第 2 章　C#程序设计基础

第 1 章学习了 ASP.NET 的特性和.NET Framework 的一些基本知识，如果要深入学习 ASP.NET 应用程序开发，需要对开发语言有更进一步的了解。而在.NET 平台上，微软主推的编程语言就是 C#，本章将会从 C#的语法、结构和特性来讲解，以便读者能够深入地了解 C#程序设计。

C#语言是专门用于.NET 的编程语言，是为在.NET Framework 上运行的多种应用程序而设计的。C#语言简单、功能强大、类型安全，是一种面向对象的语言，从 C、C++以及 Java 演化而来，吸收了其他语言的优点，并解决了它们存在的一些问题。

C#语言凭借自身的多项创新，实现了应用程序的快速开发，几乎可以开发出所有的 Windows 程序。

2.1　C#程序

2.1.1　C#程序的结构

我们来做一个最简单的 C#程序——这是一个把信息写到屏幕上的控制台应用程序。

实例：编写第一个控制台程序。

解决方案：

启动 Visual Studio，新建一个控制台应用程序，在 main()中输入以下代码：

```
//这是我的第一个 C#程序
Console.WriteLine("Hello World!");
Console.ReadLine();
```

运行程序，将在 DOS 模式下显示字符“Hello World!”，并等待输入。同时，在 Bin\debug 目录下会生成一个可执行文件 ConsoleApplication1.exe。点击此文件，会显示同样的效果。

详细介绍：在 C#中，每个语句都必须用一个分号（;）结尾，语句可以写在多个代码行上；用花括号（{...}）把语句组合为块；单行注释以两个斜杠字符开头（//），多行注释以一个斜杠和一个星号（/*）开头，以一个星号和一个斜杠（*/）结尾。

2.1.2　C#的代码设置

代码格式也是程序设计中一个非常重要的组成环节，它可以帮助用户组织代码和改进代码，也让代码具有可读性。具有良好可读性的代码能够让更多的开发人员更加轻松地了解和认知代码。按照约定的格式书写代码是一个非常良好的习惯，下面的代码示例说明了应用缩进、大小写敏感、空白区和注释等格式的原则。

```
using System;
using System.Collections.Generic;
using System.Linq;                                    //使用 LINQ 命名空间
using System.Text;
namespace mycsharp                                    //声明命名空间
{
    class Program                                     //主程序类
    {
        static void Main(string[] args)               //静态方法
        {
            Console.WriteLine("Hello World");         //这里输出 Hello World
            Console.WriteLine("按任意键退出.."); Console.ReadKey();
                                                  //这里让用户按键后退出，保持等待状态
        }
    }
}
```

1. 缩　进

缩进可以帮助开发人员阅读代码，同样能够给开发人员带来层次感。读者可以从以上代码看出这一串代码让人能够很好地分辨区域，非常方便地就能找到 Main 方法的代码区域，这是因为括号都是有层次的。

缩进让代码保持优雅，同一语句块中的语句应该缩进到同一层次，这是一个非常重要的约定，因为它直接影响到代码的可读性。虽然缩进不是必须的，同样也没有编译器强制，但是为了在不同人员的开发中能够进行良好的协调，这是一个值得去遵守的约定。

2. 大小写敏感

C#是一种对大小写敏感的编程语言。在 C#程序中，同名的大写和小写代表不同的对象，因此在输入关键字、变量和函数时必须使用适当的字符。但是在 C#中，其语法规则的确是对字符串中字母的大小写是敏感的，如“ C Sharp ”“ c Sharp ”“ c sHaRp ”是不同的字符串，在编程中应当注意。对于关键字基本上都采用小写，对于私有变量的定义一般都以小写字母开头，而公共变量的定义则以大写字母开头。

3. 空　白

C#编译器会忽略空白。使用空白能够改善代码的格式，提高代码的可读性。但是值得注意的是，编译器不对引号内的任何空白做忽略，在引号内的空格作为字符串存在。

4. 注　释

在 C/C++里，编译器支持开发人员编写注释，良好的注释习惯能够增强代码的可读性，以便开发人员能够方便地阅读代码。当然，在 C#里也一样继承了这个良好的习惯。

C#提供了三种注释的类型：

第一种：单行注释，注释符号是“ // ”，例如：

```
int a;     //一个整型变量，存储整数
```

第二种：多行注释，注释符号是“ /* ”和“ */ ”，任何在符号“ /* ”和“ */ ”之间的内容都会被编译器忽略，例如：

```
/*一个整型变量，存储整数*/
int a;
```

第三种：注释符号“ /// ”也可以用来对 C#程序进行注释，例如：

```
///一个整型变量
  ///存储整数
  int a;
```

5. 布局风格

```
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Hello World");          //这里输出 Hello World
        Console.WriteLine("按任意键退出.."); Console.ReadKey();
                                                   //这里让用户按键后退出，保持等待状态
    }
}
```

从以上代码可以看出，程序中使用了缩进、大小写敏感、空白区和注释等，但是这个代码风格依旧不是最好，可以修改代码让代码更加“ 好看 ”。这里能够将代码进行修正，修正后的示例代码如下：

```
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Hello World");          //这里输出 Hello World
        Console.WriteLine("按任意键退出..");
        Console.ReadKey();                         //这里让用户按键后退出，保持等待状态
    }
}
```

这种布局风格让开发人员感觉到耳目一新，这样能方便更多的开发人员阅读源代码。如果打开一千行或更多代码量的源文件时，其编码格式都是标准的风格，不管是谁再接手去阅读，都能尽快上手。不仅如此，在软件开发当中，应该规定好每个人都使用同样的布局风格，让团队能够协调运作。

2.2 C#的数据类型

在任何编程语言中，无论是传统的面向过程还是面向对象都必须使用变量。因此，变量都有自己的数据类型。数据类型在程序设计中是一个很重要的内容。C#的数据类型包括值类型、引用类型和指针类型。指针类型是不安全类型，一般不推荐使用。

2.2.1 值类型

值类型直接存储值。值类型包括所有简单数据类型、枚举类型和结构类型。

值类型声明语法如下：

```
Type name;
name=TypeVaue;
```

或者：

```
Type name=new Type();   //声明加初始化
```

1. 简单数据类型

简单数据类型是 C#预先定义的结构类型，又称纯量类型，是直接由一系列元素构成的数据类型，它包括整数类型、浮点类型和字符类型等。

（1）整数类型如表 2-1 所示。

表 2-1 整数类型

类 型	CTS 类型	说 明	范 围
sbyte	System.SByte	8 位有符号整数	－128~127
short	System.Int16	16 位有符号整数	－32 768～32 767
int	System.Int32	32 位有符号整数	－2 147 483 648～2 147 483 647
long	System.Int64	64 位有符号整数	－263~263
byte	System.Byte	8 位无符号整数	0～255
ushort	System.Unit16	16 位无符号整数	0～65 535
unit	System.Unit32	32 位无符号整数	0～4 284 867 285
ulong	System.Ulong	64 位无符号整数	0～264
float	System.Single	32 位单精度浮点数	1 038
double	System.Double	64 位双精度浮点数	10 308
decimal	System.Decimal	128 位双精度浮点数	1 028

（2）浮点类型。

小数在 C#中采用两种数据类型来表示：单精度 float 和双精度 double。它们的差别在于取值范围和精度不同。

计算机对浮点数的运算速度大大低于对整数的运算，对精度要求不是很高的浮点数计算，我们可以采用 float 型，而采用 double 型获得的结果将更为精确，如果在程序中大量地使用双精度类浮点数将会占用更多的内存单元，而且计算机的处理任务也将更加繁重。

（3）字符类型。

包括一般字符和转义字符，采用 Unicode 字符集。C#的 char 类型为双字节型，它的数据可以占有 2 个字节。以下方法给一个字符变量赋值，如：

```
char c = 'A';
```

C#中用转义符在程序中指代特殊的控制字符。例如，字符串常量“ c:\\windows\\system32 ”的真实含义是路径 c:\windows\system32。

C#可以用反转符@去掉反斜杠的转义。例如，字符串常量@“ c:\windows\system32 ”也表示路径 c:\windows\system32。

2. 枚举类型

枚举类型实际上是为一组在逻辑上密不可分的整数值提供便于记忆的符号。枚举类型可以有效地限定变量的取值范围，达到减少赋值出错的概率。

格式：enum <枚举类型名> {<枚举表>}；

例如，定义一个代表星期的枚举类型的变量：

```
enum WeekDay{
Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday
};
```

大括号中的表示符为枚举元素，枚举元素默认基础类型为 int 类型，默认情况下，第一个枚举元素的值为 0，后面每个枚举元素值依次递增 1，也可以直接给枚举元素赋值。例如：Sunday=2，则 Monday 为 3，以此类推。

3. 结构类型

如果要把不同类型的数据组合到一起便于使用，就要用到结构类型。结构就是相互关联的不同类型数据的组合。结构类型是一种可包含构造函数、常数、字段、方法、属性、索引器、运算符、事件和嵌套类型的值类型。

在 C#中，结构类型是用来封装简单对象，把它们封装成一个实体来统一使用，结构不能从其他的类继承，也不能作为其他类的基类。结构类型以 struct 关键字进行声明。

例如，定义一个描述学生信息的结构。

```
using System；
struct Student
{   public int no;
    public string name;
    Public string phone;
    Public Student(int stu_no,string stu_name,string stu_phone)
    {     no=stu_no;
```

```
            name=stu_name;
            phone=stu_phone;
        }
    }
```

2.2.2 引用类型

引用类型存储的是对值的引用。引用类型变量又称为对象，在 C#中，引用类型有类类型（Class）、接口类型（Interface）、委托类型（Delegate）等。和值类型相比，引用类型不存储它们所代表的实际数据，只存储对实际数据的引用。具体情况是，当将一个数值保存到一个值类型变量后，该数值被复制到变量中，把一个值类型赋值给一个引用类型时，仅仅是引用（保存数值的变量地址）被复制，而实际值仍然保留在相同的内存位置。

多个引用变量可以附加于一个对象，而且某些引用可以不附加于任何对象，如果声明了一个引用类型的变量却不给它赋给任何对象，那么它的默认值就是 null。相比之下，值类型的值不能为 null。

C#有两个内置的引用类型，分别为 Object 和 String 类型。

对象类型（Object）在.NET 框架中是 System.Object 的别名，在 C#的统一类型系统中，所有类型（预定义类型、用户定义类型、引用类型和值类型）都是直接或间接从 Object 继承的。可以将任何类型的值赋给 Object 类型变量。也就是说，对象类型是其他类型的基类。

C#有 string 关键字，在翻译为.NET 类时，它就是 System.String，专门用于对字符串的操作。有了它，像字符串连接和字符串复制这样的操作就很简单了，可以用加号+ 合并两个字符串：

```
string str1 = "Hello ";
string str2 = "World";
string str3 = str1 + str2; // string concatenation
```

2.3 变量和常量

程序所处理的数据不仅分为不同的类型，而且每种类型的数据还有常量与变量之分。从用户角度来看，变量就是存储信息的基本单元；从系统角度来看，变量就是计算机内存中的一个存储空间。常量是指在程序运行的整个过程中其值始终不可改变的量，常量的值仅可在编译时指定，平时不允许更改。

2.3.1 变　量

变量是指在程序运行过程中其值可以不断变化的量。变量通常用来保存程序运行过程

中的输入数据、计算获得的中间结果和最终结果。在 C#中，变量可分为静态变量、实例变量、数组变量、局部变量、参数值、引用参数和输出参数这 7 种。

变量的命名规则必须符合标识符的命名规则，并且变量命名要人性化，以便理解。

C#的变量名区分大小写，变量命名的规则包括：

（1）变量名只能由字母、数字和下划线组成，且变量名的第一个符号只能是字母或下划线；

（2）不能使用关键字来作变量名。

2.3.2 声明并初始化变量

C#规定使用变量前必须先声明。声明的同时规定了变量的数据类型和变量名。

1. 语　法

声明变量的语法非常简单，即在数据类型之后编写变量名，如一个人的年龄（age）和一辆车的颜色（color），声明代码如下：

```
int age;                    //声明一个叫 age 的整型变量，代表年龄
string color;               //声明一个叫 color 的字符串变量，代表颜色
```

上述代码声明了一个整型变量 age 和一个字符串型变量 color。

2. 初始化变量

变量在声明后还需要初始化，例如“我年龄 21 岁，很年轻，我想买一辆红色的车”，那么就需要对相应的变量进行初始化，示例代码如下：

```
int age;                    //声明一个叫 age 的整型变量，代表年龄
string color;               //声明一个叫 color 的字符串变量，代表颜色
age = 21;                   //声明始化，年龄 21 岁
color = "red";              //声明始化，车的颜色为红色
```

上述代码也可以合并为一个步骤简化编程开发，示例代码如下：

```
int age=1;                  //声明并初始化一个叫 age 的整型变量，代表年龄
string color="red";         //声明初始化
```

3. 赋　值

在声明了一个变量之后，就可以给这个变量赋值了，但是当编写以下代码时就会出错，示例代码如下：

```
float a = 1.1;              //错误地声明浮点类型变量
```

当运行了以上代码后会提示错误信息：不能隐式地将 Double 类型转换为“float”类型，应使用“F”后缀创建此类型。从错误中可以看出，将变量后缀增加一个“F”即可，示例代码如下：

```
float a = 1.1F;                    //正确地声明浮点类型变量
```

运行程序，程序就能够编译并运行了。这是因为若无其他指定，C#编译器将默认所有带小数点的数字都是 Double 类型，如果要声明成其他类型，可以通过后缀来指定数据类型。

表 2-2 将展示一些可用的后缀，并且后缀可用小写。

表 2-2　可用的后缀表

后　缀	描　述
U	无符号
L	长整型
UL	无符号长整型
F	浮点型
D	双精度浮点型
M	十进制
L	长整型

2.3.3　变量的分类

在 C#中，变量可分为静态变量、实例变量、数组变量、局部变量、参数值、引用参数和输出参数这 7 种。

数组是一个引用类型，开发人员能够声明数组并初始化数据进行相应的数组操作，数组是一种常用的数据存放方式。

1. 静态变量

通过 static 修饰符声明的变量称为静态变量。静态变量只有被创建并加载后才会生效，同样被卸载后失效。声明一个整型静态变量 a 的代码如下：

```
Static int a=0;                    //声明静态变量并赋值
```

2. 实例变量

声明变量时，没有 static 修饰的变量称为实例变量。当类被实例化时，将生成属于该类的实例变量。当不再对该实例进行引用，并且已执行实例的析构函数后，此实例变量将失效。类中实例变量的初始值是该类型变量的默认值。为了方便进行赋值检查，类中的实例变量应是初始化的。例如，声明一个整型的实例变量 a，初始化代码如下：

```
int a;                             //声明实例变量
```

3．数组变量

数组元素随着数组的存在而存在，当任意一个数组实例被创建时，该数组元素也被同时创建。每个数组元素的初始值是该数组元素类型的默认值。声明一个整型数组变量的代码如下：

```
Int[] arry]=new int[5];            //声明数组变量
```

4. 局部变量

具有局部作用的变量称为局部变量，它只在定义它的块内起作用。所谓块是指大括号“｛”和“｝”之间的所有内容。局部变量从被声明的位置开始起作用，当块结束时，局部变