WPF レイアウト

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

サンプル

・WPF レイアウト サンプル

レイアウトの原則

・レイアウトを他のレイアウトにネストできる
・子コントロールは親レイアウトと「コントラクト」でやり取りすることで、任意のレイアウトが任意のコントロールをホスト可能

レイアウトコントラクト

・レイアウトと子コントロールがやり取りする方法

■コンテンツに合わせたサイズ設定

・各コントロールはコンテンツに合わせてサイズを決定する必要がある

■ 2 段階レイアウト

・コントロールが必要とするサイズは、測定と配置の 2 段階で判断する

測定段階

・各要素に対して、サイズを判断するよう要求

配置段階

・各要素の親が各要素に対し、特定のサイズおよび位置に自らを設定するように指示

サイズの協議

・3 つのサイズについて協議
1. 使用可能なサイズ
2. 目的のサイズ
3. 実際のサイズ

```
desiredSize <= actualSize <= availableSize
```

■クラス階層

・レイアウトは UIElement 型によって、WPF のクラス階層に導入される

```
public class UIElement : Visual {
  ...
  public bool Visibility Visibility { get; set; }
```

```
    public void Arrange(Rect finalRect);
    protecterd virtual void ArrangeCore(Rect finalRect);
    public void Measure(Size availableSize);
    protected virtual Size MeasureCore(Size availableSize);
}
```

一貫したレイアウト

・これらのパターンを一貫して実装するレイアウトを簡単に作成するために、レイアウトコントラクトの上位に位置するレイヤーを備える
・共通レイアウトを制御するプロパティは、System.Windows.FrameworkElement で定義される

レイアウト制約

・レイアウト制約を制御するために、FrameworkElement は 6 つのプロパティを定義

プロパティ	備考
Height	通常直接指定しない (コンテンツに合わせたサイズ設定が無効となる)
MaxHeight	
MinHeight	
ActualHeight	出力プロパティ。最終的なサイズを出力。レイアウトが完了するまで無効
Width	通常直接指定しない (コンテンツに合わせたサイズ設定が無効となる)
MaxWidth	
MinWidth	
ActualWidth	出力プロパティ。最終的なサイズを出力。レイアウトが完了するまで無効

スロットモデル

・親は「スロット」を設定し、子は「スロット」内の任意の部分を自由に占有できる
・この機能は以下の 3 つのプロパティで公開される

プロパティ	備考
Margin	子がスロットの内側に緩衝スペースを配置
HorizontalAlignment	子が残りのスペースをどのように占有するか
VerticalAlignment	子が残りのスペースをどのように占有するか

変換

・コントロールの最終的な位置を決定するために 2 つのプロパティを使用して任意の変換を行うことができる
・System.Windows.Media.Transform 型

プロパティ	備考

RenderTransform	レンダリング直前に適用 (レンダリングのみに影響を与える)
LayoutTransform	レイアウトの前に適用 (レイアウトに影響を与える)

■一般的なサブクラス

ScaleTransform

・x/y ズーム

RotateTransform

・点を中心とした回転

TranslateTransform

・配置をオフセットによって変更

TransformGroup

・上記を組み合わせる

例

```
<StackPanel Background="..." Orientagion="Horizontal">
  <Buttojn Width="75" Height="25">
    <Button.RenderTransform>
      <RotateTransform Angle="15" />
    </Button>
  </Button>
</StackPanel>
```

■ z インデックス

・重なりあったコントロールに Panel.ZIndex でレイア順序を指定

レイアウトライブラリ

・一連のレイアウトパネルを提供

Canvas

・もっともシンプル
・子要素を任意のオフセットに配置
・いかなるサイズ制約も適用しない

StackPanel

・子要素を重ねる
・Orientagion プロパティで垂直方向に重ねるか、水平方向に重ねるか制御
・それぞれの子スロットには方向に応じてコントロールの幅全体または高さ全体が与えられる
・子の最大サイズに応じて、自らの好ましいサイズを判断

DockPanel

・StackPanel によく似ているが、同じレイアウトコンテナ内で異なる縁から要素を重ねることができる。
・LastChildFill プロパティを使用すると、最後の子が残りの領域をすべて占有するようにできる

WrapPanel

・DockPanel が複数の縁を持つ StackPanel だとすると、WrapPanel は折り返しをサポートする StackPanel といえる
・利用可能な領域に合わせて要素を配置。スペースがなくなったら折り返す

UniformGrid

・System.Windows.Controls.Primitives に含まれる
・基本的なグリッドレイアウト
・各セルが同じサイズ
・要素の位置は子コレクションの順序によって決まる

Grid

・複数のセルにまたがる項目をもつグリッド
・行および列の間隔設定が一様ではないグリッド
・最も簡単な使い方は、RowDefinitions プロパティと ColumnDefinitions プロパティを設定し、添付プロパティの Grid.Row、Grid.Column を使用してどの子がどのスロットに入るかを指定する

■グリッドの概念

レイアウトと構造の分離

・通常はネストしたレイアウトパネルの数が少なくても複雑な結果を実現できる
・Grid の行、列情報はプロパティ値に基づいているため、要素の順序を入れ替えることなくレイアウトを大幅に変更することができる
・要素の順序がレイアウトの順序に依存していないため、重なりの制御が容易
・プログラムモデルに影響を与えずに、レイアウトを定義できる

柔軟なサイズ設定モデル

・パーセントによるサイズ設定
  ・幅、高さを スター (*) 谷を使用して指定できる
  ・コンテンツに合わせた、または絶対値によるサイズ設定が完了した後に、グリッド内のスペースをパーセントに応じて占有する
・例

```
<Grid>
  <Grid.RowDefinitions>
    <RowDefinition Height="2*" />
    <RowDefinition Height="1*" />
  </Grid.RowDefinitions>
  <Grid.ColoumDefinitions>
    <ColumnDefinition Width="2*" />
    <ColumnDefinition Width="1*" />
  </Grid.ColoumDefinitions>
    :
</Grid>
```

共有サイズ情報

・グリッドは同じ列内に配置されるすべてのコントロール間でサイズ情報を共有する
・有効にするには、上の列または行で、SharedSizeGroup プロパティを設定、次にコントロールで IsSharedSizeScope プロパティを設定
・例

```
<Grid IsSharedSizeScope="true">
  :
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="Auto" SharedSizeGroup="a" />
    <ColumnDefinition Width="Auto" SharedSizeGroup="a" />
  </Grid.ColumnDefinitions>
  :
</Grid>
```

グリッドのレイアウト

・グリッドのレイアウトは 2 つの段階で構成される
  ・行と列の定義
  ・スロットへの子の割り当て