

Мета данни

Мета данни – система, която свързва системата на типовете и изпълнителната машина.

Причини за описване на типовете чрез мета данни в CLR компилаторите:

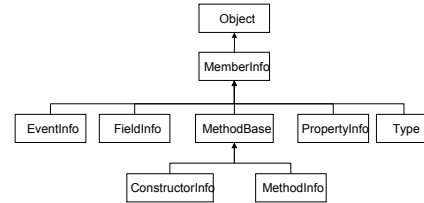
1. Позволяват типове, дефинирани в един език, да се използват в друг език – взаимна работа в CLR.
2. Изпълнителната система използва мета данните, за да управлява обектите – разполагане на обектите в паметта, управление на паметта и сигурност.

Съществуват типове и методи за четене и запис на мета данни – постига се по-висока степен на абстракция.

Отражение (Reflection)

Отражение – фундаментален механизъм в CLR за прегледане на мета данните на типовете по време на изпълнение.

Иерархия на класовете за отражение



```

using System;
using System.Reflection;
namespace TestReflection
{
    public class TestReflection
    {
        static void Main(string[] args)
        {
            Type t = typeof(Object);
            Console.WriteLine(t.Name);
            MemberInfo[] members = t.GetMembers();
            foreach (MemberInfo m in members)
                Console.WriteLine("!" + m);
        }
    }
}
    
```

Връща името на типа.

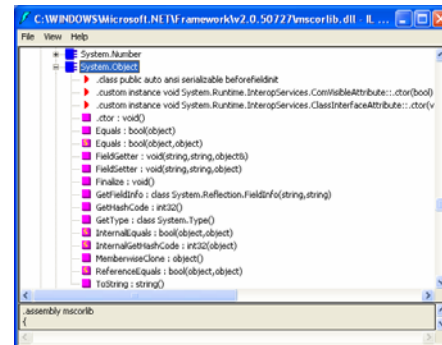
Връща масив от тип MemberInfo, представящ членовете на типа.

Резултати:

```

Object
System.Type GetType()
System.String ToString()
Boolean Equals(System.Object)
Boolean Equals(System.Object, System.Object)
Boolean ReferenceEquals(System.Object, System.Object)
Int32 GetHashCode()
Void .ctor()
    
```

Четене и изобразяване на мета данни – ILDASM (Intermediate Language Disassembler)



Атрибути

Атрибути – механизъм за добавяне на декларативна информация към програмен елемент, която се извлича по време на изпълнение.

1. Дефиниране на атрибути

- а) производен клас на базовия клас System.Attribute;
- б) суфикс на атрибутния клас Attribute;
- в) суфиксът се изпуска при използването.

```

public class име_на_атрибут : System.Attribute
{
}
    
```

2. Параметри на атрибути

- а) позиционни параметри – всеки public конструктор дефинира последователност от позиционни параметри;
 - б) наименовани параметри – всяко нестатично public поле и свойство за запис/четене.
3. Добавяне на атрибут – инстанция на атрибутния клас се добавя към тип или член и присъства в мета данните за типа

```

[име_на_атрибут(списък от позиционни параметри,
име_на_наименован_параметър = стойност, ...)]
    
```

3. Видове атрибути

- а) глобални атрибути (за асембли или модул);
- б) декларация на типове;
- в) декларация на членове на класове;
- г) декларация на членове на интерфейси;
- д) декларация на членове на структури;
- е) декларация на членове на изброими типове;
- ж) декларация на достъп до свойства;
- з) декларация на достъп до събития;
- и) списъци от формални параметри.

Атрибут `STAThreadAttribute` (STA – Single Threaded Apartment) – модел за приложение с една нишка.

```
class Class1
{
    [STAThread]
    static void Main(string[] args)
    {
    }
}
```

Атрибут `AttributeUsageAttribute` – определя използването на друг атрибутен клас.

```
[AttributeUsage(AttributeTargets.validOn, AllowMultiple = true/false,
    Inherited = true/false)]
```

`validOn` – валиден програмен елемент

`AttributeTargets`.Module | Class | Struct | Enum | Constructor | Method | Property | Field | Event | Interface | Parameter | Delegate | ReturnValue | GenericParameter | All

Пример:

Потребителски атрибут `DeveloperAttribute`

- запазва името на разработчика като позиционен параметър;
- прилага се за клас или структура;
- допуска прилагане на много инстанции на атрибута.

Потребителски атрибут `IsTestedAttribute`

- запазва датата на тестване на тип или член като незадължителен (наименован) параметър;
- прилага се за всички типове и членове;
- допуска прилагане на много инстанции на атрибута.

Потребителски атрибут `ValidLengthAttribute`

- запазва минималната и максималната дължина на поле като позиционни параметри и съобщение като незадължителен (наименован) параметър;
- прилага се за поле или свойство.

```
using System;
namespace CustomAttributes
{
    [AttributeUsage(AttributeTargets.Class | AttributeTargets.Struct,
        AllowMultiple = true)]
    public class DeveloperAttribute : Attribute // Разработчик
    {
        private string name; // име
        public DeveloperAttribute(string name)
        {
            this.name = name;
        }
        public string Developer
        {
            get { return name; }
        }
        public override string ToString()
        {
            return "Разработчик : " + Developer;
        }
    }
}
```

```
using System;
namespace CustomAttributes
{
    [AttributeUsage(AttributeTargets.All, AllowMultiple = true)]
    public class IsTestedAttribute : Attribute // Тестван
    {
        private string date; // дата
        public string Date
        {
            get { return date; }
            set { date = value; }
        }
        public IsTestedAttribute()
        {
            date = null;
        }
        public override string ToString()
        {
            string value = "Тестван!";
            if (date != null)
                value += " Дата: " + date;
            return value;
        }
    }
}
```

```
using System;
namespace CustomAttributes
{
    [AttributeUsage(AttributeTargets.Property | AttributeTargets.Field)]
    public class ValidLengthAttribute : Attribute
    {
        private int min;           // минимална дължина
        private int max;           // максимална дължина
        private string message;    // съобщение
        public ValidLengthAttribute(int min, int max)
        {
            this.min = min;
            this.max = max;
        }
        public string Message
        {
            get { return (message); }
            set { message = value; }
        }
        public bool IsValid(string theValue)
        {
            int length = theValue.Length;
            if (length >= min && length <= max) return true;
            return false;
        }
    }
}
```

```
using System;
using System.Reflection;

namespace TestAttributes
{
    [CustomAttributes.Developer("Александър Русев")]
    [CustomAttributes.Developer("Владислав Здравков")]
    [CustomAttributes.IsTested]
    public class TestAttributes
    {
        [CustomAttributes.ValidLength(4, 30,
            Message = "Заглавието трябва да има дължина между 4 и 30 символа")]
        public string title;           // заглавие

        [CustomAttributes.IsTested(Date = "01.02.2008")]
        [CustomAttributes.IsTested(Date = "05.02.2008")]
        public void MethodA() {}

        [CustomAttributes.IsTested(Date = "19.02.2008")]
        public void MethodB() {}

        public void MethodC() {}
    }
}
```

Съкратеното име на атрибут се използва само за присъединяване на атрибут

```
static void Main(string[] args)
{
    // Изобразява атрибутите за класа TestAttributes
    Type t = typeof(TestAttributes);
    object[] attributes = t.GetCustomAttributes(true);
    Console.WriteLine("Атрибути за: " + t.Name);
    foreach (object o in attributes)
    {
        Console.WriteLine("t" + o);
    }
    Console.WriteLine();
}
```

Връща масив от тип **Object** с всички атрибути за дадения тип – **true** показва да се търси в наследниците

Резултати:
Атрибути за: TestAttributes
Тестван!
Разработчик: Александър Русев
Разработчик: Владислав Здравков

```
// Изобразява атрибутите за методите на класа TestAttributes
MemberInfo[] members = t.GetMethods();
Console.WriteLine("Атрибути за методите на класа: " + t.Name);
foreach (MethodInfo method in members)
{
    bool flag = false;
    foreach (Attribute attr in method.GetCustomAttributes(true))
    {
        Console.WriteLine("\t{0}: {1}", method.Name, attr);
        if (attr is CustomAttributes.IsTestedAttribute)
            flag = true;
    }
    if (!flag)
        Console.WriteLine("\t{0}: Не е тестван!", method.Name);
}
Console.WriteLine();
```

Връща масив от тип **MemberInfo** за публичните методи на типа

Връща масив от тип **Attribute** за метод

```
Резултати:
Атрибути за: TestAttributes
Тестван!
Разработчик: Александър Русев
Разработчик: Владислав Здравков

Атрибути за методите на класа: TestAttributes
MethodA: Тестван! Дата: 05.02.2008
MethodA: Тестван! Дата: 01.02.2008
MethodB: Тестван! Дата: 19.02.2008
MethodC: Не е тестван!
GetType: Не е тестван!
ToString: Не е тестван!
Equals: Не е тестван!
GetHashCode: Не е тестван!

Атрибути за полетата на класа: TestAttributes
title: CustomAttributes.ValidLengthAttribute
Коректна дължина!
```

```
// Изобразява атрибутите за полетата на класа TestAttributes
TestAttributes test = new TestAttributes();
test.title = "Тестване на роторителски атрибути";
FieldInfo[] fields = t.GetFields();
Console.WriteLine("Атрибути за полетата на класа: " + t.Name);
foreach (FieldInfo field in fields)
    foreach (Attribute attr in field.GetCustomAttributes(true))
    {
        Console.WriteLine("\t{0}: {1}", field.Name, attr);
        CustomAttributes.ValidLengthAttribute via =
            attr as CustomAttributes.ValidLengthAttribute;
        if (null != via)
        {
            string theValue = (string)field.GetValue(test);
            Console.WriteLine(via.IsValid(theValue) ?
                "\tКоректна дължина!" : "\tНекоректна дължина!");
        }
    }
Console.WriteLine();
}
```

Връща масив от тип **FieldInfo** за публичните полета на типа

Връща стойността на поле

4. Правила:

- а) първо се определят позиционните параметри, а след това наименованите в произволен ред;**
- б) позиционен параметър не може да се именува;**
- в) наименованите параметри са полета с `public` достъп или свойства с нестатичен `set` метод;**
- г) параметрите на атрибутите могат да бъдат:**
 - `bool, byte, char, double, float, int, long, short, string;`
 - `System.Type;`
 - `object;`
 - `enum` и тип, в който е вграден, с `public` достъп;
 - едномерен масив от горните типове.
- д) конструкторът на атрибут не може да има клас като параметър (атрибутите се присъединяват по време на проектирането – все още не са създадени обекти на класове).**

5. Предефинирани атрибути

.NET атрибут	Отнася се за	Описание
<code>AttributeUsage</code>	клас	Определя правилното използване на друг атрибутен клас.
<code>CLSCompliant</code>	всички	Показва съвместимост на програмния елемент със CLS (Common Language Specification).
<code>Conditional</code>	метод	Компилаторът ще компилира метод, за който е дефиниран дадения низ.
<code>DllImport</code>	метод	Определя местоположението на DLL, съдържащ реализацията на външен метод.
<code>MTAThread</code>	Метод (<code>Main</code>)	Подразбира се многонишков модел за приложението.
<code>NonSerialized</code>	поле	Полетата не ще бъдат сериализирани.
<code>Obsolete</code>	всички без асембли, модул, параметър и <code>return</code>	Остарял елемент, който ще бъде премахнат в бъдещите версии на продукта.
<code>ParamArray</code>	параметър	Единствен параметър може да се разглежда като <code>params</code> .

.NET атрибут	Отнася се за	Описание
<code>Serializable</code>	клас, структура, изброим тип, делегат	Всички <code>public</code> и <code>private</code> полета могат да се сериализират.
<code>STAThread</code>	Метод (<code>Main</code>)	Подразбира се еднонишков модел за приложението.
<code>StructLayout</code>	клас, структура	Определя <code>Auto, Explicit</code> или <code>Sequential</code> данни.
<code>ThreadStatic</code>	поле (<code>static</code>)	Реализира локално нишково съхранение (TLS) – всяка нишка има свое собствено копие на статично поле, което не се споделя от нишките.

Пример: Атрибут `ObsoleteAttribute` – ако вторият параметър е `true`, компилаторът ще изведе съобщение за грешка при извикване на метода; ако е `false` – кодът се компилира без предупреждения и грешки; първият параметър е част от диагностиката на компютъра при генериране на грешка.

```
[Obsolete("Използвайте NewMethod вместо OldMethod", true)]
public static void OldMethod()
{
    Console.WriteLine("Old");
}
public static void NewMethod()
{
    Console.WriteLine("New");
}
...
TestAttributes.OldMethod();
```

Грешка: 'TestAttributes.TestAttributes.OldMethod()' is obsolete: 'Използвайте NewMethod вместо OldMethod'

Пример: Атрибут `DllImportAttribute` – определя, че достъпът до ASCII версията на външния метод `MessageBoxA` се намира в `user32.dll`.

```
using System.Runtime.InteropServices;
...
[DllImport("user32.dll")]
public static extern int MessageBoxA(int p, string m, string h, int t);
...
MessageBoxA(0, "Добре, дошли!", "Пример с DllImport", 0);
```

