予約属性

属性の属性

属性は、要素の情報を明記し、プログラムレベルで要素と情報を関連付けられます
この概念は、XML を学習することでより本質的に理解できると思いますが
そもそも .NET は XML との連帯のために存在するプラットフォームであり
.NET のためにある C# 言語が XML の要素のような性質があるのは偶然ではありません

しかし、C# 言語には属性の性質を定める手段が、言語レベルでは存在しません
属性によっては、クラス専用に作成されていて、メンバに指定されると都合が悪いかもしれません
XML には、要素と属性を関連付ける文書型定義がありますが C# にはないのです

そこで、こういった属性の性質を**予約属性**を用いてコンパイラに知らせます
予約属性は、言語自体の情報をコンパイラに知らせるための属性です
このうち **AttributeUsage** 属性は、属性の情報を明記する属性です

```
[AttributeUsage(
   validon,
   AllowMultiple=allowmultiple,
   Inherited=inherited
)]
```

validon には、列挙型 **AttributeTargets** 型の値を指定します
この値は、位置パラメータで必須です
このパラメータによって、属性が有効な対象を指定することができます

AllowMultiple には、多重適応に関する情報を
Inherited には、継承に関する情報を指定する名前付きパラメータです
これらのパラメータに付いては、後で説明します

AttributeTargets のメンバは以下のようなものがあり、これらは組み合わせて使うこともできます

メンバ	解説
All	属性は、全てに適応可能である
Assembly	属性は、アセンブリに適応可能である
Class	属性は、クラスに適応可能である
Constructor	属性は、コンストラクタに適応可能である
Delegate	属性は、デリゲートに適応可能である
Enum	属性は、列挙型に適応可能である
Event	属性は、イベントに適応可能である
Field	属性は、フィールドに適応可能である
Interface	属性は、インターフェイスに適応可能である
Method	属性は、メソッドに適応可能である
Module	属性は、モジュールに適応可能である
Parameter	属性は、パラメータに適応可能である
Property	属性は、プロパティに適応可能である
ReturnValue	属性は、戻り値に適応可能である
Struct	属性は、構造体に適応可能である

例えば、属性クラスが AttributeTargets.Class パラメータを指定した
AttributeUsage 属性を持っているのならば、その属性はクラスにのみ有効です

```
using System;

enum KittyName { RENA , YUKI , MIMI }

[AttributeUsage(AttributeTargets.Class)]
class KittyAttribute : Attribute {
 public readonly KittyName name;
 public KittyAttribute(KittyName name) {
  this.name = name;
 }
}

[Kitty(KittyName.RENA)] class Kitty {
 [Kitty(KittyName.YUKI)] public void KittyMethod() {} //エラー
}

class Test {
 public static void Main() {}
}
```

このプログラムの Kitty 属性は、クラスにのみ指定できる属性です
しかし、Kitty クラスの KittyMethod() メソッドにも用いているため、コンパイルできません

AttributeUsage は、名前付きパラメータで属性の重複定義も指定できます
属性は、基本的に一つの要素にいくつ指定してもかまいません
XML では、一つの要素が複数の属性を持つことはよくあることです

しかし **同じ属性を重複定義**することは望ましいことではなく

コンパイラは、デフォルトで属性の重複定義を見つけるとエラーを出します
すなわち、一つの要素に対して同じ属性を1回以上指定することはできないということです

AttributeUsage の AllowMultiple 名前付きパラメータを指定すれば
その属性が重複定義可能であるかどうかを明示することができます
AllowMultiple は bool 型のパラメータで、デフォルトで false に設定されています
true を指定すれば重複定義が可能であることを表します

その属性が、一つの要素に対して複数指定されたもかまわないのであれば
この名前付きパラメータを true に設定すればよいのです

```
using System;

enum KittyName { RENA , YUKI , MIMI }

[AttributeUsage(AttributeTargets.Class , AllowMultiple=true)]
class KittyAttribute : Attribute {
 public readonly KittyName name;
 public KittyAttribute(KittyName name) {
  this.name = name;
 }
}

[Kitty(KittyName.RENA)]
[Kitty(KittyName.YUKI)]
[Kitty(KittyName.MIMI)]
class Kitty {}

class Test {
 public static void Main() {
  Type t = typeof(Kitty);
  foreach(Object tmp in t.GetCustomAttributes(false)) {
   KittyAttribute attrKitty = tmp as KittyAttribute;
   if (attrKitty != null)
    Console.WriteLine("名前 : " + attrKitty.name);
  }
 }
}
```

このプログラムの Kitty クラスは Kitty 属性を3つも重複定義しています
本来ならば、このプログラムはコンパイルすることはできません
しかし、Kitty クラスは AttributeUsage で重複可能であることを明示しているため
プログラムは問題なくコンパイルすることができます

さらに、オブジェクト思考における属性の問題は**継承するかどうか**です
通常、属性とは対象の要素にのみ関連付けられる情報であるため
属性を持つクラスを継承したクラスは、基底クラスの属性と関連はありません

クラスを継承する時、その属性を継承するかどうかは
AttributeUsage 属性の Inherited 名前付きパラメータで指定します
このパラメータも bool 型で、true を指定すれば継承可能であることを意味します
逆に false を指定すれば継承できません

```
using System;

enum KittyName { RENA , YUKI , MIMI }

[AttributeUsage(AttributeTargets.All , Inherited=false)]
class KittyAttribute : Attribute {
 public readonly KittyName name;
 public KittyAttribute(KittyName name) {
  this.name = name;
 }
}

[Kitty(KittyName.RENA)] class Kitty {}
class KittyEx : Kitty {}

class Test {
 public static void Main() {
  Type t = typeof(KittyEx);
  foreach(Object tmp in t.GetCustomAttributes(true)) {
   KittyAttribute attrKitty = tmp as KittyAttribute;
   if (attrKitty != null)
    Console.WriteLine("名前 : " + attrKitty.name);
  }
 }
}
```

このプログラムの Kitty 属性は Inherited 名前付きパラメータで
継承不可であることを明示しています
Type クラスの GetCustomAttributes() メソッドのパラメータを true にし
Kitty クラスの派生クラスである KittyEx から属性をたどりますが、属性は見つかりません
プログラムを実行しても、何も出力されないでしょう

逆に Inherited に true を指定すれば、プログラムは RENA を出力します
KittyEx クラスが、基底クラス Kitty の Kitty 属性を継承していることを確認できます

Inherited が true の場合は AllowMultiple パラメータが状態が重要になります
Inherited と AllowMultiple の両方が true の場合、属性は純粋に加算されていきますが
AllowMultiple が false であれば、**属性はオーバーライド**されます

```
using System;

enum KittyName { RENA , YUKI , MIMI }

[AttributeUsage(AttributeTargets.All , AllowMultiple=false , Inherited=true)]
class KittyAttribute : Attribute {
 public readonly KittyName name;
 public KittyAttribute(KittyName name) {
  this.name = name;
 }
}

[Kitty(KittyName.RENA)] class Kitty {}
[Kitty(KittyName.YUKI)] class KittyEx : Kitty {}

class Test {
 public static void Main(String[] args) {
  Type t = typeof(KittyEx);
  foreach(Object tmp in t.GetCustomAttributes(true)) {
   KittyAttribute attrKitty = tmp as KittyAttribute;
   if (attrKitty != null)
    Console.WriteLine("名前 : " + attrKitty.name);
  }
 }
}
```

このプログラムの KittyEx は Kitty クラスの Kitty 属性を継承しています
さらに、KittyEx でも Kitty 属性を指定していますが、この場合は重複定義にはなりません
継承した Kitty 属性がオーバーライドされ、プログラムは YUKI を出力します

この他にも、いくつかの予約属性が存在しますが、この場では省略しいます

前のページへ　戻る　次のページへ