第 3 章

C#和 ASP.NET 4.5

本章要点：

- 了解 C#语言特点和编程规范。
- 了解常用.NET Framework 命名空间。
- 结合 ASP.NET 4.5 页面熟悉 C#语言的运用。
- 能结合 ASP.NET 4.5 页面创建简单的类。
- 掌握 ASP.NET 4.5 页面调试的方法。

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#的类型即为.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 4.5 本身就采用 C#语言开发，所以 C#不仅适用于 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 .NET Framework 命名空间

.NET Framework 提供了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。

要在 ASP.NET 网站中使用这些命名空间，需要使用 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，则 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 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 string 表示一种数据类型。

常用于 ASP.NET 4.5 页面的命名空间有：

- System——提供基本类。
- System.Configuration——提供处理配置文件中数据的类。
- System.Data——提供对 ADO.NET 类的访问。
- System.Ling——提供使用 LINQ 进行查询的类和接口。
- System.Web——提供使浏览器与服务器相互通信的类和接口。
- System.Web.Security——提供实现 ASP.NET 安全性的类。
- System.Web.UI——提供用于创建 Web 应用程序用户界面的类和接口。
- System.Web.UI.HtmlControls——提供在 Web 窗体上创建 HTML 服务器控件的类。
- System.Web.UI.WebControls——提供在 Web 窗体上创建 Web 服务器控件的类。
- System.Web.UI.WebControls.WebParts——提供用于创建个性化 Web 部件页的类和接口。
- System.Xml.Linq——提供用于 LINQ to XML 的类。

3.3　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// …。

（2）类、接口头部应进行 XML 注释。注释应列出内容摘要、版本号、作者、完成日期、修改信息等。

（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，特别是多分支、多重嵌套的条件语句或循环语句。

（5）对分支语句（条件分支、循环语句等）应编写注释。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 形式将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
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	strName

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。

（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。建议采用如表 3-2 所示的常用控件名简写规范。

（6）接口命名在名字前加上 I 前缀，如 IDisposable。

表 3-2　建议的常用控件名简写规范表

控　件　名	简　　写	控　件　名	简　　写	控　件　名	简　　写
Label	lbl	TextBox	txt	RadioButton	rdo
Button	btn	LinkButton	lnkbtn	Image	img
ImageButton	imgbtn	DropDownList	ddl	RangeValidator	rv
ListBox	lst	GridView	gv	RequiredFieldValidator	rfv
DataList	dl	CheckBox	chk	CompareValidator	cv
CheckBoxList	chkls	AdRotator	ar	ValidatorSummary	vs
RadioButtonList	rdolt	Table	tbl	RegularExpressionValidator	rev
Panel	pnl	Calendar	cld		

3.4　常量与变量

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";  //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

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　访问修饰符的作用范围表

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

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。

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

3.4.4　局部变量作用范围

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 btnSubmit_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 btnSubmit_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。值类型变量直接包含它们的数据，而引用类型变量存储它们的数据的引用。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。

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。

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 c1='A';         //一般方式，值为字符 A
char c2='中';        //值为汉字“中”
char c3='\x0041';    //十六进制方式，值为字符 A
char c4='\u0041';    //Unicode 方式，值为字符 A
char c5='\'';   //转义符方式，值为单引号'，其中等号右边是“单引号、\、单引号、单引号”
```

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 c6="A";”编译时将

出错。

4）实数类型

实数类型分为 float 单精度类型、double 双精度类型和 decimal 十进制类型。其中 float、double 类型常用于科学计算，decimal 类型常用于金融计算。

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来改变。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　运用枚举类型变量

本实例首先定义枚举类型 Color，再声明 enumColor 枚举变量，最后以两种形式输出 enumColor 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
{
  enum Color  //声明枚举类型 Color
  {
    Red = 1, Green, Blue
  }
  protected void Page_Load(object sender, EventArgs e)
  {
```

```
        Color enumColor = Color.Green;
        int i = (int)Color.Green;
        Response.Write("enumColor 的值为: " + enumColor + "<br />");  //输出 Green
        Response.Write("i 的值为: " + i);  //输出 2
    }
}
```

操作步骤：

在 Chap3 文件夹中新建 Enum.aspx 和 Enum.aspx.cs。在 Enum.aspx.cs 中输入阴影部分内容。浏览 Enum.aspx 呈现如图 3-1 所示的界面。

图 3-1　Enum.aspx 浏览效果

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 中实质是 System.Object 类的别名。object 类型在 C#的统一类型系统中有特殊作用，所有其他类型（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\等字符的字符串，要使用转义符形式，如下面的示例代码：

```
string strPath = "C:\\ASP.NET\\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，字符串中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.NET\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换为 string 类型的情形，这可以通过 ToString()方法实现，如：

```
string strInt = 23.ToString();  //int 类型转换为 string 类型
```

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%
//输出当前系统日期，形式如“2013 年 12 月 21 日”
Response.Write(DateTime.Now.ToString("D"));
```

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

```
int iString = Int32.Parse("1234");     //将 string 类型转换为 int32 类型
//将日期类型转换为 string 类型
string strDatetime = Convert.ToString(DateTime.Now);
```

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

委托是一种安全的封装方法的类型，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换为对象类型，也就是创建一个对象并把值赋给对象，如：

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

拆箱的核心是把对象类型转换为值类型，即把值从对象实例中复制出来，如：

```
int i = 100;
object objNum = i;   //装箱
int j = (int)objNum;  //拆箱
```

3.6 运 算 符

表 3-4 总结了 C#中常用的运算符，并按优先级从高到低的顺序列出。

表 3-4 运算符对应表

类 别	表 达 式	说 明
基本	x.m	成员访问
	x(…)	方法和委托调用
	x[…]	数组和索引器访问
	x++	后增量
	x--	后减量
	new T(…)	对象和委托创建
	new T(…){…}	使用初始值设定项创建对象
	new {…}	匿名对象初始值设定项
	new T[…]	数组创建
	typeof(T)	获得 T 的 System.Type 对象
一元	-x	求相反数
	!x	逻辑求反
	~x	按位求反
	++x	前增量
	--x	前减量
	(T)x	显式地将 x 转换为类型 T
乘除	x * y	乘法
	x / y	除法
	x % y	求余
加减	x + y	加法、字符串串联、委托组合
	x – y	减法、委托移除
移位	x << y	左移
	x >> y	右移
关系和类型检测	x < y	小于
	x > y	大于
	x <= y	小于或等于
	x >= y	大于或等于
	x is T	如果 x 属于 T 类型，则返回 true，否则返回 false
	x as T	返回转换为类型 T 的 x，如果 x 不是 T，则返回 null
逻辑操作	x == y	若 x 等于 y，则为 true，否则 false
	x != y	若 x 不等于 y，则为 true，否则 false
	x & y	整型按位 AND、布尔逻辑 AND
	x ^ y	整型按位 XOR、布尔逻辑 XOR
	x | y	整型按位 OR、布尔逻辑 OR
	x && y	仅当 x 为 true 时才对 y 求值，再执行布尔逻辑 AND 操作
	x || y	仅当 x 为 false 时才对 y 求值，再执行布尔逻辑 OR 操作
条件	x ? y : z	如果 x 为 true，则对 y 求值并返回 y 的值；如果 x 为 false，则对 z 求值并返回 z 的值
赋值或匿名函数	x = y	赋值
	x op y	复合赋值；支持 op 运算符有： *= /= %= += -= <<= >>= &= ^= |=
	(T x) => y	Lambda 表达式

3.7　流 程 控 制

与其他语言类似，C#提供了选择、循环等结构。用于选择结构的有 if 和 switch 语句；用于循环结构的有 while、do-while、for 和 foreach 语句。

3.7.1　选择结构

1. if 语句

语法格式一：

```
if (条件表达式) { 语句序列 }
```

执行顺序：计算条件表达式。若值为 true，则执行“语句序列”；否则执行 if 语句的后续语句。

语法格式二：

```
if (条件表达式) { 语句序列 1 }
else { 语句序列 2 }
```

执行顺序：计算条件表达式。若值为 true，则执行“语句序列 1”；否则执行“语句序列 2”。

注意：条件表达式在判断是否相等时一定要用==。

2. switch 语句

if 语句实现的是两路分支功能，若要用 if 语句实现两路以上的分支时，必须嵌套 if 语句。而使用 switch 语句能很方便地实现多路分支功能。语法格式如下：

```
switch (控制表达式)
{
  case 常量 1:
   语句序列 1
  case 常量 2:
   语句序列 2
      ⋮
  default:
   语句序列 n
}
```

执行顺序：计算控制表达式。若值与某一个 case 后面的常量值匹配，则执行此 case 块中的语句；若值与所有 case 后面的常量值均不匹配，则执行 default 语句块。

实例 3-2　运用 switch 语句

如图 3-2 所示，本实例根据今天是星期几在页面上输出相应信息。

图 3-2　Switch.aspx 浏览效果

源程序：Switch.aspx 部分代码

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Switch.aspx.cs"
 Inherits="Chap3_Switch" %>
…（略）
```

源程序：Switch.aspx.cs

```
using System;
public partial class Chap3_Switch : System.Web.UI.Page
{
  protected void Page_Load(object sender, EventArgs e)
  {
    DateTime dtToday = DateTime.Today;    //获取今天的系统日期
    switch (dtToday.DayOfWeek.ToString())  //枚举值转换为字符型
    {
      case "Monday":
        Response.Write("今天是星期一！");
        break;
      case "Tuesday":
        Response.Write("今天是星期二！");
        break;
      case "Wednesday":
        Response.Write("今天是星期三！");
        break;
      case "Thursday":
        Response.Write("今天是星期四！");
        break;
      case "Friday":
        Response.Write("今天是星期五！");
        break;
      default:
        Response.Write("今天可以休息了！");
        break;
    }
  }
}
```

3.7.2 循环结构

1. while 语句

while 语句根据条件表达式的值，执行 0 次或多次循环体。语法格式如下：

```
while (条件表达式) { 语句序列 }
```

执行顺序：

（1）计算条件表达式。

（2）若条件表达式的值为 true，则执行循环体中语句序列，然后返回（1）；否则执行 while 后续语句。

实例 3-3　运用 while 语句

本实例在页面上的文本框中输入一个值 n，单击“确定”按钮后计算 1+3+…+n，再在一个标签控件中输出计算值。

源程序：While.aspx 部分代码

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="While.aspx.cs"
 Inherits="Chap3_While" %>
          ⋮
<form id="form1" runat="server">
  <div>
    <asp:TextBox ID="txtInput" runat="server">请输入一个数字</asp:TextBox>
    <asp:Label ID="lblOutput" runat="server"></asp:Label><br/>
    <asp:Button ID="btnSubmit" runat="server"Text="确定" OnClick=
      "btnSubmit_Click" />
  </div>
</form>
   ⋮
```

源程序：While.aspx.cs

```
using System;
public partial class Chap3_While : System.Web.UI.Page
{
  protected void btnSubmit_Click(object sender, EventArgs e)
  {
    int iSum = 0;                    //iSum 存放和
    int iInput = int.Parse(txtInput.Text);//iInput 存放类型转换后的文本框输入值
    int i=1;                        //循环变量 i
    while (i<=iInput)
    {
      iSum+=i;
      i+=2;
    }
    lblOutput.Text="和为："+iSum.ToString();
  }
}
```

操作步骤：

在 Chap3 文件夹中建立 While.aspx，添加 TextBox、Label 和 Button 控件各一个，参考源程序设置各控件属性；在 While.aspx.cs 中输入代码。浏览 While.aspx 呈现如图 3-3 所示的界面；在文本框中输入 100，单击“确定”按钮后呈现如图 3-4 所示的界面。

图 3-3　While.aspx 浏览效果（1）

图 3-4　While.aspx 浏览效果（2）

2. do-while 循环

语法格式如下：

```
do { 语句序列 }
while (条件表达式)
```

执行顺序：

（1）执行循环体内语句序列。

（2）计算条件表达式，若值为 true，则返回（1）；否则执行后续语句。

注意：与 while 语句不同，do-while 循环体内语句序列会在计算条件表达式之前执行一次。

3. for 语句

for 语句适用于循环次数已知的循环，循环体内语句序列可能执行 0 次或多次。语法格式如下：

```
for (循环变量初始化; 条件表达式; 循环控制表达式) { 语句序列 }
```

执行顺序：

（1）初始化循环变量，并赋初值。

（2）计算条件表达式，若值为 true，则执行循环体内语句序列；否则跳出循环。

（3）根据循环控制表达式改变循环变量的值，返回（2）。

注意：当使用 for (; ;)形式时表示死循环，需要使用 break 语句跳出。

实例 3-4　运用 for 语句

本实例利用 for 语句在页面上输出三角形。

源程序：For.aspx 部分代码

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="For.aspx.cs"
  Inherits="Chap3_For" %>
…（略）
```

源程序：For.aspx.cs

```
using System;
public partial class Chap3_For : System.Web.UI.Page
{
  protected void Page_Load(object sender, EventArgs e)
  {
    for (int i=1;i<5;i++)            //i 控制行数
    {
      for (int k=1;k<=20-2*i;k++)    //控制输出每行前的空格数
      {
        Response.Write(" ");
      }
      for (int j=1;j<=2*i-1;j++)     //控制输出每行的*数
      {
        Response.Write("*");
      }
      Response.Write("<br/>");       //换行
```

```
    }
  }
}
```

操作步骤：

在 Chap3 文件夹中建立 For.aspx 和 For.aspx.cs，浏览 For.aspx 呈现如图 3-5 所示的界面。

图 3-5　For.aspx 浏览效果

4. foreach 语句

foreach 语句常用于枚举数组、集合中的每个元素，并针对每个元素执行循环体内语句序列。foreach 语句不能改变集合中各元素的值。语法格式如下：

```
foreach (数据类型 循环变量 in 集合) { 语句序列 }
```

实例 3-5　运用 foreach 语句

本实例先给一个 strNames 数组赋值，再逐个输出数组元素。

源程序：Foreach.aspx 部分代码

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Foreach.aspx.cs"
  Inherits="Chap3_Foreach" %>
…（略）
```

源程序：Foreach.aspx.cs

```
using System;
public partial class Chap3_Foreach : System.Web.UI.Page
{
  protected void Page_Load(object sender, EventArgs e)
  {
    string[] strNames = { "张犯", "周振", "王涛" };    //数组赋值
    Array.Sort(strNames);                              //升序排列数组
    foreach (string n in strNames)                     //逐个输出数组元素
    {
      Response.Write("姓名: "+n+"<br/>");
    }
  }
}
```

操作步骤：

在 Chap3 文件夹中建立 Foreach.aspx 和 Foreach.aspx.cs。浏览 Foreach.aspx 呈现如图 3-6 所示的界面。

图 3-6　Foreach.aspx 浏览效果

3.7.3　异常处理

异常的产生常由于触发了某个异常的条件，使得操作无法正常进行，如算术运算中的除零操作、内存不足、数组索引越界等。异常处理能使程序更加健壮，容易让程序员对捕获的错误进行处理。异常处理常使用两种形式：throw 语句和 try…catch…finally 结构。

1. throw 语句

throw 语句用于抛出异常错误信息。它可以在 try…catch…finally 结构的 catch 块中使用，也可以在其他的结构中使用，如 if 语句。

实例 3-6　运用 throw 语句

本实例实现当除零操作时，抛出“除数不能为零！”的错误信息。

源程序：Throw.aspx 部分代码

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Throw.aspx.cs"
  Inherits="Chap3_Throw" %>
…（略）
```

源程序：Throw.aspx.cs

```
using System;
public partial class Chap3_Throw : System.Web.UI.Page
{
  protected void Page_Load(object sender, EventArgs e)
  {
    int i=10;
    int j=0;
    int k;
    if (j==0)
    {
      throw new Exception("除数不能为零！");
    }
    else
    {
      k=i/j;
      Response.Write(k);
    }
  }
}
```

操作步骤：

在 Chap3 文件夹中建立 Throw.aspx 和 Throw.aspx.cs。浏览 Throw.aspx 呈现如图 3-7 所示的界面。

程序说明：

本实例主要为了说明 throw 语句的应用。在实际工程中，*j* 变量直接赋值为 0 再进行判断是否为 0 毫无意义。

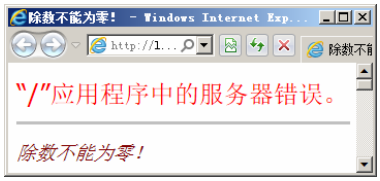

图 3-7　Throw.aspx 浏览效果

2.　try…catch…finally 结构

在 try…catch…finally 结构中，异常捕获由 try 块完成，处理异常的代码放在 catch 块中，而在 finally 块中的代码不论是否有异常发生总会被执行。其中，catch 块可包含多个，而 finally 块是可选的。在实际应用中，finally 块常完成一些善后工作，如网盘文件读写操作中的文件关闭等。语法格式如下：

```
try  { 可能出错的语句序列 }
catch (异常声明 1)  { 捕获异常后执行的语句序列 1 }
catch (异常声明 2)  { 捕获异常后执行的语句序列 2 }
    ⋮
finally  { 总是执行的语句块 }
```

执行顺序：

（1）执行 try 块，若出错转（2），否则转（3）。

（2）将捕获的异常信息逐个查找 catch 块中的异常声明，若匹配则执行内嵌语句序列。

（3）执行 finally 块。

实例 3-7　运用 try…catch…finally 结构

本实例的 ExceptionNo.aspx.cs 未包含 try…catch…finally 结构，浏览 ExceptionNo.aspx 时因为将读取的文件块存放到 buffer 数组时超出了数组界限而给出系统报错信息，如图 3-8 所示。Exception.aspx.cs 中包含了 try…catch…finally 结构，当 try 块执行出错时将执行 catch 块，因此，浏览 Exception.aspx 时显示系统错误信息和开发人员定义的错误信息，如图 3-9 所示。

图 3-8　ExceptionNo.aspx 浏览效果

图 3-9　Exception.aspx 浏览效果

源程序：ExceptionNo.aspx 部分代码

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="ExceptionNo.aspx.cs"
  Inherits="Chap3_ExceptionNo" %>
…（略）
```

源程序：ExceptionNo.aspx.cs

```
using System;
using System.IO;
public partial class Chap3_ExceptionNo : System.Web.UI.Page
{
  protected void Page_Load(object sender, EventArgs e)
  {
    //定义要读取文件的物理路径
    string filePath = @"F:\Book\ChapSite\Chap3\Test.txt";
    //定义 streamReader 对象
    StreamReader streamReader = new StreamReader(filePath);
    char[] buffer = new char[5];
    //从文件中读取内容到 buffer 数组
    streamReader.ReadBlock(buffer, 0, 10);
    //关闭 streamReader 对象，释放占用的资源
    streamReader.Close();
  }
}
```

源程序：Exception.aspx 部分代码

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Exception.aspx.cs"
  Inherits="Chap3_Exception" %>
…（略）
```

源程序：Exception.aspx.cs

```
using System;
using System.IO;
public partial class Chap3_Exception : System.Web.UI.Page
{
  protected void Page_Load(object sender, EventArgs e)
  {
    string filePath = @"F:\Book\ChapSite\Chap3\Test.txt";
    StreamReader streamReader = new StreamReader(filePath);
    char[] buffer = new char[5];
    try
    {
      streamReader.ReadBlock(buffer, 0, 10);
    }
    catch (Exception ee)
    {
      Response.Write("系统提示：" + ee.Message + "<br/>");//输出捕获的错误信息
```

```
        Response.Write("用户自定义：" + "读取文件错误！");  //输出用户自定义的错误信息
      }
      finally
      {
        streamReader.Close();
      }
    }
  }
```

3.8　自定义 ASP.NET 4.5 类

ASP.NET 4.5 是完全面向对象的，任何对象都由类生成，而自定义类能进一步扩展功能。

3.8.1　类的常识

.NET 的底层全部是用类实现的，不管是界面上的按钮，还是前面介绍的数据类型。在考虑实现 ASP.NET 4.5 网站功能时要尽量从类的角度去实现。那么，什么是类呢？简单地说，类就是一种模板，通过类的实际例子（实例）就能使用模板中定义的属性、方法等。类具有封装性、继承性和多态性的特点。封装性指的是将具体实现方法封闭起来，只向用户暴露属性、方法等。也就是说，用户不需要知道类内部到底如何实现的，只要会调用属性和方法就可以了。继承性指的是一个类可以继承另一个类的特征（属性、方法、事件等）。多态性指的是具有继承关系的不同类拥有相同的方法名称，当调用这些类的相同方法时，执行的动作却不一样。与多态性概念容易混淆的是重载方法，它常用于在同一个类中定义多个方法名相同但参数不同的方法。

与 ASP.NET 4.5 页面对应的类包含在.aspx.cs 文件中。而对自定义的类应该放在 App_Code 文件夹中，VSEW 2012 会自动编译该文件夹中包含的类，并且在使用这些类时能得到智能感知的支持。

创建类的语法格式如下：

```
修饰符 class 类名 {…}
```

类创建完后，使用 new 关键字可建立类的实例对象。类的常用修饰符主要有访问修饰符 abstract、static、partial、sealed。

abstract 修饰符表示该类只能是其他类的基类，又称为抽象类，对这种类中的成员必须通过继承来实现。

static 修饰符表示该类为静态类，这种类在使用时不能使用 new 创建类的实例，但能够直接访问数据和方法。

partial 修饰符在 ASP.NET 4.5 网站开发中使用相当频繁，在每个.aspx 文件对应的.aspx.cs 文件中定义的类都包含了该修饰符。使用 partial 可以将类的定义拆分到两个或多个源文件中。每个源文件包含定义的一部分，当编译 Web 应用程序时，.NET Framework 会将所有部分组合起来形成一个类。

sealed 修饰符表示该类为密封类，意味着该类不能被继承。

下面将结合一个简单的银行账户类 Account 说明创建一个类时通常涉及的属性、构造函

数、方法、事件和继承等。

3.8.2 属性

通过属性可以获取或改变类中私有字段的内容，这种方式充分地体现了封装性，即不直接操作类的数据内容，而是通过访问器进行访问。访问器有 get 访问器和 set 访问器，分别用于获取和设置属性值。当仅包含 get 访问器时，表示该属性是只读的。

实例 3-8　定义 Account 类的属性

本实例定义 Account 类的三个属性：账户编号（ID）、账户所有者姓名（Name）、账户金额（Balance）。

源程序：Account.cs 属性代码

```
public class Account
{
  private string _ID;        //定义_ID 私有字段，对应 ID 属性。注意下划线前有一个空格
  private string _Name;     //定义_Name 私有字段，对应 Name 属性
  private decimal _Balance;  //定义_Balance 私有字段，对应 Balance 属性
  public string ID
  {
    get { return _ID; }
    set { _ID = value; }
  }
  public string Name
  {
    get { return _Name; }
    set { _Name = value; }
  }
  public decimal Balance
  {
    get { return _Balance; }
    set { _Balance = value; }
  }
}
```

操作步骤：

右击 App_Code 文件夹，在弹出的快捷菜单中选择“添加”→“添加新项”命令，然后在呈现的对话框中选择“类”模板，输入文件名 Account.cs，单击“添加”按钮建立文件。再输入源程序内容。

3.8.3 构造函数

当使用 new 关键字实例化一个对象时，将调用对象的构造函数，所以说，在使用一个类时，最先执行的语句就是构造函数中的语句。每个类都有构造函数，如果没有定义构造函数，编译器会自动提供一个默认的构造函数。

注意：构造函数名与类名相同且总是 public 类型。

实例 3-9　定义 Account 类的构造函数

本实例在银行账户类 Account 中构建一个对应的构造函数。

源程序：Account.cs 构造函数代码

```
public Account(string id, string name, decimal balance)
{
  _ID = id;                    //将 id 参数值传递给_ID 私有字段
  _Name = name;                //将 name 参数值传递给_Name 私有字段
  _Balance = balance;          //将 balance 参数值传递给_Balance 私有字段
}
```

操作步骤：

在 Account.cs 文件中输入源程序内容。

程序说明：

从源程序中可看出，构造函数常用于实例化类时将参数值带入对象中的情形，如建立对象时使用：

```
Account account = new Account("03401", "李明", 140);
```

这表示将"03401"、"李明"、140 等参数值分别传递给对象中的_ID、_Name、_Balance 等私有字段。

3.8.4　方法

方法反映了对象的行为。方法的常用修饰符有访问修饰符 void 等。其中，void 修饰符指定的方法不返回值。

实例 3-10　定义 Account 类的存款和取款方法

存款方法先检查存款的金额是否大于 0，若大于 0 则将原账户金额与存款金额相加保存为新的账户金额，否则抛出异常信息。取款方法先检查取款金额是否小于原账户金额，若是则将原账户金额减去取款金额，再保存为新的账户金额，否则抛出异常。

源程序：Account.cs 方法代码

```
/// <summary>
/// 存款方法
/// </summary>
/// <param name="amount">存款金额</param>
public void Deposit(decimal amount)
{
  if (amount > 0)
  {
    _Balance+=amount;
  }
  else
  {
    throw new Exception("存款金额不能小于或等于 0！");
```

```
  }
}
/// <summary>
/// 取款方法
/// </summary>
/// <param name="amount">取款金额</param>
public void Acquire(decimal amount)
{
  if (amount<_Balance)
  {
    _Balance-=amount;
  }
  else
  {
    throw new Exception("账户金额不足！");
  }
}
```

操作步骤：

在 Account.cs 文件中输入源程序内容。

实例 3-11　结合 Account 类和 ASP.NET 4.5 页面

源程序：AccountPage.aspx 部分代码

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="AccountPage.aspx.cs"
Inherits="Chap3_AccountPage" %>
…（略）
```

源程序：AccountPage.aspx.cs

```
using System;
public partial class Chap3_AccountPage : System.Web.UI.Page
{
  protected void Page_Load(object sender, EventArgs e)
  {
    Account account=new Account("03401", "李明", 200);  //建立 account 对象
    //输出初始金额信息
    Response.Write("初始金额为：" + account.Balance.ToString()+"<br/>");
   account.Deposit(100);  //存款 100
   //输出存款 100 后账户金额信息
    Response.Write("存款 100 后，"+account.Name+"的账户金额为："
       +account.Balance.ToString()+"<br />");
    account.Acquire(150);  //取款 150
    //输出取款 150 后账户金额信息
    Response.Write("取款 150 后，"+account.Name+"的账户金额为："
      +account.Balance.ToString());
  }
}
```

操作步骤：

在 Chap3 文件夹中建立 AccountPage.aspx 和 AccountPage.aspx.cs。浏览 AccountPage.aspx 呈现如图 3-10 所示的界面。

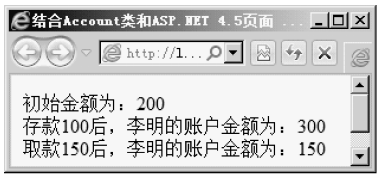

图 3-10　AccountPage.aspx 浏览效果

程序说明：

new Account("03401", "李明", 200)调用 Account()构造函数创建实例对象。

account.Balance.ToString()获取 account 对象的 Balance 属性值，并转化为 string 类型数据。

account.Deposit(100)表示调用 account 对象的 Deposit()方法。

3.8.5　事件

事件是一种用于类和类之间传递消息或触发新的行为的编程方式。通过提供事件的句柄，能够把控件和可执行代码联系在一起，如用户单击 Button 控件触发 Click 事件后就执行相应的事件处理代码。

事件的声明通过委托来实现。先定义委托，再用委托定义事件，触发事件的过程实质是调用委托。事件声明语法格式如下：

```
public delegate void EventHandler(object sender, EventArgs e);  //定义委托
public event EventHandler MyEvent;                              //定义事件
```

EventHandler 委托定义了两个参数，分别属于 object 类型和 EventArgs 类型。如果需要更多参数，可以通过派生 EventArgs 类实现。sender 表示触发事件的对象，e 用于在事件中传递参数。例如，若用户单击 Button 按钮，则 sender 表示 Button 按钮，e 表示 Click 事件参数。

MyEvent 事件使用 EventHandler 委托定义，其中使用了 public 修饰符，也可以使用 private、protected 等修饰符。

实例 3-12　在 AccountEvent 类中增加账户金额不足事件并运用事件

为避免与 Account 类冲突，本实例在 Account 类基础上新建一个 AccountEvent 类，定义的账户金额不足事件 Overdraw 将在取款时账户金额不足的情况下被触发。

源程序：AccountEvent.cs 中 Overdraw 事件代码

```
public event EventHandler Overdraw;  //定义 Overdraw 事件
  ⋮
public void OnOverdraw(object sender, EventArgs e)
{
  if (Overdraw != null)
  {
    Overdraw(this, e);
```

```
    }
  }
```

注意：定义的事件名前无 On，而对应的方法名前加 On，如 OnOverdraw 方法对应 Overdraw 事件。

定义完事件后，还需要在其他方法中设置事件的触发点。下面在 Account 类的基础上修改 Acquire()方法，在其中加入触发事件的代码。

源程序：AccountEvent.cs 中 Acquire()方法代码

```
public void Acquire(decimal amount)
{
  if (amount < _Balance)
  {
    _Balance -= amount;
  }
  else
  {
    OnOverdraw(this, EventArgs.Empty);
    return;
  }
}
```

至此，已经声明了事件并增加了事件触发点。但若要在 ASP.NET 4.5 页面上使用事件，还需要使用运算符“+=”注册事件，并要编写事件处理代码。下面说明如何运用 Overdraw 事件。

源程序：AccountEventPage.aspx 部分代码

```
<%@ Page Language="C#" AutoEventWireup="true"
  CodeFile="AccountEventPage.aspx.cs"Inherits="Chap3_AccountEventPage"%>
…（略）
```

源程序：AccountEventPage.aspx.cs

```
using System;
public partial class Chap3_AccountEventPage : System.Web.UI.Page
{
  protected void Page_Load(object sender, EventArgs e)
  {
    AccountEvent accountEvent = new AccountEvent("03401", "李明", 200);
    //注册 Overdraw 事件
    accountEvent.Overdraw += new EventHandler(account_Overdraw);
    accountEvent.Acquire(400);                              //取款 400
  }
  //Overdraw 事件处理代码
  private void account_Overdraw(object sender, EventArgs e)
  {
    Response.Write("账户金额不足了！");
```

```
    }
}
```

操作步骤：

在 Chap3 文件夹中建立 AccountEventPage.aspx 和 AccountEventPage.aspx.cs，在 AccountEventPage.aspx.cs 中输入阴影部分代码。浏览 AccountEventPage.aspx 呈现如图 3-11 所示的界面。

图 3-11　AccountEvent.aspx 浏览效果

程序说明：

当程序执行“account.Acquire(400);”时将触发 Overdraw 事件，再执行 account_Overdraw() 方法，输出信息“账户金额不足了！”。

3.8.6　继承

继承可以重用现有类的数据和行为，并扩展新的功能。继承以基类为基础，通过向基类添加成员创建派生类。通常基类又称为超类或父类，派生类又称为子类。

例如，在 Account 类中，如果针对企业账户需要增加 Type 属性，那么利用类的继承性，只要添加一个新的属性就可以了。

实例 3-13　实现继承类

本实例建立的 EnterpriseAccount 类在继承 Account 类的基础上增加了 Type 属性。

源程序：EnterpriseAccount.cs

```
public class EnterpriseAccount : Account
{
  private string _Type;  //定义_Type 私有字段，对应 Type 属性
  public string Type
  {
    get { return _Type; }
    set { _Type = value; }
  }
}
```

3.9　ASP.NET 4.5 页面调试

在实际项目开发过程中，ASP.NET 4.5 页面调试非常重要，可以说，不会调试的人永远不会编程。通过程序调试，可以检查代码并验证它们是否能够正常地运行，从而发现 VSEW 2012 编译页面过程中不能捕获的错误。对于正确执行的程序，使用调试功能还能真正地理解

程序的运行过程。

要对 ASP.NET 4.5 网站启用调试，必须将 Web 应用程序配置成调试模式，这需要配置 Web.config 文件中<system.web>元素的子元素<compilation>，示例代码如下：

```
<compilation debug="true" targetFramework="4.5">
```

断点设置是 ASP.NET 4.5 页面调试中最常用的操作。使用断点，可以通知调试器在某个特定点上暂时挂起程序的执行。此时，程序的运行处于中断模式。这种模式并不是终止或结束程序的执行，而是在任何时候都能根据调试情况确定是否需要继续执行。在中断模式下，可以检查变量的状态，还可以更改变量值以便人为地控制程序的执行过程。具体操作时，右击需要设置断点的语句，在弹出的快捷菜单中选择“断点”→“插入断点”命令即可在该语句处设置断点。

设置断点后，按 F5 键可启动调试过程。在调试过程中，使用较多的窗口包括“局部变量”、“监视”、“即时窗口”，如图 3-12 所示。“局部变量”窗口用于显示当前变量。“监视”窗口用于监视变量或表达式的值，也可以用于更改变量的值。“即时窗口”用于计算表达式、输出变量值、更改变量值等。

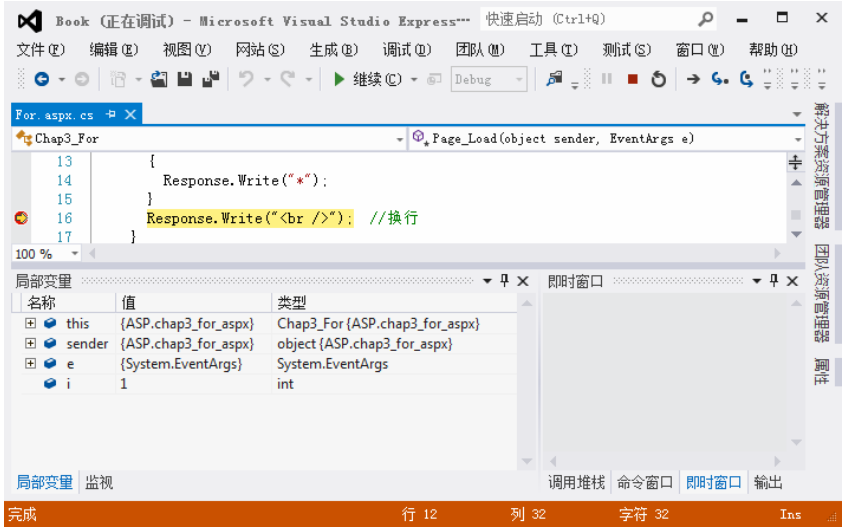

图 3-12　程序调试界面

调试过程中常用的快捷键包括 F10、F11 键。其中，F10 键用于逐过程地执行程序，而 F11 键用于逐语句地执行程序。另外，组合键 Shift+F5 用于结束程序的调试过程。

3.10　小　　结

本章主要介绍 C#基础知识，并结合 ASP.NET 4.5 页面说明 C#在网站开发中的应用。C#作为 Microsoft 专门为.NET 打造的编程语言，非常适合 ASP.NET 4.5 页面的开发。.NET Framework 命名空间提供了.NET 类的组织方式。良好的编程规范是开发人员应当遵守的规则。掌握 C#基础语法是 ASP.NET 4.5 页面开发的基础。通过装箱和拆箱能较深入地理解 C#中任何东西都可作为对象对待的实质。流程控制提供了程序的运行逻辑。异常处理能使程序

更健壮，在编程过程中需要熟练地使用。尽管.NET 类库提供了强大的功能支持，但仍有一些功能需进一步扩展，此时就需要自定义类。页面调试可以检查代码执行过程中的正确性，还能真正地理解代码的执行过程，因此，在平时的编程实践中必须加强页面调试能力的培养。

3.11　习　　题

1. 填空题

（1）C#使用的类库就是_____提供的类库。

（2）要在一个类中包含 System.Data 命名空间的语句是_____。

（3）使用_____修饰符能调用未实例化的类中的方法。

（4）C#中的数据类型包括_____和_____。

（5）_____是由一组命名常量组成的类型。

（6）在 C#统一类型系统中，所有类型都是直接或间接地从_____继承。

（7）装箱实质是把_____转化为_____。

（8）至少会执行一次循环的循环语句是_____。

（9）较适用于已知循环次数的循环语句是_____。

（10）如果类名为 UserInfo，那么它的构造函数名为_____。

（11）_____可以重用现有类的数据和行为，并扩展新的功能。

2. 是非题

（1）decimal 类型必须在数据末尾添加 M 或 m，否则编译器以 double 类型处理。（　　）

（2）访问结构类型中成员的方式通常使用“结构名.成员名”形式。（　　）

（3）枚举类型的变量可能同时取到枚举中两个元素的值。（　　）

（4）数组可以由一组数据类型不相同的元素组成。（　　）

（5）foreach 语句适用于枚举数组中的元素。（　　）

（6）当一个类实例化时，它的构造函数中包含的代码肯定会执行。（　　）

3. 选择题

（1）下列数据类型属于值类型的是（　　）。

A．struct　　B．class　　C．interface　　D．delegate

（2）下列数据类型属于引用类型的是（　　）。

A．bool　　B．char　　C．string　　D．enum

（3）下列运算符中（　　）具有三个操作数。

A．>>=　　B．&&　　C．++　　D．?

（4）下面有关数据类型的描述中不正确的是（　　）。

A．两个引用类型变量可能引用同一个对象　　B．bool 类型中可以用数字 1 表示 true

C．byte 类型的取值范围是 0~255　　D．可以通过转义符方式输入字符

（5）下面对 protected 修饰符说法正确的是（　　）。

A．只能在派生类中访问　　B．只能在所属的类中访问

C．能在当前应用程序中访问　　D．能在所属的类或派生类中访问

（6）以下有关属性的说法错误的是（　　）。

A．通过属性能获取类中 private 字段的数据

B．当定义属性时，若仅包含 set 访问器，则表示该属性为只读属性

C．属性的访问形式是“对象名.属性名”

D．属性体现了对象的封装性

4. 简答题

（1）请说明修饰符 public、internal、protected、protected internal、private 的区别。

（2）值类型和引用类型有什么区别。

（3）举例说明装箱和拆箱的作用。

5. 上机操作题

（1）建立并调试本章的所有实例。

（2）调试实例 3-4 源程序，要求在“Response.Write("<br />");”语句处设置断点，查看循环变量 *i*、*k* 和 *j* 的值，通过更改 *i* 变量人为地控制循环次数。

（3）设计一个 ASP.NET 4.5 页面，其中包含 TextBox 和 Button 控件各一个。当在 TextBox 中输入一个成绩，再单击 Button 控件时在页面上输出相应的等级信息。

（4）在 ASP.NET 4.5 页面上输出九九乘法表。

（5）在 ASP.NET 4.5 页面上输出如下形状：

```
  A
 BBB
CCCCC
 DDD
  E
```

（6）设计一个 ASP.NET 4.5 页面，其中包含 TextBox 和 Button 控件各一个。当在 TextBox 中输入一组以空格间隔的一组数字后，再单击 Button 控件时在页面上输出该组数字的降序排列（要求使用数组）。

（7）设计一个 ASP.NET 4.5 页面，其中包含两个 TextBox 和一个 Button 控件。当在 TextBox 中各输入一个数值，再单击 Button 控件时在页面上输出两者相除的数值（要求包含异常处理）。

（8）设计一个用于用户注册页面的用户信息类 UserInfo，它包括两个属性：姓名（Name）、生日（Birthday）；一个方法 DecideAge()：用于判断用户是否达到规定年龄，对大于等于 18 岁的在页面上输出“您是成人了！”，而小于 18 岁的在页面上输出“您还没长大呢？”。

（9）改写第（8）题中 DecideAge()方法，增加一个事件 ValidateBirthday：当输入的生日值大于当前日期或小于 1900-1-1 时被触发。

（10）设计 ASP.NET 4.5 页面并应用自己定义的 UserInfo 类。