



Механизм Отражения (Reflection) в C#

Что такое Отражение?



Отражение – механизм, позволяющий во время выполнения обнаруживать и использовать типы и их члены, о которых во время компиляции ничего не было известно.

Большая часть необходимых для этого типов находится в пространстве имён **System.Reflection**.

Метаданные сборки



assembly (сборка), файл с расширением .exe или .dll

метаданные

код на языке
CIL

манифест

ресурсы (иконки, звуки, картинки и т.д)



КОМПИЛЯЦИЯ

исходный код на
языке C#

- Метаданные в .Net обязательны и универсальны
- Метаданные в .Net общедоступны
- Метаданные в .Net исчерпывающи
- Метаданные в .Net расширяемы
- Метаданные в .Net конструируемы

- Не контролируется безопасность типов на этапе компиляции
- Отражение работает медленно

Динамическая загрузка сборок



```
namespace System.Reflection
{
    // Summary:
    //     Represents an assembly, which is a reusable, versionable, and self-describing
    //     building block of a common language runtime application.
    public abstract class Assembly : _Assembly, IEvidenceFactory,
    ICustomAttributeProvider, ISerializable
    {
        public static Assembly Load(AssemblyName assemblyRef);
        public static Assembly Load(string assemblyString);
        public static Assembly LoadFile(string path);
        public static Assembly LoadFrom(string assemblyFile);
        public static Assembly ReflectionOnlyLoad(string assemblyString);
        public static Assembly ReflectionOnlyLoadFrom(string assemblyFile);

        ...
    }
}
```

Полное имя сборки



1. Имя (без расширения и пути)
2. Версия
3. Информация о локализации
4. Маркер открытого ключа
5. Архитектура процессора (опционально)

Пример:

```
"SomeAssembly, Version=2.0.0.0, Culture=neutral,  
PublicKeyToken=01234567890abcde,  
ProcessorArchitecture=MSIL"
```

Пример загрузки сборки по полному имени



```
using System;
using System.Reflection;

public class Example
{
    public static void Main()
    {
        string longName = "system, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089";
        Assembly assem = Assembly.Load(longName);
        Console.WriteLine(assem.FullName);
    }
}
```

Вывод:

System, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089

Assembly.LoadFrom vs. Assembly.LoadFile

LoadFrom сначала вызывает статический метод **GetAssemblyName**, который возвращает полное имя сборки из метаданных файла по указанному пути. После этого он передаёт это имя методу **Load**, который далее загружает сборку стандартным образом.

LoadFile же сразу загружает файл по указанному пути как новую сборку в текущий домен приложения. При этом он не разрешает автоматически зависимости этой сборки, т. е. все остальные сборки, на которые она ссылается, придётся загружать вручную.

Загрузка сборки из встроенного ресурса



```
AppDomain.CurrentDomain.AssemblyResolve += (sender, args) =>
{
    String resourceName = "MyDefaultNamespace" +
        new AssemblyName(args.Name).Name + ".dll";

    using (var stream = Assembly.GetExecutingAssembly().
        GetManifestResourceStream(resourceName))
    {
        Byte[] assemblyData = new Byte[stream.Length];
        stream.Read(assemblyData, 0, assemblyData.Length);
        return Assembly.Load(assemblyData);
    }
};
```



При первом обращении в домене приложений к типу CLR создаёт экземпляр **System.Type** и инициализирует поля объекта информацией о типе.

С его помощью можно динамически создавать экземпляры описываемого им типа, а также исполнять методы и работать со свойствами полученного объекта.

Поэтому тип **System.Type** является отправной точкой для операций с типами и объектами через механизмы Отражения.

Получение экземпляра System.Type



1. Через оператор **typeof**

```
using System;

namespace App1
{
    class Class1
    {
        static void Main()
        {
            Type type = typeof(Class1);
            Console.WriteLine(type.FullName);
            Console.ReadLine();
        }
    }
}
```



2. Через метод типа **System.Object** – **GetType**

```
Class1 app = new Class1();  
Type type = app.GetType();
```

3. Через статические методы типа **System.Type**

```
Type type = Type.GetType("System.Int32, mscorlib, Version=2.0.0.0, Culture=neutral,  
PublicKeyToken=b77a5c561934e089");  
Console.WriteLine(type.FullName);
```

Получение экземпляра System.Type



4. Через методы типа

Assembly

```
using System;
using System.Reflection;

namespace App1
{
    class Class1
    {
        static void Main()
        {
            foreach (Type type in Assembly.GetExecutingAssembly().GetTypes())
                Console.WriteLine(type.Name);
            Console.ReadLine();
        }
    }
}
```

Характеристики типа

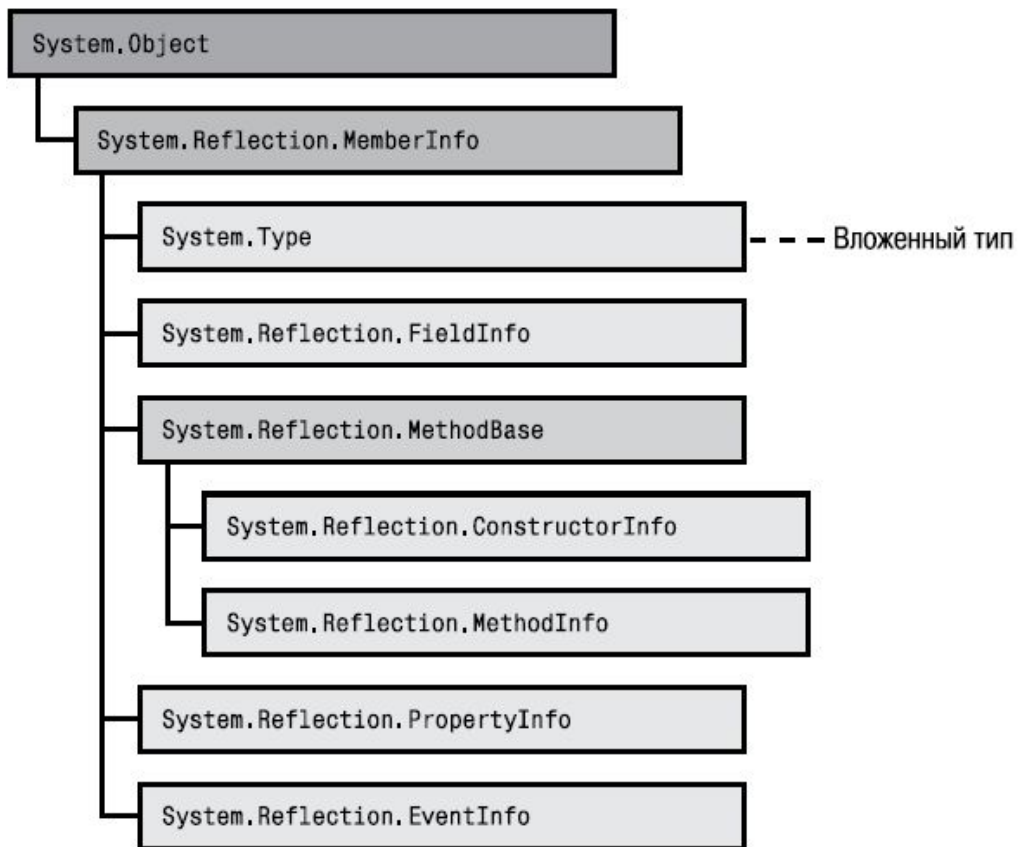


- IsAbstract
- IsArray
- IsClass
- IsCOMObject
- IsEnum
- IsInterface
- IsPrimitive
- IsNestedPrivate
- IsNestedPublic
- IsSealed
- IsValueType

Получение информации о членах типа

- `GetConstructors()`
- `GetEvents()`
- `GetFields()`
- `GetInterfaces()`
- `GetMembers()`
- `GetMethods()`
- `GetNestedTypes()`
- `GetProperties()`

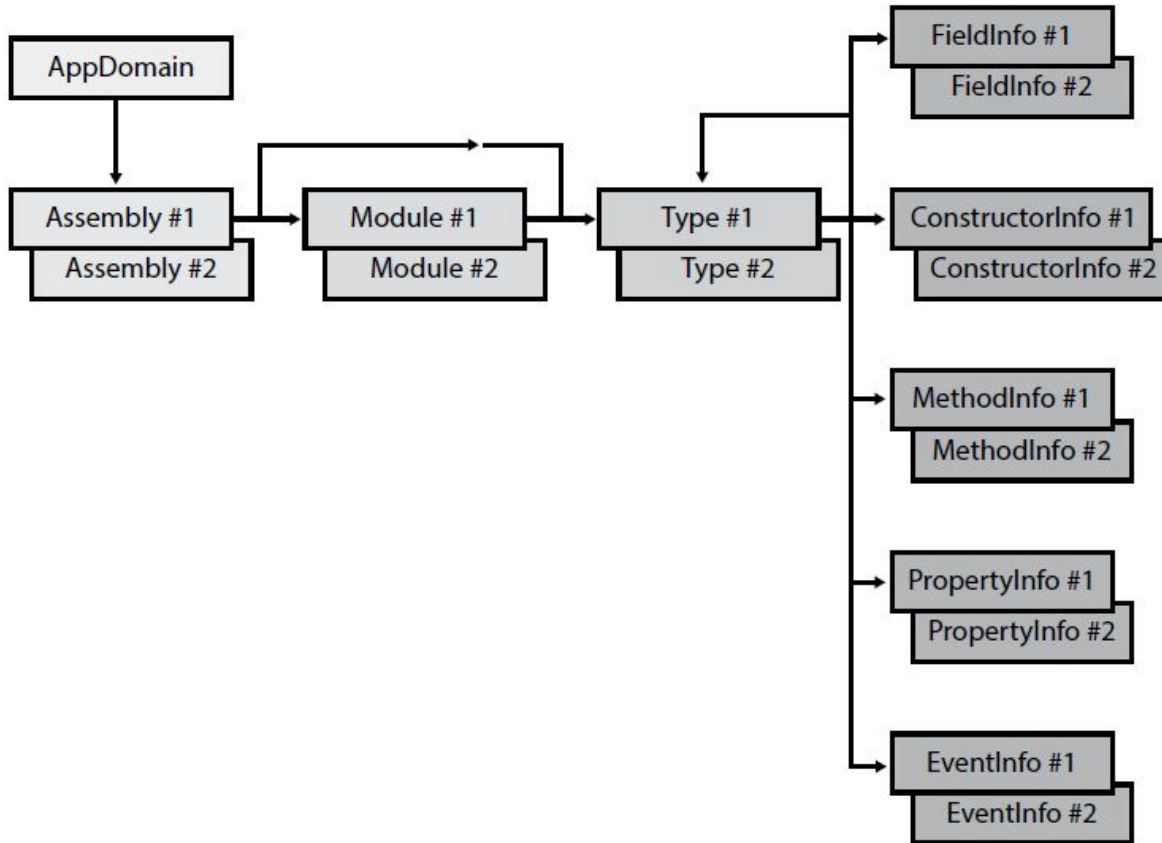
Иерархия MemberInfo



Пример получения информации о типе

```
string trace = "";
ConstructorInfo[] arrCI = type.GetConstructors();
foreach (ConstructorInfo ci in arrCI)
{
    trace += (ci.IsStatic ? "static " : "")
        + (ci.IsPrivate ? "private " : "")
        + (ci.IsFamily ? "protected " : "")
        + (ci.IsAssembly ? "internal " : "")
        + ci.Name;
    ParameterInfo[] arrParamInfo = ci.GetParameters();
    trace += "(";
    for (int i = 0; i != arrParamInfo.Length; i++)
    {
        ParameterInfo parInf = arrParamInfo[i];
        trace += (i != 0 ? ", " : "")
            + (parInf.IsIn ? "in " : "")
            + (parInf.IsOut ? "out " : "")
            + (parInf.IsOptional ? "optional " : "")
            + parInf.ParameterType.Name + " "
            + parInf.Name
            + ((parInf.DefaultValue != DBNull.Value)
                ? (" = " + parInf.DefaultValue) : "");
    }
    trace += ");\r\n";
}
```

Связи типов Отражения



Фильтрация возвращаемых членов типа

- BindingFlags.Default
- BindingFlags.IgnoreCase
- BindingFlags.DeclaredOnly
- BindingFlags.Instance
- BindingFlags.Static
- BindingFlags.Public
- BindingFlags.NonPublic
- BindingFlags.FlattenHierarchy

```
MethodInfo[] methods = type.GetMethods(BindingFlags.Instance | BindingFlags.NonPublic);  
foreach (MethodInfo method in methods)  
{  
    Console.WriteLine(method.Name);  
}
```

Создание экземпляра типа



1. Через методы типа

System.Activator

```
using System;
using System.Reflection;

namespace ReflectionTestConsoleApplication
{
    class Class1
    {
        public string someField = "Some test field";

        static void Main()
        {
            string className = "ReflectionTestConsoleApplication.Class1";
            Type type = Type.GetType(className);
            Object data = Activator.CreateInstance(type);
            Console.WriteLine((data as Class1).someField);
            Console.ReadLine();
        }
    }
}
```

Создание экземпляра типа



2. Через методы **System.AppDomain**

```
class Class1
{
    public string someField = "Some test field";

    static void Main()
    {
        string className = "ReflectionTestConsoleApplication.Class1";
        Type type = Type.GetType(className);
        ObjectHandle data =
AppDomain.CurrentDomain.CreateInstance("ReflectionTestConsoleApplication", type.FullName);
        Console.WriteLine((data.Unwrap() as Class1).someField);
        Console.ReadLine();
    }
}
```

Создание экземпляра типа



3. Через метод `InvokeMember` объекта `System.Type`

```
class Class1
{
    public string someField = "Some test field";

    static void Main()
    {
        string className = "ReflectionTestConsoleApplication.Class1";
        Type type = Type.GetType(className);
        Object data = type.InvokeMember(null, BindingFlags.CreateInstance, null, null,
null);
        Console.WriteLine((data as Class1).someField);
        Console.ReadLine();
    }
}
```

Создание экземпляра типа



4. Через метод `Invoke` объекта

`System.Reflection.ConstructorInfo`

```
class Class1
{
    public string someField = "Some test field";

    static void Main()
    {
        string className = "ReflectionTestConsoleApplication.Class1";
        Type type = Type.GetType(className);
        ConstructorInfo constructor = type.GetConstructor(new Type[0]);
        Object data = constructor.Invoke(null);
        Console.WriteLine((data as Class1).someField);
        Console.ReadLine();
    }
}
```


Метод InvokeMember



```
public abstract class Type : MemberInfo, ...
{
    public Object InvokeMember
        (
            String name,                // Имя члена
            BindingFlags invokeAttr,    // Способ поиска членов
            Binder binder,              // Способ сопоставления членов и аргументов
            Object target,              // Объект, на котором нужно вызвать член
            Object[] args,              // Аргументы, которые нужно передать методу
            CultureInfo culture          // Региональные стандарты, которые используются
при связывании
        );
    ...
}
```

Пример использования метода InvokeMember

```
class Class1
{
    public string someField = "Some test field";

    public override string ToString()
    {
        return someField;
    }

    static void Main()
    {
        string className = "ReflectionTestConsoleApplication.Class1";
        Type type = Type.GetType(className);
        ConstructorInfo constructor = type.GetConstructor(new Type[0]);
        Object data = constructor.Invoke(null);


        String s = (String)type.InvokeMember("ToString",
            BindingFlags.DeclaredOnly |
            BindingFlags.Public | BindingFlags.NonPublic |
            BindingFlags.Instance | BindingFlags.InvokeMethod, null, data, null);
        Console.WriteLine("ToString: " + s);
        Console.ReadLine();
    }
}
```

An aerial photograph of a large brick fortress wall on a hillside. The wall is made of red brick with a crenellated top. In the foreground, there is a green lawn and some trees. In the middle ground, a prominent round tower with a conical roof stands on a small hill. To the right, a long, multi-story red brick building with a dark roof is visible. In the background, a wide river flows through a city, with industrial buildings and a bridge visible in the distance. The sky is overcast.

Вопросы?

Список литературы



 Джеффри Рихтер. CLR via C# (3е издание)

 RSDN Magazine. Метаданные в среде .Net.
<https://rdsn.ru/article/dotnet/refl.xml>

Задание для работы в аудитории



Написать программу, которая выводит на экран иерархию всех типов, производных от **Exception** (или любого другого базового часто используемого типа). Сделать возможность задания пользователем сборки для поиска (или нескольких).

Задание на дом



Написать простое динамически расширяемое приложение. Сделать отдельную библиотеку, в которой определить интерфейсы, которые будут использовать разработчики модулей приложения. В главном хосте приложения сделать динамический поиск и загрузку таких модулей из заданной папки. К примеру, в хосте сделать выпадающее меню, которое будет заполняться информацией из загруженных модулей, и при нажатии на пункты которого будут вызываться методы из этих модулей (через определённые до этого интерфейсы).