WinUI 3 | Ep3. 简单的数据绑定

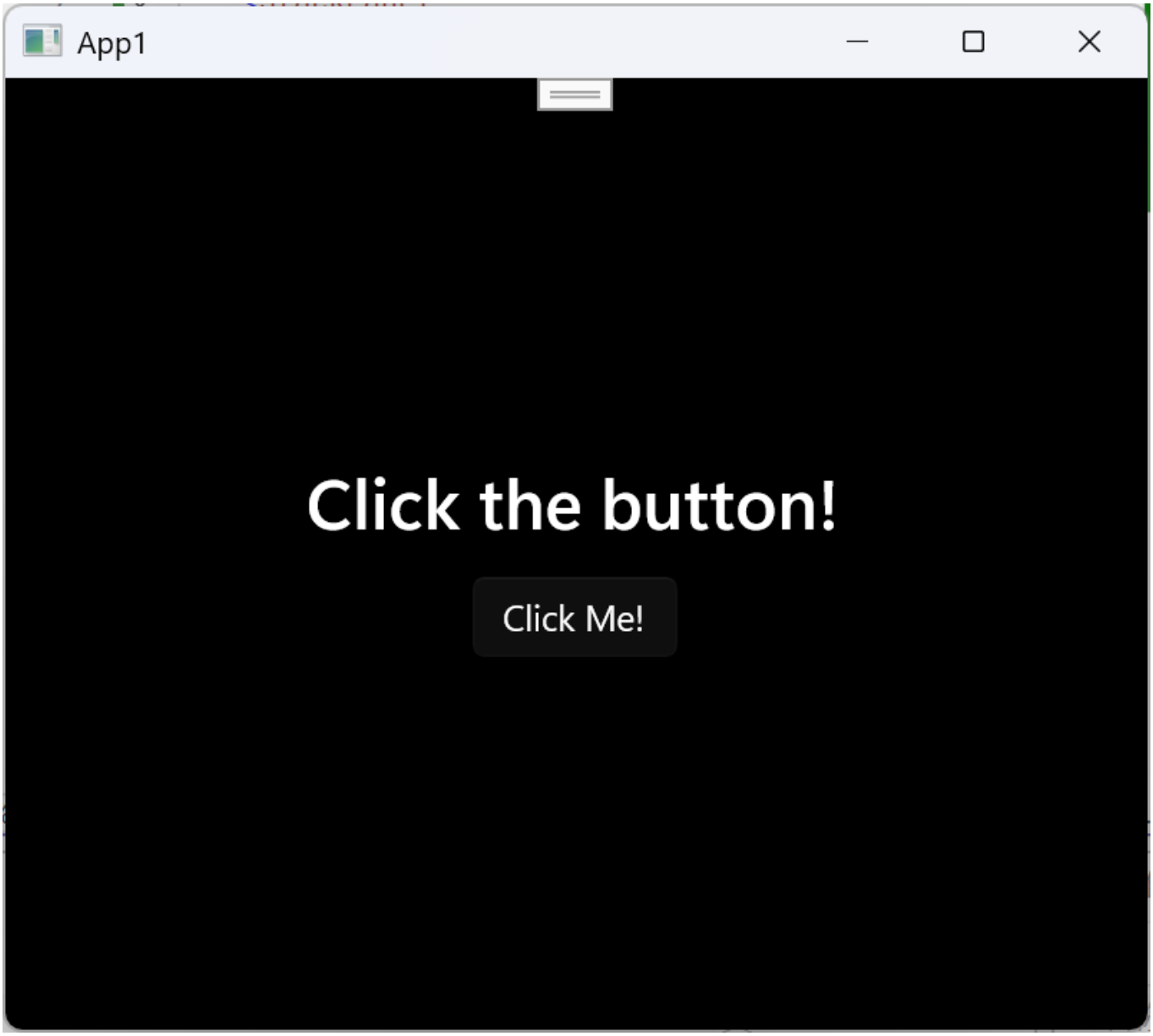
一个简单的点击计数器应用

对于一个应用，它无外乎两个功能——输入和输出。

让我们来做一个最简单的计数器吧。修改之前的 `MainWindow.xaml`：

```
<?xml version="1.0" encoding="utf-8"?>
<Window
    x:Class="App1.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="App1">
    <StackPanel
        VerticalAlignment="Center"
        HorizontalAlignment="Center"
        Spacing="12">
        <TextBlock
            x:Name="CounterTextBlock"
            HorizontalAlignment="Center"
            Style="{ThemeResource TitleTextBlockStyle}"
            Text="Click the button!" />
        <Button
            x:Name="IncrementButton"
            HorizontalAlignment="Center"
            Content="Click Me!" />
    </StackPanel>
</Window>
```

运行得到的大概是这样一个界面

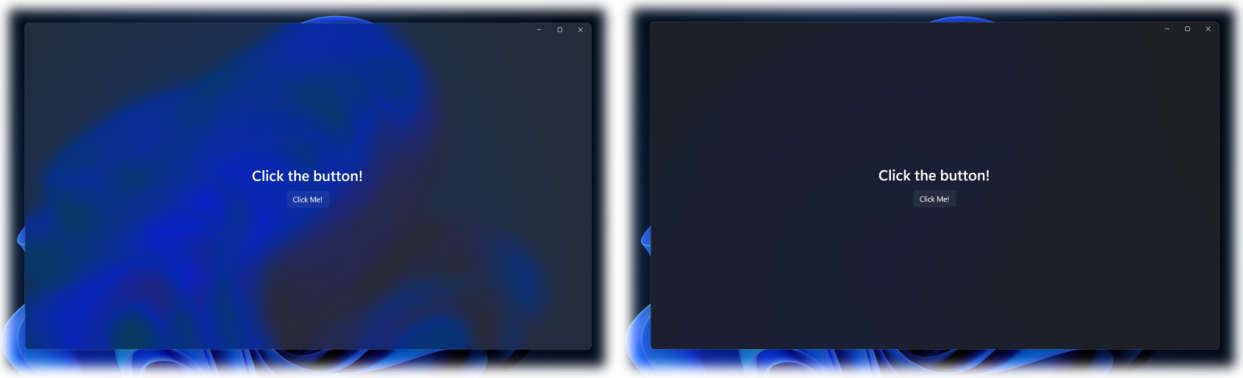


一点美化

默认的黑底不太好看，但是WinUI 3可以让我们非常方便的使用Windows 11的Mica与毛玻璃背景。只需要在 `Window` 类中设置属性 `SystemBackdrop` 的值为 `MicaBackdrop` 类型或者 `DesktopAcrylicBackdrop` 即可。我们可以在XAML中轻松完成：

```
<Window ...>
    <Window.SystemBackdrop>
        <DesktopAcrylicBackdrop />
    </Window.SystemBackdrop>
        ...
</Window>
```

我们也可以禁用标题栏。在 `MainWindow` 的构造函数中，设置 `ExtendsContentIntoTitleBar = true;` 。

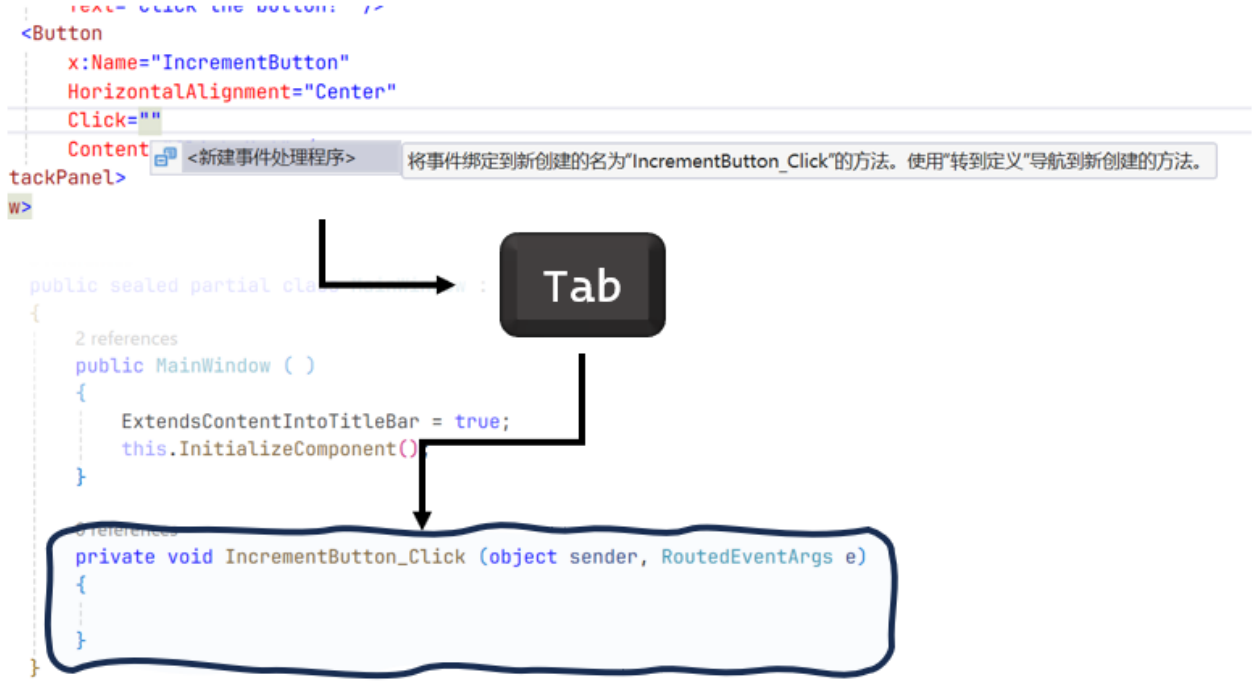


`<DesktopAcrylicBackdrop />` `<MicaBackdrop />`

这样漂亮多了，和Windows本身的风格完美融合在一起。

实现计数逻辑

既然我们要实现每点击一次按钮次数加一，我们应该要对按钮点击事件做出响应。设置按钮的`Click`事件，Visual Studio会提出自动创建一个事件处理函数，按下`tab`接受建议。



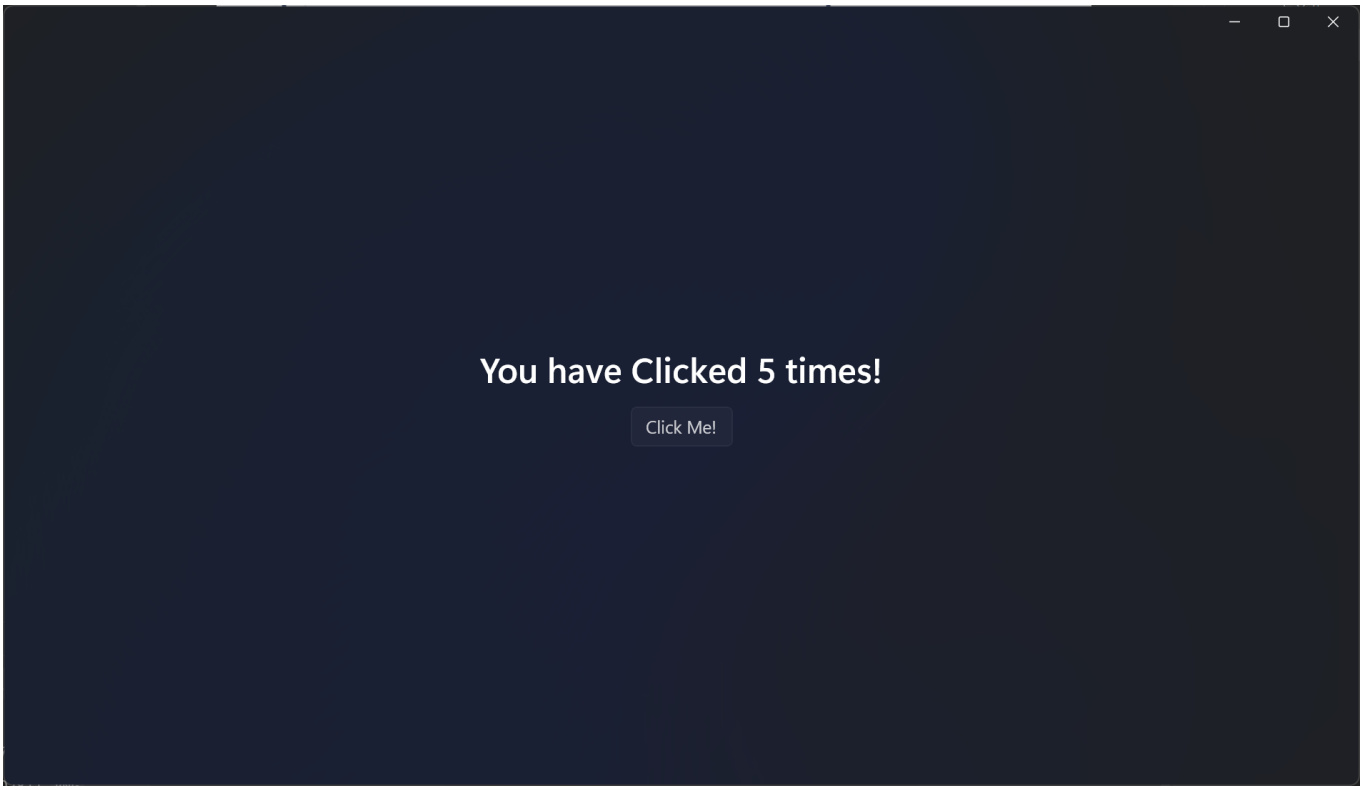
完成代码。

```
public sealed partial class MainWindow : Window
{
    int ClickCount { get; set; } = 0;

    public MainWindow ( )
    {
        ExtendsContentIntoTitleBar = true;
        this.InitializeComponent();
    }

    private void IncrementButton_Click (object sender, RoutedEventArgs e)
    {
        ClickCount++;
        CounterTextBlock.Text = $"You have clicked {ClickCount} times!";
    }
}
```

运行，点击按钮。



增加一点功能

目前，点击一次按钮只会使次数+1。有人嫌点击太慢了，希望有一个点一次次数+10的按钮。

为了实现这个功能，我们只需要在XAML中添加一个按钮，将按钮的`Click`事件的处理函数设置为刚才的方法。借助事件传入的事件参数，以及为两个按钮分别设置Tag，我们可以区分究竟是

哪个按钮触发了事件。

```
<Button
        x:Name="IncrementButton"
        Click="IncrementButton_Click"
        Tag="1" .../>
<Button
        x:Name="Increment10TimesButton"
        Click="IncrementButton_Click"
        Tag="10" .../>
```

```
private void IncrementButton_Click (object sender, RoutedEventArgs e)
{
        ClickCount += sender switch
        {
                Button { Tag: "1" }     => 1,  // 如果触发事件的是Tag为1的按钮
                Button { Tag: "10" }    => 10, // 如果触发事件的是Tag为10的按钮
                _                       => 0,
        };
        CounterTextBlock.Text = $"You have clicked {ClickCount} times!";
}
```

如果我想把两个按钮的内容也设置成点击次数：

```
...
CounterTextBlock.Text = $"You have clicked {ClickCount} times!";
IncrementButton.Content = $"Clicked {ClickCount}";
Increment10TimesButton.Content = $"Clicked {ClickCount}";
...
```

你会发现，为了展示数据最新的值，每当ClickCount被修改的时候，我们都需要手动更新每个按钮和TextBlock的内容。这样很麻烦。因此XAML中有一种新方法来完成数据的更新：

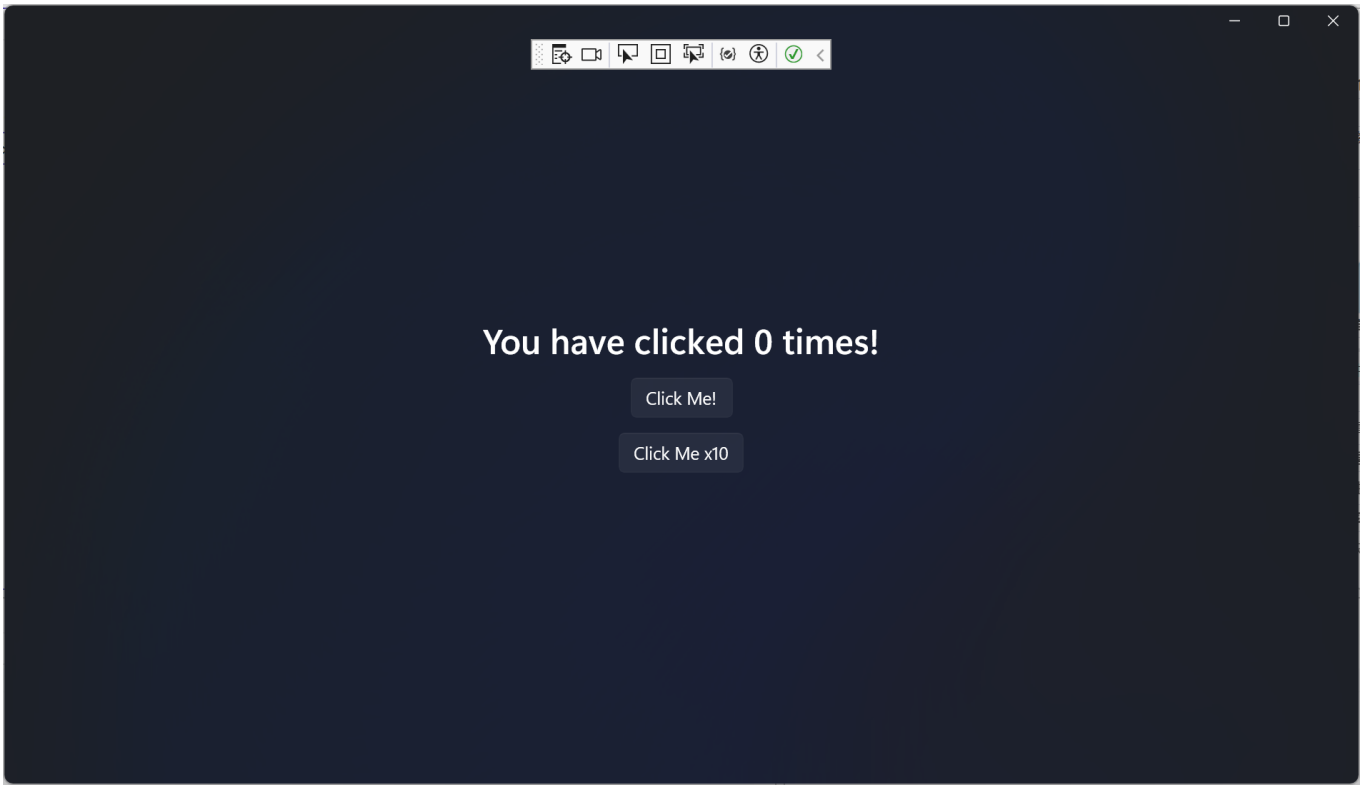
数据绑定

数据绑定语法是XAML的一大特色。借助数据绑定，我们不再需要手动去更新UI控件的内容了，只需要声明UI控件的属性值绑定到后台代码的某个属性上，然后框架会自动完成UI控件内容的更新。

使用 `x:Bind` 这一XAML标记扩展，我们可以将TextBlock的Text属性绑定到后台代码的 `string TitleText => $"You have clicked {ClickCount} times!";` ：

```
<TextBlock ... Text="{x:Bind TitleText}" />
```

删除点击事件中直接操作控件内容的代码，运行程序



你会发现无论如何点击都不会改变。这是由两个原因造成的：

- `{x:Bind}` 出于性能考虑，默认仅在初始化时更新一次
  - 只需要设置 `{x:Bind ..., Mode=OneWay}` 即可
- 数据绑定需要有东西通知它属性的值被更改了。

对于第二个问题，我们需要让MainWindow实现INotifyPropertyChanged接口，这个接口有一个叫做 `PropertyChanged` 的事件。在后台代码中加入：

```
public event PropertyChangedEventHandler PropertyChanged;

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

当 `ClickCount` 被改变时，调用OnPropertyChanged，表示属性被更改了。

```
private int clickCount = 0;

int ClickCount
{
        get => clickCount;
        set
        {
                clickCount = value;
                OnPropertyChanged(nameof(TitleText));
        }
}
```

这时，界面上展示的值就能自动更新了，不需要我们手动设置每个控件。这让我们编写逻辑代码的时候，不用再去考虑用户界面，可以极大减少我们编程时的心智负担，并且降低出现Bug的概率。换句话说，我们在一定程度上实现了业务逻辑与用户界面的解耦。

数据绑定是MVVM架构的重要组成部分，MVVM架构的核心就是业务逻辑与用户界面的解耦。在下一节我们将着重介绍什么是MVVM，以及借助MVVM架构完成进一步的解耦。