عنوان: **دسترسی سریع به مقادیر خواص توسط Reflection.Emit**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۵/۱۵ ۱:۳
آدرس: www.dotnettips.info
برچسب‌ها: C#, CIL, CLR, IL, MSIL, Reflection

اگر پروژه‌های چندسال اخیر را مرور کرده باشید خصوصا در زمینه ORMها و یا Serializerها و کلا مواردی که با Reflection زیاد سروکار دارند، تعدادی از آن‌ها پیشوند fast را یدک می‌کشند و با ارائه نمودارهایی نشان می‌دهند که سرعت عملیات و کتابخانه‌های آن‌ها چندین برابر کتابخانه‌های معمولی است و ... سؤال مهم اینجا است که رمز و راز این‌ها چیست؟
فرض کنید تعاریف کلاس User به صورت زیر است:

```
public class User
{
     public int Id { set; get; }
}
```

همانطور که در قسمت‌های قبل نیز عنوان شد، خاصیت Id در کدهای IL نهایی به صورت متدهای get_Id و set_Id ظاهر می‌شوند.
حال اگر یک متد پویا ایجاد کنیم که بجای هر بار Reflection جهت دریافت مقدار Id، خود متد get_Id را مستقیما صدا بزند، چه خواهد شد؟
پیاده سازی این نکته را در ادامه ملاحظه می‌کنید:

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Reflection;
using System.Reflection.Emit;

namespace FastReflectionTests
{
    /// <summary>
    /// کلاسی برای اندازه گیری زمان اجرای عملیات
    /// </summary>
    public class Benchmark : IDisposable
    {
        Stopwatch _watch;
        string _name;

        public static Benchmark Start(string name)
        {
            return new Benchmark(name);
        }

        private Benchmark(string name)
        {
            _name = name;
            _watch = new Stopwatch();
            _watch.Start();
        }

        public void Dispose()
        {
            _watch.Stop();
            Console.WriteLine("{0} Total seconds: {1}"
                              , _name, _watch.Elapsed.TotalSeconds);
        }
    }

    public class User
    {
        public int Id { set; get; }
    }

    class Program
    {
        public static Func<object, object> GetFastGetterFunc(string propertyName, Type ownerType)
        {
            var propertyInfo = ownerType.GetProperty(propertyName, BindingFlags.Instance |
BindingFlags.Public);
```

```
            if (propertyInfo == null)
                return null;

            var getter = ownerType.GetMethod("get_" + propertyInfo.Name,
                                             BindingFlags.Instance | BindingFlags.Public |
BindingFlags.FlattenHierarchy);
            if (getter == null)
                return null;

            var dynamicGetterMethod = new DynamicMethod(
                                              name: "_",
                                              returnType: typeof(object),
                                              parameterTypes: new[] { typeof(object) },
                                              owner: propertyInfo.DeclaringType,
                                              skipVisibility: true);
            var il = dynamicGetterMethod.GetILGenerator();

            il.Emit(OpCodes.Ldarg_0); // Load input to stack
            il.Emit(OpCodes.Castclass, propertyInfo.DeclaringType); // Cast to source type
            // نکته مهم در اینجا فراخوانی نهایی متد گت بدون استفاده از ریفلکشن است //
            il.Emit(OpCodes.Callvirt, getter); //calls its get method

            if (propertyInfo.PropertyType.IsValueType)
                il.Emit(OpCodes.Box, propertyInfo.PropertyType);//box

            il.Emit(OpCodes.Ret);

            return (Func<object, object>)dynamicGetterMethod.CreateDelegate(typeof(Func<object,
object>));
        }


        static void Main(string[] args)
        {
            //تهیه لیستی از داده‌ها جهت آزمایش
            var list = new List<User>();
            for (int i = 0; i < 1000000; i++)
            {
                list.Add(new User { Id = i });
            }

            // دسترسی به اطلاعات لیست به صورت متداول از طریق ریفلکشن معمولی
            var idProperty = typeof(User).GetProperty("Id");
            using (Benchmark.Start("Normal reflection"))
            {
                foreach (var item in list)
                {
                    var id = idProperty.GetValue(item, null);
                }
            }

            // دسترسی از طریق روش سریع دستیابی به اطلاعات خواص
            var fastIdProperty = GetFastGetterFunc("Id", typeof(User));
            using (Benchmark.Start("Fast Property"))
            {
                foreach (var item in list)
                {
                    var id = fastIdProperty(item);
                }
            }
        }
    }
}
```

توضیحات:

از کلاس Benchmark برای نمایش زمان انجام عملیات دریافت مقادیر Id از یک لیست، به دو روش Reflection متداول و روش صدا زدن مستقیم متد get_Id استفاده شده است.

در متد GetFastGetterFunc، ابتدا به متد get_Id خاصیت Id دسترسی پیدا خواهیم کرد. سپس یک متد پویا ایجاد می‌کنیم تا این get_Id را مستقیما صدا بزند. حاصل کار را به صورت یک delegate بازگشت می‌دهیم. شاید عنوان کنید که در اینجا هم حداقل در ابتدای کار متد، یک Reflection اولیه وجود دارد. پاسخ این است که مهم نیست؛ چون در یک برنامه واقعی، تهیه delegates در زمان آغاز برنامه انجام شده و حاصل کش می‌شود. بنابراین در زمان استفاده نهایی، به هیچ عنوان با سربار Reflection مواجه نخواهیم بود.

خروجی آزمایش فوق بر روی سیستم معمولی من به صورت زیر است:

```
Normal reflection Total seconds: 2.0054177
Fast Property Total seconds: 0.0552056
```

بله. نتیجه روش GetFastGetterFunc واقعا سریع و باور نکردنی است!

چند پروژه که از این روش استفاده می‌کنند

Dapper

AutoMapper

fastJson

در سورس این کتابخانه‌ها روش‌های فراخوانی مستقیم متدهای set نیز پیاده سازی شده‌اند که جهت تکمیل بحث می‌توان به آن‌ها مراجعه نمود.

ماخذ اصلی

این کشف و استفاده خاص، از اینجا شروع و عمومیت یافته است و پایه تمام کتابخانه‌هایی است که پیشوند fast را به خود داده‌اند:

faster using dynamic method calls 2000%