Рефлексия

Юрий Литвинов
yurii.litvinov@gmail.com

12.10.2018г

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
  - ...
- Проблемы:
  - Медленно
  - Нет помощи от системы типов

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");
```

Выгружать сборки нельзя

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

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);
    }
}
```

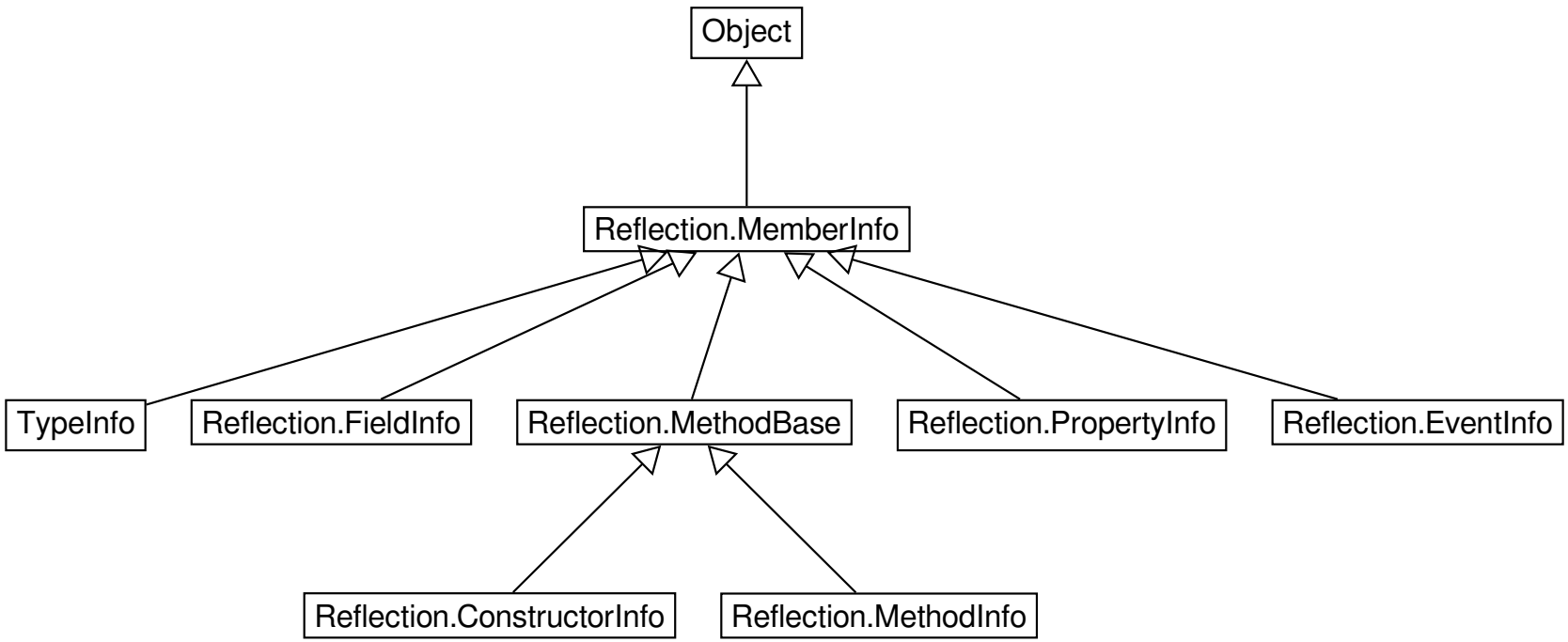
Пример: плагины

```
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 = string.Empty;
               if (mi is FieldInfo) typeName = "FieldInfo";
               if (mi is MethodInfo) typeName = "MethodInfo";
               if (mi is ConstructorInfo) typeName = "ConstructoInfo";
               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}");
   }
}
```

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");
}
```