Attribute trong C#

Một **attribute** trong C#, là một thẻ tường thuật, được sử dụng để truyền thông tin tới runtime về các hành vi của các phần tử đa dạng như các lớp, phương thức, cấu trúc, enum, assembly, … trong chương trình của bạn. Bạn có thể thêm thông tin tường thuật tới một chương trình bởi việc sử dụng một **Attribute**. Một thẻ tường thuật được miêu tả bởi các dấu ngoặc móc vuông ([]) được đặt bên trên phần tử mà nó được sử dụng cho.

Các Attribute được sử dụng để thêm metadata, ví dụ như chỉ lệnh biên dịch và thông tin khác như comment, miêu tả, phương thức và các lớp tới một chương trình. .Net Framework cung cấp hai kiểu Attribute: các Attribute được định nghĩa trước và các Custom Attribute.

Xác định một Attribute trong C#

Cú pháp để xác định một Attribute trong C# như sau:

```
[attribute(positional_parameters, name_parameter = value, ...)]
element
```

Tên của Attribute và giá trị của nó được xác định bên trong dấu ngoặc vuông, ở trước phần tử từ đó thuộc tính được áp dụng cho. *positional_parameters* xác định thông tin thiết yếu và*name_parameter* xác định thông tin tùy ý.

Attribute được định nghĩa trước trong C#

.Net Framework cung cấp 3 Attribute được định nghĩa trước:

- AttributeUsage
- Conditional
- Obsolete

AttributeUsage trong C#

Attribute được định nghĩa trước **AttributeUsage** miêu tả cách một lớp custom Attribute có thể được sử dụng. Nó xác định kiểu của các item, mà từ đó Attribute có thể áp dụng cho.

Cú pháp để xác định Attribute này trong C# như sau:

```
[AttributeUsage(
   validon,
```

```
   AllowMultiple=allowmultiple,
   Inherited=inherited
)]
```

Tại đây:

- Tham số *validon* xác định các phần tử ngôn ngữ mà Attribute có thể được đặt. Nó là một sự tổ hợp giá trị của một **AttributeTargets** enumerator. Giá trị mặc định là**AttributeTargets.All**.
- Tham số *allowmultiple* (tùy ý) cung cấp giá trị cho thuộc tính **AllowMultiple** của attribute này, một giá trị Boolean. Nếu điều này là true, Attribute là multiuse. Giá trị mặc định là false (tức là single-use).
- Tham số *inherited* (tùy ý) cung cấp giá trị cho thuộc tính **Inherited** của attribute này, một giá trị Boolean. Nếu nó là true, Attribute được kế thừa bởi các lớp kế thừa. Giá trị mặc định là false (không được kế thừa).

Ví dụ:

```
[AttributeUsage(AttributeTargets.Class |
AttributeTargets.Constructor |
AttributeTargets.Feild |
AttributeTargets.Method |
AttributeTargets.Property,
AllowMultiple = true)]
```

Conditional trong C#

Attribute tiền định nghĩa này đánh dấu một phương thức có điều kiện mà sự thực thi của nó phụ thuộc vào một tiến trình tiền xử lý định danh đã cho.

Nó tạo sự biên dịch có điều kiện của các lời gọi phương thức, phụ thuộc vào giá trị đã cho, như **Debug** hoặc **Trace**. Ví dụ: nó hiển thị các giá trị của các biến trong khi debug một code.

Cú pháp để xác định Attribute này trong C# là như sau:

```
[Conditional(
   conditionalSymbol
```

```
)]
```

Ví dụ:

```
[Conditional("DEBUG")]
```

Sau đây là ví dụ minh họa Conditional trong C#:

```
#define DEBUG
using System;
using System.Diagnostics;

public class Myclass
{
   [Conditional("DEBUG")]
   public static void Message(string msg)
   {
      Console.WriteLine(msg);
   }
}

class Test
{
   static void function1()
   {
      Myclass.Message("In Function 1.");
      function2();
   }
   static void function2()
   {
      Myclass.Message("In Function 2.");
   }

   public static void Main()
   {
```

```
      Myclass.Message("In Main function.");
      function1();
      Console.ReadKey();
   }
}
```

Khi code trên được biên dịch và thực thi, nó sẽ cho kết quả:

```
In Main function
In Function 1
In Function 2
```

Obsolete trong C#

Attribute tiền định nghĩa này trong C# đánh dấu một thực thể chương trình mà không nên được sử dụng. Nó cho bạn khả năng để thông báo cho compiler để loại bỏ một phần tử target cụ thể. Ví dụ, khi một phương thức mới đang được sử dụng trong một lớp và nếu bạn vẫn muốn giữ lại phương thức cũ trong lớp này, bạn có thể đánh dấu nó là obsolete bằng việc hiển thị một thông báo là phương thức mới nên được sử dụng, thay cho phương thức cũ.

Cú pháp để xác định Attribute này trong C# là như sau:

```
[Obsolete(
   message
)]
[Obsolete(
   message,
   iserror
)]
```

Tại đây,

- Tham số *message* là một chuỗi miêu tả lý do tại sao item là obsolete và cái gì được sử dụng thay cho nó.
- Tham số *iserror* là một giá trị Boolean. Nếu giá trị của nó là true, compiler nên đối xử sự sử dụng của item này như là một lỗi. Giá trị mặc định là false (tức là compiler tạo một warning).

Ví dụ sau minh họa obsolete trong C#:

```
using System;


public class MyClass
{
   [Obsolete("Don't use OldMethod, use NewMethod instead", true)]
   static void OldMethod()
   {
      Console.WriteLine("It is the old method");
   }
   static void NewMethod()
   {
      Console.WriteLine("It is the new method");
   }
   public static void Main()
   {
      OldMethod();
   }
}
```

Khi bạn cố biên dịch chương trình trên, compiler sẽ cho một thông báo lỗi:

```
Don't use OldMethod, use NewMethod instead
```

Tạo Custom Attribute trong C#

Còn gọi là Attribute tùy biến hay Attribute do người dùng tự định nghĩa. .Net Framework cho phép tạo các Custom Attribute mà có thể được sử dụng để lưu giữ thông tin tường thuật và có thể được thu nhận tại runtime. Thông tin này có thể liên quan tới bất kỳ phần tử target nào phụ thuộc vào chuẩn thiết kế và yêu cầu ứng dụng.

Tạo và sử dụng Custom Attribute trong C# bao gồm 4 bước sau:

- Khai báo một Custom Attribute
- Xây dựng Custom Attribute

- Áp dụng Attribute trên một phần tử chương trình target
- Truy cập các Attribute thông qua Reflection

Bước cuối cùng liên quan tới việc viết một chương trình đơn giản để đọc qua metadata để tìm ra các notation đa dạng. Metadata là dữ liệu hoặc thông tin được sử dụng để miêu tả dữ liệu khác. Chương trình này nên sử dụng các Reflection để truy cập các Attribute tại runtime. Chúng ta sẽ bàn luận điều này trong chương tới.

Khai báo một Custom Attribute trong C#

Một Custom Attribute mới nên được kế thừa từ lớp **System.Attribute** trong C#. Ví dụ:

```
//a custom attribute BugFix to be assigned to a class and its members
[AttributeUsage(AttributeTargets.Class |
AttributeTargets.Constructor |
AttributeTargets.Field |
AttributeTargets.Method |
AttributeTargets.Property,
AllowMultiple = true)]

public class DeBugInfo : System.Attribute
```

Trong code trên, chúng ta đã khai báo một Custom Attribute là *DeBugInfo*.

Xây dựng Custom Attribute trong C#

Chúng ta cùng xây dựng Custom Attribute có tên là *DeBugInfo*, mà lưu giữ thông tin thu được bởi việc debug bất kỳ chương trình nào. Nó có thể giữ thông tin sau:

- Số hiệu code để bug
- Tên lập trình viên, người nhận diện bug đó
- Ngày review cuối cùng của code đó
- Một thông báo dạng chuỗi để lưu giữ các lưu ý của lập trình viên

Lớp *DeBugInfo* có 3 thuộc tính private để lưu giữ 3 thông tin đầu tiên và một thuộc tính public để lưu giữ thông báo đó. Vì thế, số hiệu bug, tên lập trình viên, và ngày review là các tham số vị trí tương ứng của lớp DeBugInfo và thông báo là một tham số tùy ý.

Mỗi Attribute phải có ít nhất một constructor. Các tham số vị trí tương ứng nên được truyền thông qua constructor đó. Ví dụ sau minh họa lớp *DeBugInfo* trên:

```
//a custom attribute BugFix to be assigned to a class and its members
[AttributeUsage(AttributeTargets.Class |
AttributeTargets.Constructor |
AttributeTargets.Field |
AttributeTargets.Method |
AttributeTargets.Property,
AllowMultiple = true)]

public class DeBugInfo : System.Attribute
{
   private int bugNo;
   private string developer;
   private string lastReview;
   public string message;

   public DeBugInfo(int bg, string dev, string d)
   {
      this.bugNo = bg;
      this.developer = dev;
      this.lastReview = d;
   }

   public int BugNo
   {
      get
      {
         return bugNo;
      }
   }
```

```
   public string Developer
   {
      get
      {
         return developer;
      }
   }

   public string LastReview
   {
      get
      {
         return lastReview;
      }
   }

   public string Message
   {
      get
      {
         return message;
      }
      set
      {
         message = value;
      }
   }
}
```

Áp dụng Custom Attribute trong C#

Custom Attribute trong C# được áp dụng bằng việc đặt nó ngay trước target của nó:

```
[DeBugInfo(45, "Zara Ali", "12/8/2012", Message = "Return type mismatch")]
[DeBugInfo(49, "Nuha Ali", "10/10/2012", Message = "Unused variable")]
```

```
class Rectangle
{
   //member variables
   protected double length;
   protected double width;
   public Rectangle(double l, double w)
   {
      length = l;
      width = w;
   }
   [DeBugInfo(55, "Zara Ali", "19/10/2012", Message = "Return type mismatch")]

   public double GetArea()
   {
      return length * width;
   }
   [DeBugInfo(56, "Zara Ali", "19/10/2012")]

   public void Display()
   {
      Console.WriteLine("Length: {0}", length);
      Console.WriteLine("Width: {0}", width);
      Console.WriteLine("Area: {0}", GetArea());
   }
}
```

Trong chương tới, chúng ta thu hồi thông tin Attribute bởi sử dụng một đối tượng lớp Reflection trong C#.