C# 反射（Reflection） C# 索引器（Indexer）

C# 属性（Property）

属性（Property）是类（class）、结构（structure）和接口（interface）的命名（named）成员。类或结构中的成员变量或方法称为 **域（Field）**。属性（Property）是域（Field）的扩展，且可使用相同的语法来访问。它们使用 **访问器（accessors）** 让私有域的值可被读写或操作。

属性（Property）不会确定存储位置。相反，它们具有可读写或计算它们值的 **访问器（accessors）**。

例如，有一个名为 Student 的类，带有 age、name 和 code 的私有域。我们不能在类的范围以外直接访问这些域，但是我们可以拥有访问这些私有域的属性。

访问器（Accessors）

属性（Property）的**访问器（accessor）**包含有助于获取（读取或计算）或设置（写入）属性的可执行语句。访问器（accessor）声明可包含一个 get 访问器、一个 set 访问器，或者同时包含二者。例如：

```
// 声明类型为 string 的 Code 属性
public string Code
{
   get
   {
      return code;
   }
   set
   {
      code = value;
   }
}

// 声明类型为 string 的 Name 属性
public string Name
{
   get
   {
     return name;
   }
   set
   {
     name = value;
   }
}

// 声明类型为 int 的 Age 属性
public int Age
{
   get
   {
      return age;
   }
```

```
   set
   {
      age = value;
   }
}
```

实例

下面的实例演示了属性（Property）的用法：

实例

```
using System;
namespace tutorialspoint
{
   class Student
   {

      private string code = "N.A";
      private string name = "not known";
      private int age = 0;

      // 声明类型为 string 的 Code 属性
      public string Code
      {
         get
         {
            return code;
         }
         set
         {
            code = value;
         }
      }

      // 声明类型为 string 的 Name 属性
      public string Name
      {
         get
         {
            return name;
         }
         set
         {
            name = value;
         }
      }

      // 声明类型为 int 的 Age 属性
      public int Age
      {
         get
         {
            return age;
```

```
         }
         set
         {
            age = value;
         }
      }
      public override string ToString()
      {
         return "Code = " + Code +", Name = " + Name + ", Age = " + Age;
      }
   }
   class ExampleDemo
   {
      public static void Main()
      {
         // 创建一个新的 Student 对象
         Student s = new Student();

         // 设置 student 的 code、name 和 age
         s.Code = "001";
         s.Name = "Zara";
         s.Age = 9;
         Console.WriteLine("Student Info: {0}", s);
         // 增加年龄
         s.Age += 1;
         Console.WriteLine("Student Info: {0}", s);
         Console.ReadKey();
      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Student Info: Code = 001, Name = Zara, Age = 9
Student Info: Code = 001, Name = Zara, Age = 10
```

抽象属性（Abstract Properties）

抽象类可拥有抽象属性，这些属性应在派生类中被实现。下面的程序说明了这点：

实例

```
using System;
namespace tutorialspoint
{
   public abstract class Person
   {
      public abstract string Name
      {
         get;
         set;
      }
```

```
        public abstract int Age
        {
           get;
           set;
        }
    }
    class Student : Person
    {

        private string code = "N.A";
        private string name = "N.A";
        private int age = 0;

        // 声明类型为 string 的 Code 属性
        public string Code
        {
           get
           {
              return code;
           }
           set
           {
              code = value;
           }
        }

        // 声明类型为 string 的 Name 属性
        public override string Name
        {
           get
           {
              return name;
           }
           set
           {
              name = value;
           }
        }

        // 声明类型为 int 的 Age 属性
        public override int Age
        {
           get
           {
              return age;
           }
           set
           {
              age = value;
           }
        }
        public override string ToString()
        {
           return "Code = " + Code +", Name = " + Name + ", Age = " + Age;
```

```
        }
    }
    class ExampleDemo
    {
        public static void Main()
        {
            // 创建一个新的 Student 对象
            Student s = new Student();

            // 设置 student 的 code、name 和 age
            s.Code = "001";
            s.Name = "Zara";
            s.Age = 9;
            Console.WriteLine("Student Info:- {0}", s);
            // 增加年龄
            s.Age += 1;
            Console.WriteLine("Student Info:- {0}", s);
            Console.ReadKey();
        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Student Info: Code = 001, Name = Zara, Age = 9
Student Info: Code = 001, Name = Zara, Age = 10
```

C# 反射（Reflection） C# 索引器（Indexer）

1 篇笔记

写笔记

抽象属性例子代码的简化版（使用C# 6.0 语言新特性）

```
using System;
namespace Demo.cs
{
    class Program
    {
        public abstract class Person
        {
            public abstract string Name { get; set; }
            public abstract int Age { get; set; }
        }
        public class Student : Person
        {
            public string Code { get; set; } = "N.A";
            public override string Name { get; set; } = "N.A";
```

```
            public override int Age { get; set; } = 0;
            public override string ToString()
            {
                return $"Code:{Code},Name:{Name},Age:{Age}";
            }
        }

        static void Main(string[] args)
        {
            var s = new Student()
            {
                Code = "001",
                Name = "Zara",
                Age = 10
            };
            System.Console.WriteLine($"Student Info:={s}");

            s.Age++;
            System.Console.WriteLine($"Student Info:={s}");
        }
    }
}
```

qqzzft 9个月前 (06-22)