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.