你必须知道的.NET之特性和属性

2008-10-13 来源：网络

1. 引言

attribute是.NET框架引入的有一技术亮点，因此我们有必要花点时间走进一个发现attribute登堂入室的入口。因为.NET Framework中使用了大量的定制特性来完成代码约定，[Serializable]、[Flags]、[DllImport]、[AttributeUsage]这些的构造，相信我们都见过吧，那么你是否了解其背后的技术。

提起特性，由于高级语言发展的历史原因，不免让人想起另一个耳熟能详的名字：属性。特性和属性，往往给初学者或者从C++转移到C#的人混淆的概念冲击。那么，什么是属性，什么是特性，二者的概念和区别，用法与示例，将在本文做以概括性的总结和比较，希望给你的理解带来收获。另外本文的主题以特性的介绍为主，属性的论述重点突出在二者的比较上，关于属性的更多论述将在另一篇主题中详细讨论，敬请关注。

2. 概念引入

2.1. 什么是特性?

MADN的定义为：公共语言运行时允许添加类似关键字的描述声明，叫做attributes, 它对程序中的元素进行标注，如类型、字段、方法和属性等。Attributes和Microsoft .NET Framework文件的元数据保存在一起，可以用来向运行时描述你的代码，或者在程序运行的时候影响应用程序的行为。

我们简单的总结为：定制特性attribute，本质上是一个类，其为目标元素提供关联附加信息，并在运行期以反射的方式来获取附加信息。具体的特性实现方法，在接下来的讨论中继续深入。

2.2. 什么是属性?

属性是面向对象编程的基本概念，提供了对私有字段的访问封装，在C#中以get和set访问器方法实现对可读可写属性的操作，提供了安全和灵活的数据访问封装。关于属性的概念，不是本文的重点，而且相信大部分的技术人员应该对属性有清晰的概念。以下是简单的属性示例：

```
public class MyProperty
{
    //定义字段
    private string _name;
    private int _age;

    //定义属性，实现对_name字段的封装
    public string Name
```

```
        {
            get { return (_name == null) ? string.Empty : _name; }
            set { _name = value; }
        }

        //定义属性，实现对_age字段的封装
        //加入对字段的范围控制
        public int Age
        {
            get { return _age; }
            set
            {
                if ((value > 0) && (value < 150))
                {
                    _age = value;
                }
                else
                {
                    throw new Exception("Not a real age");
                }
            }
        }
    }

    public class MyTest
    {
        public static void Main(string[] args)
        {
            MyProperty myProperty = new MyProperty();
            //触发set访问器
            myProperty.Name = "Anytao";
            //触发get访问器
            Console.WriteLine(myProperty.Name);
            myProperty.Age = 66;
            Console.WriteLine(myProperty.Age.ToString());

            Console.ReadLine();
        }
    }
```

2.3. 区别与比较

通过对概念的澄清和历史的回溯，我们知道特性和属性只是在名称上有过纠葛，在MSDN上关于attribute的中文解释甚至还是属性，但是我同意更通常的称呼：特性。在功能上和应用上，二者其实没有太多模糊的概念交叉，因此也没有必要来比较其应用的异同点。本文则以特性的概念为重点，来讨论其应用的场合和规则。

我理解的定制特性，就是为目标元素，可以是数据集、模块、类、属性、方法、甚至函数参数等加入附加信息，类似于注释，但是可以在运行期以反射的方式获得。定制特性主要应用在序列化、编译器指令、设计模式等方面。

3. 通用规则

定制特性可以应用的目标元素可以为：程序集(assembly)、模块(module)、类型(type)、属性(property)、事件(event)、字段(field)、方法(method)、参数(param)、返回值(return)，应该全了。

定制特性以[,]形式展现，放在紧挨着的元素上，多个特性可以应用于同一元素，特性间以逗号隔开，以下表达规则有效：[AttributeUsage][Flags]、[AttributeUsage, Flags]、[Flags, AttibuteUsageAttribute]、[AttributeUsage(), FlagesAttribute()]

attibute实例，是在编译期进行初始化，而不是运行期。

C#允许以指定的前缀来表示特性所应用的目标元素，建议这样来处理，因为显式处理可以消除可能带来的二义性。例如： using System;

```
using System;
```

```
namespace Anytao.net
{
    [assembly: MyAttribute(1)]          //应用于程序集
    [moduel: MyAttribute(2)]            //应用于模块
    pubic class Attribute_how2do
    {
        //
    }
}
```

定制特性类型，必须直接或者间接的继承自System.Attribute类，而且该类型必须有公有构造函数来创建其实例。

所有自定义的特性名称都应该有个Attribute后缀，这是习惯性约定。

定制特性也可以应用在其他定制特性上，这点也很好理解，因为定制特性本身也是一个类，遵守类的公有规则。例如很多时候我们的自定义定制特性会应用AttributeUsageAttribute特性，来控制如何应用新定义的特性。

```
[AttributeUsageAttribute(AttributeTarget.All),
```

```
[AttributeUsageAttribute(AttributeTarget.All),
AllowMultiple = true,
Inherited = true]
class MyNewAttribute: System.Attribute
{
//
}
```

定制特性不会影响应用元素的任何功能，只是约定了该元素具有的特质。

所有非抽象特性必须具有public访问限制。

特性常用于编译器指令，突破#define, #undefine, #if, #endif的限制，而且更加灵活。

定制特性常用于在运行期获得代码注释信息，以附加信息来优化调试。

定制特性可以应用在某些设计模式中，如工厂模式。

定制特性还常用于位标记，非托管函数标记、方法废弃标记等其他方面。

4. 特性的应用

4.1. 常用特性

常用特性，也就是.NET已经提供的固有特性，事实上在.NET框架中已经提供了丰富的固有特性由我们发挥，以下精选出我认为最常用、最典型的固有特性做以简单讨论，当然这只是我的一家之言，亦不足道。我想了解特性，还是从这里做为起点，从.NET提供的经典开始，或许是一种求知的捷径，希望能给大家以启示。

AttributeUsage

AttributeUsage特性用于控制如何应用自定义特性到目标元素。关于AttributeTargets、AllowMultiple、Inherited、ValidOn，请参阅示例说明和其他文档。我们已经做了相当的介绍和示例说明，我们还是在实践中自己体会更多吧。

Flags

以Flags特性来将枚举数值看作位标记，而非单独的数值，例如：

```
enum Animal
{
    Dog     = 0x0001,
    Cat     = 0x0002,
    Duck    = 0x0004,
  Chicken = 0x0008
}
```

因此，以下实现就相当轻松

因此，以下实现就相当轻松：

```
Animal animals = Animal.Dog | Animal.Cat;
Console.WriteLine(animals.ToString());
```

请猜测结果是什么，答案是:"Dog, Cat"。如果没有Flags特别，这里的结果将是"3"。关于位标记，也将在本系列的后续章回中有所交代，在此只做以探讨止步。

DllImport

DllImport特性，可以让我们调用非托管代码，所以我们可以使用DllImport特性引入对Win32 API函数的调用，对于习惯了非托管代码的程序员来说，这一特性无疑是救命的稻草。

```
using System;
using System.Runtime.InteropServices;

namespace Anytao.net
{
    class MainClass
    {
       [DllImport("User32.dll")]
       public static extern int MessageBox(int hParent, string msg, string caption, int type);

       static int Main()
       {
          return MessageBox(0, "How to use attribute in .NET", "Anytao_net", 0);
      }
    }
}
```

Serializable

Serializable特性表明了应用的元素可以被序列化(serializated)，序列化和反序列化是另一个可以深入讨论的话题，在此我们只是提出概念，深入的研究有待以专门的主题来呈现，限于篇幅，此不赘述。

Conditional

Conditional特性，用于条件编译，在调试时使用。注意：Conditional不可应用于数据成员和属性。

还有其他的重要特性，包括：Description、DefaultValue、Category、ReadOnly、BrowerAble等，有时间可以深入研究。

4.2. 自定义特性

既然attribute，本质上就是一个类，那么我们就可以自定义更特定的attribute来满足个性化要求，只要遵守上述的12条规则，实现一个自定义特性其实是很容易的，典型的实现方法为：

定义特性

```
[AttributeUsage(AttributeTargets.Class |
      AttributeTargets.Method,
      Inherited = true)]
  public class TestAttribute : System.Attribute
  {
      public TestAttribute(string message)
      {
          throw new Exception("error:" + message);
      }
      public void RunTest()
      {
          Console.WriteLine("TestAttribute here.");
      }
  }
```

应用目标元素 [Test("Error Here.")]

```
[Test("Error Here.")]
    public void CannotRun()
    {
        //
    }
```

获取元素附加信息

如果没有什么机制来在运行期来获取Attribute的附加信息，那么attribute就没有什么存在的意义。因此，.NET中以反射机制来实现在运行期获取attribute信息，实现方法如下：

```
 public static void Main(string[] args)
    {
        Tester t = new Tester();
        t.CannotRun();

        Type tp = typeof(Tester);
        TestAttribute myAtt = (TestAttribute)Attribute.GetCustomAttribute((MemberInfo)tp, typeof(TestAttribute));
        myAtt.RunTest();
    }
```

5. 经典示例

5.1 小菜一碟

啥也不说了，看注释吧。

```
  using System;
using System.Reflection;                                    //应用反射技术获得特性信息

namespace Anytao.net
{
    //定制特性也可以应用在其他定制特性上，
    //应用AttributeUsage，来控制如何应用新定义的特性
    [AttributeUsageAttribute(AttributeTargets.All,          //可应用任何元素
        AllowMultiple = true,                               //允许应用多次
        Inherited = false)]                                 //不继承到派生类
    //特性也是一个类，
    //必须继承自System.Attribute类，
    //命名规范为："类名"+Attribute。
    public class MyselfAttribute : System.Attribute
    {
        //定义字段
        private string _name;
        private int _age;
        private string _memo;

        //必须定义其构造函数，如果不定义有编译器提供无参默认构造函数
        public MyselfAttribute()
        {
        }
        public MyselfAttribute(string name, int age)
        {
            _name = name;
            _age = age;
        }
```

```
    //定义属性
    //显然特性和属性不是一回事儿
    public string Name
    {
        get { return _name == null ? string.Empty : _name; }
    }

    public int Age
    {
        get { return _age; }
    }

    public string Memo
    {
        get { return _memo; }
        set { _memo = value; }
    }

    //定义方法
    public void ShowName()
    {
        Console.WriteLine("Hello, {0}", _name == null ? "world." : _name);
    }
}

//应用自定义特性
//可以以Myself或者MyselfAttribute作为特性名
//可以给属性Memo赋值
[Myself("Emma", 25, Memo = "Emma is my good girl.")]
public class Mytest
{
    public void SayHello()
    {
        Console.WriteLine("Hello, my.net world.");
    }
}

public class Myrun
{
    public static void Main(string[] args)
    {
        //如何以反射确定特性信息
        Type tp = typeof(Mytest);
        MemberInfo info = tp;
        MyselfAttribute myAttribute =
            (MyselfAttribute)Attribute.GetCustomAttribute(info, typeof(MyselfAttribute));
        if (myAttribute != null)
        {
            //嘿嘿，在运行时查看注释内容，是不是很爽
            Console.WriteLine("Name: {0}", myAttribute.Name);
            Console.WriteLine("Age: {0}", myAttribute.Age);
            Console.WriteLine("Memo of {0} is {1}", myAttribute.Name, myAttribute.Memo);
            myAttribute.ShowName();
        }

        //多点反射
```

```
            //多点反射
            object obj = Activator.CreateInstance(typeof(Mytest));

            MethodInfo mi = tp.GetMethod("SayHello");
            mi.Invoke(obj, null);
            Console.ReadLine();
        }
    }
}
```

啥也别想了，自己做一下试试。

5.2 他山之石

MSDN认为，特性 (Attribute) 描述如何将数据序列化，指定用于强制安全性的特性，并限制实时 (JIT) 编译器的优化，从而使代码易于调试。属性 (Attribute) 还可以记录文件名或代码作者，或在窗体开发阶段控制控件和成员的可见性。

dudu Boss收藏的系列文章《Attribute在.net编程中的应用》，给你应用方面的启示会很多，值得研究。

亚历山大同志 的系列文章《手把手教你写ORM(六)》中，也有很好的诠释。

idior的文章《Remoting基本原理及其扩展机制》也有收获，因此补充。

6. 结论

Attribute是.NET引入的一大特色技术，但在博客园中讨论的不是很多，所以拿出自己的体会来分享，希望就这一技术要点进行一番登堂入室的引导。更深层次的应用，例如序列化、程序安全性、设计模式多方面都可以挖掘出闪耀的金子，这就是.NET在技术领域带来的百变魅力吧。希望大家畅所欲言，来完善和补充作者在这方面的不全面和认知上的不深入，那将是作者最大的鼓励和动力。