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