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.