عنوان: **اجرای متد از طریق Reflection**
نویسنده: سیدمجتبی حسینی
تاریخ: ۱۳۹۱/۰۷/۱۷ ۱۴:۰
آدرس: www.dotnettips.info
برچسب‌ها: C#, Reflection

برای اجرای متد درون یک کلاس از طریق Reflection ابتدا نوع آن کلاس را به دست می‌آوریم و سپس از طریق کلاس Activator.CreateInstance یک نمونه از آن کلاس را ساخته و در متغیری از نوع object ذخیره کرده و با استفاده از GetMethod اطلاعات متد مورد نظر خود را در متغیری ذخیره کرده و سپس از طریق دستور Invoke آن متد را اجرا می‌کنیم. دستور Invoke دو سربارگذاری دارد که در یک نوع از آن، متغیر حاوی نمونه کلاس و پارامترهای متد مورد نظر، در قالب یک آرایه از نوع object، به عنوان آرگومان پذیرفته می‌شود. با امضای زیر

```
public Object Invoke(Object obj, Object[] parameters)
```

به مثال زیر که چگونگی این عملیات را شرح می‌دهد، توجه کنید:

```
public class TestMath
    {
        public int Squar(int i)
        {
            return i*i;
        }
    }

static void Main(string[] args)
        {
            Type type = typeof (TestMath);//به دست آوردن نوع کلاس
            object obj = Activator.CreateInstance(type);//ساختن نمونه‌ای از نوع مورد نظر
            MethodInfo methodInfo = type.GetMethod("Squar");//یافتن اطلاعات متد مورد نظر
            Console.WriteLine(methodInfo.Invoke(obj, new object[] { 100 }));// و نمایش نتیجه object[]
ارسال عدد 100 به صورت
            Console.Read();
        }
```

توجه کنید که دو متد GetMethod و Invoke در فضای نام System.Reflection قرار دارند.

روش دیگر

در شیوه دیگر برای انجام این کار، نیازی به استفاده از GetMethod و Invoke نیست و فراخوانی متد مورد نظر بسیار شبیه فراخوانی عادی متدهاست و نیازی به ساخت متغیر ویژه‌ای از نوع object[] برای ارسال پارامترها نیست. برای انجام این کار فقط کافیست نوع متغیری که نوع نمونه‌سازی شده را نگه‌می‌دارد (در اینجا نمونه ای از کلاس را نگه‌می‌دارد) به صورت dynamic باشد:

```
static void Main(string[] args)
        {
            Type type = typeof (TestMath);
            dynamic obj = Activator.CreateInstance(type);
            Console.WriteLine(obj.Square(100));
            Console.Read();
        }
```

توجه کنید که بعد از تعریف obj، با درج نقطه در کنار آن، منوی Code Insight متد Square را شامل نمی‌شود اما کامپایلر آن را می‌پذیرد.

نظرات خوانندگان

نویسنده: KishIsland
تاریخ: ۱۳۹۱/۱۲/۱۵ ۱۹:۴۷

برای تست کد بالا من یک کلاس بشکل زیر تعریف کردم:

```
class Test
    {
        public void func1()
        {
            Console.WriteLine("Hello World!");
        }
    }
```

و در قسمت main برنامه فراخوانی رو بشکل زیر نوشتم مطابق روش دوم ولی برنامه Exception داد:

```
Type type = typeof(Test);
            dynamic obj = Activator.CreateInstance(type);
            Console.WriteLine(obj.func1());
```

علت بروز استثناء void بودن متد هست.می خواستم بدونم برای مواقعی که متد بصورت void تعریف شده چکار باید کرد؟
سپاس

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۲/۱۵ ۲۱:۲۰

در این حالت بجای

```
Console.WriteLine(obj.func1());
```

فقط کافی هست بنویسید

```
obj.func1();
```

عنوان: **چگونگی دسترسی به فیلد و خاصیت غیر عمومی**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۳/۱۵ ۱۶:۱۰
آدرس: www.dotnettips.info
گروه‌ها: C#, Reflection, read-only

یک از ابتدایی‌ترین مواردی که در یادگیری دات نت آموزش داده می‌شود مباحث مربوط به کپسوله سازی است. برای مثال فیلدها و خواص Private که به صورت خصوصی هستند یا Protected هستند از خارج کلاس قابل دسترسی نیستند. برای دسترسی به این کلاس‌ها باید از خواص یا متدهای عمومی استفاده کرد.

```
public class Book
    {
        private int code = 10;

        public int GetCode()
        {
            return code;
        }
    }
```

یا فیلدها و خواصی که به صورت فقط خواندنی هستند،(RealOnly) امکان تغییر مقدار برای اون‌ها وجود ندارد. برای مثال کد پایین کامپایل نخواهد شد.

```
public class Book
    {
        private readonly int code = 10;

        public int GetCode()
        {
            return code = 20;
        }
    }
```

اما در دات نت با استفاده از Reflectionها می‌تونیم تمام قوانین بالا رو نادیده بگیریم. یعنی می‌تونیم هم به خواص و فیلدهای غیر عمومی کلاس دسترسی پیدا کنیم و هم می‌تونیم مقدار فیلدهای فقط خواندنی رو تغییر بدیم. به مثال‌های زیر دقت کنید.
#مثال اول

```
using System.Reflection;

 public class Book
 {
        private int code = 10;
 }

 public class Program
 {
        static void Main( string[] args )
        {
            Book book = new Book();
            var codeField = book.GetType().GetField( "code", BindingFlags.NonPublic |
BindingFlags.Instance );
            codeField.SetValue( book, 20 );
            var value = codeField.GetValue( book );
        }
    }
```

ابتدا یک کلاس که دارای یک متغیر به نام کد است ساخته ایم که مقدار 10 را دارد. فیلد به صورت private است. بعد از اجرا به راحتی مقدار Code را به دست می‌آوریم.

```
class Program
{
   static void Main( string[] args )
   {
      Book book = new Book();
      var codeField = book.GetType().GetField( "code", BindingFlags.NonPublic | BindingFlags.Instance );
      var value = codeField.GetValue( book );
   }
}
```

value 10

حتی امکان تغییر مقدار فیلد private هم امکان پذیر است.

```
class Program
{
   static void Main( string[] args )
   {
      Book book = new Book();
      var codeField = book.GetType().GetField( "code", BindingFlags.NonPublic | BindingFlags.Instance );
      codeField.SetValue( book, 100 );
      var value = codeField.GetValue( book );
   }
}
```

value 100

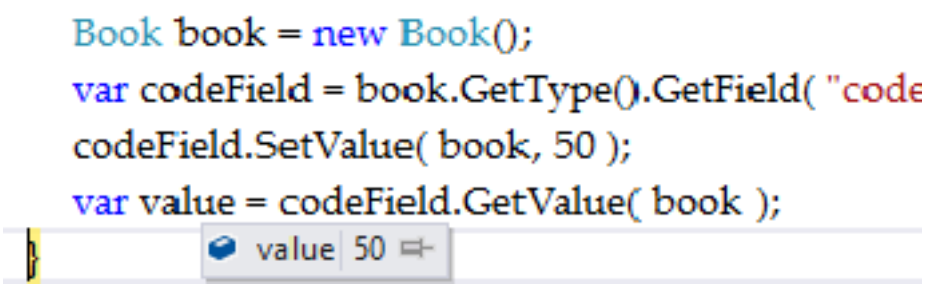
#مثال دوم.
در این مثال قصد داریم مقدار یک فیلد، از نوع فقط خواندنی رو تغییر دهیم.

```
using System.Reflection;

 public class Book
 {
        private readonly int code = 10;
 }

 public class Program
 {
        static void Main( string[] args )
        {
            Book book = new Book();
            var codeField = book.GetType().GetField( "code", BindingFlags.NonPublic |
BindingFlags.Instance );
            codeField.SetValue( book, 50);
            var value = codeField.GetValue( book );
        }
    }
```

بعد از اجرا مقدار متغیر code به 50 تغییر می‌یابد.



مطالب تکمیلی

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۳/۱۷ ۱۰:۱۰

البته مباحث Reflection، تابع سطح دسترسی کد فراخوان است (همان لینک آخر بحث جهت تاکید بیشتر و همچنین تنویر مقدمه):

« Security Considerations for Reflection »

برای نمونه در حالت medium trust، گزینه ReflectionPermission غیرفعال است.

برای آزمایش این مسایل می‌شود از دو برنامه Permview و Permcalc استفاده کرد.

نویسنده: سالار خلیل زاده
تاریخ: ۱۳۹۲/۰۳/۱۸ ۹:۱۸

ReflectionMagic جهت همین کار طراحی شده

http://nuget.org/packages/ReflectionMagic

نویسنده: reza
تاریخ: ۱۳۹۳/۰۵/۲۵ ۱۷:۹

آیا می‌توان به کمک رفلکشن به خصوصیتی که مثلا Set ندارد مقدار دهی کرد. بعنوان مثال

```
class Program
{
  static void Main(string[] args)
  {
      Book book = new Book();
      var codeprop = book.GetType().GetProperty("Code", BindingFlags.Public | BindingFlags.Instance);
      codeprop.SetValue(book, 20, null);
      var value = codeprop.GetValue(book, null);
  }
}

public class Book
{
  public int code;
  public int Code
  {
      get { return code; }
  }
}
```

نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۳/۰۵/۲۶ ۱۴:۰

خیر! با یک ArgumentException و پیغام Property set method not found مواجه خواهید شد. اما در مثال بالا می‌توان مقدار فیلد code را تغییر داد که در نتیجه خاصیت Code نیز مقدار جدید را برگشت می‌دهد.

عنوان: **فراخوانی داینامیک اسمبلی‌های آفیس در برنامه**
نویسنده: مهران ر
تاریخ: ۱۳۹۲/۱۱/۰۷ ۱۶:۲۵
آدرس: www.dotnettips.info
گروه‌ها: C#, Reflection, Dynamic Programming

در بسیاری از پروژه‌های دات نت، نیاز به استفاده از فایلهای نرم افزار آفیس، از قبیل ورد و اکسل و ... وجود دارد. برای مثال گاهی لازم است اطلاعات یک گرید، یا هر منبع داده‌ای، در قالب اکسل به کاربر نمایش داده شود. بدین شکل که این فایلها در زمان اجرا ساخته شده و به کاربر نمایش داده شود .حال فرض کنید شما روی سیستم خودتان Office2007 را نصب کرده اید و به اسمبلی‌های این ورژن دسترسی دارید. البته بدون نیاز به نصب آفیس نیز میتوان به این توابع دسترسی داشت و از آنها در برنامه استفاده کرد که همان استفاده از Primary Interop Assemblies میباشد. مشکلی که ممکن است پیش آید این است که در کامپیوترهای کاربران ممکن است ورژن‌های مختلفی از آفیس نصب باشد مانند 2003 -2007-2010-2013 و اگر با ورژن اسمبلی‌هایی که فراخوانی‌های فایلهای اکسل از طریق آن انجام شده باشد متفاوت باشد، برنامه اجرا نمی‌شود .

در حالت معمول برای نمایش یک فایل آفیس مثل اکسل در برنامه، ابتدا اسمبلی مربوطه را (اکسل در این مثال) که به نام Microsoft.Office.Interop.Excel میباشد به اسمبلی‌های برنامه اضافه کرده (از طریق add reference) و برای نمایش یک فایل اکسل در زمان اجرا از کدهای زیر استفاده مینماییم :

```
try
            {
                var application =
(Microsoft.Office.Interop.Excel.Application)Activator.CreateInstance(Type.GetTypeFromProgID("Excel.Appl
ication"));
               Workbook  wrkBook;
                var  wbk = application.Workbooks;
                 wrkBook = wbk.Add();
                wrkBook.Activate();
                application.Visible = true;
             }
            catch (Exception ex)
            {
                Error(ex.Message);
            }
```

حال اگر آفیس 2010 به عنوان مثال در سیستم ما نصب باشد، ورژن این اسمبلی 14 می‌باشد و اگر این برنامه را در کامپیوتر کلاینتی که آفیس 2007 بر روی آن نصب باشد انتشار دهیم اجرا نمیشود. برای حل این مشکل بنده با استفاده از روش dynamic این موضوع را حل کردم و بنظر می‌رسد راه‌های دیگری نیز برای حل آن وجود داشته باشد.

در این روش با توجه به ورژن آفیسی که بر روی سیستم کاربر نصب شده اسمبلی مربوطه را از سیستم کاربر لود کرده و فایلهای آفیس را اجرا مینماییم. در ابتدا تشخیص میدهیم چه ورژنی از آفیس بر روی سیستم کاربر نصب است :

```
string strVersion = null;
               dynamic objEApp =Activator.CreateInstance(Type.GetTypeFromProgID("Excel.Application"));
               if (objEApp.Version == "12.0")
               {
                   strVersion = "2007";
               }
               else if (objEApp.Version == "14.0")
               {
                   strVersion = "2010";
               }
```

روش دیگر برای انجام اینکار استفاده از اطلاعات رجیستری ویندوز است :

```
string strEVersionSubKey = "\\Excel.Application\\CurVer";
            string strValue = null; //Value Present In Above Key
            string strVersion = null; //Determines Excel Version
            RegistryKey rkVersion = null; //Registry Key To Determine Excel Version
            rkVersion = Registry.ClassesRoot.OpenSubKey(strEVersionSubKey, false); //Open Registry Key
```

```
            if ((rkVersion != null)) //If Key Exists
            {
                strValue = (string)rkVersion.GetValue(string.Empty); //Get Value
                strValue = strValue.Substring(strValue.LastIndexOf(".") + 1); //Store Value
                switch (strValue) //Determine Version
                {
                    case "11":
                        strVersion = "2003";
                        break;

                    case "12":
                        strVersion = "2007";
                        break;

                    case "14":
                        strVersion = "2010";
                        break;

                }
```

حال با استفاده از تابع assembly.load() اسمبلی مورد نیاز را لود کرده و در برنامه استفاده مینماییم :

```
if (strVersion == "2007")
                {
                    string strAssemblyOff2007 =
                        "Microsoft.Office.Interop.Excel, Version=12.0.0.0, Culture=neutral,
PublicKeyToken=71e9bce111e9429c";
                    try
                    {
                        Assembly xslExcelAssembly = Assembly.Load(strAssemblyOff2007); //Load Assembly
                        Type type = xslExcelAssembly.GetTypes().Single(t => t.Name ==
"ApplicationClass");
                        dynamic application = Activator.CreateInstance(type);
                        var workbooks = application.Workbooks;
                        var workbook = workbooks.Add();
                        var worksheet = workbook.Worksheets[1];
                        workbook.Activate();
                        application.Visible = true;
                    }
                    catch (Exception ex)
                    {
                    }
                }
```

در این حالت بدون اینکه بدانیم بر روی سیستم کاربر چه ورژنی از آفیس نصب است میتوان فایلهای آفیس را در زمان اجرا لود کرده و استفاده کرد .

نظرات خوانندگان

نویسنده: حسین
تاریخ: ۱۳۹۲/۱۱/۱۹ ۲۰:۳۳

با تشکر از مقاله مفید شما
من یه بار توی یه پروژه یک تمپلت ورد ایجاد کردم و توش انواع اقسام چارت‌ها و جدول‌ها رو توش رسم کردم و کلی هم روش کار کردم تا گزارش خوبی از کار در بیارد
واقعا اطلاع نداشتم با ورژن‌ها مختلف اجرا نمیشه!
الان عذاب وجدان گرفتم :)

عنوان: **استفاده ازExpressionها جهت ایجاد Strongly typed view در ASP.NET MVC**
نویسنده: محسن موسوی
تاریخ: ۱۳۹۳/۰۴/۱۵ ۰:۰
آدرس: www.dotnettips.info
گروه‌ها: MVC, Reflection, Kendo UI, jqGrid

مدل زیر را در نظر بگیرید:

```
/// <summary>
    ///
    /// </summary>
    public class CompanyModel
    {
        /// <summary>
        /// Table Identity
        /// </summary>
        public int Id { get; set; }

        /// <summary>
        /// Company Name
        /// </summary>
        [DisplayName("نام شرکت")]
        public string CompanyName { get; set; }

        /// <summary>
        /// Company Abbreviation
        /// </summary>
        [DisplayName("نام اختصاری شرکت")]
        public string CompanyAbbr { get; set; }

    }
```

از View زیر جهت نمایش لیستی از شرکت‌ها متناظر با مدل جاری استفاده میشود:

```
@{
    const string viewTitle = "شرکت ها";
    ViewBag.Title = viewTitle;
    const string gridName = "companies-grid";
}
<div class="col-md-12">
    <div class="form-panel">
        <header>
            <div class="title">
                <i class="fa fa-book"></i>
                @viewTitle
            </div>
        </header>
        <div class="panel-body">
            <div id="@gridName">
            </div>
        </div>
    </div>
</div>
</div>
@section scripts
{
    <script type="text/javascript">
        $(document).ready(function () {
            $("#@gridName").kendoGrid({
                dataSource: {
                    type: "json",
                    transport: {
                        read: {
                            url: "@Html.Raw(Url.Action(MVC.Company.CompanyList()))",
                            type: "POST",
                            dataType: "json",
                            contentType: "application/json"
                        }
                    },
                    schema: {
                        data: "Data",
                        total: "Total",
                        errors: "Errors"
                        }
                    },
                    pageSize: 10,
```

```
                    serverPaging: true,
                    serverFiltering: true,
                    serverSorting: true
                },
                pageable: {
                    refresh: true
                },
                sortable: {
                    mode: "multiple",
                    allowUnsort: true
                },
                editable: false,
                filterable: false,
                scrollable: false,
                columns: [ {
                    field: "CompanyName",
                    title: "نام شرکت",
                    sortable: true,
                }, {
                    field: "CompanyAbbr",
                    title: "مخفف نام شرکت",
                    sortable: true
                }]
            });
        });
    </script>
}
```

مشکلی که در کد بالا وجود دارد این است که با تغییر نام هر یک از متغییر هایمان ، اطلاعات گرید در ستون مربوطه نمایش داده نمیشود.همچنین عناوین ستونها نیز از DisplayName مدل پیروی نمیکنند.توسط متدهای الحاقی زیر این مشکل برطرف شده است.

```
/// <summary>
    ///
    /// </summary>
    public static class PropertyExtensions
    {
        /// <summary>
        ///
        /// </summary>
        /// <typeparam name="T"></typeparam>
        /// <param name="expression"></param>
        /// <returns></returns>
        public static MemberInfo GetMember<T>(this Expression<Func<T, object>> expression)
        {
            var mbody = expression.Body as MemberExpression;

            if (mbody != null) return mbody.Member;
            //This will handle Nullable<T> properties.
            var ubody = expression.Body as UnaryExpression;
            if (ubody != null)
            {
                mbody = ubody.Operand as MemberExpression;
            }
            if (mbody == null)
            {
                throw new ArgumentException("Expression is not a MemberExpression", "expression");
            }
            return mbody.Member;
        }

        /// <summary>
        ///
        /// </summary>
        /// <typeparam name="T"></typeparam>
        /// <param name="expression"></param>
        /// <returns></returns>
        public static string PropertyName<T>(this Expression<Func<T, object>> expression)
        {
            return GetMember(expression).Name;
        }

        /// <summary>
        ///
        /// </summary>
```

```
        /// <typeparam name="T"></typeparam>
        /// <param name="expression"></param>
        /// <returns></returns>
        public static string PropertyDisplay<T>(this Expression<Func<T, object>> expression)
        {
            var propertyMember = GetMember(expression);
            var displayAttributes = propertyMember.GetCustomAttributes(typeof(DisplayNameAttribute),
true);
            return displayAttributes.Length == 1 ?
((DisplayNameAttribute)displayAttributes[0]).DisplayName : propertyMember.Name;
        }
    }
```

```
public static string PropertyName<T>(this Expression<Func<T, object>> expression)
```

جهت بدست آوردن نام متغییر هایمان استفاده مینماییم.

```
public static string PropertyDisplay<T>(this Expression<Func<T, object>> expression)
```

جهت بدست آوردن DisplayNameAttribute استفده میشود. درصورتیکه این DisplayNameAttribute یافت نشود نام متغییر بازگشت داده میشود.

بنابراین View مربوطه را اینگونه بازنویسی میکنیم:

```
@using Models
@{
    const string viewTitle = "شرکت ها";
    ViewBag.Title = viewTitle;
    const string gridName = "companies-grid";
}
<div class="col-md-12">
    <div class="form-panel">
        <header>
            <div class="title">
                <i class="fa fa-book"></i>
                @viewTitle
            </div>
        </header>
        <div class="panel-body">
            <div id="@gridName">
            </div>
        </div>
    </div>
</div>
</div>
@section scripts
{
    <script type="text/javascript">
        $(document).ready(function () {
            $("#@gridName").kendoGrid({
                dataSource: {
                    type: "json",
                    transport: {
                        read: {
                            url: "@Html.Raw(Url.Action(MVC.Company.CompanyList()))",
                            type: "POST",
                            dataType: "json",
                            contentType: "application/json"
                        }
                    },
                    schema: {
                        data: "Data",
                        total: "Total",
                        errors: "Errors"
                        }
```

```
                    },
                    pageSize: 10,
                    serverPaging: true,
                    serverFiltering: true,
                    serverSorting: true
                },
                pageable: {
                    refresh: true
                },
                sortable: {
                    mode: "multiple",
                    allowUnsort: true
                },
                editable: false,
                filterable: false,
                scrollable: false,
                columns: [ {
                    field: "@(PropertyExtensions.PropertyName<CompanyModel>(a => a.CompanyName))",
                    title: "@(PropertyExtensions.PropertyDisplay<CompanyModel>(a => a.CompanyName))",
                    sortable: true,
                }, {
                    field: "@(PropertyExtensions.PropertyName<CompanyModel>(a => a.CompanyAbbr))",
                    title: "@(PropertyExtensions.PropertyDisplay<CompanyModel>(a => a.CompanyAbbr))",
                    sortable: true
                }]
            });
        });
    </script>
}
```

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۴/۱۶ ۱۲:۳۸

با تشکر از شما. حالت پیشرفته‌تر این مساله، کار با مدل‌های تو در تو هست. برای مثال:

```
public class CompanyModel
    {
        public int Id { get; set; }
        public string CompanyName { get; set; }
        public string CompanyAbbr { get; set; }

        public Product Product { set; get; }
    }

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

در اینجا اگر بخواهیم Product.Id را بررسی کنیم:

```
var data = PropertyExtensions.PropertyName<CompanyModel>(x => x.Product.Id);
```

فقط Id آن دریافت می‌شود.
راه حلی که از کدهای EF برای این مساله استخراج شده به صورت زیر است (نمونه‌اش متد Include تو در تو بر روی چند خاصیت):

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;

namespace PropertyExtensionsApp
{
    public class PropertyHelper : ExpressionVisitor
    {
        private Stack<string> _stack;
        public string GetNestedPropertyPath(Expression expression)
        {
            _stack = new Stack<string>();
            Visit(expression);
            return _stack.Aggregate((s1, s2) => s1 + "." + s2);
        }

        protected override Expression VisitMember(MemberExpression expression)
        {
            if (_stack != null)
                _stack.Push(expression.Member.Name);
            return base.VisitMember(expression);
        }

        public string GetNestedPropertyName<TEntity>(Expression<Func<TEntity, object>> expression)
        {
            return GetNestedPropertyPath(expression);
        }
    }
}
```

در این حالت خواهیم داشت:

```
var name = new PropertyHelper().GetNestedPropertyName<CompanyModel>(x => x.Product.Id);
```

که خروجی Product.Id را بر می‌گرداند.

نویسنده: محسن موسوی
تاریخ: ۱۳۹۳/۰۷/۲۶ ۱۸:۸

در نهایت این متد به این شکل اصلاح شود:

```
/// <summary>
        ///
        /// </summary>
        /// <typeparam name="T"></typeparam>
        /// <param name="expression"></param>
        /// <returns></returns>
        public static string PropertyName<T>(this Expression<Func<T, object>> expression)
        {
            return new PropertyHelper().GetNestedPropertyName(expression);
        }
```

عنوان: **کدام سلسله متدها، متد جاری را فراخوانی کرده‌اند؟**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۷/۰۶ ۹:۴۰
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, Reflection, Profiler, StackTrace

یکی از نیازهای نوشتن یک برنامه‌ی پروفایلر، نمایش اطلاعات متدهایی است که سبب لاگ شدن اطلاعاتی شده‌اند. برای مثال در طراحی interceptorهای EF 6 به یک چنین متدهایی می‌رسیم:

```
public void ScalarExecuted(DbCommand command,
                                    DbCommandInterceptionContext<object> interceptionContext)
        {
        }
```

سؤال: در زمان اجرای ScalarExecuted دقیقا در کجا قرار داریم؟ چه متدی در برنامه، در کدام کلاس، سبب رسیدن به این نقطه شده‌است؟
تمام این اطلاعات را در زمان اجرا توسط کلاس StackTrace می‌توان بدست آورد:

```
public static string GetCallingMethodInfo()
        {
            var stackTrace = new StackTrace(true);
            var frameCount = stackTrace.FrameCount;

            var info = new StringBuilder();
            var prefix = "-- ";
            for (var i = frameCount - 1; i >= 0; i--)
            {
                var frame = stackTrace.GetFrame(i);
                var methodInfo = getStackFrameInfo(frame);
                if (string.IsNullOrWhiteSpace(methodInfo))
                    continue;

                info.AppendLine(prefix + methodInfo);
                prefix = "-" + prefix;
            }

            return info.ToString();
        }
```

ایجاد یک نمونه جدید از کلاس StackTrace با پارامتر true به این معنا است که می‌خواهیم اطلاعات فایل‌های متناظر را نیز در صورت وجود دریافت کنیم.
خاصیت stackTrace.FrameCount مشخص می‌کند که در زمان فراخوانی متد GetCallingMethodInfo که اکنون برای مثال درون متد ScalarExecuted قرار گرفته‌است، از چند سطح بالاتر این فراخوانی صورت گرفته‌است. سپس با استفاده از متد stackTrace.GetFrame می‌توان به اطلاعات هر سطح دسترسی یافت.
در هر StackFrame دریافتی، با فراخوانی stackFrame.GetMethod می‌توان نام متد فراخوان را بدست آورد. متد stackFrame.GetFileLineNumber دقیقا شماره سطری را که فراخوانی از آن صورت گرفته، بازگشت می‌دهد و stackFrame.GetFileName نیز نام فایل مرتبط را مشخص می‌کند.

یک نکته:
شرط عمل کردن متدهای stackFrame.GetFileName و stackFrame.GetFileLineNumber در زمان اجرا، وجود فایل PDB اسمبلی در حال بررسی است. بدون آن اطلاعات محل قرارگیری فایل سورس مرتبط و شماره سطر فراخوان، قابل دریافت نخواهند بود.

اکنون بر اساس این اطلاعات، متد getStackFrameInfo چنین پیاده سازی را خواهد داشت:

```
private static string getStackFrameInfo(StackFrame stackFrame)
        {
            if (stackFrame == null)
                return string.Empty;

            var method = stackFrame.GetMethod();
```

```
            if (method == null)
                return string.Empty;

            if (isFromCurrentAsm(method) || isMicrosoftType(method))
            {
                return string.Empty;
            }

            var methodSignature = method.ToString();
            var lineNumber = stackFrame.GetFileLineNumber();
            var filePath = stackFrame.GetFileName();

            var fileLine = string.Empty;
            if (!string.IsNullOrEmpty(filePath))
            {
                var fileName = Path.GetFileName(filePath);
                fileLine = string.Format("[File={0}, Line={1}]", fileName, lineNumber);
            }

            var methodSignatureFull = string.Format("{0} {1}", methodSignature, fileLine);
            return methodSignatureFull;
        }
```

و خروجی آن برای مثال چنین شکلی را خواهد داشت:

```
Void Main(System.String[]) [File=Program.cs, Line=28]
```

که وجود file و line آن تنها به دلیل وجود فایل PDB اسمبلی مورد بررسی است.

در اینجا خروجی نهایی متد GetCallingMethodInfo به شکل زیر است که در آن چند سطح فراخوانی را می‌توان مشاهده کرد:

```
-- Void Main(System.String[]) [File=Program.cs, Line=28]
--- Void disposedContext() [File=Program.cs, Line=76]
---- Void Opened(System.Data.Common.DbConnection,
System.Data.Entity.Infrastructure.Interception.DbConnectionInterceptionContext)
[File=DatabaseInterceptor.cs,Line=157]
```

جهت تعدیل خروجی متد GetCallingMethodInfo، عموما نیاز است مثلا از کلاس یا اسمبلی جاری صرفنظر کرد یا اسمبلی‌های مایکروسافت نیز در این بین شاید اهمیتی نداشته باشند و بیشتر هدف بررسی سورس‌های موجود است تا فراخوانی‌های داخلی یک اسمبلی ثالث:

```
private static bool isFromCurrentAsm(MethodBase method)
        {
            return method.ReflectedType == typeof(CallingMethod);
        }

        private static bool isMicrosoftType(MethodBase method)
        {
            if (method.ReflectedType == null)
                return false;

            return method.ReflectedType.FullName.StartsWith("System.") ||
                   method.ReflectedType.FullName.StartsWith("Microsoft.");
        }
```

کد کامل CallingMethod.cs را از اینجا می‌توانید دریافت کنید:

CallingMethod.cs

نظرات خوانندگان

نویسنده: علیرضا
تاریخ: ۱۳۹۳/۰۷/۱۰ ۲۳:۳۸

چه موقعی GetMethod میتواند Null برگرداند؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۷/۱۱ ۰:۳۷

زمانیکه کامپایلر مباحث inlining متدها را جهت بهینه سازی اعمال کند.