XAML Do-It-Yourself シリーズ

第 3 回 イベントとトリガー

Microsoft®

XAML Do-It-Yourself
第 3 回 イベントとトリガー

XML Do-It-Yourself 第 3 回目は、イベント処理とトリガーについて学習します。Windows フォーム アプリケーションでは、たとえば「ボタンが押された」というイベントに対応する処理 (イベント ハンドラー) を記述することで、アプリケーションのユーザー インターフェイスを実現していました。XAML を用いる WPF アプリケーションでも同様に、ユーザーの操作などによって発生するイベントの処理を、C# や Visual Basic などのプログラミング言語で記述します (本書では、プログラミング言語に C# を用いています)。

今回は主にイベントについて、次のことを学習します。

- XAML ファイルとクラス (ソース ファイル) の対応付け
- イベントの設定とイベント ハンドラーの記述
- イベント ルーティングの利用
- トリガーによるイベント処理

■イベント

XAML により記述したコントロールで発生するイベントをプログラミング言語で処理するには、まず XAML ファイルと、それに対応するイベント ハンドラーが記述されたクラス (ソース ファイル) を対応付けておく必要があります。このようなソース ファイルは、ASP.NET と同様、コード ビハインド ファイルと呼ばれます。

●XAML ファイルとコード ビハインド ファイルをマッピング

ここではまず、イベント ハンドラーが記述されたクラスを作成します。そして、XAML のルート要素 (<Window> 要素) に、XAML の名前空間を表すプレフィックス ("x:") を付けた x:Class 属性を記述し、属性の値として、そのクラス名 (名前空間を含む) を記述します。

例えば、作成中の XAML ファイルに対応する MainWindow.xaml.cs というファイルを作成し、次のような名前空間とクラスの定義を行います。Visual Studio 2010 を使って WPF アプリケーションのテンプレートからプロジェクトを作成した場合は、イベント ハンドラーを記述するソース ファイル (MainWindow.xaml.cs など) はすでに作成されています。

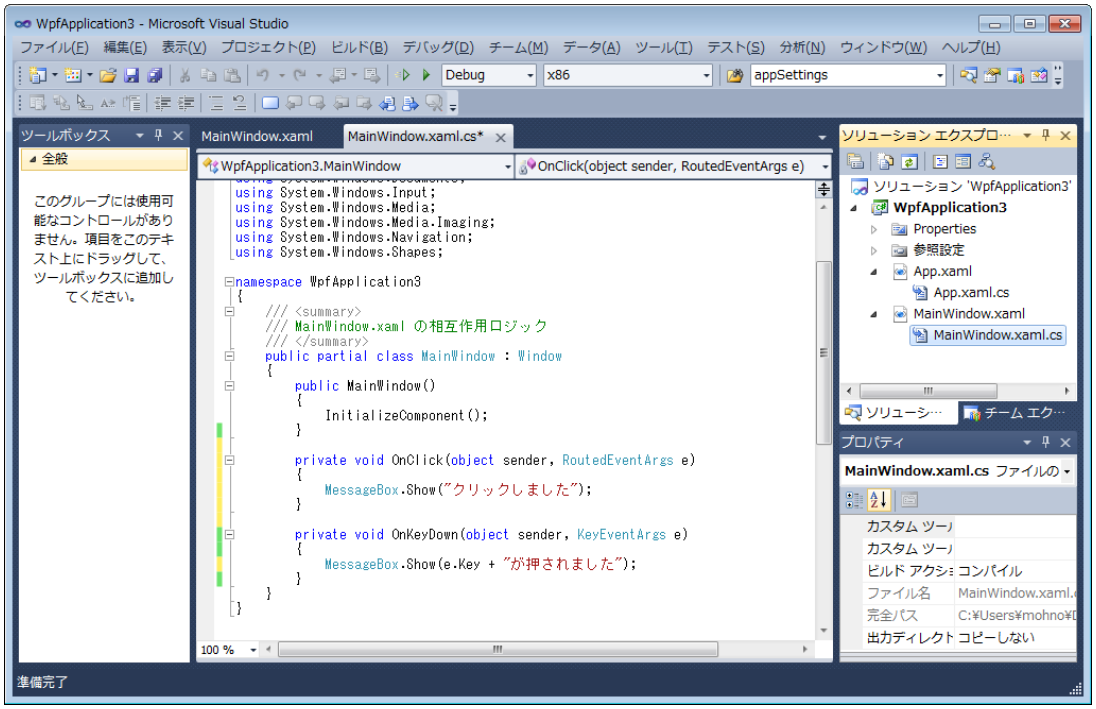
```
namespace WpfApplication
{
  public partial class MainWindow : Window
  {
  }
}
```

続いて、対応する XAML ファイルの <Window> 要素に、x:Class 属性を追加します。属性の値には、名前空間を含む C# のクラス名を記述します。

```
<Window x:Class="WpfApplication.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525">
```

これは、<Window> 要素で定義するウィンドウが、"WpfApplication.MainWindow" というクラスに対応することを示しています。ファイル名はここには記述しませんが、XAML ファイルを上記のクラスが定義されたソース ファイルとともにビルドすることで、これらが対応付けされます。ビルド時に x:Class 属性が示すクラスが存在しない場合はビルド エラーとなります。

Visual Studio 2010 で WPF アプリケーション プロジェクトを作成した場合は、x:Class 属性はすでに作成されています。次の画面は、Visual Studio 2010 でのコード ビハインド ファイルを編集しているところです。ソリューション エクスプローラーには、XAML ファイルに対応するコード ビハインド ファイルがツリー状に表示されます。



■イベント ハンドラーの指定

次に、イベント処理を行いたいコントロールで、イベント ハンドラーの指定を行います。イベント ハンドラーの指定は、イベント名を表す属性を記述し、その値にイベント ハンドラー名 (メソッド名) を記述します。

たとえば、ボタンがクリックされた際にイベント ハンドラーである OnClick メソッドが実行されるようにするには、次のように Click 属性を記述します。

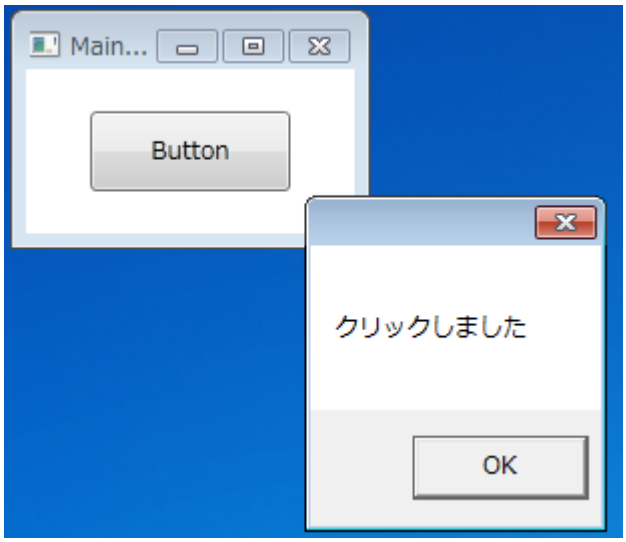
```
<Window x:Class="WpfApplication.MainWindow"
       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
       Title="MainWindow" Height="120" Width="180">
  <Grid>
    <Button Width="100" Height="40" Name="button1"
            Click="OnClick">Button</Button>
  </Grid>
</Window>
```

これにより、ボタンがクリックされると、<Window> 要素の x:Class 属性で指定したクラスにある OnClick メソッドが呼び出されます。

一方、イベント ハンドラーである OnClick メソッドは、次のように記述します。

```
private void OnClick(object sender, RoutedEventArgs e)
{
  MessageBox.Show("クリックしました");
}
```

これにより、ボタンが押されるとメッセージ ボックスが表示されます。



●イベント ハンドラーのパラメーター

イベント ハンドラーは、通常 2 つのパラメーターをとります。1 つは object 型のパラメーターで、もう 1 つは RoutedEventArgs 型のパラメーターです。

```
private void OnClick(object sender, RoutedEventArgs e)
{
  MessageBox.Show("クリックしました");
```

```
}
```

sender は、イベント ハンドラーを呼び出したオブジェクトを示します。この例では sender は Button コントロールの button1 です。

2 番目のパラメーターの RoutedEventArgs オブジェクトには、後述するイベントのルーティングを含めたイベント情報が含まれています。これには、イベントが発生した大本のオブジェクト、コントロールがネストしている場合にイベントの伝搬を続けるかどうかを指定するプロパティなどが含まれます。

なお、2 番目のパラメーターの型はイベントの種類によって変化します。たとえばキーが押された場合のイベント ハンドラーでは、2 番目のパラメーターは、押されたキーの情報を含んだ KeyEventArgs 型のオブジェクトとなります。

```
<Button KeyDown="OnKeyDown">Button</Button>
```

この場合のイベント ハンドラーを、次のように定義します。

```
private void OnKeyDown(object sender, KeyEventArgs e)
{
  MessageBox.Show(e.Key + "が押されました");
}
```

コントロールで発生するイベントの種類や、イベント ハンドラーのパラメーターとして渡されるオブジェクトを確認するには、MSDN ライブラリを参照してください。

■イベントのルーティング

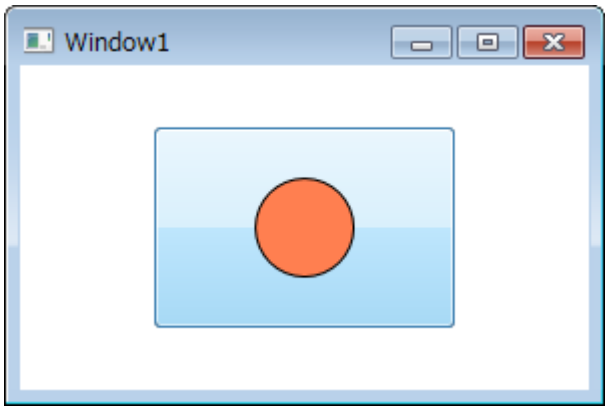
XAML の構造について説明した際に、XAML では要素（コントロール）を階層的に記述することを説明しました。XAML では、この階層内のいずれかのコントロールでイベントが発生すると、その階層のすべて

のコントロールにイベントが伝搬します。これは、イベント ルーティングと呼ばれる機能です。

例えば以下のようなボタンを用意します。

```
<Grid>
  <Button Name="button1" Width="150" Height="100" >
    <Ellipse Width="50" Name="ellipse1" Stroke="Black" Height="50" Fill="Coral" />
  </Button>
</Grid>
```

ここでは、Grid の中に Button を配置し、さらにボタンの文字列の代わりにオレンジ色の円 (Ellipse) を描画しています。



ここで、円の内側をマウスでクリックすると、Ellipse、Button、Grid の順番に Click イベントが発生します。そのため、円がクリックされただけでも、Button の Click イベントのイベント ハンドラーは Click イベントを捉えることができます。

さらに Button ではなく Grid で Click イベントに対するイベント ハンドラーを用意しておき、この中でイベントが発生したオブジェクトを特定することで、ボタンがクリックされたことを判断することも可能です。

```
<Grid Button.Click="OnPanelClicked">
  <Button Name="button1" Width="150" Height="100" >
    <Ellipse Width="50" Name="ellipse1" Stroke="Black" Height="50" Fill="Coral" />
  </Button>
</Grid>
```

```
public partial class MainWindow : Window
{
  public MainWindow()
  {
    InitializeComponent();
  }

  private void OnPanelClicked(object sender, RoutedEventArgs e)
```

```
  {
    // イベントが発生したコントロールの名前を表示
    MessageBox.Show(((Button)e.Source).Name + "が押されました");
  }
}
```

●複数のイベント処理を 1 カ所でまとめて処理する

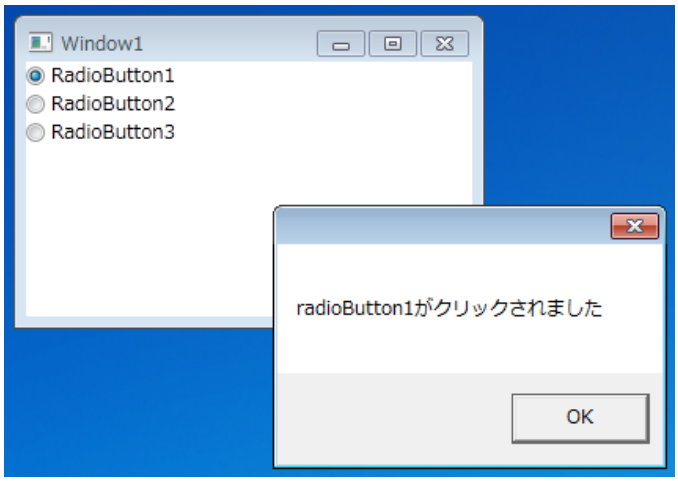
上記の仕組みを利用して、複数のコントロールで発生するイベントを 1 つのイベント ハンドラーにまとめることもできます。

次の例では、StackPanel にラジオボタン (RadioButton) を 3 つ配置し、すべての RadioButton のクリックを StackPanel で処理しています。

```
<StackPanel RadioButton.Click="OnClicked">
  <RadioButton Name="radioButton1" >RadioButton1</RadioButton>
  <RadioButton Name="radioButton2" >RadioButton2</RadioButton>
  <RadioButton Name="radioButton3" >RadioButton3</RadioButton>
</StackPanel>
```

StackPanel に設定したイベント ハンドラーでは、イベントの発生元 (e.Source) の Name プロパティを参照することにより、どの RadioButton が押されたのかを判別します。

```
public void OnClicked(object sender, RoutedEventArgs e)
{
  RadioButton btn = (RadioButton)e.Source;
  MessageBox.Show(btn.Name + "がクリックされました");
}
```



■コマンドを利用する

イベント ハンドラーを用いてイベント処理を実装する方法に加えて、コマンド (Command) と呼ばれる

機能を用いてイベント処理を行うことも可能です。WPF ではファイルのオープンやクローズ、コピー、ペーストといった汎用的な処理がコマンドとして用意されています。これらのコマンドに対応したコントロールでは、コマンドを利用することで、プログラミングの手間を省くことができます。

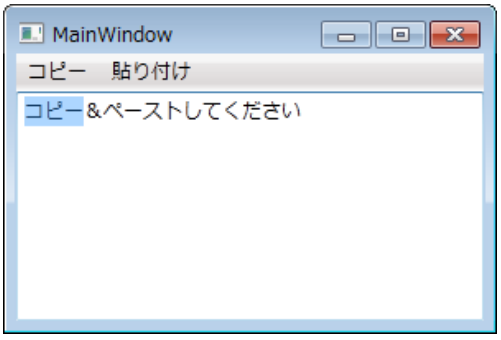
例として、クリップボードのコピーおよびペースト (貼り付け) 処理を挙げます。メニュー項目に [コピー] と [ペースト] の項目を作成します。その際にイベント ハンドラーを指定するのではなく、"Command" 属性を指定します。

コマンドの内容は、コピーのメニュー項目には "ApplicationCommands.Copy" を、ペーストのメニュー項目には "ApplicationCommands.Paste" を指定します。さらに <TextBox> 要素を配置して文字列の入力を可能にしておきます。

```
<Window x:Class="WpfApplication.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">
  <DockPanel>
    <Menu DockPanel.Dock="Top">
      <MenuItem Command="ApplicationCommands.Copy"/>
      <MenuItem Command="ApplicationCommands.Paste"/>
    </Menu>
    <TextBox>
      コピー＆ペーストしてください
    </TextBox>
  </DockPanel>
</Window>
```

リスト 3-2 コマンドを使ったメニュー

これを実行して、ウィンドウに表示されている文字列を選択した後、メニューから [コピー] そして [ペースト] を選択すると、コピーした文字列がテキストボックスに追加されるのを確認できます。



このようにコマンドを利用することで、汎用的な処理を WPF のフレームワークに任せてしまうこともできます。

■トリガー

今回説明したとおり、イベントは、ユーザーの操作やアプリケーションの状態の変化を受けて、プログラミング言語で記述されたロジックを実行するためのものです。XAML ではこれに似た機能として「トリガー」があります。トリガーを用いると、コードを記述することなく、イベントの発生時にアプリケーションの表示内容を変更できます。

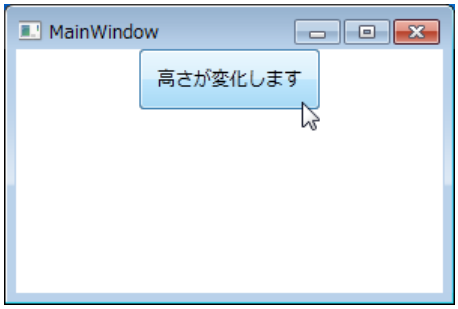
トリガーは、テンプレートまたはスタイルでのみ使用できる機能です。テンプレートおよびスタイルについては、まだ紹介していませんので、ここではトリガーによって実現できる一例を紹介するにとどめます。詳細は「スタイル」の回で説明します。

例として、ボタンの上にマウスが移動した際に、ボタンの高さを変える記述を示します。

```
<Window x:Class="WpfApplication.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">
  <Window.Resources>
    <Style  TargetType="Button">
      <Style.Triggers>
        <Trigger Property="IsMouseOver" Value="true">
          <Trigger.Setters>
            <Setter Property = "Height" Value="40"/>
          </Trigger.Setters>
        </Trigger>
      </Style.Triggers>
    </Style>
  </Window.Resources>
  <StackPanel>
    <Button Width="120">高さが変化します</Button>
  </StackPanel>
</Window>
```

リスト 3-3 トリガーを用いたイベント処理

実行して、マウスをボタンの上に移動すると、ボタンの高さが変化するのを確認できます。



■まとめ

今回はイベントについて紹介しました。ウィンドウにコントロールを配置し、プログラミング言語でイベント処理を記述するスタイルは、これまでのプログラミングとそれほど変わらないため、容易に理解できたのではないでしょうか。

次回はデータバインディングを用いて、XML ファイルやデータベースの内容をコントロールに表示する方法を学習します。