C# 枚举（Enum） C# 继承

C# 类（Class）

当你定义一个类时，你定义了一个数据类型的蓝图。这实际上并没有定义任何的数据，但它定义了类的名称意味着什么，也就是说，类的对象由什么组成及在这个对象上可执行什么操作。对象是类的实例。构成类的方法和变量成为类的成员。

类的定义

类的定义是以关键字 **class** 开始，后跟类的名称。类的主体，包含在一对花括号内。下面是类定义的一般形式：

```
<access specifier> class  class_name
{
    // member variables
    <access specifier> <data type> variable1;
    <access specifier> <data type> variable2;
    ...
    <access specifier> <data type> variableN;
    // member methods
    <access specifier> <return type> method1(parameter_list)
    {
        // method body
    }
    <access specifier> <return type> method2(parameter_list)
    {
        // method body
    }
    ...
    <access specifier> <return type> methodN(parameter_list)
    {
        // method body
    }
}
```

请注意：

- 访问标识符 <access specifier> 指定了对类及其成员的访问规则。如果没有指定，则使用默认的访问标识符。类的默认访问标识符是 **internal**，成员的默认访问标识符是 **private**。
- 数据类型 <data type> 指定了变量的类型，返回类型 <return type> 指定了返回的方法返回的数据类型。
- 如果要访问类的成员，你要使用点（.）运算符。
- 点运算符链接了对象的名称和成员的名称。

下面的实例说明了目前为止所讨论的概念：

实例

```
using System;
namespace BoxApplication
```

```
{
    class Box
    {
       public double length;   // 长度
       public double breadth;  // 宽度
       public double height;   // 高度
    }
    class Boxtester
    {
        static void Main(string[] args)
        {
            Box Box1 = new Box();        // 声明 Box1，类型为 Box
            Box Box2 = new Box();        // 声明 Box2，类型为 Box
            double volume = 0.0;         // 体积

            // Box1 详述
            Box1.height = 5.0;
            Box1.length = 6.0;
            Box1.breadth = 7.0;

            // Box2 详述
            Box2.height = 10.0;
            Box2.length = 12.0;
            Box2.breadth = 13.0;

            // Box1 的体积
            volume = Box1.height * Box1.length * Box1.breadth;
            Console.WriteLine("Box1 的体积：  {0}",  volume);

            // Box2 的体积
            volume = Box2.height * Box2.length * Box2.breadth;
            Console.WriteLine("Box2 的体积：  {0}", volume);
            Console.ReadKey();
        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Box1 的体积：  210
Box2 的体积：  1560
```

成员函数和封装

类的成员函数是一个在类定义中有它的定义或原型的函数，就像其他变量一样。作为类的一个成员，它能在类的任何对象上操作，且能访问该对象的类的所有成员。

成员变量是对象的属性（从设计角度），且它们保持私有来实现封装。这些变量只能使用公共成员函数来访问。

让我们使用上面的概念来设置和获取一个类中不同的类成员的值：

实例

```
using System;
namespace BoxApplication
{
    class Box
    {
       private double length;   // 长度
       private double breadth;  // 宽度
       private double height;   // 高度
       public void setLength( double len )
       {
            length = len;
       }

       public void setBreadth( double bre )
       {
            breadth = bre;
       }

       public void setHeight( double hei )
       {
            height = hei;
       }
       public double getVolume()
       {
           return length * breadth * height;
       }
    }
    class Boxtester
    {
        static void Main(string[] args)
        {
            Box Box1 = new Box();        // 声明 Box1，类型为 Box
            Box Box2 = new Box();                // 声明 Box2，类型为 Box
            double volume;                              // 体积


            // Box1 详述
            Box1.setLength(6.0);
            Box1.setBreadth(7.0);
            Box1.setHeight(5.0);

            // Box2 详述
            Box2.setLength(12.0);
            Box2.setBreadth(13.0);
            Box2.setHeight(10.0);

            // Box1 的体积
            volume = Box1.getVolume();
            Console.WriteLine("Box1 的体积:  {0}" ,volume);

            // Box2 的体积
            volume = Box2.getVolume();
            Console.WriteLine("Box2 的体积:  {0}", volume);
```

```
            Console.ReadKey();
        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Box1 的体积:  210
Box2 的体积:  1560
```

C# 中的构造函数

类的 **构造函数** 是类的一个特殊的成员函数，当创建类的新对象时执行。

构造函数的名称与类的名称完全相同，它没有任何返回类型。

下面的实例说明了构造函数的概念：

实例

```
using System;
namespace LineApplication
{
   class Line
   {
      private double length;   // 线条的长度
      public Line()
      {
         Console.WriteLine("对象已创建");
      }

      public void setLength( double len )
      {
         length = len;
      }
      public double getLength()
      {
         return length;
      }

      static void Main(string[] args)
      {
         Line line = new Line();
         // 设置线条长度
         line.setLength(6.0);
         Console.WriteLine("线条的长度:  {0}", line.getLength());
         Console.ReadKey();
      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
对象已创建
线条的长度： 6
```

默认的构造函数没有任何参数。但是如果你需要一个带有参数的构造函数可以有参数，这种构造函数叫做**参数化构造函数**。这种技术可以帮助你在创建对象的同时给对象赋初始值，具体请看下面实例：

实例

```
using System;
namespace LineApplication
{
   class Line
   {
      private double length;   // 线条的长度
      public Line(double len)  // 参数化构造函数
      {
         Console.WriteLine("对象已创建，length = {0}", len);
         length = len;
      }

      public void setLength( double len )
      {
         length = len;
      }
      public double getLength()
      {
         return length;
      }

      static void Main(string[] args)
      {
         Line line = new Line(10.0);
         Console.WriteLine("线条的长度： {0}", line.getLength());
         // 设置线条长度
         line.setLength(6.0);
         Console.WriteLine("线条的长度： {0}", line.getLength());
         Console.ReadKey();
      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
对象已创建，length = 10
线条的长度： 10
线条的长度： 6
```

C# 中的析构函数

类的 **析构函数** 是类的一个特殊的成员函数，当类的对象超出范围时执行。

析构函数的名称是在类的名称前加上一个波浪形（~）作为前缀，它不返回值，也不带任何参数。

析构函数用于在结束程序（比如关闭文件、释放内存等）之前释放资源。析构函数不能继承或重载。

下面的实例说明了析构函数的概念：

实例

```
using System;
namespace LineApplication
{
   class Line
   {
      private double length;   // 线条的长度
      public Line()  // 构造函数
      {
         Console.WriteLine("对象已创建");
      }
      ~Line() //析构函数
      {
         Console.WriteLine("对象已删除");
      }

      public void setLength( double len )
      {
         length = len;
      }
      public double getLength()
      {
         return length;
      }

      static void Main(string[] args)
      {
         Line line = new Line();
         // 设置线条长度
         line.setLength(6.0);
         Console.WriteLine("线条的长度:  {0}", line.getLength());
      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
对象已创建
线条的长度:  6
对象已删除
```

C# 类的静态成员

我们可以使用 **static** 关键字把类成员定义为静态的。当我们声明一个类成员为静态时，意味着无论有多少个类的对象被创建，只会有一个该静态成员的副本。

关键字 **static** 意味着类中只有一个该成员的实例。静态变量用于定义常量，因为它们的值可以通过直接调用类而不需要创建类的实例来获取。静态变量可在成员函数或类的定义外部进行初始化。你也可以在类的定义内部初始化静态变量。

下面的实例演示了**静态变量**的用法：

实例

```
using System;
namespace StaticVarApplication
{
    class StaticVar
    {
       public static int num;
        public void count()
        {
            num++;
        }
        public int getNum()
        {
            return num;
        }
    }
    class StaticTester
    {
        static void Main(string[] args)
        {
            StaticVar s1 = new StaticVar();
            StaticVar s2 = new StaticVar();
            s1.count();
            s1.count();
            s1.count();
            s2.count();
            s2.count();
            s2.count();
            Console.WriteLine("s1 的变量 num:  {0}", s1.getNum());
            Console.WriteLine("s2 的变量 num:  {0}", s2.getNum());
            Console.ReadKey();
        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
s1 的变量 num:  6
s2 的变量 num:  6
```

你也可以把一个**成员函数**声明为 **static**。这样的函数只能访问静态变量。静态函数在对象被创建之前就已经存在。下面的实例演示了**静态函数**的用法：

实例

```
using System;
namespace StaticVarApplication
{
    class StaticVar
    {
       public static int num;
        public void count()
        {
            num++;
        }
        public static int getNum()
        {
            return num;
        }
    }
    class StaticTester
    {
        static void Main(string[] args)
        {
            StaticVar s = new StaticVar();
            s.count();
            s.count();
            s.count();
            Console.WriteLine("变量 num:  {0}", StaticVar.getNum());
            Console.ReadKey();
        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
变量 num:  3
```

C# 枚举（Enum）

C# 继承