Xamarin XAML 语言教程页面布局篇

（内部资料 v1.0）

大学霸
www.daxueba.net

前　言

Xamarin 是一个跨平台开发框架。它可以用来开发 iOS、Android、Windows Phone 和 Mac 的应用程序。使用 Xamarin 框架中的 Forms 子框架，用户可以一次性的开发多个平台的应用，如 iOS、Android、Windows Phone，从而节省大量的开发时间。

在 Xamarin.Forms 中，用户可以直接使用 XAML 语言直接进行界面设计。这样，就可以将界面和逻辑代码分离，使得应用程序的结构更加清晰。为了满足大家的开发需求，本教程着眼于 Xamarin.Forms 开发，详细讲解 XAML 语言在界面设计中的使用。同时为了方便大家理解，我们为每个知识点都配以小实例。

1.学习所需的系统和软件

- 安装 Windows 10 操作系统
- Xamarin 4.2.0.719
- 安装 OS X 10.11
- 安装 Xcode 8.0

2.学习建议

大家学习之前，可以到百度网盘（xxxxxxxxxxxxxxxxxxx）获取相关的资料和软件。如果大家在学习过程遇到问题，也可以将问题发送到邮箱 xxxxxxxxxxxxxxxxxxxx。我们尽可能给大家解决。

目　录

第 14 章　基本页面和视图

基本页面和基本视图都是在开发应用程序时最为常用的。本章将讲解有关基本页面 ContentPag、基本视图 ContentView、控件模板 ControlTemplate、模板页面 TemplatedPage 和模板视图 TemplatedView 等内容。

14.1　基本页面 ContentPage

在 Xamarin.Forms 中，每个 App 的界面都是一个页面 Page。页面的种类有很多种。其中，最常见、最为基础的页面为 ContentPage 页面，也称为内容页面。当开发者在创建一个项目后，默认就带有一个 ContentPage 页面。本节将将要有关内容页面的内容，其中包括内容页面的占用面积、内容页面的添加、派生关系以及属性等。

注意：默认的 ContentPage 页面其实是 ContentPage 派生的一个子类。

1.占用面积

ContentPage 页面占用屏幕的面积在各个平台下是有区别的。在 Android 中，页面不会占用屏幕顶部的状态栏，如果 Android 屏幕底部有按钮也不会占用，除此之外是都占用的，如图 14.1 所示。在 iOS 中会占据整个屏幕，包括顶部的状态栏，如图 14.2 所示。在 Windows Phone 中页面不会占用屏幕顶部的状态栏，除此之外是都占用的，如图 14.3 所示。

图 14.1　Android 的运行效果

图 14.2　iOS 的运行效果

图 14.3　Windows Phone 的运行效果

注意：在图 14.1~14.3 中，为了让 ContentPage 页面可以更加形象，我们为 ContentPage 页面设置了青色的背景。

2.内容页面的添加

为了方便用户添加 ContentPage 类页面，VS（Visual Studio）和 XS（Xamarin Studio）都提供专门的命令。

在 VS 创建的项目中，右击*** (可移植的) 项目，在弹出的快捷菜单中选择“添加（D）”|“新建项(w)”名，会弹出“添加新项”对话框，选择 Cross-Platform 选项。此时，我们就会看到创建 ContentPage 的两个选项，一个为 Forms Page，另一个为 Forms Xaml Page，如图 14.4 所示。

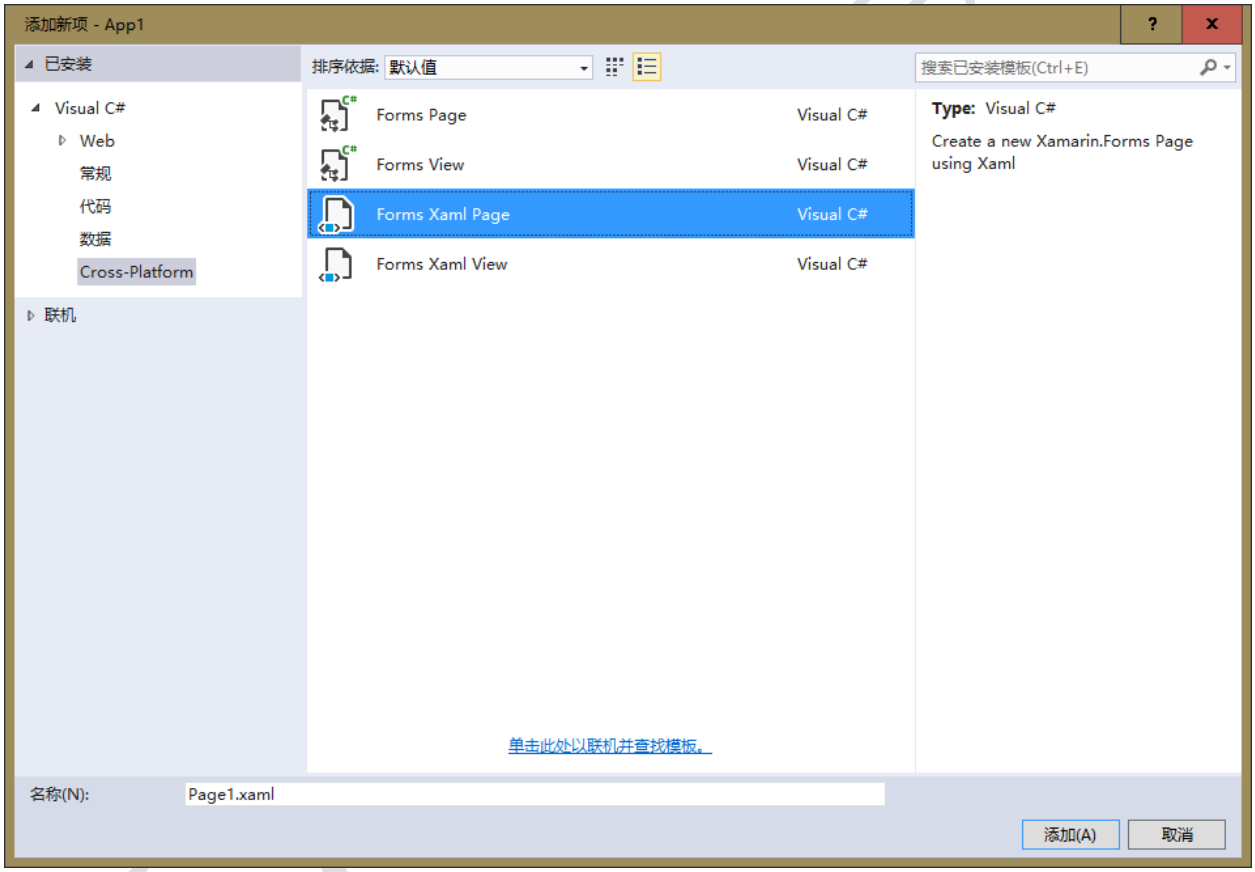


图 14.4　“添加新项”对话框

在 XS 创建的项目中，右击项目名称下方的项目名，在弹出的快捷菜单中选择 Add|New File…命令，弹出 New File 对话框，此时我们就会看到创建 ContentPage 的两个选项，一个为 Forms Page，另一个为 Forms Xaml Page，如图 14.5 所示。

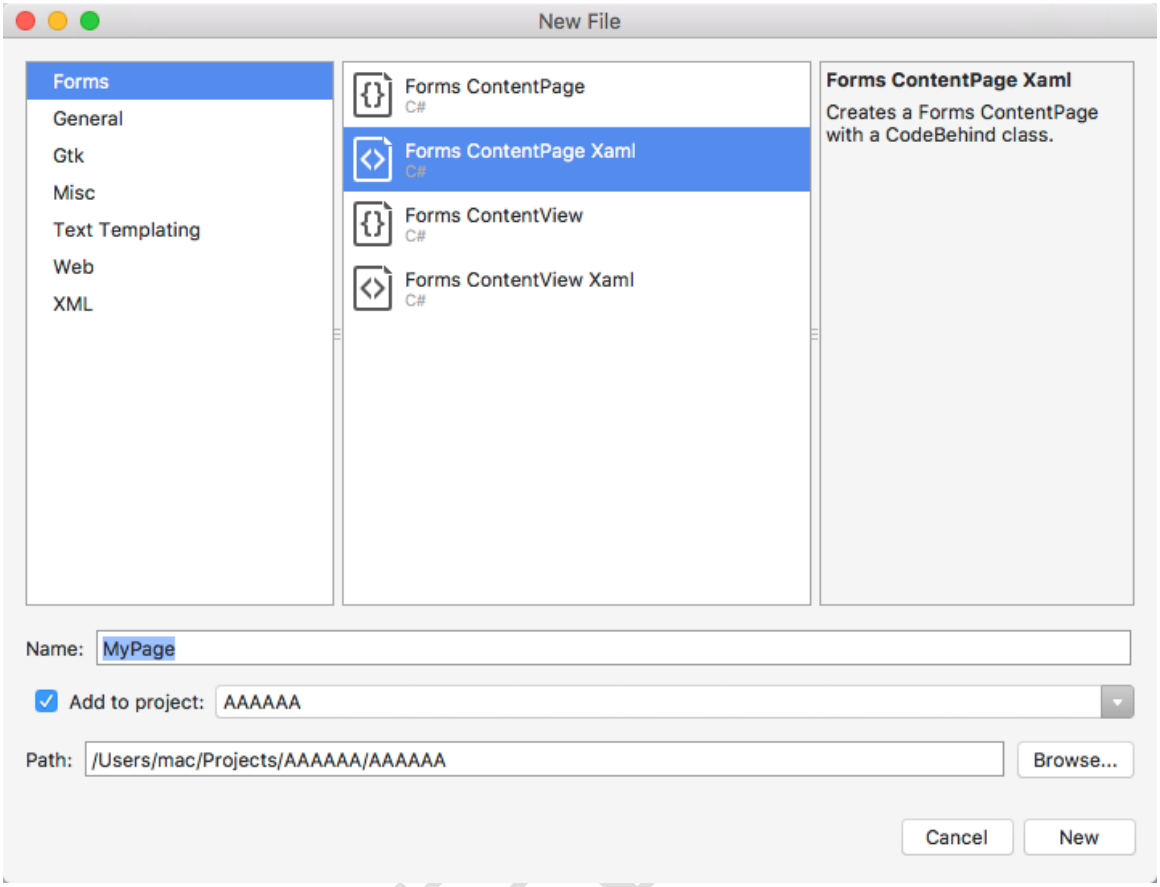


图 14.5　New File 对话框

注意：（1）使用 Forms Page 创建的 ContentPage 类是一个代码文件，而 Forms Xaml Page 创建的 ContentPage 类是一个 XAML 文件。（2）使用 Forms Xaml Page 选项在 VS 和 XS 中创建的.xaml 文件中的代码是不一样的，在 VS 中创建的代码如下：

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="App1.Page1">
  <Label Text="{Binding MainText}" VerticalOptions="Center" HorizontalOptions="Center" />
</ContentPage>
```

在 XS 中创建的代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="AAAAAA.MyPage">
    <ContentPage.Content>
    </ContentPage.Content>
</ContentPage>
```

3.内容页面的派生关系

ContentPage 页面派生自 Page，同时它又是其他页面的父类。派生关系图如图 14.6 所示。

```
Page
    TemplatedPage
        ContentPage
    NavigationPage
    MasterDetailPage
    MultiPage<T>
        TabbedPage
        CarouselPage
```

图 14.6　派生关系

注意：ContentPage 页面可以作为其他页面的一个元素使用。

4.属性

ContentPage 页面定义了一个用来设置页面内容的属性 Content 属性，开发者可以将这个属性设置为一个控件、一个视图或者是一个布局。

（1）开发者可以将 Content 属性的属性设置为按钮控件、标签控件等，如以下的代码片段：

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:App2"
             x:Class="App2.MainPage">
  <Label Text="Welcome to Xamarin Forms!"
         VerticalOptions="Center"
         HorizontalOptions="Center" />
</ContentPage>
```

在此代码中我们将 Content 属性设置为了标签控件。

注意：在属性和属性值一章中，我们提到了内容属性是可以省略的。Content 属性就是一个内容属性，所以我们在代码中将此属性进行了省略。

（2）Content 属性除了可以设置为控件外，还可以设置为一个内容视图，如以下的代码：

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:App2"
             x:Class="App2.MainPage">
  <ContentView>
    <Label Text="Accept what was and what is, and you'll have more positive energy to pursue what will be."
           VerticalOptions="Center"
           HorizontalOptions="Center" />
  </ContentView>
</ContentPage>
```

（3）ContentPage 页面的 Content 属性也可以设置为一个对象。当我们将其设置为控件或者是内容视图时，只会在页面上看到一个元素。如果开发者要在页面上出现多个元素，就需要使用到布局，在布局中可以有多个视图或者控件。如以下的代码：

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:App2"
             x:Class="App2.MainPage">
  <StackLayout Spacing="10"
```

```
                VerticalOptions="CenterAndExpand"
                  HorizontalOptions="Center">
    <Label Text="静夜思"
        FontSize="30"
        FontAttributes="Bold"
        HorizontalOptions="Center"/>
    <Label Text="床前明月光，"
            FontSize="18"/>
    <Label Text="疑是地上霜。"
            FontSize="18"/>
    <Label Text="举头望明月，"
            FontSize="18"/>
    <Label Text="低头思故乡。"
            FontSize="18"/>
  </StackLayout>
</ContentPage>
```

在此代码中，我们将布局设置为了堆栈布局，在此布局中又放置了 5 个标签控件。

14.2 基本视图 ContentView

视图是用来呈现具体内容，根据呈现内容不同，使用的视图也不同。其中，最常用的视图为 ContentView 视图，它也被称为内容视图。本节将讲解有关内容视图的内容。

1.内容视图的添加

为了方便用户添加 ContentView 视图，VS（Visual Studio）和 XS（Xamarin Studio）都提供专门的命令。

在 VS 创建的项目中，右击*** (可移植的) 项目，在弹出的快捷菜单中选择“添加（D）”|“新建项(w)”名，会弹出“添加新项”对话框，选择 Cross-Platform 选项，此时我们就会看到创建 ContentView 的两个选项，一个为 Forms View，另一个为 Forms Xaml View，如图 14.4 所示。

在 XS 创建的项目中，右击项目名称下方的项目名，在弹出的快捷菜单中选择 Add|New File…命令，弹出 New File 对话框，此时我们就会看到创建 ContentView 的两个选项，Forms ContentView，另一个为 Forms Xaml ContentView，如图 14.5 所示。

2.内容视图的派生关系

ContentView 页面派生自 Layout，同时它也是 Frame 帧布局的父类，派生关系图如图 14.7 所示。

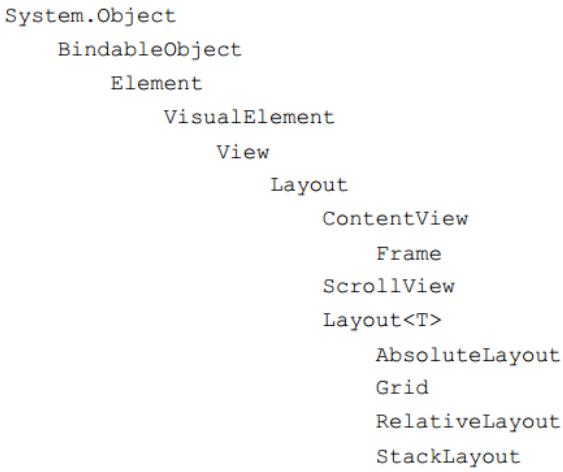


图 14.7　派生关系图

3.作用

ContentView 视图基本上有三个作用，下面依次介绍。

（1）范围框架：ContentView 视图可以构建一个范围框架，用来约束其中的子元素。

【示例 14-1：ContentViewScopeFrame】以下将使用 ContentView 来构建一个范围框架，以此来约束其子元素。代码如下：

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:ContentViewScopeFrame"
             x:Class="ContentViewScopeFrame.MainPage">
  <ContentView BackgroundColor="#B3EE3A"
               WidthRequest="300"
               HeightRequest="300"
               HorizontalOptions="Center"
               VerticalOptions="Center">
    <Label Text="Hello,Xamarin.Forms"
           FontAttributes="Bold"
           FontSize="18"
           HorizontalOptions="End"
           VerticalOptions="End"/>
  </ContentView>
</ContentPage>
```

在此代码中，我们将 ContentView 视图作为了范围框架，Label 只可以在此框架中进行布局。此时运行程序，会看到如图 14.8~14.9 所示的效果。

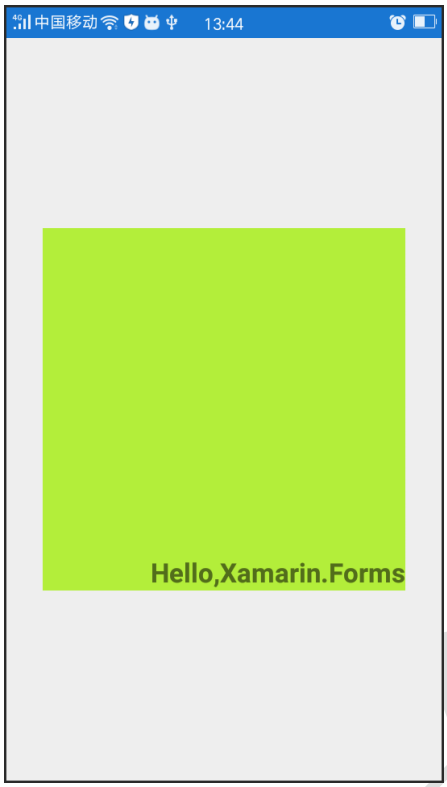


图 14.8　Android 的运行效果

图 14.9　iOS 的运行效果

（2）自定义视图的父类：ContentView 视图可以作为自定义视图的父类。

【示例 14-2】以下将自定义一个颜色视图。具体的操作步骤如下：

（1）创建一个 Forms Xaml View 文件，命名为 ColorView。

（2）打开 ColorView.xaml 文件，编写代码，构建自定义颜色视图。代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<ContentView xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="ContentViewCustomControls.ColorView">
  <Frame OutlineColor="Accent">
    <StackLayout Orientation="Horizontal">
      <BoxView x:Name="boxView"
               WidthRequest="70"
      HeightRequest="70" />
      <StackLayout>
        <Label x:Name="colorNameLabel"
               FontSize="Large"
               VerticalOptions="CenterAndExpand" />
        <Label x:Name="colorValueLabel"
               VerticalOptions="CenterAndExpand" />
      </StackLayout>
    </StackLayout>
  </Frame>
</ContentView>
```

（3）打开 ColorView.xaml.cs 文件，编写代码，实现一些与颜色视图相关的属性。代码如下：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Forms;
namespace ContentViewCustomControls
{
    public partial class ColorView : ContentView
    {
        string colorName;
        ColorTypeConverter colorTypeConv = new ColorTypeConverter();
        public ColorView()
        {
            InitializeComponent();
        }
        //颜色名称
        public string ColorName
        {
            set
            {
                colorName = value;
                colorNameLabel.Text = value;
                Color color = (Color)colorTypeConv.ConvertFromInvariantString(colorName);
                boxView.Color = color;
                colorValueLabel.Text = String.Format("{0:X2}-{1:X2}-{2:X2}",
                (int)(255 * color.R),
                (int)(255 * color.G),
                (int)(255 * color.B));
            }
            get
            {
                return colorName;
            }
        }
    }
}
```

（4）打开 MainPage.xaml 文件，编写代码，通过颜色视图实现对内容页面的布局。代码如下：

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:ContentViewCustomControls"
             x:Class="ContentViewCustomControls.MainPage">
  <ContentPage.Padding>
    <OnPlatform x:TypeArguments="Thickness"
                iOS="0, 20, 0, 0" />
  </ContentPage.Padding>
  <StackLayout Padding="6, 0">
    <local:ColorView ColorName="Aqua" />
    <local:ColorView ColorName="Black" />
```

```
        <local:ColorView ColorName="Blue" />
        <local:ColorView ColorName="Fuchsia" />
        <local:ColorView ColorName="Gray" />
    </StackLayout>
</ContentPage>
```

此时运行程序，会看到如图 14.10~14.11 所示的效果。

图 14.10　Android 的运行效果

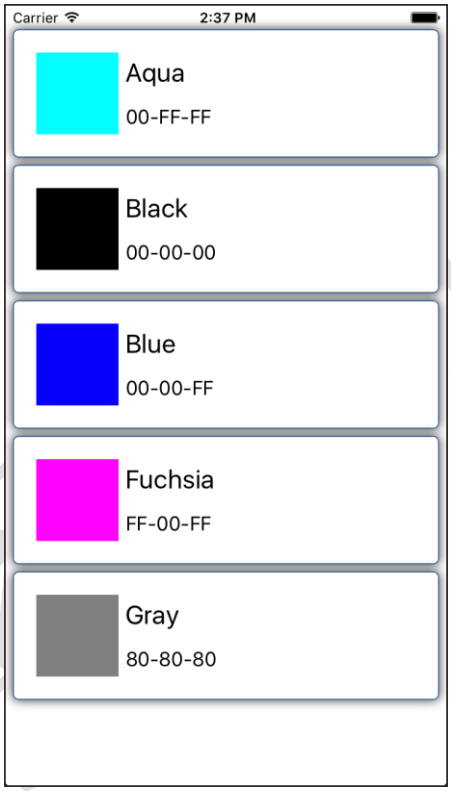


图 14.11　iOS 的运行效果

（3）构建更复杂的布局模式：在 ContentView 中可以包含视图，还可以包括布局，从而构建更为复杂的布局模式。

14.3　控件模板 ControlTemplate

ControlTemplate 是从 Xamarin.Forms 2.1.0 开始被引入的。ControlTemplate 被称为控件模板，它将页面的外观和内容进行了分离，从而让开发者可以更方便的创建基于主题的页面。本节将讲解控件模板相关的内容，其中包括构建控件模板以及控件模板的模板绑定等内容。

14.3.1　构建控件模板

控件模板可以在应用程序级别中构建，也可以在页面级别中构建。以下将对这两个构建方式进行讲解。

1.应用程序级别构建

如果开发者要在应用程序级别构建控件模板，首先必须将 ResourceDictionary 添加到 App 类中，然后在 ResourceDictionary 中实现模板的构建。其语法形式如下：

```
<Application>
  <Application.Resources>
    <ResourceDictionary>
      <ControlTemplate x:Key="KeyName">
      ……
      </ControlTemplate>
    </ResourceDictionary>
  </Application.Resources>
</Application>
```

其中，KeyName 指定一个字典键，用来指代控件模板。

构建好模板后，我们需要将这个模板控件显示出来，此时就需要可以模板化的视图。在这些视图中都会存在一个 ControlTemplate 属性。将此属性设置为创建的控件模板后，控件模板就可以进行显示了。在 Xamarin.Forms 目前只有 4 个视图包含 ControlTemplate 属性，这 4 个视图如下：

- ContentPage：内容页面
- ContentView：内容视图
- TemplatedPage：模板页面
- TemplatedView：模板视图

【示例 14-3：ControlTemplateDemo】下面将在应用程序级别中构建控件模板，实现应用程序主题的切换。具体的操作步骤如下：

（1）打开 App.xaml 文件，编写代码，实现在应用程序级别中构建控件模板，代码如下：

```
<?xml version="1.0" encoding="utf-8" ?>
<Application xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="ControlTemplateDemo.App">
  <Application.Resources>
    <ResourceDictionary>
      <!--构建控件模板-->
      <ControlTemplate x:Key="TealTemplate">
        <Grid>
          <Grid.RowDefinitions>
            <RowDefinition Height="0.1*" />
            <RowDefinition Height="0.8*" />
            <RowDefinition Height="0.1*" />
          </Grid.RowDefinitions>
          <Grid.ColumnDefinitions>
            <ColumnDefinition Width="0.05*" />
            <ColumnDefinition Width="0.95*" />
          </Grid.ColumnDefinitions>
          <BoxView Grid.ColumnSpan="2"
                   Color="Teal" />
          <Label Grid.Column="1"
                 Text="Knowledge is power."
                 TextColor="White"
                 FontSize="18"
                 VerticalOptions="Center" />
```

```
            <ContentPresenter Grid.Row="1"
                              Grid.ColumnSpan="2" />
            <BoxView Grid.Row="2"
                     Grid.ColumnSpan="2"
                     Color="Teal" />
            <Label Grid.Row="2"
                   Grid.Column="1"
                   Text="Xamarin.Froms XAML"
                   TextColor="White"
                   FontSize="18"
                  VerticalOptions="Center" />
        </Grid>
    </ControlTemplate>
    <!--构建控件模板-->
    <ControlTemplate x:Key="AquaTemplate">
        <Grid>
            <Grid.RowDefinitions>
                <RowDefinition Height="0.1*" />
                <RowDefinition Height="0.8*" />
                <RowDefinition Height="0.1*" />
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="0.05*" />
                <ColumnDefinition Width="0.95*" />
            </Grid.ColumnDefinitions>
            <BoxView Grid.ColumnSpan="2"
                     Color="Aqua" />
            <Label Grid.Column="1"
                   Text="Knowledge is power."
                   TextColor="Blue"
                   FontSize="18"
                   VerticalOptions="Center" />
            <ContentPresenter Grid.Row="1"
                              Grid.ColumnSpan="2" />
            <BoxView Grid.Row="2"
                     Grid.ColumnSpan="2"
                     Color="Aqua" />
            <Label Grid.Row="2"
                   Grid.Column="1"
                   Text="Xamarin.Froms XAML"
                   TextColor="Blue"
                   FontSize="18"
                   VerticalOptions="Center" />
        </Grid>
    </ControlTemplate>
  </ResourceDictionary>
 </Application.Resources>
</Application>
```

在此代码中，我们构建了两个控件模板，一个为 TealTemplate 控件模板，另一为 AquaTemplate 控件模板。

（2）打开 MainPage.xaml 文件，编写代码，将构建的控件模板应用于 ContentView 中。代码如下：

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:ControlTemplateDemo"
             x:Class="ControlTemplateDemo.MainPage">
  <ContentView x:Name="contentView"
               Padding="0,20,0,0"
               ControlTemplate="{StaticResource TealTemplate}">
    <StackLayout Spacing="20"
                 VerticalOptions="Center">
      <Label Text="If a jewel falls into the mire, it remains as precious as before; and though dust should ascend to heaven, its former worthlessness will not be altered."
             FontSize="20"
             FontAttributes="Bold"/>
      <Button Text="改变主题"
              Clicked="OnButtonClicked" />
    </StackLayout>
  </ContentView>
</ContentPage>
```

在此代码中 TealTemplate 控件模板通过使用 StaticResource 标记扩展分配给 ContentView.ControlTemplate 属性。ContentView.Content 属性设置为 StackLayout，用于定义要在 ContentPage 上显示的内容。 此内容将由 TealTemplate 中包含的 ContentPresenter 显示。

（3）打开 MainPage.xaml.cs 文件，编写代码，实现主题的切换功能。代码如下：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Forms;
namespace ControlTemplateDemo
{
    public partial class MainPage : ContentPage
    {
        bool originalTemplate = true;
        ControlTemplate tealTemplate;
        ControlTemplate aquaTemplate;
        public MainPage()
        {
            InitializeComponent();
            //实例化控件模板
            tealTemplate = (ControlTemplate)Application.Current.Resources["TealTemplate"];
            aquaTemplate = (ControlTemplate)Application.Current.Resources["AquaTemplate"];
        }
        //实现模板的切换
        void OnButtonClicked(object sender, EventArgs e)
        {
            originalTemplate = !originalTemplate;
            contentView.ControlTemplate = (originalTemplate) ? tealTemplate : aquaTemplate;
```

```
            }
        }
    }
```

此时运行程序，会看到如图 14.12~14.14 所示的效果。当开发者轻拍“改变主题”按钮后，会看到如图 14.15~14.17 所示的效果。

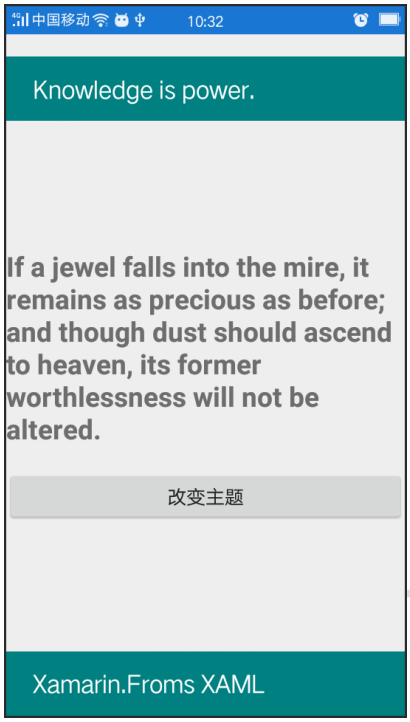


图 14.12　Android 的运行效果

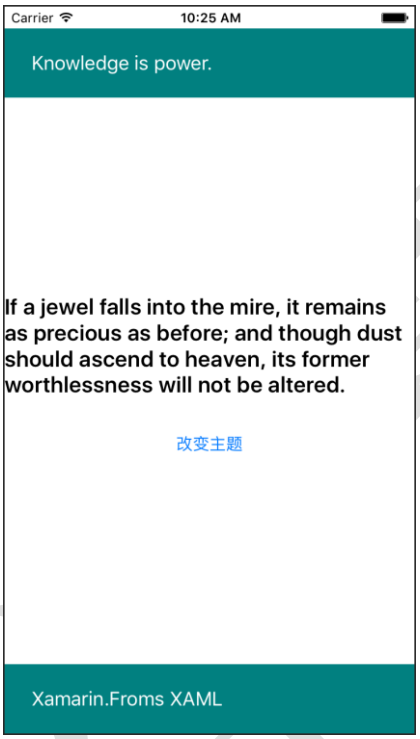
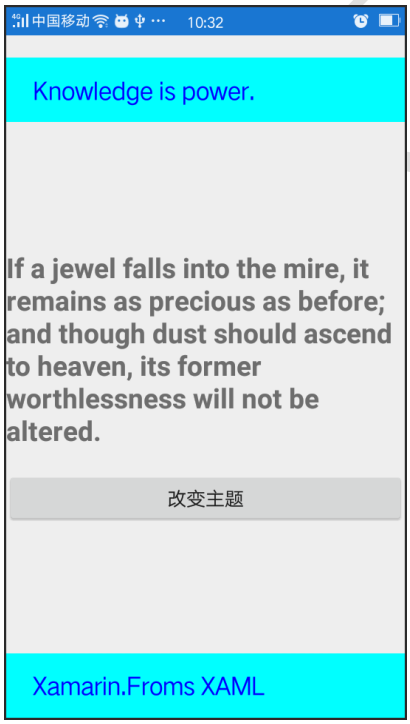


图 14.13　iOS 的运行效果

图 14.14　Windows Phone 的运行效果



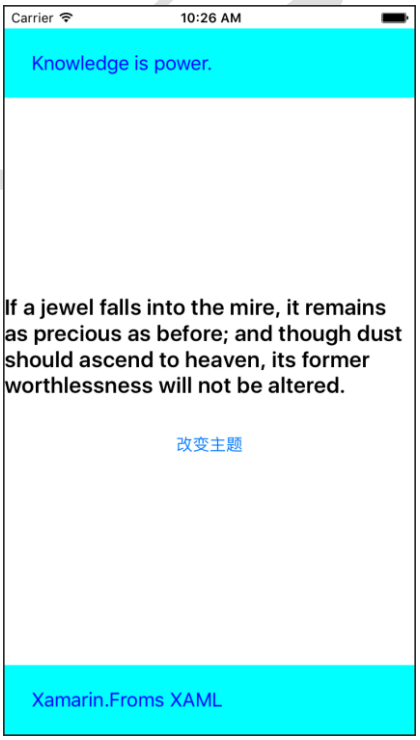


图 14.15　Android 的运行效果　　图 14.16　iOS 的运行效果　　图 14.17　Windows Phone 的运行效果

2.在页面级别中构建控件模板

如果开发者要在页面级别中构建控件模板，首先必须将 ResourceDictionary 添加到页面中，然后在 ResourceDictionary 中实现模板的构建即可，其语法形式如下：

```
<Page>
  <Page.Resources>
    <ResourceDictionary>
      <ControlTemplate x:Key="KeyName">
      ……
      </ControlTemplate>
    </ResourceDictionary>
  </Page.Resources>
</Page>
```

其中，Page 表示的是页面以及页面的子类。KeyName 用来指定一个字典键，此键指代的就是控件模板。

【示例 14-4：ControlTemplateContentPage】以下将在内容页面中构建控件模板。代码如下：

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:ControlTemplateContentPage"
             x:Class="ControlTemplateContentPage.MainPage">
  <ContentPage.Resources>
    <ResourceDictionary>
      <!--构建控件模板-->
      <ControlTemplate x:Key="TealTemplate">
        <StackLayout VerticalOptions="CenterAndExpand"
                     Spacing="20"
                     Padding="20">
          <Entry Placeholder="Username" />
          <Entry Placeholder="Password"
                 IsPassword="True"/>
          <Button Text="Click Here To Log In"     />
          <ContentPresenter />
        </StackLayout>
      </ControlTemplate>
    </ResourceDictionary>
  </ContentPage.Resources>
  <ContentView x:Name="contentView"
               Padding="0,20,0,0"
               ControlTemplate="{StaticResource TealTemplate}">
    <Frame OutlineColor="Accent">
      <Label Text="请在确认环境安全后，输入账号和对应的密码"
             FontAttributes="Bold"
             FontSize="18"/>
    </Frame>
  </ContentView>
</ContentPage>
```

此时运行程序，会看到如图 14.18~14.20 所示的效果。

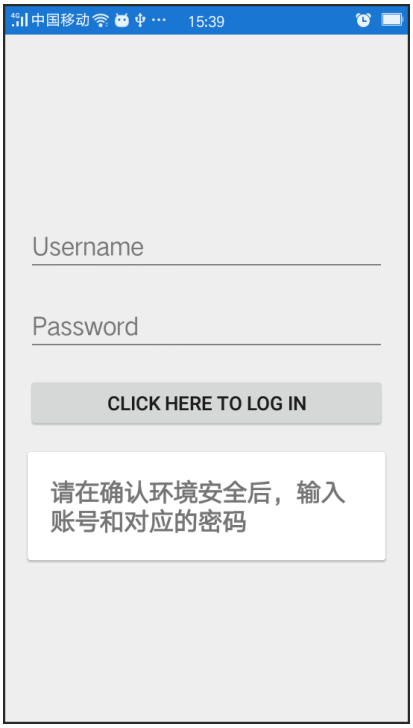


图 14.18　Android 的运行效果

图 14.19　iOS 的运行效果

图 14.20　Windows Phone 的运行效果

14.3.2　控件模板的模板绑定

为了可以轻松更改控件模板中控件上的属性值，可以在控件模板中实现模板绑定功能。模板绑定允许控件模板中的控件将数据绑定到公共属性上。这时需要使用 TemplateBinding。它可以将控件模板中的控件的属性绑定到拥有控件模板的目标视图的父级上的可绑定属性上。

注意：（1）TemplateBinding 类似于现有的 Binding，不同之处在于 TemplateBinding 的源总是自动设置为拥有控件模板的目标视图的父级。（2）不支持在控件模板之外使用 TemplateBinding。

【示例 14-5：ControlTemplateDemo】以下将以项目 ControlTemplateDemo 为基础，在控件模板中实现模板绑定功能。具体的操作步骤如下：

（1）打开 MainPage.xaml 文件，编写代码，实现可绑定属性的定义。代码如下：

```
namespace ControlTemplateDemo
{
    public partial class MainPage : ContentPage
    {
        bool originalTemplate = true;
        ControlTemplate tealTemplate;
        ControlTemplate aquaTemplate;
        public static readonly BindableProperty HeaderTextProperty = BindableProperty.Create("HeaderText",
                                                                     typeof(string),
                                                                     typeof(MainPage),
                                                                     "Knowledge is power.");
        public static readonly BindableProperty FooterTextProperty = BindableProperty.Create("FooterText",
                                                                   typeof(string),
                                                                   typeof(MainPage),
```

```
                                                                    "Xamarin.Froms XAML");
        public MainPage()
        {
            InitializeComponent();
            ……                    //此处省略了对 tealTemplate 和 aquaTemplate 对象的实例化
        }
        public string HeaderText
        {
            get
            {
                return (string)GetValue(HeaderTextProperty);
            }
        }
        public string FooterText
        {
            get
            {
                return (string)GetValue(FooterTextProperty);
            }
        }
        ……                        //此处省略了对 OnButtonClicked 方法的实现
    }
}
```

（2）打开 App.xaml 文件，编写代码，在第一个构建的 ControlTemplate 中实现模板绑定功能。代码如下：

```
<ControlTemplate x:Key="TealTemplate">
  <Grid>
    <Grid.RowDefinitions>
      <RowDefinition Height="0.1*" />
      <RowDefinition Height="0.8*" />
      <RowDefinition Height="0.1*" />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="0.05*" />
      <ColumnDefinition Width="0.95*" />
    </Grid.ColumnDefinitions>
    <BoxView Grid.ColumnSpan="2"
             Color="Teal" />
    <Label Grid.Column="1"
           Text="{TemplateBinding Parent.HeaderText}"
           TextColor="White"
           FontSize="18"
           VerticalOptions="Center" />
    <ContentPresenter Grid.Row="1"
                      Grid.ColumnSpan="2" />
    <BoxView Grid.Row="2"
             Grid.ColumnSpan="2"
             Color="Teal" />
    <Label Grid.Row="2"
           Grid.Column="1"
```

```
            Text="{TemplateBinding Parent.FooterText}"
            TextColor="White"
            FontSize="18"
            VerticalOptions="Center" />
    </Grid>
</ControlTemplate>
```

在此代码中，我们将两个 Label 控件的 Text 属性实现了模板绑定功能，在上文中我们提到了属性使用模板绑定将其绑定到拥有 ControlTemplate 的目标视图的父级上的可绑定属性上。但是，在我们的代码中，模板绑定绑定到 Parent.HeaderText 和 Parent.FooterText 上，而不是 HeaderText 和 FooterText 上。这是因为在此代码中，可绑定属性是在目标视图的祖父级上定义的，而不是父级。

注意：模板绑定的源始终自动设置为拥有控件模板的目标视图的父级，在此项目中是 ContentView 实例。模板绑定使用 Parent 属性返回 ContentView 实例的父元素，这是 ContentPage 实例。

此时运行程序，会看到和图 14.12~14.14 一样的运行效果。

14.4 模板页面 TemplatedPage

在上文中我们提到了 TemplatedPage，它被称为模板页面，用来显示控件模版。TemplatedPage 用作基类，将 ContentPage 替换为最基本的页。与 ContentPage 不同，TemplatedPage 没有 Content 属性。因此开发者不能直接将内容包装进去。这意味着在 TemplatedPage 中获取内容的唯一方法是设置 ControlTemplate，否则它将显示为空白。

14.5 模板视图 TemplatedView

与模板页面相对的是 TemplatedView，它被称为模板视图，它的功能和模板页面类似，也是用来显示控件模板的，只不过比模板页面更加灵活。TemplatedView 提供 ControlTemplate 属性，实现对控件模板的关联，从而展现对应的界面。

【示例 14-6：TemplatedViewDemo】以下将使用模板视图显示控件模板，并实现模板的切换。具体的操作步骤如下：

（1）打开 App.xaml 文件，编写代码，实现在应用程序级别中构建控件模板，代码如下：

```
<?xml version="1.0" encoding="utf-8" ?>
<Application xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="TemplatedViewDemo.App">
  <Application.Resources>
    <ResourceDictionary>
      <!--构建控件模板-->
      <ControlTemplate x:Key="ChineseTemplate">
        <StackLayout>
          <StackLayout VerticalOptions="End">
            <BoxView Color="Aqua" />
          </StackLayout>
          <StackLayout Spacing="35"
            VerticalOptions="CenterAndExpand"   >
```

```
                <Frame OutlineColor="Accent">
                  <StackLayout Spacing="20"
                               VerticalOptions="CenterAndExpand"
                               HorizontalOptions="Center">
                    <Label Text="山居秋暝"
                           FontSize="30"
                           FontAttributes="Bold"
                           HorizontalOptions="Center"/>
                    <Label Text="空山新雨后，天气晚来秋。"
                           FontSize="18"/>
                    <Label Text="明月松间照，清泉石上流。"
                           FontSize="18"/>
                    <Label Text="竹喧归浣女，莲动下渔舟。"
                           FontSize="18"/>
                    <Label Text="随意春芳歇，王孙自可留。"
                           FontSize="18"/>
                  </StackLayout>
                </Frame>
                <Button Command="{TemplateBinding Parent. CommandEnglish}"
                        Text="Enter English Template" />
              </StackLayout>
            </StackLayout>
          </ControlTemplate>
          <!--构建控件模板-->
          <ControlTemplate x:Key="EnglishTemplate">
            <StackLayout>
              <StackLayout VerticalOptions="End">
                <BoxView Color="Green" />
              </StackLayout>
              <StackLayout Spacing="35"
                           VerticalOptions="CenterAndExpand"    >
                <Frame OutlineColor="Accent">
                  <Label Text="your life only lasts for a few decades, so be sure that you don\'t leave any regrets.
laugh or cry as you like, and it's meaningless to oppress yourself."
                         FontAttributes="Bold"
                         FontSize="18"/>
                </Frame>
                <Button Command="{TemplateBinding Parent.CommandChinese}"
                        Text="Enter Chinese Template" />
              </StackLayout>
            </StackLayout>
          </ControlTemplate>
        </ResourceDictionary>
      </Application.Resources>
    </Application>
```

在此代码中，我们构建了两个控件模板，一个为 ChineseTemplate 控件模板，另一为 EnglishTemplate 控件模板。

（2）打开 MainPage.xaml 文件，编写代码，将构建的控件模板应用于中 TemplatedView。代码如下：

```
<?xml version="1.0" encoding="utf-8" ?>
```

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:TemplatedViewDemo"
             x:Class="TemplatedViewDemo.MainPage">
  <TemplatedView x:Name="templatedView"
                 Padding="0,20,0,0"
                 ControlTemplate="{StaticResource ChineseTemplate}">
  </TemplatedView>
</ContentPage>
```

（3）打开 MainPage.xaml.cs 文件，编写代码，实现控件模板的切换功能。代码如下：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Input;
using Xamarin.Forms;
namespace TemplatedViewDemo
{
    public partial class MainPage : ContentPage
    {
        ControlTemplate chineseTemplate;
        ControlTemplate englishTemplate;
        public ICommand CommandEnglish { private set; get; }
        public ICommand CommandChinese { private set; get; }
        public MainPage()
        {
            CommandEnglish = new Command(() => OnEnterEnglishTemplate());
            CommandChinese = new Command(() => OnEnterChineseTemplate());
            InitializeComponent();
            chineseTemplate = (ControlTemplate)Application.Current.Resources["ChineseTemplate"];
            englishTemplate = (ControlTemplate)Application.Current.Resources["EnglishTemplate"];
        }
        //设置为 EnglishTemplate 控件模板
        public void OnEnterEnglishTemplate()
        {
            templatedView.ControlTemplate = englishTemplate;
        }
        //设置为 ChineseTemplate 控件模板
        public void OnEnterChineseTemplate()
        {
            templatedView.ControlTemplate = chineseTemplate;
        }
    }
}
```

此时运行程序，会看到如图 14.21~14.23 所示的效果。当开发者轻拍 Enter English Template 按钮后，会看到如图 14.24~14.26 所示的效果。当开发者轻拍 Enter Chinese Template 按钮后，会看到如图 14.21~14.23 所示的效果。

图 14.21　Android 的运行效果

图 14.22　iOS 的运行效果

图 14.23　Windows Phone 的运行效果

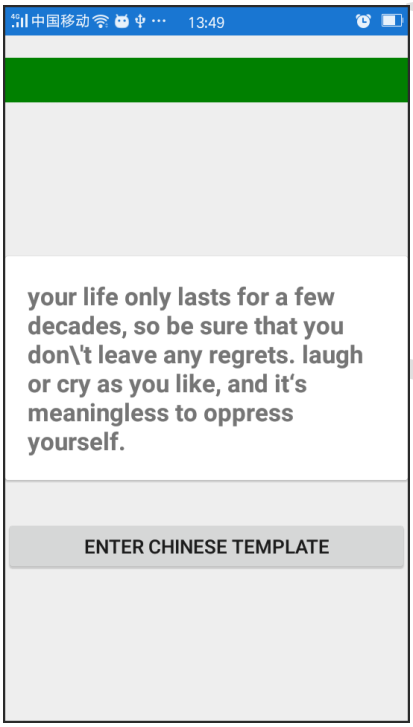


图 14.24　Android 的运行效果

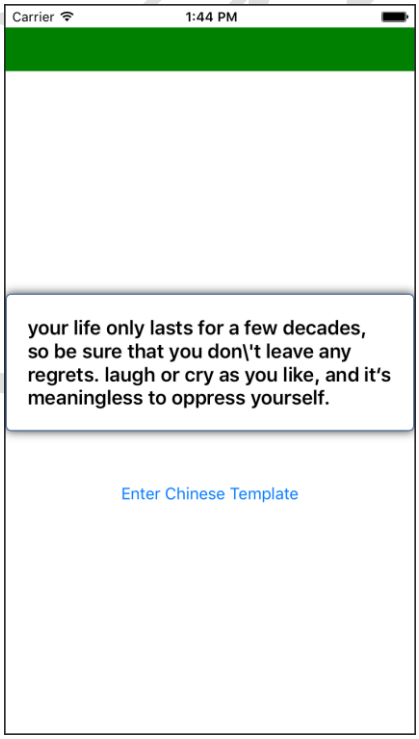


图 14.25　iOS 的运行效果

图 14.26　Windows Phone 的运行效果