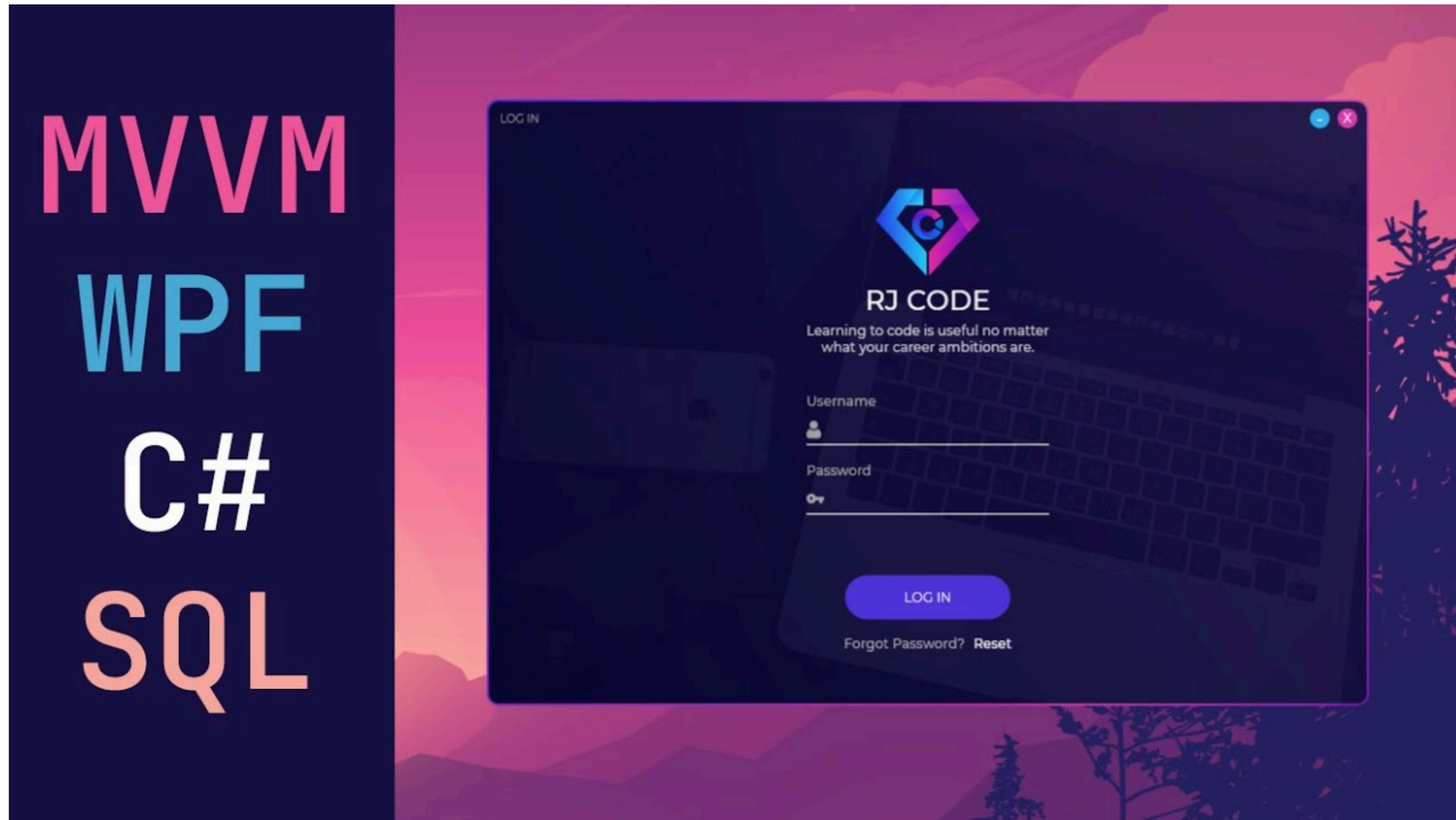


WPF开发新境界：MVVM设计模式解析与实战，构建清晰可维护的用户界面！

作者：微信公众号：【架构师老卢】

11-29 10:12

192



**概述：** MVVM是一种在WPF开发中广泛应用的设计模式，通过将应用程序分为模型、视图、和视图模型，实现了解耦、提高可维护性的目标。典型应用示例展示了如何通过XAML、ViewModel和数据绑定创建清晰、可测试的用户界面。

## 什么是MVVM？

MVVM (Model-View-ViewModel) 是一种用于构建用户界面的软件设计模式，它将应用程序分为三个核心组件：模型 (Model)、视图 (View) 和视图模型 (ViewModel)。MVVM的目标是实现界面逻辑与用户界面的分离，提高代码的可维护性和可测试性。

## 为什么要用MVVM？

MVVM带来了以下优点：

- 松散耦合：** 模型、视图、和视图模型相互独立，降低了各个组件之间的耦合度。
- 可维护性：** 分离关注点使得代码更易于理解和维护。
- 可测试性：** 视图模型可以方便地进行单元测试，无需依赖具体的UI元素。

## MVVM应用实例：

### 1. 创建模型 (Model)：定义数据模型。

```
1 public class PersonModel
2 {
3     public string FirstName { get; set; }
4     public string LastName { get; set; }
5 }
```

### 2. 创建视图模型 (ViewModel)：实现业务逻辑和与视图相关的命令。

```
1 public class PersonViewModel : INotifyPropertyChanged
2 {
3     private PersonModel _person;
4
5     public PersonViewModel()
6     {
7         _person = new PersonModel();
8     }
9
10    public string FirstName
11    {
12        get { return _person.FirstName; }
13        set
14        {
15            if (_person.FirstName != value)
16            {
17                _person.FirstName = value;
18                OnPropertyChanged(nameof(FirstName));
19            }
20        }
21    }
22
23    public string LastName
24    {
25        get { return _person.LastName; }
26        set
27        {
28            if (_person.LastName != value)
29            {
30                _person.LastName = value;
31                OnPropertyChanged(nameof(LastName));
32            }
33        }
34    }
35
36    // INotifyPropertyChanged实现省略...
37
38    private void OnPropertyChanged(string propertyName)
39    {
40        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
41    }
42
43    public event PropertyChangedEventHandler PropertyChanged;
44 }
```

### 3. 创建视图 (View)：利用XAML定义用户界面。

```
1 <Window x:Class="MVVMSample.MainWindow"
2     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4     xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
5     xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
6     xmlns:local="clr-namespace:MVVMSample"
7     mc:Ignorable="d"
8     Title="MainWindow" Height="200" Width="300">
9     <Grid>
10        <StackPanel Margin="10">
11            <TextBox Text="{Binding FirstName}" Margin="0 0 0 5"/>
12            <TextBox Text="{Binding LastName}" Margin="0 0 0 5"/>
13            <Button Content="Submit" Command="{Binding SubmitCommand}"/>
14        </StackPanel>
15    </Grid>
16 </Window>
```

### 4. 将视图与视图模型关联：在视图的代码-behind或XAML中，将DataContext设置为视图模型的实例。

```
1 public partial class MainWindow : Window
2 {
3     public MainWindow()
4     {
5         InitializeComponent();
6
7         // 关联视图模型
8         DataContext = new PersonViewModel();
9     }
10 }
```

### 5. 实现提交命令 (Command)：在视图模型中定义和实现命令。

```
1 public class PersonViewModel : INotifyPropertyChanged
2 {
3     // 其他代码省略...
4
5     public ICommand SubmitCommand => new RelayCommand(Submit);
6
7     private void Submit()
8     {
9         MessageBox.Show($"Submitted: {FirstName} {LastName}");
10    }
11 }
```

MVVM设计模式通过将应用程序分为模型、视图和视图模型，实现了解耦和分离关注点的目标。上述实例演示了如何在WPF中应用MVVM，通过数据绑定和命令使得界面逻辑更清晰、易于测试和维护。