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.