عنوان: **ساخت Attribute های دلخواه یا خصوصی سازی شده**
نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۴/۰۳/۰۲ ۲۲:۴۵
آدرس: www.dotnettips.info
گروه‌ها: C#, Validation, Reflection

در قسمت‌های مختلفی از منابع آموزشی این سایت از متادیتاها attributes استفاده شده و در برخی آموزش هایی چون EF و MVC حداقل یک قسمت کامل را به خود اختصاص داده‌اند. متادیتاها کلاس‌هایی هستند که به روشی سریع و کوتاه در بالای یک Type معرفی شده و ویژگی‌هایی را به آن اضافه می‌کنند. به عنوان مثال متادیتای زیر را ببینید. این متادیتا در بالای یک متد در یک کلاس تعریف شده است و این متد را منسوخ شده اعلام می‌کند و به برنامه نویس می‌گوید که در نسخه‌ی جاری کتابخانه، این متد که احتمال میرود در نسخه‌های پیشین کاربرد داشته است، الان کارآیی خوبی برای استفاده نداشته و بهتر است طبق مستندات آن کلاس، از یک متد جایگزین که برای آن فراهم شده است استفاده کند.

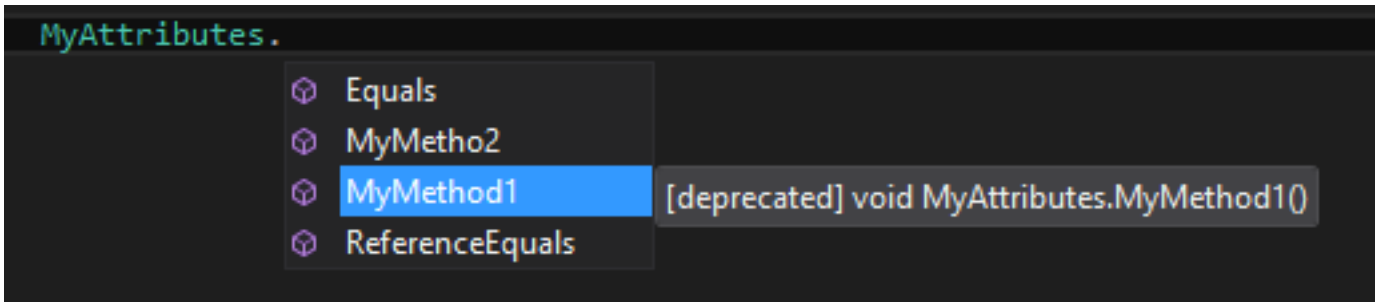
```
public static class  MyAttributes
    {
        [Obsolete]
        public static void MyMethod1()
        {

        }

        public static void MyMetho2()
        {

        }
    }
```

همانطور که ملاحظه می‌کنید می‌توانید اخطار آن را مشاهده کنید:



البته توصیه می‌کنم از ابزارهایی چون Resharper در کارهایتان استفاده کنید، تا طعم کدنویسی را بهتر بچشید. نحوه‌ی نمایش آن در Resharper به مراتب واضح‌تر و گویاتر است:

حال در این بین این سؤال پیش می‌آید که چگونه ما هم می‌توانیم متادیتاهایی را با سلیقه‌ی خود ایجاد کنیم.
برای تهیه‌ی یک متادیتا از کلاس system.attribute استفاده می‌کنیم:

```
public  class  MyMaxLength:Attribute
    {

    }
```

در چنین حالتی شما یک متادیتا ساخته‌اید که می‌توان از آن به شکل زیر استفاده کرد:

```
[MyMaxLength]
    public class GetCustomProperties
    {
//...
     }
```

ولی اگر بخواهید توسط این متادیتا اطلاعاتی را دریافت کنید، می‌توانید به روش زیر عمل کنید. در اینجا من دوست دارم یک متادیتا به اسم MyMaxLength را ایجاد کرده تا جایگزین MaxLength دات نت کنم، تا طبق میل من رفتار کند.

```
public  class  MyMaxLength:Attribute
    {
        private int max;
        public string ErrorText = "";

        public MyMaxLength(int max)
        {
            this.max = max;
            ErrorText = string.Format("max Length is {0} chars", max);
        }
    }
```

در کد بالا، یک متادیتا با یک پارامتر اجباری در سازنده تعریف شده است. این کلاس هم می‌تواند مثل سایر کلاس‌ها سازنده‌های مختلفی داشته باشد تا چندین شکل تعریف متادیتا داشته باشیم. متغیر ErrorText به عنوان یک پارامتر معرفی نشده، ولی از آن جا که public تعریف شده است می‌تواند مورد استفاده‌ی مستقیم قرار بگیرد و استفاده‌ی از آن نیز اختیاری است. نحوه‌ی معرفی این متادیتا نیز به صورت زیر است:

```
[MyMaxLength(30)]
    public class GetCustomProperties
    {
//...
     }

//or
 [MyMaxLength(30,ErrorText = "شما اجازه ندارید بیش از 30 کاراکتر وارد نمایید")]
    public class GetCustomProperties
    {
//...
     }
```

در حالت اول از آنجا که متغیر ErrorText اختیاری است، تعریف نشده‌است. پس در نتیجه با مقدار (x=max) Max length is chars پر خواهد شد ولی در حالت دوم برنامه نویس متن خطا را به خود کلاس واگذار نکرده است و آن را طبق میل خود تغییر داده است.

اجباری کردن Type

هر متادیتا می‌تواند مختص یک نوع Type باشد که این نوع می‌تواند یک کلاس، متد، پراپرتی یا ساختار و ... باشد. نحوه‌ی محدود سازی آن توسط یک متادیتا مشخص می‌شود:

```
[System.AttributeUsage(System.AttributeTargets.Class | System.AttributeTargets.Struct)]
    public  class  MyMaxLength:Attribute
    {
        private int max;
        public string ErrorText = "";

        public MyMaxLength(int max)
        {
            this.max = max;
            ErrorText = string.Format("max Length is {0} chars", max);
        }
    }
```

الان این کلاس توسط متادیتای AttributeUsage که پارامتر ورودی آن Enum است محدود به دو ساختار کلاس و Struct شده است. البته در ویژوال بیسیک با نام Structure معرفی شده است. اگر ساختار شمارشی AttributeTarget را مشاهده کنید، لیستی از نوع‌ها را چون All (همه موارد) ، دلیگیت، سازنده، متد و ... را مشاهده خواهید کرد و از آن جا که این متادیتای ما کاربردش در پراپرتی‌ها خلاصه می‌شود، از متادیتای زیر بر روی آن استفاده می‌کنیم:

```
[AttributeUsage(AttributeTargets.Property)]
```

```
public class User
    {
        [MyMaxLength(30, ErrorText = "شما اجازه ندارید بیش از 30 کاراکتر وارد نمایید")]
        public string Name { get; set; }
}
```

یکی دیگر از ویژگی‌های AttributeUsage خصوصیتی به اسم AllowMultiple است که اجازه می‌دهد بیش از یک بار این متادیتا، بر روی یک نوع استفاده شود:

```
[AttributeUsage(AttributeTargets.Property,AllowMultiple = true)]
    public  class  MyMaxLength:Attribute
    {
        //....
     }
```

که تعریف چندگانه آن به شکل زیر می‌شود:

```
[MyMaxLength(40, ErrorText = "شما اجازه ندارید بیش از 40 کاراکتر وارد نمایید")]
        [MyMaxLength(50, ErrorText = "شما اجازه ندارید بیش از 50 کاراکتر وارد نمایید")]
        [MyMaxLength(30, ErrorText = "شما اجازه ندارید بیش از 30 کاراکتر وارد نمایید")]
        public string Name { get; set; }
```

در این مثال ما فقط اجازه‌ی یکبار استفاده را خواهیم داد؛ پس مقدار این ویژگی را false قرار می‌دهم.

آخرین ویژگی که این متادیتا در دسترس ما قرار میدهد، استفاده از خصوصیت ارث بری است که به طور پیش فرض با True مقداردهی شده است. موقعی که شما یک متادیتا را به ویژگی ارث بری مزین کنید، در صورتی که آن کلاس که برایش متادیتا تعریف می‌کنید به عنوان والد مورد استفاده قرار بگیرد، فرزند آن هم به طور خودکار این متادیتا برایش منظور می‌گردد. به مثال‌های زیر دقت کنید:
دو عدد متادیتا تعریف شده که یکی از آن‌ها ارث بری در آن فعال شده و دیگری خیر.

```
public class MyAttribute : Attribute
{
    //...
}
```

```
[AttributeUsage(AttributeTargets.Method, Inherited = false)]
public class YourAttribute : Attribute
{
    //...
}
```

هر دو متادیتا بر سر یک متد در یک کلاسی که بعدا از آن ارث بری می‌شود تعریف شده اند.

```
public class MyClass
{
    [MyAttribute]
    [YourAttribute]
    public virtual void MyMethod()
    {
        //...
    }
}
```

در کد زیر کلاس بالا به عنوان والد معرفی شده و متد کلاس فرزند الان شامل متادیتایی به اسم MyAttribute است، ولی متادیتای YourAttribute بر روی آن تعریف نشده است.

```
public class YourClass : MyClass
{
    public override void MyMethod()
    {
        //...
    }

}
```

الان که با نحوه‌ی تعریف یکی از متادیتاها آشنا شدیم، این بحث پیش می‌آید که چگونه Type مورد نظر را تحت تاثیر این متادیتا قرار دهیم. الان چگونه میتوانم حداکثر متنی که یک پراپرتی می‌گیرد را کنترل کنم. در اینجا ما از مفهومی به نام Reflection استفاده می‌کنیم. با استفاده از این مفهوم ما میتوانیم به تمامی قسمت‌های یک Type دسترسی داشته باشیم. متاسفانه دسترسی مستقیمی از داخل کلاس متادیتا به نوع مورد نظر نداریم. کد زیر تمامی پراپرتی‌های یک کلاس را چک میکند و سپس ویژگی‌های هر پراپرتی را دنبال کرده و در صورتیکه متادیتای مورد نظر به آن پراپرتی ضمیمه شده باشد، حالا می‌توانید عملیات را انجام دهید. کد زیر میتواند در هر جایی نوشته شود. داخل کلاسی که به آن متادیتا ضمیمه می‌کنید یا داخل تابع Main در اپلیکشین‌ها و هر جای دیگر. مقدار True که به متد GetCustomAttributes پاس می‌شود باعث می‌شود تا متادیتاهای ارث بری شده هم لحاظ گردند.

```
Type type = typeof (User);

                foreach (PropertyInfo property in type.GetProperties())
                {
                    foreach (Attribute attribute in property.GetCustomAttributes(true))
                    {
                        MyMaxLength max = attribute as MyMaxLength;
                        if (max != null)
                        {
                            string Max = max.ErrorText;
                            //انجام عملیات
                        }
                    }
                }
```

البته یک ترفند جهت دسترسی به کلاس‌ها از داخل کلاس متادیتا وجود دارد و آن هم این هست که نوع را از طریق پارامتر به سمت متادیتا ارسال کنید. هر چند این کار زیبایی ندارد ولی به هر حال روش خوبی برای کنترل از داخل کلاس متادیتا و هچنین منظم سازی و دسته بندی و کم کردن کد دارد.

```
[MyMaxLength(30, typeof(User))]
```