

# Рефлексия

Юрий Литвинов  
y.litvinov@spbu.ru

28.10.2025

# 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
  - ▶ Плохо работает с Ahead-of-time-компиляцией

# 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
    - ▶ Для .NET Framework это только почти всегда так, 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 (поставляется с VS), ILSpy, DotPeek

# 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);  
    ...  
}
```

например,

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

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

## Загрузка только метаданных сборки

- ▶ `MetadataLoadContext` — словарь, отображающий имя сборки в загруженные метаданные
- ▶ Работает, даже когда сборку нельзя загрузить по-настоящему
  - ▶ Например, для другой целевой платформы
- ▶ Живёт в отдельном NuGet-пакете  
`System.Reflection.MetadataLoadContext`
- ▶ `MetadataAssemblyResolver` — абстракция алгоритма поиска нужной сборки
  - ▶ `PathAssemblyResolver` — ищет сборки в переданных путях
  - ▶ Пути должны включать не только целевую сборку, но и все её зависимости (как минимум, `typeof(object).Assembly.Location`)

# Сильные и слабые имена сборок

- ▶ 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 раньше считал от неё SHA-1-хеш и проверял, что он совпал с подписанным
- ▶ Пример сильного имени:  
`"MyTypes, Version=1.0.8123.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"`
- ▶ PublicKeyToken — короткий хеш публичного ключа
- ▶ Сборка с сильным именем может ссылаться только на сборки с сильными именами
- ▶ В современном .NET не очень актуально
  - ▶ В силу уклона в веб и контейнеризацию

# Пример

Распечатать имена всех типов в сборке

```
using System.Reflection;
```

```
var assembly = Assembly.Load("Example");
```

```
foreach (Type t in assembly.GetTypes())
```

```
{
```

```
    Console.WriteLine(t.FullName);
```

```
}
```

- ▶ Уже загруженную сборку так не загрузить (ошибки не будет, но список типов будет пустым)
- ▶ Бывает `GetTypes`, `DefinedTypes`, `GetExportedTypes`

## 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

```
Type openType = typeof(Dictionary<,>);  
Type closedType = openType.MakeGenericType(  
    typeof(string), typeof(int));  
object o = Activator.CreateInstance(closedType);  
Console.WriteLine(o.GetType());  
  
internal sealed class Dictionary<TKey, TValue> { }
```

## 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)  
        => "AddInA: " + x.ToString();  
}
```

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

## Пример: ядро системы

```
using System.Reflection;
```

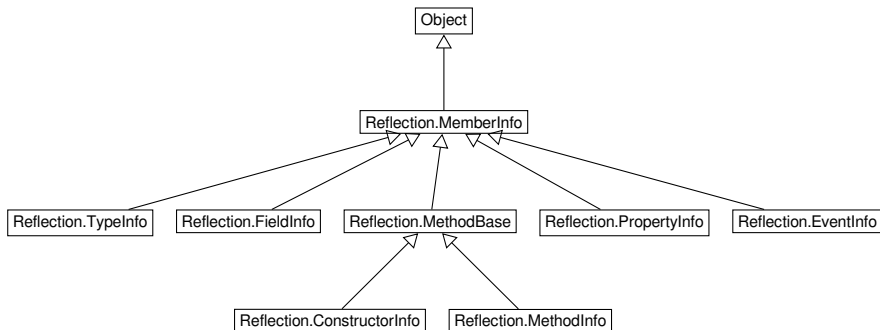
```
string addInDir = Path.GetDirectoryName(Assembly.GetEntryAssembly()?.Location)  
?? throw new Exception("Unable to load assembly");
```

```
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 = Activator.CreateInstance(t) as IAddIn;  
    Console.WriteLine(addIn?.DoSomething(5));  
}
```

# Информация о типах



# Пример: распечатать информацию о полях и методах

```
using System.Reflection;
```

```
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.Reflection;
```

```
Type t = typeof(SomeType);
Type ctorArgument = Type.GetType("System.Int32") ?? throw new Exception("Failed to load Int32");
ConstructorInfo ctor = t.GetTypeInfo().DeclaredConstructors.First(
    c => c.GetParameters()[0].ParameterType == ctorArgument);
object obj = ctor.Invoke([12]);
MethodInfo mi = obj.GetType().GetTypeInfo().GetDeclaredMethod("DoSomething")
    ?? throw new Exception("Failed to find method");

int result = (int)(mi.Invoke(obj, [3]) ?? throw new Exception("Method returned null"));
Console.WriteLine($"result = {result}");
```

```
internal sealed class SomeType
```

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

# UnsafeAccessorAttribute

- ▶ Атрибут, генерирующий код доступа к закрытому члену класса
- ▶ Появился в .NET 8
- ▶ «Рефлексия для бедных», но в десятки раз быстрее рефлексии

Пример:

```
using System.Runtime.CompilerServices;
```

```
static void CallStaticPrivateMethod() {
    StaticPrivateMethod(null);
```

```
[UnsafeAccessor(UnsafeAccessorKind.StaticMethod, Name = nameof(StaticPrivateMethod))]
extern static void StaticPrivateMethod(Class c);
```

```
}
```

```
static void GetSetStaticPrivateField() {
    ref int f = ref GetSetStaticPrivateField(null);
```

```
[UnsafeAccessor(UnsafeAccessorKind.StaticField, Name = "StaticPrivateField")]
extern static ref int GetSetStaticPrivateField(Class c);
```

```
}
```

```
public class Class {
    static void StaticPrivateMethod() { }
    static int StaticPrivateField;
```

```
}
```

# 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

```
using System.Reflection;
```

```
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 {mInfo.Name} has a pet " +  
                $"{((AnimalTypeAttribute)attr).Pet} attribute.");  
    }  
}
```

# Ограничение области применения атрибута

```
namespace System;
```

```
[AttributeUsage(AttributeTargets.Enum, Inherited = false)]
```

```
public class FlagsAttribute : System.Attribute {
```

```
    public FlagsAttribute() {
```

```
    }
```

```
}
```

Атрибуты у атрибутов!



# Ключевое слово dynamic

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

# Динамические сборки

- ▶ Динамическая сборка — сборка, существующая только в памяти
- ▶ Генерируется в рантайме средствами из `System.Reflection.Emit`
- ▶ Может быть сохранена на диск
  - ▶ Всегда хотели написать свой язык и компилятор, но боитесь машинно-зависимых оптимизаций?
- ▶ Для .NET неотличима от обычной сборки

# 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");
}

```