Анимация

Основы анимаций

Наличие встроенных возможностей анимации представляют одну из ключевых особенностей платформы WPF. Анимации в WPF - это действиетельно мощное средство, которое при этом очень легко использовать. Но перед тем, как перейти к созданию анимаций, сразу надо сказать об ограничениях:

- Одна анимация выполняется только для одного свойства зависимостей
- Для анимации свойства необходим класс анимации, который бы поддерживал тип этого свойства. Например, для изменения таких свойств, как длина, ширина, которые представляют тип double, предназначен класс **DoubleAnimation**. Для изменения цвета фона или шрифта - **ColorAnimation**, для изменения свойства Margin - **ThiknessAnimation**.

За анимацию в WPF отвечает пространство имен System.Windows.Media.Animation. Оно содержит довольно большой набор классов, позволяющих анимировать различные свойства. Но в реальности все классы анимаций можно разделить условно на три группы:

- Классы, которые производят линейную интерполяцию значений, благодаря чему при анимации свойство плавно изменяет свое значение. Как правило, такие классы называются по шаблону TypeAnimation, где Type - тип данных, которое представляет анимируемое свойства, например, DoubleAnimation:

- ByteAnimation
- ColorAnimation
- DecimalAnimation
- DoubleAnimation
- Int16Animation
- Int32Animation
- Int64Animation
- PointAnimation
- Point3DAnimation
- QuarternionAnimation
- RectAnimation
- Rotation3DAnimation
- SingleAnimation
- SizeAnimation

- ThicknessAnimation
- VectorAnimation
- Vector3DAnimation

- Классы, которые производят анимацию по ключевым кадрам или фреймам (покадровая анимация). Такие классы, как правило, называются по шаблону TypeAnimationUsingKeyFrames, например, DoubleAnimationUsingKeyFrames

- BooleanAnimationUsingKeyFrames
- ByteAnimationUsingKeyFrames
- CharAnimationUsingKeyFrames
- ColorAnimationUsingKeyFrames
- DecimalAnimationUsingKeyFrames
- DoubleAnimationUsingKeyFrames
- Int16AnimationUsingKeyFrames
- Int32AnimationUsingKeyFrames
- Int64AnimationUsingKeyFrames
- MatrixAnimationUsingKeyFrames
- ObjectAnimationUsingKeyFrames
- PointAnimationUsingKeyFrames
- Point3DAnimationUsingKeyFrames
- QuarternionAnimationUsingKeyFrames
- RectAnimationUsingKeyFrames
- Rotation3DAnimationUsingKeyFrames
- SingleAnimationUsingKeyFrames
- SizeAnimationUsingKeyFrames
- StringAnimationUsingKeyFrames
- ThicknessAnimationUsingKeyFrames
- VectorAnimationUsingKeyFrames
- Vector3DAnimationUsingKeyFrames

- Классы, которые используют для анимации объект PathGeometry. Такие классы, как правило, называются по шаблону TypeAnimationUsingPath, например, DoubleAnimationUsingPath

- DoubleAnimationUsingPath
- MatrixAnimationUsingPath
- PointAnimationUsingPath

Анимацию можно создать и использовать как декларативно в коде XAML, так и программно в коде C#.

Программная анимация

Пусть в XAML определена кнопка:

```
<Button x:Name="helloButton" Width="70" Height="30" Content="Hello" />
```

Проанимируем ее свойство Width:

```
using System;
using System.Windows;
using System.Windows.Controls;

using System.Windows.Media.Animation;

namespace AnimationApp
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();

            DoubleAnimation buttonAnimation = new DoubleAnimation();
            buttonAnimation.From = helloButton.ActualWidth;
            buttonAnimation.To = 150;
            buttonAnimation.Duration = TimeSpan.FromSeconds(3);
            helloButton.BeginAnimation(Button.WidthProperty, buttonAnimation);
        }
    }
}
```

В данном случае мы сначала задаем тип анимации и создаем ее. Поскольку изменяется свойство Width, это будет **DoubleAnimation**.

У любого класса линейной анимации есть набор свойств, с помощью которых мы можем управлять анимацией:

- From: начальное значение, с которого будет начинается анимация
- To: конечное значение
- Duration: время анимации в виде объекта TimeSpan

- By: значение, которое указывает, насколько должно увеличиться анимируемое свойство. Свойство By используется вместо свойства To
- RepeatBehavior: позволяет установить, как анимация будет повторяться
- AccelerationRatio: задает ускорение анимации
- DecelerationRatio: устанавливает замедление анимации
- SpeedRatio: устанавливает скорость анимации. По умолчанию значение 1.0
- AutoReverse: при значении true анимация выполняется в противоположную сторону
- FillBehavior: определеяет поведение после окночания анимации. Если оно имеет значение Stop, то после окончания анимации объект возвращает прежние значения: buttonAnimation.FillBehavior = FillBehavior.Stop. Если же оно имеет значение HoldEnd, то анимация присваивает анимируемому свойству новое значение.

Чтобы запустить анимацию, у элемента вызывается метод BeginAnimation(). В этот метод передается свойство зависимости, которое надо анимировать, и сам объект анимации.

В данном случае произойдет изменение ширины кнопки от текущего значения до 150 пикселей. И данное изменение будет длиться 3 секунды.

Если мы хотим повторения анимации, можно использовать свойство RepeatBehavior:

```
DoubleAnimation buttonAnimation = new DoubleAnimation();
buttonAnimation.From = helloButton.ActualWidth;
buttonAnimation.To = 150;
buttonAnimation.Duration = TimeSpan.FromSeconds(3);
buttonAnimation.RepeatBehavior = new RepeatBehavior(2);
helloButton.BeginAnimation(Button.WidthProperty, buttonAnimation);
```

В данном случае анимация будет повторяться два раза. Также мы можем задать время повторения:

```
buttonAnimation.Duration = TimeSpan.FromSeconds(3);
buttonAnimation.RepeatBehavior = new RepeatBehavior(TimeSpan.FromSeconds(7));
```

Здесь время повторения - 7 секунд. Анимация длится 3 секунды, а это значит, что будет 7 / 3 повторений: два полноценных повторения и в последнем случае ширина увелится только до трети требуемой ширины.

Чтобы задать плавное изменение свойства в обратную сторону, применим свойство AutoReverse:

```
buttonAnimation.AutoReverse = true;
buttonAnimation.RepeatBehavior = new RepeatBehavior(5);
```

Событие Completed

При завершении анимации генерируется событие Completed, которое мы можем обработать:

```
public MainWindow()
{
    InitializeComponent();

    DoubleAnimation buttonAnimation = new DoubleAnimation();
    buttonAnimation.From = helloButton.ActualWidth;
    buttonAnimation.To = 150;
    buttonAnimation.Duration = TimeSpan.FromSeconds(5);
    buttonAnimation.Completed += ButtonAnimation_Completed;
    helloButton.BeginAnimation(Button.WidthProperty, buttonAnimation);
}

private void ButtonAnimation_Completed(object sender, EventArgs e)
{
    MessageBox.Show("Анимация завершена");
}
```

Анимация нескольких свойств

Одна анимация может работать только с одним свойством объекта. Однако если нам надо анимировать сразу несколько свойств, то мы можем задать для каждоо свой объект анимации:

```
// анимация для ширины
DoubleAnimation widthAnimation = new DoubleAnimation();
widthAnimation.From = helloButton.ActualWidth;
widthAnimation.To = 150;
widthAnimation.Duration = TimeSpan.FromSeconds(5);

// анимация для высоты
DoubleAnimation heightAnimation = new DoubleAnimation();
heightAnimation.From = helloButton.ActualHeight;
heightAnimation.To = 60;
heightAnimation.Duration = TimeSpan.FromSeconds(5);

helloButton.BeginAnimation(Button.WidthProperty, widthAnimation);
helloButton.BeginAnimation(Button.HeightProperty, heightAnimation);
```

Анимация в XAML

Для определения анимации в XAML применяется объект **EventTrigger** или триггер событий. Этот объект имеет свойство **Actions**, которое определяет ряд действий, возникающих в результате генерации события. Само возникающее действие описывается через элемент **BeginStoryboard**, который и запускает анимацию.

Непосредственно для определения анимации используется объект **Storyboard**. Он объявляет объект анимации со всеми ее свойствами и параметрами. Например, проанимируем ширину кнопки:

```
<Window x:Class="AnimationApp.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:AnimationApp"
        mc:Ignorable="d"
        Title="MainWindow" Height="250" Width="300">
    <Window.Triggers>
        <EventTrigger RoutedEvent="Loaded">
            <EventTrigger.Actions>
                <BeginStoryboard>
                    <Storyboard TargetProperty="Width" TargetName="helloButton">
                        <DoubleAnimation From="70" To="150"
    AutoReverse="True"
    RepeatBehavior="0:0:10"
    Duration="0:0:3"
    Completed="ButtonAnimation_Completed" />
                    </Storyboard>
                </BeginStoryboard>
            </EventTrigger.Actions>
        </EventTrigger>
    </Window.Triggers>
    <Grid>
        <Button x:Name="helloButton" Width="70" Height="30" Content="Hello" />
    </Grid>
</Window>
```

У объекта EventTrigger с помощью атрибута RoutedEvent определяется событие, которое будет запускать анимацию. В данном случае это событие Loaded - загрузка окна.

Ряд настроек анимации устанавливает элемент Storyboard: TargetName задает анимируемый элемент, а TargetProperty определяет свойство элемента, которое будет анимироваться.

В принципе эти настройки также можно было бы вынести в объект анимации в виде прикрепляемых свойств:

```
<Storyboard>
    <DoubleAnimation Storyboard.TargetProperty="Width" Storyboard.TargetName="helloButton"
                     From="70" To="150"  ...>
```

И далее в самом объекте Storyboard определяется объект анимации DoubleAnimation с рядом настроек. Все настройки в прицнипе тут те же, что использовались для анимации в прошлой теме. Правда, при определении значений свойств здесь есть некоторые отличия.

Так, свойство **RepeatBehavior** инициализируется временем - "0:0:10", которое будет повторяться анимация. Чтобы указать число повторов, структуру **RepeatBehavior** надо инициализировать так: **RepeatBehavior="2x"**, где 2 - количество повторов, а x - просто префикс, указывающий, что речь идет о количестве итераций, без него бы число интерпретировалось как количество дней. Третий способ задания этого свойства - **RepeatBehavior="Forever"** - в этом случае анимация будет продолжаться все время работы приложения.

Например, ниже установлено **RepeatBehavior="Forever"**, а свойство **DecelerationRatio** замедляет анимацию, что создает эффект подскока шара в верх, где он, достигая максимальной точки, теряет скорость, а при падении вновь ее увеличивает:

```
<Window x:Class="AnimationApp.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:AnimationApp"
        mc:Ignorable="d"
        Title="MainWindow" Height="250" Width="300">
    <Window.Triggers>
        <EventTrigger RoutedEvent="Button.Click">
            <EventTrigger.Actions>
                <BeginStoryboard>
                    <Storyboard Timeline.DesiredFrameRate="60">
                        <DoubleAnimation Storyboard.TargetName="ball" Storyboard.TargetProperty=
                                 From="0" To="160" AutoReverse="True" Duration="0:0:2.5" RepeatB
                                 DecelerationRatio="1"/>
                    </Storyboard>
                </BeginStoryboard>
            </EventTrigger.Actions>
        </EventTrigger>
    </Window.Triggers>
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition />
            <RowDefinition Height="Auto" />
        </Grid.RowDefinitions>
        <Canvas Background="LightPink">
            <Ellipse Name="ball" Fill="Red" Stroke="Black"  Width="15" Height="15"
                        Canvas.Left="130" Canvas.Bottom="0" />
        </Canvas>
        <Button Width="70" Height="25" Content="Кнопка" Grid.Row="1" Margin="10" />
    </Grid>
</Window>
```

В данном случае так как в EventTrigger в качестве события определено Button.Click, то анимация будет запускаться по нажатию на кнопку.

По умолчанию анимация вызывается 60 раз в секунду, однако с помощью прикрепленного свойства **Timeline.DesiredFrameRate** можно задать частоту кадров в секунду, как в предыдущем примере.

Анимация свойств вложенных объектов

Анимация может применяться и к свойствам вложенных объектов, которые являются свойствами, например:

```
<Ellipse Name="ball" Stroke="Black"  Width="20" Height="20" Canvas.Left="130" Canvas.Bottom="0">
    <Ellipse.Fill>
        <RadialGradientBrush RadiusX="1" RadiusY="1" GradientOrigin="0.3, 0.3">
            <GradientStop Color="White" Offset="0" />
            <GradientStop Color="Blue" Offset="1" />
        </RadialGradientBrush>
    </Ellipse.Fill>
    <Ellipse.Triggers>
        <EventTrigger RoutedEvent="Window.Loaded">
            <BeginStoryboard>
                <Storyboard>
                    <ColorAnimation Storyboard.TargetProperty="Fill.GradientStops[1].Color"
                            To="Yellow" Duration="0:0:8" AutoReverse="True"
                            RepeatBehavior="Forever" />
                </Storyboard>
            </BeginStoryboard>
        </EventTrigger>
    </Ellipse.Triggers>
</Ellipse>
```

Здесь свойство **Fill** элемента Ellipse инициализируется кистью RadialGradientBrush, которая имеет коллекцию **GradientStops**. Анимация же применяется ко второму объекту коллекции и его свойству **Color**.

Комплексные анимации

С помощью объекта Storyboard можно создавать и более комплексные анимации. Например, сделаем кнопку, которая одновременно меняет ширину, длину и цвет:

```
<Button x:Name="helloButton" Foreground="White" Width="70" Height="25" Content="Кнопка">
    <Button.Background>
        <SolidColorBrush x:Name="buttonColor" Color="Black" />
    </Button.Background>
    <Button.Triggers>
        <EventTrigger RoutedEvent="Loaded">
            <EventTrigger.Actions>
                <BeginStoryboard>
                    <Storyboard>
                        <DoubleAnimation Storyboard.TargetProperty="Width" Storyboard.TargetName
                              From="80" To="150"  AutoReverse="True" RepeatBehavior="0:0:10" Dur
                        <DoubleAnimation Storyboard.TargetProperty="Height" Storyboard.TargetNam
                              From="30" To="100" AutoReverse="True" RepeatBehavior="0:0:10" Dura
                        <ColorAnimation Storyboard.TargetName="buttonColor" Storyboard.TargetPro
                              From="{Binding ElementName=buttonColor, Path=Color}" To="Red"
                              AutoReverse="True" RepeatBehavior="0:0:10" Duration="0:0:2" />
                    </Storyboard>
                </BeginStoryboard>
            </EventTrigger.Actions>
        </EventTrigger>
    </Button.Triggers>
</Button>
```

Определение анимации в стиле

При этом необязательно знать имя элемента для анимации, можно прикрепить анимацию ко всем элементам одного типа и установить ее через стиль:

```
<Window x:Class="AnimationApp.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:AnimationApp"
        mc:Ignorable="d"
        Title="MainWindow" Height="250" Width="300">
    <Window.Resources>
        <Style TargetType="Button">
            <Style.Triggers>
                <EventTrigger RoutedEvent="Button.Click">
                    <EventTrigger.Actions>
                        <BeginStoryboard>
                            <Storyboard>
                                <ColorAnimation Storyboard.TargetProperty="Background.Color"
   To="Red" AutoReverse="True" Duration="0:0:2" />
                            </Storyboard>
                        </BeginStoryboard>
                    </EventTrigger.Actions>
                </EventTrigger>
            </Style.Triggers>
        </Style>
    </Window.Resources>
    <StackPanel>
        <Button Width="70" Height="25" Content="Кнопка 1" Margin="10" />
        <Button Width="70" Height="25" Content="Кнопка 2" Margin="10" />
    </StackPanel>
</Window>
```

Так как у стиля задан атриубут TargetType="Button", то анимация внутри триггера будет применяться ко всем кнопкам. В качестве анимируемого свойства указывается "Background.Color". То есть у класса Button есть свойство Background, представленное объектом SolidColorBrush, а у этого объекта есть свойство Color. Напрямую анимировать свойство Background мы не можем, так как ColorAnimation требует типа Color.

Так как стиль применяется глобально к кнопкам, то в настройках анимации название анимируемого элемента не указывается. Также не указывается свойство From, а это значит, что анимаця будет идти от текущего цвета кнопки.

Теперь после нажатия натажатая кнопка будет ненадолго окрашиваться в красный, а затем возвращать исходный цвет.

Анимации по ключевым кадрам

Все вышеприведенные типы анимаций, такие как ColorAnimation или DoubleAnimation, основывались на плавном смене значения. Есть еще второй род анимаций - анимации по ключевым кадрам, которые анимирует те же свойства, что и обычные анимации. Например, **ColorAnimationUsingKeyFrames** анимирует цвет, а **DoubleAnimationUsingKeyFrames** - свойства, представляющие тип double.

Анимации по ключевым кадрам имеет практически все те же свойства, что и обычные анимации. Основное отличие - покадровые анимации имеют свойство **KeyFrames**, которое позволяет установить коллекцию кадров. Каждый кадр (KeyFrame) определяет значение для анимируемого свойства с помощью свойства **Value** и продолжительность кадра с помощью свойства **KeyTime**

```
<Window x:Class="AnimationApp.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:AnimationApp"
        mc:Ignorable="d"
        Title="MainWindow" Height="250" Width="300">
    <Canvas Background="Black">
        <Ellipse Width="20" Height="20" Canvas.Bottom="0" Canvas.Left="120">
            <Ellipse.Fill>
                <RadialGradientBrush RadiusX="1" RadiusY="1" GradientOrigin="0.3, 0.3">
                    <GradientStop Color="White" Offset="0" />
                    <GradientStop Color="Black" Offset="1" />
                </RadialGradientBrush>
            </Ellipse.Fill>
            <Ellipse.Triggers>
                <EventTrigger RoutedEvent="Window.Loaded">
                    <BeginStoryboard>
                        <Storyboard>
                            <DoubleAnimationUsingKeyFrames
                                Storyboard.TargetProperty="(Canvas.Bottom)"
                                Duration="0:0:3" AutoReverse="True"
                                RepeatBehavior="Forever" DecelerationRatio="1" >
                                <LinearDoubleKeyFrame KeyTime="0%" Value="0" />
                                <LinearDoubleKeyFrame KeyTime="33%" Value="60" />
                                <LinearDoubleKeyFrame KeyTime="66%" Value="120" />
                                <LinearDoubleKeyFrame KeyTime="99%" Value="180" />
                            </DoubleAnimationUsingKeyFrames>

                            <ColorAnimationUsingKeyFrames
                                Storyboard.TargetProperty="Fill.GradientStops[1].Color"
                                Duration="0:0:3" RepeatBehavior="Forever" AutoReverse="True">

                                <ColorAnimationUsingKeyFrames.KeyFrames>
                                    <LinearColorKeyFrame KeyTime="0:0:1" Value="Green" />
                                    <LinearColorKeyFrame KeyTime="0:0:2" Value="Red" />
                                </ColorAnimationUsingKeyFrames.KeyFrames>
                            </ColorAnimationUsingKeyFrames>
                        </Storyboard>
                    </BeginStoryboard>
                </EventTrigger>
            </Ellipse.Triggers>
        </Ellipse>
    </Canvas>
</Window>
```

Тип кадра анимации определяется следующим образом: тип анимации задает тип изменяемого значения, например, ColorAnimationUsingKeyFrames изменяемое значение типа Color. Тип кадра

состоит из трех частей: вид кадра, например, линейный (Linear), тип изменяемого значения, и ключевого слова KeyFrame. То есть выше тип кадра был LinearColorKeyFrame - линейный кадр, изменяющий значение Color.

Свойство KeyTime позволяет задать время, с которого начинается кадр. Например, KeyTime="66%" - действие кадра начинается, когда будет пройдено 66% времени анимации. Или KeyTime="0:0:2" - кадр начинается со 2 секунды.

Кроме линейных есть еще несколько видов кадров анимации:

- Дискретные кадры, например, **DiscreteColorKeyFrame**, они обеспечивают резкую смену кадров.
- Кадры, на основе сплайнов, например, **SplineDoubleKeyFrame**. В этом случае анимация строиться на основе кривой Безье. Свойство **KeySpline** определяет начальную и конечную точки участка сплайна. При чем начальная точка сплайна находится в начале координат (0,0), а конечная - в конце (1,1). Получаемый сплайн является отношением между временем (координата X) и анимируемым значением (координата Y).

Например, используем сплайновую анимацию:

```
<DoubleAnimationUsingKeyFrames
        Storyboard.TargetProperty="(Canvas.Bottom)"
        Duration="0:0:3" AutoReverse="True"
        RepeatBehavior="Forever" DecelerationRatio="1" >
    <SplineDoubleKeyFrame KeySpline="0,1 1,0" KeyTime="0%" Value="0" />
    <SplineDoubleKeyFrame KeySpline="0,1 1,0" KeyTime="33%" Value="60" />
    <SplineDoubleKeyFrame KeySpline="0,1 1,0" KeyTime="66%" Value="120" />
    <SplineDoubleKeyFrame KeySpline="0,1 1,0" KeyTime="100%" Value="180" />
</DoubleAnimationUsingKeyFrames>
```

Анимация пути

Еще один вид анимаций - это анимация пути (path-based animation). Она также анимирует определенное свойство. Ее особенность в том, что для анимации свойства используется объект **PathGeometry**.

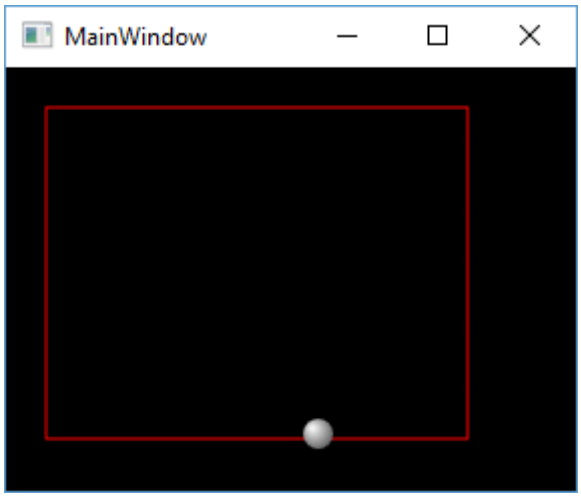
Анимации пути представлены тремя классами: DoubleAnimationUsingPath, MatrixAnimationUsingPath, PointAnimationUsingPath.

Каждый объект анимации с помощью свойства PathGeometry устанавливает ссылку на объект PathGeometry, а свойство Source получает или задает аспект объекта анимации PathGeometry, определяющий ее выходное значение.

Особенно полезны такие анимации при позиционировании объекта в окне приложения:

```
<Window x:Class="AnimationApp.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:AnimationApp"
        mc:Ignorable="d"
        Title="MainWindow" Height="250" Width="300">
    <Window.Resources>
        <PathGeometry x:Key="geometryPath">
            <PathFigure IsClosed="True" StartPoint="10, 10">
                <PolyLineSegment Points="220,10 220,175 10,175" />
            </PathFigure>
        </PathGeometry>
    </Window.Resources>
    <Canvas Background="Black">
        <Path Stroke="Red" Data="{StaticResource geometryPath}" Canvas.Top="10" Canvas.Left="10"
        <Ellipse Width="15" Height="15" Canvas.Top="177" Canvas.Left="120">
            <Ellipse.Fill>
                <RadialGradientBrush RadiusX="1" RadiusY="1" GradientOrigin="0.3, 0.3">
                    <GradientStop Color="White" Offset="0" />
                    <GradientStop Color="Black" Offset="1" />
                </RadialGradientBrush>
            </Ellipse.Fill>
            <Ellipse.Triggers>
                <EventTrigger RoutedEvent="Window.Loaded">
                    <BeginStoryboard>
                        <Storyboard>
                            <DoubleAnimationUsingPath Storyboard.TargetProperty="(Canvas.Top)"
  Duration="0:0:5" RepeatBehavior="Forever"
  PathGeometry="{StaticResource geometryPath}" Source="Y" >
                            </DoubleAnimationUsingPath>
                            <DoubleAnimationUsingPath Storyboard.TargetProperty="(Canvas.Left)"
  Duration="0:0:5" RepeatBehavior="Forever"
  PathGeometry="{StaticResource geometryPath}" Source="X" >
                            </DoubleAnimationUsingPath>
                        </Storyboard>
                    </BeginStoryboard>
                </EventTrigger>
            </Ellipse.Triggers>
        </Ellipse>
    </Canvas>
</Window>
```

При запуске мы увидим, как небольшой шарик бегает по прямоугольному контуру:



Здесь одна из применяемых анимаций имеет следующую форму:

```
<DoubleAnimationUsingPath Storyboard.TargetProperty="(Canvas.Left)"
        Duration="0:0:5" RepeatBehavior="Forever"
        PathGeometry="{StaticResource geometryPath}" Source="X" >
```

В данном случае анимация применяется к свойству Canvas.Left объекта Ellipse. Длится 5 секунд и бесконечно повторяется.

PathGeometry устанавливает ссылку на ресурс, а атрибут Source указывает, что для нового значения свойства Canvas.Left будет использоваться параметр X (координата по оси X) у точек, которые составляют PathGeometry.

Плавность анимации

Для создания плавности анимации используется различные функции плавности или эластичности (Easing functions). Их основная задача - сделать ход анимации более плавным, более естественным.

Рассмотрим самую простейшую функцию анимации - **ElasticEase**. С помощью ее свойства **Oscillations** можно определить количество колебательных движений во время анимации, в итоге анимируемый элемент как бы пружинится.

Свойство **Springiness** позволяет установить жесткость, и чем больше это значение, тем быстрее затухают колебания.

Еще одно свойство **EasingMode** указывает на режим эластичности и может принимать одно из трех значений:

EaseIn: функция плавности применяется в начале анимации

EaseOut: функция плавности применяется в конце анимации

EaseInOut: функция плавности применяется в начале и в конце анимации

Например,

```
<Button Width="70" Height="25" Content="Кнопка">
    <Button.Triggers>
        <EventTrigger RoutedEvent="Loaded">
            <EventTrigger.Actions>
                <BeginStoryboard>
                    <Storyboard>
                        <DoubleAnimation Storyboard.TargetProperty="Width"
   To="150"
   Duration="0:0:5">

                            <DoubleAnimation.EasingFunction>
                                <ElasticEase EasingMode="EaseOut" Oscillations="5" Springiness="
                            </DoubleAnimation.EasingFunction>

                        </DoubleAnimation>
                    </Storyboard>
                </BeginStoryboard>
            </EventTrigger.Actions>
        </EventTrigger>
    </Button.Triggers>
</Button>
```

Настройка функции анимации Oscillations="5" указывает, что будет пять колебаний. А свойсто EasingMode="EaseOut" устанавливает, что эти колебания будут в конце анимации. Если бы мы захотели увидить колебания в начале, то соответственно надо было бы использовать настройку EasingMode="EaseIn"

В WPF определено всего 11 функций плавности анимации:

- BackEase возвращает анимацию назад, а ее свойство Amplitute задает амплитуду возврата
- BounceEase создает эффект отскока, а свойство Bounces устанавливает количество отскоков.
- ElasticEase создает анимацию осцилляций пружины назад и вперед до момента ее полной остановки
- SineEase создает анимацию с помощью формулы синуса
- ExponentialEase создает анимацию с помощью экспоненциальной формулы.
- CircleEase создает анимацию с помощью цикличной функции.
- QuadraticEase функция плавности на основе квадратичной функции f(t) = t2
- CubicEase функция плавности на основе кубической функции f(t) = t3

- QuarticEase создает анимацию с помощью формулы f(t) = t4
- QuinticEase создает анимацию с помощью формулы f(t) = t5
- PowerEase ускоряет анимацию вначале и замедляет в конце, используя функцию f(t) = tp. В зависимости от значения экспоненты p посредством этой функции мы можем замещать функции QuadraticEase, QuarticEase и QuinticEase

Например, применение функции PowerEase:

```
<DoubleAnimation.EasingFunction>
    <PowerEase Power="6" />
</DoubleAnimation.EasingFunction>
```

Или функция BounceEase:

```
<DoubleAnimation.EasingFunction>
    <BounceEase EasingMode="EaseOut" Bounces="3" />
</DoubleAnimation.EasingFunction>
```

Создание своей функции анимации

WPF имеет ограниченный набор функций плавности, и, возможно, в какой-то момент их станет недостаточно. В этом случае можно созать свою функцию. К счастью, определение функции плавности представляет довольно тривиальный процесс.

Например, определим функцию, которая использует для выисления значений функцию f(t) = t6. Для этого добавим в проект следующий класс:

```
public class SexticEase : EasingFunctionBase
{
    protected override double EaseInCore(double normalizedTime)
    {
        return normalizedTime * normalizedTime * normalizedTime
                 * normalizedTime * normalizedTime * normalizedTime;
    }


    protected override Freezable CreateInstanceCore()
    {
        return new SexticEase();
    }
}
```

Метод EaseInCore() выполняет всю работу функции плавности. Он получает текущее значение анимируемого свойства в виде параметра normalizedTime. Здесь мы просто возводит его в

шестую степень. А метод CreateInstanceCore() должен возвращать объект функции анимации.

Применение функции в XAML:

```
<DoubleAnimation Storyboard.TargetProperty="Width"
                 To="150"
                 Duration="0:0:5">

    <DoubleAnimation.EasingFunction>
        <local:SexticEase EasingMode="EaseOut" />
    </DoubleAnimation.EasingFunction>
</DoubleAnimation>
```