عنوان: **آشنایی با Fluent interfaces**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۳/۰۴ ۱۳:۲۱:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: Design patterns

تعریف مقدماتی fluent interface در ویکی پدیا به شرح زیر است: (+)

In software engineering, a fluent interface (as first coined by Eric Evans and Martin Fowler) is a way of implementing an object oriented API in a way that aims to provide for more readable code.

به صورت خلاصه هدف آن فراهم آوردن روشی است که بتوان متدها را زنجیر وار فراخوانی کرد و به این ترتیب خوانایی کد نوشته شده را بالا برد. پیاده سازی آن هم شامل دو نکته است:
الف) نوع متد تعریف شده باید مساوی با نام کلاس جاری باشد.
ب) در این حالت خروجی متدهای ما کلمه کلیدی this خواهند بود.

برای مثال:

```
using System;

namespace FluentInt
{
 public class FluentApiTest
 {
   private int _val;

   public FluentApiTest Number(int val)
   {
       _val = val;
       return this;
   }

   public FluentApiTest Abs()
   {
       _val = Math.Abs(_val);
       return this;
   }

   public bool IsEqualTo(int val)
   {
       return val == _val;
   }
 }
}
```

مثالی هم از استفاده‌ی آن به صورت زیر می‌تواند باشد:

```
if (new FluentApiTest().Number(-10).Abs().IsEqualTo(10))
{
 Console.WriteLine("Abs(-10)==10");
}
```

که در آن توانستیم تمام متدها را زنجیر وار و با خوانایی خوبی شبیه به نوشتن جملات انگلیسی در کنار هم قرار دهیم.
خوب! این مطلبی است که همه جا پیدا می‌کنید و مطلب جدیدی هم نیست. اما موردی را که سخت می‌شود یافت این است که طراحی کلاس فوق ایراد دارد. برای مثال شما می‌توانید ترکیب‌های زیر را هم تشکیل دهید و کار می‌کند؛ یا به عبارتی برنامه کامپایل می‌شود و این خوب نیست:

```
if(new FluentApiTest().Abs().Number(-10).IsEqualTo(10)) ...
if (new FluentApiTest().Abs().IsEqualTo(10)) ...
```

می‌شود در کدهای برنامه یک سری throw new exception را هم قرار داد که ... هی! اول باید اون رو فراخوانی کنی بعد این رو! ولی ... این روش هم صحیح نیست. از ابتدای کار نباید بتوان متد بی‌ربطی را در طول این زنجیره مشاهده کرد. اگر قرار نیست استفاده گردد، نباید هم در intellisense ظاهر شود و پس از آن هم نباید قابل کامپایل باشد.

بنابراین صورت مساله به این ترتیب اصلاح می‌شود:
می‌خواهیم پس از نوشتن FluentApiTest و قرار دادن یک نقطه، در intellisense فقط Number ظاهر شود و نه هیچ متد دیگری. پس از ذکر متد Number فقط متد Abs یا مواردی شبیه به آن مانند Sqrt ظاهر شوند. پس از انتخاب مثلا Abs آنگاه متد IsEqualTo توسط Intellisense قابل دسترسی باشد. در روش اول فوق، به صورت دوستانه همه چیز در دسترس است و هر ترکیب قابل کامپایلی را می‌شود با متدها ساخت که این مورد نظر ما نیست.
اینبار پیاده سازی اولیه به شرح زیر تغییر خواهد کرد:

```
using System;

namespace FluentInt
{
 public class FluentApiTest
 {
   public MathMethods<FluentApiTest> Number(int val)
   {
       return new MathMethods<FluentApiTest>(this, val);
   }
 }

 public class MathMethods<TParent>
 {
   private int _val;
   private readonly TParent _parent;

   public MathMethods(TParent parent, int val)
   {
       _val = val;
       _parent = parent;
   }

   public Restrictions<MathMethods<TParent>> Abs()
   {
       _val = Math.Abs(_val);
       return new Restrictions<MathMethods<TParent>>(this, _val);
   }
 }

 public class Restrictions<TParent>
 {
   private readonly int _val;
   private readonly TParent _parent;

   public Restrictions(TParent parent, int val)
   {
       _val = val;
       _parent = parent;
   }

   public bool IsEqualTo(int val)
   {
       return _val == val;
   }
 }
}
```

در اینجا هم به همان کاربرد اولیه می‌رسیم:

```
if (new FluentApiTest().Number(-10).Abs().IsEqualTo(10))
{
 Console.WriteLine("Abs(-10)==10");
}
```

با این تفاوت که intellisense هربار فقط یک متد مرتبط در طول زنجیره را نمایش می‌دهد و تمام متدها در همان ابتدای کار قابل انتخاب نیستند.

در پیاده سازی کلاس MathMethods از Generics استفاده شده به این جهت که بتوان نوع متد Number را بر همین اساس تعیین کرد تا متدهای کلاس MathMethods در Intellisense (یا به قولی در طول زنجیره مورد نظر) ظاهر شوند. کلاس Restrictions نیز به همین ترتیب معرفی شده است و از آن جهت تعریف نوع متد Abs استفاده کردیم. هر کلاس جدید در طول زنجیره، توسط سازنده خود به وهله‌ای از کلاس قبلی به همراه مقادیر پاس شده دسترسی خواهد داشت. به این ترتیب زنجیره‌ای را تشکیل داده‌ایم که سازمان یافته است و نمی‌توان در آن متدی را بی‌جهت پیش یا پس از دیگری صدا زد و همچنین دیگر نیازی به بررسی نحوه‌ی فراخوانی‌های یک مصرف کننده نیز نخواهد بود زیرا برنامه کامپایل نمی‌شود.

نظرات خوانندگان

نویسنده: Saber Soleymani
تاریخ: ۱۳۹۰/۰۳/۰۴ ۱۵:۰۰:۳۲

مقاله خوبی بود. اما درست است که استفاده از فراخوانی زنجیره‌ای با این روش ساده‌تر و ایمن‌تر می‌شود، اما خوانایی کد را (در مقایسه با روش اول) پایین‌تر می‌آورد.
در کل روش جالبی بود.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۳/۰۴ ۱۶:۲۱:۱۹

بله روش دوم ساده نیست اما نتیجه نهایی آن برای کسی که قرار است از API شما استفاده کند یکی است و به همان اندازه ساده. در کل طراحی API خوب کار مشکلی است. برای نمونه ما از LINQ لذت می‌بریم (به عنوان استفاده کننده نهایی) ولی واقعا پیاده سازی اون مشکل بوده و پشت صحنه ساده‌ای نداره.

نویسنده: Alidoosti
تاریخ: ۱۳۹۰/۰۳/۰۵ ۱۴:۲۴:۲۳

Constructor کلاس Restrictions به صورت (Public Restriction(MathMethods parent, int valنیز درست است ؟؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۳/۰۶ ۰۲:۱۸:۵۴

خیر؛ یک پارامتر جنریک به عنوان ورودی یک پارامتر جنریک قابل تعریف نیست.

پ.ن.
بلاگر در ارسال یک سری کاراکترها حساسیت دارد حتی در کامنت‌ها. همان کاراکترهای غیرمجاز در XML که باید به صورت escape شده معرفی شوند و گرنه خیلی ساده یا آن‌ها را نمایش نمی‌دهد یا حذف می‌کند (مشکلی که به نظر با تعریف جنریک‌ها داشتید و کاراکترها حدف شده بود؛ علتش این مبحث است).

نویسنده: امیرحسین
تاریخ: ۱۳۹۱/۰۶/۰۶ ۳:۴۵

سلام
به نظر من این امکانات فقط برای پروژه‌های مانند LINQ خوبه و واقعا در محیطهای واقعی قابل پیاده سازی نیست.
چون این کار خودش نیاز به تحلیل جدا داره ... و پیاده ساز ی اون هم وقت گیره.
مگر در مواردی که قرار لایه بندی برنامه در بالاترین صطح و کیفیت قرار داشته باشه ، که باز هم بعید می‌دونم در این حد نیاز باشه.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۰۶ ۹:۱۱

- نمونه پیاده سازی شده اون رو در پروژه نسبتا بزرگ fluent nhibernate می‌تونید مشاهده کنید.
- پروژه بزرگ دیگری که از این روش استفاده می‌کنه ASP.NET MVC Extensions شرکت telerik است (برای طراحی API نهایی قابل استفاده از آن).
- همچنین اکثر افزونه‌ها و کتابخانه‌های کمکی طراحی شده برای ASP.NET MVC از روش Fluent interfaces استفاده می‌کنند. مثلا fluent validation ، fluent security و غیره.
- اخیرا هم اعضای تیم Entity framework، قسمتی از کار تنظیم نگاشت‌ها را توسط روشی به نام Fluent API طراحی کرده‌اند(در

.(EF Code first

نویسنده: حسام
تاریخ: ۱۳۹۱/۰۶/۰۶ ۱۰:۳۴

در این روش ضرورت استفاده از genericها چیست ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۰۶ ۱۰:۳۷

لطفا در مطلب فوق از قسمت «بنابراین صورت مساله به این ترتیب اصلاح می‌شود» را مطالعه کنید. هدف مقید کردن استفاده کننده از API به انتخاب متدهایی خاص است و نه هر متد ممکنی در طول یک زنجیره.