

地址：050011 河北省石家庄市平安南大街 32 号 河北省地方税务局计算机管理中心 甘冀平

(gwok@yeah.net Oicq=90255)

C#，深入浅出全接触

青苹果工作室（编译）

目 录 表

一、什么是 **C#**?

二、**Java** 与 **C#**

三、**C#** 的主要特征

四、安装运行环境

五、**C#** 编辑器

六、**C#** 的程序结构

七、**C#** 和名称空间

八、**C#** 中一个经典例程的编写与编译

九、**C#** 编程实战演习 **ABC**

1、使用 **Visual Studio .NET** 编写 **C#** 程序

2、使用 **Visual C#** 创建 **Windows** 应用程序

3、创建 **C#** 类库（**Dll**）

一、什么是 C#?

C# 是由 Microsoft 开发的一种新型编程语言，由于它是从 C 和 C++ 中派生出来的，因此具有 C++ 的功能。同时，由于是 Microsoft 公司的产品，它又同 VB 一样简单。对于 web 开发而言，C# 象 Java，同时具有 Delphi 的一些优点。Microsoft 宣称：C# 是开发 .NET 框架应用程序的最好语言。

二、Java 与 C#

要学习 C#，不能不首先看一看 Java 语言。

相对于其他编程语音，Java 有一个毋庸置疑的优点：用户以及编译器第一次不必了解生成可执行代码的特定 CPU 细节。Java 引入了一个编译代码中间层，叫做字节代码，并使用一个虚拟抽象的机器，而不是一个真实的机器。当 Java 编译器结束了一个源文件的编译后，你所得到的不是可以立即在一个给定平台上运行的代码，而是可以在任何真实的平台上运行的字节代码，唯一的条件就是这个平台要理解和支持 Java。这些发展包含着一个文化的变革。作为一个开发人员，你只需要确定 Java 虚拟机 (JVM) 提供的抽象层，不同的 OS 销售商负责执行代码层，从而将中立于平台的字节代码映射到主机平台的机构中。在这种情况下，Java 似乎是统一分布式计算机世界的领袖候选人了。“编写一次，永远运行”（并且无论在哪里）一度成为 Java 诱人但却真实的口号。

那么为什么说 Java 只是“似乎”是一个好东西呢？跨平台理论的发展很好地证明了这一点。我们看到，将 Java 代码从一个平台移植到另一个平台—Java 这个语言最重要和最受吹捧的特点—并不象宣传的那样容易。任何 Java 平台都有其自己的虚拟机，它可以理解通用的字节代码，并且及时地将其编译为本地代码。矛盾由此产生，不同虚拟机的执行也很不相同，这一点足以使代码的移植比预期耗费多得多的时间，而且基本上不是自动的。

那么，Java 模型的好处在哪里呢？首先，Java 是一种先进的、面向对象的语言，包含了预防常见错误的内置功能，并在仅仅一两个对象中携带了许多经常需要用到的功能。与 C++ 相比，Java 更易于读写，不容易出错，而且更加美观，但是它速度较慢也不太灵活。想实现在任何软件和硬件平台上都可虚拟移植，Java 尽可能少地使用了公分母模型，也就是说放弃了将每个平台开发到极限的能力。第二，虚拟机的概念本身就是可移植和可共用的，因此对于分布式环境来说是理想的。Java 对于为非 Windows 平台开发代码是最好的语言。

那么对于 Windows 平台来说，Java 又怎么样呢？让 Java 适应 Windows 是不可能的，这是由于 Sun 的许可约束问题。但是 Java 实在是太吸引人了，Microsoft 比谁都能更清楚这一点。因此，Microsoft 又一次采取了“拿来主义”的手法，很好地利用了 Java 的众多特性，隆重推出了 Windows 平台的新锐力量，它就是相当简单但十分强大的面向对象的 C# 编程语言。C# 超过了 C++，它天生就包含了 .NET 框架类库中的所有类，并使语法简单化。

可以说，Java 具备的优点，C#都可以或者都将具备！

三、C#的主要特征

C# 是 .NET 的关键性语言，它是整个 .NET 平台的基础。与 C#相比，.NET 所支持的其它语言显然是配角身份。比如，VB.NET 的存在主要是对千万个 VB 开发人员的负责。对于 JScript.NET 和 Managed C++ 也同样可以这么说，后者只是增加了调用 .NET 类的 C++ 语言。C#是唯一没有在设计思路中加入了前辈语言某种遗传的新事物。

.NET 平台将 C#作为其固有语言，重温了许多 Java 的技术规则。C#中也有一个虚拟机，叫做公用语言运行环境(CLR)，它的对象也具有同样的层次。但是 C#的设计意图是要使用全部的 Win32 API 甚至更多。由于 C#与 Windows 的体系结构相似，因此 C# 很容易被开发人员所熟悉。

Java 的目的是要拯救分布式计算世界，C# 则不同。C#本质上是 C++的进化产物，使用了包括声明、表达式及操作符在内的许多 C++特征，但是 C#还有更多的增强功能，比如类型安全 (type-Safe)、事件处理、碎片帐集、代码安全性等。在 C#中，除了可以使用许多 API，更能使用 .NET 类。特别地是，我们可以处理 COM 的自动化和 C 类型的函数。

C#还让你调用无管理的代码，也就是在 CLR 引擎控制之外的代码。这种不安全的模式允许你操作原始指针来读和写内置碎片帐集控制以外的内存。

四、安装运行环境

安装 .NET SDK 是在机器上运行 C# 的第一步。.NET SDK 可以安装在 Windows ME、Windows NT 或 Windows 2000 上，但是最好的选择是 Windows 2000 上。选择了操作系统后，再执行以下步骤：

- 安装 IE 5.5
- 安装 Microsoft .NET Framework SDK。它是免费的，可以从以下站点下载。 [NET Framework SDK](#).
- 完成以上安装后，就可以在任何文本编辑器中编写代码了，最后保存为扩展名为 .cs 的文件。

五、C# 编辑器

编写 C#程序可以在文本编辑器中进行，或者在集成开发环境 Visual Studio 中进行。市场上还有一些第三方编辑器，其中一些是免费的。更多信息请查看[这里](#)。

六、C#的程序结构

一个 C# 程序包含一个类，这个类中至少有一个公用的静态方法 Main，这个方法对程序进行初始化并终止程序。在 Main 方法中创建子对象、执行方法并实现软件的逻辑处理。下面是一个典型的微型 C# 例程：

```
using System;
class MyFirstApp
{
    static int Main(String[] args)
    {
        System.Console.WriteLine ("Hello .NET");
        return 1;
    }
}
```

在 C# 中，要使用下面的声明来引入外部定义，而不是用象 C++ 中的 #include：

```
using System;
using System.Data;
```

然后，使用 C# 编译器 csc.exe 编译代码。假定将前面的代码保存为文件 hello.cs，使用以下命令：

```
csc hello.cs
```

结果就生成了 hello.exe，它向控制台输出窗口写入信息 "Hello .NET"。

尽管编译后的结果文件包含 .exe 后缀，但 hello.exe 却不是一个真正的、明确的 CPU 代码段。实际上，它包含了 .NET 字节代码。当启动 hello.exe 时，CLR 将提取编译器写入代码中的重要元数据。接着，一个叫做 Just-In-Time 编译器的模块将代码映射到特定的 CPU 中，开始实际的执行过程。

七、C# 和名称空间

实际中的 C# 程序通常包含多个文件，其中每个文件都可以包含一个或多个名称空间。一个名称空间就是一个名字，它向编译器描绘出一些软件实体，如类、界面、枚举以及嵌入的名称空间。名称空间和数据类型一样必须有唯一的名称。在一个 C# 程序中，可以通过一个元素的完整资格名称来识别它，这个资格名称表明出层次关系。例如，System.String 是 .NET String 类型完整的资格名称。但是为了简化代码起见，只要声明正在使用 System 名称空间：

```
using System;
```

就可以使用一个相对名称如 String 来作为完整名称的同义词，而最后依然代表 System.String。

通过使用 namespace 关键字，我们还可以将 C# 程序或者类包裹在自身的名称空间中，比如：

```
namespace MyOwn
{
    using System;        // for String
    class MyFirstApp
    {
        static int Main(String[] args)
```

```

    {
        System.Console.WriteLine ("Hello .NET");
        return 1;
    }
}

```

名称空间 `MyOwn` 是全局名称空间的一部分。调用它不需要再使用前缀，因为其完整资格名称就是简单的 `MyOwn`。定义一个名称空间是保持公共名称唯一性的一个途径。实际上，如果两个类的名称发生冲突，但只要它们分别属于不同的名称空间，两个类仍然是各自唯一的。

八、C# 中一个经典例程的编写与编译

1、编写代码

“Hello World”几乎是学习任何一门编程语言都要涉及的第一个例程。下面也让我们用 C# 完成这个工作。在上面提到的任意一个 C# 编辑器中（比如写字板），键入以下代码：

```

using System;
class MyClass
{
    static void Main() {
        Console.WriteLine("Hello World!");
    }
}

```

然后保存为文件 `myclass.cs`。

2、编译程序

注意：C# 编译器要求至少一个自变量，比如文件名。假设你的 C# 文件名是 `myclass.cs`，现在用命令程序 `csc.exe` 来编译上面的 `myclass.cs` 文件：

```
csc myclass.cs
```

于是，C# 编译器在工程文件的 `bin` 目录下生成了一个 `myclass.exe` 文件。运行这个 `exe`，看输出是什么。

3、代码含义

下面我们逐行看看这些代码的含义：

程序的第一行是 `using System`。为什么要 `using System` 呢？因为 `System` 是存储系统类的名称空间，程序中用来在控制台上显示输出的 `Console`（控制台）类就是在 `System` 名称空间中定义的。

下一行是 `class MyClass`。C# 中的 `class` 关键字用于创建一个新类。每个类都有一个静态的 `void Main()` 函数，这个函数就是一个 C# 程序的入口。

`Console` 类的 `WriteLine` 方法负责向控制台输出文本信息。

九、C#编程实战演习 ABC

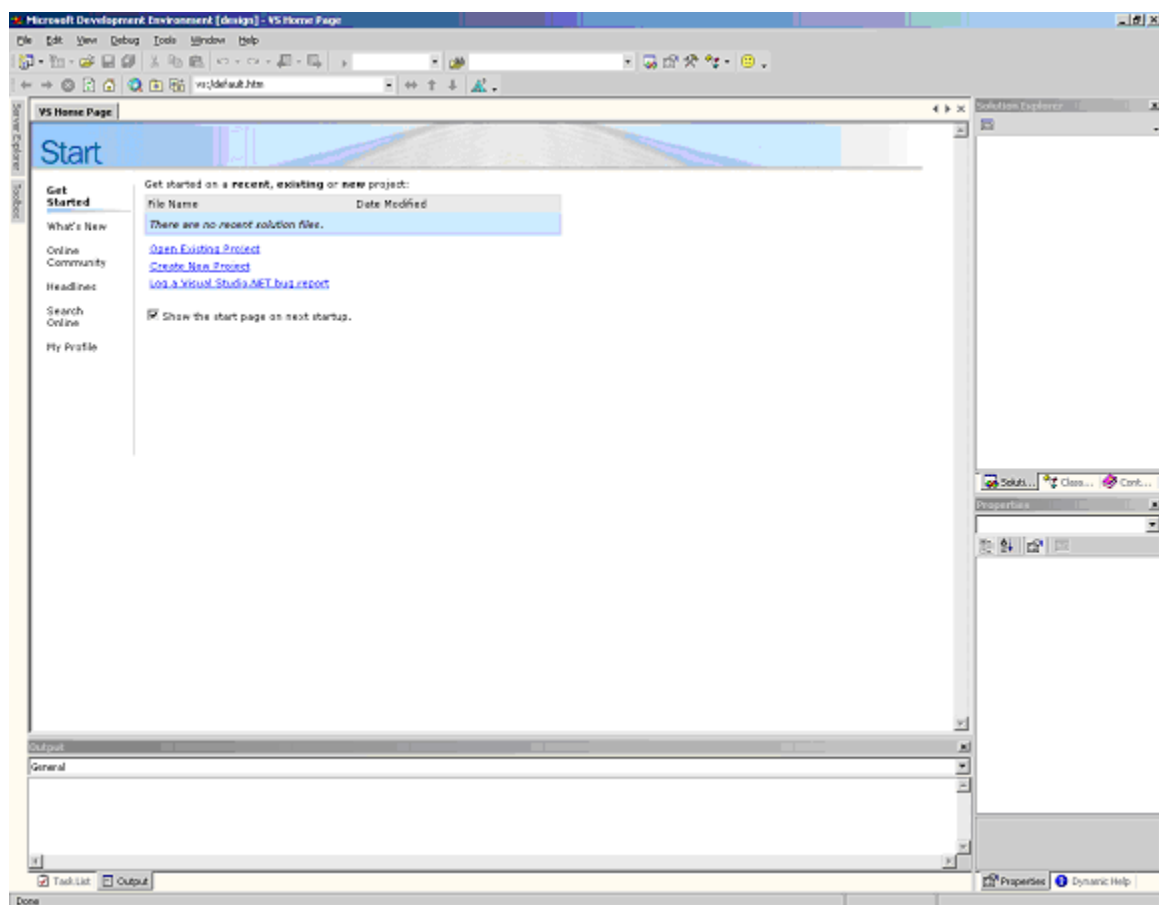
1、用 Visual Studio .NET 编写 C#程序

Visual Studio.NET 是 Microsoft 新一代的旗舰开发环境，在这个环境中，我们能够看到 Microsoft 将所有开发工具都集成到一个 IDE 中。我们惊喜地发现，我们拥有了一个所有编程语言都适用的代码编辑器。而且，这个环境中还具一个 HTML 编辑器、一个 XML 编辑器、一个 SQL Server 界面以及一个 Server Explorer。

下面，我们将学习如何在 Visual Studio .NET 中编写 C#程序。

初始页面

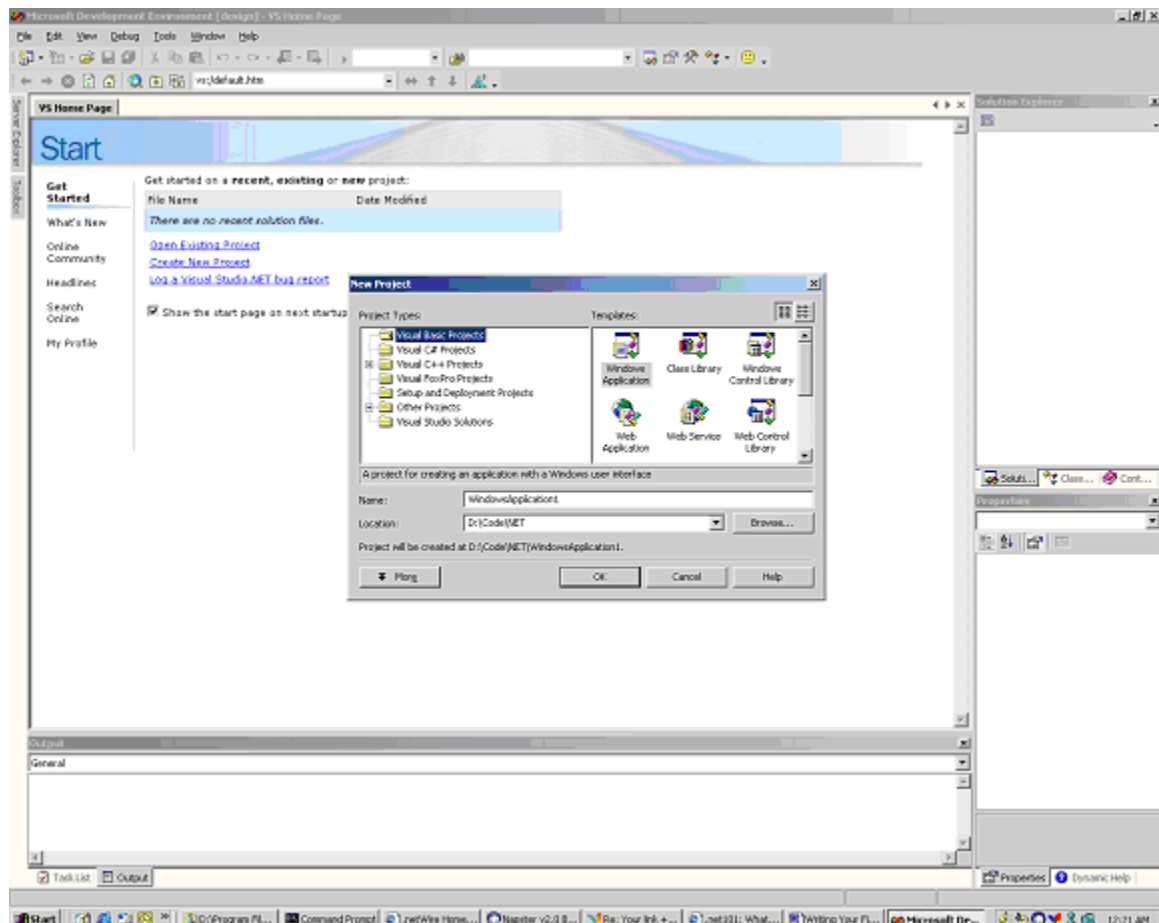
启动 VS.NET 后，我们会看到与以前版本完全不同的景象。实际上，它看起来更象 Visual J++。一开始出现的是初始页面，这是一个 HTML 格式的页面。



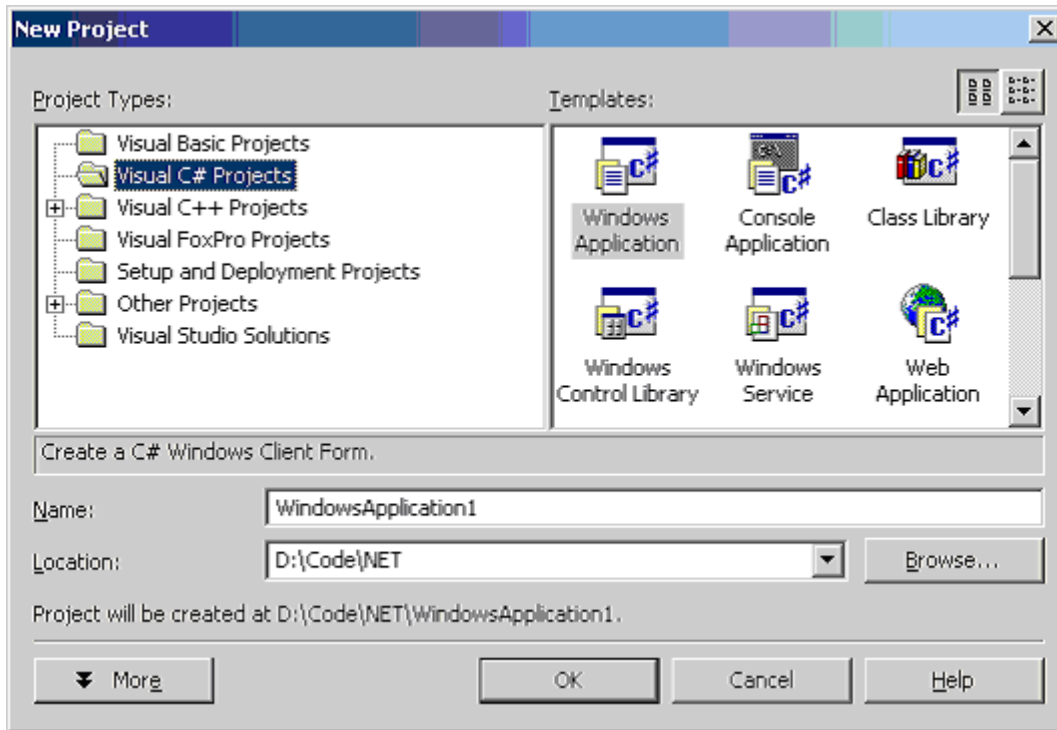
从上图中，我们能看到在线 Microsoft web 站点的链接、现有解决方案列表以及创建属于自己的个性文件（Profile）的功能。

创建 C# 控制台应用程序

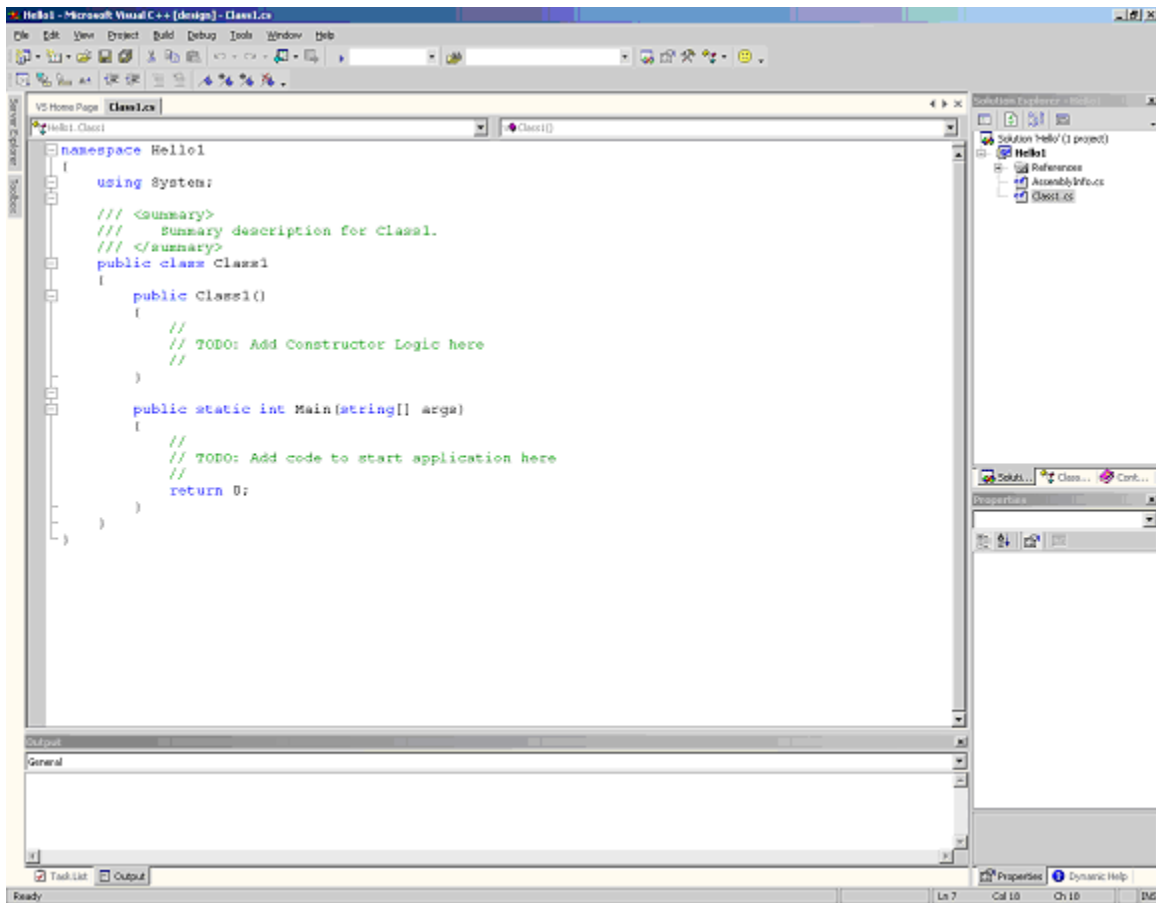
在 Visual Studio.Net 中创建 C# 应用程序是简单得不能再简单的事情。点击 "Create New Project"（创建新工程）链接 后，会出现以下的对话窗口。从这个窗口中的多种工程样本中，选择需要的一个：



在这里，我们选择 Visual C# 工程文件。选中后，出现下面的对话框：



然后再选择“Console Application”（控制台应用程序），命名为 "Hello1"（不需要加引号），点击“OK”，VS.NET 开始生成应用程序的壳（shell）：



现在让我们来分析一下这个壳中已经具备的代码。

首先我们会看到关键字 **namespace**（名称空间）。你可以将名称空间简单地理解为是将类归在一起的东西。**C#** 中的名称空间与 **C++** 中的名称空间相似，但还具备一些额外的功能。另外，它也与 **Java** 中的 **package** 关键字相似。

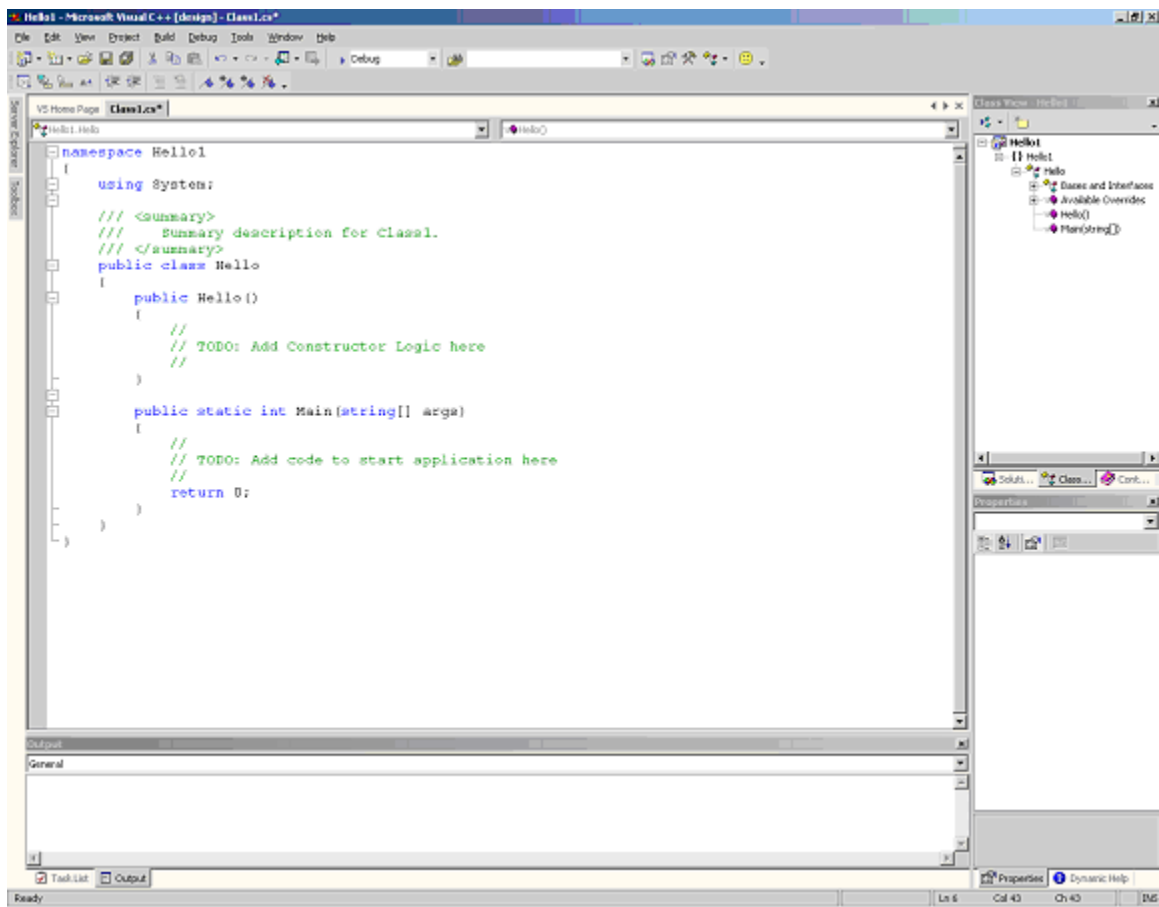
下面的语句是 **using** 命令，它负责告诉编译器在 **System** 库中寻找未知的类。**.NET** 携带了一套可扩展的系统库，由它们的名称空间名确定其范围。在 **C#** 中，所有的 **I/O** 操作都是系统库的一部分，而不是语言本身的一部分。

接着我们看到 **"public class Class1"** 声明。作为一个先进的面向对象的开发语言，**C#** 中的所有代码都必须包含在一个类里面，不存在全局函数或数据。

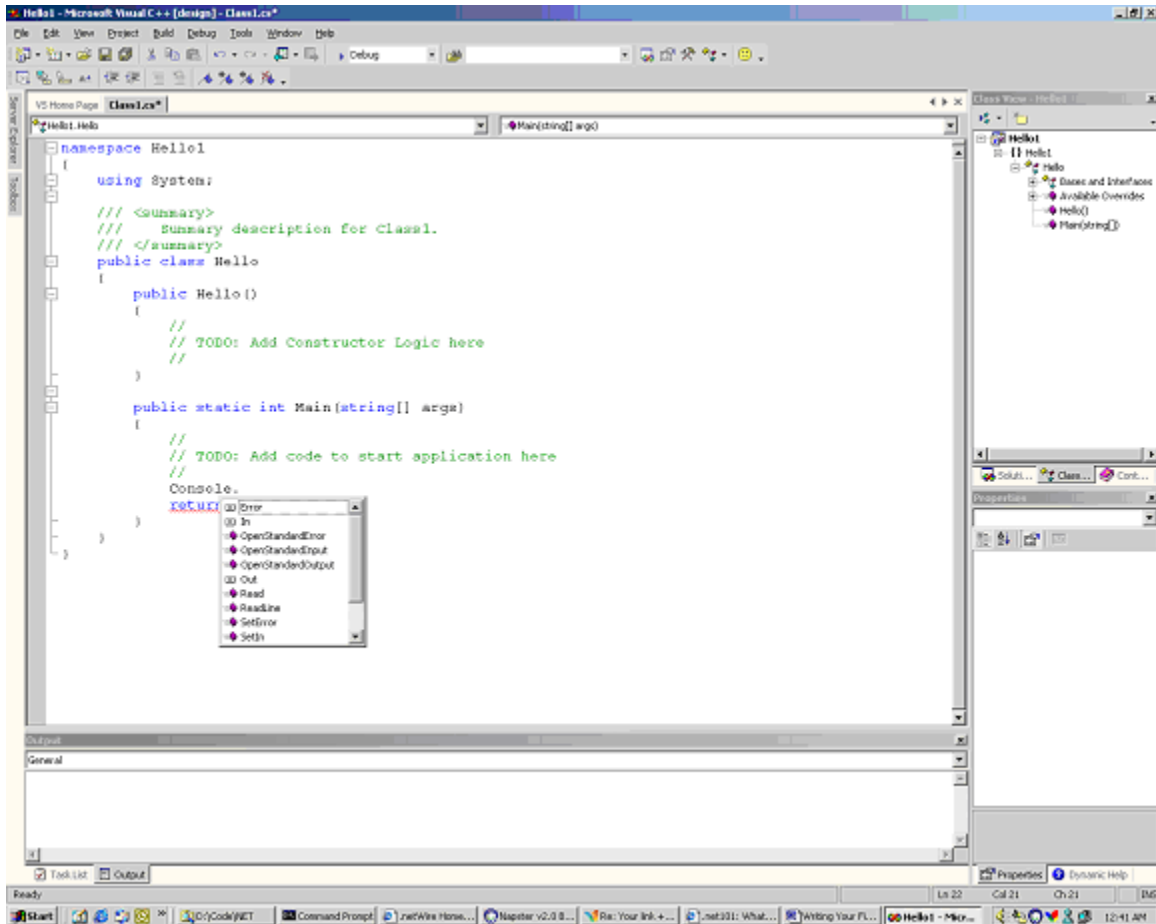
程序中还包含一个方法 **Main**，它是所有 **C#** 程序的进入大门。

修改一下代码

现在我们要增加一些代码来执行一些有用的操作。第一件事是将右上角的窗口切换到 **Class View**（类视图），然后展开 **"Hello1"** 名称空间，点击 **"Class1"** 类。在属性窗口中将名称修改为 **Hello**。现在的代码窗口变为：



将光标移到 **Main** 方法内的 **TODO** 注释之后，输入“**Console**”这个命令。请注意发生了什么：



你能看到系统自动列出了 **Console** 类的相关方法。选择 **WriteLine**，然后写入下面这一行：

```
Console.WriteLine("Hello from VS.NET!");
```

运行

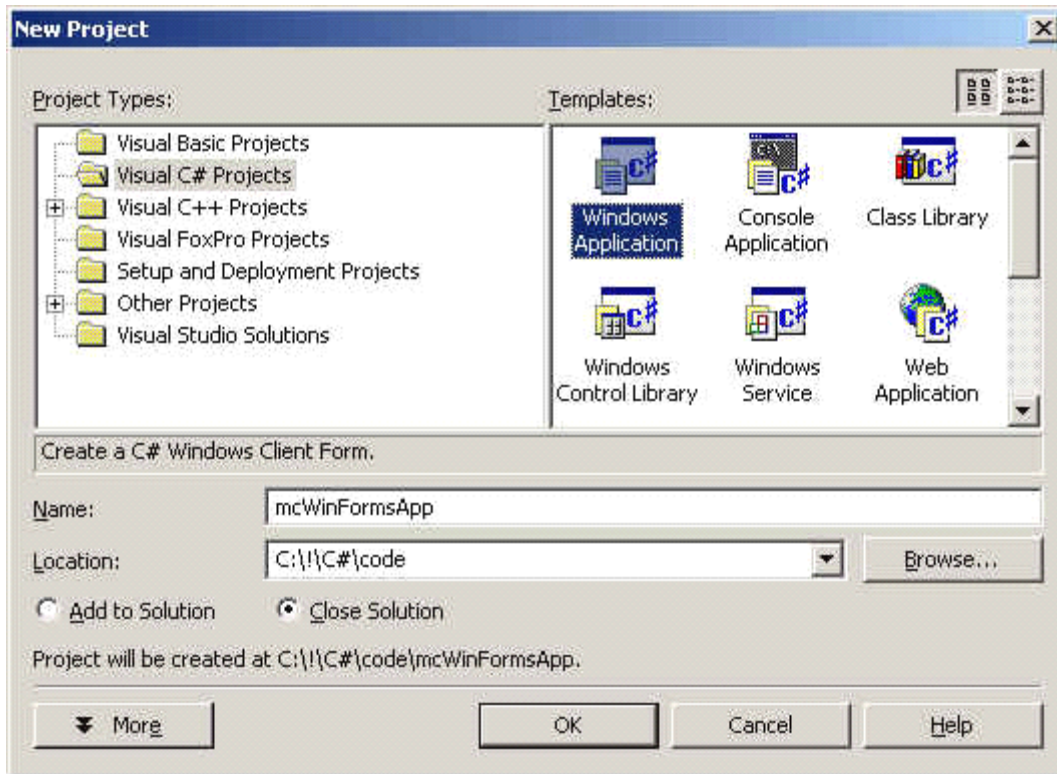
现在，从“Build”菜单中选择“Build”项，然后从“Debug”（调试）菜单中选择“Start Without Debugging”（不调试启动）。最后，控制台应该显示出“Hello From VS.NET!”的信息。这说明，我们已经大功告成了 :-)

2、用 Visual C# 创建 Windows 应用程序

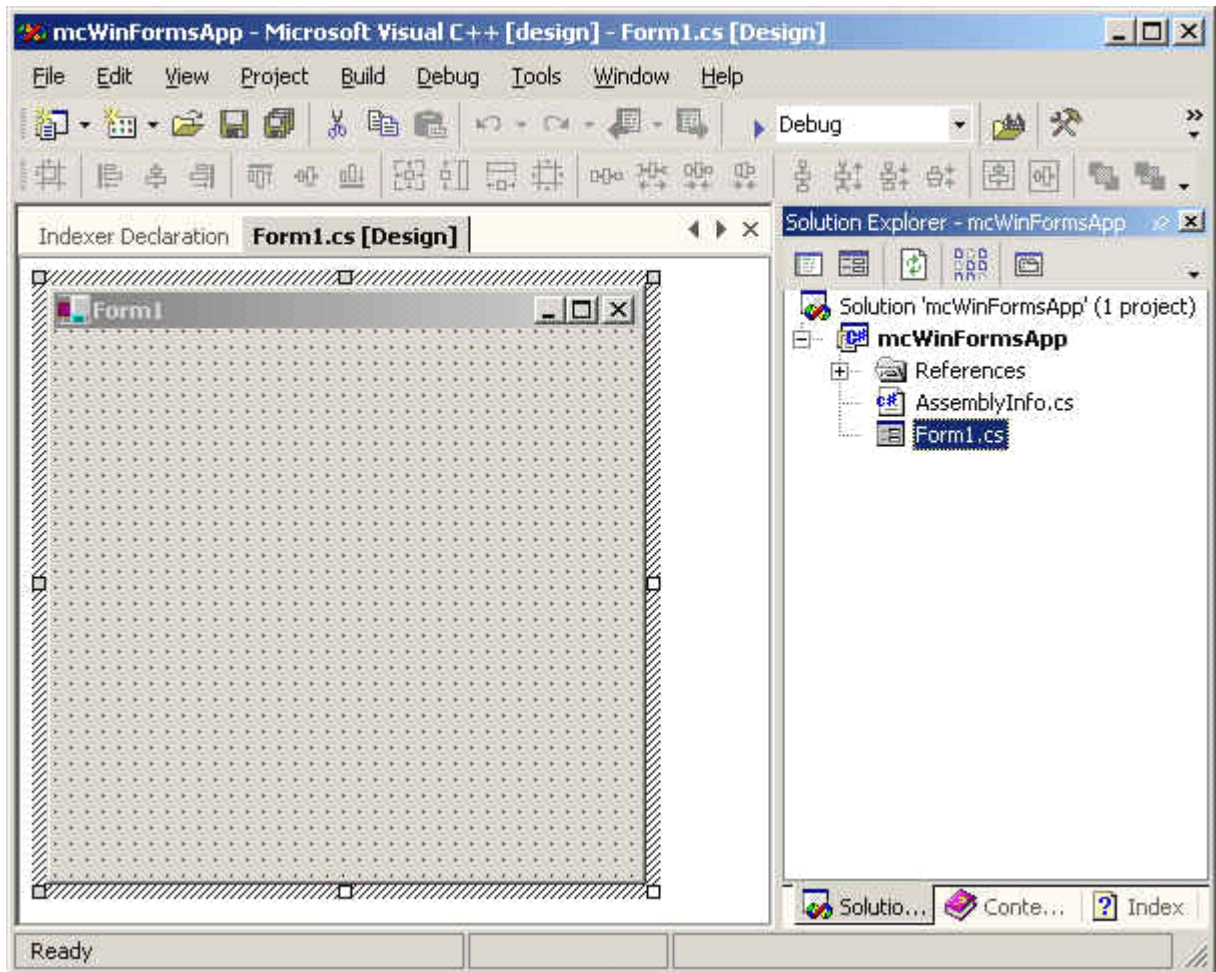
在 Visual C# 创建一个 Windows (GUI) 应用程序要以前版本的 VC++ 容易得多。下面将介绍用 Visual C# 工程文件向导创建 Windows 应用程序的过程。

创建应用程序框架

在 VS .NET IDE 中选择“新建->工程文件->Visual C# 工程文件->Windows 应用程序”：



然后点击 OK，出现一个表单设计视图（这与 VB 或 Delphi 相同）。在右侧我们看到了一个解决方案导航器（Solution Explorer）。向导为新表单增加了一个 Form1.cs 文件，其中包括了这个表单及其所有子窗口的的代码：



双击 Form1.cs 就能看到这个代码：

```
namespace mcWinFormsApp
{
    using System;
    using System.Drawing;
    using System.Collections;
    using System.ComponentModel;
    using System.Windows.Forms;
    using System.Data;
    /// <summary>
    /// Summary description for Form1.
    /// </summary>
    public class Form1 : System.Windows.Forms.Form
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.Container components;
        public Form1()
        {

```

```

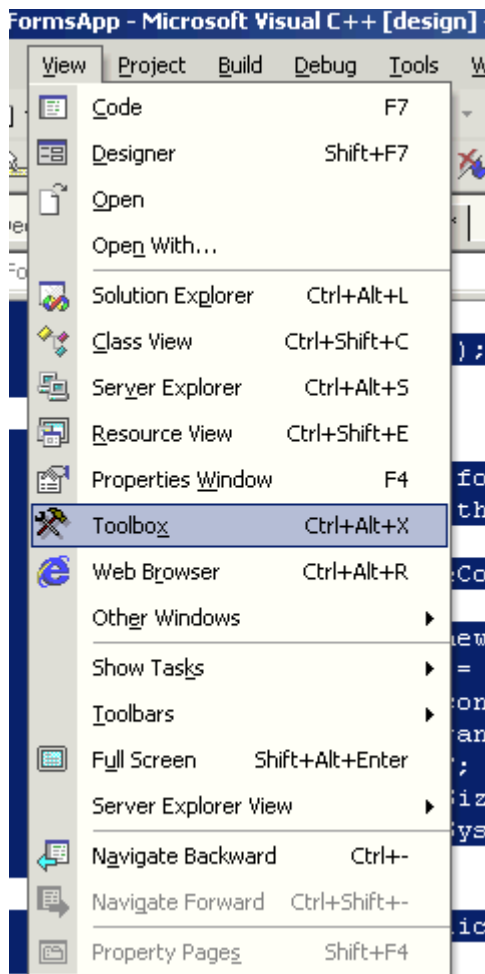
//
// Required for Windows Form Designer support
//
InitializeComponent();
//
// TODO: Add any constructor code after InitializeComponent call
//
}
/// <summary>
/// Clean up any resources being used.
/// </summary>
public override void Dispose()
{
    base.Dispose();
    components.Dispose();
}
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
    this.components = new System.ComponentModel.Container ();
    //@this.TrayHeight = 0;
    //@this.TrayLargeIcon = false;
    //@this.TrayAutoArrange = true;
    this.Text = "Form1";
    this.AutoScaleBaseSize = new System.Drawing.Size (5, 13);
    this.Click += new System.EventHandler (this.Form1_Click);
}
protected void Form1_Click (object sender, System.EventArgs e)
{
}
/// <summary>
/// The main entry point for the application.
/// </summary>
public static void Main(string[] args)
{
    Application.Run(new Form1());
}
}
}
}

```

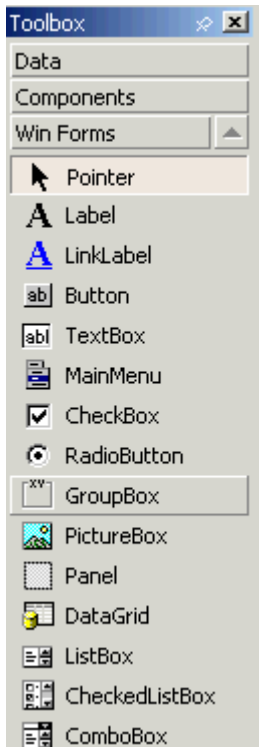
从以上代码中，我们看到：向导增加了一个默认的名称空间以及对 WinForms 所要求的不同名称空间的引用；Form1 类是从 System.WinForms.Form 中派生出来的；InitializeComponent 方法负责初始化（创建）表单及其控件（当在表单中托放下一些控件时，可以看到它的更多细节）；Dispose 方法负责清除所有不再使用的资源。

添加控件

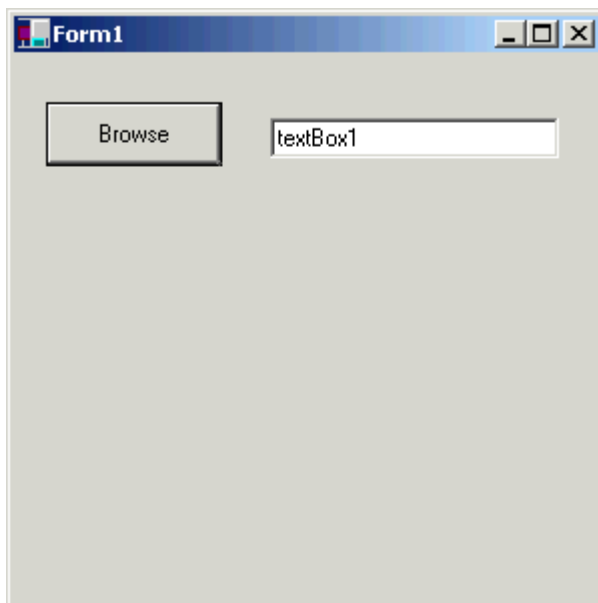
要向一个表单中添加控件或者子窗口，需要打开 工具箱 ToolBox。这个工具箱的概念来自 VB。点击菜单“视图—>工具箱”，激活工具箱功能：



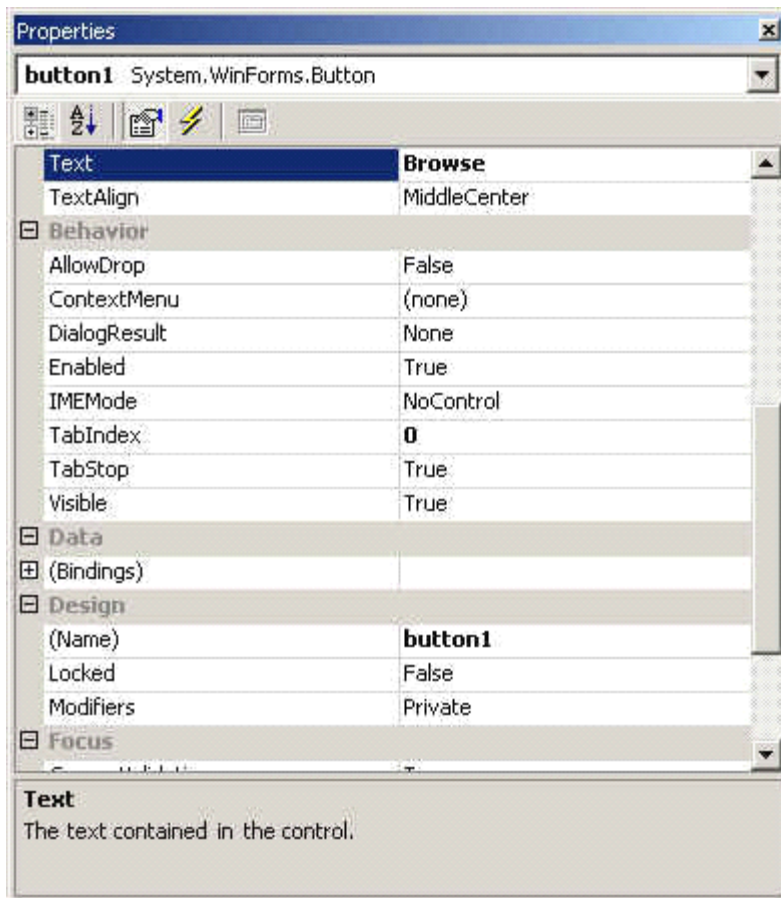
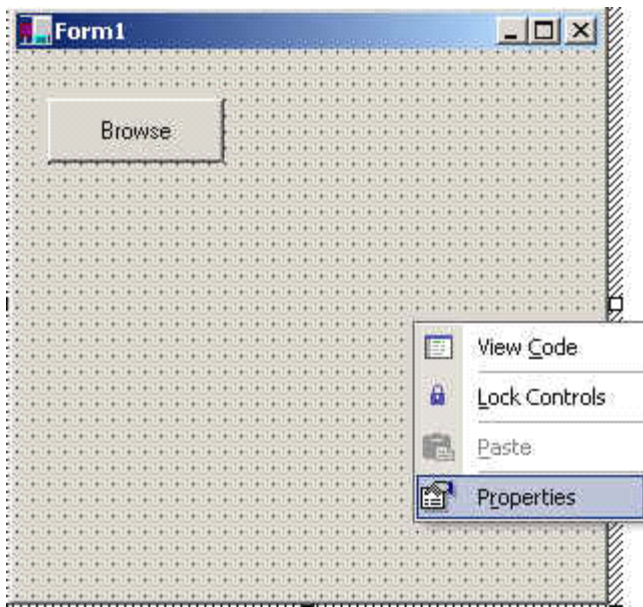
ToolBox（工具箱）窗口的样子如下图所示。现在就可以添加控件了，添加方法与 Visual Studio 的以前版本一样，拖放或者双击控件都可以。



首先在表单上托放下一个按钮和一个编辑框，然后让我们看看系统向初始组件（InitializeComponent）中增加了什么东西。



接着在属性窗口中设置控件的属性，这与 VB 中的操作方式一样。在控件上点击右键，并点中“属性”菜单条就可以调出属性窗口。



现在看看 `InitializeComponent` 方法，就会发现这些代码已经增加到其中了。接着手工修改一下这些代码：

```

this.components = new System.ComponentModel.Container ();
this.button1 = new System.Windows.Forms.Button ();
this.textBox1 = new System.Windows.Forms.TextBox ();
//@this.TrayHeight = 0;
//@this.TrayLargeIcon = false;
//@this.TrayAutoArrange = true;
button1.Location = new System.Drawing.Point (16, 24);
button1.Size = new System.Drawing.Size (88, 32);
button1.TabIndex = 0;
button1.Text = "Browse";
button1.Click += new System.EventHandler (this.button1_Click);
textBox1.Location = new System.Drawing.Point (128, 32);
textBox1.Text = "textBox1";
textBox1.TabIndex = 1;
textBox1.Size = new System.Drawing.Size (144, 20);
this.Text = "Form1";
this.AutoScaleBaseSize = new System.Drawing.Size (5, 13);
this.Click += new System.EventHandler (this.Form1_Click);
this.Controls.Add (this.textBox1);
this.Controls.Add (this.button1);

```

添加事件处理器

最后，要为按钮增加一个事件处理器，实现浏览文件的目的。在按钮上双击，打开 `Button1_Click` 事件处理器。同理，使用同样的方法可以为任何控件编写事件处理器。

```

protected void button1_Click (object sender, System.EventArgs e)
{
    OpenFileDialog fdlg = new OpenFileDialog();
    fdlg.Title = "C# Corner Open File Dialog" ;
    fdlg.InitialDirectory = @"c:\\" ;
    fdlg.Filter = "All files (*.*)|*.*|All files (*.*)|*.*" ;
    fdlg.FilterIndex = 2 ;
    fdlg.RestoreDirectory = true ;
    if(fdlg.ShowDialog() == DialogResult.OK)
    {
        textBox1.Text = fdlg.FileName ;
    }
}

```

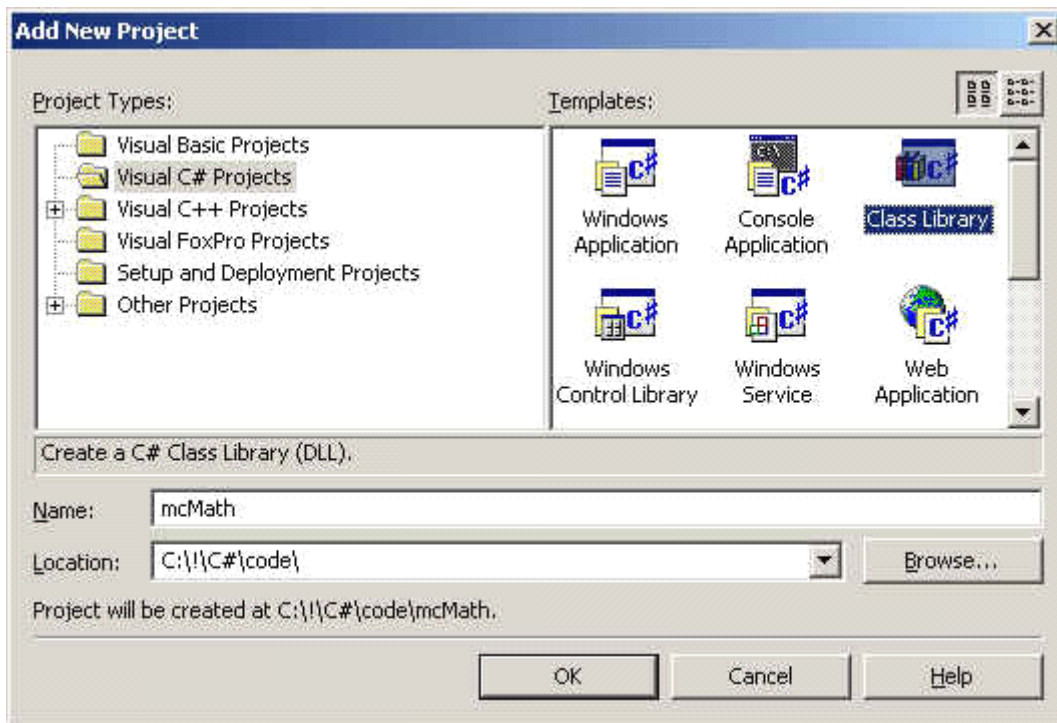
到此就完成了所有步骤，剩下的就是运行这个程序。它实现了浏览一个文件，然后将选择的文件名装进文本框的功能。请下载相关代码：[winFormApp.zip](#)。

3、创建 C# 类库 (DLL)

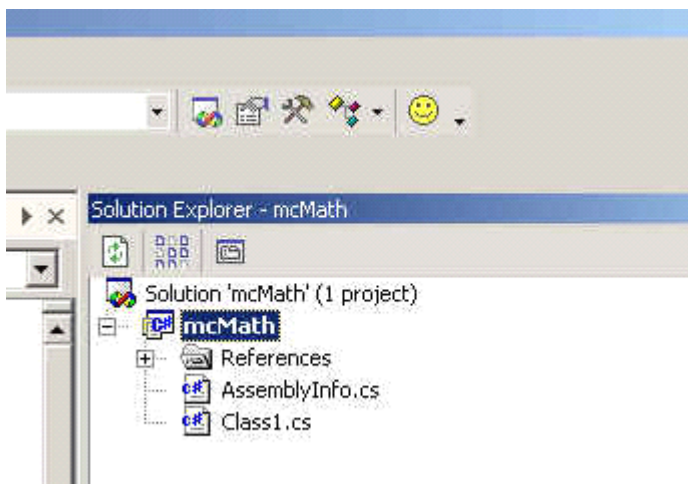
以前在 VC++ 中创建一个 dll 文件不能说简单，但在 Visual C# 中，这将同样是轻而易举的事情。下面的介绍分成两部分：1、创建 DLL 2、在客户端测试 dll。

(1) 创建 DLL

首先创建一个空的类库工程。在 VS.NET 集成环境（IDE）中选择“文件->新建->工程文件->Visual C# 工程->类库”，点击 Browse（浏览）按钮选择工程文件名和相应的目录，再点击 OK：



接着看看工程和它的相关文件。Solution Explorer（解决方案探测器）向工程中增加两个 C# 类，第一个是 AssemblyInfo.cs，第二个是 Class1.cs。我们并讨论 AssemblyInfo，重点介绍 Class1.cs。



双击 Class1.cs，就能看到一个名称空间 mcMath。我们将在客户机引用这个名称空间以使用这个类库：

```
namespace mcMath
{
```

```

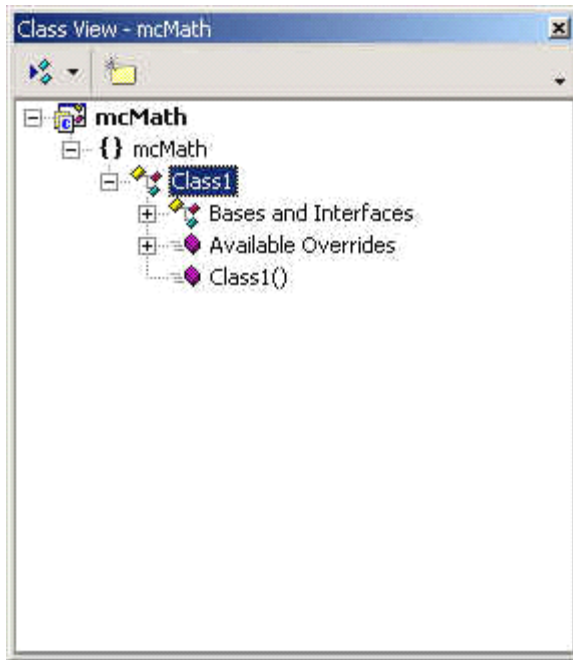
using System;
/// <summary>
/// Summary description for Class1.
/// </summary>
public class Class1
{
    public Class1()
    {
        //
        // TODO: Add Constructor Logic here
        //
    }
}
}

```

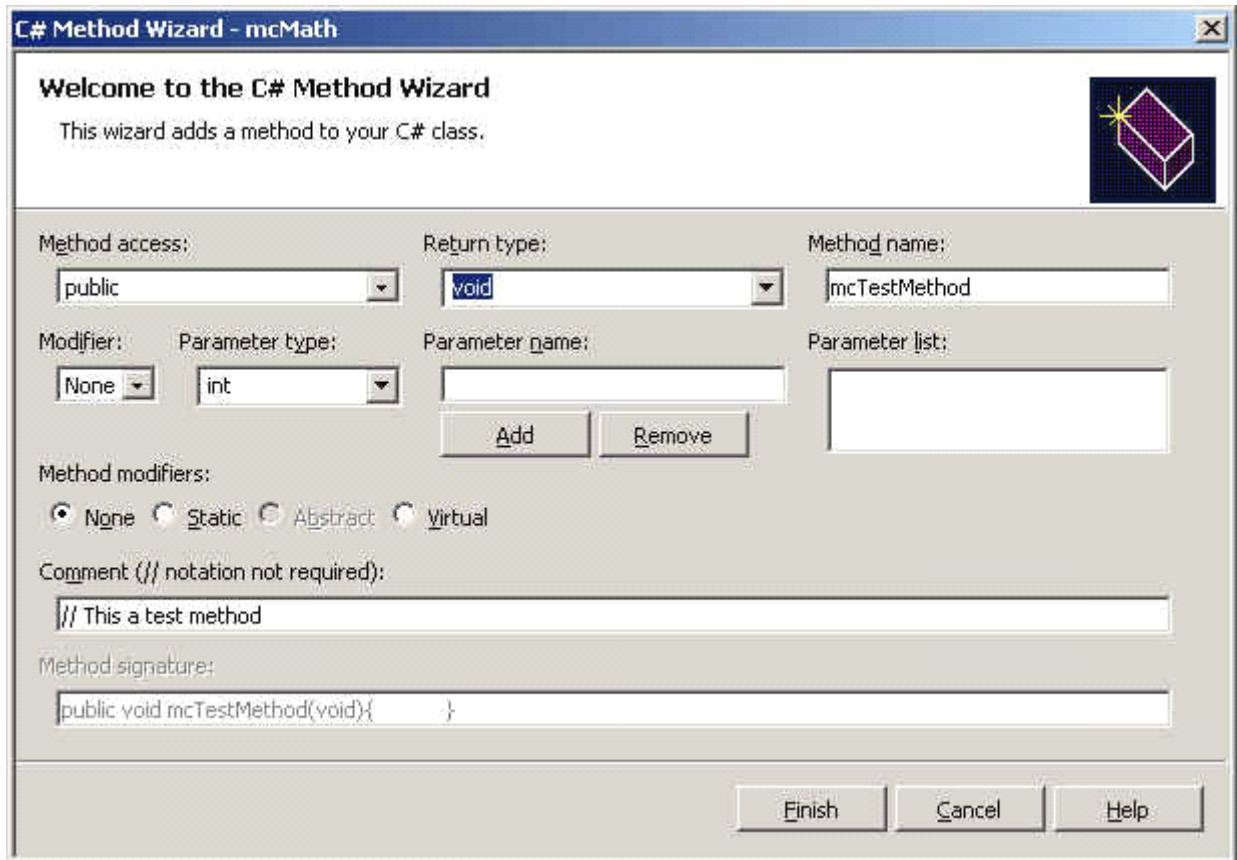
现在就可以 Build（构造）这个工程了。Build（构造）完毕后，就会在工程文件的 bin/debug 目录中生成 mcMath.dll 文件。

增加一个方法

从 View（视图）菜单中打开 ClassView（类视图），开始只显示 Class1，没有方法和属性。现在来增加一个方法和一个属性。



用鼠标右键单击“Class1”，选择“Add（增加）-> Add Method（增加方法）”，这时将弹出 C# 方法生成向导：

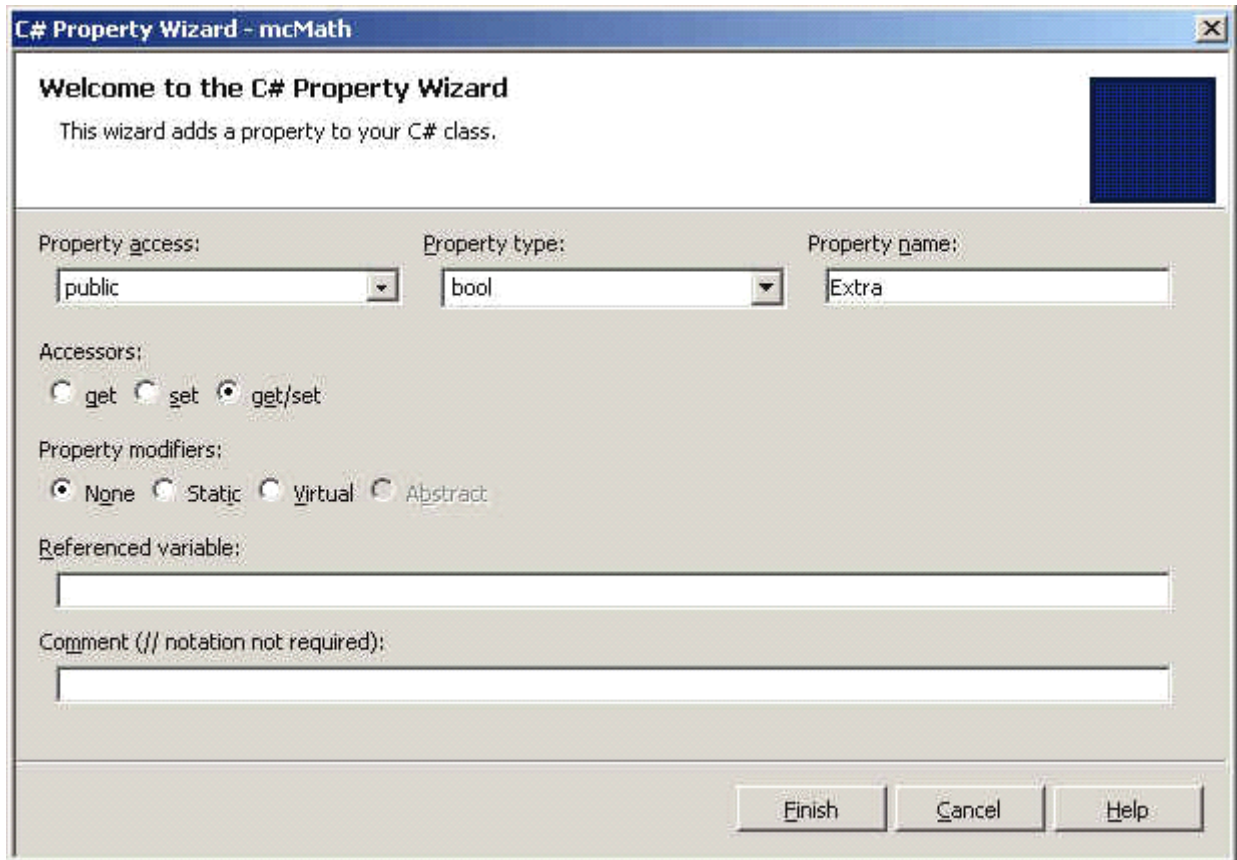


The image shows a Windows-style dialog box titled "C# Method Wizard - mcMath". It has a standard title bar with a close button. The main area is divided into several sections. At the top, it says "Welcome to the C# Method Wizard" and "This wizard adds a method to your C# class." with a small icon of a box with a star. Below this, there are three groups of controls: "Method access:" with a dropdown set to "public"; "Return type:" with a dropdown set to "void"; and "Method name:" with a text box containing "mcTestMethod". The next row has "Modifier:" with a dropdown set to "None", "Parameter type:" with a dropdown set to "int", "Parameter name:" with an empty text box, and "Parameter list:" with an empty list box. Below the "Parameter name:" text box are "Add" and "Remove" buttons. The "Method modifiers:" section has four radio buttons: "None" (selected), "Static", "Abstract", and "Virtual". Below that is a "Comment (// notation not required):" section with a text box containing "// This a test method". The "Method signature:" section has a text box containing "public void mcTestMethod(void){ }". At the bottom right are three buttons: "Finish", "Cancel", and "Help".

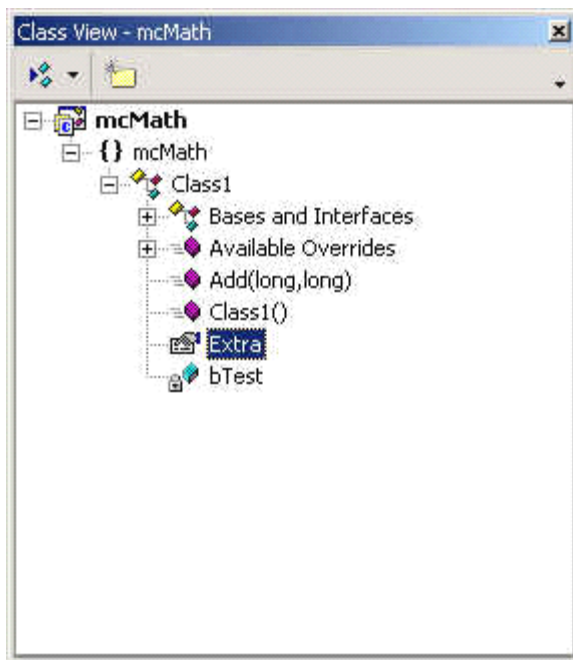
在这个窗口中增加方法名、存取类型、返回类型、参数以及注释信息。使用 Add（增加）和 Remove（取消）按钮可分别从参数列表中增加和取消参数。这里增加了一个方法 long Add(long val1, long val2)，它负责将两个数字相加并返回和。

增加一个属性

同理可以 C# 属性生成向导，向类中增加一个属性：



增加了一个方法和一个属性后，Class1 变成下图所示的样子：



仔细观察这个 Class1，你会发现 C# 的向导程序向类中增加了如下两个函数：

```
public long Add (long val1, long val2)
```

```

{
    return 0;
}
public bool Extra
{
    get
    {
        return true;
    }
    set
    {
    }
}
}

```

向类中增加代码

这里把 Class1 修改成为 mcMathComp，因为 Class1 是个容易造成混淆的名字，当想将这个类用在一个客户应用程序中时会造成问题。下面的代码对上面的做了些调整：

```

namespace mcMath
{
    using System;
    public class mcMathComp
    {
        private bool bTest = false;
        public mcMathComp()
        {
        }
        public long Add (long val1, long val2)
        {
            return val1 + val2;
        }
        public bool Extra
        {
            get
            {
                return bTest;
            }
            set
            {
                bTest = Extra ;
            }
        }
    }
}

```

构造 dll

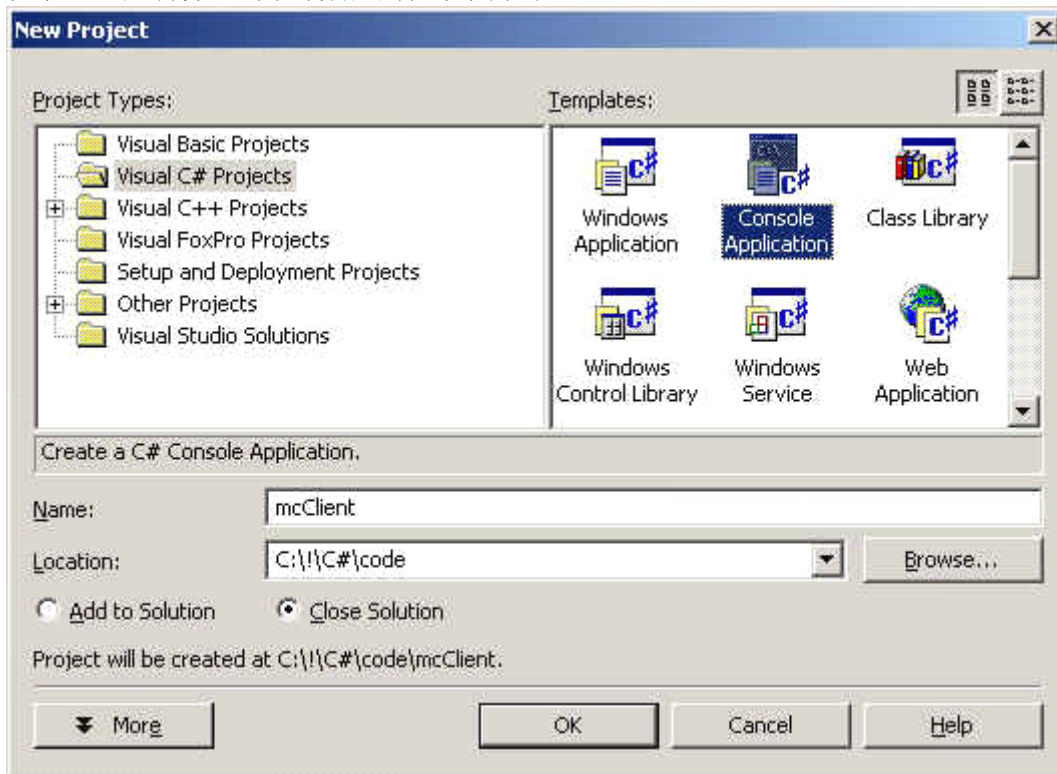
选择 Build 菜单创建 dll 文件，如果一切 OK，就会在工程文件的 bin\debug 目录生成 dll 文件。

(2) 在客户端测试 dll

在客户端调用 dll 的方法和属性也是非常简单的工作，请遵照下面的步骤执行：

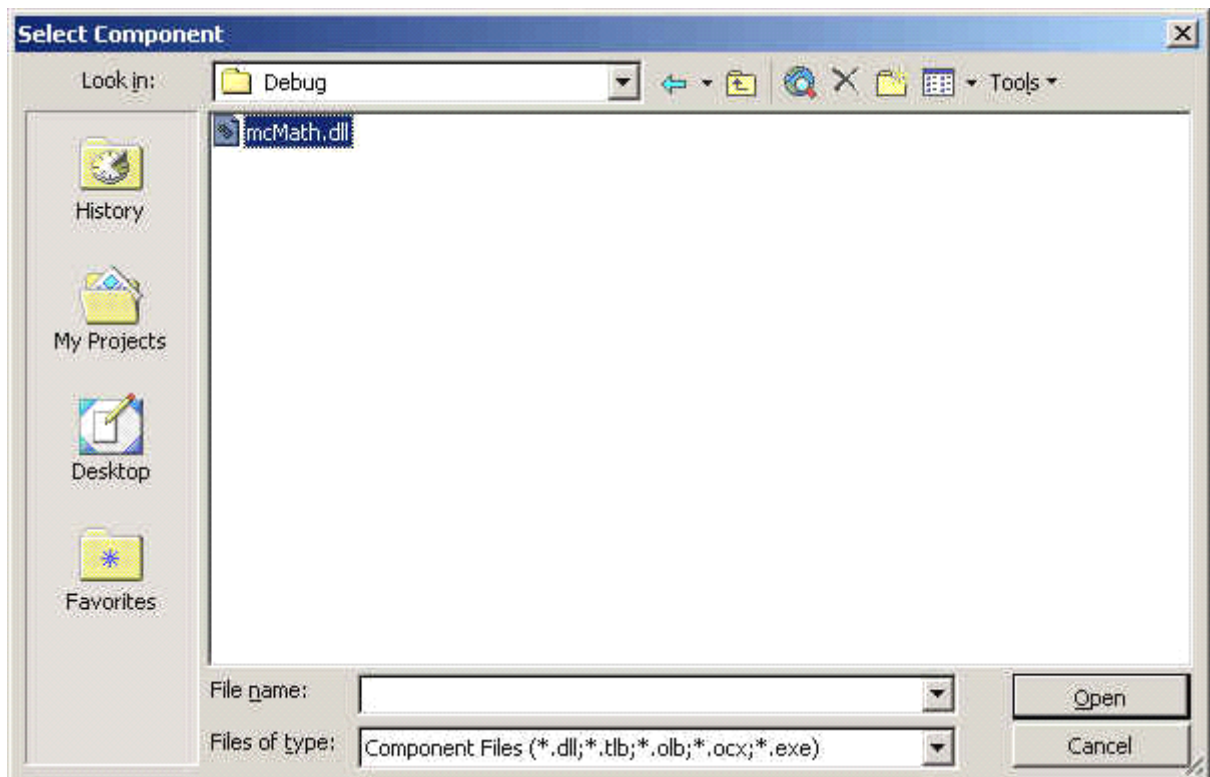
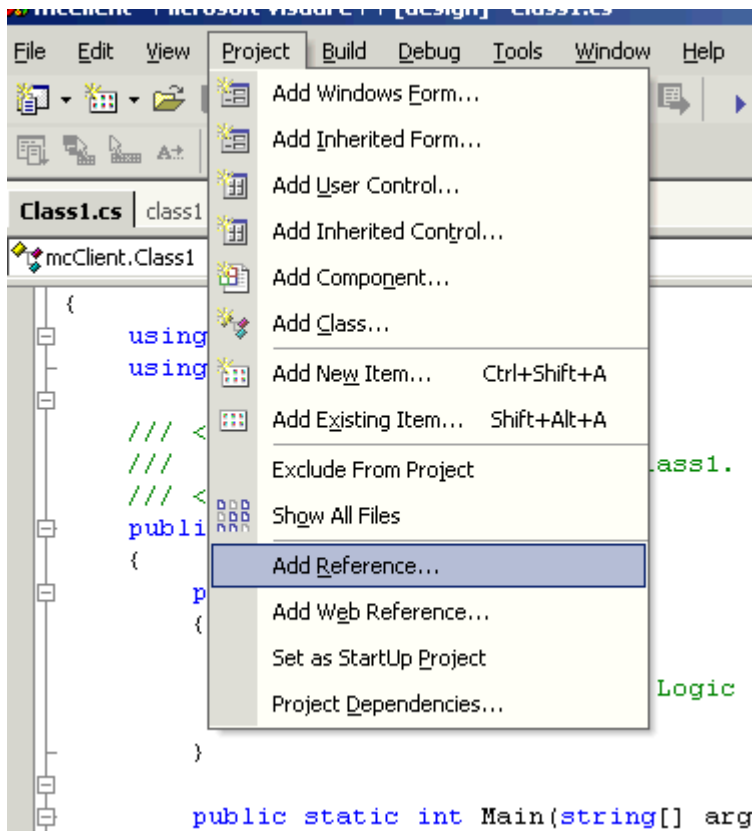
1、创建控制台应用程序

在 VS.NET IDE 集成环境中选择“文件→新建→工程文件→Visual C#工程文件→控制台应用程序”，最终将在这个控制台应用程序中测试 dll。

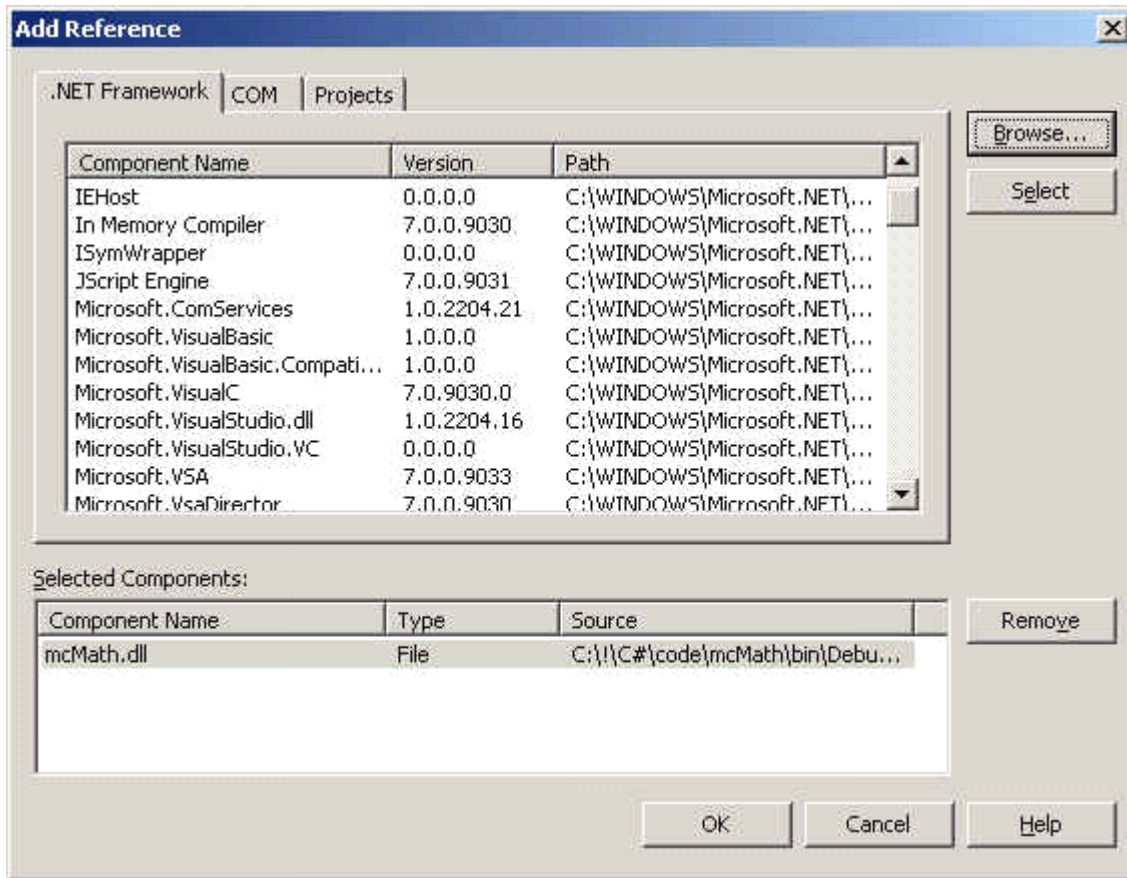


2、增加名称空间的引用

选择“工程→添加引用”（Project→Add reference），然后浏览文件找到 dll，点击 Ok：



引用添加向导程序将向当前工程文件中增加对相关库的引用：



3、调用 mcMath 名称空间，创建 mcMathComp 的对象，并调用其方法和属性。

现在距离调用组件的方法和属性只有一步之遥了。请按照以下步骤进行：

- 引用名称空间：using mcMath
- 创建一个 mcMathComp 的对象：mcMathComp cls = new mcMathComp();
- 调用方法和属性
mcMathComp cls = new mcMathComp();
long lRes = cls.Add(23, 40);
cls.Extra = false;

以下是完整的工程文件代码：

```

namespace mcClient
{
using System;
using mcMath;
/// <summary>
/// Summary description for Class1.
/// </summary>
public class Class1
{
public Class1()
{
//
// TODO: Add Constructor Logic here

```

```
//  
}  
public static int Main(string[] args)  
{  
    mcMathComp cls = new mcMathComp();  
    long lRes = cls.Add( 23, 40 );  
    cls.Extra = false;  
    return 0;  
}  
}  
}
```

请下载工程文件: [mcMath.zip](#)