**[Attribute在.net编程中的应用（一）](http://www.cnblogs.com/dudu/articles/4449.html)**

Attribute的基本概念

经常有朋友问，Attribute是什么？它有什么用？好像没有这个东东程序也能运行。实际上在.Net中，Attribute是一个非常重要的组成部分，为了帮助大家理解和掌握Attribute，以及它的使用方法，特地收集了几个Attribute使用的例子，提供给大家参考。

在具体的演示之前，我想先大致介绍一下Attribute。我们知道在类的成员中有property成员，二者在中文中都做属性解释，那么它们到底是不是同一个东西呢？从代码上看，明显不同，首先就是它们的在代码中的位置不同，其次就是写法不同（Attribute必须写在一对方括符中）。

**什么是Atrribute**

首先，我们肯定Attribute是一个类，下面是msdn文档对它的描述：  
公共语言运行时允许你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。

在.NET中,Attribute被用来处理多种问题，比如序列化、程序的安全特征、防止即时编译器对程序代码进行优化从而代码容易调试等等。下面，我们先来看几个在.NET中标准的属性的使用，稍后我们再回过头来讨论Attribute这个类本身。（文中的代码使用C#编写，但同样适用所有基于.NET的所有语言）

**Attribute作为编译器的指令**

在C#中存在着一定数量的编译器指令，如：#define DEBUG, #undefine DEBUG, #if等。这些指令专属于C#,而且在数量上是固定的。而Attribute用作编译器指令则不受数量限制。比如下面的三个Attribute:

* Conditional：起条件编译的作用，只有满足条件，才允许编译器对它的代码进行编译。一般在程序调试的时候使用。
* DllImport：用来标记非.NET的函数，表明该方法在一个外部的DLL中定义。
* Obsolete：这个属性用来标记当前的方法已经被废弃，不再使用了。

下面的代码演示了上述三个属性的使用：

#define DEBUG //这里定义条件

using System;

using System.Runtime.InteropServices;

using System.Diagnostics;

namespace AttributeDemo

{

class MainProgramClass

{

[DllImport("User32.dll")]

public static extern int MessageBox(int hParent, string Message, string Caption, int Type);

static void Main(string[] args)

{

DisplayRunningMessage();

DisplayDebugMessage();

MessageBox(0,"Hello","Message",0);

Console.ReadLine();

}

[Conditional("DEBUG")]

private static void DisplayRunningMessage()

{

Console.WriteLine("开始运行Main子程序。当前时间是"+DateTime.Now);

}

[Conditional("DEBUG")]

[Obsolete]

private static void DisplayDebugMessage()

{

Console.WriteLine("开始Main子程序");

}

}

}

如果在一个程序元素前面声明一个Attribute,那么就表示这个Attribute被施加到该元素上，前面的代码，[DllImport]施加到MessageBox函数上, [Conditional]施加到DisplayRuntimeMessage方法和DisplayDebugMessage方法，[Obsolete]施加到DisplayDebugMessage方法上。

根据上面涉及到的三个Attribute的说明，我们可以猜到程序运行的时候产生的输出：DllImport Attribute表明了MessageBox是User32.DLL中的函数，这样我们就可以像内部方法一样调用这个函数。

重要的一点就是Attribute就是一个类，所以DllImport也是一个类，Attribute类是在编译的时候被实例化的，而不是像通常的类那样在运行时候才实例化。Attribute实例化的时候根据该Attribute类的设计可以带参数，也可以不带参数，比如DllImport就带有"User32.dll"的参数。Conditional对满足参数的定义条件的代码进行编译，如果没有定义DEBUG,那么该方法将不被编译，读者可以把#define DEBUG一行注释掉看看输出的结果（release版本，在Debug版本中Conditional的debug总是成立的）。Obsolete表明了DispalyDebugMessage方法已经过时了，它有一个更好的方法来代替它，当我们的程序调用一个声明了Obsolete的方法时，那么编译器会给出信息，Obsolete还有其他两个重载的版本。大家可以参考msdn中关于的ObsoleteAttribute 类的描述。

**Attribute类**

除了.NET提供的那些Attribute派生类之外，我们可以自定义我们自己的Attribute，所有自定义的Attribute必须从Attribute类派生。现在我们来看一下Attribute 类的细节：

protected Attribute(): 保护的构造器，只能被Attribute的派生类调用。

三个静态方法：

static Attribute GetCustomAttribute():这个方法有8种重载的版本，它被用来取出施加在类成员上指定类型的Attribute。

static Attribute[] GetCustomAttributes(): 这个方法有16种重载版本，用来取出施加在类成员上指定类型的Attribute数组。

static bool IsDefined():由八种重载版本，看是否指定类型的定制attribute被施加到类的成员上面。

实例方法：

bool IsDefaultAttribute(): 如果Attribute的值是默认的值，那么返回true。

bool Match():表明这个Attribute实例是否等于一个指定的对象。

公共属性： TypeId: 得到一个唯一的标识，这个标识被用来区分同一个Attribute的不同实例。

我们简单地介绍了Attribute类的方法和属性，还有一些是从object继承来的。这里就不列出来了。

下面介绍如何自定义一个Attribute: 自定义一个Attribute并不需要特别的知识，其实就和编写一个类差不多。自定义的Attribute必须直接或者间接地从Attribute这个类派生，如：

public MyCustomAttribute : Attribute { ... }

这里需要指出的是Attribute的命名规范，也就是你的Attribute的类名+"Attribute",当你的Attribute施加到一个程序的元素上的时候，编译器先查找你的Attribute的定义，如果没有找到，那么它就会查找“Attribute名称"+Attribute的定义。如果都没有找到，那么编译器就报错。

对于一个自定义的Attribute，你可以通过AttributeUsage的Attribute来限定你的Attribute 所施加的元素的类型。代码形式如下： [AttriubteUsage(参数设置)] public 自定义Attribute : Attribute { ... }

非常有意思的是，AttributeUsage本身也是一个Attribute，这是专门施加在Attribute类的Attribute. AttributeUsage自然也是从Attribute派生，它有一个带参数的构造器，这个参数是AttributeTargets的枚举类型。下面是AttributeTargets 的定义：

public enum AttributeTargets

{

All=16383,

Assembly=1,

Module=2,

Class=4,

Struct=8,

Enum=16,

Constructor=32,

Method=64,

Property=128,

Field=256,

Event=512,

Interface=1024,

Parameter=2048,

Delegate=4096,

ReturnValue=8192

}

作为参数的AttributeTarges的值允许通过“或”操作来进行多个值得组合，如果你没有指定参数，那么默认参数就是All 。 AttributeUsage除了继承Attribute 的方法和属性之外，还定义了以下三个属性：

AllowMultiple: 读取或者设置这个属性，表示是否可以对一个程序元素施加多个Attribute 。

Inherited:读取或者设置这个属性，表示是否施加的Attribute 可以被派生类继承或者重载。

ValidOn: 读取或者设置这个属性，指明Attribute 可以被施加的元素的类型。

**AttributeUsage 的使用例子：**

using System;

namespace AttTargsCS

{

// 该Attribute只对类有效.

[AttributeUsage(AttributeTargets.Class)]

public class ClassTargetAttribute : Attribute

{

}

// 该Attribute只对方法有效.

[AttributeUsage(AttributeTargets.Method)]

public class MethodTargetAttribute : Attribute

{

}

// 该Attribute只对构造器有效。

[AttributeUsage(AttributeTargets.Constructor)]

public class ConstructorTargetAttribute : Attribute

{

}

// 该Attribute只对字段有效.

[AttributeUsage(AttributeTargets.Field)]

public class FieldTargetAttribute : Attribute

{

}

// 该Attribute对类或者方法有效（组合）.

[AttributeUsage(AttributeTargets.Class|AttributeTargets.Method)]

public class ClassMethodTargetAttribute : Attribute

{

}

// 该Attribute对所有的元素有效.

[AttributeUsage(AttributeTargets.All)]

public class AllTargetsAttribute : Attribute

{

}

//上面定义的Attribute施加到程序元素上的用法

[ClassTarget] //施加到类

[ClassMethodTarget]//施加到类

[AllTargets] //施加到类

public class TestClassAttribute

{

[ConstructorTarget] //施加到构造器

[AllTargets] //施加到构造器

TestClassAttribute()

{

}

[MethodTarget] //施加到方法

[ClassMethodTarget] //施加到方法

[AllTargets] //施加到方法

public void Method1()

{

}

[FieldTarget] //施加到字段

[AllTargets] //施加到字段

public int myInt;

static void Main(string[] args)

{

}

}

}

至此，我们介绍了有关Attribute类和它们的代码格式。你一定想知道到底如何在你的应用程序中使用Attribute，如果仅仅是前面介绍的内容，还是不足以说明Attribute有什么实用价值的话，那么从后面的章节开始我们将介绍几个Attribute的不同用法，相信你一定会对Attribute有一个新的了解。（待续）

原文: <http://www.csdn.net/Develop/Read_Article.asp?Id=19546>