عنوان: **آشنایی با الگوی M-V-VM- قسمت اول**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۲/۰۱ ۱۴:۰۴:۰۰
آدرس: www.dotnettips.info
گروه‌ها: MVVM

در مورد الگوی MVVM پیشتر دو مطلب در این سایت منتشر شده‌اند : + و +
مشکل عمده‌ای هم که در مورد این الگو وجود دارد کمبود منابع آموزشی آن به زبان ساده است. هر چند این الگو از طرف خود مایکروسافت ارائه شده اما همانند ASP.Net MVC به آن پر و بال ندادند و شاهد چند ده کتاب منتشر شده در مورد آن نیستیم.
به همین جهت خلاصه‌ای چند قسمتی را در این مورد تهیه کرده‌ام که در طی روزهای آتی ارائه خواهند شد.

فهرست قسمت اول:
M-V-VM چیست؟
آشنایی با اجزای مختلف الگوی M-V-VM
مزایای استفاده از الگوی M-V-VM
اصول کاری و بایدها و نبایدهای الگوی M-V-VM
بایدها و نبایدهای یک View
بایدها و نبایدهای ViewModel
بایدها و نبایدهای Model
مروری بر معایب الگوی M-V-VM

دریافت قسمت اول

لینک کمکی **دریافت این سری**

نظرات خوانندگان

نویسنده: reza khanmirzaee
تاریخ: ۱۳۸۹/۰۲/۰۱ ۱۳:۴۵:۲۴

این الگو فقط در wpf و silverlight استفاده می شود درسته؟

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۲/۰۱ ۱۴:۱۰:۱۴

به دلیل قابلیت‌های binding پیشرفته‌ی WPF و Silverlight، الگوی MVVM بیشتر با این دو فناوری سازگار است؛ هر چند محدودیتی هم برای استفاده از آن در سایر حالات نیست.

نویسنده: hossein
تاریخ: ۱۳۸۹/۱۰/۲۴ ۱۲:۰۱:۴۰

نمام لینک هاتون رو فیلتر کردن

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۱۰/۲۴ ۱۲:۰۴:۲۸

اخیرا کل دامین box.net رو فیلتر کردن (به همراه تمام زیر مجموعه‌ها). هر از چندگاهی این اتفاق میفته بعد از مدتی درست میشه!

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۱۰/۲۴ ۱۴:۵۱:۲۶

آدرس دیگری برای دریافت تمام قسمت‌ها: (+)

عنوان: **آشنایی با الگوی M-V-VM - قسمت دوم**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۲/۰۲ ۰۰:۰۵:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: MVVM

در این قسمت، یک مثال ساده، بدون استفاده از فریم ورک‌های متداول M-V-VM بررسی شده است. در قسمت‌های بعدی با یک سری از فریم ورک‌های موجود آشنا خواهیم شد.

فهرست مطالب:

دریافت قسمت دوم
دریافت مثال قسمت دوم

نظرات خوانندگان

نویسنده: mohammad
تاریخ: ۱۳۸۹/۰۵/۰۶ ۱۶:۴۵:۳۶

اول اینکه واقعا دستتون درد نکنه ،وبلاگ بسیار خوبی دارین.
و اما در مثالی که آوردین اگر رکوردی select نشه برنامه error میده .پس باید متد canexecute درکلاس DoDecreaseCommand بار اول false برگردونه ولی canexecute فقط یکبار توسط دکمه مورد نظر فراخونی میشه.
این مشکل رو چطور باید رفع کرد که در ضمن موقعی که age>60 شد هم کلید غیر فعال شه

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۵/۰۶ ۱۸:۵۵:۳۲

- بله. اینجا یک بررسی نال بودن آیتم انتخاب شده باید قبل از کم و زیاد کردن مقادیر اضافه شود.
- دو breakpoint داخل هر دو متد CanExecute موجود قرار دهید. خواهید دید که به ازای هر بار کلیک بر روی دکمه‌های متناظر (افزایش یا کاهش)، متد CanExecute مرتبط هم در ابتدا یکبار فراخوانی می‌شود.
- غیرفعال کردن خودکار دکمه را من در این مثال ابتدایی پیاده سازی نکردم و مرتبط است با EventHandler تعریف شده‌ای به نام CanExecuteChanged. اگر CanExecuteChanged در متدهای CanExecute صدا زده شود این غیر فعال سازی هم رخ خواهد داد.
این مثال رو بر اساس توضیحات ذکر شده به روز کردم و از اینجا قابل دریافت است:
http://www.box.net/shared/zc27lmyvku

+ کلا در مورد روش‌های بهتر Model Validation در قسمت پنجم بیشتر بحث شده است. همچنین در قسمت‌های بعد این کلاس‌های خام مشتق شده از ICommand با نمونه‌های بهتر جایگزین می‌شوند (DelegateCommand و یا RelayCommand).

عنوان: **آشنایی با الگوی M-V-VM - قسمت سوم**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۲/۰۳ ۰۰:۰۸:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: MVVM

در این قسمت، WPF MVVM Toolkit مایکروسافت به صورت کامل بررسی شده است (دریافت، نصب، ارائه یک مثال به همراه توضیحات و ایجاد آزمون‌های واحد).

فهرست مطالب:

دریافت قسمت سوم
دریافت مثال قسمت سوم

عنوان: **آشنایی با الگوی M-V-VM - قسمت چهارم**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۲/۰۴ ۰۰:۰۴:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: MVVM

در این قسمت، MVVM Light Toolkit مورد بررسی قرار گرفته است (دریافت، نصب، به همراه ارائه 4 مثال جهت معرفی توانمندی‌های آن)

فهرست مطالب:

دریافت قسمت چهارم
دریافت مثال‌های قسمت چهارم

نظرات خوانندگان

نویسنده: رضا
تاریخ: ۱۳۹۱/۰۷/۱۱ ۱۷:۵۷

سلام. من توی پروژه WPF ام از MVVM Light استفاده کردم. سوالی که داشتم اینه که کِی و کجا باید Messenger رو Unregister کرد (به دلیل اینکه باعث Memory Leak نشه و هم اینکه یک Action چندین بار صدا زده نشود)؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۱۱ ۱۹:۱۲

هر زمان که به آن‌ها نیازی نداشتید (اتمام کار مورد نظر، بسته شدن یک پنجره و امثال آن)، حذفشان کنید تا ارجاع به متدهای ثبت شده توسط آن‌ها از بین برود و GC بتواند کارش را انجام دهد. برای مثال در زمان بسته شدن یک پنجره (این مورد تمام ارجاعات تعریف شده توسط پنجره جاری را یکجا حذف می‌کند):

```
Messenger.Default.Unregister(this);
```

عنوان: **آشنایی با الگوی M-V-VM - قسمت پنجم**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۲/۰۴ ۱۴:۱۸:۰۰
آدرس: www.dotnettips.info
گروه‌ها: MVVM

در این قسمت قصد داریم از امکانات جدید اعتبار سنجی تعریف شده در فضای نام استاندارد System.ComponentModel.DataAnnotations استفاده نمائیم. از سیلورلایت سه به بعد امکان استفاده از این فضای نام به سادگی در برنامه‌های سیلورلایت میسر است (همچنین در برنامه‌های ASP.Net MVC)؛ اما برای کار با آن در WPF نیاز به تعدادی متد کمکی می‌باشد...

فهرست مطالب:

دریافت قسمت پنجم
دریافت مثال قسمت پنجم

تعدادی از منابع و مآخذ مورد استفاده در این سری:

1. Model-View-ViewModel (MVVM) Explained
2. Model View ViewModel
3. DataModel-View-ViewModel pattern
4. Minute Overview of MVVM in Silverlight 5
5. A Field Guide to WPF Presentation Patterns
6. An attempt at simple MVVM with WPF
7. WPF: If Heineken did MVVM Frameworks Part 1 of n
8.

Modal dialogs with MVVM and Silverlight 4

9.

How do I do... With the Model-View-ViewModel pattern

10.

Intro to WPF MVVM

11.

Introduction to MVVM pattern in WPF

12.

Learning WPF M-V-VM

13.

Binding Combo Boxes in WPF with MVVM

14.

Model-View-ViewModel Pattern

15.

Unit Testable WCF Web Services in MVVM and Silverlight 4

16.

MVVM Part 1: Overview

17.

?Which came first, the View or the Model

18.

Stackoverflow's questions tagged with MVVM

19.

WPF: MVVM (Model View View-Model) Simplified

20.

WPF and MVVM tutorial 01, Introduction

21.

?...WPF patterns : MVC, MVP or MVVM or

22.

Silverlight, MVVM and Validation Part III

23.

'DotNetKicks.com - Stories recently tagged with 'MVVM

24.

DotNetShoutout - Stories tagged with MVVM

25.

MVVM Light Toolkit

26.

MVVM screen casts

27.

What’s new in MVVM Light V3

28.

Using RelayCommands in Silverlight 3 and WPF

29.

WPF Apps With The Model-View-ViewModel Design Pattern

30.

WPF MVVM and Showing Dialogs

نظرات خوانندگان

نویسنده: sAeid
تاریخ: ۱۳۸۹/۰۲/۰۴ ۱۴:۳۳:۴۳

با سلام
ممنون از مطالب جالب تون در این که بطور کلی در سایت هست و مخصوص در باره MVVM
می خواستم تشکر کنم که فایل ها رو بصورت PDF قرار دادید و خواهش کنم اگر امکان دارد مطالب مربوط به NHibrinate رو هم لطف کنید و بصورت PDF قرار بدهید
چون من مشکل پرینت و مطالعه این مطالب رو داشتم
می دونم کار وقت گیری هست اما زکات علم نشر آن است
یک دنیا تشکر
سعید محمدهاشم

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۲/۰۴ ۱۴:۴۱:۲۷

سلام
ممنون. برای پرینت می‌تونید از فایل chm قرار داده شده (خلاصه وبلاگ، منوی سمت راست بالای سایت) هم استفاده کنید.

نویسنده: C# Builder
تاریخ: ۱۳۸۹/۰۴/۱۴ ۲۰:۱۱:۳۳

بسیار بسیار مفید بود با تشکر فراوان

نویسنده: داود رمضانی
تاریخ: ۱۳۸۹/۰۹/۱۸ ۱۲:۰۳:۱۷

لطفا این آموزش رو ادامه بدید

نویسنده: رضا
تاریخ: ۱۳۹۱/۰۷/۰۸ ۱۱:۴۵

آقای نصیری من با توجه به مطالب این فصل یک سوال داشتم: وقتی ما از Entity Freamework Code First استفاده میکنیم، و تمام کلاس‌های مربوطه را داخل یک پروژه دیگر به اسم DAL قرار میدهیم، روش درست این است که به ازای هر موجودیت یک کلاس دیگر ساخته و INotifyPropertyChanged و ... را در آن پیاده سازی کنیم؟
به عنوان مثال وقتی کلاس Code First ما Customer می‌باشد ما یک کلاس دیگر مثلاً به اسم CustomerModel بسازیم (برای پیاده سازی INotifyPropertyChanged و IDataErrorInfo و ...) و بعد هنگام انجام عملیات (ثبت و ویرایش و حذف) این دو کلاس رو به هم Map کنیم؟
ممنون.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۰۸ ۱۱:۵۶

دو نوع کلاس اینجا وجود دارند:
domain models : کلاس‌های معادل جداول و موجودیت‌های بانک اطلاعاتی
viewmodels : مقصود از این viewmodelها، کلاس مدل معادل عناصر بصری UI است و منظور viewmodel تعریف شده در MVVM نیست که دقیقا معادل Controller در MVC است.
بنابراین اگر domain model شما با مدل معادل view یکی است، همه رو یکجا هم می‌تونید تعریف کنید ولی عموما این‌ها یکی

نیستند. بنابراین نیاز است بین این دو فرق گذاشت و در صورت نیاز نگاشت لازم را انجام داد.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۲۰ ۹:۴۲

لینک کمکی دریافت این سری

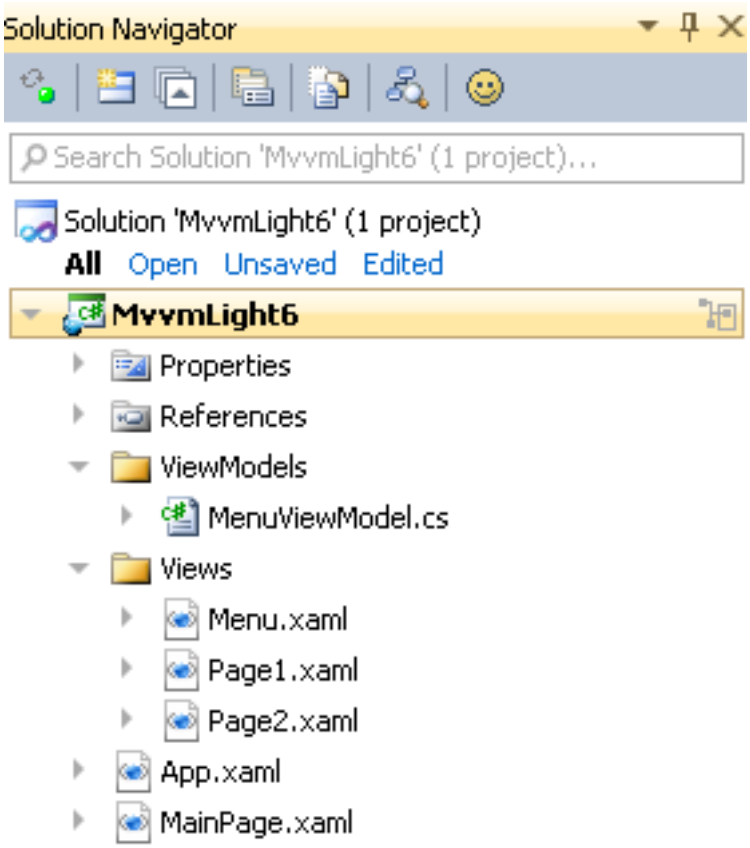
عنوان: **راهبری در Silverlight به کمک الگوی MVVM**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۵/۰۸ ۲۲:۲۵:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: MVVM

مقدمات راهبری (Navigation) در سیلورلایت را در اینجا می‌توانید مطالعه نمائید : +
مطلبی را که در فصل فوق نخواهید یافت در مورد نحوه‌ی بکارگیری الگوی MVVM جهت پیاده سازی Navigation در یک برنامه‌ی سیلورلایت است؛ علت آن هم به این بر می‌گردد که این فصل پیش از مباحث Binding مطرح شد.

صورت مساله:
یکی از اصول MVVM این است که در ViewModel نباید ارجاعی از View وجود داشته باشد (ViewModel باید در بی‌خبری کامل از وجود اشیاء UI و ارجاع مستقیم به آن‌ها طراحی شود)، اما برای پیاده سازی مباحث Navigation نیاز است به نحوی به شیء Frame قرار داده شده در صفحه‌ی اصلی یا قالب اصلی برنامه دسترسی یافت تا بتوان درخواست رهنمون شدن به صفحات مختلف را صادر کرد. اکنون چکار باید کرد؟

راه حل:
یکی از راه حل‌های جالبی که برای این منظور وجود دارد استفاده از امکانات کلاس Messenger مجموعه‌ی MVVM Light toolkit است. از طریق ViewModel برنامه، آدرس صفحه‌ی مورد نظر را به صورت یک پیغام به View مورد نظر ارسال می‌کنیم و سپس View برنامه که به این پیغام‌ها گوش فرا می‌دهد، پس از دریافت آدرس مورد نظر، نسبت به فراخوانی تابع Navigate شیء Frame رابط کاربری برنامه اقدام خواهد کرد. به این صورت ViewModel برنامه به View خود جهت اعمال راهبری برنامه، گره نخواهد خورد.

روش پیاده سازی:
ابتدا ساختار پروژه را در نظر بگیرید (این شکل دگرگون شده‌ی Solution explorer مرتبط است با productivity tools نصب شده):



در پوشه‌ی Views ، دو صفحه اضافه شده‌اند که توسط user control ایی به نام menu لیست شده و راهبری خواهند شد. مونتاژ نهایی هم در MainPage.xaml صورت می‌گیرد.

کدهای XAML مرتبط با منوی ساده برنامه به شرح زیر هستند (Menu.xaml) :

```
<UserControl x:Class="MvvmLight6.Views.Menu"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
   xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
   xmlns:vm="clr-namespace:MvvmLight6.ViewModels" mc:Ignorable="d"
   FlowDirection="RightToLeft" d:DesignHeight="300" d:DesignWidth="400">
   <UserControl.Resources>
       <vm:MenuViewModel x:Key="vmMenuViewModel" />
   </UserControl.Resources>
   <StackPanel DataContext="{Binding Source={StaticResource vmMenuViewModel}}">
       <HyperlinkButton Content="صفحه یک" Margin="5"
                        Command="{Binding DoNavigate}"
                        CommandParameter="/Views/Page1.xaml"
                        />
       <HyperlinkButton Content="صفحه دو" Margin="5"
                        Command="{Binding DoNavigate}"
                        CommandParameter="/Views/Page2.xaml"
                        />
   </StackPanel>
</UserControl>
```

کدهای ViewModel مرتبط با این View که کار Command گردانی را انجام خواهد داد به شرح زیر است:

```
using GalaSoft.MvvmLight.Command;
using GalaSoft.MvvmLight.Messaging;

namespace MvvmLight6.ViewModels
```

```
{
public class MenuViewModel
{
    public RelayCommand<string> DoNavigate { set; get; }

    public MenuViewModel()
    {
        DoNavigate = new RelayCommand<string>(doNavigate);
    }

    private static void doNavigate(string url)
    {
        Messenger.Default.Send(url, "MyNavigationService");
    }
}
}
```

تمام آیتم‌های منوی فوق یک روال را صدا خواهند زد : DoNavigate . تنها تفاوت آن‌ها در CommandParameter ارسالی به RelayCommand ما است که حاوی آدرس قرارگیری فایل‌های صفحات تعریف شده است. این آدرس‌ها با کمک امکانات کلاس Messenger مجموعه‌ی MVVM light toolkit به View اصلی برنامه ارسال می‌گردند.
کدهای XAML مرتبط با MainPage.xaml به شرح زیر هستند:

```
<UserControl x:Class="MvvmLight6.MainPage"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
   xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
   xmlns:sdk="clr-namespace:System.Windows.Controls;assembly=System.Windows.Controls.Navigation"
   xmlns:usr="clr-namespace:MvvmLight6.Views"
   mc:Ignorable="d" d:DesignHeight="300" d:DesignWidth="400">
   <Grid x:Name="LayoutRoot" Background="White">
       <Grid.ColumnDefinitions>
           <ColumnDefinition Width="*" />
           <ColumnDefinition Width="268" />
       </Grid.ColumnDefinitions>
       <usr:Menu Grid.Column="1" />
       <sdk:Frame Margin="5"
                  Name="frame1"
                  HorizontalContentAlignment="Stretch"
                  VerticalContentAlignment="Stretch"
                  Grid.Column="0" />
   </Grid>
</UserControl>
```

و کار دریافت پیغام‌ها (یا همان آدرس صفحات جهت انجام راهبری) و عکس العمل نشان دادن به آن‌ها توسط کدهای ذیل صورت خواهد گرفت:

```
using System;
using GalaSoft.MvvmLight.Messaging;

namespace MvvmLight6
{
public partial class MainPage
{
    public MainPage()
    {
        registerMessenger();
        InitializeComponent();
    }

    private void registerMessenger()
    {
        Messenger.Default.Register<string>(this, "MyNavigationService", doNavigate);
    }

    private void doNavigate(string uri)
    {
        frame1.Navigate(new Uri(uri, UriKind.Relative));
    }
```

```
}
}
```

ابتدا یک Messenger در اینجا رجیستر می‌شود و سپس به ازای هر بار دریافت پیغامی با token مساوی MyNavigationService ، متد doNavigate فراخوانی خواهد گردید.
کدهای این مثال را از اینجا می‌توانید دریافت کنید.

نظرات خوانندگان

نویسنده: iMAN
تاریخ: ۱۳۸۹/۰۵/۰۸ ۲۳:۳۵:۳۰

سلام آقای نصیری
جا داره یک خسته نباشید بهتون بگم، من واقعاً از آموزش های MVVM شما استفاده کردم و از این بابت از شما ممنونم.

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۵/۰۸ ۲۳:۴۵:۱۸

لطف دارید. سلامت باشید.

نویسنده: mohammad
تاریخ: ۱۳۸۹/۰۶/۲۰ ۱۱:۱۲:۴۴

با سلام من یک سوالی در باره RelayCommand دارم من از وبی استفاده می کنم و زمان استفاده از RelayCommand اون رو نمی شناسه میشه بگین تو چه nameSpace هست

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۶/۲۰ ۱۷:۴۰:۲۲

ارجاعی را به اسمبلی‌های MVVM Light toolkit اضافه کنید.

نویسنده: Meysam
تاریخ: ۱۳۸۹/۰۶/۲۰ ۲۱:۰۹:۳۶

سلام
چطوری یک Concrete Class رو از یک صفحه به یک صفحه دیگر پاس بدم؟(با استفاده از این الگو یا هر روش دیگر)

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۶/۲۰ ۲۱:۴۰:۱۰

یک روش : برای پاس دادن یک وهله از یک کلاس به کلاسی دیگر در Silverlight یا WPF (مثلا در حین Navigation) می‌تونید از امکانات کلاس Messenger مجموعه‌ی MVVM Light toolkit استفاده کنید.
توضیحات مرتبط با هر کدام در جزوه MVVM (برای WPF) یا کتاب Silverlight منتشر شده موجود است.

مثال فوق هم به همین ترتیب یک رشته را از یک کلاس به کلاس دیگر پاس داده است.

نویسنده: Meysam
تاریخ: ۱۳۸۹/۰۶/۲۰ ۲۲:۱۵:۵۷

صورت مسئله اینجور هستش که ، یک صفحه دارم که از Frame برای راهبری استفاده میکنه(بدون هیچگونه کتابخونه ای و با استفاده از الگوی MVVM) ، به این صورت که متد Navigate فریم مربوطه به یک Func در یک VM وصل شده که در Command یک دکمه آنرا فراخوانی میکند. در واقع هیچ اختیاری بین اشیا VM ندارم!

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۶/۲۰ ۲۳:۰۸:۵۹

این هم خوبه؛ شبیه به ارسال اطلاعات به کمک یک delegate یا event . از سر ناچاری!

- ضمنا MVVM Light toolkit سورس باز است. کلاس Messenger آن‌را جدا کنید و استفاده کنید (اگر از کل آن نمی‌خواهید استفاده کنید).

یک روش هم اینجا دیدم که خیلی جالب است:
http://forums.silverlight.net/forums/p/198684/463126.aspx
از NavigateUri یک HyperlinkButton استفاده کرده. فقط UriMapper را هم تنظیم کرده برای زیبایی کار.
(نیاز به هیچ کتابخانه جانبی هم ندارد. نیازی به دخالت MVVM هم ندارد.و مهم‌تر از همه، نیازی به کد نویسی هم اصلا ندارد.)

ولی برای پاس دادن یک وهله از صفحه جاری به صفحه بعد (مثل لینک داخل صفحات وب)، کلاس Messenger واقعا تر و تمیز و عالی است. (نیازی هم به استفاده از کوئری استرینگ یا هر روش دیگری نیست)

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۶/۲۰ ۲۳:۱۴:۳۶

علاوه بر این‌ها برای انتقال وهله از MEF هم می‌شود استفاده کرد (managed extensibility framework)، که می‌تواند singleton باشد یا به ازای هر بار استفاده وهله سازی شود. (در موردش قبلا مطلب نوشتم در سایت)

نویسنده: farbod
تاریخ: ۱۳۸۹/۰۸/۰۹ ۲۳:۲۱:۰۲

از زحمتی که برای مجموعه آموزشی سیلورلایت کشیدید واقعا متشکرم.

عنوان: **معرفی یک ابزار گزارشگیری رایگان مخصوص WPF**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۸/۰۱ ۱۴:۵۷:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: MVVM

تا صحبت از گزارشگیری به میان بیاید احتمالا معرفی ابزارهای تجاری مانند Reporting services ، کریستال ریپورت، fast-report.com ، stimulsoft.com و امثال آن درصدر لیست توصیه کنندگان و مشاوران قرار خواهند داشت. اما خوب برای ایجاد یک گزارشگیری ساده حتما نیازی نیست تا به این نوع ابزارهای تجاری مراجعه کرد. ابزار رایگان و سورس باز جالبی هم در این باره جهت پروژه‌های WPF در دسترس است:

Open-Source .NET WPF Reporting Engine

در ادامه در طی یک مثال قصد داریم از این کتابخانه استفاده کنیم:

1) تنظیم وابستگی‌ها
پس از دریافت کتابخانه فوق، ارجاعات زیر باید به پروژه شما اضافه شوند:
CodeReason.Reports.dll (از پروژه فوق) و ReachFramework.dll (جزو اسمبلی‌های استاندارد دات نت است)

2) تهیه منبع داده گزارش
کتابخانه‌ی فوق به صورت پیش فرض با DataTable کار می‌کند. بنابراین کوئری‌های شما یا باید خروجی DataTable داشته باشد یا باید از یک سری extension methods برای تبدیل IEnumerable به DataTable استفاده کرد (در پروژه پیوست شده در پایان مطلب، این موارد موجود است).
برای مثال فرض کنید می‌خواهیم رکوردهایی را از نوع کلاس Product زیر در گزارش نمایش دهیم:

```
namespace WpfRptTests.Model
{
  public class Product
  {
      public string Name { set; get; }
      public int Price { set; get; }
  }
}
```

3) تعریف گزارش
الف) اضافه کردن فایل تشکیل دهنده ساختار و ظاهر گزارش
گزارش‌های این کتابخانه مبتنی است بر اشیاء FlowDocument استاندارد WPF . بنابراین از منوی پروژه گزینه‌ی Add new item در قسمت WPF آن یک FlowDocument جدید را به پروژه اضافه کنید (باید دقت داشت که Build action این فایل باید به Content تنظیم گردد). ساختار ابتدایی این FlowDocument به صورت زیر خواهد بود که به آن FlowDirection و FontFamily مناسب جهت گزارشات فارسی اضافه شده است. همچنین فضای نام مربوط به کتابخانه‌ی گزارشگیری CodeReason.Reports نیز باید اضافه گردد.

```
<FlowDocument xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
            xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
            FlowDirection="RightToLeft" FontFamily="Tahoma"
            xmlns:xrd="clr-namespace:CodeReason.Reports.Document;assembly=CodeReason.Reports"
            PageHeight="29.7cm" PageWidth="21cm" ColumnWidth="21cm">

</FlowDocument>
```

مواردی که در ادامه ذکر خواهند شد محتوای این گزارش را تشکیل می‌دهند:
ب) مشخص سازی خواص گزارش

```
<xrd:ReportProperties>
      <xrd:ReportProperties.ReportName>SimpleReport</xrd:ReportProperties.ReportName>
      <xrd:ReportProperties.ReportTitle>گزارش از محصولات</xrd:ReportProperties.ReportTitle>
  </xrd:ReportProperties>
```

در اینجا ReportName و ReportTitle باید مقدار دهی شوند (دو dependency property که در کتابخانه‌ی CodeReason.Reports تعریف شده‌اند)

ج) مشخص سازی Page Header و Page Footer
اگر می‌خواهید عباراتی در بالا و پایین تمام صفحات گزارش تکرار شوند می‌توان از SectionReportHeader و SectionReportFooter این کتابخانه به صورت زیر استفاده کرد:

```
<xrd:SectionReportHeader PageHeaderHeight="2" Padding="10,10,10,0" FontSize="12">
      <Table CellSpacing="0">
          <Table.Columns>
              <TableColumn Width="*" />
              <TableColumn Width="*" />
          </Table.Columns>
          <TableRowGroup>
              <TableRow>
                  <TableCell>
                      <Paragraph>
                          <xrd:InlineContextValue PropertyName="ReportTitle" />
                      </Paragraph>
                  </TableCell>
                  <TableCell>
                      <Paragraph TextAlignment="Right">
                          <xrd:InlineDocumentValue PropertyName="PrintDate" Format="dd.MM.yyyy
HH:mm:ss" />
                      </Paragraph>
                  </TableCell>
              </TableRow>
          </TableRowGroup>
      </Table>
  </xrd:SectionReportHeader>

  <xrd:SectionReportFooter PageFooterHeight="2" Padding="10,0,10,10" FontSize="12">
      <Table CellSpacing="0">
          <Table.Columns>
              <TableColumn Width="*" />
              <TableColumn Width="*" />
          </Table.Columns>
          <TableRowGroup>
              <TableRow>
                  <TableCell>
                      <Paragraph>
                          نام کاربر:
                          <xrd:InlineDocumentValue PropertyName="RptBy" Format="dd.MM.yyyy HH:mm:ss" />
                      </Paragraph>
                  </TableCell>
                  <TableCell>
                      <Paragraph TextAlignment="Right">
                          صفحه
                          <xrd:InlineContextValue PropertyName="PageNumber" FontWeight="Bold" /> از
                          <xrd:InlineContextValue PropertyName="PageCount" FontWeight="Bold" />
                      </Paragraph>
                  </TableCell>
              </TableRow>
          </TableRowGroup>
      </Table>
  </xrd:SectionReportFooter>
```

دو نکته در اینجا حائز اهمیت هستند: xrd:InlineDocumentValue و xrd:InlineContextValue
InlineDocumentValue را می‌توان در کدهای برنامه به صورت سفارشی اضافه کرد. بنابراین هر جایی که نیاز بود مقدار ثابتی از

طریق کد نویسی به گزارش تزریق و اضافه شود می‌توان از InlineDocumentValue استفاده کرد. برای مثال در کدهای ViewModel برنامه که در ادامه ذکر خواهد شد دو مقدار PrintDate و RptBy به صورت زیر تعریف و مقدار دهی شده‌اند:

```
data.ReportDocumentValues.Add("PrintDate", DateTime.Now);
data.ReportDocumentValues.Add("RptBy", "وحید");
```

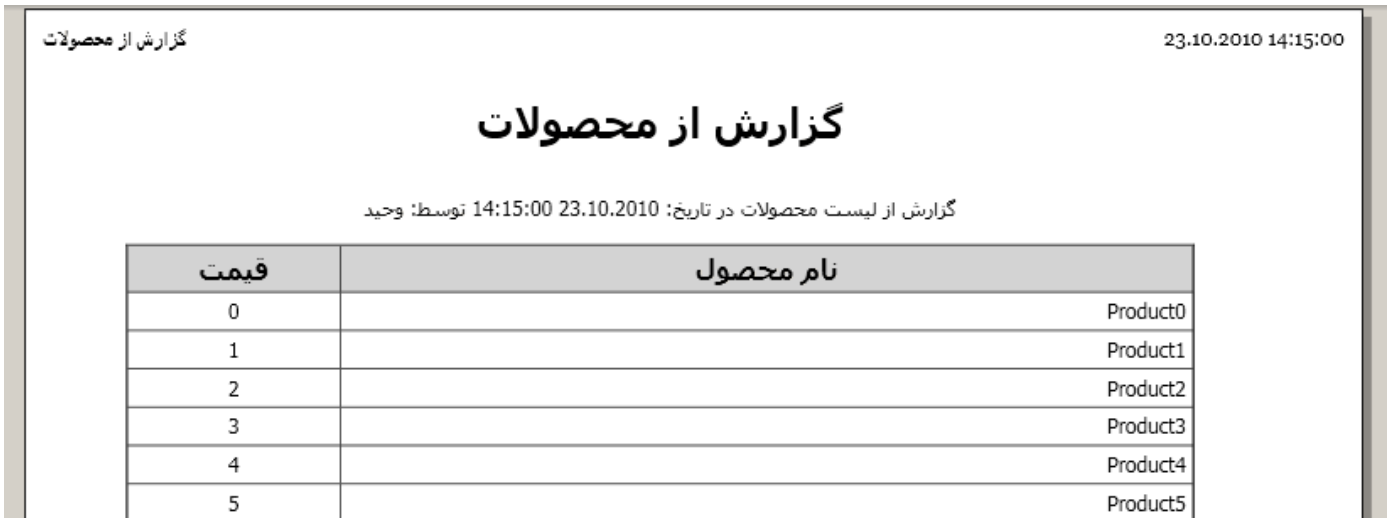
برای مشاهده مقادیر مجاز مربوط به InlineContextValue به فایل ReportContextValueType.cs سورس کتابخانه مراجعه کنید که شامل PageNumber, PageCount, ReportName, ReportTitle است و توسط CodeReason.Reports به صورت پویا تنظیم خواهد شد.

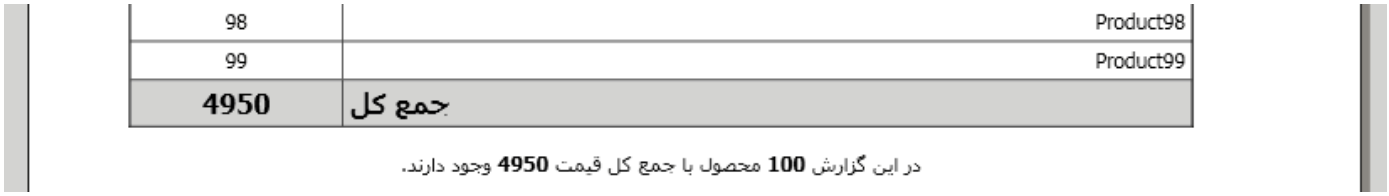
د) مشخص سازی ساختار تولیدی گزارش

```
<Section Padding="80,10,40,10" FontSize="12">
      <Paragraph FontSize="24" TextAlignment="Center" FontWeight="Bold">
          <xrd:InlineContextValue PropertyName="ReportTitle" />
      </Paragraph>
      <Paragraph  TextAlignment="Center">
          گزارش از لیست محصولات در تاریخ:
          <xrd:InlineDocumentValue PropertyName="PrintDate" Format="dd.MM.yyyy HH:mm:ss" />
          توسط:
          <xrd:InlineDocumentValue PropertyName="RptBy" Format="dd.MM.yyyy HH:mm:ss" />
      </Paragraph>
      <xrd:SectionDataGroup DataGroupName="ItemList">
          <Table CellSpacing="0" BorderBrush="Black" BorderThickness="0.02cm">
              <Table.Resources>
                  <!-- Style for header/footer rows. -->
                  <Style x:Key="headerFooterRowStyle" TargetType="{x:Type TableRowGroup}">
                      <Setter Property="FontWeight" Value="DemiBold"/>
                      <Setter Property="FontSize" Value="16"/>
                      <Setter Property="Background" Value="LightGray"/>
                  </Style>

                  <!-- Style for data rows. -->
                  <Style x:Key="dataRowStyle" TargetType="{x:Type TableRowGroup}">
                      <Setter Property="FontSize" Value="12"/>
                  </Style>

                  <!-- Style for data cells. -->
                  <Style TargetType="{x:Type TableCell}">
                      <Setter Property="Padding" Value="0.1cm"/>
                      <Setter Property="BorderBrush" Value="Black"/>
                      <Setter Property="BorderThickness" Value="0.01cm"/>
                  </Style>
              </Table.Resources>

              <Table.Columns>
                  <TableColumn Width="0.8*" />
                  <TableColumn Width="0.2*" />
              </Table.Columns>
              <TableRowGroup Style="{StaticResource headerFooterRowStyle}">
                  <TableRow>
                      <TableCell>
                          <Paragraph TextAlignment="Center">
                              <Bold>نام محصول</Bold>
                          </Paragraph>
                      </TableCell>
                      <TableCell>
                          <Paragraph TextAlignment="Center">
                              <Bold>قیمت</Bold>
                          </Paragraph>
                      </TableCell>
                  </TableRow>
              </TableRowGroup>

              <TableRowGroup Style="{StaticResource dataRowStyle}">
                  <xrd:TableRowForDataTable TableName="Product">
                      <TableCell>
                          <Paragraph>
                              <xrd:InlineTableCellValue PropertyName="Name" />
                          </Paragraph>
                      </TableCell>
                      <TableCell>
                          <Paragraph TextAlignment="Center">
                              <xrd:InlineTableCellValue PropertyName="Price" AggregateGroup="Group1" />
```

```
                        </Paragraph>
                    </TableCell>
                </xrd:TableRowForDataTable>
            </TableRowGroup>

            <TableRowGroup Style="{StaticResource headerFooterRowStyle}">
                <TableRow>
                    <TableCell>
                        <Paragraph TextAlignment="Right">
                            <Bold>جمع کل</Bold>
                        </Paragraph>
                    </TableCell>
                    <TableCell>
                        <Paragraph TextAlignment="Center">
                            <Bold>
                                <xrd:InlineAggregateValue AggregateGroup="Group1"
                                                          AggregateValueType="Sum"
                                                          EmptyValue="0"
                                                          FontWeight="Bold" />
                            </Bold>
                        </Paragraph>
                    </TableCell>
                </TableRow>
            </TableRowGroup>

        </Table>

        <Paragraph TextAlignment="Center" Margin="5">
            در این گزارش
            <xrd:InlineAggregateValue AggregateGroup="Group1"
                                      AggregateValueType="Count"
                                      EmptyValue="هیچ"
                                      FontWeight="Bold" /> محصول با جمع کل قیمت
            <xrd:InlineAggregateValue AggregateGroup="Group1"
                                      AggregateValueType="Sum"
                                      EmptyValue="0"
                                      FontWeight="Bold" /> وجود دارند.
        </Paragraph>
    </xrd:SectionDataGroup>
</Section>
```

برای اینکه بتوان این قسمت‌ها را بهتر توضیح داد، نیاز است تا تصاویر مربوط به خروجی این گزارش نیز ارائه شوند:





Product98	98
Product99	99
جمع کل	4950

در این گزارش **100** محصول با جمع کل قیمت **4950** وجود دارند.

در ابتدا توسط دو پاراگراف، عنوان گزارش و یک سطر زیر آن نمایش داده شده‌اند. بدیهی است هر نوع شیء و فرمت مجاز در FlowDocument را می‌توان در این قسمت نیز قرار داد.
سپس یک SectionDataGroup جهت نمایش لیست آیتم‌ها اضافه شده و داخل آن یک جدول که بیانگر ساختار جدول نمایش رکوردهای گزارش می‌باشد، ایجاد گردیده است.
سه TableRowGroup در این جدول تعریف شده‌اند.
TableRowGroup های اولی و آخری دو سطر اول و آخر جدول گزارش را مشخص می‌کنند (سطر عناوین ستون‌ها در ابتدا و سطر جمع کل در پایان گزارش)
از TableRowGroup میانی برای نمایش رکوردهای مرتبط با نام جدول مورد گزارشگیری استفاده شده است. توسط TableRowForDataTable آن نام این جدول باید مشخص شود که در اینجا همان نام کلاس مدل برنامه است. به کمک InlineTableCellValue، خاصیت‌هایی از این کلاس را که نیاز است در گزارش حضور داشته باشند، ذکر خواهیم کرد. نکته‌ی مهم آن AggregateGroup ذکر شده است. توسط آن می‌توان اعمال جمع، محاسبه تعداد، حداقل و حداکثر و امثال آن‌را که در فایل InlineAggregateValue.cs سورس کتابخانه ذکر شده‌اند، به فیلدهای مورد نظر اعمال کرد. برای مثال می‌خواهیم جمع کل قیمت را در پایان گزارش نمایش دهیم به همین جهت نیاز بود تا یک AggregateGroup را برای این منظور تعریف کنیم.
از این AggregateGroup در سومین TableRowGroup تعریف شده به کمک xrd:InlineAggregateValue جهت نمایش جمع نهایی استفاده شده است.
همچنین اگر نیاز بود در پایان گزارش اطلاعات بیشتری نیز نمایش داده شود به سادگی می‌توان با تعریف یک پاراگراف جدید، اطلاعات مورد نظر را نمایش داد.

4) نمایش گزارش تهیه شده
نمایش این گزارش بسیار ساده است. View برنامه به صورت زیر خواهد بود:

```
<Window x:Class="WpfRptTests.MainWindow"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:c="clr-namespace:CodeReason.Reports.Controls;assembly=CodeReason.Reports"
      xmlns:vm="clr-namespace:WpfRptTests.ViewModel"
      Title="MainWindow" WindowState="Maximized" Height="350" Width="525">
  <Window.Resources>
      <vm:ProductViewModel x:Key="vmProductViewModel" />
  </Window.Resources>
  <Grid DataContext="{Binding Source={StaticResource vmProductViewModel}}">
      <c:BusyDecorator IsBusyIndicatorHidden="{Binding RptGuiModel.IsBusyIndicatorHidden}">
          <DocumentViewer Document="{Binding RptGuiModel.Document}"  />
      </c:BusyDecorator>
  </Grid>
</Window>
```

تعریف ابتدایی RptGuiModel به صورت زیر است (جهت مشخص سازی مقادیر IsBusyIndicatorHidden و Document در حین بایندینگ اطلاعات):

```
using System.ComponentModel;
using System.Windows.Documents;

namespace WpfRptTests.Model
```

```
{
  public class RptGuiModel
  {
      public IDocumentPaginatorSource Document { get; set; }
      public bool IsBusyIndicatorHidden { get; set; }
  }
}
```

و این View اطلاعات خود را از ViewModel زیر دریافت خواهد نمود:

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Threading;
using CodeReason.Reports;
using WpfRptTests.Helper;
using WpfRptTests.Model;

namespace WpfRptTests.ViewModel
{
  public class ProductViewModel
  {
      #region Constructors (1)

      public ProductViewModel()
      {
          RptGuiModel = new RptGuiModel();
          if (Stat.IsInDesignMode) return;
          //انجام عملیات نمایش گزارش در یک ترد دیگر جهت قفل نشدن ترد اصلی برنامه
          showReportAsync();
      }

      #endregion Constructors

      #region Properties (1)

      public RptGuiModel RptGuiModel { set; get; }

      #endregion Properties

      #region Methods (3)

      // Private Methods (3)

      private static List<Product> getProducts()
      {
          var products = new List<Product>();
          for (var i = 0; i < 100; i++)
              products.Add(new Product { Name = string.Format("Product{0}", i), Price = i });

          return products;
      }

      private void showReport()
      {
          try
          {
              //Show BusyIndicator
              RptGuiModel.IsBusyIndicatorHidden = false;

              var reportDocument =
                  new ReportDocument
                  {
                      XamlData = File.ReadAllText(@"Report\SimpleReport.xaml"),
                      XamlImagePath = Path.Combine(Environment.CurrentDirectory, @"Report\")
                  };

              var data = new ReportData();

              // تعریف متغیرهای دلخواه و مقدار دهی آن‌ها
              data.ReportDocumentValues.Add("PrintDate", DateTime.Now);
              data.ReportDocumentValues.Add("RptBy", "وحید");

              // استفاده از یک سری اطلاعات آزمایشی به عنوان منبع داده
              data.DataTables.Add(getProducts().ToDataTable());

              var xps = reportDocument.CreateXpsDocument(data);
              //انقیاد آن به صورت غیر همزمان در ترد اصلی برنامه
```

```
                DispatcherHelper.DispatchAction(
                    () => RptGuiModel.Document = xps.GetFixedDocumentSequence()
                    );
            }
            catch (Exception ex)
            {
                //وجود این مورد ضروری است زیرا بروز استثناء در یک ترد به معنای خاتمه آنی برنامه است
                //todo: log errors
            }
            finally
            {
                //Hide BusyIndicator
                RptGuiModel.IsBusyIndicatorHidden = true;
            }
        }

        private void showReportAsync()
        {
            var thread = new Thread(showReport);
            thread.SetApartmentState(ApartmentState.STA); //for DocumentViewer
            thread.Start();
        }

        #endregion Methods
    }
}
```

توضیحات:
برای اینکه حین نمایش گزارش، ترد اصلی برنامه قفل نشود، از ترد استفاده شد و استفاده ترد به همراه DocumentViewer کمی نکته دار است:
- ترد تعریف شده باید از نوع STA باشد که در متد showReportAsync مشخص شده است.
- حین بایندیگ Document تولید شده توسط کتابخانه‌ی گزارشگیری به خاصیت Document کنترل، حتما باید کل عملیات در ترد اصلی برنامه صورت گیرد که سورس کلاس DispatcherHelper را در فایل پیوست خواهید یافت.

کل عملیات این ViewModel در متد showReport رخ می‌دهد، ابتدا فایل گزارش بارگذاری می‌شود، سپس متغیرهای سفارشی مورد نظر تعریف و مقدار دهی خواهند شد. در ادامه یک سری داده آزمایشی تولید و به DataTables گزارش ساز اضافه می‌شوند. در پایان XPS Document متناظر آن تولید شده و به کنترل نمایشی برنامه بایند خواهد شد.

دریافت سورس این مثال

نظرات خوانندگان

نویسنده: حسین مرادی نیا
تاریخ: ۱۳۸۹/۰۸/۱۹ ۲۳:۰۸:۰۴

سلام
خسته نباشید
من احساس می کنم بهتر باشه از همان ابزارهای تجاری استفاده کرد.چون اگه در آینده نیاز باشه برنامه و گزارش مربوطه پیشرفته تر بشته و یا توسعه داده بشه بهتر میشه این کار رو انجام داد.

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۸/۲۰ ۰۰:۱۵:۳۴

این پروژه در حال حاضر دو ایراد داره:
- سر ستون‌های گزارش‌ها در صفحات مختلف تکرار نمی‌شوند.
- امکان گروه بندی ندارد.

مشکل حاد دیگری من ندیدم.

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۸/۲۴ ۱۳:۳۲:۰۲

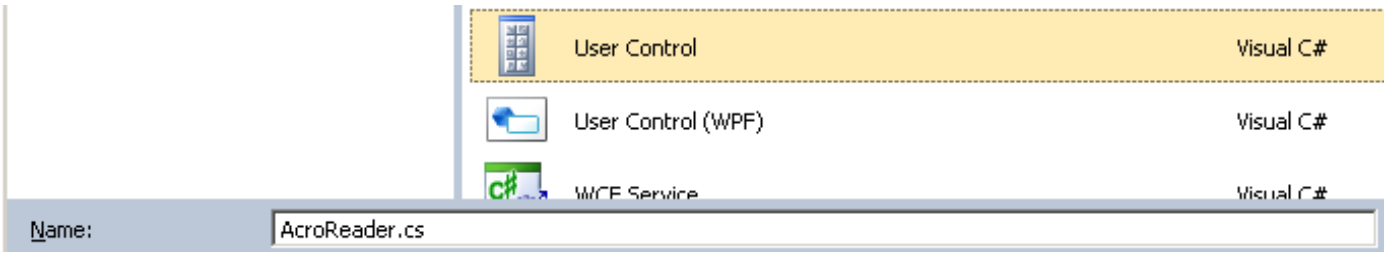
اگر علاقمند به تهیه خروجی از این گزارش‌ها باشید، کلاس‌های تهیه خروجی PDF/PNG/XPS را اینجا آپلود کردم:
(+)

همچنین اگر مایل باشید که به کنترل DocumentViewer دکمه‌های دلخواه تهیه خروجی را اضافه کنید می‌توان از قالب زیر کمک گرفت:
(+)

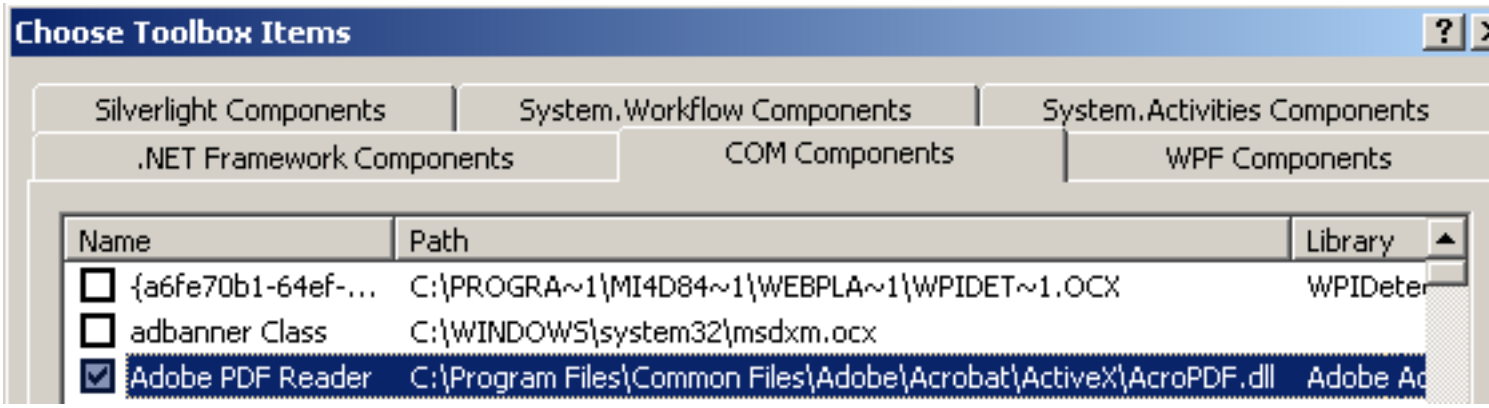
عنوان: **استفاده از کنترل‌های Active-X در WPF**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۹/۰۸ ۰۰:۰۰:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: MVVM

گاهی از اوقات شاید نیاز شود تا از یک کنترل Active-X در WPF استفاده شود؛ مثلا هیچ نمایش دهنده‌ی PDF ایی را در ویندوز نمی‌توان یافت که امکانات و کیفیت آن در حد Acrobat reader و Active-X آن باشد. یک روش استفاده از آن‌را به کمک کنترل WebBrowser در WPF پیشتر در این سایت مطالعه کرده‌اید . روش معرفی شده برای WinForm هم در WPF قابل استفاده است که در ادامه شرح آن خواهد آمد.

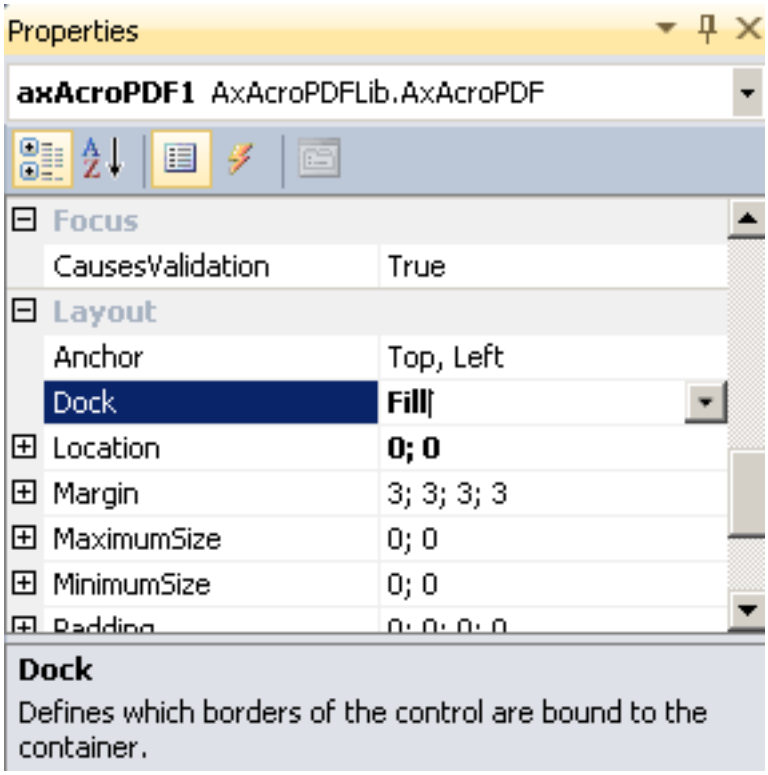
الف) بجای اضافه کردن یک User control مخصوص WPF یک user control از نوع WinForms را به یک پروژه WPF اضافه کنید.



سپس مراحل مشابهی را مانند حالت WinForms، باید طی کرد:
ب) در VS.NET از طریق منوی Tools گزینه‌ی Choose toolbox items ، برگه‌ی Com components را انتخاب کنید.
ج) سپس گزینه‌ی Adobe PDF reader را انتخاب نمائید و بر روی دکمه‌ی OK کلیک کنید.



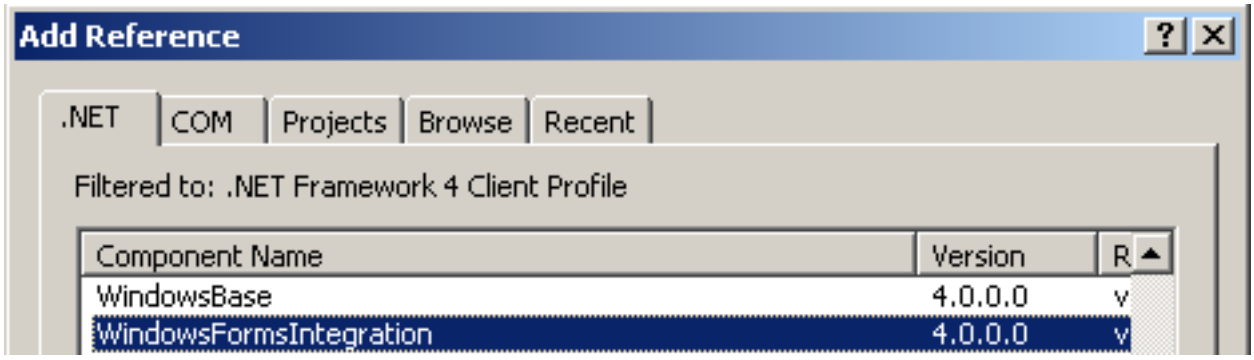
د) اکنون این کنترل جدید را بر روی فرم user control قسمت الف برنامه قرار دهید. به صورت خودکار COMReference های متناظر هم به پروژه اضافه می‌شوند.
پس از اینکه کنترل بر روی فرم قرار گرفت بهتر است به خواص آن مراجعه کرده و خاصیت Dock آن‌را با Fill مقدار دهی کرد تا کنترل به صورت خودکار در هر اندازه‌ای کل ناحیه‌ی متناظر را پوشش دهد.



کدهای مرتبط با نمایش فایل PDF این کنترل هم به شرح زیر است:

```
using System.Windows.Forms;

namespace WpfPdfViewer.Controls
{
    public partial class AcroReader : UserControl
    {
        public AcroReader(string fileName)
        {
            InitializeComponent();
            ShowPdf(fileName);
        }

        public void ShowPdf(string fileName)
        {
            if (string.IsNullOrWhiteSpace(fileName)) return;
            axAcroPDF1.LoadFile(fileName);
            axAcroPDF1.setShowToolbar(true);
            axAcroPDF1.Show();
        }
    }
}
```

خوب، ما تا اینجا یک کنترل Active-X را از طریق یک User controls مخصوص WinForms به پروژه‌ی WPF جاری اضافه کرده‌ایم. برای اینکه بتوانیم این کنترل را درون مثلا یک User control از جنس WPF و XAML نمایش دهیم باید از کنترل WindowsFormsHost استفاده کرد. برای این منظور نیاز است تا ارجاعی را به اسمبلی WindowsFormsIntegration اضافه کنیم. پس از آن کنترل یاد شده قابل استفاده خواهد بود.



برای نمونه کدهای XAML پنجره اصلی برنامه می‌تواند به صورت زیر باشد:

```
<Window x:Class="WpfPdfViewer.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525">
    <Grid>
        <WindowsFormsHost x:Name="WindowsFormsHost1" />
    </Grid>
</Window>
```

سپس جهت استفاده از کنترل WindowsFormsHost خواهیم داشت:

```
using WpfPdfViewer.Controls;

namespace WpfPdfViewer
{
    public partial class MainWindow
    {
        public MainWindow()
        {
            InitializeComponent();
            WindowsFormsHost1.Child = new AcroReader(@"PageSummary.pdf");
        }
    }
}
```

فقط کافی است شیء Child این کنترل را با وهله‌ای از یوزرکنترل AcroReader اضافه شده به برنامه مقدار دهی کنیم.

سؤال: این روش زیاد MVVM friendly نیست. به عبارتی Child را نمی‌توان از طریق Binding مقدار دهی کرد. آیا راهی برای آن وجود دارد؟
پاسخ: بله. روش متداول برای حل این نوع مشکلات، نوشتن یک DependencyObject و Attached property مناسب می‌باشد که به آن‌ها Behaviors هم می‌گویند. برای مثال یک نمونه از این پیاده سازی را در ذیل مشاهده می‌کنید:

```
using System;
using System.Windows;
using System.Windows.Forms;
using System.Windows.Forms.Integration;

namespace WpfPdfViewer.Behaviors
{
    public class WindowsFormsHostBehavior : DependencyObject
```

```
    {
        public static readonly DependencyProperty BindableChildProperty =
                                    DependencyProperty.RegisterAttached("BindableChild",
                                    typeof(Control),
                                    typeof(WindowsFormsHostBehavior),
                                    new UIPropertyMetadata(null, BindableChildPropertyChanged));

        public static Control GetBindableChild(DependencyObject obj)
        {
            return (Control)obj.GetValue(BindableChildProperty);
        }

        public static void SetBindableChild(DependencyObject obj, Control value)
        {
            obj.SetValue(BindableChildProperty, value);
        }

        public static void BindableChildPropertyChanged(DependencyObject o,
DependencyPropertyChangedEventArgs e)
        {
            var windowsFormsHost = o as WindowsFormsHost;
            if (windowsFormsHost == null)
                throw new InvalidOperationException("This behavior can only be attached to a
WindowsFormsHost.");

            var control = (Control)e.NewValue;
            windowsFormsHost.Child = control;
        }
    }
}
```

که نهایتا برای استفاده از آن خواهیم داشت:

```
<WindowsFormsHost
       Behaviors:WindowsFormsHostBehavior.BindableChild="{Binding ...}" />
```

و در ViewModel برنامه هم مانند مثال فوق، فقط کافی است یک وهله از new AcroReader به این خاصیت قابل انقیاد از نوع Control، انتساب داده شود.
یا حتی می‌توان بجای نوشتن یک BindableChild، برای مثال مسیر فایل pdf را به DependencyObject تعریف شده ارسال کرد و سپس در همانجا این وهله سازی و انتسابات صورت گیرد (بجای ViewModel برنامه که اینبار فقط مسیر را تنظیم می‌کند).

عنوان: **تکمیل کلاس DelegateCommand**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۹/۱۹ ۱۳:۱۱:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: MVVM

مدت‌ها از کلاس DelegateCommand معرفی شده در این آدرس استفاده می‌کردم. این کلاس یک مشکل جزئی دارد و آن هم عدم بررسی مجدد قسمت canExecute به صورت خودکار هست.

خلاصه‌ای برای کسانی که بار اول هست با این مباحث برخورد می‌کنند؛ یا MVVM به زبان بسیار ساده:

در برنامه نویسی متداول سیستم مایکروسافتی، در هر سیستمی که ایجاد کرده و در هر فناوری که ارائه داده از زمان VB6 تا امروز، شما روی یک دکمه مثلا دوبار کلیک می‌کنید و در فایل اصطلاحا code behind این فرم و در روال رخدادگردان آن شروع به کد نویسی خواهید کرد. این مورد تقریبا در همه جا صادق است؛ از WinForms تا WPF تا Silverlight تا حتی ASP.NET Webforms . به عمد هم این طراحی صورت گرفته تا برنامه نویس‌ها در این محیط‌ها زیاد احساس غریبی نکنند. اما این روش یک مشکل مهم دارد و آن هم «توهم» جداسازی رابط کاربر از کدهای برنامه است. به ظاهر یک فایل فرم وجود دارد و یک فایل جدای code behind ؛ اما در عمل هر دوی این‌ها یک partial class یا به عبارتی «یک کلاس» بیشتر نیستند. «فکر می‌کنیم» که از هم جدا شدند اما واقعا یکی هستند. شما در code behind صفحه به صورت مستقیم با عناصر رابط کاربری سروکار دارید و کدهای شما به این عناصر گره خورده‌اند.
شاید بپرسید که چه اهمیتی دارد؟
مشکل اول: امکان نوشتن آزمون‌ها واحد برای این متدها وجود ندارد یا بسیار سخت است. این متدها فقط با وجود فرم و رابط کاربری متناظر با آن‌ها هست که معنا پیدا می‌کنند و تک تک عناصر آن‌ها وهله سازی می‌شوند.
مشکل دوم: کد نوشته فقط برای همین فرم جاری آن قابل استفاده است؛ چون به صورت صریح به عناصر موجود در فرم اشاره می‌کند. نمی‌تونید این فایل code behind رو بردارید بدون هیچ تغییری برای فرم دیگری استفاده کنید.
مشکل سوم: نمی‌تونید طراحی فرم رو بدید به یک نفر، کد نویسی اون رو به شخصی دیگر. چون ایندو لازم و ملزوم یکدیگرند.

این سیستم کد نویسی دهه 90 است.
چند سالی است که طراحان سعی کرده‌اند این سیستم رو دور بزنند و روش‌هایی رو ارائه بدن که در آن‌ها فرم‌های برنامه و فایل‌های پیاده سازی کننده‌ی منطق آن هیچگونه ارتباط مستقیمی باهم نداشته باشند؛ به هم گره نخورده باشند؛ ارجاعی به هیچیک از عناصر بصری فرم را در خود نداشته باشند. به همین دلیل ASP.NET MVC به وجود آمده و در همان سال‌ها مثلا MVVM .

سؤال:
الان که رابط کاربری از فایل پیاده سازی کننده منطق آن جدا شده و دیگر Code behind هم نیست (همان partial class های متداول)، این فایل‌ها چطور متوجه می‌شوند که مثلا روی یک فرم، شیءایی قرار گرفته؟ از کجا متوجه خواهند شد که روی دکمه‌ای کلیک شده؟ این‌ها که ارجاعی از فرم را در درون خود ندارند.
در الگوی MVVM این سیم کشی توسط امکانات قوی Binding موجود در WPF میسر می‌شود. در ASP.NET MVC چیزی شبیه به آن به نام Model binder و همان مکانیزم‌های استاندارد HTTP این کار رو می‌کنه. در MVVM شما بجای code behind خواهید داشت ViewModel (اسم جدید آن). در ASP.NET MVC این اسم شده Controller. بنابراین اگر این اسامی رو شنیدید زیاد تعجب نکنید.
این‌ها همان Code behind قدیمی هستند اما ... بدون داشتن ارجاعی از رابط کاربری در خود که ... اطلاعات موجود در فرم به نحوی به آن‌ها Bind و ارسال می‌شوند.
این سیم کشی‌ها هم نامرئی هستند. یعنی فایل ViewModel یا فایل Controller نمی‌دونند که دقیقا از چه کنترلی در چه فرمی این اطلاعات دریافت شده.
این ایده هم جدید نیست. شاید بد نباشه به دوران طلایی Win32 برگردیم. همان توابع معروف PostMessage و SendMessage را به خاطر دارید؟ شما در یک ترد می‌تونید با مثلا PostMessage شیءایی رو به یک فرم که در حال گوش فرا دادن به تغییرات است ارسال کنید (این سیم کشی هم نامرئی است). بنابراین پیاده سازی این الگوها حتی در Win32 و کلیه فریم ورک‌های ساخته شده بر

پایه آن‌ها مانند WinForms ، VB6 ، VCL و غیره ... «از روز اول» وجود داشته و می‌تونستند بعد از 10 سال نیان بگن که اون روش‌های RAD ایی رو که ما پیشنهاد دادیم، می‌شد خیلی بهتر از همان ابتدا، طور دیگری پیاده سازی بشه.

ادامه بحث!
این سیم کشی یا اصطلاحا Binding ، در مورد رخدادها هم در WPF وجود داره و اینبار به نام Commands معرفی شده‌است. به این معنا که بجای اینکه بنویسید:

```
<Button  Click="btnClick_Event">Last</Button>
```

بنویسید:

```
<Button Command="{Binding GoLast}">Last</Button>
```

حالا باید مکانیزمی وجود داشته باشه تا این پیغام رو به ViewModel برنامه برساند. اینکار با پیاده سازی اینترفیس ICommand قابل انجام است که معرفی یک کلاس عمومی از پیاده سازی آن‌را در ابتدای بحث مشاهده نمودید.
در یک DelegateCommand، توسط متد منتسب به executeAction، مشخص خواهیم کرد که اگر این سیم کشی برقرار شد (که ما دقیقا نمی‌دانیم و نمی‌خواهیم که بدانیم از کجا و کدام فرم دقیقا)، لطفا این اعمال را انجام بده و توسط متد منتسب به canExecute به سیستم Binding خواهیم گفت که آیا مجاز هستی این اعمال را انجام دهی یا خیر. اگر این متد false برگرداند، مثلا دکمه یاد شده به صورت خودکار غیرفعال می‌شود.
اما مشکل کلاس DelegateCommand ذکر شده هم دقیقا همینجا است. این دکمه تا ابد غیرفعال خواهد ماند. در WPF کلاسی وجود دارد به نام CommandManager که حاوی متدی استاتیکی است به نام InvalidateRequerySuggested . اگر این متد به صورت دستی فراخوانی شود، یکبار دیگر کلیه متدهای منتسب به تمام canExecute های تعریف شده، به صورت خودکار اجرا می‌شوند و اینجا است که می‌توان دکمه‌ای را که باید مجددا بر اساس شرایط جاری تغییر وضعیت پیدا کند، فعال کرد. بنابراین فراخوانی متد InvalidateRequerySuggested یک راه حل کلی رفع نقیصه‌ی ذکر شده است.
راه حل دومی هم برای حل این مشکل وجود دارد. می‌توان از رخدادگردان CommandManager.RequerySuggested استفاده کرد. روال منتسب به این رخدادگردان هر زمانی که احساس کند تغییری در UI رخ داده، فراخوانی می‌شود. بنابراین پیاده سازی بهبود یافته کلاس DelegateCommand به صورت زیر خواهد بود:

```
using System;
using System.Windows.Input;

namespace MvvmHelpers
{
    // Ref.
    // - http://johnpapa.net/silverlight/5-simple-steps-to-commanding-in-silverlight/
    // - http://joshsmithonwpf.wordpress.com/2008/06/17/allowing-commandmanager-to-query-your-icommand-
objects/
    public class DelegateCommand<T> : ICommand
    {
        readonly Func<T, bool> _canExecute;
        bool _canExecuteCache;
        readonly Action<T> _executeAction;

        public DelegateCommand(Action<T> executeAction, Func<T, bool> canExecute = null)
        {
            if (executeAction == null)
                throw new ArgumentNullException("executeAction");

            _executeAction = executeAction;
            _canExecute = canExecute;
        }

        public event EventHandler CanExecuteChanged
        {
            add { if (_canExecute != null) CommandManager.RequerySuggested += value; }
            remove { if (_canExecute != null) CommandManager.RequerySuggested -= value; }
```

```
        }

        public bool CanExecute(object parameter)
        {
            return _canExecute == null ? true : _canExecute((T)parameter);
        }

        public void Execute(object parameter)
        {
            _executeAction((T)parameter);
        }
    }
}
```

استفاده از آن هم در ViewModel ساده است. یکبار خاصیتی به این نام تعریف می‌شود. سپس در سازنده کلاس مقدار دهی شده و متدهای متناظر آن تعریف خواهند شد:

```
public DelegateCommand<string> GoLast { set; get; }

//in ctor
GoLast = new DelegateCommand<string>(goLast, canGoLast);

private bool canGoLast(string data)
{
    //ex.
    return ListViewGuiData.CurrentPage != ListViewGuiData.TotalPage - 1;
}

private void goLast(string data)
{
  //do something
}
```

مزیت کلاس DelegateCommand جدید هم این است که مثلا متد canGoLast فوق، به صورت خودکار با به روز رسانی UI ، فراخوانی و تعیین اعتبار مجدد می‌شود.

نظرات خوانندگان

نویسنده: mohammad azad
تاریخ: ۱۳۹۰/۰۹/۱۹ ۱۷:۱۰:۴۷

سلام.این کلاس تفاوتی با کلاس RelayCommand داره؟ من از mvvmlight استفاده می کنم جالب اینجاست این بررسی خودکار canexequte رو در ورژن آخریش که البته بتا هست برداشته شایدم فراموش کرده!!!!

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۹/۱۹ ۱۷:۲۱:۳۹

سلام؛ اگر از mvvm light استفاده می‌کنید نیازی به این نیست.
البته به نظر من این فریم ورک‌های تهیه شده فقط یک جمع آوری مطلب از وب هستند. بنابراین بهتر است کمی هم با پشت صحنه این‌ها آشنا شد که چرا مثلا از CommandManager.RequerySuggested استفاده شده. چرا روش جان پاپا که ابتدای بحث مثال زدم درست کار نمی‌کنه (حداقل برای WPF البته) یا مثلا متد CommandManager.InvalidateRequerySuggested چی هست و به چه علتی پیش بینی شده؟

نویسنده: mohammad azad
تاریخ: ۱۳۹۰/۰۹/۱۹ ۲۰:۱۴:۱۹

حق با شماست آقای نصیری.اما CommandManager.RequerySuggested تو آخریت نسخه mvvmlight استفاده نشده .البته نسخه 4.0 تا زمانی که من اطلاع داشتم نسخه بتا بود.برام جای سوال بود که تو ورژن های قبلی mvvmlight بود و بعد تو ورژن بالاتر حذف شد.البته شایدم فراموش شده یا هر چیز دیگه...سوالم هم این بود این کلاس با RelayCommand josh Smith چه فرقی داره؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۹/۱۹ ۲۰:۵۵:۰۷

RelayCommand موجود در mvvm light دقیقا نسخه معادل mvvm foundation است (یا بود یک زمانی).

نویسنده: امیری
تاریخ: ۱۳۹۰/۰۹/۲۰ ۰۰:۰۷:۵۷

درود بر شما؛ در راستای فرمایشات شما، ویدیوهای زیر میتونه برای دوستان جالب باشه:

http://channel9.msdn.com/posts/NYC-DevReady-MVVM-Session-2-of-5-Programming-with-MVVM-Part-1
http://channel9.msdn.com/posts/NYC-DevReady-MVVM-Session-3-of-5-Programming-with-MVVM-Part-2
تو بخش اول مفصلا راجع به DelegateCommand بحث شده؛ پاینده باشید.

نویسنده: Arcabdelahi
تاریخ: ۱۳۹۰/۰۹/۲۰ ۱۳:۱۸:۴۴

بسیار عالی با تشکر ازمطلب خوب شما که بسیار خوب توضیح داده اید.

نویسنده: mohsen bahrzadeh
تاریخ: ۱۳۹۰/۱۰/۰۵ ۰۰:۴۸:۰۱

راهی برای پیاده سازی این موضوع در سیلورلایت وجود دارد؟!!!

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۱۰/۰۵ ۰۱:۰۳:۳۹

command manager implementation for Silverlight

عنوان: **مروری سریع بر اصول مقدماتی MVVM**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۹/۲۱ ۰۰:۵۰:۰۰
آدرس: www.dotnettips.info
گروه‌ها: MVVM

در قسمت قبل
، فلسفه وجودی MVVM و MVC و امثال آن‌را به بیانی ساده مطالعه کردید. همچنین به اینجا رسیدیم که بجای نوشتن روال رخدادگردان، از Commands استفاده کنید.
در این قسمت «تفکر MVVM ایی» بررسی خواهد شد! بنابراین سطح این قسمت را هم مقدماتی درنظر بگیرید.

در سیستم متداول مایکروسافتی ما همیشه یک فرم داریم به همراه یک سری کنترل. برای استفاده از این‌ها هم در فایل code behind فرم مرتبط، امکان دسترسی به این کنترل‌ها وجود دارد. مثلا textBox1.Text یعنی ارجاعی مستقیم به شیء textBox1 و سپس دسترسی به خاصیت متنی آن.
«تفکر MVVM ایی» می‌گه که: خیر! اینکار رو نکنید؛ ارجاع مستقیم به یک کنترل روش کار من نیست! فرم رو طراحی کنید؛ برای هیچکدام از کنترل‌ها هم نامی را مشخص نکنید (برخلاف رویه متداول). یک فایل درست کنید به نام Model ، داخل آن معادل textBox1.Text را که می‌خواهید استفاده کنید، پیش بینی و تعریف کنید؛ مثلا Public string Name . همین!
ما نمی‌خواهیم بدانیم که اصلا textBox1 وجود خارجی دارد یا نه. ما فقط با خاصیت متنی آن که در ادامه نحوه‌ی سیم کشی آن‌را هم بررسی خواهیم کرد، کار داریم.
بنابراین بجای اینکه بنویسید:

```
<TextBox Name="txtName" />
```

که ممکن است بعدا وسوسه شوید تا از txtName.Text آن استفاده کنید، بنویسید:

```
<TextBox Text="{Binding Name}" />
```

این مهم‌ترین قسمت «تفکر MVVM ایی» به زبان ساده است. یعنی قرار است تا حد ممکن از Binding استفاده کنیم. مثلا در قسمت قبل هم دیدید که بجای نوشتن روال رخدادگردان، فرمان مرتبط با آن‌را به جای دیگری Bind کردیم.

بنابراین تا اینجا Model ما به این شکل خواهد بود:

```
using System.ComponentModel;

namespace SL5Tests
{
    public class MainPageModel : INotifyPropertyChanged
    {
        string _name;
        public string Name
        {
            get { return _name; }
            set
            {
                if (_name == value) return;
                _name = value;
                raisePropertyChanged("Name");
            }
        }
```

```
        public event PropertyChangedEventHandler PropertyChanged;
        void raisePropertyChanged(string propertyName)
        {
            var handler = PropertyChanged;
            if (handler == null) return;
            handler(this, new PropertyChangedEventArgs(propertyName));
        }
    }
}
```

سؤال مهم:
تا اینجا یک فایل Model داریم که خاصیت Name در آن تعریف شده؛ یک فرم (View) هم داریم که فقط در آن نوشته شده Binding Name. الان این‌ها چطور به هم متصل خواهند شد؟
پاسخ: اینجا است که کلاس دیگری به نام ViewModel (همان فایل Code behind قدیمی است با این تفاوت که به هیچ فرم خاصی گره نخورده است و اصلا نمی‌داند که در برنامه فرمی وجود دارد یا نه)، کار خودش را شروع خواهد کرد:

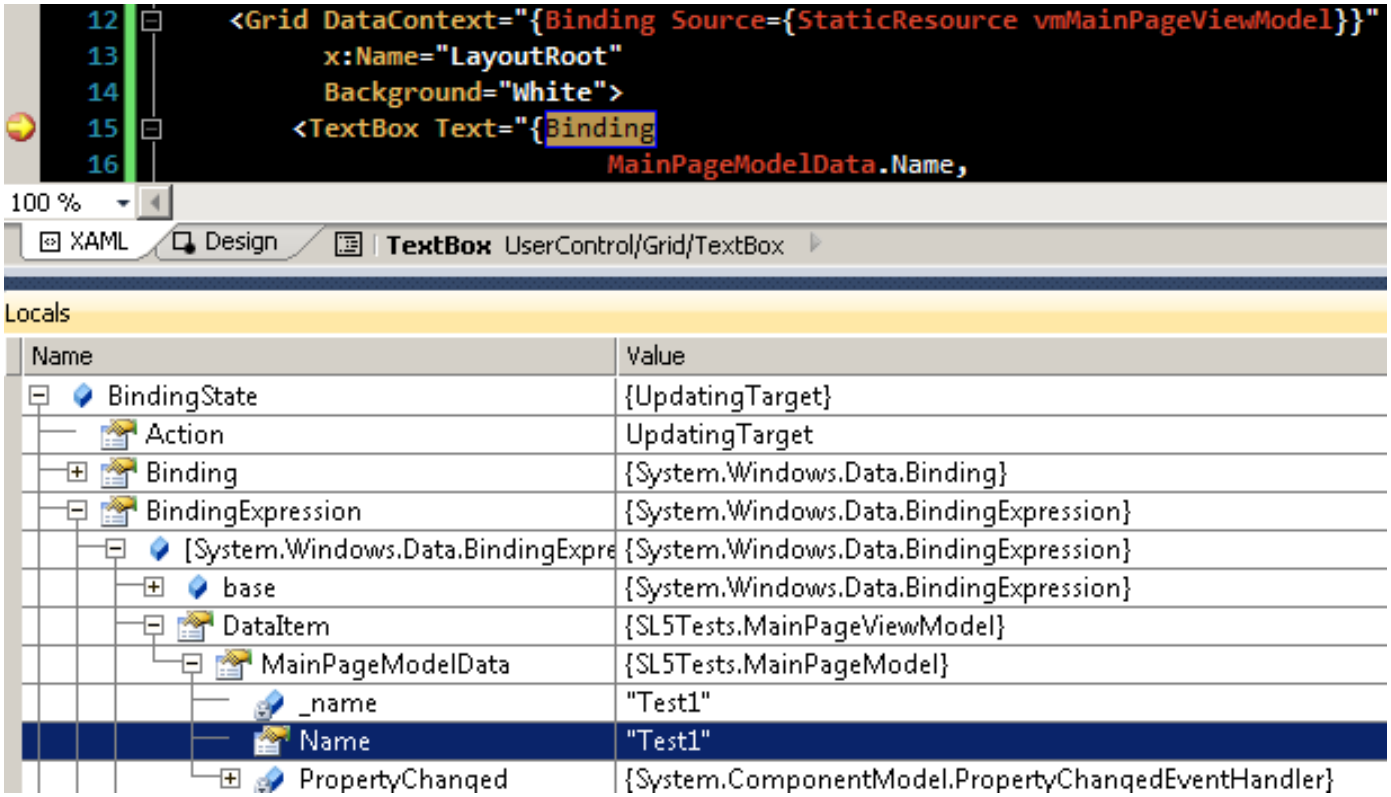
```
namespace SL5Tests
{
    public class MainPageViewModel
    {
        public MainPageModel MainPageModelData { set; get; }
        public MainPageViewModel()
        {
            MainPageModelData = new MainPageModel();
            MainPageModelData.Name = "Test1";
        }
    }
}
```

ما در این کلاس یک وهله از MainPageModel را ایجاد خواهیم کرد. اگر فرمی (که ما دقیقا نمی‌دانیم کدام فرم) در برنامه نیاز به یک ViewModel بر اساس مدل یاد شده داشت، می‌تواند آن‌را مورد استفاده قرار دهد. مقدار دهی آن در ViewModel موجب مقدار دهی خاصیت Text در فرم مرتبط خواهد شد و برعکس (البته به شرطی که مدل ما INotifyPropertyChanged را پیاده سازی کرده باشد و در فرم برنامه Binding Mode دو طرفه تعریف شود).

در قسمت بعد هم کار اتصال نهایی صورت می‌گیرد:
ابتدا xmlns:VM تعریف می‌شود تا بتوان به ViewModelها در طرف XAML دسترسی پیدا کرد. سپس در قسمت مثلا UserControl.Resources، این ViewModel را تعریف کرده و به عنوان DataContext بالاترین شیء فرم مقدار دهی خواهیم کرد:

```
<UserControl x:Class="SL5Tests.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:VM="clr-namespace:SL5Tests"
    mc:Ignorable="d" Language="fa"
    d:DesignHeight="300" d:DesignWidth="400">
    <UserControl.Resources>
        <VM:MainPageViewModel x:Name="vmMainPageViewModel" />
    </UserControl.Resources>
    <Grid DataContext="{Binding Source={StaticResource vmMainPageViewModel}}"
          x:Name="LayoutRoot"
          Background="White">
        <TextBox Text="{Binding
                            MainPageModelData.Name,
                            Mode=TwoWay,
                            UpdateSourceTrigger=PropertyChanged}" />
    </Grid>
</UserControl>
```

اکنون اگر یک breakpoint روی این سطر Binding قرار دهیم و برنامه را اجرا کنیم، جزئیات این سیم کشی را در عمل بهتر می‌توان مشاهده کرد:



البته این قابلیت قرار دادن breakpoint روی Bindingهای تعریف شده در View فعلا به سیلورلایت 5 اضافه شده و هنوز در WPF موجود نیست.

حداقل مزیتی را که اینجا می‌توان مشاهده کرد این است که فایل MainPageViewModel چون نمی‌داند که قرار است در کدام View وهله سازی شود، به سادگی در Viewهای دیگر نیز قابل استفاده خواهد بود یا تغییر و تعویض کلی View آن کار ساده‌ای است.
Commanding قسمت قبل را هم اینجا می‌شود اضافه کرد. تعاریف DelegateCommandهای مورد نیاز در ViewModel قرار می‌گیرند.
مابقی عملیات تفاوتی نمی‌کند و یکسان است.

نظرات خوانندگان

نویسنده: hossein moradinia
تاریخ: ۱۳۹۰/۰۹/۲۱ ۱۲:۲۰:۱۳

کاملا واضحه که الگوی MVVM برای جداسازی رابط کاربری نرم افزار (View) از مدل برنامه طراحی شده. هچنین میدونیم که الگویی به نام Repository وجود داره که بر روی ORM برای مثال Entity Framework پیاده میشه و عملکرد این دو الگو متفاوت هست.مزایای استفاده از Repository هم که مشخصه...
حال سوال اینجاست که آیا میشه از این دو الگو در کنار هم دیگر استفاده کرد؟!!!

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۹/۲۱ ۱۲:۳۸:۰۲

بله. در همان ViewModel عنوان شده، الگوی مخزن را با توجه به وجود مثلا شیء MainPageModelData فراخوانی و مقدار دهی کنید.

نویسنده: Ahmadxml
تاریخ: ۱۳۹۰/۰۹/۲۱ ۱۳:۲۸:۲۲

بسیار عالی و قابل فهم

نویسنده: alireza
تاریخ: ۱۳۹۰/۰۹/۲۱ ۱۹:۱۸:۴۷

سلام
تشکر از مطالب بسیار مفیدتون
من چندین ساله دارم برنامه نویسی میکنم و جدیدن با تکنولوژی WPF آشناشدم. متاسفانه هر مرجعی که برای یادگیری این اصول (MVVM , WPF,...) که بکار بردم در اول کار مطالب خیلی پیش پا افتاده رو بیان میکنند و بدون گفتن پیش زمینه های لازم وارد مباحث بسیار سنگین میشن. که باعث میشه آموزنده از مطلب زده بشه.(یه جورایی هم احساس حقارت در مورد سواد کم خودش بهش دست بده) در هر صورت من علاقه بسیار زیادی به برنامه نویسی داشتم و دارم وخیلی دوست دارم با تکنولوژی های جدید بیشتر کار کنم و سبک کاریم رو بروز کنم. از شما که در این زمینه تجربه کافی دارین میخوام لطف کنین یک منبع و مرجع برای یادگیری این مباحث (مباحث جدید که یادگیریش واسه برنامه نویسی الان از نون شب واجب تره) چه فارسی چه انگلیسی معرفی کنین. لازم به ذکره که مطالب آموزشی که خود شما میذارید تقریبا از سواد الان من فراتره و خیلی از قسمت هاشو درک نمیکنم(که قطعا به خاطر سواد کم من در این زمینه است).
پیشاپیش از لطفتون ممنونم.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۹/۲۱ ۱۹:۳۰:۱۳

پیشنیاز MVVM مباحث Binding در Silverlight و WPF است. یک کتاب فارسی رو در این زمینه در اینجا می‌تونید دریافت کنید: (^)
مرتبط با سیلورلایت است اما ... مباحث کلی آن با WPF تفاوتی ندارد و اصول یکی است.

نویسنده: Alimomen54
تاریخ: ۱۳۹۰/۰۹/۲۶ ۱۳:۵۹:۳۵

سلام
از مطالب خوبتون تشکر می کنم. دیگه تقریبا مشتری ثابت و ساعتی سایتتون شدم.
mvvm & wpf هنوز تبدیل به یه ابزار کامل برای تولید یه برنامه کاربردی حرفه ای نشده. من پوستم کنده شد تا تونستم یه برنامه

کامل باهاش بنویسم. واسه راتباطش با ssrs چه مشقاتی که نکشیدم.
ناچار شدم یه فرم ویندوزی به برنامه اضافه کنم و گزارش را توی اون نمایش بدم.
راهم درسته فعلا؟ در نسخه 2011 فکری به حال این مشکل نشده؟
راستی چطور میشه توسط یه کلید و بدون نوشتم کد یه ویو جدید را نمایش داد. راهی برای بایند کردن کلید به ویو وجود داره.
بدون نوشتن Command ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۹/۲۶ ۱۷:۱۴:۲۴

احتمالا این مطلب راهبری برای شما مفید باشد.

نویسنده: saman
تاریخ: ۱۳۹۱/۰۴/۰۴ ۲۳:۵۵

سلام.با تشکر از مطالب مفیدتون.راستش رو بخواین من نمیدونم باید اینجا سوالم رو مطرح کنم یانه؟چون تاپیک مرتبطتری پیدا نکردم.
من با الگوی MVVMکار میکنم.برای نمایش خطاهای اعتبار سنجی هم از IDataErrorInfo استفاده کردم.
مشکل من اینجاست که وقتی یک پروپرتی از نوع int رو به یکی از تکست باکسهام بایند میکنم و میخوام که کاربر مقدار اون فیلد رو همیشه پر کنه یعنی not nullable
هستش.وقتی متن داخل تکست باکس رو پاک میکنم بجای خطای در نظر گرفته شده براش عبارت زیر داخل tooltipنمایش داده میشه:
value "" could not be converted
ممنون میشم اگه راهنماییم کنین.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۰۵ ۰:۱۳

باید از ValidationRules استفاده کنید. مثلا : (^)

عنوان: **MVVM و رویدادگردانی**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۹/۲۳ ۱۹:۰۷:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: MVVM

در دو قسمت قبل به اینجا رسیدیم که بجای شروع به کدنویسی مستقیم در code behind یک View (یک پنجره، یک user control ...)، کلاس مجزای دیگری را به نام ViewModel به برنامه اضافه خواهیم کرد و این کلاس از وجود هیچ فرمی در برنامه مطلع نیست.
بنابراین جهت انتقال رخدادها به ViewModel، بجای روش متداول تعریف روال‌های رخدادگردان در Code behind:

```
<Button  Click="btnClick_Event">Last</Button>
```

آن‌ها را با Commands به ViewModel ارسال خواهیم کرد:

```
<Button Command="{Binding GoLast}">Last</Button>
```

همچنین بجای اینکه مستقیما بخواهیم از طریق نام یک شیء به مثلا خاصیت متنی آن دسترسی پیدا کنیم:

```
<TextBox Name="txtName" />
```

از طریق Binding، اطلاعات مثلا متنی آن‌را به ViewModel منتقل خواهیم کرد:

```
<TextBox Text="{Binding Name}" />
```

و همینجا است که 99 درصد آموزش‌های MVVM موجود در وب به پایان می‌رسند؛ البته پس از مشاهده 10 تا 20 ویدیو و خواندن بیشتر از 30 تا مقاله! و اینجا است که خواهید گفت: فقط همین؟! با این‌ها میشه یک برنامه رو مدیریت کرد؟!
البته همین‌ها برای مدیریت قسمت عمده‌ای از اکثر برنامه‌ها کفایت می‌کنند؛ اما خیلی از ریزه کاری‌ها وجود دارند که به این سادگی‌ها قابل حل نیستند و در طی چند مقاله به آن‌ها خواهیم پرداخت.

سؤال: در همین مثال فوق، اگر متن ورودی در TextBox تغییر کرد، چگونه می‌توان بلافاصله از تغییرات آن در ViewModel مطلع شد؟ قدیم‌ترها می‌شد نوشت:

```
<TextBox TextChanged="TextBox_TextChanged" />
```

اما الان که قرار نیست در code behind کد بنویسیم (تا حد امکان البته)، باید چکار کرد؟
پاسخ: امکان Binding به TextChanged وجود ندارد، پس آن‌را فراموش می‌کنیم. اما همان Binding معمولی را به این صورت هم می‌شود نوشت (همان مثال قسمت قبل):

```
<TextBox Text="{Binding
                            MainPageModelData.Name,
                            Mode=TwoWay,
                            UpdateSourceTrigger=PropertyChanged}" />
```

و نکته مهم آن UpdateSourceTrigger است. اگر روی حالت پیش فرض باشد، ViewModel پس از تغییر focus از این TextBox به کنترلی دیگر، از تغییرات آگاه خواهد شد. اگر آن‌را صریحا ذکر کرده و مساوی PropertyChanged قرار دهیم (این مورد در سیلورلایت 5 جدید است؛ هر چند از روز نخست WPF وجود داشته است)، با هر تغییری در محتوای TextBox، خاصیت MainPageModelData.Name به روز رسانی خواهد شد.
اگر هم بخواهیم این تغییرات آنی‌را در ViewModel تحت نظر قرار دهیم، می‌توان نوشت:

```
using System.ComponentModel;

namespace SL5Tests
{
    public class MainPageViewModel
    {
        public MainPageModel MainPageModelData { set; get; }
        public MainPageViewModel()
        {
            MainPageModelData = new MainPageModel();
            MainPageModelData.Name = "Test1";
            MainPageModelData.PropertyChanged += MainPageModelDataPropertyChanged;
        }

        void MainPageModelDataPropertyChanged(object sender, PropertyChangedEventArgs e)
        {
            switch (e.PropertyName)
            {
                case "Name":
                    //do something
                    break;
            }
        }
    }
}
```

تعریف MainPageModel را در قسمت قبل مشاهده کرده‌اید و این کلاس اینترفیس INotifyPropertyChanged را پیاده سازی می‌کند. بنابراین می‌توان از رویدادگردان PropertyChanged آن در ViewModel هم استفاده کرد.
به این ترتیب همان کار رودیدادگردان TextChanged را اینطرف هم می‌توان شبیه سازی کرد و تفاوتی نمی‌کند. البته با این تفاوت که در ViewModel فقط به اطلاعات به روز موجود در MainPageModelData.Name دسترسی داریم، اما نمی‌دانیم و نمی‌خواهیم هم بدانیم که منبع آن دقیقا کدام شیء رابط کاربری برنامه است.

سؤال: ما قبلا مثلا می‌توانستیم بررسی کنیم که اگر کاربر حین تایپ در یک TextBox بر روی دکمه‌ی Enter کلیک کرد، آن‌گاه برای نمونه، جستجویی بر اساس اطلاعات وارد شده صورت گیرد. الان این فشرده شدن دکمه‌ی Enter را چگونه دریافت و چگونه به ViewModel ارسال کنیم؟
این مورد کمی پیشرفته‌تر از حالت‌های قبلی است. برای حل این مساله ابتدا باید UpdateSourceTrigger یاد شده را مساوی Explicit قرار داد. یعنی اینبار می‌خواهیم نحوه ی به روز رسانی خاصیت MainPageModelData.Name را از طریق Binding خودمان مدیریت کنیم. این مدیریت کردن هم با استفاده از امکاناتی به نام Attached properties قابل انجام است که به آن‌ها Behaviors هم می‌گویند. مثلا:

```
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
```

```
namespace SL5Tests
{
    public static class InputBindingsManager
    {
        public static readonly DependencyProperty UpdatePropertySourceWhenEnterPressedProperty
                    = DependencyProperty.RegisterAttached(
                                "UpdatePropertySourceWhenEnterPressed",
                                typeof(bool),
                                typeof(InputBindingsManager),
                                new PropertyMetadata(false,
OnUpdatePropertySourceWhenEnterPressedPropertyChanged));

        static InputBindingsManager()
        { }

        public static void SetUpdatePropertySourceWhenEnterPressed(DependencyObject dp, bool value)
        {
            dp.SetValue(UpdatePropertySourceWhenEnterPressedProperty, value);
        }

        public static bool GetUpdatePropertySourceWhenEnterPressed(DependencyObject dp)
        {
            return (bool)dp.GetValue(UpdatePropertySourceWhenEnterPressedProperty);
        }

        private static void OnUpdatePropertySourceWhenEnterPressedPropertyChanged(DependencyObject dp,
    DependencyPropertyChangedEventArgs e)
        {
            var txt = dp as TextBox;
            if (txt == null)
                return;

            if ((bool)e.NewValue)
            {
                txt.KeyDown += HandlePreviewKeyDown;
            }
            else
            {
                txt.KeyDown -= HandlePreviewKeyDown;
            }
        }

        static void HandlePreviewKeyDown(object sender, KeyEventArgs e)
        {
            if (e.Key != Key.Enter) return;

            var txt = sender as TextBox;
            if (txt == null)
                return;

            var binding = txt.GetBindingExpression(TextBox.TextProperty);
            if (binding == null) return;
            binding.UpdateSource();
        }
    }
}
```

تعریف Attached properties یک قالب استاندارد دارد که آن را در کد فوق ملاحظه می‌کنید. یک تعریف به صورت static و سپس تعریف متدهای Get و Set آن. با تغییر مقدار آن که اینجا از نوع bool تعریف شده، متد OnUpdatePropertySourceWhenEnterPressedPropertyChanged به صورت خودکار فراخوانی می‌شود. اینجا است که ما از طریق آرگومان dp به textBox جاری دسترسی کاملی پیدا می‌کنیم. مثلا در اینجا بررسی شده که آیا کلید فشرده شده enter است یا خیر. اگر بله، یک سری فرامین را انجام بده. به عبارتی ما توانستیم، قطعه کدی را به درون شیءایی موجود تزریق کنیم. Txt تعریف شده در اینجا، واقعا همان کنترل TextBox ایی است که به آن متصل شده‌ایم.

و برای استفاده از آن خواهیم داشت:

```
<UserControl x:Class="SL5Tests.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
```

```
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:VM="clr-namespace:SL5Tests"
    mc:Ignorable="d" Language="fa"
    d:DesignHeight="300" d:DesignWidth="400">
    <UserControl.Resources>
        <VM:MainPageViewModel x:Name="vmMainPageViewModel" />
    </UserControl.Resources>
    <Grid DataContext="{Binding Source={StaticResource vmMainPageViewModel}}"
          x:Name="LayoutRoot"
          Background="White">
        <TextBox Text="{Binding
                            MainPageModelData.Name,
                            Mode=TwoWay,
                            UpdateSourceTrigger=Explicit}"
                 VerticalAlignment="Top"
                 VM:InputBindingsManager.UpdatePropertySourceWhenEnterPressed="True"  />
    </Grid>
</UserControl>
```

همانطور که مشاهده می‌کنید، UpdateSourceTrigger به Explicit تنظیم شده و سپس InputBindingsManager.UpdatePropertySourceWhenEnterPressed به این کنترل متصل گردیده است. یعنی تنها زمانیکه در متد HandlePreviewKeyDown ذکر شده، متد UpdateSource فراخوانی گردد، خاصیت MainPageModelData.Name به روز رسانی خواهد شد (کنترل آن‌را خودمان در دست گرفته‌ایم نه حالت‌های از پیش تعریف شده).

این روش، روش متداولی است برای تبدیل اکثر حالاتی که Binding و Commanding متداول در مورد آن‌ها وجود ندارد. مثلا نیاز است focus را به آخرین سطر یک ListView از داخل ViewModel انتقال داد. در حالت متداول چنین امری میسر نیست، اما با تعریف یک Attached properties می‌توان به امکانات شیء ListView مورد نظر دسترسی یافت (به آن متصل شد، یا نوعی تزریق)، آخرین عنصر آن‌را یافته و سپس focus را به آن منتقل کرد یا به هر اندیسی مشخص که بعدا در ViewModel به این Behavior از طریق Binding ارسال خواهد شد.

عنوان: **استفاده از MVVM زمانیکه امکان Binding وجود ندارد**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۹/۲۶ ۱۸:۴۱:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: MVVM

ساده‌ترین تعریف MVVM، نهایت استفاده از امکانات Binding موجود در WPF و Silverlight است. اما خوب، همیشه همه چیز بر وفق مراد نیست. مثلا کنترل WebBrowser را در WPF در نظر بگیرید. فرض کنید که می‌خواهیم خاصیت Source آن‌را در ViewModel مقدار دهی کنیم تا صفحه‌ای را نمایش دهد. بلافاصله با خطای زیر متوقف خواهیم شد:

```
A 'Binding' cannot be set on the 'Source' property of type 'WebBrowser'.
A 'Binding' can only be set on a DependencyProperty of a DependencyObject.
```

بله؛ این خاصیت از نوع DependencyProperty نیست و نمی‌توان چیزی را به آن Bind کرد. بنابراین این نکته مهم را توسعه دهنده‌های کنترل‌های WPF و Silverlight همیشه باید بخاطر داشته باشند که اگر قرار است کنترل‌های شما MVVM friendly باشند باید کمی بیشتر زحمت کشیده و بجای تعریف خواص ساده دات نتی، خواص مورد نظر را از نوع DependencyProperty تعریف کنید.

الان که تعریف نشده چه باید کرد؟

پاسخ متداول آن این است: مهم نیست؛ خودمان می‌توانیم این‌کار را انجام دهیم! یک Attached property یا به عبارتی یک Behavior را تعریف و سپس به کمک آن عملیات Binding را میسر خواهیم ساخت. برای مثال:

در این Attached property قصد داریم یک خاصیت جدید به نام BindableSource را جهت کنترل WebBrowser تعریف کنیم:

```
using System;
using System.Windows;
using System.Windows.Controls;

namespace WebBrowserSample.Behaviors
{
    public static class WebBrowserBehaviors
    {
        public static readonly DependencyProperty BindableSourceProperty =
            DependencyProperty.RegisterAttached("BindableSource",
                            typeof(object),
                            typeof(WebBrowserBehaviors),
                            new UIPropertyMetadata(null, BindableSourcePropertyChanged));

        public static object GetBindableSource(DependencyObject obj)
        {
            return (string)obj.GetValue(BindableSourceProperty);
        }

        public static void SetBindableSource(DependencyObject obj, object value)
        {
            obj.SetValue(BindableSourceProperty, value);
        }

        public static void BindableSourcePropertyChanged(DependencyObject o,
DependencyPropertyChangedEventArgs e)
        {
            WebBrowser browser = o as WebBrowser;
            if (browser == null) return;

            Uri uri = null;

            if (e.NewValue is string)
            {
                var uriString = e.NewValue as string;
                uri = string.IsNullOrWhiteSpace(uriString) ? null : new Uri(uriString);
            }
            else if (e.NewValue is Uri)
```

```
            {
                uri = e.NewValue as Uri;
            }

            if (uri != null) browser.Source = uri;
        }
    }
}
```

یک مثال ساده از استفاده‌ی آن هم به صورت زیر می‌تواند باشد:
ابتدا ViewModel مرتبط با فرم برنامه را تهیه خواهیم کرد. اینجا چون یک خاصیت را قرار است Bind کنیم، همینجا داخل ViewModel آن‌را تعریف کرده‌ایم. اگر تعداد آن‌ها بیشتر بود بهتر است به یک کلاس مجزا مثلا GuiModel منتقل شوند.

```
using System;
using System.ComponentModel;

namespace WebBrowserSample.ViewModels
{
    public class MainWindowViewModel : INotifyPropertyChanged
    {
        Uri _sourceUri;
        public Uri SourceUri
        {
            get { return _sourceUri; }
            set
            {
                _sourceUri = value;
                raisePropertyChanged("SourceUri");
            }
        }

        public MainWindowViewModel()
        {
            SourceUri = new Uri(@"C:\path\arrow.png");
        }

        #region INotifyPropertyChanged Members
        public event PropertyChangedEventHandler PropertyChanged;
        void raisePropertyChanged(string propertyName)
        {
            var handler = PropertyChanged;
            if (handler == null) return;
            handler(this, new PropertyChangedEventArgs(propertyName));
        }
        #endregion
    }
}
```

در ادامه بجای استفاده از خاصیت Source که قابلیت Binding ندارد، از Behavior سفارشی تعریف شده استفاده خواهیم کرد.
ابتدا باید فضای نام آن تعریف شود، سپس BindableSource مرتبط آن در دسترس خواهد بود:

```
<Window x:Class="WebBrowserSample.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:VM="clr-namespace:WebBrowserSample.ViewModels"
        xmlns:B="clr-namespace:WebBrowserSample.Behaviors"
        Title="MainWindow" Height="350" Width="525">
    <Window.Resources>
        <VM:MainWindowViewModel x:Key="vmMainWindowViewModel" />
    </Window.Resources>
    <Grid DataContext="{Binding Source={StaticResource vmMainWindowViewModel}}">
        <WebBrowser B:WebBrowserBehaviors.BindableSource="{Binding SourceUri}" />
    </Grid>
</Window>
```

نمونه مشابه این مورد را در مثال « استفاده از کنترل‌های Active-X در WPF » پیشتر در این سایت دیده‌اید.

نظرات خوانندگان

نویسنده: ZB
تاریخ: ۱۳۹۰/۰۹/۲۷ ۱۳:۱۴:۲۳

سلام آقای نصیری
سوالی داشتم از حضورتون. این لایه بندی که شما انجام میدین در برنامه هاتون بر چه اساسی هست؟ من این برنامه قرار دادن پست از گوگل پلاس رو گرفتم و دیدم که پروژه های زیادی داخلشه ممنون میشم بفرمایید هر کدوم رو به چه دلیلی گذاشتین
با تشکر

نویسنده: ZB
تاریخ: ۱۳۹۰/۰۹/۲۷ ۱۳:۱۵:۱۴

منظورم اینه که لایه بندی شما یک جور MVVM گسترش یافته است؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۹/۲۷ ۱۳:۴۳:۳۵

خیر. همان MVVM متداول است. زمانیکه شما با MVVM کار می‌کنید خودبخود به Viewهایی می‌رسید که خبری از وجود Code behind که در اینجا به آن ViewModel گفته می‌شود ندارند. بنابراین راحت می‌شود این‌ها را جدا کرد. همچنین ViewModel ها رو هم می‌شود جدا کرد در یک پروژه Class library دیگر. این یکی از اهداف MVVM است. اینکه راحت بشود طراحی رابط کاربری را از کدنویسی جدا کرد. حداقل دو نفر به صورت جداگانه بتوانند روی رابط کاربری و کد نویسی مرتبط با آن کار کنند بدون اینکه نگران باشند چیزی را به هم می‌ریزند.

نویسنده: ZB
تاریخ: ۱۳۹۰/۰۹/۲۷ ۱۴:۴۳:۲۷

ممنونم آقای نصیری میشه یک توضیح یک خطی راجع به هر کدوم از پروژه های اون سیستم بفرمایید؟ تریس کد تو MVVM سخته و برای رسیدن به جریان کد خیلی باید وقت صرف کرد
با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۹/۲۷ ۱۶:۲۸:۴۰

روال متداول پروژه‌های MVVM این است که سه پوشه به نام‌های ViewModel ، Model و Views در آن‌ها درست می‌شود. چون این‌ها با کمک این الگو از هم جدا می‌شوند، امکان قرار دادن آن‌ها در پروژه‌های class library مجزا هم فراهم خواهد شد.
روال من به این صورت است که Model و ViewModel را در یک پروژه جدید Class library به نام infrastructure قرار می‌دم.
تمام View ها رو بجای یک پوشه در پروژه اصلی به یک class library دیگر به نام Shell منتقل می‌کنم.
Common هم یک سری کد خیلی عمومی مشترک است که عموما در infrastructure استفاده می‌شود.
خلاصه بجای سه تا پوشه در یک پروژه می‌شود سه تا پروژه Class library مجزا از هم داشت. به این ترتیب هم زمان کامپایل کاهش پیدا می‌کند چون اگر تمام این‌ها داخل یک پروژه باشد هربار باید کامپایل شوند. همچنین این جداسازی نگهداری برنامه رو هم ساده‌تر می‌کنه چون هر قسمت به صورت مجزا و خیلی مشخص نگهداری میشه.

نویسنده: Z_farzani
تاریخ: ۱۳۹۰/۰۹/۲۷ ۱۹:۵۵:۱۰

ممنون خیلی لطف کردین

نویسنده: hossein moradinia

تاریخ: ۱۳۹۰/۰۹/۳۰ ۱۷:۳۹:۳۲

در برنامه های تجاری لازم است بعد از واکشی داده ها از بانک اطلاعات ، محاسباتی بر روی این داده ها انجام شده و در نهایت اطلاعات جدید حاصل شده به صورت یک گزارش ، لیست نمودار و یا مواردی از این قبیل نمایش داده شود.
سوال اینجاست که در یک برنامه سیلورلایت که با مدل MVVM توسعه یافته ، عملیاتهای محاسباتی برنامه در کدام بخش انجام میگیرد.
لازم به ذکر است که در بعضی برنامه ها نیاز است قبل از ثبت اطلاعات در بانک نیز محاسباتی بر روی آنها انجام شده و سپس نتیجه حاصل شده در بانک قرار گیرد.حال این محاسبات کجای پروژه و در کدام لایه قرار میگیرند؟!

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۹/۳۰ ۱۸:۵۰:۴۷

خارج از ViewModel . در اینجا ViewModel فقط مصرف کننده‌ی نهایی منطقی است که در جای دیگری از برنامه در لایه‌ای دیگر تهیه می‌شود.

نویسنده: hossein moradinia
تاریخ: ۱۳۹۰/۱۰/۰۱ ۲۰:۳۰:۳۲

مرسی
ولی روش درست پیاده سازی این موضوع برای من قدری مشکل است.از این نظر که پیاده سازیی که انجام می شود فاقد استاندارد و الگوهای برنامه نویسی نباشد.در اهداف MVVM و جداسازی لایه های برنامه خللی وارد نکند.
آیا نمونه هایی از چنین پیاده سازی هایی وجود دارد؟!!

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۱۰/۰۱ ۲۱:۲۵:۴۷

مثلا: MVVM Sample for WCF RIA Services

عنوان: **MVVM و امکان استفاده از یک وهله از ViewModel جهت چند View مرتبط**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۹/۳۰ ۱۸:۵۳:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: MVVM

عموما هنگام طراحی یک View، خیلی زود به حجم انبوهی از کدهای XAML خواهیم رسید. در ادامه بررسی خواهیم کرد که چطور می‌توان یک View را به چندین View خرد کرد، بدون اینکه نیازی باشد تا از چندین ViewModel (یا همان code behind عاری از ارجاعات بصری سابق قرار گرفته در یک پروژه جدای دیگر) استفاده شود و تمام این View های خرد شده هم تنها از یک وهله از ViewModel ایی خاص استفاده کنند و با اطلاعاتی یکپارچه سروکار داشته باشند؛ یا در عمل یکپارچه کار کنند.
این مشکل از جایی شروع می‌شود که مثلا خرد کردن یک user control به چند یوزر کنترل، یعنی کار کردن با چند وهله از اشیایی متفاوت. هر چند نهایتا تمام این‌ها قرار است در یک صفحه در کنار هم قرار گیرند اما در عمل از هم کاملا مجزا هستند و اگر به ازای هر کدام یکبار ViewModel را وهله سازی کنیم، به مشکل برخواهیم خورد؛ چون هر وهله نسبت به وهله‌ای دیگر ایزوله است. اگر در یکی Name مثلا Test بود در دیگری ممکن است مقدار پیش فرض نال را داشته باشد؛ چون با چند وهله از یک کلاس، در یک فرم نهایی سروکار خواهیم داشت.

ابتدا Model و ViewModel ساده زیر را در نظر بگیرید:

```
using System.ComponentModel;

namespace SplittingViewsInMvvm.Models
{
    public class GuiModel : INotifyPropertyChanged
    {
        string _name;
        public string Name
        {
            get { return _name; }
            set
            {
                _name = value;
                raisePropertyChanged("Name");
            }
        }

        string _lastName;
        public string LastName
        {
            get { return _lastName; }
            set
            {
                _lastName = value;
                raisePropertyChanged("LastName");
            }
        }

        #region INotifyPropertyChanged Members
        public event PropertyChangedEventHandler PropertyChanged;
        void raisePropertyChanged(string propertyName)
        {
            var handler = PropertyChanged;
            if (handler == null) return;
            handler(this, new PropertyChangedEventArgs(propertyName));
        }
        #endregion
    }
}
```

```
using SplittingViewsInMvvm.Models;

namespace SplittingViewsInMvvm.ViewModels
{
    public class MainViewModel
    {
```

```
        public GuiModel GuiModelData { set; get; }

        public MainViewModel()
        {
            GuiModelData = new GuiModel();
            GuiModelData.Name = "Name";
            GuiModelData.LastName = "LastName";
        }
    }
}
```

سپس View زیر هم از این اطلاعات استفاده خواهد کرد:

```
<UserControl x:Class="SplittingViewsInMvvm.Views.Main"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             mc:Ignorable="d"
             xmlns:VM="clr-namespace:SplittingViewsInMvvm.ViewModels"
             d:DesignHeight="300" d:DesignWidth="300">
    <UserControl.Resources>
        <VM:MainViewModel x:Key="vmMainViewModel" />
    </UserControl.Resources>
    <StackPanel DataContext="{Binding Source={StaticResource vmMainViewModel}}">
        <GroupBox Margin="2" Header="Group 1">
            <TextBlock Text="{Binding GuiModelData.Name}" />
        </GroupBox>
        <GroupBox Margin="2" Header="Group 2">
            <TextBlock Text="{Binding GuiModelData.LastName}" />
        </GroupBox>
    </StackPanel>
</UserControl>
```

اکنون فرض کنید که می‌خواهیم Group 1 و Group 2 را جهت مدیریت ساده‌تر View اصلی در دو user control مجزا قرار دهیم؛ مثلا:

```
<UserControl x:Class="SplittingViewsInMvvm.Views.Group1"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             mc:Ignorable="d"
             d:DesignHeight="300" d:DesignWidth="300">
    <Grid>
        <GroupBox Margin="2" Header="Group 1">
            <TextBlock Text="{Binding GuiModelData.Name}" />
        </GroupBox>
    </Grid>
</UserControl>
```

و

```
<UserControl x:Class="SplittingViewsInMvvm.Views.Group2"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             mc:Ignorable="d"
             d:DesignHeight="300" d:DesignWidth="300">
    <Grid>
        <GroupBox Margin="2" Header="Group 2">
            <TextBlock Text="{Binding GuiModelData.LastName}" />
        </GroupBox>
    </Grid>
```

```
</UserControl>
```

اکنون اگر این دو را مجددا در همان View اصلی ساده شده قبلی قرار دهیم (بدون اینکه در هر user control به صورت جداگانه data context را تنظیم کنیم):

```
<UserControl x:Class="SplittingViewsInMvvm.Views.Main"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             mc:Ignorable="d"
             xmlns:V="clr-namespace:SplittingViewsInMvvm.Views"
             xmlns:VM="clr-namespace:SplittingViewsInMvvm.ViewModels"
             d:DesignHeight="300" d:DesignWidth="300">
    <UserControl.Resources>
        <VM:MainViewModel x:Key="vmMainViewModel" />
    </UserControl.Resources>
    <StackPanel DataContext="{Binding Source={StaticResource vmMainViewModel}}">
        <V:Group1 />
        <V:Group2 />
    </StackPanel>
</UserControl>
```

باز هم .... برنامه همانند سابق کار خواهد کرد و ViewModel وهله سازی شده در user control فوق به صورت یکسانی در اختیار هر دو View اضافه شده قرار می‌گیرد و نهایتا یک View یکپارچه را در زمان اجرا می‌توان مورد استفاده قرار داد. علت هم بر می‌گردد به مقدار دهی خودکار DataContext هر View اضافه شده به بالاترین DataContext موجود در Visual tree که ذکر آن الزامی نیست:

```
<UserControl x:Class="SplittingViewsInMvvm.Views.Main"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             mc:Ignorable="d"
             xmlns:V="clr-namespace:SplittingViewsInMvvm.Views"
             xmlns:VM="clr-namespace:SplittingViewsInMvvm.ViewModels"
             d:DesignHeight="300" d:DesignWidth="300">
    <UserControl.Resources>
        <VM:MainViewModel x:Key="vmMainViewModel" />
    </UserControl.Resources>
    <StackPanel DataContext="{Binding Source={StaticResource vmMainViewModel}}">
        <V:Group1 DataContext="{Binding}" />
        <V:Group2 DataContext="{Binding}"/>
    </StackPanel>
</UserControl>
```

بنابراین به صورت خلاصه زمانیکه از MVVM استفاده می‌کنید لازم نیست کار خاصی را جهت خرد کردن یک View به چند Sub View انجام دهید! فقط این‌ها را در چند User control جدا کنید و بعد مجددا به کمک فضای نامی که تعریف خواهد (مثلا V در اینجا) در همان View اصلی تعریف کنید. بدون هیچ تغییر خاصی باز هم برنامه همانند سابق کار خواهد کرد.

عنوان: **MVVM و الگوی ViewModel Locator**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۱۰/۰۴ ۱۴:۰۰:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: MVVM

اگر ViewModel را همان فایل code behind عاری از ارجاعاتی به اشیاء بصری بدانیم، یک تفاوت مهم را علاوه بر مورد ذکر شده نسبت به Code behind متداول خواهد داشت: وهله سازی آن باید دستی انجام شود و خودکار نیست.
اگر به ابتدای کلاس‌های code behind دقت کنید همیشه واژه‌ی partial قابل رویت است، به این معنا که این کلاس در حقیقت جزئی از همان کلاس متناظر با XAML ایی است که مشاهده می‌کنید؛ یا به عبارتی با آن یکی است. فقط جهت زیبایی یا مدیریت بهتر، در دو کلاس قرار گرفته‌اند اما واژه کلیدی partial این‌ها را نهایتا به صورت یکسان و یکپارچه‌ای به کامپایلر معرفی خواهد کرد.
بنابراین وهله سازی code behind هم خودکار خواهد بود و به محض نمایش رابط کاربری، فایل code behind آن هم وهله سازی می‌شود؛ چون اساسا و در پشت صحنه، از دیدگاه کامپایلر تفاوتی بین این دو وجود ندارد.

اکنون سؤال اینجا است که آیا می‌توان با ViewModel ها هم همین وهله سازی خودکار را به محض نمایش یک View متناظر، پیاده سازی کرد؟
البته صحیح آن این است که عنوان شود ViewModel متناظر با یک View و نه برعکس. چون روابط در الگوی MVVM از View به ViewModel به Model است و نه حالت عکس؛ مدل نمی‌داند که ViewModel ایی وجود دارد. ViewModel هم از وجود View ها در برنامه بی‌خبر است و این «بی‌خبری‌ها» اساس الگوهایی مانند MVP ، MVVM ، MVC و غیره هستند. به همین جهت شاعر در وصف ViewModel فرموده‌اند که:

ای در درون برنامه‌ام و View از تو بی خبر_____وز تو برنامه‌ام پر است و برنامه از تو بی خبر :)

پاسخ:
بله. برای این منظور الگوی دیگری به نام ViewModel Locator طراحی شده است؛ روش‌های زیادی برای پیاده سازی این الگو وجود دارند که ساده‌ترین آن‌ها مورد زیر است:
فرض کنید ViewModel ساده زیر را قصد داریم به کمک الگوی ViewModel Locator به View ایی تزریق کنیم:

```
namespace WpfViewModelLocator.ViewModels
{
    public class MainWindowViewModel
    {
        public string SomeText { set; get; }
        public MainWindowViewModel()
        {
            SomeText = "Data ...";
        }
    }
}
```

برای این منظور ابتدا کلاس ViewModelLocatorBase زیر را تدارک خواهیم دید:

```
using WpfViewModelLocator.ViewModels;

namespace WpfViewModelLocator.ViewModelLocator
{
    public class ViewModelLocatorBase
    {
        public MainWindowViewModel MainWindowVm
        {
            get { return new MainWindowViewModel(); }
```

```
        }
    }
}
```

در اینجا یک وهله از کلاس MainWindowViewModel توسط خاصیتی به نام MainWindowVm در دسترس قرار خواهد گرفت. برای اینکه بتوان این کلاس را در تمام Viewهای برنامه قابل دسترسی کنیم، آن‌را در App.Xaml تعریف خواهیم کرد:

```
<Application x:Class="WpfViewModelLocator.App"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:vml="clr-namespace:WpfViewModelLocator.ViewModelLocator"
             StartupUri="MainWindow.xaml">
    <Application.Resources>
        <vml:ViewModelLocatorBase x:Key="ViewModelLocatorBase" />
    </Application.Resources>
</Application>
```

اکنون فقط کافی است در View خود DataContext را به نحو زیر مقدار دهی کنیم تا در زمان اجرا به صورت خودکار بتوان به خاصیت MainWindowVm یاد شده دسترسی یافت:

```
<Window x:Class="WpfViewModelLocator.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525">
    <Grid DataContext="{Binding Path=MainWindowVm, Source={StaticResource ViewModelLocatorBase}}">
        <TextBlock Text="{Binding SomeText}" VerticalAlignment="Top" Margin="5" />
    </Grid>
</Window>
```

در مورد ViewModel ها و Viewهای دیگر هم به همین ترتیب خواهد بود. یک وهله از آن‌ها به کلاس ViewModelLocatorBase اضافه می‌شود. سپس Binding Path مرتبط به DataContext به نام خاصیتی که در کلاس ViewModelLocatorBase مشخص خواهیم کرد، Bind خواهد شد.

روش دوم:

اگر در اینجا بخواهیم Path را حذف کنیم و فقط دسترسی عمومی به ViewModelLocatorBase را ذکر کنیم، باید یک Converter نوشت (چون به این ترتیب می‌توان به اطلاعات Binding در متد Convert دسترسی یافت). سپس یک قرار داد را هم تعریف خواهیم کرد؛ به این صورت که ما در Converter به نام View دسترسی پیدا می‌کنیم (از طریق ریفلکشن). سپس نام viewModel ایی را که باید به دنبال آن گشت مثلا ViewName به علاوه کلمه ViewModel در نظر خواهیم گرفت. در حقیقت یک نوع Convection over configuration است:

```
using System;
using System.Globalization;
using System.Linq;
using System.Windows.Data;

namespace WpfViewModelLocator.ViewModelLocator
{
    public class ViewModelLocatorBaseConverter : IValueConverter
    {
        public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
        {
            //مقدار در اینجا همان مشخصات ویوو است
            if (value == null) return null;
```

```
            string viewTypeName = value.GetType().Name;

            //قرار داد ما است
            //ViewModel Name = ViewName + "ViewModel"
            string viewModelName = string.Concat(viewTypeName, "ViewModel");

            //یافتن اسمبلی که حاوی ویوو مدل ما است
            var asms = AppDomain.CurrentDomain.GetAssemblies();
            var viewModelAsmName = "WpfViewModelLocator"; //نام پروژه مرتبط
            var viewModelAsm = asms.Where(x => x.FullName.Contains(viewModelAsmName)).First();

            //یافتن این کلاس ویوو مدل مرتبط
            var viewModelType = viewModelAsm.GetTypes().Where(x =>
x.FullName.Contains(viewModelName)).FirstOrDefault();
            if (viewModelType == null)
                throw new InvalidOperationException(string.Format("Could not find view model '{0}'",
viewModelName));

            //وهله سازی خودکار آن
            return Activator.CreateInstance(viewModelType);
        }

        public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
        {
            throw new NotImplementedException();
        }
    }
}
```

کار این تبدیلگر بسیار ساده و واضح است. Value دریافتی، وهله‌ای از view است. پس به این ترتیب می‌توان نام آن‌را یافت. سپس قرارداد ویژه خودمان را اعمال می‌کنیم به این ترتیب که "ViewModel Name = ViewName + "ViewModel و سپس به دنبال اسمبلی که حاوی این نام است خواهیم گشت. آن‌را یافته، کلاس مرتبط را در آن پیدا می‌کنیم و در آخر، به صورت خودکار آن‌را وهله سازی خواهیم کرد.

اینبار تعریف عمومی این Conveter در فایل App.Xaml به صورت زیر خواهد بود:

```
<Application x:Class="WpfViewModelLocator.App"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:vml="clr-namespace:WpfViewModelLocator.ViewModelLocator"
             StartupUri="MainWindow.xaml">
    <Application.Resources>
        <vml:ViewModelLocatorBaseConverter x:Key="ViewModelLocatorBaseConverter" />
    </Application.Resources>
</Application>
```

و استفاده‌ی آن در تمام View های برنامه به شکل زیر می‌باشد (بدون نیاز به ذکر هیچ نام خاصی و بدون نیاز به کلاس ViewModelLocatorBase یاد شده در ابتدای مطلب):

```
<Window x:Class="WpfViewModelLocator.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        DataContext="{Binding RelativeSource={RelativeSource Self},
                              Converter={StaticResource ViewModelLocatorBaseConverter}}"
        Title="MainWindow" Height="350" Width="525">
    <Grid>
        <TextBlock Text="{Binding SomeText}" VerticalAlignment="Top" Margin="5" />
    </Grid>
</Window>
```

نظرات خوانندگان

نویسنده: hossein moradinia
تاریخ: ۱۳۹۰/۱۰/۰۴ ۲۲:۵۷:۱۹

خب حالا مزیب و یا بهتر بگم کاربرد این کجا میتونه باشه؟!!!

نویسنده: rahmat rezaei
تاریخ: ۱۳۹۰/۱۰/۰۴ ۲۳:۴۳:۳۰

دارم یه چیزی شبیه mvc در asp.net webpages طراحی میکنم که فقط از یک httpHandler استفاده شده. می خواستم با توجه به یک پارامتر در queryString، به طور خودکار کلاس مربوطه ساخته و اجرا شود. فکر کنم راهش همین مطلب شماست.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۱۰/۰۵ ۱۲:۴۵:۲۹

سورس ASP.NET MVC در سایت کدپلکس در دسترس هست. این مورد و نحوه طراحی باز آن، تابحال یک مزیت منحصربفردی رو به همراه داشته که میشه گفته بی سابقه هست:
چندین فریم ورک MVC جدید توسط برنامه نویس‌های مستقل برای ASP.NET طراحی شده.
مثلا:

FubuMVC

Nancy

Bistro MVC

OpenRasta

و ...

در کل این‌ها هم می‌تونه ایده‌ای باشه برای کسانی که نمی‌خواهند در چارچوب‌های بسته مایکروسافت کار کنند و علاقمند هستند کنترل بیشتری روی محصول نهایی داشته باشند.

نویسنده: A. Karimi
تاریخ: ۱۳۹۰/۱۰/۰۶ ۰۰:۴۹:۰۴

در خصوص MVC ظاهراً Razor بر خلاف ASP.NET MVC بسته است. Engineهایی شبیه به Razor اما Open Source (ترجیحاً با لایسنسهایی مثل MS-PL و نه GPL) می‌شناسید؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۱۰/۰۶ ۰۱:۰۵:۱۵

ASP.NET MVC طراحی فوق العاده‌ای داره. تقریبا تمام قسمت‌های اون قابل تعویض است منجمله View Engine آن. لیستی از موارد پیاده سازی شده رو می‌تونید اینجا پیدا کنید: (^)

نویسنده: A. Karimi
تاریخ: ۱۳۹۰/۱۰/۰۶ ۱۴:۳۰:۳۱

جالب بود ولی هیچ کدام شبیه Razor نبودند شاید Razor دارای پتنت باشد.

نویسنده: amiry
تاریخ: ۱۳۹۰/۱۰/۰۷ ۱۲:۵۷:۲۱

سلام. قبل از ASP.NET MVC من کاری شبیه به این رو با الگوبرداری از RoR انجام داده بودم. دو تا موضوع مطرحه: 1- اگه برای

خودتون اینکارو انجام میدید، خیلی عالیه؛ چون تجربه ی به شدت غنی و ارزشمندی هست. 2- اگه برای پروژه انجام میدید، اگه کارتون پروژه های معمول توی بازار باشه اصلا ارزش نداره و به دردسرش نمی ارزه؛ مگه اینکه برای یه پروژه ی بزرگ کار کنید که در مجموع و کلیت براتون مقرون به صرفه باشه. پاینده و پیروز باشید.

نویسنده: rahmat rezaei
تاریخ: ۱۳۹۰/۱۰/۰۹ ۱۵:۴۳:۰۴

خوشبختانه کارم هر چند در مراحل ابتدایی است و چون تنها روی آن کار میکنم اشکالات بسیاری دارد اما مورد توجه و استقبال فراوان شرکتی قرار گرفته و در یکی از پروژه های بزرگش این امکان را به من داده که کارم را با آن تست کنم و برنامه نویس های پروژه از این فریمورک استفاده کنند. از لحاظ مالی هم بد نبوده.
اما در کل نمی دانم چرا قالب های موجود مثل mvc هم راضیم نمی کند و احساس می کنم در فریمورک های تولید صفحات وب باید یک انقلاب اساسی صورت بگیرد و چیزهایی مثل mvc قدمهای اول هستند.

عنوان: **MVVM و رویدادگردانی - قسمت دوم**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۱۰/۰۷ ۱۰:۲۵:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: MVVM

قسمت اول این بحث و همچنین پیشنیاز آن‌را در اینجا و اینجا می‌توانید مطالعه نمائید.
همه‌ی این‌ها بسیار هم نیکو! اما ... آیا واقعا باید به ازای هر روال رویدادگردانی یک Attached property نوشت تا بتوان از آن در الگوی MVVM استفاده کرد؟ برای یکی دو مورد شاید اهمیتی نداشته باشد؛ اما کم کم با بزرگتر شدن برنامه نوشتن این Attached properties تبدیل به یک کار طاقت فرسا می‌شود و اشخاص را از الگوی MVVM فراری خواهد داد.
برای حل این مساله، تیم Expression Blend راه حلی را ارائه داده‌اند به نام Interaction.Triggers که در ادامه به توضیح آن پرداخته خواهد شد.
ابتدا نیاز خواهید داشت تا SDK مرتبط با Expression Blend را دریافت کنید: (^)
سپس با فایل System.Windows.Interactivity.dll موجود در آن کار خواهیم داشت.

یک مثال عملی:
فرض کنید می‌خواهیم رویداد Loaded یک View را در ViewModel دریافت کنیم. زمان وهله سازی یک ViewModel با زمان وهله سازی View یکی است، اما بسته به تعداد عناصر رابط کاربری قرار گرفته در View ، زمان بارگذاری نهایی آن ممکن است متفاوت باشد به همین جهت رویداد Loaded برای آن درنظر گرفته شده است. خوب، ما الان در ViewModel نیاز داریم بدانیم که چه زمانی کار بارگذاری یک View به پایان رسیده.
یک راه حل آن‌را در قسمت قبل مشاهده کردید؛ باید برای این کار یک Attached property جدید نوشت چون نمی‌توان Command ایی را به رویداد Loaded انتساب داد یا Bind کرد. اما به کمک امکانات تعریف شده در System.Windows.Interactivity.dll به سادگی می‌توان این رویداد را به یک Command استاندارد ترجمه کرد:

```
<Window x:Class="WpfEventTriggerSample.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:i="http://schemas.microsoft.com/expression/2010/interactivity"
        xmlns:vm="clr-namespace:WpfEventTriggerSample.ViewModels"
        Title="MainWindow" Height="350" Width="525">
    <Window.Resources>
        <vm:MainWindowViewModel x:Key="vmMainWindowViewModel" />
    </Window.Resources>
    <Grid DataContext="{Binding Source={StaticResource vmMainWindowViewModel}}">
        <i:Interaction.Triggers>
            <i:EventTrigger EventName="Loaded">
                <i:InvokeCommandAction Command="{Binding DoLoadCommand}"
                                       CommandParameter="I am loaded!" />
            </i:EventTrigger>
        </i:Interaction.Triggers>

        <TextBlock Text="Testing InvokeCommandAction..."
                   Margin="5" VerticalAlignment="Top" />
    </Grid>
</Window>
```

ابتدا ارجاعی به اسمبلی System.Windows.Interactivity.dll باید به پروژه اضافه شود. سپس فضای نام xmlns:i باید به فایل XAML جاری مطابق کدهای فوق اضافه گردد. در نهایت به کمک Interaction.Triggers آن، ابتدا نام رویداد مورد نظر را مشخص می‌کنیم (EventName) و سپس به کمک InvokeCommandAction، این رویداد به یک Command استاندارد ترجمه می‌شود.
ViewModel این View هم می‌تواند به شکل زیر باشد که با کلاس DelegateCommand آن در پیشنیازهای بحث جاری آشنا شده‌اید.

```
using WpfEventTriggerSample.Helper;
```

```
namespace WpfEventTriggerSample.ViewModels
{
    public class MainWindowViewModel
    {
        public DelegateCommand<string> DoLoadCommand { set; get; }
        public MainWindowViewModel()
        {
            DoLoadCommand = new DelegateCommand<string>(doLoadCommand, canDoLoadCommand);
        }

        private void doLoadCommand(string param)
        {
            //do something
        }

        private bool canDoLoadCommand(string param)
        {
            return true;
        }
    }
}
```

به این ترتیب حجم قابل ملاحظه‌ای از کد نویسی Attached properties مورد نیاز، به ساده‌ترین شکل ممکن، کاهش خواهد یافت. بدیهی است این Interaction.Triggers را جهت تمام عناصر UI ایی که حداقل یک رویداد منتسب تعریف شده داشته باشند، می‌توان بکار گرفت؛ مثلا تبدیل رویداد Click یک دکمه به یک Command استاندارد:

```
<Button>
            <i:Interaction.Triggers>
                <i:EventTrigger EventName="Click">
                    <i:InvokeCommandAction Command="{Binding DoClick}"
                                           CommandParameter="I am loaded!" />
                </i:EventTrigger>
            </i:Interaction.Triggers>
</Button>
```

نظرات خوانندگان

نویسنده: محمد صاحب
تاریخ: ۱۳۹۰/۱۰/۰۷ ۱۱:۱۱:۱۰

ممنون

سوال اول:
این قسمت رو من درست متوجه نشدم
<>
رویداد کلیک رو که میشه مستقیم بایند کرد؟

سوال دوم:
فکر میکنید موارد ی که کار با این الگو رو راحت میکنن بصورت توکار برای WPF و SL اضافه بشه.
دقیقا چیزی که تو MVC داریم مثلا ساختار پروژه و نحوه نامگذاری (اضافه کردن controller به نام و...)

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۱۰/۰۷ ۱۱:۳۴:۰۶

- برای مثال نمی‌شود نوشت Click = Binding DoLoadCommand ، به همین جهت نیاز هست تا این event handler را تبدیل به یک command استاندارد کرد تا در ViewModel قابل دسترسی شود.
در کل هدف من یک مثال کلی بود که بگم این همه جا کاربرد دارد، مثلا اینطوری هم میشه با آن کار کرد.
- مایکروسافت همین الان یک فریم ورک MVVM تمام عیار به نام PRISM دارد: (^)

عنوان: **MVVM و فراخوانی متدهای اشیاء View از طریق ViewModel**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۱۰/۱۰ ۱۲:۲۰:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: MVVM

ما در ViewModel دسترسی مستقیمی به هیچ یک از اشیاء موجود در View نداریم (و درستش هم همین است). الان فرض کنید که می‌خواهیم از طریق ViewModel یک View را ببندیم؛ مثلا متد Close آن پنجره را فراخوانی کنیم. به عبارتی در حالت کلی می‌خواهیم یکی از متدهای تعریف شده یکی از عناصر بصری موجود در View را از طریق ViewModel فراخوانی نمائیم.
برای حل این مساله از فایل‌های همان SDK مرتبط با Expression blend استفاده خواهیم کرد.

ابتدا ارجاعاتی را به اسمبلی‌های System.Windows.Interactivity.dll و Microsoft.Expression.Interactions.dll اضافه می‌کنیم.
سپس دو فضای نام مرتبط هم باید اضافه شوند:

```
xmlns:i="http://schemas.microsoft.com/expression/2010/interactivity"
  xmlns:ei="http://schemas.microsoft.com/expression/2010/interactions"
```

یک مثال عملی:
قصد داریم از طریق ViewModel ، پنجره‌ای را ببندیم. کدهای XAML این مثال را در ادامه مشاهده خواهید کرد:

```
<Window x:Class="WpfCallMethodActionSample.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:i="http://schemas.microsoft.com/expression/2010/interactivity"
        xmlns:ei="http://schemas.microsoft.com/expression/2010/interactions"
        xmlns:vm="clr-namespace:WpfCallMethodActionSample.ViewModels"
        Name="ThisWindow"
        Title="MainWindow" Height="350" Width="525">
    <Window.Resources>
        <vm:MainWindowViewModel x:Key="vmMainWindowViewModel" />
    </Window.Resources>
    <Grid DataContext="{Binding Source={StaticResource vmMainWindowViewModel}}">

        <Button Content="Save &amp;amp; Close" VerticalAlignment="Top" Margin="5">
            <i:Interaction.Triggers>
                <!--فراخوانی متدی در ویوو مدل-->
                <i:EventTrigger EventName="Click">
                    <ei:CallMethodAction
                            TargetObject="{Binding}"
                            MethodName="SaveButtonClicked" />
                </i:EventTrigger>

                <!--فراخوانی متدی در شیء جاری از طریق ویوو مدل-->
                <i:EventTrigger SourceObject="{Binding}"  EventName="CloseMainWindow">
                    <ei:CallMethodAction
                            TargetObject="{Binding ElementName=ThisWindow}"
                            MethodName="Close"/>
                </i:EventTrigger>
            </i:Interaction.Triggers>
        </Button>
    </Grid>
</Window>
```

همچنین ViewModel تعریف شده نیز همین چند سطر زیر است:

```
using System;

namespace WpfCallMethodActionSample.ViewModels
{
    public class MainWindowViewModel
    {
        public void SaveButtonClicked()
        {
            close();
        }

        public event EventHandler CloseMainWindow;
        private void close()
        {
            if (CloseMainWindow != null) CloseMainWindow(this, EventArgs.Empty);
        }
    }
}
```

توضیحات:
اگر به ViewModel دقت کنید خبری از DelegateCommand در آن نیست. بله، به کمک ترکیبی از EventTrigger و CallMethodAction می‌توان جایگزینی را جهت DelegateCommand معرفی شده در قسمت‌های قبل این سری مباحث MVVM ارائه داد.
EventTrigger در اینجا به این معنا است که اگر EventName ذکر شده رخ داد، آنگاه این اعمال را انجام بده. مثلا در اینجا CallMethodAction را فراخوانی کن.
CallMethodAction در اسمبلی Microsoft.Expression.Interactions.dll تعریف شده است و تنها متدی از نوع void و بدون پارامتر را می‌تواند به صورت خودکار فراخوانی کند (محدودیت مهم آن است).
اینکه این متد کجا قرار دارد، توسط TargetObject آن مشخص می‌شود. اگر TargetObject را مساوی Binding قرار دادیم، یعنی به دنبال متدی که در DataContext گرید وجود دارد بگرد. به عبارتی به صورت خودکار به SaveButtonClicked تعریف شده در ViewModel ما متصل خواهد شد و آن‌را فراخوانی می‌کند.

تا اینجا رخداد Click دکمه تعریف شده را به متد SaveButtonClicked موجود در ViewModel سیم کشی کردیم.

در مرحله بعد می‌خواهیم از طریق ViewModel ، متدی را در View فراخوانی کنیم. نکته آن هم پیشتر ذکر شد؛ TargetObject صحیحی را باید انتخاب کرد. در اینجا برای پنجره جاری نام ThisWindow تعریف شده است و از طریق تعریف:

```
TargetObject="{Binding ElementName=ThisWindow}"
```

به CallMethodAction خواهیم گفت که قرار است متد Close را در شیء ThisWindow فراخوانی کنی.
همچنین نحوه تعریف EventTrigger ما هم در اینجا برعکس شده است:

```
<i:EventTrigger SourceObject="{Binding}"  EventName="CloseMainWindow">
```

قبلا به دنبال مثلا رخداد Click یک دکمه بودیم، اکنون با توجه به SourceObject تعریف شده، در ViewModel به دنبال این رخداد که برای نمونه در اینجا CloseMainWindow نام گرفته خواهیم گشت.

بنابراین View اینبار به رخداد CloseMainWindow تعریف شده در ViewModel سیم کشی خواهد شد. اکنون اگر این رخداد در ViewModel فراخوانی شود، CallMethodAction متناظر فعال شده و متد Close پنجره را فراخوانی می‌کند.

نظرات خوانندگان

نویسنده: Milad
تاریخ: ۱۳۹۰/۱۰/۱۰ ۱۸:۳۴:۳۲

با سلام خدمت استاد نصیری
من مطالب شما رو در خصوص MVVM دنبال کردم و بسیار عالی بیان کردید که واقعاً از شما سپاسگذارم.
اما من بیشتر با ASP.Net برنامه می نویسم. سرچ کردم دیدم تو این آدرس http://aspnetmvvm.codeplex.com دیگه از سال 2009 به بعد MVVM ASP.net توسعه داده نشده. آیا دلیل خاصی داره؟
کلا برای ASP.net هم MVVM رو پیشنهاد میکنید یا خیر دنبال چیز دیگری باشم؟
ممنون از لطف شما.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۱۰/۱۰ ۱۹:۱۰:۵۳

علتش رو اینجا توضیح دادم: (^) .
ASP.NET Webforms از نظر مایکروسافت در رده Done قرار دارد. فقط این اواخر کمی «ماله کشی و صافکاری» روی آن انجام شده و خواهد شد.
ضمنا الگوی MVVM به درد ASP.NET نمی‌خوره. نیاز به سیستمی State full داره که سیستم‌های وب در این رده قرار نمی‌گیرند. ASP.NET اساسا Stateless است. به همین جهت در پروژه‌های وب تمایل به MVC بیشتر است تا هر الگوی دیگری.
همچنین یکی از اعضای تیم ASP.NET MVC ، اخیرا فریم ورک MVVM JavaScript ایی را به نام knockoutjs ارائه داده ((^)). علت ارائه برای جاوا اسکریپت هم دقیقا به State full آن بر می‌گردد، زمانیکه داخل مرورگر کاربر اجرا می‌شود. مانند Silverlight که آن هم State full است.

نویسنده: shahin kiassat
تاریخ: ۱۳۹۰/۱۰/۱۱ ۱۰:۱۰:۵۱

سلام.
آقای نصیری می تونم بپرسم چرا دیگه خلاصه اشتراکات رو منتشر نمی کنید ؟
ممنونم.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۱۰/۱۱ ۱۰:۳۱:۰۲

من هر از چندگاهی می‌ایستم و به کارهایی که کردم نگاه می‌کنم. بعد از 100 روز اینکار تقریبا هیچ بازخوردی نداشت. بنابراین حذف شد.
آخر امسال هم در مورد این وبلاگ تصمیم گیری می‌کنم. خصوصا به غیرعمومی کردنش.
باز بودن، باز کار کردن، سورس باز بودن، به اشتراک گذاری، انتشار مطلب و همه این‌ها خوب. عالی. من پشتیبانش هستم. اما در فضایی متقابل هست که معنا پیدا می‌کنه که من متاسفانه این فضا رو نمی‌بینم. جمع کسانی که این دور و اطراف فنی نویس هستند به 5 نفر نمیرسه.

نویسنده: shahin kiassat
تاریخ: ۱۳۹۰/۱۰/۱۱ ۱۰:۵۰:۳۱

آقای نصیری خلاصه اشتراک هایی که به اشتراک می گذاشتید مورد توجه خیلی ها قرار می گرفت این مسئله از +1 ها و رای هایی که می گرفت مشخص بود.
تا حدی که برای من و خیلی ها عادت شده بود که شب ها قبل از خواب ابتدا اشتراک های شما رو بررسی کنیم.
جدا از خلاصه اشتراک ها بدون تملق این وبلاگ "مهمترین" دلیل اندک پیشرفت من و امثال من که در ابتدای راه هستیم بوده .
امیدوارم همچنان به تلاشتون جهت پیشرفت بار فنی و فکر برنامه نویسان ایرانی ادامه بدید. و فضایی که ذکر کردید بیشتر از الآن

متقابل شود.
ممنون.

نویسنده: Hossein Raziee
تاریخ: ۱۳۹۰/۱۰/۱۱ ۱۱:۰۴:۳۰

با سلام.
وقتی که شاهین کیاست به من ایمیل زد و موضوع "خلاصه اشتراکهای روزانه" و پاسخی که جناب نصیری دادند رو به من اطلاع داد ابتدا گفتم نه! آخه چرا؟! ولی چند دقیقه بعد که کمی فکر کردم دیدم ایشون حق دارند. وقتی فقط یه نفر باشه که بروز باشه ، اطلاعات رو به اشتراک بگذاره و بازخوردی نبینه خسته ککنده هست.
متاسفانه این چند وقت اخیر اکثر دوستان دیگه هم مثل "بهروز راد" دست از نوشتن برداشتن. وبلاگ "مهدی موسوی" رو نگاه کنید. آخرین به روز رسانی برای چه تاریخی هست.
کسانی مثل مهدی موسوی ، وحید نصیری ، بهروز راد و دیگر دوستان که دانش زیادی برخوردار هستند ، گرفتاری هاشون هم بیشتر اما با تمام این گرفتاری ها به فکر ارتقاء سطح علمی دیگران هم هستند.
اما همونطور که جناب نصیری گفتند تعداد فنی نویسان بسیار کم هست.
وحید نصیری یکی از کسانی هست که من هر وقت میخوام به کسی یک فعال در زمینه ی IT رو معرفی کنم ایشون اولین نفر هست. من شخصا همیشه این بلاگ رو بررسی میکنم و از مطالب مفیدش استفاده میکنم.
امیدوارم که این وضع عوض بشه و همه ی دوستان اطلاعاتشون رو در هر سطحی که هست به اشتراک بگذارند. منظورم با خودم و شاهین هم هست. باید شروع کرد.
مرسی وحید نصیری.

نویسنده: rahmat rezaei
تاریخ: ۱۳۹۰/۱۰/۱۱ ۱۱:۱۳:۱۰

از طریق همین اشتراکهای روزانه من با خیلی از وبلاگهای ایرانی و خارجی و سایتهای مفید آشنا شدم.
یک راه میانبر بود برای تازه کارهایی مثل من.

نویسنده: hossein moradinia
تاریخ: ۱۳۹۰/۱۰/۱۱ ۱۲:۲۲:۵۶

من هم هر روز و هر شب چک میکنم ...
ای کاش بشه ادامه بدید ...

نویسنده: Javad Darvish Amiry
تاریخ: ۱۳۹۰/۱۰/۱۱ ۱۲:۳۸:۴۹

درود بر شما جناب نصیری و همینطور شاهین کیاست که لطف کردن و موضوع رو مطرح کردن.
من هم با جناب نصیری موافقم که ادامه چنین کاری نیاز به فضای متقابل و طبیعتا دلگرمی و انگیزه داره؛ اما صحبت اینجاست که «من اگر بنشینم، تو اگر بنشینی، چه کسی برخیزد؟». فکر میکنم به گفته Hossein Raziee باید یه حرکتی بهرحال انجام داد؛ اینطور که من برداشت کردم، فکر میکنم دو تا موضوع وبلاگ نویسی IT در ایران رو تحت الشعاع قرار داده: یکی گرایش به تالارهاست. و دومی گرایش به بلاگ های خارجی. اکثر برنامه نویسای قوی که دست به قلم هستن، توی تالارها فعالیت میکنن؛ ظاهرا اهمیت وبلاگ ها هنوز برای ما جا نیفتاده. دوستانی هم که یه مقداری از مشق و تمرین بالاتر میرن، معمولا نه تنها خودشون شروع به انتقال نمیکنن، بلکه حتی برای ارتقای خودشون و پیگیری مطالب سراغ بلاگ های غیر فارسی میرن؛ فکر میکنم نیاز به یه حرکت اصولی و درست و حساب شده داریم تا موضوع تولید محتوی IT به زبان فارسی رو تو یه مسیر درست بندازیم. یکی از راههایی که به ذهن من میرسید، ایجاد یه شبکه بین نویسنده های ایرانی بود؛ حدود یه سال پیش این موضوع رو با چند تا از دوستان اینترنتیم مطرح کردم که استقبال نشد. حالا دوباره از دوستان میپرسم که نظرشون چیه؟
نمونه بسیار مفید (ولی تنها یه شمّه) خلاصه اشتراک هایی هست که جناب نصیری منتشر میکنن؛ وقتی یه نفر مثل وحید نصیری لینکی رو منتشر میکنه من مطمئنم که مطلب مفیدی هست؛ پس حتما میرم و میخونم؛ و چه بسا جزو خوانندگان دائمی اون سایت

هم بشم؛ حالا عرض بنده اینه که این موضوع باید منسجم و شبکه ای باشه؛
با جناب نصیری موافقم؛ با شاهین هم موافقم؛ اما موافق این نیستم که آقای نصیری بگن میخوام بکشم کنار و من و شاهین کیاست هم بگیم نه اینکارو نکنید؛ چون وحید نصیری شاید امروز احساساتی بشه و بخاطر من و امثال من ادامه بده، اما فردا دوباره بی انگیزگی یا خستگی میاد سراغش و باز همین داستان؛
من فکر میکنم باید راه چاره رو پیدا کرد؛ راهی که به نظر من میرسید رو عرض کردم؛ از دوستان که تا جاییکه من میشناسم و مطالب رو دنبال میکنم، میدونم همشون از افراد تاثیر گذار در IT ایران و با دانش و تجربه زیاد هستن هم خواهش میکنم که هر راهی به ذهنشون میرسه عنوان کنن که با همفکری هم بتونیم به یه راه حل درست برسیم؛
به سهم خودم هم از جناب نصیری عمیقا تشکر میکنم و از پرحرفی و زیاده گوییم پوزش میخوام. پاینده باشید؛

نویسنده: Mohammad Safdel
تاریخ: ۱۳۹۰/۱۰/۱۱ ۱۳:۱۴:۴۳

سلام. من مدتی بود که به دلیل گرفتاری و درگیری زیاد فرصت بررسی وبلاگها و سایتها رو برای بروز شدن نداشتم ولی با دیدن خلاصه اشتراکهای شما خودمو ملزم کرده بودم که اونها را هر شب و یا اگه شب فرصت نمی کردم به عنوان اولین کار در روز بعد، مطالعه و بررسی کنم.
البته دلیل شما برای ادامه ندادن دقیقا همون دلیلی بود که دو سال پیش باعث شد تا منم وبلاگ نویسی را کنار بذارم. چه قصد ادامه دادن داشته باشید چه نداشته باشید به خاطر همه اون مطالب از شما واقعا ممنونم.

نویسنده: محمد صاحب
تاریخ: ۱۳۹۰/۱۰/۱۱ ۱۴:۲۸:۴۰

خبر خیلی بدی بود ...
امیدوارم به کسی بر نخوره ولی متاسفانه Leecher بودن داره تبدیل به یک فرهنگ میشه.

خوندن این مطلب هم خالی از لطف نیست
چگونه وبلاگ بخوانیم

نویسنده: Shima
تاریخ: ۱۳۹۰/۱۰/۱۱ ۱۹:۱۴:۲۴

استاد عزیز آخه چرا این تصمیم کشنده رو گرفتید؟؟!!
البته حق با شماست و کاملاً حرفاتون متین.
اما اگر شده بلاگ رو اشتراکی کنید و مثلاً اشتراک سالانه رو عدد X اعلام بفرمائید، اما لطفاً کنار نکشید. یک محل واقعاً علمی که از بهترین دانشگاهامون قوی تر هستش رو لطفاً ازمون نگیرید.
من ارشد IT میخونم تهران تربیت.م.د اما کل مطالبی که تو مدت تحصیل در مقطع کارشناسی و ارشد یاد گرفتن به مفت نمی ارزد و اساتیدش هم که ... اما از طریق وبلاگ و خلاصه اشتراکهای شما کلی مطالب به روز، به دردبخور، خلاصه و گل مطلب رو گرفتم.
البته خیلی وقتا چند بار باید یک پست شما را مطالعه کرد تا موضوع را فهمید که طبیعی است زیرا شما از بهترین اساتید ما که 2 کارشناسی اشد و یک دکتری داشت هزار پله بالاترید.
در کل لطفاً تجدید نظر فرموده و تنها دانشگاه به روز و بی منت کشور را از ما نگیرید.
ممنون

نویسنده: Shima 6489578840
تاریخ: ۱۳۹۰/۱۰/۱۱ ۱۹:۱۶:۰۵

استاد عزیز آخه چرا این تصمیم کشنده رو گرفتید؟؟!!البته حق با شماست و کاملاً حرفاتون متین.اما اگر شده بلاگ رو اشتراکی کنید و مثلاً اشتراک سالانه رو عدد X اعلام بفرمائید، اما لطفاً کنار نکشید. یک محل واقعاً علمی که از بهترین دانشگاهامون قوی تر هستش رو لطفاً ازمون نگیرید.من ارشد IT میخونم تهران تربیت.م.د اما کل مطالبی که تو مدت تحصیل در مقطع کارشناسی و ارشد یاد گرفتن به مفت نمی ارزد و اساتیدش هم که ... اما از طریق وبلاگ و خلاصه اشتراکهای شما کلی مطالب به روز، به

دردبخور، خلاصه و گل مطلب رو گرفتم. البته خیلی وقتا چند بار باید یک پست شما را مطالعه کرد تا موضوع را فهمید که طبیعی است زیرا شما از بهترین اساتید ما که 2 کارشناسی اشد و یک دکتری داشت هزار پله بالاترید.در کل لطفاً تجدید نظر فرموده و تنها دانشگاه به روز و بی منت کشور را از ما نگیرید.ممنون

نویسنده: Ahmadxml
تاریخ: ۱۳۹۰/۱۰/۱۱ ۲۰:۴۹:۵۵

سلام استاد عزیز
من یک برنامه نویس آماتور بودم که طی سه سال آشنایی با وبلاگ شما سطح خود را بالا بردم . طی این سالها به ندرت اتفاق افتاده که روزی مطالب شما رو نخونده باشم. بعضی از مطالب رو هم چندین و چند بار خوندم.NH رو از شما یاد گرفتم و در کل بهترین استاد کامپیوتر زندگی من هستید. من حاضرم برای استفاده از مطالب شما هزینه پردازم. بسته شدن این وبلاگ یعنی مرگ برنامه نویسان نیمه حرفه ای و حرفه ای.

نویسنده: mohsen bahrzadeh
تاریخ: ۱۳۹۰/۱۰/۱۱ ۲۲:۴۶:۲۰

استاد خواهش می کنم این کار رو نکنید، خدایی من SL رو با وحید نصیری شناختم هر موقعه می خوام اسم یه حرفه ای رو نام ببرم اولین اسمی که می یارم استاد وحید نصیری هست، استاد شما خودتون می بینید که من همیشه جزو طرفدارای پروپاقرص وبلاگتون هستم خدایی همیشه به عنوان یه اسطوره واسه ما بودید. پس خواهش می کنم اط این تصمیم منصرف بشید

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۱۰/۱۲ ۰۹:۱۷:۵۱

من امیدی به این جماعت ندارم! همین الان حداقل 2 دو شبکه مخصوص برنامه نویس‌ها در این دور و اطراف هست که ... دارند خاک می‌خورند. یک نمونه idevcenter.com است و نمونه دیگر pspcommunity.org . هر دو توسط تعدادی کمتر از 2 تا 3 نفر سرپا نگه داشته شدن. هر دو هم شاید هفته‌ای یک مطلب یا ماهی 4 تا 5 مطلب جدید داشته باشند. این جماعت خیر و برکت نداره! دست و باز و گشاده‌ای نداره.
اما ... تنها راهی رو که عده‌ای تجربه کردن و جواب داده فعالیت‌های محدود، بسته و غیرعمومی است. همین الان هم هست؛ شاید باور نکنید که لینک‌های referrer در کنترل پنل این وبلاگ به یک سری انجمن و سایت غیرعمومی برنامه نویسی ایرانی داره ختم میشه. حتی انجمن‌های عمومی که نمی‌خوام نام ببرم، قسمت‌های خصوصی دارند؛ و قسمت عمومی آن‌ها جهت بهره کشی از عموم کاربران است. مطالب به درد بخور، در قسمت‌های خصوصی و بسته مطرح می‌شود. آری! اینچنین است، بردار!

نویسنده: Javad Darvish Amiry
تاریخ: ۱۳۹۰/۱۰/۱۲ ۱۲:۲۸:۰۶

خوب نسبت به مسایلی که فرمودید، من هیچ ایده ای ندارم. یعنی راهی به ذهنم نمیرسه جز همونی که گفتم. هرچند منظور من از شبکه، شبکه ای از ارتباطات منسجم و هدفمند بود؛ مثلا یه تیم بلاگ نویس تشکیل بشه، با یه سیستم یه پارچه که توسط سرویس هایی با هم در ارتباط باشن؛ بعنوان مثال بخشی تو بلاگ باشه، شامل خلاصه آخرین پست های بلاگ های دیگه؛ از طرفی با هماهنگی تیم، هر کدوم از نویسنده ها تو بلاگشون فقط راجع به یه موضوع بصورت تخصصی بحث کنن؛ یکی منحصرا WPF یکی دیگه ASP.NET و ... . اینطوری بعد از یه مدت یه شبکه منسجم و غنی بدست میاد؛ از طرفی میشه با پیش بینی راهکارهایی که مهمترینش هم تبلیغات هست، هزینه های فنی رو تامین کرد؛ هرچند که کار سختیه؛ مخصوصا جمع شدن چهار نفر ایرانی کنار همدیگه و کار گروهی!!! بگذریم؛ دقیقا با شما موافقم که اینچنین است...
اگه تصمیم بگیرید که بلاگتون رو خصوصی کنید و هزینه ای هم براش در نظر بگیرید، من با کمال میل اعلان آمادگی میکنم برای ثبت نام. برای شخص بنده، پای درس وحید نصیری نشستن غنیمت ارزشمندی هست که حاضر نیستم از دست بدم. از جزوه های آموزش ASP.NET تا «الگوهای طراحی شیئ گرا» و «امنیت در ASP.NET» تا blogspot و حالا هم dotnettips با جناب نصیری بودم و کماکان خواهم بود؛
آرزوی پیروزی و بهروزی و شادکامی برای شما معلم گرامی و بزرگوار. دنیا به روح بزرگ و سخاوتمندی چون شما نیاز دارد؛ زنده باشید.

نویسنده: VB_ASP_NET
تاریخ: ۱۳۹۰/۱۰/۱۲ ۱۳:۴۹:۱۷

استاد یه خواهشی که ما از شما داریم(میگم ما چون فکر کنم نظر تمام بچه ها همین باشه) اینه که واقعاً این کار رو نکنید چون خدایی شما الان تو اینترنت که بگردید توی هر تالاری یه اسمی از استاد نصیری هست حالا به خاطر چند نفری که واقعاً دارن از مطالب شما استفاده می کنند(من به شخصه دارم یه پروژه بزرگ رو با توجه به مطالب این وبلاگ هدایت می کنم) واقعاً من روزانه 4-3 بار از وبلاگتون میام و مطالبتون رو می خونم واقعاً خیلی خیلی خیلی ناراحت شدم وقتی که این پست رو دیدم اینو واقعاً از ته دل می گم.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۱۰/۱۲ ۱۴:۱۸:۳۱

سلام؛ من نگفتم بسته میشه. من کارم همینه! فقط می‌تونه به اشتراک گذاشته بشه یا نشه. می‌تونه عمومی باشه یا نباشه. ولی زندگی من همینه.
چون باز هم تکرار می‌کنم، «به اشتراک می‌گذارم، به اشتراک می‌گذاری» خوبه.
الان جدا در طول ماه 4 تا 5 نفر فعال هستند که جمعا 5 تا مطلب «شاید» منتشر کنند. این خوب نیست.
فقط تصمیمی که گرفتم این است که برای سال بعد عمومی نباشه. رایگان هم نباشه.

نویسنده: rahmat rezaei
تاریخ: ۱۳۹۰/۱۰/۱۲ ۱۶:۲۵:۲۳

همون اوایل که اشتراکهای روزانه قطع شد می خواستم بپرسم که چرا قطع شد؟ دیدم شاید پررویی باشه نپرسیدم. ولی بزارید یه سوالی که برام پیش اومده بپرسم :
چرا شما و کسانی مثل شما که به علم روز برنامه نویسی و نرم افزار تسلط دارید، پروژه های اوپن سورس تولید نمی کنید که هم آموزش است و هم می تواند به خوبی درآمد زا باشد.
واقعا دلیلش چیست که در ایران این کار اتفاق نمی افتد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۱۰/۱۲ ۱۷:۳۲:۴۱

البته من چند پروژه سورس باز دارم در کدپلکس : (^) و اینکه گفتم این نوع طرز فکر را پشتیبانی می‌کنم، در عمل هم رخ داده.
الان اتفاقا دو پروژه هم هست که مدتی است دارم روی آن‌ها کار می‌کنم: یک گزارش ساز جامع هست بر پایه iTextSharp و یک برنامه نوشته شده با ASP.NET به عنوان معادل دات نتی این رپیدلیج PHP کارها (بهتره بگم PHP باز ... چون این برنامه حتی تعریف یک «صف» هم ندارد) که خیلی خیلی از آن کاملتر است. من روی فروش این‌ها نمی‌تونم حساب باز کنم چون زمانیکه ارائه شد ... یعنی رفته. اما می‌شود روی پشتیبانی غیر رایگان این‌ها حساب کرد. شاید برای سال بعد این کار رو کردم. برای امسال برنامه‌ای ندارم.

نویسنده: shahin kiassat
تاریخ: ۱۳۹۰/۱۰/۱۲ ۲۱:۴۸:۰۳

آقای نصیری به عنوان کسب تجربه سوالی داشتم :
زمانی که در آموزش های iTextSharp خودتون به این مسئله که مشغول تهیه ی گذارش ساز هستید اشاره کردید من هم تصمیم گرفتم کمی با iTextSharp برای چاپ گذارش ها کار کنم.(با توجه به مشکلاتی که با ابزار های آماده داشتم) که انصافا خیلی راضی هستم و از شما متشکرم.
حالا می خوام بدونم آیا به نظر شما درست هست نتیجه ی زحمت رو به صورت سورس باز در اینترنت قرار داد ؟ و گروهی از اون منفعت مالی ببرن و فقط استفاده کننده باشند.

مثلا همین پروژه های کد باز شما چند نفر به غیر از خود شما روی توسعه آن وقت گذاشتن ؟

در شرکت ما (و احتمالا خیلی شرکت ها) بسیار پیش آمده که مدیر پروژه یا برنامه نویس نتیجه ی زحمت صاحب یک وبلاگ رو به اسم خودش تمام می کنه (نمونش jQueyr User control loader شما یا خیلی نمونه های دیگه)
خیلی ممنون از شما.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۱۰/۱۲ ۲۲:۵۷:۲۴

خوب، سخت بودن کار سورس باز هم همینجا است. در کارهای سورس باز در تمام آن‌ها، شما مجاز هستید کار مشتق شده رو بفروشید. تنها تفاوت در اینجا است که یکی می‌گه حتما باید سورس رو هم کنارش قرار بدی و یکی می‌گه مهم نیست و گرنه تمامشون با این مساله مالی مشکلی ندارند.
در کل به درجه‌ی روانی به اشتراک گذاری اطلاعات رسیدن، کار سختی است. به همین جهت باز هم تکرار می‌کنم این دور و اطراف بگردید، ماهی 5 نفر رو شاید پیدا کنید که حاضر باشند مطلب فنی مهمی رو به رایگان منتشر کنند و قید همه چیز آن‌را بزنند. زمانیکه هم که روز به روز تعدادشون کمتر بشه، انگیزه رو از بقیه خواهند گرفت.

نویسنده: مهدی موسوی
تاریخ: ۱۳۹۰/۱۰/۱۳ ۱۵:۲۰:۳۲

سلام.
ما در تولید محتوای فنی (در زمینه کاری خودمون و به زبان فارسی) به شکلی صحیح، مولفین انگشت شماری داریم. به نظر من برخی از مهمترین دلائل این مساله عبارت است از:

1. برخی از افراد انگل هستن - تمام! (به کتاب Harley Hahn در این زمینه مراجعه کنید).

2. بعضی ها فکر میکنن که اگر فلان مطلب رو به اشتراک بذارم، ممکنه همکارم، دوستم و ... با خوندن اون مطلب، فاصله دانسته هاشو با من کم کنه، و به این ترتیب منو با زحمت مواجه کنه (متاسفانه این خصیصه در بسیاری از شرکت های خصوصی، و تقریبا تمامی سازمان های دولتی دیده میشه).

3. وقتی گوگل و بسیاری از شرکت های شناخته شده در صنعت آگهی های Online بر اساس ضوابط کاری کشورشون، اجازه سرویس دهی به سایت هایی با مطالب فارسی رو ندارن، من نوعی از چه طریقی می تونم از نوشتن، انتفاع حاصل کنم؟ چه کنم که به شرکتهای ایرانی فعال در زمینه Ads نیز اعتمادی ندارم؟ جدا از اینکه تبلیغات این شرکت ها، اکثرا Animation هستش و من دوست دارم خواننده مطالب من، هنگام مطالعه یه مطلب فنی، احساس آرامش کنه، نه اینکه چشم هاش مدام به خاطر وجود یه آگهی َ«ـِـً]ُ\[]!,@# پر پر بزنه.

4. نبود قانون Copyright از دیگر دلائل عمده ای هستش که باعث شده در این زمینه ما پیشرفتی نکنیم. وقتی یکی از مطالب فارسی ای که نوشته بودم (در سال 1996 (یا 1994)، دقیق خاطرم نیست)، کپی و در یکی از جرائد کشور به اسم فرد دیگه ای منتشر شد، در همون ابتدای راه تصمیم گرفتم دیگه فارسی ننویسم. متاسفانه هنوز که هنوزه، کم و بیش شاهد این اتفاقات هستیم.

5. ...

اما در مورد مطلبی که در مورد بلاگ من فرمودید. حقیقتش بعد از اینکه فردی چند سال پیش، منو به دلیل مطالبی که به اشتراک میذاشتم به سخره گرفت و ازم پرسید که "تو اصلا میدونی معمار کیه؟"، به خودم اومدم و تصمیم گرفتم مثل وبلاگ های دیگه، به مطالب بزن و برو اکتفا نکنم. به همین دلیل، از 2010/1/2 به بعد، مطالب ارسالیم شکل مقاله به خودشون گرفتن که طبیعتا، نوشتنشون در مقایسه با نوشتن یکی دو وجب مطلب فنی، بسیار دشوارتر و زمان بر تر هست. من از March 2008 تا February 20110 در واقع برای Search Engine گوگل می نوشتم، نه برای خوانندگان. اما از اون تاریخ به بعد، مطالبی که در وبلاگم گذاشتم، توجه بسیاری از افراد رو بخودش جلب کرد و ... (بی ارتباط با موضوع گفتگو هستش، بنابراین بیش از این در این مورد توضیح نمیدم).

در هر حال، بنظر من، عدم ارائه مطالب فنی یه وبلاگ بصورت رایگان (در فرهنگ ما)، موفقیتی در پی نداره (امیدوارم برای آقای نصیری اینطور نباشه، البته اگر این وبلاگ رو از حالت رایگان در آوردن). من هنوز یادم نرفته افرادیکه برای شرکت در کنفرانس کذایی ای که بهروز راد، من و یکی دو نفر دیگه قرار بود در مورد JavaScript Performance، HTML5 و ... مطلب ارائه بدیم، ابراز خرسندی کردن، اما وقت پول دادن که شد، تعداد افراد ثبت نام کننده به حداقل تعداد مورد نیاز نرسید و اون جلسه Cancel شد. جای تاسفه اگر بدونید برای یه جلسه 1 ساعته، هر نفر فقط باید 5-6 هزارتومان پرداخت می کرد...

نویسنده: امیرحسین جلوداری
تاریخ: ۱۳۹۰/۱۰/۱۵ ۲۲:۳۳:۴۸

سلام استاد ... خیلی نوکریم ... :دی
والا من نمیدونم شما رو چه دلیلی میفرمایین که خلاصه ی اشتراکات مورد استقبال قرار نگرفت!!! ... ولی من میدونم که این کار شما رو نمونشو تو هیچ بلاگ ایرانی به شخصه ندیده بودم و واقعا (واقعا!) کار قشنگی بود و واقعا(واقعا!!) مفید واقع شد و من و خیلی از دوستان صمیمانه انتظار داریم که به این کار ادامه بدین!

من با این کارتون با بسیاری از بلاگ های مفید ایرانی و اونور آبی :دی آشنا شدم و خیلی هاشو الان دارم follow میکنم ...

در ضمن شک نداریم که وبلاگ شما بهترین وبلاگ فارسی موجود در برنامه نویسی حرفه ای و نیمه حرفه ای هست ... (خودتونم شک نکنید :دی)

آره! قبول دارم که ما وبلاگ خونایه خوبی نیستیم!!! ... زورمون میاد یه لایک معمولی کنیم یا کامنت تشکر بذاریم! ... ولی این دلیل نمیشه که از کار شما استقبال نمیشه ... من وقتی تو گودرم تایتلایه شما رو وقتی نوشته بود خلاصه اشتراکایه ... میدیدم بی درنگ expand میکردم لینک رو!

اصن شما یه پست بزنین در مورد این قضیه و کارایی که قصد دارین در آینده انجام بدینو مطرح کنین ... مطمئنم اونوقته که تازه متوجه میشین که چقدر طرفدار دارین!!! ... که خیلی ها شما رو اسطوره ی خودشون میدونن ... که خیلی ها شما رو دعا میکنن ... که خوش به حالتون تو اون دنیا :دی ... که ...

موفق باشید ...
یا علی ...

نویسنده: بهروز راد
تاریخ: ۱۳۹۰/۱۰/۱۶ ۱۷:۱۲:۲۴

@وحید نصیری

از تمام این Commentها فقط یک جمله رو پسندیدم... "خوش به حالت توی اون دنیا" :)

عنوان: **MVVM و نمایش دیالوگ‌ها**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۱۰/۱۲ ۱۰:۱۱:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: MVVM

بسیاری از برنامه‌های دسکتاپ نیاز به نمایش پنجره‌های دیالوگ استاندارد ویندوز مانند OpenFileDialog و SaveFileDialog را دارند و سؤال اینجا است که چگونه اینگونه موارد را باید از طریق پیاده سازی صحیح الگوی MVVM مدیریت کرد؛ از آنجائیکه خیلی راحت در فایل ViewModel می‌توان نوشت new OpenFileDialog و الی آخر. این مورد هم یکی از دلایل اصلی استفاده از الگوی MVVM را زیر سؤال می‌برد : این ViewModel دیگر قابل تست نخواهد بود. همیشه شرایط آزمون‌های واحد را به این صورت در نظر بگیرید:
سروری وجود دارد در جایی که به آن دسترسی نداریم. روی این سرور با اتوماسیونی که راه انداخته‌ایم، آخر هر روز آزمون‌های واحد موجود به صورت خودکار انجام شده و یک گزارش تهیه می‌شود (مثلا یک نوع continuous integration سرور). بنابراین کسی دسترسی به سرور نخواهد داشت تا این OpenFileDialog ظاهر شده را مدیریت کرده، فایلی را انتخاب و به برنامه آزمون واحد معرفی کند. به صورت خلاصه ظاهر شدن هر نوع دیالوگی حین انجام آزمون‌های واحد «مسخره» است!
یکی از روش‌های حل این نوع مسایل، استفاده از dependency injection یا تزریق وابستگی‌ها است و در ادامه خواهیم دید که چگونه WPF بدون نیاز به هیچ نوع فریم ورک تزریق وابستگی خارجی، از این مفهوم پشتیبانی می‌کند.

مروری مقدماتی بر تزریق وابستگی‌ها

امکان نوشتن آزمون واحد برای new OpenFileDialog وجود ندارد؟ اشکالی نداره، یک Interface بر اساس نیاز نهایی برنامه درست کنید (نیاز نهایی برنامه از این ماجرا فقط یک رشته LoadPath است و بس) سپس در ViewModel با این اینترفیس کار کنید؛ چون به این ترتیب امکان « تقلید » آن فراهم می‌شود.

یک مثال عملی:

ViewModel نیاز دارد تا مسیر فایلی را از کاربر بپرسد. این مساله را با کمک dependency injection در ادامه حل خواهیم کرد.
ابتدا سورس کامل این مثال:

ViewModel برنامه (تعریف شده در پوشه ViewModels برنامه):

```
namespace WpfFileDialogMvvm.ViewModels
{
    public interface IFilePathContract
    {
        string GetFilePath();
    }

    public class MainWindowViewModel
    {
        IFilePathContract _filePathContract;
        public MainWindowViewModel(IFilePathContract filePathContract)
        {
            _filePathContract = filePathContract;
        }

        //...

        private void load()
        {
            string loadFilePath = _filePathContract.GetFilePath();
            if (!string.IsNullOrWhiteSpace(loadFilePath))
            {
                // Do something
            }
        }
    }
}
```

دو نمونه از پیاده سازی اینترفیس IFilePathContract تعریف شده (در پوشه Dialogs برنامه):

```
using Microsoft.Win32;
using WpfFileDialogMvvm.ViewModels;

namespace WpfFileDialogMvvm.Dialogs
{
    public class OpenFileDialogProvider : IFilePathContract
    {
        public string GetFilePath()
        {
            var ofd = new OpenFileDialog
            {
                Filter = "XML files (*.xml)|*.xml"
            };
            string filePath = null;
            bool? dialogResult = ofd.ShowDialog();
            if (dialogResult.HasValue && dialogResult.Value)
            {
                filePath = ofd.FileName;
            }
            return filePath;
        }
    }

    public class FakeOpenFileDialogProvider : IFilePathContract
    {
        public string GetFilePath()
        {
            return @"c:\path\data.xml";
        }
    }
}
```

و View برنامه:

```
<Window x:Class="WpfFileDialogMvvm.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:vm="clr-namespace:WpfFileDialogMvvm.ViewModels"
        xmlns:dialogs="clr-namespace:WpfFileDialogMvvm.Dialogs"
        Title="MainWindow" Height="350" Width="525">
    <Window.Resources>
        <ObjectDataProvider x:Key="mainWindowViewModel"
                            ObjectType="{x:Type vm:MainWindowViewModel}">
            <ObjectDataProvider.ConstructorParameters>
                <dialogs:OpenFileDialogProvider/>
            </ObjectDataProvider.ConstructorParameters>
        </ObjectDataProvider>
    </Window.Resources>
    <Grid DataContext="{Binding Source={StaticResource mainWindowViewModel}}">

    </Grid>
</Window>
```

توضیحات:

ما در ViewModel نیاز داریم تا مسیر نهایی فایل را دریافت کنیم و این عملیات نیاز به فراخوانی متد ShowDialog ایی را دارد که امکان نوشتن آزمون واحد خودکار را از ViewModel ما سلب خواهد کرد. بنابراین بر اساس نیاز برنامه یک اینترفیس عمومی به نام IFilePathContract را طراحی می‌کنیم. در حالت کلی کلاسی که این اینترفیس را پیاده سازی می‌کند، قرار است مسیری را برگرداند. اما به کمک استفاده از اینترفیس، به صورت ضمنی اعلام می‌کنیم که «برای ما مهم نیست که چگونه». می‌خواهد OpenFileDialogProvider ذکر شده باشد، یا نمونه تقلیدی مانند FakeOpenFileDialogProvider. از نمونه واقعی

OpenFileDialogProvider در برنامه اصلی استفاده خواهیم کرد، از نمونه تقلیدی FakeOpenFileDialogProvider در آزمون واحد و نکته مهم هم اینجا است که ViewModel ما چون بر اساس اینترفیس IFilePathContract پیاده سازی شده، با هر دو DialogProvider یاد شده می‌تواند کار کند.

مرحله آخر نوبت به وهله سازی نمونه واقعی، در View برنامه است. یا می‌توان در Code behind مرتبط با View نوشت:

```
namespace WpfFileDialogMvvm
{
    public partial class MainWindow
    {
        public MainWindow()
        {
            InitializeComponent();
            this.DataContext = new MainWindowViewModel(new OpenFileDialogProvider());
        }
    }
}
```

و یا از روش ObjectDataProvider توکار WPF هم می‌شود استفاده کرد؛ که مثال آن‌را در کدهای XAML مرتبط با View ذکر شده می‌توانید مشاهده کنید. ابتدا دو فضای نام vm و dialog تعریف شده (با توجه به اینکه مثلا در این مثال، دو پوشه ViewModels و Dialogs وجود دارند). سپس کار تزریق وابستگی‌ها به سازنده کلاس MainWindowViewModel، از طریق ObjectDataProvider.ConstructorParameters انجام می‌شود:

```
<ObjectDataProvider x:Key="mainWindowViewModel"
                            ObjectType="{x:Type vm:MainWindowViewModel}">
            <ObjectDataProvider.ConstructorParameters>
                <dialogs:OpenFileDialogProvider/>
            </ObjectDataProvider.ConstructorParameters>
</ObjectDataProvider>
```

نظرات خوانندگان

نویسنده: Salar
تاریخ: ۱۳۹۰/۱۰/۱۲ ۱۹:۳۸:۵۰

خواستم تشکر کنم از شما که اندک ته مانده وب فارسی رو غنی می کنید. از سری MVVM هم تشکر می کنم.
نظرتون در مورد کتابچه 54 صفحه ای Advanced MVVM چی هست؟ می خواستم در اولین فرصت مطالعش کنم.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۱۰/۱۲ ۱۹:۵۲:۰۱

سلام؛ ممنون. شما لطف دارید.
این کتابچه در حقیقت شرح نوشتن یک برنامه بازی به کمک الگوی MVVM است. بد نیست ولی انتظار نداشته باشید که به صورت قدم به قدم چیزی را توضیح داده باشد. فقط شرح برنامه است.

نویسنده: مومن
تاریخ: ۱۳۹۰/۱۰/۱۴ ۰۰:۵۰:۴۷

سلام
از مطالب خوب مفیدتون تشکر می کنم.
سوال: راهی برای مدیریت دسترسی کاربران به Command ها وجود داره.
به این صورت که در فرم تعریف دسترسی ها لیست فرامین نمایش داده شود و سپس آنها را برای کاربران یا گروههای کاربری فعال و غیر فعال کنیم.
خلاصه کلام: راهی برای نمایش لیست کردن فرامین یک ویومدل وجود داره یانه؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۱۰/۱۴ ۰۱:۰۲:۰۹

- شما تمام خواص عمومی (و حتی غیرعمومی) رو می‌تونید به کمک Reflection لیست کنید. نوع آن‌ها هم که مشخص است از جنس مثلا ICommand یا DelegateCommand هستند. بنابراین یافتن و لیست کردن خودکار Commands تعریف شده در یک ViewModel به این ترتیب امکان پذیر است.
- در حالت کلی برای مدیریت Commands فقط کافی است قسمت canExecute آن‌ها را مدیریت کنید (مثلا در delegate command معرفی شده در همین سری). اگر false برگرداند (مثلا بر اساس سطح دسترسی کاربر جاری)، خودبخود عنصر مرتبط با آن غیرفعال خواهد شد.

نویسنده: مومن
تاریخ: ۱۳۹۰/۱۰/۱۴ ۰۷:۴۶:۵۱

سپاس

نویسنده: مهرناز توکلی
تاریخ: ۱۳۹۱/۰۷/۱۹ ۱۶:۶

خیلی خوب بود .ممنون

عنوان: **بارگذاری UserControl در WPF به کمک الگوی MVVM**
نویسنده: شاهین کیاست
تاریخ: ۱۳۹۱/۱۰/۲۷ ۳:۳۵
آدرس: www.dotnettips.info
برچسب‌ها: MVVM, WPF

در نرم افزارهای تحت ویندوز روشها و سلیقه‌های متفاوتی برای چینش فرمها ، منوها و دیگر اجزای برنامه وجود دارد. در یک نرم افزار اتوماسیون اداری که فرمهای ورود اطلاعات زیادی دارد فضای کافی برای نمایش همه‌ی فرم‌ها به کاربر نیست. یکی از روش هایی که می‌تواند به کار رود تقسیم قسمت‌های مختلف نرم افزار در Viewهای جداگانه است. این کار استفاده‌ی مجدد از قسمت‌های مختلف و نگهداری کد را سهولت می‌بخشد.

الگوی متداولی که در نرم افزارهای WPF و Silverlight استفاده می‌شود الگوی MVVM است. (این الگو در جاوااسکریپت هم به سبب Statefull بودن استفاده می‌شود.) قبلا مطالب زیادی در این سایت جهت آموزش و توضیح این الگوی منتشر شده است.
فرض کنید نرم افزار از چند بخش تشکیل شده :

صفحه‌ی اصلی
منو
یک صفحه‌ی خوش آمدگویی
صفحه‌ی ورود و نمایش اطلاعات

می توان اجزا و تعریف هر یک از این قسمت‌ها را در یک UserControl قرار داد و در زمان مناسب آن را بارگذاری کرد.
سوالی که مطرح است بارگذاری UserControlها به کمک الگوی MVVM چگونه است ؟
کدهای XAML صفحه‌ی اصلی :

```
<Window x:Class="TwoViews.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        Title="MVVM Light View Switching"
        d:DesignHeight="300"
        d:DesignWidth="300"
        DataContext="{Binding Main,
                              Source={StaticResource Locator}}"
        ResizeMode="NoResize"
        SizeToContent="WidthAndHeight"
        mc:Ignorable="d">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
        </Grid.RowDefinitions>

        <ContentControl Content="{Binding CurrentViewModel}" />

        <DockPanel Grid.Row="1" Margin="5">
            <Button Width="75"
                    Height="23"
                    Command="{Binding SecondViewCommand}"
                    Content="Second View"
                    DockPanel.Dock="Right" />
            <Button Width="75"
                    Height="23"
                    Command="{Binding FirstViewCommand}"
                    Content="First View"
                    DockPanel.Dock="Left" />
        </DockPanel>
    </Grid>
</Window>
```

2 دکمه در صفحه‌ی اصلی وجود دارد ، یکی از آنها وظیفه‌ی بارگذاری View اول و دیگری وظیفه‌ی بارگذاری View دوم را دارد ، این دکمه‌ها نقش منو را در یک نرم افزار واقعی به عهده دارند.

کدهای View-Model گره خورده (به کمک الگوی ViewModolLocator) به View اصلی :

```
/// This is our MainViewModel that is tied to the MainWindow via the
    /// ViewModelLocator class.
    /// </summary>
    public class MainViewModel : ViewModelBase
    {
        /// <summary>
        /// Static instance of one of the ViewModels.
        /// </summary>
        private static readonly SecondViewModel SecondViewModel = new SecondViewModel();
        /// <summary>
        /// Static instance of one of the ViewModels.
        /// </summary>
        private static readonly FirstViewModel FirstViewModel = new FirstViewModel();
        /// <summary>
        /// The current view.
        /// </summary>
        private ViewModelBase _currentViewModel;
        /// <summary>
        /// Default constructor.  We set the initial view-model to 'FirstViewModel'.
        /// We also associate the commands with their execution actions.
        /// </summary>
        public MainViewModel()
        {
            CurrentViewModel = FirstViewModel;
            FirstViewCommand = new RelayCommand(ExecuteFirstViewCommand);
            SecondViewCommand = new RelayCommand(ExecuteSecondViewCommand);
        }
        /// <summary>
        /// The CurrentView property.  The setter is private since only this
        /// class can change the view via a command.  If the View is changed,
        /// we need to raise a property changed event (via INPC).
        /// </summary>
        public ViewModelBase CurrentViewModel
        {
            get { return _currentViewModel; }
            set
            {
                if (_currentViewModel == value)
                    return;
                _currentViewModel = value;
                RaisePropertyChanged("CurrentViewModel");
            }
        }
        /// <summary>
        /// Simple property to hold the 'FirstViewCommand' - when executed
        /// it will change the current view to the 'FirstView'
        /// </summary>
        public ICommand FirstViewCommand { get; private set; }
        /// <summary>
        /// Simple property to hold the 'SecondViewCommand' - when executed
        /// it will change the current view to the 'SecondView'
        /// </summary>
        public ICommand SecondViewCommand { get; private set; }
        /// <summary>
        /// Set the CurrentViewModel to 'FirstViewModel'
        /// </summary>
        private void ExecuteFirstViewCommand()
        {
            CurrentViewModel = FirstViewModel;
        }
        /// <summary>
        /// Set the CurrentViewModel to 'SecondViewModel'
        /// </summary>
        private void ExecuteSecondViewCommand()
        {
            CurrentViewModel = SecondViewModel;
        }
    }
```

این ViewModel از کلاس پایه‌ی چارچوب MVVM Light مشتق شده است. Commandها جهت Handle کردن کلیک دکمه‌ها هستند .
نکته‌ی اصلی این ViewModel پراپرتی CurrentViewModel می‌باشد. این پراپرتی به ویژگی Content کنترل ContentControl مقید (Bind) شده است. با کلیک شدن روی دکمه‌ها View مورد نظر به کاربر نمایش داده می‌شود.
WPF از کجا می‌داند کدام View را به ازای ViewModel خاص render کند ؟
در فایل App.xaml یک سری DataTemplate تعریف شده است :

```
<Application.Resources>
        <vm:ViewModelLocator x:Key="Locator" d:IsDataSource="True" />
        <!--
            We define the data templates here so we can apply them across the
            entire application.

            The data template just says that if our data type is of a particular
            view-model type, then render the appropriate view.  The framework
            takes care of this dynamically.  Note that the DataContext for
            the underlying view is already set at this point, so the
            view (UserControl), doesn't need to have it's DataContext set
            directly.
        -->
        <DataTemplate DataType="{x:Type vm:SecondViewModel}">
            <views:SecondView />
        </DataTemplate>
        <DataTemplate DataType="{x:Type vm:FirstViewModel}">
            <views:FirstView />
        </DataTemplate>
    </Application.Resources>
```

به کمک این DataTemplateها مشخص شده اگر نوع داده‌ی ما از یک نوع View-Model خاص می‌باشد View مناسب را به ازای آن Render کند. با تعریف DataTemplateها در App.Xaml می‌توان از آنها در سطح نرم افزار استفاده کرد. می‌توان DataTemplateها را جهت خلوت کردن App.xaml به Resource دیگری انتقال داد.

دریافت مثال : TwoViews.zip
منبع مثال

نظرات خوانندگان

نویسنده: افشار محبی
تاریخ: ۱۳۹۱/۱۰/۲۷ ۱۹:۲۰

کاربرد DataTemplate را نمی‌دانستم. اینجا یاد گرفتم. ممنون.

نویسنده: سعید
تاریخ: ۱۳۹۱/۱۰/۲۸ ۱۴:۲۳

علت خاصی داره که viewmodelهای دوم و اول رو به صورت static readonly تعریف کردید؟ اگر تعداد viewmodelها زیاد شد برنامه به مشکلات مصرف زیاد حافظه بر نمی‌خوره؟ شاید استفاده از خواص lazy در اینجا مناسب‌تر باشه. یا اینکه این وهله سازی فقط در زمان نیاز انجام بشه.

عنوان: **آزمون واحد در MVVM به کمک تزریق وابستگی**
نویسنده: شاهین کیاست
تاریخ: ۱۳۹۲/۰۱/۰۴ ۱۷:۰
آدرس: www.dotnettips.info
برچسب‌ها: MVVM, Unit testing, Dependency Injection

یکی از خوبی‌های استفاده از Presentation Patternها بالا بردن تست پذیری برنامه و در نتیجه نگهداری کد می‌باشد.
MVVM الگوی محبوب برنامه نویسان WPF و Silverlight می‌باشد. به صرف استفاده از الگوی MVVM نمی‌توان اطمینان داشت که ViewModel کاملا تست پذیری داریم. به عنوان مثلا اگر در ViewModel خود مستقیما DialogBox کنیم یا ارجاعی از View دیگری داشته باشیم نوشتن آزمون‌های واحد تقریبا غیر ممکن می‌شود. قبلا درباره‌ی این مشکلات و راه حل آن مطلب در سایت منتشر شده است :

- MVVM و نمایش دیالوگ‌ها

در این مطلب قصد داریم سناریویی را بررسی کنیم که ViewModel از Background Worker جهت انجام عملیات مانند دریافت داده‌ها استفاده می‌کند.

Background Worker کمک می‌کند تا اعمال طولانی در یک Thread دیگر اجرا شود در نتیجه رابط کاربری Freeze نمی‌شود.

به این مثال ساده توجه کنید :

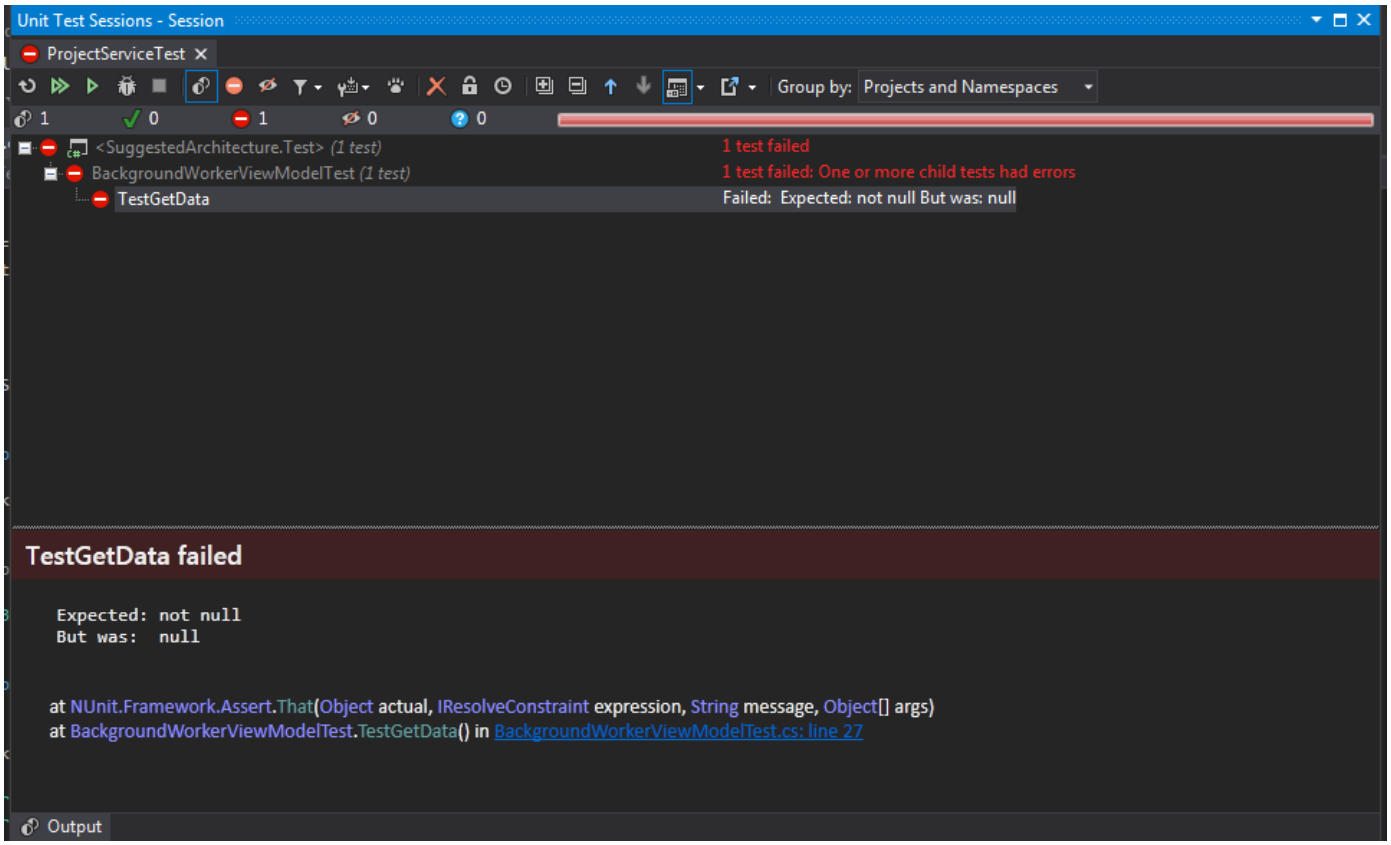
```
public class BackgroundWorkerViewModel : BaseViewModel
    {
        private List<string> _myData;

        public BackgroundWorkerViewModel()
        {
            LoadDataCommand = new RelayCommand(OnLoadData);
        }

        public RelayCommand LoadDataCommand { get; set; }

        public List<string> MyData
        {
            get { return _myData; }
            set
            {
                _myData = value;
                RaisePropertyChanged(() => MyData);
            }
        }

        public bool IsBusy { get; set; }

        private void OnLoadData()
        {
            var backgroundWorker = new BackgroundWorker();
            backgroundWorker.DoWork += (sender, e) =>
                                {
                                    MyData = new List<string> {"Test"};
                                    Thread.Sleep(1000);
                                };
            backgroundWorker.RunWorkerCompleted += (sender, e) => { IsBusy = false; };
            backgroundWorker.RunWorkerAsync();
        }
    }
```

در این ViewModel با اجرای دستور LoadDataCommand داده‌ها از یک منبع داده دریافت می‌شود. این عمل می‌تواند چند ثانیه طول بکشد ، در نتیجه برای قفل نشدن رابط کاربر این عمل را به کمک Background Worker به صورت Async در پشت صحنه انجام شده است.
آزمون واحد این ViewModel اینگونه خواهد بود :

```
[TestFixture]
```

```
public class BackgroundWorkerViewModelTest
{
    #region Setup/Teardown

    [SetUp]
    public void SetUp()
    {
        _backgroundWorkerViewModel = new BackgroundWorkerViewModel();
    }

    #endregion

    private BackgroundWorkerViewModel _backgroundWorkerViewModel;

    [Test]
    public void TestGetData()
    {

        _backgroundWorkerViewModel.LoadDataCommand.Execute(_backgroundWorkerViewModel);

        Assert.NotNull(_backgroundWorkerViewModel.MyData);
        Assert.IsNotEmpty(_backgroundWorkerViewModel.MyData);
    }
}
```

با اجرای این آزمون واحد نتیجه با آن چیزی که در زمان اجرا رخ می‌دهد متفاوت است و با وجود صحیح بودن کدها آزمون واحد شکست می‌خورد.
چون Unit Test به صورت همزمان اجرا می‌شود و برای عملیات‌های پشت صحنه صبر نمی‌کند در نتیجه این آزمون واحد شکست می‌خورد.



یک راه حل تزریق BackgroundWorker به صورت وابستگی به ViewModel می‌باشد. همانطور که قبلا اشاره شده یکی از مزایای استفاده از تکنیک‌های تزریق وابستگی سهولت Unit testing می‌باشد.

در نتیجه یک Interface عمومی و 2 پیاده سازی همزمان و غیر همزمان جهت استفاده در برنامه‌ی واقعی و آزمون واحد تهیه می‌کنیم :

```
public interface IWorker
    {
        void Run(DoWorkEventHandler doWork);
        void Run(DoWorkEventHandler doWork, RunWorkerCompletedEventHandler onComplete);
    }
```

جهت استفاده در برنامه‌ی واقعی :

```
public class AsyncWorker : IWorker
    {
        public void Run(DoWorkEventHandler doWork)
        {
            Run(doWork, null);
        }

        public void Run(DoWorkEventHandler doWork, RunWorkerCompletedEventHandler onComplete)
        {
            var backgroundWorker = new BackgroundWorker();
            backgroundWorker.DoWork += doWork;
            if (onComplete != null)
                backgroundWorker.RunWorkerCompleted += onComplete;
            backgroundWorker.RunWorkerAsync();


        }
    }
```

جهت اجرا در آزمون واحد :

```
public class SyncWorker : IWorker
    {
        #region IWorker Members

        public void Run(DoWorkEventHandler doWork)
        {
            Run(doWork, null);
        }

        public void Run(DoWorkEventHandler doWork, RunWorkerCompletedEventHandler onComplete)
        {
            Exception error = null;
            var doWorkEventArgs = new DoWorkEventArgs(null);
            try
            {
                doWork(this, doWorkEventArgs);
            }
            catch (Exception ex)
            {
                error = ex;
                throw;
            }
            finally
            {
                onComplete(this, new RunWorkerCompletedEventArgs(doWorkEventArgs.Result, error,
doWorkEventArgs.Cancel));
            }
        }

        #endregion
    }
```

در نتیجه ViewModel اینگونه تغییر خواهد کرد :

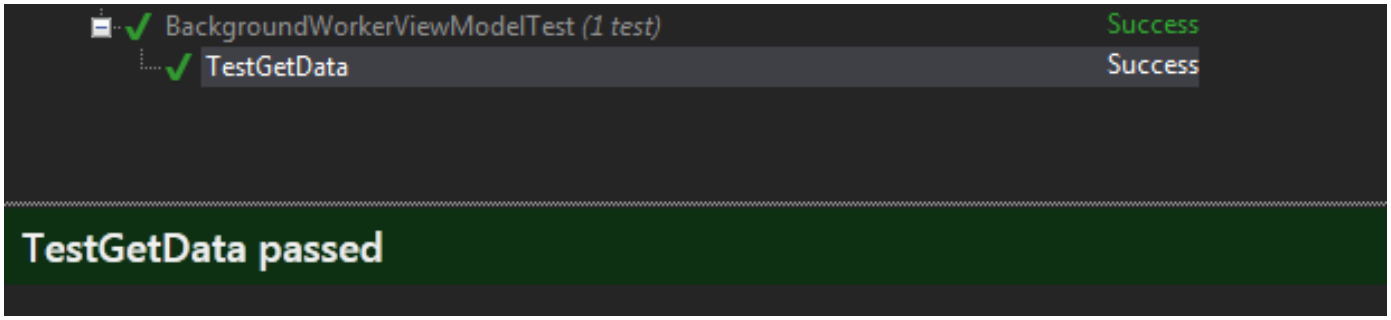
```
public class BackgroundWorkerViewModel : BaseViewModel
    {
        private readonly IWorker _worker;
        private List<string> _myData;
```

```
        public BackgroundWorkerViewModel(IWorker worker)
        {
            _worker = worker;
            LoadDataCommand = new RelayCommand(OnLoadData);
        }

        public RelayCommand LoadDataCommand { get; set; }

        public List<string> MyData
        {
            get { return _myData; }
            set
            {
                _myData = value;
                RaisePropertyChanged(() => MyData);
            }
        }

        public bool IsBusy { get; set; }

        private void OnLoadData()
        {
            IsBusy = true; // view is bound to IsBusy to show 'loading' message.

            _worker.Run(
                (sender, e) =>
                    {
                        MyData = new List<string> {"Test"};
                        Thread.Sleep(1000);
                    },
                (sender, e) => { IsBusy = false; });
        }
    }
```

کلاس مربوطه به آزمون واحد را مطابق با تغییرات ViewModel :

```
[TestFixture]
    public class BackgroundWorkerViewModelTest
    {
        #region Setup/Teardown

        [SetUp]
        public void SetUp()
        {
            _backgroundWorkerViewModel = new BackgroundWorkerViewModel(new SyncWorker());
        }

        #endregion

        private BackgroundWorkerViewModel _backgroundWorkerViewModel;

        [Test]
        public void TestGetData()
        {

            _backgroundWorkerViewModel.LoadDataCommand.Execute(_backgroundWorkerViewModel);

            Assert.NotNull(_backgroundWorkerViewModel.MyData);
            Assert.IsNotEmpty(_backgroundWorkerViewModel.MyData);
        }
    }
```

اکنون اگر Unit Test را اجرا کنیم نتیجه اینگونه خواهد بود :


BackgroundWorkerViewModelTest (1 test)
Success
TestGetData
Success
TestGetData passed

عنوان: **مقیدسازی (DataBinding) در WPF زمانی که دسترسی به DataContext وجود ندارد**
نویسنده: محبوبه محمدی
تاریخ: ۱۳۹۲/۰۳/۰۲ ۱۴:۵
آدرس: www.dotnettips.info
برچسب‌ها: MVVM, WPF

در WPF و Silverlight می‌توان با استفاده از مقید سازی (DataBinding) کنترل‌ها را به منبع‌های داده متصل کرد. این منابع به چند شیوه مختلف مانند استفاده مستقیم از خصوصیت Source قابل دسترسی هستند. یکی از این روش ها، ارث بری از DataContext نزدیک‌ترین والد است.

همانطور که گفته شد DataContext هر کنترل، توسط تمامی فرزندان آن قابل دسترسی است. اما در بعضی مواقع، زمانیکه کنترل فرزند، بخشی از visual یا logical tree نباشند، دسترسی به DataContext وجود ندارد.

برای مثال زمانی که نیاز است خصوصیت ItemsSource مربوط به یک به لیستی خارج از ItemsSource کنترل DataGrid DataGridTemplateColumn مثلا به لیستی درون ViewModel مربوط به Window در مثال زیر مقید شود، به صورت معمول باید به این صورت عمل کرد:

ViewModel :

```
public List<People> ComboBoxDataSource{get; set;}
```

XAML :

```
<Window x:Class="WpfApplication1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow"
        x:Name="this">
    <Grid>
        <DataGrid ItemsSource="{Binding DataCollection}">
            <DataGrid.Columns>
                <DataGridComboBoxColumn ItemsSource="{Binding DataContext.ComboBoxDataSource,
ElementName=this}"/>
            </DataGrid.Columns>
        </DataGrid>
    </Grid>
</Window>
```

با اینکه همه چیز درست به نظر می‌رسد اما در عمل هیچ اتصالی صورت نمی‌گیرد و در پنجره Output ویژوال استادیو خطای زیر مشاهده می‌شود:

```
System.Windows.Data Error: 2 : Cannot find governing FrameworkElement or FrameworkContentElement for
target element.
BindingExpression:Path=ComboBoxDataSource; DataItem=null;
target element is 'DataGridComboBoxColumn' (HashCode=17334644); target property is 'ItemsSource' (type
'IEnumerable')
```

این خطا مشخص میکند که WPF نمیتواند تشخیص بدهد که کدام FrameWorkElement قرار است از DataContext استفاده کند؛ چرا که همانطور که قبلا عنوان شد DataGridTemplateColumn بخشی از visual یا logical tree نیست.

برای مشکل فوق در صورتیکه خصوصیت مورد نظر، یک خصوصیت از فرزندان کنترل باشد، از طریق استایل‌ها می‌توان مشکل را حل کرد. برای مثال به جای ItemSource مربوط به DataGridComboBoxColumn می‌توان خصوصیت ItemSource کنترل ComboBox درون آن را تنظیم کرد.

```
<DataGridComboBoxColumn DisplayMemberPath="FirstName">
        <DataGridComboBoxColumn.EditingElementStyle>
              <Style TargetType="ComboBox">
                    <Setter Property="ItemsSource" Value="{Binding DataContext.ComboBoxDataSource ,
ElementName=this}"/>
               </Style>
         </DataGridComboBoxColumn.EditingElementStyle>
   </DataGridComboBoxColumn>
```

اما در صورتیکه نیاز باشد یک خصوصیت از خود DataGridComboBoxColumn مانند Visibility مقید سازی شود، روش بالا کارساز نخواهد بود. برای حل مشکل فوق میتوان از کلاس‌های Freezable استفاده کرد؛ چرا که این کلاسها می‌توانند از DataContext ارث بری کنند حتی زمانیکه بخشی از visual یا logical tree نباشند. برای این کار می‌توان کلاس زیر را ایجاد کرد:

```
public class DataBindingHelper : Freezable
    {
        protected override Freezable CreateInstanceCore()
        {
            return new DataBindingHelper();
        }
        public object Data
        {
            get { return (object)GetValue(DataProperty); }
            set { SetValue(DataProperty, value); }
        }

        public static readonly DependencyProperty DataProperty =
            DependencyProperty.Register("Data", typeof(object), typeof(DataBindingHelper), new
UIPropertyMetadata(null));
    }
```

و یک نمونه از آن را در Resourceهای DataGrid ساخت:

```
<DataGrid.Resources>
       <local:DataBindingHelper x:Key="bindingHelper"Data="{Binding}"/>
</DataGrid.Resources>
```

و هنگام مقید سازی خصوصیت Visibility مربوط به DataGridComboBoxColumn، از نمونه ساخته شده به عنوان Source استفاده نمود.

```
<DataGridComboBoxColumn Visibility="{Binding Data.IsVisible,Converter={StaticResource
visibilityConverter},Source={StaticResource bindingHelper}}"/>
```

عنوان: **استفاده از SimpleIoc به عنوان یک IoC Container**
نویسنده: مهدی ع
تاریخ: ۱۳۹۲/۰۳/۱۱ ۱۶:۲۰
آدرس: www.dotnettips.info
برچسب‌ها: Design patterns, MVVM, Dependency Injection, SimpleIoc, Ioc, mvvm light

SimpleIoc به صورت پیش فرض در پروژه های MVVM Light موجود می‌باشد. قطعه کد پایین به صورت پیش فرض در پروژه‌های MVVM Lightایجاد می‌شود.
در کلاس ViewModelLocator ما تمام میانجی(Interface)ها و اشیا(Objects)ی مورد نیازمان را ثبت(register) می‌کنیم.
در ادامه اجزای مختلف آن را شرح می‌دهیم.

```
class ViewModelLocator
{
    static ViewModelLocator()
    {
        ServiceLocator.SetLocatorProvider(() => SimpleIoc.Default);
        if (ViewModelBase.IsInDesignModeStatic)
        {
            SimpleIoc.Default.Register<IDataService, Design.DesignDataService>();
        }
        else
        {
            SimpleIoc.Default.Register<IDataService, DataService>();
        }
        SimpleIoc.Default.Register<MainViewModel>();
        SimpleIoc.Default.Register<SecondViewModel>();
    }

    public MainViewModel Main
    {
        get
        {
            return ServiceLocator.Current.GetInstance<MainViewModel>();
        }
    }
}
```

1) هر شیء که به صورت پیش فرض ایجاد می‌شود با الگوی Singlton ایجاد می‌شود.

```
SimpleIoc.Default.GetInstance<MainViewModel>(Guid.NewGuid().ToString());
```

2) جهت ثبت یک کلاس مرتبط با میانجی آن از روش زیر استفاده می‌شود.

```
SimpleIoc.Default.Register<IDataService, Design.DesignDataService>();
```

3) جهت ثبت یک شی مرتبط با میانجی از روش زیر استفاده می‌شود.

```
SimpleIoc.Default.Register<IDataService>(myObject);
```

4) جهت ثبت یک نوع (Type) به طریق زیر عمل می‌کنیم.

```
SimpleIoc.Default.Register<MainViewModel>();
```

5) جهت گرفتن وهله (Instance) از یک میانجی خاص، از روش زیر استفاده می‌کنیم.

```
SimpleIoc.Default.GetInstance<IDataService>();
```

6) جهت گرفتن وهله ای به صورت مستقیم، 'ایجاد و وضوح وابستگی(dependency resolution)' از روش زیر استفاد می‌کنیم.

```
SimpleIoc.Default.GetInstance();
```

7) برای ایجاد داده‌های زمان طراحی از روش زیر استفاده می‌کنیم.

```
if (ViewModelBase.IsInDesignModeStatic)
        {
            SimpleIoc.Default.Register<IDataService, Design.DesignDataService>();
        }
        else
        {
            SimpleIoc.Default.Register<IDataService, DataService>();
        }
```

در حالت زمان طراحی، سرویس‌های زمان طراحی به صورت خودکار ثبت می‌شوند. و می‌توان این داده‌ها را در ViewModelها و Viewها حین طراحی مشاهده نمود.

منبع

عنوان: **آشنایی با Catel MVVM Frameowork**
نویسنده: محمّد زارع
تاریخ: ۱۳۹۲/۰۳/۱۳ ۱۶:۱۵
آدرس: www.dotnettips.info
گروه‌ها: MVVM, WPF, Catel

در این مقاله به بررسی اولیه فریمورک Catel و برخی ویژگی‌های آن خواهیم پرداخت.
همانطور که می‌دانید فریمورک‌های متعددی برای MVVM به وجود آمده اند، مانند MVVM Light یا Caliburn و Chinch و ... که هر کدام از آن‌ها دارای ویژگی هایی می‌باشند اما Catel تنها یک فریمورک برای MVVM نیست بلکه دارای قسمت‌های دیگری مانند کنترل‌های اختصاصی و سرویس‌های متعدد و پرکاربرد و Extensionهای مفید و ... نیز می‌باشد که کار توسعه یک برنامه MVVM را فوق العاده لذتبخش می‌کند.
برای شروع کار با این فریمورک ابتدا بایستی قالب پروژه را از این آدرس دریافت نمایید . بعد از دریافت و نصب آن یک زیرگروه جدید به نام Catel به قسمت افزودن پروژه جدید اضافه خواهد شد که شامل قالب پروژه برای WPF و Silverlight و Windows Phone و Windows Store می باشد. در این قسمت گزینه WPF Application with Catel را انتخاب نمایید و پروژه را ایجاد کنید.
بعد از ایجاد پروژه نوبت به نصب بسته های nuget مورد نیاز Catel می رسد. تنها بسته مورد نیاز Catel.Extensions.Controls می باشد که به صورت خودکار بسته های Catel.MVVM و Catel.Core را نیز نصب خواهد کرد. البته بسته‌های دیگری مانند Catel.Extensions.Prism, Catel.Extensions.FluentValidation و Catel.Extensions.Data و Catel.Fody و ... نیز برای این فریمورک وجود دارد که در این مطلب به آن‌ها نیازی نداریم.
اکنون ساختار اصلی پروژه ما ایجاد شده است. در این ساختار پوشه‌های Views، Models و ViewModels به صورت پیش فرض وجود دارند. Catel برای برقراری ارتباط بین View و ViewModel از IViewModelLocator ، IViewLocator و یکسری قواعد نام گذاری پیروی میکند تا نیاز به رجیستر کردن تک تک ویوها و ویومدل‌ها به صورت دستی نباشد که البته این قواعد قابل تغییر و شخصی سازی هستند. قرارداد پیش فرض برای پروژه‌های کوچک ممکن است مناسب باشد ولی در پروژه‌های بزرگ نیاز به سفارشی سازی دارد که در قسمت‌های بعد به آن خواهیم پرداخت.

View و ViewModel:

برای ایجاد یک ViewModel جدید، باید از منوی Add New Item قسمت Catel گزینه ViewModel (Catel) را انتخاب نمایید. با توجه به code snippet های تهیه شده برای این فریمورک، کار تهیه ViewModelها فوق العاده سریع انجام می‌شود. به عنوان مثال برای اضافه کردن یک Command در ویومدل، از vmcommand و یا vmcommandwithcanexecute و برای ایجاد پروپرتی هم از vmprop و vmpropchanged میتوان استفاده نمود. همانطور که ملاحظه می‌کنید نام این snippetها کاملا واضح می‌باشد و نیاز به توضیح اضافی ندارند.
همینطور برای ایجاد یک View گزینه DataWindow (WPF with Catel) را انتخاب نمایید. ViewModelها در Catel از کلاس پایه ViewModelBase و Viewها نیز از کلاس DataWindow مشتق می‌شوند.
DataWindow یک Window پیشرفته با قابلیت هایی مانند افزودن خودکار دکمه‌های Ok / Cancel یا Ok / Cancel / Apply یا Close می‌باشد که می‌تواند باعث تسریع روند ایجاد Windowهای تکراری شود. اما اگر به هیچ کدام از این دکمه‌های ذکر شده نیاز نداشتید DataWindowMode.Custom را انتخاب می‌کنید. نشان دادن Validation در بالای پنجره به صورت popup نیز یکی دیگر از قابلیت‌های این Window پیشرفته است. البته DataWindow دارای overloadهای مختلفی است که می‌توانید به کمک آن ویژگی‌های ذکر شده را فعال یا غیر فعال کنید.
حال برای درک بهتر commandها و نحوه تعریف و بکارگیری آن‌ها یک command جدید در MainWindowViewModel با استفاده از vmcommand ایجاد کنید. مانند قطعه کد زیر:

```
public class MainWindowViewModel : ViewModelBase
    {
        public MainWindowViewModel()
            : base()
        {
            ShowPleaseWait = new Command(OnShowPleaseWaitExecute);
        }

        public override string Title { get { return "View model title"; } }

        public Command ShowPleaseWait { get; private set; }
        private void OnShowPleaseWaitExecute()
        {
            var pleaseWaitService = GetService<IPleaseWaitService>();
```

```
            pleaseWaitService.Show(() =>
            {
                Thread.Sleep(3000);
            });
        }
    }
```

در داخل بدنه این command از PleaseWaitService استفاده کردیم که در ادامه توضیح داده خواهد شد. در MainView نیز یک button اضافه کنید و پروپرتی Command آن را به صورت زیر تنظیم کنید:

```
<Button Margin="6"
                Command="{Binding ShowPleaseWait}"
                Content="Show PleaseWait!" />
```

اکنون با فشردن button کد داخل بدنه command اجرا خواهد شد.

سرویس ها:

کتابخانه Catel.MVVM دارای سرویس‌های مختلف و پرکاربردی می‌باشد که در ادامه به بررسی آن‌ها خواهیم پرداخت:

PleaseWaitService: از این سرویس برای نشان دادن یک loading به کاربر در حین انجام یک کار سنگین استفاده می‌شود و نحوه استفاده از آن به صورت زیر است:

```
var pleaseWaitService = GetService<IPleaseWaitService>();
pleaseWaitService.Show(() =>
{
        Thread.Sleep(3000);
});
```

UIVisualizerService: از این سرویس برای باز کردن پنجره‌های برنامه استفاده می‌شود. هر View در برنامه دارای یک ViewModel می باشد. برای باز کردن View ابتدا یک نمونه از ViewModel مربوطه را ایجاد میکنیم و با دادن viewmodel به متد Show یا ShowDialog پنجره مورد نظر را باز میکنیم.

```
var uiService = GetService<IUIVisualizerService>();
var viewModel = new AnotherWindowViewModel();
uiService.Show(viewModel);
```

OpenFileService: برای نشان دادن OpenFileDialog جهت باز کردن یک فایل در برنامه.

```
var openFileService = GetService<IOpenFileService>();
openFileService.Filter = "ZIP files (*.zip)|*.zip";
openFileService.IsMultiSelect = false;
openFileService.Title = "Open file";
if (openFileService.DetermineFile())
{
       // ?
}
```

SaveFileService: برای نشان دادن SaveFileDialog جهت ذخیره سازی.

```
var saveFileService = GetService<ISaveFileService>();
saveFileService.Filter = "ZIP files (*.zip)|*.zip";
saveFileService.FileName = "test";
saveFileService.Title = "Save file";
if (saveFileService.DetermineFile())
{
       // ?
```

```
}
```

ProcessService: برای اجرا کردن یک process. به عنوان مثال برای باز کردن ماشین حساب ویندوز به صورت زیر عمل می‌کنیم:

```
var processService = GetSetvice<IProcessService>();
processService.StartProcess(@"C:\Windows\System32\calc.exe");
```

SplashScreenService: برای نشان دادن SplashScreen در ابتدای برنامه هایی که سرعت بالا آمدن پایینی دارند.

```
var splashScreenService = GetService<ISplashScreenService>();
splashScreenService.Enqueue(new ActionTask("Creating the shell", OnCreateShell));
splashScreenService.Enqueue(new ActionTask("Initializing modules", OnInitializeModules));
splashScreenService.Enqueue(new ActionTask("Starting application", OnStartApplication));
```

MessageService: برای نشان دادن MessageBox به کاربر.

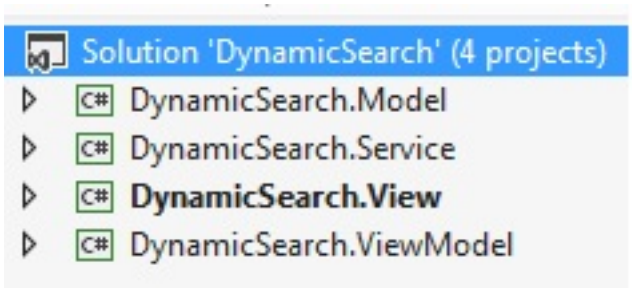
```
var messageService = GetService<IMessageService>();
if (messageService.Show("Are you sure?", "?", MessageButton.YesNo, MessageImage.Warning) ==
MessageResult.Yes)
{
	// ?
}
```

همانطور که ملاحظه کردید اکثر کارهای مورد نیاز یک پروژه با کمک سرویس‌های ارائه شده در این فریمورک به آسانی انجام می‌شود.

دریافت مثال و پروژه کامل این قسمت: TestApp.zip

عنوان: **ایجاد فرم جستجوی پویا با استفاده از Expression ها**
نویسنده: محمد عیدی مراد
تاریخ: ۱۳۹۲/۰۳/۲۵ ۹:۲۵
آدرس: www.dotnettips.info
برچسب‌ها: LINQ, MVVM, Expression, DynamicLINQ

در مواردی نیاز است کاربر را جهت انتخاب فیلدهای مورد جستجو آزاد نگه داریم. برای نمونه جستجویی را در نظر بگیرید که کاربر قصد دارد: "دانش آموزانی که نام آنها برابر علی است و شماره دانش آموزی آنها از 100 کمتر است" را پیدا کند در شرایطی که فیلدهای نام و شماره دانش آموزی و عمل گر کوچک‌تر را خود کاربر به دلخواه برگزیرده.
روش‌های زیادی برای پیاده سازی این نوع جستجوها وجود دارد. در این مقاله سعی شده گام‌های ایجاد یک ساختار پایه برای این نوع فرم‌ها و یک ایجاد فرم نمونه بر پایه ساختار ایجاد شده را با استفاده از یکی از همین روش‌ها شرح دهیم.
اساس این روش تولید عبارت Linq بصورت پویا با توجه به انتخاب‌های کاربرمی باشد.
1- برای شروع یک سلوشن خالی با نام DynamicSearch ایجاد می‌کنیم. سپس ساختار این سلوشن را بصورت زیر شکل می‌دهیم.



در این مثال پیاده سازی در قالب ساختار MVVM در نظر گرفته شده. ولی محدودتی از این نظر برای این روش قائل نیستیم.
2- کار را از پروژه مدل آغاز می‌کنیم. جایی که ما برای سادگی کار، 3 کلاس بسیار ساده را به ترتیب زیر ایجاد می‌کنیم:

```
namespace DynamicSearch.Model
{
    public class Person
    {
        public Person(string name, string family, string fatherName)
        {
            Name = name;
            Family = family;
            FatherName = fatherName;
        }

        public string Name { get; set; }
        public string Family { get; set; }
        public string FatherName { get; set; }
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DynamicSearch.Model
{
    public class Teacher : Person
    {
        public Teacher(int id, string name, string family, string fatherName)
            : base(name, family, fatherName)
        {
            ID = id;
        }

        public int ID { get; set; }

        public override string ToString()
```

```
        {
            return string.Format("Name: {0}, Family: {1}", Name, Family);
        }
    }
}

namespace DynamicSearch.Model
{
    public class Student : Person
    {
        public Student(int stdId, Teacher teacher, string name, string family, string fatherName)
            : base(name, family, fatherName)
        {
            StdID = stdId;
            Teacher = teacher;
        }

        public int StdID { get; set; }
        public Teacher Teacher { get; set; }
    }
}
```

-3 در پروژه سرویس یک کلاس بصورت زیر ایجاد می‌کنیم:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using DynamicSearch.Model;

namespace DynamicSearch.Service
{
    public class StudentService
    {
        public IList<Student> GetStudents()
        {
            return new List<Student>
                {
                    new Student(1,new Teacher(1,"Ali","Rajabi","Reza"),"Mohammad","Hoeyni","Sadegh"),
                    new Student(2,new Teacher(2,"Hasan","Noori","Mohsen"),"Omid","Razavi","Ahmad"),
                };
        }
    }
}
```

-4 تا اینجا تمامی داده‌ها صرفا برای نمونه بود. در این مرحله ساخت اساس جستجو گر پویا را شرح می‌دهیم.

جهت ساخت عبارت، نیاز به سه نوع جزء داریم:

-اتصال دهنده عبارات ("و" ، "یا")

-عملوند (در اینجا فیلدی که قصد مقایسه با عبارت مورد جستجوی کاربر را داریم)

-عملگر (">" ، "<" ، "=" ، ....)

برای ذخیره المان‌های انتخاب شده توسط کاربر، سه کلاس زیر را ایجاد می‌کنیم (همان سه جزء بالا):

```
using System;
using System.Linq.Expressions;

namespace DynamicSearch.ViewModel.Base
{
    public class AndOr
    {
        public AndOr(string name, string title,Func<Expression,Expression,Expression> func)
        {
            Title = title;
            Func = func;
            Name = name;
        }

        public string Title { get; set; }
        public Func<Expression, Expression, Expression> Func { get; set; }
        public string Name { get; set; }
```

```
    }
}

using System;

namespace DynamicSearch.ViewModel.Base
{
    public class Feild : IEquatable<Feild>
    {
        public Feild(string title, Type type, string name)
        {
            Title = title;
            Type = type;
            Name = name;
        }

        public Type Type { get; set; }
        public string Name { get; set; }
        public string Title { get; set; }
        public bool Equals(Feild other)
        {
            return other.Title == Title;
        }
    }
}

using System;
using System.Linq.Expressions;

namespace DynamicSearch.ViewModel.Base
{
    public class Operator
    {
        public enum TypesToApply
        {
            String,
            Numeric,
            Both
        }

        public Operator(string title, Func<Expression, Expression, Expression> func, TypesToApply
typeToApply)
        {
            Title = title;
            Func = func;
            TypeToApply = typeToApply;
        }

        public string Title { get; set; }
        public Func<Expression, Expression, Expression> Func { get; set; }
        public TypesToApply TypeToApply { get; set; }
    }
}
```

توسط کلاس زیر یک سری اعمال متداول را پیاده سازی کرده ایم و پیاده سازی اضافات را بعهده کلاس‌های ارث برنده از این کلاس گذاشته ایم:

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

namespace DynamicSearch.ViewModel.Base
{
    public abstract class SearchFilterBase<T> : BaseViewModel
    {
        protected SearchFilterBase()
        {
            var containOp = new Operator("شامل باشد", (expression, expression1) =>
Expression.Call(expression, typeof(string).GetMethod("Contains"), expression1),
Operator.TypesToApply.String);
            var notContainOp = new Operator("شامل نباشد", (expression, expression1) =>
            {
                var contain = Expression.Call(expression, typeof(string).GetMethod("Contains"),
expression1);
                return Expression.Not(contain);
            }, Operator.TypesToApply.String);
```

```
            var equalOp = new Operator("=", Expression.Equal, Operator.TypesToApply.Both);
            var notEqualOp = new Operator("<>", Expression.NotEqual, Operator.TypesToApply.Both);
            var lessThanOp = new Operator("<", Expression.LessThan, Operator.TypesToApply.Numeric);
            var greaterThanOp = new Operator(">", Expression.GreaterThan,
Operator.TypesToApply.Numeric);
            var lessThanOrEqual = new Operator("<=", Expression.LessThanOrEqual,
Operator.TypesToApply.Numeric);
            var greaterThanOrEqual = new Operator(">=", Expression.GreaterThanOrEqual,
Operator.TypesToApply.Numeric);

            Operators = new ObservableCollection<Operator>
                {
                      equalOp,
                      notEqualOp,
                      containOp,
                      notContainOp,
                      lessThanOp,
                      greaterThanOp,
                      lessThanOrEqual,
                      greaterThanOrEqual,
                };


            SelectedAndOr = AndOrs.FirstOrDefault(a => a.Name == "Suppress");
            SelectedFeild = Feilds.FirstOrDefault();
            SelectedOperator = Operators.FirstOrDefault(a => a.Title == "=");
        }

        public abstract IQueryable<T> GetQuarable();

        public virtual ObservableCollection<AndOr> AndOrs
        {
            get
            {
                return new ObservableCollection<AndOr>
                    {
                        new AndOr("And","و", Expression.AndAlso),
                        new AndOr("Or","یا",Expression.OrElse),
                        new AndOr("Suppress","نادیده",(expression, expression1) => expression),
                    };
            }
        }
        public virtual ObservableCollection<Operator> Operators
        {
            get { return _operators; }
            set { _operators = value; NotifyPropertyChanged("Operators"); }
        }
        public abstract ObservableCollection<Feild> Feilds { get; }

        public bool IsOtherFilters
        {
            get { return _isOtherFilters; }
            set { _isOtherFilters = value; }
        }
        public string SearchValue
        {
            get { return _searchValue; }
            set { _searchValue = value; NotifyPropertyChanged("SearchValue"); }
        }
        public AndOr SelectedAndOr
        {
            get { return _selectedAndOr; }
            set { _selectedAndOr = value; NotifyPropertyChanged("SelectedAndOr");
NotifyPropertyChanged("SelectedFeildHasSetted"); }
        }
        public Operator SelectedOperator
        {
            get { return _selectedOperator; }
            set { _selectedOperator = value; NotifyPropertyChanged("SelectedOperator"); }
        }
        public Feild SelectedFeild
        {
            get { return _selectedFeild; }
            set
            {
                Operators = value.Type == typeof(string) ? new
ObservableCollection<Operator>(Operators.Where(a => a.TypeToApply == Operator.TypesToApply.Both ||
a.TypeToApply == Operator.TypesToApply.String)) : new ObservableCollection<Operator>(Operators.Where(a
=> a.TypeToApply == Operator.TypesToApply.Both || a.TypeToApply == Operator.TypesToApply.Numeric));
                if (SelectedOperator == null)
                {
```

```
                    SelectedOperator = Operators.FirstOrDefault(a => a.Title == "=");
                }

                NotifyPropertyChanged("SelectedOperator");
                NotifyPropertyChanged("SelectedFeild");
                _selectedFeild = value;
                NotifyPropertyChanged("SelectedFeildHasSetted");
            }
        }
        public bool SelectedFeildHasSetted
        {
            get
            {
                return SelectedFeild != null &&
                       (SelectedAndOr.Name != "Suppress" || !IsOtherFilters);
            }
        }

        private ObservableCollection<Operator> _operators;
        private Feild _selectedFeild;
        private Operator _selectedOperator;
        private AndOr _selectedAndOr;
        private string _searchValue;
        private bool _isOtherFilters = true;
    }
}
```

توضیحات: در این ویو مدل پایه سه لیست تعریف شده که برای دو تای آنها پیاده سازی پیش فرضی در همین کلاس دیده شده ولی برای لیست فیلدها پیاده سازی به کلاس ارث برنده واگذار شده است.

در گام بعد، یک کلاس کمکی برای سهولت ساخت عبارات ایجاد می‌کنیم:

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

namespace DynamicSearch.ViewModel.Base
{
  public static  class ExpressionExtensions
    {
        public static List<T> CreateQuery<T>(Expression whereCallExpression, IQueryable entities)
        {
            return entities.Provider.CreateQuery<T>(whereCallExpression).ToList();
        }

        public static MethodCallExpression CreateWhereCall<T>(Expression condition, ParameterExpression
pe, IQueryable entities)
        {
            var whereCallExpression = Expression.Call(
                typeof(Queryable),
                "Where",
                new[] { entities.ElementType },
                entities.Expression,
                Expression.Lambda<Func<T, bool>>(condition, new[] { pe }));
            return whereCallExpression;
        }

        public static void CreateLeftAndRightExpression<T>(string propertyName, Type type, string
searchValue, ParameterExpression pe, out Expression left, out Expression right)
        {
            var typeOfNullable = type;
            typeOfNullable = typeOfNullable.IsNullableType() ? typeOfNullable.GetTypeOfNullable() :
typeOfNullable;
            left = null;

            var typeMethodInfos = typeOfNullable.GetMethods();
            var parseMethodInfo = typeMethodInfos.FirstOrDefault(a => a.Name == "Parse" &&
a.GetParameters().Count() == 1);

            var propertyInfos = typeof(T).GetProperties();
            if (propertyName.Contains("."))
```

```
            {
                left = CreateComplexTypeExpression(propertyName, propertyInfos, pe);
            }
            else
            {
                var propertyInfo = propertyInfos.FirstOrDefault(a => a.Name == propertyName);
                if (propertyInfo != null) left = Expression.Property(pe, propertyInfo);
            }

            if (left != null) left = Expression.Convert(left, typeOfNullable);

            if (parseMethodInfo != null)
            {
                var invoke = parseMethodInfo.Invoke(searchValue, new object[] { searchValue });
                right = Expression.Constant(invoke, typeOfNullable);
            }
            else
            {
                //type is string
                right = Expression.Constant(searchValue.ToLower());
                var methods = typeof(string).GetMethods();
                var firstOrDefault = methods.FirstOrDefault(a => a.Name == "ToLower" &&
!a.GetParameters().Any());
                if (firstOrDefault != null) left = Expression.Call(left, firstOrDefault);
            }
        }

        public static Expression CreateComplexTypeExpression(string searchFilter,
IEnumerable<PropertyInfo> propertyInfos, Expression pe)
        {
            Expression ex = null;
            var infos = searchFilter.Split('.');
            var enumerable = propertyInfos.ToList();
            for (var index = 0; index < infos.Length - 1; index++)
            {
                var propertyInfo = infos[index];
                var nextPropertyInfo = infos[index + 1];
                if (propertyInfos == null) continue;
                var propertyInfo2 = enumerable.FirstOrDefault(a => a.Name == propertyInfo);
                if (propertyInfo2 == null) continue;
                var val = Expression.Property(pe, propertyInfo2);
                var propertyInfos3 = propertyInfo2.PropertyType.GetProperties();
                var propertyInfo3 = propertyInfos3.FirstOrDefault(a => a.Name == nextPropertyInfo);
                if (propertyInfo3 != null) ex = Expression.Property(val, propertyInfo3);
            }

            return ex;
        }

        public static Expression AddOperatorExpression(Func<Expression, Expression, Expression> func,
Expression left, Expression right)
        {
            return func.Invoke(left, right);
        }

        public static Expression JoinExpressions(bool isFirst, Func<Expression, Expression, Expression>
func, Expression expression, Expression ex)
        {
            if (!isFirst)
            {
                return func.Invoke(expression, ex);
            }

            expression = ex;
            return expression;
        }
    }
}
```

-5 ایجاد کلاس فیلتر جهت معرفی فیلدها و معرفی منبع داده و ویو مدلی ارث برنده از کلاس‌های پایه ساختار، جهت ایجاد فرم نمونه:

```
using System.Collections.ObjectModel;
using System.Linq;
using DynamicSearch.Model;
using DynamicSearch.Service;
```

```
using DynamicSearch.ViewModel.Base;

namespace DynamicSearch.ViewModel
{
    public class StudentSearchFilter : SearchFilterBase<Student>
    {
        public override ObservableCollection<Feild> Feilds
        {
            get
            {
                return new ObservableCollection<Feild>
                    {
                        new Feild("نام دانش آموز",typeof(string),"Name"),
                         new Feild("نام خانوادگی دانش آموز",typeof(string),"Family"),
                        new Feild("نام خانوادگی معلم",typeof(string),"Teacher.Name"),
                        new Feild("شماره دانش آموزی",typeof(int),"StdID"),
                    };
            }
        }

        public override IQueryable<Student> GetQuarable()
        {
            return new StudentService().GetStudents().AsQueryable();
        }
    }
}
```

-6 ایجاد ویو نمونه:

در نهایت زمل فایل موجود در پروژه ویو:

```
<Window x:Class="DynamicSearch.View.MainWindow"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:viewModel="clr-namespace:DynamicSearch.ViewModel;assembly=DynamicSearch.ViewModel"
        xmlns:view="clr-namespace:DynamicSearch.View"
        mc:Ignorable="d"
             d:DesignHeight="300" d:DesignWidth="300">
    <Window.Resources>
        <viewModel:StudentSearchViewModel x:Key="StudentSearchViewModel" />
        <view:VisibilityConverter x:Key="VisibilityConverter" />
    </Window.Resources>
    <Grid   DataContext="{StaticResource StudentSearchViewModel}">
        <WrapPanel Orientation="Vertical">
            <DataGrid AutoGenerateColumns="False" Name="asd" CanUserAddRows="False"
ItemsSource="{Binding BindFilter}">
                <DataGrid.Columns>
                    <DataGridTemplateColumn>
                        <DataGridTemplateColumn.CellTemplate>
                            <DataTemplate DataType="{x:Type viewModel:StudentSearchFilter}">
                                <ComboBox MinWidth="100"  DisplayMemberPath="Title"
ItemsSource="{Binding AndOrs}" Visibility="{Binding IsOtherFilters,Converter={StaticResource
VisibilityConverter}}"
                                          SelectedItem="{Binding
SelectedAndOr,Mode=TwoWay,UpdateSourceTrigger=PropertyChanged}" />
                            </DataTemplate>
                        </DataGridTemplateColumn.CellTemplate>
                    </DataGridTemplateColumn>
                    <DataGridTemplateColumn >
                        <DataGridTemplateColumn.CellTemplate>
                            <DataTemplate DataType="{x:Type viewModel:StudentSearchFilter}">
                                <ComboBox IsEnabled="{Binding SelectedFeildHasSetted}" MinWidth="100"
DisplayMemberPath="Title" ItemsSource="{Binding Feilds}" SelectedItem="{Binding
SelectedFeild,Mode=TwoWay,UpdateSourceTrigger=PropertyChanged }"/>
                            </DataTemplate>
                        </DataGridTemplateColumn.CellTemplate>
                    </DataGridTemplateColumn>
                    <DataGridTemplateColumn>
                        <DataGridTemplateColumn.CellTemplate>
                            <DataTemplate DataType="{x:Type viewModel:StudentSearchFilter}">
                                <ComboBox MinWidth="100"  DisplayMemberPath="Title"
ItemsSource="{Binding Operators}" IsEnabled="{Binding SelectedFeildHasSetted}"
                                          SelectedItem="{Binding
```

```
SelectedOperator,Mode=TwoWay,UpdateSourceTrigger=PropertyChanged}" />
                            </DataTemplate>
                        </DataGridTemplateColumn.CellTemplate>
                    </DataGridTemplateColumn>
                    <DataGridTemplateColumn Width="*">
                        <DataGridTemplateColumn.CellTemplate>
                            <DataTemplate DataType="{x:Type viewModel:StudentSearchFilter}">
                                <TextBox IsEnabled="{Binding SelectedFeildHasSetted}" MinWidth="200"
Text="{Binding SearchValue,Mode=TwoWay,UpdateSourceTrigger=PropertyChanged}"/>
                                <!--<TextBox Text="{Binding
SearchValue,Mode=TwoWay,UpdateSourceTrigger=PropertyChanged}"/>-->
                            </DataTemplate>
                        </DataGridTemplateColumn.CellTemplate>
                    </DataGridTemplateColumn>
                </DataGrid.Columns>
            </DataGrid>

            <Button Content="+" HorizontalAlignment="Left" Command="{Binding AddFilter}"/>
            <Button Content="Result" Command="{Binding ExecuteSearchFilter}"/>
            <DataGrid ItemsSource="{Binding Results}">

            </DataGrid>
        </WrapPanel>
    </Grid>
</Window>
```

در این مقاله، هدف معرفی روند ایجاد یک جستجو گر پویا با قابلیت استفاده مجدد بالا بود و عمدا از توضیح جزء به جزء کدها صرف نظر شده. علت این امر وجود منابع بسیار راجب ابزارهای بکار رفته در این مقاله و سادگی کدهای نوشته شده توسط اینجانب می‌باشد.

برخی منابع جهت آشنایی با Expression ها:

http://msdn.microsoft.com/en-us/library/bb882637.aspx

انتخاب پویای فیلدها در LINQ

http://www.persiadevelopers.com/articles/dynamiclinqquery.aspx

نکته: کدهای نوشته شده در این مقاله، نسخه‌های نخستین هستند و طبیعتا جا برای بهبود بسیار دارند. دوستان می‌توانند در این امر به بنده کمک کنند.

پیشنهادات جهت بهبود:
- جداسازی کدهای پیاده کننده منطق از ویو مدل‌ها جهت افزایش قابلیت نگهداری کد و سهولت استفاده در سایر ساختارها
- افزودن توضیحات به کد
- انتخاب نامگذاری‌های مناسب تر

DynamicSearch.zip

عنوان: **آموزش Prism #1**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۴/۰۱ ۸:۱۵
آدرس: www.dotnettips.info
گروه‌ها: MVVM, Silverlight, WPF, prism

امروزه تقریبا تمام کسانی که پروژه‌های WPF یا Silverlight رو توسعه می‌دهند با مدل برنامه نویسی MVVM آشنایی دارند. فریم ورک‌های مختلفی برای توسعه پروژه‌ها به صورت MVVM وجود دارد. نظیر:

MVVM Light

Prism

Caliburn

Cinch

WAF

Catel

Onyx

MVVM helpers

و...

هر کدوم از فریم ورک‌های بالا مزایا، معایب و طرفداران خاص خودشون رو دارند(^) ولی به جرات می‌تونیم Prism رو به عنوان قوی‌ترین فریم ورک برای پیاده سازی پروژهای بزرگ و قوی و ماژولار با تکنولوژی WPF یا Silverlight بنامیم. در این پست به معرفی و بررسی مفاهیم اولیه Prism خواهیم پرداخت و در پست‌های دیگر به پیاده سازی عملی همراه با مثال می‌پردازیم.

*اگر به هر دلیلی مایل به یادگیری و استفاده از Prism نیستید، بهتون پیشنهاد می‌کنم از WAF استفاده کنید.

پیش نیازها:

برای یادگیری PRISM ابتدا باید با مفاهیم زیر در WPF یا Silverlight آشنایی داشته باشید.(فرض بر این است که به UserControl و Xaml و Dependency Properties، **تسلط** کامل دارید)

Data binding

Resources

Commands

Behaviors

چرا Prism ؟

Prism به صورت کامل از Modular Programming برای پروژه‌های WPF و Silverlight پشتیانی می‌کند*

از Prism هم می‌توانیم در پروژه‌های WPF استفاده کنیم و هم Silverlight.

Prism به صورت کامل از الگوی MVVM برای پیاده سازی پروژه‌ها پشتیبانی می‌کند.

پیاده سازی مفاهیمی نظیر Composite Command و Command Behavior و Asynchronous Interacion به راحتی در Prism امکان پذیر است.

مفاهیم تزریق وابستگی به صورت توکار در Prism فراهم است که برای پیاده سازی این مفاهیم به طور پیش فرض امکان استفاده از UnityContainer و MEF در Prism تدارک دیده شده است.

پیاده سازی Region navigation در Prism به راحتی امکان پذیر است.

به وسیله امکان Event Aggregation به راحتی می‌توانیم بین ماژول‌های مختلف ارتباط برقرار کنیم.

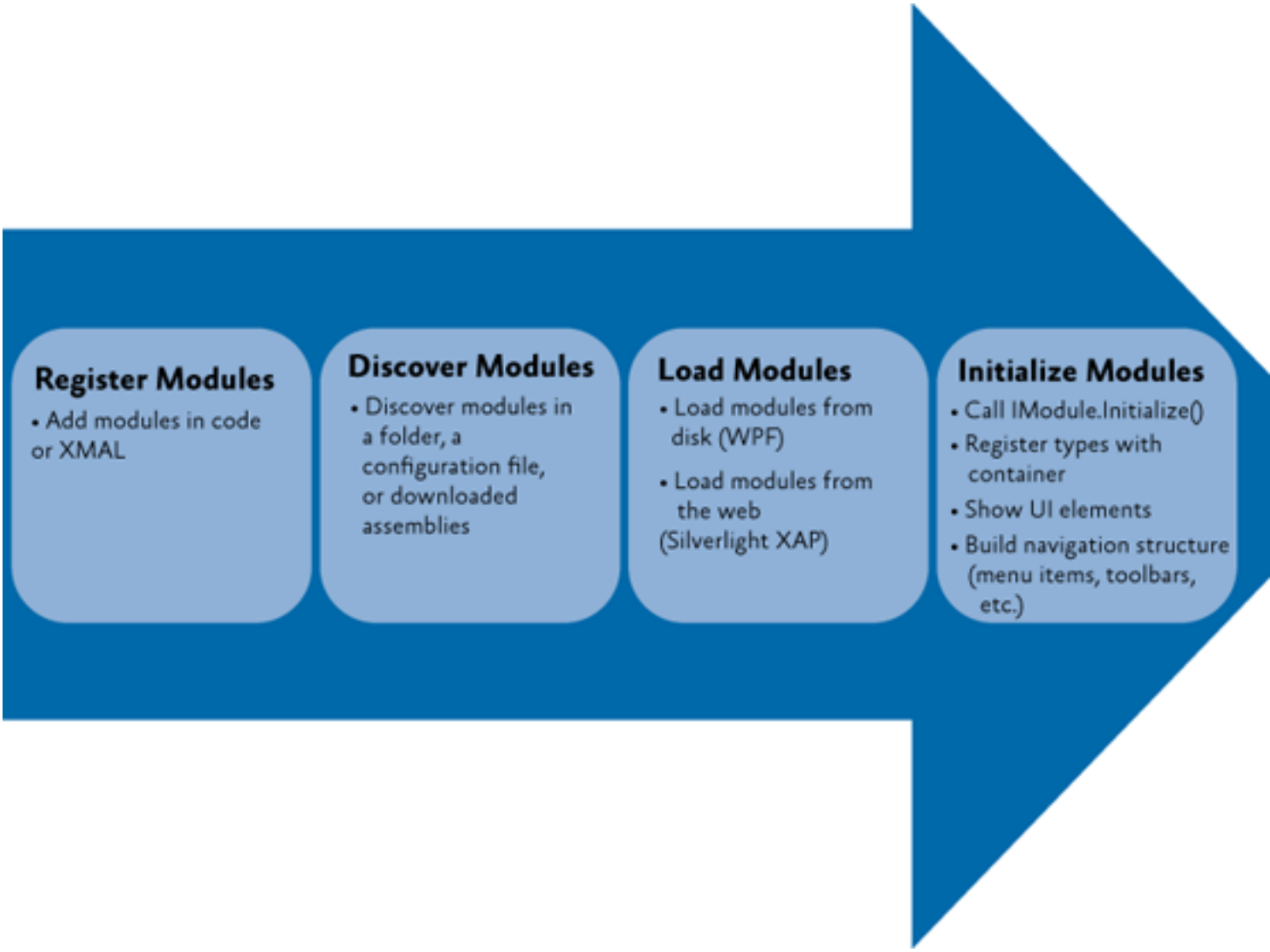
***توضیح درباره برنامه‌های ماژولار**

در تولید پروژهای نرم افزاری بزرگ هر چه قدر هم اگر در تهیه فایل‌های اسمبلی، کلاس ها، اینترفیس‌ها و کلا طراحی پروژه به صورت شی گرا دقت به خرج دهیم باز هم ممکن است پروژه به صورت یک پارچه طراحی نشود. یعنی بعد از اتمام پروژه، توسعه، تست پذیری و نگهداری آن سخت و در بعضی مواقع غیر ممکن خواهد شد. برنامه نویسی ماژولار این امکان را فراهم می‌کنه که یک پروژه با مقیاس برزگ به چند پروژه کوچک تقسیم شده و همه مراحل طراحی و توسعه و تست برای هر کدام از این ماژول‌ها به صورت جدا انجام شود.

Prism امکاناتی رو برای طراحی و توسعه این گونه پروژه‌ها به صورت ماژولار فراهم کرده است:

ابتدا باید نام و مکان هر ماژول رو به Prism معرفی کنیم که می‌تونیم اون‌ها رو در کد یا Xaml یا Configuration File تعریف کنیم.
با استفاده از Metadata باید وابستگی‌ها و مقادیر اولیه برای هر ماژول مشخص شود.
با کمک تزریق وابستگی‌ها ارتباطات بین ماژول‌ها میسر می‌شود.
ماژول مورد نظر به دو صورت OnDemand و Available لود خواهد شد.

در شکل زیر مراحل بالا قابل مشاهده است:

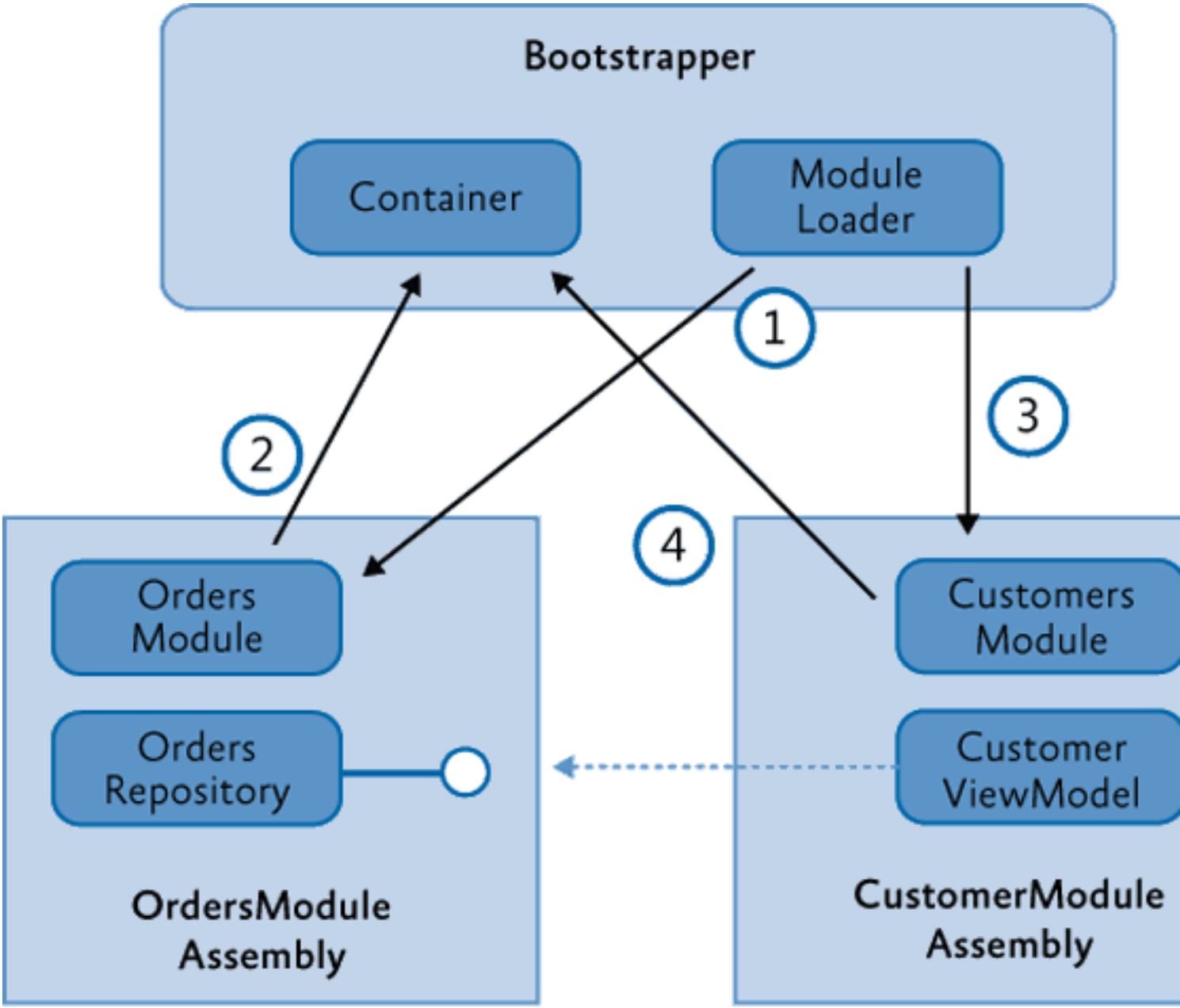


Bootstrapper چیست؟

در هر پروژه ماژولار (مختص Prism نیست) برای اینکه ماژول‌های مختلف یک پروژه، قابلیت استفاده به صورت یک پارچه رو در یک Application داشته باشند باید مفهومی به نام Bootstapper رو پیاده سازی کنیم که وظیفه اون شناسایی و پیکربندی و لود ماژول هاست. در Prism دو نوع Bootstrapper پیش فرض وجود دارد.

MefBootstrapper : کلاس پایه Bootstrapper که مبنای آن MEF است. اگر قصد استفاده از MEF رو در پروژه‌های خود دارید(^) Bootstrapper شما باید از این کلاس ارث ببرد.
UnityBootstrapper : کلاس پایه Bootstrapper که مبنای آن UnityContainer است. اگر قصد استفاده از UnityContainer یا Service Locator (^) رو در پروژه‌های خود دارید Bootstrapper شما باید از این کلاس ارث ببرد.

تصویری از ارتباط Bootstrapper با ماژول‌های سیستم



مفهوم Shell

در پروژه‌های WPF، در فایل App.xaml توسط یک Uri نقطه شروع پروژه را تعیین می‌کنیم. در پروژه‌های Silverlight به وسیله خاصیت RootVisual نقطه شروع سیستم تعیین می‌شود. در Prism نقطه شروع پروژه توسط bootsrapper تعیین می‌شود. دلیل این امر این است که Shell در پروژه‌های مبتنی بر Prism متکی بر Region Manager است. از Region برای لود و نمایش ماژول‌ها استفاده خواهیم کرد.

ادامه دارد...

نظرات خوانندگان

نویسنده: محمد احمدی
تاریخ: ۱۳۹۲/۰۴/۰۱ ۱۰:۶

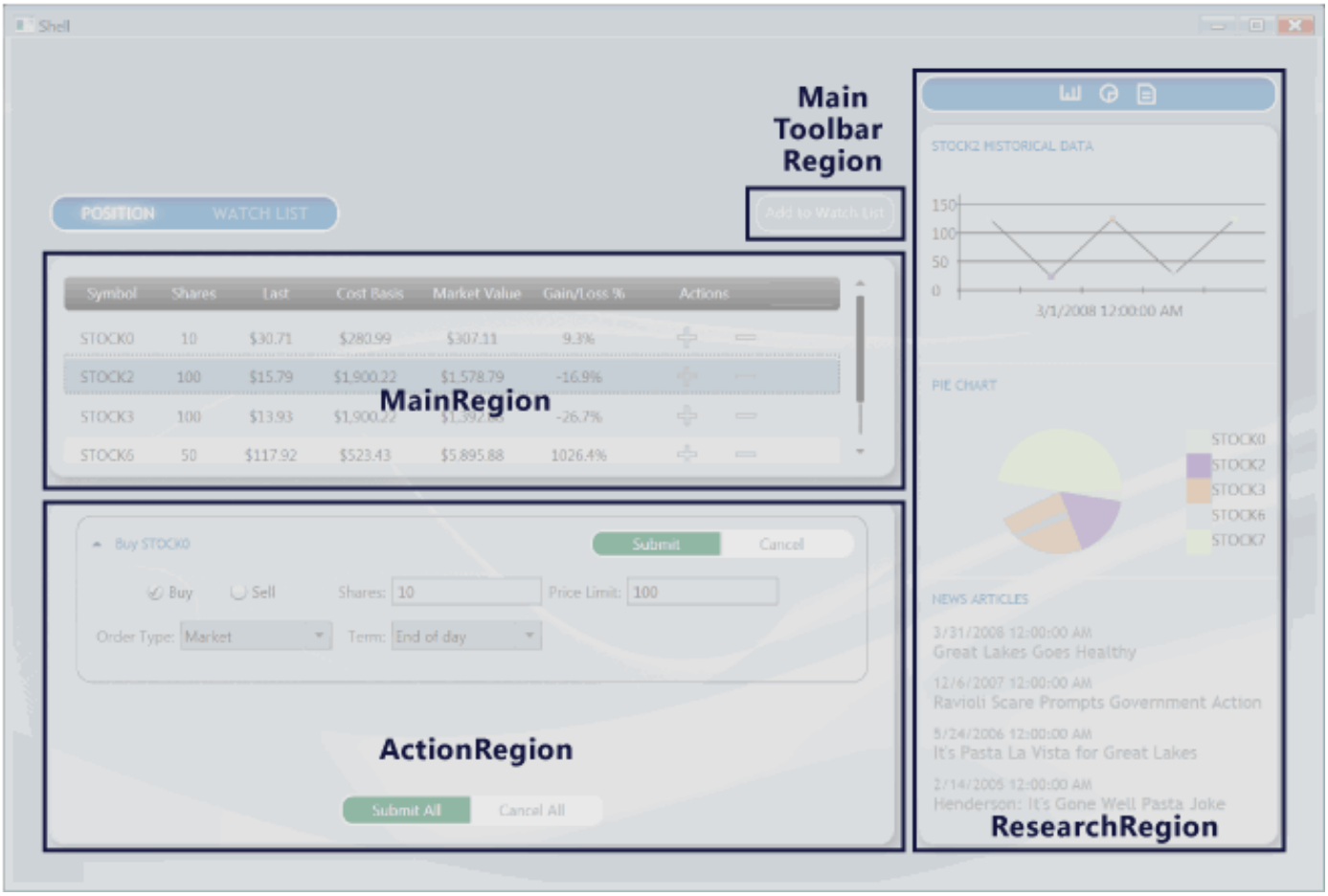
با سلام و تشکر از مطلب مفیدتون
همانطور که می‌دانید مدل‌های مختلف توسعه MVVM برای مقاصد مختلف بهتر است و به طور کلی نمی‌توان گفت که کدام بهتر است لطفا در ادامه مطلب این فریم ورک را با MVVM Light هم مقایسه بفرمائید تا موارد استفاده هر کدام بهتر مشخص شود

نویسنده: مسعود م.پاکدل
تاریخ: ۱۳۹۲/۰۴/۰۱ ۱۱:۵۶

ممنون.
من از Prism به عنوان بهترین فریم ورک نام نبردم بلکه از عنوان قوی‌ترین فریم ورک استفاده کردم
"می‌تونیم Prism رو به عنوان قوی‌ترین فریم ورک برای پیاده سازی پروژهای بزرگ و قوی و ماژولار با تکنولوژی WPF یا Silverlight بنامیم." که لزوما به معنی بهترین نیست.
MVVM Light در حال حاضر به عنوان محبوب‌ترین فریم ورک برای MVVM است که این محبوبیت بیشتر به خاطر راحتی کار با اون هست.

MVVM Light نظیر Prism هم قابلیت استفاده در WPF را دارد و هم Silverlight (مزیت). MVVM Light راهکار مشخصی برای پیاده سازی پروژه‌های ماژولار ندارد(منظور Modular Composite Application است) در حالی که Prism برای تولید Modular Composite Applicationها طراحی شده است. برای اینکه بتونید، بعضی از قابلیت‌ها موجود در Prism را برای پروژه‌های ماژولار شبیه سازی کنید باید از ترکیب MEF و MVVM Light استفاده کنید.

Prism به شما این امکان رو می‌ده که حتی برای Popup Windowها هم Region طراحی کنید(مزیت). با Prism می‌تونید به راحتی برای یک Command رفتار تعریف کنید(به صورت توکار از Interactionها استفاده می‌کنه(مزیت)) برای این کار در MVVM Light شما باید از EventToCommandها استفاده کنید که اصلا قابل مقایسه به مباحث Composite Command و Command Behavior نیست.
معادل Messaging در MVVM Light در Prism شما EventAggregatorها رو در اختیار دارید.
Prism به صورت توکار از dependency Injection استفاده میکنه و دو فریم ورک هم به شما پیشنهاد میده یکی MEF و دیگری UnityContainer(مزیت).
Prism به صورت توکار از Composite UI هم پشتیبانی می‌کند. به تصویر زیر دقت کنید:



به راحتی می‌تونید با استفاده از RegionManager موجود در Prism نواحی هر صفحه رو تقسیم بندی کنید و هر ناحیه هم می‌تونه توسط یک ماژول لود شود. برای طراحی و مدیریت صفحات در MVVM Light باید خودتون دست به کار بشید.
یادگیری و استفاده از قابلیت‌های MVVM Light در حد دو یا سه روز زمان می‌برد در حالی که برای یادگیری قابلیت‌های Prism یک کتاب نوشته شده است(^)
*در پایان دوباره تاکید می‌کنم که اگر نیازی به تولید و توسعه پروژه به صورت ماژولار رو ندارید بهتره که اصلا به Prism فکر نکنید.

نویسنده: Petek
تاریخ: ۱۳۹۲/۰۴/۰۲ ۰:۳۶

با سلام
دوست عزیز ممنون میشم این مطلب جالب و مفید رو هر چه بیشتر و سریعتر ادامه بدید . با تشکر

نویسنده: محمد احمدی
تاریخ: ۱۳۹۲/۰۴/۰۲ ۱۳:۱۴

دوست عزیز
ممنونم از راهنمایی جامع و مفیدتون . امیدوارم هر چه زودتر مطالب بیشتری در این زمینه از شما یاد بگیریم

عنوان: **آموزش Prism #2**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۴/۰۳ ۹:۵
آدرس: www.dotnettips.info
گروه‌ها: MVVM, WPF, prism

در پست قبلی توضیح کلی درباره فریم ورک Prism داده شد. در این بخش قصد داریم آموزش‌های داده شده در پست قبلی را با هم در یک مثال مشاهده کنیم. در پروژه‌های ماژولار طراحی و ایجاد زیر ساخت قوی برای مدیریت ماژول‌ها بسیار مهم است. Prism فریم ورکی است که فقط چارچوب و قواعد اصول طراحی این گونه پروژه‌ها را در اختیار ما قرار می‌دهد. در پروژه‌های ماژولار هر ماژول باید در یک اسمبلی جدا قرار داشته باشد که ساختار پیاده سازی آن می‌تواند کاملا متفاوت با پیاده سازی سایر ماژول‌ها باشد.
برای شروع باید فایل‌های اسمبلی Prism رو دانلود کنید(لینک دانلود).

تشریح پروژه:

می‌خواهیم برنامه ای بنویسیم که دارای سه ماژول زیر است.:

ماژول Navigator : برای انتخاب و Switch کردن بین ماژول‌ها استفاده می‌شود؛
ماژول طبقه بندی کتاب‌ها : لیست طبقه بندی کتاب‌ها را به ما نمایش می‌دهد؛
ماژول لیست کتاب‌ها : عناوین کتاب‌ها به همراه نویسنده و کد کتاب را به ما نمایش می‌دهد.

*در این پروژه از UnityContainer برای مباحث Dependency Injection استفاده شده است.
ابتدا یک پروژه WPF در Vs.Net ایجاد کنید(در اینجا من نام آن را FirstPrismSample گذاشتم). قصد داریم یک صفحه طراحی کنیم که دو ماژول مختلف در آن لود شود. ابتدا باید Shell پروژه رو طراحی کنیم. یک Window جدید به نام Shell بسازید و کد زیر را

در آن کپی کنید.

```
<Window x:Class="FirstPrismSample.Shell"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:com="http://www.codeplex.com/CompositeWPF"
    Title="Prism Sample By Masoud Pakdel" Height="400" Width="600"
WindowStartupLocation="CenterScreen">
    <DockPanel>
      <ContentControl com:RegionManager.RegionName="WorkspaceRegion" Width="400"/>
      <ContentControl com:RegionManager.RegionName="NavigatorRegion"  DockPanel.Dock="Left" Width="200"
/>
    </DockPanel>
</Window>
```

در این صفحه دو ContentControl تعریف کردم یکی به نام Navigator و دیگری به نام Workspace. به وسیله RegionName که یک AttachedProperty است هر کدوم از این نواحی را برای Prism تعریف کردیم. حال باید یک ماژول برای Navigator و دو ماژول دیگر یکی برای طبقه بندی کتاب‌ها و دیگری برای لیست کتاب‌ها بسازیم.

#پروژه Common

قبل از هر چیز یک پروژه Common می‌سازیم و مشترکات بین ماژول‌ها رو در آن قرار می‌دهیم(این پروژه باید به تمام ماژول‌ها رفرنس داده شود). این مشترکات شامل :

کلاس پایه ViewModel

کلاس ViewRequestEvent

کلاس ModuleService

کد کلاس ViewModelBase که فقط اینترفیس INotifyPropertyChanged رو پیاده سازی کرده است:

```
using System.ComponentModel;

namespace FirstPrismSample.Common
{
    public abstract class ViewModelBase : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;
        protected void RaisePropertyChangedEvent( string propertyName )
        {
            if ( PropertyChanged != null )
            {
                PropertyChangedEventArgs e = new PropertyChangedEventArgs( propertyName );
                PropertyChanged( this, e );
            }
        }
    }
}
```

کلاس ViewRequestEvent که به صورت زیر است:

```
using Microsoft.Practices.Composite.Presentation.Events;

namespace FirstPrismSample.Common.Events
{
    public class ViewRequestedEvent : CompositePresentationEvent<string>
    {
    }
}
```

توضیح درباره CompositePresentationEvent :

در طراحی و توسعه پروژه‌های ماژولار نکته ای که باید به آن دقت کنید این است که ماژول‌های پروژه نباید به هم وابستگی مستقیم داشته باشند در عین حال ماژول‌ها باید بتوانند با هم در ارتباط باشند. CPE یا **C** omposite **P** resentation **E** vent دقیقا برای این

منظور به وجود آمده است. CPE که در این جا طراحی کردم فقط کلاسی است که از CompositePresentationEventارث برده است و دلیل آن که به صورت string generic استفاده شده است این است که می‌خواهیم در هر درخواست نام ماژول درخواستی را داشته باشیم و به همین دلیل نام آن را ViewRequestedEvent گذاشتم.

توضیح درباره EventAggregator

EventAggregator یا به اختصار EA مکانیزمی است در پروژهای ماژولار برای اینکه در Composite UIها بتوانیم بین کامپوننت‌ها ارتباط برقرار کنیم. استفاده از EA وابستگی بین ماژول‌ها را از بین خواهد برد. برنامه نویسانی که با MVVM Light آشنایی دارند از قابلیت Messaging موجود در این فریم ورک برای ارتباط بین View و ViewModel استفاده می‌کنند. در Prism این عملیات توسط EA انجام می‌شود. یعنی برای ارتباط با Viewها باید از EA تعبیه شده در Prism استفاده کنیم. در ادامه مطلب، چگونگی استفاده از EA را خواهید آموخت.

اینترفیس IModuleService که فقط شامل یک متد است:

```
namespace FirstPrismSample .Common
{
    public interface IModuleServices
    {
        void ActivateView(string viewName);
    }
}
```

کلاس ModuleService که اینترفیس بالا را پیاده سازی کرده است:

```
using Microsoft.Practices.Composite.Regions;
using Microsoft.Practices.Unity;

namespace FirstPrismSample.Common
{
    public class ModuleServices : IModuleServices
    {
        private readonly IUnityContainer m_Container;

        public ModuleServices(IUnityContainer container)
        {
            m_Container = container;
        }

        public void ActivateView(string viewName)
        {
            var regionManager = m_Container.Resolve<IRegionManager>();

            // غیر فعال کردن ویو
            IRegion workspaceRegion = regionManager.Regions["WorkspaceRegion"];
            var views = workspaceRegion.Views;
            foreach (var view in views)
            {
                workspaceRegion.Deactivate(view);
            }

            //فعال کردن ویو انتخاب شده
            var viewToActivate = regionManager.Regions["WorkspaceRegion"].GetView(viewName);
            regionManager.Regions["WorkspaceRegion"].Activate(viewToActivate);
        }
    }
}
```

متد ActivateView نام view مورد نظر برای فعال سازی را دریافت می‌کند. برای فعال کردن View ابتدا باید سایر viewهای فعال در RegionManager را غیر فعال کنیم. سپس فقط view مورد نظر در RegionManager انتخاب و فعال می‌شود.

*نکته: در هر ماژول ارجاع به اسمبلی‌های Prism مورد نیاز است.

#ماژول طبقه بندی کتاب ها:

برای شروع یک Class Library جدید به نام ModuleCategory به پروژه اضافه کنید. یک UserControl به نام CategoryView

بسازید و کدهای زیر را در آن کپی کنید.

```
<UserControl x:Class="FirstPrismSample.ModuleCategory.CategoryView "
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             Background="LightGray" FlowDirection="RightToLeft" FontFamily="Tahoma">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>
        <TextBlock Text=" طبقه بندی ها"/>
        <ListView Grid.Row="1"  Margin="10" Name="lvCategory">
            <ListView.View>
                <GridView>
                    <GridViewColumn Header="کد" Width="50" />
                    <GridViewColumn Header="عنوان" Width="200"  />
                </GridView>
            </ListView.View>
        </ListView>
    </Grid>
</UserControl>
```

یک کلاس به نام CategoryModule بسازید که اینترفیس IModule رو پیاده سازی کند.

```
using Microsoft.Practices.Composite.Events;
using Microsoft.Practices.Composite.Modularity;
using Microsoft.Practices.Composite.Regions;
using Microsoft.Practices.Unity;
using FirstPrismSample.Common;
using FirstPrismSample.Common.Events;
using Microsoft.Practices.Composite.Presentation.Events;

namespace FirstPrismSample.ModuleCategory
{
    [Module(ModuleName = "ModuleCategory")]
    public class CategoryModule : IModule
    {
        private readonly IUnityContainer m_Container;
        private readonly string moduleName = "ModuleCategory";

        public CategoryModule(IUnityContainer container)
        {
            m_Container = container;
        }

        ~CategoryModule()
        {
            var eventAggregator = m_Container.Resolve<IEventAggregator>();
            var viewRequestedEvent = eventAggregator.GetEvent<ViewRequestedEvent>();
            viewRequestedEvent.Unsubscribe(ViewRequestedEventHandler);
        }

        public void Initialize()
        {
            var regionManager = m_Container.Resolve<IRegionManager>();
            regionManager.Regions["WorkspaceRegion"].Add(new CategoryView(), moduleName);

            var eventAggregator = m_Container.Resolve<IEventAggregator>();
            var viewRequestedEvent = eventAggregator.GetEvent<ViewRequestedEvent>();
            viewRequestedEvent.Subscribe(this.ViewRequestedEventHandler, true);
        }

        public void ViewRequestedEventHandler(string moduleName)
        {
            if (this.moduleName != moduleName) return;

            var moduleServices = m_Container.Resolve<IModuleServices>();
            moduleServices.ActivateView(moduleName);
        }
    }
}
```

چند نکته :

*ModuleAttribute استفاده شده در بالای کلاس برای تعیین نام ماژول استفاده می‌شود. این Attribute دارای دو خاصیت دیگر

هم است :

OnDemand : برای تعیین اینکه ماژول باید به صورت OnDemand (بنا به درخواست) لود شود.
StartupLoaded : برای تعیین اینکه ماژول به عنوان ماژول اول پروژه لود شود.(البته این گزینه Obsolute شده است)

*برای تعریف ماژول کلاس مورد نظر حتما باید اینترفیس IModule را پیاده سازی کند. این اینترفیس فقط شامل یک متد است به نام Initialize.

*در این پروژه چون Viewهای برنامه صرفا جهت نمایش هستند در نتیجه نیاز به ایجاد ViewModel برای آن‌ها نیست. در پروژه‌های اجرایی حتما برای هر View باید ViewModel متناظر با آن تهیه شود.

توضیح درباره متد Initialize

در این متد ابتدا با استفاده از Container موجود RegionManager را به دست می‌آوریم. با استفاده از RegionManager می‌تونیم یک CompositeUI طراحی کنیم. در فایل Shell مشاهده کردید که یک صفحه به دو ناحیه تقسیم شد و به هر ناحیه هم یک نام اختصاص دادیم. دستور زیر به یک ناحیه اشاره خواهد داشت:

```
regionManager.Regions["WorkspaceRegion"]
```

در خط بعد با استفاده از EA یا Event Aggregator توانستیم CPE را بدست بیاوریم. متد Subscribe در کلاس CPE یک ارجاع قوی به delegate مورد نظر ایجاد می‌کند(پارامتر دوم این متد که از نوع boolean است) که به این معنی است که این delegate هیچ گاه توسط GC جمع آوری نخواهد شد. در نتیجه، قبل از اینکه ماژول بسته شود باید به صورت دستی این کار را انجام دهیم که مخرب را برای همین ایجاد کردیم. اگر به کدهای مخرب دقت کنید می‌بینید که با استفاده از EA توانستیم ViewRequestEventHandler را Unsubscribe کنیم به دلیل اینکه از ارجاع قوی با strong Reference در متد Subscribe استفاده شده است. دستور moduleService.ActiveateView ماژول مورد نظر را در region مورد نظر هاست خواهد کرد.

#ماژول لیست کتاب ها:

ابتدا یک Class Library به نام ModuleBook بسازید و همانند ماژول قبلی نیاز به یک Window و یک کلاس داریم:
BookWindow که کاملا مشابه به CategoryView است.

```
<UserControl x:Class="FirstPrismSample.ModuleBook.BookView"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Background="LightGray" FontFamily="Tahoma" FlowDirection="RightToLeft">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>
        <TextBlock Text="لیست کتاب ها"/>
        <ListView Grid.Row="1" Margin="10" Name="lvBook">
            <ListView.View>
                <GridView>
                    <GridViewColumn Header="کد" Width="50"  />
                    <GridViewColumn Header="عنوان" Width="200" />
                    <GridViewColumn Header="نویسنده" Width="150" />
                </GridView>
            </ListView.View>
        </ListView>
    </Grid>
</UserControl>
```

کلاس BookModule که پیاده سازی و توضیحات آن کاملا مشابه به CategoryModule می‌باشد.

```
using Microsoft.Practices.Composite.Events;
using Microsoft.Practices.Composite.Modularity;
using Microsoft.Practices.Composite.Presentation.Events;
using Microsoft.Practices.Composite.Regions;
using Microsoft.Practices.Unity;
using FirstPrismSample.Common;
using FirstPrismSample.Common.Events;

namespace FirstPrismSample.ModuleBook
{
    [Module(ModuleName = "moduleBook")]
    public class BookModule : IModule
    {
        private readonly IUnityContainer m_Container;
        private readonly string moduleName = "ModuleBook";

        public BookModule(IUnityContainer container)
        {
            m_Container = container;
        }

        ~BookModule()
        {
            var eventAggregator = m_Container.Resolve<IEventAggregator>();
            var viewRequestedEvent = eventAggregator.GetEvent<ViewRequestedEvent>();

            viewRequestedEvent.Unsubscribe(ViewRequestedEventHandler);
        }

        public void Initialize()
        {
            var regionManager = m_Container.Resolve<IRegionManager>();
            var view = new BookView();
            regionManager.Regions["WorkspaceRegion"].Add(view, moduleName);
            regionManager.Regions["WorkspaceRegion"].Deactivate(view);

            var eventAggregator = m_Container.Resolve<IEventAggregator>();
            var viewRequestedEvent = eventAggregator.GetEvent<ViewRequestedEvent>();
            viewRequestedEvent.Subscribe(this.ViewRequestedEventHandler, true);
        }

        public void ViewRequestedEventHandler(string moduleName)
        {
            if (this.moduleName != moduleName) return;

            var moduleServices = m_Container.Resolve<IModuleServices>();
            moduleServices.ActivateView(m_WorkspaceBName);
        }
    }
}
```

#ماژول Navigator

برای این ماژول هم ابتدا View مورد نظر را ایجاد می‌کنیم:

```
<UserControl x:Class="FirstPrismSample.ModuleNavigator.NavigatorView"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" >
    <Grid>
        <StackPanel VerticalAlignment="Center">
            <TextBlock Text="انتخاب ماژول" Foreground="Green" HorizontalAlignment="Center"
                VerticalAlignment="Center" FontFamily="Tahoma" FontSize="24" FontWeight="Bold" />
        <Button Command="{Binding ShowModuleCategory}" Margin="5" Width="125">طبقه بندی کتاب
ها</Button>
        <Button Command="{Binding ShowModuleBook}" Margin="5" Width="125">لیست کتاب ها</Button>
        </StackPanel>
        </Grid>
</UserControl>
```

حال قصد داریم برای این View یک ViewModel بسازیم. نام آن را INavigatorViewModel خواهیم گذاشت:

```
public interface INavigatorViewModel
    {
        ICommand ShowModuleCategory { get; set; }
        ICommand ShowModuleBook { get; set; }
```

```
        string ActiveWorkspace { get; set; }

        IUnityContainer Container { get; set; }

        event PropertyChangedEventHandler PropertyChanged;
    }
```

*در اینترفیس بالا دو Command داریم که هر کدام وظیفه لود یک ماژول را بر عهده دارند.
*خاصیت ActiveWorkspace برای تعیین workspace فعال تعریف شده است.

حال به پیاده سازی مثال بالا می‌پردازیم:

```
public class NavigatorViewModel : ViewModelBase, INavigatorViewModel
    {
        public NavigatorViewModel(IUnityContainer container)
        {
            this.Initialize(container);
        }

        public ICommand ShowModuleCategory { get; set; }

        public ICommand ShowModuleBook { get; set; }

        public string ActiveWorkspace { get; set; }

        public IUnityContainer Container { get; set; }

        private void Initialize(IUnityContainer container)
        {
            this.Container = container;
            this.ShowModuleCategory = new ShowModuleCategoryCommand(this);
            this.ShowModuleBook = new ShowModuleBookCommand(this);
            this.ActiveWorkspace = "ModuleCategory";
        }
    }
```

تنها نکته مهم در کلاس بالا متد Initialize است که دو Command مورد نظر را پیاده سازی کرده است. ماژول پیش فرض هم ماژول طبقه بندی کتاب‌ها یا ModuleCategory در نظر گرفته شده است. همان طور که می‌بینید پیاده سازی Commandها بالا توسط دو کلاس ShowModuleCategoryCommand و ShowModuleBookCommand انجام شده که در زیر کدهای آن‌ها را می‌بینید.

#کد کلاس ShowModuleCategoryCommand

```
public class ShowModuleCategoryCommand : ICommand
    {
        private readonly NavigatorViewModel viewModel;
        private const string workspaceName = "ModuleCategory";

        public ShowModuleCategoryCommand(NavigatorViewModel viewModel)
        {
            this.viewModel = viewModel;
        }

        public bool CanExecute(object parameter)
        {
            return viewModel.ActiveWorkspace != workspaceName;
        }

        public event EventHandler CanExecuteChanged
        {
            add { CommandManager.RequerySuggested += value; }
            remove { CommandManager.RequerySuggested -= value; }
        }

        public void Execute(object parameter)
        {
            CommandServices.ShowWorkspace(workspaceName, viewModel);
        }
    }
```

#کد کلاس ShowModuleBookCommand

```
public class ShowModuleBookCommand : ICommand
    {
        private readonly NavigatorViewModel viewModel;
        private readonly string workspaceName = "ModuleBook";

        public ShowModuleBookCommand( NavigatorViewModel viewModel )
        {
            this.viewModel = viewModel;
        }

        public bool CanExecute( object parameter )
        {
            return viewModel.ActiveWorkspace != workspaceName;
        }

        public event EventHandler CanExecuteChanged
        {
            add { CommandManager.RequerySuggested += value; }
            remove { CommandManager.RequerySuggested -= value; }
        }

        public void Execute( object parameter )
        {
            CommandServices.ShowWorkspace( workspaceName , viewModel );
        }
    }
```

با توجه به این که فرض است با متدهای Execute و CanExecute و CanExecuteChanged آشنایی دارید از توضیح این مطالب خودداری خواهم کرد. فقط کلاس CommandServices در متد Execute دارای متدی به نام ShowWorkspace است که کدهای زیر را شامل می‌شود:

```
public static void ShowWorkspace(string workspaceName, INavigatorViewModel viewModel)
  {
            var eventAggregator = viewModel.Container.Resolve<IEventAggregator>();
            var viewRequestedEvent = eventAggregator.GetEvent<ViewRequestedEvent>();
            viewRequestedEvent.Publish(workspaceName);

            viewModel.ActiveWorkspace = workspaceName;
 }
```

در این متد با استفاده از CPE که در پروژه Common ایجاد کردیم ماژول مورد نظر را لود خواهیم کرد. و بعد از آن مقدار ActiveWorkspace جاری در ViewModel به نام ماژول تغییر پیدا می‌کند. متد Publish در CPE این کار را انجام خواهد دارد.

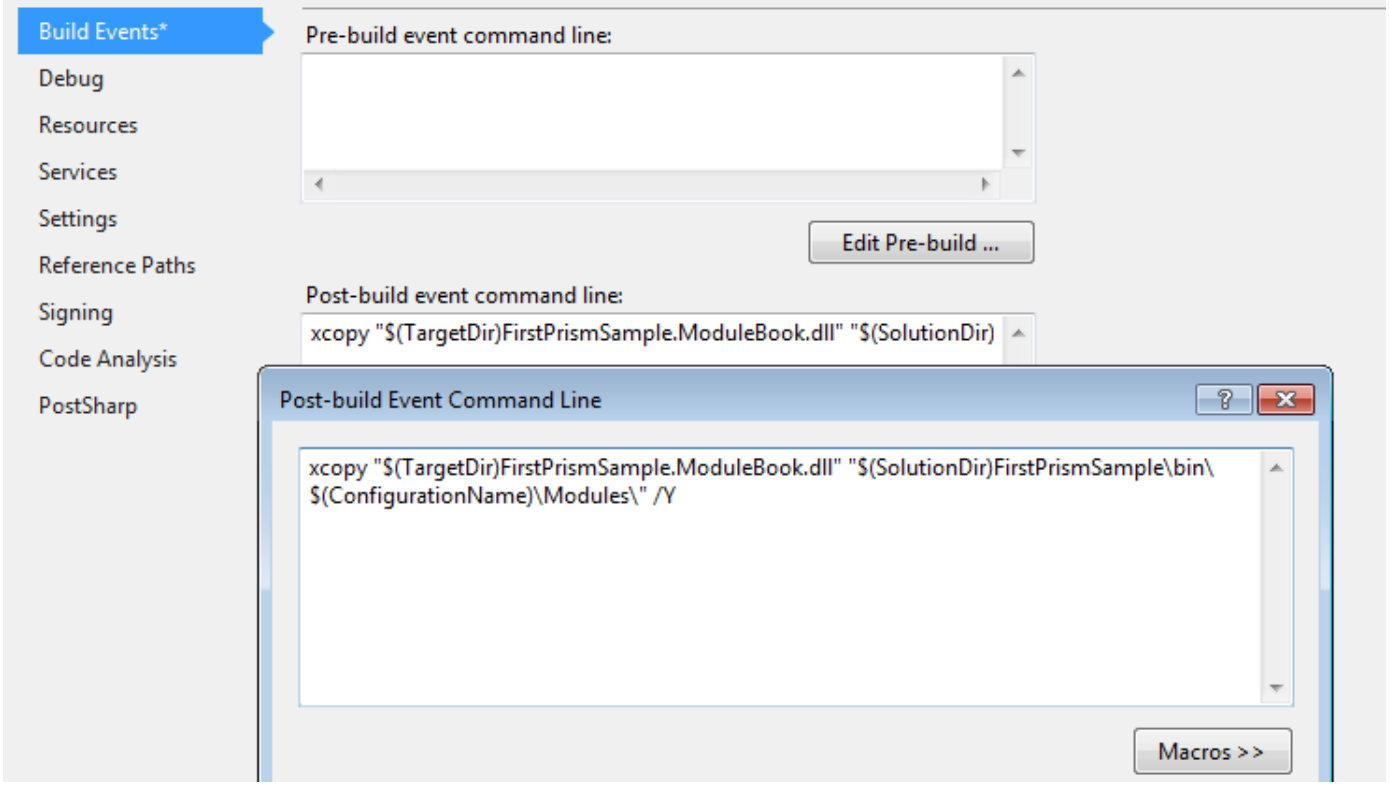
عدم وابستگی ماژول ها

همان طور که می‌بینید ماژول‌های پروژه به هم Reference داده نشده اند حتی هیچ Reference هم به پروژه اصلی یعنی جایی که فایل App.xaml قرار دارد، داده نشده است ولی در عین حال باید با هم در ارتباط باشند. برای حل این مسئله این ماژول‌ها باید در فولدر bin پروژه اصلی خود را کپی کنند. بهترین روش استفاده از Pre-Post Build Event خود VS.Net است. برای این کار از پنجره Project Properties وارد برگه Build Events شوید و از قسمت Post Build Event Command Line استفاده کنید و کد زیر را در آن کپی نمایید:

```
xcopy "$(TargetDir)FirstPrismSample.ModuleBook.dll"
"$(SolutionDir)FirstPrismSample\bin\$(ConfigurationName)\Modules\" /Y
```

قطعا باید به جای FirstPrismSample نام Solution خود و به جای ModuleBook نام ماژول را وارد نمایید.

مانند:



مراحل بالا برای هر ماژول باید تکرار شود(ModuleNavigation , ModuleBook , ModuleCategory). بعد از Rebuild پروژه در فولدر bin پروژه اصلی یک فولدر به نام Module ایجاد می‌شود که اسمبلی هر ماژول در آن کپی خواهد شد.

ایجاد Bootstrapper

حال نوبت به Bootstrapper میرسد(در پست قبلی در باره مفهوم Bootstrapper شرح داده شد). در پروژه اصلی یعنی جایی که فایل App.xaml قرار دارد کلاس زیر را ایجاد کنید.

```
public class Bootstrapper : UnityBootstrapper
    {
        protected override void ConfigureContainer()
        {
            base.ConfigureContainer();
            Container.RegisterType<IModuleServices, ModuleServices>();
        }

        protected override DependencyObject CreateShell()
        {
            var shell = new Shell();
            shell.Show();
            return shell;
        }

        protected override IModuleCatalog GetModuleCatalog()
        {
            var catalog = new DirectoryModuleCatalog();
            catalog.ModulePath = @".\Modules";
            return catalog;
        }
    }
```

متد ConfigureContainer برای تزریق وابستگی به وسیله UnityContainer استفاده می‌شود. در این متد باید تمامی Registrationهای مورد نیاز برای DI را انجام دهید. نکته مهم این است که عملیات وهله سازی و Initialization برای Container در متد base کلاس UnityBootstrapper انجام خواهد شد پس همیشه باید متد base این کلاس در ابتدای این متد فراخوانی شود در غیر این صورت با خطا متوقف خواهید شد.

متد CreateShell برای ایجاد و وهله سازی از Shell پروژه استفاده می‌شود. در این جا یک وهله از Shell Window برگشت داده می‌شود.
متد GetModuleCatalog برای تعیین مسیر ماژول‌ها در پروژه کاربرد دارد. در این متد با استفاده از خاصیت ModulePath کلاس DirectoryModuleCatalog تعیین کرده ایم که ماژول‌های پروژه در فولدر Modules موجود در bin اصلی پروژه قرار دارد. اگر به دستورات کپی در Post Build Event قسمت قبل توجه کنید می‌بینید که دستور ساخت فولدر وجود دارد.

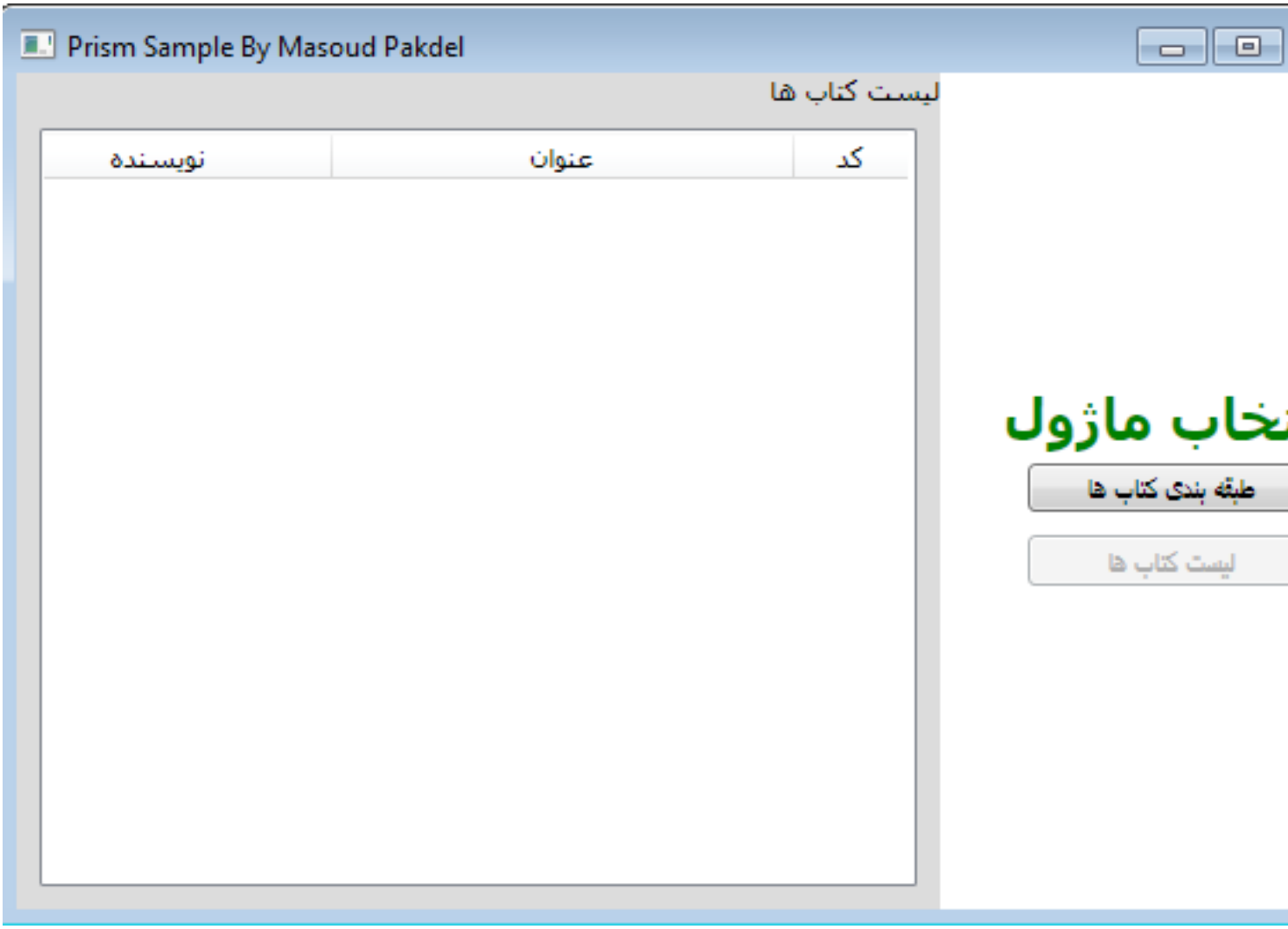
```
"$(SolutionDir)FirstPrismSample\bin\$(ConfigurationName)\Modules\" /Y
```

***نکته:** اگر استفاده از این روش برای شناسایی ماژول‌ها توسط Bootstrapper را چندان جالب نمی‌دانید می‌تونید از MEF استفاده کنید که اسمبلی ماژول‌های پروژه را به راحتی شناسایی می‌کند و در اختیار Bootsrtapper قرار می‌دهد(از آن جا در مستندات مربوط به Prism، بیشتر به استفاده از MEF تاکید شده است من هم در پست‌های بعدی، مثال‌ها را با MEF پیاده سازی خواهم کرد)

در پایان باید فایل App.xaml را تغییر دهید به گونه ای که متد Run در کلاس Bootstapper ابتدا اجرا شود.

```
public partial class App : Application
    {
        protected override void OnStartup(StartupEventArgs e)
        {
            base.OnStartup(e);
            var bootstrapper = new Bootstrapper();
            bootstrapper.Run();
        }
    }
```

اجرای پروژه:
بعد از اجرا، با انتخاب ماژول مورد نظر اطلاعات ماژول در Workspace Content Control لود خواهد شد.


Prism Sample By Masoud Pakdel
لیست کتاب ها
کد
عنوان
نویسنده
طبقه بندی کتاب ها
لیست کتاب ها

ادامه دارد...

نظرات خوانندگان

نویسنده: Petek
تاریخ: ۱۳۹۲/۰۴/۰۳ ۱۰:۲۷

با سلام مهندس
خیلی عالیه امیدوارم ادامه بدید . با تشکر

نویسنده: مهدی
تاریخ: ۱۳۹۲/۰۴/۰۳ ۱۹:۵۶

ممنون از آموزش خوبتون ، نظرتون در مورد استفاده از Prism به همراه StructerMap چیه ؟

نویسنده: مسعود م.پاکدل
تاریخ: ۱۳۹۲/۰۴/۰۳ ۲۲:۲۳

شدنی است. فقط همانند UnityBootstrapper نیاز به یک StructureMapBootstrapper دارید. این کار قبلا توسط Richard Cerirol انجام شده. می‌تونید از nuget استفاده کنید:

```
PM> Install-Package Prism.StructureMapExtensions
```

نویسنده: بهنام
تاریخ: ۱۳۹۲/۰۴/۰۵ ۱:۲۶

با سلام و با تشکر مطلب مفیدتان
چند اصلاح کوچک در مطلب هست که اینجا بیان می‌کنم
بخش اول (مبدا)دستور xcopy باید به دستور زیر تبدیل شود:

```
xcopy "$(SolutionDir)\PrismProject.ModuleBook\bin\$(ConfigurationName)\PrismProject.ModuleBook.dll"
"$(SolutionDir)PrismProject\bin\$(ConfigurationName)\Modules\" /Y
```

همچنین متد GetModuleCatalog به CreateModuleCatalog تبدیل شده است.
با تشکر مجدد

نویسنده: مسعود م.پاکدل
تاریخ: ۱۳۹۲/۰۴/۰۵ ۹:۳۰

ممنونم دوست عزیز.
در مورد دستور اول روش ذکر شده کاملا صحیح است و نیازی به اصلاح نیست.

$TargetDir دقیقا به مسیر فایل‌های اجرایی اشاره می‌کند و $ConfigurationName را در خودش پشتیبانی می‌کنه. یعنی اگر پروژه در حال Relase باشد با استفاده از $TargetDir دقیقا به فایل‌های موجود در فولدر Release در bin پروژه اشاره می‌کند و در حالت Debug به فایل‌های موجود در فولدر Debug در bin پروژه. با استفاده از گزینه Macros درقسمت Edit Post-Build مشاهده می‌کنید که مقدار $TargetDir دقیقا صحیح است. اما دلیل اینکه چرا در بخش دوم دستور از $SolutionDir استفاده شده است به این دلیل است که می‌خواهیم به فولدر bin پروژه اصلی اشاره داشته باشیم و چون این پروژه حتما در مسیر Solution جاری خواهد بود در نتیجه از این آدرس استفاده شده است.(در این جا TargetDir و TargetPath نمی‌تواند کمکی به ما بکند). به تصویر زیر دقت کنید:(چون پروژه در حالت release است در نتیجه مقادیر TargetDir و TargetPath به release ختم می‌شود)

Macro	Value
OutDir	bin\Release\
ConfigurationName	Release
ProjectName	XLIFFProject
TargetName	WpfApplication١
TargetPath	E:\Workspace\Projects\XLIFFProject\XLIFFProject\bin\Release\WpfApplication١.exe
ProjectPath	E:\Workspace\Projects\XLIFFProject\XLIFFProject\XLIFFProject.csproj
ProjectFileName	XLIFFProject.csproj
TargetExt	.exe
TargetFileName	WpfApplication١.exe
DevEnvDir	C:\Program Files (x٨٦)\Microsoft Visual Studio ١٠,٠\Common٧\IDE\
TargetDir	E:\Workspace\Projects\XLIFFProject\XLIFFProject\bin\Release\
ProjectDir	E:\Workspace\Projects\XLIFFProject\XLIFFProject\
SolutionFileName	XLIFFProject.sln
SolutionPath	E:\Workspace\Projects\XLIFFProject\XLIFFProject.sln
SolutionDir	E:\Workspace\Projects\XLIFFProject\
SolutionName	XLIFFProject
PlatformName	x٨٦
ProjectExt	.csproj
SolutionExt	.sln

به تفاوت مقادیر بین $TargetDir و $TargetPath و $SolutionDir و ... دقت کنید.

در مورد متد GetModuleCatalog هم باید عنوان کنم که این متد در اسمبلی Microsoft.Practices.Composite.UnityExtensions ورژن 2.0.1.0 وجود دارد. در ورژن 4 نسخه Prism این متد به این نام تغییر کرده است. در این جا می‌تونید تغییرات بین Prism Library 2 و Prism Library 4 رو ببینید

نویسنده: یوسف
تاریخ: ۱۳۹۲/۰۴/۲۲ ۱۹:۴۹

درود؛

لطفاً سورس پروژهٔ مثال را هم جهت دانلود اینجا بذارین، چون توی مقاله اشاره‌ای به اینکه پروژه‌ها از چه نوعی باشند و کدوم رفرنس‌ها را لازم دارند نشده و برای یکی مثل من که کلاً آشناییش با مقالات شما آغاز شده پیشرفت کار خیلی کند میشه.
سپاسگزارم.

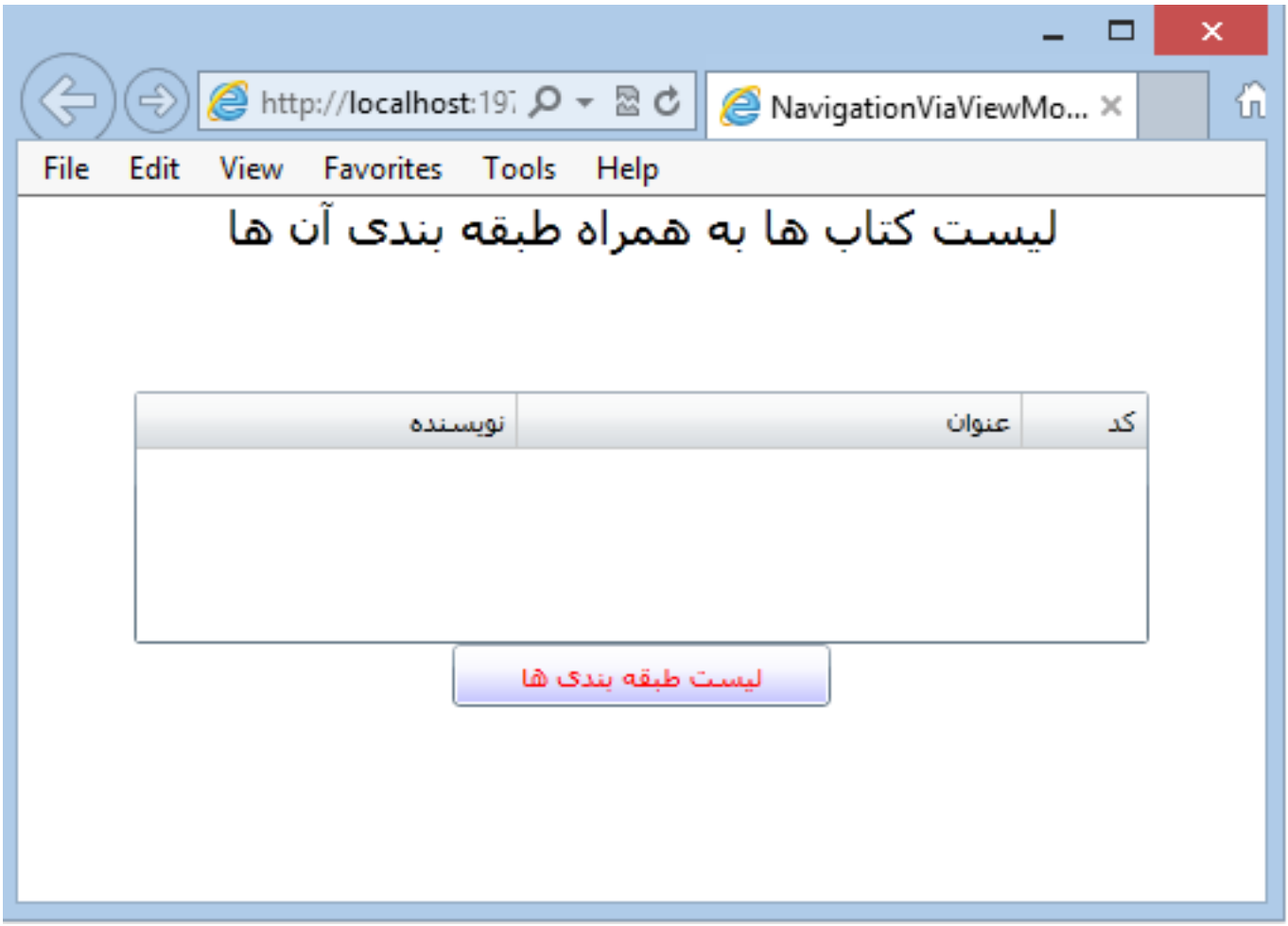
نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۴/۲۲ ۲۰:۱۶

در قسمت سوم ، سورس پیوست شده

عنوان: **آموزش Prism #3**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۴/۱۶ ۸:۳۰
آدرس: www.dotnettips.info
گروه‌ها: MVVM, Silverlight, WPF, prism

در پست‌های قبلی با Prism و روش استفاده از آن آشنا شدیم (قسمت اول) و (قسمت دوم). در این پست با استفاده از Mef قصد ایجاد یک پروژه Silverlight رو به صورت ماژولار داریم. مثال پیاده سازی شده در پست قبلی را در این پست به صورت دیگر پیاده سازی خواهیم کرد.
تفاوت‌های پیاده سازی مثال پست قبلی با این پست:
در مثال قبل پروژه به صورت Desktop و با WPF پیاده سازی شده بود ولی در این مثال با Silverlight می‌باشد؛
در مثال قبل از UnityBootstrapper استفاده شده بود ولی در این مثال از MefBootstrapper؛
در مثال قبل هر View در یک ماژول قرار داشت ولی در این مثال هر دو View را در یک ماژول قرار دادم؛
در مثال قبل از Prism Libary 2.x استفاده شده بود ولی در این مثال از PrismLibrary 4.x؛
و...

نکته : برای فهم بهتر مفاهیم، آشنایی اولیه با MEF و مفاهیمی نظیر Export و Import و AggregateCatalog و AssemblyCatalog نیاز است. در صورتی که با این مطالب آشنایی ندارید می‌توانید از (^) شروع کنید.



برای شروع یک پروژه Silverlight ایجاد کنید. بعد از اضافه شدن دو پروژه Silverlight و Web، یک Silverlight Class

Library جدید بسازید.
ابتدا یک Page ایجاد کنید و کدهای زیر را در آن کپی کنید.

```
<UserControl
             x:Class="Module1.Module1View1"
             xmlns:sdk="http://schemas.microsoft.com/winfx/2006/xaml/presentation/sdk"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" FlowDirection="RightToLeft"
FontFamily="Tahoma">
    <StackPanel>
        <sdk:DataGrid Height="100">
            <sdk:DataGrid.Columns>
                <sdk:DataGridTextColumn Header="کد" Width="50"  />
                <sdk:DataGridTextColumn Header="عنوان" Width="200" />
                <sdk:DataGridTextColumn Header="نویسنده" Width="150"  />
            </sdk:DataGrid.Columns>
        </sdk:DataGrid>
        <Button x:Name="NextViewButton"
                Width="150"
                Height="25"
                Foreground="Red"
                Background="Blue"
                Content="لیست طبقه بندی ها" />
    </StackPanel>
</UserControl>
```

بر روی Page مربوطه راست کلیک کنید و گزینه ViewCode را انتخاب کنید و کدهای زیر را در آن کپی کنید.

```
[Export(typeof(Module1View1))]
    public partial class Module1View1 : UserControl
    {
        [Import]
        public IRegionManager TheRegionManager { private get; set; }

        public Module1View1()
        {
            InitializeComponent();

            NextViewButton.Click += NextViewButton_Click;
        }

        void NextViewButton_Click(object sender, RoutedEventArgs e)
        {
            TheRegionManager.RequestNavigate
            (
                "MyRegion1",
                new Uri("Module1View2", UriKind.Relative),
                a => { }
            );
        }
    }
```

ابتدا خود این View باید حتما Export شود. در رویداد کلیک با استفاده از متد RequestNavigate می‌توانیم به View مورد نظر برای نمایش در Shell اشاره کنیم و این View در Region نمایش داده می‌شود. به دلیل اینکه در این کلاس به RegionManager نیاز داریم از ImportAttribute استفاده کردیم. این بدین معنی است که کلاس Module1View1 وابستگی مستقیم به IRegionManager دارد.

حال یک Page دیگر برای طبقه بندی کتاب‌ها ایجاد کنید و کدهای زیر را در آن کپی کنید.

```
<UserControl
             xmlns:sdk="http://schemas.microsoft.com/winfx/2006/xaml/presentation/sdk"
x:Class="Module1.Module1View2"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" FlowDirection="RightToLeft"
FontFamily="Tahoma">
    <StackPanel>
        <sdk:DataGrid Height="100">
            <sdk:DataGrid.Columns>
                <sdk:DataGridTextColumn Header="کد" Width="150"/>
                <sdk:DataGridTextColumn Header="عنوان" Width="150"/>
```

```
            </sdk:DataGrid.Columns>
        </sdk:DataGrid>
        <Button x:Name="NextViewButton"
                Width="150"
                Height="25"
                Foreground="Green"
                Background="Yellow"
                Content="لیست کتاب ها" />
    </StackPanel>
</UserControl>
```

در Code Behind این Page نیز کدهای زیر را قرار دهید.

```
using Microsoft.Practices.Prism.Regions;
using System;
using System.ComponentModel.Composition;
using System.Windows;
using System.Windows.Controls;

namespace Module1
{
    [Export]
    public partial class Module1View2 : UserControl
    {
        IRegion _region1;

        [ImportingConstructor]
        public Module1View2( [Import] IRegionManager regionManager )
        {
            InitializeComponent();

            ViewModel viewModel = new ViewModel();
            DataContext = viewModel;

            viewModel.ShouldNavigateFromCurrentViewEvent += () => { return true; };

            _region1 = regionManager.Regions["MyRegion1"];

            NextViewButton.Click += NextViewButton_Click;
        }

        void NextViewButton_Click( object sender, RoutedEventArgs e )
        {
            _region1.RequestNavigate
            (
                new Uri( "Module1View1", UriKind.Relative ),
                a => { }
            );
        }
    }
}
```

در این ماژول برای اینکه بتوانیم حالت گردشی در فراخوانی ماژول‌ها را داشته باشیم ابتدا DataContext این کلاس را برابر با ViewModel ساخته شده قرار دادیم. با استفاده از رویداد ShoudlNavigateFromCurrentViewEvent که در کلاس ViewModel وجود دارد تعیین می‌کنیم که آیا باید از این View به View قبلی برگشت داشته باشیم یا نه. در صورتی که مقدار false برگشت داده شود خواهید دید که امکان فراخوانی View1 از View2 امکان پذیر نیست. در رویداد کلیک نیز همانند Page قبلی با استفاده از RegionManager و متد RequestNavigate به View مورد نظر راهبری کرده ایم.

نکته: اگر یک کلاس، سازنده با پارامتر داشته باشد باید با استفاده از ImportingConstructor حتما سازنده مورد نظر را هنگام وهله سازی مشخص کنیم در غیر این صورت با Exception مواجه خواهید شد.

حال قصد ایجاد کلاس ViewModel بالا را داریم:

```
using System;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Ink;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
```

```
using System.Windows.Shapes;
using System.ComponentModel.Composition;
using Microsoft.Practices.Prism.Regions;

namespace Module1
{
    public class ViewModel : IConfirmNavigationRequest
    {
        public event Func<bool> ShouldNavigateFromCurrentViewEvent;

        public bool IsNavigationTarget( NavigationContext navigationContext )
        {
            return true;
        }

        public void OnNavigatedTo( NavigationContext navigationContext )
        {

        }

        public void OnNavigatedFrom( NavigationContext navigationContext )
        {

        }

        public void ConfirmNavigationRequest( NavigationContext navigationContext, Action<bool>
continuationCallback )
        {
            bool shouldNavigateFromCurrentViewFlag = false;

            if ( ShouldNavigateFromCurrentViewEvent != null )
                shouldNavigateFromCurrentViewFlag = ShouldNavigateFromCurrentViewEvent();

            continuationCallback( shouldNavigateFromCurrentViewFlag );
        }
    }
}
```

توضیح متدهای بالا:

IsNavigateTarget : برای تعیین اینکه آیا کلاس پیاده سازی کننده اینترفیس، می‌تواند عملیات راهبری را مدیریت کند یا نه.

OnNavigateTo : زمانی عملیات راهبری وارد View شود(بهتره بگم View مورد نظر در Region صفحه لود شود) این متد فراخوانی می‌شود.

OnNavigateFrom : زمانی که راهبری از این View خارج می‌شود (View از حالت لود خارج می‌شود) این متد فراخوانی خواهد شد.

ConfirmNavigationRequest : برای تایید عملیات راهبری توسط کلاس پیاده سازی کننده اینترفیس استفاده می‌شود.

حال یک کلاس برای پیاده سازی و مدیریت ماژول می‌سازیم.

```
using Microsoft.Practices.Prism.MefExtensions.Modularity;
using Microsoft.Practices.Prism.Modularity;
using Microsoft.Practices.Prism.Regions;
using System.ComponentModel.Composition;

namespace Module1
{
    [ModuleExport(typeof(Module1Impl))]
    public class Module1Impl : IModule
    {
        [Import]
        public IRegionManager TheRegionManager { private get; set; }

        public void Initialize()
        {
            TheRegionManager.RegisterViewWithRegion("MyRegion1", typeof(Module1View1));

            TheRegionManager.RegisterViewWithRegion("MyRegion1", typeof(Module1View2));
        }
    }
}
```

همان طور که مشاهده می‌کنید از ModuleExportAttribute برای شناسایی ماژول توسط MefBootstrapper استفاده کردیم و نوع آن را ModuleImpl قرار دادیم. ImportAttribute استفاده شده در این کلاس و خاصیت TheRegionManager برای این است که در هنگام ساخت Instance از این کلاس IRegionManager موجود در Container باید در اختیار این کلاس قرار گیرد(نشان دهنده وابستگی مستقیم این کلاس با IRegionManager است). روش دیگر این است که در سازنده این کلاس هم این اینترفیس را تزریق کنیم.

در متد Initialize برای RegionManager دو View ساخته شده را رجیستر کردیم. این کار باید به تعداد Viewهای موجود در ماژول انجام شود.

Shell

در پروژه اصلی یک Page به نام Shell ایجاد کنید و کدهای زیر را در آن کپی کنید.

```
<UserControl x:Class="NavigationViaViewModel.Shell"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:prism="http://www.codeplex.com/prism" FlowDirection="RightToLeft"
FontFamily="Tahoma">

    <Grid x:Name="LayoutRoot"
          Background="White">
        <TextBlock Text="لیست کتاب‌ها به همراه طبقه بندی آن ها"
                   FontSize="19"
                   Foreground="Black"
                   HorizontalAlignment="Center"
                   VerticalAlignment="Top" />
        <ContentControl HorizontalAlignment="Center"
                        VerticalAlignment="Center"
                        prism:RegionManager.RegionName="MyRegion1" />
    </Grid>
</UserControl>
```

همانند مثال قبلی یک ContentControl داریم و به وسیله RegionName که یک AttachedProperty است یک Region به نام MyRegion1 ایجاد کردیم. تمام ماژول‌های این مثال در این محدوده نمایش داده خواهند شد.

Bootstrapper

حال نیاز به یک Bootstrapper داریم. برای این کار یک کلاس به نام TheBootstrapper بسازید:

```
using Microsoft.Practices.Prism.MefExtensions;
using Microsoft.Practices.Prism.Modularity;
using System.ComponentModel.Composition.Hosting;
using System.Windows;

namespace NavigationViaViewModel
{
    public class TheBootstrapper : MefBootstrapper
    {
        protected override void InitializeShell()
        {
            base.InitializeShell();

            Application.Current.RootVisual = (UIElement)Shell;
        }

        protected override DependencyObject CreateShell()
        {
            return Container.GetExportedValue<Shell>();
        }

        protected override void ConfigureAggregateCatalog()
        {
            base.ConfigureAggregateCatalog();
            AggregateCatalog.Catalogs.Add(new AssemblyCatalog(this.GetType().Assembly));
        }

        protected override IModuleCatalog CreateModuleCatalog()
        {
            ModuleCatalog moduleCatalog = new ModuleCatalog();

            moduleCatalog.AddModule
            (
                new ModuleInfo
                {
```

```
                    InitializationMode = InitializationMode.WhenAvailable,
                    Ref = "Module1.xap",
                    ModuleName = "Module1Impl",
                    ModuleType = "Module1.Module1Impl, Module1, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=null"
                }
            );

            return moduleCatalog;
        }
    }
}
```

متد CreateShell اولین متد در این کلاس است که اجرا خواهد شد. بعد از متد CreateShell، متد InitializeShell اجرا خواهد شد. خاصیت Shell دقیقا به مقدار برگشتی متد CreateShell اشاره خواهد کرد. در متد InitializeShell مقدار خاصیت Shell به RootVisual این پروژه اشاره می‌کند(مانند MainWindow در کلاس Application پروژه‌های WPF).

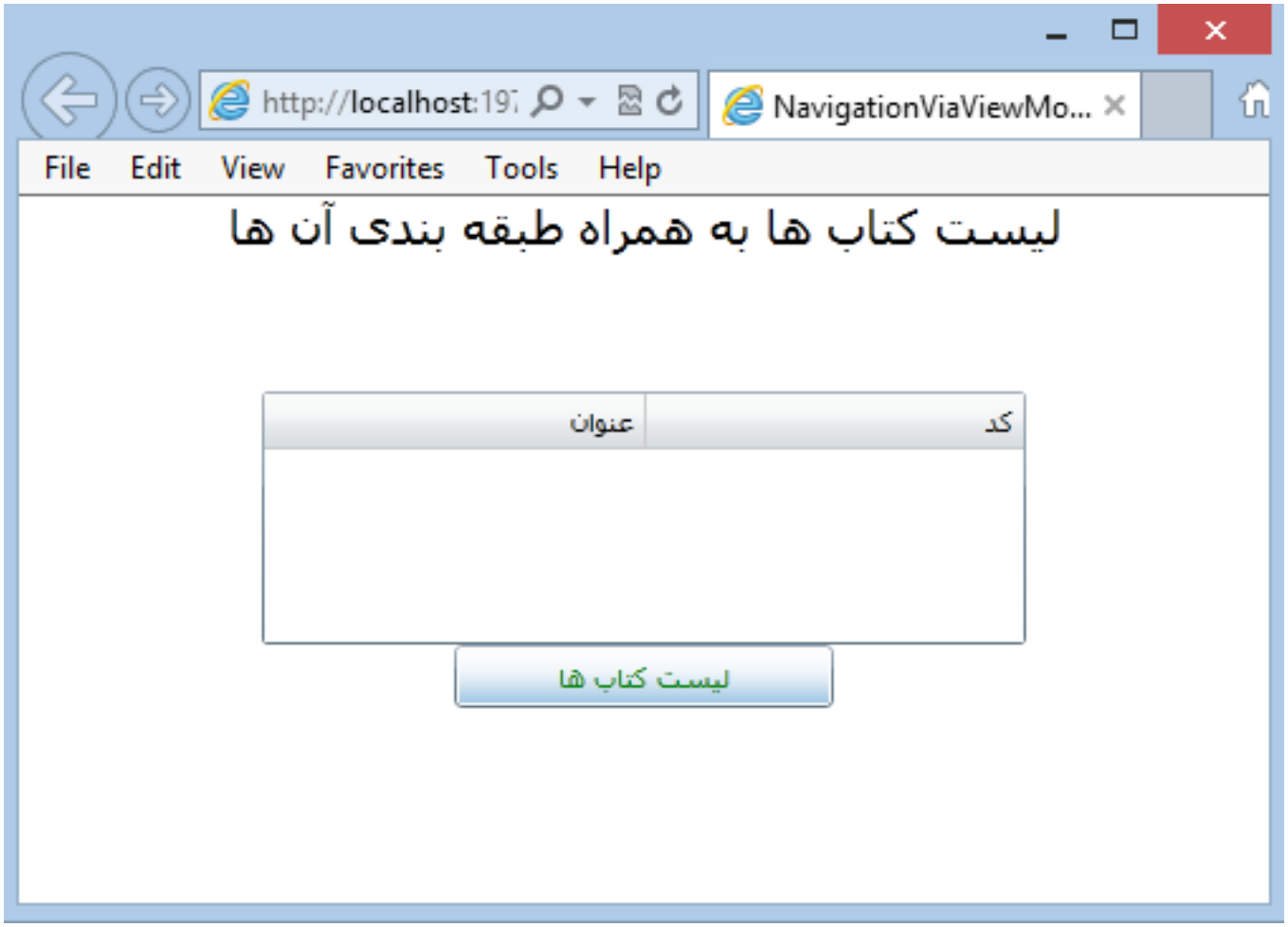
متد ConfigureAggregateCatalog برای مدیریت کاتالوگ‌ها و ماژول‌ها که هر کدام در یک اسمبلی جدا وجود خواهند شد استفاده می‌شود. در این متد من از AssemblyCatalog استفاده کردم. AssemblyCatalog تمام کلاس هایی که ExportAttribute را به همراه دارند شناسایی می‌کند و آن‌ها را در Container نگهداری خواهد کرد(^). مانند یک ServiceLocator در Microsoft unity Service Locator(^) .

متد آخر به نام CreateModuleCatalog است و باید در آن تمام ماژول‌های برنامه را به کلاس ModuleCatalog اضافه کنیم. در مثال پست قبلی به دلیل استفاده از UnityBootstrapper باید این کار را از طریق BuildEvent ها مدیریت می‌کردیم ولی در این جا Mef به راحتی این کار را انجام خواهد داد.

تغییرات زیر را در فایل App.Xaml قرار دهید و پروژه را اجرا کنید.

```
public partial class App : Application
 {
        public App()
        {
            this.Startup += this.Application_Startup;
            InitializeComponent();
        }

        private void Application_Startup(object sender, StartupEventArgs e)
        {
           var bootstrapper = new TheBootstrapper();
            bootstrapper.Run();
        }
}
```

با کلیک بر روی ماژول عملیات راهبری برای ماژول انجام خواهد شد.



دریافت سورس پروژه

ادامه دارد..

نظرات خوانندگان

نویسنده: javad
تاریخ: ۱۳۹۲/۰۵/۰۵ ۱۲:۱

سلام
اگه می‌شه آموزش استفاده از Entity Framwork در prism را نیز قرار دهید . می‌خوام ماژول‌های مختلف از یک دیتا بیس استفاده کنند و یک EF مشترک داشته باشند ؟

نویسنده: مسعود م.پاکدل
تاریخ: ۱۳۹۲/۰۵/۰۵ ۱۳:۱۰

بسیار ساده است. شما نیاز به طراحی یک UnitOfWork بر اساس EF دارید(^). بعد از آن کافیست کدهای مورد نظر برای عملیات CRUD رو در ViewModelهای هر ماژول بنویسید. در پروژه‌های Silverlight هم می‌تونید از RIA Service و EF استفاده کنید. سعی می‌کنم در صورت داشتن زمان کافی یک پست را به این مطلب اختصاص بدم.

نویسنده: imo0
تاریخ: ۱۳۹۲/۰۶/۲۰ ۱۶:۳۴

سلام . دستتون درد نکنه اقای پاکدل . فقط یه چیزی!
یکی اینکه این اموزشتونو اگه میشه یکم سریعتر بدید . اون روش قبلیه که گفتید رو من خوندم خیلی واضح‌تر توضیح داده بودین . اما از این یکی زیاد نمیتونم درکش کنم.
اگه میشه لطفا رو یه ساختار کنین . یعنی مثلا همین Prism رو با همون الگویه MVVM ای که داره تویه WPF بگین که ما هم بتونیم استفاده کنیم . شما یکی شو با یه روش، یکی دیگشو با یه روشه دیگه و باز اینارو هر کدوم یکی تو Silver و اون یکی تو WPF . این نظر منه . اگه شما یه دونشونو انتخاب کنید و همینطوری ادامه بدین بهتره که ما هم بتونیم برای خودمون یه جمع بندی و یه راه مشخص پیدا کنیم . سایت واقعا عالی دارین . خیلی چیزا من از این سایت یاد گرفتم . این ماژولار بودن تو این سبک و تا این سطح خیلی برام کاربردی و مهمه . میخوام پایه پروژه‌های شرکتو بر همین روال قرار بدم . اگه میشد شما از همین Prism و این MEF یه پروژه WPF بسازین فقط یکی دوتا ماژول ساده براش پیاده سازه کنین و یه فیلم بگیرین خیلی ممنون میشم . میخوام ا این روش استفاده کنم اما روال کار برام مبهمه . اگه کتاب یا سری آموزشی در این باره هم دارین بزارین ما استفاده کنیم . اموزش هاتونم من هر روز میام میخونم و چک میکنم اما خیلی دیر دیر مطلب میزارین . حتما این اموزشو ادامه بدین . مخصوصا Prism With MEF In WPF . خلیی باحالین....

نویسنده: imo0
تاریخ: ۱۳۹۲/۰۹/۲۵ ۱۷:۱۵

سلام . خسته نباشید . من اگه بخوام تمام ماژول‌ها به صورت داینامیک از تو یک فولدر بخونه باید چیکار کرد. داخل WPF از کلاس DirectoryCatalog استفاده میشه کرد . اما برای سیلورلایت این کلاس وجود نداره . اگه میشه راهنمایی بفرمایین .

نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۹/۲۵ ۱۷:۲۸

ابتدا اسمبلی System.ComponentModel.Composition را به پروژه خود اضافه نمایید. در فضای نام System.ComponentModel.Composition.Hosting کلاس DirectoryCatalog موجود است.

نویسنده: imo0

تاریخ: ۱۳۹۲/۰۹/۲۵ ۱۷:۴۲

با تشکر ولی به نظر سیلورلایت نداره . لطفا اینجا رو یه چک بکنید . نوشته که

"**Note:** DirectoryCatalog is not supported in Silverlight".

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۹/۲۵ ۲۲:۴۴

در همون لینکی که دادید یک پیاده سازی کمکی ذکر شده: A DirectoryCatalog class for Silverlight

DeploymentCatalog هم هست

عنوان: **انقیاد RadioButtonها در WPF به یک Enum**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۰/۲۷ ۱۲:۰
آدرس: www.dotnettips.info
گروه‌ها: MVVM, WPF

فرض کنید قصد دارید برای انتخاب بین چند گزینه‌ی محدود، از RadioButtonها بجای سایر کنترل‌های موجود استفاده کنید. این گزینه‌ها نیز توسط یک Enum تعریف شده‌اند. اکنون نیاز است گزینه‌های مختلف این Enum را به RadioButtonهای تعریف شده Bind کنیم.
تعریف Enum برنامه به صورت زیر است:

```
namespace WpfBindRadioButtonToEnum.Models
{
    public enum Gender
    {
        Female,
        Male
    }
}
```

در ادامه با توجه به اینکه RadioButtonها با خاصیت IsChecked از نوع bool کار می‌کنند، نیاز است بتوانیم گزینه‌های Enum را به bool و یا برعکس تبدیل کنیم . برای این منظور از تبدیلگر EnumBooleanConverter ذیل می‌توان استفاده کرد:

```
using System;
using System.Globalization;
using System.Windows;
using System.Windows.Data;

namespace WpfBindRadioButtonToEnum.Converters
{
    public class EnumBooleanConverter : IValueConverter
    {
        public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
        {
            if (Enum.IsDefined(value.GetType(), value) == false)
                return DependencyProperty.UnsetValue;

            return Enum.Parse(value.GetType(), parameter.ToString()).Equals(value);
        }

        public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
        {
            return Enum.Parse(targetType, parameter.ToString());
        }
    }
}
```

پیش‌فرض تبدیلگر تهیه شده بر این است که مقدار ثابت Enum را از طریق سومین پارامتر، یعنی ConverterParameter تنظیم شده در حین عملیات Binding، دریافت می‌کند. پارامتر value مقداری است که از طریق Binding خاصیت IsChecked دریافت خواهد شد.

اکنون اگر ViewModel برنامه به شکل زیر باشد که GenderValue را در اختیار View قرار می‌دهد:

```
using System.ComponentModel;
using WpfBindRadioButtonToEnum.Models;

namespace WpfBindRadioButtonToEnum.ViewModels
{
    public class MainWindowViewModel : INotifyPropertyChanged
    {
        Gender _genderValue;
        public Gender GenderValue
        {
            get { return _genderValue; }
            set
            {
                _genderValue = value;
```

```
                notifyPropertyChanged("GenderValue");
            }
        }



        #region INotifyPropertyChanged Members
        public event PropertyChangedEventHandler PropertyChanged;
        private void notifyPropertyChanged(string propertyName)
        {
            if (PropertyChanged != null)
            {
                PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
            }
        }
        #endregion
    }
}
```

View متناظری که از آن و همچنین Enum و تبدیلگر تهیه شده استفاده می‌کند، به شرح ذیل خواهد بود:

```
<Window x:Class="WpfBindRadioButtonToEnum.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:VM="clr-namespace:WpfBindRadioButtonToEnum.ViewModels"
        xmlns:C="clr-namespace:WpfBindRadioButtonToEnum.Converters"
        xmlns:Models="clr-namespace:WpfBindRadioButtonToEnum.Models"
        Title="MainWindow" Height="350" Width="525">
    <Window.Resources>
        <VM:MainWindowViewModel x:Key="VMainWindowViewModel" />
        <C:EnumBooleanConverter x:Key="CEnumBooleanConverter" />
    </Window.Resources>
    <StackPanel DataContext="{Binding Source={StaticResource VMainWindowViewModel}}">
        <TextBlock Text="Gender" Margin="3" />
        <RadioButton Content="{x:Static Models:Gender.Male}"
                     IsChecked="{Binding GenderValue, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged,
                                 Converter={StaticResource CEnumBooleanConverter},
                                 ConverterParameter={x:Static Models:Gender.Male}}"
                     Margin="3" GroupName="G1" />
        <RadioButton Content="{x:Static Models:Gender.Female}"
                     IsChecked="{Binding GenderValue, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged,
                                 Converter={StaticResource CEnumBooleanConverter},
                                 ConverterParameter={x:Static Models:Gender.Female}}"
                     Margin="3" GroupName="G1" />
    </StackPanel>
</Window>
```

در این View از یک markup extension به نام x:Static برای دسترسی به فیلدهای ثابت برنامه کمک گرفته شده‌است. از x:Static در ConverterParameter و همچنین Content می‌توان استفاده کرد. برای دسترسی به Enum تعریف شده در برنامه، فضای نام آن توسط xmlns:Models در ابتدای کار تعریف گردیده‌است.
در اینجا EnumBooleanConverter تهیه شده، کار تبدیل مقدار true و false دریافتی از IsChecked را به معادل Enum آن و برعکس، انجام می‌دهد.

به صورت خلاصه: ابتدا تبدیلگر EnumBooleanConverter باید اضافه شود. سپس به ازای هر گزینه‌ی Enum، یک RadioButton در صفحه قرار می‌گیرد که ConverterParameter خاصیت IsChecked آن مساوی است با یکی از گزینه‌های Enum متناظر.

عنوان: **بستن یک پنجره از طریق ViewModel با استفاده از خصوصیت های پیوست شده هنگام استفاده از الگوی MVVM**
نویسنده: محبوبه محمدی
تاریخ: ۱۳۹۲/۱۱/۰۲ ۱۴:۲۰
آدرس: www.dotnettips.info
گروه‌ها: MVVM, WPF

در نظر بگیرید که یک پروژه WPF را با الگوی MVVM پیاده سازی کرده اید و نیاز پیدا می‌کنید تا یک پنجره را از طریق کد ببندید. از آنجایی که به کنترل Window درون ViewModel دسترسی ندارید، نمی‌توانید از متد Close آن برای اینکار استفاده کنید. راه‌های مختلفی برای اینکار وجود دارند، مثلا اگر از MVVM Light Toolkit استفاده می‌کنید با ارسال یک Message و نوشتن یک تکه کد در CodeBehind پنجره می‌توانید اینکار را انجام بدهید.
اما برای اینکار یک راه حل ساده‌تری بدون نیاز به نوشتن کد در CodeBehind و استفاده از Toolkit خاصی وجود دارد و آن استفاده ازخاصیت‌های پیوست شده یا *Attached Properties* است. برای اینکار یک خاصیت از نوع Boolean مانند زیر تعریف می‌کنیم و آن را به پنجره ای که می‌خواهیم Colse شود پیوست می‌کنیم.

```
namespace TestProject.XamlServices
{
    public class CloseBehavior
    {
        public static readonly DependencyProperty CloseProperty =
DependencyProperty.RegisterAttached("Close", typeof(bool), typeof(CloseBehavior), new
UIPropertyMetadata(false, OnClose));

        private static void OnClose(DependencyObject sender, DependencyPropertyChangedEventArgs e)
        {
            if (!(e.NewValue is bool) || !((bool) e.NewValue)) return;
            var win = GetWindow(sender);
            if (win != null)
                win.Close();
        }

        private static Window GetWindow(DependencyObject sender)
        {
            Window w = null;
            if (sender is Window)
                w = (Window)sender;
            return w ?? (w = Window.GetWindow(sender));
        }

        public static bool GetClose(Window target)
        {
            return (bool)target.GetValue(CloseProperty);
        }

        public static void SetClose(DependencyObject target, bool value)
        {
            target.SetValue(CloseProperty, value);
        }
    }
}
```

در تکه کد بالا یک خصوصیت از نوع Boolean ایجاد کردیم که می‌تواند به هر پنجره ای که قرار است از طریق کد بسته شود، پیوست شود. خصوصیت‌های پیوست شده یک Callback مربوط به تغییر مقدار دارند که یک متداستاتیک است و مقدار جدید، از طریق EventArg و شیءایی که این خاصیت به آن پیوست شده نیز بعنوان Source به آن ارسال می‌شود. هر وقت مقدار خصوصیت، تغییر کند این متد فراخوانی می‌گردد. در کد بالا متد OnClose ایجاد شده است و زمانی که مقدار این خصوصیت برابر true می‌شود پنجره close خواهد شد. برای استفاده از این خصوصیت و اتصال آن باید یک خصوصیت از نوع Boolean نیز در ViewModel مربوط به Window ایجاد کنید:

```
private bool _isClose;
 public bool IsClose
   {
      get { return _isClose; }
      set
      {
       _isClose = value;
      OnClosed();
```

```
      RaisePropertyChanged("IsClose");
      }
}
```

و آن را به صورت زیر Bind کنید:

```
<Window x:Class="TestProject.TestView"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:xamlServices="clr-namespace:TestProject.XamlServices;assembly=TestProject.XamlServices"
       xamlServices:CloseBehavior.Close="{Binding IsClose}">
        ...
</Window>
```

پس از انجام اتصالات فوق، کافیست هر جایی از ViewModel که نیاز است پنجره بسته شود،مقدار این خصوصیت برابر False بشود.

نظرات خوانندگان

نویسنده: نفیسه الف
تاریخ: ۱۳۹۳/۰۳/۳۱ ۲:۷

سلام
وقتی مقدار تغییر میکنه propertychangedcallback اجرا نمیشه
onclosed() که تو set پراپرتی isclose هست از کجا اومده؟
ممنون از مطلب مفیدتون

عنوان: **حرکت روی سلول های دیتا گرید با فشردن کلید Enter در برنامه های WPF**
نویسنده: محبوبه محمدی
تاریخ: ۱۳۹۲/۱۱/۲۳ ۱۳:۳۵
آدرس: www.dotnettips.info
گروه‌ها: MVVM, WPF, XAML

کاربران بیشتر برنامه‌های فارسی تمایل دارند که توسط کلیدEnter درون فرم‌ها حرکت کنند.در برنامه‌های WPF و مخصوصا زمانی که شما از الگوی MVVM استفاده می‌کنید ،انجام این کار اگر از روش‌های مناسب استفاده نکنید تا حدودی سخت می‌شود.
برای حرکت روی TextBoxها و کنترل‌های مشابه می‌توانید این کار را به راحتی با Register کردن رویداد مربوط به آن نوع کنترل‌ها توسط EventManager یک بار در ابتدای برنامه انجام دهید.

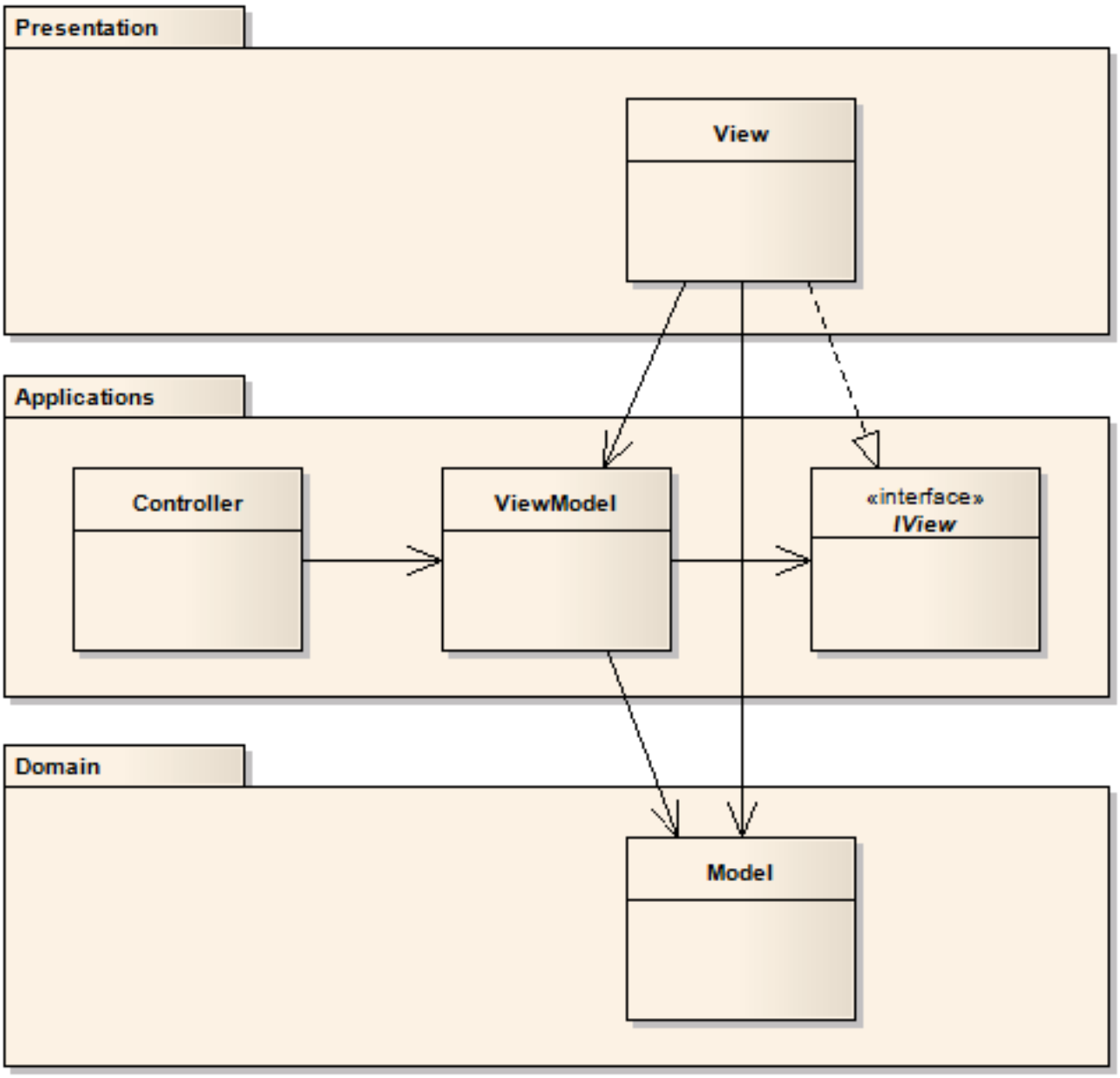
```
public partial class App : Application
{
EventManager.RegisterClassHandler(typeof(TextBox), TextBox.KeyDownEvent, new
KeyEventHandler(TextBox_KeyDown));
     ...
}
 private void TextBox_KeyDown(object sender, KeyEventArgs e)
   {
         if (e.Key != Key.Enter)
             return;
         var focusedElement = Keyboard.FocusedElement as TextBox;
          focusedElement.MoveFocus(new TraversalRequest(FocusNavigationDirection .Next));
   }
```

اما همانطور که در عنوان مطلب آورده شده است در این مطلب تصمیم دارم حرکت روی سلول‌های دیتا گرید توسط کلید Enter را شرح بدهم.
برای این کار نیز یک راه حل ساده وجود دارد و آن شبیه سازی فراخوانی کلید Tab هنگام فشرده شدن کلید Enter است.چون همانطور که می‌دانید کلید Tab به صورت پیش فرض حرکت روی سلول‌ها را انجام می‌دهد.برای انجام آن کافی ست دیتاگرید خود را سفارشی کرده و در متد OnPreviewKeyDown عملیات زیر را انجام دهید:

```
public class CommonDataGrid : DataGrid
{
        protected override void OnPreviewKeyDown(KeyEventArgs e)
        {
            base.OnPreviewKeyDown(e);
            if (e.Key != Key.Enter || Keyboard.PrimaryDevice.ActiveSource == null) return;
            this.CommitEdit();
            var args = new KeyEventArgs
                (Keyboard.PrimaryDevice,
                Keyboard.PrimaryDevice.ActiveSource, 0, Key.Tab) { RoutedEvent = Keyboard.KeyDownEvent
};
            InputManager.Current.ProcessInput(args);
        }
}
```

عنوان: **آموزش WAF**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۳/۰۱/۱۳ ۹:۲۵
آدرس: www.dotnettips.info
گروه‌ها: MVVM, WPF, WAF

در طراحی پروژه‌های مقیاس بزرگ و البته به صورت ماژولار همیشه ساختار پروژه اهمیت به سزایی دارد. متاسفانه این مورد خیلی در طراحی پروژه‌ها در نظر گرفته نمی‌شود و اغلب اوقات شاهد آن هستیم که یک پروژه بسیار بزرگ دقیقا به همان صورت پروژهای کوچک و کم اهمیت‌تر مدیریت و پیاده سازی می‌شود که این مورد هم مربوط به پروژه‌های تحت وب و هم پروژه‌های تحت ویندوز و WPF است. برای مدیریت پروژه‌های WPF و Silverlight در این پست به اختصار درباره PRISM بحث شد. مزایا و معایب آن بررسی و در طی این پست ها(^ و ^) مثال هایی را پیاده سازی کردیم. اما در این پست مفتخرم شما را با یکی دیگر از کتابخانه‌های مربوط به پیاده سازی مدل MVVM آشنا کنم. کتابخانه ای متن باز، بسیار سبک با کارایی بالا.
اما نکته ای که ذکر آن خالی از لطف نیست این است که قبلا از این کتابخانه در یک پروژه بزرگ و ماژولار WPF استفاده کردم و نتیجه مطلوب نیز حاصل شد.
معرفی:
WPF Application Framework یا به اختصار WAF کتابخانه کم حجم سبک و البته با کارایی عالی برای طراحی پروژه‌های ماژولار WPF در مقیاس بزرگ طراحی شده است که مدل پیاده سازی ان بر مبنای مدل MVVM و MVC است. شاید برایتان جالب باشد که این کتابخانه دقیقا مدل MVC را با مدل MVVM ترکیب کرده در نتیجه مفاهیم آن بسیار شبیه به پروژه‌های تحت وب MVC است. همانطور که از نام آن پیداست این کتابخانه صرفا برای پروژه‌های WPF طراحی شده، در نتیجه در پروژه‌های Silverlight نمی‌توان از آن استفاده کرد.
ساختار کلی آن به شکل زیر می‌باشد:



همانطور که مشاهده می‌کنید پروژه‌های مبتنی بر این کتابخانه همانند سایر کتابخانه‌های MVVM از سه بخش تشکیل شده اند. بخش اول با عنوان Shell یا Presentation معرف فایل‌های Xaml پروژه است، بخش دوم یا Application معرف ViewModel و Controller و البته IView می‌باشد. بخش Domain نیز در برگیرنده مدل‌های برنامه است.

معرفی برخی مفاهیم:

»Shell : این کلاس معادل یک فایل Xaml است که حتما باید یک اینترفیس IView را پیاده سازی نماید.

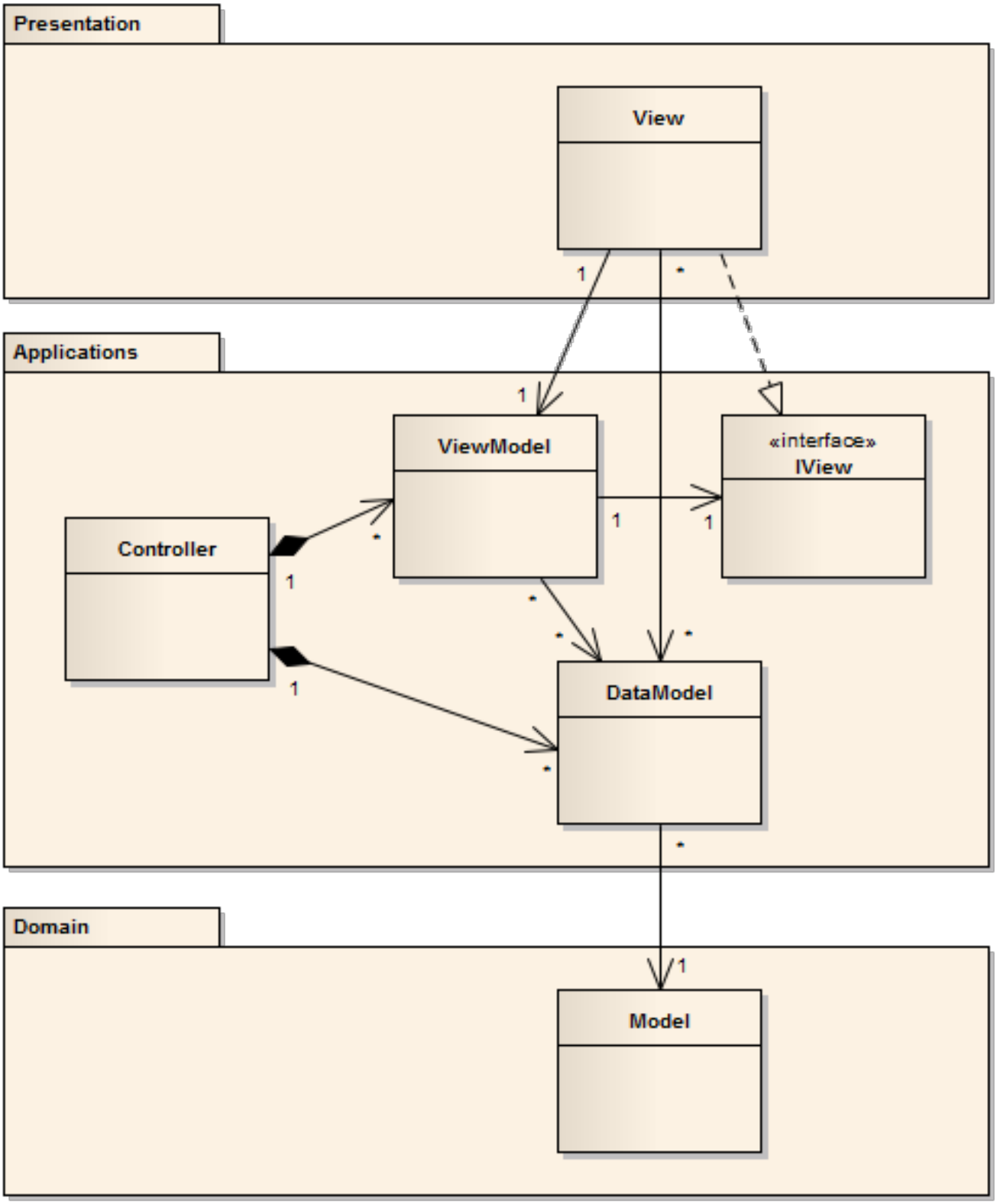
»IView : معرف یک اینترفیس جهت برقراری ارتباط بین ViewModel و Shell

»ViewModel : در این جا ViewModel با مفهوم ViewModel در سایر کتابخانه‌های MVVM کمی متفاوت است. در این کتابخانه ViewModel فقط شامل تعاریف است و هیچ گونه پیاده سازی در اینجا صورت نمی‌گیرد. دقیقا معادل مفهوم ViewModel در پروژه‌های MVC تحت وب.

»Controller : پیاده سازی ViewModel و تعریف رفتارها در این قسمت انجام می‌گیرد.

اما در بسیاری از پروژها نیاز به پیاده سازی الگوی DataModel-View-ViewModel است که این کتابخانه با دراختیار داشتن برخی

کلاس‌های پایه این مهم را برایمان میسر کرده است.



همانطور که می‌بینید در این حالت بر خلاف حالت قبلی ViewModel و کنترلرهای پروژه به جای ارتباط با مدل با مفهوم DataModel تغذیه می‌شوند که یک پیاده سازی سفارشی از مدل‌های پروژه است. هم چنین این کتابخانه یک سری Converterهای سفارشی

جهت تبدیل Model به DataModel و برعکس را ارائه می‌دهد.
سرویس‌های پیش فرض: که شامل DialogBox جهت نمایش پیغام‌ها و Save|Open File Dialog سفارشی نیز می‌باشد.
»برای پیاده سازی Modularity از کتابخانه MEF استفاده شده است.
Commandهای سفارشی: پیاده سازی خاص از اینترفیس ICommand
»مفاهیم مربوط به Weak Event Pattern به صورت توکار در این کتابخانه تعبیه شده است.
»به صورت پیش فرض مباحث مربوط به اعتبارسنجی با استفاده از DataAnnotation و IDataErrorInfo در این کتابخانه تعبیه شده است.
»ارائه Extensionهای مربوط به UnitTest نظیر Exceptions و CanExecuteChangedEvent و PopertyChanged جهت سهولت در تهیه unit test

دانلود و نصب
با استفاده از nuget و دستور زیر می‌توانید این کتابخانه را نصب نمایید:

```
Install-Package waf
```

هم چنین می‌توانید سورس آن به همراه فایل‌های باینری را از اینجا دریافت کنید. در پست بعدی یک نمونه از پیاده سازی مثال با این کتابخانه را بررسی خواهیم کرد.

عنوان: **آموزش WAF (بررسی ساختار همراه با پیاده سازی یک مثال)**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۳/۰۱/۱۳ ۱۶:۳۵
آدرس: www.dotnettips.info
گروه‌ها: MVVM, WPF, WAF

در این پست با مفاهیم اولیه این کتابخانه آشنا شدید. برای بررسی و پیاده سازی مثال، ابتدا یک Blank Solution را ایجاد نمایید. فرض کنید قصد پیاده سازی یک پروژه بزرگ ماژولار را داریم. برای این کار لازم است مراحل زیر را برای طراحی ساختار مناسب پروژه دنبال نمایید.

نکته: آشنایی اولیه با مفاهیم MEF از ملزومات این بخش است.

»ابتدا یک Class Library به نام Views ایجاد نمایید و اینترفیس زیر را به صورت زیر در آن تعریف نمایید. این اینترفیس رابط بین کنترلر و View از طریق ViewModel خواهد بود.

```
public interface IBookView : IView
    {
        void Show();
        void Close();
    }
```

اینترفیس IView در مسیر System.Waf.Applications قرار دارد. در نتیجه از طریق nuget اقدام به نصب Package زیر نمایید:

```
Install-Package WAF
```

»حال در Solution ساخته شده یک پروژه از نوع WPF Application به نام Shell ایجاد کنید. با استفاده از نیوگت، Waf Package را نصب نمایید؛ سپس ارجاعی از اسمبلی Views را به آن ایجاد کنید. output type اسمبلی Shell را به نوع ClassLibrary تغییر داده، همچنین فایل‌های موجود در آن را حذف نمایید. یک فایل Xaml جدید را به نام BookShell ایجاد نمایید و کدهای زیر را در آن کپی نمایید:

```
<Window x:Class="Shell.BookShell"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Book View" Height="350" Width="525">
    <Grid>
        <DataGrid ItemsSource="{Binding Books}" HorizontalAlignment="Left" Margin="10,10,0,0"
VerticalAlignment="Top" Width="400" Height="200">
            <DataGrid.Columns>
                <DataGridTextColumn Header="Code" Binding="{Binding Code}"
Width="100"></DataGridTextColumn>
                <DataGridTextColumn Header="Title" Binding="{Binding Title}"
Width="300"></DataGridTextColumn>
            </DataGrid.Columns>
        </DataGrid>

    </Grid>
</Window>
```

این فرم فقط شامل یک دیتاگرید برای نمایش اطلاعات کتاب‌هاست. دیتای آن از طریق ViewModel تامین خواهد شد، در نتیجه ItemsSource آن به خاصیتی به نام Books بایند شده است. حال ارجاعی به اسمبلی System.ComponentModel.Composition دهید. سپس در Code behind این فرم کدهای زیر را کپی کنید:

```
[Export(typeof(IBookView))]
    [PartCreationPolicy(CreationPolicy.NonShared)]
    public partial class BookShell : Window, IBookView
    {
        public BookShell()
        {
            InitializeComponent();
        }
    }
```

کاملا واضح است که این فرم اینترفیس IBookView را پیاده سازی کرده است. از آنجاکه کلاس Window به صورت پیش فرض دارای متدهای Show و Close است در نتیجه نیازی به پیاده سازی مجدد متدهای IBookView نیست. دستور Export باعث می‌شود که این کلاس به عنوان وابستگی به Composition Container اضافه شود تا در جای مناسب بتوان از آن وهله سازی کرد. نکته‌ی مهم این است که به دلیل آنکه این کلاس، اینترفیس IBookView را پیاده سازی کرده است در نتیجه نوع Export این کلاس حتما باید به صورت صریح از نوع IBookView باشد.

»یک Class Library به نام Models بسازید و بعد از ایجاد آن، کلاس زیر را به عنوان مدل Book در آن کپی کنید:

```
public class Book
    {
        public int Code { get; set; }

        public string Title { get; set; }
    }
```

»یک Class Library دیگر به نام ViewModels ایجاد کنید و همانند مراحل قبلی، Package مربوط به WAF را نصب کنید. سپس کلاسی به نام BookViewModel ایجاد نمایید و کدهای زیر را در آن کپی کنید (ارجاع به اسمبلی‌های Views و Models را فراموش نکنید):

```
[Export]
    [Export(typeof(ViewModel<IBookView>))]
    public class BookViewModel : ViewModel<IBookView>
    {
        [ImportingConstructor]
        public BookViewModel(IBookView view)
            : base(view)
        {
        }

        public ObservableCollection<Book> Books { get; set; }

    }
```

ViewModel مورد نظر از کلاس ViewModel of T ارث برده است. نوع این کلاس معادل نوع View مورد نظر ماست که در اینجا مقصود IBookView است. این کلاس شامل خاصیتی به نام ViewCore است که امکان فراخوانی متدها و خاصیت‌های View را فراهم می‌نماید. وظیفه اصلی کلاس پایه ViewModel، وهله سازی از View سپس ست کردن خاصیت DataContext در View مورد نظر به نمونه وهله سازی شده از ViewModel است. در نتیجه عملیات مقید سازی در Shell به درستی انجام خواهدشد.
به دلیل اینکه سازنده پیش فرض در این کلاس وجود ندارد حتما باید از ImportingConstructor استفاده نماییم تا CompositionContainer در هنگام عملیات وهله سازی Exception صادر نکند.

»بخش بعدی ساخت یک Class Library دیگر به نام Controllers است. در این Library نیز بعد از ارجاع به اسمبلی‌های زیر کتابخانه WAF را نصب نمایید.

Views

Models

ViewModels

System.ComponentModel.Composition

کلاسی به نام BookController بسازید و کدهای زیر را در آن کپی نمایید:

```
[Export]
    public class BookController
    {
        [ImportingConstructor]
        public BookController(BookViewModel viewModel)
        {
            ViewModelCore = viewModel;
        }

        public BookViewModel ViewModelCore
        {
```

```
            get;
            private set;
        }

        public void Run()
        {
            var result = new List<Book>();
            result.Add(new Book { Code = 1, Title = "Book1" });
            result.Add(new Book { Code = 2, Title = "Book2" });
            result.Add(new Book { Code = 3, Title = "Book3" });

            ViewModelCore.Books = new ObservableCollection<Models.Book>(result);

            (ViewModelCore.View as IBookView).Show();
        }
    }
```

نکته مهم این کلاس این است که BookViewModel به عنوان وابستگی این کنترلر تعریف شده است. در نتیجه در هنگام وهله سازی از این کنترلر Container مورد نظر یک وهله از BookViewModel را در اختیار آن قرار خواهد داد. در متد Run نیز ابتدا مقدار Book که به ItemsSource دیتا گرید در BookShell مقید شده است مقدار خواهد گرفت. سپس با فراخوانی متد Show از اینترفیس IBookView، متد Show در BookShell فراخوانی خواهد شد که نتیجه آن نمایش فرم مورد نظر است.

طراحی Bootstrapper

در پروژه‌های ماژولار Bootstrapper از ملزومات جدانشدنی این گونه پروژه هاست. برای این کار ابتدا یک WPF Application دیگر به نام Bootstrapper ایجاد نماید. سپس ارجاعی به اسمبلی‌های زیر را در آن قرار دهید:

»Controllers

»Views

»ViewModels

»Shell

»System.ComponentModel.Composition

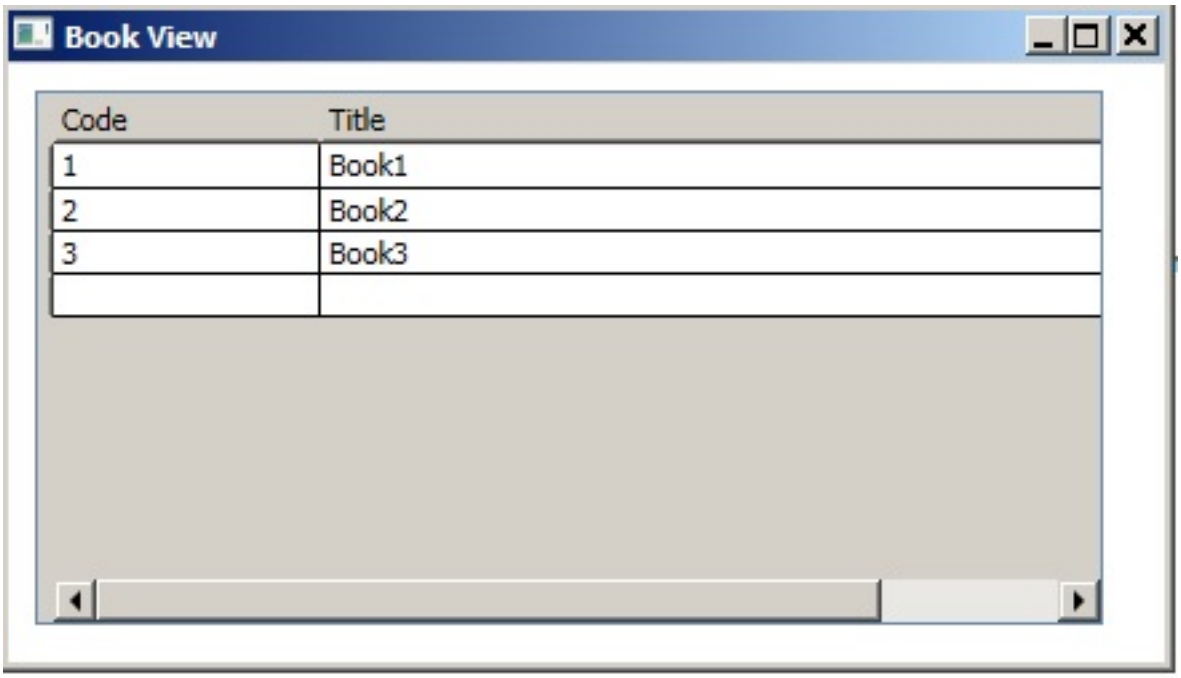
»نصب بسته WAF با استفاده از nuget

حال یک کلاس به نام AppBootstrapper ایجاد نمایید و کدهای زیر را در آن کپی نمایید:

```
public class AppBootstrapper
    {
        public CompositionContainer Container
        {
            get;
            private set;
        }

        public AggregateCatalog Catalog
        {
            get;
            private set;
        }

        public void Run()
        {
            Catalog = new AggregateCatalog();
            Catalog.Catalogs.Add(new AssemblyCatalog(Assembly.GetExecutingAssembly()));

            Catalog.Catalogs.Add(new AssemblyCatalog(String.Format("{0}\\{1}",
Environment.CurrentDirectory, "Shell.dll")));
            Catalog.Catalogs.Add(new AssemblyCatalog(String.Format("{0}\\{1}",
Environment.CurrentDirectory, "ViewModels.dll")));
            Catalog.Catalogs.Add(new AssemblyCatalog(String.Format("{0}\\{1}",
Environment.CurrentDirectory, "Controllers.dll")));

            Container = new CompositionContainer(Catalog);

            var batch = new CompositionBatch();
            batch.AddExportedValue(Container);
            Container.Compose(batch);

            var bookController = Container.GetExportedValue<BookController>();
```

```
            bookController.Run();

        }
    }
```

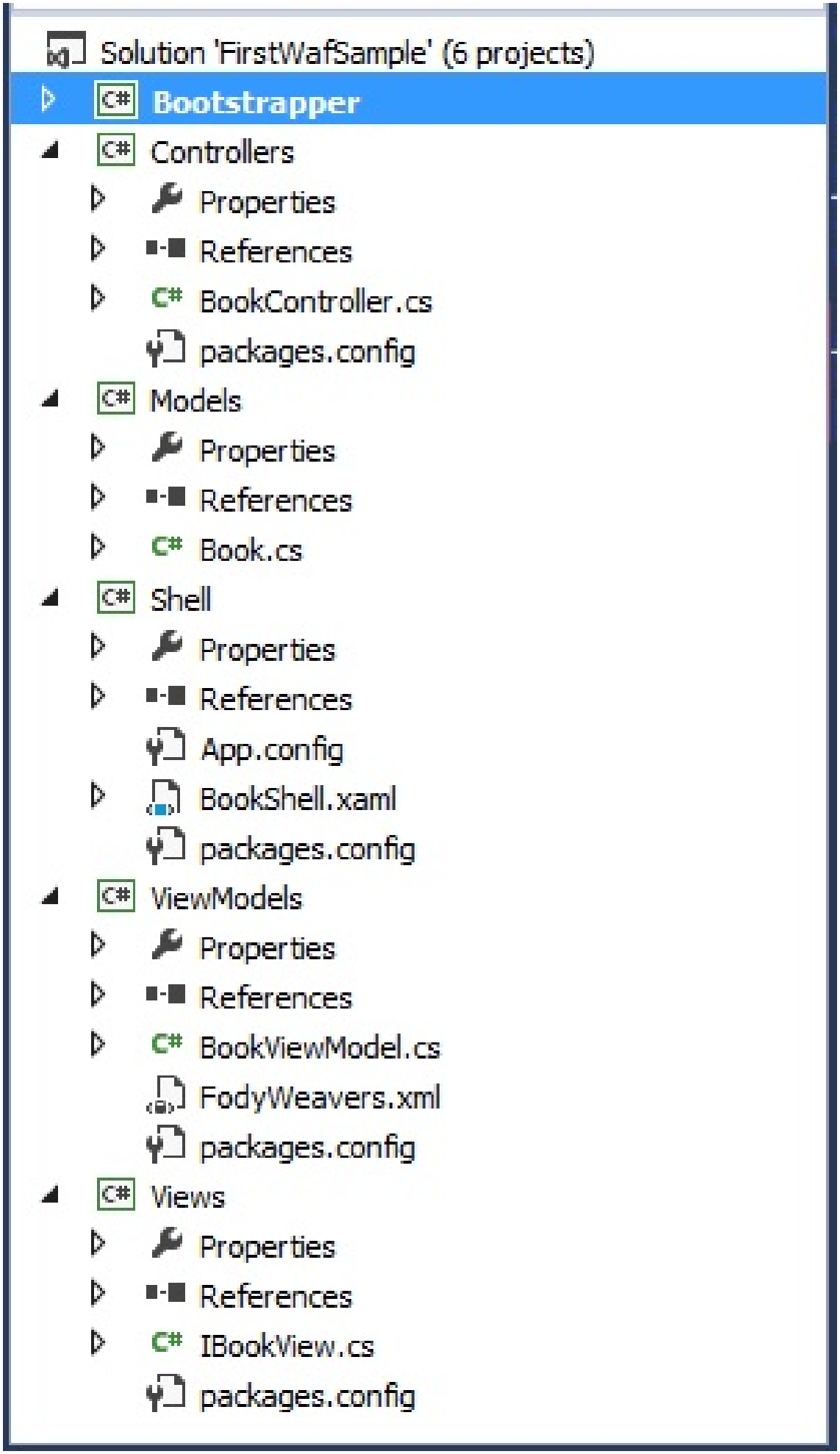
اگر با MEF آشنا باشید کدهای بالا نیز برای شما مفهوم مشخصی دارند. در متد Run این کلاس ابتدا Catalog ساخته می‌شود. سپس با اسکن اسمبلی‌های مورد نظر تمام Exportها و Importهای لازم واکشی شده و به Conrtainer مورد نظر رجیستر می‌شوند. در انتها نیز با وهله سازی از BookController و فراخوانی متد Run آن خروجی زیر نمایان خواهد شد.



نکته بخش Startup را از فایل App.Xaml خذف نمایید و در متد Startup این فایل کد زیر را کپی کنید:

```
public partial class App : Application
    {
        protected override void OnStartup(StartupEventArgs e)
        {
            new Bootstrapper.AppBootstrapper().Run();
        }
    }
```

در پایان، ساختار پروژه به صورت زیر خواهد شد:


Solution 'FirstWafSample' (6 projects)
Bootstrapper
Controllers
Properties
References
BookController.cs
packages.config
Models
Properties
References
Book.cs
Shell
Properties
References
App.config
BookShell.xaml
packages.config
ViewModels
Properties
References
BookViewModel.cs
FodyWeavers.xml
packages.config
Views
Properties
References
IBookView.cs
packages.config

نکته: می‌توان بخش اسکن اسمبلی‌ها را توسط یک DirecotryCatalog به صورت زیر خلاصه کرد:

```
Catalog.Catalogs.Add(new DirectoryCatalog(Environment.CurrentDirectory));
```

در این صورت تمام اسمبلی‌های موجود در این مسیر اسکن خواهند شد.
نکته: می‌توان به جای جداسازی فیزیکی لایه‌ها آن‌ها را از طریق Directoryها به صورت منطقی در قالب یک اسمبلی نیز مدیریت کرد.
نکته: بهتر است به جای رفرنس مستقیم اسمبلی‌ها به Bootstrapper با استفاده از Pre post build در قسمت Build Event، اسمبلی‌های مورد نظر را در یک مسیر Build کپی نمایید که روش آن به تفصیل در این پست و این پست شرح داده شده است.
دانلود سورس پروژه

عنوان: **آموزش WAF (بررسی Commandها)**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۳/۰۱/۱۶ ۲۲:۵۰
آدرس: www.dotnettips.info
گروه‌ها: MVVM, WPF, WAF

در این پست قصد داریم مثال قسمت قبل را توسعه داده و پیاده سازی Commandها را در آن در طی یک مثال بررسی کنیم. از این جهت دکمه‌ای، جهت حذف آیتم انتخاب شده در دیتا گرید، به فرم BookShell اضافه می‌نماییم. به صورت زیر:

```
<Button Content="RemoveItem" Command="{Binding RemoveItemCommand}" HorizontalAlignment="Left"
VerticalAlignment="Top" Width="75"/>
```

Command تعریف شده در Button مورد نظر به خاصیتی به نام RemoveItemCommand در BookViewModel که نوع آن ICommand است اشاره می‌کند. پس باید تغییرات زیر را در ViewModel اعمال کنیم:

```
public ICommand RemoveItemCommand { get; set; }
```

از طرفی نیاز به خاصیتی داریم که به آیتم جاری در دیتاگرید اشاره کند.

```
public Book CurrentItem
        {
            get
            {
                return currentItem;
            }
            set
            {
                if(currentItem != value)
                {
                    currentItem = value;
                    RaisePropertyChanged("CurrentItem");
                }
            }
        }
        private Book currentItem;
```

همان طور که در پست قبلی توضیح داده شد پیاده سازی‌ها تعاریف ViewModel در Controller انجام می‌گیرد برای همین منظور باید تعریف DelegateCommand که یک پیاده سازی خاص از ICommand است در کنترلر انجام شود. :

```
[Export]
    public class BookController
    {
        [ImportingConstructor]
        public BookController(BookViewModel viewModel)
        {
            ViewModelCore = viewModel;
        }

        public BookViewModel ViewModelCore
        {
            get;
            private set;
        }

        public DelegateCommand RemoveItemCommand
        {
            get;
            private set;
        }

        private void ExecuteRemoveItemCommand()
        {
            ViewModelCore.Books.Remove(ViewModelCore.CurrentItem);
        }

        private void Initialize()
        {
            RemoveItemCommand = new DelegateCommand(ExecuteRemoveItemCommand);
            ViewModelCore.RemoveItemCommand = RemoveItemCommand;
```

```
        }

        public void Run()
        {

            var result = new List<Book>();
            result.Add(new Book { Code = 1, Title = "Book1" });
            result.Add(new Book { Code = 2, Title = "Book2" });
            result.Add(new Book { Code = 3, Title = "Book3" });

            Initialize();

            ViewModelCore.Books = new ObservableCollection<Models.Book>(result);

            (ViewModelCore.View as IBookView).Show();
        }
    }
```

تغییرات:

»خاصیتی به نام RemoveItemCommand که از نوع DelegateCommand است تعریف شده است؛

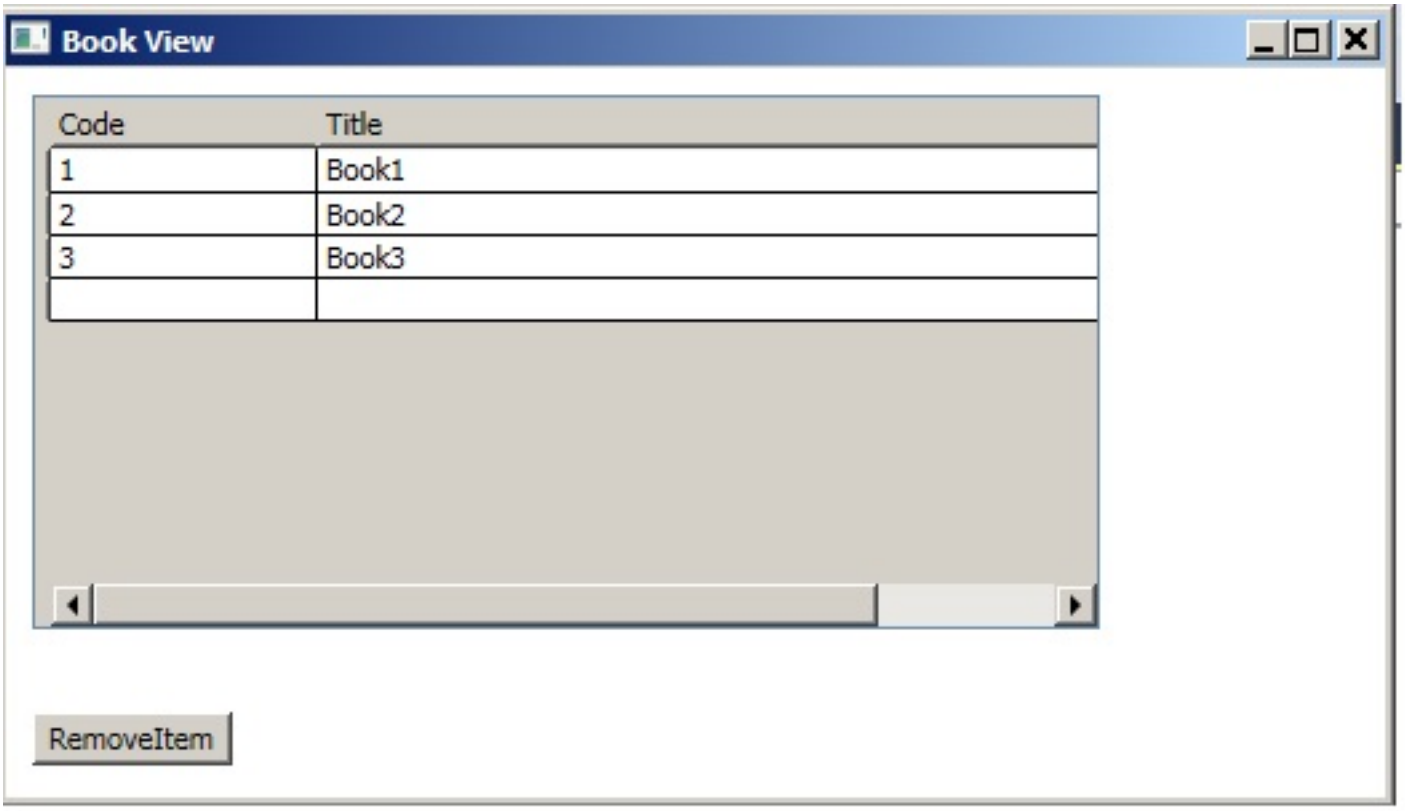
»متدی به نام Initialize اضافه شد که متدهای Execute و CanExecute برای Commandها را در این قسمت رجیستر می‌کنیم.

»در نهایت Command تعریف شده در کنترلر به Command مربوطه در ViewModel انتساب داده شد.

حال کافیست خاصیت SelectedItem دیتاگرید BookShell به خاصیت CurrentItem موجود در ViewModel مقید شود:

```
<DataGrid ItemsSource="{Binding Books}" SelectedItem="{Binding CurrentItem
,Mode=TwoWay,UpdateSourceTrigger=PropertyChanged}" HorizontalAlignment="Left" Margin="10,10,0,0"
VerticalAlignment="Top" Width="400" Height="200">
            <DataGrid.Columns>
                <DataGridTextColumn Header="Code" Binding="{Binding Code}"
Width="100"></DataGridTextColumn>
                <DataGridTextColumn Header="Title" Binding="{Binding Title}"
Width="300"></DataGridTextColumn>
            </DataGrid.Columns>
        </DataGrid>
```

اگر پروژه را اجرا نمایید، بعد از انتخاب سطر مورد نظر و کلیک بر روی دکمه RemoveItem مورد زیر قابل مشاهده است:




```
1 reference
private void ExecuteRemoveItemCommand()
{
    ViewModelCore.Books.Remove(ViewModelCore.CurrentItem);
}
```

عنوان: آموزش WAF (مشاهده تغییرات خواص ViewModel در Controller)
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۳/۰۱/۱۷ ۹:۳۵
آدرس: www.dotnettips.info
گروه‌ها: MVVM, WPF, WAF

قصد داریم در مثال پست قبلی برای Command مورد نظر، عملیات اعتبارسنجی را فعال کنیم. اگر با الگوی MVVM آشنایی داشته باشید می‌دانید که می‌توان برای Commandها اکشنی به عنوان CanExecute تعریف کرد و در آن عملیات اعتبارسنجی را انجام داد. اما از آن جا که پیاده سازی این روش زمانی مسیر است که تغییرات خواص ViewModel در دسترس باشد در نتیجه در WAF مکانیزمی جهت ردیابی تغییرات خواص ViewModel در کنترلر فراهم شده است. در نسخه‌های قبلی WAF (قبل از نسخه 3) هر کنترلر از کلاس پایه ای به نام Controller ارث می‌برد که متد هایی جهت ردیابی تغییرات در آن در نظر گرفته شده بود به صورت زیر:

```
public class MyController : Controller
    {
        [ImportingConstructor]
        public MyController(MyViewModel viewModel)
        {
            ViewModelCore = viewModel;
        }

        public MyViewModel ViewModelCore
        {
            get;
            private set;
        }

        public void Run()
        {
            AddWeakEventListener(ViewModelCore , ViewModelCoreChanged)
        }

        private void ViewModelCoreChanged(object sender , PropertyChangedEventArgs e)
        {
            if(e.PropertyName=="CurrentItem")
            {

            }
        }
    }
```

همان طور که مشاهده می‌کنید با استفاده از متد AddWeakEventListener توانستیم تمامی تغییرات خواص ViewModel مورد نظر را از طریق متد ViewModelCoreChanged ردیابی کنیم. این متد بر مبنای الگوی WeakEvent پیاده سازی شده است. البته این تغییرات فقط زمانی قابل ردیابی هستند که در ViewModel متد RaisePropertyChanged برای متد set خاصیت فراخوانی شده باشد.
از آنجا که در دات نت 4.5 یک پیاده سازی خاص از الگوی WeakEvent در کلاس PropertyChangedEventManager موجود در اسمبلی WindowsBase و فضای نام System.ComponentModel انجام شده است در نتیجه توسعه دهندگان این کتابخانه نیز تصمیم به استفاده از این روش گرفتند که نتیجه آن Obsolete شدن کلاس پایه کنترلر در نسخه‌های 3 به بعد آن است. در روش جدید کافیست به صورت زیر عمل نمایید:

```
[Export]
    public class BookController
    {
        [ImportingConstructor]
        public BookController(BookViewModel viewModel)
        {
            ViewModelCore = viewModel;
        }

        public BookViewModel ViewModelCore
        {
            get;
            private set;
        }

        public DelegateCommand RemoveItemCommand
```

```
        {
            get;
            private set;
        }

        private void ExecuteRemoveItemCommand()
        {
            ViewModelCore.Books.Remove(ViewModelCore.CurrentItem);
        }

        private bool CanExecuteRemoveItemCommand()
        {
            return ViewModelCore.CurrentItem != null;
        }
        private void Initialize()
        {
            RemoveItemCommand = new DelegateCommand(ExecuteRemoveItemCommand ,
CanExecuteRemoveItemCommand);
            ViewModelCore.RemoveItemCommand = RemoveItemCommand;
        }

        public void Run()
        {
            var result = new List<Book>();
            result.Add(new Book { Code = 1, Title = "Book1" });
            result.Add(new Book { Code = 2, Title = "Book2" });
            result.Add(new Book { Code = 3, Title = "Book3" });

            Initialize();
            ViewModelCore.Books = new ObservableCollection<Models.Book>(result);

            PropertyChangedEventManager.AddHandler(ViewModelCore, ViewModelChanged, "CurrentItem");

            (ViewModelCore.View as IBookView).Show();
        }

        private void ViewModelChanged(object sender,PropertyChangedEventArgs e)
        {
            if(e.PropertyName == "CurrentItem")
            {
                RemoveItemCommand.RaiseCanExecuteChanged();
            }
        }
    }
```

تغییرات:

»ابتدا متدی به نام CanExecuteRemoveItemCommand ایجاد کردیم و کدهای اعتبارسنجی را در آن قرار دادیم؛

»هنگام تعریف Command مربوطه متد بالا را به DelegateCommand رجیستر کردیم:

```
RemoveItemCommand = new DelegateCommand(ExecuteRemoveItemCommand , CanExecuteRemoveItemCommand);
```

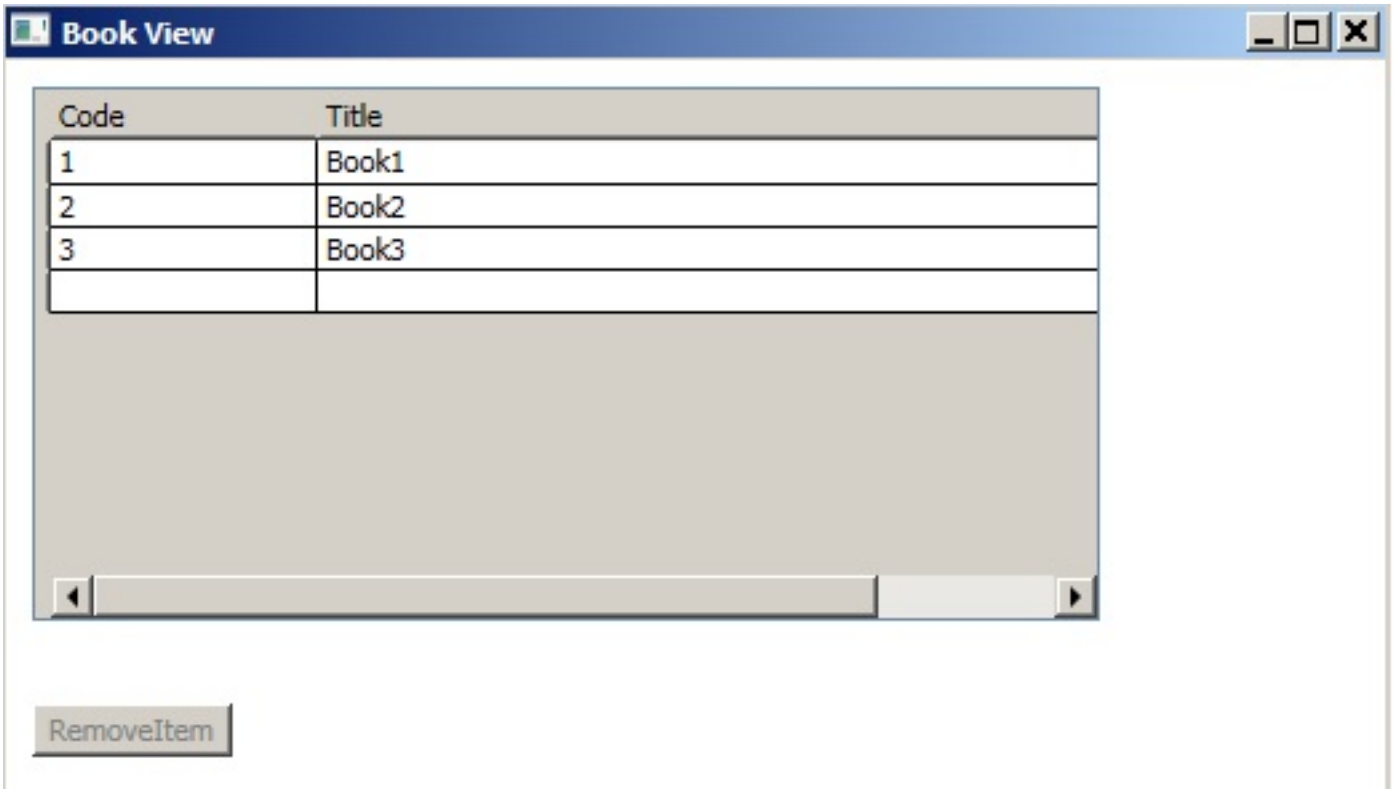
در این حالت بعد از اجرای برنامه همواره دکمه RemoveItem غیر فعال خواهد بود. دلیل آن این است که بعد از انتخاب آیتم مورد نظر از لیست باید کنترلر را متوجه تغییر در مقدار خاصیت CurrentItem نماییم. بدین منظور کد زیر را به متد Run اضافه کردم:

```
PropertyChangedEventManager.AddHandler(ViewModelCore, ViewModelChanged, "CurrentItem");
```

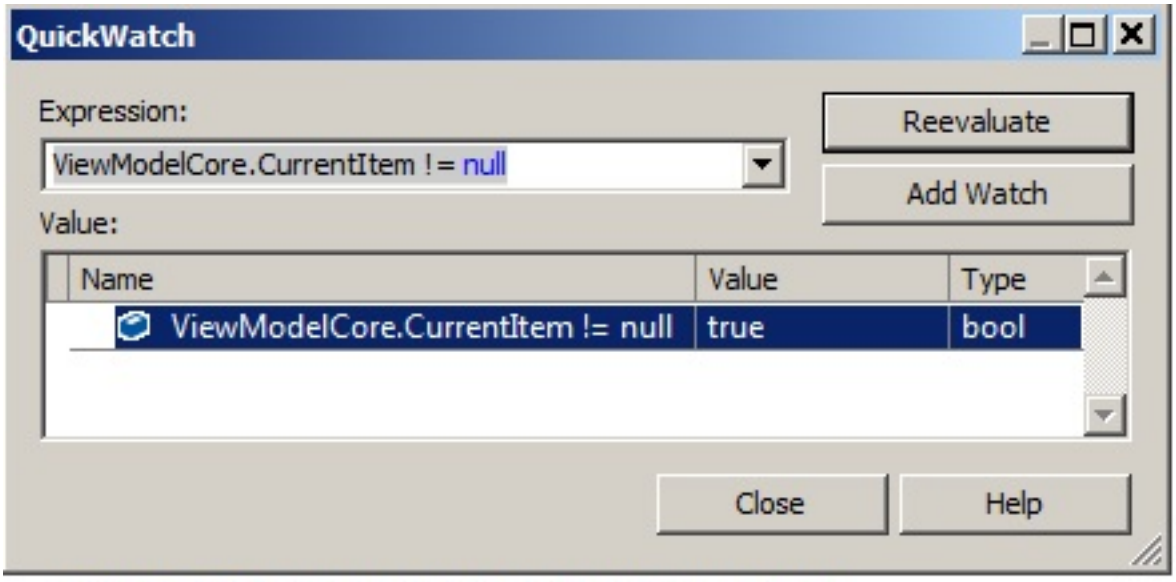
دستور بالا دقیقا معادل دستور AddWeakEventListener موجود در نسخه‌های قدیمی WAF است. سپس در صورتی که نام خاصیت مورد نظر CurrentItem بود با استفاده از دستور RaiseCanExecuteChanged در کلاس DelegateCommand کنترلر را ملزم به اجرای دوباره متد CanExecuteRemoveItemCommand می‌کنیم.

اجرای برنامه:

ابتدا دکمه RemoveItem غیر فعال است:

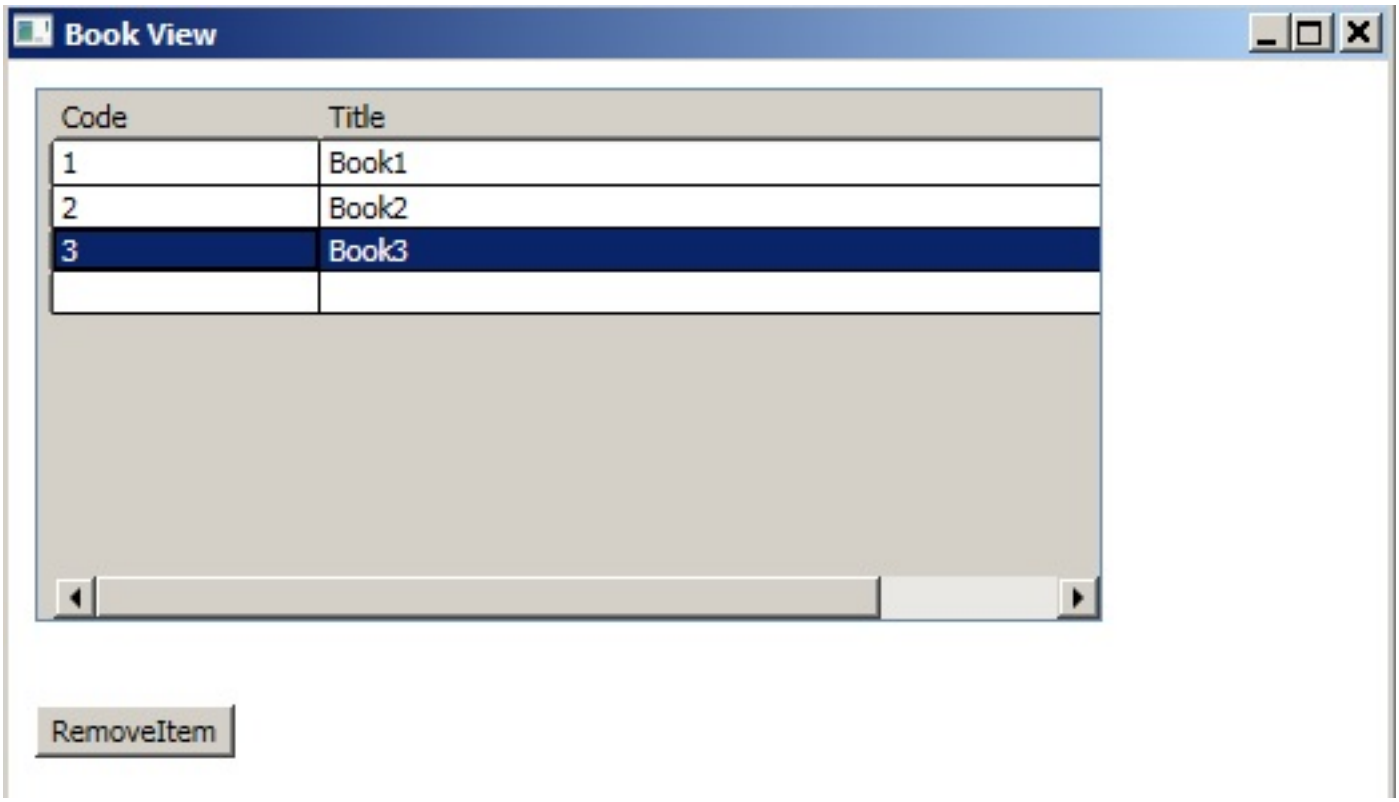


بعد از انتخاب یکی از گزینه و فراخوانی مجدد متد CanExecuteRemoveItemCommand دکمه مورد نظر فعال می‌شود:




```
private bool CanExecuteRemoveItemCommand()
{
    return ViewModelCore.CurrentItem != null;
}
```

و در نهایت دکمه RemoveItem فعال خواهد شد:



دانلود سورس پروژه