Вызов функций в OverScript коде из сторонней программы

admin
January 24 edited March 18

Допустим, вы написали на C# или VB.NET свою библиотеку (dll) и хотите не только вызывать её функции, а и чтобы она вызывала ваши OverScript-функции. Сделать это можно двумя способами:
1. Можно создать делегат Func или Action функцией Delegate() и передать его в библиотеку.
2. Можно передать экземпляр или тип, и вызывать через рефлекшн метод Call, который у них есть.

Через делегаты

Сначала посмотрим, как создаются делегаты.

```
object d=Delegate(Test(int\, long\), "System.Func`3[System.Int32,System.Int64,System.String]");
WriteLine(d.Raise(567, 123L)); //a=567; b=123;
ReadKey();

string Test(int a, long b){
    return $"a={a}; b={b};";
}
```

Второй аргумент в Delegate() - это строковое представление типа Func<int, long, string> в .NET. Проверим в C#:

```
Console.WriteLine(typeof(Func<int, long, string>)); //System.Func`3[System.Int32,System.Int64,System.String]
```

Тип возвращаемого значения указывается последним.

Создадим такую DLL (C#):

```
using System;
public class SimpleTest
{
    public static void SomeLibFunc(Func<int, long, string> delegateByScript)
    {
        Console.WriteLine("Calling from lib: " + delegateByScript(10, 5L));
    }
}
```

Скомпилируем код в SimpleTestLib.dll и положим его в папку с OverScript-скриптом.
Теперь вызовем функцию в библиотеке с передачей её делегата.

```
const object Lib=typeof("SimpleTestLib.dll", "SimpleTest"); //typeof умеет загружать тип из файла (указывается первым)
object d=Delegate(Test(int\, long\), "System.Func`3[System.Int32,System.Int64,System.String]");
Lib->SomeLibFunc(d);
//Calling from lib: a=10; b=5;

ReadKey();

string Test(int a, long b){
    return $"a={a}; b={b};";
}
```

Работает! Функция SomeLibFunc в DLL вызвала функцию Test в скрипте.
Добавлю, что лучше создавать делегат так:

```
const object FuncType=typeof("System.Func`3[System.Int32,System.Int64,System.String]");
object d=Delegate(Test(int\, long\), FuncType);
//object d=Test(int\, long\).Delegate(FuncType); //можно так, смотрится неплохо
```

Так тип делегата получается при загрузке скрипта (так со всеми константами), и если вы ошибётесь со строковым представлением типа, то сразу получите сообщение об ошибке, а не во время выполнения. К тому же, функции Delegate не придётся искать каждый раз тип по строке.

Создавать делегаты EventHandler и EventHandler<T> можно функцией EventHandler:

```
object eh=EventHandler(OnEvent(object\, object\));
eh.Raise(this); //в качестве sender-а передаём текущий инстанс, а в args, если не указывать, будет EventArgs.Empty
//sender: Instance of App; args=System.EventArgs

OnEvent(object sender, object args){
    WriteLine($"sender: {sender}; args={args}");
}
```

Обобщенный EventHandler<T>:

```
object eh=EventHandler(OnEvent(object\, string\), string); //создаём EventHandler<string>
eh.Raise(this, "Hello!");
//sender: Instance of App; arg=Hello!

OnEvent(object sender, string arg){
    WriteLine($"sender: {sender}; arg={arg}");
}
```

Вызов методом Call

DLL:

```
using System;
public class SimpleTest
{
    public static void SomeLibFunc(object obj)
    {
        object v=CallInScript(obj, "Test", "Hello", Environment.TickCount); //сначала сам объект, потом имя функции, пото
        Console.WriteLine("Calling from lib: "+v);
    }
    static object CallInScript(object obj, params object[] args)
    {
        return obj.GetType().InvokeMember("Call", System.Reflection.BindingFlags.InvokeMethod, null, obj, args);
    }
}
```

Функцию CallInScript можно ещё так написать:

```
static object CallInScript(object obj, string funcName, params object[] args)
{
    var mi = obj.GetType().GetMethod("Call");
    return mi.Invoke(obj, new object[] { funcName, args });
}
```

OverScript:

```
const object Lib=typeof("SimpleTestLib.dll", "SimpleTest");
Lib->SomeLibFunc(this);
//Calling from lib: a=Hello; b=1019739031;

string Test(string a, int b){
```

```
    return $"a={a}; b={b};";
}
```

Ещё пример:

```
const object Lib=typeof("SimpleTestLib.dll", "SimpleTest");
Lib->SomeLibFunc(new Foo());
//Calling from lib: a=Hello; b=1020536312;

class Foo{
    public string Test(string a, int b){
        return $"a={a}; b={b};";
    }
}
```

Делать Test() публичным не обязательно.

Можно передавать не экземпляр, а тип (+ нужно ещё экзекутор передать):

```
const object Lib=typeof("SimpleTestLib.dll", "SimpleTest");
Lib->SomeLibFunc(Foo, Executor()); // Executor() возвращает текущий экзекутор
//Calling from lib: a=Hello; b=1020345968;

class Foo{
    public static string Test(string a, int b){
        return $"a={a}; b={b};";
    }
}
```

Тогда в DLL код такой должен быть:

```
public static void SomeLibFunc(object obj, object exec)
{
    object v = CallInScript(obj, exec, "Test",  new object[] { "Hello", Environment.TickCount });
    Console.WriteLine("Calling from lib: " + v);
}
```

Экзекутор - это объект, который выполняет код. Его нужно передавать потому, что в типе (CustomType), в отличие от экземпляра класса (ClassInstance), нет информации о том, какой экзекутор должен выполнять код (тип принадлежит скрипту, а экземпляр экзекутору).
