第3章 值和方法

3.1 知识点回顾

本章介绍类和结构的基本成员类型以及变量和常量的用法。成员的访问限制规则、静态和非静态成员的区别等都是大家应当牢固掌握的基础知识。本章的难点在于方法中不同参数类型的用法，以及委托对方法的封装技术。

基本知识点：

(1) 作为复合类型，类和结构可以有字段成员、方法成员，以及嵌套成员。一个类型中不能有两个同名的成员。

(2) 常数常量使用关键字 const 定义，它在定义的同时进行赋值；只读常量使用关键字 readonly 定义，它可在字段定义或类型初始化时进行赋值。

(3) 类型的可变实例叫做变量，它必须先定义后使用。

(4) 一般方法成员包括返回类型、方法名、参数列表和执行体 4 个部分。方法定义中的参数叫做形式参数，而调用方法时所使用的参数叫做实际参数。参数类型有普通参数、引用型参数、输出型参数和数组型参数，其中普通参数不能改变实参的值。

(5) 非静态成员(包括字段和方法)属于类型的实例(对象)，而静态成员属于类型本身。

(6) 字段和变量是对数据的封装，委托则是对方法(过程)的封装。把一个方法封装到一个委托变量中后，就可以将方法作为另一个方法的参数或返回值。

(7) C＃通过访问限制修饰符控制对类型和成员的访问。私有成员只有类型本身才能访问，保护成员还可被派生类型访问，内部成员则只能在当前程序集中访问。

3.2 实验目的和要求

- 理解字段和类型的默认值；
- 掌握成员方法的定义和调用；
- 理解静态成员的作用和使用方式；
- 掌握通过委托对方法进行封装和调用的基本技术。

3.3 实验内容

(1) 在控制台输出各种常见数据类型的默认值。

(2) 学习.NET 类库中的 Random 类,通过它随机生成不同类型的数。

(3) 学习使用 System.Windows.Forms 命名空间下的 Application 类,通过其静态成员控制应用程序。

(4) 学习定义委托类型,并使用委托类型的变量来封装和调用方法。

3.4 实验指导

实验 3-1 输出类型的默认值

(1) 创建 C# 控制台应用程序 L3_1。

(2) 在程序中新建一个 MyClass 类,在其中分别定义一个嵌套结构 InnerStruct 和一个嵌套类 InnerClass,然后为类分别添加 short、int、long、float、double、bool、string、InnerStruct 和 InnerClass 类型的字段各一个。参考源代码如下:

```
class MyClass
{
    public short s;
    public int i;
    public long l;
    public float f;
    public double d;
    public bool b;
    public string str;
    public InnerStruct m_struct;
    public InnerClass m_class;

    public struct InnerStruct
    {
    }

    public class InnerClass
    {
    }
}
```

(3) 在程序主方法 Main 中创建一个 MyClass 的实例变量,并输出其各个字段的值(即类型的默认值)。参考源代码如下:

```
static void Main()
```

```
{
    MyClass obj = new MyClass();
    Console.WriteLine("default value of short: {0}", obj.s);
    Console.WriteLine("default value of int: {0}", obj.i);
    Console.WriteLine("default value of long: {0}", obj.l);
    Console.WriteLine("default value of float: {0}", obj.f);
    Console.WriteLine("default value of double: {0}", obj.d);
    Console.WriteLine("default value of bool: {0}", obj.b);
    Console.WriteLine("default value of string: {0}", obj.str);
    Console.WriteLine("default value of Struct: {0}", obj.m_struct);
    Console.WriteLine("default value of Class: {0}", obj.m_class);
}
```

(4) 使用普通方式编译程序，看编译过程中有哪些提示信息，然后修改编译选项取消提示，并重新编译程序。

(5) 执行程序并查看其输出结果，如图 3.1 所示。

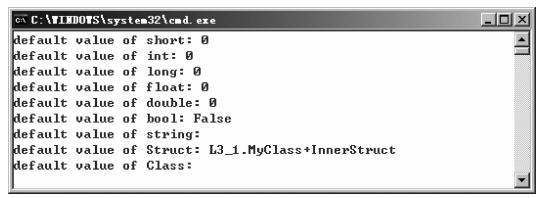

图 3.1 程序 L3_1 的输出结果

实验 3-2 使用 Random 类生成随机数

(1) 创建 C# 控制台应用程序 L3_2。

(2) 在程序主方法 Main 中创建一个 Random 类型的对象 r，用于在程序中随机生成不同的数。

(3) Random 类型的 Next 方法有多种重载形式，其中不带参数的方法用于生成一个随机正整数。在程序主方法中调用该方法生成 3 个随机数。参考源代码如下：

```
Console.WriteLine("输出任意 3 个随机整数: ");
Console.WriteLine("{0}, {1}, {2}", r.Next(), r.Next(), r.Next());
```

(4) Random 类型的 Next(int n)方法用于生成一个 0 到 n 之间的随机正整数，Next(int $n1$, int $n2$)方法则生成一个 $n1$ 到 $n2$ 之间的随机整数。在程序主方法中继续添加代码，分别调用这两个方法生成随机数。

(5) Random 类型的 NextDouble 方法用于生成一个 0 到 1 之间的随机实数，在程序中调用该方法生成 3 个随机实数。

(6) 继续在程序中生成 3 个 0 到 10 之间的随机实数。

(7) Random 类型的 NextBytes 方法用于随机生成一组 byte 类型的数，并将这些数写入一个 byte[]型数组。在程序中创建一个长度为 3 的数组，调用该方法写入随机数，并输出数组内容。参考源代码如下：

```
Console.WriteLine("输出一个随机数数组：");
byte[] buf = new byte[3];
r.NextBytes(buf);
Console.WriteLine("{0}, {1}, {2}", buf[0], buf[1], buf[2]);
```

(8) 执行程序并查看其输出结果(如图 3.2 所示)。

图 3.2　程序 L3_2 的输出结果

实验 3-3　使用 Application 类

(1) 创建 Windows 窗体应用程序 L3_3。

(2) 从工具箱中拖放 4 个 Label 控件到窗体上，然后在设计视图中双击窗体，在自动生成的窗体启动事件处理方法中输入代码，在各个 Label 控件上分别显示 Application 类的静态属性 ProductName、ProductVersion、StartupPath 和 CurrentCulture，它们分别表示当前程序的名称、版本、路径和语言。参考源代码如下：

```
private void Form1_Load(object sender, EventArgs e)
{
    label1.Text = "程序名称:" + Application.ProductName;
    label2.Text = "程序版本:" + Application.ProductVersion;
    label3.Text = "程序路径:" + Application.StartupPath;
    label4.Text = "程序语言:" + Application.CurrentCulture;
}
```

(3) 再向窗体中添加两个 Button 控件，设置其显示文本分别为“重启”和“退出”。为两个按钮添加单击事件处理方法，在其中分别调用 Application 类的静态方法 Restart 和 Exit，分别用于重启程序和退出程序。参考源代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    Application.Restart();
}

private void button2_Click(object sender, EventArgs e)
{
    Application.Exit();
}
```

(4) 编译运行程序，查看窗体输出效果，如图 3.3 所示。

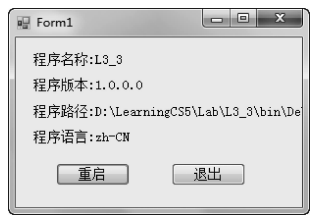

图 3.3　程序 L3_3 的 Windows 窗体输出结果

实验 3-4　使用委托封装和调用数学函数

（1）创建 Windows 窗体应用程序 L3_4。

（2）从工具箱中向主窗体 Form1 拖放 3 个 Label 控件、一个 NumericUpDown 控件、一个 ListBox 控件、一个 Button 控件和一个 TextBox 控件，窗体界面设计如图 3.4 所示。

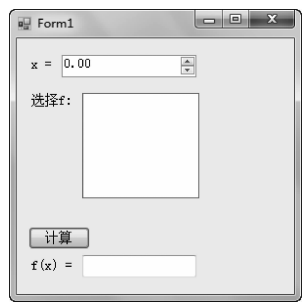

图 3.4　程序 L3_4 的 Windows 窗体设计界面

（3）为项目添加一个新类 MyFunc，定义一个输入和输出参数类型均为 double 的委托类型 Func，并为 MyFunc 添加一个 Func 类型的字段 f 和一个 string 类型的字段 name，分别表示一个一元函数及其名称。再为 MyFunc 添加一个 ToString 方法，该方法使用 override 修饰符，并返回 name 字段值。参考源代码如下：

```
delegate double Func(double x);

class MyFunc
{
    public Func f;
    public string name;

    public override string ToString()
    {
        return name;
    }
}
```

（4）回到主窗体 Form1.cs 的代码视图，在窗体构造函数中输入代码，创建一组 MyFunc 对象，每个对象封装一个特定的一元数学函数（由.NET 类库中的 Math 类提供），

然后将这些对象加入到 ListBox 控件的项集中。参考源代码如下：

```
public Form1()
{
    InitializeComponent();
    listBox1.Items.Add(new MyFunc() { f = Math.Exp, name = "Exp" });
    listBox1.Items.Add(new MyFunc() { f = Math.Log, name = "Ln" });
    listBox1.Items.Add(new MyFunc() { f = Math.Sin, name = "Sin" });
    listBox1.Items.Add(new MyFunc() { f = Math.Cos, name = "Cos" });
    listBox1.Items.Add(new MyFunc() { f = Math.Asin, name = "Asin" });
    listBox1.Items.Add(new MyFunc() { f = Math.Acos, name = "Acos" });
}
```

(5) 为窗体的按钮控件添加 Click 事件处理方法，在其中获取 ListBox 控件中所选定的项(MyFunc 对象)，从中得到相应的一元数学函数，调用函数并输出计算结果。参考源代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    double x = (double)numericUpDown1.Value;
    Func f = ((MyFunc)listBox1.SelectedItem).f;
    textBox1.Text = f(x).ToString();
}
```

(6) 编译运行程序并查看计算结果。

提示：ListBox 控件的项集成员类型为 object，因此可以把任意对象装箱为 object 类型后放入项集，并在需要时拆箱取出相应的对象。这些成员在 ListBox 控件中所显示的内容由对象类型的 ToString 方法决定。

3.5 补充上机练习

(1) 编写 C#控制台应用程序，在其中定义方法 Calculate，它可以接受一个 double[]数组的输入参数，计算数组中所有元素的和与积，并分别保存在两个输出参数中。在程序主方法中调用该方法。

(2) 编写 C#控制台应用程序，在其中定义重载的静态方法 Max，用于得到两个数或 3 个数的最大值，且数值类型可以是 int、double 和 decimal。类似地，定义求最小值的静态方法 Min。在程序主方法中调用这两个静态方法的不同重载形式。

(3) 参照实验 3-3，编写 WPF 应用程序，对比它与 Windows 窗体应用程序的程序对象类型有何区别。

(4) 扩展实验 3-4 中的程序 L3_4，使其支持二元数学函数的计算。

(5) 编写一个 C#类库应用程序，在其中定义访问限制分别为 public、protected、private，以及 protected internal 的 4 个方法。然后创建一个 C#控制台应用程序，看看从中可以调用类库程序中的哪些方法。