Рефлексия

Юрий Литвинов
y.litvinov@spbu.ru

26.10.2023

Рефлексия

- Позволяет во время выполнения получать информацию о типах
  - И главное, создавать объекты этих типов и вызывать их методы
- Зачем:
  - Плагины
  - Анализаторы кода
  - Тестовые системы
  - Библиотеки сериализации
  - Библиотеки для работы с базами данных
  - Inversion of Control-контейнеры
  - ...
- Проблемы:
  - Медленно
  - Нет помощи от системы типов

Рефлексия в .NET

- Пространство имён System.Reflection
- Байт-код хранит всю информацию о типах
  - И даже параметрах-типах у генериков, в отличие от Java
- Самая крупная штука, которой оперирует рефлексия — **сборка**
  - .dll или .exe, единица развёртывания программы
    - На самом деле это неправда, детали см. в CLR via C#
- Сборка хранит метаинформацию:
  - Таблицы модулей, типов, методов, полей, параметров, свойств и событий
- На всё это можно посмотреть в ILDasm
- AppDomain — логическая группа сборок во время выполнения

Загрузка сборки

```
public class Assembly {
   public static Assembly Load(AssemblyName assemblyRef);
   public static Assembly Load(String assemblyString);
   public static Assembly Load(byte[] rawAssembly)
   public static Assembly LoadFrom(String path);
   public static Assembly ReflectionOnlyLoad(String assemblyString);
   public static Assembly ReflectionOnlyLoadFrom(String assemblyFile);
}
```

например,

```
var a = Assembly.LoadFrom(@"http://example.com/ExampleAssembly.dll");
```

Такая сборка должна быть ещё не загружена. Выгружать сборки нельзя.

Сильные и слабые имена сборок (1)

- Сильные сборки подписаны асимметричным шифром
  - Публичная часть ключа внедряется в саму сборку
  - От сборки считается SHA-1-хеш, шифруется приватным ключом и внедряется в сборку
  - CLR при загрузке сборки считает от неё SHA-1-хеш и проверяет, что он совпал с подписанным
  - Позволяет проверить, что сборка именно такая, какой её собирал автор, но не позволяет проверить идентичность автора

Сильные и слабые имена сборок (2)

- Пример сильного имени:
  "MyTypes, Version=1.0.8123.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
  - Однозначно идентифицирует сборку
  - PublicKeyToken — короткий хеш публичного ключа
    - Естественно, не криптостоек, поэтому CLR проверяет настоящий ключ при загрузке сборки
- Сборка с сильным именем может ссылаться только на сборки с сильными именами
- Сборка с сильным именем может быть установлена в GAC
  - GACUtil — утилита для манипуляции GAC, в стандартной поставке

Пример

Распечатать имена всех типов в сборке

```
using System;
using System.Reflection;

public static class Program {
   public static void Main() {
      string dataAssembly = "System.Data, version=4.0.0.0, "
            + "culture=neutral, PublicKeyToken=b77a5c561934e089";
      LoadAssemblyAndShowPublicTypes(dataAssembly);
   }

   private static void LoadAssemblyAndShowPublicTypes(string assemblyId) {
      var a = Assembly.Load(assemblyId);
      foreach (Type t in a.ExportedTypes) {
         Console.WriteLine(t.FullName);
      }
   }
}
```

Создание экземпляра объекта

- System.Activator.CreateInstance — можно передавать тип или строку с именем типа
  - Версии со строкой возвращают System.Runtime.Remoting.ObjectHandle, надо вызвать Unwrap()
- System.Activator.CreateInstanceFrom — вызывает LoadFrom для сборки
- System.Reflection.ConstructorInfo.Invoke — просто вызов конструктора (несколько дольше писать, чем предыдущие варианты)
- Рефлексия ничего не знает о синонимах
  - То есть int везде называется System.Int32

Пример:

```
var zero = Activator.CreateInstance("mscorlib.dll", "System.Int32").Unwrap();
```

Создание экземпляра типа-генерика

```
using System;
using System.Reflection;

internal sealed class Dictionary<TKey, TValue> { }

public static class Program {
   public static void Main() {
      Type openType = typeof(Dictionary<,>);
      Type closedType = openType.MakeGenericType(
            typeof(String), typeof(Int32));
      Object o = Activator.CreateInstance(closedType);
      Console.WriteLine(o.GetType());
   }
}
```

Пример: как сделать свою плагинную систему

- Сделать отдельную сборку с описанием интерфейса плагина и типов данных, которые он использует
  - Менять её будет очень проблематично
- Сделать “ядро системы” — отдельную сборку, ссылающуюся на сборку с интерфейсом плагина
- Делать набор плагинов, ссылающихся на сборку с интерфейсом плагина и реализующих его

Пример: интерфейс плагина

```
namespace MyCoolSystem.SDK {
    public interface IAddIn {
        string DoSomething(int x);
    }
}
```

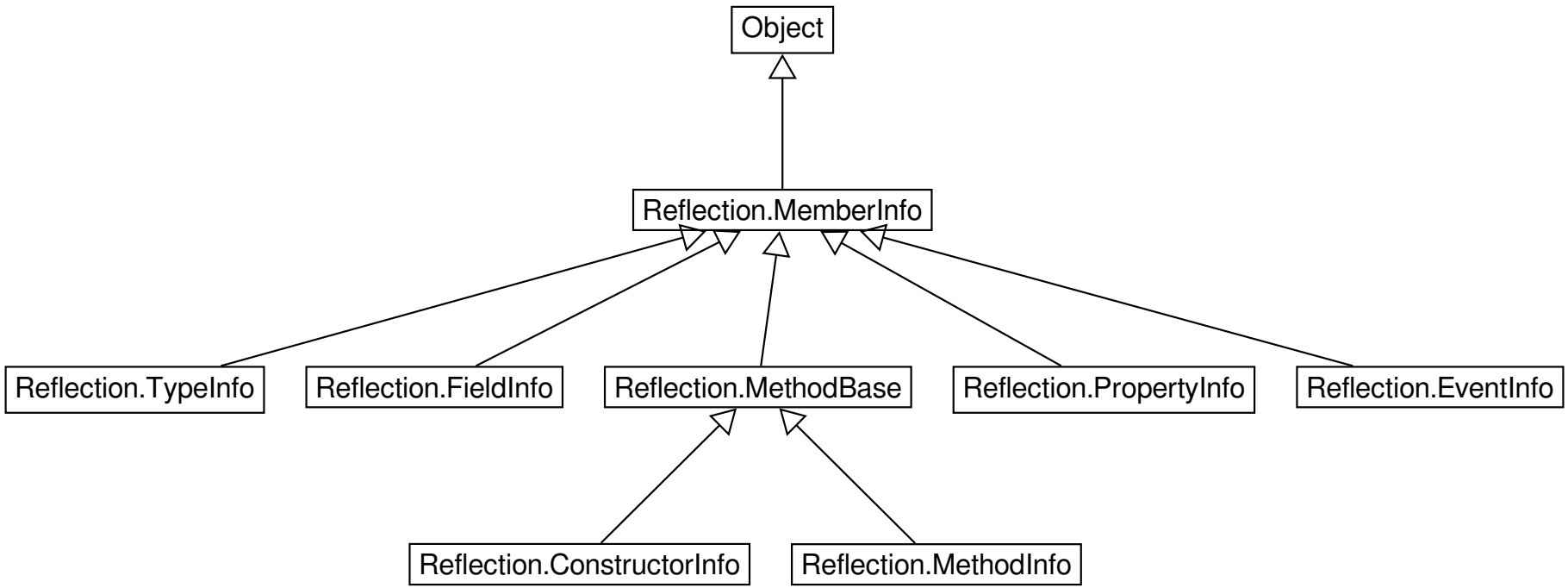
Пример: плагины

```
using MyCoolSystem.SDK;

public sealed class AddInA : IAddIn {
    public String DoSomething(int x) {
        return "AddInA: " + x.ToString();
    }
}

public sealed class AddInB : IAddIn {
    public String DoSomething(int x) {
        return "AddInB: " + (x * 2).ToString();
    }
}
```

Пример: ядро системы

```
public static class Program
{
    public static void Main()
    {
        string addInDir = Path.GetDirectoryName(Assembly.GetEntryAssembly().Location);
        var addInAssemblies = Directory.EnumerateFiles(addInDir, "*.dll");
        var addInTypes =
            addInAssemblies.Select(Assembly.Load)
                .SelectMany(a => a.ExportedTypes)
                .Where(t => t.IsClass
                        && typeof(IAddIn).GetTypeInfo().IsAssignableFrom(t.GetTypeInfo()));

        foreach (Type t in addInTypes)
        {
            var addIn = (IAddIn)Activator.CreateInstance(t);
            Console.WriteLine(addIn.DoSomething(5));
        }
    }
}
```

Информация о типах



Пример: распечатать информацию о полях и методах

```
using System;
using System.Reflection;

public static class Program {
   public static void Main() {
      Assembly[] assemblies = AppDomain.CurrentDomain.GetAssemblies();
      foreach (Assembly a in assemblies) {
         Console.WriteLine($"Assembly: {a}");
         foreach (Type t in a.ExportedTypes) {
            Console.WriteLine($"  Type: {t}");
            foreach (MemberInfo mi in t.GetTypeInfo().DeclaredMembers) {
               var typeName = mi switch {
                  FieldInfo _ => "FieldInfo",
                  MethodInfo _ => "MethodInfo",
                  ConstructorInfo _ => "ConstructorInfo",
                  _ => ""
               };
               Console.WriteLine($"    {typeName}: {mi}");
            }
         }
      }
   }
}
```

Полезные свойства MemberInfo

- Name (string) — имя члена класса
- DeclaringType (Type) — тип
- Module (Module) — модуль, в котором он объявлен
- CustomAttributes (IEnumerable<CustomAttributeData>) — коллекция атрибутов, соответствующих этому члену класса
  - Пример — модульные тесты

Как что-нибудь сделать с MemberInfo

- GetValue и SetValue для FieldInfo и PropertyInfo
- Invoke для ConstructorInfo и MethodInfo
- AddEventHandler и RemoveEventHandler для EventInfo

Пример: создать объект и вызвать его метод

```
using System;
using System.Reflection;
using System.Linq;

internal sealed class SomeType {
   public SomeType(int test) { }
   private int DoSomething(int x) => x * 2;
}

public static class Program {
   public static void Main() {
      Type t = typeof(SomeType);
      Type ctorArgument = Type.GetType("System.Int32");
      ConstructorInfo ctor = t.GetTypeInfo().DeclaredConstructors.First(
            c => c.GetParameters()[0].ParameterType == ctorArgument);
      Object[] args = { 12 };
      Object obj = ctor.Invoke(args);
      MethodInfo mi = obj.GetType().GetTypeInfo().GetDeclaredMethod("DoSomething");
      int result = (int)mi.Invoke(obj, new object[]{3});
      Console.WriteLine($"result = {result}");
   }
}
```

Атрибуты

- Способ добавить произвольную информацию к коду во время компиляции
- Эта информация может быть использована потом во время компиляции или во время выполнения
  - Типичный пример — атрибуты юнит-тестов (Test, ExpectedException, ...)
- Могут быть применены к сборке, типу, полю, методу, параметру метода, возвращаемому значению, свойству, событию, параметру-типу
- Могут иметь параметры
- На самом деле, экземпляры классов-наследников System.Attribute

Объявление своего атрибута

```
public enum Animal
{
    Dog = 1,
    Cat,
    Bird,
}

public class AnimalTypeAttribute : Attribute
{
    public AnimalTypeAttribute(Animal pet)
    {
        this.Pet = pet;
    }

    public Animal Pet { get; set; }
}
```

Использование атрибута

```
class AnimalTypeTestClass
{
    [AnimalType(Animal.Dog)]
    public void DogMethod() { }

    [AnimalType(Animal.Cat)]
    public void CatMethod() { }

    [AnimalType(Animal.Bird)]
    public void BirdMethod() { }
}
```

Получение атрибута рефлексией

```
static void Main(string[] args)
{
    var testClass = new AnimalTypeTestClass();
    Type type = testClass.GetType();

    foreach (MethodInfo mInfo in type.GetMethods())
    {
        foreach (Attribute attr in Attribute.GetCustomAttributes(mInfo))
        {
            // Check for the AnimalType attribute.
            if (attr.GetType() == typeof(AnimalTypeAttribute))
                Console.WriteLine(
                    "Method {0} has a pet {1} attribute.",
                    mInfo.Name, ((AnimalTypeAttribute)attr).Pet);
        }
    }
}
```

Ограничение области применения атрибута

```
namespace System {
    [AttributeUsage(AttributeTargets.Enum, Inherited = false)]
    public class FlagsAttribute : System.Attribute {
        public FlagsAttribute() {
        }
    }
}
```

Атрибуты у атрибутов!

Библиотеки сериализации

- Нужны для передачи объектов по сети или сохранения объектов
- Часто используемые форматы
  - XML
  - JSON
  - Protobuf
- Активно используют рефлексию

Пример, сериализация в XML, System.Xml.Serialization

Данные

```
public class Point
{
    public Point() { }
    public Point(int x, int y) { this.X = x; this.Y = y; }
    public int X { get; set; }
    public int Y { get; set; }
}

public class SomeData
{
    public int Radius { get; set; }
    public Point Center { get; set; }
    public List<Point> Points { get; } = new List<Point>();
}
```

Сама сериализация

```
var data = new SomeData { Radius = 10, Center = new Point(5, 5)};
data.Points.Add(new Point(1, 1));
data.Points.Add(new Point(2, 2));

var serializer = new XmlSerializer(typeof(SomeData));
using (var textWriter = new StringWriter())
{
    serializer.Serialize(textWriter, data);
    Console.WriteLine(textWriter.ToString());
}
```

Результат

```
<?xml version="1.0" encoding="utf-16"?>
<SomeData xmlns:xsi="..." xmlns:xsd="...">
  <Radius>10</Radius>
  <Center>
    <X>5</X>
    <Y>5</Y>
  </Center>
  <Points>
    <Point>
      <X>1</X>
      <Y>1</Y>
    </Point>
    <Point>
      <X>2</X>
      <Y>2</Y>
    </Point>
  </Points>
</SomeData>
```

То же с Newtonsoft.Json

```
var data = new SomeData { Radius = 10, Center = new Point(5, 5)};
data.Points.Add(new Point(1, 1));
data.Points.Add(new Point(2, 2));

var jsonSerializer = new JsonSerializer() { Formatting = Formatting.Indented };
using (var textWriter = new StringWriter())
{
    jsonSerializer.Serialize(textWriter, data);
    Console.WriteLine(textWriter.ToString());
}
```

Результат


```
{
  "Radius": 10,
  "Center": {
    "X": 5,
    "Y": 5
  },
  "Points": [
    {
      "X": 1,
      "Y": 1
    },
    {
      "X": 2,
      "Y": 2
    }
  ]
}
```

Сериализация графа объектов

Дьявол в деталях

```
var center = new Point(5, 5);
var data = new SomeData { Radius = 10, Center = center };
data.Points.Add(new Point(1, 1));
data.Points.Add(center);

var jsonSerializer = new JsonSerializer() {
    Formatting = Formatting.Indented,
    PreserveReferencesHandling = PreserveReferencesHandling.All
};
using (var textWriter = new StringWriter())
{
    jsonSerializer.Serialize(textWriter, data);
    Console.WriteLine(textWriter.ToString());
}
```

Результат


```
{
  "$id": "1",
  "Radius": 10,
  "Center": {
    "$id": "2",
    "X": 5,
    "Y": 5
  },
  "Points": [
    {
      "$id": "3",
      "X": 1,
      "Y": 1
    },
    {
      "$ref": "2"
    }
  ]
}
```

Десериализация

```
var deserializedData =
    JsonConvert.DeserializeObject<SomeData>(serializedData);
Console.WriteLine(deserializedData.Radius);
```

Управление именами полей

```
public class Videogame
{
    [JsonProperty("name")]
    public string Name { get; set; }

    [JsonProperty("release_date")]
    public DateTime ReleaseDate { get; set; }
}
```

© https://www.newtonsoft.com

Управление тем, что надо сериализовать, а что нет

```
[JsonObject(MemberSerialization.OptIn)]
public class File
{
    // Это поле не сериализуется,
    // потому что у него нет JsonPropertyAttribute
    public Guid Id { get; set; }

    [JsonProperty]
    public string Name { get; set; }

    [JsonProperty]
    public int Size { get; set; }
}
```

Ключевое слово dynamic

```
using System;

internal static class DynamicDemo
{
    public static void Main()
    {
        dynamic value;
        for (int demo = 0; demo < 2; demo++)
        {
            value = (demo == 0) ? (dynamic)5 : (dynamic)"A";
            value = value + value;
            M(value);
        }
    }

    private static void M(int n) { Console.WriteLine("M(int): " + n); }
    private static void M(string s) { Console.WriteLine("M(string): " + s); }
}
```

Генерация кода “на лету”

```
public static void Main() {
  AssemblyName assemblyName = new AssemblyName {Name = "HelloEmit"};
  AppDomain appDomain = AppDomain.CurrentDomain;
  AssemblyBuilder assemblyBuilder = appDomain.DefineDynamicAssembly(
    assemblyName, AssemblyBuilderAccess.Save);
  ModuleBuilder moduleBuilder =
    assemblyBuilder.DefineDynamicModule(assemblyName.Name, "Hello.exe");
  TypeBuilder typeBuilder = moduleBuilder.DefineType("Test.MainClass",
    TypeAttributes.Public | TypeAttributes.Class);
  MethodBuilder methodBuilder = typeBuilder.DefineMethod("Main",
    MethodAttributes.Public | MethodAttributes.Static,
    typeof(int), new[] { typeof(string[]) });

  ILGenerator ilGenerator = methodBuilder.GetILGenerator();
  ilGenerator.Emit(OpCodes.Ldstr, "Hello, World!");
  ilGenerator.Emit(OpCodes.Call,
    typeof(Console).GetMethod("WriteLine", new[] { typeof(string) }));
  ilGenerator.Emit(OpCodes.Ldc_I4_0);
  ilGenerator.Emit(OpCodes.Ret);

  typeBuilder.CreateType();
  assemblyBuilder.SetEntryPoint(methodBuilder, PEFileKinds.ConsoleApplication);
  assemblyBuilder.Save("Hello.exe");
}
```