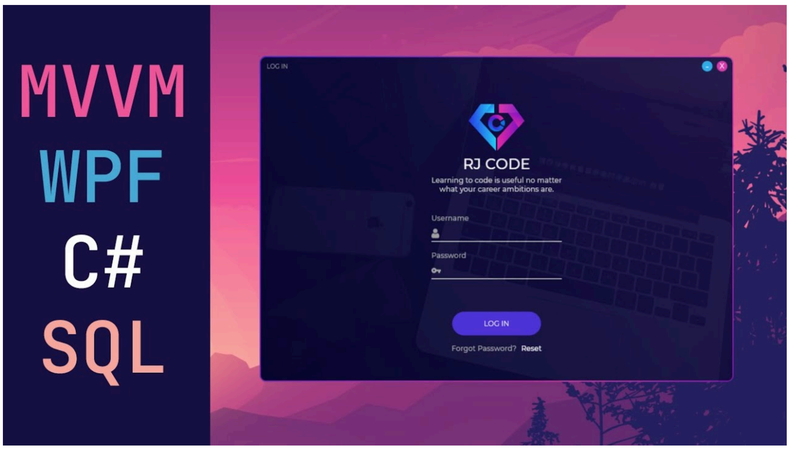
WPF开发新境界：MVVM设计模式解析与实战，构建清晰可维护的用户界面！

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：

1. **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。
2. **可维护性：** 分离关注点使得代码更易于理解和维护。
3. **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。

```
public class PersonModel
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
}
```

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

```
public class PersonViewModel : INotifyPropertyChanged
{
    private PersonModel _person;

    public PersonViewModel()
    {
        _person = new PersonModel();
    }

    public string FirstName
    {
        get { return _person.FirstName; }
        set
        {
            if (_person.FirstName != value)
            {
                _person.FirstName = value;
                OnPropertyChanged(nameof(FirstName));
            }
        }
    }

    public string LastName
    {
        get { return _person.LastName; }
        set
        {
            if (_person.LastName != value)
            {
                _person.LastName = value;
                OnPropertyChanged(nameof(LastName));
            }
        }
    }

    // INotifyPropertyChanged实现省略...

    private void OnPropertyChanged(string propertyName)
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    }

    public event PropertyChangedEventHandler PropertyChanged;
}
```

3. **创建视图（View）：** 利用XAML定义用户界面。

```
<Window x:Class="MVVMSample.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"
        xmlns:local="clr-namespace:MVVMSample"
        mc:Ignorable="d"
        Title="MainWindow" Height="200" Width="300">
    <Grid>
        <StackPanel Margin="10">
            <TextBox Text="{Binding FirstName}" Margin="0 0 0 5"/>
            <TextBox Text="{Binding LastName}" Margin="0 0 0 5"/>
            <Button Content="Submit" Command="{Binding SubmitCommand}"/>
        </StackPanel>
    </Grid>
</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。

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();

        // 关联视图模型
        DataContext = new PersonViewModel();
    }
}
```

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

```
public class PersonViewModel : INotifyPropertyChanged
{
    // 其他代码省略...

    public ICommand SubmitCommand => new RelayCommand(Submit);

    private void Submit()
    {
        MessageBox.Show($"Submitted: {FirstName} {LastName}");
    }
}
```

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。