عنوان: **ساده سازی و بالا بردن سرعت عملیات Reflection با استفاده از Dynamic Proxy**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۳/۰۹ ۱۲:۴۰
آدرس: www.dotnettips.info
گروه‌ها: Reflection, Performance, AOP

فرض کنید یک چنین کلاسی طراحی شده‌است:

```
public class NestedClass
{
    private int _field2;
    public NestedClass()
    {
        _field2 = 12;
    }
}

public class MyClass
{
    private int _field1;
    private NestedClass _nestedClass;

    public MyClass()
    {
        _field1 = 1;
        _nestedClass = new NestedClass();
    }

    private string GetData()
    {
        return "Test";
    }
}
```

می‌خواهیم از طریق Reflection مقادیر فیلدها و متدهای مخفی آن‌را بخوانیم.
حالت متداول دسترسی به فیلد خصوصی آن از طریق Reflection، یک چنین شکلی را دارد:

```
var myClass = new MyClass();

var field1Obj = myClass.GetType().GetField("_field1", BindingFlags.NonPublic | BindingFlags.Instance);
if (field1Obj != null)
{
    Console.WriteLine(Convert.ToInt32(field1Obj.GetValue(myClass)));
}
```

و یا دسترسی به مقدار خروجی متد خصوصی آن، به نحو زیر است:

```
var getDataMethod = myClass.GetType().GetMethod("GetData", BindingFlags.NonPublic |
BindingFlags.Instance);
if (getDataMethod != null)
{
    Console.WriteLine(getDataMethod.Invoke(myClass, null));
}
```

در اینجا دسترسی به مقدار فیلد مخفی NestedClass، شامل مراحل زیر است:

```
var nestedClassObj = myClass.GetType().GetField("_nestedClass", BindingFlags.NonPublic |
BindingFlags.Instance);
if (nestedClassObj != null)
{
    var nestedClassFieldValue = nestedClassObj.GetValue(myClass);
    var field2Obj = nestedClassFieldValue.GetType()
        .GetField("_field2", BindingFlags.NonPublic | BindingFlags.Instance);
    if (field2Obj != null)
    {
        Console.WriteLine(Convert.ToInt32(field2Obj.GetValue(nestedClassFieldValue)));
    }
}
```

البته این مقدار کد فقط برای دسترسی به دو سطح تو در تو بود.

چقدر خوب بود اگر می‌شد بجای این همه کد، نوشت:

```
myClass._field1
myClass._nestedClass._field2
myClass.GetData()
```

نه؟!
برای این مشکل راه حلی معرفی شده‌است به نام Dynamic Proxy که در ادامه به معرفی آن خواهیم پرداخت.

معرفی Dynamic Proxy

Dynamic Proxy یکی از مفاهیم AOP است. به این معنا که توسط آن یک محصور کننده‌ی نامرئی، اطراف یک شیء تشکیل خواهد شد. از این غشای نامرئی عموما جهت مباحث ردیابی اطلاعات، مانند پروکسی‌های Entity framework، همانجایی که تشخیص می‌دهد کدام خاصیت به روز شده‌است یا خیر، استفاده می‌شود و یا این غشای نامرئی کمک می‌کند که در حین دسترسی به خاصیت یا متدی، بتوان منطق خاصی را در این بین تزریق کرد. برای مثال فرآیند تکراری logging سیستم را به این غشای نامرئی منتقل کرد و به این ترتیب می‌توان به کدهای تمیزتری رسید.
یکی دیگر از کاربردهای این محصور کننده یا غشای نامرئی، ساده سازی مباحث Reflection است که نمونه‌ای از آن در پروژه‌ی EntityFramework.Extended بکار رفته‌است.
در اینجا، کار با محصور سازی نمونه‌ای از کلاس مورد نظر با Dynamic Proxy شروع می‌شود. سپس کل عملیات Reflection فوق در همین چند سطر ذیل به نحوی کاملا عادی و طبیعی قابل انجام است:

```
// Accessing a private field
dynamic myClassProxy = new DynamicProxy(myClass);
dynamic field1 = myClassProxy._field1;
Console.WriteLine((int)field1);

// Accessing a nested private field
dynamic field2 = myClassProxy._nestedClass._field2;
Console.WriteLine((int)field2);

// Accessing a private method
dynamic data = myClassProxy.GetData();
Console.WriteLine((string)data);
```

خروجی Dynamic Proxy از نوع dynamic دات نت 4 است. پس از آن می‌توان در اینجا هر نوع خاصیت یا متد دلخواهی را به شکل dynamic تعریف کرد و سپس به مقادیر آن‌ها دسترسی داشت.

بنابراین با استفاده از Dynamic Proxy فوق می‌توان به دو مهم دست یافت:
1) ساده سازی و زیبا سازی کدهای کار با Reflection
2) استفاده‌ی ضمنی از مباحث Fast Reflection . در کتابخانه‌ی Dynamic Proxy معرفی شده، دسترسی به خواص و متدها، توسط کدهای IL بهینه سازی شده‌است و در دفعات آتی کار با آن‌ها، دیگر شاهد سربار بالای Reflection نخواهیم بود.

کدهای کامل این مثال را از اینجا می‌توانید دریافت کنید:
DynamicProxyTests.zip