**C# 特性**

1.特性功能：

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。特性与程序实体相关联后，即可在运行时用反射技术查询特性。

2.使用特性：

特性可以放置在几乎所有的声明中（但特定的特性可能限制在其上有效的声明类型）。

2.1.普通特性：

**[csharp]** [view plain](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "view plain) [copy](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "copy)

1. [System.Serializable]
2. **public** **class** SampleClass
3. {
4. // Objects of this type can be serialized.
5. }

2.2.具有 [DllImportAttribute](https://msdn.microsoft.com/zh-cn/library/system.runtime.interopservices.dllimportattribute.aspx" \t "http://blog.csdn.net/zhulongxi/article/details/_blank) 特性的方法的声明如下：

**[csharp]** [view plain](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "view plain) [copy](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "copy)

1. **using** System.Runtime.InteropServices;

**[csharp]** [view plain](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "view plain) [copy](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "copy)

1. [DllImport("user32.dll")]
2. **extern** **static** **void** SampleMethod();

2.3.一个声明上可放置多个特性：

**[csharp]** [view plain](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "view plain) [copy](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "copy)

1. **using** System.Runtime.InteropServices;

**[csharp]** [view plain](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "view plain) [copy](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "copy)

1. **static** **void** MethodA([In][Out] **ref** **double** x)
2. {
4. }
5. **static** **void** MethodB([Out][In] **double** x)
6. {
8. }
9. **static** **void** MethodC([Out, In]**double** x)
10. {
12. }

2.4.某些特性对于给定实体可以指定多次，如， ConditionalAttribute就是一个可以多次使用的特性：

**[csharp]** [view plain](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "view plain) [copy](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "copy)

1. [Conditional("DEBUG"), Conditional("TEST1")]
2. **void** TraceMethod()
3. {
4. // ...
5. }

3.特性参数：

定位参数（特性类的构造函数的参数）、属性参数（特性类中的属性），也叫命名参数

定位参数是必填的且必须按顺序指定， 属性参数是可选的且可以按任意顺序指定。定位参数必须在前。

**[csharp]** [view plain](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "view plain) [copy](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "copy)

1. [DllImport("user32.dll")]
2. [DllImport("user32.dll", SetLastError=**false**, ExactSpelling=**false**)]
3. [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，请参考各个特性的文档。

4.特性目标：

特性的目标是应用该特性的实体，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类，方法还是它的参数或返回值。

若要显式标识特性目标，请使用下面的语法来应用特性到目标上：

**[csharp]** [view plain](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "view plain) [copy](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "copy)

1. [target : attribute-list]

特性的目标列表：

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

举例：

将特性应用于程序集和模块：

**[csharp]** [view plain](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "view plain) [copy](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "copy)

1. [assembly:AssemblyTitle("Production assembly4")]

**[csharp]** [view plain](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "view plain) [copy](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "copy)

1. [module:CLSCompliant(**true**)]

将特性应用于方法、方法参数和方法返回值：

**[csharp]** [view plain](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "view plain) [copy](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "copy)

1. [method:MyAttr]
2. **static** **int** Method()
3. {
4. **return** 1;
5. }
6. 或
7. [MyAttr]
8. **static** **int** Method()
9. {
10. **return** 1;
11. }

**[csharp]** [view plain](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "view plain) [copy](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "copy)

1. // applies to return value
2. [**return**: SomeAttr]
3. **int** Method3() { **return** 0; }

**[csharp]** [view plain](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "view plain) [copy](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "copy)

1. [property:SomeAttr]
2. **public** **int** Age{**get**;**set**;};
3. 或
4. [SomeAttr]
5. **public** **int** Age{**get**;**set**;};

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。  
  
5.特性的用途：

5.1.在Web服务中， 使用WebMethod特性来标记方法， 以指示该方法可以通过SOAP协议进行调用，

5.2.描述当与本机代码进行交互操作时如何封送方法参数（MarshalAsAttribute）

5.3.描述类、方法和接口的 COM 属性。

5.4.使用 DllImportAttribute 类调用非托管代码。

5.5.在标题、版本、说明或商标方面描述您的程序集。

5.6.描述要持久性序列化类的哪些成员。

5.7.描述如何映射类成员和 XML 节点以便进行 XML 序列化。

5.8.描述方法的安全要求。

5.9.指定用于强制安全性的特性。

5.11.由实时 (JIT) 编译器控制优化，以便易于调试代码。

5.12.获取有关调用方的信息的方法。

6.C#常用特性：

6.1.全局特性：

用于整个程序集或模块：如：AssemblyVersionAttribute用于向程序集中嵌入版本信息：

**[csharp]** [view plain](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "view plain) [copy](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "copy)

1. [assembly: AssemblyVersion("1.0.0.0")]

6.2.过时特性：

Obsolete 属性指示某个程序实体标记为建议不再使用的一个。 每次使用Obsolete标记过的实体会出现警告或错误提示。

**[csharp]** [view plain](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "view plain) [copy](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "copy)

1. [System.Obsolete("use NewMethod", **true**)]
2. **public** **void** OldMethod() { }

6.3.条件特性：Conditional

利用 Conditional 属性，程序员可以定义条件方法。Conditional 属性通过**[测试](http://lib.csdn.net/base/softwaretest" \o "软件测试知识库" \t "http://blog.csdn.net/zhulongxi/article/details/_blank)**条件编译符号来确定适用的条件。当运行到一个条件方法调用时，是否执行该调用，要根据出现该调用时是否已定义了此符号来确定。如果定义了此符号，则执行该调用；否则省略该调用（包括对调用的参数的计算）：

**[csharp]** [view plain](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "view plain) [copy](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "copy)

1. <strong>#define TRACE\_ON</strong>
2. **using** System;
3. **using** System.Diagnostics;
5. **public** **class** Trace
6. {
7. [Conditional("TRACE\_ON")]
8. **public** **static** **void** Msg(**string** msg)
9. {
10. Console.WriteLine(msg);
11. }
12. }
14. **public** **class** ProgramClass
15. {
16. **static** **void** Main()
17. {
18. Trace.Msg("Now in Main...");
19. Console.WriteLine("Done.");
20. }<pre name="code" **class**="csharp">#**if** DEBUG
21. **void** ConditionalMethod()
22. {
23. }
24. #endif

}

如果没#define TRANCE\_ON这一指令，则在main函数中不执行Msg方法，直接输出Done.Conditional 一般用于在程序Debug版本中使用DEBUG标识符来启用跟踪并记录功能：

**[csharp]** [view plain](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "view plain) [copy](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "copy)

1. [Conditional("DEBUG")]//DEBUG指令是系统指令，不用再用#define DEBUG定义
2. **static** **void** DebugMethod()
3. {
4. }

此时，在Debug版本中会调用DebugMethod，在Release版本中会跳过对他的调用。效果和#if DEBUG...#endif一样

如果一个方法有多个Condigional特性，则只要有一个条件符号被定义了，则该方法就会被调用。（相当于用or连接多个特性）

**[csharp]** [view plain](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "view plain) [copy](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "copy)

1. [Conditional("A"), Conditional("B")]
2. **static** **void** DoIfAorB()
3. {
4. // ...
5. }//A,B中只要有一个被定义了， 就会调用。

Condigional也可用于特性本身：

**[csharp]** [view plain](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "view plain) [copy](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "copy)

1. [Conditional("DEBUG")]
2. **public** **class** DocumentationAttribute : System.Attribute
3. {
4. **string** text;
6. **public** DocumentationAttribute(**string** text)
7. {
8. **this**.text = text;
9. }
10. }
12. **class** SampleClass
13. {
14. // This attribute will only be included if DEBUG is defined.
15. [Documentation("This method displays an integer.")]//只有定义了DEBUG命令时，Dowork才能应用Documentation特性。
16. **static** **void** DoWork(**int** i)
17. {
18. System.Console.WriteLine(i.ToString());
19. }
20. }

6.4.调用方信息特性：

使用调用方信息特性， 可以获取调用方的信息， 并将它传递给方法。可以获取源代码文件路径，行号和调用方的成员名称：

CallerFilePathAttribute：包含调用方源文件在编译时的完整路径。

CallerLineNumberAttribute：在调用方法的源文件中的行号。

CallerMemberNameAttribute：方法名称或调用方的属性名称。

**[csharp]** [view plain](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "view plain) [copy](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "copy)

1. **static** **void** Main(**string**[] args)
2. {
3. TraceMessage("Something happened.");
4. Console.ReadKey();
5. }
6. **public** **static** **void** TraceMessage(**string** message,
7. [System.Runtime.CompilerServices.CallerMemberName] **string** memberName = "",
8. [System.Runtime.CompilerServices.CallerFilePath] **string** sourceFilePath = "",
9. [System.Runtime.CompilerServices.CallerLineNumber] **int** sourceLineNumber = 0)
10. {
11. System.Diagnostics.Trace.WriteLine("message: " + message);
12. System.Diagnostics.Trace.WriteLine("member name: " + memberName);
13. System.Diagnostics.Trace.WriteLine("source file path: " + sourceFilePath);
14. System.Diagnostics.Trace.WriteLine("source line number: " + sourceLineNumber);
15. }
16. 输出：
17. message: Something happened.
18. member name: Main
19. source file path: d:\Myprojects\MyTestProjects\AttributeTest\Program.cs
20. source line number: 17

6.5.DllImportAttribute特性：using System.Runtime.InteropServices

http://blog.csdn**[.NET](http://lib.csdn.net/base/dotnet" \o ".NET知识库" \t "http://blog.csdn.net/zhulongxi/article/details/_blank)**/hbqhdlc/article/details/6843650

6.5.1.使用DllImport特性可以直接调用一些已经存在的功能（如windows中的一些功能， C++中已经编写好的一些方法）,它的功能是提供从非托管DLL导出的函数进行调用所必须的信息。该特性只能应用于方法， 要求

最少要提供包含入口点的dll的名称。

（托管DLL：指完全由.NET 托管代码实现的DLL，完全依赖于.NET平台的CLR运行， 受.NET CLR管控， 支持内存自动回收，对.NET平台是安全的DLL，非托管DLL：指完全或部分不是用.NET代码实现，

不依赖于.NET平台即可运行， 如COM方式的DLL，不支持内存自动回收， 对.NET平台而言，也是非安全的。）

6.5.2.说明：

6.5.2.1.DllImport只能放置在方法声明上。

6.5.2.2.DllImport具有单个定位参数：指定包含被导入方法的dll名称的dllName参数。

6.5.2.3.DllImport具有五个命名参数：

a.CallingConvention参数指示入口点的调用约定，如果未指定CallingConvention，则使用默认值CallingConvention.Winapi.

b.CharSet参数指定用再入口点的字符集， 如果未指定， 则默认CharSet.Auto.

c.EntryPoint参数给出dll中入口点的名称， 如果未指定， 则默认使用方法本身的名称。

d.ExactSpelling参数指示EntryPoint是否必须与指示的入口点的拼写完全匹配，如果未指定， 则默认false.

e.PreserveSig参数指示方法的签名被保留还是被转换，当签名被转换时， 它被转换为一个具有HRESULT返回值和该返回值的一个名为retval的附加输出参数的签名，如果未指定，则true.

f.SetLastError参数指示方法是否保留Win32"上一错误"，默认false.

6.5.2.4.DllImport修饰的方法必须具有extern修饰符。

6.5.3.举例：

使用DllImport特性导入Win32的MessageBox函数：

**[csharp]** [view plain](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "view plain) [copy](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "copy)

1. [DllImport("user32.dll", CharSet = CharSet.Unicode)]
2. **public** **static** **extern** **int** MessageBox(IntPtr hWnd, String text, String caption, **uint** type);
4. MessageBox(<span style="color:Blue;">**new**</span> IntPtr(0), <span style="color:#A31515;">"Hello World!"</span>, <span style="color:#A31515;">"Hello Dialog"</span>, 0);

在调用MessageBox时就会弹出提示框。

**[csharp]** [view plain](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "view plain) [copy](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "copy)

1. [DllImport("kernel32.dll")]
2. ublic **static** **extern** **bool** Beep(**int** frequency, **int** duration);//调用Beep()API来发出声音

6.5.4.DllImport路径问题：

会按照顺序自动去寻找dll:exe所在目录->System32所在目录->环境变量目录。

所以只需要你把引用的DLL 拷贝到这三个目录下 就可以不用写路径了。

7.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。

**[csharp]** [view plain](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "view plain) [copy](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "copy)

1. [System.AttributeUsage(System.AttributeTargets.Class |
2. System.AttributeTargets.Struct,
3. AllowMultiple = **true**)  // multiuse attribute
4. ]
5. **public** **class** Author : System.Attribute
6. {
7. <span style="color:Blue;">**private**</span> <span style="color:Blue;">**string**</span> name;
8. <span style="color:Blue;">**public**</span> <span style="color:Blue;">**double**</span> version;
9. <span style="color:Blue;">**public**</span> Author(<span style="color:Blue;">**string**</span> name)
10. {
11. <span style="color:Blue;">**this**</span>.name = name;
12. version = 1.0;
13. }
14. }

如果特性类包含一个属性，则该属性必须为读写属性。

8.使用反射访问（检索）自定义特性：

    如果没有检索自定义特性的信息和对其进行操作的方法，则定义自定义特性并将其放置在源代码中就没有意义。使用反射，可检索用自定义特性定义的信息。主要方法是 **GetCustomAttributes**，它返回对象数组，这些对象在运行时等效于源代码特性。

**[csharp]** [view plain](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "view plain) [copy](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "copy)

1. **private** **static** **void** PrintAuthorInfo(Type t)
2. {
3. Console.WriteLine("Author information for {0}",t);
4. Attribute[] attr = Attribute.GetCustomAttributes(t);
5. **foreach** (Attribute item **in** attr)
6. {
7. **if** (item **is** Author)
8. {
9. Author a = (Author)item;
10. Console.WriteLine( "{0},Version{1:f}",a.GetName(),a.version);
11. }
12. }
13. }

**[csharp]** [view plain](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "view plain) [copy](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "copy)

1. [System.AttributeUsage(System.AttributeTargets.Class |
2. System.AttributeTargets.Struct,
3. AllowMultiple = **true**)  // Multiuse attribute.
4. ]
5. **public** **class** Author : System.Attribute
6. {
7. **string** name;
8. **public** **double** version;
10. **public** Author(**string** name)
11. {
12. **this**.name = name;
14. // Default value.
15. version = 1.0;
16. }
18. **public** **string** GetName()
19. {
20. **return** name;
21. }
22. }

**[csharp]** [view plain](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "view plain) [copy](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "copy)

1. // Class with the Author attribute.
2. [Author("P. Ackerman")]
3. **public** **class** FirstClass
4. {
5. // ...
6. }

**[csharp]** [view plain](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "view plain) [copy](http://blog.csdn.net/zhulongxi/article/details/51605574" \o "copy)

1. PrintAuthorInfo(**typeof**(FirstClass));