عنوان: **تولید پویای کد در زمان اجرا توسط Reflection.Emit**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۵/۱۳ ۱۹:۲۰
آدرس: www.dotnettips.info
برچسب‌ها: C#, CIL, CLR, IL, MSIL, Reflection

در ادامه قصد داریم توسط امکانات Reflection به همراه کدهای IL، اشیایی را در زمان اجرا ایجاد کنیم.

Reflection چیست؟

Reflection چیزهایی هستند که با نگاه در یک آینه قابل مشاهده‌اند. در این حالت شخص می‌تواند قسمت‌های مختلف ظاهر خود را برانداز کرده یا قسمتی را تغییر دهید. اما این مساله چه ربطی به دنیای دات نت دارد؟ در دات نت با استفاده از Reflection می‌توان به اطلاعات اشیاء یک برنامه‌ی در حال اجرا دسترسی یافت. برای مثال نام کلاس‌های مختلف آن چیست یا درون کلاسی خاص، چه متدهایی قرار دارند. همچنین با استفاده از Reflection می‌توان رفتارهای جدیدی را نیز به کلاس‌ها و اشیاء افزود یا آن‌ها را تغییر داد.
همواره عنوان می‌شود که از Reflection به دلیل سربار بالای آن پرهیز کنید و تنها از آن به عنوان آخرین راه حل موجود استفاده نمائید و این دقیقا موردی است که در مباحث جاری بیشتر از آن استفاده خواهد شد: ساخت اشیاء جدید در زمان اجرا به کمک کدهای IL و امکانات Reflection

نگاهی به امکانات متداول Reflection

در مثال بعد، نگاهی خواهیم داشت به امکانات متداول Reflection، مانند دسترسی به متدها و خواص یک کلاس و تعویض مقدار یا فراخوانی آن‌ها:

```
using System;

namespace FastReflectionTests
{
    class Person
    {
        public string Name { set; get; }

        public string Speak()
        {
            return string.Format("Hello, my name is {0}.", this.Name);
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            //روش متداول
            var vahid = new Person { Name = "Vahid" };
            Console.WriteLine(vahid.Speak());

            var type = vahid.GetType();

            //نمایش متدهای یک کلاس
            var methods = type.GetMethods();
            foreach (var method in methods)
            {
                Console.WriteLine(method.Name);
            }

            //تغییر مقدار یک خاصیت
            var setNameMethod = type.GetMethod("set_Name");
            setNameMethod.Invoke(obj: vahid, parameters: new[] { "Ali" });

            //فراخوانی یک متد
            var speakMethod = type.GetMethod("Speak");
            var result = speakMethod.Invoke(obj: vahid, parameters: null);
            Console.WriteLine(result);
        }
    }
}
```

با خروجی ذیل

```
Hello, my name is Vahid.
set_Name
get_Name
Speak
ToString
Equals
GetHashCode
GetType
Hello, my name is Ali.
```

توضیحات:

در اینجا یک کلاس شخص با خاصیت نام او تعریف شده است؛ به همراه متدی که رشته‌ای را نمایش خواهد داد.

در متد Main برنامه، ابتدا یک وهله جدید از این شخص ایجاد شده و سپس به روش متداول، متد Speak آن فراخوانی گردیده است. در ادامه کار از امکانات Reflection برای انجام همین امور کمک گرفته شده است.

کار با دریافت نوع یک وهله شروع می‌شود. برای نمونه در اینجا توسط vahid.GetType به نوع وهله ساخته شده دسترسی یافته‌ایم. سپس با داشتن این type، می‌توان به کلیه امکانات Reflection دسترسی یافت. برای مثال توسط GetMethods، لیست کلیه متدهای موجود در کلاس شخص بازگشت داده می‌شود.

اگر به خروجی فوق دقت کنید، پس از سطر اول، 7 سطر بعدی نمایانگر متدهای موجود در کلاس شخص هستند. شاید عنوان کنید که این کلاس به نظر یک متد بیشتر ندارد. اما در دات نت اشیاء از شیء Object مشتق می‌شوند و چهار متد ،Equals ،ToString GetHashCode و GetType متعلق به آن هستند. همچنین خواص تعریف شده نیز در اصل به دو متد set و get به صورت خودکار در کدهای IL برنامه ترجمه خواهند شد. از همین متد set_Name در ادامه برای مقدار دهی خاصیت نام وهله ایجاد شده استفاده شده است.

همانطور که ملاحظه می‌کنید برای فراخوانی یک وهله از طریق Reflection، ابتدا توسط متد type.GetMethod می‌توان به آن دسترسی یافت و سپس با فراخوانی متد Invoke، می‌توان متد مدنظر را بر روی یک شیء مهیا با پارامترهایی که ذکر می‌کنیم، فراخوانی کرد. اگر این متد پارامتری ندارد، آن‌را نال قرار خواهیم داد.

تا اینجا مقدمه‌ای را ملاحظه نمودید که بیشتر جهت تکمیل بحث، حفظ روابط منطقی قسمت‌های مختلف آن و یادآوری مباحث مرتبط با Reflection ذکر شدند.

ایجاد اشیاء در زمان اجرای برنامه

یکی از کلاس‌های مهم Reflection که در منابع مختلف کمتر به آن پرداخته شده است، کلاس DynamicMethod آن است که از آن می‌توان برای ایجاد اشیاء و یا متدهایی پویا در زمان اجرا استفاده کرد. این کلاس قرار گرفته در فضای نام System.Reflection.Emit، دارای یک ILGenerator است که می‌توان به آن OpCodeهایی را اضافه کرد. زمانیکه کار ایجاد این متدپویا به پایان رسید، با استفاده از Delegates امکان دسترسی و اجرای این متد پویا وجود خواهد داشت.

یک مثال کامل را در این زمینه در ادامه ملاحظه می‌نمائید:

```
using System;
using System.Reflection.Emit;

namespace FastReflectionTests
{
    class Program
    {
        static double Divider(int a, int b)
        {
            return a / b;
        }

        delegate double DividerDelegate(int a, int b);
        static void Main(string[] args)
        {
            //روش متداول
            Console.WriteLine(Divider(10, 2));
```

```
            //تعریف امضای متد
            var myMethod = new DynamicMethod(
                                        name: "DividerMethod",
                                        returnType: typeof(double),
                                        parameterTypes: new[] { typeof(int), typeof(int) },
                                        m: typeof(Program).Module);
            //تعریف بدنه متد
            var il = myMethod.GetILGenerator();
            il.Emit(opcode: OpCodes.Ldarg_0); //بارگذاری پارامتر اول بر روی پشته ارزیابی
            il.Emit(opcode: OpCodes.Ldarg_1); //بارگذاری پارامتر دوم بر روی پشته ارزیابی
            il.Emit(opcode: OpCodes.Div); // دو پارامتر از پشته ارزیابی دریافت و تقسیم خواهند شد
            il.Emit(opcode: OpCodes.Ret); // دریافت نتیجه نهایی از پشته ارزیابی و بازگشت آن

            //فراخوانی متد پویا
            //روش اول
            var result = myMethod.Invoke(obj: null, parameters: new object[] { 10, 2 });
            Console.WriteLine(result);

            //روش دوم
            var method = (DividerDelegate)myMethod.CreateDelegate(delegateType:
typeof(DividerDelegate));
            Console.WriteLine(method(10, 2));
        }
    }
}
```

توضیحات

در ابتدای این مثال جدید یک متد متداول تقسیم کننده دو عدد را ملاحظه می‌کنید. در ادامه قصد داریم overload دیگری از این متد را توسط کدهای MSIL در زمان اجرا ایجاد کنیم که دو پارامتر int را قبول می‌کند.

کار با وهله سازی کلاس DynamicMethod موجود در فضای نام System.Reflection.Emit شروع می‌شود. در اینجا کار تعریف امضای متد جدید باید صورت گیرد. برای مثال نام آن چیست، نوع خروجی آن کدام است. نوع پارامترهای آن چیست و نهایتا این متدی که قرار است به صورت پویا به برنامه اضافه شود، باید در کجا قرار گیرد. برای اینکار از Module خود کلاس Program برنامه استفاده شده است.

پس از تعریف امضای متد پویا، نوبت به تعریف بدنه‌ی آن می‌رسد. کار با دریافت یک ILGenerator که می‌توان در آن کدهای IL را وارد کرد شروع می‌شود. مابقی آن تعریف کدهای IL توسط متد Emit است و پیشتر با مقدمات اسمبلی دات نت در قسمت‌های قبلی مبحث جاری آشنا شده‌ایم. ابتدا دو Ldarg فراخوانی شده‌اند تا دو پارامتر ورودی متد را دریافت کنند. سپس Div بر روی آن‌ها صورت گرفته و نهایتا نتیجه بازگشت داده شده است.

خوب؛ تا اینجا موفق شدیم اولین متد پویای خود را ایجاد نمائیم. برای اجرا آن حداقل دو روش وجود دارد:

الف) فراخوانی متد Invoke بر روی آن. با توجه به اینکه قرار نیست این متد بر روی وهله‌ی خاصی اجرا شود، اولین پارامتر آن null وارد شده است و سپس پارامترهای این متد پویا توسط آرگومان دوم متد Invoke وارد شده‌اند.

ب) می‌توان این عملیات را اندکی شکیل‌تر کرد. برای اینکار پیش از متد Main برنامه یک delegate به نام DividerDelegate تعریف شده است. سپس با استفاده از متد CreateDelegate، خروجی این متد پویا را تبدیل به یک delegate کرده‌ایم. اینبار فراخوانی متد پویا بسیار شبیه به متدهای معمولی می‌شود.

نظرات خوانندگان

نویسنده: پویا امینی
تاریخ: ۱۳۹۲/۰۵/۲۲ ۲۰:۶

زمانیکه یک کلاس همراه با یه سری property با استفاده از Reflection.Emit ایجاد کنیم آیا امکانش هست که از این کلاس یک نمونه ایجاد کنیم و به property های آن مقدار بدیم؟ ممنون میشم راهنمایی کنید

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۵/۲۲ ۲۰:۹

بله. در ادامه بحث در مطلب « ایجاد یک کلاس جدید پویا و وهله‌ای از آن در زمان اجرا توسط Reflection.Emit » به آن پرداخته شده.

نویسنده: پویا امینی
تاریخ: ۱۳۹۲/۰۵/۲۳ ۱۱:۴۵

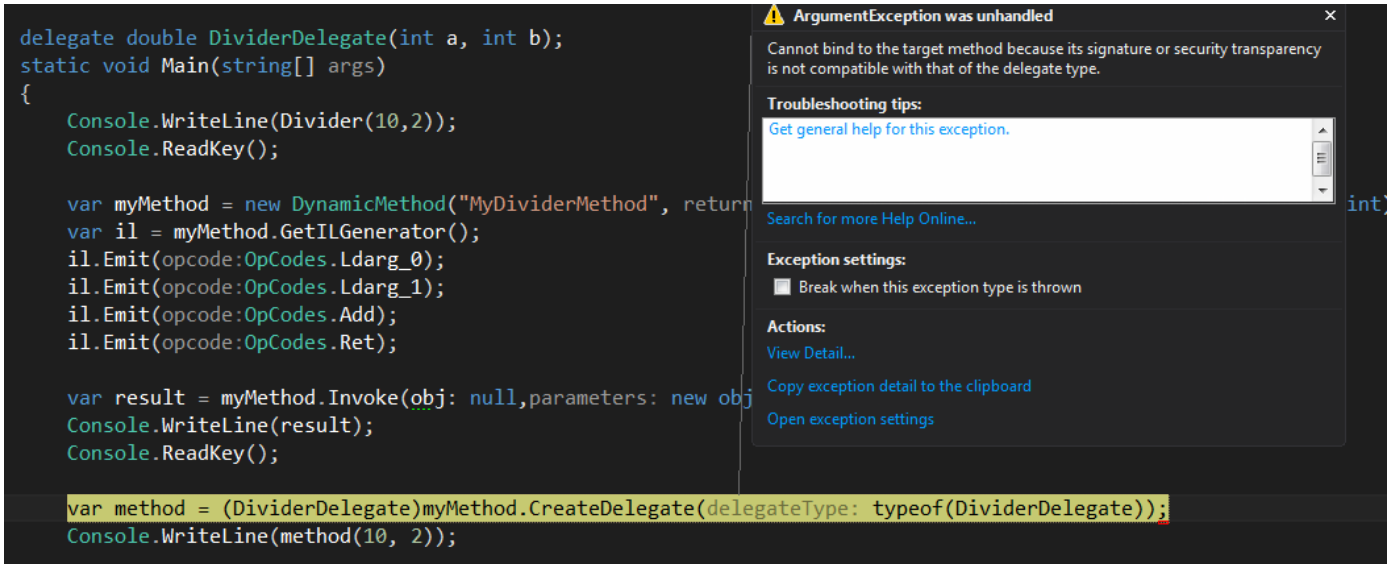
با سلام، من زمانی که می‌خواهم از روش دوم فراخوانی متد استفاده کنم با خطای زیر مواجه می‌شوم

```
var myMethod = new DynamicMethod("MyDividerMethod", returnType: typeof(int), parameterTypes: new[] {
typeof(int), typeof(int) }, m: typeof(Program).Module);
            var il = myMethod.GetILGenerator();
            il.Emit(opcode:OpCodes.Ldarg_0);
            il.Emit(opcode:OpCodes.Ldarg_1);
            il.Emit(opcode:OpCodes.Add);
            il.Emit(opcode:OpCodes.Ret);

            var result = myMethod.Invoke(obj: null,parameters: new object[] { 10, 2 });
            Console.WriteLine(result);
            Console.ReadKey();

            var method = (DividerDelegate)myMethod.CreateDelegate(delegateType:
typeof(DividerDelegate));
            Console.WriteLine(method(10, 2));
```

خطا



نویسنده: پویا امینی
تاریخ: ۱۳۹۲/۰۵/۲۳ ۱۱:۴۸

مشکل را متوجه شدم Signature تعریف Delegate من با متدی که تعریف کردم همخونی نداشت (int و double)
ممنونم

نویسنده: پویا امینی
تاریخ: ۱۳۹۲/۰۵/۲۳ ۱۱:۵۰

امکانش هست این قسمت را بیشتر توضیح بدید چون درست مفهومش رو متوجه نشدم
و **نهایتا این متدی که قرار است به صورت پویا به برنامه اضافه شود، باید در کجا قرار گیرد. برای اینکار از Module خود کلاس Program برنامه استفاده شده است.**
ممنون

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۵/۲۳ ۱۲:۵

تصویر قسمت‌ها و اجزای مختلف تشکیل دهنده یک اسمبلی، برای توضیحات بیشتر در مطلب « ایجاد یک اسمبلی جدید توسط Reflection.Emit » ارائه شده‌اند.