XAML Do-It-Yourself シリーズ

第 6 回 スタイル

Microsoft®

XAML Do-It-Yourself
第 6 回 スタイル

XAML Do-It-Yourself 第 6 回は、スタイルについて学習します。スタイルというと、Microsoft Word での書体や段落、文字の大きさなどをまとめたものや、HTML の CSS (Cascading Style Sheet) を思い浮かべるかもしれません。XAML で使用するスタイルもそれらと同じく、ユーザー インターフェイスの表示に関わる設定をまとめて管理するものです。

今回は、次の項目について学習します

・スタイルの作成
・スタイルの継承
・トリガーの作成

■スタイルとは

XAML におけるスタイルとは、XAML で定義するさまざまな要素にセットするプロパティやトリガーなどの設定をまとめる働きをします。これにより、ウィンドウやコントロールの描画設定をまとめて行うことができます。

たとえば、アプリケーションで使用するボタンはすべて横幅が 80 px (ピクセル)、高さが 40 px で統一したい、といった場合は、スタイルを利用してデフォルトの Button の大きさを決めておけば、それをすべての Button に適用できます。

また、スタイルの一部としてトリガーを記述することで、特定のプロパティの変化に応じて、別のプロパティの値を変更することもできます。

■スタイルの作成

まずは Button に適用できるスタイルを作成してみましょう。ボタンのボーダー (外側の輪郭) とボタンの背景にグラデーションを加えるスタイルです。

なお、スタイルはコントロールごとに個々に定義することも可能ですが、スタイルにはユーザー インターフェイスの外観を統一する目的もありますので、個々のコントロールそれぞれにスタイルを指定してもあまり意味がありません。通常は <Window> 要素の Resources プロパティでリソースとして定義します。これにより <Window> 要素に含まれるコントロールに対してスタイルを適用できます。

スタイルの定義は <Style> 要素で行います。<Style> 要素には TargetType 属性を記述して、そのスタイルを適用するコントロールを指定します。

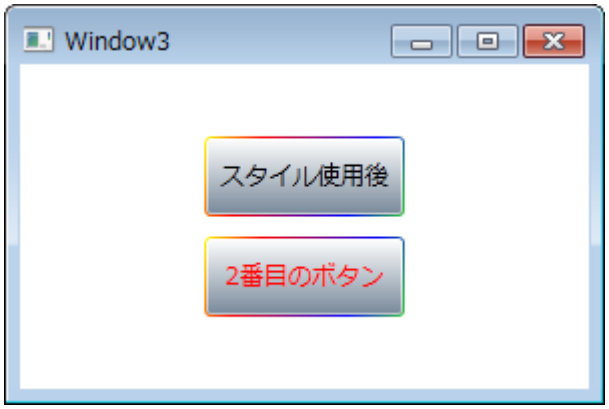
ここでは Button をターゲットとし、コントロールの幅 (Width) と高さ (Height)、前景色 (Foreground)、背景色 (Background) を指定します。前景色と背景色には、前回でも使用した LinearGradientBrush (線形グラデーション ブラシ) を使っています。

```
<Window.Resources>
  <Style TargetType="Button">
    <Setter Property="Foreground" Value="Black"/>
    <Setter Property="Height" Value="40"/>
    <Setter Property="Width" Value="100"/>
    <Setter Property="Margin" Value="5" />
    <Setter Property="BorderBrush">
      <Setter.Value>
        <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>
      </Setter.Value>
    </Setter>
    <Setter Property="Background" >
      <Setter.Value>
        <LinearGradientBrush StartPoint="0,0" EndPoint="0,1">
          <GradientStop Color="White" Offset="0.0"/>
          <GradientStop Color="LightSlateGray" Offset="1.0"/>
        </LinearGradientBrush>
      </Setter.Value>
     </Setter>
  </Style>
</Window.Resources>
```

次にウィンドウに Button を配置します。上下に 2 個並べて配置しますが、上側の Button ではボタンに表示する文字列以外は何も設定しません。2 番目の Button では Foreground 属性のみを設定します。

```
<StackPanel VerticalAlignment="Center">
  <Button>スタイル使用後</Button>
  <Button Foreground="Red">2 番目のボタン</Button>
</StackPanel>
```

実行すると、<Style> 要素で設定したプロパティが有効になっているのが分かります。下側のボタンは Foreground 属性を改めて指定 (オーバーライド) していますので、<Style> 要素で設定した内容とは異なっています。個々の要素でプロパティを再度指定した場合は、そちらの内容が優先されます。



■スタイルの継承

既存のスタイルを継承して、新たなスタイルを定義することも可能です。上記の例の下側のボタンでは、ボタンの文字の色を直接指定していますが、これをスタイルで行ってみましょう。

先ほど作成した XAML ファイルと同様、<Window.Resources> 要素に <Style> 要素を追加しますが、ここでは x:Key 属性を使ってスタイルに明示的に名前を付けます。さらに TargetType 属性を記述してターゲットのコントロールの指定し、BaseOn 属性を用いて継承元のスタイルを指定します。

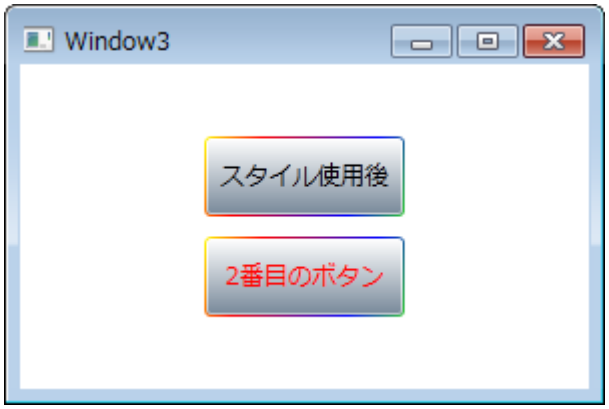
BaseOn 属性の記述では、継承元のスタイルを定義しているリソースを "x:Type Button" という具合に、"x:Type" をつけて指定する必要があります。設定するプロパティは、上述したプロパティの設定と同様に、<Setter> 要素によりプロパティ名と値を指定します。

```
<Style x:Key="myRedButton" TargetType="Button" BasedOn="{StaticResource {x:Type Button}}">
  <Setter Property="Foreground" Value="Red" />
</Style>
```

●継承したスタイルを参照する

継承したスタイルを利用する場合は、利用したいコントロールで Style 属性により、スタイルの名前を指定します。

```
<Button Style="{StaticResource myRedButton}">2 番目のボタン</Button>
```



■TargetType を指定しないスタイル

さらに <Style> 要素で TargetType 属性を記述しないことで、一部のコントロールのみを対象にしたスタイルの定義も可能です。この場合は x:Key 属性を記述してスタイルを識別します。TargetType 属性は記述しません。

この場合、スタイルとして設定するプロパティがどういったコントロールのプロパティかを示す必要があります。Button や TextBox などのユーザー インターフェイスを提供するクラスの基本クラスである Control を指定することもできます。これにより、特定のタイプのコントロールに依存しないスタイルを定義できます。

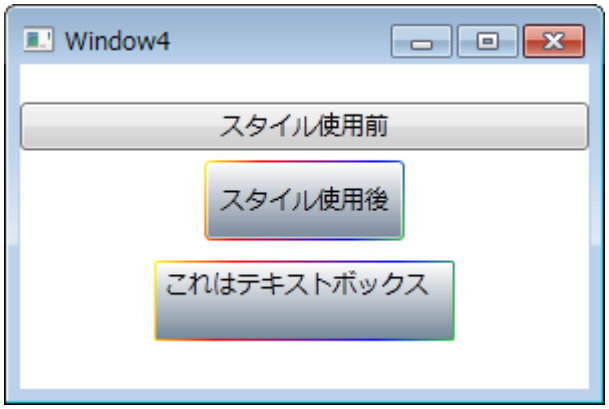
例として、先ほどのスタイルを変更してみましょう。

```
<Window x:Class="WpfApplication5.Window4"
       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
       Title="Window4" Height="200" Width="300">
  <Window.Resources>
    <Style x:Key="myStyle" >
      <Setter Property="Control.Foreground" Value="Black"/>
      <Setter Property="Control.Height" Value="40"/>
      <Setter Property="Control.Width" Value="100"/>
      <Setter Property="Control.Margin" Value="5" />
      <Setter Property="Control.BorderBrush">
        <Setter.Value>
          <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>
        </Setter.Value>
      </Setter>
      <Setter Property="Control.Background" >
        <Setter.Value>
```

```
        <LinearGradientBrush StartPoint="0,0" EndPoint="0,1">
          <GradientStop Color="White" Offset="0.0"/>
          <GradientStop Color="LightSlateGray" Offset="1.0"/>
        </LinearGradientBrush>
      </Setter.Value>
    </Setter>
  </Style>
 </Window.Resources>
 <StackPanel VerticalAlignment="Center">
  <Button >スタイル使用前</Button>
  <Button Style="{StaticResource myStyle}">スタイル使用後</Button>
  <TextBox Style="{StaticResource myStyle}" Width="150">これはテキストボックス
</TextBox>
 </StackPanel>
</Window>
```

ここでは同じスタイル myStyle を適用した Button と TextBox をパネルに配置しています。

myStyle では、プロパティ設定する対象をそれらの基本クラスである Control としていますので、そのスタイルは Button にも TextBox にも適用されます。一方、最上部に配置した Button はスタイルを指定していないので、スタイルは適用されていません。



■トリガー

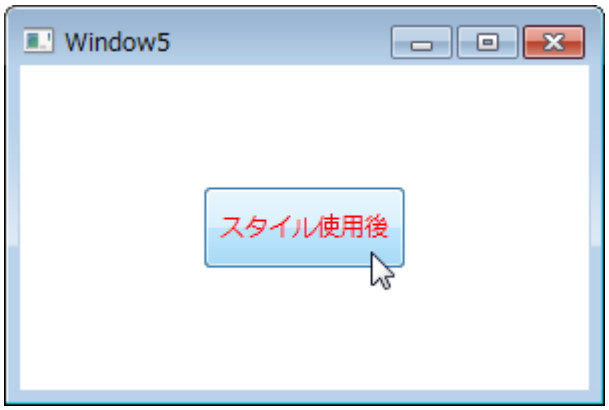
さらに、スタイルにトリガーを設定することで、特定の条件下でスタイルを変更できるようになります。これにより、コントロールのプロパティが変化した際に、あらかじめ決められたプロパティを変更することができます。

例として、Button に含まれる IsMouseOver プロパティが False から True に変化したときに、Button の背景が異なるブラシで描画されるように、トリガーを追加してみましょう。

トリガーを記述するには <Style> 要素に <Style.Triggers> 要素を追加し、動作条件と、条件にマッチした際に設定するプロパティの値を記述します。

ここでは、IsMouseOver が True になったら、Foreground プロパティの値を <Setter.Value> 要素で設定した色 (Red) で描画するようにします。マウスが Button から離れると、IsMouseOver が False になり、Background プロパティの値は自動的に元の内容に戻ります。

```
<Window x:Class="WpfApplication5.Window5"
       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
       Title="Window5" Height="200" Width="300">
  <Window.Resources>
    <Style TargetType="Button">
      <Setter Property="Height" Value="40"/>
      <Setter Property="Width" Value="100"/>
      <Style.Triggers>
        <Trigger Property="IsMouseOver" Value="True">
          <Trigger.Setters>
            <Setter Property="Foreground" Value="Red" />
          </Trigger.Setters>
        </Trigger>
      </Style.Triggers>
    </Style>
  </Window.Resources>
  <StackPanel VerticalAlignment="Center">
    <Button>スタイル使用後</Button>
  </StackPanel>
</Window>
```



●イベント トリガー

プロパティの変化ではなく、マウスがクリックされたなどの、イベントに応じてスタイルを変更する場合は、<EventTrigger> 要素を用いてプロパティを設定します。ただし、<EventTrigger> 要素を利用する場合には、注意点が 2 つあります。

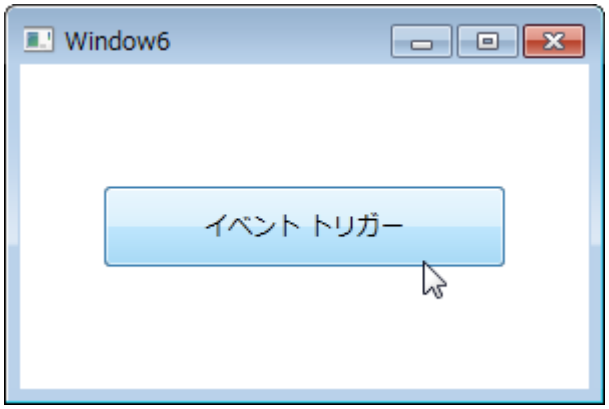
1 つは、通常のトリガーと異なり、<Setter> 要素を使ってプロパティの値を直接設定することができない点です。<EventTrigger> 要素で指定できるのは、ストーリー ボード (Storyboard) に限られていま

す。ストーリー ボードについては、アニメーションの回で学習します。

もう 1 つは、<EventTrigger> 要素でイベントを処理した場合は、イベントが発生する前の状態には自動的に戻らないという点です。

例えば、マウスの状態を監視するなら、MouseEnter イベントと MouseLeave イベントを対にしてトリガーを記述する必要があります。ここでは、MouseEnter イベントで Button の幅を 200 px に変更し、MouseLeave イベントで元の幅の 100 px に戻しています。

```
<Window x:Class="WpfApplication5.Window6"
       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
       Title="Window6" Height="200" Width="300">
  <Window.Resources>
    <Style TargetType="Button">
      <Setter Property="Height" Value="40"/>
      <Setter Property="Width" Value="100"/>
      <Setter Property="Margin" Value="5" />
      <Style.Triggers>
        <EventTrigger RoutedEvent="MouseEnter">
          <BeginStoryboard>
            <Storyboard>
              <DoubleAnimation To="200" Duration="0:0:0.1"
                 Storyboard.TargetProperty="(Button.Width)"/>
            </Storyboard>
          </BeginStoryboard>
        </EventTrigger>
        <EventTrigger RoutedEvent="MouseLeave">
          <BeginStoryboard>
            <Storyboard>
              <DoubleAnimation To="100" Duration="0:0:1"
                Storyboard.TargetProperty="(Button.Width)" />
            </Storyboard>
          </BeginStoryboard>
        </EventTrigger>
      </Style.Triggers>
    </Style>
  </Window.Resources>
  <StackPanel VerticalAlignment="Center">
    <Button>イベント トリガー</Button>
  </StackPanel>
</Window>
```



アプリケーションを実行し、マウスをボタンの上に移動する (MouseEnter) と、幅が 200px に変化し、マウスをボタンの外側に移動すると (MouseLeave)、100 px に戻ります (実際には、元に戻るのではなく、100 px に再設定しているだけです)。

■まとめ

今回学習したスタイルを利用することで、ユーザー インターフェイスの外見を効率よく作成できることが理解できました。またコントロールの外見を、コード ビハインドで処理することなく、ユーザーの操作に応じて変化させることが可能であることも学習しました。

次回は、既存のコントロールを拡張できるテンプレートについて学習します。