عنوان: **ایجاد یک کلاس جدید پویا و وهله‌ای از آن در زمان اجرا توسط Reflection.Emit**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۵/۱۴ ۲۱:۴۷
آدرس: www.dotnettips.info
برچسب‌ها: C#, CIL, CLR, IL, MSIL, Reflection

توانایی‌های Reflection.Emit صرفا به ایجاد متدهایی کاملا جدید و پویا در زمان اجرا محدود نمی‌شود. برای نمونه کلاس ذیل را درنظر بگیرید:

```
public class Person
    {
        private string _name;
        public string Name
        {
            get { return _name; }
        }

        public Person(string name)
        {
            _name = name;
        }
    }
```

در ادامه قصد داریم معادل این کلاس را به همراه وهله‌ای از آن، به صورتی کاملا پویا در زمان اجرا ایجاد کرده (تصور کنید این کلاس در برنامه وجود خارجی نداشته و تنها جهت درک بهتر کدهای IL ادامه بحث، معرفی گردیده است) و سپس مقداری را به سازنده آن ارسال کنیم.
کدهای کامل و توضیحات این typeBuilder را در ادامه ملاحظه می‌کنید:

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

namespace FastReflectionTests
{
    class Program
    {
        static void Main(string[] args)
        {
            //اسمبلی محل قرارگیری کدهای پویای نهایی در اینجا تعیین می‌شود//
            //حالت دسترسی به آن اجرایی درنظر گرفته شده، امکان تعیین حالت‌های دیگری مانند ذخیره سازی نیز
وجود دارد
            var assemblyBuilder = AppDomain.CurrentDomain.DefineDynamicAssembly(
                                    name: new AssemblyName("Demo"), access:
AssemblyBuilderAccess.Run);

            // اکنون داخل این اسمبلی یک ماژول جدید را برای قرار دادن کلاس جدید خود تعریف می‌کنیم
            var moduleBuilder = assemblyBuilder.DefineDynamicModule(name: "PersonModule");

            // کار ساخت نوع و کلاس جدید شخص عمومی از اینجا شروع می‌شود
            var typeBuilder = moduleBuilder.DefineType(name: "Person", attr: TypeAttributes.Public);

            // افزودن فیلد خصوصی نام تعریف شده در سطح کلاس شخص
            var nameField = typeBuilder.DefineField(fieldName: "_name",
                                                    type: typeof(string),
                                                    attributes: FieldAttributes.Private);

            // تعریف سازنده عمومی کلاس شخص که دارای یک آرگومان رشته‌ای است
            var ctor = typeBuilder.DefineConstructor(
                                    attributes: MethodAttributes.Public,
                                    callingConvention: CallingConventions.Standard,
                                    parameterTypes: new[] { typeof(string) });
            // تعریف بدنه سازنده کلاس شخص
            // در اینجا فیلد خصوصی تعریف شده در سطح کلاس باید مقدار دهی شود
            var ctorIL = ctor.GetILGenerator();
            // نکته‌ای در مورد سازنده‌ها
            ctorIL.Emit(OpCodes.Ldarg_0); // اندیس صفر در سازنده کلاس به وهله‌ای از کلاس جاری اشاره
می‌کند
            ctorIL.Emit(OpCodes.Ldarg_1); // بارگذاری آرگومان سازنده و قرار دادن آن روی پشته
            // مقدار دهی فیلد خصوصی نام که به وهله‌ای از کلاس جاری و مقدار آرگومان دریافتی نیاز دارد
            ctorIL.Emit(OpCodes.Stfld, nameField);
            ctorIL.Emit(OpCodes.Ret); // پایان کار سازنده

            // تعریف خاصیت رشته‌ای نام در کلاس شخص
```

```
            var nameProperty = typeBuilder.DefineProperty(
                                            name: "Name",
                                            attributes: PropertyAttributes.HasDefault,
                                            returnType: typeof(string),
                                            parameterTypes: null); // خاصیت پارامتر ورودی ندارد

            var namePropertyGetMethod = typeBuilder.DefineMethod(
                                            name: "get_Name",
                                            attributes: MethodAttributes.Public |
                //متد ویژه‌ای است که توسط کامپایلر پردازش و تشخیص داده می‌شود
                                                        MethodAttributes.SpecialName |
                                                        MethodAttributes.HideBySig,
                                            returnType: typeof(string),
                                            parameterTypes: Type.EmptyTypes);
            // اتصال گت متد به خاصیت رشته‌ای نام که پیشتر تعریف شد
            nameProperty.SetGetMethod(namePropertyGetMethod);

            // بدنه گت متد در اینجا تعریف خواهد شد
            var namePropertyGetMethodIL = namePropertyGetMethod.GetILGenerator();
            namePropertyGetMethodIL.Emit(OpCodes.Ldarg_0); // بارگذاری اشاره‌گری به وهله‌ای از کلاس جاری
در پشته
            namePropertyGetMethodIL.Emit(OpCodes.Ldfld, nameField); // بارگذاری فیلد نام
            namePropertyGetMethodIL.Emit(OpCodes.Ret);

            var t = typeBuilder.CreateType(); // نهایی سازی کار ایجاد نوع جدید

            // ایجاد وهله‌ای از نوع جدید که پارامتری رشته‌ای به سازنده آن ارسال می‌شود
            var instance = Activator.CreateInstance(t, "Vahid");

            // دسترسی به خاصیت نام
            var nProperty = t.GetProperty("Name");
            // و دریافت مقدار آن برای نمایش
            var result = nProperty.GetValue(instance, null);

            Console.WriteLine(result);
        }
    }
}
```

در اینجا ایجاد یک کلاس جدید با ایجاد یک TypeBuilder واقع در فضای نام System.Reflection.Emit آغاز می‌شود. پیش از آن نیاز است یک اسمبلی پویا و ماژولی در آن‌را برای قرار دادن کدهای پویای این TypeBuilder ایجاد کنیم. توضیحات مرتبط با دستورات مختلف را به صورت کامنت در کدهای فوق ملاحظه می‌کنید. با استفاده از TypeBuilder و متد DefineField آن می‌توان یک فیلد در سطح کلاس ایجاد کرد و یا توسط متد DefineConstructor آن، سازنده کلاس را با امضایی ویژه تعریف نمود و سپس با دسترسی به ILGenerator آن، بدنه این سازنده را همانند متدهای پویا ایجاد کرد.

اگر به کدهای فوق دقت کرده باشید، متد get_Name به خاصیت Name انتساب داده شده است. علت را در قسمت معرفی اجمالی Reflection زمانیکه لیست متدهای کلاس Person را نمایش دادیم، ملاحظه کرده‌اید. تمام خواص Auto implemented در دات نت، هر چند ظاهر ساده‌ای دارند اما در عمل به دو متد get_Name و set_Name در کدهای IL توسط کامپایلر تبدیل می‌شوند. به همین جهت در اینجا نیاز بود تا get_Name را نیز تعریف کنیم.

چند مثال تکمیلی

Populating a PropertyGrid using Reflection.Emit

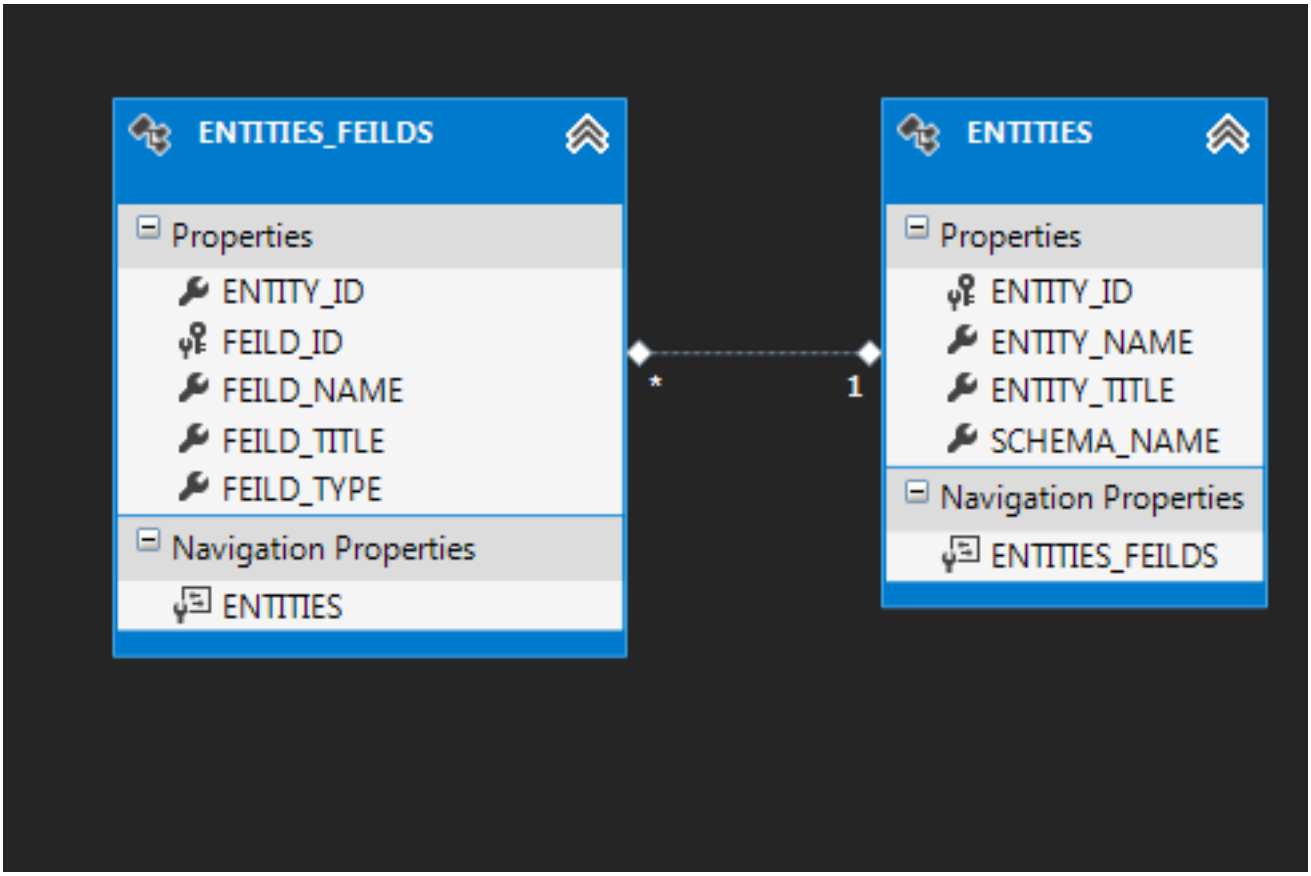
Dynamically adding RaisePropertyChanged to MVVM Light ViewModels using Reflection.Emit

نظرات خوانندگان

نویسنده: پویا امینی
تاریخ: ۱۳۹۲/۰۵/۲۳ ۱۴:۳۴

با سلام، من می‌خواهم در یکی از پروژه‌ها از این روش استفاده کنم و سناریویی که من روی اون کار می‌کنم به صورت زیر است:

درون دیتابیس، یک Table دارم که درون این Table نام تمام موجودیت‌های سیستم خودم رو نگه می‌دارم و در یک Table دیگر تمام فیلدهای موجودیت‌ها را همراه با نوع داده آنها ذخیره می‌کنم



برای یک سری شرایط خاص می‌خواهم کار زیر را انجام دهم:

یک فرم طراحی کردم که برای تمام موجودیت‌های تعریف شده درون جدول Entities کاربرد داره ، می‌خواهم زمانی که این فرم اجرا شده با توجه به اینکه این فرم برای کدام موجودیت فراخوانی شده است یک کلاس برای آن موجودیت ایجاد کنم و پس از آن یک لیست از کلاسی که ایجاد شده ، ایجاد بکنم و درون آن لیست مقادیری را قرار دهم (مقادیر را از دیتابیس خوانده می‌شود) و در آخر مقادیر لیست را در یک کنترل مثل gridview نمایش دهم

حال من برای انجام این کار به چند مشکل برخوردم . کدی که نوشتم به صورت زیر است

```
var ctx = new Entities();
            var Fields = ctx.ENTITIES_FEILDS.ToList();
            var assemblyBuilder = AppDomain.CurrentDomain.DefineDynamicAssembly(
                                    name: new AssemblyName("Demo"), access: AssemblyBuilderAccess.Run);

            var moduleBuilder = assemblyBuilder.DefineDynamicModule(name: "Module");

            var typeBuilder = moduleBuilder.DefineType(name:
```

```
Fields.First(c=>c.FEILD_ID==1).ENTITIES.ENTITY_NAME, attr: TypeAttributes.Public);

            foreach (var item in Fields)
            {

            }
```

در اینجا یک کلاس همنام با نام موجودیت ایجاد کردم و تمام فیلدهای این موجودیت را واکشی کردم حال می‌خواهم به ازای هر فیلد، یک Property ایجاد کنم. با توجه به مطلبی که در بالا فرمودید اگر ما تعداد فیلدهامون از قبل مشخص بود به راحتی می‌توانستیم این کار رو انجام بدیم ولی الان که مشخص نیست چگونه می‌توانیم Property خودمان را اضافه کنم؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۵/۲۳ ۱۴:۵۶

- در متن فوق جایی عنوان نشده که تنها اگر تعداد فیلدها از قبل مشخص بود، اینکار قابل انجام است. همچنین اگر به مثال بحث دقت کنید، پارامتر name رشته‌ای است. یعنی هر نام خاصیت دلخواهی قابل تعریف است. نوع آن نیز قابل مقدار دهی و تغییر است.
- در حلقه‌ای که نوشتید، کدهای «افزودن فیلد خصوصی» مثال بحث، «تعریف خاصیت رشته‌ای نام» ، «اتصال گت متد به خاصیت رشته‌ای نام» و «تعریف بدنه گت متد» باید به ازای هر خاصیت، تکرار شوند (پارامتر name را با نام خاصیت‌ها جایگزین کنید؛ نوع آن هم قابل تغییر است). اگر set هم دارد، علاوه بر متد گت، متد set_XYZ هم باید اضافه شود و روش کار یکی است.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۵/۲۳ ۱۵:۸

برای ساده سازی و همچنین کپسوله کردن این عملیات، مراجعه کنید به مطالب زیر. در اینجا یک ClassGenerator با استفاده از Reflection Emit تهیه کرده‌اند:

Power of Reflection Emit

How to create a class with properties at run time

نویسنده: پویا امینی
تاریخ: ۱۳۹۲/۰۵/۲۳ ۱۵:۵۵

من کد رو به صورت زیر تغییر دادم

```
var ctx = new Entities();
            var Fields = ctx.ENTITIES_FEILDS.ToList();
            var assemblyBuilder = AppDomain.CurrentDomain.DefineDynamicAssembly(
                                    name: new AssemblyName("Demo"), access: AssemblyBuilderAccess.Run);

            var moduleBuilder = assemblyBuilder.DefineDynamicModule(name: "Module");

            var typeBuilder = moduleBuilder.DefineType(name: Fields.First(c => c.FEILD_ID ==
1).ENTITIES.ENTITY_NAME, attr: TypeAttributes.Public);

            foreach (var item in Fields)
            {
                switch (item.FEILD_TYPE)
                {
                    case 0://int
                        {
                            var intField = typeBuilder.DefineField(fieldName: string.Format("_{0}",
item.FEILD_NAME), type: typeof(string),
                                                   attributes: FieldAttributes.Private);

                            var intProperty = typeBuilder.DefineProperty(
                                                name: item.FEILD_NAME,
                                                attributes: PropertyAttributes.HasDefault,
                                                returnType: typeof(string),
                                                parameterTypes: null); // خاصیت پارامتر ورودی ندارد

                            //تعریف گت
                            var intpropertyGetMethod = typeBuilder.DefineMethod(
                                                               name: string.Format("get_{0}",
item.FEILD_NAME),
```

```
                                                    attributes: MethodAttributes.Public |
                           MethodAttributes.SpecialName |
                                                                MethodAttributes.HideBySig,
                                                    returnType: typeof(string),
                                                    parameterTypes: Type.EmptyTypes);

                                // اتصال گت متد به خاصیت عددی
                                intProperty.SetGetMethod(intpropertyGetMethod);


                                //تعریف ست
                                var propertySetMethod =
                                    typeBuilder.DefineMethod(name: string.Format("set_{0}",
item.FEILD_NAME),
                                    attributes: MethodAttributes.Public |
                                   MethodAttributes.SpecialName |
                                                                MethodAttributes.HideBySig,
                                                      returnType: typeof(void),
                                                    parameterTypes: new[] { typeof(string)
});
                                //اتصال ست متد
                                intProperty.SetSetMethod(propertySetMethod);


                                // بدنه گت متد در اینجا تعریف خواهد شد
                                var propertyGetMethodIL = intpropertyGetMethod.GetILGenerator();
                                propertyGetMethodIL.Emit(OpCodes.Ldarg_0); // بارگذاری اشاره‌گری به وهله‌ای از
کلاس جاری در پشته
                                propertyGetMethodIL.Emit(OpCodes.Ldfld, intField); // بارگذاری فیلد نام
                                propertyGetMethodIL.Emit(OpCodes.Ret);

                                //بدنه ست متد در اینجا تعریف شده است
                                var propertySetIL = propertySetMethod.GetILGenerator();
                                propertySetIL.Emit(OpCodes.Ldarg_0);
                                propertySetIL.Emit(OpCodes.Ldarg_1);
                                propertySetIL.Emit(OpCodes.Stfld, intField);
                                propertySetIL.Emit(OpCodes.Ret);

                            }

                            break;
                        case 1://string
                            {

                            } break;
                    }

                }
                var t = typeBuilder.CreateType();



                var instance = Activator.CreateInstance(t);

                var type = instance.GetType();

                //تغییر مقدار یک خاصیت
                var setNameMethod = type.GetMethod("set_CoOrder");
                setNameMethod.Invoke(obj: instance, parameters: new[] {"1"});


                // دسترسی به خاصیت نام
                var nProperty = t.GetProperty("CoOrder");
                // و دریافت مقدار آن برای نمایش
                var result = nProperty.GetValue(instance, null);

                Console.WriteLine(result);




                Console.ReadKey();
```

تا اینجا درست کار می‌کنه حال می‌خواهم از کلاسی که برای من ایجاد می‌کند یک لیست ایجاد کنم و بتونم بهش مقدار بدم. ولی هر چی تلاش کردم نتونستم کلاس خودم رو ایجاد کنم. ممنون میشم راهنمایی کنید

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۵/۲۳ ۱۷:۴۹

الان لیست رو به صورت زیر ایجاد کنید

```
List<object> items = new List<object>();
```

هر آیتم این لیست، یک وهله از شیء پویایی خواهد بود که تهیه کردید.
گرید شما هم اگر حالت auto generate columns را پشتیبانی کند، بدون مشکل کار خواهد کرد.

نویسنده: پویا امینی
تاریخ: ۱۳۹۲/۰۵/۲۴ ۱۳:۵۱

یعنی اصلاً نمی‌توانم یه لیست از نوع کلاس ایجاد شده، تعریف کنم؟ چون گرید من auto generate columns نیست و من به صورت داینامیکcolumnها را مشخص می‌کنم

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۵/۲۴ ۱۴:۱۳

- وهله شیء تولیدی شما از نوع object است. آن‌را به لیست اضافه کنید و استفاده نمائید.
+ نوع جنریک در دات نت پویا نیست و نمی‌شود آن را به صورت یک متغیر تعریف کرد. مثلا حالت زیر مجاز نیست:

```
var myType = typeof(something);
List<myType> list = new List<myType>();
```

علت هم این است که هدف از نوع جنریک، compile time safety است و زمانیکه نوع در زمان کامپایل مشخص نباشد، این مساله قابل حصول نخواهد بود. تنها حالت پویای آن استفاده از نوع object است.
- البته می‌شود با استفاده Reflection نوع جنریک را به صورت متغیر تعریف کنید .