第3章

C#和ASP.NET 3.5

本章要点：

- 了解C#语言特点和编程规范。
- 了解常用.NET命名空间。
- 熟悉C#基础语法、流程控制。
- 能创建简单的类。

3.1 C# 概 述

C#是Microsoft专门为.NET量身打造的一种全新的编程语言。目前，C#已经分别被ECMA和ISO/IEC组织接受并形成ECMA-334标准和ISO/IEC 23270标准。它与.NET Framework有密不可分的关系，C#的类型即.NET Framework所提供的类型，并直接使用.NET Framework所提供的类库。另外，C#的类型安全检查、结构化异常处理等都交给CLR处理。实际上，ASP.NET 3.5本身就采用C#语言开发，所以C#不仅非常适用于Web应用程序的开发，也适用于开发强大的系统程序。总体来说，它具有以下典型特点：

(1) C#代码在.NET Framework提供的环境下运行，不允许直接操作内存，增强了程序的安全性。C#不推荐使用指针，若要使用指针，就必须添加unsafe修饰符，且在编译时使用/unsafe参数。

(2) 使用C#能构建健壮的应用程序。C#中的垃圾回收将自动回收不再使用的对象所占用的内存；异常处理提供了结构化和可扩展的错误检测和恢复方法；类型安全的设计则避免了读取未初始化的变量、数组索引超出边界等情形。

(3) 统一的类型系统。所有C#类型都继承于一个唯一的根类型object。因此，所有类型都共享一组通用操作。

(4) 完全支持组件编程。现代软件设计日益依赖自包含和自描述功能包形式的软件组件，通过属性、方法和事件来提供编程模型。C#可以容易地创建和使用这些软件组件。

3.2 Framework 命名空间

.NET Framework提供几千个类用于对系统功能的访问，这些类是建立应用程序、组件和控件的基础。在.NET Framework中，组织这些类的方式即是命名空间。作为组织类的

逻辑单元，命名空间即成了应用程序的内部组织形式，也成了应用程序的外部组织形式。而且，使用命名空间可以解决类名冲突问题。

要在 ASP.NET 网站中使用这些命名空间，可以使用 using 语句，如“using system;”表示导入 system 命名空间。导入命名空间后使得要访问包含的类时可省略命名空间。例如，若没有使用“using system;”语句，“string strNum = "100";”这个语句就会出现编译错误，此时就应该用“System. String strNum = "100";”代替。

注意：C#语言区分大小写。语句“System. String strNum = "100";”中的 String 首字母大写，其实这里的 String 是 System 命名空间中的一个类。而“string strNum = "100";”中的 string 表示一种数据类型。

常用于 ASP.NET 3.5 页面的命名空间有：

- System——提供基本类，如提供字符串操作的 String 类。
- System. Configuration——提供处理配置文件中数据的类，如能获取 web. config 文件中数据库连接字符串的 ConnectionStringSettings 类。
- System. Data——提供对 ADO.NET 类的访问，如提供数据缓存的 Dataset 类。
- System. Ling——提供使用 LINQ 进行查询的类和接口，如包含标准查询运算符的 Queryable 类。
- System. Web——提供使浏览器与服务器相互通信的类和接口，如用于读取客户端信息的 HttpRequest 类。
- System. Web. Security——提供在 Web 服务器实现 ASP.NET 安全性的类，如用于验证用户凭据的 MemberShip 类。
- System. Web. UI——提供用于创建 ASP.NET 网站用户界面的类和接口，如每个 Web 窗体都继承的 Page 类。
- System. Web. UI. HtmlControls——提供在 Web 窗体页上创建 HTML 服务器控件的类，如对应元素＜a＞的 HtmlAnchor 类。
- System. Web. UI. WebControls——提供在 Web 窗体页上创建 Web 服务器控件的类，如按钮 Button 控件类。
- System. Web. UI. WebControls. WebParts——提供用于创建个性化 Web 部件页的类和接口，如呈现模块化用户界面的 Part 类。
- System. Xml. Linq——提供用于 LINQ to XML 的类，如获取 XML 元素的 Xelement 类。

3.3 编 程 规 范

良好的编程规范能极大地提高程序的可读性。本节将从程序注释和命名规则两方面来介绍编程规范。

3.3.1 程序注释

注释有助于理解代码，有效的注释是指在代码的功能、意图层次上进行注释，提供有用、

额外的信息，而不是代码表面意义的简单重复。程序注释需要遵守下面的规则：

（1）类、方法、属性的注释采用 XML 文档格式注释；多行代码注释采用“/* … */”，单行代码注释采用“// …”。示例如下：

```
public class Sample
    {
        //数据成员  (单行注释)
        private int  _Property;

        /// <summary>  (XML 注释)
        /// 示例属性
        /// </summary>
        public int Property
        {
            get
            {
                return _Property;
            }
        /* set        (多行注释)
            {
                _Property = value;
            } */
        }
```

（2）类、接口头部应进行 XML 注释。注释必须列出内容摘要、版本号、作者、完成日期、修改信息等。示例如下：

```
/// <summary>
/// 版权所有：版权所有(C) 2004,
/// 内容摘要：本类的内容是…
/// 完成日期：2008 年 3 月 1 日
/// 版　　本：V1.0
/// 作　　者：小张
///
/// 修改记录 1：
/// 修改日期：2008 年 3 月 10 日
/// 版 本 号：V1.2
/// 修 改 人：小张
/// 修改内容：对方法…进行修改，修正故障 BUG…
/// 修改记录 2：
/// 修改日期：2008 年 3 月 20 日
/// 版 本 号：V1.3
/// 修 改 人：小张
/// 修改内容：对方法…进行进一步改进，修正故障…
/// </summary>
```

（3）公共方法前面应进行 XML 注释，列出方法的目的/功能、输入参数、返回值等。

（4）在{}中包含较多代码行的结束处应加注释，便于阅读，特别是多分支、多重嵌套的条件语句或循环语句。此时注释常用英文，方便查找对应的语句。示例如下：

```
void Main()
```

```
{
    if (...)
     {
     ⋮
          while (...)
          {
          ⋮
          }  / * end of while (...) * /      // 指明该条 while 语句结束
          ⋮
    }  / * end of if (...) * /               // 指明是 if 语句结束
} / * end of void main() * /                 // 指明方法的结束
```

(5) 对分支语句(条件分支、循环语句等)必须编写注释。这些语句往往是程序实现某一特殊功能的关键,对于维护人员来说,良好的注释有助于更好地理解程序,有时甚至优于看设计文档。

3.3.2 命名规则

命名时常考虑字母的大小写规则,主要有 Pascal 和 Camel 两种形式。Pascal 形式指将标识符的首字母和后面连接的每个单词的首字母都大写,如 BackColor。Camel 形式指标识符的首字母小写,而每个后面连接的单词的首字母都大写,如 backColor。常用标识符的大小写方式如表 3-1 所示。

表 3-1 常用标识符的大小写方式对应表

标识符	方式	示例	标识符	方式	示例
类	Pascal	AppDomain	接口	Pascal	IDisposable
枚举类型	Pascal	ErrorLevel	方法	Pascal	ToString
枚举值	Pascal	FatalError	命名空间	Pascal	System. Drawing
事件	Pascal	ValueChanged	参数	Camel	typeName
异常类	Pascal	WebException	属性	Pascal	BackColor
只读的静态字段	Pascal	RedValue	变量名	Camel	dateConnection

下面是命名时应遵守的其他规则:

(1) 用正确的反义词组命名具有互斥意义的变量或相反动作的函数等。常用的反义词组有 add/remove、begin/end、create/destroy、insert/delete、first/last、get/release、increment/decrement、put/get、add/delete、lock/unlock、open/close、min/max、old/new、start/stop、next/previous、source/target、show/hide、send/receive、cut/paste、up/down。

(2) 常量名都要使用大写字母,用下划线"_"分割单词,如 MIN_VALUE 等。

(3) 一般变量名不得取单个字符(如 i、j、k 等)作为变量名,局部循环变量除外。

(4) 类的成员变量(属性所对应的变量)使用前缀"_",如属性名为 Name,则对应的成员变量名为_Name。

(5) 控件命名采用"控件名简写+英文描述"形式,英文描述首字母大写。常用控件名简写对照如表 3-2 所示。

表 3-2　常用控件名简写对照表

控件名	简写	控件名	简写
Label	lbl	TextBox	txt
Button	btn	LinkButton	lnkbtn
ImageButton	imgbtn	DropDownList	ddl
ListBox	lst	DataGrid	dg
DataList	dl	CheckBox	chk
CheckBoxList	chkls	AdRotator	ar
RadioButtonList	rdolt	Table	tbl
Panel	pnl	Calender	cld
RadioButton	rdo	Image	img
RangeValidator	rv	RequiredFieldValidator	rfv
CompareValidator	cv	ValidatorSummary	vs
RegularExpressionValidator	rev		

(6) 接口命名在名字前加上 I 前缀，如 IDisposable。

3.4　常量与变量

常量和变量的使用在程序设计中必不可少。本节将介绍常量声明、变量声明、修饰符和局部变量使用范围等。

3.4.1　常量声明

常量具有在编译时值保持不变的特性，声明时使用 const 关键字，同时必须初始化。使用常量的好处主要有：常量用易于理解的名称替代了“含义不明确的数字或字符串”，使程序更易于阅读；常量使程序更易于修改。如个人所得税计算中，若使用 TAX 常量代表税率，当税率改变时，只需修改常量值而不必在整个程序中修改相应的税率。

常量的访问修饰符有 public、internal、protected internal 和 private 等。如：

```
public const string CORP = "一舟网络";
```

表示定义公共的字符型常量 CORP，值为“一舟网络”。

3.4.2　变量声明

变量具有在程序运行过程中值可以变化的特性，必须先声明再使用。变量名长度任意，可以由数字、字母、下划线等组成，但第一个字符必须是字母或下划线。C＃是区分大小写的，因此 strName 和 strname 代表不同的变量。变量的修饰符有 public、internal、protected、protected internal、private、static 和 readonly，C＃中将具有这些修饰符的变量称为字段，而把方法中定义的变量称为局部变量。

注意：局部变量前不能添加 public、internal、protected、protected internal、private、

static 和 readonly 等修饰符。

3.4.3 修饰符

1. 访问修饰符

public、internal、protected、protected internal、private 修饰符都用于设置变量的访问级别，在变量声明中只能使用这些修饰符中的一个。它们的作用范围如表 3-3 所示。

表 3-3 访问修饰符作用范围表

修饰符	作用范围
public	访问不受限制，任何地方都可访问
internal	在当前程序中能被访问
protected	在所属的类或派生类中能被访问
protected internal	在当前的程序或派生类中能被访问
private	在所属的类中能被访问

2. static

使用 static 声明的变量称静态变量，又称为静态字段。对于类中的静态字段，在使用时即使创建了多个类的实例，都仅对应一个实例副本。访问静态字段时只能通过类直接访问，而不能通过类的实例来访问。

3. readonly

使用 readonly 声明的变量称为只读变量，这种变量被初始化后在程序中不能修改它的值。

3.4.4 变量作用范围

1. 块级

块级变量是作用域范围最小的变量，如包含在 if、while 等语句段中的变量。这种变量仅在块内有效，在块结束后即被删除。如下面程序段中的 strName 变量，在程序段结束之后即不能被访问。

```
if (nSum == 1)
{
      string strName = "张三";          // strName 是块级变量
 }
lblMessage.Text = strName;              //不能访问 strName，会产生编译错误
```

2. 方法级

方法级变量作用于声明变量的方法中，在方法外即不能访问。

```
protected void Page_Load(object sender, EventArgs e)
{
    string strName = "张三";          // strName 是方法级变量
}
protected void Button1_Click(object sender, EventArgs e)
{
    lblMessage.Text = strName;        //不能访问 strName,会产生编译错误
}
```

3. 对象级

对象级变量可作用于定义类的所有方法中,只有相应的 ASP.NET 页面结束时才被删除。

```
public partial class _Default : System.Web.UI.Page
{
     string strName = "张三";          // strName 是对象级变量
     protected void Page_Load(object sender, EventArgs e)
     {
         strName = "李四";
     }
     protected void Button1_Click(object sender, EventArgs e)
     {
         lblMessage.Text = strName;    //能访问 strName
     }
}
```

3.5　数 据 类 型

C#数据类型有值类型和引用类型两种。值类型的变量直接包含它们的数据,而引用类型存储对它们的数据的引用。对于值类型,一个变量的操作不会影响另一个变量;而对于引用类型,两个变量可能引用同一个对象,因此对一个变量的操作可能会影响到另一个变量。本节将介绍值类型、引用类型、装箱和拆箱等。

3.5.1　值类型

值类型分为简单类型、结构类型、枚举类型。简单类型再分为整数类型、布尔类型、字符类型和实数类型。

1. 简单类型

1) 整数类型

整数类型的值都为整数,在具体编程时应根据实际需要选择合适的整数类型,以免造成存储资源浪费。各整数类型的位数、取值范围等如表 3-4 所示。

表 3-4 整数类型对应表

类别	位数	类型	范围/精度
有符号整型	8	sbyte	−128～127
	16	short	−32 768～32 767
	32	int	−2 147 483 648～2 147 483 647
	64	long	−9 223 372 036 854 775 808～9 223 372 036 854 775 807
无符号整型	8	byte	0～255
	16	ushort	0～65 535
	32	uint	0～4 294 967 295
	64	ulong	0～18 446 744 073 709 551 615

2）布尔类型

布尔类型表示“真”和“假”，用 true 和 false 表示。

注意：布尔类型不能用整数类型代替，如数字 0 不能代替 false。

3）字符类型

字符类型采用 Unicode 字符集标准，一个字符长度为 16 位。字符类型的赋值形式有：

```
char x1 = 'A';        // 一般方式，值为字符 A
char x2 = '中';       //值为汉字“中”
char x3 = '\x0041';   // 十六进制方式，值为字符 A
char x4 = '\u0041';   //Unicode 方式，值为字符 A
char x5 = '\'';       //转义符方式，值为单引号'，其中等号右边输入是“单引号、\、单引号、单引号”
```

常用的转义符如表 3-5 所示。

表 3-5 常用转义符对应表

转义符	对应字符	转义符	对应字符
\'	单引号	\a	感叹号
\"	双引号	\n	换行
\\	反斜杠	\r	回车
\0	空字符	\b	退格

注意：char 类型变量声明时必须包含在一对单引号中。如语句“char x6＝"A";”编译时将出错。

4）实数类型

实数类型分为单精度 float 类型、双精度 double 类型和十进制 decimal 类型。其中 float、double 类型常用于科学计算，decimal 常用于金融计算，相应的位数、取值范围等如表 3-6 所示。

表 3-6 实数类型对应表

类别	位数	类型	范围/精度
浮点型	32	Float	1.5×10^{-45}～3.4×10^{38}，7 位精度
	64	Double	5.0×10^{-324}～1.7×10^{308}，15 位精度
小数	128	Decimal	1.0×10^{-28}～7.9×10^{28}，28 位精度

注意：float 类型必须在数据后添加 F 或 f，decimal 类型必须添加 M 或 m，否则编译器以 double 类型处理，如“float fNum＝12.6f;”。

2. 结构类型

把一系列相关的变量组织在一起形成一个单一实体，这种类型叫结构类型，结构体内的每个变量称为结构成员。结构类型的声明使用 struct 关键字。下面的示例代码声明学生信息 StudentInfo 结构，其中包括 Name、Phone、Address 成员。

```
public struct StudentInfo
{
public string Name;
public string phone;
public string Address;
}
StudentInfo stStudent;     // stStudent 为一个 StudentInfo 结构类型变量
```

对结构成员访问使用“结构变量名.成员名”形式，如“stStudent.Name＝"张三";”。

3. 枚举类型

枚举类型是由一组命名常量组成的类型，使用 enum 关键字声明。枚举中每个元素默认是整数类型，且第一个值为 0，后面每个连续的元素依次加 1 递增。若要改变默认起始值 0，可以通过直接给第一个元素赋值的方法。枚举类型的变量在某一时刻只能取某一枚举元素的值。

实例 3-1　枚举类型变量应用

本实例首先定义枚举类型 Color，再声明 enTest 枚举变量，最后以两种形式输出 enTest 值。

源程序：enum.aspx 部分代码

```
＜％@ Page Language＝"C＃" AutoEventWireup＝"true" CodeFile＝"enum.aspx.cs" Inherits＝
"chap3_enum" ％＞
...(略)
```

源程序：enum.aspx.cs

```
using System;
public partial class chap3_enum ：System.Web.UI.Page
{
    //声明枚举类型 Color
    enum Color
    {
        Red ＝ 1, Green, Blue
    }
    protected void Page_Load(object sender, EventArgs e)
    {
        Color enTest＝Color.Green;
        int i＝(int) Color.Green;
        Response.Write("enTest 的值为：" ＋ enTest ＋ "＜/br＞");     //输出 Green
```

```
            Response.Write("i 的值为：" + i);    //输出 2
        }
    }
```

操作步骤：

(1) 在 chap3 文件夹中新建 enum.aspx。

(2) 在 enum.aspx.cs 中输入阴影部分内容。最后，浏览 enum.aspx，呈现如图 3-1 所示的界面。

图 3-1　enum.aspx 浏览效果图

3.5.2　引用类型

C#引用类型包括 class 类型、接口类型、数组类型和委托类型。

1. class 类型

class 类型定义了一个包含数据成员(字段)和函数成员(方法、属性等)的数据结构，声明使用 class 关键字。在 3.8 节中将较详细地介绍有关类的内容。

1) object 类型

作为 class 类型之一的 object 类型，在.NET Framework 实质是 System.Object 类的别名。object 类型在 C#的统一类型系统中有特殊作用，所有类型(预定义类型、用户定义类型、引用类型和值类型)都是直接或间接地从 System.Object 类继承，因此，可以将任何类型的数据转化为 object 类型。

2) string 类型

另外一种作为 class 类型的 string 类型在 C#中实质是一种数组，即字符串可看作是一个字符数组。在声明时要求放在一对双引号之间。对于包含“\”字符的字符串，要使用转义符形式，如下面的示例代码：

```
string strPath = "c:\\ASP\\default.aspx";
```

对需要转义符定义的字符串，C#中的@字符提供了另一种解决方法，即在字符串前加上@后，字符串的所有字符都会被看作原来的含义，如上面的示例代码可写成：

```
string strPath = @"c:\ASP\default.aspx";
```

另外，[]运算符可访问字符串中各个字符，如：

```
string strTest = "abcdefg";
char x = strTest[2];     //x 的值为'c'
```

注意：string 类型声明需要一对双引号，而 char 类型声明需要一对单引号。

实际编程时经常遇到要将其他数据类型转化为 string 类型的情形，这可以通过 ToString()方法实现。如：

```
string strInt = 23.ToString();
```

ToString()方法还提供了很实用的用于转换成不同格式的参数，如下面示例中 P 表示百分比格式，D 表示长日期格式，其他的参数详见 MSDN。

```
Response.Write(0.234.ToString("P"));          //输出 23.4 %
Response.Write(DateTime.Now.ToString("D")); //输出系统日期“2008 年 4 月 21 日”
```

若要将 string 类型转化为其他类型，可使用 Parse()方法或 Convert 类的相应方法，如：

```
int iString = Int32.Parse("1234");                    //将 string 类型转化为 int 类型
string strDatetime = Convert.ToString(DateTime.Now);  //将日期型转化为字符型
```

2. 接口类型

接口常用来描述组件对外能提供的服务，如组件与组件之间、组件和用户之间的交互都是通过接口完成。接口中不能定义数据，只能定义方法、属性、事件等。包含在接口中的方法不定义具体实现，而是在接口的继承类中实现。

3. 数组类型

数组是一组数据类型相同的元素集合。要访问数组中的元素时，可以通过“数组名[下标]”形式获取，其中下标编号从 0 开始。数组可以是一维的，也可以是多维的。下面是数组声明的多种形式：

```
string[] s1;                                    //定义一维数组，但未初始化值
int[] s2 = new int[] { 1, 2, 3 };               //定义一维数组并初始化
int[,] s3 = new int[,] { { 1, 2 }, { 4, 5 } };  //定义二维数组并初始化
```

4. 委托类型

委托是一种安全地封装方法的类型，类似于 C 和 C++中的函数指针。与 C 中的函数指针不同，委托是类型安全的。通过委托可以将方法作为参数或变量使用。

3.5.3 装箱和拆箱

装箱和拆箱是实现值类型和引用类型相互转换的桥梁。装箱的核心是把值类型转化为对象类型，也就是创建一个对象并把值赋给对象。示例代码如下：

```
int i = 100;
object objNum = i;     //装箱
```