C#基础系列：Attribute特性使用 - 文章 - 伯乐在线

前言：总结了下反射得基础用法，这章我们来看看C#的另一个基础技术——特性。

1、什么是特性：就博主的理解，特性就是在类的类名称、属性、方法等上面加一个标记，使这些类、属性、方法等具有某些统一的特征，从而达到某些特殊的需要。比如：方法的异常捕捉，你是否还在某些可能出现异常的地方（例如数据库的操作、文件的操作等）经常使用try…catch。这个时候如果使用特性，就可以大大减少方法里面的try…catch的使用。你只需要定义一个专门捕捉异常的特性类ExceptionExAttribute，然后给这个特性类做些特殊处理，比如给它增加一个AOP拦截的功能（AOP拦截的方式很多，有兴趣可以搜搜看，园子里面很多类似的文章）。那么在可能出现异常的方法名称上面加上一个[ExceptionEx]特性标签，这个方法就具有自动捕捉异常的能力。还是加上官方定义：

特性提供功能强大的方法，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。 reflection.” data-guid=”716c0768f610f38427afe934e71f1d47″ style=”margin: 0px; padding: 0px;”>特性与程序实体关联后，即可在运行时使用名为“反射”的技术查询特性。

特性具有以下属性：

- 特性可向程序中添加元数据。 Metadata is information about the types defined in a program.” data-guid=”01e24c05eb8a815d9d031ae958a894ae” style=”margin: 0px; padding: 0px;”>元数据是有关在程序中定义的类型的信息。 所有的 .NET 程序集都包含指定的一组元数据，这些元数据描述在程序集中定义的类型和类型成员。 可以添加自定义特性，以指定所需的任何附加信息。
- 可以将一个或多个特性应用到整个程序集、模块或较小的程序元素（如类和属性）。
- 特性可以与方法和属性相同的方式接受参数。
- 程序可以使用反射检查自己的元数据或其他程序内的元数据。

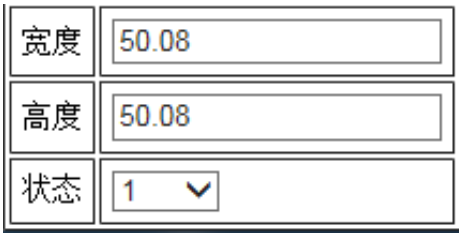
（以上来自MSDN）

2、为什么需要特性：这个上面已经简单介绍过，特性能大大减少统一需求的代码量。其他不说，至少它能让我们的代码看上去更大气点吧~~

3、特性的使用：博主这次还是打算从三个方便分别介绍下特性的常规使用方法。当然这几种方式都是博主原来用过的，可能不是最好的举例场景，但是也算比较典型的特性用法吧。

（1）类的属性上面特性的用法：
之所以将这个放在最前面介绍是因为博主最近做的一个BS项目正好用到，并且使用场景也比较典型。首先介绍下使用场景：最近项目有一个需求，BS界面需要一个拖拽的功能。如下图

当将左边的3个div拖到右边来时，每个div都有自己的特有属性，比如2部门拖过来时，要显示如下属性：



设计思路：每个div对应的Model，每个Model里面有自己特有的属性，然后属性上面加上特性显示属性的名称和默认值，以及界面应该呈现的html标签。

实现代码：

首先来看自定义的一个特性类：

```
C#
public class Factory
        {
                [Detail(AttrName="宽度", Html="<input type='text' />", DefaultValue="50",
DataSource=null)]
                public string Width { set; get; }
                [Detail(AttrName = "高度", Html = "<input type='text' />", DefaultValue =
"50", DataSource = null)]
                public string Height { set; get; }
                [Detail(AttrName = "状态", Html = "<select></select>", DefaultValue = null,
DataSource = "select text,value from status")]
                public string Status { set; get; }
                [Detail(AttrName = "Tag值", Html = "<input type='text' />", DefaultValue =
"", DataSource = null)]
                public string Tag { set; get; }
        }
        public class FactoryDetail
        {
                [Detail(AttrName = "宽度", Html = "<input type='text' />", DefaultValue =
```

```
"50", DataSource = null)]
                public string Width { set; get; }
                [Detail(AttrName = "高度", Html = "<input type='text' />", DefaultValue =
"50", DataSource = null)]
                public string Height { set; get; }
                [Detail(AttrName = "状态", Html = "<select></select>", DefaultValue = null,
DataSource = "select text,value from status")]
                public string Status { set; get; }
                [Detail(AttrName = "Tag值", Html = "<input type='text' />", DefaultValue =
"", DataSource = null)]
                public string Tag { set; get; }
                [Detail(AttrName = "描述", Html = "<input type='text' />", DefaultValue =
"", DataSource = null)]
                public string Desc { set; get; }
        }
```

然后在界面的拖放事件结束时通过js发送ajax请求来得到界面要呈现的html：

```
XHTML
public JsonResult GetModelByType(string strType)
                {
              //strType传过来的是Factory或者FactoryDetail
                        var assembly = Assembly.Load("Ewin.Client.Web");//参数为程序集的名
称
                        var oType = assembly.GetType("Ewin.Client.Web.Controllers." +
strType);
                //得到类的所有属性
                        var lstProperties = oType.GetProperties();
                        foreach (var oProperty in lstProperties)
                        {
                    //得到每一个属性的特性类集合
                                IList<CustomAttributeData> lstAttr =
oProperty.GetCustomAttributesData();
                                foreach (var oAttr in lstAttr)
                                {
                        //得到每一个特性类的全称
                                        Console.WriteLine("特性类的名称" +
oAttr.AttributeType.FullName);
                                        Console.WriteLine("特性类成员如下：");
                        //得到特性类的所有参数
                                        var lstAttrArgu = oAttr.NamedArguments;
                                        foreach (var oAttrAru in lstAttrArgu)
                                        {
```

```
                                    //取每个特性类参数的键值对
                                                Console.WriteLine(oAttrAru.MemberName + "="
+ oAttrAru.TypedValue.Value);
                                            }
                                            //Console.WriteLine(oAttr.AttributeType+"———"+oAtt
r.NamedArguments);
                                    }
                                }
                                return Json(new { }, JsonRequestBehavior.AllowGet);
                            }
```

GetModelByType方法结果简单构造下然后将属性的键值对返回给js方法，然后再由js追加到界面上面。这样通过特性和反射的结合能很快完成这个小功能的设计。

（2）类的方法上面特性的用法：

这个用法.Net framework里面就很多，如果MVC里面Filter过滤器的用法：

C#