WPF データ

[WPF][.Net][Silverlight][Universal Windows Platform][C#]

- http://www.atmarkit.co.jp/ait/articles/1010/08/news123.html

データの原則

.NET データモデル

- " データモデル " は、データの提供元と、利用者の間の取り決めを記述する
- .NET の登場により、データモデルは API に固有のものから、フレームワーク全体で共通したものへと変化
- WPF のすべてのデータ操作は、基盤である、.NET データモデルに基づいているため、WPF コントロールでは任意の CLR オブジェクトからデータを取得することが可能
- CLR を通じてアクセス出来るかぎり、WPF で視覚化することが可能

バインディングの多用

- バインディングがシステム全体で統合されている
  - コントロールテンプレートはテンプレートバインディングを利用
  - リソースはリソースバインディングを通じてロード
  - コントロールもデータバインディングに大きく依存するコンテンツモデルに基づいている

データソースにプロパティをバインドし、依存関係を追跡「、表示を自動的に更新するという概念は WPF のすべての部分に共通

データ変換

- バインディングが多用されるフレームワークでは、データへのアクセスは、変換可能な場合にのみ可能となる
- WPF は 2 種類の主要な変換、値変換とデータテンプレートをサポート
  - 値コンバータは、値の形式変換を行う。コンバーターは変換前後、どちらの形式でも受け取り変換可能 (データの双方向変換)
  - データテンプレートを使用すると、データを表現するためのコントロールをその場で作成できる

リソース

- 一般に、表示とデータの分離に最初に遭遇するのは、リソースを使用するとき
- すべての要素には、Resources プロパティがある
- Resources プロパティは単純なディクショナリで、リソースはキーを通じた単純な階層型参照を提供
- テーマ、スタイル、データバインディングはいずれもこのテクニックを利用している

C# でコードを記述

- 後で使用する変数を簡単に定義できる

```
public class Window1 : Window {
  public Window() {
    Title = " リソース ";
    Burush toShare = new SolidColorBrush(Colors.Yellow);
```

```
    Button b = new Button();
    b.Background = toShare;
  }
}
```

マークアップ

- 名前付きオブジェクトのリストを保持できる共通のプロパティ (Resource) を任意の子要素で使用できる
- 参照は階層的に行われ、要素の親に変数が含まれてない場合、その親、次の親へとたどる

```
<Window
  Text ="Resource"
  ...>
  <Window.Resource>
    <SolidColorBrush x:Key="toShare">Yellow</SolidColorBrush>
  </Window.Resource>
  <Button Background="{StaticResource toShare}">
  </Button>
</Window>
```

参照パス

- リソースの参照パスは多数の場所からデータを取得できる
- リソースをアプリケーションレベルで定義出来る
  - ページ、ウィンドウ、コントロール間でリソースを共有可能

リソース参照順	
要素階層	
Application.Resource	
型テーマ	
システムテーマ	

リソースは、データバインディングの特殊な形式であり、更新頻度が低く数が多いバインディングに最適化されている

バインディングの基本

- " バインディング " とは、2 つのデータポイントの同期を保つこと。
- WPF では Binding クラスがデータポイントを表します。
- バインディングを構築するには、このクラスにソース (データソース) とパス (クエリ) を渡す

TextBox オブジェクトの Text プロパティを参照するデータポイントを作成

```
Binding bind = new Binding();
bind.Source = textBox1;
bind.Path = new PropertyPath("Text");
```

同期させる 2 つ目のデータポイントが必要

```
contentControl1.SetBinding(ContentControl1.Content, bind);
```

XAML で同様のコードを記述

・マークアップで宣言されたバインディングでは、ElementName プロパティを使用してソースを指定
・下例のように、Text プロパティを FontFamily プロパティ (文字列ではない) などのまったく異なるものにバインドできる
・値変換のためのメカニズムには以下の 2 つがある
  ・TypeConver
  ・IValueConverter
・FontFamily の場合、TypeCoverter が FontFamily 型に関連付けられ、これによって変換が自動的に発生する

```
<Window ... >
  <StackPanel>
    <TextBox x:Name="textBox1" />
    <ContentControl x:Name="contentControl1"
                    Content="{Binding ElementName=textBox1, Path=Text}"
                    FontFamily="{Binding ElementName=textBox2,Path=Text}"/>
  </StackPanel>
</Window>
```

■ {x:Bind} マークアップ拡張

・https://msdn.microsoft.com/ja-jp/library/windows/apps/mt204783.aspx

Windows 10 では、{Binding} に代わり、{x:Bind} マークアップ拡張が新たに提供されています。{x:Bind} では、{Binding} の機能のいくつかが省略されていますが、{Binding} よりも短い時間および少ないメモリで動作し、より適切なデバッグをサポートしています。

・XAML の読み込み時、{x:Bind} は、バインディング オブジェクトと考えることのできるオブジェクトに変換され、このオブジェクトがデータ ソースのプロパティから値を取得します。
・{x:Bind} と {Binding} によって作成されたバインディング オブジェクトは、ほとんど機能的に同等
・{x:Bind} は、コンパイル時に生成される特定用途のコードを実行し、{Binding} は、汎用的なランタイム オブジェクト検査を実行します
・{x:Bind} バインディング (多くの場合、コンパイル済みバインドと呼ばれます) はパフォーマンスが高く、コンパイル時にバインド式を検証したり、ページの部分クラスとして生成されたコード ファイル内にブレークポイントを設定し、デバッグを行ったりできます

これらのファイルは obj フォルダー内にあり、<view name>.g.cs (C# の場合) などの名前が付けられています

■コンバーター

・バインディングに関連付けられる値コンバーターを使用して必要な値変換を実行できる。
・IValueConverter からクラスを派生させ、2 つのメソッドを実装する

```
public class HumanConverter : IValueConverter {
  public object Convert(object value, Type targetType,
                        object parameter, CultureInfo culture) {
    Human h = new Human();
    h.Name = (string)value;
    return h;
  }
  public objectConvertBack(object value, Type targetType,
                           object paramter, CultureInfo culture) {
    return ((Human)value).Name;
```

```
  }
}
```

・値コンバーターをバインディングに結び付ける

```
<ContentContorol
  Margin="5"
  FontFamily="{Biding ElementName=textBox2,Path=Text}">
  <Content.Control.Content>
    <Binding
       ElementName="textBox1"
       Path="Text">
      <Binding.Converter>
        <l:HumanConverter xmlns:l="clr-namespace:EssentialWPF"/>
      </Binding.Converter>
    </Binding>
  </Content.Control.Content>
</ContentControl>
```

■データテンプレート

・データ (DataType プロパティによって記述される) を受け取り表示ツリーを構築
・表示ツリー内で、データの各部分にバインドすることができる
・独自の型に対するテンプレートを構築し、データをプロパティにバインド

```
<DataTemplate
  xmlns:l="clr-namespace:EssentialWPF"
  DataType="{x:Type l:Human}">
  <Border Margin="5" Padding="5" BorderBrush="Black"
    BorderThickness="3" CornerRadius="5">
    <TextBlock Text="{Binding Path=Name}" />
  </Border>
</DataTemplate>
```

・データテンプレートを ContentControl に関連付ける方法はさまざま (例えばリソースを通じて)
・ContentTemplate プロパティで設定する例

```
<ContentControl
  Margin="5"
  FontFamily="{Binding ElementName=textBox2,Pat=Text}">
  <ContentControl.Content>
    <Binding .../>
  </ContentControl.Content>
  <ContentControl.ContentTemplate>
    <DataTemplate
      xmlns:l="crl-namespace:EssentialWPF"
      DataType="{x:Type l:Human}">
      ...
    </DataTemplate>
  </ContentControl.ContentTemplate>
 </ContentControl>
```

WPF では環境データコンテキストを要素に関連付けることができる

・上記例では、バインディングのデータソースが指定されていない

```
<TextBloxk Text="{Binding Path=Name}" />
```

・データテンプレートの場合、データコンテキストは、テンプレートが変換しているデータに自動的に設定される。
・任意の要素で DataContext プロパティを明示的に設定することが可能
・そのデータソースが当該の要素およびその子すべてのバインディングに使用される

CLR オブジェクトへのバインディング

・CLR オブジェクトへのデータのバインドは、プロパティおよびリスト (IEnumerable を実装する任意の型) を通じて行う
・バインディングは、ソースとターゲットの間の関係を確立する
・オブジェクトバインディングの場合、ソースに選択される項目は、プロパティパスによって決まる
・プロパティパスはドットで区切られた名前付きプロパティまたはインデックス

■オブジェクトバインディングのプロパティ名の識別子

・単純な CLR プロパティ用
・WPF の DependencyProperty ベースのプロパティ用

TextBox の Text プロパティを別の TextBox にバインド

```
<Window
  xmlns="..."
  >
  <StackPanel>
    <TextBox Name="text1">Hello</TextBox>
    <TextBox Text="{Binding ElementName=text1, Path=Text}" />
  </StackPanel>
</Window>
```

・次の例と同じ
・クラス修飾形式のプロパティ識別子を使用
・CLR リフレクションを使用してバインディング式内の Text という名前を解決する処理を行わない
  ・リフレクションを使用することによるパフォーマンスへの影響を回避
  ・添付プロパティへのバインディングが可能になる (TextBox オブジェクトの Grid.Row プロパティにバインドする場合、{Binding Element Name=text1, Path=(Grid.Row)} のようにする)

```
<Window
  xmlns="..."
  >
  <StackPanel>
    <TextBox Name="text1">Hello</TextBox>
    <TextBox Text="{Binding ElementName=text1, Path=(TextBox.Text)}"/>
  </StackPanel>
</Window>
```

■編集

・値を編集するには、値がいつ変更されたかを知る方法が必要
・いくつかのインターフェースは、変更通知をブロードキャストすることを可能にする
・バインディングシステムがデータの変更時に応答できるようにするには、データソースが変更通知を提供することが重要

変更通知をサポートするには 3 つの選択肢がある

方法	備考

INotifyPropertyChanged を実装する	.NET2.0 で導入、データバインディングのシナリオに最適化、通常変更通知シナリオにはいくぶん大げさだが、一般にデータモデルを作成する場合、賢明な選択
変更をプロパティに報告するイベントを追加する	.NET1.0 で導入。Windows Forms ASP.NET のデータバインディングでサポート
DependencyProperty ベースのプロパティを作成する	使用は比較的簡単。このプロパティをし y 法すると、sparse storage を保持したり、WPF のほかのサービスに接続したりすることが可能となる。DependencyObject からの派生が必須となってしまう

```
public class Name : INotifyPropertyChanged {
  ...
  public string First {
    get { return _first; }
    set { _first = value; NotifyChanged("First"); }
  }
  ...
  public event PropertyChangeEventHandler PropertyChanged;
  void NotifyChanged(string property) {
    if (ProeprtyChanged != null) {
      PropertyChanged(this, new PropertyChangedEventArgs(property));
    }
  }
}
```

・編集用に TextBox、表示用に TextBlock を使用する
・Name クラスが INotifyPropertyChanged をサポートするので、値が更新されるとバインディングシステムに通知が送られ、TextBlock オブジェクトが更新される
・規定では TextBox はフォーカスを失ったときにデータを更新する (UpdateSourceTrigger プロパティで変更できる)

```
:
<TextBlock Text="{Binding Path=Name.Last}"/>
<TextBlock Text="{Binding Path=Name.First}"/>
:
<TextBox Text="{Binding Path=Name.Last}"/>
<TextBox Text="{Binding Path=Name.First}"/>
```

リストの場合

・リストの場合、単なるプロパティの変更よりも複雑
・リストの項目が追加もしくは削除されたタイミングを把握するために、INotifyCollectionChanged が存在する
・INotifyCollectionChanged は次のイベントを持つ ChollectionChanged イベントを提供

```
public class NotifyCollectionChangedEventArgs : EventArgs {
  public NotifyCollecionChangedEventArgs Action { get; }
  public IList NewItems { get; }
  public int NewStartingIndex { get; }
  public IList OldItems { get; }
  public int OldStartingIndex{ get; }
}
public enum NotifyCollectionChangedAction {
  Add,Remove,Replace,Move,Reset,
}
```

・もっとも簡単な方法は、INotifyCollectionChangedINotifyCollectionChanged をサポートする ObservableCollection<T> を使用すること

```
public class Person : INotifyPropertyChanged {
  IList<Address> _addresses = new ObservableCollection<Address>();
 :
}


:
<StackPanel.Resources>
  <!-- 住所のリストを表示するためのテンプレート -->
  <DataTemplate x:Key="addressTemplate">
    <StackPanel Orientation="Horizontal">
      <TextBlock Text="{Binding Path=Zip}"/>
      <TextBlock Text="{Binding Path=Province}"/>
      <TextBlock Text="{Binding Path=City}"/>
      <TextBlock Text="{Binding Path=Street}"/>
    </StackPanel>
  </DataTemplate>
</StackPanel.Resources>
 :
<!-- 住所のリスト -->
<ListBox ItemSource="{Binding Path="Addresses}"
 ItemTemplate="{DynamicResource addresTemplate}" />
 :
```

XML へのバインディング

・WPF の XML サポートは、System.Xml 名前空間で提供される DOM を基盤としている
・任意の XmlDocument、XmlElement、XmlNode をソースとして使用することができる
・プロパティは要素の属性またはコンテンツにのみあバインドできる
・リストは任意の要素セットにバインドできる

■ XPath の基本

・WPF のバインディングは大きく XPath に依存している

```
<Media>
  <Book Author="a1" Title="t1"/>
  <Book Author="a2" Title="t2"/>
  <Book Author="a3" Title="t3"/>
  <CD Artist="a4" title="t4"/>
  <DVD Directory="d1" Title="t5">
    <Actor>A1</Actor>
    <Actor>A2</Actor>
  </DVD>
</Media>
```

・「/」 は最も一般的な演算子で目的の要素へのパスを構築できる (eg:Media/CD)
  ・ Media/Book を選択すると以下が生成される

```
<Book Author="a1" Title="t1"/>
<Book Author="a2" Title="t2"/>
<Book Author="a3" Title="t3"/>
```

・XPath によって、ノードのリストまたは単一のノードが生成されるという考え方は XML バインディングを学ぶ上で極めて重要
・XML では要素の属性の両方が XML ノードとみなされる
・XPath は実際には要素だけでなくノードを選択することによって機能する
・属性名を参照するには 「@」を演算子を使用する
  ・ Media/Book/@Title を使用すると、次の内容が返る (XmlAttributeNode 型)

```
Title="t1"
```

```
Title="t2"
Title="t3"
```

・「*」演算子を使用すると、任意の名前付きノード (属性または要素) を取得できる
・「[]」演算子を使用すると、位置または属性によってノードを選択できる (インデックス 1 ベース)
  ・ Media/Book[1] を選択すると以下が生成される

```
<Book Author="a1" Title="t1"/>
```

・属性による選択
  ・ Media/Book/[@Author="t1"]

```
<Book Author="a1" Title="t1"/>
```

XPath	説明	例
/	ルート以下を選択	/
//NAME	任意の子孫の NAME というタグにマッチ	//Book
@NAME	NAME という属性にマッチ	//Book/@Author
*	任意のタグにマッチ	//*/@Author
@*	任意の属性にマッチ	//Book/@*
NAME	NAME というタグにマッチ	/Media
[]	位置または属性によって子タグを選択	//Book[1],//Book[@Author='a1']

■ XML バインディング

バインディングを使用しない例

```
XmlDocument doc = new XmlDocument();
doc.LoadXml(@"
<Media xmlns=''>
  <Book Author='a1' Title='t1'/>
  <CD Artist='a1' Title='t2'/>
  <DVD Director='d1' Title='t3'>
    <Actor>A1</Actor>
  </DVD>
 </Media>");
ListBox list = new ListBox();
list.ItemSource = doc.SelectNodes("/Media/Book/@Title");
```

バインドを使用する

・バインディングを使用すると変更を追跡できる
・XmlDataProvider オブジェクトを更新、変更すると ListBox が自動的に更新される

```
XmlDocument doc = new XmlDocument();
doc.LoadXml(@"
<Media xmlns=''>
  <Book Author='a1' Title='t1'/>
  <CD Artist='a1' Title='t2'/>
```

```
  <DVD Director='d1' Title='t3'>
    <Actor>A1</Actor>
  </DVD>
 </Media>");
XmlDataProvider dataSource = new XmlDataProvider();
dataSource.Document = doc;
Binding bind = new Binding();
bind.Source = dataSource;
bind.XPath = "/Media/Book/@Title";
ListBox list = new ListBox();
list.SetBinding(ListBox.ItemSourceProperty, bind);
```

マークアップ

- XmlDataProvider
  - XmlDocument オブジェクトを構築して XPath を適用するためのマークアップフレンドリーな方法
  - データソースに対してフィルタリングを直接実行できる
  - XML データをデータソースに移動するための一般的な方法
  - 多くの場合、XmlDataProvider を使用せずに、XmlDocument、XmlElement オブジェクトをバインディングソースとして直接使用できる

```
<Window
  :>
  <Window.Resource>
   <XmlDataProvider x:Key="dataSource">
      <x:XData>
        <Media xmlns=''>
          <Book Author='a1' Title='t1'/>
          <CD Artist='a1' Title='t2'/>
          <DVD Director='d1' Title='t3'>
            <Actor>A1</Actor>
          </DVD>
        </Media>
      </x:XData>
    </XmlDataProvider>
  </Window.Resource>
 :
<ListBox ItemSource = "{Binding XPath=/Media/Book/@Title}, Source={StaticResource dataSource}"/>
```

データソースを動的に判断する

- データソースを (動的リソース参照を使用、またはデータソースを判断するためのバインディングを通じて) 動的に判断する必要がある場合は、DataContext プロパティを使用できる

```
<Window
  :
 DatraContext="{DynamicResource dataSource}">
  <Window.Resource>
   <XmlDataProvider x:Key="dataSource">
      <x:XData>
        <Media xmlns=''>
          <Book Author='a1' Title='t1'/>
          <CD Artist='a1' Title='t2'/>
          <DVD Director='d1' Title='t3'>
            <Actor>A1</Actor>
          </DVD>
        </Media>
      </x:XData>
    </XmlDataProvider>
  </Window.Resource>
 :
<ListBox ItemSource = "{Binding XPath=/Media/Book/@Title}" />
```

データテンプレート

・データテンプレートを使用するとデータの表示方法を定義できる