Contents

Windows Presentation Foundation

2022/2/12 •

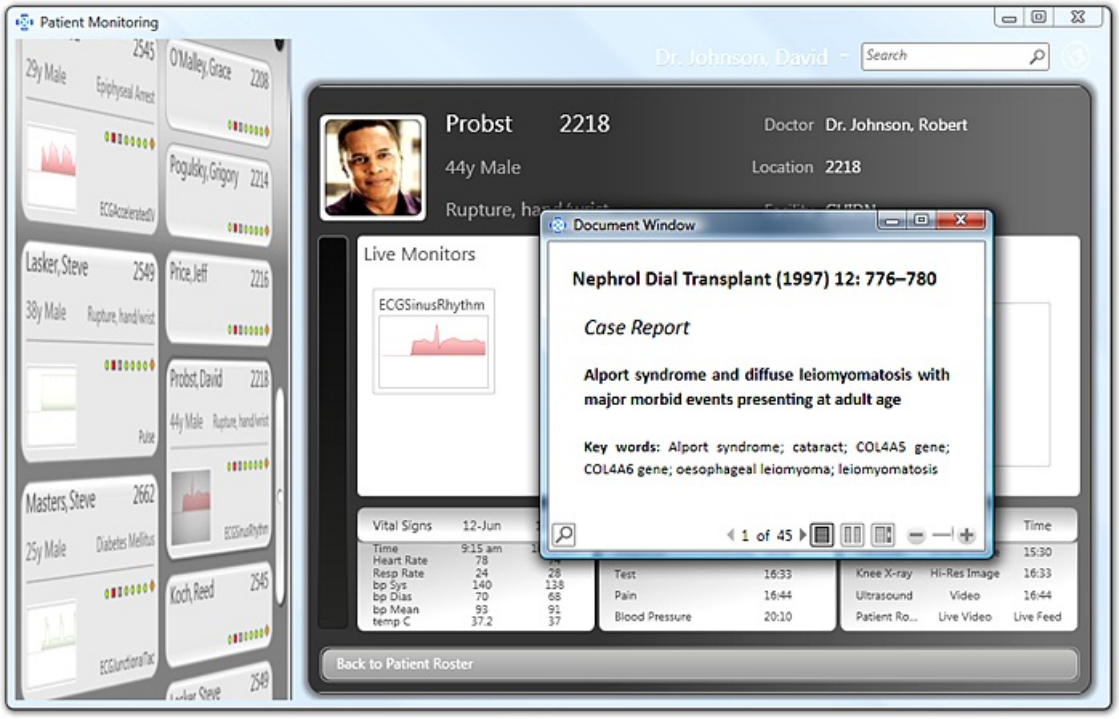
Windows Presentation Foundation (WPF) 向开发人员提供了一个统一的编程模型，可用于在 Windows 上生成业务线桌面应用程序。

- WPF 介绍
- 入门
- 应用程序开发
- 高级
- 控件
- 数据
- 图形和多媒体
- 安全
- WPF 示例
- 类库

WPF 概述

2022/2/12 •

使用 Windows Presentation Foundation (WPF)，你可以创建适用于 Windows 且具有非凡视觉效果的桌面客户端应用程序。



WPF 的核心是一个与分辨率无关且基于矢量的呈现引擎，旨在充分利用现代图形硬件。WPF 通过一套完善的应用程序开发功能对该核心进行了扩展，这些功能包括可扩展应用程序标记语言 (XAML)、控件、数据绑定、布局、二维和三维图形、动画、样式、模板、文档、媒体、文本和版式。WPF 属于 .NET，因此可以生成整合 .NET API 其他元素的应用程序。

本概述适用于新用户，介绍了 WPF 的主要功能和概念。

使用 WPF 进行编程

WPF 作为大部分位于 System.Windows 命名空间中的 .NET 类型的一个子集存在。如果你之前使用托管技术（如 ASP.NET 和 Windows 窗体）通过 .NET 生成过应用程序，则不会对基本的 WPF 编程体验感到陌生；你可以使用最喜欢的 .NET 编程语言（如 C# 或 Visual Basic）来完成实例化类、设置属性、调用方法以及处理事件等操作。

WPF 还包括增强属性和事件的其他编程构造：依赖项属性 和 路由事件。

标记和代码隐藏

通过 WPF，可以使用标记和代码隐藏开发应用程序，这是 ASP.NET 开发人员已经熟悉的体验。通常使用 XAML 标记实现应用程序的外观，同时使用托管编程语言（代码隐藏）来实现其行为。这种外观和行为的分离具有以下优点：

- 降低了开发和维护成本，因为特定于外观的标记与特定于行为的代码不紧密耦合。
- 开发效率更高，因为设计人员在实现应用程序外观的同时，开发人员可以实现应用程序的行为。
- WPF 应用程序的全球化和本地化 得以简化。

标记

XAML 是一种基于 XML 的标记语言，以声明形式实现应用程序的外观。通常用它创建窗口、对话框、页和用户控件，并填充控件、形状和图形。

下面的示例使用 XAML 来实现包含一个按钮的窗口的外观：

```
<Window
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    Title="Window with Button"
    Width="250" Height="100">

  <!-- Add button to window -->
  <Button Name="button">Click Me!</Button>

</Window>
```

具体而言，此 XAML 通过分别使用 `Window` 和 `Button` 元素来定义窗口和按钮。每个元素均配置了特性（如 `Window` 元素的 `Title` 特性）来指定窗口的标题栏文本。在运行时，WPF 会将标记中定义的元素和特性转换为 WPF 类的实例。例如，`Window` 元素被转换为 Window 类的实例，该类的 Title 属性是 `Title` 特性的值。

下图显示上一个示例中的 XAML 定义的用户界面 (UI)：

由于 XAML 是基于 XML 的，因此使用它编写的 UI 汇集在嵌套元素的层次结构中，称为 元素树。元素树提供了一种直观的逻辑方式来创建和管理 UI。

代码隐藏

应用程序的主要行为是实现响应用户交互的功能，包括处理事件（例如，单击菜单、工具栏或按钮）以及相应地调用业务逻辑和数据访问逻辑。在 WPF 中，在与标记相关联的代码中实现此行为。此类代码称为代码隐藏。下面的示例演示上一个示例的更新标记和代码隐藏：

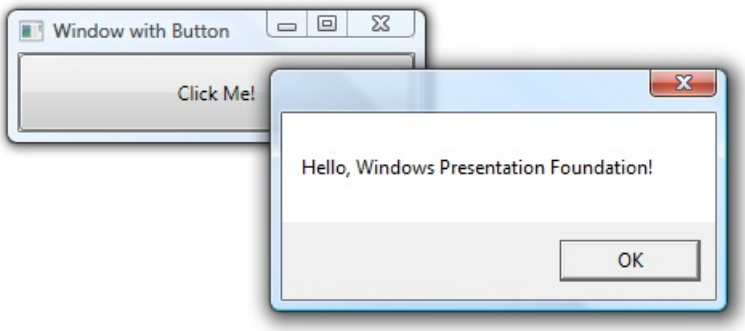
```
<Window
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="SDKSample.AWindow"
    Title="Window with Button"
    Width="250" Height="100">

  <!-- Add button to window -->
  <Button Name="button" Click="button_Click">Click Me!</Button>

</Window>
```

```
using System.Windows; // Window, RoutedEventArgs, MessageBox

namespace SDKSample
{
    public partial class AWindow : Window
    {
        public AWindow()
        {
            // InitializeComponent call is required to merge the UI
            // that is defined in markup with this class, including
            // setting properties and registering event handlers
            InitializeComponent();
        }

        void button_Click(object sender, RoutedEventArgs e)
        {
            // Show message box when button is clicked.
            MessageBox.Show("Hello, Windows Presentation Foundation!");
        }
    }
}
```

```
Namespace SDKSample

    Partial Public Class AWindow
        Inherits System.Windows.Window

        Public Sub New()

            ' InitializeComponent call is required to merge the UI
            ' that is defined in markup with this class, including
            ' setting properties and registering event handlers
            InitializeComponent()

        End Sub

        Private Sub button_Click(ByVal sender As Object, ByVal e As RoutedEventArgs)

            ' Show message box when button is clicked.
            MessageBox.Show("Hello, Windows Presentation Foundation!")

        End Sub

    End Class

End Namespace
```

在此示例中，代码隐藏实现派生自 Window 类的类。`x:Class` 特性用于将标记与代码隐藏类相关联。从代码隐藏类的构造函数调用 `InitializeComponent`，以将标记中定义的 UI 与代码隐藏类合并在一起。`InitializeComponent` 生成应用程序时，将为你生成 (，这就是你不需要手动实现它的原因。) 的组合 `x:Class` 并 `InitializeComponent` 确保你的实现在创建时正确初始化。代码隐藏类还可实现按钮的 Click 事件的事件处理程序。单击该按钮后，事件处理程序会通过调用 System.Windows.MessageBox.Show 方法显示一个消息框。

下图显示单击该按钮后的结果：



控制

应用程序模型带来的用户体验是构造的控件。在 WPF 中，*控件* 是一个涵盖性术语，适用于在窗口或页中承载的 WPF 类的类别、具有用户界面并实现某些行为。

有关详细信息，请参阅 控件。

按功能分类的 WPF 控件

下面列出了内置的 WPF 控件：

- **按钮**：Button 和 RepeatButton。
- **数据显示**：DataGrid、ListView 和 TreeView。
- **日期显示和选项**：Calendar 和 DatePicker。
- **对话框**：OpenFileDialog、PrintDialog和 SaveFileDialog。
- **数字墨迹**：InkCanvas 和 InkPresenter。
- **文档**：DocumentViewer、FlowDocumentPageViewer、FlowDocumentReader、FlowDocumentScrollViewer和 StickyNoteControl。
- **输入**：TextBox、RichTextBox和 PasswordBox。
- **布局**：Border、BulletDecorator、Canvas、DockPanel、Expander、Grid、GridView、GridSplitter、GroupBox、Panel、ResizeGrip、Separator、ScrollBar、ScrollViewer、StackPanel、Thumb、Viewbox、VirtualizingStackPanel、Window和 WrapPanel。
- **媒体**：Image、MediaElement和 SoundPlayerAction。
- **菜单**：ContextMenu、Menu和 ToolBar。
- **导航**：Frame、Hyperlink、Page、NavigationWindow和 TabControl。
- **选项**：CheckBox、ComboBox、ListBox、RadioButton和 Slider。
- **用户信息**：AccessText、Label、Popup、ProgressBar、StatusBar、TextBlock和 ToolTip。

输入和命令

最常检测和响应用户输入的控件。WPF 输入系统使用直接事件和路由事件来支持文本输入、焦点管理和鼠标定位。

应用程序通常具有复杂的输入要求。WPF 提供了 命令系统，用于将用户输入操作与对这些操作做出响应的代码分隔开来。

Layout

创建用户界面时，按照位置和大小排列控件以形成布局。任何布局的一项关键要求都是适应窗口大小和显示设

置的变化。WPF 为你提供一流的可扩展布局系统，而不强制你编写代码以适应这些情况下的布局。

布局系统的基础是相对定位，这提高了适应不断变化的窗口和显示条件的能力。此外，该布局系统还可管理控件之间的协商以确定布局。协商是一个两步过程：首先，控件将需要的位置和大小告知父级；其次，父级将控件可以有的空间告知控件。

该布局系统通过基 WPF 类公开给子控件。对于通用的布局（如网格、堆叠和停靠），WPF 包括若干布局控件：

- Canvas：子控件提供其自己的布局。
- DockPanel：子控件与面板的边缘对齐。
- Grid：子控件由行和列定位。
- StackPanel：子控件垂直或水平堆叠。
- VirtualizingStackPanel：子控件在水平或垂直的行上虚拟化并排列。
- WrapPanel：子控件按从左到右的顺序定位，在当前行上的控件超出允许的空间时，换行到下一行。

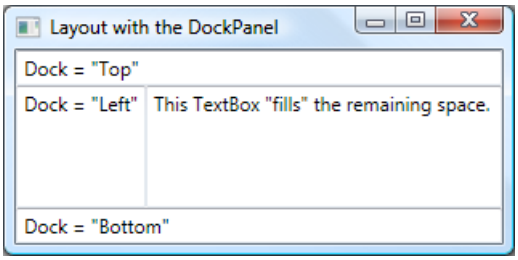
下面的示例使用 DockPanel 布置几个 TextBox 控件：

```
<Window
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="SDKSample.LayoutWindow"
    Title="Layout with the DockPanel" Height="143" Width="319">

  <!--DockPanel to layout four text boxes-->
  <DockPanel>
    <TextBox DockPanel.Dock="Top">Dock = "Top"</TextBox>
    <TextBox DockPanel.Dock="Bottom">Dock = "Bottom"</TextBox>
    <TextBox DockPanel.Dock="Left">Dock = "Left"</TextBox>
    <TextBox Background="White">This TextBox "fills" the remaining space.</TextBox>
  </DockPanel>

</Window>
```

DockPanel 允许子 TextBox 控件，以告诉它如何排列这些控件。为了完成此操作，DockPanel 实现 `Dock` 附加了属性，该属性公开给子控件，以允许每个子控件指定停靠样式。

> **NOTE**
> 由父控件实现以便子控件使用的属性是 WPF 构造，称为附加属性。

下图显示上一个示例中的 XAML 标记的结果：：



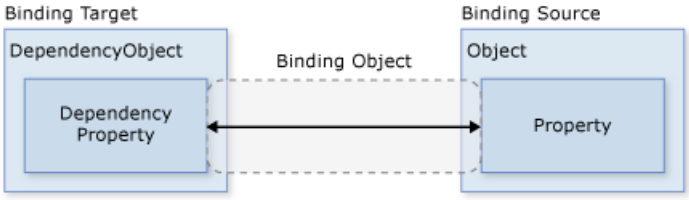
数据绑定

大多数应用程序旨在为用户提供查看和编辑数据的方法。对于 WPF 应用程序，已对存储和访问数据的工作提供技术（如 SQL Server 和 ADO.NET）。访问数据并将数据加载到应用程序的托管对象后，WPF 应用程序的复杂工作开始。从根本上来说，这涉及到两件事：

1. 将数据从托管对象复制到控件，在控件中可以显示和编辑数据。

2. 确保使用控件对数据所做的更改将复制回托管对象。

为了简化应用程序开发，WPF 提供了一个数据绑定引擎来自动执行这些步骤。数据绑定引擎的核心单元是 Binding 类，其工作是将控件（绑定目标）绑定到数据对象（绑定源）。下图阐释了这种关系：



下一示例演示如何将 TextBox 绑定到自定义 `Person` 对象的实例。下面的代码演示了 `Person` 实现：

```
Namespace SDKSample

    Class Person

        Private _name As String = "No Name"

        Public Property Name() As String
            Get
                Return _name
            End Get
            Set(ByVal value As String)
                _name = value
            End Set
        End Property

    End Class

End Namespace
```

```
namespace SDKSample
{
    class Person
    {
        string name = "No Name";

        public string Name
        {
            get { return name; }
            set { name = value; }
        }
    }
}
```

下面的标记将绑定 TextBox 到自定义对象的实例 `Person` ：

```
 <Window
     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
     x:Class="SDKSample.DataBindingWindow">

   <!-- Bind the TextBox to the data source (TextBox.Text to Person.Name) -->
   <TextBox Name="personNameTextBox" Text="{Binding Path=Name}" />

 </Window>
```

```
Imports System.Windows ' Window

Namespace SDKSample

    Partial Public Class DataBindingWindow
        Inherits Window

        Public Sub New()
            InitializeComponent()

            ' Create Person data source
            Dim person As Person = New Person()

            ' Make data source available for binding
            Me.DataContext = person

        End Sub

    End Class

End Namespace
```

```
using System.Windows; // Window

namespace SDKSample
{
    public partial class DataBindingWindow : Window
    {
        public DataBindingWindow()
        {
            InitializeComponent();

            // Create Person data source
            Person person = new Person();

            // Make data source available for binding
            this.DataContext = person;
        }
    }
}
```

在此示例中，`Person` 类在代码隐藏中实例化并被设置为 `DataBindingWindow` 的数据上下文。在标记中，Text 的 TextBox 属性被绑定至 `Person.Name` 属性（使用“`{Binding ... }`”XAML 语法）。此 XAML 告知 WPF 将 TextBox 控件绑定至窗口的 `Person` 属性中存储的 DataContext 对象。

WPF 数据绑定引擎提供了额外支持，包括验证、排序、筛选和分组。此外，数据绑定支持在标准 WPF 控件显示的用户界面不恰当时，使用数据模板来为数据绑定创建自定义的用户界面。

有关详细信息，请参阅数据绑定概述。

显卡

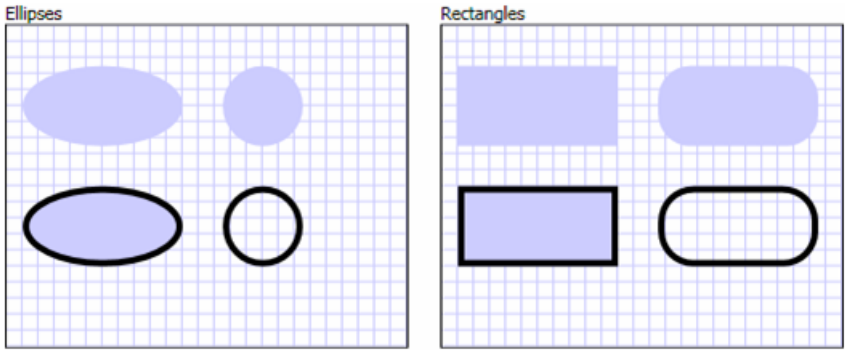
WPF 引入了一组广泛、可伸缩的灵活图形功能，具有以下优点：

- **图形与分辨率和设备均无关**。WPF 图形系统中的基本度量单位是与设备无关的像素（即 1/96 英寸），且不考虑实际屏幕分辨率，并为实现与分辨率和设备无关的呈现提供了基础。每个与设备无关的像素都会自动缩放，以匹配呈现它的系统的每英寸点数 (dpi) 设置。
- **精度更高**。WPF 坐标系统使用双精度浮点数字度量，而不是单精度数字。转换和不透明度值也表示为双精度数字。WPF 还支持广泛的颜色域 (scRGB)，并集成了对管理来自不同颜色空间的输入的支持。

- **高级图形和动画支持**。WPF 通过为你管理动画场景简化了图形编程，你无需担心场景处理、呈现循环和双线性内插。此外，WPF 还提供了点击测试支持和全面的 alpha 合成支持。
- **硬件加速**。WPF 图形系统充分利用图形硬件来尽量降低 CPU 使用率。

二维形状

WPF 提供一个常用矢量绘制的二维形状库，如下图中所示的矩形和椭圆：



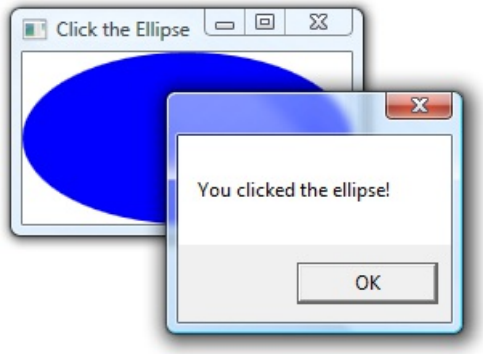
形状的一个有趣功能是它们不只是用于显示；形状实现许多你期望的控件功能，包括键盘和鼠标输入。下面的示例演示 MouseUp 正在处理的事件 Ellipse：

```
<Window
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="SDKSample.EllipseEventHandlingWindow"
    Title="Click the Ellipse">
    <Ellipse Name="clickableEllipse" Fill="Blue" MouseUp="clickableEllipse_MouseUp" />
</Window>
```

```
Imports System.Windows ' Window, MessageBox
Imports System.Windows.Input ' MouseButtonEventArgs

Namespace SDKSample

    Public Class EllipseEventHandlingWindow
        Inherits Window

        Public Sub New()
            InitializeComponent()
        End Sub

        Private Sub clickableEllipse_MouseUp(ByVal sender As Object, ByVal e As MouseButtonEventArgs)
            MessageBox.Show("You clicked the ellipse!")
        End Sub

    End Class

End Namespace
```

```
using System.Windows; // Window, MessageBox
using System.Windows.Input; // MouseButtonEventHandler

namespace SDKSample
{
    public partial class EllipseEventHandlingWindow : Window
    {
        public EllipseEventHandlingWindow()
        {
            InitializeComponent();
        }

        void clickableEllipse_MouseUp(object sender, MouseButtonEventArgs e)
        {
            // Display a message
            MessageBox.Show("You clicked the ellipse!");
        }
    }
}
```

下图显示了前面的代码生成的内容：



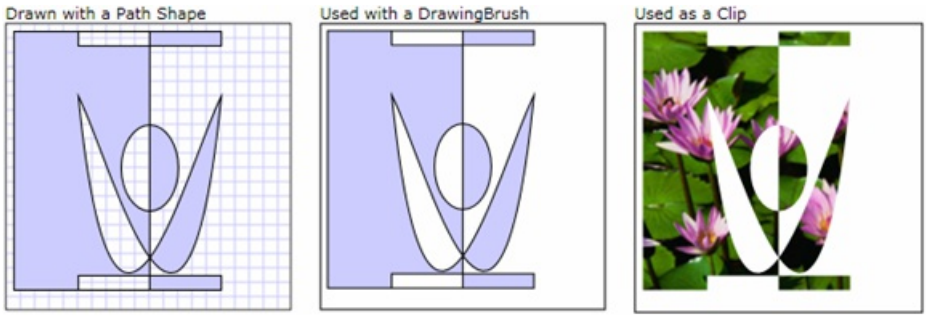
有关详细信息，请参阅 WPF 中的形状和基本绘图概述。

二维几何图形

WPF 提供的二维形状包含基本形状的标准集。但是，你可能需要创建自定义形状以帮助改进自定义用户界面的设计。为此，WPF 提供了几何图形。下图演示了使用几何图形来创建可直接绘制、用作画笔或用于剪辑其他形状和控件的自定义形状。

Path 对象可用于绘制封闭式或开放式形状、多个形状，甚至曲线形状。

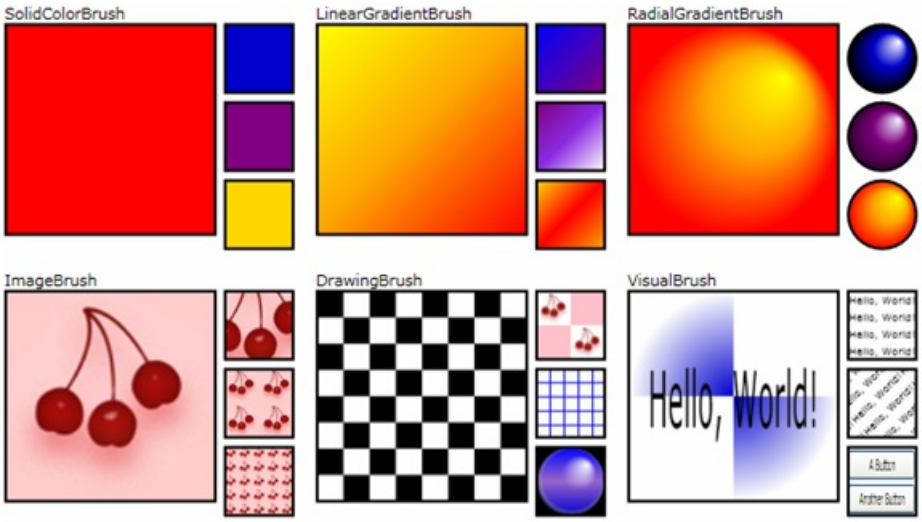
Geometry 对象可用于剪辑、命中测试以及呈现二维图形数据。



有关详细信息，请参阅几何图形概述。

二维效果

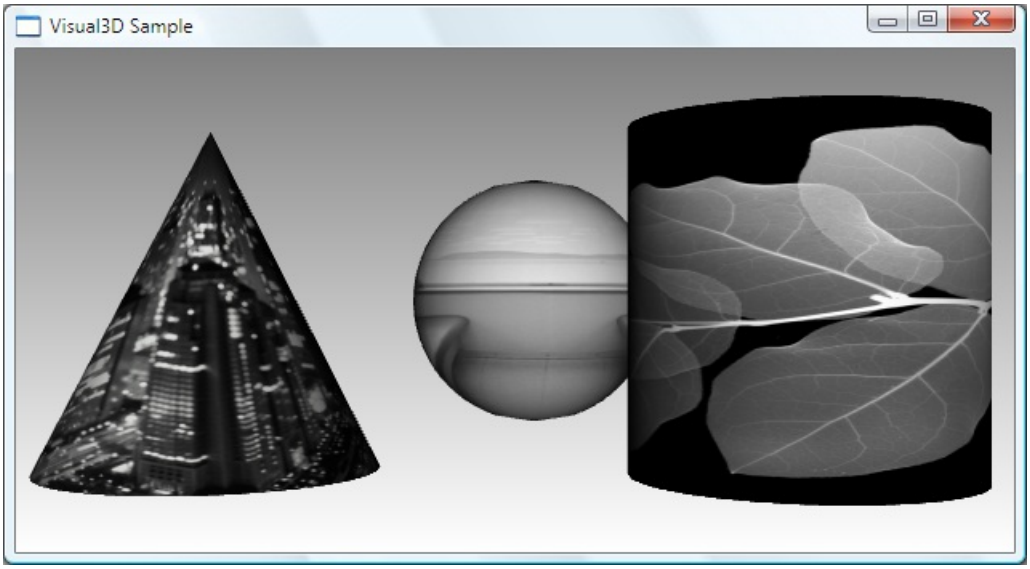
WPF 二维功能的子集包括视觉效果，如渐变、位图、绘图、用视频绘画、旋转、缩放和倾斜。这些都是通过画笔实现的;下图显示了一些示例：



有关详细信息，请参阅 WPF 画笔概述。

三维呈现

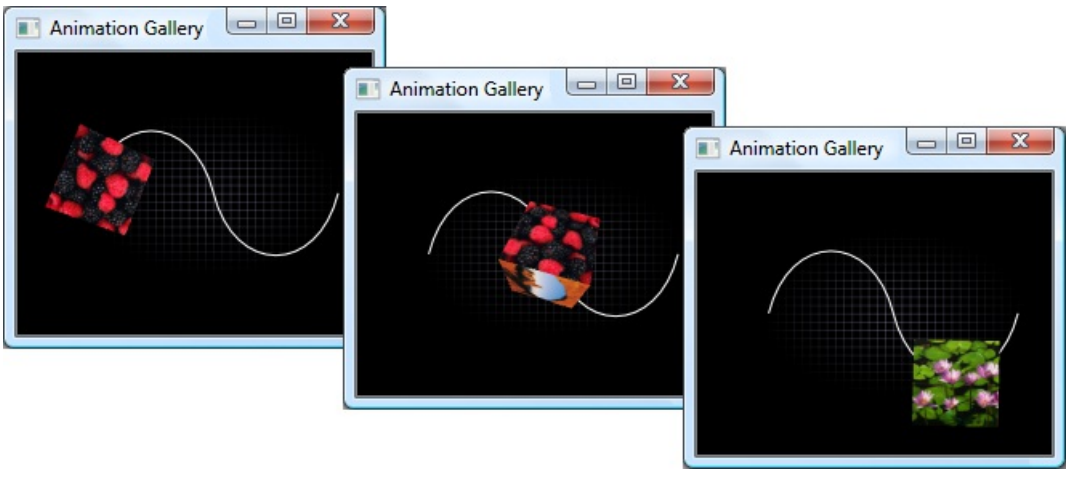
WPF 还包括三维呈现功能，这些功能与二维图形集成，以创建更精彩、更有趣的用户界面。例如，下图显示呈现在三维形状上的二维图像：



有关详细信息，请参阅三维图形概述。

动画

WPF 动画支持可以使控件变大、抖动、旋转和淡出，以形成有趣的页面过渡等。你可以对大多数 WPF 类，甚至自定义类进行动画处理。下图显示了运行中的一个简单动画：



有关详细信息，请参阅动画概述。

媒体

传达丰富内容的一种方法是使用视听媒体。WPF 为图像、视频和音频提供特殊支持。

映像

图像对大多数应用程序很常见，WPF 提供多种方式来使用它们。下图显示一个用户界面，该用户界面中的列表框中包含缩略图图像。选中一个缩略图后，将显示该图像的原尺寸。

有关详细信息，请参阅 图像处理概述。

视频和音频

MediaElement 控件能够播放视频和音频，并且其足够灵活，可以用作其他自定义媒体播放器的基础。以下 XAML 标记实现了媒体播放器：

```
<MediaElement
  Name="myMediaElement"
  Source="media/wpf.wmv"
  LoadedBehavior="Manual"
  Width="350" Height="250" />
```

下图中的窗口显示了 MediaElement 操作中的控件：

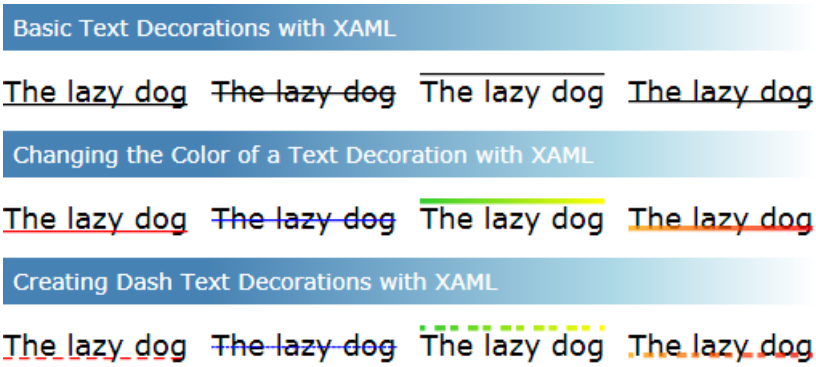
有关详细信息，请参阅 图形和多媒体。

文本和版式

为了促进高质量的文本呈现，WPF 提供以下功能：

- OpenType 字体支持。
- ClearType 增强功能。
- 利用硬件加速的高性能。
- 文本与媒体、图形和动画的集成。
- 国际字体支持和回退机制。

作为文本与图形集成的演示，下图显示了文本修饰的应用程序：



有关详细信息，请参阅 Windows Presentation Foundation 中的版式。

自定义 WPF 应用

到目前为止，你已经了解用于开发应用程序的核心 WPF 构建块。你可以使用该应用程序模型来托管和交付应用程序内容，它主要由控件组成。若要简化用户界面中控件的排列，并确保保持该排列能够应对窗口大小和显示设置的更改，你可以使用 WPF 布局系统。由于大多数应用程序允许用户与数据交互，因此你可以使用数据绑定来减少将用户界面与数据集成的工作。若要增强你应用程序的可视化外观，可以使用 WPF 提供的综合图形、动画和媒体支持。

不过，在创建和管理真正独特且视觉效果非凡的用户体验时，基础知识通常是不够的。标准的 WPF 控件可能无法与你所需的应用程序外观集成。数据可能不会以最有效的方式显示。你应用程序的整体用户体验可能不适合 Windows 主题的默认外观和感觉。在许多方面，演示技术需要视觉扩展性，需要的程度与任何其他类型的扩展性一样。

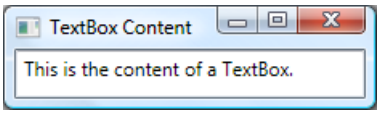
为此，WPF 提供了多种机制用于创建独特的用户体验，包括控件、触发器、控件和数据模板、样式、用户界面资源以及主题和皮肤的丰富内容模型。

内容模型

大多数 WPF 控件的主要用途是显示内容。在 WPF 中，可以构成控件内容的项的类型和数目称为控件的 *内容模型*。某些控件可以包含一种内容类型的一个项；例如， TextBox 的内容是分配给 Text 属性的一个字符串值。下面的示例设置的内容 TextBox ：

```
<Window
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="SDKSample.TextBoxContentWindow"
    Title="TextBox Content">

    <TextBox Text="This is the content of a TextBox." />
</Window>
```

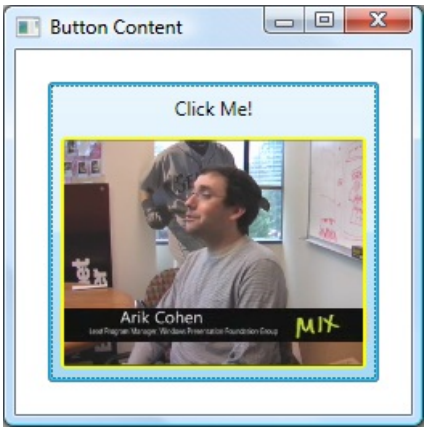
下图显示了结果：



但是，其他控件可以包含不同内容类型的多个项；Button的内容（由 Content 属性指定）可以包含各种项（包括布局控件、文本、图像和形状）。下面的示例演示了 Button 包含、、、DockPanel Label Border 和的内容 MediaElement：

```
<Window
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="SDKSample.ButtonContentWindow"
    Title="Button Content">

  <Button Margin="20">
    <!-- Button Content -->
    <DockPanel Width="200" Height="180">
      <Label DockPanel.Dock="Top" HorizontalAlignment="Center">Click Me!</Label>
      <Border Background="Black" BorderBrush="Yellow" BorderThickness="2"
        CornerRadius="2" Margin="5">
        <MediaElement Source="media/wpf.wmv" Stretch="Fill" />
      </Border>
    </DockPanel>
  </Button>
</Window>
```

下图显示了此按钮的内容：



有关各种控件支持的内容类型的详细信息，请参阅 WPF 内容模型。

触发器

尽管 XAML 标记的主要用途是实现应用程序的外观，你也可以使用 XAML 来实现应用程序行为的某些方面。其中一个示例是使用触发器来基于用户交互更改应用程序的外观。有关详细信息，请参阅样式和模板。

控件模板

WPF 控件的默认用户界面通常是从其他控件和形状构造的。例如，Button 由 ButtonChrome 和

ContentPresenter 控件组成。ButtonChrome 提供了标准按钮外观，而 ContentPresenter 显示按钮的内容，正如 Content 属性所指定。

有时，某个控件的默认外观可能与应用程序的整体外观不一致。在这种情况下，可以使用 ControlTemplate 更改控件的用户界面的外观，而不更改其内容和行为。

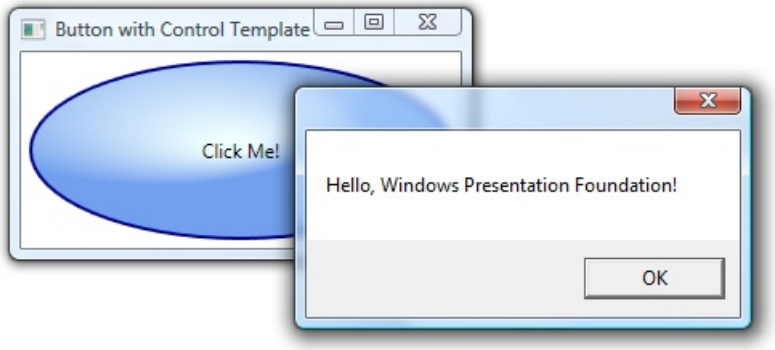
下面的示例演示如何使用更改的外观 Button ControlTemplate：

```
<Window
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  x:Class="SDKSample.ControlTemplateButtonWindow"
  Title="Button with Control Template" Height="158" Width="290">

  <!-- Button using an ellipse -->
  <Button Content="Click Me!" Click="button_Click">
    <Button.Template>
      <ControlTemplate TargetType="{x:Type Button}">
        <Grid Margin="5">
          <Ellipse Stroke="DarkBlue" StrokeThickness="2">
            <Ellipse.Fill>
              <RadialGradientBrush Center="0.3,0.2" RadiusX="0.5" RadiusY="0.5">
                <GradientStop Color="Azure" Offset="0.1" />
                <GradientStop Color="CornflowerBlue" Offset="1.1" />
              </RadialGradientBrush>
            </Ellipse.Fill>
          </Ellipse>
          <ContentPresenter Name="content" HorizontalAlignment="Center"
            VerticalAlignment="Center"/>
        </Grid>
      </ControlTemplate>
    </Button.Template>

  </Button>

</Window>
```

```
using System.Windows; // Window, RoutedEventArgs, MessageBox

namespace SDKSample
{
    public partial class ControlTemplateButtonWindow : Window
    {
        public ControlTemplateButtonWindow()
        {
            InitializeComponent();
        }

        void button_Click(object sender, RoutedEventArgs e)
        {
            // Show message box when button is clicked
            MessageBox.Show("Hello, Windows Presentation Foundation!");
        }
    }
}
```

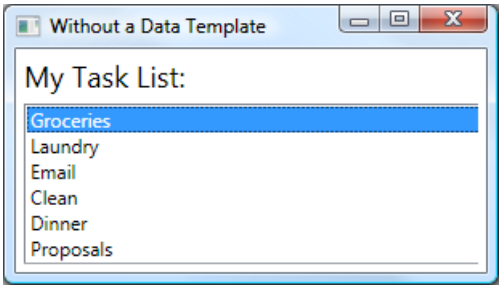
```
Imports System.Windows ' Window, RoutedEventArgs, MessageBox

Namespace SDKSample

    Public Class ControlTemplateButtonWindow
        Inherits Window

        Public Sub New()

            InitializeComponent()

        End Sub

        Private Sub button_Click(ByVal sender As Object, ByVal e As RoutedEventArgs)
            MessageBox.Show("Hello, Windows Presentation Foundation!")
        End Sub

    End Class

End Namespace
```

在此示例中，默认按钮用户界面已被替换为 Ellipse，它具有深蓝色边框并使用 RadialGradientBrush进行填充。 ContentPresenter 控件显示 Button的内容，“单击我!”单击 Button 后，在 Click 控件的默认行为中，仍将引发 Button 事件。结果如下图所示：



数据模板

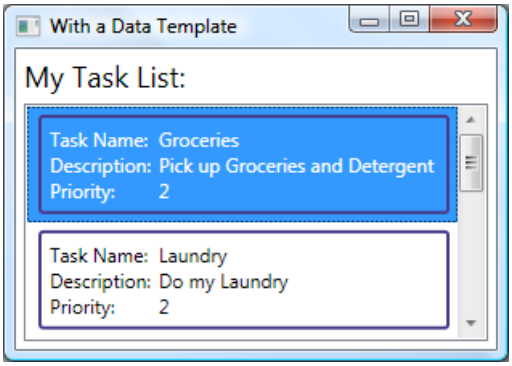
使用控件模板可以指定控件的外观，而使用数据模板则可以指定控件内容的外观。数据模板经常用于改进绑定数据的显示方式。下图显示 ListBox 的默认外观，它绑定到 `Task` 对象的集合，其中每个任务都具有名称、描述和优先级：



默认外观是你对 ListBox的期望。但是，每个任务的默认外观仅包含任务名称。若要显示任务名称、描述和优先级，必须使用 ListBox 更改 DataTemplate控件绑定列表项的默认外观。下面的 XAML 定义了这样的 DataTemplate，它通过使用特性应用于每个任务 ItemTemplate：

```
<Window
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  x:Class="SDKSample.DataTemplateWindow"
  Title="With a Data Template">
  <Window.Resources>
    <!-- Data Template (applied to each bound task item in the task collection) -->
    <DataTemplate x:Key="myTaskTemplate">
      <Border Name="border" BorderBrush="DarkSlateBlue" BorderThickness="2"
        CornerRadius="2" Padding="5" Margin="5">
        <Grid>
          <Grid.RowDefinitions>
            <RowDefinition/>
            <RowDefinition/>
            <RowDefinition/>
          </Grid.RowDefinitions>
          <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto" />
            <ColumnDefinition />
          </Grid.ColumnDefinitions>
          <TextBlock Grid.Row="0" Grid.Column="0" Padding="0,0,5,0" Text="Task Name:"/>
          <TextBlock Grid.Row="0" Grid.Column="1" Text="{Binding Path=TaskName}"/>
          <TextBlock Grid.Row="1" Grid.Column="0" Padding="0,0,5,0" Text="Description:"/>
          <TextBlock Grid.Row="1" Grid.Column="1" Text="{Binding Path=Description}"/>
          <TextBlock Grid.Row="2" Grid.Column="0" Padding="0,0,5,0" Text="Priority:"/>
          <TextBlock Grid.Row="2" Grid.Column="1" Text="{Binding Path=Priority}"/>
        </Grid>
      </Border>
    </DataTemplate>
  </Window.Resources>

  <!-- UI -->
  <DockPanel>
    <!-- Title -->
    <Label DockPanel.Dock="Top" FontSize="18" Margin="5" Content="My Task List:"/>

    <!-- Data template is specified by the ItemTemplate attribute -->
    <ListBox
      ItemsSource="{Binding}"
      ItemTemplate="{StaticResource myTaskTemplate}"
      HorizontalContentAlignment="Stretch"
      IsSynchronizedWithCurrentItem="True"
      Margin="5,0,5,5" />

 </DockPanel>
</Window>
```

下图显示了此代码的效果：



注意，ListBox 已保留其行为和整体外观；仅列表框显示的内容外观已更改。

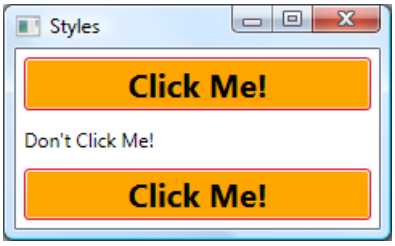
有关详细信息，请参阅数据模板化概述。

样式

通过样式功能，开发人员和设计人员能够对其产品的特定外观进行标准化。WPF 提供了一个强样式模型，其基础是 Style 元素。下面的示例创建一个样式，该样式将窗口上的每个的背景色设置 Button 为 `Orange` ：

```
<Window
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="SDKSample.StyleWindow"
    Title="Styles">

    <Window.Resources>
        <!-- Style that will be applied to all buttons for this window -->
        <Style TargetType="{x:Type Button}">
            <Setter Property="Background" Value="Orange" />
            <Setter Property="BorderBrush" Value="Crimson" />
            <Setter Property="FontSize" Value="20" />
            <Setter Property="FontWeight" Value="Bold" />
            <Setter Property="Margin" Value="5" />
        </Style>
    </Window.Resources>
    <StackPanel>

        <!-- This button will have the style applied to it -->
        <Button>Click Me!</Button>

        <!-- This label will not have the style applied to it -->
        <Label>Don't Click Me!</Label>

        <!-- This button will have the style applied to it -->
        <Button>Click Me!</Button>

    </StackPanel>
</Window>
```

由于此样式针对所有 Button 控件，因此将自动应用于窗口中的所有按钮，如下图所示：



有关详细信息，请参阅样式和模板。

资源

应用程序中的控件应共享相同的外观，它可以包括从字体和背景色到控件模板、数据模板和样式的所有内容。你可以对用户界面资源使用 WPF 支持，以将这些资源封装在一个位置以便重复使用。

下面的示例定义 Button 和 Label共享的通用背景色：

```
<Window
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="SDKSample.ResourcesWindow"
    Title="Resources Window">

  <!-- Define window-scoped background color resource -->
  <Window.Resources>
    <SolidColorBrush x:Key="defaultBackground" Color="Red" />
  </Window.Resources>

  <!-- Button background is defined by window-scoped resource -->
  <Button Background="{StaticResource defaultBackground}">One Button</Button>

  <!-- Label background is defined by window-scoped resource -->
  <Label Background="{StaticResource defaultBackground}">One Label</Label>
</Window>
```

此示例通过使用 `Window.Resources` 属性元素实现背景色资源。此资源可供 Window的所有子级使用。有各种资源作用域，具体如下(按解析顺序列出)：

1. 单个控件(使用继承的 System.Windows.FrameworkElement.Resources 属性)。
2. Window 或 Page (也使用继承的 System.Windows.FrameworkElement.Resources 属性)。
3. Application (使用 System.Windows.Application.Resources 属性)。

这些不同种类的作用域在定义和共享资源的方式方面为你提供了灵活性。

作为直接将你的资源与特定作用域关联的替代方法，可以通过使用单独的 ResourceDictionary (可以在应用程序的其他部分引用)打包一个或多个资源。例如，下面的示例定义资源字典中的默认背景色：

```
<ResourceDictionary
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

  <!-- Define background color resource -->
  <SolidColorBrush x:Key="defaultBackground" Color="Red" />

  <!-- Define other resources -->
</ResourceDictionary>
```

下面的示例引用上一个示例中定义的资源字典，以便在应用程序之间共享：

```
<Application
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="SDKSample.App">

  <Application.Resources>
    <ResourceDictionary>
      <ResourceDictionary.MergedDictionaries>
        <ResourceDictionary Source="BackgroundColorResources.xaml"/>
      </ResourceDictionary.MergedDictionaries>
    </ResourceDictionary>
  </Application.Resources>
</Application>
```

资源和资源字典是 WPF 主题和皮肤支持的基础。

有关详细信息，请参阅资源。

自定义控件

尽管 WPF 提供了大量自定义支持，但你仍可能会遇到现有 WPF 控件不满足你的应用程序或其用户的需求的情况。出现这种情况的原因有：

- 不能通过自定义现有 WPF 实现的外观和感觉创建所需的用户界面。
- 现有 WPF 实现不支持（或很难支持）所需的行为。

但是，此时，你可以充分利用三个 WPF 模型中的一个来创建新的控件。每个模型都针对一个特定的方案并要求你的自定义控件派生自特定 WPF 基类。下面列出了这三个模型：

- **用户控件模型**。自定义控件派生自 UserControl 并由一个或多个其他控件组成。
- **控件模型**。自定义控件派生自 Control，并用于生成使用模板将其行为与其外观分隔开来的实现，非常类似大多数 WPF 控件。派生自 Control 使得你可以更自由地创建自定义用户界面（相较用户控件），但它可能需要花费更多精力。
- **框架元素模型**。当其外观由自定义呈现逻辑（而不是模板）定义时，自定义控件派生自 FrameworkElement 。

下面的示例演示一个派生自的自定义数字向上/向下控件 UserControl ：

```
<UserControl
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  x:Class="SDKSample.NumericUpDown">

  <Grid>

    <Grid.RowDefinitions>
      <RowDefinition/>
      <RowDefinition/>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
      <ColumnDefinition/>
      <ColumnDefinition/>
    </Grid.ColumnDefinitions>

    <!-- Value text box -->
    <Border BorderThickness="1" BorderBrush="Gray" Margin="2" Grid.RowSpan="2"
      VerticalAlignment="Center" HorizontalAlignment="Stretch">
      <TextBlock Name="valueText" Width="60" TextAlignment="Right" Padding="5"/>
    </Border>

    <!-- Up/Down buttons -->
    <RepeatButton Name="upButton" Click="upButton_Click" Grid.Column="1"
      Grid.Row="0">Up</RepeatButton>
    <RepeatButton Name="downButton" Click="downButton_Click" Grid.Column="1"
      Grid.Row="1">Down</RepeatButton>

  </Grid>

</UserControl>
```

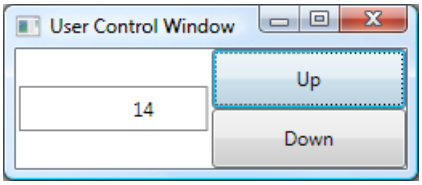
```
using System; // EventArgs
using System.Windows; // DependencyObject, DependencyPropertyChangedEventArgs,
                      // FrameworkPropertyMetadata, PropertyChangedCallback,
                      // RoutedPropertyChangedEventArgs
using System.Windows.Controls; // UserControl

namespace SDKSample
{
    public partial class NumericUpDown : UserControl
    {
        // NumericUpDown user control implementation
    }
}
```

```
imports System 'EventArgs
imports System.Windows 'DependencyObject, DependencyPropertyChangedEventArgs,
                       ' FrameworkPropertyMetadata, PropertyChangedCallback,
                       ' RoutedPropertyChangedEventArgs
imports System.Windows.Controls 'UserControl

Namespace SDKSample

    ' Interaction logic for NumericUpDown.xaml
    Partial Public Class NumericUpDown
        Inherits System.Windows.Controls.UserControl

        'NumericUpDown user control implementation

    End Class

End Namespace
```

下面的示例演示了将用户控件合并到中所需的 XAML Window：

```
<Window
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="SDKSample.UserControlWindow"
    xmlns:local="clr-namespace:SDKSample"
    Title="User Control Window">

  <!-- Numeric Up/Down user control -->
  <local:NumericUpDown />

</Window>
```

下图显示 `NumericUpDown` 了中托管的控件 Window：



有关自定义控件的详细信息，请参阅控件创作概述。

WPF 最佳做法

与任何开发平台一样，可以采用多种方式使用 WPF 以实现所需的结果。为确保你的 WPF 应用程序提供所需的用户体验并满足一般用户的需求，针对辅助功能、全球化和本地化以及性能提供了一些建议的最佳做法。有关详

细信息，请参阅：

- 辅助功能
- WPF 全球化和本地化
- WPF 应用性能
- WPF 安全性

后续步骤

我们已经了解了 WPF 的主要功能。现在可以生成你的第一个 WPF 应用。

演练：我的第一个 WPF 桌面应用

请参阅

- WPF 入门
- Windows Presentation Foundation
- WPF 社区资源

入门 (WPF)

2022/2/12 •

Windows Presentation Foundation (WPF) 是一个可创建桌面客户端应用程序的 UI 框架。WPF 开发平台支持广泛的应用开发功能，包括应用模型、资源、控件、图形、布局、数据绑定、文档和安全性。它是 .NET Framework 的子集，因此，如果你曾经使用 ASP.NET 或 Windows 窗体通过 .NET Framework 构建应用程序，应该会熟悉此编程体验。WPF 使用 Extensible Application Markup Language (XAML)，为应用编程提供声明性模型。本节包含 WPF 简介及入门帮助等主题。

我应从何处着手？

我希望直接开始使用...	演练：我的第一个 WPF 桌面应用程序
如何设计应用程序 UI？	在 Visual Studio 中设计 XAML
是否是初次使用 .NET？	.NET Framework 概述 Visual C# 和 Visual Basic 入门
有关 WPF 的详细信息...	Visual Studio 中的 WPF 简介 WPF 中的 XAML 控件 数据绑定概述
是否是 Windows 窗体开发人员？	Windows 窗体控件和等效的 WPF 控件 WPF 和 Windows 窗体互操作

另请参阅

- 类库
- 应用程序开发
- .NET framework 开发人员中心

应用程序开发

2022/2/12 •

Windows Presentation Foundation (WPF) 是一个演示框架，可用于开发以下类型的应用程序：

- 独立应用程序（传统风格的 Windows 应用程序，这些应用程序作为要安装到客户端计算机并从客户端计算机运行的可执行程序集来生成）。
- XAML 浏览器应用程序 (XBAP)（由导航页组成的应用程序，这些应用程序作为可执行程序集生成并由 Microsoft Internet Explorer 或 Mozilla Firefox 等 Web 浏览器托管）。
- 自定义控件库（包含可重用控件的非可执行程序集）。
- 类库（包含可重用类的非可执行程序集）。

> **NOTE**
> 强烈建议不要在 Windows 服务中使用 WPF 类型。如果尝试在 Windows 服务中使用这些功能，这些功能可能无法按预期工作。

为了生成这样一组应用程序，WPF 要实现众多服务。本主题对这些服务以及在何处可以找到更多相关信息进行了概述。

应用程序管理

可执行的 WPF 应用程序通常需要一组核心功能，其中包括：

- 创建和管理常见的应用程序基础结构（包括创建入口点方法和 Windows 消息循环，以接收系统和输入消息）。
- 对应用程序的生存期进行跟踪并与之进行交互。
- 检索和处理命令行参数。
- 共享应用程序范围的属性和 UI 资源。
- 检测和处理未经处理的异常。
- 返回退出代码。
- 管理独立应用程序中的窗口。
- 跟踪 XAML 浏览器应用程序 (XBAP) 以及使用导航窗口和框架的独立应用程序中的导航。

以上功能由通过 *应用程序定义* 添加到应用程序的 Application 类来实现。

有关详细信息，请参阅应用程序管理概述。

WPF 应用程序资源、内容和数据文件

通过为以下三类非可执行数据文件提供支持，WPF 扩展了 Microsoft .NET Framework 对于嵌入资源的核心支持：资源、内容和数据。有关详细信息，请参阅 WPF 应用程序资源、内容和数据文件。

在对于 WPF 非可执行数据文件的众多支持中，其中的一项重要支持就是能够通过唯一的 URI 来识别和加载这些

文件。有关详细信息，请参阅 WPF 中的 Pack URI。

窗口和对话框

用户通过窗口与 WPF 独立应用程序交互。窗口旨在托管应用程序内容并提供通常允许用户与内容交互的应用程序功能。在 WPF 中，通过 Window 类封装窗口，该类支持：

- 创建和显示窗口。
- 建立所有者/所拥有窗口关系。
- 配置窗口外观（例如，大小、位置、图标、标题栏文本、边框）。
- 对窗口的生存期进行跟踪并与之进行交互。

有关详细信息，请参阅 WPF 窗口概述。

Window 支持用于创建被称为对话框的特殊窗口类型的功能。可以创建两种类型的对话框，即模式和无模式对话框。

为了方便起见，也为了享受可重用性所带来的益处，以及在应用程序间实现一致的用户体验，WPF 提供了三种常用的 Windows 对话框：OpenFileDialog、SaveFileDialog 和 PrintDialog。

消息框是一种特殊类型的对话框，用于向用户显示重要的文本信息并询问简单的“是/否/确定/取消”问题。使用 MessageBox 类创建并显示消息框。

有关详细信息，请参阅对话框概述。

导航

WPF 支持使用页面 (Page) 和超链接 (Hyperlink) 进行Web 式导航。导航可以通过多种方式来实现，其中包括：

- 在 Web 浏览器中承载的独立页面。
- 被编译到 XBAP 中并在 Web 浏览器中托管的页面。
- 被编译到独立应用程序中并由导航窗口 (NavigationWindow) 承载的页面。
- 由框架 (Frame) 托管的页面（可能在独立页面中托管），或是被编译到 XBAP 或独立应用程序中的页面。

为了便于导航，WPF 实现了：

- NavigationService，供 Frame、NavigationWindow 和 XBAP 用于处理导航请求以支持应用程序内导航的共享导航引擎。
- 用于启动导航的导航方法。
- 各种导航事件，用于对导航的生存期进行跟踪并与之进行交互。
- 记住通过日志实现的后向和前向导航，还可以检查和操控这些导航。

有关信息，请参阅导航概述。

WPF 还支持一种被称为结构化导航的特殊导航类型。结构化导航可用于调用一个或多个页面，这些页面能以结构化的可预测方式返回与调用函数一致的数据。此功能将取决于 PageFunction<T> 类；有关该类的进一步描述，请参阅结构化导航概述。PageFunction<T> 还可用于简化导航拓扑概述中所述的复杂导航拓扑的创建。

Hosting

XBAP 可托管在 Microsoft Internet Explorer 或 Firefox 中。每个承载模型都有各自的一些注意事项和约束，这些内容在承载中均有涵盖。

生成和部署

尽管简单的 WPF 应用程序可以在命令提示符下使用命令行编译器来生成，但 WPF 仍与 Visual Studio 实现了集成以提供简化了开发和生成过程的额外支持。有关详细信息，请参阅生成 WPF 应用程序。

根据所生成的应用程序类型，会有一个或多个部署选项可供选择。有关详细信息，请参阅部署 WPF 应用程序。

相关主题

TITLE	
应用程序管理概述	简要介绍 Application 类，包括管理应用程序生存期、窗口、应用程序资源和导航。
WPF 中的窗口	详细介绍如何在应用程序中管理窗口，包括如何使用 Window 类和对话框。
导航概述	概述如何管理应用程序的各个页面间的导航。
承载	概述了 XAML 浏览器应用程序 (XBAP)。
生成和部署	描述如何生成和部署 WPF 应用程序。
Visual Studio 中的 WPF 简介	介绍 WPF 的主要功能。
演练：我的第一个 WPF 桌面应用程序	一项演练，用于演示如何使用页面导航、布局、控件、图像、样式和绑定来创建 WPF 应用程序。

高级 (Windows Presentation Foundation)

2022/2/12 •

本节介绍 WPF 中的部分高级区域。

本节内容

WPF 体系结构
WPF 中的 XAML
基元素类
元素树和序列化
WPF 属性系统
WPF 中的事件
输入
拖放
资源
文档
全球化和本地化
布局
从 WPF 迁移到 System.Xaml 的类型
迁移和互操作性
性能
线程模型
非托管 WPF API 参考\

控制

2022/2/12 •

Windows Presentation Foundation (WPF) 附带许多几乎在所有 Windows 应用程序中都会使用的常见 UI 组件，如 Button、Label、TextBox、Menu 和 ListBox。以前，这些对象称为控件。虽然 WPF SDK 继续使用术语“控件”泛指任何代表应用程序中可见对象的类，但请务必注意，类不必从 Control 类继承即可具有可见外观。从 Control 类继承的类包含一个 ControlTemplate，它允许控件的使用方在无需创建新子类的情况下从根本上改变控件的外观。本主题讨论在 WPF 中使用控件（包括从 Control 类继承的控件以及不从该类继承的控件）的常见方式。

创建控件的实例

可以通过使用 可扩展应用程序标记语言 (XAML) 或以代码形式向应用程序添加控件。下面的示例演示如何创建一个向用户询问其姓名的简单应用程序。此示例在 XAML 中创建六个控件：两个标签、两个文本框及两个按钮。所有控件均可使用相似方式创建。

```
<Grid>
  <Grid.RowDefinitions>
    <RowDefinition Height="30"/>
    <RowDefinition Height="30"/>
    <RowDefinition Height="30"/>
    <RowDefinition/>
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition/>
    <ColumnDefinition/>
  </Grid.ColumnDefinitions>

  <Label>
    Enter your first name:
  </Label>
  <TextBox Grid.Row="0" Grid.Column="1"
           Name="firstName" Margin="0,5,10,5"/>

  <Label Grid.Row="1" >
    Enter your last name:
  </Label>
  <TextBox Grid.Row="1" Grid.Column="1"
           Name="lastName" Margin="0,5,10,5"/>

  <Button Grid.Row="2" Grid.Column="0"
          Name="submit" Margin="2">
    View message
  </Button>

  <Button Grid.Row="2" Grid.Column="1"
          Name="Clear" Margin="2">
    Clear Name
  </Button>
</Grid>
```

以下示例在代码中创建相同的应用程序。为简洁起见，示例中省略了 Grid、`grid1` 的创建过程。`grid1` 的列和行定义与前面的 XAML 示例中所示的相同。

```
Label firstNameLabel;
Label lastNameLabel;
TextBox firstName;
TextBox lastName;
Button submit;
Button clear;

void CreateControls()
{
    firstNameLabel = new Label();
    firstNameLabel.Content = "Enter your first name:";
    grid1.Children.Add(firstNameLabel);

    firstName = new TextBox();
    firstName.Margin = new Thickness(0, 5, 10, 5);
    Grid.SetColumn(firstName, 1);
    grid1.Children.Add(firstName);

    lastNameLabel = new Label();
    lastNameLabel.Content = "Enter your last name:";
    Grid.SetRow(lastNameLabel, 1);
    grid1.Children.Add(lastNameLabel);

    lastName = new TextBox();
    lastName.Margin = new Thickness(0, 5, 10, 5);
    Grid.SetColumn(lastName, 1);
    Grid.SetRow(lastName, 1);
    grid1.Children.Add(lastName);

    submit = new Button();
    submit.Content = "View message";
    Grid.SetRow(submit, 2);
    grid1.Children.Add(submit);

    clear = new Button();
    clear.Content = "Clear Name";
    Grid.SetRow(clear, 2);
    Grid.SetColumn(clear, 1);
    grid1.Children.Add(clear);
}
```

```
Private firstNameLabel As Label
Private lastNameLabel As Label
Private firstName As TextBox
Private lastName As TextBox
Private submit As Button
Private clear As Button

Sub CreateControls()
    firstNameLabel = New Label()
    firstNameLabel.Content = "Enter your first name:"
    grid1.Children.Add(firstNameLabel)

    firstName = New TextBox()
    firstName.Margin = New Thickness(0, 5, 10, 5)
    Grid.SetColumn(firstName, 1)
    grid1.Children.Add(firstName)

    lastNameLabel = New Label()
    lastNameLabel.Content = "Enter your last name:"
    Grid.SetRow(lastNameLabel, 1)
    grid1.Children.Add(lastNameLabel)

    lastName = New TextBox()
    lastName.Margin = New Thickness(0, 5, 10, 5)
    Grid.SetColumn(lastName, 1)
    Grid.SetRow(lastName, 1)
    grid1.Children.Add(lastName)

    submit = New Button()
    submit.Content = "View message"
    Grid.SetRow(submit, 2)
    grid1.Children.Add(submit)

    clear = New Button()
    clear.Content = "Clear Name"
    Grid.SetRow(clear, 2)
    Grid.SetColumn(clear, 1)
    grid1.Children.Add(clear)


End Sub
```

更改控件的外观

更改控件的外观以适应应用程序的外观，这是很常见的操作。可以根据要达到的效果，通过执行以下操作之一来更改控件的外观：

- 更改控件的属性值。
- 为控件创建 Style。
- 为控件新建 ControlTemplate。

更改控件的属性值

许多控件具有支持更改控件外观的属性，例如 Button 的 Background。可以在 XAML 和代码中设置值属性。下面的示例在 XAML 中的 Button 上设置 Background、FontSize 和 FontWeight 属性。

```
<Button FontSize="14" FontWeight="Bold">
  <!--Set the Background property of the Button to
    a LinearGradientBrush.-->
  <Button.Background>
    <LinearGradientBrush StartPoint="0,0.5"
                         EndPoint="1,0.5">
      <GradientStop Color="Green" Offset="0.0" />
      <GradientStop Color="White" Offset="0.9" />
    </LinearGradientBrush>

  </Button.Background>
  View message
</Button>
```

下面的示例在代码中设置相同的属性。

```
LinearGradientBrush buttonBrush = new LinearGradientBrush();
buttonBrush.StartPoint = new Point(0, 0.5);
buttonBrush.EndPoint = new Point(1, 0.5);
buttonBrush.GradientStops.Add(new GradientStop(Colors.Green, 0));
buttonBrush.GradientStops.Add(new GradientStop(Colors.White, 0.9));

submit.Background = buttonBrush;
submit.FontSize = 14;
submit.FontWeight = FontWeights.Bold;
```

```
Dim buttonBrush As New LinearGradientBrush()
buttonBrush.StartPoint = New Point(0, 0.5)
buttonBrush.EndPoint = New Point(1, 0.5)
buttonBrush.GradientStops.Add(New GradientStop(Colors.Green, 0))
buttonBrush.GradientStops.Add(New GradientStop(Colors.White, 0.9))

submit.Background = buttonBrush
submit.FontSize = 14
submit.FontWeight = FontWeights.Bold
```

为控件创建样式

利用 WPF，通过创建 Style，可以同时为许多控件指定相同的外观，而不是为应用程序中的每个实例设置属性。下面的示例创建一个 Style，它将应用于应用程序中的每个 Button。Style 定义通常在 ResourceDictionary（例如 FrameworkElement 的 Resources 属性）中以 XAML 形式定义。

```
<Style TargetType="Button">
  <Setter Property="FontSize" Value="14"/>
  <Setter Property="FontWeight" Value="Bold"/>
  <Setter Property="Background">
    <Setter.Value>
      <LinearGradientBrush StartPoint="0,0.5"
                           EndPoint="1,0.5">
        <GradientStop Color="Green" Offset="0.0" />
        <GradientStop Color="White" Offset="0.9" />
      </LinearGradientBrush>

    </Setter.Value>
  </Setter>
</Style>
```

通过将键分配给样式并在控件的 `Style` 属性中指定该键，还可将样式仅应用于某些特定类型的控件。有关样式的详细信息，请参阅样式设置和模板化。

创建 ControlTemplate

利用 Style，可以一次为多个控件设置属性，但有时除了通过创建 Style 可执行的操作之外，可能还希望自定义 Control 的外观。从 Control 类继承的类具有 ControlTemplate，它用于定义 Control 的结构和外观。Control 的 Template 属性是公共属性，因此可以为 Control 指定非默认的 ControlTemplate。通常可以为 Control 指定新的 ControlTemplate（而不是从控件继承）以自定义 Control 的外观。

请考虑一个很常用的控件 Button。Button 的主要行为是当用户单击它时让应用程序采取某些操作。默认情况下，WPF 中的 Button 显示为一个凸出的矩形。开发应用程序时，用户可能希望利用 Button 的行为（即通过处理按钮的单击事件），不过，除了通过更改按钮的属性可以执行的操作外，也可以更改按钮的外观。在这种情况下，可以创建新的 ControlTemplate。

下面的示例为 Button 创建了一个 ControlTemplate。ControlTemplate 创建一个带圆角和渐变背景的 Button。ControlTemplate 包含一个 Border，其 Background 是带有两个 GradientStop 对象的 LinearGradientBrush。第一个 GradientStop 使用数据绑定将 GradientStop 的 Color 属性绑定到按钮背景的颜色。当设置 Button 的 Background 属性时，该值的颜色将用作第一个 GradientStop。有关数据绑定的详细信息，请参阅数据绑定概述。此示例还创建一个 Trigger，用于在 IsPressed 为 `true` 时更改 Button 的外观。

```
<!--Define a template that creates a gradient-colored button.-->
<Style TargetType="Button">
  <Setter Property="Template">
    <Setter.Value>
      <ControlTemplate TargetType="Button">
        <Border
          x:Name="Border"
          CornerRadius="20"
          BorderThickness="1"
          BorderBrush="Black">
          <Border.Background>
            <LinearGradientBrush StartPoint="0,0.5"
                                 EndPoint="1,0.5">
              <GradientStop Color="{Binding Background.Color,
                    RelativeSource={RelativeSource TemplatedParent}}"
                            Offset="0.0" />
              <GradientStop Color="White" Offset="0.9" />
            </LinearGradientBrush>
          </Border.Background>
          <ContentPresenter
            Margin="2"
            HorizontalAlignment="Center"
            VerticalAlignment="Center"
            RecognizesAccessKey="True"/>
        </Border>
        <ControlTemplate.Triggers>
          <!--Change the appearance of
          the button when the user clicks it.-->
          <Trigger Property="IsPressed" Value="true">
            <Setter TargetName="Border" Property="Background">
              <Setter.Value>
                <LinearGradientBrush StartPoint="0,0.5"
                                     EndPoint="1,0.5">
                  <GradientStop Color="{Binding Background.Color,
                    RelativeSource={RelativeSource TemplatedParent}}"
                                Offset="0.0" />
                  <GradientStop Color="DarkSlateGray" Offset="0.9" />
                </LinearGradientBrush>
              </Setter.Value>
            </Setter>
          </Trigger>

        </ControlTemplate.Triggers>
      </ControlTemplate>
    </Setter.Value>
  </Setter>
</Style>
```

```
<Button Grid.Row="2" Grid.ColumnSpan="2" Name="submitName"
        Background="Green">View message</Button>
```

> **NOTE**
>
> 为使此示例正常工作，Button 的 Background 属性必须设置为 SolidColorBrush。

订阅事件

可以使用 XAML 或代码来订阅控件的事件，但只能在代码中处理事件。下面的示例演示如何订阅 Button 的 `Click` 事件。

```
<Button Grid.Row="2" Grid.ColumnSpan="2" Name="submitName" Click="submit_Click"
  Background="Green">View message</Button>
```

```
submit.Click += new RoutedEventHandler(submit_Click);
```

```
AddHandler submit.Click, AddressOf submit_Click
```

下面的示例处理 Button 的 `Click` 事件。

```
void submit_Click(object sender, RoutedEventArgs e)
{
    MessageBox.Show("Hello, " + firstName.Text + " " + lastName.Text);
}
```

```
Private Sub submit_Click(ByVal sender As Object, ByVal e As RoutedEventArgs)
    MessageBox.Show("Hello, " + firstName.Text + " " + lastName.Text)

End Sub
```

控件中的丰富内容

从 Control 类继承的大多数类具有包含丰富内容的能力。例如，Label 可以包含任意对象，例如字符串、Image 或 Panel。下列类支持丰富内容，可以用作 WPF 中大多数控件的基类。

- ContentControl-- 从此类继承的类的部分示例包括 Label、Button 和 ToolTip。
- ItemsControl-- 从此类继承的类的部分示例包括 ListBox、Menu 和 StatusBar。
- HeaderedContentControl-- 从此类继承的类的部分示例包括 TabItem、GroupBox 和 Expander。
- HeaderedItemsControl-- 从此类继承的类的部分示例包括 MenuItem、TreeViewItem 和 ToolBar。

有关这些基类的详细信息，请参阅 WPF 内容模型。

请参阅

- 样式设置和模板化
- 按类别分类的控件
- 控件库
- 数据模板化概述
- 数据绑定概述
- 输入
- 启用命令
- 演练：创建自定义的动画按钮
- 控件自定义

数据

2022/2/12 •

Windows Presentation Foundation (WPF) 数据绑定为应用程序呈现数据并与数据交互提供了一种简单且一致的方式。元素能够以公共语言运行时 (CLR) 对象和 XML 的形式绑定到多种数据源中的数据。Windows Presentation Foundation (WPF) 还提供了通过拖放操作传输数据的机制。

本节内容

数据绑定
拖放

参考

System.Windows.Data

Binding

DataTemplate

DataTemplateSelector

相关章节

控件

样式设置和模板化

数据绑定

另请参阅

- 演练：我的第一个 WPF 桌面应用程序
- 演练：在 WPF 应用程序中缓存应用程序数据

图形和多媒体

2022/2/12 •

Windows Presentation Foundation (WPF) 为多媒体、矢量图形、动画和内容复合提供支持，使开发者可以轻松生成有趣的用户界面和内容。使用 Visual Studio，可以创建矢量图形或复杂动画，并将媒体集成到应用程序中。

本主题介绍 WPF 的图形、动画和媒体功能，可用于向应用程序添加图形、转换效果、声音和视频。

> **NOTE**
> 强烈建议不要在 Windows 服务中使用 WPF 类型。如果尝试在 Windows 服务中使用 WPF 类型，该服务可能无法按预期工作。

WPF 4 中的图形和多媒体新增功能

进行了与图形和动画相关的多项更改。

- 布局舍入

  当对象边缘落在像素设备中间位置时，与 DPI 无关的图形系统可以创建呈现项目，如模糊或半透明边缘。WPF 的以前版本包含像素捕捉以帮助处理这种情况。Silverlight 2 引入了布局舍入，这是移动元素以使边缘落在整个像素边界上的另一种方法。WPF 现在支持在 FrameworkElement 上使用 UseLayoutRounding 附加属性进行布局舍入。

- 缓存复合

  通过使用新的 BitmapCache 和 BitmapCacheBrush 类，可以将可视化树的复杂部分缓存为位图，并大大缩短呈现时间。位图仍然能够响应用户输入（如鼠标单击），并且可以像任何画笔一样将其绘制到其他元素上。

- 像素着色器 3 支持

  WPF 4 基于 WPF 3.5 SP1 中引入的 ShaderEffect 支持而生成，允许应用程序使用像素着色器 (PS) 版本 3.0 写入效果。PS 3.0 着色器模型比 PS 2.0 更复杂，从而允许在支持的硬件上使用更多效果。

- 缓动函数

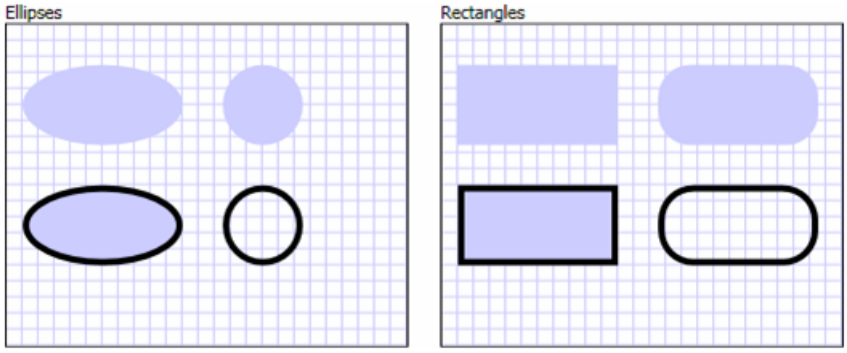
  可以使用缓动函数增强动画，从而提供对动画行为的额外控制。例如，可以将 ElasticEase 应用到动画，以提供类似弹簧的行为。有关详细信息，请参阅 System.Windows.Media.Animation 命名空间中的缓动类型。

图形和呈现

WPF 引入了对高质量 2D 图形的支持。功能包括画笔、几何、图像、形状和转换。有关详细信息，请参阅图形。图形元素的呈现基于 Visual 类。屏幕上的视觉对象的结构由可视化树描述。有关详细信息，请参阅 WPF 图形呈现概述。

2D 形状

WPF 提供常用矢量绘制的 2D 形状库，如下图中所示的矩形和椭圆形。



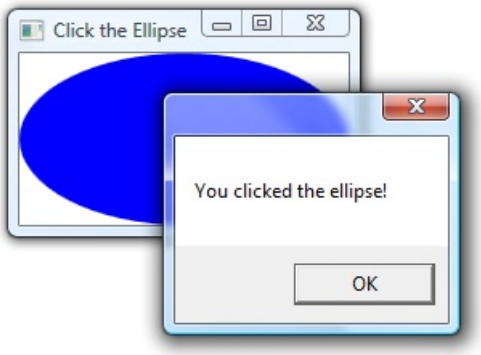
这些内部 WPF 形状不仅仅是形状：它们是可编程的元素，用于实现常见控件的许多预期功能，包括键盘和鼠标输入。下面的示例演示如何处理因单击 Ellipse 元素而引发的 MouseUp 事件。

```
<Window
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  x:Class="Window1" >
  <Ellipse Fill="LightBlue" MouseUp="ellipseButton_MouseUp" />
</Window>
```

```
public partial class Window1  : Window
{
    void ellipseButton_MouseUp(object sender, MouseButtonEventArgs e)
    {
        MessageBox.Show("You clicked the ellipse!");
    }
}
```

```
Partial Public Class Window1
    Inherits Window
    Private Sub ellipseButton_MouseUp(ByVal sender As Object, ByVal e As MouseButtonEventArgs)
        MessageBox.Show("You clicked the ellipse!")
    End Sub
End Class
```

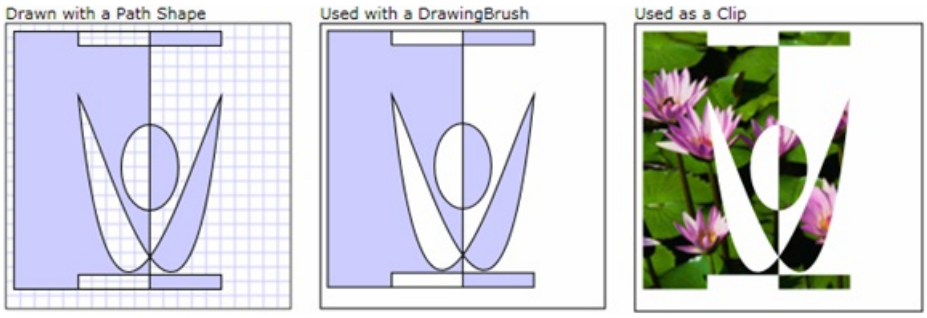
下图显示前面的 XAML 标记和代码隐藏的输出。



有关详细信息，请参阅 WPF 中的形状和基本绘图概述。有关介绍性示例，请参阅形状元素示例。

2D 几何图形

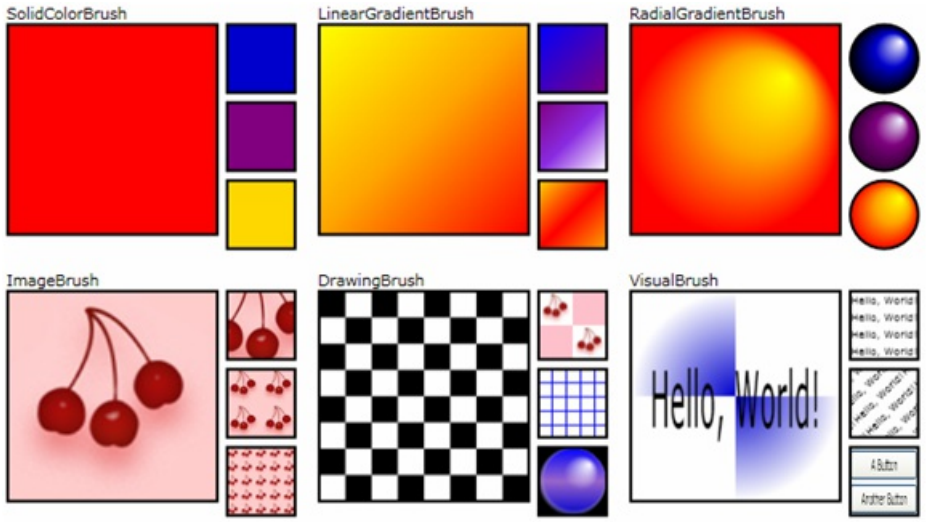
当 WPF 提供的 2D 形状不足时，可以使用对几何和路径的 WPF 支持来创建自己的形状。下图显示如何使用几何创建形状作为图形画笔和剪裁其他 WPF 元素。



有关详细信息，请参阅 Geometry 概述。有关介绍性示例，请参阅 Geometry 示例。

2D 效果

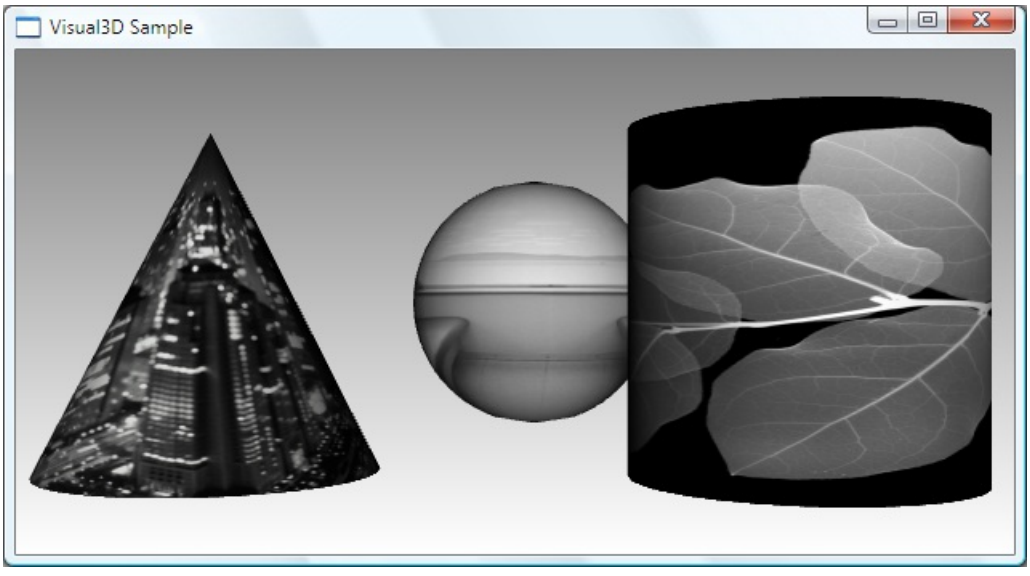
WPF 提供了 2D 类的库，可用于创建各种效果。借助 WPF 的 2D 呈现功能，可以绘制具有渐变、位图、绘图和视频的 UI 元素，并且可以使用旋转、缩放和倾斜来操作它们。下图演提供了可使用 WPF 画笔实现的许多效果的示例。



有关详细信息，请参阅 WPF 画笔概述。有关详细信息，请参阅画笔示例。

3D 呈现

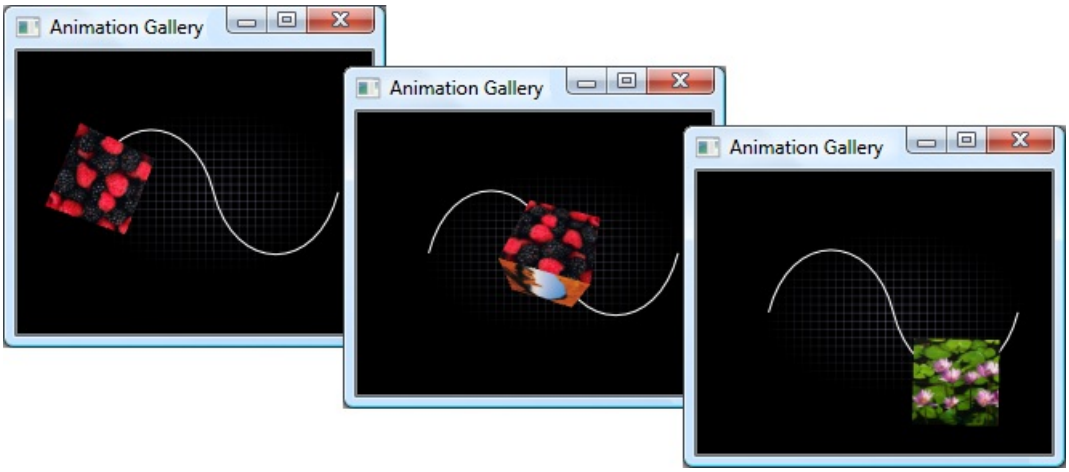
WPF 提供一组 3D 呈现功能，这些功能与 WPF 中的 2D 图形支持集成，以便创建更精彩的布局、UI 和数据可视化。另一方面，WPF 支持将 2D 图像呈现到 3D 形状的表面，下图演示了此功能。



有关详细信息，请参阅三维图形概述。有关介绍性示例，请参阅 3D 实体示例。

动画

使用动画，可以让控件和元素变大、抖动、旋转和淡出，并创建有趣的转换等。由于 WPF 支持对大多数属性进行动画处理，因此，不仅可以对大多数 WPF 对象进行动画处理，还可以使用 WPF 对创建的自定义对象进行动画处理。



有关详细信息，请参阅 动画概述。有关介绍性示例，请参阅动画示例库。

媒体

图像、视频和音频是传达信息和用户体验的富媒体方法。

映像

图像（包括图标、背景甚至动画部分）是大部分应用程序的核心部分。由于经常需要使用图像，因此 WPF 公开以各种方式处理它们的功能。下图显示其中一种方法。

有关详细信息，请参阅 图像概述。

视频和音频

WPF 的图形功能的核心功能是为处理多媒体（包括视频和音频）提供本机支持。以下示例介绍如何将媒体播放器插入到应用程序中。

```
<MediaElement Source="media\numbers.wmv" Width="450" Height="250" />
```

MediaElement 能够播放视频和音频，并且可扩展，足以轻松创建自定义的 UI。

有关详细信息，请参阅多媒体概述。

请参阅

- System.Windows.Media
- System.Windows.Media.Animation

- System.Windows.Media.Media3D
- 二维图形和图像处理
- WPF 中的形状和基本图形概述
- 使用纯色和渐变进行绘制概述
- 使用图像、图形和视觉对象进行绘制
- 动画和计时帮助主题
- 三维图形概述
- 多媒体概述

安全性 (WPF)

2022/2/12 •

在Windows Presentation Foundation (独立) 浏览器托管的应用程序中开发 WPF 时，必须考虑安全模型。无论 (使用 Windows Installer (.msi) 、XCopy 或 ClickOnce 部署，WPF 独立应用程序都使用 CAS FullTrust 权限集) 的无限制权限执行。不支持使用 ClickOnce 部署部分信任的独立 WPF 应用程序。但是，完全信任主机应用程序可以使用 .NET Framework AppDomain 模型创建部分信任。有关详细信息，请参阅 WPF Add-Ins 概述。

WPF 浏览器托管的应用程序由 Windows Internet Explorer 或 Firefox 托管，可以是 XAML 浏览器应用程序 (XBAP) 或松散文档。有关详细信息，请参阅 可扩展应用程序标记语言 (XAML) WPF XAML 浏览器应用程序概述。

默认情况下，WPF 浏览器托管应用程序在部分信任安全沙盒中执行，该沙盒仅限于默认的 CAS Internet 区域权限集。这有效地将 WPF 浏览器托管的应用程序与客户端计算机隔离，其方式与隔离典型的 Web 应用程序的方式相同。XBAP 最高可以将权限提升到“完全信任”，具体取决于部署 URL 的安全区域和客户端的安全配置。有关详细信息，请参阅 WPF 部分信任安全性。

本主题讨论独立和浏览器托管Windows Presentation Foundation (WPF) 的安全模型。

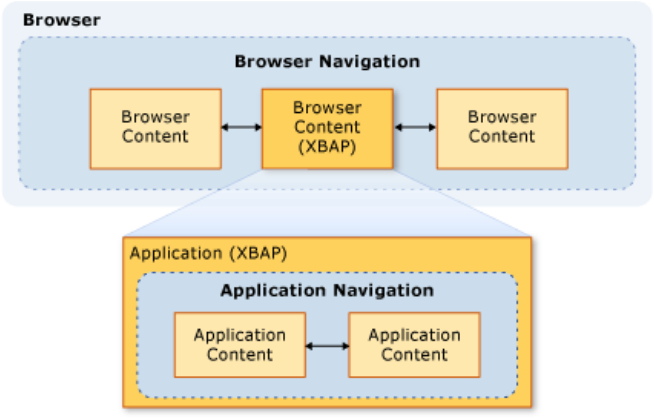
本主题包含以下各节：

- 安全导航
- Web 浏览软件安全设置
- WebBrowser 控件和功能控件
- 对部分受信任的客户端应用程序禁用 APTCA 程序集
- 宽松 XAML 文件的沙盒行为
- 用于开发可提高安全性的 WPF 应用程序的资源

安全导航

对于 XBAP，WPF 区分了两种类型的导航：应用程序和浏览器。

应用程序导航是指在浏览器托管的应用程序内的内容项之间进行导航。浏览器导航是指可更改浏览器自身的内容和位置 URL 的导航。下图显示了应用程序导航 (XAML) 浏览器导航 (HTML) 之间的关系：



XBAP 导航到的内容类型主要取决于是使用应用程序导航还是浏览器导航。

应用程序导航安全性

如果可以使用支持四种类型的内容的包 URI 标识应用程序导航，则应用程序导航会被视为安全：

□□□□	□□	URI □□
资源	添加到生成类型为 Resource 的项目 ■ ▮。	`pack://application:,,,/MyResourceFile.xaml`
Content	添加到生成类型为 Content 的项目 ■■。	`pack://application:,,,/MyContentFile.xaml`
源站点	添加到生成类型为 None 的项目 ■■。	`pack://siteoforigin:,,,/MySiteOfOriginFile.xa`
应用程序代码	具有已编译代码隐藏的 XAML 资源。 - 或 - 添加到生成类型为 Page 的项目的 XAML ■。	`pack://application:,,,/MyResourceFile.xaml`

> **NOTE**
> 有关应用程序数据文件和包 URI 的信息，请参阅 WPF 应用程序资源、内容和数据文件。

可以由用户导航到这些内容类型的文件，也可以通过编程方式导航到这些内容类型的文件：

- **用户导航**。用户通过单击 元素进行 Hyperlink 导航。
- **编程导航**。应用程序在无需用户参与的情况下进行导航，例如，通过设置 NavigationWindow.Source 属性。

浏览器导航安全性

浏览器导航仅在以下条件下被视为安全：

- **用户导航**。用户通过单击主（而不是嵌套）Hyperlink NavigationWindow 中的元素进行导航 Frame 。
- **区域**。要导航到的内容位于 Internet 或本地 Intranet 上。
- 协议。使用的协议是 **http、https、**文件 或 mailto。

如果 XBAP 尝试以不符合这些条件的方式导航到内容，则 SecurityException 会引发 。

Web 浏览软件安全设置

计算机上的安全设置决定了任何 Web 浏览软件被授予的访问权限。Web 浏览包括任何使用 WinINet 或 UrlMon API 的应用程序或组件，包括 Internet Explorer 和 PresentationHost.exe。

Internet Explorer提供了一种机制，通过该机制可以配置允许由用户执行或从Internet Explorer的功能，包括：

- .NET Framework组件
- ActiveX 控件和插件
- 下载
- 脚本编写
- 用户身份验证

可以按区域为 Internet、Intranet、受信任的站点和受限站点区域配置可通过此方法保护 的功能 集合。以下步骤描述如何配置安全设置：

1. 打开“控制面板”

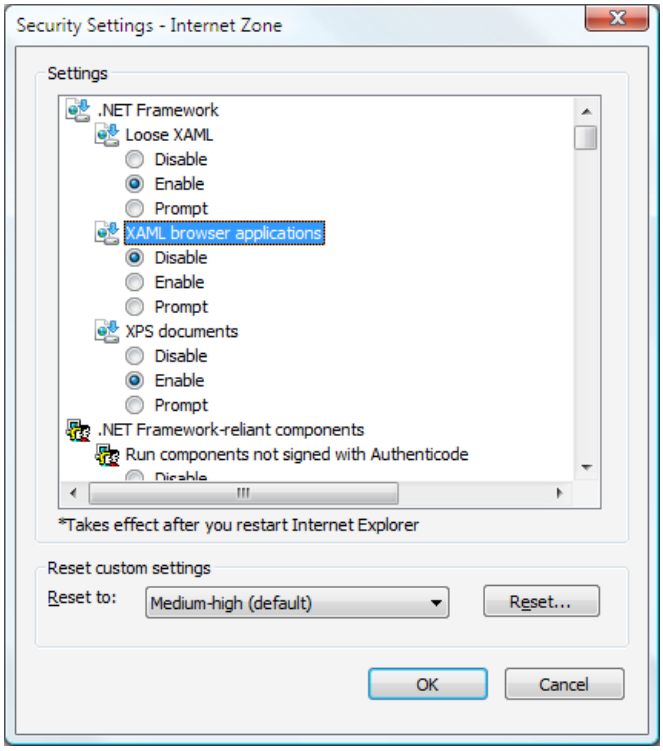
2. 单击 **"网络和 Internet"**，然后单击 **"Internet 选项"**。

   将显示“Internet 选项”对话框。

3. 在 **" 安全性 "** 选项卡上，选择要配置其安全设置的区域。

4. 单击 **" 自定义级别 " 按钮**。

   将出现 **"设置"** 对话框，你可以为所选区域配置安全设置。



> **NOTE**
>
> 也可以从 Internet Explorer 中进入“Internet 选项”对话框。单击 **"■"**，然后单击 **"Internet ■"**。

从 Windows Internet Explorer 7 开始，包含以下专门用于.NET Framework安全设置：

- **宽松 XAML**。控制Internet Explorer是否可导航到和松散 XAML 文件。（“启用”、“禁用”和“提示”选项）。
- **XAML 浏览器应用程序**。控制Internet Explorer导航到并运行 XBAP。（“启用”、“禁用”和“提示”选项）。

默认情况下，这些设置都为 **Internet**、本地 **Intranet** 和受信任的站点区域启用，对受限站点区域 **禁用**。

与安全相关的 WPF 注册表设置

除了可通过“Internet 选项”设置的安全设置之外，还可以通过设置以下注册表值有选择地阻止许多安全敏感 WPF 功能。这些值在以下注册表项下定义：

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\.NETFramework\Windows Presentation Foundation\Features
```

下表列出了可以设置的值。

■■■	■■■	■■■
XBAPDisallow	REG_DWORD	1 为禁止；0 为允许。
LooseXamlDisallow	REG_DWORD	1 为禁止；0 为允许。
WebBrowserDisallow	REG_DWORD	1 为禁止；0 为允许。

MediaAudioDisallow	REG_DWORD	1 为禁止;0 为允许。
MediaImageDisallow	REG_DWORD	1 为禁止;0 为允许。
MediaVideoDisallow	REG_DWORD	1 为禁止;0 为允许。
ScriptInteropDisallow	REG_DWORD	1 为禁止;0 为允许。

WebBrowser 控件和功能控件

WPF WebBrowser 控件可用于托管 Web 内容。WPF WebBrowser 控件包装基础 WebBrowser ActiveX控件。使用 WPF 控件托管不受信任的 Web 内容时，WPF 为保护应用程序提供了 WebBrowser 一些支持。但是，某些安全功能必须由使用 控件的应用程序直接 WebBrowser 应用。有关 WebBrowser 控件ActiveX，请参阅WebBrowser 控件概述和教程。

> **NOTE**
> 本部分也适用于 控件，Frame 因为它使用 WebBrowser 导航到 HTML 内容。

如果 WPF 控件用于托管不受信任的 Web 内容，则应用程序应该使用部分信任来帮助避免应用程序代码与 WebBrowser AppDomain 潜在的恶意 HTML 脚本代码隔离。如果应用程序使用 方法和 属性与托管脚本交互，则 InvokeScript 尤其 ObjectForScripting 如此。有关详细信息，请参阅 WPF Add-Ins 概述。

如果应用程序使用 WPF 控件，则提高安全性和缓解攻击的另一种方式是启用Internet Explorer WebBrowser 控制。功能控制是 Internet Explorer 的新增功能，允许管理员和开发人员配置 Internet Explorer 和托管 WebBrowser ActiveX 控件（WPF 控件包装）的应用程序的功能。WebBrowser 可以通过使用 CoInternetSetFeatureEnabled 函数或更改注册表中的值来配置功能控件。有关功能控件详细信息，请参阅 功能控件和 Internet 功能控件简介。

如果要开发使用 WPF 控件的独立 WPF WebBrowser 应用程序，WPF 会自动为应用程序启用以下功能控件。

FEATURE_MIME_HANDLING
FEATURE_MIME_SNIFFING
FEATURE_OBJECT_CACHING
FEATURE_SAFE_BINDTOOBJECT
FEATURE_WINDOW_RESTRICTIONS
FEATURE_ZONE_ELEVATION
FEATURE_RESTRICT_FILEDOWNLOAD
FEATURE_RESTRICT_ACTIVEXINSTALL
FEATURE_ADDON_MANAGEMENT
FEATURE_HTTP_USERNAME_PASSWORD_DISABLE

FEATURE_SECURITYBAND
FEATURE_UNC_SAVEDFILECHECK
FEATURE_VALIDATE_NAVIGATE_URL
FEATURE_DISABLE_TELNET_PROTOCOL
FEATURE_WEBOC_POPUPMANAGEMENT
FEATURE_DISABLE_LEGACY_COMPRESSION
FEATURE_SSLUX

由于这些功能控件是无条件启用的，因此它们可能会对完全信任的应用程序造成损害。在这种情况下，如果特定应用程序及其承载的内容没有安全风险，则可以禁用相应的功能控件。

功能控件由实例化 WebBrowser 对象ActiveX应用。因此，如果要创建可导航到不受信任的内容的独立应用程序，则应该认真考虑启用附加功能控件。

> **NOTE**
> 此建议是根据 MSHTML 和 SHDOCVW 主机安全性的一般性建议提出的。有关详细信息，请参阅 MSHTML 主机安全性常见问题解答:II 的第 I 部分和 MSHTML 主机安全性常见问题解答:II 的第 II 部分。

对于可执行文件，请考虑通过将注册表值设置为 1 来启用以下功能控件。

FEATURE_ACTIVEX_REPURPOSEDETECTION
FEATURE_BLOCK_LMZ_IMG
FEATURE_BLOCK_LMZ_OBJECT
FEATURE_BLOCK_LMZ_SCRIPT
FEATURE_RESTRICT_RES_TO_LMZ
FEATURE_RESTRICT_ABOUT_PROTOCOL_IE7
FEATURE_SHOW_APP_PROTOCOL_WARN_DIALOG
FEATURE_LOCALMACHINE_LOCKDOWN
FEATURE_FORCE_ADDR_AND_STATUS
FEATURE_RESTRICTED_ZONE_WHEN_FILE_NOT_FOUND

对于可执行文件，请考虑通过将注册表值设置为 0 来禁用以下功能控件。

[[[[
FEATURE_ENABLE_SCRIPT_PASTE_URLACTION_IF_PROMPT

如果运行在 Windows Internet Explorer 中包含 WPF 控件的部分信任 XAML 浏览器应用程序 (XBAP)，WPF 在 Internet Explorer 进程的地址空间中托管 WebBrowser WebBrowser ActiveX 控件。由于 WebBrowser ActiveX 控件托管在 Internet Explorer 进程中，因此 Internet Explorer 的所有功能控件也会为 WebBrowser 控件ActiveX启用。

与普通的独立应用程序相比，运行于 Internet Explorer 中的 XBAP 还将另外获得一层安全保护。这一附加安全性 Internet Explorer WebBrowser ActiveX 控件默认在 Windows Vista 和 Windows 7 上以受保护模式运行。有关受保护模式的信息，请参阅了解和在受保护 模式下Internet Explorer。

> **NOTE**
> 如果尝试在 Firefox 中运行包含 WPF 控件的 XBAP，而在 Internet 区域中，WebBrowser SecurityException 将引发 。这是由于 WPF 安全策略造成的。

对部分受信任的客户端应用程序禁用 APTCA 程序集

当托管程序集安装到 GAC (全局程序集缓存)，这些程序集将完全受信任，因为用户必须提供显式权限才能安装它们。因为这些程序集是完全受信任的，所以只有完全受信任的托管客户端应用程序才可以使用它们。若要允许部分受信任的应用程序使用它们，必须使用 AllowPartiallyTrustedCallersAttribute APTCA (标记) 。仅当程序集经过测试，可在部分信任的情况下安全执行时，才应该为其标记此特性。

但是，APTCA 程序集在安装到 GAC 后可能会表现出安全缺陷。一旦发现安全漏洞，程序集发布者可以生成安全更新来修复现有安装上的问题，还可以阻止问题发现后进行的安装操作。其中一个更新选项是卸载程序集，即使这可能中断使用该程序集的其他完全受信任的客户端应用程序。

WPF 提供了一种机制，通过该机制，可以在不卸载 APTCA 程序集的情况下为部分受信任的 XBAP 禁用 APTCA 程序集。

若要禁用 APTCA 程序集，必须创建一个特殊的注册表项：

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\.NETFramework\policy\APTCA\<AssemblyFullName>, FileVersion=
<AssemblyFileVersion>
```

示例如下：

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\.NETFramework\policy\APTCA\aptcagac, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=215e3ac809a0fea7, FileVersion=1.0.0.0
```

此项建立 APTCA 程序集的条目。还必须在此项中创建值来启用或禁用程序集。下面是该值的详细信息：

- 值名称：APTCA_FLAG。
- 值类型：REG_DWORD。
- 值数据：1 要禁用;要启用的 0。

如果必须为部分受信任的客户端应用程序禁用某程序集，可以编写一个用于创建注册表项和值的更新。

> **NOTE**
> 核心.NET Framework程序集不受这种方式禁用的影响，因为它们是托管应用程序运行所需的。对禁用 APTCA 程序集的支持主要面向第三方应用程序。

宽松 XAML 文件的沙盒行为

松散 XAML 文件是仅标记的 XAML 文件，不依赖于任何代码隐藏、事件处理程序或特定于应用程序的程序集。当松散文件直接从浏览器导航到 时，它们根据默认的 Internet 区域权限集加载到安全 XAML 沙盒中。

但是，当松散文件从 或独立应用程序中导航到 时， XAML NavigationWindow 安全 Frame 行为会有所不同。

在这两种情况下，导航到 的松散文件 XAML 将继承其主机应用程序的权限。但是，从安全角度来看，此行为可能并不可取，尤其是在松散文件由不受信任的或未知的 XAML 实体生成时。此类型的内容称为 *外部内容*， Frame 并且 NavigationWindow 可以配置为在导航到时将其隔离。可以通过将 **SandboxExternalContent** 属性设置为 true 来实现隔离，如和的以下示例所 Frame 示 NavigationWindow ：

```
<Frame
  Source="ExternalContentPage.xaml"
  SandboxExternalContent="True">
</Frame>
```

```
<!-- Sandboxing external content using NavigationWindow-->
<NavigationWindow
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  Source="ExternalContentPage.xaml"
  SandboxExternalContent="True">
</NavigationWindow>
```

使用此设置，外部内容将加载到不同于承载应用程序的进程的进程中。此进程被限制在默认 Internet 区域权限集中，从而有效地将其与承载应用程序和客户端计算机隔离。

> **NOTE**
> 即使 XAML 从 NavigationWindow 或独立应用程序中的松散文件导航到 Frame 基于 WPF 浏览器宿主基础结构（涉及 presentationhost.exe 过程），安全级别也略小于直接在 Internet Explorer 中的内容加载到 Windows Vista 和 Windows 7 (，这仍然是通过 presentationhost.exe) 。这是因为使用 Web 浏览器的独立 WPF 应用程序不提供 Internet Explorer 的额外“保护模式”安全功能。

用于开发可提高安全性的 WPF 应用程序的资源

下面是一些其他资源，可帮助开发可提高安全性的 WPF 应用程序：

托管代码	应用程序的模式和实践安全指南
CAS	代码访问安全性
ClickOnce	ClickOnce 安全和部署
WPF	WPF 部分信任安全

请参阅

- WPF 部分信任安全
- WPF 安全策略 - 平台安全性
- WPF 安全策略 - 安全工程
- 应用程序的模式和实践安全指南
- 代码访问安全性
- ClickOnce 安全和部署

- WPF 中的 XAML

WPF 部分信任安全

2022/2/12 •

一般情况下，应该限制 Internet 应用程序直接访问关键系统资源，防止恶意损坏。默认情况下，HTML 和客户端脚本语言不能访问关键系统资源。由于 Windows Presentation Foundation (WPF) 浏览器承载的应用程序可以从浏览器启动，因此它们应符合一组类似的限制。为了强制实施这些限制，wpf 依赖于代码访问安全性 (CAS) 和 ClickOnce (请参阅WPF 安全策略-平台安全性) 。默认情况下，浏览器承载的应用程序请求 Internet 区域 CA 权限集，不管它们是从 Internet、本地 intranet 还是本地计算机启动。如果应用程序的运行权限小于完整权限集，则说明该应用程序正在部分信任环境下运行。

WPF 提供了广泛的支持，以确保在部分信任环境中可以安全地使用尽可能多的功能，并与 CA 一起为部分信任编程提供附加支持。

本主题包含以下各节：

- WPF 功能部分信任支持
- 部分信任编程
- 管理权限

WPF 功能部分信任支持

下表列出了可在 Internet 区域权限集限制范围内安全使用 (WPF) Windows Presentation Foundation 的高级功能。

表 1：在部分信任环境中安全的 WPF 功能

ZONE	FEATURE
常规	浏览器窗口 源站点访问 IsolatedStorage（512KB 限制） UIAutomation 提供程序 命令 输入法编辑器 (IME) 触笔和墨迹 使用鼠标捕获和移动事件模拟的拖/放 OpenFileDialog XAML 反序列化（通过 XamlReader.Load）

|||	FEATURE
Web 集成	浏览器下载对话框 顶级用户启动的导航 mailto:links 统一资源标识符参数 HTTPWebRequest IFRAME 中托管的 WPF 内容 使用框架托管同一站点 HTML 页 使用 WebBrowser 托管同一站点 HTML 页 Web 服务 (ASMX) Web 服务(使用 Windows Communication Foundation) 脚本编写 文档对象模型
视觉对象	2D 和 3D 动画 媒体(源站点和跨域) 图像处理/音频/视频
正在读取	流文档 XPS 文档 嵌入式字体与系统字体 CFF 字体与 TrueType 字体
正在编辑	拼写检查 RichTextBox 纯文本和墨迹剪贴板支持 用户启动的粘贴 复制选定内容
控件	常规控件

此表包含高级别的 WPF 功能。有关更多详细信息，Windows SDK 记录 WPF 中每个成员所需的权限。此外，以下功能含有部分信任执行的相关详细信息，其中包括特殊注意事项。

- XAML (参见 WPF) 中的 XAML 。
- 弹出 (参阅 System.Windows.Controls.Primitives.Popup) 。
- 拖放 (请参阅 拖放概述) 。

- 剪贴板 (参阅 System.Windows.Clipboard) 。
- 图像 (参阅 System.Windows.Controls.Image) 。
- 序列化 (XamlReader.Load 参阅 XamlWriter.Save) 。
- "打开文件" 对话框 (参阅 Microsoft.Win32.OpenFileDialog) 。

下表概述了在 Internet 区域权限集限制内不能安全运行的 WPF 功能。

表 2:在部分信任环境中不安全的 WPF 功能

|||	FEATURE
常规	窗口(应用程序定义的窗口和对话框) SaveFileDialog 文件系统 注册表访问 拖放 XAML 序列化(通过 XamlWriter.Save) UIAutomation 客户端 源窗口访问 (HwndHost) 完全语音支持 Windows 窗体互操作性
视觉对象	位图效果 图像编码
正在编辑	RTF 格式剪贴板 完全 XAML 支持

部分信任编程

对于 XBAP 应用程序，超过默认权限集的代码将具有不同的行为，具体取决于安全区域。在某些情况下，用户会在尝试安装它时收到警告。用户可以选择继续或取消安装。下表描述每个安全区域的应用程序的行为，以及为了使应用程序接收完全信任而必须执行的操作。

本地计算机	自动完全信任	无需采取任何措施。
Intranet 和受信任的站点	提示完全信任	使用证书对 XBAP 进行签名，以便用户在提示中看到源。
Internet	失败，并显示“未授予信任”	使用证书对 XBAP 进行签名。

> **NOTE**
>
> 上表中描述的行为针对不遵循 ClickOnce 受信任部署模型的完全信任 XBAP。

通常，超出允许权限的代码可能是在独立应用程序和浏览器托管的应用程序之间共享的公用代码。CAS 和 WPF 提供了几种管理此方案的方法。

使用 CAS 检测权限

在某些情况下，独立应用程序和 Xbap 可以使用库程序集中的共享代码。这时，代码执行的功能所需要的权限可能超出应用程序的授权权限集允许的权限。应用程序可以使用 Microsoft .NET Framework 安全性来检测它是否有特定权限。具体而言，它可以通过 Demand 对所需权限的实例调用方法来测试它是否有特定权限。以下示例对此进行了演示，示例中的代码查询其是否能够将文件保存到本地磁盘：

```
using System.IO;
using System.IO.IsolatedStorage;
using System.Security;
using System.Security.Permissions;
using System.Windows;

namespace SDKSample
{
    public class FileHandling
    {
        public void Save()
        {
            if (IsPermissionGranted(new FileIOPermission(FileIOPermissionAccess.Write, @"c:\newfile.txt")))
            {
                // Write to local disk
                using (FileStream stream = File.Create(@"c:\newfile.txt"))
                using (StreamWriter writer = new StreamWriter(stream))
                {
                    writer.WriteLine("I can write to local disk.");
                }
            }
            else
            {
                MessageBox.Show("I can't write to local disk.");
            }
        }

        // Detect whether or not this application has the requested permission
        bool IsPermissionGranted(CodeAccessPermission requestedPermission)
        {
            try
            {
                // Try and get this permission
                requestedPermission.Demand();
                return true;
            }
            catch
            {
                return false;
            }
        }
```

```
Imports System.IO
Imports System.IO.IsolatedStorage
Imports System.Security
Imports System.Security.Permissions
Imports System.Windows

Namespace SDKSample
    Public Class FileHandling
        Public Sub Save()
            If IsPermissionGranted(New FileIOPermission(FileIOPermissionAccess.Write, "c:\newfile.txt"))
Then
                ' Write to local disk
                Using stream As FileStream = File.Create("c:\newfile.txt")
                Using writer As New StreamWriter(stream)
                    writer.WriteLine("I can write to local disk.")
                End Using
                End Using
            Else
                MessageBox.Show("I can't write to local disk.")
            End If
        End Sub

        ' Detect whether or not this application has the requested permission
        Private Function IsPermissionGranted(ByVal requestedPermission As CodeAccessPermission) As Boolean
            Try
                ' Try and get this permission
                requestedPermission.Demand()
                Return True
            Catch
                Return False
            End Try
        End Function
```

```
    }
}
```

```
    End Class
End Namespace
```

如果应用程序没有所需的权限，则对的调用 Demand 将引发安全异常。如果没有引发异常，则表示已授予该权限。`IsPermissionGranted` 封装此行为，并 `true` 根据 `false` 需要返回或。

功能下降

对可从不同区域执行的代码而言，能够检测代码是否具有完成所需操作的权限是很有意义的。能够检测区域固然不错，但如果能够为用户提供替代方法，则要好得多。例如，完全信任应用程序通常使用户能够在所需的任何位置创建文件，而部分信任应用程序只能在独立存储中创建文件。如果用于创建文件的代码存在于完全信任(独立)应用程序和部分信任(浏览器托管的)应用程序共享的程序集中，并且这两个应用程序都希望用户能够创建文件，则共享代码应首先检测其是在部分信任环境还是完全信任环境中运行，然后才能在适当的位置创建文件。下面的代码对这两种情况进行了演示。

```
using System.IO;
using System.IO.IsolatedStorage;
using System.Security;
using System.Security.Permissions;
using System.Windows;

namespace SDKSample
{
    public class FileHandlingGraceful
    {
        public void Save()
        {
            if (IsPermissionGranted(new FileIOPermission(FileIOPermissionAccess.Write, @"c:\newfile.txt")))
            {
                // Write to local disk
                using (FileStream stream = File.Create(@"c:\newfile.txt"))
                using (StreamWriter writer = new StreamWriter(stream))
                {
                    writer.WriteLine("I can write to local disk.");
                }
            }
            else
            {
                // Persist application-scope property to
                // isolated storage
                IsolatedStorageFile storage = IsolatedStorageFile.GetUserStoreForApplication();
                using (IsolatedStorageFileStream stream =
                    new IsolatedStorageFileStream("newfile.txt", FileMode.Create, storage))
                using (StreamWriter writer = new StreamWriter(stream))
                {
                    writer.WriteLine("I can write to Isolated Storage");
                }
            }
        }

        // Detect whether or not this application has the requested permission
        bool IsPermissionGranted(CodeAccessPermission requestedPermission)
        {
            try
            {
                // Try and get this permission
                requestedPermission.Demand();
                return true;
            }
            catch
            {
                return false;
            }
        }
```

```
Imports System.IO
Imports System.IO.IsolatedStorage
Imports System.Security
Imports System.Security.Permissions
Imports System.Windows

Namespace SDKSample
    Public Class FileHandlingGraceful
        Public Sub Save()
            If IsPermissionGranted(New FileIOPermission(FileIOPermissionAccess.Write, "c:\newfile.txt"))
Then
                ' Write to local disk
                Using stream As FileStream = File.Create("c:\newfile.txt")
                Using writer As New StreamWriter(stream)
                    writer.WriteLine("I can write to local disk.")
                End Using
                End Using
            Else
                ' Persist application-scope property to
                ' isolated storage
                Dim storage As IsolatedStorageFile = IsolatedStorageFile.GetUserStoreForApplication()
                Using stream As New IsolatedStorageFileStream("newfile.txt", FileMode.Create, storage)
                Using writer As New StreamWriter(stream)
                    writer.WriteLine("I can write to Isolated Storage")
                End Using
                End Using
            End If
        End Sub

        ' Detect whether or not this application has the requested permission
        Private Function IsPermissionGranted(ByVal requestedPermission As CodeAccessPermission) As Boolean
            Try
                ' Try and get this permission
                requestedPermission.Demand()
                Return True
            Catch
                Return False
            End Try
        End Function
```

```
    }
}
```

```
    End Class
End Namespace
```

在很多情况下，应该能够找到部分信任替代方法。

在受控环境（例如 intranet）中，可在客户端基础上将自定义托管框架安装到全局程序集缓存 (GAC) 。这些库可以执行需要完全信任的代码，并使用 (从仅允许部分信任的应用程序中进行引用 AllowPartiallyTrustedCallersAttribute 。有关详细信息，请参阅 安全性 和 WPF 安全策略-平台安全性) 。

浏览器主机检测

当你需要按权限检查时，使用 CA 检查权限是一种合适的方法。然而，这一技巧依赖于在正常处理过程中捕获异常（通常不鼓励这样做），并且可能导致性能问题。相反，如果 XAML 浏览器应用程序 (XBAP) 只在 Internet 区域沙盒中运行，则可以使用 BrowserInteropHelper.IsBrowserHosted 属性，该属性对于 XAML 浏览器应用程序 (xbap) 返回 true。

NOTE

IsBrowserHosted 仅区分应用程序是否在浏览器中运行，而不区分应用程序运行时使用的权限集。

管理权限

默认情况下，Xbap 以部分信任的方式运行 (默认 Internet 区域权限集) 。但是，根据应用程序的要求，可以更改默认的权限集。例如，如果从本地 intranet 启动 Xbap，它可以利用增加的权限集，如下表所示。

表 3：LocalIntranet 和 Internet 权限

		LOCALINTRANET	INTERNET
DNS	访问 DNS 服务器	是	否
环境变量	读取	是	否
文件对话框	打开	是	是
文件对话框	非受限	是	否
独立存储	按用户隔离程序集	是	否
独立存储	未知隔离	是	是
独立存储	无限制用户配额	是	否
媒体	安全音频、视频和图像	是	是
打印	默认打印	是	否
打印	安全打印	是	是
反射	发出	是	否
安全性	托管代码执行	是	是
安全性	声明授予的权限	是	否
用户界面	非受限	是	否
用户界面	安全顶级窗口	是	是
用户界面	自己的剪贴板	是	是
Web 浏览器	HTML 中的安全框架导航	是	是

NOTE

如果由用户启动，则剪切和粘贴只允许以部分信任方式执行。

如果需要增加权限，则需要更改项目设置和 ClickOnce 应用程序清单。有关详细信息，请参阅 WPF XAML 浏览器应用程序概述。以下各个文档可能也会有帮助。

- Mage.exe (清单生成和编辑工具) 。
- MageUI.exe (清单生成和编辑工具，图形客户端) 。
- 保护 ClickOnce 应用程序。

如果 XBAP 需要完全信任，可以使用同一工具来增加请求的权限。尽管 XBAP 仅在从本地计算机、Intranet 或浏览器的受信任站点或允许的站点中列出的 URL 上安装和启动时，才接收完全信任。如果从 Intranet 或受信任站点安装应用程序，则用户会收到标准 ClickOnce 提示，通知用户提升了权限。用户可以选择继续或取消安装。

或者，可以使用 ClickOnce 受信任部署模型从任何安全区域中进行完全信任部署。有关详细信息，请参阅受信任的应用程序部署概述和安全。

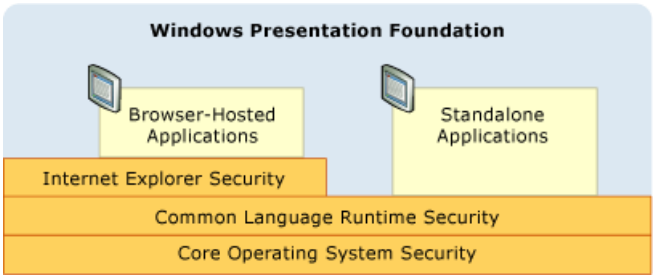
请参阅

- 安全性
- WPF 安全策略 - 平台安全性
- WPF 安全策略 - 安全工程

WPF 安全策略 - 平台安全性

2022/2/12 •

虽然 Windows Presentation Foundation (WPF) 提供各种安全服务，但它还利用基础平台（包括操作系统、CLR 和 Internet Explorer）的安全功能。这些层组合在一起为 WPF 提供强大的深层防御安全模型，尝试避免任何单点故障，如下图所示：



本主题的其余部分将讨论专门适用于 WPF 的每个层中的功能。

操作系统安全性

Windows 的核心提供了几种安全功能，这些功能构成了所有 Windows 应用程序（包括用 WPF 构建的应用程序）的安全基础。本主题讨论了对 WPF 重要的这些安全功能的广度，以及 WPF 如何与它们集成以提供进一步的深层防御。

Microsoft Windows XP Service Pack 2 (SP2)

除了对 Windows 进行一般检查和强化外，本主题还介绍 Windows XP SP2 的三个主要功能：

- /GS 编译
- Microsoft Windows 更新。

/GS 编译

WindowsXP SP2 通过重新编译许多核心系统库（包括所有 WPF 依赖项（如 CLR）来提供保护，以帮助缓解缓冲区溢出。通过使用/GS 形参和 C/C++ 命令行编译器可实现这一点。虽然应显式避免缓冲区溢出，但 /GS 编译针对由它们无意或恶意创建的潜在漏洞提供了深层防御示例。

以前，缓冲区溢出已导致出现了许多影响较大的安全漏洞。当攻击者利用代码漏洞时就会发生缓冲区溢出，代码漏洞可让注入的恶意代码通过缓冲区边界写入。从而让攻击者可以通过重写导致执行攻击者代码的函数返回地址执行代码进程。结果，恶意代码可以执行具有截获进程相同特权的任意代码。

在高级别上，-GS 编译器标志通过注入特殊安全 cookie 来保护具有本地字符串缓冲区的函数的返回地址，从而防止某些潜在的缓冲区溢出。函数返回后，安全 cookie 将与其上一个值进行比较。如果值已更改，可能已发生缓冲区溢出，并且该进程已停止并显示错误条件。停止的进程将阻止执行潜在的恶意代码。有关更多详细信息，请参阅 -GS (缓冲区安全检查) 。

WPF 用/GS 标志进行编译，以便向 WPF 应用程序添加另一层防御。

Windows Vista

Windows Vista 上的 WPF 用户将受益于操作系统的附加安全增强功能，其中包括 "最小特权用户访问权限"、代码完整性检查和特权隔离。

用户帐户控制 (UAC)

目前，Windows 用户常常会以管理员权限运行，因为许多应用程序都需要它们进行安装或执行，或者两者都需要。其中一个示例就是，可以将默认应用程序设置写入到注册表。

使用管理员特权运行实际上就是指应用程序从授予管理员特权的进程执行。此方法的安全影响在于，可截获使用管理员特权运行的进程的任何恶意代码都将自动继承这些特权，包括对关键系统资源的访问权限。

保护计算机免受此安全威胁的一种方法就是使用所需的最少特权数运行应用程序。这称为最小特权原则，是 Windows 操作系统的核心功能。此功能称为用户帐户控制 (uac)，并通过两种主要方式 Windows uac 使用：

- 若要在默认情况下使用 UAC 特权运行大多数应用程序，即使用户是管理员，也只有需要使用管理员特权的应用程序才会使用管理员特权运行。要使用管理特权运行，必须以应用程序清单形式或作为安全策略中的一个条目显式标记应用程序。
- 提供兼容性解决方案(如虚拟化)。例如，许多应用程序尝试写入受限位置，例如 C:\Program Files。对于在 UAC 中执行的应用程序，存在基于用户的可选位置无需管理员特权就能写入。对于在 UAC 中运行的应用程序，UAC 可虚拟化 C:\Program Files，这样认为其写入到其中的应用程序实际上是写入到基于用户的可选位置。这种兼容性工作可使操作系统来运行许多以前无法在 UAC 中运行的应用程序。

代码完整性检查

WindowsVista 合并了更深入的代码完整性检查，以帮助防止恶意代码在负载/运行时注入到系统文件或内核中。这超出了系统文件保护。

浏览器承载的应用程序的受限权限进程

浏览器承载的 WPF 应用程序在 Internet 区域沙箱内执行。与 Microsoft Internet Explorer 的 WPF 集成扩展了此保护，并提供了更多支持。

由于 XAML 浏览器应用程序 (Xbap) 通常由 Internet 区域权限集进行沙盒处理，因此，删除这些权限不会损害 XAML 浏览器应用程序 (Xbap) 从兼容性角度来看。反而会创建一个附加的深层防御层；如果经过沙箱处理的应用程序能够利用其他层截获此进程，该进程将仍然只有有限特权。

请参阅 使用 Least-Privileged 的用户帐户。

公共语言运行时的安全性

公共语言运行时 (CLR) 提供了许多重要的安全优势，其中包括验证和验证、代码访问安全性 (CAS) 和安全关键方法。

确认和验证

为了提供程序集隔离和完整性，CLR 使用验证过程。CLR 验证通过验证程序集的可移植可执行文件 (PE) 文件格式来确定是否隔离程序集。CLR 验证还验证嵌入到程序集内的元数据的完整性。

为了确保类型安全，帮助防止常见安全问题 (例如缓冲区溢出)，并通过子进程隔离启用沙盒，CLR 安全使用验证概念。

托管应用程序被编译为 Microsoft 中间语言 (MSIL)。当执行托管应用程序中的方法时，将采用实时 (JIT) 编译方式把 MSIL 编译为本机代码。JIT 编译包括的验证过程将应用许多众多安全和可靠规则，从而确保代码不会：

- 违反类型合约
- 引入缓冲区溢出
- 随意访问内存。

不允许不符合验证规则的托管代码执行，除非它被视为受信任代码。

可验证代码的优点是 WPF 在 .NET Framework 上构建的主要原因。从使用验证代码而言，利用潜在漏洞的可能性明显降低。

代码访问安全性

客户端计算机公开了托管应用程序可以访问的各种资源，包括文件系统、注册表、打印服务、用户界面、反射和环境变量。在托管应用程序可以访问客户端计算机上的任何资源之前，它必须具有 .NET Framework 的权限。CA 中的权限是的子类 CodeAccessPermission ;CA 为托管应用程序可以访问的每个资源实现一个子类。

CA 在开始执行时授予托管应用程序的权限集称为权限集，由应用程序提供的证据确定。对于 WPF 应用程序，提供的证据为从中启动应用程序的位置或区域。CA 标识以下区域：

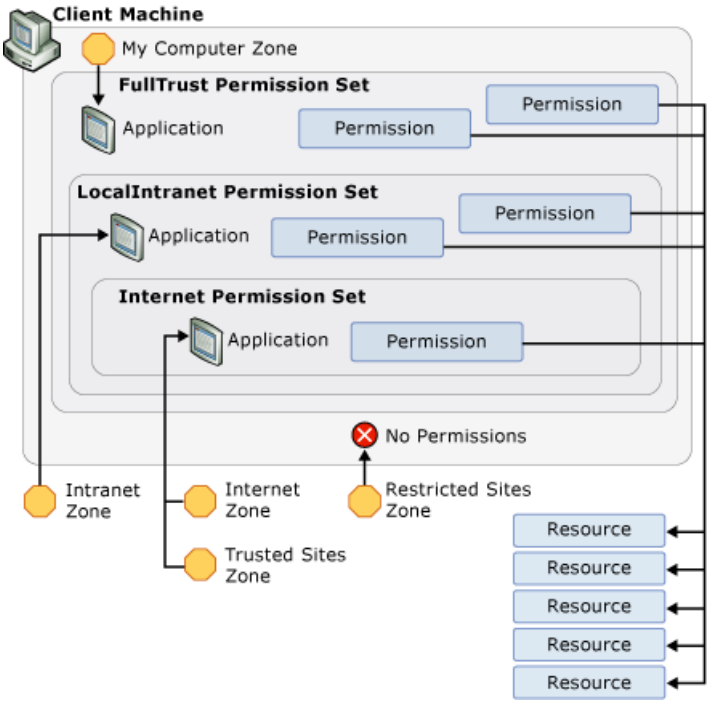
- **我的**电脑。从客户端计算机（完全受信任）上启动的应用程序。
- **本地 Intranet**。从 Intranet 启动的应用程序。（某种程度上受信任）。
- **Internet**。从 Intranet 启动的应用程序。（最不受信任）。
- **受信任的站点**。由受信任用户标识的应用程序。（最不受信任）。
- **不受信任的站点**。由不受信任的用户标识的应用程序。（不受信任）。

对于其中的每个区域，CA 都提供预定义的权限集，该权限集包含与每个相关联的信任级别相匹配的权限。其中包括：

- **FullTrust**。对于从 **我的**电脑 区域启动的应用程序。将授予全部可能的权限。
- **LocalIntranet**。对于从 **本地 Intranet** 区域启动的应用程序。将授予权限的子集，以提供对客户端计算机资源适度的访问权限，包括隔离存储、用户界面的无限制访问、无限制使用文件对话框、有限的反射和有限访问环境变量。不提供对关键资源（如注册表）的权限。
- **Internet**。对于从 Internet或受信任的 **站点区域启动的应用程序**。将授予权限的子集，以提供对客户端计算机资源有限的访问权限，包括隔离存储、仅限打开文件和有限的用户界面。实质上，此权限集将应用程序与客户端计算机隔离。

CAS 不会向标识为来自不 **受信任的站点** 区域的应用程序授予任何权限。因此，对它们而言，就不存在预定义的权限集。

下图说明了区域、权限集、权限和资源之间的关系：



Internet 区域安全沙盒的限制同样适用于 XBAP 从系统库（包括 WPF）导入的任何代码。这可确保锁定代码的每一位，甚至是 WPF。遗憾的是，为了能够执行，XBAP 需要执行所需的权限比 Internet 区域安全沙盒启用的权限更多的功能。

请考虑包含以下页面的 XBAP 应用程序：

```
FileIOPermission fp = new FileIOPermission(PermissionState.Unrestricted);
fp.Assert();

// Perform operation that uses the assert

// Revert the assert when operation is completed
CodeAccessPermission.RevertAssert();
```

```
Dim fp As New FileIOPermission(PermissionState.Unrestricted)
fp.Assert()

' Perform operation that uses the assert

' Revert the assert when operation is completed
CodeAccessPermission.RevertAssert()
```

若要执行此 XBAP，基础 WPF 代码必须执行的功能必须多于调用 XBAP 可用的功能，包括：

- 使用 HWND (窗口句柄) 呈现
- 调度消息
- 加载 Tahoma 字体

从安全角度而言，允许从沙盒应用程序直接访问上述任何操作将会导致灾难性后果。

幸运的是，WPF 通过允许这些操作代表沙盒应用程序以提升的权限执行，从而满足这种情况。虽然针对 XBAP 的应用程序域的受限 Internet 区域安全权限检查所有 WPF 操作，但 WPF (与其他系统库一样) 会被授予包含所有可能的权限的权限集。

这要求 WPF 接收提升的权限，同时防止这些特权受主机应用程序域的 Internet 区域权限集控制。

WPF 通过使用权限的 Assert 方法实现此要求。以下代码演示了这种方法。

```
FileIOPermission fp = new FileIOPermission(PermissionState.Unrestricted);
fp.Assert();

// Perform operation that uses the assert

// Revert the assert when operation is completed
CodeAccessPermission.RevertAssert();
```

```
Dim fp As New FileIOPermission(PermissionState.Unrestricted)
fp.Assert()

' Perform operation that uses the assert

' Revert the assert when operation is completed
CodeAccessPermission.RevertAssert()
```

Assert 实质上可防止 WPF 所需的无限制权限受到 XBAP 的 Internet 区域权限的限制。

从平台的角度来看，WPF 负责正确使用 Assert; 错误使用 Assert 可能导致恶意代码提升特权。因此，必须仅根据需要调用 Assert，并确保沙盒限制保持不变。例如，禁止沙盒代码打开任意文件，但允许其使用字体。WPF 允许沙盒应用程序通过调用 Assert 来使用字体功能，对于 WPF，可以代表沙盒应用程序读取已知包含这些字体的文件。

ClickOnce 部署

ClickOnce是一种全面的部署技术，.NET Framework与 Visual Studio (集成，请参阅 ClickOnce 安全和部署，了解) 。独立 WPF 应用程序可以使用 ClickOnce 部署，而浏览器托管的应用程序必须使用 ClickOnce。

使用 CLICKONCE 部署的应用程序在 CAS (上具有额外的) ;实质上，ClickOnce应用程序请求所需的权限。如果它们不超过在其中部署应用程序的区域的权限集，几乎仅授予它们这些权限。通过仅将权限集减少到所需的权限集，即使它们小于启动区域的权限集提供的权限集，应用程序有权访问的资源数将减少到最少。因此，如果截获到应用程序，将可以降低对客户端计算机的潜在损坏几率。

安全-关键方法

使用权限为 XBAP 应用程序启用 Internet 区域沙盒的 WPF 代码必须受到尽可能高的安全审核和控制。为了简化此要求，.NET Framework为管理提升特权的代码提供新的支持。具体而言，CLR 使你能够识别提升特权的代码，然后使用 将其标记为 ;任何未标记为 的代码都使用此方法 SecurityCriticalAttribute SecurityCriticalAttribute 变得透明。反之，禁止未标有 SecurityCriticalAttribute 的托管代码提升特权。

使用Security-Critical方法可以组织 WPF 代码，以将特权提升为安全关键内核，其余代码是透明的。隔离安全关键代码可使 WPF 工程团队将额外的安全分析和源代码管理集中在标准安全做法之上和之外的安全关键内核上 (请参阅 WPF 安全策略 - 安全工程) 。

请注意，.NET Framework允许受信任的代码扩展 XBAP Internet 区域沙盒，允许开发人员编写使用 (APTCA) 标记并部署到用户的全局程序集缓存 AllowPartiallyTrustedCallersAttribute (GAC) 的托管程序集。将程序集标记为 APTCA 是高度敏感的安全操作，因为它允许任何代码调用该程序集，包括来自 Internet 的恶意代码。执行此操作时，要特别注意并且必须采用最佳做法，用户必须选择信任该软件才能完成安装。

Microsoft Internet Explorer 安全

除了减少安全问题和简化安全配置外，Microsoft Internet Explorer 6 (SP2) 还包含多项功能，这些功能可增强 XAML 浏览器应用程序 (XBAP) 用户的安全性。这些功能的主旨是尝试允许用户更好地控制它们的浏览体验。

在 IE6 SP2 之前，用户可能需遵守以下任一条件：

- 随机弹出窗口。
- 混淆的脚本重定向。
- 某些网站上出现大量安全对话框。

在某些情况下，不受信任的网站会尝试通过欺骗安装或反复显示 Microsoft ActiveX 安装对话框来欺骗用户，即使用户已 用户界面 (UI) 取消安装。使用这些技术，有可能会有相当多的用户上当受骗，从而导致安装间谍软件应用程序。

IE6 SP2 包括几个功能来缓解这些类型的问题，这围绕用户启动的概念。IE6 SP2 检测用户何时在操作（称为用户启动）之前单击了链接或页面元素，并像页面上的脚本改为触发类似操作时一样处理它。例如，IE6 SP2 包含一个弹出窗口阻止程序，用于检测用户在创建弹出窗口之前单击按钮时。这使 IE6 SP2 能够允许大多数无影响弹出窗口，同时阻止用户既不要求也不想要弹出的弹出窗口。阻止的弹出窗口被捕获到新的信息栏 下，这允许用户手动重写 块并查看弹出窗口。

相同的用户启动逻辑也适用于"**打开保存 / "** 安全提示。ActiveX安装对话框始终捕获在信息栏下，除非它们表示从以前安装的控件升级。这些度量值组合在一起，可提供用户更安全、更可控的用户体验，因为诱导他们安装不需要的软件或恶意软件的站点受到了保护。

这些功能还保护使用 IE6 SP2 浏览允许他们下载和安装 WPF 应用程序的网站的客户。具体而言，这是因为 IE6 SP2 提供了更好的用户体验，减少了用户安装恶意或恶意应用程序的可能性，而不管使用什么技术（包括 WPF）来构建它。WPF 通过使用 ClickOnce来添加这些保护，以便通过 Internet 下载其应用程序。由于 XAML 浏览器应用程序 (XBAP) Internet 区域安全沙盒中执行，因此可以无缝启动这些应用程序。另一方面，独立 WPF 应用程序需要完全信任来执行。对于这些应用程序，ClickOnce在启动过程中显示一个安全对话框，以通知应用程序的其他安全要求的使用。但是，必须由用户启动，必须由用户启动的逻辑进行管理并且可以取消。

Internet Explorer 7 包含并扩展了 IE6 SP2 的管理功能，这是持续承诺安全性的一部分。

请参阅

- 代码访问安全性
- 安全性
- WPF 部分信任安全性
- WPF 安全策略 - 安全工程

WPF 安全策略 - 安全工程

2022/2/12 •

可信计算是 Microsoft 为确保生成安全代码而首创的一项技术。可信计算计划的关键要素是 (SDL) 的 Microsoft 安全开发生命周期。SDL 是一种工程实践，与标准工程流程结合使用，以促进安全代码的交付。SDL 包含10个阶段，其中包含规范化结合、可度量性和其他结构的最佳实践，其中包括：

- 安全设计分析
- 基于工具的质量检查
- 渗透测试
- 最终安全评审
- 发布后产品安全管理

WPF 详细信息

WPF工程团队共同应用和扩展 SDL，其中包括以下重要方面：

威胁建模

安全分析和编辑工具

测试技术

关键代码管理

威胁建模

威胁建模是 SDL 的核心组件，用于分析系统，以确定潜在的安全漏洞。一旦确定漏洞，威胁模型还可以确保采取适当的缓解措施。

我们以一个杂货店为例，说明威胁模型在高级别上所涉及的以下关键步骤：

1. **确定**资产。杂货店的资产可能包括员工、保险箱、收银机和库存。
2. **枚举入口点**。杂货店的入口点可能包括前门和后门、窗户、装货区和空调设备。
3. **使用入口点**调查针对资产**的攻**击。可能进行的攻击包括通过空调入口点来对杂货店的保险箱资产进行攻击；有人可能会将空调设备拆掉，将保险箱通过空调处拉出杂货店。

威胁建模应用于整个 WPF，包含以下各项：

- XAML 分析器读取文件、将文本映射到相应的对象模型类并创建实际代码的方式。
- 创建窗口句柄 (hWnd) 并通过其发送消息和呈现窗口内容的方式。
- 数据绑定获取资源以及与系统交互的方式。

这些威胁模型对于在开发过程中确定安全设计需求以及缓解威胁非常重要。

源分析和编辑工具

除了 SDL 的手动安全代码评审元素外，WPF 团队使用多个工具进行源分析和关联编辑以减少安全漏洞。使用了多种源工具，包括以下各项：

- **FXCop**：查找托管代码中的常见安全问题，包括继承规则、代码访问安全性的使用情况以及安全地与非托管代码交互操作的方式。请参阅FXCop。
- **Prefix/Prefast**：查找非托管代码中的安全漏洞和常见安全问题，例如缓冲区溢出、格式字符串问题以及错误检查。
- **已禁止的 API**：搜索源代码，以识别出众所周知的因意外使用而引发安全问题的函数，例如 `strcpy`。一旦确定，这些函数将被替换为更安全的替代项。

测试技术

WPF 使用多种安全测试技术，包括：

- **白盒测试**：测试人员查看源代码，然后构建利用测试。
- **黑盒测试**：测试人员尝试通过检查 API 和功能来查找安全问题，然后尝试对产品进行攻击。
- **对来自其他产品的安全问题进行回归测试**：对来自相关产品的相关安全问题进行测试。例如，已识别出 Internet Explorer 的大约60安全问题的适当变体，并尝试将其适用性用于 WPF 。
- **借助文件模糊化执行基于工具的渗透测试**：文件模糊化指利用文件读取器多种输入的输入范围。在 WPF 中的一个示例，使用此技术的一个示例就是检查图像解码代码的错误。

关键代码管理

对于 XAML 浏览器应用程序 (Xbap)， WPF 通过使用 .NET Framework 对标记和跟踪安全关键代码的支持来生成安全沙盒，(请参阅 WPF 安全策略-平台) 安全性中的 **安全关键方法**。考虑到对安全关键代码有较高的安全质量要求，因此需要对此类代码进行其他级别的源管理控制和安全审核。大约有 5% 到 10% 的 WPF 由安全关键代码组成，这些代码由专门的审核团队进行审核。通过跟踪安全关键代码和将每个关键实体(即，包含关键代码的方法)映射到其签署状态来对源代码和签入过程进行管理。签署状态包括一个或多个审阅者的姓名。WPF 的每个日常版本都将关键代码与前一版本中的该代码进行比较，以检查未经审批的更改。如果工程师未经审核团队的批准而自行修改关键代码，则将识别并立即修复该代码。通过这一过程，可以对 WPF 沙盒代码应用级别特高的审核并加以维护。

请参阅

- 安全性
- WPF 部分信任安全
- WPF 安全策略 - 平台安全性
- 可信计算
- .NET 中的安全性

WPF 示例

2022/2/12 •

有关演示 Windows Presentation Foundation (WPF) 的示例，请参阅 GitHub 上的Microsoft/WPF 示例存储库。

类库 (WPF)

2022/2/12 •

以下链接引用包含 api 的命名空间 Windows Presentation Foundation (WPF) 。

本节内容

参考

- Microsoft.Build.Tasks.Windows
- Microsoft.Win32 (共享)
- Microsoft.Windows.Themes
- System.Collections.ObjectModel (共享)
- System.Collections.Specialized (共享)
- System.ComponentModel (共享)
- System.Diagnostics (共享)
- System.IO (共享)
- System.IO.Packaging
- System.Printing
- System.Printing.IndexedProperties
- System.Printing.Interop
- System.Security.Permissions (共享)
- System.Security.RightsManagement
- System.Windows
- System.Windows.Annotations
- System.Windows.Annotations.Storage
- System.Windows.Automation
- System.Windows.Automation.Peers
- System.Windows.Automation.Provider
- System.Windows.Automation.Text
- System.Windows.Controls
- System.Windows.Controls.Primitives
- System.Windows.Converters
- System.Windows.Data

- System.Windows.Documents
- System.Windows.Documents.DocumentStructures
- System.Windows.Documents.Serialization
- System.Windows.Forms.Integration
- System.Windows.Ink
- System.Windows.Input
- System.Windows.Input.StylusPlugIns
- System.Windows.Interop
- System.Windows.Markup (共享)
- System.Windows.Markup.Localizer
- System.Windows.Markup.Primitives
- System.Windows.Media
- System.Windows.Media.Animation
- System.Windows.Media.Converters
- System.Windows.Media.Effects
- System.Windows.Media.Imaging
- System.Windows.Media.Media3D
- System.Windows.Media.Media3D.Converters
- System.Windows.Media.TextFormatting
- System.Windows.Navigation
- System.Windows.Resources
- System.Windows.Shapes
- System.Windows.Threading
- System.Windows.Xps
- System.Windows.Xps.Packaging
- System.Windows.Xps.Serialization
- UIAutomationClientsideProviders

.NET 4 中的 XAML 支持

以下命名空间包含来自 System.object 程序集的类型。对于在 .NET Framework 4 上构建的 WPF（如 WPF），.Xaml 提供常见的 XAML 语言支持。

- System.Windows.Markup (共享)
- System.Xaml
- System.Xaml.Permissions

- System.Xaml.Schema