WPF系列之一

WPF技术概述

北京理工大学计算机学院
金旭亮

了解 WPF

什么是WPF？

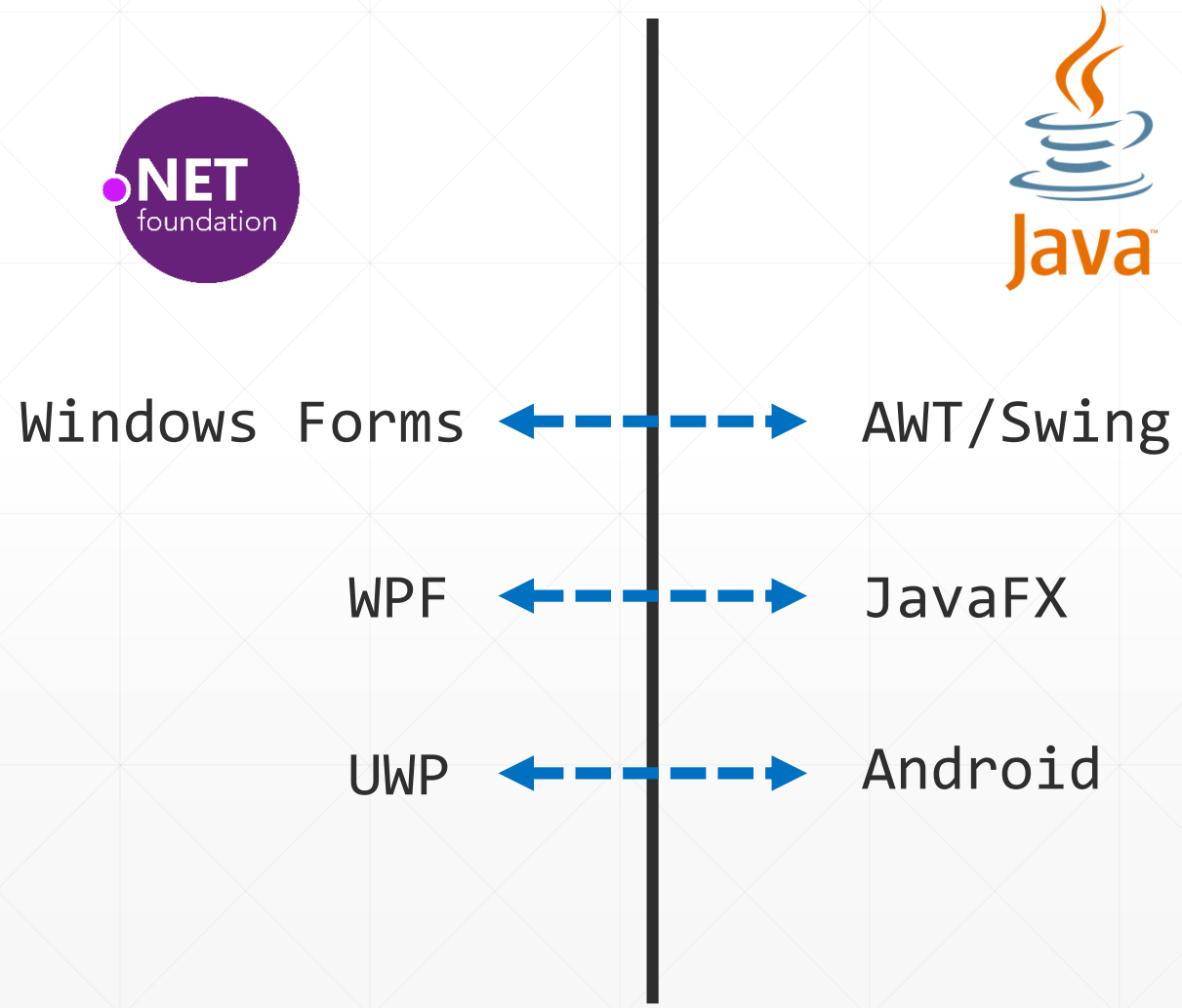
WPF: Windows Presentation Foundation

一个基于.NET的Windows桌面应用开发框架

Windows Forms vs. WPF

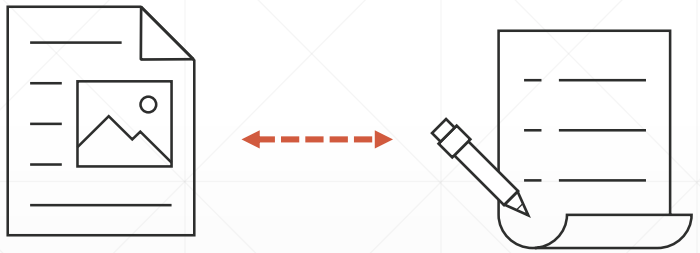
	Windows Forms	WPF
优点	开发效率高，易学易用	功能强大，极强的UI可定制性，较丰富的多媒体支持
缺点	UI要定制比较困难，缺少对于图形和多媒体方面的支持	学习曲线陡峭，编程模型过于复杂
适用场景	使用标准控件构建的小规模的Windows桌面应用	强调用户独特体验的、功能集合比较庞大的需要长期维护的桌面应用

.NET桌面应用技术与Java相关技术间的对应关系



WPF采用声明式编程风格

WPF使用XAML定义UI界面，这是一种“声明式的编程风格（declarative program style）”，使用C#来编写事件响应等交互逻辑。



.Xaml　　.cs

设计界面　　UI交互逻辑

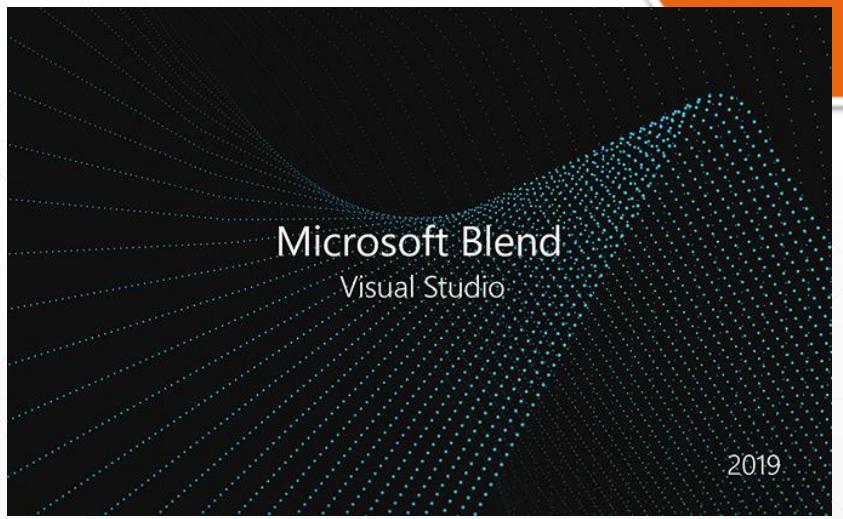
eXtensible Application Markup Language
（扩展应用程序标记语言）

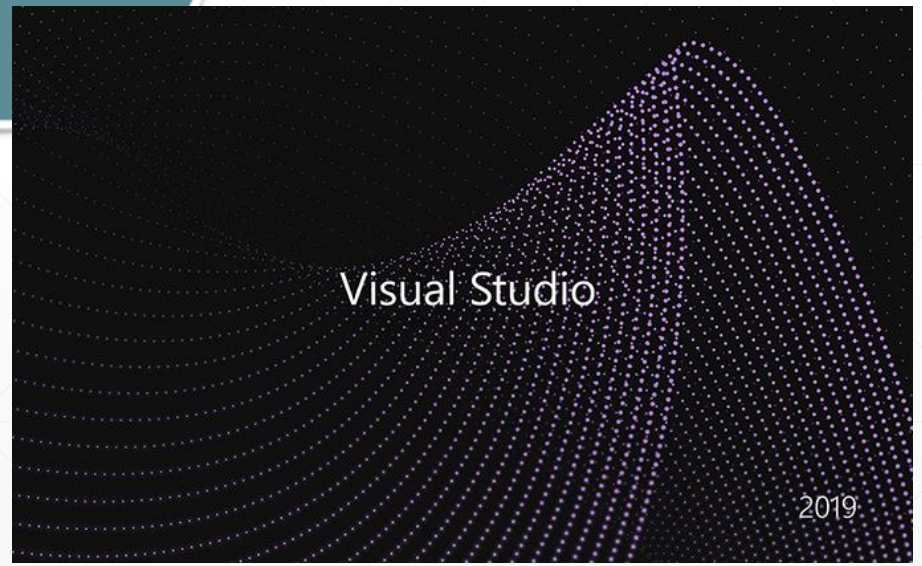
虽然WPF中的XAML与UWP和Xamarin中所使用的UI设计语言看上很类似，但并不完全一样，它们可以看成是“XAML的不同方言”。

“UI设计”与“功能实现”的分离

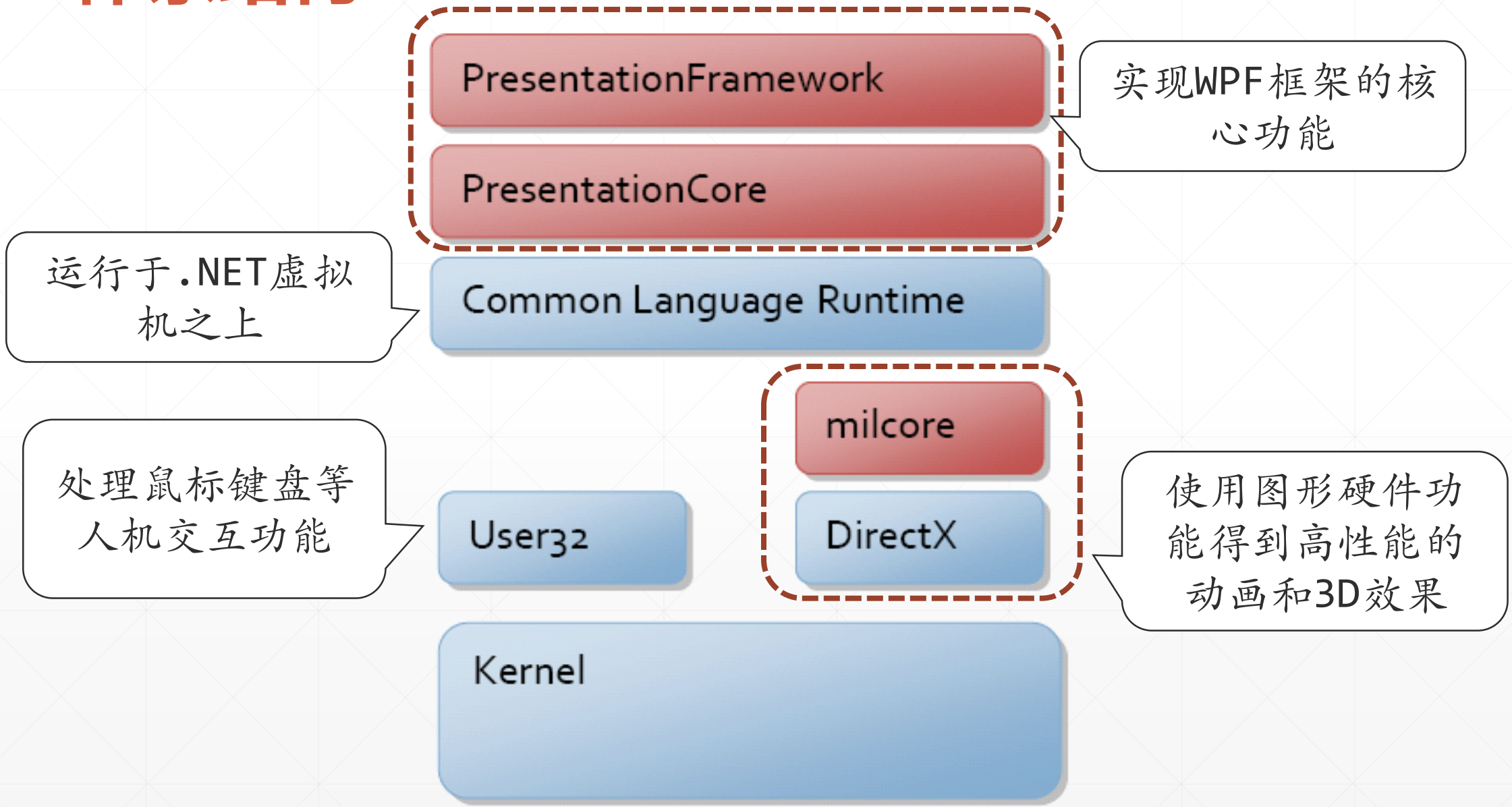
UI设计师

程序员

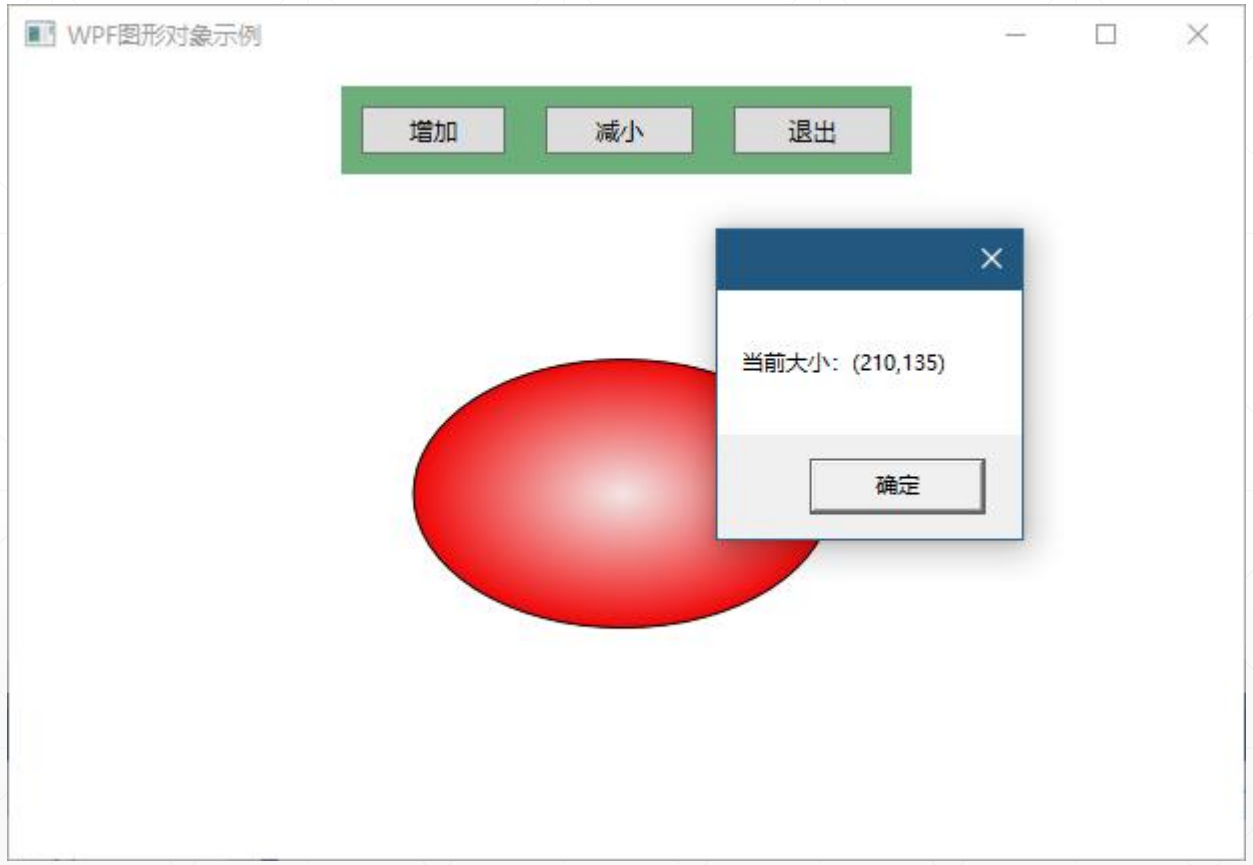




WPF体系结构


PresentationFramework
PresentationCore
实现WPF框架的核心功能
运行于.NET虚拟机之上
Common Language Runtime
milcore
DirectX
使用图形硬件功能得到高性能的动画和3D效果
处理鼠标键盘等人机交互功能
User32
Kernel

WPF二维矢量图形功能示例

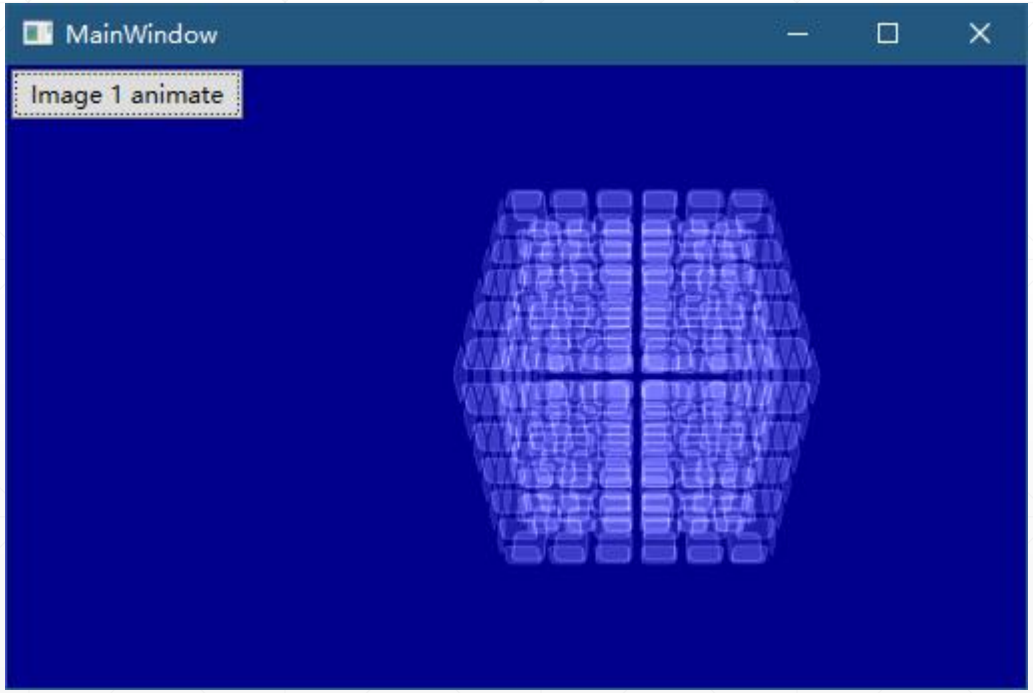


WPF内置了一些矢量图形对象，这些对象不仅能实现平滑的重绘，还能响应鼠标和键盘，从而为开发各种数据可视化程序提供了方便。

示例：WpfShapeDemos

WPF 3D 功能示例

由于WPF可以使用Windows内置的DirectX组件，因此，WPF框架提供了Viewport3D、MeshGeometry3D、Storyboard等组件，WPF应用程序可以用它们实现平滑的三维场景动画。

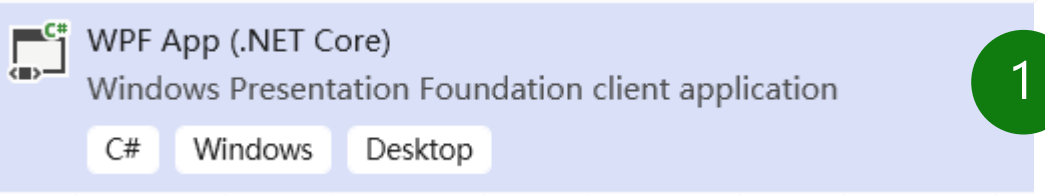


示例项目：WpfCubeAnimation

Hello World, WPF!

创建WPF项目

这两个是最常用的WPF项目类型



1 用于构建一个基于.NET Core的WPF应用。

2 用于放置WPF用户控件，用户控件通常是组合现有控件而得到的。

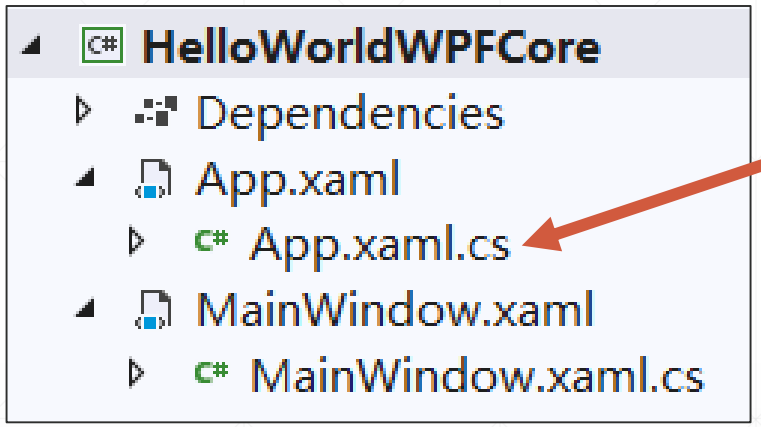
3 用于放置WPF自定义控件，自定义控件通常是通过自定义控件模板而得到的。

WPF项目模板分为“.NET Framework”和“.NET Core”两套，建议新项目采用以“.NET Core”为目标框架的项目模板。


Configure your new project
WPF App (.NET Core)
C#
Windows
Desktop
Project name
HelloWorldWPFCore
Location
I:\
...
Solution name
WPFCoreDemos
Place solution and project in the same directory
WPF App (.NET Core)
Windows Presentation Foundation client application
C#
Windows
Desktop
Back
Create
使用这个模板，创建一个WPF示例项目

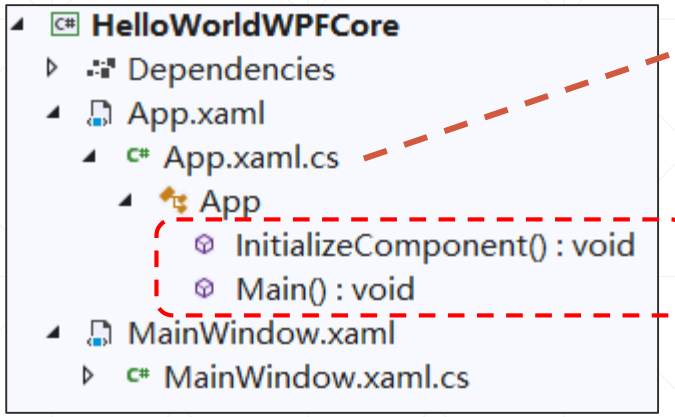
WPF项目的基础构成

WPF使用Xaml来定义UI界面，放在“.xaml”文件中。



每个“.xaml”关联着一个同名的“.cs”文件，用于放置程序员手写的C#代码，实现各种功能。

“.xaml”遵循XML相关规范，是一个纯文本文件



App.xmal.cs

```
namespace HelloWorldWPFCore
{
    /// <summary>
    /// Interaction logic for App.xaml
    /// </summary>
    3 references
    public partial class App : Application
    {
    }
}
```

App.g.i.cs

```
namespace HelloWorldWPFCore {

    /// <summary> App
    3 references
    public partial class App : System.Windows.Application {

        /// <summary> InitializeComponent
        [System.Diagnostics.DebuggerNonUserCodeAttribute()]
        [System.CodeDom.Compiler.GeneratedCodeAttribute("Presentatior
        1 reference
        public void InitializeComponent() ...

        /// <summary> Application Entry Point.
        [System.STAThreadAttribute()]
        [System.Diagnostics.DebuggerNonUserCodeAttribute()]
        [System.CodeDom.Compiler.GeneratedCodeAttribute("Presentatior
        0 references
        public static void Main() {
            HelloWorldWPFCore.App app = new HelloWorldWPFCore.App();
            app.InitializeComponent();
            app.Run();
        }
    }
}
```

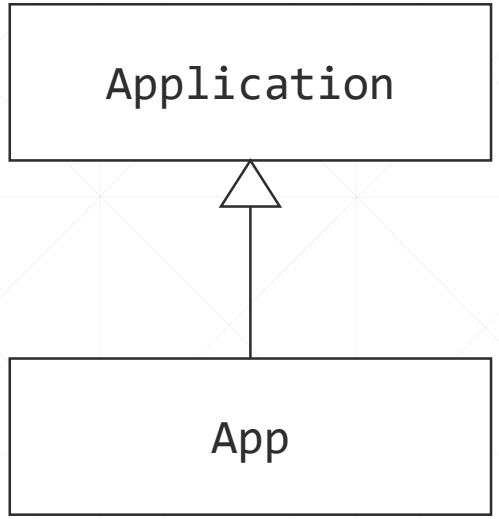
WPF项目构建工具会依据Xaml文件的内容，动态地创建一个.g.i.cs文件参与编译，这些文件是由工具生成的，不要手工更改。

程序的入口点

App.xaml

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

    </Application.Resources>
</Application>
```



- WPF程序的入口点放在App.xaml中，里面放置了一个Application元素，其x:Class属性指定了一个App类，此类派生于.NET基类库中的System.Windows.Application类，用于管理WPF应用程序的生命周期。

- Application元素的StartupUri属性指定了WPF程序的主窗体，Application类的MainWindow属性引用它。

编译器“玩”的把戏

编译器会依据App.xaml内容自动创建一个分部类App，放到了App.g.i.cs文件中，里面就定义了两个方法：

App.g.i.cs

```
3 references
public partial class App : System.Windows.Application {
    1 reference
    public void InitializeComponent() {
        this.StartupUri = new System.Uri("MainWindow.xaml",
            System.UriKind.Relative);
    }
    0 references
    public static void Main() {
        HelloWorldWPFCore.App app = new HelloWorldWPFCore.App();
        app.InitializeComponent();
        app.Run();
    }
}
```

Main()方法是WPF程序的入口点

WPF应用主窗体

WPF窗体是一个派生自Window的自定义类，包容多个WPF控件，以及相应的C#交互逻辑代码。

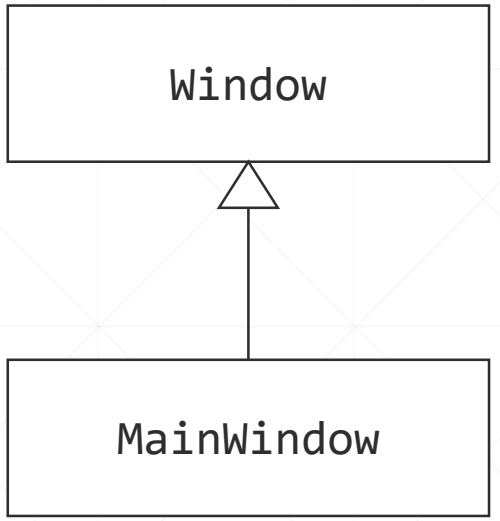
x:Class属性，将MainWindow类与<Window>元素关联了起来。

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="450" Width="800">
    <Grid>

    </Grid>
</Window>
```

这些属性值，将成为MainWindows对象的属性值，指定窗体的标题和大小。



当WPF程序运行时，看到的窗体是MainWindow类的实例。

编译器玩的花样

你在这个类中写代码……

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

MainWindow.xaml.cs

这个文件由编译器自动生成，其中LoadComponent()方法负责实例化窗体中的所有WPF控件。

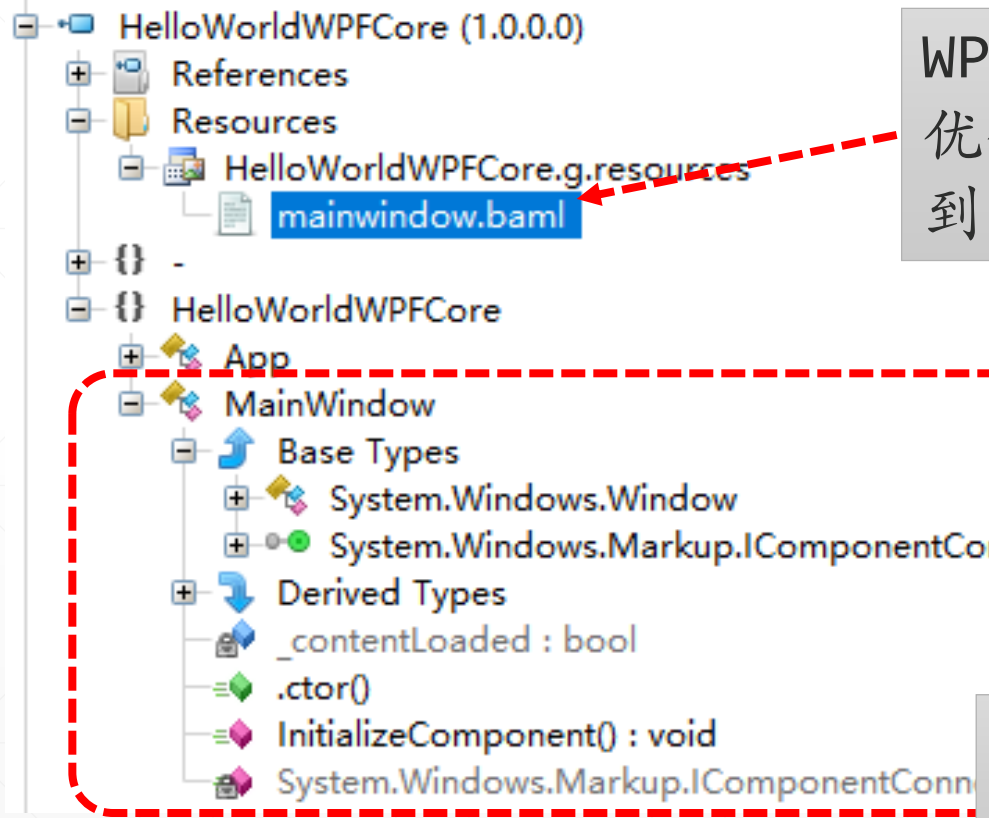
```
2 references
public partial class MainWindow : System.Windows.Window,
    System.Windows.Markup.IComponentConnector {

    private bool _contentLoaded;
    1 reference
    public void InitializeComponent() {
        if (_contentLoaded) {
            return;
        }
        _contentLoaded = true;
        Uri resourceLocater =
            new Uri("/HelloWorldWPFCore;component/mainwindow.xaml",
            UriKind.Relative);

        System.Windows.Application.LoadComponent(this, resourceLocater);
    }
    0 references
    void IComponentConnector.Connect(int connectionId, object target) {
        this._contentLoaded = true;
    }
}
```

MainWindow.g.i.cs

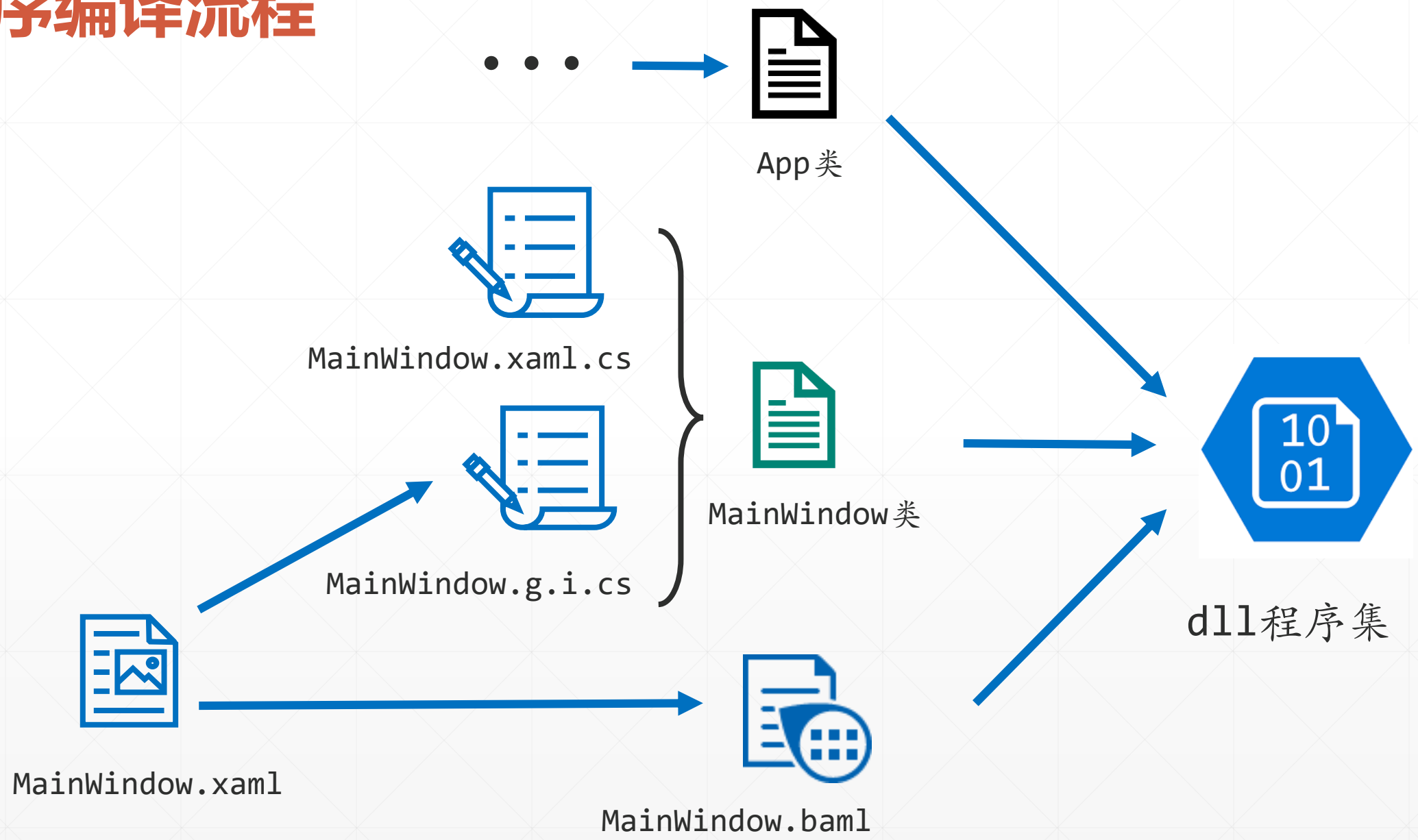
使用ILSpy工具查看编译器最终生成的程序集

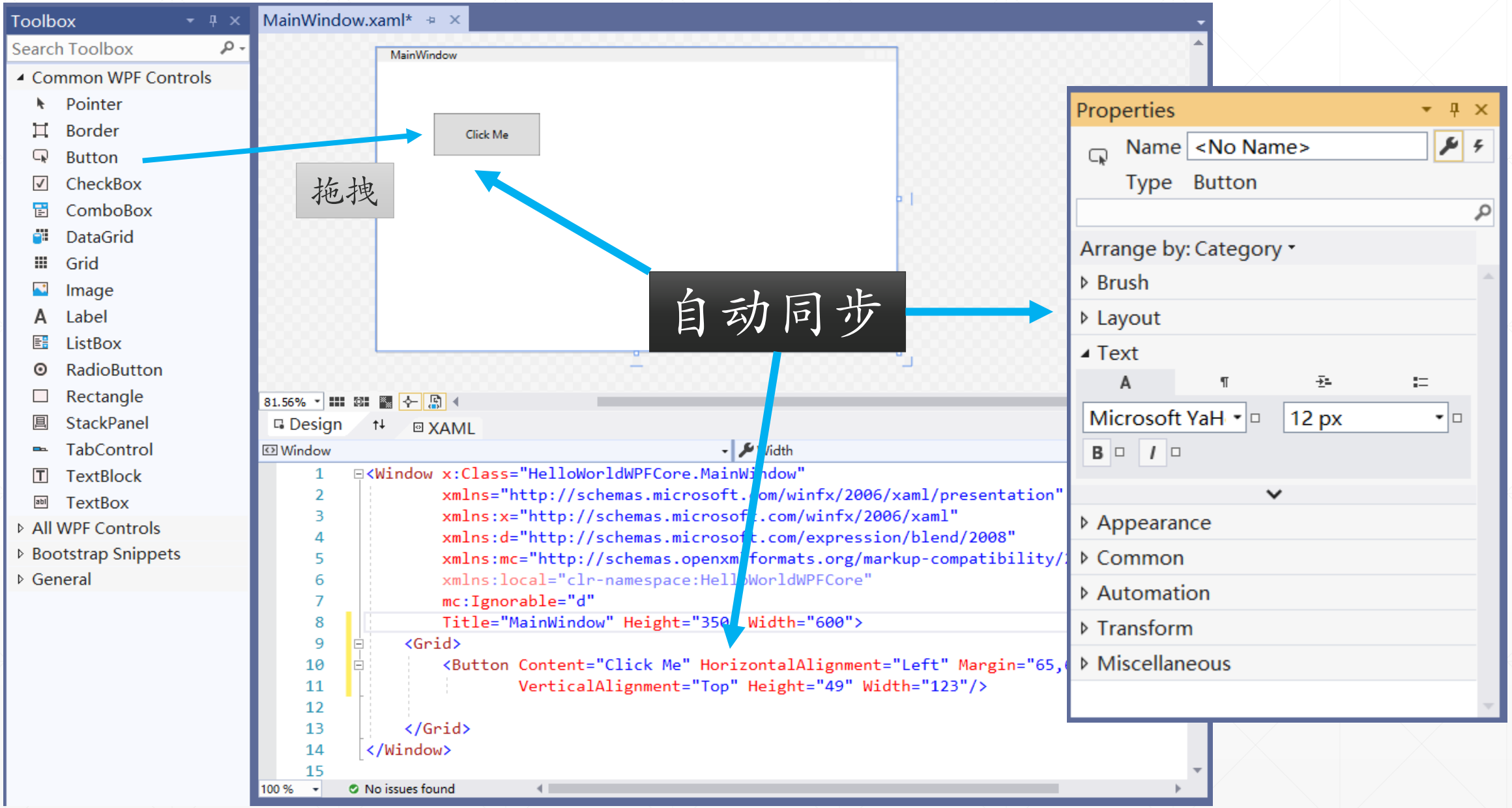


WPF应用程序编译过程中，会将XAML文件压缩与优化转换为二进制的BAML文件，并作为资源嵌入到最终的程序集中。

MainWindow.cs + MainWindow.g.i.cs
合并编译的结果

WPF程序编译流程


App类
MainWindow.xaml.cs
MainWindow.g.i.cs
MainWindow类
MainWindow.xaml
MainWindow.baml
dll程序集


Toolbox
Search Toolbox
Common WPF Controls
Pointer
Border
Button
CheckBox
ComboBox
DataGrid
Grid
Image
Label
ListBox
RadioButton
Rectangle
StackPanel
TabControl
TextBlock
TextBox
All WPF Controls
Bootstrap Snippets
General
MainWindow.xaml*
MainWindow
Click Me
拖拽
自动同步
81.56%
Design
XAML
Window
Width
<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/
xmlns:local="clr-namespace:HelloWorldWPFCore"
mc:Ignorable="d"
Title="MainWindow" Height="350" Width="600">
<Grid>
<Button Content="Click Me" HorizontalAlignment="Left" Margin="65,
VerticalAlignment="Top" Height="49" Width="123"/>
</Grid>
</Window>
100 %
No issues found
Properties
Name <No Name>
Type Button
Arrange by: Category
Brush
Layout
Text
Microsoft YaH
12 px
Appearance
Common
Automation
Transform
Miscellaneous

编码访问XAML元素（控件）对象

要在代码中访问XAML元素，必须给它们命名。

```
<Button x:Name="btnClick" ...  />
```

之后，在C#代码中即可通过“btnClick”名字来访问此按钮对象。

控件名字“背后的故事”

给Button控件一个“btnClick”名字之后，这将导致编译器生成的MainWindow类中自动添加一个私有字段(放在MainWindow.g.cs中)：

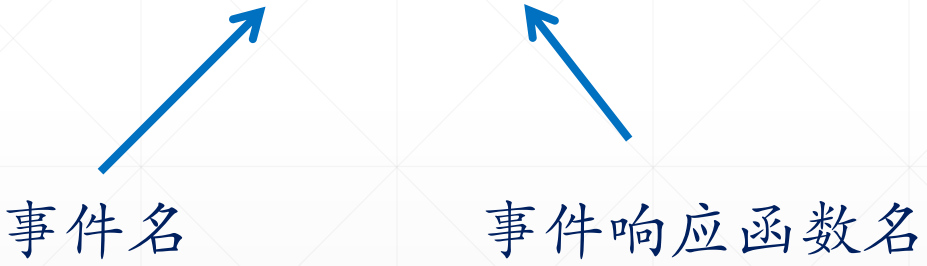
```
internal System.Windows.Controls.Button btnClick;
```

如果不指定特定的名字，编译器也会按照它自己的规则为每个控件生成一个默认的名字。

给XAML元素挂接事件响应代码

最简单的方法是直接设定XAML元素的事件属性

```
<Button x:Name="btnClick" Click="BtnClick_Click" ……>
```

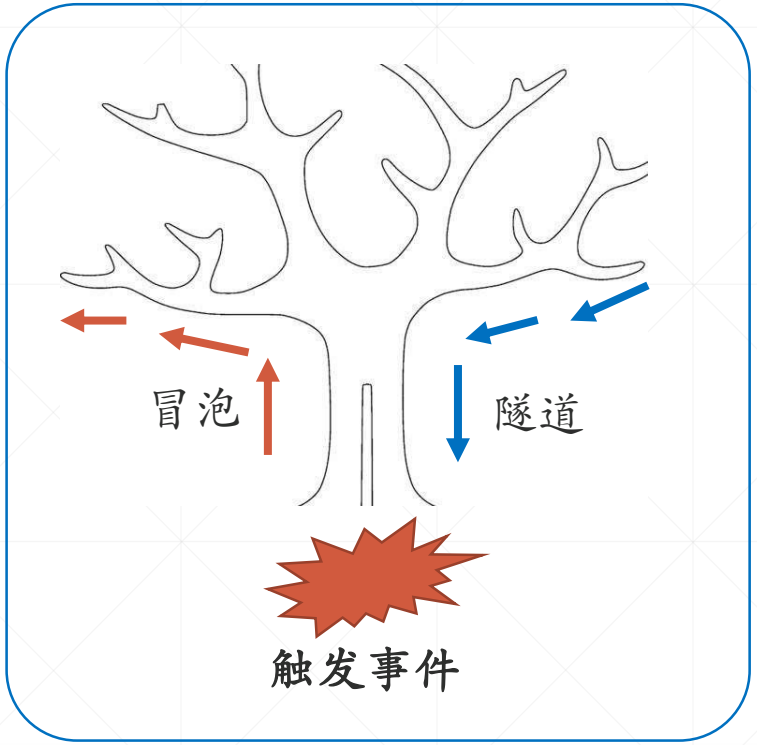


使用代码挂接事件

创建元素所对应的对象实例，使用委托挂接事件

```
var btnClick = new Button();
btnClick.Click += new RoutedEventHandler(BtnClick_Click);
```

WPF独有的路由事件，是Windows Form程序中不具备的特性



事件在控件树中的传递

如果是使用前面介绍的事件声明方式，则WPF构建工具会在后台生成的“.g.i.cs”文件中加上类似于上面的示例代码，你可以自己去找一下……

示例程序中的事件驱动代码

```
<Button x:Name="btnClick"
        Click="BtnClick_Click"
        Content="Click Me" HorizontalAlignment="Left"
        Margin="65,60,0,0"
        VerticalAlignment="Top" Height="49" Width="123"/>
```

使用x:Name来标识控件，Click属性用于指定单击事件响应代码

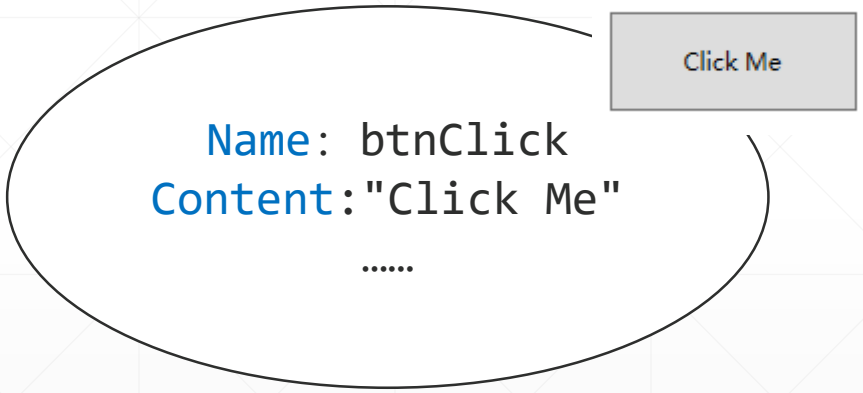
```
public partial class MainWindow : Window
{
    0 references
    public MainWindow()
    {
        InitializeComponent();
    }
    //按钮单击事件响应代码
    0 references
    private void BtnClick_Click(object sender, RoutedEventArgs e)
    {
        MessageBox.Show("Hello,WPF!");
    }
}
```

这个方法由WPF构建工具自动生成，里面有代码完成Click事件响应函数的挂接

XAML元素与WPF控件对象

在XAML中定义一个元素，在程序运行时，相当于实例化一个对应的WPF对象，XAML元素中的属性，对应着相应类所定义的（依赖）属性。

```
<Button
    x:Name="btnClick"
    Content="Click Me"
     ... >
```



XAML元素

Button实例

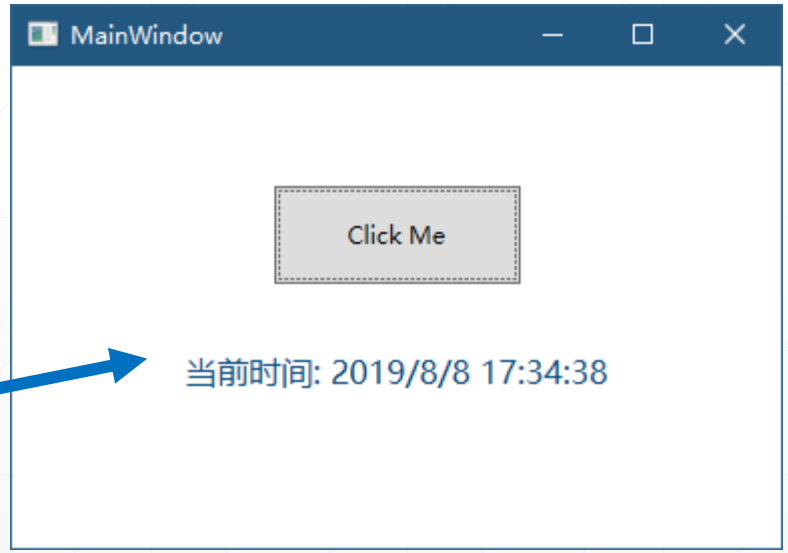
XAML和代码是相互对应的，可以用纯代码编程实现所有XAML所实现的功能，反之则不行。

向示例中添加一个新控件

```
<Label x:Name="lblInfo" Content="当前时间: 2019/1/1 00:00:00"
       HorizontalAlignment="Center" Margin="0,138,0,0"
       VerticalAlignment="Top" FontSize="16"
       Foreground="#FF074C82"/>
```

Label对象的名字为“lblInfo”

```
public partial class MainWindow : Window
{
    0 references
    public MainWindow()
    {
        InitializeComponent();
    }
    //按钮单击事件响应代码
    1 reference
    private void BtnClick_Click(object sender, RoutedEventArgs e)
    {
        //MessageBox.Show("Hello,WPF!");
        //通过控件名字访问控件属性，调用控件方法
        lblInfo.Content = $"当前时间: {DateTime.Now}";
    }
}
```



对标签控件Content依赖属性值的修改，会引发UI的重绘。

小结

- WPF项目中的每个XAML文档其实都代表了一个类，文档中的元素则成为这个类的属性。
- WPF中的窗体是基类库中Window类的子类，其界面由XAML文档所定义，XAML文档中声明的各个控件将成为 Window子类的成员，在InitializeComponent()方法中被实例化，此函数的实现由编译器提供，在构造函数中被调用。
- WPF窗体的XAML文件在编译时会被转换为“.baml”文件，作为资源被嵌入到程序集中。
- 与Windows Forms类似，WPF项目同样应用了C#的“分部类（partial）”语法，将程序员手写的C#代码放在以.cs为扩展名的代码文件中，而将编译器生成的代码放到以“.g.i.cs”为扩展名的文件中，编译时，将这两个文件合并起来编译。
- 与Windows Forms一样，WPF项目通过名字访问控件，同样采用“事件驱动”的程序运行模式，但编程模型与编译流程要比Windows Forms复杂。