

实验一 C#类库及控件编程

1 背景知识

1.1 C#语言简介

1998年12月,微软(Microsoft)公司开始开发.NET平台上的面向对象的程序设计语言,由Anders Hejlsberg(Turbo Pascal和Delphi的开发者)领导设计;2000年2月,将其正式命名为C#(C Sharp)。C#是专门为.NET平台而设计的,也是.NET编程的首选语言。学习C#语言,就必须理解.NET的运行机制,了解.NET Framework(框架)。

.NET Framework简称.NET,是微软公司为开发应用程序创建的一个富有革命性的新平台。.NET框架定义了.NET应用程序的开发和运行环境,包括可在.NET应用程序中使用的类库,它与Java的虚拟机类似。开发人员可以在此平台或者环境下开发各种应用程序,就像火车必须在铁轨上才能行驶一样,用C#编写的程序必须在.NET Framework上运行,所以首先要在计算机中安装.NET Framework。

.NET Framework是支持生成和运行下一代应用程序与XML(Extensible Markup Language,可扩展标记语言) Web Services的内部Windows组件。.NET Framework旨在实现下列目标:(1)提供一个一致的面向对象的编程环境,无论对象代码是在本地存储和执行,还是在本地执行但在Internet上分布,或者是在远程执行;(2)提供一个将软件部署和版本控制冲突最小化的代码执行环境;(3)提供一个可提高代码(包括由未知的或不完全受信任的第三方创建的代码)执行安全性的代码执行环境;(4)提供一个可消除脚本环境或解释环境的性能问题的代码执行环境;(5)提供一个开发不同类型的应用程序(如基于Windows的应用程序和基于Web的应用程序)的一致方法;(6)使用工业标准XML进行数据通信,确保基于.NET Framework的代码可与其他代码集成。

.NET Framework有两个主要组件:公共语言运行库(CLR, Common Language Runtime)和.NET Framework类库。公共语言运行库是.NET Framework的基础,它提供内存管理、线程管理和远程处理等核心服务,并强制实施严格的类型安全检查来提高安全性和可靠性。以运行库为目标的代码称为托管代码,而不以运行库为目标的代码则称为非托管代码。.NET Framework的另外一个主要组件是类库,它是一个综合性的、面向对象的、可重用的类型集合,供开发者开发各种类型的应用程序。

C#源程序要经编译器转换为微软中间语言MSIL(Microsoft Intermediate Language),再利用JIT即时编译器(Just-In-Time Compiler)和通用语言运行时CLR生成操作系统的可执行程序,如图1-1所示。

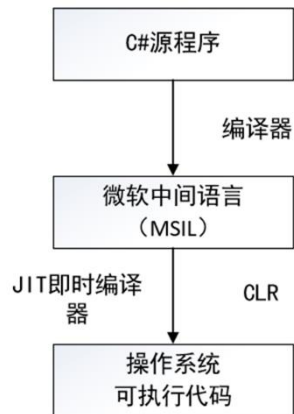


图 1-1 C#源程序的编译

微软公司已经为 C#语言推出了两个开发工具：Visual Studio .NET 继承开发环境 IDE（Integrated Development Environment）和 Microsoft .NET Framework 软件开发包 SDK（Software Development Kit）。SDK 是一个免费的 .NET 平台上的 C#编译器 csc；.NET 集成开发环境 IDE 提供了更优秀的多种编程语言的编辑和编译统一开发平台，它是一个快速开发企业级 Web 应用程序以及高性能桌面应用的工具。

1.2 C#的面向对象特性

C#中的面向对象（OO，Object-Oriented）特性可以分为三个层次：（1）初级特性，OO 最基本的概念，即类和对象；（2）中级特性，OO 最核心的概念，即封装、继承和多态；（3）高级特性，由初级特性和中级特性引出的一些问题，如构造函数的使用、覆盖的规则、静态变量和函数等。

其中，面向对象技术的三个核心概念：（1）封装，将数据和操作组合到一起，并决定哪些数据和操作对外是可见的；（2）继承，父类中的变量和行为，子类可以同样使用，本质是代码重用；（3）多态，由继承引出的一种机制，父类型的引用变量可以指向子类型的对象。

封装把对象的所有组成部分组合在一起，如图 1-2 所示，有三个作用：（1）隐藏类的实现细节，使用方法将类的数据隐藏起来；（2）迫使用户去使用同一个界面去访问数据，定义程序如何引用对象的数据，控制用户对类的修改和访问数据的程度；（3）使代码更好维护，类的内部实现改变，对外接口可以不变。

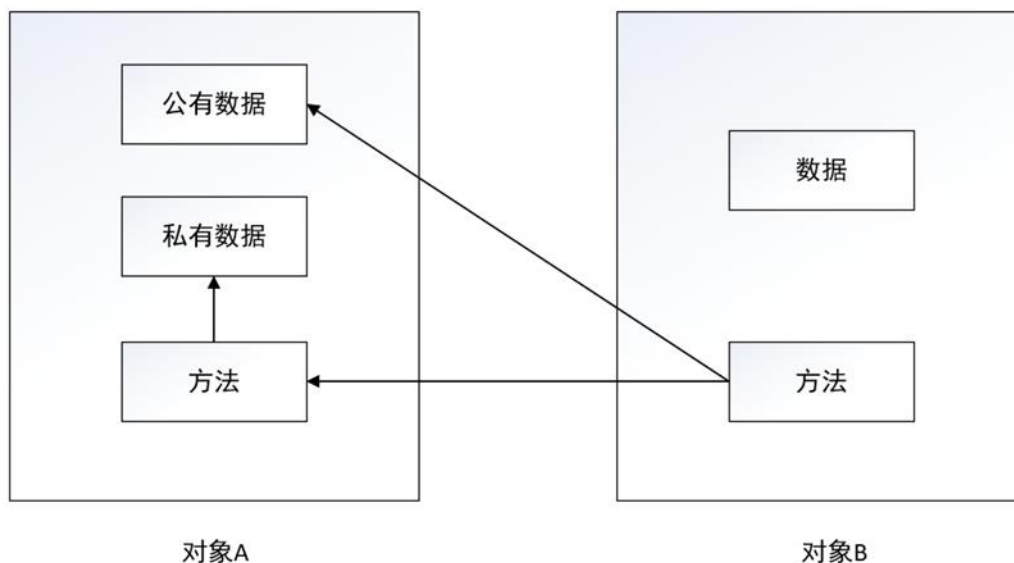


图 1-2 C#中对象的封装

继承提供了创建新类的一种方法，继承对开发者来说就是代码共享（如图 1-3 所示），有三个作用：（1）通过继承创建的子类是作为另一个类的扩充或修正所定义的一个类；（2）子类从超类(父类)中继承所有方法和变量；（3）子类和超类之间是特化与范化的关系。

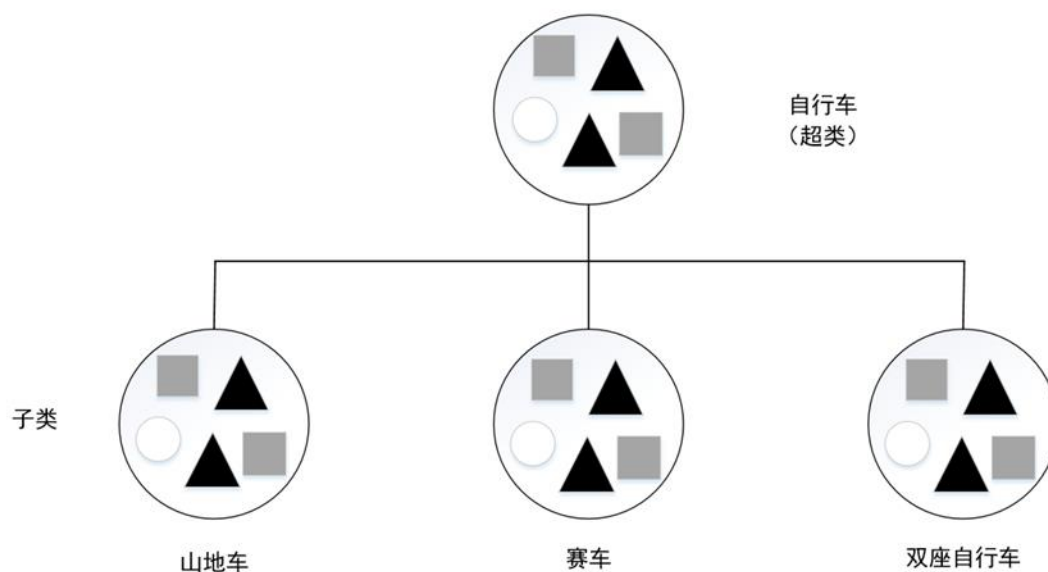


图 1-3 C#中类的继承

C#中的继承主要有以下 3 种特性：

（1）继承的可传递性

如果 C 从 B 中派生，B 又从 A 中派生，那么 C 不仅继承了 B 中声明的成员，同样也继承了 A 中的成员。

（2）继承的单一性

继承的单一性是指派生类只能从一个基类中继承，不能同时继承多个基类。C#不支持类的多重继承，也就是说儿子只能有一个亲生父亲，不能同时拥有多个

亲生父亲。C#主要通过接口实现多重继承。

(3) 继承中的访问修饰符

C#中的访问修饰符有 public、protected、private、internal 和 protected internal 共 5 种，可以使用这些访问修饰符指定可访问性级别，在继承时各个访问修饰符的访问权限如表 1-1 所示。

表 1-1 C#中的访问修饰符

访问性修饰符	类内部	派生类
public	访问不受限制	不受限制
protected	访问仅限于包含类或从包含类派生的类型	可以访问
internal	访问仅限于当前项目	可以访问
protected internal	访问仅限于从包含类派生的当前项目或类型	可以访问
private	访问仅限于包含类型	不可访问

基类中的成员如果用 public 修饰，任何类都可以访问；如果用 private 修饰，它将作为私有成员，只有类本身可以访问，其他任何类都无法访问。在 C#中，我们使用 protected 修饰符，使用这个访问修饰符的成员可以被其派生类访问，而不允许其他非派生类访问。

多态性是指不同的对象收到相同的消息时，会产生不同动作。C#支持两种类型的多态性：

(1) 编译时的多态性是通过重载和覆盖(/new)来实现的，系统在编译时，根据传递的参数个数、类型信息决定实现何种操作。

重载是在同一个作用域内发生（比如一个类里面），定义一系列同名方法，但是方法的参数列表不同，就是签名不同，签名由方法名和参数组成。能通过传递不同的参数来决定到底调用哪一个同名方法。注意返回值类型不同不能构成重载，因为签名不包括返回值。

基类中的方法不声明为 virtual（默认为非虚方法）时，在派生类中声明与基类同名方法时，需使用 new 关键字，以隐藏基类同名方法。

(2) 运行时的多态性是指在运行时根据实际情况决定实行何种操作，C#中运行时的多态性通过虚函成员实现(virtual/override)。

基类方法中使用 virtual 关键字声明的方法和派生类中使用 override 关键字声明的方法名称相同，参数列表也相同，就是基类方法和派生类方法的签名相同，实现了派生类重写基类中的同名方法。

1.3 C#中的类库和控件

类库(Class Library)是一个面向对象的可重用类型集合，这些类型包括接口和类。一个大的应用程序往往会划分成很多模块，而每一个功能模块可能由不同的开发人员来开发，每一个功能模块生成一个类库，类库可以单独开发、单独编译，甚至单独调试和测试。当所有的类库开发完成后，把它们组合在一起就得到了完成的应用系统。自定义类库可以对外提供一些调用的接口和方法等，但是不对外提供详细的代码，这样就保证了代码的安全性。

控件主要指提供或实现了用户界面功能的类库。在 Windows 窗体控件中，.NET Framework 为控件提供的基类是 System.Windows.Forms.Control，C#中

所有的控件都直接或间接从这个类继承。开发者也可以根据应用的需要开发自定义控件，从而提供更好的复用功能。

2 演示实例

2.1 “HelloWorld！”程序演示

本例实现在 Windows 控制台输出“Hello World!”的功能。具体步骤如下：

（1）鼠标右键以管理员身份运行“Visual Studio”，选择菜单栏【文件】→【新建】→【项目】，在弹出的【新建项目】对话框左侧栏选择【Visual C#】→【Windows 桌面】模板，中间栏单击选择【控制台应用】，修改下侧栏的新建项目“名称”为“HelloWorld”，“位置”为新建项目的物理存储位置，“解决方案名称”为“Test1”，点击【确定】按钮，如图 1-4 所示。

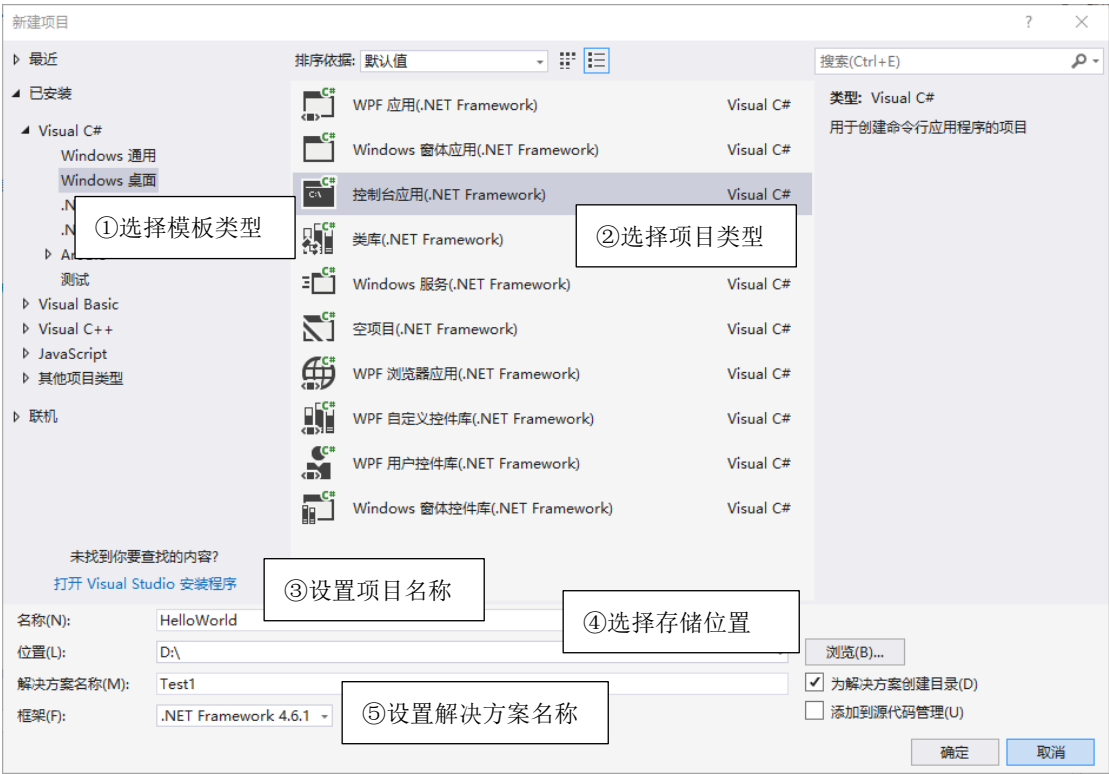


图 1-4 新建控制台应用项目

（2）在解决方案资源管理器中单击并打开 Program.cs 源文件，在 Main() 函数内添加代码行(如图 1-5 所示)： Console.WriteLine("Hello World!");

源文件 Program.cs 文件的内容如下：

Program.cs	功能：输出“HelloWord!”
<pre>using System; //导入System命名空间 using System.Collections.Generic; //导入System.Collections.Generic命名空间 using System.Linq; //导入System.Linq命名空间 using System.Text; //导入System.Text命名空间 using System.Threading.Tasks; //导入System.Threading.Tasks命名空间</pre>	

```

namespace HelloWorld//声明命名空间HelloWorld
{
    class Program//声明类Program
    {
        //程序入口点，Main 的返回类型为 void
        static void Main(string[] args)
        {
            //控制台的 WriteLine() 方法用于显示输出结果
            Console.WriteLine("Hello World!");
        }
    }
}

```

C#程序一般包括以下几部分：（1）名称空间的引用，使用 using 关键字指明引用的名称空间；（2）名称空间的声明，使用 namespace 关键字声明名称空间；（3）类，使用 class 关键字声明类；（4）Main() 方法，Main() 方法是 C# 程序的入口。一个文件中可以有 1 个或者多个类。C#所有语句都以分号“;”结束。

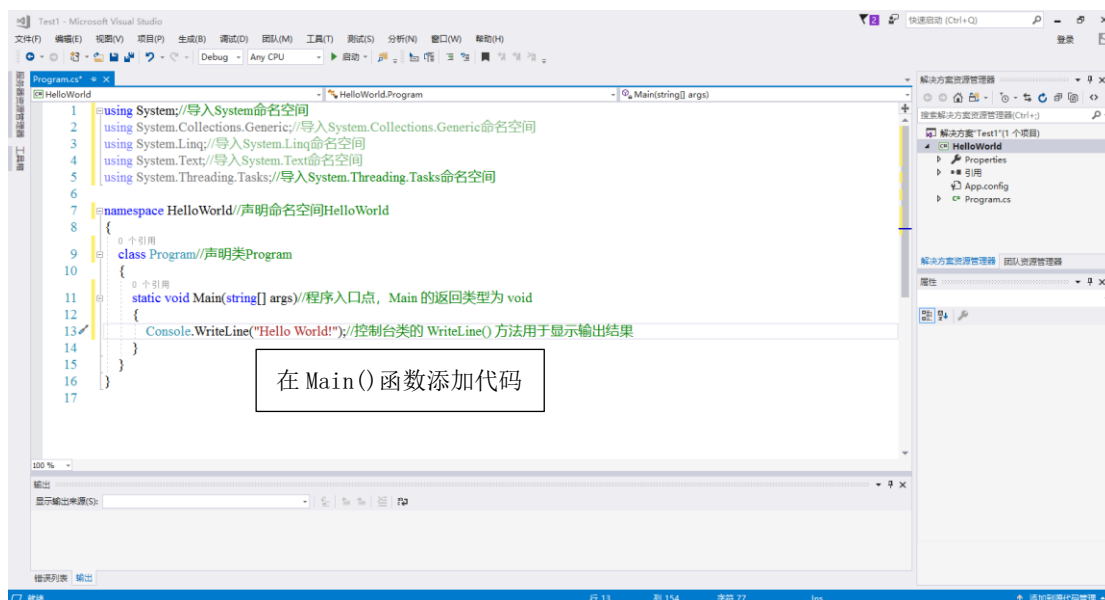


图 1-5 修改 Main 函数

（3）编译生成 C#程序，选择菜单栏目【生成】→【生成解决方案】，该过程将编译项目中包括的所有文件，编译结果显示在【输出】窗口中。结果显示“生成：成功 1 个，失败 0 个，最新 0 个，跳过 0 个”，则说明已经成功生成应用程序，如图 1-6 所示，然后即可执行该应用程序。

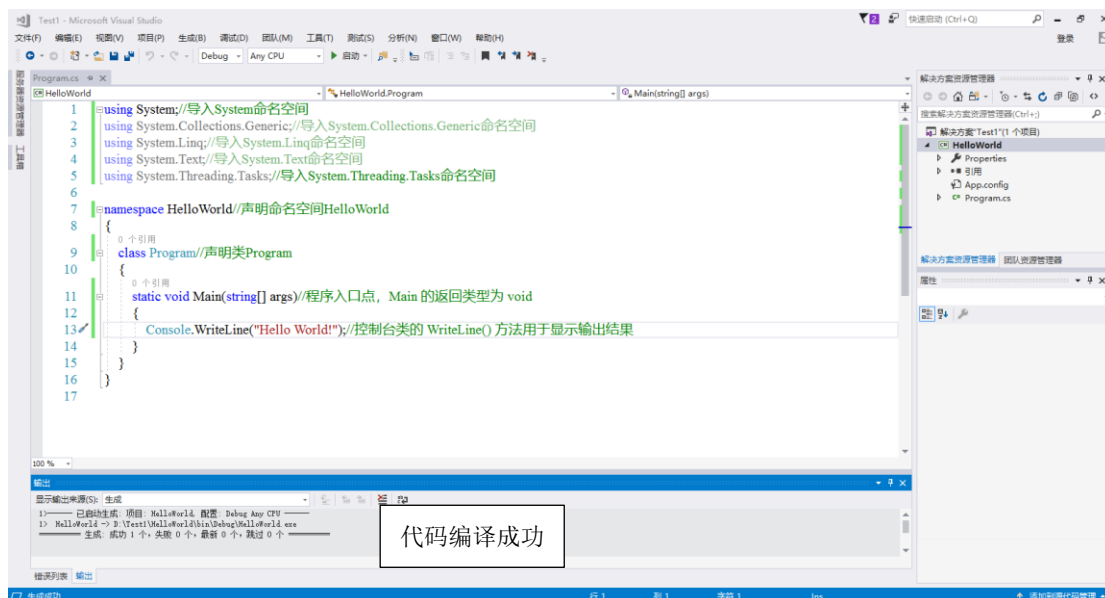


图 1-6 C#程序的编译生成

(4) 选择菜单栏【调试】→【开始执行（不调试）】，代码编译并生成应用程序后，可在弹出的控制台窗口可以看到程序运行结果，如图 1-7 所示。

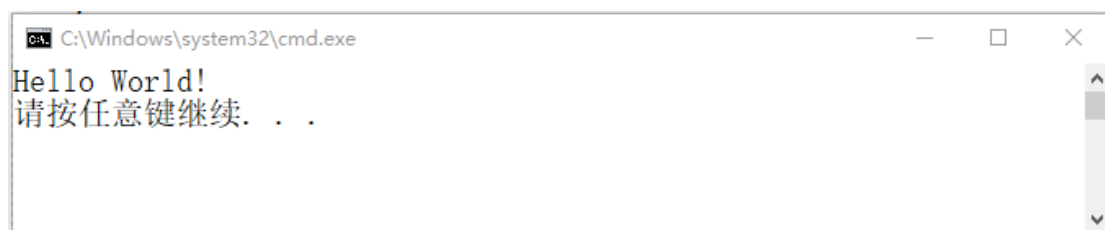


图 1-7 HelloWorld 程序运行结果

(5) 在“解决方案资源管理器”中选中项目“HelloWorld”右键选择【在文件资源管理器中打开文件夹】。可以看到 Visual Studio .NET 创建了一个与项目 HelloWorld 同名的文件夹和一个 HelloWorld.csproj 项目文件。HelloWorld 文件夹包含文件 bin, obj, Properties 三个文件夹和其他源文件如 Program.cs（默认主程序源代码）。Properties 文件夹下有文件 AssemblyInfo.cs，主要用来设定生成的有关程序集的常规信息及动态链接库 DLL(Dynamic Link Library)文件的一些参数。bin 和 obj 这两个文件夹下都有一个 Debug 子目录，其中都包含可执行文件 HelloWorld.exe，如图 1-8 所示。

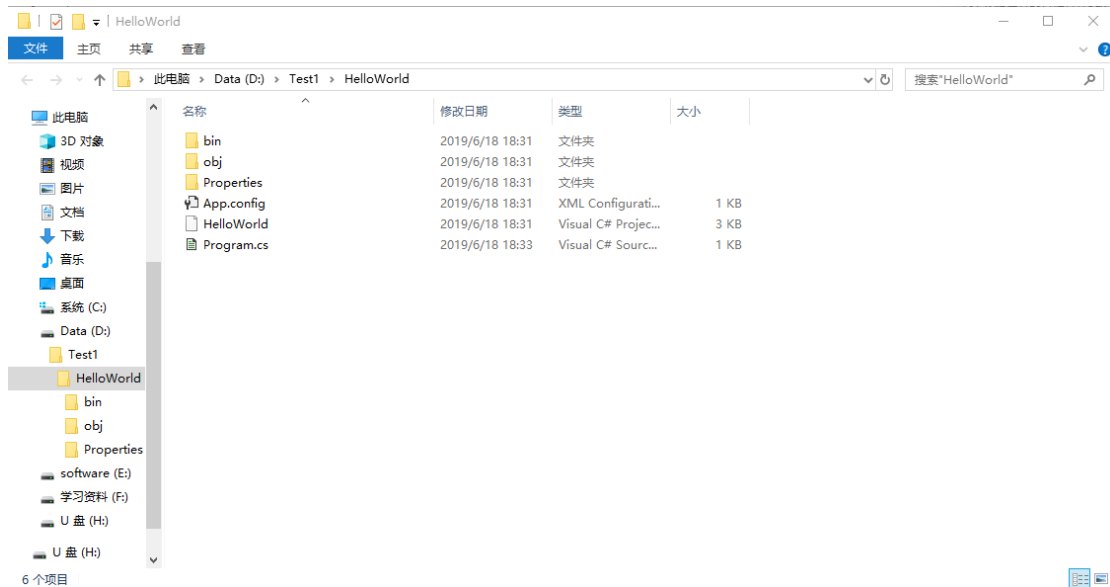


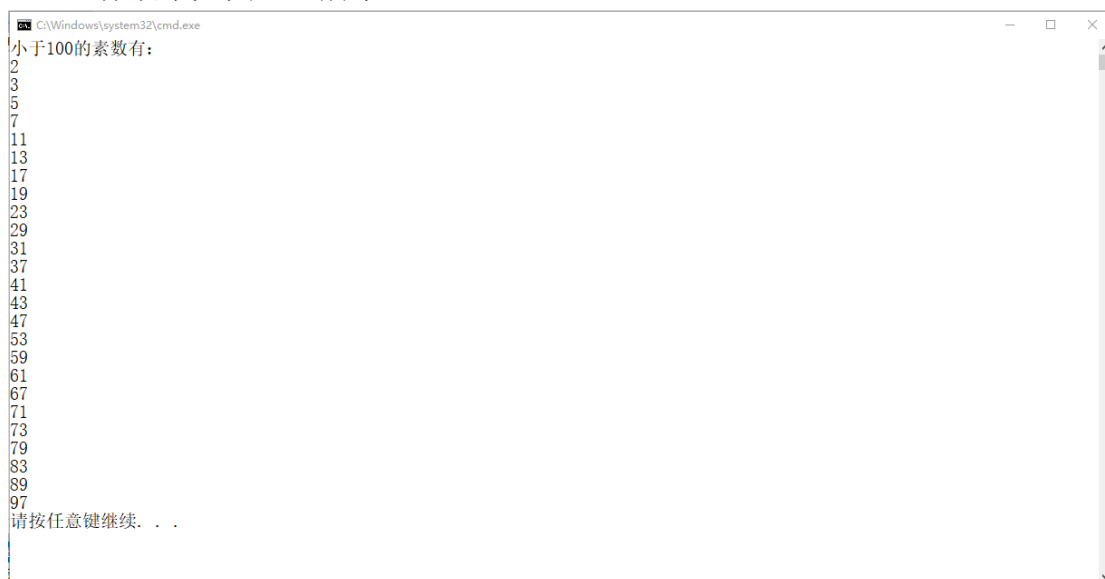
图 1-8 项目文件目录

2.2 查找 100 以内的素数

一个大于 1 的自然数，除了 1 和它自身外，不能被其他自然数整除的数叫做素数。在 Main() 函数下添加以下代码实现查找 100 以内的素数功能。

Program.cs (节选)	功能：查找 100 以内的素数
<pre> static void Main(string[] args) { Console.WriteLine("小于100的素数有："); for (int i = 2; i <= 100; i++) { bool isPrime = true; for (int j = 2; j < Math.Sqrt(i); j++) { //如果能被小与它的数整除，则不是素数 if (i % j == 0) { isPrime = false; break; } } if (isPrime) { //输出素数 Console.WriteLine("{0}", i); } } } </pre>	

运行效果如图 1-9 所示：



```
C:\Windows\system32\cmd.exe
小于100的素数有:
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
53
59
61
67
71
73
79
83
89
97
请按任意键继续. . .
```

图 1-9 输出 100 以内的素数程序运行结果

2.3 小于 100 范围内验证哥德巴赫猜想

小于 100 范围内验证哥德巴赫（Goldbach）猜想（即是否每个大于 2 的偶数都可写成两个素数之和？），在 Main()函数下面添加以下代码实现验证哥德巴赫猜想功能。

Program.cs(节选)	功能：小于 100 范围内验证哥德巴赫猜想
<pre>static void Main(string[] args) { Console.WriteLine("大于2的偶数都可写成两个素数之和?"); List<int> liPrime = new List<int>(); //不定长素数数组List for (int i = 2; i <= 100; i++) { bool isPrime = true; for (int j = 2; j < Math.Sqrt(i); j++) { if (i % j == 0) { isPrime = false; break; } } if (isPrime) { liPrime.Add(i); //找到素数 } } }</pre>	

```

if (i > 2 && i % 2 == 0)//大于2的偶数
{
    bool bGuess = false;
    foreach (int m in liPrime)
    {
        foreach (int n in liPrime)
        {
            if (m + n == i)
            {
                bGuess = true;
                Console.WriteLine("{0}={1}+{2}", i, m, n);
                break;
            }
        }
        if (bGuess)
        {
            break;
        }
    }
    if (!bGuess)
    {
        Console.WriteLine("The guess is false.");
    }
}
}
}

```

运行效果如图 1-10 所示：

```

C:\Windows\system32\cmd.exe
44=3+41
46=3+43
48=5+43
50=3+47
52=5+47
54=7+47
56=3+53
58=5+53
60=7+53
62=3+59
64=3+61
66=5+61
68=7+61
70=3+67
72=5+67
74=3+71
76=3+73
78=5+73
80=7+73
82=3+79
84=5+79
86=3+83
88=5+83
90=7+83
92=3+89
94=5+89
96=7+89
98=19+79
100=3+97
请按任意键继续...

```

图 1-10 小于 100 范围内验证哥德巴赫猜想程序运行结果

3 实验目的

熟悉 Visual Studio .NET 集成开发环境 IDE 及程序调试过程，了解 C# 中自定义类库、控件的开发和使用。

4 实验内容

- (1) 利用 C# 接口编程制作一个类库，并且在 Windows 窗体程序中使用该类库的 DLL，以及捕获处理交互输入中的异常；
- (2) 用 C# 制作自定义控件和用户控件，并在 Windows 窗体程序中使用控件；
- (3) 使用 Visual Studio .NET 中的 C# 程序调试工具。

5 实验步骤

5.1 制作类库并使用

用 Visual C# 生成的 DLL 文件在程序设计中更多的表现为类(Class)或者类库(Class Library)。类库制作和调用通常包括以下几个步骤：

- (1) 基于类库模板的新建项目，开发环境自动生成包含对命名空间定义等的代码。新建项目，使用【Visual C#】→【Windows 桌面】→【类库】模板。项目文件名称改为 ClsLib，并且通过“浏览”按钮选择项目要存放的目录，将其添加到 Test1 解决方案中，然后单击“确定”按钮，如图 1-11 所示。

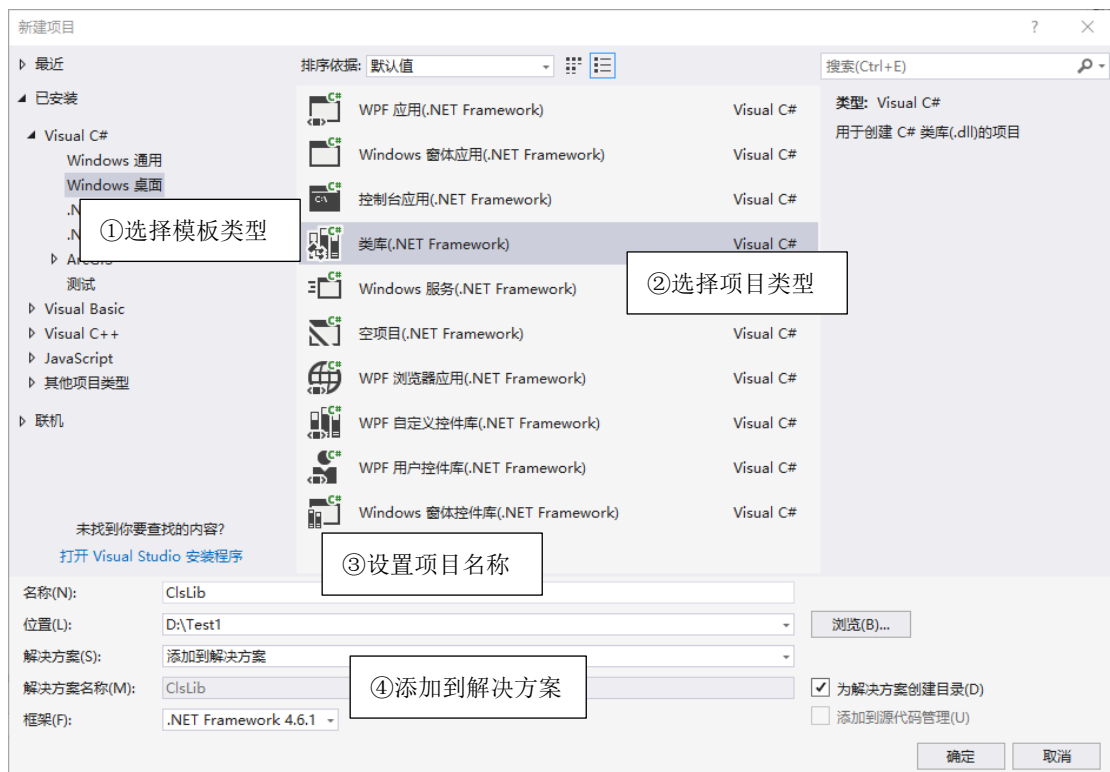


图 1-11 新建 windows 桌面类库项目

查看项目文件夹和它所包含的文件，此时项目文件中的解决方案浏览器已经添加了两个 C# 的源文件。分别是 AssemblyInfo.cs 和 Class1.cs。先删除类 Class1 带代码段，再重命名“Class1.cs”为“ClsLib.cs”，如图 1-12 所示。

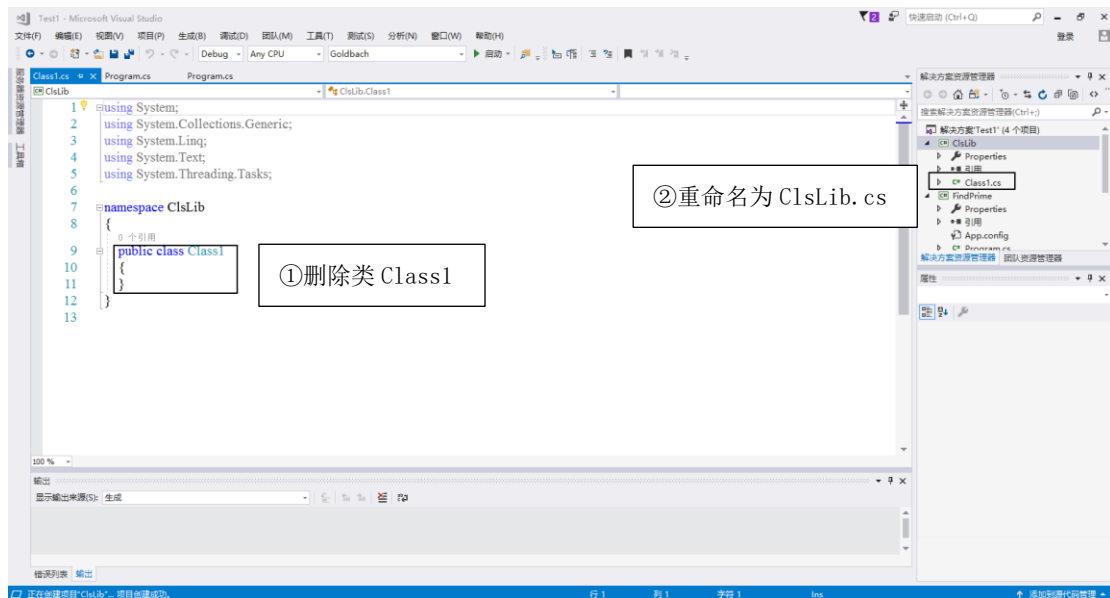


图 1-12 源文件重命名为 ClsLib.cs

(2) 在 ClsLib 命名空间中添加一个接口 ICalculateAreaAndVolume(含有 Area 和 Volume 两个只读属性)，如下：

ClsLib.cs (节选)	添加接口 ICalculateAreaAndVolume
<pre>namespace ClsLib { public interface ICalculateAreaAndVolume { double dArea { get; } double dVolume { get; } } }</pre>	

接口用来描述一种程序的规定，可定义属于任何类或结构的一组相关行为，定义接口的语法如下：

```
<访问修饰符> interface 接口标识符 [:基接口列表]
{
    //接口主体
}
```

接口成员访问权限为 **public**，但不能加访问修饰符；接口成员不能有定义；接口的成员必须是方法，属性，事件或索引器，不能包含常数、字段、运算符、实例构造函数、析构函数或类型，也不能包含任何种类的静态成员。

(3)在项目中添加已一组类的定义，并添加类的方法、属性、事件和字段等，实现计算球、圆锥、圆柱的表面积和体积的功能，代码如下：

ClsLib.cs (节选)	功能：计算球，圆锥，圆柱的表面积和体积
<pre>namespace ClsLib { public interface ICalculateAreaAndVolume { double dArea { get; } double dVolume { get; } } public class Circle //基类，不需要计算表面积和体积 { protected double radius; public Circle(double r) { radius = r; } public double dRadius { get { return radius; } set { radius = value; } } } public class Sphere : Circle, ICalculateAreaAndVolume//球体类 {</pre>	

```

public Sphere(double r) : base(r)
{
}

public double dArea
{
    get { return (4 * Math.PI * radius * radius); }
}

public double dVolume
{
    get { return (4 * Math.PI * Math.Pow(radius, 3) / 3); }
}
}

public class Cylinder : Circle, ICalculateAreaAndVolume//圆柱类
{
    private double height;//高度字段
    public Cylinder(double r, double h) : base(r)
    {
        height = h;
    }

    public double dArea
    {
        get
        {
            return (2 * Math.PI * radius * radius
                + 2 * Math.PI * radius * height);
        }
    }

    public double dVolume
    {
        get { return (Math.PI * radius * radius * height); }
    }

    public double dHeight
    {
        get { return height; }
        set { height = value; }
    }
}

public class Cone : Circle, ICalculateAreaAndVolume//圆锥类
{
    private double height;//高度字段
    public Cone(double r, double h) : base(r)
    {
        height = h;
    }
}

```

```

public double dArea
{
    get
    {
        return (Math.PI * radius * (radius
+ Math.Sqrt(height * height + radius * radius)));
    }
}

public double dVolume
{
    get { return (Math.PI * radius * radius * height / 3); }
}

public double dHeight
{
    get { return height; }
    set { height = value; }
}
}
}

```

(4) 点击菜单栏目【生成】→【生成解决方案】，编译工程文件生成类库文件，该类库文件会在项目文件夹的 bin\debug 目录里，文件扩展名是 DLL。

(5) 在客户端，调用我们刚才生成的类库。新建一个 Windows 窗体应用程序，选择【Visual C#】→【Windows 桌面】→【Windows 窗体应用】模板。通过这个应用程序来调用我们的类库，程序名为 Tester，如图 1-13 所示。

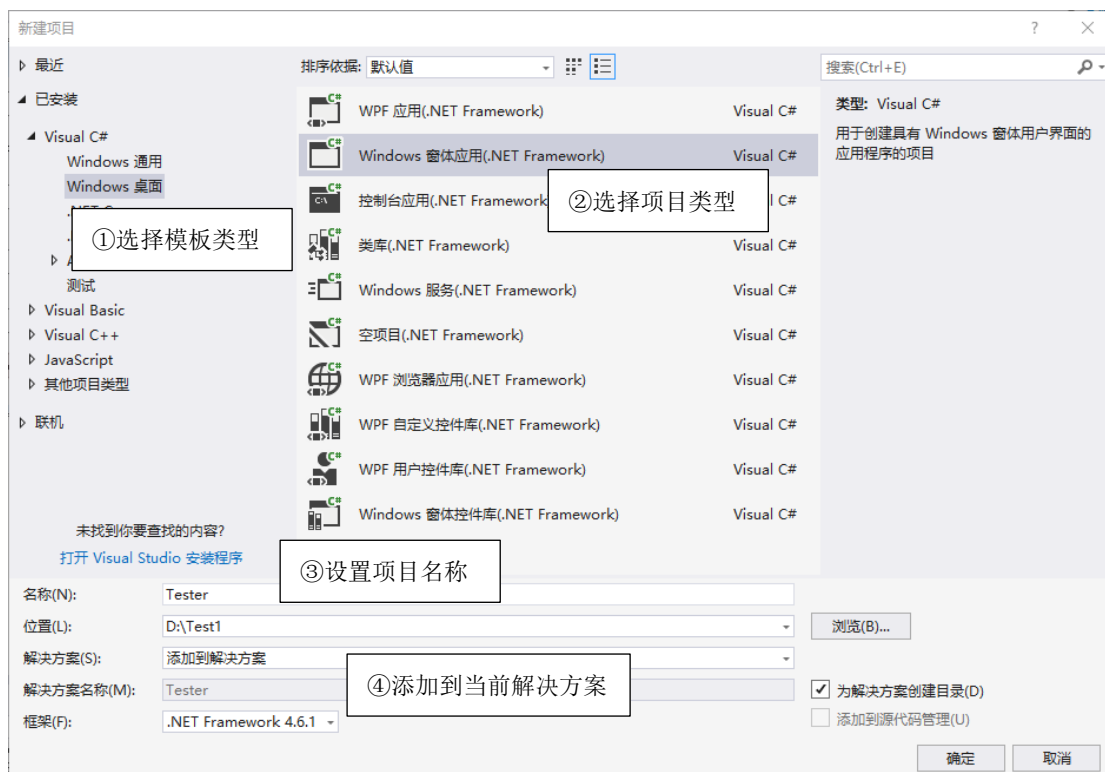


图 1-13 新建 Windows 窗体应用项目

(6) 添加对类库的引用。选择解决方案管理器中 Tester 项目的【引用】，右键【添加引用】命令，浏览到刚才生成的 DLL，然后单击“确定”按钮。添加引用向导将会把引用 ClsLib.dll 加到当前的 Tester 项目文件中。如图 1-14 所示。

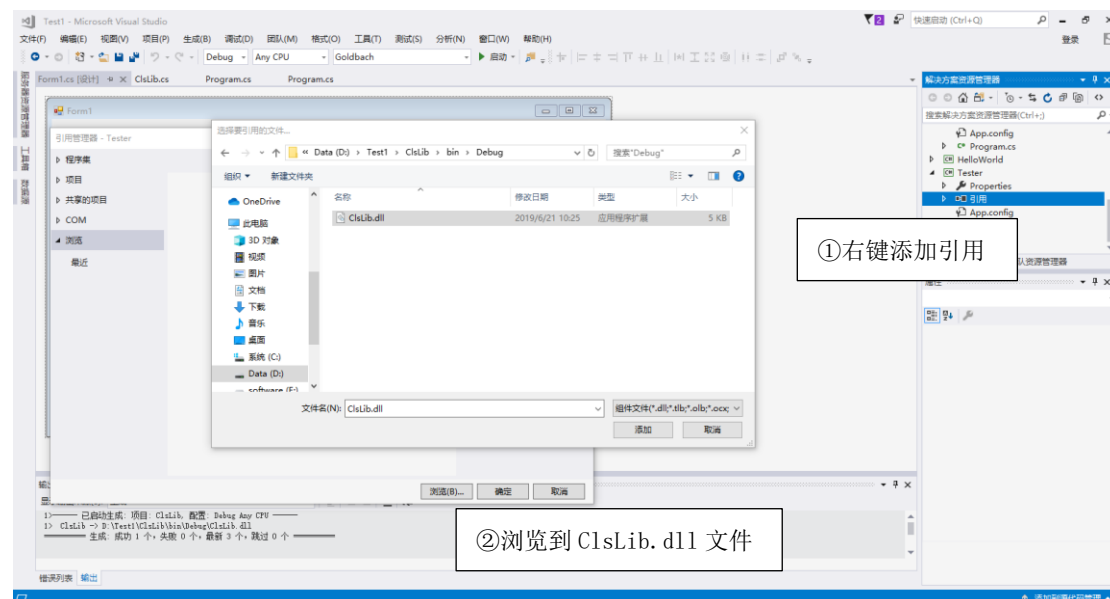


图 1-14 引用 ClsLib.dll 加到当前项目中

(7) 在窗体上添加若干 Label 控件和 TextBox 控件，以及一个 Button 控件，分别修改它们的 Text 属性值和 Name 属性值等，如图 1-15，1-16 所示。

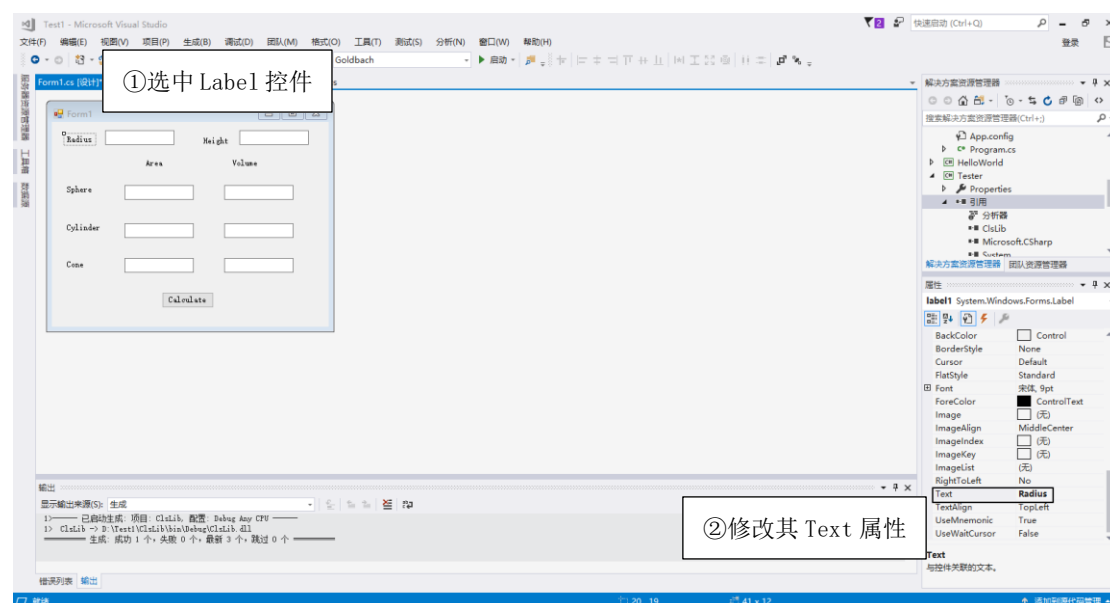


图 1-15 修改 Label 控件的 Text 属性

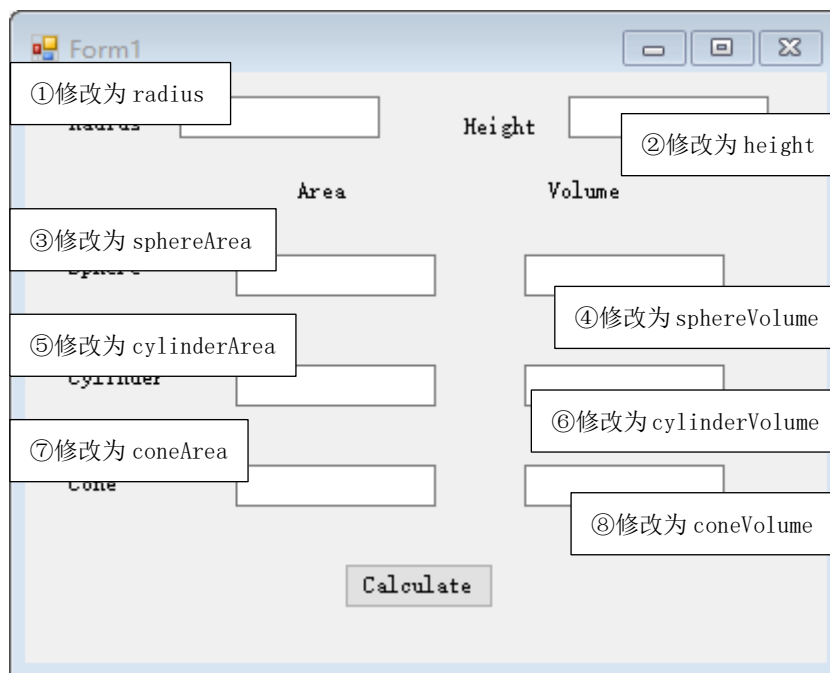


图 1-16 修改 TextBox 的 Name 属性

(8) 在 Form1.cs 开头部分导入 ClsLib 命名空间: `using ClsLib;`

双击“Calculate”按钮, 添加按钮单击事件响应代码, 调用类库中接口的方法和属性;

Form1.cs (节选)	功能: “Calculate” 按钮的单击事件响应
<pre> private void button1_Click(object sender, EventArgs e) { ICalculateAreaAndVolume[] c = new ICalculateAreaAndVolume[3]; try { double r = Convert.ToDouble(radius.Text); double h = Convert.ToDouble(height.Text); if (r <= 0 h <= 0) { throw new Exception("半径或高度不能小于等于0"); } c[0] = new Sphere(r); sphereArea.Text = c[0].dArea.ToString(); sphereVolume.Text = c[0].dVolume.ToString(); c[1] = new Cylinder(r, h); cylinderArea.Text = c[1].dArea.ToString(); cylinderVolume.Text = c[1].dVolume.ToString(); c[2] = new Cone(r, h); coneArea.Text = c[2].dArea.ToString(); coneVolume.Text = c[2].dVolume.ToString(); } } </pre>	

```

catch (Exception err)
{
    MessageBox.Show(err.Message + "\n"
        + err.Source + "\n"
        + err.TargetSite + "\n"
        + err.StackTrace);
}
}

```

(9) 在解决方案管理器中选中 Tester 项目右键选择【设置为启动项目】，编译并运行应用程序 Tester，至此完成了一个类库从制作到调用的全部过程，如图 1-17 所示。

	Area	Volume
Sphere	1256.63706143592	4188.79020478639
Cylinder	1256.63706143592	3141.59265358979
Cone	758.447559174816	1047.1975511966

图 1-17 Windows 窗体应用程序运行结果

5.2 制作自定义控件并使用

Visual Studio .NET 中开发的控件有两种类型：(1) 自定义控件，通过调用 Paint 事件中的 Graphics 对象来显示用户界面 UI (User Interface) 的控件，自定义控件通常从 Control 类派生；(2) 用户或复合控件，由其他控件组成的控件，用户控件从 UserControl 类派生，使用 Windows 窗体设计器可以创建用户控件。

下面制作一个按钮控件，该控件通过处理 OnPaint 绘制事件显示其 Color 颜色属性和 Transparent 透明度属性的值，以达到颜色和透明度渐变的效果。为了创建此控件和处理 OnPaint 事件，必须创建一个从 Button 按钮类派生的类，并创建一个重写 OnPaint 事件的方法；如果不改写 OnPaint，控件将无法自行绘制。具体步骤如下：

(1) 新建项目，使用【Visual C#】→【Windows 桌面】→【Windows 窗体控件库】模板。项目文件名称改为 CtrlLib，并且通过“浏览”按钮选择项目要存放的目录，然后单击“确定”按钮，如图 1-18 所示。

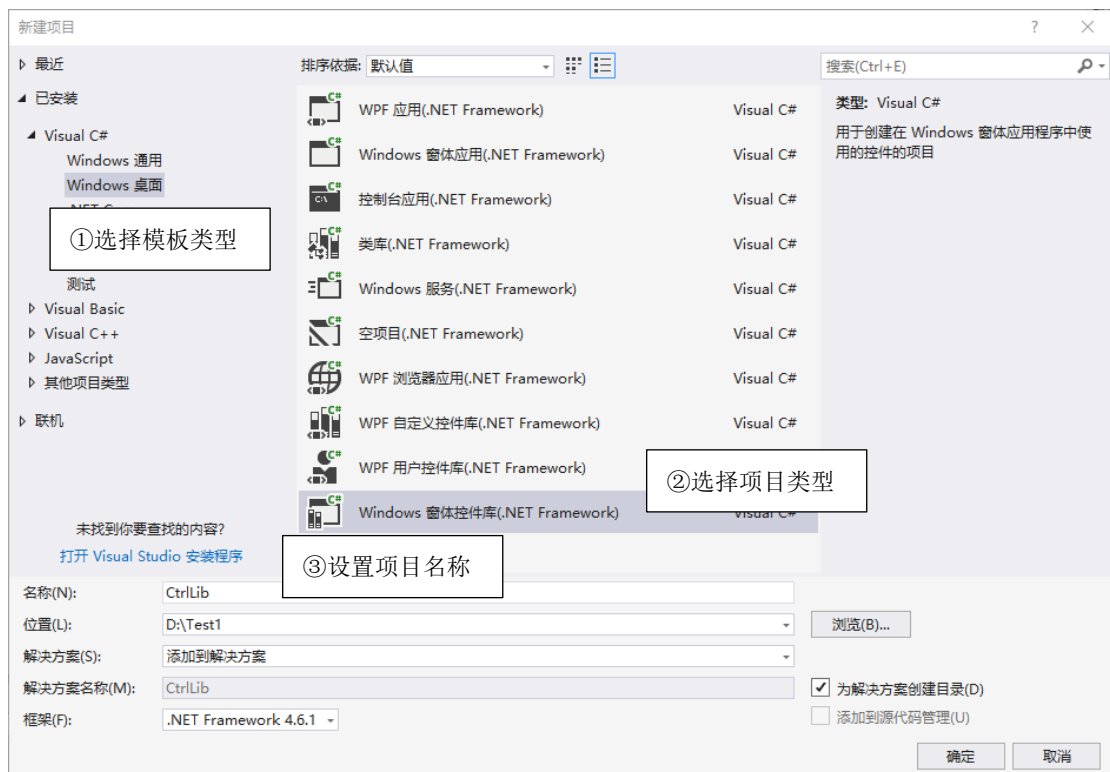


图 1-18 新建 windows 窗体控件库项目

(2) 打开项目后，从项目中删除默认生成的 UserControl1.cs, 如图 1-19 所示。

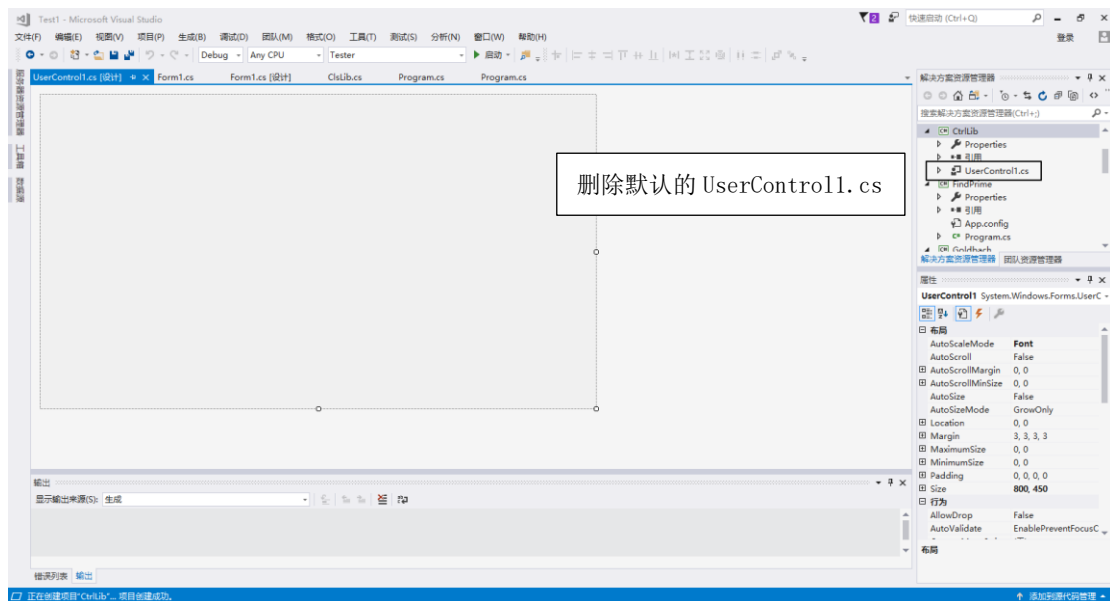


图 1-19 删除默认生成的 UserControl1.cs

(3) 从解决方案管理器选中项目“CtrlLib”，右键选择【添加新项】命令，对话框中选择【Visual C#】→【Windows Forms】→【自定义控件】。类命名为 ClrButton，然后单击【确定】按钮，一个新的用户控件就被添加到项目中，如图 1-20 所示。

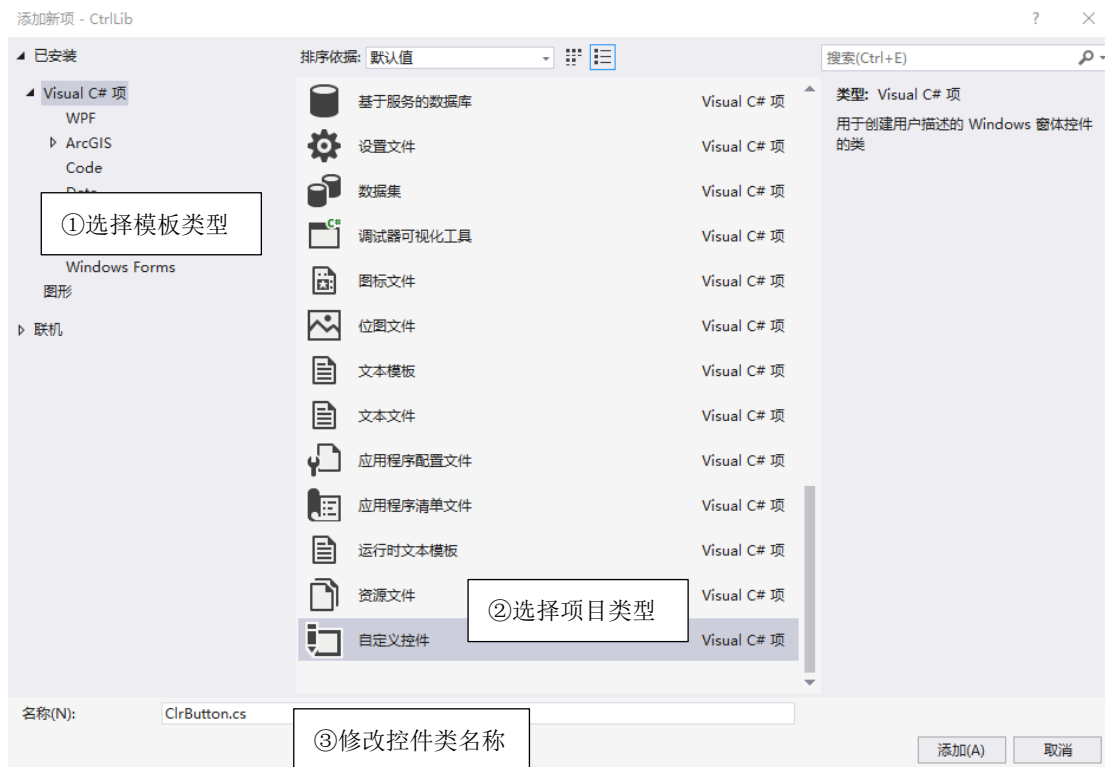


图 1-20 添加自定义控件新建项

(4) 因为要做的是按钮控件，必须改变源文件 ClrButton.cs 中类 ClrButton 的基类，将其由 `public class ClrButton: Control` 改为：`public class ClrButton: Button`。现在控件的基类是 Button 类。

(5) 给控件设置一组自定义的属性，分别为：`m_color1`、`m_color2`、`m_color1Transparent` 和 `m_color2Transparent`，用来调整按钮控件的颜色和透明度，并且通过关键字 `set` 设定属性，`get` 关键字来读取组件的属性值。在类中插入下面的代码来完成。

ClrButton.cs (节选)	功能：给控件建立 4 个设置属性
<pre> public partial class ClrButton : Button { public ClrButton() { InitializeComponent(); } private Color m_color1 = Color.LightBlue; // 第1个颜色 private Color m_color2 = Color.DarkRed; // 第2个颜色 private int m_color1Transparent = 50; // 第1个颜色透明度 private int m_color2Transparent = 50; // 第2个颜色透明度 public Color cuteColor1 { get { return m_color1; } set { m_color1 = value; Invalidate(); } } public Color cuteColor2 </pre>	

```

    {
        get { return m_color2; }
        set { m_color2 = value; Invalidate(); }
        // Invalidate() 为刷新窗口
    }

    public int cuteTransparent1
    {
        get { return m_color1Transparent; }
        set { m_color1Transparent = value; Invalidate(); }
    }

    public int cuteTransparent2
    {
        get { return m_color2Transparent; }
        set { m_color2Transparent = value; Invalidate(); }
    }

    protected override void OnPaint(PaintEventArgs pe)
    {
        base.OnPaint(pe);
    }
}

```

(6) 修改控件的 OnPaint 事件，并在这个方法中绘制控件。

ClrButton.cs (节选)	功能：实现控件的 Paint 事件
<pre> protected override void OnPaint(PaintEventArgs pe) { //调用基类OnPaint事件 base.OnPaint(pe); //创建两个半透明颜色 Color c1 = Color.FromArgb(m_color1Transparent, m_color1); Color c2 = Color.FromArgb(m_color2Transparent, m_color2); //构建线性渐变画刷对象 Brush b = new System.Drawing.Drawing2D.LinearGradientBrush(ClientRectangle, c1, c2, 10); //用渐变色填充控件矩形客户区 pe.Graphics.FillRectangle(b, ClientRectangle); //释放画刷资源 b.Dispose(); } </pre>	

(7) 编译控件。选择菜单栏【生成】→【生成解决方案】命令，生成控件。控件将会保存在该项目对应目录的 bin\Debug 子目录下。操作到此，完成制作控件的过程。

(8) 使用控件。打开 VS .NET 的项目 Tester，向“工具箱”中添加编译好的控件。选择菜单栏【工具】→【选择工具箱项】命令。点击浏览到 ClsLib.dll

文件，自定义控件 `ClrButton` 将出现在工具箱中，如图 1-21 所示。将 `ClrButton` 自定义控件拖到窗体上，制作一个清空按钮，将 `Text` 属性值和 `Name` 属性值改为“clear”，如图 1-22 所示，再把它属性值做一些改变 (`cuteColor1`, `cuteColor2`, `cuteTransparent1`, `cuteTransparent2`)，看看有什么变化。我们发现，按钮的颜色和透明度都可以根据需要进行调整，应用自己制作的按钮控件比使用普通按钮控件更加美观。

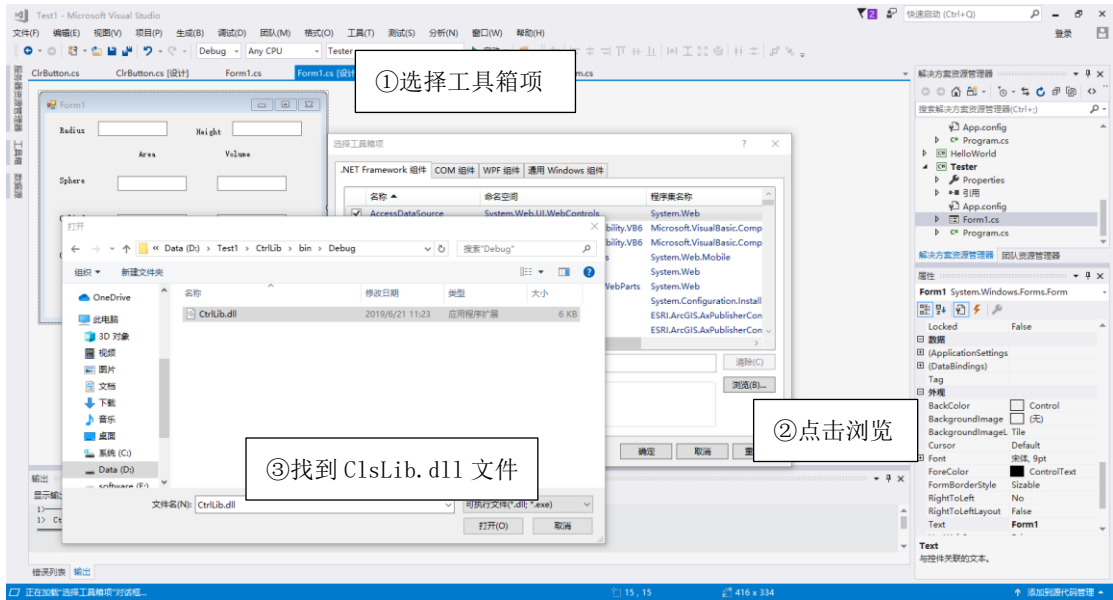


图 1-21 向“工具箱”中添加编译好的控件

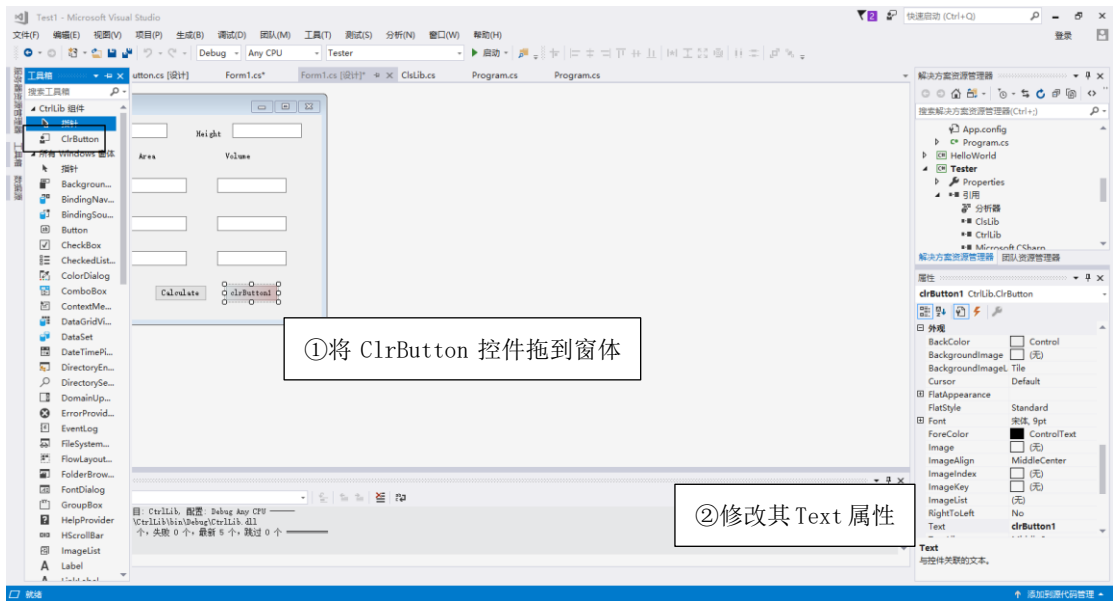


图 1-22 添加一个自定义 `ClrButton` 按钮

(9) 双击“clear”按钮，添加其单击事件相应代码，清空窗体上 `TextBox` 控件的 `Text` 内容。

Form1.cs (节选)	功能：实现按钮单击事件的清空功能
<pre>private void clear_Click(object sender, EventArgs e) { foreach (Control ctrl in Controls)</pre>	

```

//找出当前controls内的所有TextBox
{
    if (ctrl is TextBox)
    {
        ctrl.Text = "";
    }
}
}

```

(10) 右键设置 **Tester** 项目为启动项目，重新生成解决方案，运行程序即可，如图 1-23 所示。以上是建立和使用自定义控件的全部过程。

图 1-23 clear 控件清空效果

5.3 制作用户控件并使用

为了简化程序设计过程，我们经常需要开发一个复合的控件。创建了各控件后，只要将它们定义为一个复合控件，然后定制各控件的属性及事件响应，由此就简化了程序设计过程。具体步骤如下：

(1) 在解决方案管理器中选择项目“CtrlLib”，右键【添加新项】命令，选择【Visual C#】→【Windows Forms】→【用户控件】，命名为 CtrlCalculate.cs，如图 1-24 所示。

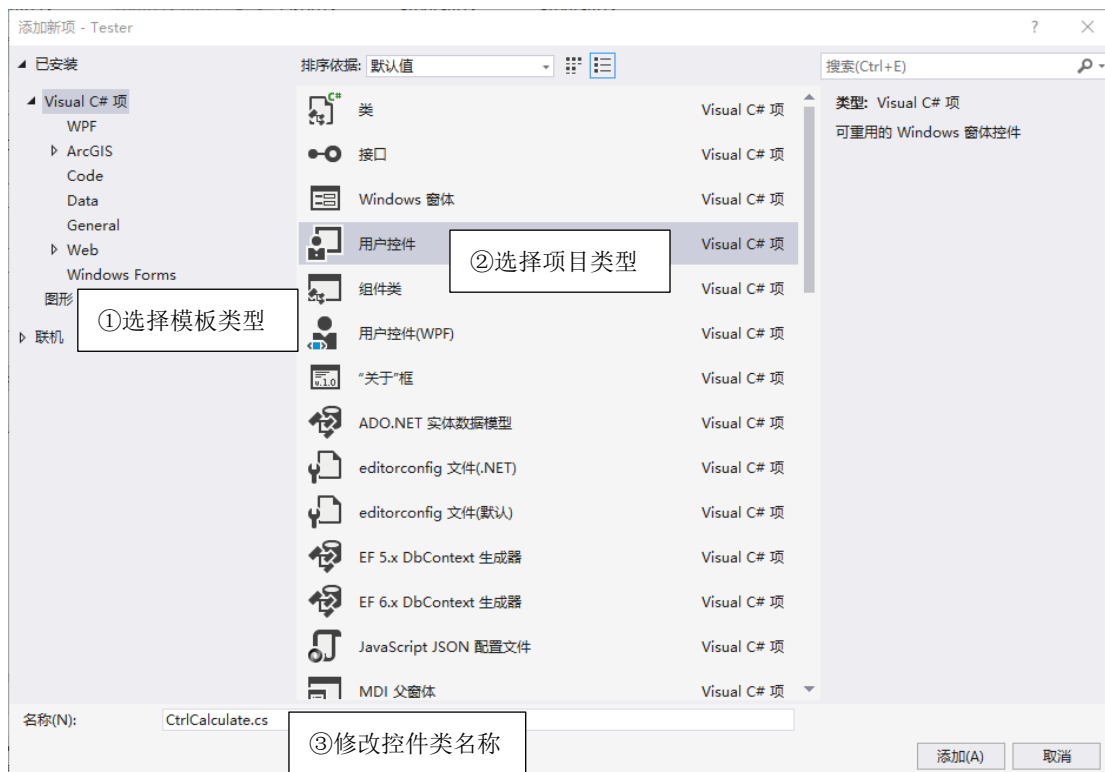


图 1-24 添加用户控件新建项

(3)参照第 5.2 中的 windows 窗体应用程序的控件布局制作一个复合控件，并代码实现, 如图 1-25 所示。

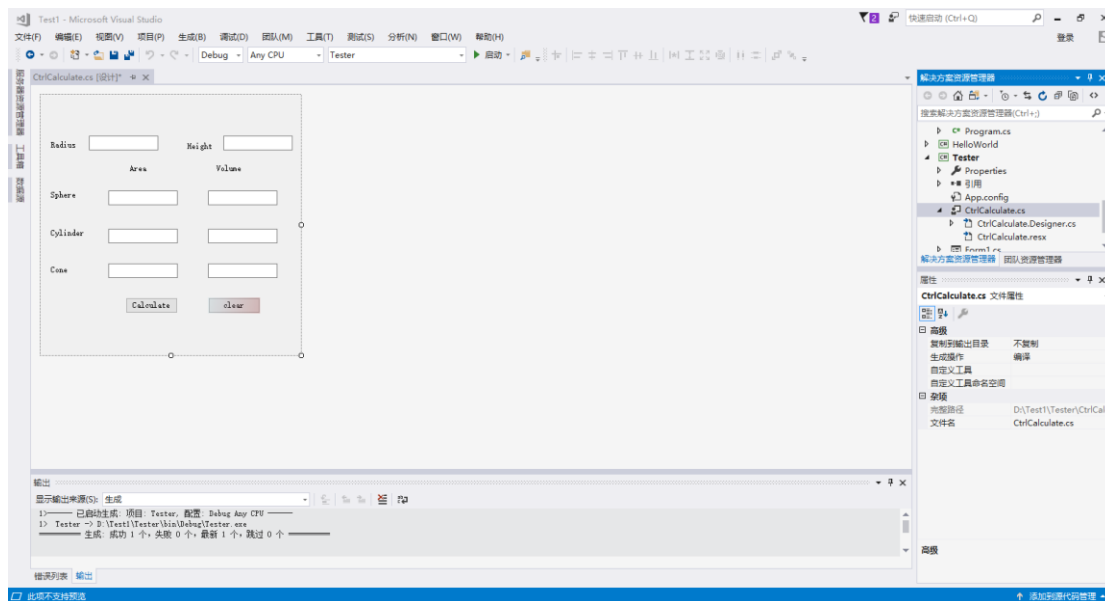


图 1-25 制作用户复合控件

(4) 在 Windows 窗体应用程序中使用复合控件。选择菜单栏【生成】→【生成解决方案】，编译成功后在 Tester 窗体应用程序中使用该复合控件，从【工具箱】中拖动 CtrlCalculate 控件到 Form1.cs 主窗体中，如图 1-26 所示。这样就完成了制作一个复合控件并使用的过程。

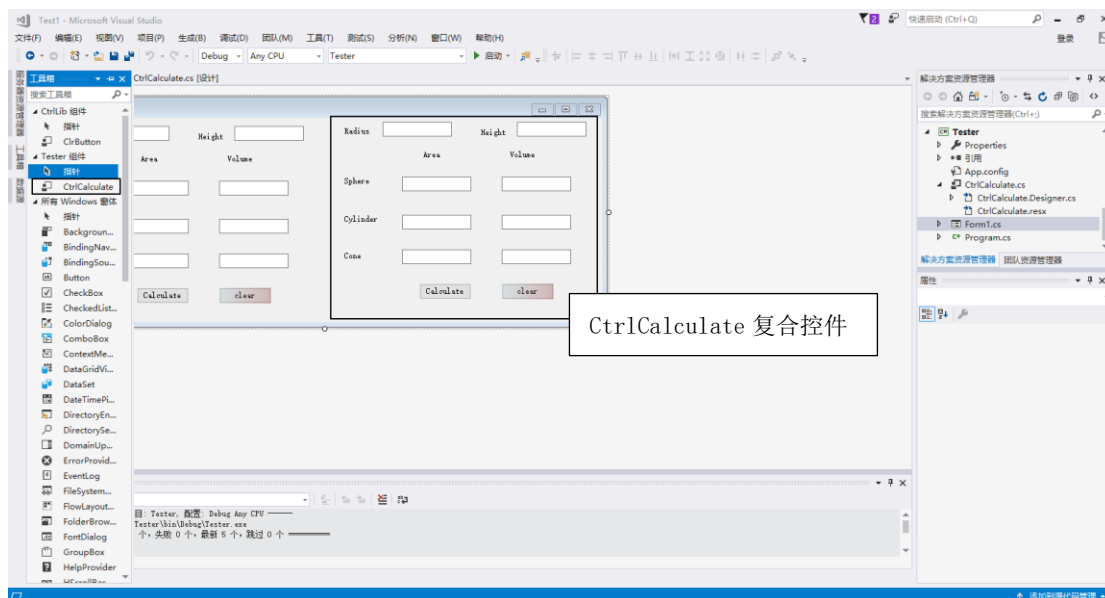


图 1-26 使用复合控件

(5) 设置项目间依赖关系并运行 Windows 窗体应用程序。选中项目“Tester”，右键选择【生成依赖项】→【项目依赖项】命令，设置项目“Tester”依赖“CtrlLib”和“ClsLib”，如图 1-27 所示。同样设置“CtrlLib”依赖“ClsLib”，重新编译项目“Tester”并运行之。

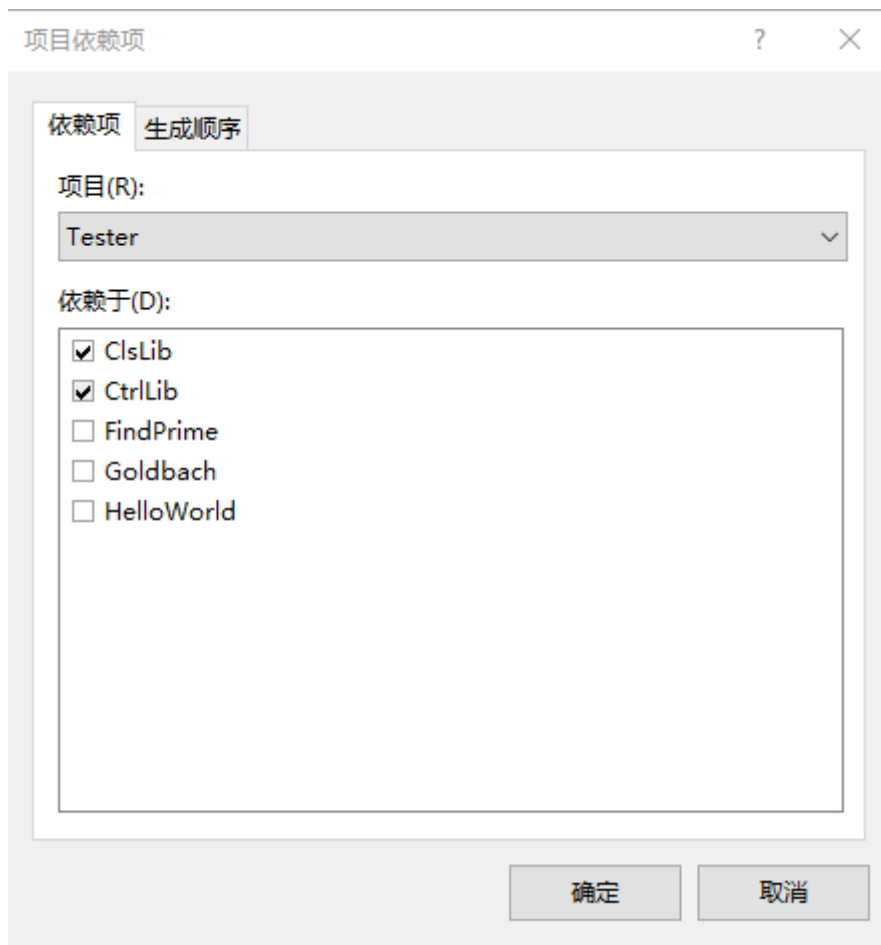


图 1-27 设置项目“Tester”的依赖项

5.4 C#程序的调试

开发的应用程序通常要安装到客户端的机器上，要保证应用程序必须无错误、无故障、可靠、稳健，在部署应用程序前必须先对其进行调试。而查找和排除错误或故障称为调试。如在计算机化的计费系统中，一旦系统发生故障，必须重新输入全部信息。因此，在事物处理过程中，系统要能显示错误消息。

错误的类型主要有：（1）**语法错误**，如缺少括号等，在编译时确定，也易于确定；（2）**逻辑错误**，一般是错误的算法导致错误结果、公式错误等，在执行过程中确定，难以调试；（3）**运行时错误**，如内存泄漏、以零作除数、异常等，在运行时确定，难以调试。

Visual Studio .NET 调试器的功能包括：跨语言调试、调试使用.NET 框架编写的应用程序以及 Win32 本机应用程序、加入正在运行的程序、调试多个程序等。.NET 集成开发环境有两种模式，Debug 模式和 Release 模式。通过调试器可以观察程序的运行时行为、跟踪变量的值、确定语义错误的位置、查看寄存器的内容、查看内存空间。

（1）设置断点。打开 FindPrime 项目代码，右键可在代码中插入“断点”，以便在特定行处暂停执行该程序。设置一个断点，程序执行到那一句就自动中断进入调试状态。在某行代码设置了断点，左侧会出现一个红点即断点，如图 1-28 所示。

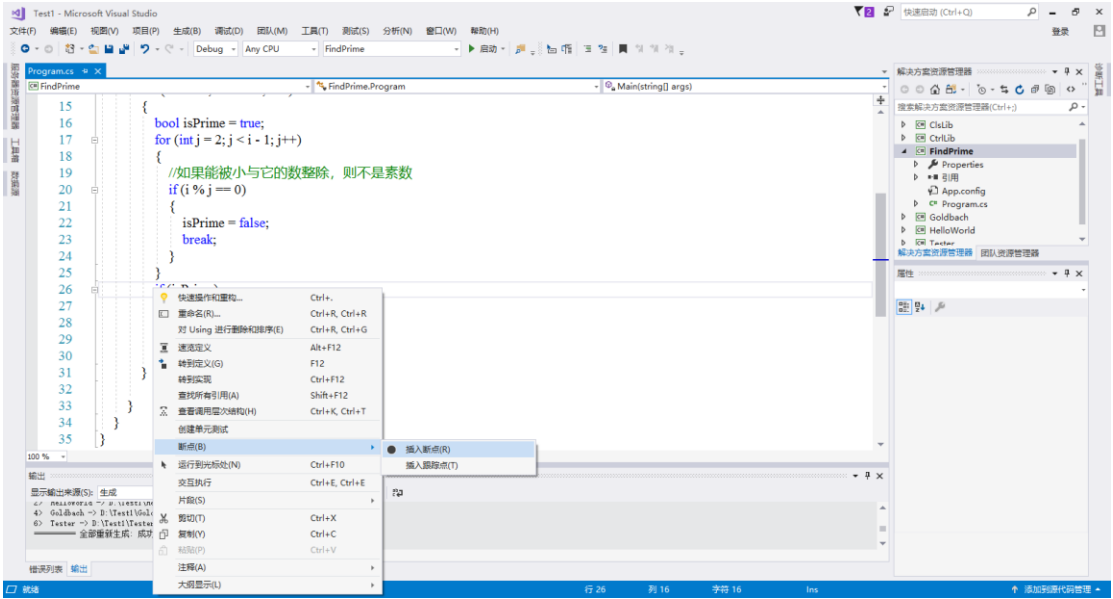


图 1-28 插入断点

程序的执行过程是连贯的，为了跟踪观察程序的运行状态，需要控制程序的运行过程，使得程序能够暂停在某些特定的位置，这种控制可以通过设置断点来实现。断点是程序暂停执行的地方，当程序运行到断点位置时，程序暂停执行，进入中断模式，程序设计者可以观察程序的运行状态，如某些变量的值，对程序进行分析。

（2）启动、继续和停止调试。选择菜单【调试】→【启动调试】（或快捷键 F5），程序运行到断点处暂停，如图 1-29 所示。

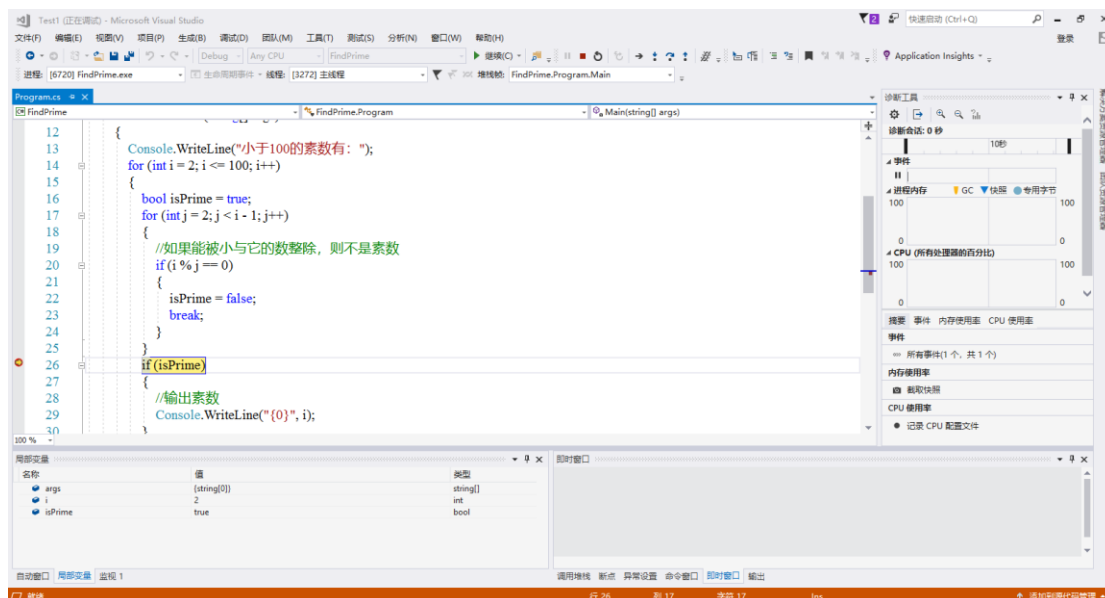


图 1-29 程序运行到断点处暂停

暂停后可以选择菜单项【调试】→【继续】（或快捷键 F5），程序继续运行，直到下一个程序断点。也可以选择程序单步执行，单步执行有三种，包括：（1）逐过程，每次执行一行，但不执行任何函数调用（快捷键 F10）；（2）逐语句，每次执行一行，但遇到函数调用就会跳到被调用的函数里，逐条语句执行代码（快捷键 F11）；（3）跳出，直接执行当前函数里剩下的指令，返回上一级函数执行当前执行点所处函数的剩余行（快捷键 Shift+F11）。选择菜单项【调试】→【停止调试】（快捷键【Shift + F5】）可以停止运行程序中的当前应用程序。

（3）使用“局部变量”窗口，调试过程中可以查看当前程序中变量的取值，如图 1-30 所示。

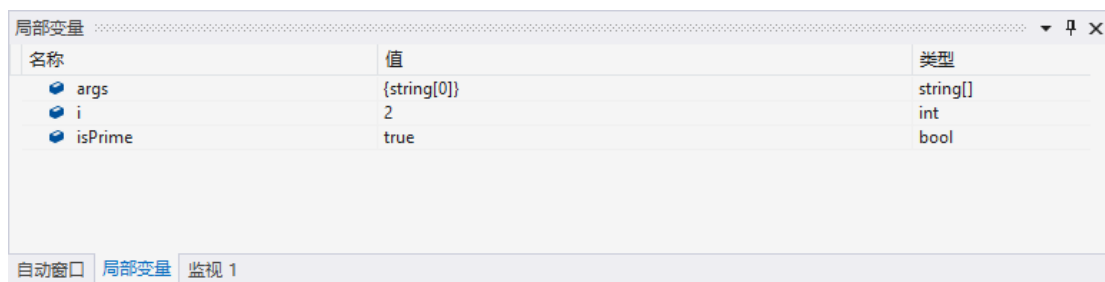


图 1-30 局部变量窗口

（4）可以将感兴趣的变量从代码窗口拖入到“监视”窗口，查看其取值变化情况，如图 1-31 所示。

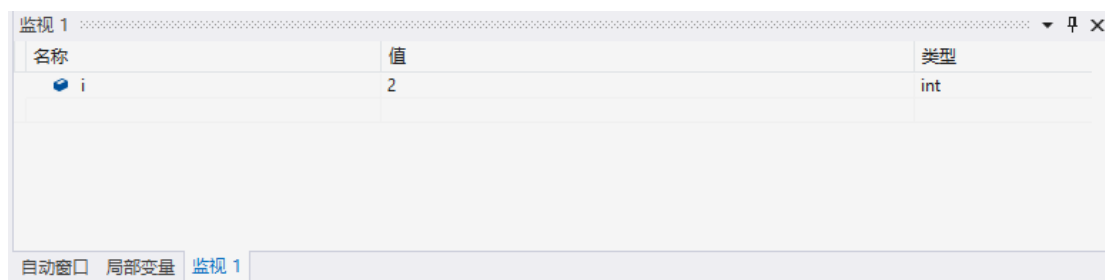


图 1-31 监视窗口

(5) 也可以通过【调试】→【快速监视】对话框查看变量或者表达式的值，也可以自定义表达式进行计算，如图 1-32 所示。

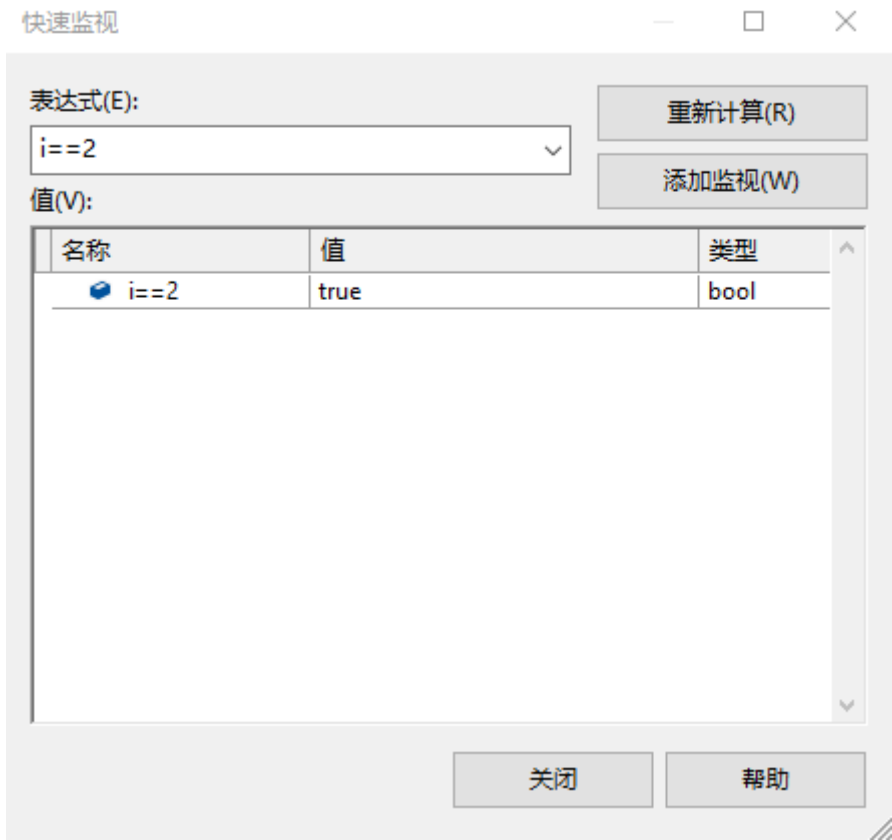


图 1-32 快速监视对话框

(6) 在程序调试过程中，可以使用“即时窗口”输入表达式，由开发语言对表达式进行计算或执行，如图 1-33 所示。

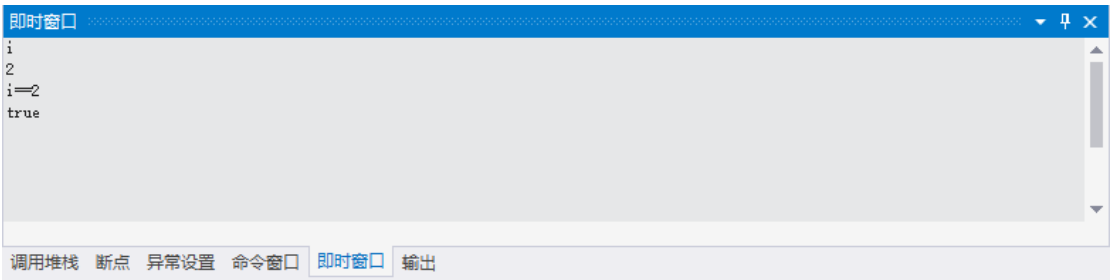


图 1-33 即时窗口