# **[C#特性详解](http://www.cnblogs.com/rohelm/archive/2012/04/19/2456088.html)**

　　　　特性提供功能强大的方法，用以将元数据或声明信息与代码（程序集、类型、方法、属性等）相关联。特性与程序实体关联后，即可在运行时使用名为“反射”的技术查询特性。这篇文章绝大部分是按照MSDN来学习的，只是加了一点点自己的东东，官方介绍的很详细，我们就一起来了解一下它的用法。

特性具有以下属性：

特性可向程序中添加元数据。元数据是有关在程序中定义的类型的信息。所有的 .NET 程序集都包含指定的一组元数据，这些元数据描述在程序集中定义的类型和类型成员。可以添加自定义特性，以指定所需的任何附加信息。

可以将一个或多个特性应用到整个程序集、模块或较小的程序元素（如类和属性）。

特性可以与方法和属性相同的方式接受参数。

* 程序可以使用反射检查自己的元数据或其他程序内的元数据。

这些都是官方的定义，那么对于一个初学者来说，看的懂汉字不难，但是上面的元数据是什么？

我这么通俗的解释下：

　　你注意过程序及编译的时候的pdb文件了吗？pdb文件里面存储了，关于程序集内部的所有的成员信息，例如，成员的数据类型，属性类型，方法返回值，方法入参类型，就是程序及内部所有的定义信息的数据信息，是存储定义信息的一类数据信息，程序集里面的所有的关于声明类的数据信息，包括方法间调用，都是存储在元数据里面。

下面开始一同学习特性的用法：

特性可以放置在几乎所有的声明中。在 C# 中，特性的指定方法为：将括在方括号中的特性名置于其应用到的实体的声明上方。

[System.Serializable]public class SampleClass

{

// Objects of this type can be serialized.

}

一个声明上可放置多个特性：

[IMG_256](http://www.cnblogs.com/rohelm/archive/2012/04/19/javascript:void(0);" \o "复制代码)

using System.Runtime.InteropServices;

...

void MethodA([In][Out] ref double x) { }void MethodB([Out][In] ref double x) { }void MethodC([In, Out] ref double x) { }

[IMG_257](http://www.cnblogs.com/rohelm/archive/2012/04/19/javascript:void(0);" \o "复制代码)

某些特性对于给定实体可以指定多次。例如，ConditionalAttribute 就是一个可多次使用的特性：

[Conditional("DEBUG"), Conditional("TEST1")]void TraceMethod()

{

// ...

}

根据约定，所有特性名称都以单词“Attribute”结束，以便将它们与“.NET Framework”中的其他项区分。但是，在代码中使用特性时，不需要指定 attribute 后缀。例如，[DllImport] 虽等效于[DllImportAttribute]，但 DllImportAttribute 才是该特性在 .NET Framework 中的实际名称

# **特性参数**

许多特性都有参数，而这些参数可以是定位参数、未命名参数或命名参数。任何定位参数都必须按特定顺序指定并且不能省略，而命名参数是可选的且可以按任意顺序指定。首先指定定位参数。例如，这三个特性是等效的：

[DllImport("user32.dll")]

[DllImport("user32.dll", SetLastError=false, ExactSpelling=false)]

[DllImport("user32.dll", ExactSpelling=false, SetLastError=false)]

第一个参数（DLL 名称）是定位参数并且总是第一个出现，其他参数为命名参数。在这种情况下，两个命名参数均默认为 false，因此可将其省略。有关默认参数值的信息，可以参考参考各个特性的文档。

# **特性目标**

特性的目标是应用该特性的实体。例如，特性可以应用于类、特定方法或整个程序集。默认情况下，特性应用于它后面的元素。但是，您也可以显式标识要将特性应用于方法还是它的参数或返回值。

若要显式标识特性目标，请使用下面的语法：

[target : attribute-list]

下表显示了可能的 target 值的列表。

|  |  |
| --- | --- |
| **C#** | **适用对象** |
| assembly | 整个程序集 |
| module | 当前程序集模块（不同于 Visual Basic 模块） |
| field | 在类或结构中的字段 |
| event | Event |
| method | 方法或 get 和 set 属性访问器 |
| param | 方法参数或 set 属性访问器参数 |
| property | Property |
| return | 方法、属性索引器或 get 属性访问器的返回值 |
| type | 结构、类、接口、枚举或委托 |

下面的示例演示如何将特性应用于程序集和模块。

using System;using System.Reflection;

[assembly: AssemblyTitleAttribute("Production assembly 4")]

[module: CLSCompliant(true)]

下面的示例演示如何在 C# 中将特性应用于方法、方法参数和方法返回值。

[IMG_258](http://www.cnblogs.com/rohelm/archive/2012/04/19/javascript:void(0);" \o "复制代码)

// default: applies to method[SomeAttr]int Method1() { return 0; }

// applies to method[method: SomeAttr]int Method2() { return 0; }

// applies to return value

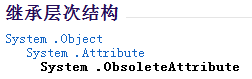
[return: SomeAttr]int Method3() { return 0; }

[IMG_259](http://www.cnblogs.com/rohelm/archive/2012/04/19/javascript:void(0);" \o "复制代码)

无论规定 SomeAttr 应用于什么目标，都必须指定 return 目标，即使 SomeAttr 被定义为仅应用于返回值也是如此。换言之，编译器将不使用 AttributeUsage 信息解析不明确的特性目标。

呀，终于了解完了，是时候自己十一下手了，我们就动手活动一下颈骨，多加点注释来关联一下学过的内容。

我们这里拿ObsoleteAttribute做下测试，它标记不再使用的程序元素。此类不能被继承。首先我们看一下它的继承结构。

当然我们看看其他的特性，我们就会发现特性其实是从System.Object类派生出来的一种特殊类。

 我们现在用这个构造来验证

public ObsoleteAttribute(string message, bool error)

  参数　　　　　　　　　　　　　　　　　　　　　　　　　类型：

  message　　　　　　　　　　　　　　　　　　 System ..::.String　　　　　　　　描述可选的变通方法的文本字符串。

　error　　　　　　　　　　　　　　　　　　　　System ..::.Boolean　　　　　　　指示是否将使用已过时的元素视为错误的布尔值。

总之我们在使用特性的时候不要产生畏惧，就当他是特殊的类，以前怎么样用构造函数现在仍旧怎么用只是格式有点微妙的变化。

[IMG_261](http://www.cnblogs.com/rohelm/archive/2012/04/19/javascript:void(0);" \o "复制代码)

using System;

namespace 特性

{

class Program

{

static void Main(string[] args)

{

OldClass old = new OldClass();//2个报错，因为使用OldClass两次  
 old.OldMethod();//警告。因为第二个参数未指定使用已过时的元素不会视为错误 Console.ReadKey();

}

}

[Obsolete("该类已经过时",true)]//使用默认的特性目标，直接作用于紧随其后的Class OldClass

//第二个参数我这里设置为true将使用已过时的元素视为错误

class OldClass

{

[method: Obsolete("该方法已经过时")]

public void OldMethod()

{

Console.WriteLine("过时的方法！");

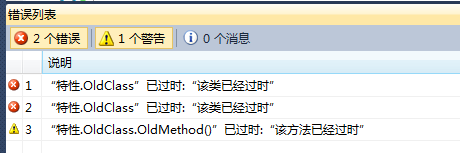
}

}

}

[IMG_262](http://www.cnblogs.com/rohelm/archive/2012/04/19/javascript:void(0);" \o "复制代码)

运行以后会出现两个错误提示，一个警告提示：



# **好了，现在我们在紧接着学习自定义特性，这个估计就算是相当简单了。 自定义特性**

通过定义一个特性类，可以创建您自己的自定义特性。该特性类直接或间接地从Attribute派生，有助于方便快捷地在元数据中标识特性定义。假设您要用编写类型的程序员的名字标记类型。可以定义一个自定义 Author特性类：

[IMG_264](http://www.cnblogs.com/rohelm/archive/2012/04/19/javascript:void(0);" \o "复制代码)

[System.AttributeUsage(System.AttributeTargets.Class |

System.AttributeTargets.Struct)

]public class Author : System.Attribute

{

private string name;

public double version;

public Author(string name)

{

this.name = name;

version = 1.0;

}

}

[IMG_265](http://www.cnblogs.com/rohelm/archive/2012/04/19/javascript:void(0);" \o "复制代码)

类名是特性的名称，即 Author。它由 System.Attribute 派生而来，因此是自定义特性类。构造函数的参数是自定义特性的定位参数。本示例中 name 是定位参数。任何公共的读写字段或属性都是命名参数。在本例中，version 是唯一的命名参数。请注意AttributeUsage 特性的用法，它使得 Author 特性仅在类声明中有效。

可以按如下所示使用此新特性：

[Author("P. Ackerman", version = 1.1)]class SampleClass

{

// P. Ackerman's code goes here...

}

AttributeUsage 有一个命名参数 AllowMultiple，使用它可以使自定义特性成为一次性使用或可以使用多次的特性。在下面的代码示例中，创建了一个使用多次的特性。

[System.AttributeUsage(System.AttributeTargets.Class |

System.AttributeTargets.Struct,

AllowMultiple = true) // multiuse attribute]public class Author : System.Attribute

在下面的代码示例中，向某个类应用了同一类型的多个特性。

[IMG_266](http://www.cnblogs.com/rohelm/archive/2012/04/19/javascript:void(0);" \o "复制代码)

[Author("P. Ackerman", version = 1.1)]

[Author("R. Koch", version = 1.2)]class SampleClass

{

// P. Ackerman's code goes here...

// R. Koch's code goes here...

}

[IMG_267](http://www.cnblogs.com/rohelm/archive/2012/04/19/javascript:void(0);" \o "复制代码)

 如果特性类包含一个属性，则该属性必须为读写属性。

 介绍完了官方的示例是不是还是云里雾里，那么我们一起来深入解剖一下。

首先我们从上面可以总结出创建自定义特性的大概步骤：

1.应用AttributeUsage特性 虽然等效，但AttributeUsageAttribute 才是该特性在 .NET Framework 中的实际名称，它也是由 System.Attribute 派生而来。

2.声明特性类，它由 System.Attribute 派生而来。

3.声明构造函数

4.声明特性

OVER!!!就这么回事，完了吗，我们继续剖析之重要的信息，AttributeUsage特性。

AttributeUsage特性，研究特性当然首要的要研究其构造函数。现在我们来看看他是怎么定义的。

public AttributeUsageAttribute( AttributeTargets validOn)

 参数   validOn 类型：System.AttributeTargets 使用按位“或”运算符组合的一组值，用于指示哪些程序元素是有效的。

用指定的 AttributeTargets、AllowMultiple 值和 Inherited 值列表初始化AttributeUsageAttribute 类的新实例。

于是乎我们返回到了研究AttributeTargets的问题了。现在我们就来细心的看看他是神马！  
原来他是一个枚举，通过该特性可使其成员值按位组合。可以通过按位“或”运算组合AttributeTargets 枚举值来获得首选组合。

成员

|  |  |  |
| --- | --- | --- |
|  | **成员名称** | **说明** |
|  | Assembly | 可以对程序集应用特性。 |
|  | Module | 可以对模块应用特性。  注意  Module 指的是可移植的可执行文件（.dll 或 .exe），而非 Visual Basic 标准模块。 |
|  | Class | 可以对类应用特性。 |
|  | Struct | 可以对结构应用特性，即值类型。 |
|  | Enum | 可以对枚举应用特性。 |
|  | Constructor | 可以对构造函数应用特性。 |
|  | Method | 可以对方法应用特性。 |
|  | Property | 可以对属性应用特性。 |
|  | Field | 可以对字段应用特性。 |
|  | Event | 可以对事件应用特性。 |
|  | Interface | 可以对接口应用特性。 |
|  | Parameter | 可以对参数应用特性。 |
|  | Delegate | 可以对委托应用特性。 |
|  | ReturnValue | 可以对返回值应用特性。 |
|  | GenericParameter | 可以对泛型参数应用特性。  注意  目前，此特性可以应用仅于 C#、Microsoft 中间语言 (MSIL) 和发出的代码。 |
|  | All | 可以对任何应用程序元素应用特性。 |

 到了这里一节也就明了了，谜底都一一展现在我们的面前。

 按照上面的经验，再次开始动手来熟悉这一切，我指定了该自定义的特性不可继承，就在不解释别的了只是为了证明一下命名参数Inherited定性成功与否，总之还是很简单的。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64 | using System;    namespace 特性  {      class Program      {          static void Main(string[] args)          {              GetAttributeInfo(typeof(OldClass));              Console.WriteLine("==============");              GetAttributeInfo(typeof(NewClass));              Console.ReadKey();          }          public static void GetAttributeInfo(Type t)          {              OldAttribute myattribute = (OldAttribute)Attribute.GetCustomAttribute(t, typeof(OldAttribute));              if (myattribute == null)              {                  Console.WriteLine(t.ToString()+"类中自定义特性不存在！");              }              else              {                  Console.WriteLine("特性描述:{0}\n加入事件{1}", myattribute.Discretion, myattribute.date);              }          }      }       [AttributeUsage(AttributeTargets.Class,Inherited=false)]//设置了定位参数和命名参数            //该特性适用于所有的类，而且是非继承的。      class OldAttribute : Attribute//继承自Attribute      {          private string discretion;            public string Discretion          {              get { return discretion; }              set { discretion = value; }          }          public DateTime date;          public OldAttribute(string discretion)          {              this.discretion = discretion;              date = DateTime.Now;          }      }      //现在我们定义两类      [Old("这个类将过期")]//使用定义的新特性      class OldClass      {          public void OldTest()          {              Console.WriteLine("测试特性");          }      }      class NewClass:OldClass      {          public void NewTest()          {              Console.WriteLine("测试特性的继承");          }      }      //我们写一个方法用来获取特性信息  } |