## MVVM简介

MVVM，即 Model-View-ViewModel，是一种针对WPF、Silverlight、Windows Phone的设计模式，从MVC，MVP等模式中演化而来。主要目的也是为了解耦。

## MVVM的好处

设想一下，当你用 Winform 开发完一个界面超级复杂的项目之后，正当心里暗自切喜时，客户突然提出 “这个界面不好看，想用滑动条代替所有的按钮”，此时你会不会有种冲动想请他爷爷的爷爷的爷爷的....爷爷喝个茶？？

因为你知道这看似很简单的改动，却是项很繁重的体力活，原来按钮的事件处理函数全都用不了了，必须要把所有这些事件处理函数中的代码手动移动到滑动条的处理函数中，前提还必须不依赖事件参数。说不定一编译还各种错误，提示你某某控件不存在。

如果用 WPF 结合 Mvvm 模式，那就是轻松加愉快了。因为你不需要移动任何代码，要做的只是重新把View和ViewModel绑定一下就可以解决了。

其它的好处，我就不啰嗦了，谷歌上太多了。

下面来演示一个最简单的示例。通过该示例，你就会更清楚的知道 MVVM 的本质及其实施方式。

本示例的最终效果如下图所示，当用户在文本框中输入数据，并点击提交之后，就会将用户输入的数据显示在文本框下面。

[](http://img.chenxu.me/Fs5_6kJnMgBtSI6obJTDJDcSYf4L)

## 准备基础代码

**1. 创建一个 ViewModelBase**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | public abstract class ViewModelBase : INotifyPropertyChanged{        //属性改变事件      public event PropertyChangedEventHandler PropertyChanged;          //当属性改变的时候，调用该方法来发起一个消息，通知View中绑定了propertyName的元素做出调整      public void RaisePropertyChanged(string propertyName)      {          PropertyChangedEventHandler handler = PropertyChanged;            if (handler != null)          {              handler(this, new PropertyChangedEventArgs(propertyName));          }      }  } |

**2. 创建一个DelegateCommand**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | public class DelegateCommand : ICommand{      readonly Action<object> \_execute;      readonly Predicate<object> \_canExecute;        public DelegateCommand(Action<object> execute)          : this(execute, null)      {      }        public DelegateCommand(Action<object> execute, Predicate<object> canExecute)      {          if (execute == null)              throw new ArgumentNullException("execute");            \_execute = execute;          \_canExecute = canExecute;      }        public void Execute(object parameter)      {          \_execute(parameter);      }        public bool CanExecute(object parameter)      {          return \_canExecute == null ? true : \_canExecute(parameter);      }        public event EventHandler CanExecuteChanged      {          add { CommandManager.RequerySuggested += value; }          remove { CommandManager.RequerySuggested -= value; }      }  } |

## 创建示例用 ViewModel

让 ViewModel 继承自 ViewModelBase。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37 | public class MainWindowViewModel : ViewModelBase{      private string \_input;      public string Input      {          get        {              return \_input;          }          set        {              \_input = value;              RaisePropertyChanged("Input");          }      }        private string \_display;      public string Display      {          get        {              return \_display;          }          set        {              \_display = value;              RaisePropertyChanged("Display");          }      }        public DelegateCommand SetTextCommand { get; set; }        private void SetText(object obj)      {          Display = Input;      }        public MainWindowViewModel()      {          SetTextCommand = new DelegateCommand(new Action<object>(SetText));      }  } |

## 创建 View

最少只需要三个控件：一个textbox拿来做输入，一个label拿来做输出，一个button拿来提交数据。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | <Window x:Class="WpfApplication1.MainWindow"          xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"          xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"          xmlns:local="clr-namespace:WpfApplication1.ViewModel"          Title="MainWindow" Height="237" Width="215">              <Grid>                      <Button Content="提 交" HorizontalAlignment="Left" Margin="37,137,0,0" VerticalAlignment="Top" Width="75"/>                      <TextBox x:Name="tb" HorizontalAlignment="Left" Height="23" Margin="37,30,0,0" TextWrapping="Wrap" VerticalAlignment="Top" Width="120"/>                      <Label HorizontalAlignment="Left" Margin="37,76,0,0" VerticalAlignment="Top" />              </Grid>  </Window> |

## 绑定 View 和 ViewModel

当 View 和 ViewModel 都已经创建完之后，最后一步就是把它哥俩绑定在一起了。这样，当 View 改变的时候，ViewModel 就会发生相应的改变，反之亦然。

|  |  |
| --- | --- |
| 1  2  3  4  5 | <Grid>          <Button Content="提 交" HorizontalAlignment="Left" Margin="37,137,0,0" VerticalAlignment="Top" Width="75" Command="{Binding SetTextCommand}"/>          <TextBox x:Name="tb" HorizontalAlignment="Left" Height="23" Margin="37,30,0,0" TextWrapping="Wrap" VerticalAlignment="Top" Width="120" Text="{Binding Input}" />          <Label HorizontalAlignment="Left" Margin="37,76,0,0" VerticalAlignment="Top" Content="{Binding Display}" />  </Grid> |

## 提供数据源

这些绑定的数据从哪来又该到哪去呢？ 这个就要通过 DataContext 来实现了。其实很简单，只需要把数据源赋给DataContext就搞定了。

**可以直接在XAML中指定**

|  |  |
| --- | --- |
| 1  2  3 | <Window.DataContext>          <local:MainWindowViewModel></local:MainWindowViewModel>      </Window.DataContext> |

最终代码为

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <Window x:Class="WpfApplication1.MainWindow"          xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"          xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"          xmlns:local="clr-namespace:WpfApplication1.ViewModel"          Title="MainWindow" Height="237" Width="215">      <Window.DataContext>          <local:MainWindowViewModel></local:MainWindowViewModel>      </Window.DataContext>      <Grid>          <Button Content="提 交" HorizontalAlignment="Left" Margin="37,137,0,0" VerticalAlignment="Top" Width="75" Command="{Binding SetTextCommand}"/>          <TextBox x:Name="tb" HorizontalAlignment="Left" Height="23" Margin="37,30,0,0" TextWrapping="Wrap" VerticalAlignment="Top" Width="120" Text="{Binding Input}" />          <Label HorizontalAlignment="Left" Margin="64,80,0,0" VerticalAlignment="Top" Content="{Binding Display}" />      </Grid>  </Window> |

**或者也可以在 CodeBehind 中指定**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | public partial class MainWindow : Window      {          public MainWindow()          {              InitializeComponent();              base.DataContext = new MainWindowViewModel();          }      } |

## 源代码管理

关于此项目的代码，您可以访问 [The simplest demo for MVVM](https://simplestmvvmdemo.codeplex.com/)

