|  |
| --- |
| **你必须懂的.NET中Attribute**[[复制链接]](javascript:;) |
|  |  |

|  |  |
| --- | --- |
| **zhaoguanghui131**  头像  Andy   * 论坛积分[655 分](http://www.51aspx.com/Help/4" \o "积分有什么用？" \t "_blank) * 注册时间2010-10-13   实名用户 | [楼主](http://bbs.51aspx.com/showtopic-33963.html###)  字体大小:  http://bbs.51aspx.com/images/groupicons/member.gif 发表于 2011-07-28 10:06 |[只看楼主](http://bbs.51aspx.com/showtopic.aspx?topicid=33963&forumpage=1&onlyauthor=1&posterid=177272)  作为一个.NET开发人员，懂Attribute的重要性，用.NET大师Jeffrey Richter的话就是“任何.NET Framework 开发人员都有必要对定制attribute有一个牢靠的掌握”，所以掌握Attitude，这是必须的！   　　什么是Attribute（特性）？和Property（属性）是什么区别？   　　我们来看看MSDN中对特性的描述：   　　Attribute 类将预定义的系统信息或用户定义的自定义信息与目标元素相关联。 目标元素可以是程序集、类、构造函数、委托、枚举、事件、字段、接口、方法、可移植可执行文件模块、参数、属性、返回值、结构或其他特性。特性在您编译代码时被发送到元数据中，并可通过运行时反射服务用于公共语言运行时以及任何自定义工具或应用程序。通俗地理解可以这么表述：你可以通过Attribute将一些额外信息加在一些目标元素上（类，字段，接口等），程序编译的时候就将这些额外的信息系列化程序集的元数据中，当你运行程序的时候可以通过反射技术从程序集元数据中读取这些额外信息，并根据这些额外信息决定你程序的行为。   　　Attribute和Property有什么区别？其实这个问题是针对中文背景的开发者而言的，因为很多中文译本把Attribute和Property都翻译成属性，在这里为了区分，我们把Attribute翻译为特性，Attribute和Property基本没有什么瓜葛，因为它们是.NET中不同层面的东西，Property就是我们再熟悉不过的定义在类中的属性，它属于面向对象理论范畴，而Attribute是编程语言文法层面的东西，其定义在上面一段已经描述。   　　你使用过.NET定义好的Attribute吗？   　　在.NET的基础类库中提供了很多定制好的Attribute供开发人员使用，这些定制的Attribute目的的方便开发者在代码中表达他们的意图。如下面三个Attribute类都是C#编译器能够理解的特性类：  　　Obsolete：这个属性用来标记不再使用的程序实体（如类或方法），每次使用标记为过时的实体时，会设设定此特性的方法，产生警告或错误。  　　Conditional：该特性可以标示出某种环境设置下某个方法是否应该被调用。   　　Serializable：指示一个类可以序列化。   　　下面以Obsolete特性的使用为例，说明Attribute是如何应用它的目标元素的。   namespace AttributeDemo {     class Program     {         static void Main(string[] args)         {             MyClass myclass = new MyClass();             myclass.OldMethod();             Console.ReadKey();         }     }      public class MyClass     {         [Obsolete("这是一个旧的方法，请调用新的方法NewMethod")]         public void OldMethod()         {             Console.WriteLine("这是旧方法");         }          public void NewMethod()         {             Console.WriteLine("这是新方法");         }     } }  　　调试这段程序的时候会发出警告信息，如下图所示：http://images.cnblogs.com/cnblogs_com/zoupeiyang/jgao.JPG  　　像Obsolete这样的定制特性，编译器能够做出相应处理，如在使用标记了Obsolete特性的方法时会发出警告信息，但如果我们自己定制的Attribute时，编译器会做什么处理呢。下面我们自己定义一个Attribute。   　　我也来定义一个Attribute   　　为了符合“公共语言规范”（CLS），定制Attribute必须直接或间接从公共抽象类System.Attribute派生。所以我们前面提到Obsolete、Conditional和Serializable都是派生于Attribute。这里需要说明下的是自定制的Attribute的命名规范，其规则是“特性名+Attribute”，也就是我们自定制必须以Attribute为后缀，那么我们上面提到的三个特性都没有Attribute为后缀的呢，原来定义它们的时候都是有Attribute后缀的，如Obsolete是ObsoleteAttribute，只是我们将一个特性应用于某个目标元素时可以将Attribute这个后缀去掉，因为编译器会先查找没有Attribute后缀的特性，如果没有找到，则会查找加了Attribute后缀的特性名称。  System.Attribute类的构造器被protected修饰，说明它不能自己实例化，只能被它的派生类调用。它有三个重要的静态方法，如下：  方法名称 说明  GetCustomAttributes 有多个重载，返回作用于目标的Attribute类实例的数组，也就是返回的类型是Attribute[]  GetCustomAttribute 有多个重载，返回作用于目标的Attribute类的一个实例，如果目标没有应用任何的Attribute则返回null，如果目标应用了指定的Attribute的多个实例，就抛出一个System.Reflection.AmbiguousMatchException异常。  IsDefined 如果至少有一个指定的Attribute派生类实例作用于目标，就返回true，否则返回false。这个方法效率很高，因为它不构建Attribute的实例，前面的两个方法返回都是Attribute实例，也就是需要从元数据中获取信息来构建实例，耗费性能多      　　通常检查一个目标元素是否被应用了某个Attribute时，就调用System.Attribute.IsDefined方法，因为它的性能比GetCustomAttributes和GetCustomAttribute要高，如果需要返回Attribute的实例，则调用GetCustomAttributes或GetCustomAttribute方法。调用这三个方法都会扫描托管模块的元数据（因为Attribute是在编译的时候保存在托管模块的元数据上的），执行字符串比较来定义指定的Attribute类。这样的操作对时间性能消耗大，如果需要反复调用这些方法，可以缓存这些方法的调用结果，也就是把实例保存在全局变量中，不需要每次都扫描和构造实例。   　　除了System.Attribute类提供的上面的三个静态方法可以检查目标元素应用Attribute的情况外，System.Reflection命名空间定义的一些类也允许你检查一个模块的元数据的内容，这些类包括Assembly，Module，ParameterInfo，MemberInfo，Type，MethodInfo，ConstrucorInfo，FieldInfo，EventInfo，PropertyInfo等，它们都提供了GetCustomAttributes和IsDefined方法。这些类GetCustomAttributes返回的类型是Object[],而System.Attribute类GetCustomAttributes方法返回的类型是Attribute[]。  下面将分别使用System.Attribute和System.Reflection.Type各自提供的GetCustomAttributes方法获取Attribute实例提供示例代码，以让您有一个更加直观的认识。  定义一个Attribute   public class MyMsgAttribute:Attribute     {         public string Msg { get; set; }         public MyMsgAttribute(string msg)         {             Msg = msg;         }     }  　　定义一个类，使自定制的MyMsgAttribute类能够应用在这个类上，如下：   [MyMsgAttribute("我的自定义Attribute")]     public class MyClass     {       }  　　使用System.Reflection.Type提供的GetCustomAttributes方法获取MyMsgAttribute类的实例，代码如下：   var attributes = typeof(MyClass).GetCustomAttributes(typeof(MyMsgAttribute), true);             MyMsgAttribute myAttribute = attributes[0] as MyMsgAttribute;             if (myAttribute != null)             {                 Console.WriteLine(myAttribute.Msg);             } 　　        使用System.Attribute提供的GetCustomAttributes方法获取MyMsgAttribute类的实例，代码如下：    var attributes2 = Attribute.GetCustomAttributes(typeof(MyClass));             MyMsgAttribute myAttribute2 = (MyMsgAttribute)attributes2[0];             Console.WriteLine(myAttribute2.Msg);  　　上面两段代码输出的结果都是Msg是属性值——“我的自定义Attribute”。   　　经过上面两段代码的分析，我们已经知道自定义一个Attribute类，并使这个Attribute应用在一个类上，同时在了解了在运行时如何从元数据构造这个Attribute的实例，我们得到Attribute对象，就可以根据这个对象的信息来执行一些逻辑分支代码，上面只是简单地输出Attribute对象Msg属性值，可见，定制Attribute是非常有用的，因为它能在运行时决定我们执行不同的逻辑分支代码。如我们可以通过IsDefined检查一个类是否应用了SerializableAttribute，从而判断这个类是否可以用于系列化操作。   　　在我们使用Attribute应用于目标元素的时候，我们会发现一个想象，就是有些Attribute可以应用于类，也可以应用于属性，如SerializableAttribute，而有些Attribute只能应用于方法这个目标元素，如ConditionalAttribute，为什么不同的Attribute会有这种应用目标元素的区别呢？我们查看这两个Attribute的定义，发现它们本身应用了一个Attribute类，这个Attribute类就是AttributeUsage。因为Attribute本身就是一个类，所以它是允许应用其它Attribute类的。而AttributeUsage的目的就是限定你的Attribute 所施加的元素的类型，比如限制你的Attribute能够应用于类还是方法或者属性上。   　　AttributeUsage的构造函数有一个参数，这个参数是AttributeTargets的枚举类型。  　　AttributeTargets的枚举成员名称说明如下：  All 可以对任何应用程序元素应用特性。   Assembly 可以对程序集应用特性。   Class 可以对类应用特性。   Constructor 可以对构造函数应用特性。   Delegate 可以对委托应用特性。   Enum 可以对枚举应用特性。   Event 可以对事件应用特性。   Field 可以对字段应用特性。   GenericParameter 可以对泛型参数应用特性。   Interface 可以对接口应用特性。   Method 可以对方法应用特性。   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。   　　如果你的自定制的Attribute没有显示应用AttributeUsage，编译器会自动给你加上一个默认的AttributeUsage，而这个默认的构造函数参数就是AttributeTargets.All，也就是你的这个自定制Attribute能够应用下面元素类型为Assembly | Module | Class | Struct | Enum | Constructor | Method | Property | Field | Event | Interface | Parameter | Delegate,  ClassMembers = Class | Struct | Enum | Constructor | Method | Property | Field | Event | Delegate | Interface 。    　　如果你的自定制Attribute只想作用于类和方法，实例代码如下：   [AttributeUsage(AttributeTargets.Class|AttributeTargets.Method)]     public class MyMsgAttribute:Attribute     {         public string Msg { get; set; }         public MyMsgAttribute(string msg)         {             Msg = msg;         }     }  　　这样定义的MyMsgAttribute只能应用于类和方法，应用于其它类型的目标元素时编译的时候会报错。   　　AttributeUsage类提供了两个公共的属性，AllowMultiple和Inherited。AllowMultiple是使用允许让多个Attribute实例应用在同一个目标元素上，当我们将AttributeUsage应用于自定制Attribute时，可以指定AllowMultiple属性值为True，这样自定制的Attribute就允许将它的多个实例应用于单个目标元素，如果不将AllowMultiple显示设为True，自定制的Attribute只能向一个选定的目标元素应用一次。Inherited属性指定自定制Attribute应用于基类时，是否同时应用于派生类和重写的方法。我们用代码演示AllowMultiple和Inherited的概念：   [AttributeUsage(AttributeTargets.Class|AttributeTargets.Method,AllowMultiple=true,Inherited=true)]     public class MyMsgAttribute:Attribute     {         public string Msg { get; set; }         public MyMsgAttribute(string msg)         {             Msg = msg;         }     }      [MyMsgAttribute("我的自定义Attribute")]     [MyMsgAttribute("也是你的自定义Attribute")]     public class MyClass     {                 public string Name { get; set; }                  public string GetName()         {             return Name;         }     }      public class YourClass : MyClass     { }   static void Main(string[] args)         {            var attributes = typeof(MyClass).GetCustomAttributes(typeof(MyMsgAttribute), true);             foreach (var attribute in attributes)             {                 MyMsgAttribute myAttribute = attribute as MyMsgAttribute;                 if (myAttribute != null)                 {                     Console.WriteLine(myAttribute.Msg);                 }             }               attributes = typeof(YourClass).GetCustomAttributes(typeof(MyMsgAttribute), true);             foreach (var attribute in attributes)             {                 MyMsgAttribute myAttribute = attribute as MyMsgAttribute;                 if (myAttribute != null)                 {                     Console.WriteLine(myAttribute.Msg);                 }             }                         Console.ReadKey();         }  　　输出的结果为：http://images.cnblogs.com/cnblogs_com/zoupeiyang/fff.JPG  　　总结  　　我们根据上面文章的分析对自定制Attribute的定义和使用进行总结：  　　1、自定制的Attribute必须派生于System.Attribute。  　　2、在自定制的Attribute应用AttributeUsageAttribute可以对自定制的Attribute进行应用目标元素、目标元素是否支持应用同一个Attribute多个实例，目标元素应用的Attribute是否能应用于的派生类和派生类的重写方法等进行控制。  　　3、自定制的Attribute是在编译时保存在模块的元数据上的，在运行时从元数据读取信息来构建Attribute实例。  　　4、获取或判断某个目标元素应用Attribute的信息，可以通过System.Attribute提供的三个镜头方法：System.Attribute.IsDefined,  System.Attribute.GetCustomAttributes和System.Attribute.GetCustomAttribute，也可以通过System.Reflection命名空间定义的一些类来检查一个模块的元数据的内容，这些类包括Assembly，Module，ParameterInfo，MemberInfo，Type，MethodInfo，ConstrucorInfo，FieldInfo，EventInfo，PropertyInfo等，它们都提供了GetCustomAttributes和IsDefined方法。 |
| [分享](http://bbs.51aspx.com/misc.aspx?action=emailfriend&tid=33963) [转发](javascript:void(0)) |