一、属性设置几种方式

属性：

属性是对XAML元素特征进行描述的方法；属性不允许在XAML中重复设置多次；允许在托管代码中改变元素的属性值

设置几种方式：

- 使用属性语法
- 使用属性元素语法
- 使用内容元素语法
- 使用集合语法

1、使用属性语法

每个属性对应一个属性值，属性值类型必须与属性匹配

一个标记中可以设置对象的多个属性

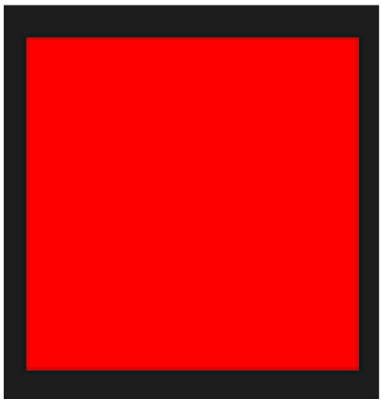
只有实例化对象才可以设置实例属性，格式如下：

```
<objectName propertyName="propertyValue"
propertyName1="propertyValue1"/>
或者
<objectName propertyName="propertyValue"
propertyName1="propertyValue1"></objectName>
```

例子：

```
<Canvas  Width="150" Height="150" Background="Red"/>
或者
<Canvas Width="150" Height="150" Background="Red"></Canvas>
```

例子效果：



2、使用属性元素语法

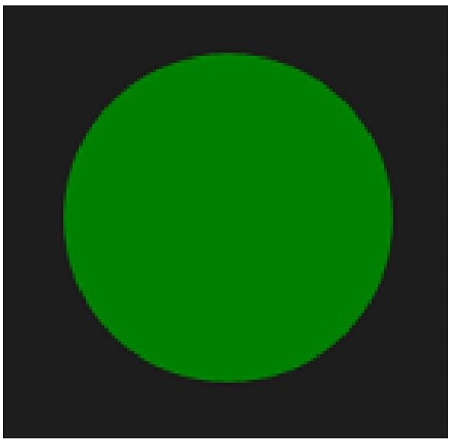
某些属性可以使用属性元素语法来设置，格式为

```
<object>
    <object.property>
        <!--元素属性值-->
    </object.property>
</object>
```

例子：

```
<Ellipse Width="150" Height="150">
    <!--设置填充颜色-->
    <Ellipse.Fill>
        <SolidColorBrush Color="Green"/>
    </Ellipse.Fill>
</Ellipse>
```

例子效果：



3、使用内容元素语法

某些元素的属性支持内容元素语法，允许忽略元素的名称

实力对象会根据XAML元素中的第一个标记值来设置属性

对于大量的格式化文本，使用内容元素语法更加灵活

属性标记之间可以插入大量的文本内容

例子：

```
<TextBlock Width="200" TextWrapping="NoWrap">
    某些元素的属性支持内容元素语法，允许忽略元素的名称
    实力对象会根据XAML元素中的第一个标记值来设置属性
    对于大量的格式化文本，使用内容元素语法更加灵活
    属性标记之间可以插入大量的文本内容
</TextBlock>
```

4、使用集合语法

元素支持一个属性元素的集合，才使用集合语法进行设置属性

使用托管代码的Add方法来增加更多的集合元素

本质是向对象的集合中添加属性项

例子：

```
<!--方式一-->
<Rectangle Width="200" Height="150">
```

```
    <Rectangle.Fill>
        <!--线性梯度画刷：-->
        <LinearGradientBrush>
            <GradientStopCollection>
                <!--属性集合语法-->
                <GradientStop Offset="0.0" Color="Gold"/>
                <GradientStop Offset="1.0" Color="Green"/>
            </GradientStopCollection>
        </LinearGradientBrush>
    </Rectangle.Fill>
</Rectangle>

<!--方式二-->
<Rectangle Width="200" Height="150">
    <Rectangle.Fill>
        <!--线性梯度画刷：-->
        <LinearGradientBrush>
            <!--省略GradientStopCollection隐式的属性设置方法-->
            <LinearGradientBrush.GradientStops>
                <!--属性集合语法-->
                <GradientStop Offset="0.0" Color="Gold"/>
                <GradientStop Offset="1.0" Color="Green"/>
            </LinearGradientBrush.GradientStops>
        </LinearGradientBrush>
    </Rectangle.Fill>
</Rectangle>
```

例子效果：

二、基本属性、附加属性和依赖属性

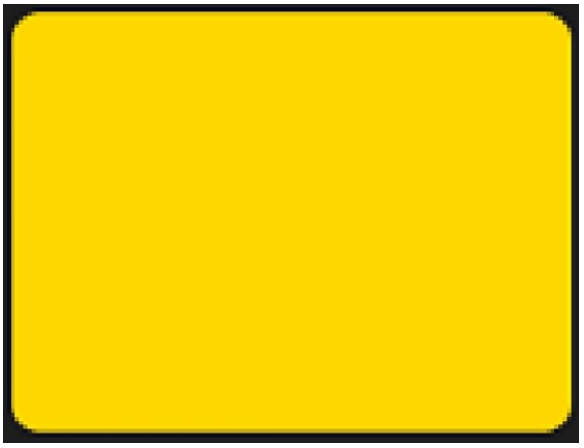
1、附加属性

- 附加属性作用于支持附加属性的元素
- 附加属性是由支持附加属性的父元素产生作用，支持附加属性的元素会继承所在的父元素的属性
- 附加属性的格式：AttachedPropertyProvider.PropertyName

例子：

```
<!--画布面板-->
<Canvas>
    <!--附加属性Canvas.Left-->
    <Rectangle Canvas.Left="50" Canvas.Top="50" Width="200" Height="150" RadiusX="10" RadiusY="10" Fill="Gold"/>
</Canvas>
```

例子效果：



2、依赖属性

- 英文名称：Dependency Properties
- 依赖属性和CRL属性类似，提供一个实例级**私有字段的访问封装**，通过GetValue和SetValue访问器实现属性的读写操作
- 最重要一个特点是属性值依赖于一个或者多个数据源，提供这些数据源的方式也可以不同
- 由于依赖多数据源的缘故，故称之为依赖属性

一、什么是依赖属性

依赖属性就是一种自己可以没有值，并且可以通过绑定从其他数据源获取值。依赖属性可支持WPF中的样式设置、数据绑定、继承、动画及默认值。

属性的场景：

1. 希望可在样式中设置属性。

2. 希望属性支持数据绑定。

3. 希望可使用动态资源引用设置属性。

4. 希望从元素树中的父元素自动继承属性值。

5. 希望属性可进行动画处理。

6. 希望属性系统在属性系统、环境或用户执行的操作或者读取并使用样式更改了属性以前的值时报告。

7. 希望使用已建立的、WPF 进程也使用的元数据约定，例如报告更改属性值时是否要求布局系统重新编写元素的可视化对象。

二、依赖属性的特点

无论什么时候，只要依赖属性的值发生改变，wpf就会自动根据属性的元数据触发一系列的动作，这些动作可以重新呈现UI元素，也可以更新当前的布局，刷新数据绑定等等，这种变更的通知最有趣的特点之一就是属性触发器，它可以在属性值改变的时候，执行一系列自定义的动作，而不需要更改任何其他的代码来实现。通过下面的示例来演示属性变更通知

示例：当鼠标移动到Button按钮上面时，文字的前景色变为红色，离开时变为默认颜色黑色，采用传统方式和依赖属性两种方式实现：

使用传统方式实现:XAML代码

```
<Window x:Class="WpfDemo.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Grid面板" Height="237" Width="525"
WindowStartupLocation="CenterScreen">
    <Grid >
        <Button Height="30" Width="200"
MouseEnter="Button_MouseEnter" MouseLeave="Button_MouseLeave" >鼠标移动
到上面，前景色变为红色</Button>
```

```
    </Grid>
</Window>

```

C#后台代码实现：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace WpfDemo
{
    /// <summary>
    /// MainWindow.xaml 的交互逻辑
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        /// <summary>
        /// 鼠标移动到按钮上面
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void Button_MouseEnter(object sender, MouseEventArgs
```

```
e)
        {
            //创建Button对象
            /*sender 是 private void Button_MouseEnter(object sender,
MouseEventArgs e) 传入的参数
             as 是类型转换
             sender as Button 意思就是将 sender这个object对象转换为
button对象
            */
             Button btn = sender as Button;
             if (btn != null)
             {
                 btn.Foreground = Brushes.Red;
             }
        }

        /// <summary>
        /// 鼠标离开按钮
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void Button_MouseLeave(object sender, MouseEventArgs
e)
        {
             Button btn = sender as Button;
             if (btn != null)
             {
                 btn.Foreground = Brushes.Black;
             }
        }
    }
}
```

（**2**）使用依赖属性实现，**XAML**界面代码：

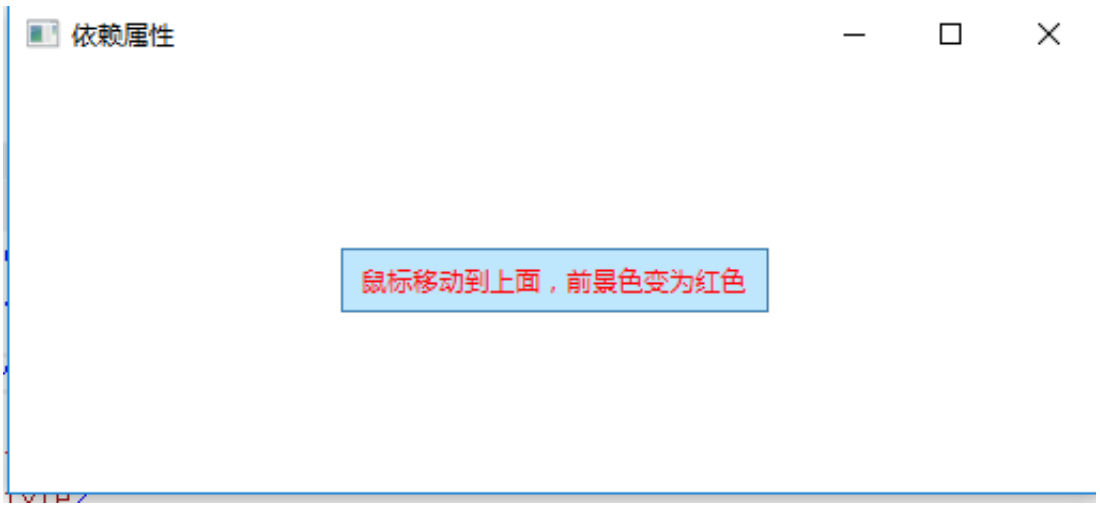
```
<Window x:Class="WpfDemo.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Grid面板" Height="237" Width="525"
```

```
WindowStartupLocation="CenterScreen">
    <Grid >
        <Button Height="30" Width="200">鼠标移动到上面，前景色变为红色
            <Button.Style>
                <Style TargetType="Button">
                    <!--样式触发器-->
                    <Style.Triggers>
                        <!--IsMouseOver(鼠标悬停)鼠标位于控件上-->
                        <Trigger Property="IsMouseOver" Value="true">
                            <!--字体颜色红色-->
                            <Setter Property="Foreground" Value="Red"
/>
                            <!--Cursor光标属性：手型的光标-->
                            <Setter Property="Cursor" Value="Hand"/>
                        </Trigger>
                    </Style.Triggers>
                </Style>
            </Button.Style>
        </Button>
    </Grid>
</Window>

```

使用上面的两种方式都可以实现Button按钮的前景色改变，效果如下：



依赖属性

可以通过多种不同类型的数据源进行赋值，不同赋值顺序影响属性值的改变

示例：XAML代码

```
<!--定义窗口私有资源-->
    <Window.Resources>
        <!--样式资源-->
        <Style x:Key="ButtonStyle" TargetType="Button">
            <!--背景色-->
            <Setter Property="Background" Value="AliceBlue"/>
            <!--字体大小-->
            <Setter Property="FontSize" Value="24"/>
        </Style>
    </Window.Resources>
   <!--页面布局-->
    <Grid>
        <!--依赖属性：绑定Style="{StaticResource ButtonStyle}"-->
        <Button Content="依赖属性测试" Style="{StaticResource ButtonStyle}" Height="80"/>
        <!--依赖属性：优先级(属性默认复制优先Background="Yellow" FontSize="39")-->
        <Button Content="依赖属性优先级" Style="{StaticResource ButtonStyle}" Background="Yellow" FontSize="39"  Height="80" Margin="0,205,0.4,35.8"/>
    </Grid>

```

例子效果

依赖属性测试

依赖属性优先级