# 08特性

## 特性Attribute是什么，和注释有什么区别

特性：中括号声明【】

每一个特性都可以带来对应的功能

例如：[Serializeable] 可以序列化和反序列化 可以影响程序的运行

**特性就是一个类，直接/间接继承Attribute**

实际上特性添加后，编译会在元素内部产生IL,但是我们是没有办法直接使用的，而且在metadata里面会有记录

## 声明和使用特性，attribute,attributeUsage

### 2.1属性的声明和使用

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace \_08属性  {  /// <summary>  /// 特性就是一个类，直接/间接继承Attribute  /// 一般来说 Attribute结尾可以省略掉  /// </summary>  [AttributeUsage(AttributeTargets.All,AllowMultiple =true)] //允许重复修饰  public class CustomAttribute:Attribute  {  public CustomAttribute()  {  }  public CustomAttribute(int id)  {  }  public string Description { get; set; }  public string Remark = null;  public void Show()  {  Console.WriteLine($"this is {nameof(CustomAttribute)}");  }  }  } |
| 使用 在编译器中可以看的到  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace \_08属性  {  [CustomAttribute] //也可以[Custom] <=>等价于[Custom()] 无参构造方式  [Custom()]  [Custom(123)] // 有参构造方式  [Custom(123),Custom(123,Description ="1234")] //多重  [Custom(123,Description ="1234",Remark ="123")] //可以构造函数，指定字段参数等，方法不行  public class People  {  public People()  {  Console.WriteLine("{0}被创建", this.GetType().FullName);  }  public int Id { get; set; }  public string Name { get; set; }  public string Description { get; set; }  public void Study()  {  Console.WriteLine($"this is syudy");  }  [Custom()] //表示给方法加特性  [return:Custom()] //表示给返回值加特性  public string Answer([Custom]string name) //表示给参数加特性  {  return $"This is {name}";  }  }  public class PeopleDTO  {  public PeopleDTO()  {  Console.WriteLine("{0}被创建", this.GetType().FullName);  }  public int Id { get; set; }  public string Name { get; set; }  public string Description { get; set; }  }  } |

## 运行中获取attribute：额外信息 额外操作

**案例：下拉列表，用户状态显示有：正常、冻结、删除，但后台数据库存储的是数字，用传统的if else判断比较麻烦 ，所有结合枚举，属性，扩展类 的方法**

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Reflection;  using System.Text;  using System.Threading.Tasks;  namespace \_08属性  {  /// <summary>  /// 用户状态  /// </summary>  public enum UesrState  {  //正常  [RemarkAttribute("正常")]  Normal=0,  //冻结  [RemarkAttribute("冻结")]  Forzem =1,  //删除  [RemarkAttribute("删除")]  Deleted =2  }  public class RemarkAttribute : Attribute  {  public RemarkAttribute(string remark)  {  this.\_Remark = remark;  }  private string \_Remark = null;  public string GetRemark()  {  return this.\_Remark;  }  }  /// <summary>  /// 扩展类  /// </summary>  public static class RemarkExtension  {  public static string GetRemark(this Enum value)  {  Type type=value.GetType();  FieldInfo field=type.GetField(value.ToString());  if (field.IsDefined(typeof(RemarkAttribute), true))  {  RemarkAttribute attribute = (RemarkAttribute)field.GetCustomAttribute(typeof(RemarkAttribute), true);  return attribute.GetRemark();  }  else  {  return value.ToString();  }    }  }  } |
| {  UesrState uesrState = UesrState.Normal;  //if(uesrState==UesrState.Normal)  //{  // Console.WriteLine("正常状态");  //}  Console.WriteLine(uesrState.GetRemark());  Console.WriteLine(UesrState.Forzem.GetRemark());  Console.WriteLine(UesrState.Deleted.GetRemark());  } |

## Remark 封装 、attribute 验证

**当需要后台数据校验时，例如 需QQ 在10001~999999999999范围内**

传统的方法如下：



利用特性，扩展类的做法如下：

|  |
| --- |
| public class People  {  public People()  {  Console.WriteLine("{0}被创建", this.GetType().FullName);  }  public int Id { get; set; }  public string Name { get; set; }  public string Description { get; set; }  /// <summary>  /// 范围 10001~999999999999  /// </summary>  [LongAttribute(10001,999999999999)]  public long QQ { get; set; } |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Reflection;  using System.Text;  using System.Threading.Tasks;  namespace \_08属性  {  public class LongAttribute:Attribute  {  private long \_Min = 0;  private long \_Max = 0;  public LongAttribute(long min,long max)  {  this.\_Max = max;  this.\_Min = min;  }  public bool Validate(object value)  {  if(value!=null&&!string.IsNullOrWhiteSpace(value.ToString()))  {  if (long.TryParse(value.ToString(),out long IResult)) //将value转换为long  {  if (IResult>this.\_Min&& IResult<this.\_Max)  {  return true;  }  }    }  return false;  }  }  /// <summary>  /// 扩展类  /// </summary>  public static class ValidateExtension  {  public static bool Validate(this object oObject)  {  Type type = oObject.GetType();  foreach(var prop in type.GetProperties())  {  if (prop.IsDefined(typeof(LongAttribute), true))  {  LongAttribute attribute=(LongAttribute)prop.GetCustomAttribute(typeof(LongAttribute), true);  if (!attribute.Validate(prop.GetValue(oObject)))  {  return false;  }    }    }  return true;  }  }  } |
| 调用  {  People p = new People()  {  Id=123,  Name="haha",  QQ=123456  };  p.Validate();  } |

升级

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Reflection;  using System.Text;  using System.Threading.Tasks;  namespace \_08属性  {  public abstract class AbstractValidateAttribute : Attribute  {  public abstract bool Validate(object value);  }  public class LongAttribute: AbstractValidateAttribute  {  private long \_Min = 0;  private long \_Max = 0;  public LongAttribute(long min,long max)  {  this.\_Max = max;  this.\_Min = min;  }  public override bool Validate(object value)  {  if(value!=null&&!string.IsNullOrWhiteSpace(value.ToString()))  {  if (long.TryParse(value.ToString(),out long IResult)) //将value转换为long  {  if (IResult>this.\_Min&& IResult<this.\_Max)  {  return true;  }  }    }  return false;  }  }  public class LengAttribute : AbstractValidateAttribute  {  private int \_Min = 0;  private int \_Max = 0;  public LengAttribute(int min, int max)  {  this.\_Max = max;  this.\_Min = min;  }  public override bool Validate(object value)  {  if (value != null && !string.IsNullOrWhiteSpace(value.ToString()))  {  int length = value.ToString().Length;  if (length > this.\_Min && length < this.\_Max)  {  return true;  }  }  return false;  }  }  /// <summary>  /// 扩展类  /// </summary>  public static class ValidateExtension  {  public static bool Validate(this object oObject)  {  Type type = oObject.GetType();  foreach (var prop in type.GetProperties())  {  if (prop.IsDefined(typeof(AbstractValidateAttribute), true))  {  object[] attributeArray = prop.GetCustomAttributes(typeof(AbstractValidateAttribute), true);  foreach (AbstractValidateAttribute attribute in attributeArray)  {  if (!attribute.Validate(prop.GetValue(oObject)))  {  return false;  }  }  }  //if (prop.IsDefined(typeof(LengAttribute), true))  //{  // LengAttribute attribute = (LengAttribute)prop.GetCustomAttribute(typeof(LengAttribute), true);  // if (!attribute.Validate(prop.GetValue(oObject)))  // {  // return false;  // }  //}  }  return true;  }  }  } |