わさわさとWPF

ソースコードと要点で学ぶWPFの活用

わさわさとは

- WPFを使用して開発されたWassr（和製ミニブログ）クライアント
- WPFの表現によりデスクトップアプリながらHTMLに引けを取らない表示を実現
- VS2008のみで開発したためコントロールのデザイン等は弄っていない
- 動作にはWindows XP以降が必要

http://hcm.wankuma.com/wasawasa.htm

セッションの内容

- スライドは参考書
- ソースコードは教科書の小説
- WPF独特の機能を使いこなす
  - リソース
    - スタイル
  - データバインディング
    - コンバータ
    - バリデータ
  - コマンド
  - ディスパッチャ
- 裏技的なテクニックを使いこなす

対象レベル

- くまーでいうと2くらい
- C#かVB.NETの文法をある程度理解していること
- XAMLの文法をある程度理解していること
- 基本は同じなのでHTML+JSな人でも大丈夫だと思います。

- ✓StaticResourceとDynamicResource
- ✓x:Static
- ✓プロジェクトの「Resource」アイテム
- ✓マージされたリソースディクショナリ
- ✓スタイル

其の壱　リソース

StaticResourceとDynamicResource

- StaticResource
  Control.Resourcesで定義されたリソースを呼び出す。
  - リソースのキーはx:Key属性で指定する。
- DynamicResource
  ソースコードから動的に設定されたリソースを呼び出す。すなわち遅延評価される。

{StaticResource resourceKey}
<StaticResource ResourceKey="resourceKey" />

http://msdn.microsoft.com/ja-jp/library/ms750950.aspx
http://msdn.microsoft.com/ja-jp/library/ms748942.aspx

x:Static

- publicクラスのpublic staticプロパティまたはフィールドを参照できる。
- マネージドリソースを参照できるが利用できる型は限られる。
- あらかじめルート要素のxmlns:ns属性でCLR名前空間と関連づけをしておく。

```
xmlns:ns="clr-namespace:NameSpace;assembly=AssemblyName“
{x:Static ns:ClassName.PropertyName}
<x:Static Member=“ns:ClassName.PropertyName” />
```

http://msdn.microsoft.com/ja-jp/library/ms742135.aspx

プロジェクトの「Resource」アイテム

- ビルド アクションを「Resource」にするとWPFリソースとして扱われる。
- 同一アセンブリのリソースのURIは、pack://application:,,,/ResourceFile.ext
  - XAMLからは相対パスで参照できる。
- 参照アセンブリのリソースのURIはpack://application:,,,/ReferencedAssembly;component/ResourceFile.xaml

http://msdn.microsoft.com/ja-jp/library/aa970678.aspx

マージされたリソースディクショナリ

- ResourceDictionaryにはリソースディクショナリファイルをマージできる。
- リソースディクショナリファイルにEventSetter（後述）を記述することはできない。

```
<ResourceDictionary>
    <ResourceDictionary.MergedDictionaries>
    <ResourceDictionary
    Source="myresourcedictionary.xaml"/>
    </ResourceDictionary.MergedDictionaries>
</ResourceDictionary>
```

http://msdn.microsoft.com/ja-jp/library/aa350178.aspx

スタイル①

- スタイルを使うとコントロールなどの既定のプロパティを設定できる。
- スタイルは通常リソースに設置される。
- TargetTypeのみ指定するとタイプの一致する要素全て、x:Keyを併用するとStaticResourceで設定した要素のみに適用される。
- スタイルはBasedOnプロパティでx:Keyが設定されているTargetTypeの一致するプロパティを継承できる。

http://msdn.microsoft.com/ja-jp/library/ms745683.aspx

スタイル②

- スタイルにはSetter、EventSetter、Triggerを含むことができる。
- Setterにはプロパティ名と値を設定する。
  - 値にはデータバインド（後述）が使用できる。
  - DataContextは適用先のコントロールのものになる。
- EventSetterにはイベント名とイベントを設定する。
  - HandlesEventTooを設定しておかないとイベントが呼び出されないことがある。
  - リソースディクショナリファイルでは使用できない。テンプレート内で使用するとデザイナが使用できない。
- Triggerについては割愛する。

- ✓ContentControlへのデータバインド
  - ✓コンバータ
  - ✓バリデーション
- ✓ListBoxへのデータバインド
- ✓TabControlへのデータバインド

其の弐　データバインディング

ContentControlへのデータバインド①

- 何を（SourceとPath）
  DataContext/Source/RelativeSource/ElementName
  未指定だと親要素やテンプレート親からDataContextが使用される。
  Pathはそこからの相対パス。
- いつ（UpdateSourceTrigger）
  LostFocus/PropertyChanged/Explicit
- どのように（ModeとConverterとValidationRules）
  OneWay/TwoWay/OneWayToSource
  Converterは未定義の変換を行うのに使用される。
  ValidationRulesは値の確認に使用される。

http://msdn.microsoft.com/ja-jp/library/ms752347.aspx

ContentControlへのデータバインド②

- ソースになるオブジェクトの種類に制限はないが、INotifyPropertyChangedイベントが実装されていると即座に反映される。
- DataContextはコントロールに属し、親要素から継承される。ソースコードから設定しやすい。
- Sourceはバインディングに属する。つまりDataContextを上書きしない。前述したStaticResourceかx:Staticが設定できる。バインディングが依存関係オブジェクトでないためDynamicResourceは設定できない。
- RelativeSourceも然り。詳しくは後述。
- ElementNameも然り。他の要素を参照できる。チェックされていれば有効といった用途に使う。

ContentControlへのデータバインド③

- Pathは指定されたオブジェクトのプロパティ。
  XMLの場合はXPath。
  存在しないプロパティの場合は後述するコンバータで変換するか拡張メソッドを使うとよい。
- たとえばtxtMessage:TextBoxの文字列の文字数を取得するにはサンプル1のようにする。
- たとえばアプリケーション設定のColorを取得するにはサンプル2のようにする。
  （Settings.Settingsのアクセス修飾子をPublicにしておく必要がある）

1. {Binding ElementName=txtMessage, Path=Text.Length, Mode=OneWay}
2. xmlns:prop="clr-namespace AppName.Properties"
   {Binding Source={x:Static prop:Settings.Default}, Path=Color}

ContentControlへのデータバインド④

- RelativeSourceは{RelativeSource Self}のように拡張機能をネストして記述する。ここは嵌りやすいので注意。
- Selfはその要素を表す。
- TemplatedParentはテンプレート親を表す。要するにTemplateを持っている要素。
- FindAncestorはAncestorType型のAncestorLevel番目に見つかった親要素を表す。たとえばAncestorType=Window, AncestorLevel=1であれば親ウィンドウを表す。
- PreviousDataは情報不足により不明です。知っている人がいたら教えてください。m(__)m

http://msdn.microsoft.com/ja-jp/library/ms743599.aspx

ContentControlへのデータバインド⑤

- LostFocusはフォーカスを失った時点で検証とバインドを行う。TextBox.Textがこの形式。
- PropertyChangedはプロパティが変更された(ry。大部分のプロパティがこの形式。
- ExplicitはUpdateTarget/UpdateSourceが呼び出された(ry。明示的に設定する必要がある。
  設定画面などはこの形式であると都合がよいが、コントロール個別に呼び出さなければいけない。

http://msdn.microsoft.com/ja-jp/library/ms754356.aspx

ContentControlへのデータバインド⑥

- OneWayはソースからターゲットへのみバインドを行う。これは内容を編集できないコントロールで使用される。
- TwoWayは両方向のバインドを行う。これは内容を編集できるコントロールで使用される。
- OneWayToSourceはターゲットからソースへのみバインドを行う。これについては後述する。
- OneWayまたはTwoWayの場合、ソースにINotifyPropertyChangedが適切に実装されていない場合、UpdateTargetを実行するまで表示が変化しない。
  ソースとして使う場合INotifyChangedの実装は事実上必須とも言える。

http://msdn.microsoft.com/ja-jp/library/ms744816.aspx

裏技①

- OneWayToSourceの使い所。データバインドで生成されたSelectorのSelectedIndexを取得するもっともスマートな方法。
- Selectorに対応するデータクラス（わさわさではIBundle）にSelectedIndexプロパティを作成する。

1. ListBoxのSelectedIndexを
   OneWayToSourceで作成したプロパティにバインドする。

```
public int SelectedIndex{ get; set; }
SelectedIndex="{Binding Path=SelectedIndex,
    Mode=OneWayToSource}"
```

コンバータ

- バインディングに型変換はつきものであるが、フレームワークによって変換できないケースもそれなりに存在する。
- こうした場合、IValueConverterを実装したクラスを定義し、リソースに配置して使用する。
- 複数のソースを使用する場合はIMultiValueConverterを(ry、ターゲットにこれを参照したMultiBindingをバインドする。

http://msdn.microsoft.com/ja-jp/library/system.windows.data.ivalueconverter.aspx

バリデーション

- ほとんどの場合バインディングのターゲットは自由に入力可能なTextBoxである。このため、数値などにバインディングしている場合入力を制限する必要がある。
- こうした場合はValidationRuleを継承したクラスを定義し、リソースに配置し、バインディングのValidationRulesプロパティに設定して使用する。
- ValidationRuleでは値が正常値かどうかをboolで返す。
- 検証を通過できなければソースに代入されない。

http://msdn.microsoft.com/ja-jp/library/system.windows.controls.validationrule.aspx

ListBoxへのデータバインド①

- スカラではなくコレクションにバインドされる。
- アイテムはテンプレートによって配置および表示される。

ListBoxへのデータバインド②

- ItemsSourceには ObservableCollection<T>を使用すると効率がよい。
- これはAdd/Insert/Removeで変更部分のみ更新されるから。
- このため、ObservableCollectionはマルチスレッドではディスパッチャ経由で操作する必要がある（後述）。
- これはItemsControl全般に言えること。

ListBoxへのデータバインド③

- ItemTemplateにはDataTemplateを使用する。
- DataTemplateはContentControl
- ContentにはGridやStackPanelを配置し、そのChildrenにコントロールを配置する。
- コントロールへのバインドは{Path=PropertyName}だけでよい。
- これは、DataContextにより自動的に設定されるから。

裏技②

- 前ページで「DataContextが自動的に設定される」と書いたが、これは別の用途にも使用できる。
- すなわち、コントロールのイベントハンドラから参照が可能。
- ListViewの列の場合は値そのものが設定されるが、TemplatedParent.DataContextでデータクラスのインスタンスが取得可能。

```
var Data = (sender as Control).DataContext as DataClass;
```

ListBoxへのデータバインド④

- ItemsPanelを設定するとは項目の配置をカスタマイズすることができる
- ItemsPanelにはItemsPanelTemplateを使用する。
- ItemsPanelTemplateにはStackPanelやWrapPanel、Canvasなどを配置できる。
- Canvasを配置した場合はListBoxのリソースにListBoxItemのスタイルを配置し、Top、Leftなどのバインディングを設定する必要がある。

TabControlへのデータバインド①

- ItemsControlにContentTemplateプロパティが追加されている。
- ただし、何故かItemsPanelは意味を成さないため、残念ながら非常に自由度の低いコントロールとなっている。

TabControlへのデータバインド②

- ItemTemplateでタブの文字列やツールチップのテンプレートを設定する。
  - 基本的にはTextBlockを配置しTextプロパティにタブの文字列を配置する。
- ContentTemplateで内容のテンプレートを設定する。
  - データクラスの設計によってはItemsControlを入れ子にすることも可能。

- ✓コマンドとは
- ✓組み込みコマンドの使用
- ✓コマンドの定義

其の参　コマンド

コマンドとは

- GoFのCommandパターンを拡張し、ターゲットごとに異なる実装をもてるようにしたもの。
- ICommandは開く、保存などの抽象的な操作を表すインターフェイスで、通常RoutedEventが使用され、public staticフィールドとして定義される。
- CommandBindingでコマンドとターゲットの実装を紐付ける。
- ボタンなどのとの紐付けはCommandプロパティにCommandを設定するだけでよい。

http://msdn.microsoft.com/ja-jp/library/ms752308.aspx

組み込みコマンドの使用

- ApplicationCommands/ComponentCommands/MediaCommands/NavigationCommandsが用意されている。
- これらのコマンドはx:Staticなどを使用せずに指定することができる。

```
<Window.CommandBindings>
    <CommandBinding x:Name="helpCommandBinding"
    Command="ApplicationCommands.Help"
    CanExecute="helpCommandBinding_CanExecute"
    Execute="helpCommandBinding_Execute" />…
<Button Command="ApplicationCommands.Help">ヘルプ
    (_P)</Command>
```

コマンドの定義

- RoutedCommand型のpublic staticフィールドを定義する。
- 名前、所有型、ショートカットを指定できる。
- 定義はサンプル1のように行う。
- 参照はサンプル2のように行う。

```
public static RoutedCommand UpdateStatusCommand = new
    RoutedCommand("UpdateStatusCommand", typeof(MainWindow),
    new InputGestureCollection(new []{ new KeyGesture(Key.Enter,
    ModifierKeys.Control) }));
```

```
xmlns:ta=“clr-namespace AssemblyName”
    {x:Static ta:MainWindow.UpdateStatusCommand}
```

http://msdn.microsoft.com/ja-jp/library/ms771540.aspx

- ✓スレッドセーフなUI操作

其の五　ディスパッチャ

スレッドセーフなUI操作

- ディスパッチャはWinFormsのControl.Invokeに相当するもの。
- UIElement. Dispatcherを取得し、Invokeメソッドにデリゲートを渡すと適宜実行される。
- デリゲートにはActionと匿名メソッドの組み合わせを使うと便利。

http://msdn.microsoft.com/ja-jp/library/ms741870.aspx

ご静聴有り難うございました

質問タイム