عنوان: **بررسی مثال‌ها و جزئیات بیشتر تولید کدهای پویا توسط Reflection.Emit**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۵/۱۴ ۹:۴۲
آدرس: www.dotnettips.info
برچسب‌ها: C#, CIL, CLR, IL, MSIL, Reflection

نحوه معرفی متغیرهای محلی در Reflection.Emit

ابتدا مثال کامل ذیل را درنظر بگیرید:

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

namespace FastReflectionTests
{
    class Program
    {
        static int Calculate(int a, int b, int c)
        {
            var result = a * b;
            return result - c;
        }

        static void Main(string[] args)
        {
            //روش متداول
            Console.WriteLine(Calculate(10, 2, 3));

            //تعریف امضای متد
            var myMethod = new DynamicMethod(
                                        name: "CalculateMethod",
                                        returnType: typeof(int),
                                        parameterTypes: new[] { typeof(int), typeof(int), typeof(int)
},
                                        m: typeof(Program).Module);
            //تعریف بدنه متد
            var il = myMethod.GetILGenerator();

            il.Emit(opcode: OpCodes.Ldarg_0); // بارگذاری اولین آرگومان بر روی پشته ارزیابی
            il.Emit(opcode: OpCodes.Ldarg_1); // بارگذاری دومین آرگومان بر روی پشته ارزیابی
            il.Emit(opcode: OpCodes.Mul); // انجام عملیات ضرب
            il.Emit(opcode: OpCodes.Stloc_0); // ذخیره سازی نتیجه عملیات ضرب در یک متغیر محلی
            il.Emit(opcode: OpCodes.Ldloc_0); // متغیر محلی را بر روی پشته ارزیابی قرار می‌دهد تا در
عملیات بعدی قابل استفاده باشد
            il.Emit(opcode: OpCodes.Ldarg_2); // آرگومان سوم را بر روی پشته ارزیابی قرار می‌دهد
            il.Emit(opcode: OpCodes.Sub); // انجام عملیات تفریق
            il.Emit(opcode: OpCodes.Ret); // بازگشت نتیجه

            //فراخوانی متد پویا
            var method = (Func<int, int, int, int>)myMethod.CreateDelegate(typeof(Func<int, int, int,
int>));
            Console.WriteLine(method(10, 2, 3));

        }
    }
}
```

در این مثال سعی کرده‌ایم معادل متد Calculate را که در ابتدای برنامه ملاحظه می‌کنید، با کدهای IL تولید کنیم. روش کار مانند قسمت قبل است. ابتدا وهله‌ی جدیدی را از کلاس DynamicMethod جهت معرفی امضای متد پویای خود ایجاد می‌کنیم. در اینجا نوع خروجی را int و نوع سه پارامتر آن‌را به نحوی که مشخص شده است توسط آرایه‌ای از typeهای int معرفی خواهیم کرد. سپس محل قرارگیری کد تولیدی پویا مشخص می‌شود.

در ادامه توسط ILGenerator، آرگومان‌های دریافتی بارگذاری شده، در هم ضرب می‌شوند. سپس نتیجه در یک متغیر محلی ذخیره شده و سپس از آرگومان سوم کسر می‌گردد. در آخر هم این نتیجه بازگشت داده خواهد شد.

در اینجا روش سومی را برای کار با متدهای پویا مشاهده می‌کنید. بجای تعریف یک delegate به صورت صریح همانند قسمت قبل، از یک Func یا حتی Action نیز بنابر امضای متد مد نظر، می‌توان استفاده کرد. در اینجا از یک Func که سه پارامتر int را قبول کرده و خروجی int نیز دارد، استفاده شده است.

اگر برنامه را اجرا کنید ... کرش خواهد کرد! با استثنای ذیل:

```
System.InvalidProgramException was unhandled
Message=Common Language Runtime detected an invalid program.
```

علت اینجا است که در حین کار با System.Reflection.Emit، نیاز است نوع متغیر محلی مورد استفاده را نیز مشخص نمائیم. اینکار را توسط فراخوانی متد DeclareLocal که باید پس از فراخوانی GetILGenerator، درج گردد، می‌توان انجام داد:

```
il.DeclareLocal(typeof(int));
```

با این تغییر، برنامه بدون مشکل اجرا خواهد شد.

نحوه تعریف برچسب‌ها در Reflection.Emit

در ادامه قصد داریم یک مثال پیشرفته‌تر را بررسی کنیم.

```
static int Calculate(int x)
        {
            int result = 0;
            for (int i = 0; i < 10; i++)
            {
                result += i * x;
            }
            return result;
        }
```

در اینجا می‌خواهیم کدهای معادل متد محاسباتی فوق را توسط امکانات System.Reflection.Emit و کدهای IL تولید کنیم.

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

namespace FastReflectionTests
{
    class Program
    {
        static int Calculate(int x)
        {
            int result = 0;
            for (int i = 0; i < 10; i++)
            {
                result += i * x;
            }
            return result;
        }

        static void Main(string[] args)
        {
            //روش متداول
            Console.WriteLine(Calculate(10));

            //تعریف امضای متد
            var myMethod = new DynamicMethod(
                                        name: "CalculateMethod",
                                        returnType: typeof(int), // خروجی متد عدد صحیح است
                                        parameterTypes: new[] { typeof(int) }, // یک پارامتر عدد صحیح
دارد
                                        m: typeof(Program).Module);
            //تعریف بدنه متد
            var il = myMethod.GetILGenerator();

            // از برچسب‌ها برای انتقال کنترل استفاده می‌شود
            // در اینجا به دو برچسب برای تعریف ابتدای حلقه
            // و همچنین برای پرش به جایی که متد خاتمه می‌یابد نیاز داریم
            var loopStart = il.DefineLabel();
            var methodEnd = il.DefineLabel();

            // variable 0; result = 0
            il.DeclareLocal(typeof(int)); //  برای تعریف متغیر محلی نتیجه عملیات
            il.Emit(OpCodes.Ldc_I4_0); // عدد ثابت صفر را بر روی پشته ارزیابی قرار می‌دهد
            il.Emit(OpCodes.Stloc_0); // و نهایتا این عدد ثابت به متغیر محلی انتساب داده خواهد شد
```

```
            // variable 1; i = 0
            il.DeclareLocal(typeof(int)); // در اینجا کار تعریف و مقدار دهی متغیر حلقه انجام می‌شود
            il.Emit(OpCodes.Ldc_I4_0); // عدد ثابت صفر را بر روی پشته ارزیابی قرار می‌دهد
            il.Emit(OpCodes.Stloc_1); // و نهایتا این عدد ثابت به متغیر حلقه در ایندکس یک انتساب داده
خواهد شد

            // در اینجا کار تعریف بدنه حلقه شروع می‌شود
            il.MarkLabel(loopStart); // شروع حلقه را علامتگذاری می‌کنیم تا بعدا بتوانیم به این نقطه پرش
نمائیم
            il.Emit(OpCodes.Ldloc_1); // در ادامه می‌خواهیم بررسی کنیم که آیا مقدار متغیر حلقه از عدد 10
کوچکتر است یا خیر
            il.Emit(OpCodes.Ldc_I4, 10); // عدد ثابت ده را بر روی پشته ارزیابی قرار می‌دهد
            // برای انجام بررسی‌های تساوی یا کوچکتر یا بزرگتر نیاز است ابتدا دو متغیر مدنظر بر روی پشته قرار
گیرند
            il.Emit(OpCodes.Bge, methodEnd);  // اگر اینطور نیست و مقدار متغیر از 10 کمتر نیست، کنترل
برنامه را به انتهای متد هدایت خواهیم کرد

            // i * x
            il.Emit(OpCodes.Ldloc_1); // مقدار متغیر حلقه را بر روی پشته قرار می‌دهد
            il.Emit(OpCodes.Ldarg_0); // مقدار اولین آرگومان متد را بر روی پشته قرار می‌دهد
            il.Emit(OpCodes.Mul); // انجام عملیات ضرب
            // نتیجه این عملیات اکنون بر روی پشته قرار گرفته است

            // result +=
            il.Emit(OpCodes.Ldloc_0); // متغیر نتیجه را بر روی پشته قرار می‌دهد
            il.Emit(OpCodes.Add); // اکنون عملیات جمع بر روی نتیجه ضرب قسمت قبل که بر روی پشته قرار
دارد و همچنین متغیر نتیجه انجام می‌شود
            il.Emit(OpCodes.Stloc_0); // ذخیره سازی نتیجه در متغیر محلی

            // i++
            // در اینجا کار افزایش متغیر حلقه انجام می‌شود
            il.Emit(OpCodes.Ldloc_1); // مقدار متغیر حلقه بر روی پشته قرار می‌گیرد
            il.Emit(OpCodes.Ldc_I4_1); // عدد ثابت یک بر روی پشته قرار می‌گیرد
            il.Emit(OpCodes.Add); // سپس این دو عدد بارگذاری شده با هم جمع خواهند شد
            il.Emit(OpCodes.Stloc_1); // نتیجه در متغیر حلقه ذخیره خواهد شد

            // مرحله بعد شبیه سازی حلقه با پرش به ابتدای برچسب آن است
            il.Emit(OpCodes.Br, loopStart);

            //در اینجا انتهای متد علامتگذاری شده است
            il.MarkLabel(methodEnd);
            il.Emit(OpCodes.Ldloc_0); // مقدار نتیجه بر روی پشته قرار داده شده
            il.Emit(OpCodes.Ret); // و بازگشت داده می‌شود

            //فراخوانی متد پویا
            var method = (Func<int, int>)myMethod.CreateDelegate(typeof(Func<int, int>));
            Console.WriteLine(method(10));
        }
    }
}
```

کد کامل معادل را به همراه کامنت گذاری سطر به سطر آن، ملاحظه می‌کنید. در اینجا نکته‌های جدید، نحوه تعریف برچسب‌ها و انتقال کنترل برنامه به آن‌ها هستند؛ جهت شبیه سازی حلقه و همچنین خاتمه آن و انتقال کنترل به انتهای متد.

فراخوانی متدها توسط کدهای پویای Reflection.Emit

در ادامه کدهای کامل یک مثال متد پویا را که متد print را فراخوانی می‌کند، ملاحظه می‌کنید:

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

namespace FastReflectionTests
{
    class Program
    {
        public static void print(int i)
        {
            Console.WriteLine("i: {0}", i);
        }

        static void Main(string[] args)
        {
            روش متداول//
```

```
            print(10);

            //تعریف امضای متد
            var myMethod = new DynamicMethod(
                                    name: "myMethod",
                                    returnType: typeof(void),
                                    parameterTypes: null, // پارامتری ندارد
                                    m: typeof(Program).Module);
            //تعریف بدنه متد
            var il = myMethod.GetILGenerator();
            il.Emit(OpCodes.Ldc_I4, 10); // عدد ثابت 10 را بر روی پشته قرار می‌دهد
            // اکنون این مقدار بر روی پشته است و از آن می‌توان برای فراخوانی متد پرینت استفاده کرد
            il.Emit(OpCodes.Call, typeof(Program).GetMethod("print"));
            il.Emit(OpCodes.Ret);


            //فراخوانی متد پویا
            var method = (Action)myMethod.CreateDelegate(typeof(Action));
            method();
        }
    }
}
```

در اینجا از OpCode مخصوص فراخوانی متدها به نام Call که در قسمت‌های قبل در مورد آن بحث شد، استفاده گردیده است.
برای اینکه امضای دقیقی را در اختیار آن قرار دهیم، می‌توان از Reflection استفاده کرد که نمونه‌ای از آن‌را در اینجا ملاحظه می‌کنید.
به علاوه چون خروجی امضای متد ما از نوع void است، اینبار delegate تعریف شده را از نوع Action تعریف کرده‌ایم و نه از نوع Func.

فراخوانی متدهای پویای Reflection.Emit توسط سایر متدهای پویای Reflection.Emit

فراخوانی یک متد پویای مشخص از طریق متدهای پویای دیگر نیز همانند مثال قبل است:

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

namespace FastReflectionTests
{
    class Program
    {
        static void Main(string[] args)
        {
            //تعریف امضای متد
            var myMethod = new DynamicMethod(
                                    name: "mulMethod",
                                    returnType: typeof(int),
                                    parameterTypes: new[] { typeof(int) },
                                    m: typeof(Program).Module);
            //تعریف بدنه متد
            var il = myMethod.GetILGenerator();
            il.Emit(OpCodes.Ldarg_0); // اولین آرگومان متد را بر روی پشته قرار می‌دهد
            il.Emit(OpCodes.Ldc_I4, 42); // عدد ثابت 42 را بر روی پشته قرار می‌دهد
            il.Emit(OpCodes.Mul); // ضرب این دو در هم
            il.Emit(OpCodes.Ret); // بازگشت نتیجه

            //فراخوانی متد پویا
            var method = (Func<int, int>)myMethod.CreateDelegate(typeof(Func<int, int>));
            Console.WriteLine(method(10));

            // فراخوانی متد پویای فوق در یک متد پویای دیگر
            var callerMethod = new DynamicMethod(
                                    name: "callerMethod",
                                    returnType: typeof(int),
                                    parameterTypes: new[] { typeof(int), typeof(int) },
                                    m: typeof(Program).Module);
            //تعریف بدنه متد
            var callerMethodIL = callerMethod.GetILGenerator();
            callerMethodIL.Emit(OpCodes.Ldarg_0); // پارامتر اول متد را بر روی پشته قرار می‌دهد
            callerMethodIL.Emit(OpCodes.Ldarg_1); // پارامتر دوم متد را بر روی پشته قرار می‌دهد
            callerMethodIL.Emit(OpCodes.Mul); // ضرب این دو در هم
            //حاصل ضرب اکنون بر روی پشته است که در فراخوانی بعدی استفاده می‌شود
            callerMethodIL.Emit(OpCodes.Call, myMethod); // فراخوانی یک متد پویای دیگر
```

```
            callerMethodIL.Emit(OpCodes.Ret);

            //فراخوانی متد پویای جدید
            var method2 = (Func<int, int, int>)callerMethod.CreateDelegate(typeof(Func<int, int,
int>));
            Console.WriteLine(method2(10, 2));
        }
    }
}
```

در مثال فوق ابتدا یک متد پویای ضرب را تعریف کرده‌ایم که عددی صحیح را دریافت و آن‌را در 42 ضرب می‌کند و نتیجه را بازگشت می‌دهد.
سپس متد پویای دومی تعریف شده است که دو عدد صحیح را دریافت و این دو را در هم ضرب کرده و سپس نتیجه را به عنوان پارامتر به متد پویای اول ارسال می‌کند.
هنگام فراخوانی OpCodes.Call، پارامتر دوم باید از نوع MethodInfo باشد. نوع یک DynamicMethod نیز همان MethodInfo است. بنابراین برای فراخوانی آن، کار خاصی نباید انجام شود و صرفا ذکر نام متغیر مرتبط با مد پویای مدنظر کفایت می‌کند.