XAML Do-It-Yourself シリーズ

第 8 回 アニメーション

Microsoft®

XAML Do-It-Yourself
第 8 回 アニメーション

XAML Do-It-Yourself 第 8 回は、アニメーションについて学習します。XAML (WPF) が提供するアニメーション機能は、時間の経過と共に、コントロールのプロパティを変化させる機能です。コントロールに含まれるほとんどのプロパティに対して、この機能を利用できます。

今回は、アニメーションについて、以下の内容を学習します。

・XAML で記述するアニメーションの基本
・アニメーションの制御

■アニメーションの基本

まずは、ウィンドウにボタン (Button) とラベル (Label) を配置し、ボタンが押されたらラベルの幅が変化するような簡単なアニメーションを作成してみましょう。

ウィンドウに StackPanel を配置し、この中に Button と Label を配置します。Label には "label1" という名前を付けておきます。そして、ボタンが押されたら、Label の幅 (Width) を 100 px から 200 px まで変化させます。これを行うのがアニメーション機能です。

ボタンのクリックによりアニメーションを開始しますので、<Button> 要素にはトリガーを記述します。これは、<Button> 要素の中に <Button.Triggers> 要素を作成し、ボタンが押された際に発生する Button.Click イベントに対応する処理としてアニメーションを記述します。

アニメーションの記述には、<BeginStoryboard> 要素と <Storyboard> 要素を使用し、その中に <DoubleAnimation> 要素を記述します。

```
<Window x:Class="WpfApplication8.MainWindow"
       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
       Title="MainWindow" Height="200" Width="300">
  <StackPanel>
    <Button Margin="20" Height="30" Width="100" Name="button1"
VerticalAlignment="Top" >Start
     <Button.Triggers>
       <EventTrigger RoutedEvent="Button.Click">
         <BeginStoryboard>
           <Storyboard >
             <DoubleAnimation
               Storyboard.TargetName="label1"
               Storyboard.TargetProperty="Width"
               FillBehavior="HoldEnd"
               From="100"
               To="200"
               Duration="0:0:1" />
```

```
          </Storyboard>
        </BeginStoryboard>
      </EventTrigger>
    </Button.Triggers>
  </Button>
  <Label Width="100" Height="40" Name="label1"
HorizontalContentAlignment="Center" VerticalContentAlignment="Center"
Background="Salmon">テキスト ラベル</Label>
 </StackPanel>
</Window>
```

アニメーションを行うには、まずアニメーションの対象となるコントロール (Storyboard.TargetName) と、そのプロパティ (Storyboard.TargetProperty) を指定します。さらに、アニメーションの実行時間の指定 (Duration 属性) と、開始時のプロパティの値 (From 属性)、そして終了時のプロパティの値 (To 属性) を指定します。

上記のリストでは、label1 の Width プロパティの値を 100 から 200 まで 1 秒かけて変化するよう指示しています。Duration 属性は、"時間:分:秒" の形式で記述します。

以上がアニメーションの基本設定です。アニメーションの対象と変化内容を指定することで簡単にアニメーションが作成できます。プログラムを実行すると、ボタンのクリックによってラベルの幅が変化するのを確認できます。

このアニメーションでは <DoubleAnimation> という要素を用いていますが、これはアニメーションを行う Width プロパティのデータ型が Double 型であるためです。

ほかにも、ByteAnimation、Int32Animation といった基本データ型に対応した要素や、座標位置を移動させる PointAnimation、色を変化させる ColorAnimation などの要素が用意されています。これらはアニメーションを行いたいプロパティの型に応じて使い分けるようにします。

なお、プロパティの開始値と終了値を指定するアニメーションでは、From の値から To の値までの間が線形補完されます。つまり、上記の場合では、ラベルの幅が一定の速さで連続的に変化します。どの程度のなめらかさでアニメーションが表示されるかは、実行するハードウェアの性能によって異なります。

●ColorAnimation

色をアニメーションにより変化させたい場合は ColorAnimation を使います。例えばラベルの背景色(Background) を Salmon から Yellow に 0.5 秒かけて変化させたい場合は、次のように記述します。<ColorAnimation> 要素では、From と To に色名を直接指定できます。

```
<ColorAnimation AutoReverse="True"
  BeginTime="0:0:0" Duration="0:0:0.5"
  From="Salmon" To="Yellow"
  Storyboard.TargetName="myBrush"
  Storyboard.TargetProperty="Color" />
```

ColorAnimation を使用する場合に、Storyboard.TargetName 属性にはコントロール名を記述して、Storyboard.TargetProperty 属性に Background を指定しても正しく動作しないことに注意してください。Background は、Color 型ではなく Brush 型なので、別途 Brush オブジェクトを用意し、その名前を指定します。

そのため、次のように <Label> 要素内に <Label.Background> 要素を作成し、この中でブラシ(SolidColorBrush) を作成しておきます。

```
<Label Width="100" Height="40" Name="label1" HorizontalContentAlignment="Center"
VerticalContentAlignment="Center" >テキスト ラベル
  <Label.Background>
    <SolidColorBrush x:Name="myBrush" Color="Salmon"/>
  </Label.Background>
</Label>
```

ブラシの色をアニメーションさせることにより、そのブラシで塗っているラベルの色が変化するというわけです。また、Label の Background プロパティに色名を指定すると、自動的に SolidColorBrush が割り当てられます。

```
<Label Width="100" Height="40" Name="label1" HorizontalContentAlignment="Center"
VerticalContentAlignment="Center" Background="Salmon">テキスト ラベル</Label>
```

このとき、ColorAnimation は次のように記述することもできます。

```
<ColorAnimation AutoReverse="True"
  BeginTime="0:0:0" Duration="0:0:0.5"
  From="Salmon" To="Yellow"
  Storyboard.TargetName="label1"
  Storyboard.TargetProperty="(Background).(SolidColorBrush.Color)" />
```

これは、label1 コントロールの Background プロパティに割り当てられている SolidColorBrush オブジェクトの Color プロパティを対象にするという意味になります。こうすることで、Brush オブジェクトに名前を付ける必要がなくなります。

●ThicknessAnimation

ThicknessAnimation は、コントロールの外側の枠線（Border)、マージン（Margin)、パディング(Padding) といった Thickness 型のプロパティをアニメーションさせたい場合に利用します。

ThicknessAnimation では From と To でそれぞれ 4 つのパラメーターを指定します。これらは枠線の高さと幅に対応します (左、上、右、下の順)。

```
<ThicknessAnimation
  Storyboard.TargetProperty="BorderThickness"
  Duration="0:0:2"
  FillBehavior="HoldEnd"
  From="1,1,1,1"
  To="28,14,28,14" />
```

●FillBehavior 属性

この記述にあるように、FillBehavior 属性を設定することで、Duration 属性で指定した時間が経過した後の振る舞いを指定できます。FillBehavior 属性が "HoldEnd" の場合は、アニメーション終了時の状態を保持します。それに対して FillBehavior 属性を "Stop" に設定すると、アニメーション終了時には、開始直前の状態に戻ります。

■キーフレームによるアニメーション

先ほどのアニメーションでは、Width プロパティを 100 px から 200 px まで 1 秒間かけて変化するように指定しました。これとは別の方法として、キーフレームを指定したアニメーションも可能です。

キーフレームによるアニメーションでは、その名前のとおり、任意の時間を指定して、その時間ごとの状態を設定することが可能になります。

例えば下記のような Height プロパティに対するアニメーションでは、アニメーション開始時の Height プロパティは 40 px で、そこから 1 秒後に 80 px まで変化し、その後 1 秒間は そのまま、そして次の 1 秒間で 80 px から 120 px に 変化します。

```
 <DoubleAnimationUsingKeyFrames
  Storyboard.TargetName="label1"
  Storyboard.TargetProperty="Height"
  Duration="0:0:4">
  <LinearDoubleKeyFrame Value="40" KeyTime="0:0:0"/>
  <LinearDoubleKeyFrame Value="80" KeyTime="0:0:1"/>
  <LinearDoubleKeyFrame Value="80" KeyTime="0:0:2"/>
  <LinearDoubleKeyFrame Value="120" KeyTime="0:0:3"/>
</DoubleAnimationUsingKeyFrames>
```

この場合は Height プロパティが Double 型であるため、<DoubleAnimationUsingKeyFrames> 要素を使います。さらに、それぞれのキーフレームの設定は <LinearDoubleKeyFrame> 要素を使って指定

します。

<LinearDoubleKeyFrame> 要素はキーフレームで指定した値を線形補完します。これに対して、<DiscreteDoubleKeyFrame> 要素を使ってキーフレームを指定すると、線形補完が行われず、指定された時間になった時点でプロパティが変更され、結果的に不連続なアニメーションとなります。

●StringAnimationUsingKeyFrames

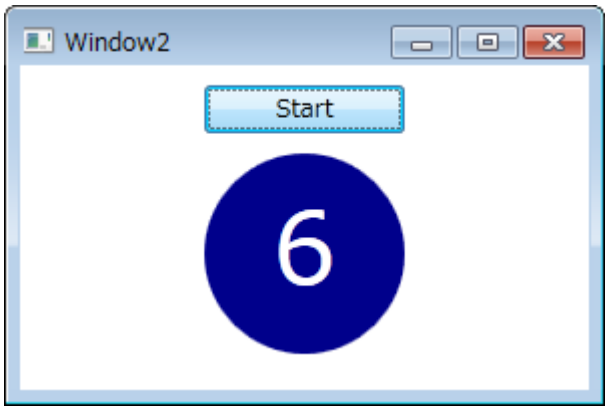
<StringAnimationUsingKeyFrames> 要素と、<DiscreteDoubleKeyFrame> 要素を使うと、時間の経過と共に文字列の表示内容が変化するアニメーションを作成できます。

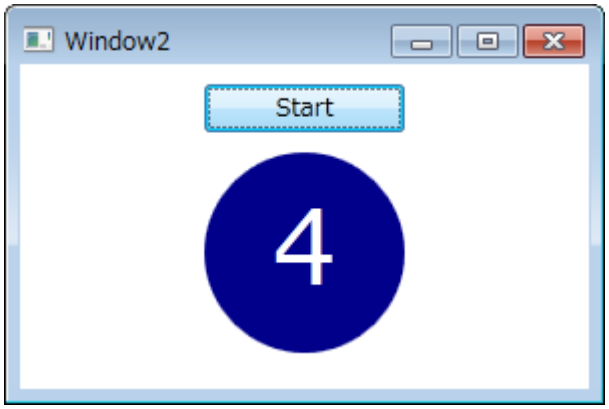
次の例では、ボタンを押すと、青く描画された円内で数字をカウントダウンします。1 秒ごとに <DiscreteStringKeyFrame> 要素を用意し、Label に表示する文字列を設定しています。

```
<Window  x:Class="WpfApplication8.Window2"
       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
       Title="Window2" Height="200" Width="300">
  <StackPanel>
    <Button Width="100" Margin="10">Start
      <Button.Triggers>
        <EventTrigger RoutedEvent="Button.Click">
          <BeginStoryboard>
            <Storyboard  >
              <StringAnimationUsingKeyFrames
                Storyboard.TargetName="label1"
                Storyboard.TargetProperty="Content"
                BeginTime="0:0:0" >
              <DiscreteStringKeyFrame Value="10" KeyTime="0:0:0" />
              <DiscreteStringKeyFrame Value="9" KeyTime="0:0:1" />
              <DiscreteStringKeyFrame Value="8" KeyTime="0:0:2" />
              <DiscreteStringKeyFrame Value="7" KeyTime="0:0:3" />
              <DiscreteStringKeyFrame Value="6" KeyTime="0:0:4" />
              <DiscreteStringKeyFrame Value="5" KeyTime="0:0:5" />
              <DiscreteStringKeyFrame Value="4" KeyTime="0:0:6" />
              <DiscreteStringKeyFrame Value="3" KeyTime="0:0:7" />
              <DiscreteStringKeyFrame Value="2" KeyTime="0:0:8" />
              <DiscreteStringKeyFrame Value="1" KeyTime="0:0:9" />
              <DiscreteStringKeyFrame Value="0" KeyTime="0:0:10" />
              </StringAnimationUsingKeyFrames>
            </Storyboard>
          </BeginStoryboard>
        </EventTrigger>
      </Button.Triggers>
    </Button>
    <Label Name="label1" Height="100" Width="100" Content="-" FontSize="50"
Foreground="White">
      <Label.Template>
        <ControlTemplate TargetType="Label">
          <Grid>
            <Ellipse x:Name="myEllipse" Fill="DarkBlue" />
```

```
          <ContentPresenter HorizontalAlignment="Center"
VerticalAlignment="Center" />
        </Grid>
      </ControlTemplate>
    </Label.Template>
   </Label>
  </StackPanel>
</Window>
```

1 秒ごとに表示文字列が変わる





■繰り返しと折り返し

アニメーションが終了すると、上述したように FillBehavior 属性によりその終了状態が決まります。これに加えて、<Storyboard> 要素に AutoReverse 属性や、RepeatBehavior 属性を記述することで、アニメーションをカスタマイズできます。

AutoReverse 属性を "True" に設定した場合、アニメーションが最後まで実行すると、今度は動画を逆再生するように、逆方向のアニメーションを行います。そのため、この属性を指定してアニメーションを実行すると、終了時はアニメーション開始時の状態に戻ります。

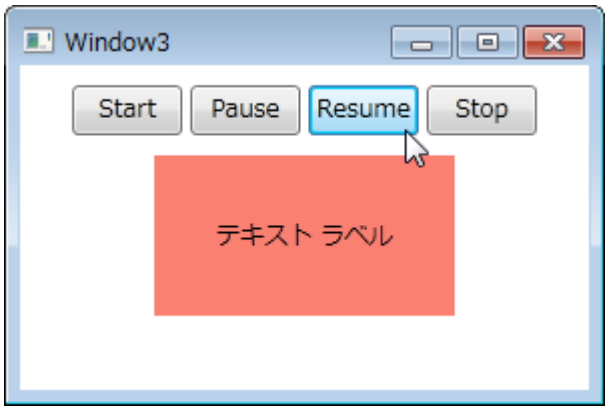
RepeatBehavior 属性は、<Storyboard> で定義したアニメーションの繰り返しを指定します。

"Forever" を指定すると、アニメーションを継続して繰り返します。時間を指定すると、その時間内でアニメーションを繰り返します。

■アニメーションをコントロールする

実行中のアニメーションを停止したり、再開させたりすることも XAML で記述できます。これは <BeginStoryboard> 要素に名前を付けておき、必要に応じて <StopStoryboard>、<ResumeStoryboard> といった要素に Storyboard 名を設定することで実現できます。

例として、「Start」、「Pause」、「Resume」、「Stop」といったボタンを配置し、それぞれのボタンのイベントでアニメーションをコントロールしてみましょう。



例えば、Stop ボタンである stopButton が押された際には、<StopStoryboard> 要素の BeginStoryboardName 属性に <BeginStoryboad> 要素の名前を設定します。これによりアニメーションの実行が停止します。他のボタンも同様にして記述できます。

```
<Window x:Class="WpfApplication8.Window3"
       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
       Title="Window3" Height="200" Width="300">
  <Window.Resources>
    <SolidColorBrush x:Key="myBrush" Color="Salmon" />
    <Style TargetType="Button">
      <Setter Property="Margin" Value="2 10 2 10"/>
      <Setter Property="Width" Value="50"/>
      <Setter Property="Height" Value="25"/>
    </Style>
  </Window.Resources>
  <StackPanel>
    <StackPanel Orientation="Horizontal" HorizontalAlignment="Center">
      <Button Name="startButton" >Start</Button>
      <Button Name="pauseButton" >Pause</Button>
      <Button Name="resumeButton" >Resume</Button>
      <Button Name="stopButton" >Stop</Button>
```

```
    <StackPanel.Triggers>
      <!-- 開始 ボタン -->
      <EventTrigger RoutedEvent="Button.Click" SourceName="startButton">
        <BeginStoryboard Name="myStoryboard">
          <Storyboard   AutoReverse="True" RepeatBehavior="Forever" >
            <!-- 幅 (Width) をアニメーション -->
            <DoubleAnimation
                Storyboard.TargetName="label1"
                Storyboard.TargetProperty="Width"
                From="100"
                To="200"
                Duration="0:0:4" >
            </DoubleAnimation>
            <!-- 高さ (Height) をアニメーション -->
            <DoubleAnimationUsingKeyFrames
                Storyboard.TargetName="label1"
                Storyboard.TargetProperty="Height"
                Duration="0:0:4">
                <DiscreteDoubleKeyFrame Value="40" KeyTime="0:0:0"/>
                <DiscreteDoubleKeyFrame Value="80" KeyTime="0:0:1"/>
                <DiscreteDoubleKeyFrame Value="120" KeyTime="0:0:2"/>
            </DoubleAnimationUsingKeyFrames>
            <!-- 背景色 (Background) をアニメーション -->
            <ColorAnimation AutoReverse="True"
                BeginTime="0:0:0" Duration="0:0:0.5"
                From="Salmon" To="Yellow"
                Storyboard.TargetName="myBrush"
                Storyboard.TargetProperty="Color">
            </ColorAnimation>
          </Storyboard>
        </BeginStoryboard>
      </EventTrigger>

      <!-- アニメーションを中断 -->
      <EventTrigger RoutedEvent="Button.Click" SourceName="pauseButton">
        <PauseStoryboard BeginStoryboardName="myStoryboard" />
      </EventTrigger>

      <!-- アニメーションを再開 -->
      <EventTrigger RoutedEvent="Button.Click" SourceName="resumeButton">
        <ResumeStoryboard BeginStoryboardName="myStoryboard" />
      </EventTrigger>

      <!-- アニメーションをストップ -->
      <EventTrigger RoutedEvent="Button.Click" SourceName="stopButton">
        <StopStoryboard BeginStoryboardName="myStoryboard" />
      </EventTrigger>

    </StackPanel.Triggers>
  </StackPanel>
  <Label Width="100" Height="40" Name="label1"
HorizontalContentAlignment="Center" VerticalContentAlignment="Center">テキスト
ラベル
    <Label.Background>
```

```
      <SolidColorBrush x:Name="myBrush" Color="Salmon"/>
    </Label.Background>
  </Label>
 </StackPanel>
</Window>
```

■まとめ

今回はアニメーションの具体例をいくつか紹介しました。コード ビハインド ファイルにコードを書かなくても、簡単な XAML の記述だけでアニメーションが実現できることを学習しました。

次回は、カスタム コントロールについて学習します。