目　录

第 1 章　C#与 Windows 简介

微软声称："C#是一个从 C 和 C++演化过来的简单的、现代的、面向对象的和类型安全的编程语言"。在使用 C#（C sharp）的时候你会注意的第一件事是这种语言的结构是多么地熟悉了。通过面向对象的设计，C#提供了访问 Visual Basic 和 Visual C++类库的能力，然而，C#并没有它自己的类库。

C#是微软在 Microsoft Visual Studio 的最新版本中实现的，它提供了对下一代 Windows 服务（NGWS）的访问能力。这些服务包含了一个专门为代码开发而设计的通用执行引擎。

1.1　Visual Studio.NET 和 C#

最新版本的 Visual Studio.NET 为开发不同的应用程序提供了语言丰富的环境。程序可以由不同的语言开发，例如 C、C++、C#、Visual Basic 等等。应用程序也可以既包括标准的控制台（命令行或者是 DOS 模式）应用，又包括 Microsoft Windows 应用程序。

C#虽然是冲击这个市场的 C 的最新版本，但它只是这个更大的开发包中的一个组件。在发布 Visual Studio.NET 的时候，微软的目标之一就是能够为项目需求提供无缝解决方案。针对一个任务的解决方案可能需要将 C++、Visual Basic 和 C#中的不同组件集成到一个无缝可执行文件中。

本书着重于在创建 Windows 应用程序的时候对 C#的运用。如果你是那种想研究一种语言的所有细枝末节的程序员，这正是一本适合的书。

1.2　创建 C#应用程序

C#应用程序分为两类：命令行或控制台应用程序和 Windows 应用程序。使用 AppWizards，你会发现依据项目中的一些必需的模板代码，这两种应用和程序都很容易创建。

在我们的第一个项目中，我们创建一个大家熟悉的 Hello World 控制台应用程序。我们将这个项目命名为 Hello World。位于本章结束处的第二个应用程序叫做 CircleArea。这是一个完整的、面向对象的 Windows 应用程序。

在每一个项目中都尽量向你介绍 Visual Studio AppWizards 的使用方法，教你如何创建本书中开发的每一个项目的基本模板代码。这些代码都是你做标记的好地方，还可以在空白的地方做一些笔记。

1.3　第一个 C#控制台应用程序

要想用 C#创建一个控制台应用程序，首先启动 Visual Studio。选择 File | New | Project Sequence 来打开“New Project”对话框，如图 1-1 所示。

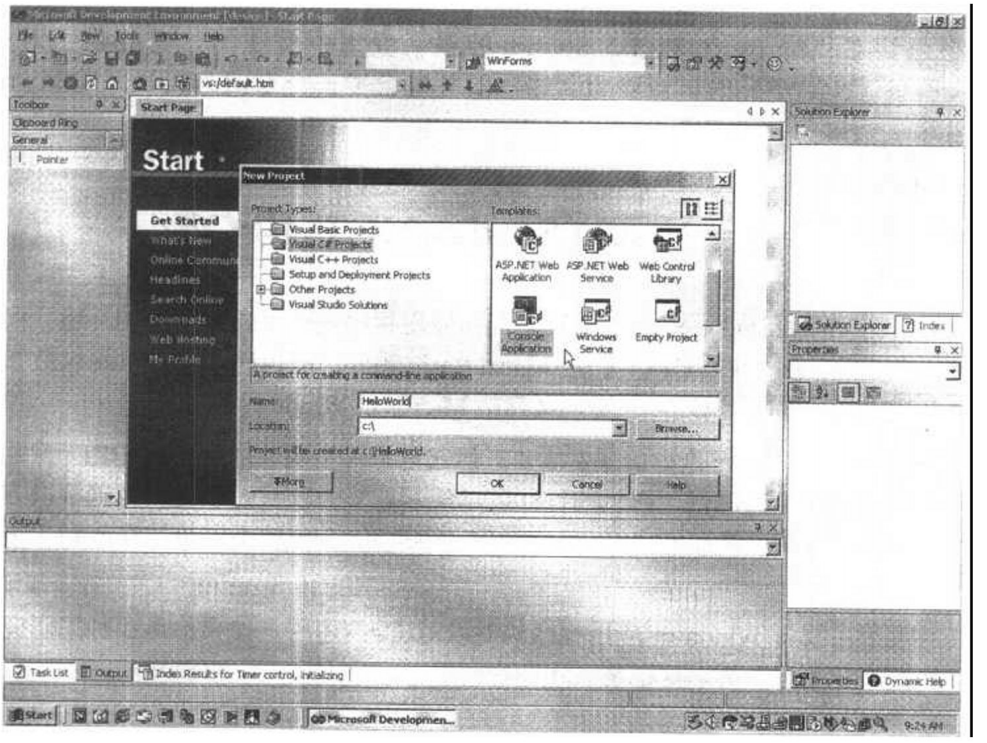

图 1-1　New Project 对话框使我们可以指定一个 C#控制台应用程序

将这个项目命名为 HelloWorld，并在根目录下面指定一个子目录，如图 1-1 所示。

当单击 OK 按钮的时候，C# AppWizard 就会创建如图 1-2 所示的模板代码。

现在可以修改这个模板代码来实现你的目标了。图 1-3 显示了我们是如何修改模板代码来完成 HelloWorld 项目的。

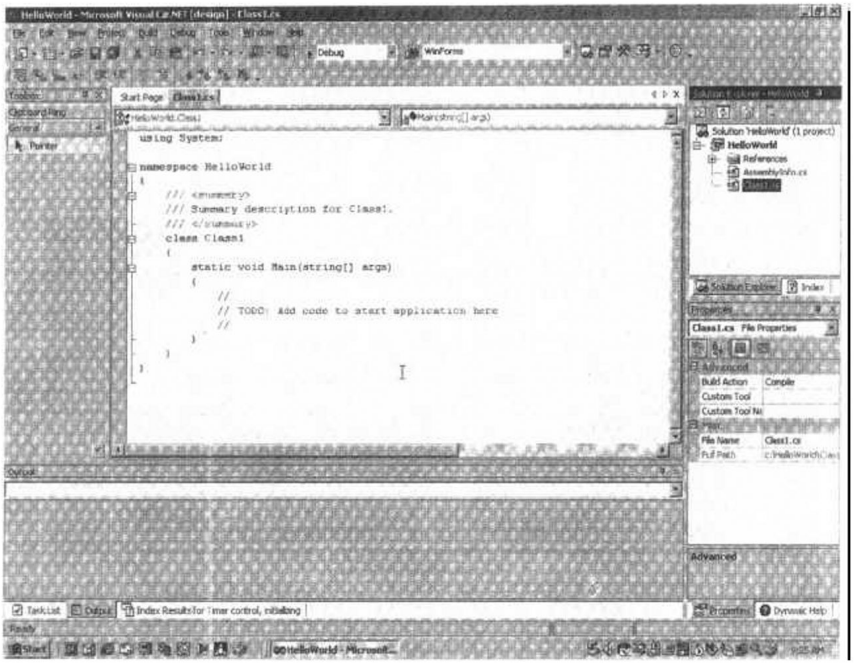

图 1-2 AppWizard 的控制台应用程序的 C#模板代码

观察图 1-3 并将其与下面的代码相比较。注意加黑了一行代码：

```
using System;

namespace HelloWorld
{
    /// <summary>
    /// Summary description for Class1.
    /// </summary>
    class Class1
    {
        static void Main(string[] args)
        {
            //
            // TODO: Add code to start application here
            //
            Console.WriteLine("Hello C# World!");
        }
    }
}
```

再次观察图 1-3，注意 Build 菜单已经被打开，而且正要选择 Rebuild 子菜单项。单击这个菜单项将创建这个项目。

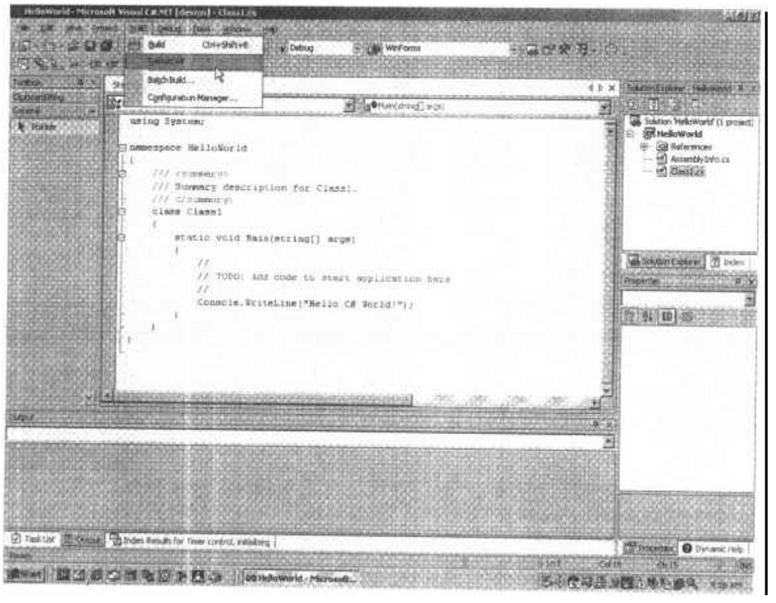

图 1-3　为 HelloWorld 项目修改模板代码

当观察这段例子代码的时候，你会发现这里的很多元素都是在 C 和 C++控制台应用程序中非常熟悉的了。图 1-4 显示了 Debug 菜单已经被打开，而且正要选择 Start Without Debugging 子菜单项时的情况。单击这个菜单项，就可以在 Visual Studio 集成开发环境中运行这个应用程序了。

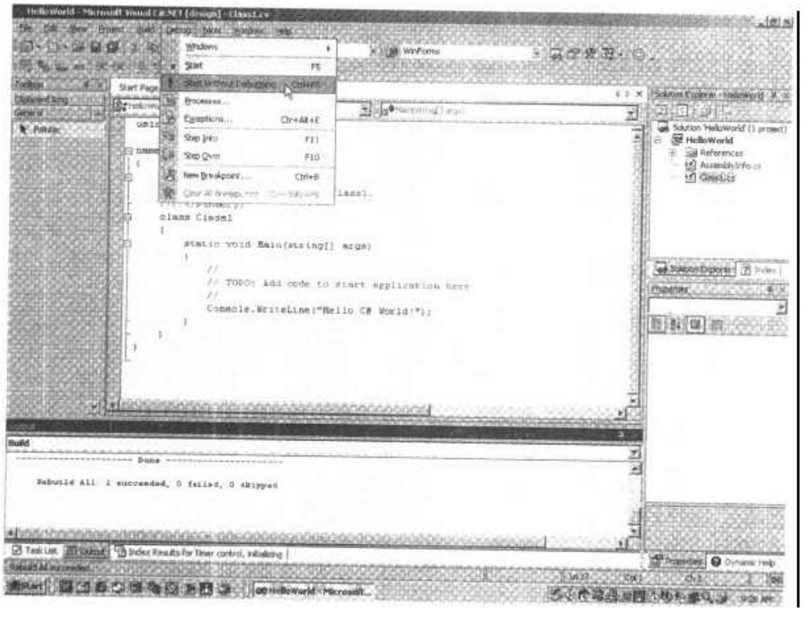

图 1-4　在 Visual Studio 集成开发环境中运行这个应用程序

当执行这个程序的时候，一个控制台（命令行或者是 DOS）窗口将会出现，并展示程序的输出。图 1-5 展示了这个应用程序的输出结果。

现在让我们简要浏览一下这个程序中那些熟悉的元素以及一些新的特性。首先，这个应用程序使用了系统指令。在运行时由 NGWS 提供的系统命名空间允许在 Main 方法中访问 Console 类。代码中所使用的 Console.WriteLine（）实际上是 System.Console.WriteLine（）的缩写形式，在这里 System 代表一个命名空间，Console 代表在这个命名空间中定义的一个类，WriteLine（）是在这个 Console 类中定义的一个静态方法。

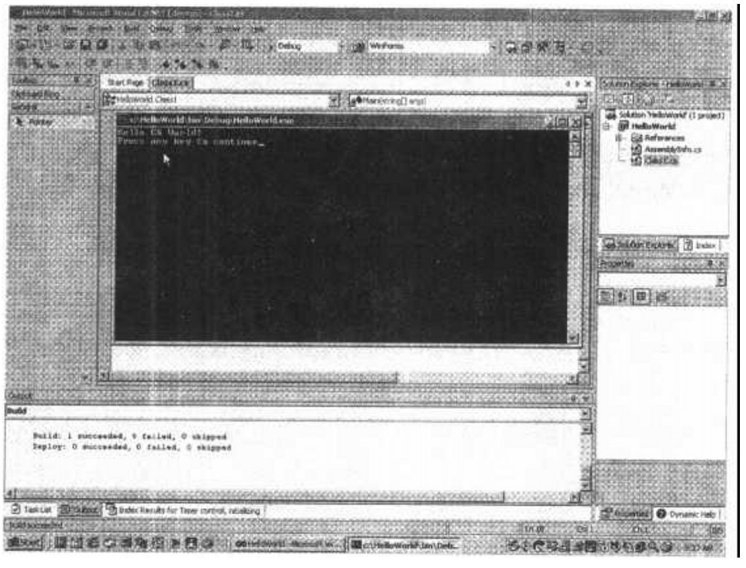

图 1-5 控制台窗口展示了项目的输出

其他程序细节

在 C#应用程序中，函数和变量都是包含在类和结构的定义中的，从来都不是全局的。

你可能会注意到在复合名之间使用“.”分隔符来加以分隔。C#使用这个分隔符来代替“::”和“->”。同时在 C#中也不需要前向声明，因为在这里顺序是不重要的。没有#include 语句表明 C#语言只是象征性地处理依赖关系。C#的另外一个特性是自动内存管理，这使得程序员可以从处理复杂的内存问题中解脱出来。

1.4 C#编程元素

这节将学习在本书中我们所要用到的关键的 C#语言编程元素。有时，我们会介绍一些额外的 C#知识，但是下面小节中的内容在今后将会多次用到。

1.4.1　数组

跟 C 和 C++语言一样，C#也支持多种数组，包括一维数组和多维数组。这种数组通常称作矩形数组，这跟变长数组相对。

要想声明一个一维整型数组，并命名为 myarray，可以使用下面的 C#语法：

```
int[ ]myarray = new int[12];
```

然后这个数据就可以被初始化为 12 个数值，我们使用下面的 for 循环来实现：

```
for (int i =o ; i <myarray.Length; i++)
    myarray [i] = 2 * i;
```

使用 for 循环和 WriteLine（）语句，数组的内容还可以写到屏幕窗口中：

```
for (int i = 0; i <myarray.Length; i++)
   Console.WriteLine ("myarray[{0}] = {1}", i, myarray[i]) ;
```

注意到在 WriteLine（）语句所提供的参数列表中 i 值被替换为{0}，myarray[]值被替换为{1}。

多维数组可以遵循相同的模式。例如，创建一个二维数组的语法如下：

```
int[ , ] my2array = new int[12, 2];
```

然后这个二维数组的值就可以用两个 for 循环来初始化，如下：

```
for (int i = 0; i < 12; i++)
    for (int j = 0; j < 2; j++)
        my2array[i, j] = 2 * i;
```

数组的内容还可以展示到控制台屏幕窗口中，同样使用 for 循环和 WriteLine（）语句，如下所示：

```
for (int i = 0; i < 12; i++)
    for (int j = 0; j < 2; j++)
        Console.WriteLine("my2array[{0}, {1}] = {2}",
                          i, j, my2array[i, j]);
```

三维数组也可以使用类似的语法来声明：

```
int[ , , ] my3array=new int[ 3, 6, 9];
```

除了能够处理多维矩形数组以外，C#还可以处理长度不等的变长数组，变长数组可以使用下面的语法来声明：

```
int[ ][ ] jagarray1;

int[ ][ ][ ] jagarray2;
```

例如，假设一个变长数组被声明为：

```
int[][] jagarray1 = new int[2][];

jagarray1[0] = new int[] {2, 4};
jagarray1[1] = new int[] {2, 4, 6, 8};
```

在这里 jagarray1 表示一个整型数组，从这个结构的参差不齐的形式可看出这个数组的类型名。下面这行代码将数值 6 打印到屏幕窗口上：

```
Console.WriteLine (jagarray1[1][2]) ;
```

希望读者能够练习编写必要的代码，将所有数组元素的值打印到屏幕窗口上。

1.4.2 属性（attribute）、事件、索引器、属性以及版本转换

当开发 Windows 应用程序的时候，将会使用到本节所学习的很多术语。如果你已经使用过了 Visual Basic 或者微软基础类库（MFC）和 C++，你会很熟悉属性、事件这些术语了。在下面这节中，我们将扩展这些定义。

译者注：在微软的术语表中，Attribute 和 Property 均称为属性，在翻译本书时为加以区别，凡出现 Attribute 时，均给出其英文，即属性（Attribute）。而 Property 直接译为属性。

1. 属性（attribute）

C#属性（attribute）使得程序员可以识别和编写新的声明信息。例如：public、private 和 protected 是识别一个方法的访问能力的属性（attribute）。

一个元素的属性（attribute）信息可以在运行时用 NGWS 的运行时反射支持来获得。

2. 事件（Event）

事件用来让类提供有关哪一个客户程序能够提供可执行代码的通知消息。这个代码是以事件处理程序的形式存在的。如果你曾经开发过 MFC C++ Windows 代码，那么，你一定已经很熟悉事件处理程序了。

这里有按钮点击事件处理程序的代码。它是从本书后面所开发的项目中节选出来的：

```
private void button1_Click(object sender, System.EventArgs e)
{
    radius = Convert.ToDouble(textBox1.Text);
```

```
        textBox2.Text = (radius * radius * 22 / 7).ToString();
        textBox3.Text = (radius * 2.0 * 22 / 7).ToString();
    }
```

事件处理程序包含了当一个按钮点击事件触发的时候将要执行的代码。Button 是一个 C# Windows 应用程序中窗体上的一个按钮。

3. 索引器（indexer）

C#中用索引器来处理类似于数组的数据结构，例如字符串数组。这种数据结构可以在一个 C# Windows 控件中使用，例如 CheckedListBox 控件。

```
    .
    .
    .
{
    private string[] items;
    public string this[int index] {
        get {
            return items[index];
        }
        set {
            items[index] = value;
            Repaint();
        }
}
```

于是 CheckedListBox 控件类可以修改为如下的代码：

```
CheckedListBox MyListBox;
MyListBox[0] = "List box title";
Console.Write(MyListBox[0]);
```

索引器对数组的访问类似于属性对字段的访问。

4. 属性（property）

属性是与一个类或者对象相关的属性（attribute）。Windows 控件提供了种类广泛的可修改的属性，包括标题名、ID 值、颜色、字体、位置、大小和文本等等。

这里是一个 C# Windows 应用程序中改变按钮控件属性的一小段代码：

```
this.button1.Location = new System.Drawing.Point(152, 192);
this.button1.Size = new System.Drawing.Size(176, 24);
this.button1.TabIndex = 6;
this.button1.Text = "Push to Calculate";
this.button1.AddOnClick(new System.EventHandler(button1_Click));
```

在需要的时候你可以读取属性值，也可以为属性赋值。

5. 版本转换（versioning）

C#通过两个层次上的兼容性来提供对版本转换的支持。第一个层次是源代码兼容，如果代码是由早期版本开发的，可以很简单的在以后版本中重新编译使用。

第二个层次是二进制代码兼容，若代码是由早期版本开发的，在新版本中不用重新编译就可以运行。

1.4.3 装箱、拆箱以及统一类型系统

C#中的所有的类型都可以被看成是对象（object）。例如，下面的代码行在 C#中是可以接受的：

```
Console.Writeline (12345.ToString ( ) ) ;
```

这时，ToString（）方法是将整数 12345 作为一个对象来处理的。

当一个值要被转换为一个引用类型的时候可以使用对象来包装，这称为装箱；拆箱是将一个引用类型还原为一个数值。例如：

```
int num1 = 12345;

object myobject = num1; // boxed

int num2 = (int) myobject; // unboxed
```

在这里，整型值 12345 首先通过装箱被转换为一个引用类型，然后再将这个对象转换回整型值（拆箱）。

1.4.4 类、结构和枚举

C#提供了简单且惟一的对这些通用的面向对象特性的实现。

1. 类（classes）

C#的类只允许单重继承。类的成员可以包括常量、构造函数、析构函数、事件、索引器、方法、属性以及操作符。每一个成员都可以具有公共的（public）、保护的（protected）、内部的（internal）、内部保护的（protected internal）或者私有的（private）访问方式。

类的结构与在 C 和 C++中使用的类很相似。例如：