WPF面向对象基础编程技巧

北京理工大学计算机学院
金旭亮

XAML中的命名空间（namespace）

MainWindow.xaml

```
<Window x:Class="HelloWorldWPFCore.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:HelloWorldWPFCore"
        mc:Ignorable="d"
        Title="MainWindow" Height="280" Width="400">
    <Grid...>
</Window>
```

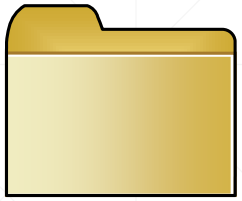
几乎每一个XAML文件都有的这个是什么东东？
有什么用？

XML命名空间声明方式：

xmlns:命名空间名字="....."

“命名空间”的隐藏含义

XAML中的命名空间



程序集中的命名空间

1. “没有名字”的命名空间：

```
xmlns="http://..."
```

映射 →

包含WPF核心对象和Grid、Button等控件类型的System.Windows.controls等多个命名空间

2. “有名字”的命名空间：

```
xmlns:x="http://..."
```

映射 → System.Windows.Markup

真实场景：XAML需要引用外部类型

- XAML文档中的元素往往直接对应到WPF相关命名空间中的类（比如TextBlock等控件）。

- 然而，在实际开发中，我们可能需要在XAML文档中使用放在第三方程序集中的类型，怎样做才能达到这个目的？

“在XAML中使用外部类型”技术要点

1. 在XAML中引入外部类所在的命令空间，并给其定义一个前缀，其格式如下：

```
xmlns:你所起的前缀名=
        "clr-namespace:完整的命名空间名;assembly=完整的程序集名称"
```

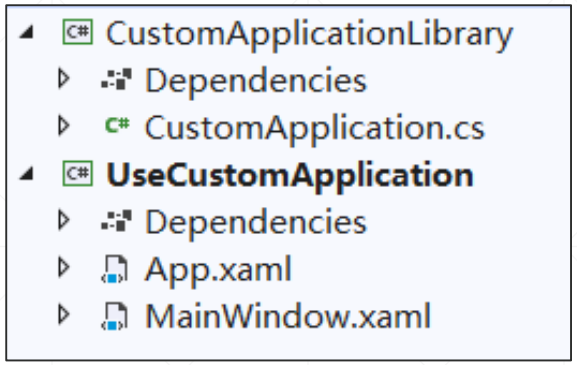
例如

```
xmlns:system="clr-namespace:System;assembly=mscorlib"
```

2. 引入新命名空间之后，即可在XAML中使用此命名空间中的类。

```
<TextBlock
        Text="{x:Static system:Environment.MachineName}" >
</TextBlock>
```

使用外部自定义程序集中的类示例



CustomApplicationLibrary**类库项目**中写了一个自定义的Application类：

```
namespace CustomApplicationLibrary
{
    //派生自WPF框架内置的Application类，重写其方法
    //扩充其功能
    3 references
    public class CustomApplication : Application
    {
        0 references
        protected override void OnStartup(StartupEventArgs e)
        {
            base.OnStartup(e);
            MessageBox.Show("你好，欢迎使用本程序!");
        }
        //获取本类的完整信息
        1 reference
        public string Info() => "CustomApplicationLibrary.CustomApplication";

    }
}
```

App.xaml (UseCustomApplication项目)

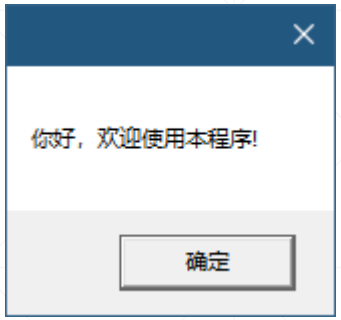
```
<custom:CustomApplication x:Class="UseCustomApplication.App"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:local="clr-namespace:UseCustomApplication"
             xmlns:custom="clr-namespace:CustomApplicationLibrary;assembly=CustomApplicationLibrary"
             StartupUri="MainWindow.xaml">
    <Application.Resources>

    </Application.Resources>
</custom:CustomApplication>
```

```
namespace UseCustomApplication
{
    3 references
    public partial class App : CustomApplication
    {
    }
}
```

注意一下WPF项目的App.xaml中引入了新的命名空间，并且指定使用放在类库项目中的自定义Application对象。

示例运行结果

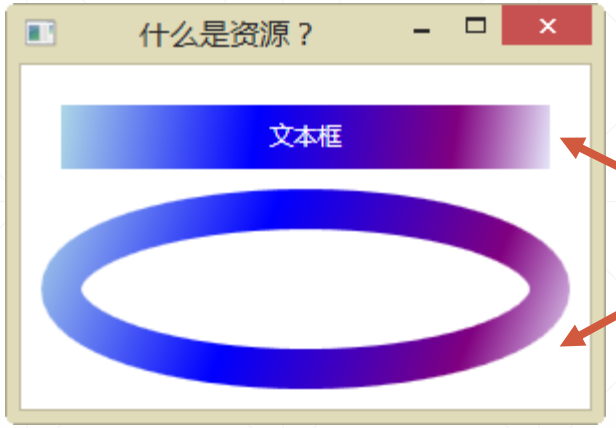
1. 首先显示一个消息框：



2. 主窗体显示，输出当前Application实例的类型信息：

XAML中的对象重用

“对象资源”—在XAML中重用对象的基本方法



两个图形，使用相同的画刷对象进行填充

示例：WhatIsResource

在WPF中，“对象资源”代表了一些可以在XAML元素定义时使用的可重用对象。

WPF对象资源的定义与使用方法

1 **定义**：存放在资源中的对象拥有一个**Key**作为标识。

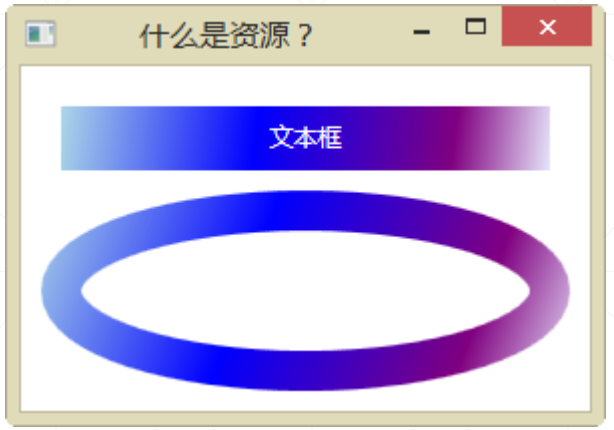
```
<Window.Resources>
    <LinearGradientBrush x:Key="backgroundBrush"...>
</Window.Resources>
```

2 **引用**：使用StaticResource扩展标记通过**Key**来引用资源对象：

```
<Ellipse StrokeThickness="20" Height="100"
         Stroke="{StaticResource backgroundBrush}" />
```

静态资源，
先定义后引用

小结：XAML中使用“对象资源”的步骤




```
<Window x:Class="StaticResource.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="什么是资源？" SizeToContent="Height" Width="300" >
    <Window.Resources>
        <LinearGradientBrush x:Key="backgroundBrush">
            <GradientStop Offset="0" Color="LightBlue"/>
            <GradientStop Offset="0.4" Color="Blue"/>
            <GradientStop Offset="0.8" Color="Purple"/>
            <GradientStop Offset="1.0" Color="Lavender"/>
        </LinearGradientBrush>
    </Window.Resources>

    <StackPanel Margin="10">
        <TextBlock Height="32" Margin="10"
                   Background="{StaticResource backgroundBrush}"
                   Foreground="White" VerticalAlignment="Center"
                   Text="文本框"
                   TextAlignment="Center" Padding="8" />
        <Ellipse StrokeThickness="20" Height="100"
                 Stroke="{StaticResource backgroundBrush}" />
    </StackPanel>
</Window>
```

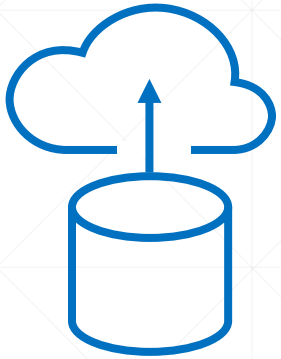
定义“画刷资源” 1

使用“画刷资源” 2

能够回到过去的“时间机器”

WPF对象状态的暂存与恢复

对象状态的序列化



对象

序列化：将对象当前状态保存到流中

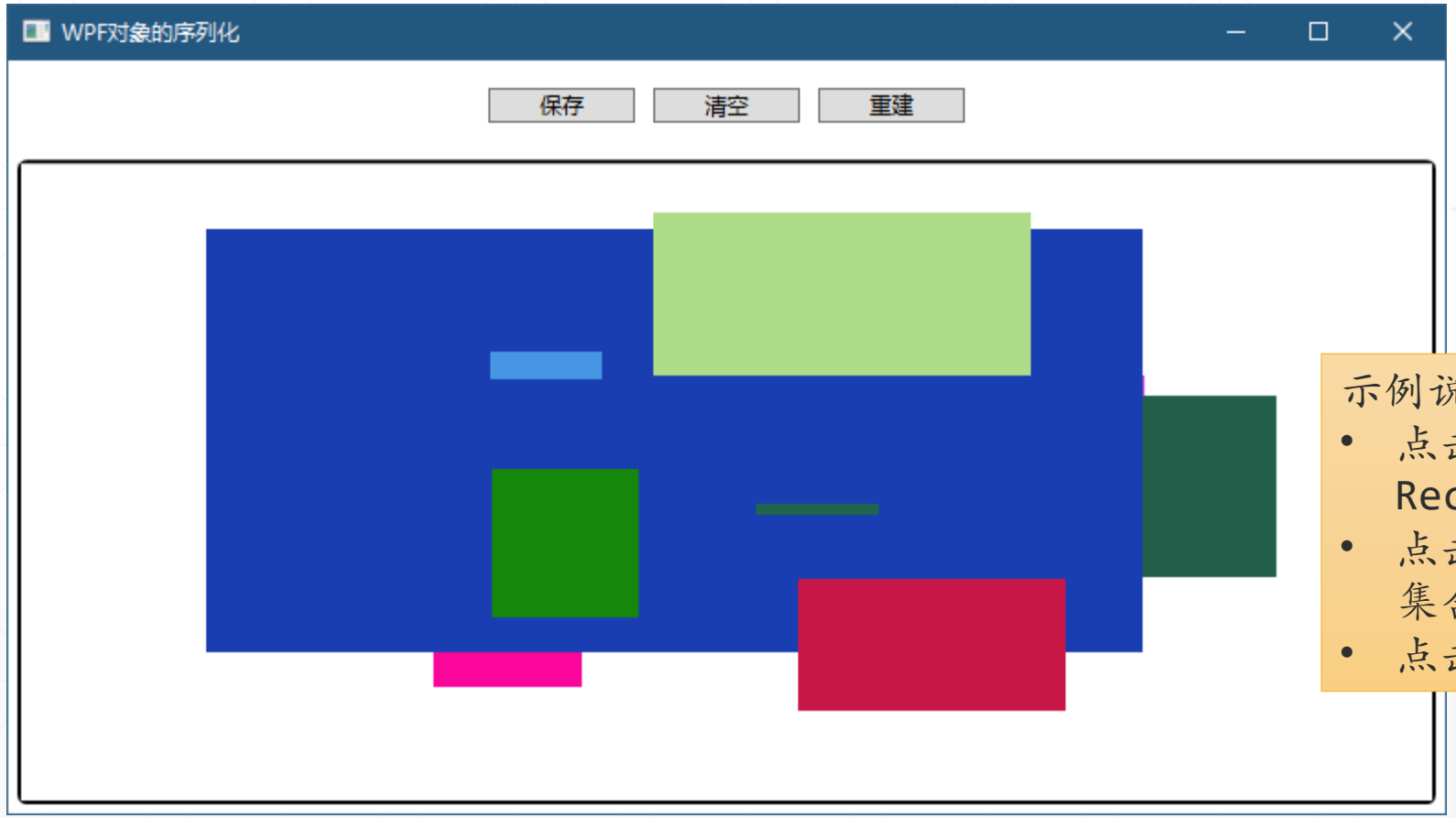
319f	a101	8b34	6713	90ab	319c	...

流

对象

反序列化：从流中读取数据，重建对象并将其状态恢复原值

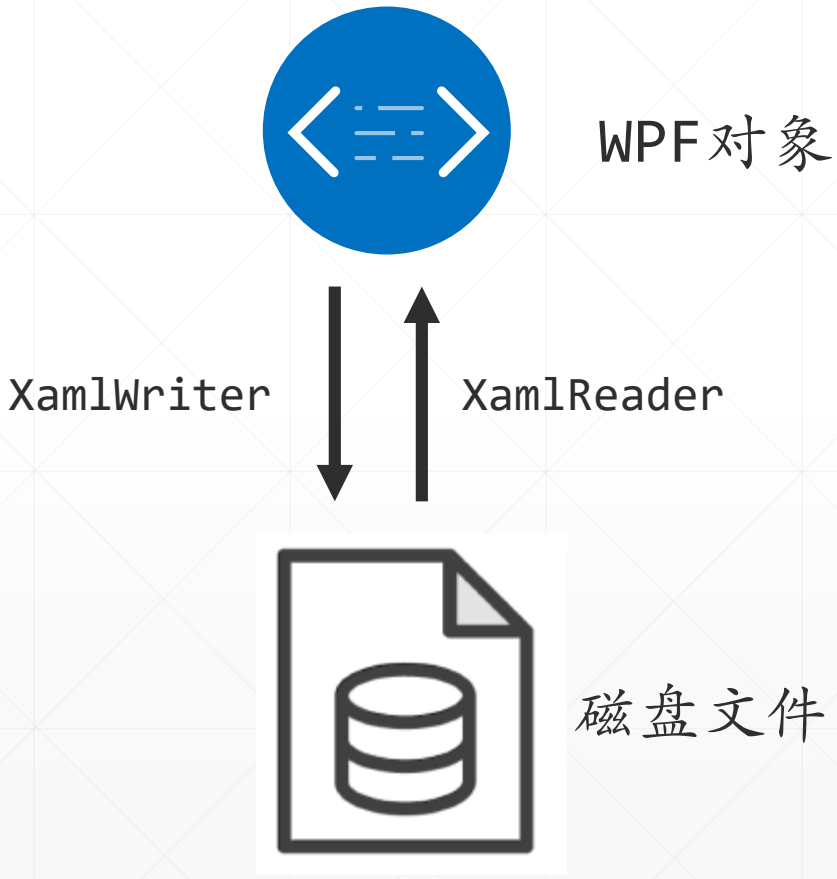
示例项目：SaveAndReloadXAMLObject



示例说明：

- 点击绘图面板，将随机添加一个Rectangle对象。
- 点击“保存”，会将当前Rectangle对象集合保存到PicData.dat文件中。
- 点击“重建”，会从文件中重建绘图结果

示例程序中的WPF对象序列化代码




```
1 reference
private void btnSave_Click(object sender, RoutedEventArgs e)
{
    using (FileStream fs = new FileStream("PicData.dat", FileMode.Create))
    {

        XamlWriter.Save(DrawPanel, fs);

        MessageBox.Show("数据已保存");
    }
}
```

保存绘图结果

```
1 reference
private void btnLoad_Click(object sender, RoutedEventArgs e)
{
    using (FileStream fs = new FileStream("PicData.dat", FileMode.Open))
    {

        Canvas obj = XamlReader.Load(fs) as Canvas;

        DrawPanelBorder.Child = obj;

    }
}
```

恢复画布

使用控件嵌套的方法构建UI界面

了解一下WPF控件的类别

布局控件

（负责排列控件，比如Grid、StackPanel等等）

内容控件
（只能有一个直接子控件，比如Button、TextBox）

列表控件
（可以有多个直接子控件，比如ListBox）

内容控件（ContentControl）的嵌套示例


```
<Button Margin="3" Padding="5">文本类型的简单按钮</Button>
<Button Margin="3" Padding="5">
    <Image Source="happyface.jpg" Stretch="None" />
</Button>
```

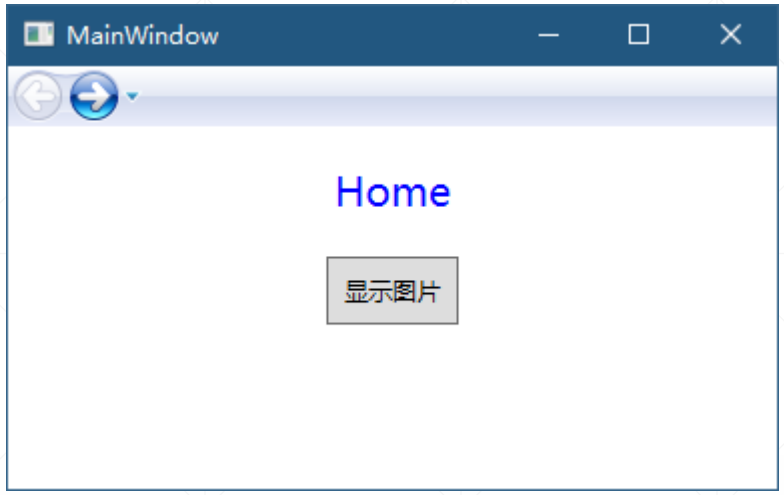
```
<Button Margin="3">
    <StackPanel Margin="5">
        <TextBlock Margin="3">图像和文本型按钮</TextBlock>
        <Image Source="happyface.jpg" Stretch="None" />
        <TextBlock Margin="3" >按钮嵌套了一个StackPanel</TextBlock>
    </StackPanel>
</Button>
```

```
<Button Margin="3" Padding="3">
    <StackPanel Margin="5">
        <TextBlock>按钮中嵌入Polygon图形</TextBlock>
        <Grid>
            <Polygon Points="100,25 125,0 200,25 125,50"
     Fill="Red" />
            <Polygon Points="100,25 75,0 0,25 75,50"
     Fill="Blue"/>
        </Grid>
    </StackPanel>
</Button>
```

示例项目：EmbedContentControl

多页面与多窗体编程

多页面的WPF应用程序示例：MultiPageDemo



主窗体是一个NavigationWindow类的实例，不同的内容放在不同的页面中（顶层元素是Page）。NavigationWindow有相关的方法在不同页面中导航。

示例中关键代码

```xaml
<NavigationWindow x:Class="MultiPageDemo.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:MultiPageDemo"
        mc:Ignorable="d"
        Source="Home.xaml"
        Title="MainWindow" Width="400" Height="250">

</NavigationWindow>
```

MainWindow.xaml

主窗体的Source属性，指定第一个要显示的页面是Home.xaml。

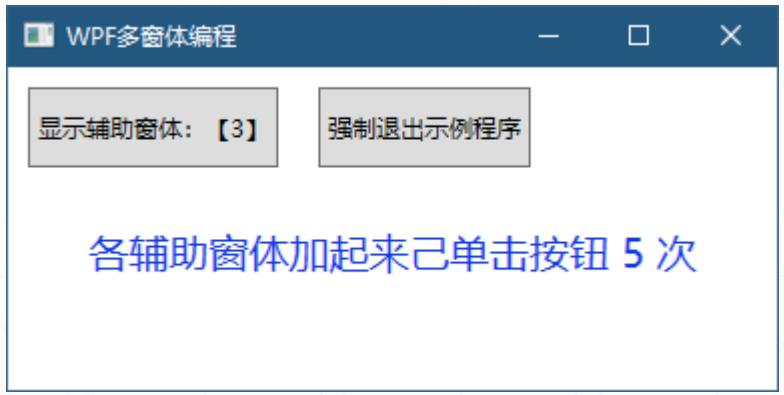
Home.xaml.cs

```csharp
public partial class Home : Page
{
    0 references
    public Home()
    {
        InitializeComponent();
    }

    1 reference
    private void Button_Click(object sender, RoutedEventArgs e)
    {
        this.NavigationService.Navigate(new ImagePage());
    }
}
```

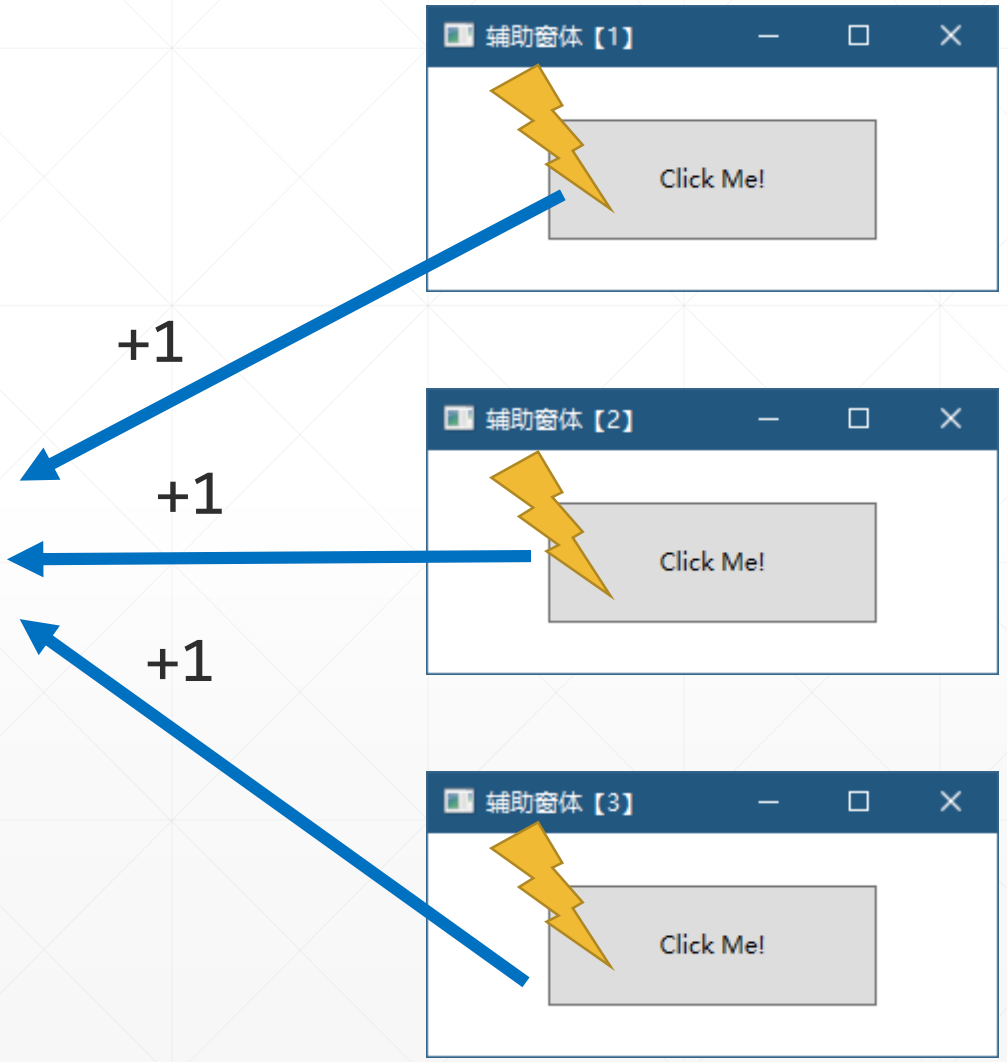
Home页面的按钮单击事件响应代码中，调用NavigationService类的相关方法导航到下一个页面。

WPF多窗体示例

主窗体

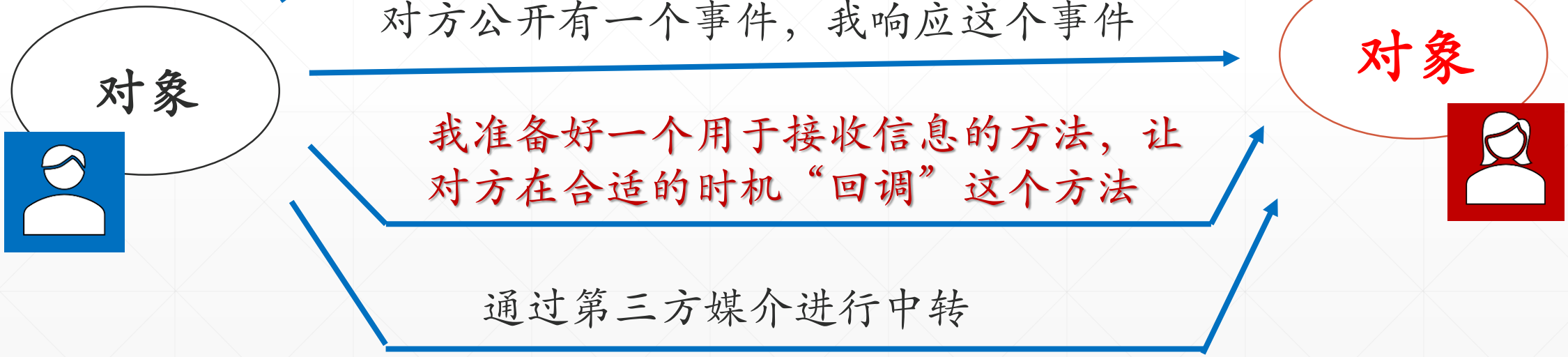


示例：WPFMultiWindow



动态创建的多个辅助窗体

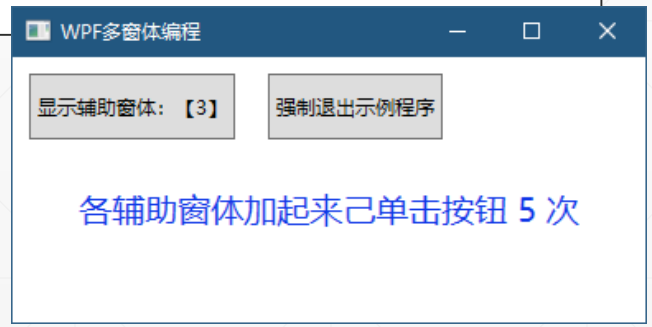
实现对象之间相互通讯的常见“编程套路”



示例关键代码

MainWindow.xaml.cs

```
1 reference
private void BtnShowOther_Click(object sender, RoutedEventArgs e)
{
    //实例化一个辅助窗体对象，将主窗体引用传给它
    var otherWindow = new OtherWindow(this);
    otherWindow.Show(); //显示辅助窗体
    //从辅助窗体的公有静态只读属性中读取信息，获取已创建的窗体个数
    (sender as Button).Content =
        $"显示辅助窗体：【{OtherWindow.Counter}】";
}

//用于累加辅助窗体按钮点击次数的计数器
private int counter = 0;
//供辅助窗体"回调"的方法
1 reference
public void IncreaseNumber()
{
    counter++;
    //显示各窗体的按钮单击统计数
    tbCount.Text = $"各辅助窗体加起来已单击按钮 {counter} 次";
}
```



主窗体

OtherWindow.xaml.cs

```
public partial class OtherWindow : Window
{
    //保存主窗体引用
    private MainWindow main = null;

    //统计已创建的辅助窗体个数（外部只读属性）
    3 references
    public static int Counter { get; private set; } = 0;
    1 reference
    public OtherWindow(MainWindow mainWindow)
    {
        InitializeComponent();
        //保存主窗体引用
        this.main = mainWindow;
        Counter++;
        this.Title = $"辅助窗体【{Counter}】";
    }

    1 reference
    private void BtnClickMe_Click(object sender, RoutedEventArgs e)
    {
        //通过主窗体引用"回调"主窗体方法
        main.IncreaseNumber();
    }
}
```

辅助窗体

小结

本讲通过几个实例，介绍了WPF开发中的一些常用技术特性和编程基础套路，理解与把握它们的前提，是你的面向对象基础是否扎实。

如果本讲所介绍的内容你学起来感觉有困难，这就是一个明显的信号，提醒你需要把面向对象这块基础好好补一补，不然，在学习更复杂的技术时，会遇到重重阻碍。