

Reflection.Emit

и все-все-все

Практика использования

MSK
DOT
NET

Андрей Курош
<http://imp.ms>
[@impworks](#)

1. Примеры решаемых задач
2. Генерация кода: язык MSIL
3. Создание структур
4. Ограничения
5. Альтернативы
6. Демо
7. Выводы

Задача 1: конвертация из DB в DTO

02

```
public class UserProfile
{
    [Key]
    public long Id;

    public string Name;
    public string Phone;
    public string Email;

    public DateTime? LastEdit;
    public bool IsSuspended;

    // ...
}
```



```
public class UserDTO
{
    public long Id;

    public string Name;
    public string Phone;
    public string Email;
}
```

Задача 1: наивное решение

03

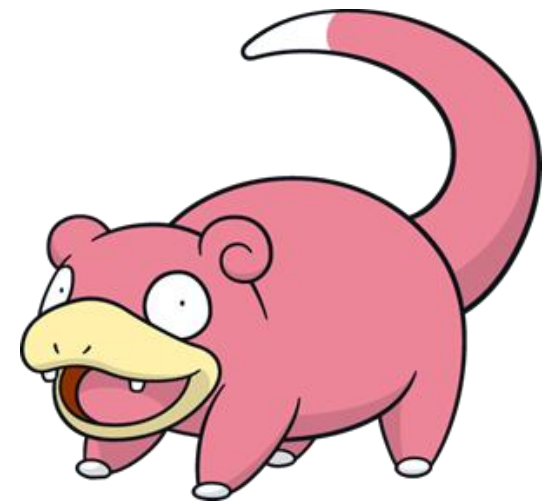
```
return new UserDTO  
{  
    Id = obj.Id,  
    Name = obj.Name,  
    Phone = obj.Phone,  
    Email = obj.Email,  
  
    // и еще много строк  
};
```



Задача 1: решение с Reflection

04

```
var dto = new UserDTO();  
var props = dto.GetType().GetFields();  
var objType = typeof(UserProfile);  
  
foreach(var prop in props)  
{  
    var objProp = dbType.GetField(prop.Name);  
    prop.SetValue(dto, objProp.GetValue(obj));  
}  
  
return dto;
```



Задача 2: DI-контейнер

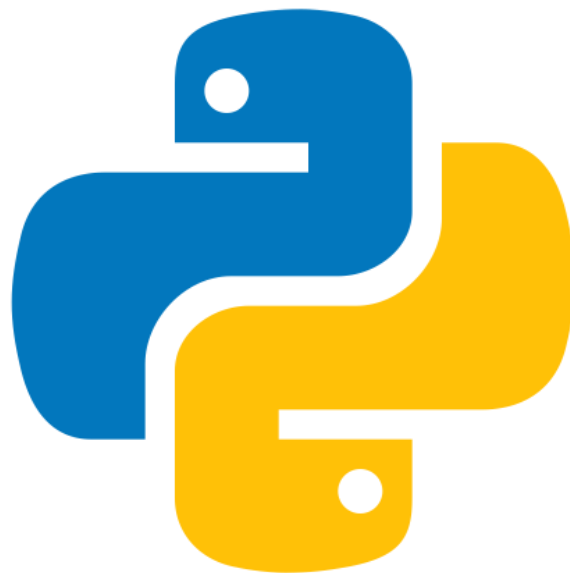
05

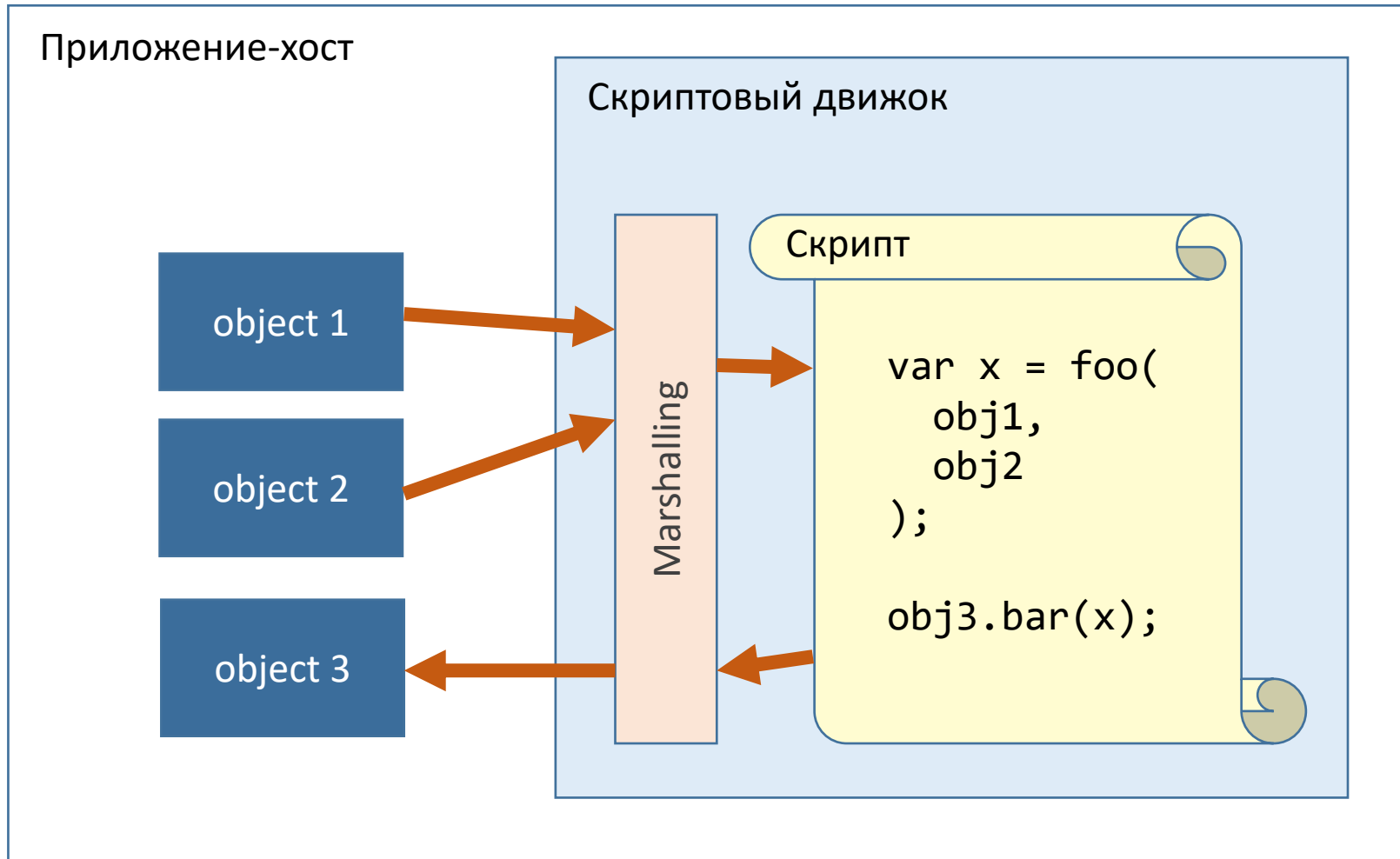
```
// регистрация
container.Register<IA, A>();

// получение
var a = container.GetInstance<IA>();

class A
{
    public Foo(B b, C c)
    {
        // ...
    }
}
```

```
return new A(
    container.GetInstance<B>(),
    container.GetInstance<C>(),
)
```





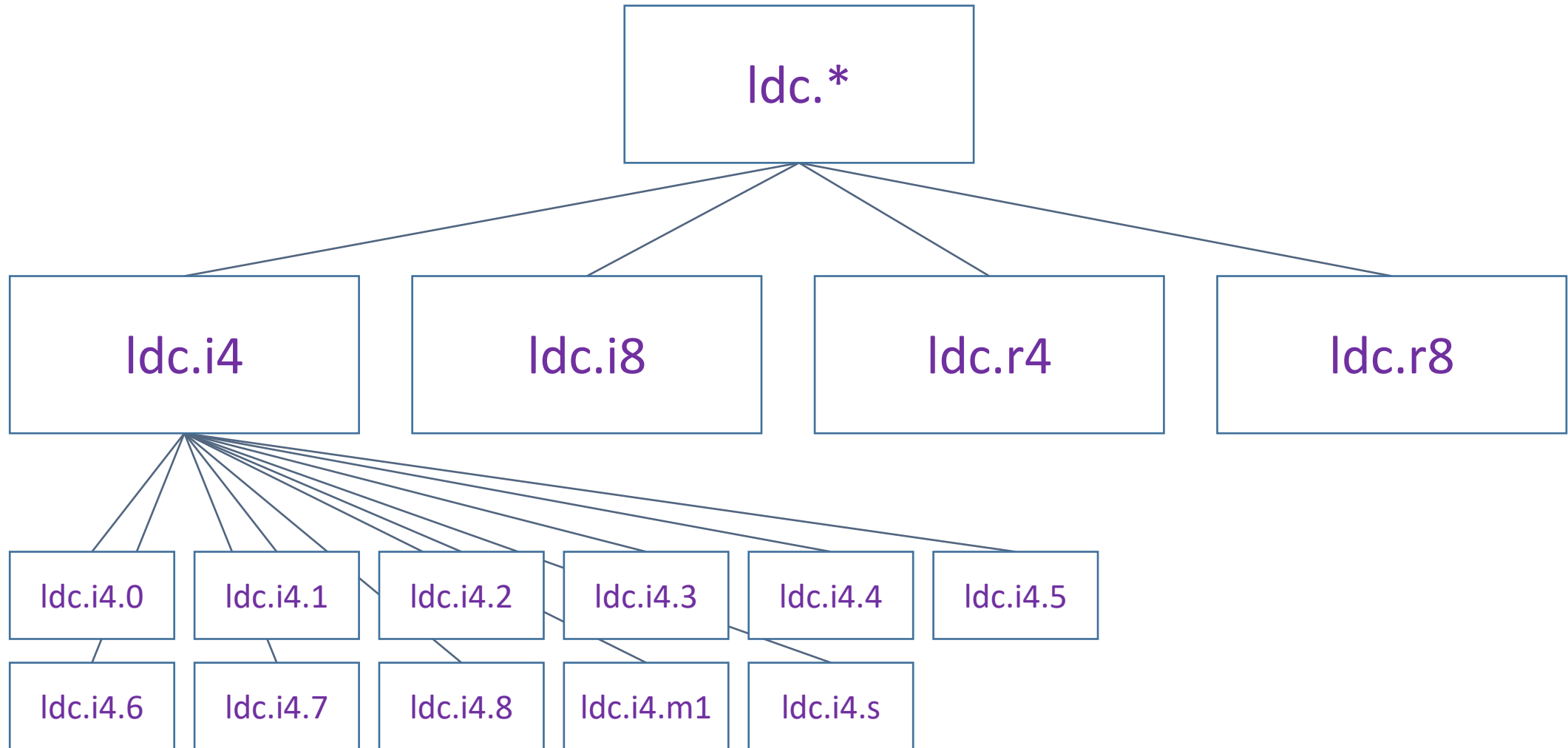

```
var write = typeof(Console).GetMethod(
    "WriteLine",
    new [] { typeof(string) }
);

var dm = new DynamicMethod("Test", typeof(void), new Type[0]);
var gen = dm.GetILGenerator();

gen.Emit(OpCodes.Ldstr, "Hello world");
gen.Emit(OpCodes.Call, write);
gen.Emit(OpCodes.Ret);

var action = (Action) dm.CreateDelegate(typeof(Action));
action();
```

Константа	<code>ldc, ldstr, ldnull, ldtoken</code>
Локальная переменная	<code>ldloc</code>
Аргумент метода	<code>ldarg</code>
Поле объекта или класса	<code>ldfld, ldsfld</code>
Элемента массива	<code>ldelem</code>



Математика

- `add`
- `sub`
- `mul`
- `div`
- `rem` // остаток от деления
- `neg` // инверсия знака

Логика / биты

- `and`
- `or`
- `xor`
- `not`
- `shl` // сдвиг влево
- `shr` // сдвиг вправо

Сравнения

- `ceq` // равенство
- `clt` // меньше
- `cgt` // больше

На стек загружается `int32` (1 или 0)

Модификации

- `*.ovf` // с проверкой переполнения
- `*.un` // для беззнаковых чисел

Статический вызов:

```
class A
{
    public static void Test(int a, int b)
    { }
}
```

```
A.Test(1, 2);
```

```
IL_0001: ldc.i4.1      // 1
IL_0002: ldc.i4.2      // 2
IL_0003: call A.Test   // A.Test(1, 2)
```

Динамический вызов:

```
class A
{
    public void Test(int a, int b)
    { }
}
```

```
new A().Test(1, 2);
```

```
IL_0001: newobj A..ctor // new A()
IL_0008: ldc.i4.1        // 1
IL_0009: ldc.i4.2        // 2
IL_000A: callvirt A.Test // a.Test(1, 2)
```

Переходы

- br
- brtrue, brfalse
- beq, bne
- bgt, bge, blt, ble
- switch <list>
- ret

Модификации


- *.s // аргумент типа <int8>
- *.un // для беззнаковых чисел

Исключения

- throw
- rethrow
- finally, endfilter

```
public string Test()
{
    if (1 > 2)
        return "foo";

    return "bar";
}
```



```
IL_0000: ldc.i4.1
IL_0001: ldc.i4.2
IL_0002: blt.s IL_000e
IL_0004: ldstr "foo"
IL_0009: br IL_0013
IL_000e: ldstr "bar"
IL_0013: ret
```

Ref.Emit: Hello world (reprise)

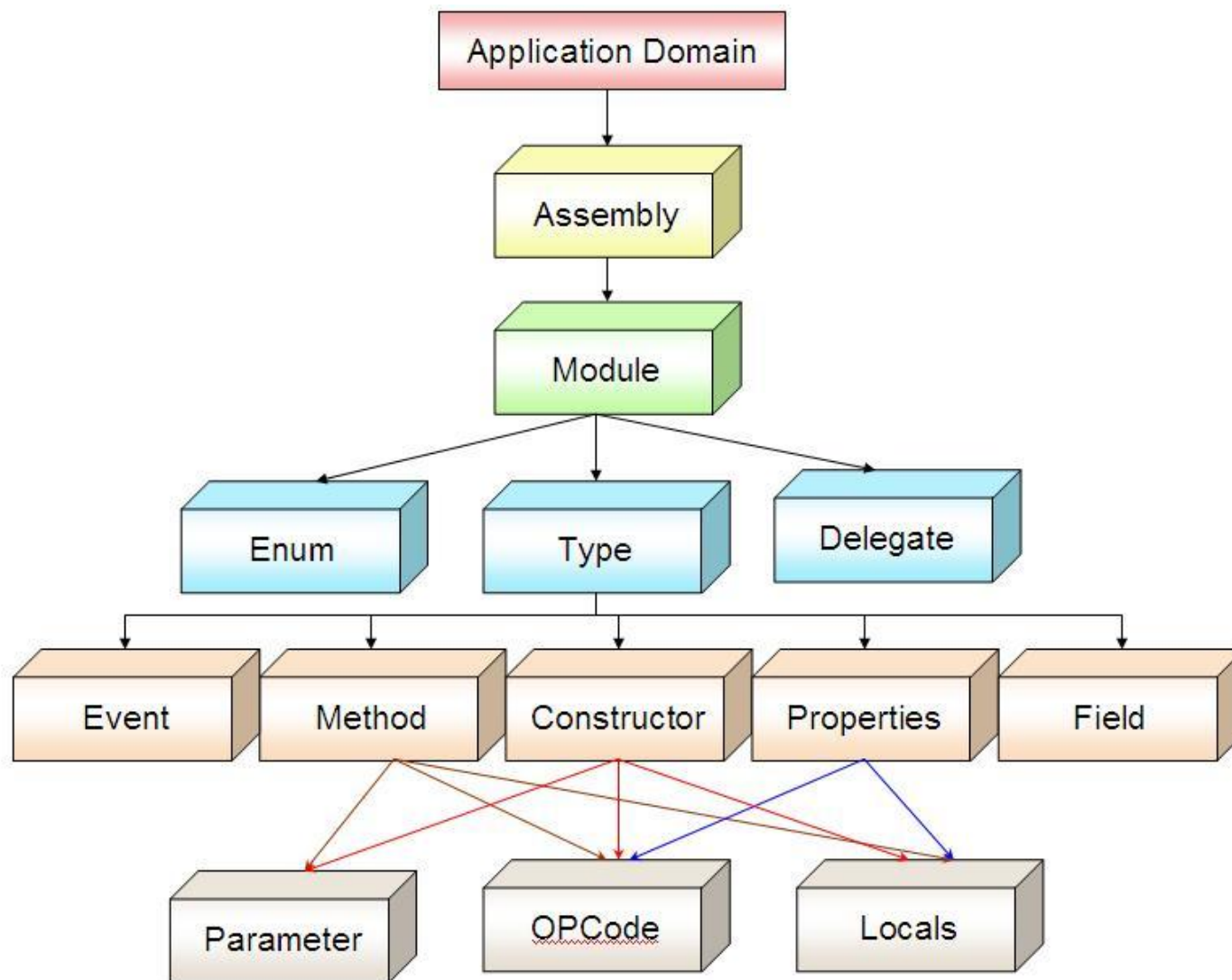
14

```
// получаем используемый метод из Reflection
var write = typeof(Console).GetMethod(
    "WriteLine", new [] { typeof(string) }
);

// создаем метод с сигнатурой: void Test()
var dm = new DynamicMethod("Test", typeof(void), new Type[0]);
var gen = dm.GetILGenerator();

gen.Emit(OpCodes.Ldstr, "Hello world"); // загрузка строки
gen.Emit(OpCodes.Call, write);           // вызов WriteLine
gen.Emit(OpCodes.Ret);                   // точка выхода

var action = (Action) dm.CreateDelegate(typeof(Action));
action();
```



Дано:

```
interface IGreeter
{
    string Greet(string name);
}
```

Нужно сгенерировать:

```
class Greeter: IGreeter
{
    public string Greet(string name)
    {
        return "Hello, " + name;
    }
}
```

Генерация типа: решение (часть 1)

17

```
var asm = AppDomain.CurrentDomain.DefineDynamicAssembly(           // сборка
    new AssemblyName("TestAssembly"),
    AssemblyBuilderAccess.Run
);

var mod = asm.DefineDynamicModule("TestModule");                   // модуль
var type = mod.DefineType("Greeter", TypeAttributes.Public);      // тип

var met = type.DefineMethod(                                       // метод
    "Greet",
    MethodAttributes.Public | MethodAttributes.Virtual,
    typeof(string),
    new [] { typeof(string) }
);

met.DefineParameter(0, ParameterAttributes.None, "name");        // аргумент
```

Генерация типа: решение (часть 2)

18

```
var concat = typeof(string).GetMethod(           // получение string.Concat
    "Concat",
    new [] { typeof(string), typeof(string) }
);

var ig = met.GetILGenerator();
ig.Emit(OpCodes.Ldstr, "Hello, ");              // префикс на стеке
ig.Emit(OpCodes.Ldarg_1);                        // аргумент на стеке
ig.Emit(OpCodes.Call, concat);                  // вызов метода
ig.Emit(OpCodes.Ret);                           // точка выхода

type.AddInterfaceImplementation(typeof(IGreeter)); // интерфейс

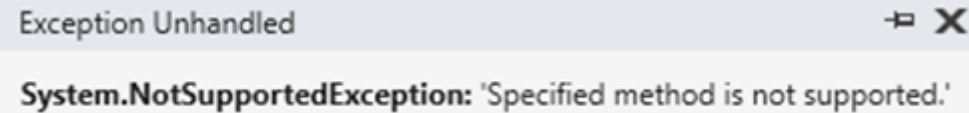
var finalType = type.CreateType();              // создание типа

dynamic obj = Activator.CreateInstance(finalType);
string result = obj.Greet("Bill Gates");        // вызов метода
```

```
class A  
{  
}
```

```
var list = new List<A>();
```

```
var listType = typeof(List<>);           // List<T>  
var myListType = listType.MakeGenericType(aType); // конкретизация в List<A>  
var ctor = myListType.GetConstructor(new Type[0]); // получение конструктора
```



Exception Unhandled

System.NotSupportedException: 'Specified method is not supported.'

Ограничения: генерики (решение)

20

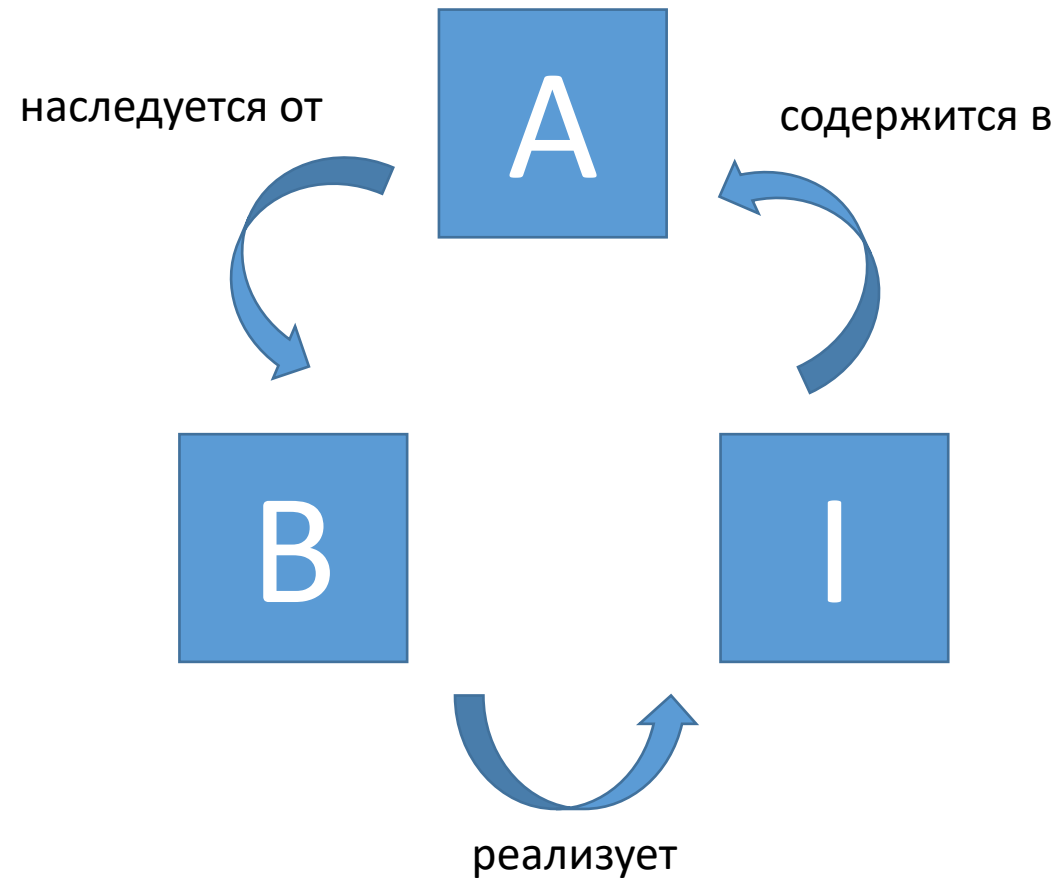
```
class A  
{  
}
```

```
var list = new List<A>();
```

```
var listType = typeof(List<>);           // List<T>  
var ctor = listType.GetConstructor(new Type[0]); // List<T>.ctor()  
  
var myListCtor = TypeBuilder.GetConstructor( // волшебный метод  
    listType.MakeGenericType(type),  
    ctor  
);
```

```
class A: B
{
    interface I
    {
    }
}

class B: A.I
{
}
```





- Expression Trees

- Объектная модель для представления выражений
- Встроен в .NET 3.0+
- Утверждения доступны с .NET 4.0+
- *Частичная* поддержка компилятора C#
- Основан на Reflection.Emit

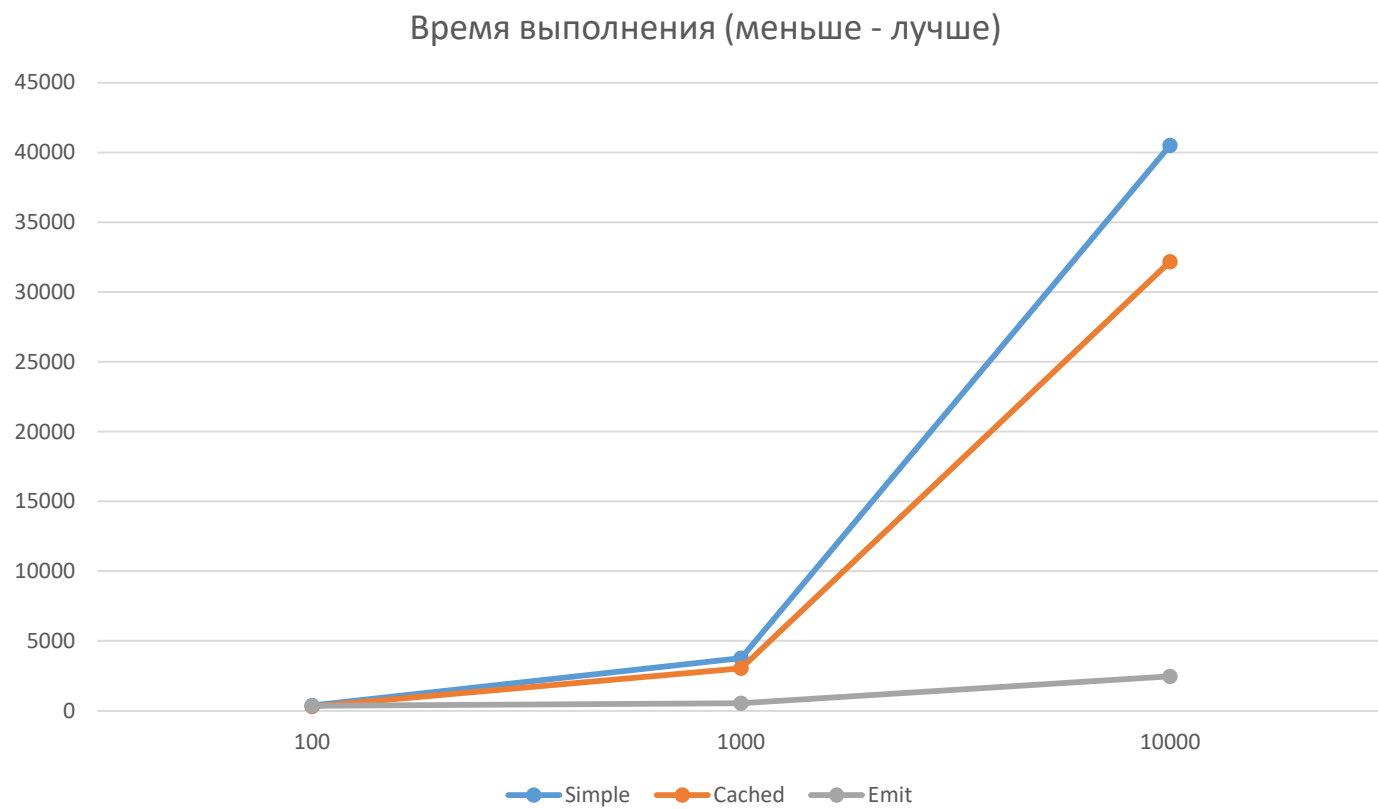
- Mono.Cecil

- Альтернативная реализация без привязки к рантайму
- Позволяет дизассемблировать существующие сборки
- Распространяется через NuGet
- Открытый исходный код на Github:
<https://github.com/jbevain/cecil>

- IKVM.Reflection.Emit



Method	Runs	Mean
Simple	100	375.8 us
SimpleCache	100	301.1 us
Emitter	100	353.0 us
Simple	1000	3,769.6 us
SimpleCache	1000	3,045.3 us
Emitter	1000	540.3 us
Simple	10000	40,501.3 us
SimpleCache	10000	32,167.4 us
Emitter	10000	2,460.2 us





- Ref.Emit – база динамической кодогенерации
- Подходит для:
 - Ускорения работы с Reflection
 - Генерации кода на основе данных
 - Экспериментов с платформой
- Не подходит для:
 - Мобильных приложений
 - Написания полнофункциональных компиляторов
 - Анализа и модификации существующих сборок

Полезные ссылки:

- <http://lingpad.net>
Утилита для быстрого запуска кода на C#/F#, показывает MSIL
- <http://ilspy.net>
Бесплатный декомпилятор .NET-сборок
- <https://github.com/impworks/emit-benchmark>
Исходный код бенчмарка
- <https://github.com/impworks/lens>
Тот самый встраиваемый язык

Каналы Telegram:

-  @CILChat
-  @CompilerDev

Спасибо за внимание!



Андрей Курош
<http://imp.ms>
[@impworks](https://twitter.com/impworks)