



POSITIVE TECHNOLOGIES

...важно сконцентрировать имеющиеся ресурсы
на основных прорывных направлениях...

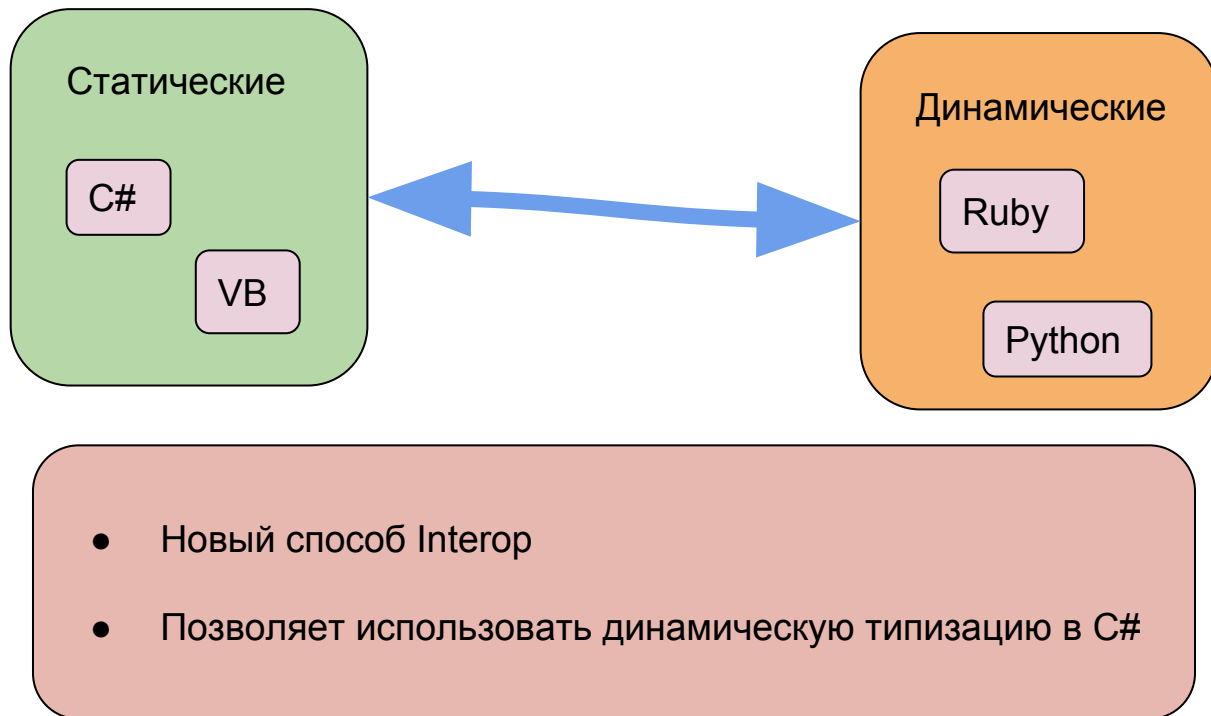
В.В. Путин

Как это работает: DLR

dynamic тип в C# или “прорывное” направление в .NET

Игорь Яковлев
iyakovlev@ptsecurity.com

Зачем нужен DLR?



Свойства типа `dynamic` в C#

`dynamic` объект - принимает значения любого типа

тип выражения с `dynamic` - `dynamic`*

`dynamic` - ссылочный "тип"

`dynamic` - объект может участвовать в любом* выражении C#

код для выражений с объектом `dynamic` создается динамически*

типизация `dynamic` объектов: динамическая

Однако

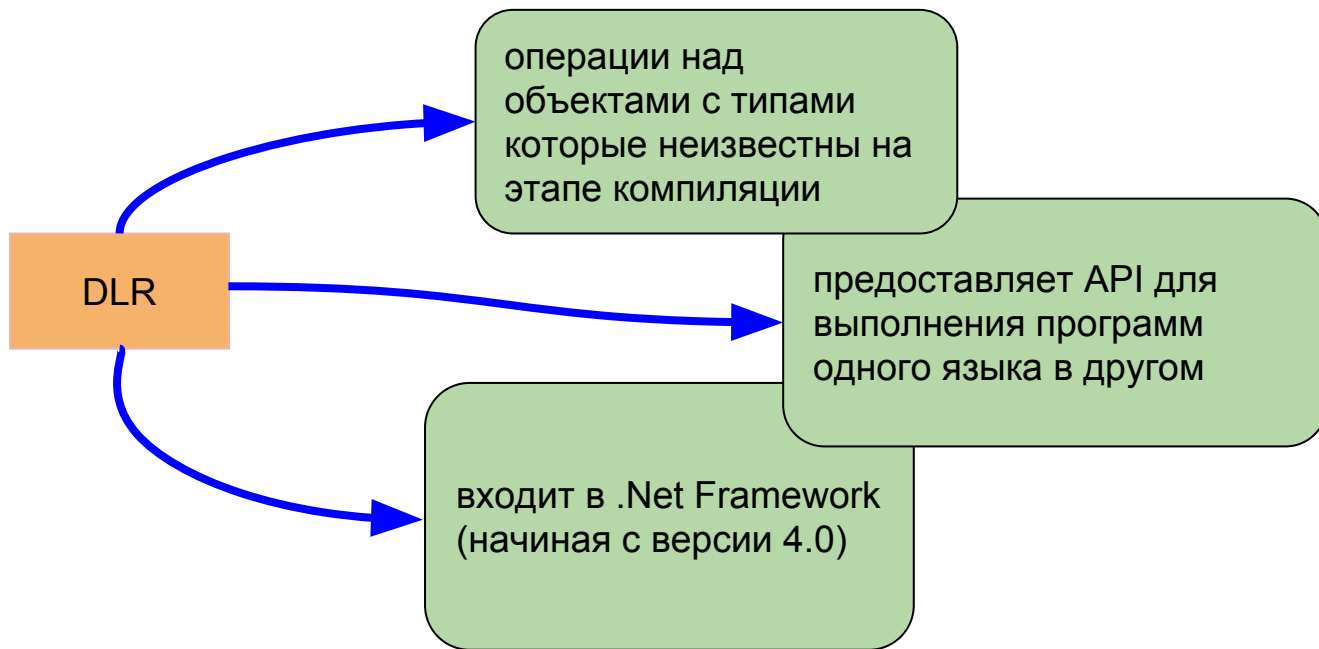
Вследствие того, что `dynamic` является не “родным” для C#, можно всегда немного потроллить компилятор:

```
new dynamic(); //compile time error
```

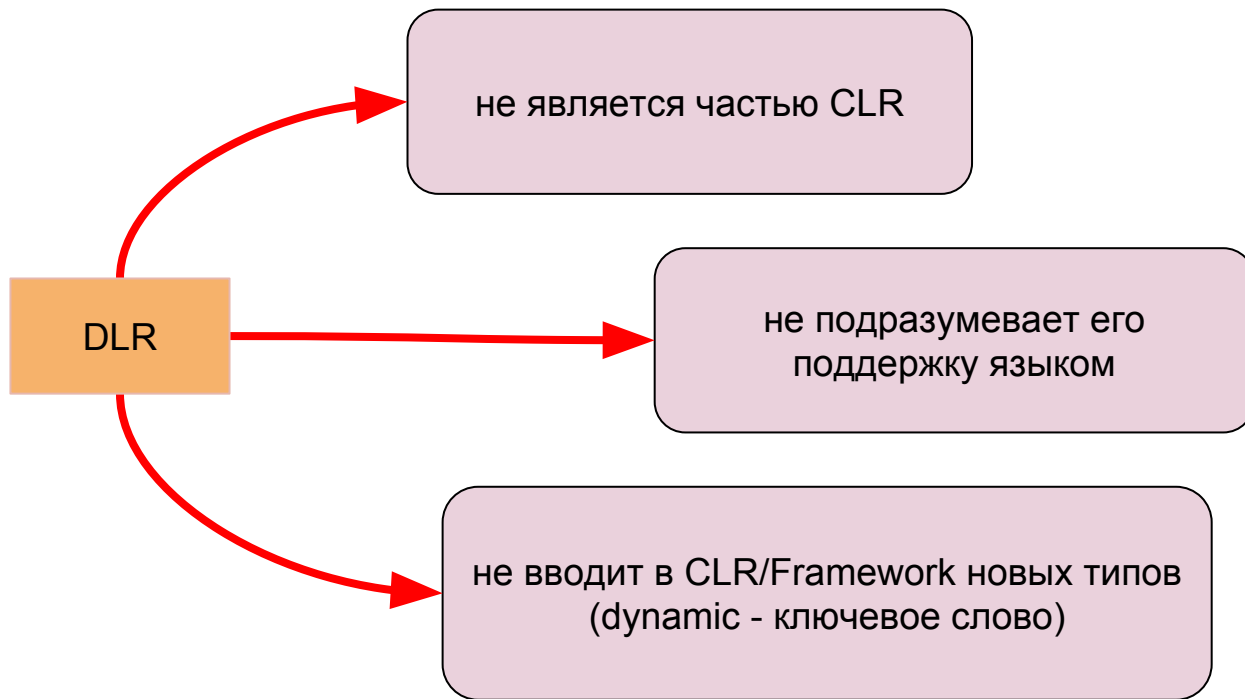
```
T Foo<T>() where T : new()  
{  
    return new T();  
}  
...  
var variable = Foo<dynamic>(); //будет new object()
```



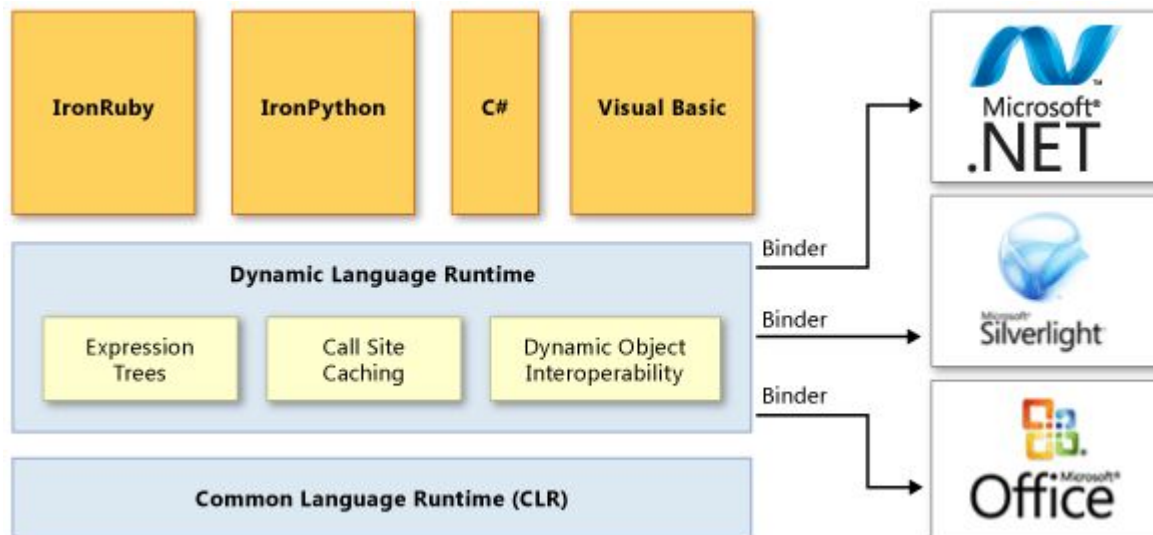
DLR - dynamic language runtime



¬DLR

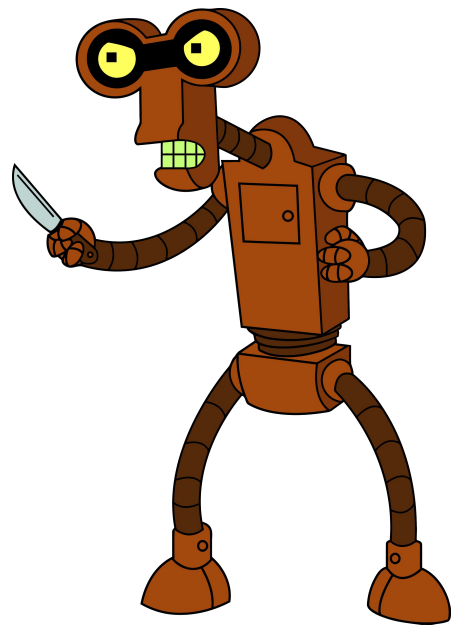


Картинка скачанная из интернета:



Пример работы с DLR из C#

```
namespace DLR46
{
    using IronPython.Hosting;
    public class CSharpType
    {
        public string HelloWorld => "Hello from CSharp";
    }
    class Program
    {
        static void Main(string[] args)
        {
            var engine = Python.CreateEngine();
            dynamic pythonObject = engine.Execute(@"
                class PythonType:
                    def Foo(self, dynamicObject):
                        print ""Python is saying hello: "" + dynamicObject.HelloWorld
                PythonType()
            ");
            var cSharpObject = new CSharpType();
            pythonObject.Foo(cSharpObject);
        }
    }
}
```



Пример работы с DLR из C#

```
namespace DLR46
{
    using IronPython.Hosting;
    public class CSharpType
    {
        public string HelloWorld => "Hello from CSharp";
    }
}
```

```
class Program
```

```
{
    static void Main(string[] args)
    {
```

```
        var engine = IronPython.Hosting.Python.CreateDefaultEngine();
```

```
        dynamic py = engine.CreateScriptedScope().GetVariable("__repl__");
```

```
        class CSharpType
```

```
        {
```

```
            public string HelloWorld => "Hello from CSharp";
```

```
        }
    }
}
```

```
        PythonEngine pyEngine = Python.CreateDefaultEngine();
```

```
        var cSharpType = new CSharpType();
```

```
        py.SetVariable("cSharpType", cSharpType);
```

```
        pythonObject.Foo(cSharpObject);
    }
}
```

```
public class CSharpType
```


```
{
```

```
    public string HelloWorld => "Hello from CSharp";
```

```
}
```

Пример работы с DLR из C#

```
namespace DLR46
{
    using IronPython.Hosting;
    public class CSharpType
    {
        public string HelloWorld { get; }
    }
    class Program
    {
        static void Main(string[] args)
        {
            var engine = Python.CreateEngine();
            dynamic pythonObject = engine.Execute(@"
                class PythonType:
                def Foo(self, dynamicObject):
                    print ""Python is saying hello: "" + dynamicObject.HelloWorld
                PythonType()
                "");
            var cSharpObject = new CSharpType();
            pythonObject.Foo(cSharpObject);
        }
    }
}
```



Пример работы с DLR из C#

namespace DLR46

```
{  
    using IronPython.Hosting;  
    public class CSharpType  
    {  
        public string HelloWorld => "Hello from CSharp";  
    }  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            var engine = Python.CreateEngine();  
            dynamic pythonObject = engine.Execute(@"  
                class PythonType:  
                    def Foo(self, dynamicObject):  
                        print ""Python is saying hello: "" + dynamicObject.HelloWorld  
                PythonType()  
            ");  
            var cSharpObject = new CSharpType();  
            pythonObject.Foo(cSharpObject);  
        }  
    }  
}
```

Запускаем, получаем:

Python is saying hello: Hello from CSharp



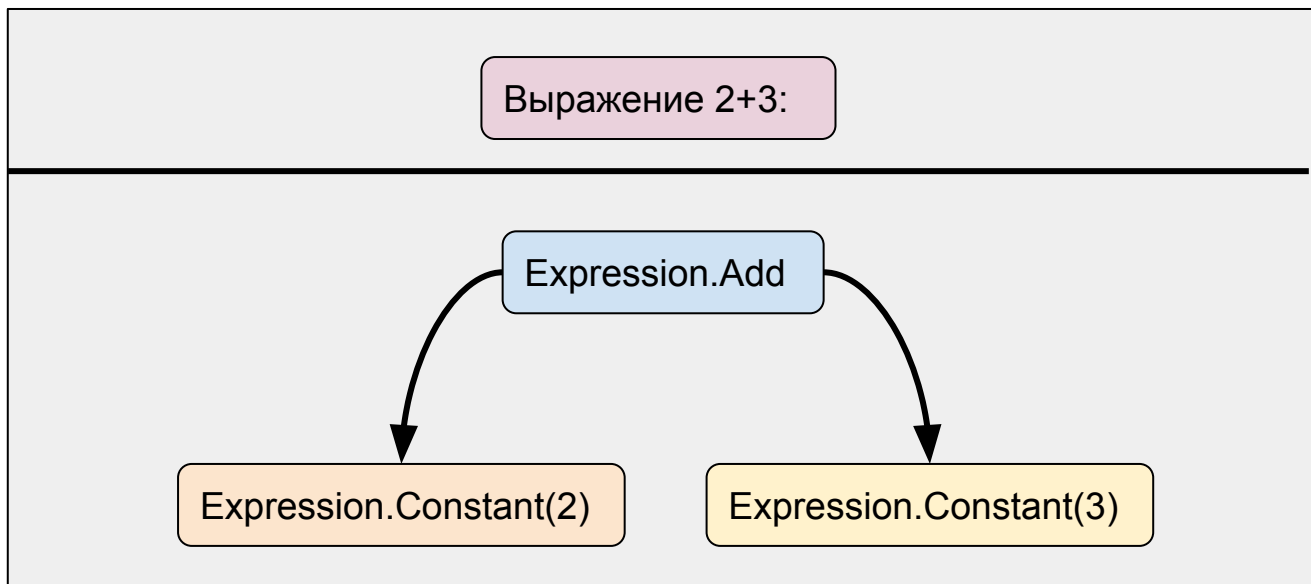
Linq Expression Tree

Как написать программу в Runtime?



LINQ Expression Trees

Деревья выражений представляют код в виде деревьев, где каждая вершина - выражение.



Пример:

```
using System;
using Expression = System.Linq.Expressions.Expression;

namespace DLR46
{
    class Program
    {
        static int Add2To3()
        {
            return 2 + 3;
        }

        static int ExpressionedAdd2To3()
        {
            Expression add2To3 = Expression.Add(Expression.Constant(2), Expression.Constant(3));

            Func<int> compiledExpression = Expression.Lambda<Func<int>>(add2To3).Compile();

            return compiledExpression();
        }

        static void Main(string[] args)
        {
            Console.WriteLine(Add2To3());
            Console.WriteLine(ExpressionedAdd2To3());
        }
    }
}
```



Статически скомпилированная функция:

```
using System;  
using Expression = System.Linq.Expressions.Expression;
```

```
namespace DLR46
```

```
{
```

```
    class Program
```

```
    {
```

```
        static int Add2To3()
```

```
        {
```

```
            return 2 + 3;
```

```
        }
```

```
        static int ExpressionedAdd2To3()
```

```
        {
```

```
            Expression add2To3 = Expression.Constant(2 + 3);
```

```
            Func<int> compiledExpr = Expression.Lambda<Func<int>>(add2To3, null);
```

```
            return compiledExpr.Invoke();
```

```
        }
```

```
        return compiledExpr.Invoke();
```

```
    }
```

```
    static void Main(string[] args)
```

```
    {
```

```
        Console.WriteLine(Add2To3());
```

```
        Console.WriteLine(ExpressionedAdd2To3());
```

```
    }
```

```
}
```

```
static int Add2To3()
```

```
{
```

```
    return 2 + 3;
```

```
}
```



Функция собранная из Expression Tree:

```
using System;
using Expression = System.Linq.Expressions.Expression;

namespace DLR46
{
    class Program
    {
        static int Add2To3()
        {
            Expression add2To3 = Expression.Add(Expression.Constant(2), Expression.Constant(3));
            Func<int> compiledExpression = Expression.Lambda<Func<int>>(add2To3).Compile();
            return compiledExpression();
        }

        static int ExpressionedAdd2To3()
        {
            Expression add2To3 = Expression.Add(Expression.Constant(2), Expression.Constant(3));
            Func<int> compiledExpression = Expression.Lambda<Func<int>>(add2To3).Compile();
            return compiledExpression();
        }

        static void Main(string[] args)
        {
            Console.WriteLine(Add2To3());
            Console.WriteLine(ExpressionedAdd2To3());
        }
    }
}
```



Результат:

```
using System;  
using Expression = System.Linq.Expressions.Expression;
```

```
namespace DLR46
```

```
{
```

```
    class Program
```

```
    {
```

```
        static int A
```

```
        {
```

```
            return
```

```
        }
```

```
        static int B
```

```
        {
```

```
            Expressi
```

```
        }
```

```
            Func<int> compiledExpression = Expression.Lambda<Func<int>>(add2To3).Compile();
```

```
            return compiledExpression();
```

```
        }
```

```
    }
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
static void Main(string[] args)
```

```
{
```

```
    Console.WriteLine(StaticExample());
```

```
    Console.WriteLine(DynamicExample());
```

```
}
```

```
static void Main(string[] args)
```

```
{
```

```
    Console.WriteLine(Add2To3());
```

```
    Console.WriteLine(ExpressionedAdd2To3());
```

```
}
```

```
C:\Win
```

```
5
```

```
5
```

```
Для продолжения продолжения продолжения . . .
```

Как работает DLR?

Что происходит когда ты пишешь `dynamic` в C#

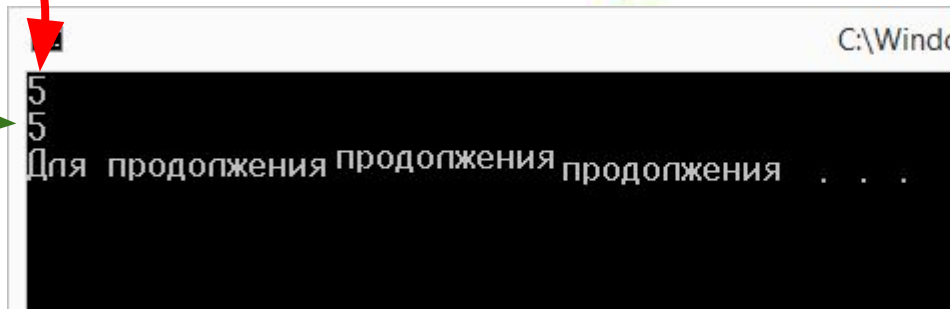


Вооружаемся дизассемблером :)

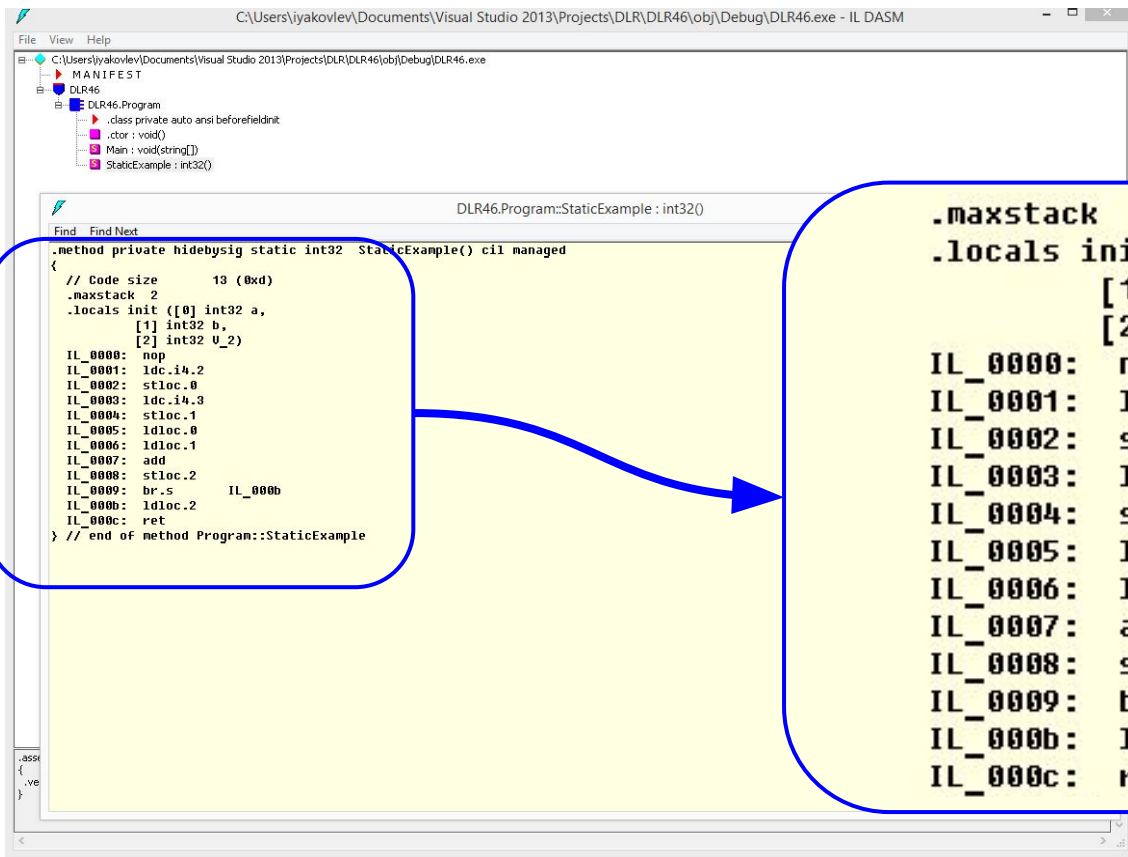
Сравним код для статического типа и динамического

```
using System;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine(StaticExample());
            Console.WriteLine(DynamicExample());
        }
        static int StaticExample()
        {
            int a = 2;
            int b = 3;
            return a + b;
        }
        static dynamic DynamicExample()
        {
            dynamic a = 2;
            dynamic b = 3;
            return a + b;
        }
    }
}
```

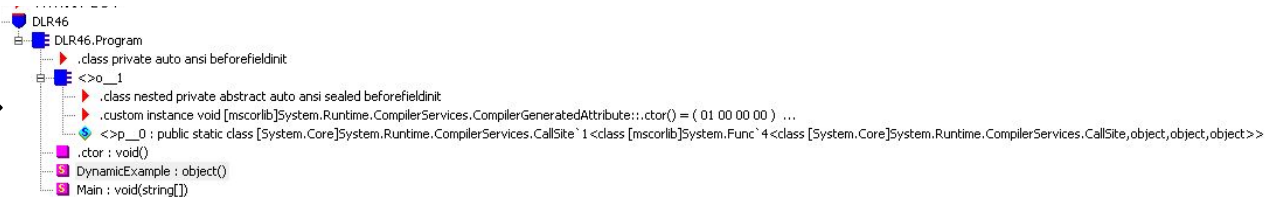


Ну в первом примере все просто



```
.maxstack 2
.locals init ([0] int32 a,
             [1] int32 b,
             [2] int32 v_2)

IL_0000:  nop
IL_0001:  ldc.i4.2
IL_0002:  stloc.0
IL_0003:  ldc.i4.3
IL_0004:  stloc.1
IL_0005:  ldloc.0
IL_0006:  ldloc.1
IL_0007:  add
IL_0008:  stloc.2
IL_0009:  br.s      IL_000b
IL_000a:  ldloc.2
IL_000c:  ret
```



DLR46.Program::DynamicExample : object()

Find	Find Next
IL_0018:	ldc.i4.0
IL_0019:	ldc.i4.0
IL_001a:	ldtoken DLR46.Program
IL_001f:	call class [mscorlib]System.Type [mscorlib]System.Type::GetTypeFromHandle(valuetype [mscorlib]System.RuntimeTypeHandle)
IL_0024:	ldc.i4.2
IL_0025:	newarr [Microsoft.CSharp]Microsoft.CSharp.RuntimeBinder.CSharpArgumentInfo
IL_002a:	dup
IL_002b:	ldc.i4.0
IL_002c:	ldc.i4.0
IL_002d:	ldnull
IL_002e:	call class [Microsoft.CSharp]Microsoft.CSharp.RuntimeBinder.CSharpArgumentInfo [Microsoft.CSharp]Microsoft.CSharp.RuntimeBinder.CShar
IL_0033:	stelem.ref
IL_0034:	dup
IL_0035:	ldc.i4.1
IL_0036:	ldc.i4.0
IL_0037:	ldnull
IL_0038:	call class [Microsoft.CSharp]Microsoft.CSharp.RuntimeBinder.CSharpArgumentInfo [Microsoft.CSharp]Microsoft.CSharp.RuntimeBinder.CShar
IL_003d:	stelem.ref
IL_003e:	call class [System.Core]System.Runtime.CompilerServices.CallSiteBinder [Microsoft.CSharp]Microsoft.CSharp.RuntimeBinder.Binder::Binar
IL_0043:	call class [System.Core]System.Runtime.CompilerServices.CallSite`1<0> class [System.Core]System.Runtime.CompilerServices.CallSite`1<
IL_0048:	stsFld class [System.Core]System.Runtime.CompilerServices.CallSite`1<class [mscorlib]System.Func`4<class [System.Core]System.Runtime.Co
IL_004d:	ldsFld class [System.Core]System.Runtime.CompilerServices.CallSite`1<class [mscorlib]System.Func`4<class [System.Core]System.Runtime.Co
IL_0052:	ldFld ?0 class [System.Core]System.Runtime.CompilerServices.CallSite`1<class [mscorlib]System.Func`4<class [System.Core]System.Runtime
IL_0057:	ldsFld class [System.Core]System.Runtime.CompilerServices.CallSite`1<class [mscorlib]System.Func`4<class [System.Core]System.Runtime.Co
IL_005c:	ldloc.0
IL_005d:	ldloc.1
IL_005e:	callvirt instance ?3 class [mscorlib]System.Func`4<class [System.Core]System.Runtime.CompilerServices.CallSite,object,object,object>::Inv

Перепишем код для $a+b$ на C#:

```
if (SiteContainer.Site1 == null)
{
    SiteContainer.Site1 =
        CallSite<Func<CallSite, object, object, object>>.Create(
            CSharpBinaryOperationBinderFactory.Create(
                ExpressionType.Add, false, false,
                new CSharpArgumentInfo[] {
                    new CSharpArgumentInfo(0, null),
                    new CSharpArgumentInfo(0, null)
                }
            )
        );
}
object c = SiteContainer.Site1.Target(SiteContainer.Site1, a, b);
```



Перепишем код для $a+b$ на C#:



System.Runtime.CompilerServices.CallSite<T>

Тот парень который знает что вызывать



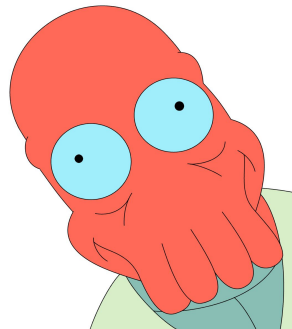
Что такое CallSite

Содержит операцию необходимую для выполнения динамического вызова

Кэширует динамические операции (часть реализации Polymorphic Inline Cache)

Создает заглушки для динамических вызовов (делегат Update)

Обращается к байндеру динамических вызовов



CallSite снаружи

```
namespace System.Runtime.CompilerServices
{
    [...]public sealed class CallSite<T> : CallSite where T : class
    {
        [...]public T Target;
        [...]public T Update { get; }
        [...]public static CallSite<T> Create(CallSiteBinder binder);
    }
}
```

T - Сигнатура операции

Делегат операции

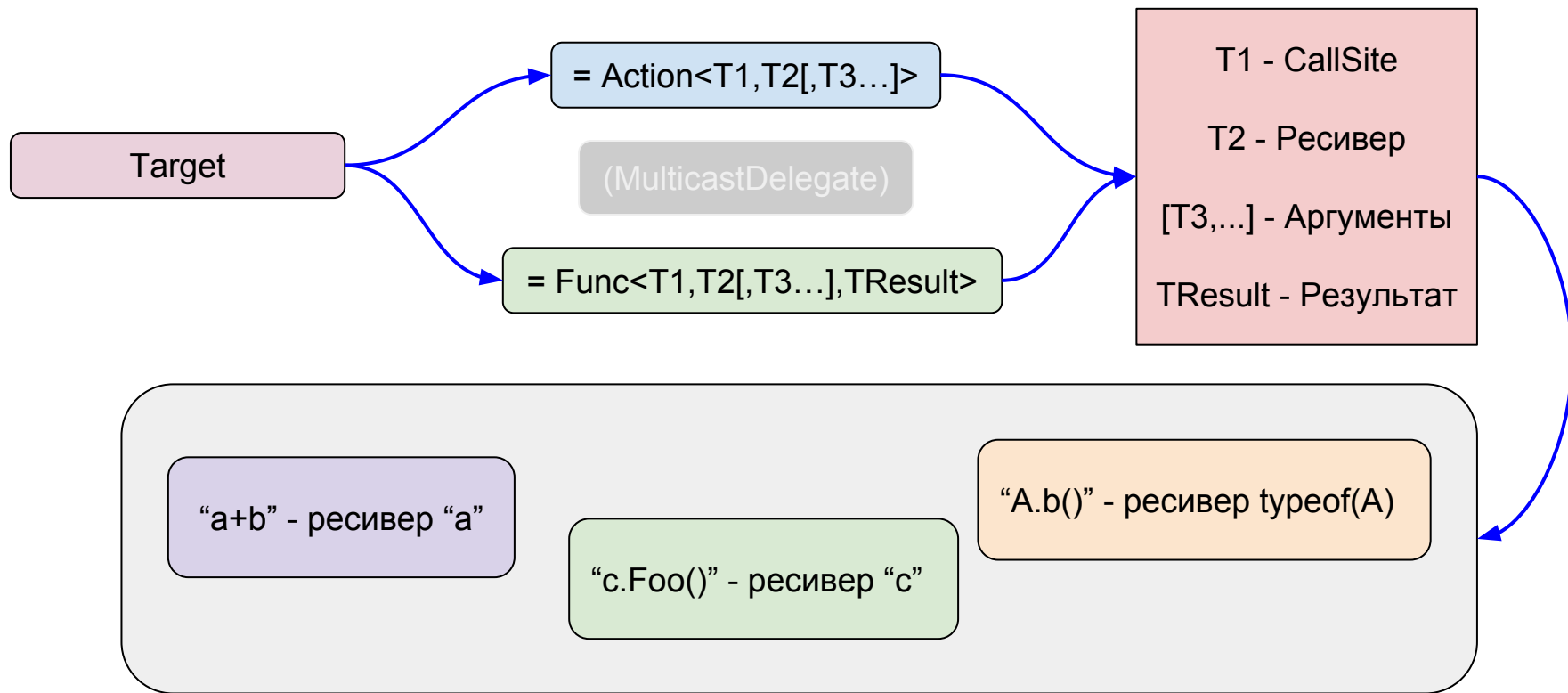
Делегат обновления

Фабрика CallSite

Что может Target:

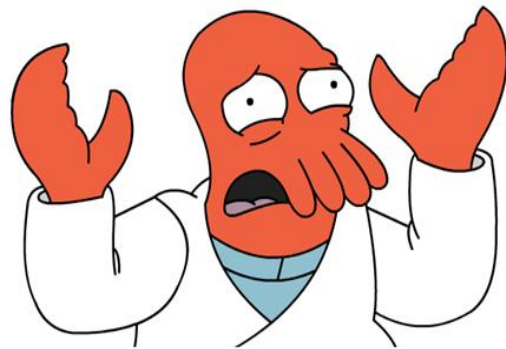


Какой тип у Target?!



Какой код внутри Target?

Пусть Target сгенерирован для операции "a+2", где a - Int32



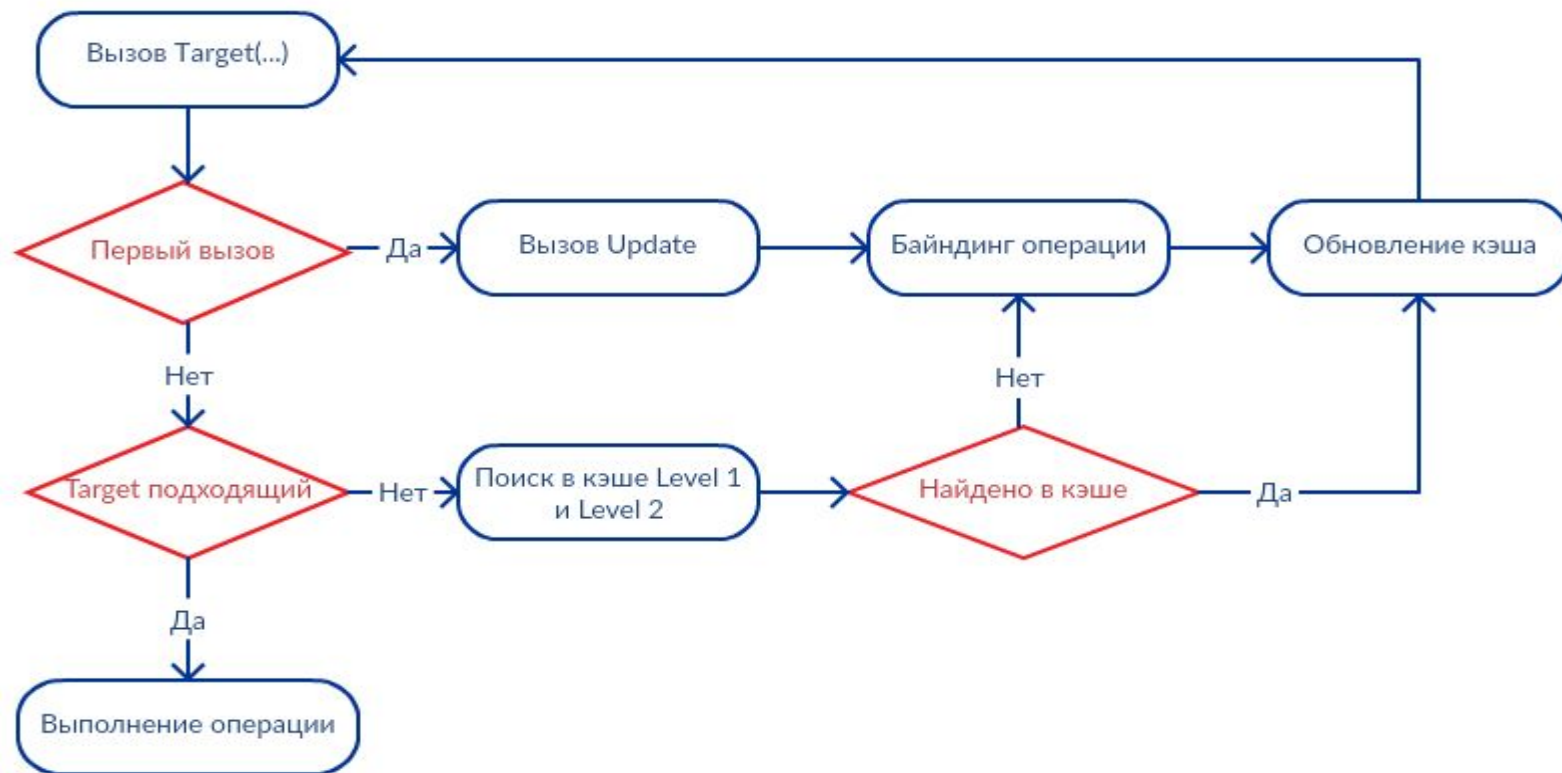
Restriction expression
Ограничение операции

```
.If ($var1 .TypeEqual System.Int32 && $var2 .TypeEqual System.Int32 && $var2 == 2)
{
    .Return { (System.Object)($var1 + 2) }
}.Else
{
    $callsite.Update()
    .Return { .Default(System.Int32) }
}
```

Body expression
Тело операции

Target - это .Net Expressions Tree v2 скомпилированный в лямбду

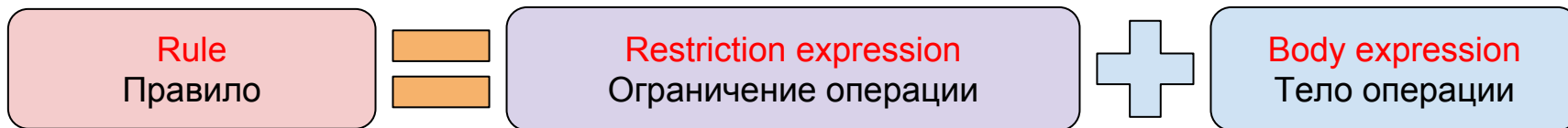
Что происходит при вызове Target(...)



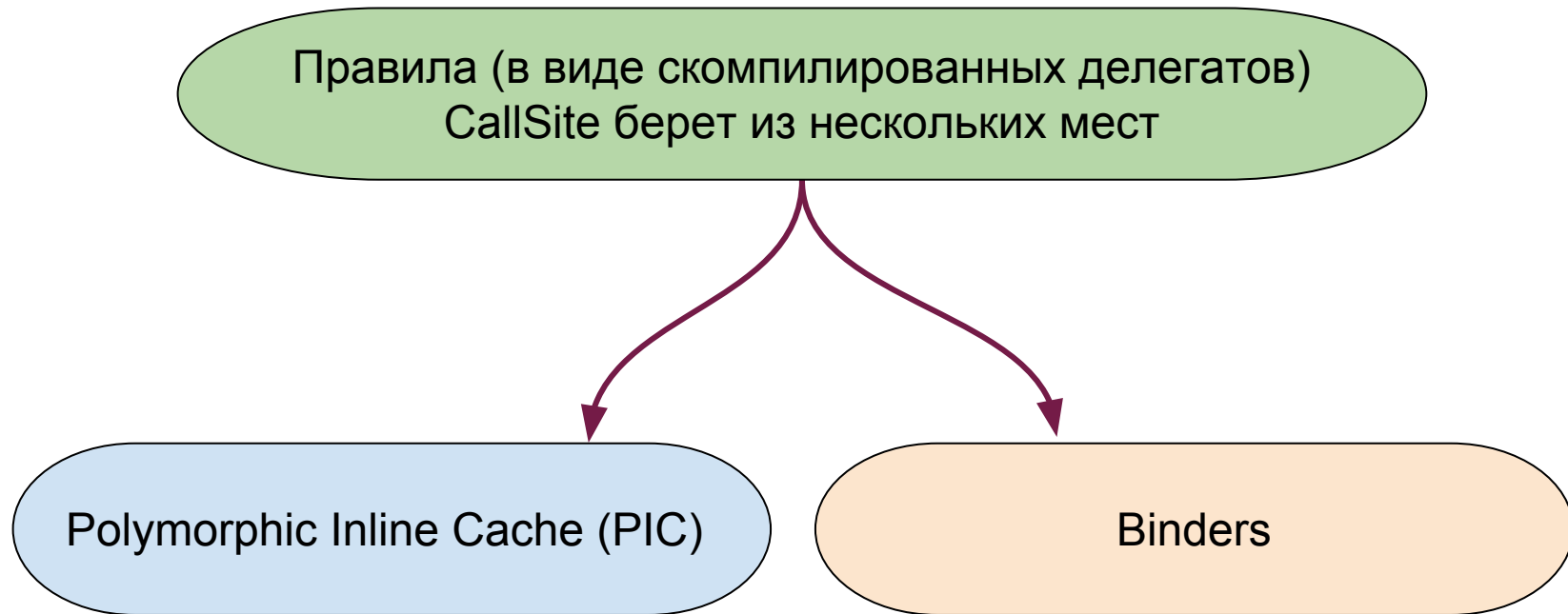
Правила (Rules)

Комбинация ограничения и тела называется правилом (Rule)

Динамический делегат - скомпилированное правило



Откуда берутся правила?!



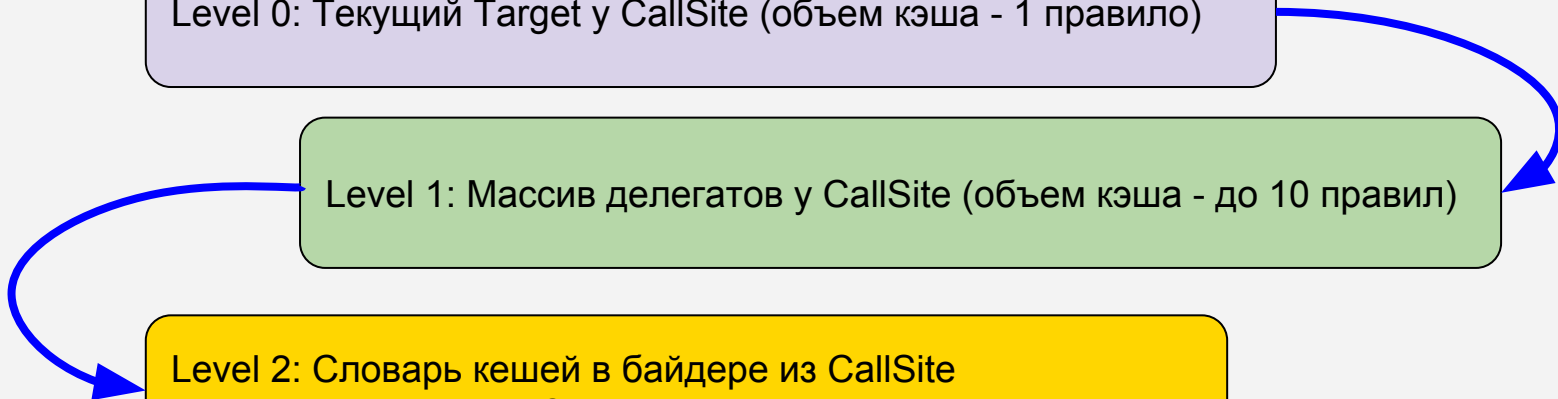
PIC - кэш правил

DLR PIC - трехуровневый кэш правил (делегатов)

Level 0: Текущий Target у CallSite (объем кэша - 1 правило)

Level 1: Массив делегатов у CallSite (объем кэша - до 10 правил)

Level 2: Словарь кешей в байдере из CallSite
(Ключ - тип правила, Значение - массив из 128 правил)

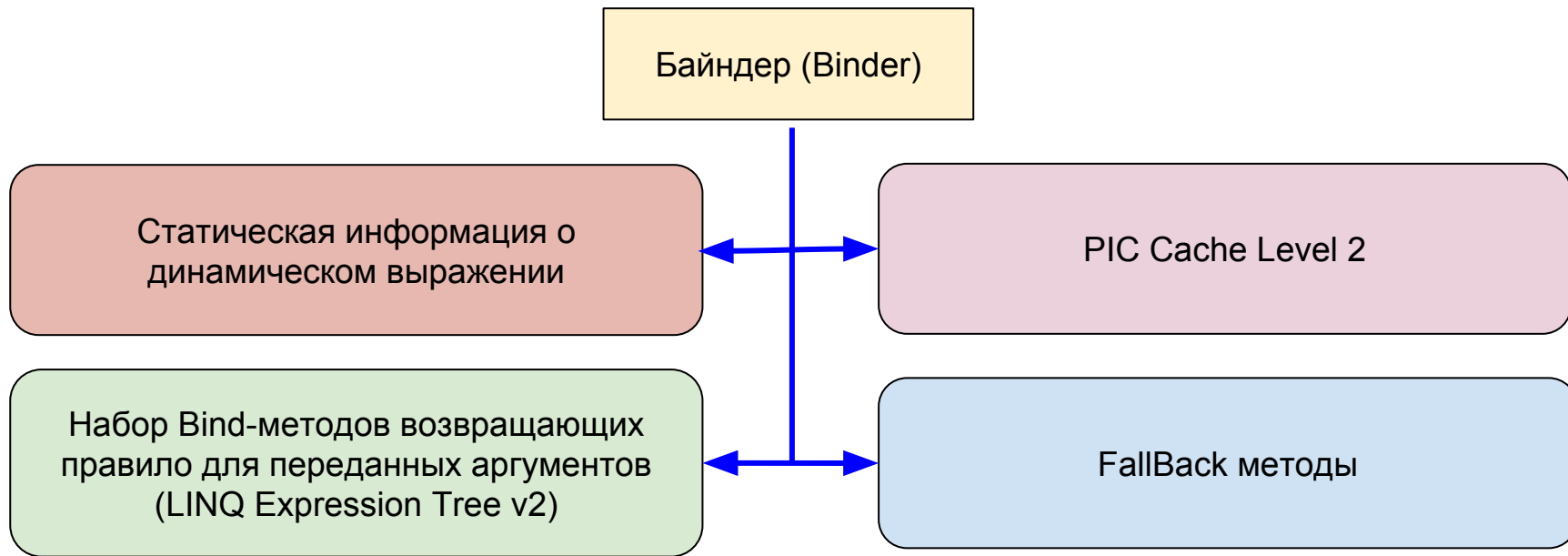


Binders

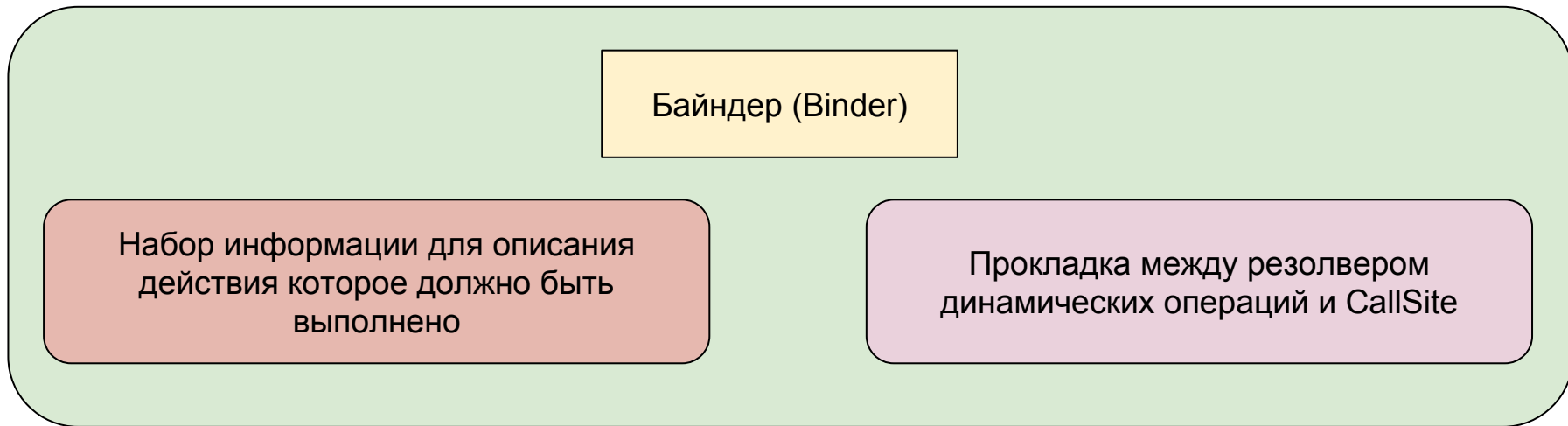
или начало нашего путешествия по DLR...



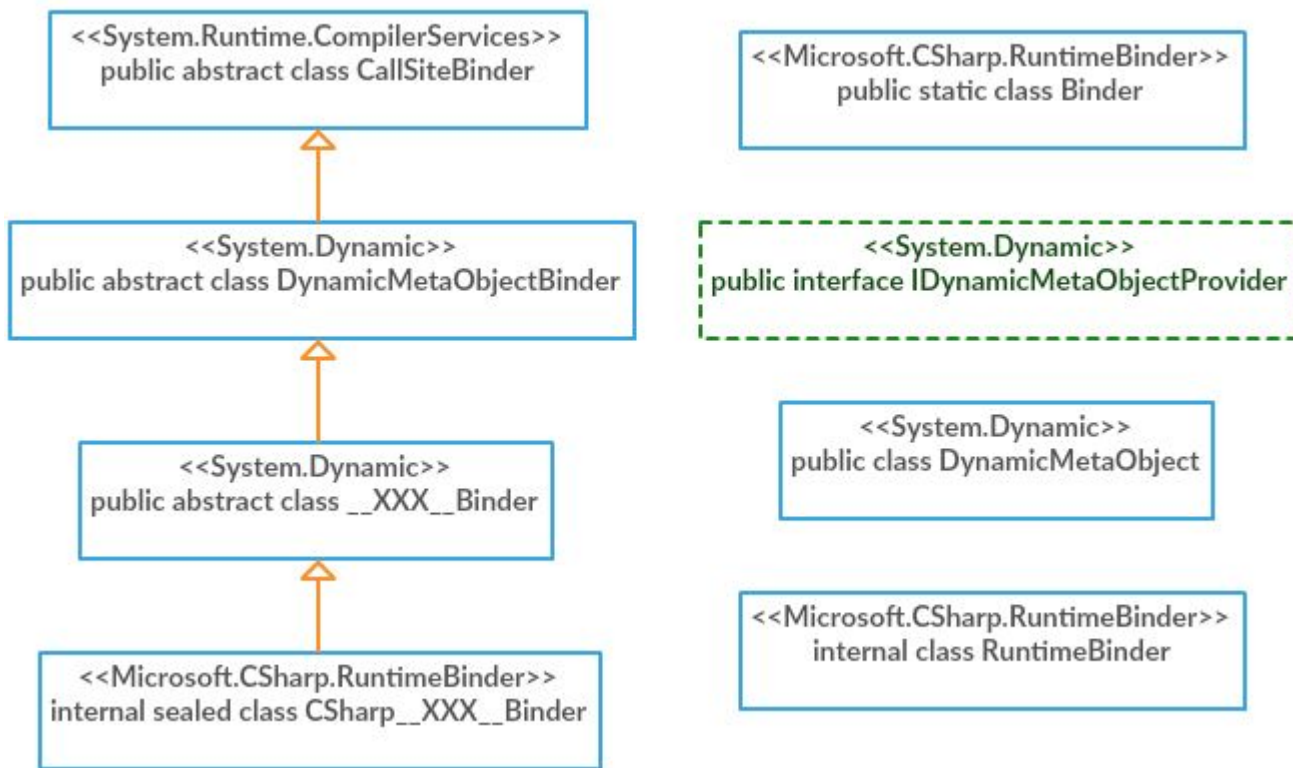
Байндеры



Высшая миссия Байндера



Иерархия инфраструктуры DLR (для C#)



abstract CallSiteBinder - базовый байндер

```
abstract Expression Bind(object[] args, + параметры для Expression Tree)
```

```
T virtual BindDelegate<T>(CallSite<T> site, object[] args)
```

```
T BindCore<T>(CallSite<T> site, object[] args)
```

1. Пытается получить готовый делегат через BindDelegate, либо
2. Получает правило в виде Expression Tree через вызов Bind
3. Компилирует правило в делегат и добавляет его в PIC Level 2

```
<<System.Runtime.CompilerServices>>  
public abstract class CallSiteBinder
```

```
<<System.Dynamic>>  
public abstract class DynamicMetaObjectBinder
```

```
<<System.Dynamic>>  
public abstract class __XXX__Binder
```

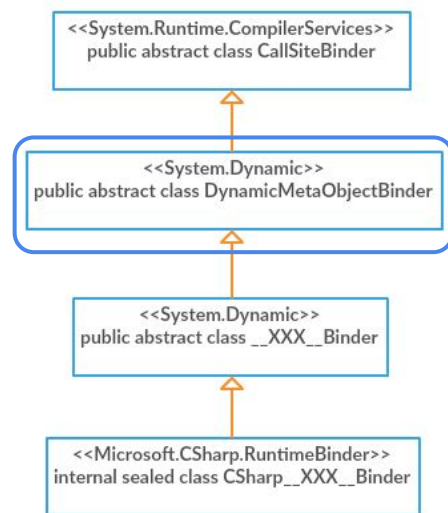
```
<<Microsoft.CSharp.RuntimeBinder>>  
internal sealed class CSharp__XXX__Binder
```

abstract DynamicMetaObjectBinder

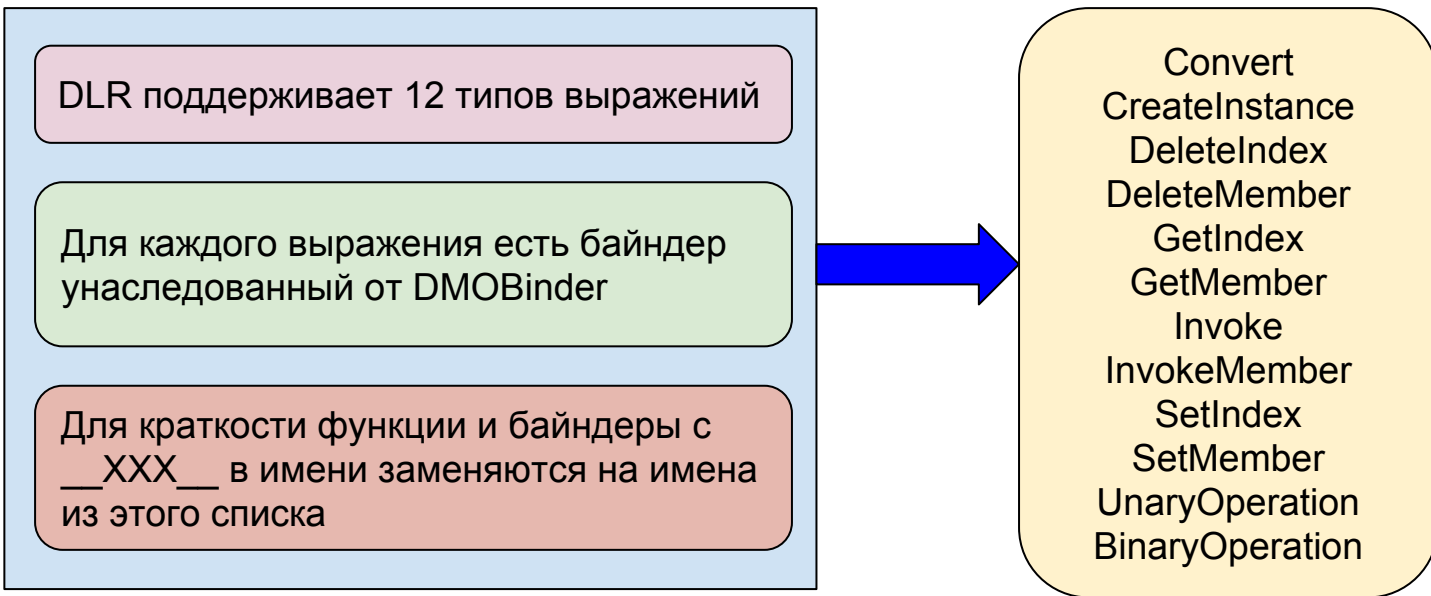
```
abstract DynamicMetaObject Bind(DynamicMetaObject reciever, DynamicMetaObject[] args)
```

```
sealed override Expression Bind(object[] args, + параметры для E.T.)
```

1. Сопоставляет аргументы args их “оберткам” типа DynamicMetaObject (DMO) через DynamicMetaObject.Create(arg)
2. Выделяет первый аргумент (типа DMO) как ресивер
3. Получает DMO с правилом через вызов Bind
4. Строит из DMO выражение .If (Restriction) .Then (.Return Body) ...



12 Разгневанных байндеров



abstract class __XXX_Binder

```
abstract DMO Fallback__XXX__(DMO target, DMO arg1, [ DMO arg2...,] DMO errorSuggestion)
```

```
DMO Fallback__XXX__(DMO reciever, DMO argDMO arg1 [,DMO arg2...])
```

Возвращает Fallback__XXX__(reciever, arg1 [,arg2], null)

```
override sealed DMO Bind(DMO reciever, DMO[] args)
```

Возвращает reciever.Bind__XXX__(this, args)

```
<<System.Runtime.CompilerServices>>  
public abstract class CallSiteBinder
```

```
<<System.Dynamic>>  
public abstract class DynamicMetaObjectBinder
```

```
<<System.Dynamic>>  
public abstract class __XXX__Binder
```

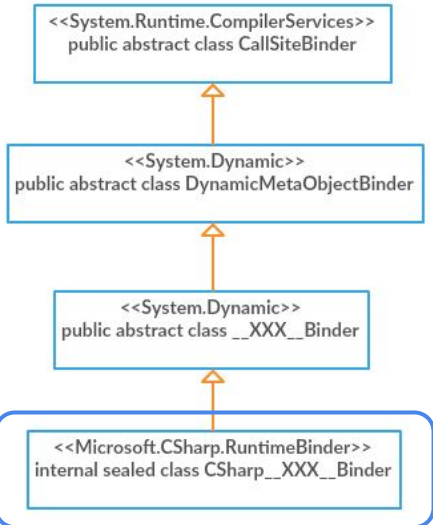
```
<<Microsoft.CSharp.RuntimeBinder>>  
internal sealed class CSharp__XXX__Binder
```

sealed CSharp_XXX_Binder

override sealed DMO Fallback__XXX__(DMO reciever, DMO arg, [DMO arg2...,] DMO errorSuggestion)

Вызывает некий RuntimeBinder :)

И это последний байндер в иерархии



И чего?! оО

Остались главные вопросы

Что это за бред?

Что такое DynamicMetaObject (DMO)?

Как из нашего object получается его DMO?

Кем составляются правила?!

Что такое FallBack?

Кому нужен IDynamicMetaObjectProvider?



DynamicMetaObject (DMO)

```
namespace System.Dynamic
```

```
{
```

```
    using [...]
```

```
    public class DynamicMetaObject
```

```
    {
```

```
        public static readonly DynamicMetaObject[];
```

```
        public Expression Expression[...]
```

```
        public bool HasValue[...]
```

```
        public Type LimitType[...]
```

```
        public BindingRestrictions Restrictions[...]
```

```
        public Type RuntimeType[...]
```

```
        public object Value[...]
```

```
        public DynamicMetaObject(Expression expression, BindingRestrictions restrictions)[...]
```

```
        public DynamicMetaObject(Expression expression, BindingRestrictions restrictions, object value)[...]
```

```
        public virtual DynamicMetaObject Bind_XXX__( __XXX__Binder binder, DynamicMetaObject arg, ...)[...]
```

```
        public static DynamicMetaObject Create(object value, Expression expression)[...]
```

```
        public virtual IEnumerable<string> GetDynamicMemberNames()[...]
```

```
    }
```

Тело правила (результат)

Ограничение правила (результат)

Обернутое значение

12 разгневанных байндер-методов

Оборачивает объект в DMO

DynamicMetaObject.Create

```
public static DynamicMetaObject Create(object value, Expression expression)
{
    ...
    IDynamicMetaObjectProvider provider = value as IDynamicMetaObjectProvider;
    if ((provider == null) && ... )
    {
        return new DynamicMetaObject(expression, BindingRestrictions.Empty, value);
    }
    DynamicMetaObject metaObject = provider.GetMetaObject(expression);
    ...
    return metaObject;
}
```

Дак тут же вроде просто?

Приводим...

Не привелось? Оборачиваем собой

Привелось? Просим вернуть

IDynamicMetaObjectProvider

```
DynamicMetaObject GetMetaObject(Expression parameter)
```

Этот скучный парень просто возвращает DynamicMetaObject


Если наш объект поддерживает этот интерфейс то для объекта можно получить DynamicMetaObject!



DynamicMetaObject::Bind__XXX__

Все 12 разгневанных байндер-методов DMO вызывают FallBack у байндера который его вызвал ;)

```
public virtual DynamicMetaObject Bind__XXX__(__XXX__Binder binder, DynamicMetaObject[] args)
{
    ContractUtils.RequiresNotNull(binder, "binder");
    return binder.Fallback__XXX__(this, args);
}
```



Ну ок, идем обратно
в байндер в метод
Fallback__XXX__



Однако они virtual...хмммм

CSharp__XXX__Binder

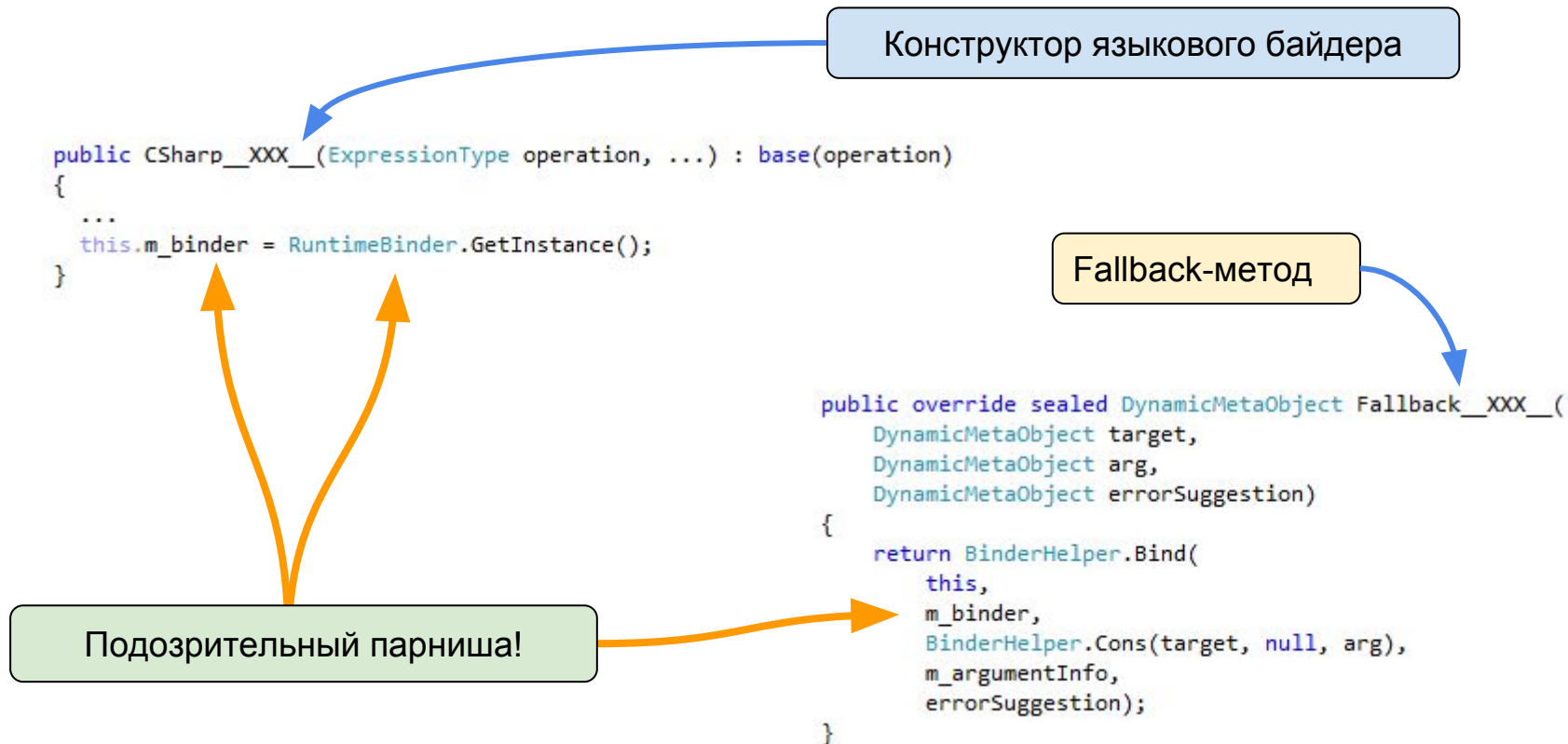
Конструктор языкового байдера

```
public CSharp__XXX__(ExpressionType operation, ...) : base(operation)
{
    ...
    this.m_binder = RuntimeBinder.GetInstance();
}
```

Fallback-метод

```
public override sealed DynamicMetaObject Fallback__XXX__(
    DynamicMetaObject target,
    DynamicMetaObject arg,
    DynamicMetaObject errorSuggestion)
{
    return BinderHelper.Bind(
        this,
        m_binder,
        BinderHelper.Cons(target, null, arg),
        m_argumentInfo,
        errorSuggestion);
}
```

Подозрительный парниша!



RuntimeBinder - МНОГО МНОГО КОДА

```
public EXPR BindStandardBinop(ExpressionKind ek, EXPR arg1, EXPR arg2)
{
    EXPRFLAG flags = (EXPRFLAG) 0;
    ExpressionBinder.BinOpArgInfo info = new ExpressionBinder.BinOpArgInfo(arg1, arg2);
    if (!this.GetBinopKindAndFlags(ek, out info.binopKind, out flags))
        return this.BadOperatorTypesError(ek, arg1, arg2);
    info.mask = (BinOpMask) (1 << (int) (info.binopKind & (BinOpKind) 31));
    List<ExpressionBinder.BinOpFullSig> list = new List<ExpressionBinder.BinOpFullSig>();
    EXPR expr = this.bindUserDefinedBinOp(ek, info);
    if (expr != null)
        return expr;
    bool flag = this.GetSpecialBinopSignatures(list, info);
    if (!flag)
        flag = this.GetStandardAndLiftedBinopSignatures(list, info);
    int index;
    if (flag)
    {
        index = list.Count - 1;
    }
    else
    {
        if (list.Count == 0)
            return (EXPR) this.bindNullEqualityComparison(ek, info);
        index = this.FindBestSignatureInList(list, info);
        if (index < 0)
            return this.ambiguousOperatorError(ek, arg1, arg2);
    }
    return this.BindStandardBinopCore(info, list[index], ek, flags);
}
```



RuntimeBinder

Парень которого нет в MSDN :)

Мы не будем приводить уважаемой публике содержание этого треша

Содержит реализацию семантического Reflection анализатора .Net

Этот парень умеет строить правила для стандартных типов .Net

Но ведь типы...эмм...не всегда стандартные!?

А если объект из Python?

```
private static DynamicMetaObject MakeBinaryOperatorResult(DynamicMetaObject[] types, DynamicMetaObjectBinder operation, PythonOpe
{
    PythonContext pythonContext = PythonContext.GetPythonContext(operation);
    ConditionalBuilder bodyBuilder = new ConditionalBuilder(operation);
    if ((op & PythonOperationKind.InPlace) != PythonOperationKind.None && !PythonProtocol.MakeOneCompareGeneric(SlotOrFunction.GetS
        return bodyBuilder.GetMetaObject(types);
    SlotOrFunction fTarget;
    SlotOrFunction rTarget;
    if (!SlotOrFunction.GetCombinedTargets(fCand, rCand, out fTarget, out rTarget) && fSlot == null && (rSlot == null && !PythonPro
        return PythonProtocol.MakeRuleForNoMatch(operation, op, errorSuggestion, types);
    if (PythonProtocol.ShouldCoerce(pythonContext, op, types[0], types[1], false) && (op != PythonOperationKind.Mod || !MetaPythonO
        PythonProtocol.DoCoerce(pythonContext, bodyBuilder, op, types, false);
    if (PythonProtocol.MakeOneTarget(PythonContext.GetPythonContext(operation), fTarget, fSlot, bodyBuilder, false, types))
    {
        if (PythonProtocol.ShouldCoerce(pythonContext, op, types[1], types[0], false))
            PythonProtocol.DoCoerce(pythonContext, bodyBuilder, op, new DynamicMetaObject[2]
            {
                types[1],
                types[0]
            }, 1 != 0);
        if (rSlot != null)
        {
            PythonProtocol.MakeSlotCall(PythonContext.GetPythonContext(operation), types, bodyBuilder, rSlot, true);
            bodyBuilder.FinishCondition(PythonProtocol.MakeBinaryThrow(operation, op, types).Expression, typeof(object));
        }
        else if (PythonProtocol.MakeOneTarget(PythonContext.GetPythonContext(operation), rTarget, rSlot, bodyBuilder, false, types))
            bodyBuilder.FinishCondition(PythonProtocol.MakeBinaryThrow(operation, op, types).Expression, typeof(object));
    }
    return bodyBuilder.GetMetaObject(types);
}
```

“0”

- объединим!



Еще раз...

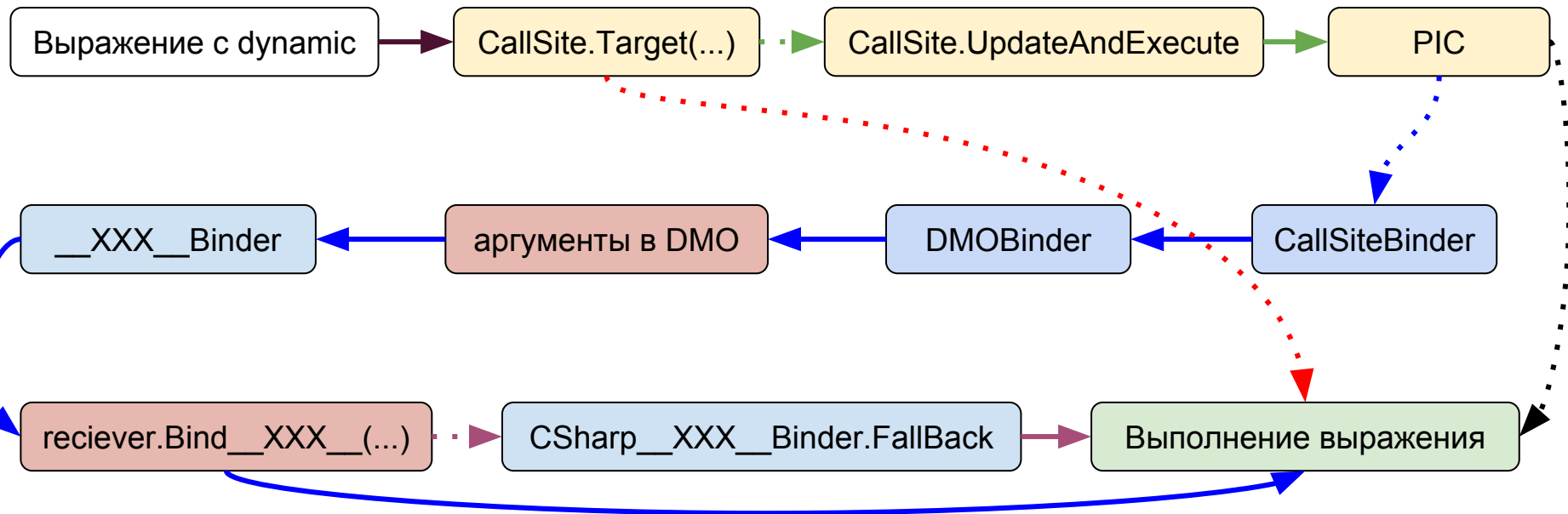
Попробуем вспомнить, что происходило и кто кого чего...

Кто такой CallSite?

Кто такие байндеры?

Кто такой DMO?

Помоги Даше найти смысл жизни...



Простой пример!

Что напечатает программа?

```
static void Main(string[] args)
{
    var myObject = new MyClass();
    Console.WriteLine(myObject + 2);
}
```

(local variable) MyClass myObject

Cannot apply operator '+' to operands of type 'DLR46.MyClass' and 'int'

Простой пример!

Что напечатает программа?

```
class Program
{
    static void Main(string[] args)
    {
        dynamic myObject = new MyClass();
        Console.WriteLine(myObject + 2);
    }
}
```



Реализация MyClass

Что напечатает программа?

```
public class MyClass : IDynamicMetaObjectProvider
{
    public DynamicMetaObject GetMetaObject(Expression parameter)
    {
        return new MyDmo(parameter, BindingRestrictions.Empty, this);
    }
}
```

Реализация MyDMO:

```
public class MyDmo : DynamicMetaObject
{
    public MyDmo(Expression expression, BindingRestrictions restrictions) : base(expression, restrictions) { }
    public MyDmo(Expression expression, BindingRestrictions restrictions, object value) : base(expression, restrictions, value) { }

    public override DynamicMetaObject BindBinaryOperation(BinaryOperationBinder binder, DynamicMetaObject arg)
    {
        if (binder.Operation != ExpressionType.Add)
        {
            return binder.FallbackBinaryOperation(this, arg);
        }
        var restriction = Restrictions.Merge(arg.Restrictions)
            .Merge(BindingRestrictions.GetTypeRestriction(Expression, typeof(MyClass)))
            .Merge(BindingRestrictions.GetTypeRestriction(arg.Expression, typeof(int)));

        var expression = Expression.TypeAs(Expression.Add(Expression.Constant(40), arg.Expression), typeof(object));

        return new DynamicMetaObject(expression, restriction);
    }
}
```

Ответ

Очевидно 42 :)

```
class Program
{
    static void Main(string[] args)
    {
        dynamic myObject = new MyClass();
        Console.WriteLine(myObject + 2);
    }
}
```



У нас будет свой язык с коллсайтом и байндерами

```
//dynamic dynamicObject = "some object"
object dynamicObject = GetDynamicObject();

//описание выражения dynamicObject.class
//binder...
var ourLanguageBinder = new OurLanguageGetMemberBinder("class", false);
//callsite...
var site = (CallSite<Func<CallSite, object, object>>)
    CallSite.Create(typeof(Func<CallSite, object, object>), ourLanguageBinder);

//dynamic result = dynamicObject.class
var result = site.Target(site, dynamicObject);

//Console.WriteLine(result);
Console.WriteLine(result);
```

C#

<module>.PythonType
Для продолжения нажмите любую клавишу

Кастомный байндер

//наш язык имеет набор байндеров

```
public class OurLanguageGetMemberBinder : GetMemberBinder
```

```
{
```

```
    public OurLanguageGetMemberBinder(string name, bool ignoreCase) : base(name, ignoreCase) { }
```

```
    private DynamicMetaObject GetErrorSuggestion(DynamicMetaObject target, DynamicMetaObject errorSuggestion)...
```

```
    public override DynamicMetaObject FallbackGetMember(DynamicMetaObject target, DynamicMetaObject errorSuggestion)
```

```
    {
```

```
        //если это пайтон-объект то может стоит добавить __{Name}__?
```

```
        if (target.Value is OldInstance && !Name.StartsWith("__"))
```

```
        {
```

```
            var newBinder = new OurLanguageGetMemberBinder($"__{Name}__", false);
```

```
            return target.BindGetMember(newBinder);
```

```
        }
```

```
        return GetErrorSuggestion(target, errorSuggestion);
```

```
    }
```

```
}
```

Что мы упустили?

да, там есть еще что-то (:



Что НЕ вошло в доклад

Классы не являющиеся частью прям-ваще реализации (ExpandableObject и DynamicObject)

Как конкретно происходит построение выражений (лучше не надо :))

Как реализовано взаимодействие с COM (вкратце - COM IDispatch)

Что такое и как работает DLR Hosting API (тема на отдельный доклад)

Оказывается там есть еще SymPL (целый функциональный язык в спецификации)

Реализацию многих мелких деталей (Defer, Stitch, генерацию UpdateAndExecute и тп)

Куда гуглить

Можно в гугле - есть статьи но мало

C# 5 Unleashed - Bart De Smet (на самом деле средненько)

Pro DLR in .NET 4 - Chaur Wu (поверхностно)

habrahabr - есть пара статей (и вся пара - не очень)

dlr.codeplex.com - developer notes (очень хорошо!)

В исходники и дебаггер - довольно полное раскрытие темы

Доклад Карлена Симоняна на .NeXT 2014: bit.do/DLR_lecture

Поиграть с примерами: bit.do/DLR_examples

Можно спросить меня: [linkedin.com/in/igoriakovlev](https://www.linkedin.com/in/igoriakovlev)

Успех :)

