عنوان: **قراردادن کلاس های Metdata در اسمبلی جداگانه برای کار با WCF Ria Service توسط FluentMetadata**
نویسنده: ایلیا اکبری فرد
تاریخ: ۱۳۹۱/۰۴/۲۳ ۲۳:۳۰
آدرس: www.dotnettips.info
برچسب‌ها: WCF RIA Services, FluentMetadata, Domain Service

اصلی‌ترین مزیت این پکیج ،امکان جداکردن dataModel و Metadata در پروژه یا اسمبلی جداگانه است . در حالیکه WCF RIA Service استاندارد فاقد این قابلیت میباشد و باید dataModel و Metadata در یک پروژه و در یک namespace تعریف شوند.
برای استفاده از FluentMetadata :
1) ابتدا فرض می‌کنیم که کلاس Product ما در یک اسمبلی دیگر به نام DataModel تعریف شده است ، بصورت زیر :

```
public class Product
{
        public int ProductId { get; set; }
        public string ProductName { get; set; }
        public long ProductPrice { get; set; }
}
```

2) حال یک پروژه جدید به نام DataModelsMetadata تعریف می‌کنیم و ارجاعی به اسمبلی بالا یعنی DataModel نیز به آن اضافه می‌کنیم .
2-1) ابتدا باید پکیج FluentMetadata را توسط Nuget نصب کرد. راهنمای نصب
2-2) سپس کلاس‌های Metadata موردنظر خود را برای کلاس Product تعریف میکنیم .

```
public class ProductMetadata : MetadataClass<Product>
{
    public ProductMetadata ()
    {
        this.Validation(x => x.ProductName).Required("عنوان محصول وارد نشده است");
        this.Validation(x => x.ProductPrice).Range(1000,100000,"قیمت محصول باید بین هزار تومان تا
صدهزار تومان باشد");
    }
}
```

2-3) سپس یک کلاس MetadataConfiguration که برای نمونه سازی تمام کلاس‌های متادیتا ایجاد می‌کنیم.

```
public class FluentMetadataConfiguration : IFluentMetadataConfiguration
{
    public void OnTypeCreation(MetadataContainer metadataContainer)
    {
        metadataContainer.Add(new ProductMetadata());
    }
}
```

2-4) در آخر اضافه کردن MetadataConfiguration ایجاد شده به Domain Service توسط ویژگی FluentMetadata.

```
[EnableClientAccess()]
[FluentMetadata(typeof(FluentMetadataConfiguration))]
public class FluentMetadataTestDomainService : DomainService
{
    ...
}
```

منابع : http://www.nikhilk.net/RIA-Services-Fluent-Metadata-API.aspx http://forums.silverlight.net/t/242023.aspx/1

عنوان: **الگوی طراحی Factory Method به همراه مثال**
نویسنده: مجتبی شاطری
تاریخ: ۱۳۹۲/۰۶/۲۸ ۰:۰
آدرس: www.dotnettips.info
برچسب‌ها: Design patterns, Domain Service, Domain Driven Design, SOLID Principals

الگوی طراحی Factory Method به همراه مثال

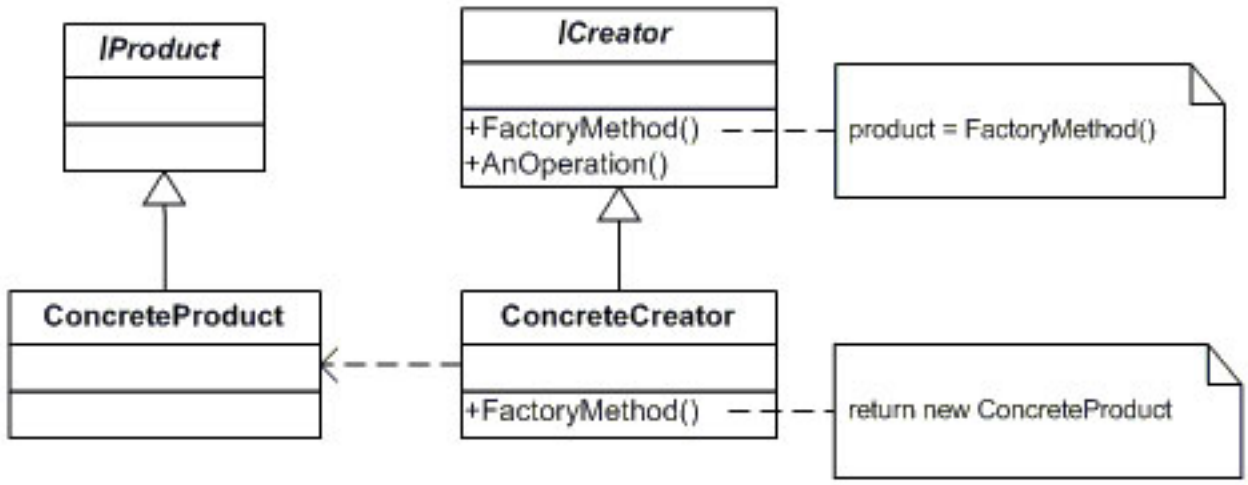
تعریف الگوی Factory Method :

این الگو پیچیدگی ایجاد اشیاء برای استفاده کننده را پنهان می‌کند. ما با این الگو میتوانیم بدون اینکه کلاس دقیق یک شیئ را مشخص کنیم آن را ایجاد و از آن استفاده کنیم. کلاینت (استفاده کننده) معمولا شیئ واقعی را ایجاد نمی‌کند بلکه با یک واسط و یا کلاس انتزاعی (Abstract) در ارتباط است و کل مسئولیت ایجاد کلاس واقعی را به Factory Method می‌سپارد. کلاس Factory Method می‌تواند استاتیک باشد . کلاینت معمولا اطلاعاتی را به متدی استاتیک از این کلاس می‌فرستد و این متد بر اساس آن اطلاعات تصمیم می‌گیرید که کدام یک از پیاده سازی‌ها را برای کلاینت برگرداند.

از مزایای این الگو این است که اگر در نحوه ایجاد اشیاء تغییری رخ دهد هیچ نیازی به تغییر در کد کلاینت‌ها نخواهد بود. در این الگو اصل DIP از اصول پنجگانه SOLID به خوبی رعایت می‌شود چون که مسئولیت ایجاد زیرکلاس‌ها از دوش کلاینت برداشته می‌شود.

دیاگرام UML :

در شکل زیر دیاگرام UML الگوی Factory Method را مشاهده می‌کنید.



شرکت کنندگان در این الگو به شرح زیل هستند :

- Iproduct یک واسط است که هر کلاینت از آن استفاده می‌کند. در اینجا کلاینت استفاده کننده نهایی است مثلا می‌تواند متد

main یا هر متدی در کلاسی خارج از این الگو باشد. ما می‌توانیم پیاده سازی‌های مختلفی بر حسب نیاز از واسط Iproduct ایجاد کنیم.

- ConcreteProduct یک پیاده سازی از واسط Iproduct است ، برای این کار بایستی کلاس پیاده سازی (ConcreteProduct) از این واسط (IProduct) مشتق شود.

- Icreator واسطیست که Factory Method را تعریف می‌کند. پیاده ساز این واسط بر اساس اطلاعاتی دریافتی کلاس صحیح را ایجاد می‌کند. این اطلاعات از طریق پارامتر برایش ارسال می‌شوند.همانطور که گفتیم این عملیات بر عهده پیاده ساز این واسط است و ما در این نمودار این وظیفه را فقط بر عهده ConcreteCreator گذاشته ایم که از واسط Icreator مشتق شده است.

پیاده سازی UMLفوق به صورت زیر است:

در ابتدا کلاس واسط IProduct تعریف شده است.

```
interface IProduct
{
      //  در اینجا  برحسب نیاز فیلدها  و یا امضای متدها قرار می‌گیرند
}
```

در این مرحله ما پند پیاده سازی از IProduct انجام می‌دهیم.

```
class ConcreteProductA : IProduct
{
     // پیاده سازی A
}

class ConcreteProductB : IProduct
{
     // پیاده سازی B
}
```

در این مرحله کلاس انتزاعی Creator تعریف می‌شود.

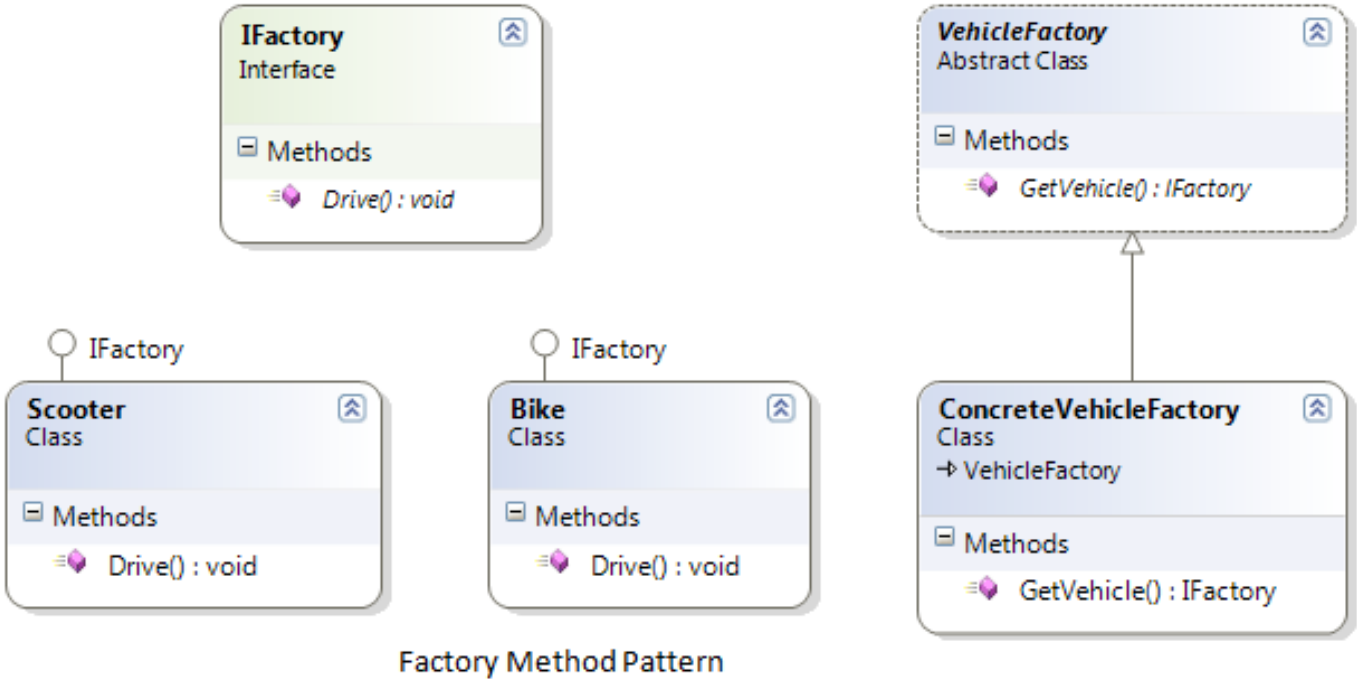
```
abstract class Creator
{
         // این متد بر اساس نوع ورودی انتخاب مناسب را انجام و باز می‌گرداند
          public abstract IProduct FactoryMethod(string type);
}
```

در این مرحله ما با ارث بری از Creator متد Abstract آن را به شیوه خودمان پیاده سازی می‌کنیم.

```
class ConcreteCreator : Creator
{
     public override IProduct FactoryMethod(string type)
    {
           switch (type)
          {
               case "A": return new ConcreteProductA();
               case "B": return new ConcreteProductB();
               default: throw new ArgumentException("Invalid type", "type");
          }
     }
}
```

مثالی از Factory Pattern در C# :

برای روشن‌تر شدن موضوع ، یک مثال کاملتر ارائه داده می‌شود. در شکل زیر طراحی این برنامه نشان داده شده است.



کد برنامه به شرح زیل است :

```
using System;

namespace FactoryMethodPatternRealWordConsolApp
{
    internal class Program
    {
        private static void Main(string[] args)
        {
            VehicleFactory factory = new ConcreteVehicleFactory();

            IFactory scooter = factory.GetVehicle("Scooter");
            scooter.Drive(10);

            IFactory bike = factory.GetVehicle("Bike");
            bike.Drive(20);

            Console.ReadKey();

        }
    }

    public interface IFactory
    {
        void Drive(int miles);
    }

    public class Scooter : IFactory
    {
        public void Drive(int miles)
        {
            Console.WriteLine("Drive the Scooter : " + miles.ToString() + "km");
        }
    }

    public class Bike : IFactory
    {
        public void Drive(int miles)
        {
```

```
            Console.WriteLine("Drive the Bike : " + miles.ToString() + "km");
        }
    }

    public abstract class VehicleFactory
    {
        public abstract IFactory GetVehicle(string Vehicle);

    }

    public class ConcreteVehicleFactory : VehicleFactory
    {
        public override IFactory GetVehicle(string Vehicle)
        {
            switch (Vehicle)
            {
                case "Scooter":
                    return new Scooter();
                case "Bike":
                    return new Bike();
                default:
                    throw new ApplicationException(string.Format("Vehicle '{0}' cannot be created",
Vehicle));
            }
        }
    }
}
```

خروجی اجرای برنامه فوق به شکل زیر است :

فایل این برنامه ضمیمه شده است، از لینک مقابل دانلود کنید FactoryMethodPatternRealWordConsolApp.zip

در مقالات بعدی مثال‌های کاربردی‌تر و جامع‌تری از این الگو و الگوهای مرتبط ارائه خواهم کرد...

نظرات خوانندگان

نویسنده: سید ایوب کوکبی
تاریخ: ۱۳۹۲/۰۷/۰۲ ۱۹:۲۲

ممنونم بابت توضیحی که در مورد این الگو ارائه دادید و همچنین مثال خوبی که ارائه کردید، ولی چند تا سوال:
-1 چرا کلاس VehicleFactory هم از نوع اینترفیس انتخاب نشده است؟ (آیا این موضوع سلیقه ای است؟)
-2 استفاده از کلمه کلید string به جای نام کلاس String آیا تفاوتی در سرعت اجرا ایجاد میکند؟
-3 چرا در دیاگرام uml رابطه بین ConcreteCreator و Concreteproduct از نوع dependency است و از نوع Association نیست؟ یعنی در مثال رابطه بین ConcreteVehicleFactory و یکی از کلاس‌های Bike و یا Scooter
-4چرا در ویژوال استودیو تولید خودکار uml از کد موجود متفاوت با دیاگرام فعلی است، مثلا نوع روابط درست نمایش داده نمیشه و همچنین رابطه ای که در مورد 3 اشاره شد در اینجا وجود نداره؟ آیا علتش نقص در این ابزار است؟ اگر بله، آیا ابزاری وجود داره که دیاگرام رو دقیقتر جنریت کنه؟
و یک نکته:
در کلاس‌های Scooter و Bike نیازی به استفاده از متد ToStringبرای تبدیل مقدار عددی miles نیست چون با یک عبارت رشته ای دیگه جمع شده به صورت درونی این متد توسط CLR فراخوانی میشه. البته این مورد رو Resharper دوست داشتنی تذکر داد. بهتره قبل از ارائه سورس پیشنهادات Resharpe هم روی کد اعمال بشه تا کد در بهترین وضعیت ارائه بشه.
ممنونم/.

نویسنده: مجتبی شاطری
تاریخ: ۱۳۹۲/۰۷/۰۳ ۰:۳

جواب سوال اول :

بله کلاس VehicleFactory میتونه اینترفیس باشه. در اینجا سلیقه ای انجام شده. اما ممکنه در جایی نیاز باشه که ما بخواهیم ورژن پذیری را تو پروژمون لحاظ کنیم که از کلاس abstract استفاده می‌کنیم. ورژن پذیر بودن یعنی اینکه اگرشما متدی به اینترفیس اضافه کنید ، بایستی در تمام کلاسهایی که از آن اینترفیس ارث بری کردند پیاده سازی اون متد را انجام دهید. در کلاس abstract شما به راحتی متدی تعریف می‌کنید که نیاز نیست برای همه استفاده کننده‌ها اون متد را override کنید. این یعنی ورژن پذیری بهتر.
جواب سوال دوم :

string در واقع یک نام مستعار برای کلاس System.String هست. مثل int برای کلاس System.Int32 . پس تفاوتی در سرعت ندارند و میشه از کلاس String هم در اینجا استفاده کرد. چند نمونه برای مثال براتون میزارم :

```
string myagebyStringClass = String.Format("My age is {0}", 27);
```

معادل با :

```
string myagebystringType = string.Format("My age is {0}", 27);
```

و اینم چند نمونه دیگه :

```
object:  System.Object
string:  System.String
bool:    System.Boolean
byte:    System.Byte
sbyte:   System.SByte
short:   System.Int16
ushort:  System.UInt16
int:     System.Int32
uint:    System.UInt32
```

```
long:    System.Int64
ulong:   System.UInt64
float:   System.Single
double:  System.Double
decimal: System.Decimal
char:    System.Char
```

جواب سوال سوم :

همونطور که میدونید رابطه Association (انجمنی) مربوط به ارتباطی یک به یک هستش. البته دو نوع هم داره که یکیش Aggregation (تجمع) و دیگری Composition (ترکیب) است. از اونجایی که نباید ConcreteProduct به ConcreteCreator وابسته باشه پس ما از رابطه Association در این مدل استفاده نمی‌کنیم. درمثال‌ها هم مشخص هست.
جواب سوال چهارم من نمی‌دونم.
درباره اون نکته Reshaper هم حرف شما صحیح هست . البته این یک مثال کلی هست. ممنون که این نکته رو یاد آوری کردید.