# XAML高级教程

郑贵锋


通过这一节课，可以使听众对XAML的资源和样式，绘图，图像处理和动画的基本概念有了更深的理解，掌握如何从代码层面实现。

# 大纲

1. 资源&样式
2. 高级绘图
3. 图像处理
4. 动画

# 资源&样式

# 资源

- 与传统WEB应用中CSS样式表类似
- 目的为了实现对象的重复调用
- 有助于XAML代码重用，有助于应用维护的一致性
- 定义资源的语法格式：

```
<根元素对象.Resources>
        <资源定义 />
</根元素对象.Resources>
```

# 资源

```
<Grid>
   <Grid.Resources>

<LinearGradientBrush x:Key="bgBrush" StartPoint="0.5,0" EndPoint="0.5,1">
            <GradientStop Color="Yellow" Offset="0.0" />
            <GradientStop Color="Blue" Offset="0.75" />
            <GradientStop Color="Green" Offset="1.0" />
        </LinearGradientBrush>
    </Grid.Resources>

    <Button>
       <Button.Resources>
                  ....
       </Button.Resources>
    </Button>

   ……
</Grid>
```

# 资源字典(ResourceDictionary)

```
<Grid x:Name="LayoutRoot">
   <Grid.Resources>
      <ResourceDictionary>
         <LinearGradientBrush x:Key="bgBrush" StartPoint=
"0.5,0" EndPoint="0.5,1">
            <GradientStop Color="Yellow" Offset="0.0" />
            <GradientStop Color="Blue" Offset="0.75" />
            <GradientStop Color="Green" Offset="1.0" />
         </LinearGradientBrush>
       </ResourceDictionary>
    </Grid.Resources>
    <Button x:Name="btnSubmit" Background="{StaticResourc
e bgBrush}" Height="60" Width="120" Margin="112,23,168,21
7"/>
</Grid>
```

# 资源字典分类

- WPF应用中，XAML资源分为StaticResource和DynamicResource
- Windows 8应用中，XAML资源仅支持StaticResource
- 资源应用域不同，XAML资源可分为FrameworkElement.Resources和Application.Resources
  - FrameworkElement.Resources是将资源对象应用于同一个对象树的不同对象上，称为页面资源，通常被定义在XAML页面根元素上
  - Application.Resources是贯穿整个应用级别的资源，通常被定义在App.xaml页面

# 合并资源字典属性

```
<Application.Resources>
   <ResourceDictionary>
      <SolidColorBrush Color="#d0157820" x:Key="m
uddyBrush"/>
      <ResourceDictionary.MergedDictionaries>
        <ResourceDictionary Source="rd1.xaml" />
        <ResourceDictionary Source="rd2.xaml" />
      </ResourceDictionary.MergedDictionaries>
   </ResourceDictionary>
</Application.Resources>
```

# 主题资源字典属性

```
<ResourceDictionary.ThemeDictionaries>
    <ResourceDictionary x:Key="Default">
        <x:String x:Key="BackButtonGlyph">  </x:String>
        <x:String x:Key="BackButtonSnappedGlyph">  </x:String>
     </ResourceDictionary>
     <ResourceDictionary x:Key="HighContrast">
        <x:String x:Key="BackButtonGlyph">  </x:String>
        <x:String x:Key="BackButtonSnappedGlyph">  </x:String>
     </ResourceDictionary>
     <ResourceDictionary x:Key="GreenTheme">
        <SolidColorBrush x:Key="MyBackgroundBrush" Color="Green"><
/SolidColorBrush>
     </ResourceDictionary>
     <ResourceDictionary x:Key="BlackTheme">
        <SolidColorBrush x:Key="MyBackgroundBrush" Color="Black"><
/SolidColorBrush>
     </ResourceDictionary>
</ResourceDictionary.ThemeDictionaries>
```

# 资源

# 演示

# 样式

| 属性样式 | 内联样式 | 引用样式 |
| --- | --- | --- |
| 直接通过UI元素的属性设置的样式 | 通过在UI元素中嵌入Style节点来设置样式 | 定义在资源字典中的样式 |

# 属性样式

- 类似于在HTML中直接通过HTML元素属性设置的样式
- 适用于复用度不高的一些属性

```
<TextBox Text="textbox" Width="200" Height="100"
Margin="40,40,40,40" HorizontalAlignment="Center"
VerticalAlignment="Top" Background="AliceBlue"/>
```

# 内联样式

- 实际上是设置元素的Style属性
- 若内联样式与属性样式对同一属性进行设置，以属性设置的样式为准

```
<TextBox Text="textbox" Width="200"
Height="100“ Margin="40,40,40,40" HorizontalAlignment="Center"
VerticalAlignment="Top" Background="AliceBlue">
   <TextBox.Style>
      <Style TargetType="TextBox">
         <Setter Property="BorderThickness" Value="3"/>
         <Setter Property="BorderBrush"  Value="Brown"/>
      </Style>
   </TextBox.Style>
</TextBox>
```

# 引用样式

- 资源字典可在多处定义，因此引用样式也可以在多处定义

- 引用样式可以分为：
  - 页面级样式
  - 应用程序级样式

# 页面级样式

页面级样式定义在页面的资源字典中，作用范围为整个页面

```
<phone:PhoneApplicationPage.Resources>
  <Style x:Key="pageStyle"  TargetType="TextBox">
    <Setter Property="FontSize" Value="40"/>
  </Style>
  <Style TargetType="TextBox">
    <Setter Property="Background" Value="LightBlue"/>
  </Style>
</phone:PhoneApplicationPage.Resources>

<TextBox Name="textBox3" Text="TextBox" Style="{StaticResource pageStyle}"
Width="400" Height="100" Margin="39,257,41,0" HorizontalAlignment="Center"
VerticalAlignment="Top" />
<TextBox Name="textBox4" Text="TextBox" Width="400" Height="100"
Margin="39,363,41,0" HorizontalAlignment="Center"  VerticalAlignment="Top" />
```

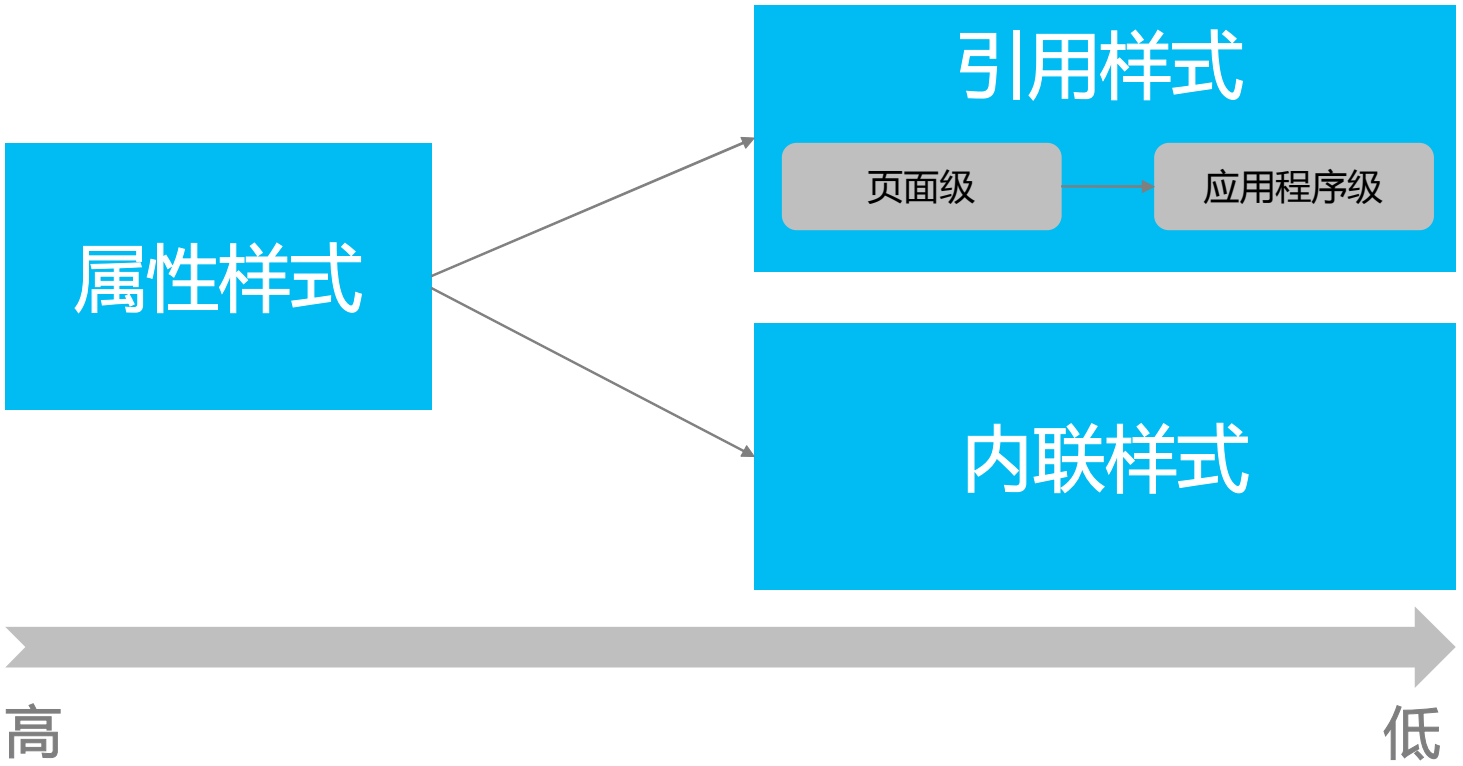
# 应用程序级样式

与页面级样式的定义方式和应用方式相同，差别有两方面：

- 定义位置不同
- 作用范围不同

```
<Application.Resources>
   <Style TargetType="TextBox" x:Key="appStyleBase">
      <Setter Property="Background" Value="LightGreen"/>
   </Style>
   <Style TargetType="TextBox" x:Key="appStyle" BasedOn="{StaticResource
appStyleBase}">
      <Setter Property="BorderThickness" Value="3"/>
      <Setter Property="BorderBrush" Value="Green"/>
   </Style>
   <Style TargetType="TextBox">
      <Setter Property="FontSize" Value="40"/>
   </Style>
</Application.Resources>
```

# 样式优先级



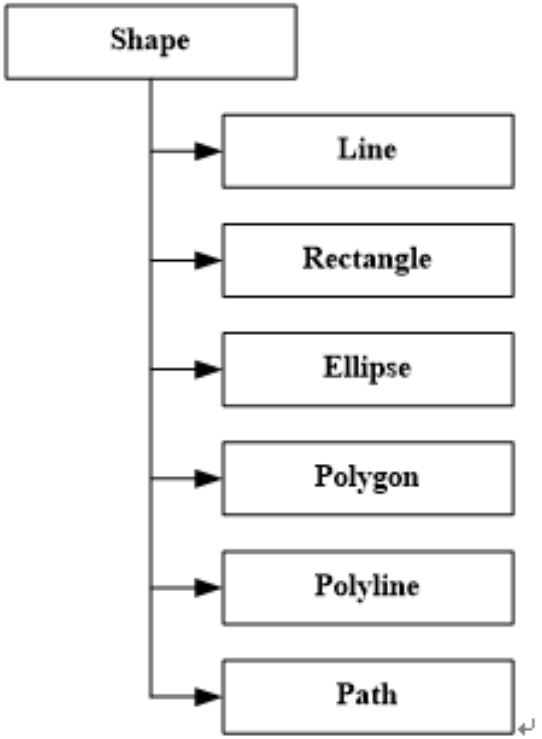
# 样式

# 演示

# 高级绘图

# 形状绘图

- 形状(Shape)是一个2D绘图类
- 位于System.Windows.Shape空间内
- 包括最常用的绘图对象
  - Line 直线
  - Rectangle 矩形
  - Ellipse 椭圆
  - Polygon 多边形
  - Path 路径
- 图形对象共有属性
  - Stroke：说明如何绘制图形的轮廓，即所使用的画刷
  - StrokeThickness：说明图形轮廓的粗细度
  - Fill：说明如何绘制图形的内部
  - 指定图形的坐标位置和顶点的数据属性，以与设备无关的像素来度量



# 直线

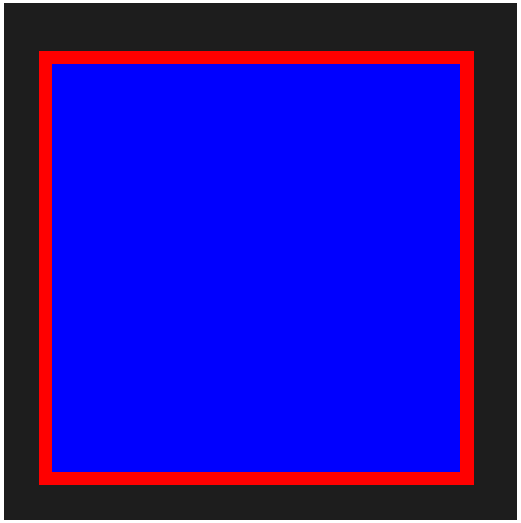
```
<Line X1="200" Y1="200" X2="300" Y2="200"
Stroke="Red" StrokeThickness="5">
</Line>
```

# 修改直线形状

```
<Line X1="200" Y1="200" X2="300" Y2="200"
  Stroke="Red"
  StrokeThickness="30"
  StrokeStartLineCap="Flat"
  StrokeEndLineCap="Triangle">
</Line>
```

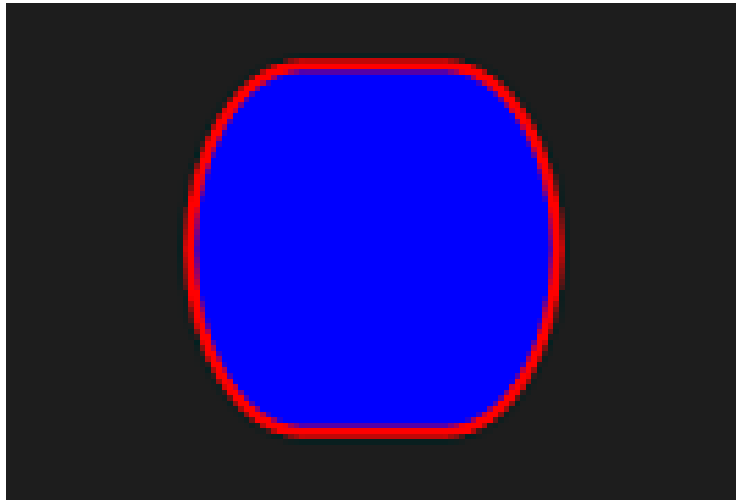
# 矩形

```
<Rectangle Width="100" Height="100" Fill="Blue"
Stroke="Red" StrokeThickness="3"
Margin="0,0,380,552">
</Rectangle>
```



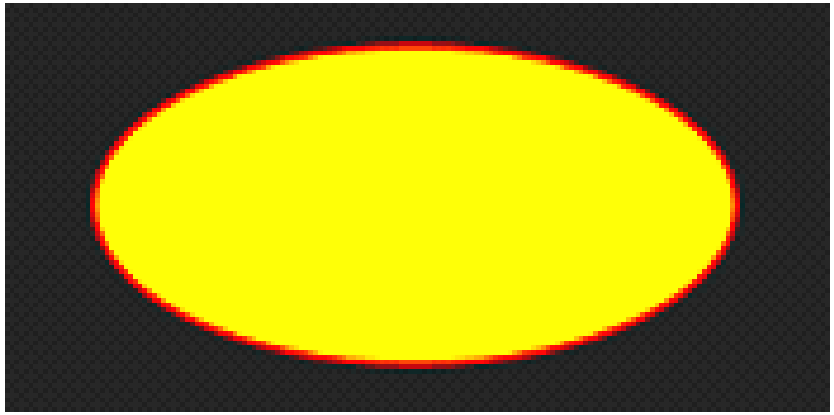
# 绘制圆角矩形

```
<Rectangle Width="100" Height="100" Fill="Blue"
Stroke="Red" StrokeThickness="3" RadiusX="30"
RadiusY="100">
</Rectangle>
```



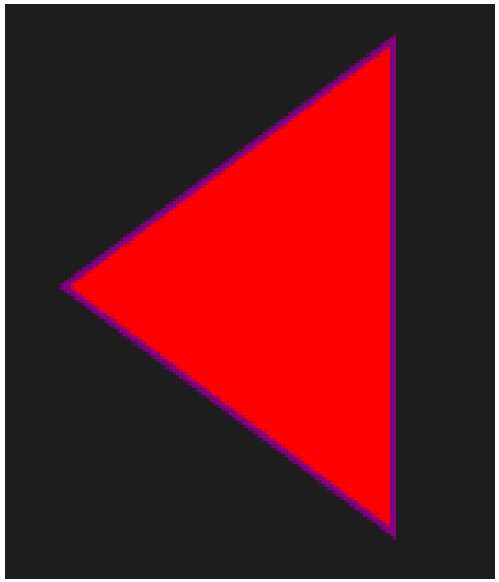
# 椭圆

```
<Ellipse Fill="Yellow" Height="100" Width="200"
StrokeThickness="2" Stroke="Red">
</Ellipse>
```



# 多边形

```
<Polygon Points="300,200 400,125 400,275 300,200"
Stroke="Purple" Fill="Red" StrokeThickness="2">
</Polygon>
```



# 多线型

```
<Polyline Points="120,20 300,20 300,120 200,120"
  Stroke="Red"
  StrokeThickness="5"
  Fill="Orange">
</Polyline>
```

# 路径绘图

- Path类可以绘制曲线和复杂形状
- Data属性使用了mini-language,即路径标记语法

```
<Path Stroke="Red" StrokeThickness="3" Data="M 100,200 C
100,25 400,350 400,175 " />
```

# 路径标记语法

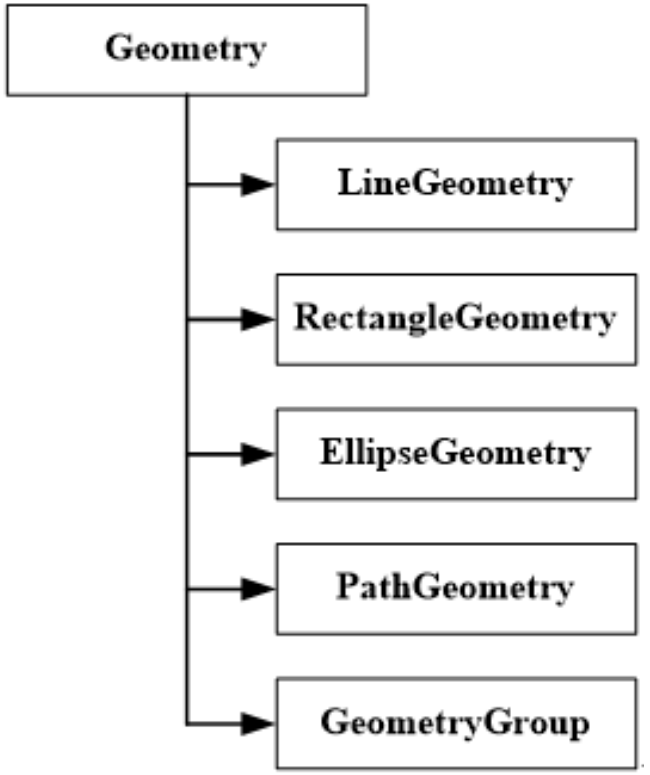
| | |
|---|---|
| 移动命令 | 指定startPoint(绘图的起始点)，用M或m表示，使用M时，表示绝对值，使用m时，表示相对于前一点的便宜量 |
| 绘制命令 | 一个指令集合，用来描述外形轮廓的内容，包含大部分的直线和曲线的绘图指令 |
| 关闭命令 | 作用是结束当前的画图，用来闭合整个Path，并在当前点和图形的起点之间画一条线段，使用字母z来表示 |

# 形状绘图

## 演示

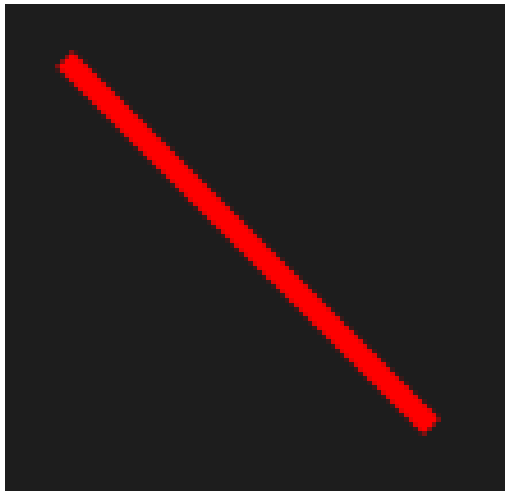
# 几何绘图

- 图形绘图与几何绘图区别
  - 图形对象可以独立存在的，可以独立绘制出具体需要的图形，
  - 几何图形对象没有具体的形体，它需要依赖于某一对象元素而存在，不能直接呈现在画板上
- 几何绘图包括5种对象
  - LineGeometry：确定两点绘制一条直线
  - RectangleGeometry：绘制矩形的几何图形
  - EllipseGeometry：绘制椭圆形的几何图形
  - GeometryGroup：组合几何对象，将多个单一的几何对象组合成一个几何对象
  - PathGeometry：路径几何对象



# LineGeometry

```
<Path Fill="Orange" Stroke="Red" StrokeThickness="5"
Canvas.Top="20" Canvas.Left="100">
  <Path.Data>
     <LineGeometry StartPoint="20,20"
EndPoint="100,100"/>
  </Path.Data>
</Path>
```

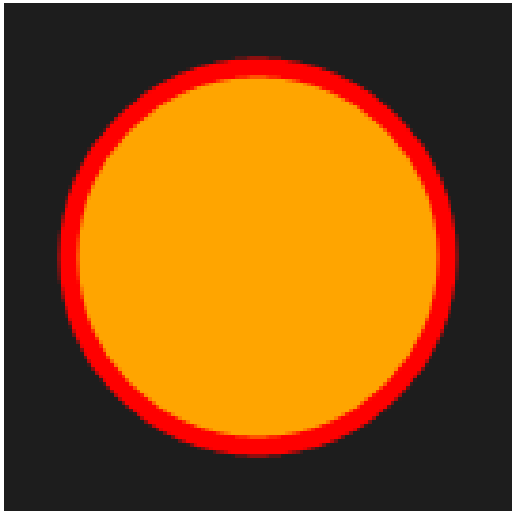


# RectangleGeometry

```
<Path Fill="Orange" Stroke="Red" StrokeThickness="5">
  <Path.Data>
     <RectangleGeometry Rect="100,50,100,50"/>
  </Path.Data>
</Path>
```

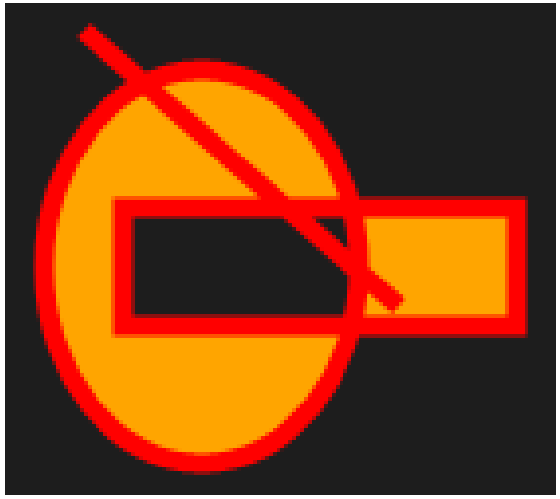
# EllipseGeometry

```
<Path Fill="Orange" Stroke="Red" StrokeThickness="5">
  <Path.Data>
     <EllipseGeometry Center="100,100"
  RadiusX="50"RadiusY="50"/>
  </Path.Data>
</Path>
```



# GeometryGroup

```
<Path Fill="Orange" Stroke="Red" StrokeThickness="5">
   <Path.Data>
      <GeometryGroup FillRule="EvenOdd">
         <LineGeometry StartPoint="20,10" EndPoint="100,80" />
         <EllipseGeometry Center="50,70" RadiusX="40" RadiusY="50" />
         <RectangleGeometry Rect="30,55 100 30" />
      </GeometryGroup>
   </Path.Data>
</Path>
```



# PathGeometry

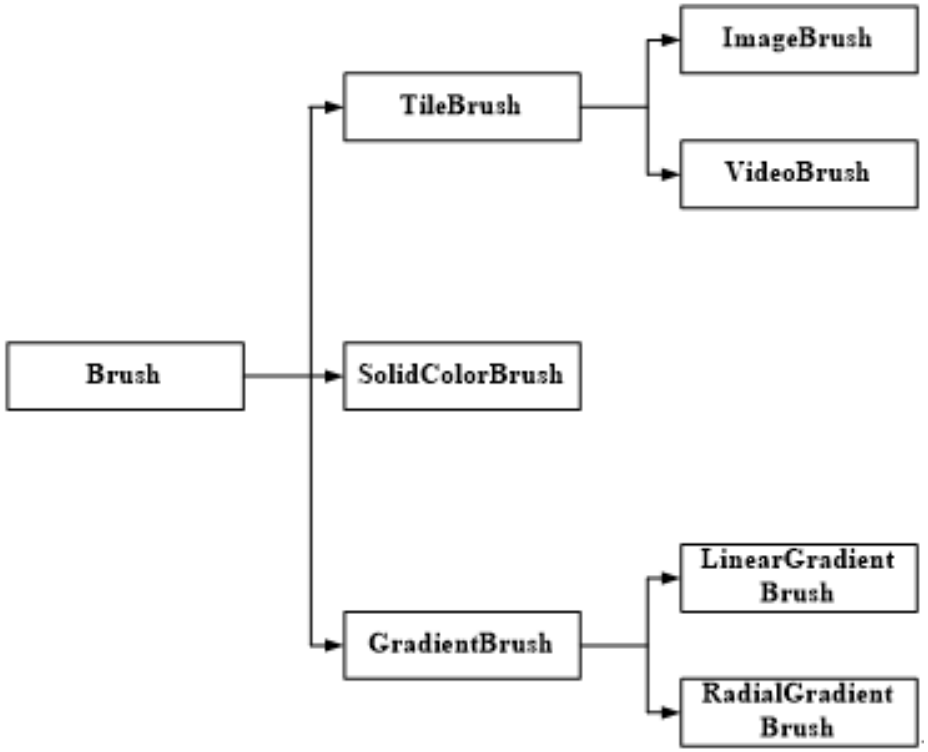
```
<Path Fill="Orange" Stroke="Red" StrokeThickness="5">
  <Path.Data>
     <PathGeometry>
        <PathFigure StartPoint="30,50">
           <LineSegment Point="300,70" />
        </PathFigure>
     </PathGeometry>
  </Path.Data>
</Path>
```

# 几何绘图

## 演示

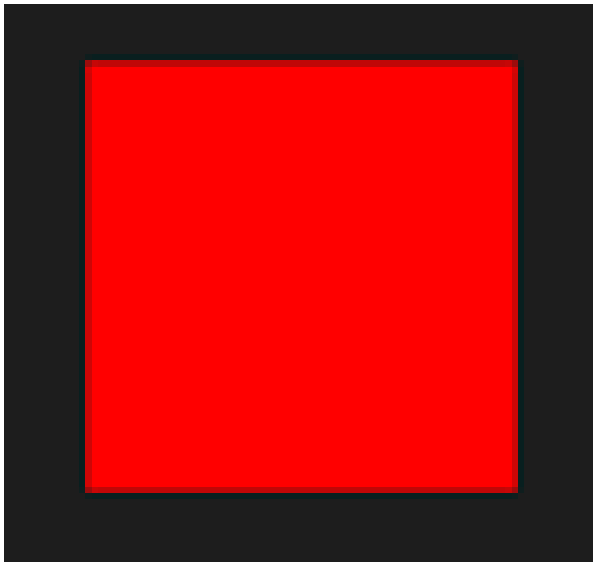
# 画刷

- 所有的UI元素都可以通过画刷的设置而改变它们的风格
- 使用不同的画刷对目标区域进行“绘制”，会有不同的效果
- 画刷种类：
  - SolidColorBrush：单色实心画刷
  - LinearGradientBrush：线性渐变画刷
  - RadialGradientBrush：径向渐变画刷(Windows 8应用中不支持)
  - ImageBrush：图片画刷



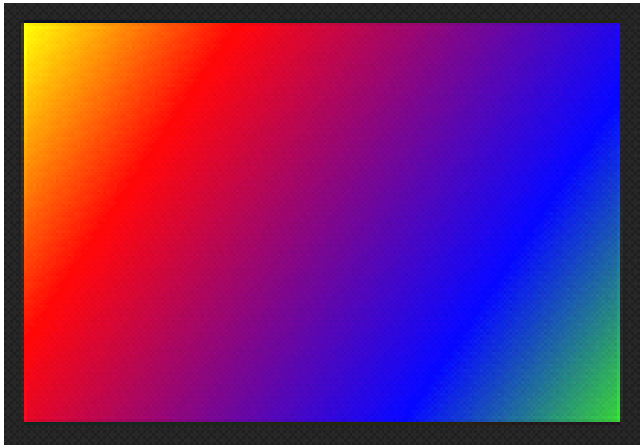
# SolidColorBrush

```
<Rectangle Width="100" Height="100">
  <Rectangle.Fill>
     <SolidColorBrush Color="Red">
     </SolidColorBrush>
  </Rectangle.Fill>
</Rectangle>
```



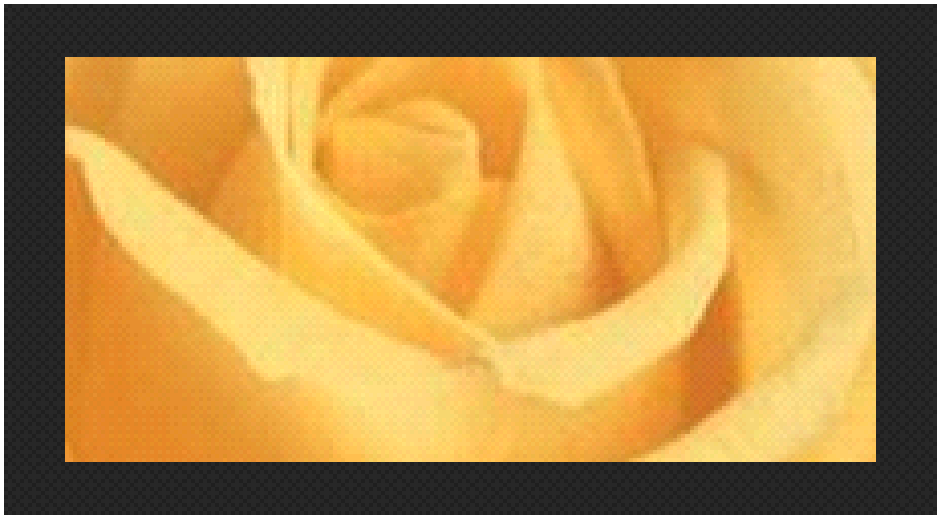
# LinearGradientBrush

```
<Rectangle Width="300" Height="200">
  <Rectangle.Fill>
     <LinearGradientBrush StartPoint="0,0" EndPoint="1,1">
        <GradientStop Color="Yellow" Offset="0.0" />
        <GradientStop Color="Red" Offset="0.25" />
        <GradientStop Color="Blue" Offset="0.75" />
        <GradientStop Color="LimeGreen" Offset="1.0" />
     </LinearGradientBrush>
  </Rectangle.Fill>
</Rectangle>
```



# ImageBrush

```
<Rectangle Width="200" Height="100">
  <Rectangle.Fill>
    <ImageBrush ImageSource="Rose.jpg" Stretch="None">
    </ImageBrush>
  </Rectangle.Fill>
</Rectangle>
```



# 画刷

# 演示

# 图像处理

# 创建图像

- 可以使用Image或者ImageBrush对象来创建一幅图像
- Image元素默认情况下会完整显示图片大小

```
<Grid x:Name="ContentGrid">
            <Image Source="Rose.jpg" />
</Grid>
```

# 图像简单处理

| 拉伸图像 | 裁切图像 | 使用图像输入文字 |
| --- | --- | --- |
| 通过设置Stretch属性实现图像的拉伸 | 通过设置Clip 属性裁切图像 | ImageBrush元素实现使用图像输入文字 |

# 拉伸图像

- Image元素的Stretch属性值类型为枚举型
- 属性值分别为:
  - 原始尺寸(None)
  - 填充拉伸(Fill)
  - 等比拉伸(Uniform)
  - 等比拉伸填充(UniformToFill)

```
<Grid x:Name="ContentGrid" Grid.Row="1">
    <Image Source= "Rose.jpg" Width="500"
    Height="500" Stretch="None"/>
</Grid>
```

None

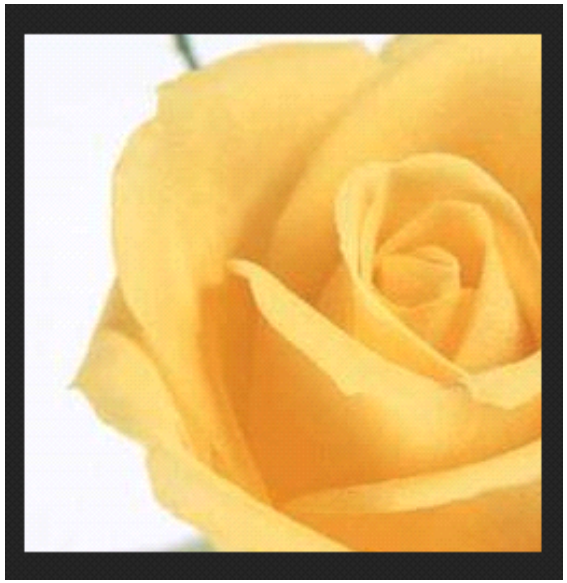
Fill

Uniform

UniformToFill

# 裁切图像

```
<Grid x:Name="ContentGrid" Grid.Row="1">
  <Image Source= "Rose.jpg" Width="500"
Height="500">
      <Image.Clip>
          <RectangleGeometry Rect="50,50 240,240"/>
      </Image.Clip>
  </Image>
</Grid>
```



# 使用图像输入文字

```
<Grid x:Name="ContentGrid" Grid.Row="1">
   <TextBlock FontSize="150" FontStyle="Italic"
FontWeight="Bold">
     Lenna
     <TextBlock.Foreground>
        <ImageBrush ImageSource="Rose.jpg" />
     </TextBlock.Foreground>
   </TextBlock>
</Grid>
```

# 透明特效

```
<Grid x:Name="ContentGrid" Grid.Row="1">
  <Image Source="Rose.jpg" Width="500"
Height="500"   Op
</Grid>
```

# 图像处理

# 演示

# 动画

# 故事板

- Storyboard(故事板)是动画的基本单元
- Storyboard控制动画的播放，暂停，停止等操作
- 需要指定TargetName和TargetProperty属性
- 动画类型声明过之后，需要使用EventTrigger(事件触发器)触发

```
<Storyboard x:Name="storyboard1">
  <DoubleAnimation Storyboard.TargetName="ellipse1"
Storyboard.TargetProperty="Opacity“ From="1.0" To="0.0"
Duration="0:0:5"/>
</Storyboard>
```

# 事件触发器

- 通过事件触发器播放BeginStoryboard故事板的动画效果

```
<Canvas Background="White">
    <Canvas.Triggers>
        <EventTrigger RoutedEvent="Canvas.Loaded">
            <EventTrigger.Actions>
                <BeginStoryboard>
                    <Storyboard x:Name="storyboard1">
                        <DoubleAnimation Storyboard.TargetName="ellipse1"
                                         Storyboard.TargetProperty="Opacity"
                                         From="1.0" To="0.0"
                                         Duration="0:0:5"/>
                    </Storyboard>
                </BeginStoryboard>
            </EventTrigger.Actions>
        </EventTrigger>
    </Canvas.Triggers>
    <Ellipse x:Name="ellipse1" Fill="GreenYellow" Width="150" Height="200"/>
</Canvas>
```

# 托管代码触发动画播放

XAML:

```
<Page.Resources>
  <Storyboard x:Name="storyboard1">
     <DoubleAnimation
Storyboard.TargetName="ellipse1“ Storyboard.TargetProperty="Opacity"
     From="1.0" To="0.0"
     Duration="0:0:5"/>
  </Storyboard>
</Page.Resources>
<Canvas Background="White">
  <Ellipse x:Name="ellipse1" Fill="GreenYellow" Width="150"
Height="200"/>
</Canvas>
```

C#:

```
public MainPage()
{

   this.InitializeComponent();

   //使用Begin方法播放动画
   storyboard1.Begin();
}
```

# 线性插值动画

## DoubleAnimation

- 属于Double类型的属性都可以使用它产生线性插值动画效果

## ColorAnimation

- 作用于属性为Color类型对象的线性插值动画，用于改变对象的填充颜色

# 线性插值动画对象属性

| | |
|---|---|
| **From** | 动画从 From 属性指定的值继续到正在进行动画处理的属性的基值或前一动画的输出值，具体取决于前一动画的配置方式。 |
| **To** | 动画从进行动画处理的属性的基值或前一动画的输出值继续到 To 属性指定的值。 |
| **By** | 动画从正在进行动画处理的属性的基值或前一动画的输出值继续到该值与 By 属性指定的值之和。 |
| **Duration** | 动画执行一次持续的时间长度，Duration的格式为时：分：秒 |

# 动画播放控制属性

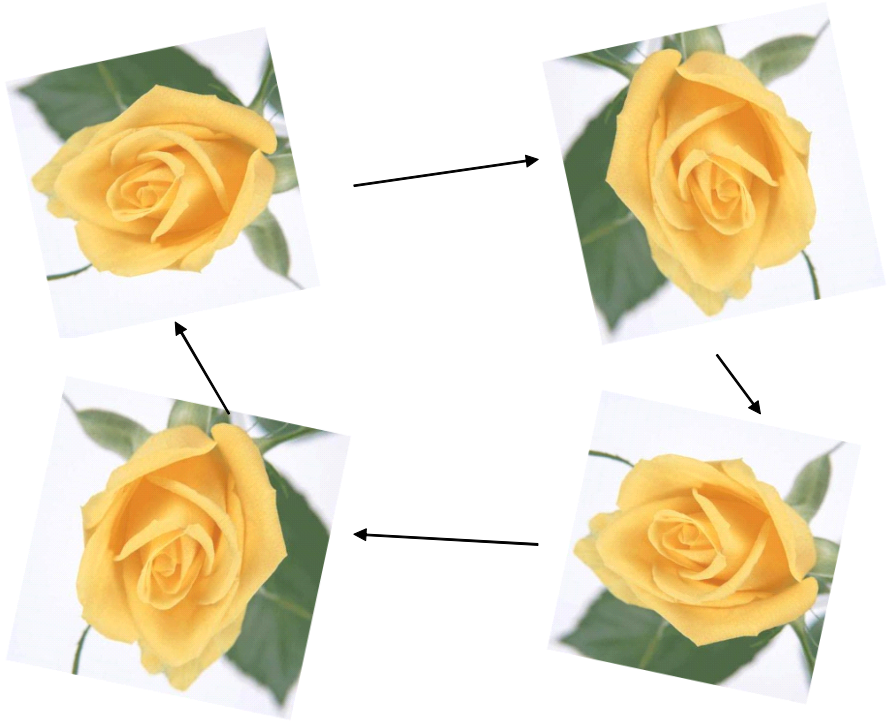
| | |
|---|---|
| **BeginTime** | 动画开始时间。默认的单位是天，也可以指定为时：分：秒 |
| **RepeatBehavior** | 用来声明动画重复次数，支持3种类型值：重复次数（格式--次数+X）;一个时间段（格式—时：分：秒）；特殊值Forever（代表无限循环） |
| **AutoReverse** | 指定动画结束后是否向后继续播放，默认只为false，若设置为true，动画结束时会回到起始位置 |
| **SpeedRatio** | 用来增加或减少动画的速度，默认值为1，若增加它，动画产生加速播放的效果 |
| **FillBehavior** | 决定什么时候发生动画，什么时候结束。默认值为HoldEnd，表示动画结束后保持当前值不变，也可以设置为Stop，表示动画结束时属性再次回到起始值 |

# DoubleAnimation动画I

```
<Canvas Background="White">
   <Canvas.Triggers>
      <EventTrigger RoutedEvent="Canvas.Loaded">
         <EventTrigger.Actions>
            <BeginStoryboard>
                <Storyboard x:Name="storyboard1">
                  <!--动画作用于RotateTransform.Angle属性值-->
                  <DoubleAnimation Storyboard.TargetName="rec"
Storyboard.TargetProperty="(UIElement.RenderTransform).(TransformGroup.Children)[2].(Rot
ateTransform.Angle)" From="0" To="360" Duration="0:0:2"
RepeatBehavior="Forever"/>
                </Storyboard>
            </BeginStoryboard>
         </EventTrigger.Actions>
     </EventTrigger>
  </Canvas.Triggers>
```

# DoubleAnimation动画II

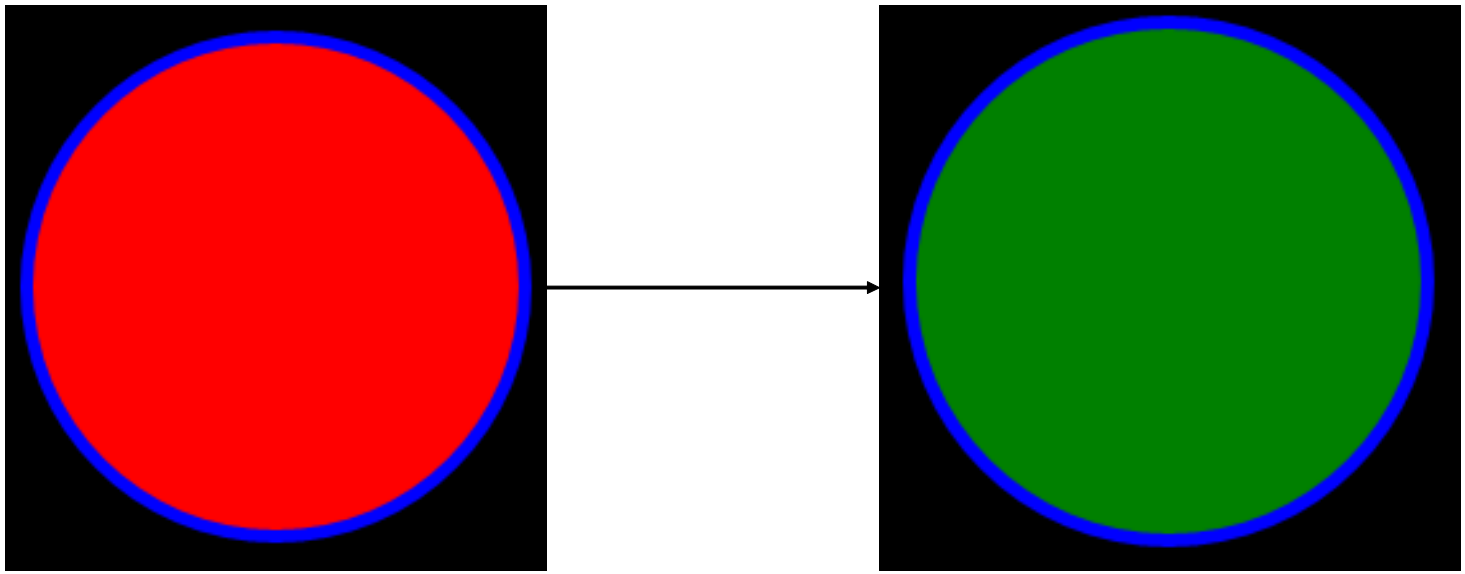
```
<!--创建用来选择的矩形-->
   <Rectangle x:Name="rec" Height="300" Width="300" Canvas.Left="100"
Canvas.Top="100" RenderTransformOrigin="0.5,0.5">
      <Rectangle.Fill>
         <ImageBrush ImageSource="Rose.jpg"/>
      </Rectangle.Fill>
      <Rectangle.RenderTransform>
         <TransformGroup>
            <ScaleTransform/>
            <SkewTransform/>
            <RotateTransform/>
            <TranslateTransform/>
         </TransformGroup>
      </Rectangle.RenderTransform>
    </Rectangle>
</Canvas>
```



# ColorAnimation动画

```
<Page.Resources>
   <Storyboard x:Name="storyboard1">
      <ColorAnimation Storyboard.TargetName="ellipse1"
      Storyboard.TargetProperty="(Ellipse.Fill).(SolidColorBrush.Color)"
      From="Red" To="Green" Duration="0:0:2">
      </ColorAnimation>
   </Storyboard>
</Page.Resources>
<Canvas Background="Black">
   <Ellipse x:Name="ellipse1" Width="200" Height="200" Fill="Red" Canvas.Left="100" Canvas.Top="80" Stroke="Blue"
StrokeThickness="5" />
</Canvas>
```

# ColorAnimation动画



# 线性插值动画

# 演示

# 关键帧动画

关键帧动画根据目标属性值之间的差异产生各种动画效果

一个关键帧动画可以在任意多个的目标属性值之间进行渐变

关键帧动画可以产生更多，更复杂的动画效果

# 关键帧动画元素

## DoubleAnimationUsingKeyFrames

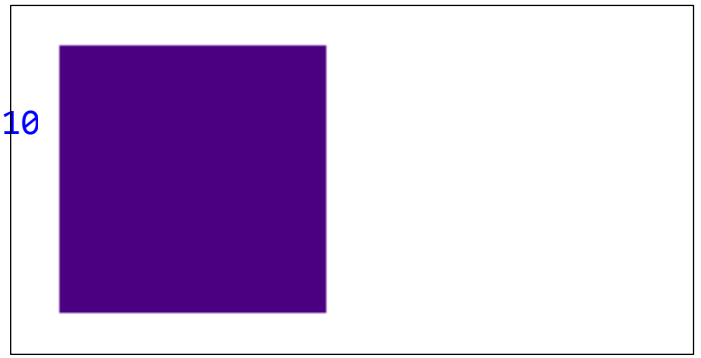
- Double关键帧动画

## ColorAnimationUsingKeyFrames

- Color关键帧动画

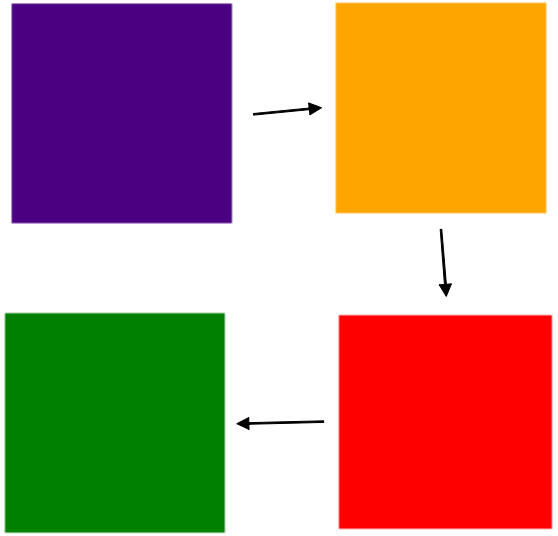
# DoubleAnimationUsingKeyFrames动画

```
<DoubleAnimationUsingKeyFrames
 Storyboard.TargetName="Scenario2KeyFrameRectangle"
 Storyboard.TargetProperty="(Canvas.Left)"
 Duration="0:0:3">
  <DiscreteDoubleKeyFrame KeyTime="0:0:0.5" Value="50" />
  <DiscreteDoubleKeyFrame KeyTime="0:0:1" Value="100" />
  <DiscreteDoubleKeyFrame KeyTime="0:0:2" Value="200" />
  <EasingDoubleKeyFrame KeyTime="0:0:3" Value="300" />
</DoubleAnimationUsingKeyFrames>

<Canvas Width="400" Height="100" Margin="0,20,0,0">
  <Rectangle Name="Scenario2KeyFrameRectangle" Width="10
Height="100" Fill="Indigo" />
</Canvas>
```



# ColorAnimationUsingKeyFrames动画

```
 <ColorAnimationUsingKeyFrames
  EnableDependentAnimation="true"
Storyboard.TargetName="Scenario2KeyFrameRectangle"
Storyboard.TargetProperty="(Rectangle.Fill).(SolidCol
orBrush.Color)“ Duration="0:0:3">
  <DiscreteColorKeyFrame KeyTime="0:0:0.5"
Value="Red" />
  <DiscreteColorKeyFrame KeyTime="0:0:1"
Value="Orange" />
  <DiscreteColorKeyFrame KeyTime="0:0:2" Value="Blue"
/>
  <EasingColorKeyFrame KeyTime="0:0:3" Value="Green"
/>
</ColorAnimationUsingKeyFrames>

<Canvas Width="400" Height="100" Margin="0,20,0,0">
  <Rectangle Name="Scenario2KeyFrameRectangle"
Width="100" Height="100" Fill="Indigo" />
</Canvas>
```



# 关键帧动画

# 演示

# 总结

资源的使用
三种样式：属性、内联、引用
绘图：图形绘图、几何绘图
图像：拉伸、裁剪、特效
动画：线性插值动画、关键帧动画

# 资源

*ResourceDictionary and StaticResource references*

*Styling and Templating*

*Drawing shapes*

*Image and ImageBrush*

*Animation Overview*

*Windows 8 Sample Code*

*《Programming Windows Sixth Edition》*

*http://shop.oreilly.com/product/0790145369079.do*

*《Programming Windows Phone 7》*

*http://shop.oreilly.com/product/0790145316707.do?intcmp=ba-ms-books-int-search-windows-phone-ct*

# Q&A

© 2011 Microsoft Corporation。保留所有权利。Microsoft、Windows、Windows Vista 及其他产品名称是或者可能是在美国和 /或其他国家/地区的注册商标和/或商标
此处包含的信息仅供参考，并代表 Microsoft Corporation 截至本演示文稿发布之日的最新观点。由于 Microsoft 必须响应不断变化的市场条件，所以不应将本文视为 Microsoft 一方的承诺，Microsoft Corporation 也无法保证所提供信息在本文发布之后的准确性。MICROSOFT 对本
演示文稿中包含的信息不做任何明示、暗示或法定的担保。

Microsoft®

© 2011 Microsoft Corporation。保留所有权利。Microsoft、Windows、Windows Vista 及其他产品名称是或者可能是在美国和 /或其他国家/地区的注册商标和/或商标
此处包含的信息仅供参考，并代表 Microsoft Corporation 截至本演示文稿发布之日的最新观点。由于 Microsoft 必须响应不断变化的市场条件，所以不应将本文视为 Microsoft 一方的承诺，Microsoft Corporation 也无法保证所提供信息在本文发布之后的准确性。MICROSOFT 对本
演示文稿中包含的信息不做任何明示、暗示或法定的担保。