XAML Do-It-Yourself シリーズ

第 7 回 テンプレート

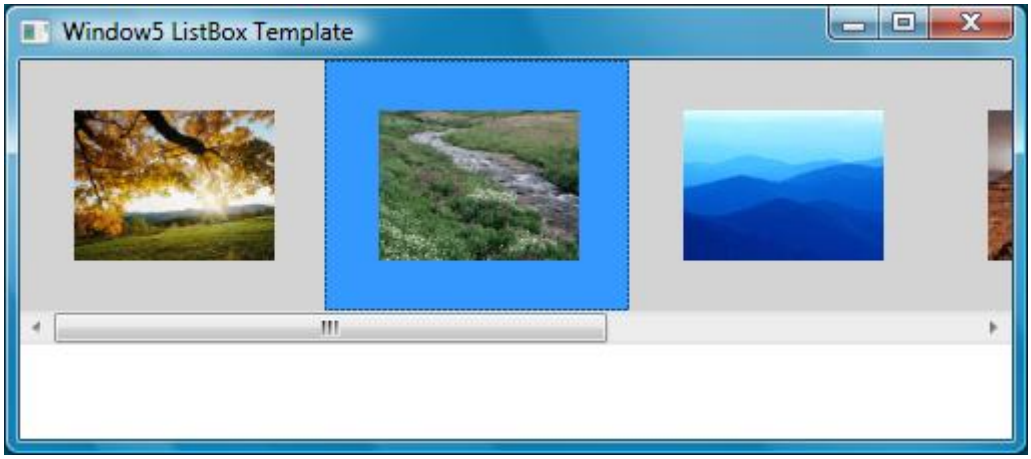
Microsoft®

XAML Do-It-Yourself
第 7 回 テンプレート

XAML Do-It-Yourself 第 7 回は、テンプレートについて学習します。テンプレートを使うと、コントロールの外観を拡張できます。

今回は、テンプレートを使って、以下の内容を学習します。

・テンプレートによるコントロールのカスタマイズ
・バインディング データの表示に使用するテンプレート
・条件に応じたテンプレートの切り替え



テンプレートにより水平方向にスクロールするように変更した ListBox コントロール

テンプレートには、コントロール テンプレート (ControlTemplate) と、データ テンプレート (DataTemplate) の 2 種類があります。まずコントロール テンプレートについて見ていきます。

■コントロール テンプレート (ControlTemplate) を使った丸いボタン

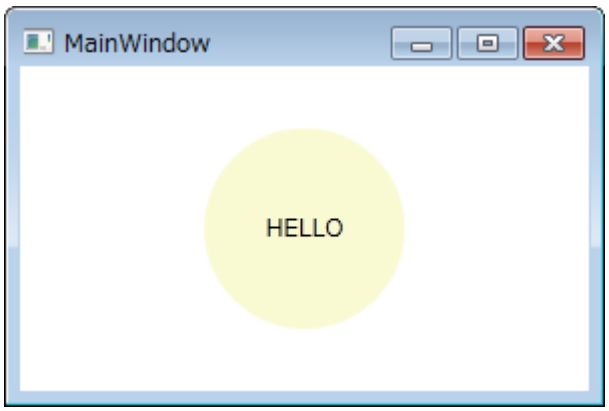
「第 6 回 スタイル」で学習したように、スタイルを用いることにより、コントロールの外観を変更することができました。例えば、ボタンの輪郭にグラデーションを付けたり、ボタンの背景を任意の色で描画したりできます。スタイルを設定する場合には、<Setter> 要素を使って、コントロールに含まれるプロパティの値を変更しました。

これに対し、テンプレートである ControlTemplate を使用すると、コントロールの外観の構成を変更できます。これにより、例えば楕円のボタンを作ることができます。Button の外観を構成する要素を定義することで、機能的にはボタンでありながら、外観が異なるコントロールを自由に作成できるわけです。

次のような XAML を記述すると、形状が円 (楕円) の Button を作成できます。

```
<Window x:Class="WpfApplication7.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">
  <Grid>
    <Button Content="HELLO" Width="100" Height="100">
      <Button.Template>
        <ControlTemplate TargetType="Button">
          <Grid>
            <Ellipse x:Name="myEllipse" Fill="LightGoldenrodYellow" />
            <ContentPresenter HorizontalAlignment="Center"
VerticalAlignment="Center" />
          </Grid>
        </ControlTemplate>
      </Button.Template>
    </Button>
  </Grid>
</Window>
```

実行すると、次のような画面が表示されます。一見するとボタンには見えませんが、マウスによるクリック イベントや Button が持つプロパティが利用できます。



ここでは、Button の Template プロパティに <ControlTemplate> 要素を用い、そこに Grid を使って描画領域を確保し、Ellipse (楕円) を描画しています。<ControlTemplate> 要素でコントロールの外観を定義し、これをコントロールの Template プロパティとして指定することで、コントロールの外観を変更できます。

ここではさらに、ボタンの文字列を描画するための ContentPresenter も配置しています。ContentPresenter は、Label や Button に表示する文字列など、Content プロパティとして設定する文字列を表示するものです。例えば、

```
<Button>こんにちは</Button>
```

といった記述は、"こんにちは" が Content プロパティとして扱われます。以前に説明しましたが、これ

は、

```
<Button Content=”こんにちは” />
```

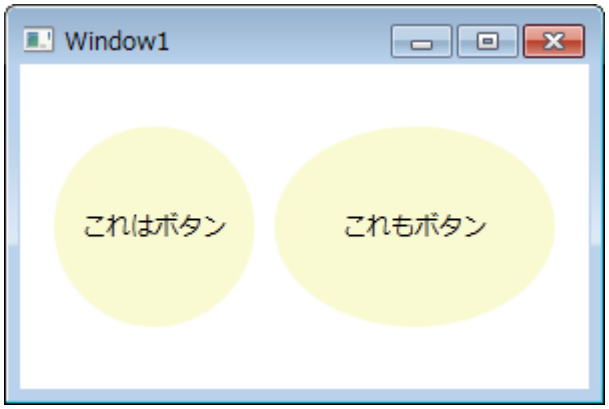
と等価です。そしてこの Content プロパティを設定するために用いるのが、ContentPresenter です。

なお、<Button> 要素内にテンプレートを記述すると、その Button でしかテンプレートが有効になりません。テンプレートを汎用的に利用したい場合は、<Window> 要素のリソースとしてテンプレートを記述します。これにより、ウィンドウに配置されたボタンをすべて丸いボタンにするといったことが可能です。

これには、<Windows.Resources> 要素の <Style> 要素に Template プロパティを指定し、<Setter> 要素を使って、テンプレートの内容を <ControlTemplate> 要素で記述します。以下のコードではさらに、BitmapEffect プロパティに BevelBitmapEffect を設定することで斜め方向の影をつけて少しボタン風にしています。

```
<Window x:Class="WpfApplication7.Window1"
       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
       Title="Window1" Height="200" Width="300">
  <Window.Resources>
    <Style TargetType="Button">
      <Setter Property = "BitmapEffect" >
        <Setter.Value>
          <BevelBitmapEffect BevelWidth="1" />
        </Setter.Value>
      </Setter>
      <Setter Property="Template">
        <Setter.Value>
          <ControlTemplate TargetType="Button">
            <Grid>
              <Ellipse x:Name="myElipse" Fill="LightGoldenrodYellow" />
              <ContentPresenter HorizontalAlignment="Center"
VerticalAlignment="Center"></ContentPresenter>
            </Grid>
          </ControlTemplate>
        </Setter.Value>
      </Setter>
    </Style>
  </Window.Resources>
  <Grid>
    <StackPanel  HorizontalAlignment="Center" VerticalAlignment="Center"
Orientation="Horizontal">
      <Button Margin="5" Width="100" Height="100">これはボタン</Button>
      <Button Margin="5" Width="140" Height="100">これもボタン</Button>
    </StackPanel>
   </Grid>
</Window>
```

これを実行すると、水平に並んだ 2 つの楕円のボタンが表示されます。ただし、ボタンが押された際のスタイルは設定していませんので、クリックしても表示は変化しません。



■水平方向にスクロールするリストボックス

以前に紹介したように、ListBox は文字列以外のものでもリストとして一覧表示できます。以下のコードは、ListBox のアイテムに画像を表示するものです。

```
<Window x:Class="WpfApplication7.Window2"
       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
       Title="Window2" Height="300" Width="300">
  <Window.Resources>
    <Style TargetType="Image">
      <Setter Property="Width" Value="100" />
      <Setter Property="Margin" Value="10" />
    </Style>
  </Window.Resources>
  <Grid>
    <ListBox>
      <Image Source="img¥Autumn Leaves.jpg"/>
      <Image Source="img¥Creek.jpg"/>
      <Image Source="img¥Lighthouse.jpg"/>
      <Image Source="img¥Desert.jpg"/>
      <Image Source="img¥Dock.jpg"/>
    </ListBox>
  </Grid>
</Window>
```

ListBox のアイテムとして設定された画像は、次のように縦に並んで表示されます。

ここではテンプレートを利用し、横方向にスクロールするリストボックスを記述してみましょう。

先ほどの Button の例と同様に、ListBox の Template プロパティに ControlTemplate を設定します。

横スクロールには、ScrollViewer を用いてスクロール バー付きの領域を作成し、その中に StackPanel を配置して Orientation プロパティを Horizontal（水平）に設定しておきます。これらを <ControlTemplate> 要素に記述すると、ListBox のアイテムとして指定された画像が、水平方向に並んで表示されるようになります。

```
<Window x:Class="WpfApplication7.Window3"
       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
       Title="Window3" Height="200" Width="500">
  <Window.Resources>
    <Style TargetType="Image">
      <Setter Property="Width" Value="120" />
      <Setter Property="Margin" Value="5" />
    </Style>
    <Style TargetType="ListBox">
      <Setter Property="Template">
        <Setter.Value>
          <ControlTemplate TargetType="ListBox">
             <ScrollViewer HorizontalScrollBarVisibility="Auto"
Background="LightGray">
               <StackPanel IsItemsHost="True" Orientation="Horizontal"/>
             </ScrollViewer>
          </ControlTemplate>
        </Setter.Value>
      </Setter>
    </Style>
```

```
  </Window.Resources>
  <StackPanel>
    <ListBox Name="listBox1">
      <Image Source="img¥Autumn Leaves.jpg"/>
      <Image Source="img¥Creek.jpg"/>
      <Image Source="img¥Lighthouse.jpg"/>
      <Image Source="img¥Desert.jpg"/>
      <Image Source="img¥Dock.jpg"/>
    </ListBox>
  </StackPanel>
</Window>
```

実行すると、画像が水平方向に並んだリストボックスが表示されます。

■データ テンプレート (DataTemplate)

次に DataTemplate を使ってみましょう。DataTemplate は、データ バインディングを行う際に、バインドした項目の表示形式を定義するためのものです。「データ バインディング」の回で紹介したように、DataTemplate を利用すると、バインドしたデータの表示をカスタマイズできます。

XML ファイルの内容を ListBox で表示する例をここでもう一度見てみましょう。まず XML データの内容を確認しておきます。

```
<?xml version="1.0" encoding="UTF-8" ?>
<Inventory xmlns="">
  <Books>
    <Book ISBN="0-7356-0562-9" Stock="in" Number="9">
      <Title>XML in Action</Title>
      <Summary>XML Web Technology</Summary>
    </Book>
    <Book ISBN="0-7356-1370-2" Stock="in" Number="8">
      <Title>Programming Microsoft Windows With C#</Title>
      <Summary>C# Programming using the .NET Framework</Summary>
    </Book>
    <Book ISBN="0-7356-1288-9" Stock="out" Number="7">
```

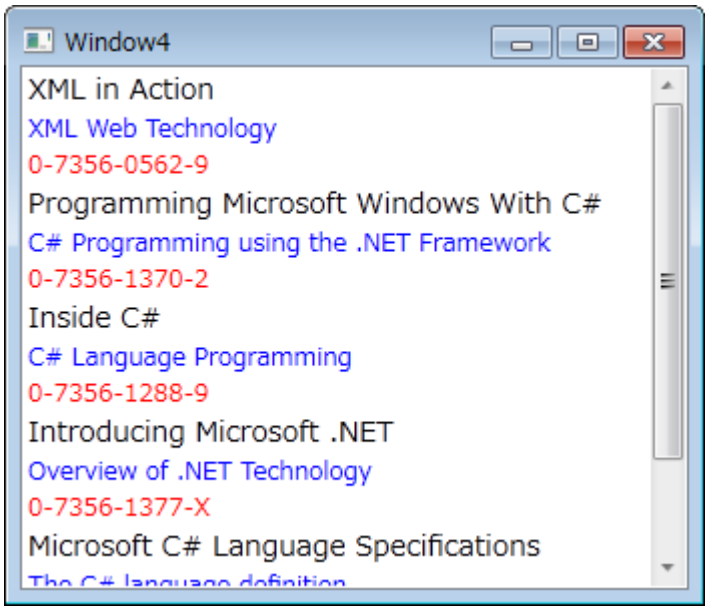
```
    <Title>Inside C#</Title>
    <Summary>C# Language Programming</Summary>
   </Book>
   <Book ISBN="0-7356-1377-X" Stock="in" Number="5">
    <Title>Introducing Microsoft .NET</Title>
    <Summary>Overview of .NET Technology</Summary>
   </Book>
   <Book ISBN="0-7356-1448-2" Stock="out" Number="4">
    <Title>Microsoft C# Language Specifications</Title>
    <Summary>The C# language definition</Summary>
   </Book>
  </Books>
</Inventory>
```

リスト bookdata.xml の内容

この内容を、次のような DataTemplate を用意して表示します。この設定では、XML データに含まれる <Title> 要素と <Summary> 要素の内容を ListBox に表示します。XML データに含まれる属性値を表示したい場合には、属性名に "@" を付けて (@ISBN のように) バインド先の XPath に指定します。

```
<Window x:Class="WpfApplication7.Window4"
       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
       Title="Window4" Height="300" Width="350">
  <Window.Resources>
    <XmlDataProvider x:Key="BookData" Source="bookdata.xml"
XPath="Inventory/Books/*"/>
    <DataTemplate x:Key="myTemplate">
      <StackPanel>
        <TextBlock FontSize="14" Text="{Binding XPath=Title}"/>
        <TextBlock Foreground="Blue" Text="{Binding XPath=Summary}"/>
        <TextBlock Foreground="Red" Text="{Binding XPath=@ISBN}"/>
      </StackPanel>
    </DataTemplate>
  </Window.Resources>
  <Grid>
    <ListBox Width="300" HorizontalContentAlignment="Stretch"
      ItemTemplate="{StaticResource myTemplate}"
      ItemsSource="{Binding Source={StaticResource BookData}}">
    </ListBox>
  </Grid>
</Window>
```

実行すると、書籍のタイトル、サマリー、ISBN が 一組となって一覧表示されます。



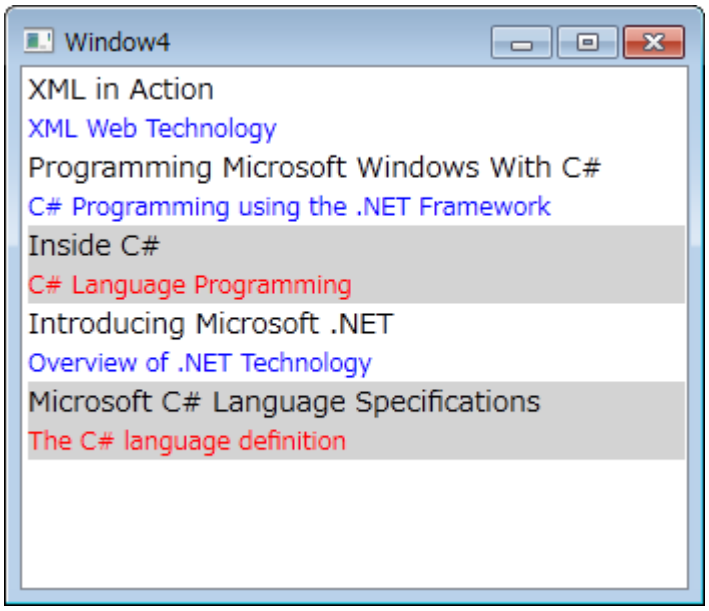
このように DataTemplate を用いることで、データ バインディングによりデータを表示する際に、表示内容をカスタマイズできるようになります。

●ItemTemplateSelector を使う

最後に ItemTemplateSelector を使い、バインドされたデータの内容に応じて、適用されるテンプレートを変更してみましょう。

上述した XML データは書籍に関する架空のデータですが、これには在庫を示す Stock 属性が含まれています。この属性には、在庫がある場合は "in" が、在庫がない場合は "out" が値として格納されています。

この属性を参照して、在庫の有無によって表示に使うテンプレートを変更します。在庫がない場合は、項目の背景を薄いグレーで描画しています。



この処理には、在庫状態に対応する 2 種類のテンプレートと、どちらのテンプレートを使用するか判断を行うロジック (C# あるいは Visual Basic のコード) が必要になります。

まず 2 種類のテンプレートを用意します。在庫がない場合のテンプレートでは、項目の背景を LightGray で塗りつぶし、Summary を赤 (Red) で表示することにします。そして、それぞれのテンプレートには x:Key 属性を使って名前 (myTemplateIN と myTemplateOUT) を付けておきます。

```
  <!-- 在庫がある場合のテンプレート -->
   <DataTemplate x:Key="myTemplateIN">
      <StackPanel>
        <TextBlock FontSize="14" Text="{Binding XPath=Title}"/>
        <TextBlock Foreground="Blue" Text="{Binding XPath=Summary}"/>
      </StackPanel>
   </DataTemplate>
   <!-- 在庫がない場合のテンプレート -->
   <DataTemplate x:Key="myTemplateOUT">
      <StackPanel Background="LightGray">
        <TextBlock FontSize="14" Text="{Binding XPath=Title}"/>
        <TextBlock Foreground="Red" Text="{Binding XPath=Summary}"/>
      </StackPanel>
   </DataTemplate>
```

次に、テンプレートの選択を行うロジックの情報をリソースとして設定します。これにはまず <Window> 要素に、ロジックを記述するコードの名前空間を指定しておきます。

```
<Window x:Class="WpfApplication7.Window4"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="clr-namespace:WpfApplication"
    Title="Window4" Height="300" Width="350">
```

これにより、この XAML からは、"WpfApplication7" という名前空間にあるクラスを "local" という XML の名前空間名を使って参照できるようになります。

さらに、<Window.Resource> 要素内 で、テンプレートの選択に利用するクラス（ここでは MyTemplateSelector）に対して、XAML 内から参照できる名前（mySelector）を x:Key 属性を使って指定します。

```
  <Window.Resources>
    <local:MyTemplateSelector x:Key="mySelector"/>
```

続いて <ListBox> 要素に、選択的にテンプレートを扱うことを示す ItemTemplateSelector プロパティを指定します。このプロパティには、ロジックが記述された MyTemplateSelector クラスに対応する名前（mySelector）を指定します。

```
    <ListBox Width="300" HorizontalContentAlignment="Stretch"
      ItemsSource="{Binding Source={StaticResource BookData}}">
      ItemTemplateSelector="{StaticResource mySelector}"
    </ListBox>
```

これにより、1 レコード(XML データでは 1 要素) を読み込むたびに、MyTemplateSelector クラスのロジックが呼び出され、適切な DataTemplate が参照されるようになります。

作成した XAML ファイル全体を以下に示します。

```
<Window x:Class="WpfApplication7.Window4"
       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
       xmlns:local="clr-namespace:WpfApplication7"
       Title="Window4" Height="300" Width="350">
  <Window.Resources>
    <XmlDataProvider x:Key="BookData" Source="bookdata.xml"
XPath="Inventory/Books/*"/>
    <local:MyTemplateSelector x:Key="mySelector"/>
    <!-- 在庫がある場合のテンプレート -->
    <DataTemplate x:Key="myTemplateIN">
       <StackPanel>
         <TextBlock FontSize="14" Text="{Binding XPath=Title}"/>
         <TextBlock Foreground="Blue" Text="{Binding XPath=Summary}"/>
       </StackPanel>
    </DataTemplate>
    <!-- 在庫がない場合のテンプレート -->
    <DataTemplate x:Key="myTemplateOUT">
       <StackPanel Background="LightGray">
         <TextBlock FontSize="14" Text="{Binding XPath=Title}"/>
         <TextBlock Foreground="Red" Text="{Binding XPath=Summary}"/>
       </StackPanel>
    </DataTemplate>
```

```
  </Window.Resources>
  <Grid>
    <ListBox Width="300" HorizontalContentAlignment="Stretch"
      ItemTemplateSelector="{StaticResource mySelector}"
      ItemsSource="{Binding Source={StaticResource BookData}}">
    </ListBox>
  </Grid>
</Window>
```

●MyTemplateSelector クラス の作成

最後に、実際にテンプレートの選択を行うロジックを、ここでは C# により記述します。このコードを記述するコード ビハインド ファイルの作成については 「データ バインディング」の回で説明しましたので、ここでは省略します。

<ListBox> 要素の ItemTemplateSelector 属性に指定するクラスは、DataTemplateSelector クラスの派生クラスとして作成し、SelectTemplate というメソッドをオーバライドする必要があります。

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

namespace WpfApplication7
{
  /// <summary>
  /// Window4.xaml の相互作用ロジック
  /// </summary>
  public partial class Window4 : Window
  {
    public Window4()
    {
      InitializeComponent();
    }
  }

  class MyTemplateSelector : DataTemplateSelector
  {
    public override DataTemplate
          SelectTemplate( object item, DependencyObject container )
    {
      XmlElement e = item as XmlElement;
      if (e != null) {
        Window window = Application.Current.MainWindow;
        if (e.Attributes["Stock"].Value == "in") {
          return window.FindResource("myTemplateIN") as DataTemplate;
        } else {
          return window.FindResource("myTemplateOUT") as DataTemplate;
        }
      }
      return null;
    }
  }
```

```
}
```

この SelectTemplate メソッドでは、データの内容を判断して、適切なテンプレート (DataTemplate) を戻り値として返します。ここでは XML データを扱っていますから、仮引数 item には、<Book> 要素の値が XmlElement 型のオブジェクト として渡されます。この要素に含まれる Stock 属性の値を参照して、どのテンプレートを使用するか判断します。

以上の作業で、データの内容に応じてテンプレートを切り替える処理を実装できました。

■まとめ

今回はテンプレートの使用について学習しました。これまでに学習したリソース、スタイルに加え、テンプレートを使用することで、ユーザー インターフェイスを柔軟に拡張することができます。

次回はユーザー インターフェイスに動きを加える「アニメーション」について学習します。