Notas das aulas
sobre
Reflexão e *Custom Attributes*

Nuno Leite, DEETC-ISEL,
E-mail: nleite@cc.isel.ipl.pt

2018

1 Reflexão

CLR via C#, Cap. 22

Sistema de tipos *Object Oriented* com informação da metadata.

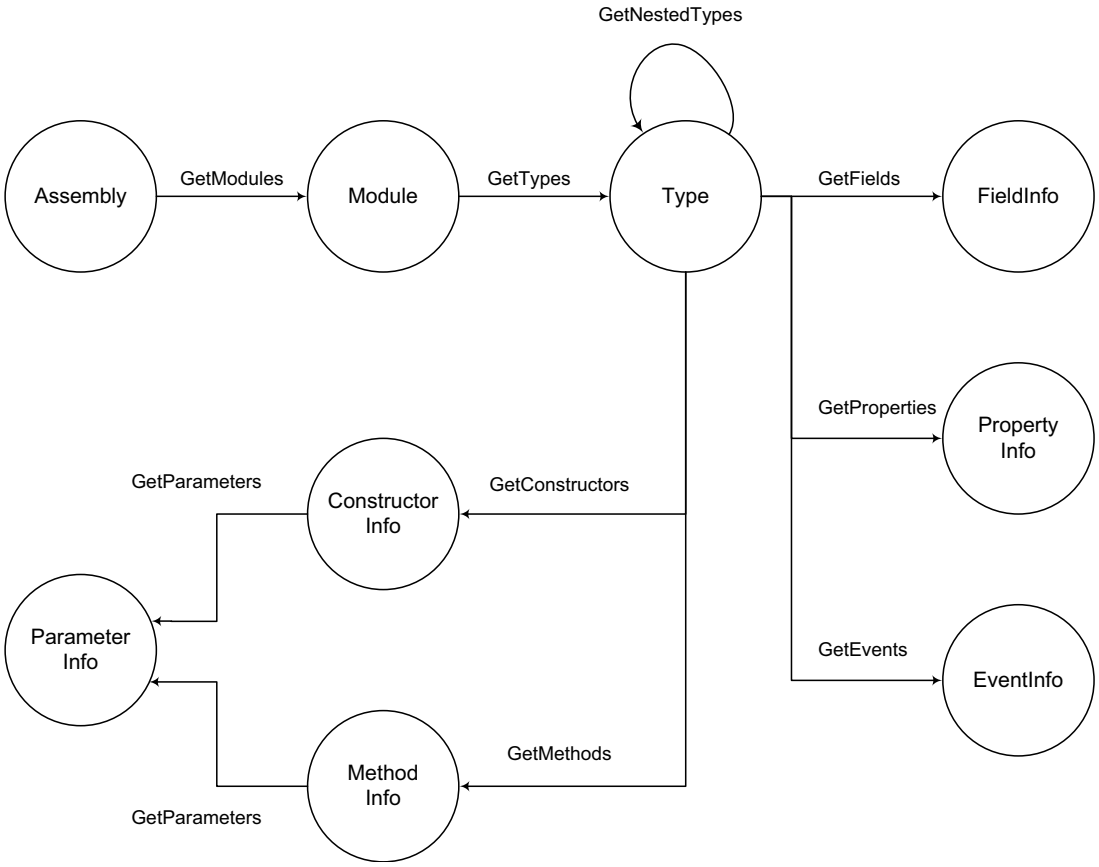

Figura 1: Modelo de classes do sistema de reflexão.

O tipo principal do *namespace* `System.Reflection` é o tipo `MemberInfo`.

- um tipo contém membros;
- a descrição de cada um destes membros é obtida através do método `System.Type.GetMembers`;
- cada descrição de membro é uma instância de um tipo derivado de `System.Reflection.MemberInfo`.

Propriedades de `MemberInfo`:

- `Name`: retorna o nome do membro
- `MemberType`: retorna uma instância do enumerado `System.Reflection.MemberTypes` indicando se o membro é um campo, método, evento, etc.
- `ReflectedType`: retorna o nome do tipo cujo membro (método, construtor, etc.) foi usado para obter esta instância de `MemberInfo`
- `DeclaringType`: retorna o nome do tipo que declarou este membro.

Nota: Existem situações em que as propriedades `ReflectedType` e `DeclaringType` retornam informação distinta. Ex.: membro declarado na classe base mas inspeccionado na classe derivada.
O código seguinte mostra todos os membros **públicos** (de instância e de tipo):

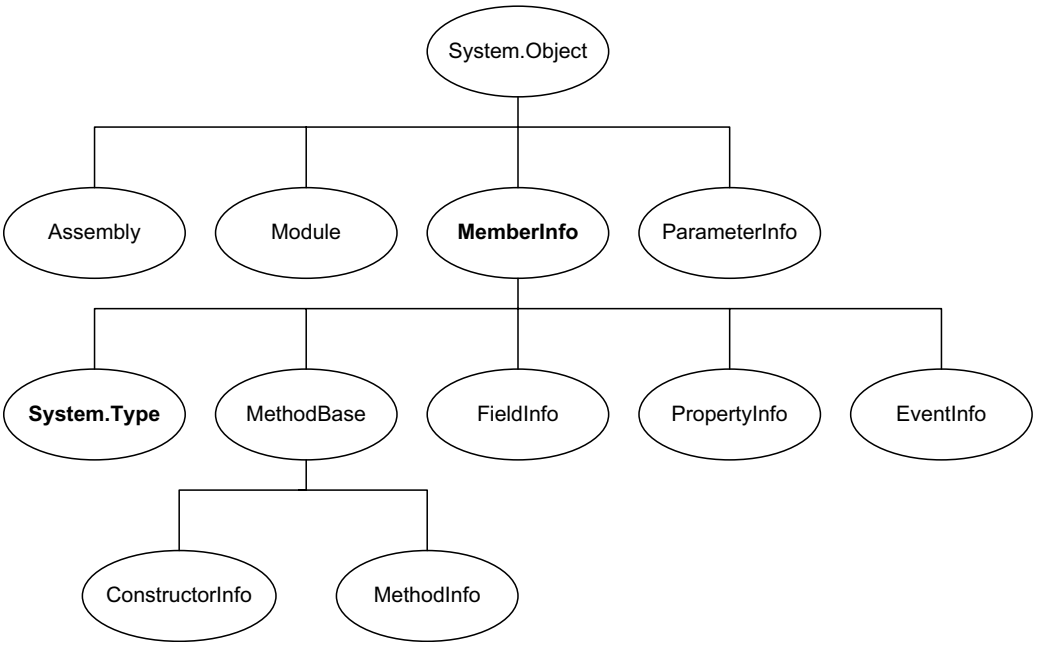

Figura 2: Hierarquia de tipos do sistema de reflexão.

```
public static void ShowPublicMembers(Type type) {
   // get all the public type's members
   MemberInfo[] members = type.GetMembers();
   // walk the list of members
   for (int i = 0; i < members.Length; i++)
      Console.WriteLine("{0}  {1} ", members[i].MemberType, members[i].Name);
}
```

Para alterar a procura de membros, deve-se usar valores do enumerado `BindingFlags`. A alteração seguinte mostra todos os membros (públicos e privados, de instância e de tipo):

```
   ...
   BindingFlags f = BindingFlags.Static
                  | BindingFlags.Instance
                  | BindingFlags.Public
                  | BindingFlags.NonPublic
                  | BindingFlags.FlattenHierarchy;

   MemberInfo[] members = type.GetMembers(f);
   ...
}
```

O valor `BindingFlags.FlattenHierarchy` especifica o seguinte:

- Membros estáticos públicos e protegidos presentes na hierarquia são retornados.
- Membros estáticos privados em classes base não são retornados.
- Membros estáticos incluem campos, métodos, eventos e propriedades.
- Tipos aninhados não são retornados.

1.1 Membros da classe `System.Type`

Propriedades:

```
IsAbstract
IsSealed
IsArray
IsClass
IsValueType
IsEnum
IsGenericTypeDefinition
IsGenericParameter
IsInterface
IsPrimitive
IsNestedPrivate
IsNestedPublic
```

Métodos:

```
GetConstructors()
GetEvents()
GetFields()
GetInterfaces()
GetMembers()
GetMethods()
GetNestedTypes()
GetProperties()
FindMembers()
GetType()
InvokeMember()
```

1.2 Obter uma referência para um tipo

1. Através da invocação `System.Object.GetType()`

   ```
   // Obtain type information using a SportsCar instance
   SportsCar sc = new SportsCar();
   Type t = sc.GetType();
   ```

   Restrições: é necessário ter conhecimento do tipo em *compile-time* e que o tipo resida em memória

2. Através da invocação `System.Type.GetType(String)`

   Recebe uma String contendo o nome completo (*full name*) do tipo do qual se pretende examinar a metadata

   Não é necessário ter uma referência para o objecto desse tipo em *compile-time*.

   Desvantagens:

   - Falível
   - Desempenho

     Vai procurar em todos os tipos dentro do *assembly* chamador se existe algum com o nome indicado. Se não encontrar, procura no *assembly* `mscorlib.dll`. Se não encontrar, lança uma excepção.

```
// Obtain type information using the static Type.GetType() method
// (don't throw an exception if SportsCar cannot be found and ignore case)
Type t = Type.GetType("CarLibrary.SportsCar", false, true);
```

3. Através do operador `typeof()`

```
// Get the Type using typeof
Type t = typeof(SportsCar);
```

Restrições: é necessário ter conhecimento do tipo em *compile-time*

Não é necessário ter uma referência para o objecto desse tipo

Se o tipo não existe, dá erro em tempo de compilação.

1.3 Carregamento de um *assembly*

`Assembly.LoadFrom(String path)`

- Dado o *path* para o módulo contendo o manifesto, faz o carregamento do respectivo *assembly*.
- Retorna uma instância de `Assembly`.

2 *Custom Attributes*

CLR via C#, Cap. 17

- Oferecem uma forma de anotar/marcar tipos e/ou membros com atributos definidos pelo utilizador.
- Podem influenciar o comportamento do compilador ou máquina virtual.
- Aparecem imediatamente antes da declaração do tipo ou membro (*target*) a que se aplica esse *custom attribute*
- Sintaxe para definição de *custom attributes*:

```
[NomeAtributo(param1, ...)]
Declaração do tipo ou membro
...
```

Regra: O nome do *custom attribute* deve ter o sufixo `Attribute`

- Em C#, este sufixo pode ser omitido na utilização

2.1 Definição de *custom attributes*

- classe derivada do tipo `Attribute`;
- os construtores da classe especificam os parâmetros que podem ser passados a esse *custom attribute*.

Exemplo:

```
class MyAttribute : Attribute {
   private int _myvalue;
   public MyAttribute(int value) {
      _myvalue = value;
   }
   public int MyValue {
      get { return _myvalue; }
   }
}

class A {
   [My] // Erro: não há construtor sem parâmetros
   [My(7)] // OK
   private void M() { ... }
}
```

2.2 Obter os *custom attributes* definidos num tipo ou membro

Existem três métodos/propriedades para verificar se um dado *custom attribute* está definido:

- método `GetCustomAttributes`: retorna um `Object[]` contendo as instâncias dos atributos aplicados a este membro. Usado quando `AllowMultiple=true`.
- método `GetCustomAttribute`: retorna um `Object` contendo a instância do atributo aplicado a este membro. . Usado quando `AllowMultiple=false`.
- propriedade `IsDefined`: retorna `true` se pelo menos uma instância do atributo está definida neste membro. Vantagem: mais rápido, pois não cria a instância de *custom attribute*.

```
static void ShowAttributes(Object o) {
   foreach (MemberInfo mi in o.GetType().GetMembers()) {
      Object[] cas = mi.GetCustomAttributes(false);
      foreach(Object obj in cas) {
       Console.WriteLine("membro {0}: , Custom Attribute: ", mi.Name, obj.ToString());

       // Se estou à procura de um atribute específico
       // MyAttribute ma = obj as MyAttribute;
       // if (ma != null)
       // Console.WriteLine(ma.MyValue);
      }
   }

}
```

- `Activator.CreateInstance(Type A, params)`: cria uma instância do tipo representado por `A` passando os parâmetros especificados ao construtor.

O âmbito de acção de um *custom attribute* é controlado pela utilização de outro *custom attribute*: `AttributeUsage`
cujo construtor recebe como parâmetro uma instância de `AttributeTargets`.

```
public enum AttributeTargets
{
All, Assembly, Class, Constructor,
Delegate, Enum, Event, Field,
Interface, Method, Module, Parameter,
Property, ReturnValue, Struct
}
```

```
[AttributeUsage(AttributeTargets.Field, AllowMultiple=false)]
class MyAttribute : Attribute {
      private int _val;
      private String _msg;
      public MyAttribute(String m) {
         _msg = m
      }
      public int Val {
         get { return _val; }
         set { _val = value; }
      }
   }
```

Notas:

- `AllowMultiple(false)` não permite segundas instâncias deste atributo no mesmo *target*.
- Propriedade `Val` é opcional na utilização. Representa um *named parameter* do atributo.
- Se não existir um construtor sem parâmetros, a especificação dos parâmetros é obrigatória na utilização do atributo.

Exemplo:

```
[My("Ola")]  // Erro
class A {
   [My("Ola", Val=7)] // OK
   int f1;
   [My("Ola 1")] // OK
   int f2;
   [My(Val=3)] // Erro
   int f3;
   [My("Ola")] // OK
   [My("Adeus")] // Erro
   int f4;
}
```

2.3 Aplicação explícita de *custom attributes*

Por vezes, é necessário remover a ambiguidade sobre a que membro um determinado atributo deve ser aplicado. Para isso usa-se na declaração o prefixo designando o *target*:

```
[assembly: MyAttribute(1)]
[module: MyAttribute(2)]

[type: MyAttribute(3)]
class SomeType {
   [property: MyAttribute(4)]
   public String SomeProp { get { ... } }
   [event: MyAttribute(5)]
   public event EventHandler SomeEvent;
   [field: MyAttribute(6)]
   public Int32 SomeField;
   [return: MyAttribute(7)]
   [method: MyAttribute(8)]
   public Int32 SomeMethod(
```

```
      [param: MyAttribute(9)]
      Int32 someParam) {
         return someParam;
   }
}
```