Silverlight UI フレームワーク

[Silverlight][C#]

・以下のサイトからメモ
・http://www.atmarkit.co.jp/fdotnet/vblab/uiframework_03/uiframework_03_01.html

概要

・コントロール・テンプレートは、スタイルの中で利用されることが多い
・スタイルはほぼ間違いなく、リソースとして定義される
・つまり、リソース、スタイル、コントロール・テンプレートの 3 つをまとめて使用することが多い

リソース

・WPF UI フレームワークでは、リソース・ディクショナリにオブジェクトを追加しておくことで、そのオブジェクトをリソースとして、XAML 上で簡単に再利用できる。

単純なリソースの例

・実行時に生成される、SolidColorBrush オブジェクトが 1 つで済むため、メモリを節約

個別に指定

```
<StackPanel>
    <Ellipse Fill="Blue" Height="150" />
    <Ellipse Fill="Blue" Height="150" />
</StackPanel>
```

リソース・ディクショナリを使用

```
<StackPanel>
    <StackPanel.Resources>
        <SolidColorBrush x:Key="BlueBrush" Color="Blue"/>
    </StackPanel.Resources>
    <Ellipse Fill="{StaticResource BlueBrush}" Height="150" />
    <Ellipse Fill="{StaticResource BlueBrush}" Height="150" />
</StackPanel>
```

リソースの種類

・追加先のリソース・ディクショナリの場所により、イミディエイト・リソースと、アプリケーション・リソースの 2 種類に分けられる

種類	概要	備考

イミディエイト・リソース	FrameworkElement オブジェクトの Resources プロパティ (ResourcesDictionary 型) に宣言されたリソース	FrameworkElement は UIElement の派生クラス。すべての UI 要素は、これを継承しているため、自身の Resources プロパティにリソースを持つことができる。リソースの宣言要素と、その子要素から参照できる。
アプリケーション・リソース	Application オブジェクトの Resources プロパティ (ResourceDictionary 型) に宣言されたリソース	アプリケーションの実装と管理で共通に使用される機能を提供。標準的な WPF/Silverlight のプロジェクトテンプレートでは、App.xaml ファイルとその分離コードで使用されている。通常は、<Application> 要素の Resource プロパティに宣言されたリソースが、アプリケーション・リソースとなり、どこかれでも参照できる

リソース・キー

・リソース・ディクショナリの各リソースはキーによって識別されるため、x:Key 属性を使用して、各リソースに対し、リソース・キーを割り当てておく必要がある。
・通常文字列が使用されるが、一部機能においてはオブジェクトが使用されることがある。

静的リソース参照

・多くの場合、リソースの参照には、StaticResource マークアップ拡張機能を使用した、静的リソース参照が使用される。

具体例

```
<Ellipse Fill="{StaticResource BlueBrush}"/>
```

WPF では、StaticResourceExtension クラスのパラメータに、BlueBrush というキーを指定するのと同意となるが、Silverlight では、XAML 専用となっており、StaticResourceExtension クラスは存在しない

■リソース検索

・リソースの検索順
1. イミディエイト・リソース
    1. リソース参照元
    2. 親要素
2. アプリケーション・リソース

スタイル

・http://msdn.microsoft.com/ja-jp/library/ms603146(v=vs.95)
・UI 要素のプロパティ値を一括管理するための機能

スタイルの機能

・FrameworkElement クラスがもつ Style プロパティに Style オブジェクトを設定するという方法で使用する。
・Resources プロパティと同様、ほぼすべての UI 要素で使用できる

プロパティ属性構文

```
<Ellipse Fill="Blue" Height="80" Width="160"/>
```

スタイルを使って書き換え

・同じプロパティがある場合、最後に宣言されたものが有効
・スタイルを利用するより、直接設定したプロパティが優先

```
<Ellipse>
    <Ellipse.Style>
        <Style TargetType="Ellipse">
            <Setter Property="Fill" Value="Blue" />
            <Setter Property="Height" Value="80" />
            <Setter Property="Width" Value="160" />
        </Style>
    </Ellipse.Style>
</Ellipse>
```

スタイルの共有

・リソースとしてリソース・ディクショナリに定義し、複数の UI 要素で共有し、初めて意味を持つ機能となる。

StackPanel のリソースに登録し、共有する例

```
<StackPanel>
    <StackPanel.Resources>
        <Style x:Key="EllipseStyle" TargetType="Ellipse">
            <Setter Property="Fill" Value="Blue" />
            <Setter Property="Height" Value="80" />
            <Setter Property="Width" Value="160" />
        </Style>
    </StackPanel.Resources>
    <Ellipse Style="{StaticResource EllipseStyle}" />
    <Ellipse Style="{StaticResource EllipseStyle}" />
</StackPanel>
```

派生元のクラスが共通で、そのクラスに存在するプロパティに限定されれば、異なるクラス感でも同じスタイルを共有できる。

スタイルの継承

・スタイルは継承元となるスタイルを、BasedOn プロパティに設定することで継承可能。

```
<StackPanel>
    <StackPanel.Resources>
        <Style x:Key="EllipseBaseStyle" TargetType="Ellipse">
```

```
            <Setter Property="Fill" Value="Blue" />
            <Setter Property="Height" Value="80" />
            <Setter Property="Width" Value="160" />
        </Style>
        <Style x:Key="EllipseStyle" TargetType="Ellipse"
               BasedOn="{StaticResource EllipseBaseStyle}">
            <Setter Property="Stroke" Value="Yellow"/>
            <Setter Property="StrokeThickness" Value="10"/>
        </Style>
    </StackPanel.Resources>

    <Ellipse Style="{StaticResource EllipseStyle}" />
    <Ellipse Style="{StaticResource EllipseStyle}" />
</StackPanel>
```

暗黙的にスタイルを適用

- x:Key を設定せずに TargetType プロパティに 型に設定すると、x:Key が暗黙的に設定されます。
- x:Key 値を指定すると、Style がすべての要素に自動的には適用されなくなります。

```
<StackPanel>
    <StackPanel.Resources>
        <Style TargetType="Ellipse">
            <Setter Property="Fill" Value="Blue" />
            <Setter Property="Height" Value="80" />
            <Setter Property="Width" Value="160" />
        </Style>
    </StackPanel.Resources>
    <Ellipse />
    <Ellipse />
</StackPanel>
```

コントロール・テンプレート

- Win32 でのオーナードローに替わるもの
- 見た目の変更もプロパティの設定だけで比較的簡単に変更できる様になっている
- Template プロパティを利用しビジュアル構造を再定義できる

コントロール・テンプレートの内容を確認する手段

- MSDN ライブラリを参照
  - WPF テーマ
- Silverlight「コントロールのスタイルとテンプレート」
  |http://msdn.microsoft.com/ja-jp/library/cc278075(VS.95).aspx]
- Expression Blend の［コントロールのパーツ（テンプレート）の編集］－［コピーの編集］機能を利用
- XamlWriter クラスの Save メソッドを使用し、実行時にコードから Template プロパティの内容をシリアライズ (WPF)

TemplateBinding

- コントロールのプロパティと、コントロール・テンプレート内の要素のプロパティとをバインドする必要がある

角丸ボタンを作成する例

```
<StackPanel>
    <StackPanel.Resources>
        <ControlTemplate x:Key="ButtonTemplate" TargetType="Button">
            <Border Background="{TemplateBinding Background}" CornerRadius="30" Padding="10">
                <TextBlock Text="{TemplateBinding Content}"
```

```
                            Foreground="White"
                            VerticalAlignment="Center" HorizontalAlignment="Center"/>
                </Border>
            </ControlTemplate>
        </StackPanel.Resources>
        <Button Content="Test1" Background="LightBlue"
                Template="{StaticResource ButtonTemplate}" Margin="30" />
        <Button Content="Test2" Background="Blue"
                Template="{StaticResource ButtonTemplate}" Margin="30" />
    </StackPanel>
```

■ ContentPresenter 要素

・Button コントロールは文字列に限らず、画像や UI 要素といったさまざまなオブジェクトをコンテンツとして表示できる ContentControl タイプのコントロール
・Button コントロールの Content プロパティには、Image 要素や Panel 要素といった文字列以外のオブジェクトを設定することができる

Content プロパティが TextBlock の Text プロパティとバインディングしていたのでは、そのようなオブジェクトを表示させることはできない。文字列以外のコンテンツが設定された場合でも、そのコンテンツをきちんと表示させるためには、そのための専用の要素である「ContentPresenter 要素」を使用する必要がある