UWP テンプレート

[Universal Windows Platform][Visual Studio][C#][WPF]

・以下の Kindle 本を参考に自分なりにメモ
・非常にわかりやすい上に安い

MVVM プロジェクトの構成

エントリーポイント (App.xaml,App.xaml.cs)

・App.xaml.cs が重要な起動処理となる

■ OnLaunch メソッド

```
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    Frame rootFrame = Window.Current.Content as Frame;
    // ウィンドウに既にコンテンツが表示されている場合は、アプリケーションの初期化を繰り返さずに、
    // ウィンドウがアクティブであることだけを確認してください
    if (rootFrame == null)
    {
        // ナビゲーション コンテキストとして動作するフレームを作成し、最初のページに移動します
        rootFrame = new Frame();
        rootFrame.NavigationFailed += OnNavigationFailed;
        if (e.PreviousExecutionState == ApplicationExecutionState.Terminated)
        {
            //TODO: 以前中断したアプリケーションから状態を読み込みます
        }
        // フレームを現在のウィンドウに配置します
        Window.Current.Content = rootFrame;
    }
    if (rootFrame.Content == null)
    {
        // ナビゲーション スタックが復元されない場合は、最初のページに移動します。
        // このとき、必要な情報をナビゲーション パラメーターとして渡して、新しいページを
        // 構成します
        rootFrame.Navigate(typeof(MainPage), e.Arguments);
    }
    // 現在のウィンドウがアクティブであることを確認します
    Window.Current.Activate();
}
```

・アプリケーション通常起動時のエントリーポイント

Window.Current.Content

・メインウィンドウに表示されるコンテンツ
・Window.Current.Content に Frame が設定されているか確認。されていなければ、Frame を生成し設定する。

Frame.Content

・Frame.Content をチェックして何もなければ、MainPage に画面遷移

フォルダ構成

・以下のフォルダを作成
1.Commons
2.Models

3.ViewModels
4.Views

MainPage.xaml

・デフォルトで作成されている MainPage.xaml を削除
・Views フォルダに、MainPage.xaml を作成 (空白のページを追加)

INotifyPropertyChanged

・MVVM アプリを作成するためには、INotifyPropertyChanged の実装が必須となる。
・一般的には、以下のような基本クラスを作成し、実装負荷を下げる。

```
public class BindableBase : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

      protected virtual bool SetProperty<T>(ref T field, T value, [CallerMemberName] string
propertyName = null)
    {
        if (Equals(field, value))
        {
            return false;
        }
        field = value;
        this.RaisePropertyChanged(propertyName);
        return true;
    }

    protected virtual void RaisePropertyChanged([CallerMemberName] string propertyName = null)
    {
        this.PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    }
}
```

ICommand

・http://sourcechord.hatenablog.com/entry/2015/08/26/002740

```
public class RelayCommand : ICommand
{
    private readonly Action _execute;
    private readonly Func<bool> _canexecute;

    public event EventHandler CanExecuteChanged;

    public RelayCommand(Action execute) : this(execute, null)
    {
    }

    public RelayCommand(Action execute, Func<bool> canExecute)
    {
        if (execute == null)
        {
            throw new ArgumentNullException("execute");
        }
        _execute = execute;
        _canexecute = canExecute;
    }

    public bool CanExecute(object parameter)
    {
        return (_canexecute == null) ? true : _canexecute();
    }

    public void Execute(object parameter)
    {
        _execute();
    }
```

```
        public void RaiseCanExecuteChanged()
        {
            CanExecuteChanged?.Invoke(this, EventArgs.Empty);
        }
    }

    public class RelayCommand<T> : ICommand
    {
        private readonly Action<T> _execute;
        private readonly Func<T, bool> _canExecute;

        public event EventHandler CanExecuteChanged;

        public RelayCommand(Action<T> execute) : this(execute,null)
        {
        }

        public RelayCommand(Action<T> execute, Func<T,bool> canExecute)
        {
            if (execute == null)
            {
                throw new ArgumentNullException("execute");
            }
            _execute = execute;
            _canExecute = canExecute;
        }

        public bool CanExecute(object parameter)
        {
            return (_canExecute == null) ? true : _canExecute((T)parameter);
        }

        public void Execute(object parameter)
        {
            _execute((T)parameter);
        }

        public void RaiseCanExecuteChanged()
        {
            CanExecuteChanged?.Invoke(this, EventArgs.Empty);
        }
    }
```

アプリケーション構成

レイアウト

ナビゲーション

- https://msdn.microsoft.com/windows/uwp/layout/navigation-basics

レイアウト

- https://msdn.microsoft.com/windows/uwp/layout/layouts-with-xaml

既定のコントロール スタイルとテンプレート

- https://msdn.microsoft.com/ja-jp/library/windows/apps/mt299122.aspx

SplitView

- SplitView コントロールの Pane にメニューを配置し、Content 部分に Frame を配置し画面遷移を行う形が一般的

■ /Views/MainPage.xaml

・MainPage 内の Frame にアクセスする必要があるため、Frame を public で公開

```
<Page
    x:Class="WakeUpOnLan.Views.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:WakeUpOnLan.Views"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">

    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
        <SplitView>
            <SplitView.Pane>
                <!-- Menu -->
                <ListView></ListView>
            </SplitView.Pane>
            <!-- Contents -->
            <Frame x:Name="RootRrame" x:FieldModifier="public" />
        </SplitView>
    </Grid>
</Page>
```

■ /App.xaml.cs

・App.OnLaunched()

```
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    var mainPage = Window.Current.Content as MainPage;

    // ウィンドウに既にコンテンツが表示されている場合は、アプリケーションの初期化を繰り返さずに、
    // ウィンドウがアクティブであることだけを確認してください
    if (mainPage == null)
    {
        // ナビゲーション コンテキストとして動作するフレームを作成し、最初のページに移動します
        mainPage = new MainPage();

        mainPage.RootRrame.NavigationFailed += OnNavigationFailed;

        if (e.PreviousExecutionState == ApplicationExecutionState.Terminated)
        {
            //TODO: 以前中断したアプリケーションから状態を読み込みます
        }

        // フレームを現在のウィンドウに配置します
        Window.Current.Content = mainPage;
    }

    if (mainPage.RootRrame.Content == null)
    {
        // TODO: 初期画面に遷移
    }
    // 現在のウィンドウがアクティブであることを確認します
    Window.Current.Activate();


}
```

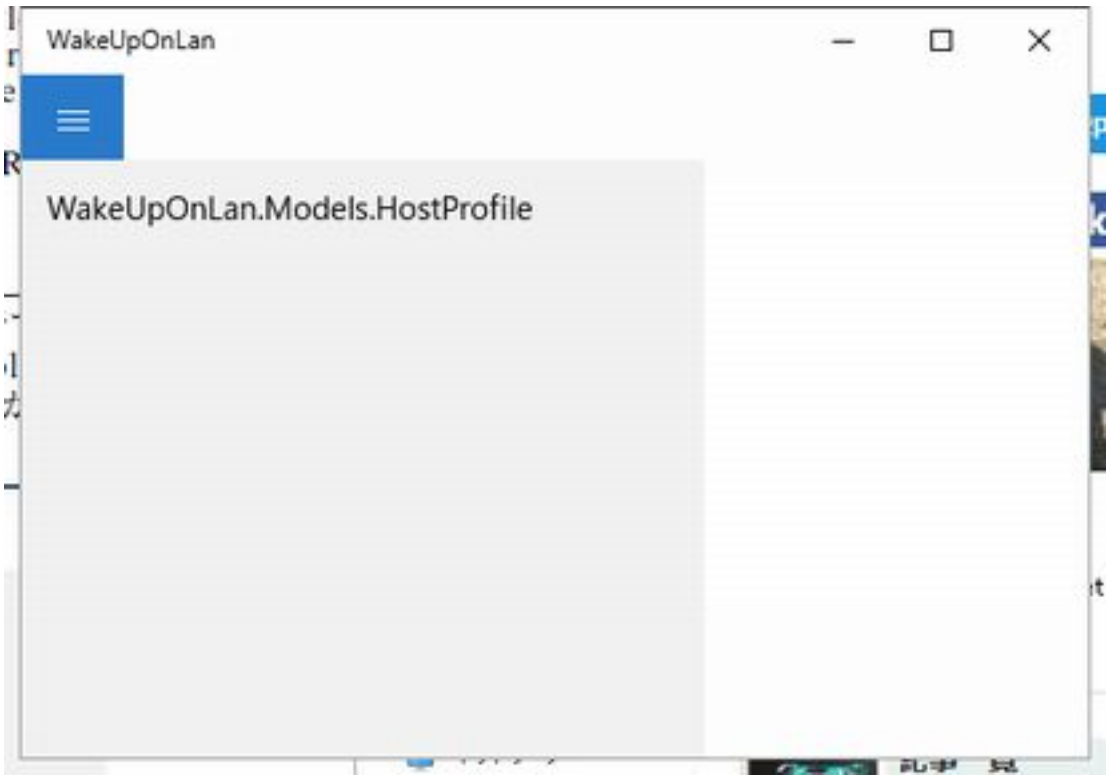
■真っ白い画面が表示されることを確認

■ハンバーガーボタン (/Views/MainPage.xaml)

・StaticResource から取得できる、SymbolThemeFontFamily の E700 に定義されている。

```
<Page
    x:Class="WakeUpOnLan.Views.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:WakeUpOnLan.Views"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">

    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition/>
        </Grid.RowDefinitions>
        <StackPanel Orientation="Horizontal">
            <ToggleButton Content="&#xE700;"
                          FontFamily="{StaticResource SymbolThemeFontFamily}"
                          IsChecked="{Binding IsPaneOpen,ElementName=SplitView, Mode=TwoWay}"
                          Width="48"
                          Height="40"/>
        </StackPanel>
        <SplitView x:Name="SplitView" Grid.Row="1">
            <SplitView.Pane>
                <!-- Menu -->
                <ListView ItemsSource="{x:Bind ViewModel.HostProfiles}">
                </ListView>
            </SplitView.Pane>
            <!-- Contents -->
            <Frame x:Name="RootRrame" x:FieldModifier="public" />
        </SplitView>
    </Grid>
</Page>
```



{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。
- WPF データ