第3章 XAML基础及布局设计

学习目的与要求

通过本章学习，要求了解 Windows Phone 开发的布局方式；掌握使用 Grid、StackPanel、Canvas、Pivot、Panorama 进行布局设计的方法。

本章主要内容

通过旅游项目的典型布局界面，介绍 Windows Phone 多种布局设计的常用方法。

3.1 XAML 简介

XAML 是 eXtensible Application Markup Language 的英文缩写，其中文名称为可扩展应用程序标记语言，是微软公司为构建应用程序用户界面而创建的一种新的描述性语言。XAML 提供了一种便于扩展和定位的语法来定义和程序逻辑分离的用户界面，而这种实现方式和 ASP.NET 中的“代码后置”模型非常类似。XAML 是一种解析性语言，尽管它也可以被编译。它的优点是简化编程中的用户创建过程，应用时要添加代码等。

3.2 布局方式

布局管理是从一个整体的角度去把握手机应用的界面设计，尽管手机应用的界面相对于 Web 程序和计算机桌面的程序界面要小了一些，但是没有一个好的界面布局也会影响整个应用的用户体验效果。Windows Phone 手机应用程序的界面布局容器主要有 Grid、Canvas 和 StackPanel 等。在一部手机应用里面，所有元素的最顶层必须是一个容

器(如 Grid、Canvas、StackPanel 等),然后在容器中摆放元素,容器中也可能包含容器。从布局的角度来说,外层的容器管理里面的容器,而里面的容器也管理它们里面的容器。

每一个布局的容器都有自己的规则。容器不仅是分配空间,还决定元素的位置。

在布局方面,Windows Phone 还提供了两个非常有用的控件,分别是枢轴控件 Pivot 和全景视图控件 Panorama。使用这两个控件可以很方便地在一个页面上完成多个视图的布局,扩充页面的展示范围,并且具有良好的用户体验。

Windows Phone 中常见的布局方式有如下几种:

(1) 网格布局(Grid 容器):按照行列方式布局程序的界面。定义一个区域,在此区域内,用户可以使用相对于 Canvas 区域的坐标显示定位子元素。

(2) 绝对布局(Canvas 容器):按照绝对坐标来布局程序的界面。

(3) 堆放布局(StackPanel 容器):按照垂直或者水平方式布局程序的界面。

(4) 枢轴视图布局(Pivot 控件):通过类似页签的方式在一个页面上展示多个视图。

(5) 全景视图布局(Panorama 控件):通过左右滑动的方式在一个页面显示多个视图。

用户可以根据自己应用的实际情况来选择布局的容器,可以选择一种布局容器或者几种布局容器结合起来进行布局设计。

3.3 使用 Grid 面板进行布局

Grid(网格布局)定义了一个表格,然后设置表格里面的行和列。

下面介绍 Grid 中几个重要属性的设置。

- RowDefinations 和 ColumnDefinations:RowDefinations 和 ColumnDefinations 属性主要是指定 Grid 控件的行数和列数。内部嵌套几个 Definition 就代表这个 Grid 有几行几列。
- Grid.Row 和 Grid.Column:当使用其他控件内置于 Grid 中时,需要使用 Grid.Row 和 Grid.Column 来指定它所在的行和列。
- VerticalAlignment 和 HorizontalAlignment:可以根据 VerticalAlignment 和 HorizontalAlignment 属性来设置行和列的位置,这样会自动根据 Grid 容器的大小来调整行和列的大小,以使整个行和列都铺满了 Grid 的所有的控件。

下面给出利用 Grid 布局的示例代码。

源代码:chapter3\chapter3-3。

Grid.xaml 文件主要代码如下:

```
<Grid x:Name="LayoutRoot" Background="Transparent">
    <Grid.RowDefinitions>
        <RowDefinition Height="600"/>
        <RowDefinition Height="108"/>
        <RowDefinition Height="60"/>
    </Grid.RowDefinitions>
```

```
    <Grid.ColumnDefinitions>
        <ColumnDefinition/>
        <ColumnDefinition/>
        <ColumnDefinition/>
    </Grid.ColumnDefinitions>
    <Image Source="img/1.png" Grid.Row="0" Grid.ColumnSpan="3"
      Margin="-5,-5,5,5"/>
    <Image Source="img/201.png" Grid.Row="1" Grid.Column="0"/>
    <Image Source="img/201.png" Grid.Row="1" Grid.Column="1"/>
    <Image Source="img/201.png" Grid.Row="1" Grid.Column="2"/>
    <TextBlock Grid.Row="2" Text="昌平旅游" HorizontalAlignment="Center"
      VerticalAlignment="Center" Grid.Column="1" Margin="42,5,38,-5"/>
</Grid>
```

程序运行的效果如图 3-1 所示。

图 3-1 Grid 布局示例

3.4 使用 StackPanel 面板进行布局

StackPanel(堆放布局)的方式是将子元素排列成一行(可沿水平或垂直方向)。StackPanel 的规则是：根据附加属性,让元素水平排列或者竖直排列。

StackPanel 为启用布局的 Panel 元素之一。在特定情形下,例如要将一组对象摆列在竖直或水平列表中,StackPanel 就能发挥很好的作用。

下面介绍 StackPanel 中几个重要属性的设置。

- Orientation 属性：设置 Orientation 属性可确定列表的方向。Orientation = "Horizontal" 表示沿水平方向放置 StackPanel 里面的元素，Orientation = "Vertical" 表示沿垂直方向放置 StackPanel 里面的子元素，Orientation 属性的默认值为 Vertical，即默认是垂直方向放置的。
- VerticalAlignment 属性：VerticalAlignment 属性表示元素垂直方向的位置。其中，VerticalAlignment = "Bottom"，表示子元素放在 StackPanel 容器的底部；VerticalAlignment = "Center"，表示子元素放在 StackPanel 容器的中间；VerticalAlignment = "Stretch"，表示子元素拉伸铺放在 StackPanel 容器中；VerticalAlignment = "Top"，表示子元素放在 StackPanel 容器的上方。VerticalAlignment 属性的默认值为 Stretch。
- HorizontalAlignment 属性：HorizontalAlignment 属性表示元素水平方向的位置，HorizontalAlignment = "Left" 表示子元素放在 StackPanel 容器的左边，HorizontalAlignment = "Center" 表示子元素放在 StackPanel 容器的中间，HorizontalAlignment = "Stretch" 表示子元素拉伸铺放在 StackPanel 容器，HorizontalAlignment = "Right" 表示子元素放在 StackPanel 容器的右边。HorizontalAlignment 属性的默认值为 Stretch。

下面给出利用 StackPanel 布局的示例代码。

源代码：chapter3\chapter3-4。

StackPanel.xaml 文件主要代码：

```
<!--LayoutRoot 是包含所有页面内容的根网格-->
    <Grid x:Name="LayoutRoot" Background="Transparent">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>
        <StackPanel Orientation="Vertical"
            VerticalAlignment="Stretch" HorizontalAlignment="Center">
            <Image Source="img/button1.png"></Image>
            <Image Source="img/button2.png"></Image>
            <Image Source="img/button3.png"></Image>
            <Image Source="img/button4.png"></Image>
            <Image Source="img/button5.png"></Image>
        </StackPanel>
</Grid>
```

程序运行的效果如图 3-2 所示。

图 3-2 StackPanel 布局示例

3.5 使用 Canvas 面板进行布局

Canvas(绝对布局)画布定义一个区域,在此区域内,用户可以使用相对于 Canvas 区域的坐标来显示定位子元素。Canvas 就像一张油布一样,所有的控件都可以堆放到这张布上,采用绝对坐标定位,可以使用附加属性(AttachedProperty)对 Canvas 中的元素进行定位,通过附加属性指定控件相对于其直接父容器 Canvas 控件的上、下、左、右坐标的位置。

Canvas 的规则是:读取附加属性 Canvas. Left、Canvas. Right、Canvas. Top 和 Canvas. Bottom,并以此来决定元素的位置。可以嵌套 Canvas 对象。嵌套对象时,每个对象使用的坐标都是相对于直接包含它们的 Canvas 而言的。每个子对象都必须是 UIElement。在 XAML 中,将子对象声明为对象元素,这些元素是 Canvas 对象元素的内部 XML。在代码中,可以通过获取由 Children 属性访问的集合来操作 Canvas 子对象的集合。由于 Canvas 为 UIElement 类型,因此可以嵌套 Canvas 对象。在很多情况下,Canvas 仅仅用作其他对象的容器,而没有任何可见属性。如果满足以下任一条件,Canvas 即不可见:

(1) Height 属性等于 0。

(2) Width 属性等于 0。

(3) Background 属性等于 null。

(4) Opacity 属性等于 0。

(5) Visibility 属性等于 Collapsed。

下面给出利用 Canvas 布局的示例代码。

源代码:chapter3\chapter3-5。

Canvas. xaml 文件主要代码如下:

```
<Grid x:Name="LayoutRoot" Background="Transparent">
```

```
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>

    <!--TitlePanel 包含应用程序的名称和页标题-->
    <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
        <TextBlock Text="昌平旅游" Style="{StaticResource
            PhoneTextNormalStyle}" Margin="12,0"/>
        <TextBlock Text="登录" Margin="9,-7,0,0" Style="{StaticResource
            PhoneTextTitle1Style}"/>
    </StackPanel>

    <Canvas>

        <TextBlock Canvas.Top="200" Canvas.Left="60">账号:</TextBlock>
        <TextBox Width="300" Canvas.Top="170" Canvas.Left="120"></TextBox>
        <TextBlock Canvas.Top="260" Canvas.Left="60">密码:</TextBlock>
        <TextBox Width="300" Canvas.Top="230" Canvas.Left="120"></TextBox>

        <CheckBox Canvas.Top="300" Canvas.Left="120">记住密码</CheckBox>
        <CheckBox Canvas.Top="300" Canvas.Left="280">自动登录</CheckBox>

        <Button Canvas.Top="360" Canvas.Left="120" Width="150" Height="80"
            Content="登录"></Button>
        <Button Canvas.Top="360" Canvas.Left="260" Width="150" Height="80"
            Content="取消"></Button>
        </Canvas>
</Grid>
```

程序运行的效果如图 3-3 所示。

图 3-3　Canvas 布局示例

3.6 使用 Pivot 面板进行枢轴视图布局

Pivot(枢轴视图布局)提供了一种快捷的方式来管理应用程序中的视图或页面,通过一种类似于标签的方式来将视图分类,这样就可以在一个界面上通过切换标签来浏览多个数据集或者切换应用视图。枢轴视图控件放置独立的视图,同时处理左侧和右侧的导航,可以通过划动或者平移手势来切换枢轴控件中的视图。

下面介绍 Pivot 中的重要属性和事件。

- HeaderTemplate:获取或设置 Pivot 控件的 Header 属性和 PivotItem 子对象的模板。
- SelectedIndex:获取或设置当前选择的 PivotItem 的索引。
- SelectItem:获取或设置当前选择的 PivotItem 的对象。
- Title:获取或设置 Pivot 的标题。
- TitleTemplate:获取或设置 Pivot 标题的模板,通过设置它可以改变 Pivot 标题的外观。
- LoadedPivotItem:该事件在加载完成后被触发。
- LoadingPivotItem:该事件在加载 PivotItem 时被触发,可以用来动态加载数据。
- SelectionChanged:选择的 PivotItem 改变时该事件被触发。
- UnloadedPivotItem:该事件在离开当前的 PivotItem 完成时被触发。
- UnloadingPivotItem:该事件在离开当前的 PivotItem 开始时被触发。

下面给出利用 Pivot 布局的示例代码。

源代码:chapter3\chapter3-6。

Pivot.xaml 文件主要代码如下:

```
<Grid x:Name="LayoutRoot" Background="Transparent">
    <!--Pivot 控件-->
    <phone:Pivot Title="Pivot 的标题" Name="myPivot"
                 LoadedPivotItem="Pivot_LoadedPivotItem">
        <phone:PivotItem Header="注册">
          <TextBlock Text="这里是注册的内容"></TextBlock>
        </phone:PivotItem>
        <phone:PivotItem Header="旅游热点">
            <TextBlock Text="这里是旅游热点的内容"></TextBlock>
        </phone:PivotItem>
    </phone:Pivot>
</Grid>
```

Pivot.xaml.cs 文件主要代码如下:

```
private void Pivot_LoadedPivotItem(object sender, PivotItemEventArgs e)
{
    myPivot.Title="加载 PivotItem 完成";
}
```

运行模拟器，效果如图 3-4 所示。

图 3-4　Pivot 布局示例 1、2

3.7　使用 Panorama 面板进行全景视图布局

Panorama(全景视图布局)是 Windows Phone 上一个独特的视图控件，给用户提供一种纵向的拉伸延长的视图布局。它通过使用一个超过屏幕宽度的长水平画布，提供一种独特显示控件、数据和服务的方式。

全景视图布局的用户接口由 3 层类型组成：背景、全景标题、全景区域。

下面给出利用 Panorama 布局的示例代码。

源代码：chapter3\chapter3-7。

Panorama.xaml 文件主要代码如下：

```
<Grid x:Name="LayoutRoot" Background="Transparent">
    <!--Panorama 控件-->
        <phone:Panorama Title="我的应用" SelectionChanged=
            "Panorama_SelectionChanged" Name="myPanorama">
        <phone:Panorama.Background>
            <ImageBrush ImageSource="PanoramaBackground.png"></ImageBrush>
        </phone:Panorama.Background>
        <phone:PanoramaItem Header="景点搜索">
            <StackPanel>
                <TextBlock Text="景点搜索" FontSize="50"></TextBlock>
                <TextBlock Text="这是第一个搜索" FontSize="50"></TextBlock>
            </StackPanel>
        </phone:PanoramaItem>
        <phone:PanoramaItem Header="购物搜索">
            <StackPanel>
                <TextBlock Text="购物搜索" FontSize="50"></TextBlock>
                <TextBlock Text="这是第二个搜索" FontSize="50"></TextBlock>
            </StackPanel>
        </phone:PanoramaItem>
```

```
    </phone:Panorama>
</Grid>
```

Panorama. xaml. cs 文件主要代码如下：

```
private void Panorama_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    myPanorama.Title="搜索";
}
```

运行模拟器，效果如图 3-5 所示。

图 3-5　Panorama 布局示例

3.8　项目案例

1. 学习目标

掌握使用 Grid 面板进行布局。

2. 案例描述

使用 Grid 面板完成导航菜单的布局。

3. 案例要点

Grid. Column 及 Grid. Row 的用法。

4. 案例实施

源代码：chapter3\chapter3-8。

(1) 新建解决方案，命名为 Application4-6Demo。

(2) GridMenu. xaml 文件主要代码如下：

```
<Grid x:Name="LayoutRoot" Background="Transparent">
    <Grid.RowDefinitions>
        <RowDefinition Height="80"/>
        <RowDefinition Height="600"/>
        <RowDefinition Height="100"/>
    </Grid.RowDefinitions>
```

```
<Grid.ColumnDefinitions>
    <ColumnDefinition/>
    <ColumnDefinition/>
    <ColumnDefinition/>
    <ColumnDefinition/>
    <ColumnDefinition/>
</Grid.ColumnDefinitions>
<StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,7,0,28"
    Grid.ColumnSpan="4">
    <TextBlock x:Name="ApplicationTitle" Text="昌平旅游"
Style="{StaticResource PhoneTextNormalStyle}" FontSize="28" Height="38"
    RenderTransformOrigin="0.506,1.016"/>
</StackPanel>
<Image Source="icon/index-0.png" Grid.ColumnSpan="5" Margin="0,61,0,19"
    Grid.RowSpan="2"/>
<Image Source="icon/icon_1_n.png" Grid.Row="1" Height="75"
    Grid.Column="0" Margin="0,582,0,18" Grid.RowSpan="2"/>
<Image Source="icon/icon_2_n.png" Grid.Row="1" Height="75"
    Grid.Column="1" Margin="0,582,0,18" Grid.RowSpan="2" />
<Image Source="icon/icon_3_n.png" Grid.Row="1" Height="75"
    Grid.Column="2" Margin="0,582,0,18" Grid.RowSpan="2" />
<Image Source="icon/icon_4_n.png" Grid.Row="1" Height="75"
    Grid.Column="3" Margin="0,582,0,18" Grid.RowSpan="2" />
<Image Source="icon/icon_5_n.png" Grid.Row="1" Height="75"
    Grid.Column="4" Margin="0,582,0,18" Grid.RowSpan="2" />
```

（3）运行模拟器，效果如图 3-6 所示。

图 3-6　GridMenu 布局示例

5. 特别提示

使用图片前，请注意处理好图片的尺寸。

6. 拓展与提高

(1) 若将图片上的时间改成动态时间，应该如何操作？

(2) 背景图片如何根据“活动情况”获取最新活动图片？

本章小结

本章对 XAML 及常用的布局方式进行了简要介绍，重点介绍了需要掌握常用的 5 种布局方法。

习　题

一、选择题

1. Grid 中几个重要属性的设置包括(　　)。

A. RowDefinations 和 ColumnDefinations

B. Grid.Row 和 Grid.Column

C. Grid.Rows 和 Grid.Columns

D. VerticalAlignment 和 HorizontalAlignment

2. 如果满足以下任一条件，Canvas 即不可见(　　)。

A. Height 属性等于 0

B. Width 属性等于 0

C. Background 属性等于 null

D. Opacity 属性等于 0

E. Visibility 属性等于 Collapsed

二、简答题

1. Windows Phone 中有哪些常见的布局方式？

2. Grid 中包括哪些重要属性？

第4章 常用控件

学习目的与要求

通过本章学习，要求了解常用控件 TextBlock、TextBox、Button、RadioButton、CheckBox、ListBox、Slider 的各种属性及方法，能够熟练运用常用控件进行编程。结合学习这些控件，并进行扩展学习，掌握更多控件的使用。

本章主要内容

学习如何在程序中使用常用控件 TextBlock、TextBox、Button、RadioButton、CheckBox、ListBox、Slider。

4.1 控件简介

Windows Phone 8 系统提供了丰富的可视化控件，熟练地掌握这些控件的使用是开发 Windows Phone 应用程序的必要条件，也是给用户良好的用户体验的保证。

大部分 Windows Phone 控件都间接或者直接继承了 System. Windows. UIElement、System. Windows. FrameworkElement和 System. Windows. Controls. Control 这 3 个基类。Windows Phone 的控件的实现，都是通过直接或者间接继承这些基类来扩展的，然后再根据控件的特性来定义和实现控件自身的属性和方法。

4.2 TextBlock 控件

TextBlock(文本块)控件是用于显示少量文本的轻量控件，可以通过 TextBlock 呈现只读的文本。经常用到的属性如下：

- FontFamily：字体名称。

- FontSize：字体大小。
- FontStyle：字体样式。
- FontWeight：文字的粗细。
- Foreground：字体前景颜色。
- Width：文字区块宽度。
- Height：文字区块高度。
- Opacity：文字透明度。
- Text：字体正文。
- TextWrapping：字体换行。Wrap 属性可强制换行。

下面给出应用 TextBlock 控件的示例代码。

源代码：chapter4\chapter4-2。

TextBlockMain.xaml 文件主要代码如下：

```
<Grid x:Name="LayoutRoot" Background="Transparent">

    <TextBlock Name="TextBlock1" Height="30" Foreground="Red"
    Margin="30,-690,0,0" Text="你好,昌平!"></TextBlock>

    <!--自定义断行-->
    <TextBlock Margin="30,64,0,0">
        你好!
        <LineBreak/>
        来自美国的朋友
        <LineBreak/>
        欢迎来北京昌平旅游!
    </TextBlock>

    <!--设置 TextBlock 控件自动折行-->
    <TextBlock TextWrapping="Wrap" Margin="30,150,0,0">
        朋朋朋朋朋朋朋朋朋朋朋朋朋朋朋朋朋朋朋朋朋朋朋朋朋朋朋朋朋
    </TextBlock>

    </Grid>
```

TextBlockMain.xaml.cs 文件主要代码如下：

```
public TextBlockMain()
{
    InitializeComponent();
    //用于本地化 ApplicationBar 的示例代码
    //BuildLocalizedApplicationBar();

    TextBlock txtBlock=new TextBlock();
    txtBlock.Height=50;
    txtBlock.Width=200;
    txtBlock.Text="动态创建的 TextBlock";
    LayoutRoot.Children.Add(txtBlock);
}
```

运行模拟器，效果如图 4-1 所示。

图 4-1　TextBlock 应用示例

4.3　TextBox 控件

TextBox(文本框)控件是表示一个可用于显示和编辑单格式、多行文本的控件。

常用属性及事件如下。

- AcceptsReturn：获取或设置一个值，该值确定文本框是否允许和显示换行符或回车键。
- CaretBrush：获取或设置用于呈现指示插入点的竖线的画笔。
- FontSource：获取或设置应用于 TextBox 以呈现内容的字体源。
- HorizontalScrollBarVisibility：获取或设置水平滚动条的可见性。
- InputScope：获取或设置此 TextBox 使用的输入的上下文。
- IsReadOnly：获取或设置一个值，该值确定用户是否能够在文本框中更改文本。
- MaxLength：获取或设置一个值，该值确定用户输入所允许的最大字符数。
- SelectedText：获取或设置文本框中当前选择的内容。
- SelectionBackground：获取或设置填充选定文本的背景的画笔。
- SelectionForeground：获取或设置用于文本框中选定文本的画笔。
- SelectionLength：获取或设置文本框中当前选定内容的字符数。
- SelectionStart：获取或设置文本框中选定文本的起始位置。
- Text：获取或设置文本框的文本内容。
- TextAlignment：获取或设置文本应在文本框中进行对齐的方式。
- TextWrapping：获取或设置当一行文本超过文本框的可用宽度后如何进行换行。
- VerticalScrollBarVisibility：获取或设置垂直滚动条的可见性。
- SelectionChanged：在文本选定内容更改后发生。
- TextChanged：在文本框中的内容更改时发生。

下面给出应用 TextBox 控件的示例代码。

源代码：chapter4\chapter4-3。

TextBoxMain.xaml 文件主要代码如下：

```
<Grid x:Name="LayoutRoot" Background="# FFF9EBEB" ShowGridLines="True">
```

```
    <TextBox Name="TextBox1" Height="100" VerticalAlignment="Top"
        TextWrapping="Wrap" TextChanged="TextBox1_TextChanged">
    </TextBox>
    <TextBlock Foreground="Black" Margin="12,172,0,0" Height="60"
        HorizontalAlignment="Left" Name="textBlock1" Text=""
        VerticalAlignment="Top" Width="364">
    </TextBlock>
</Grid>
```

TextBoxMain. xaml. cs 文件主要代码如下：

```
private void TextBox1_TextChanged(object sender, TextChangedEventArgs e)
{
    textBlock1.Text=TextBox1.Text;
}
```

运行模拟器，效果如图 4-2 所示。

图 4-2　TextBox 应用示例

4.4　Button 控件

Button(按钮)控件是一个表示按钮的控件，是单击之后要触发事件的控件。主要应用是在单击时触发 Click 事件。

下面给出应用 Button 控件的示例代码。

源代码：chapter4\chapter4-4。

ButtonMain. xaml 文件主要代码如下：

```
<Grid x:Name="LayoutRoot" Background="Transparent">
    <Button Content="按钮" Height="72" Width="300"></Button>
```

```
    <Button Name="button1" Margin="93,117,185,342" Height="150"
        Width="150" Click="button1_click">
        <StackPanel>
            <Image Source="23.png" Height="120" Width="120" Stretch="None"/>
        </StackPanel>
    </Button>
</Grid>
```

ButtonMain. xaml. cs 文件主要代码如下：

```
private void button1_click(object sender, RoutedEventArgs e)
{
    button1.Content="你单击我啦!";
}
```

运行模拟器，效果如图 4-3 所示。

图 4-3　Button 应用示例

4.5　RadioButton 控件

RadioButton(单选按钮)控件是表示一个单选的按钮。可以通过将 RadioButton 控件放到父控件或者为每个 RadioButton 设置 GroupName 属性来对 RadioButton 进行分组。当 RadioButton 元素分在一组中时，按钮之间互相排斥，用户一次只能选择 RadioButton 组中的一项。

下面给出应用 RadioButton 控件的示例代码。

源代码：chapter4\chapter4-5。

RadioButtonMain. xaml 文件主要代码如下：

```
<Grid x:Name="LayoutRoot" Background="Transparent">
```

```
    <Grid x:Name="ContentPanel" Grid.Row="1">
        <TextBlock Text="请选择您喜欢的景点:"
            Margin="16,100,0,0"></TextBlock>
        <RadioButton Margin="6,130,4,560" GroupName="MCSites" Content="华山"
            Click="RadioButton_Click" Name="a"/>
        <RadioButton Margin="6,220,4,470" GroupName="MCSites" Content="泰山"
            Click="RadioButton_Click" Name="b" />
        <RadioButton Margin="6,310,4,380" GroupName="MCSites" Content="衡山"
            Click="RadioButton_Click" Name="c" />
        <RadioButton Margin="6,400,1,290" GroupName="MCSites" Content="嵩山"
            Click="RadioButton_Click" Name="d" />
        <TextBlock Name="txtBlock1" Margin="16,490,1,200"></TextBlock>
    </Grid>
</Grid>
```

RadioButtonMain. xaml. cs 文件主要代码如下：

```
private void RadioButton_Click(object sender, RoutedEventArgs e)
{
        string stre="您选择的景点是: ";
        if (a.IsChecked==true)
            txtBlock1.Text=stre+a.Content.ToString();
        else if (b.IsChecked==true)
            txtBlock1.Text=stre+b.Content.ToString();
        else if (c.IsChecked==true)
            txtBlock1.Text=stre+c.Content.ToString();
        else
            txtBlock1.Text=stre+d.Content.ToString();
}
```

运行模拟器，效果如图 4-4 所示。

图 4-4　RadioButton 应用示例

4.6 CheckBox 控件

CheckBox(复选框)控件,表示用户可以选中或不选中的控件。

下面给出应用 CheckBox 控件的示例代码。

源代码:chapter4\chapter4-6。

CheckBoxMain.xaml 文件主要代码如下:

```
<Grid x:Name="LayoutRoot" Background="Transparent">
    <CheckBox Margin="6,120,0,580" Content="恒山" Name="CheckBox1"
        Checked="CheckBox1_Checked"/>
    <CheckBox Margin="6,180,0,520" Content="嵩山" Name="CheckBox2"
        Checked="CheckBox2_Checked" Unchecked="CheckBox2_UnChecked" />
</Grid>
```

CheckBoxMain.xaml.cs 文件主要代码如下:

```
private void CheckBox1_Checked(object sender, RoutedEventArgs e)
{
    CheckBox1.Content="选中 1";
}
private void CheckBox2_Checked(object sender, RoutedEventArgs e)
{
    CheckBox2.Content="选中 2";
}
private void CheckBox2_UnChecked(object sender, RoutedEventArgs e)
{
    CheckBox2.Content="放弃选中 2";
}
```

运行模拟器,效果如图 4-5 所示。

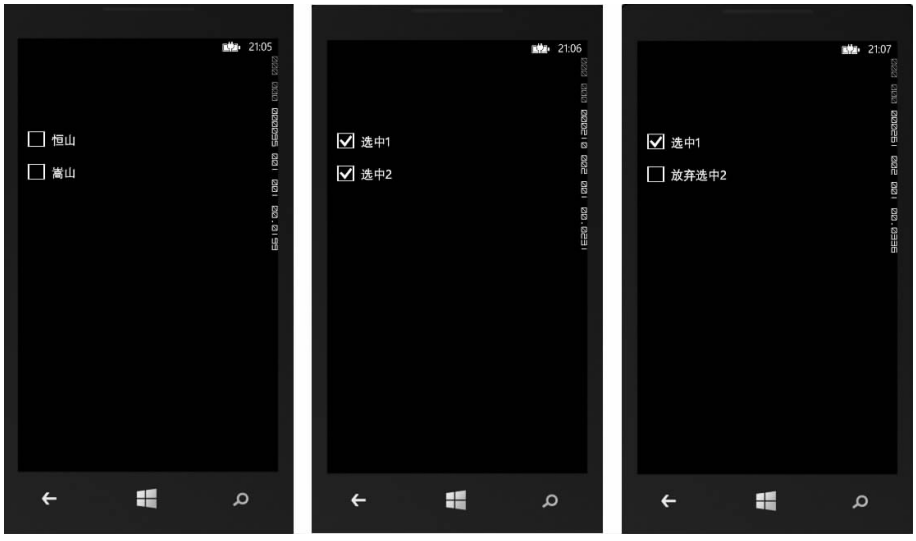

图 4-5 CheckBox 应用示例

4.7 ListBox 控件

ListBox(列表框)控件是指包含可选项列表,一次可以显示 ListBox 中的多个项。

ListBox 类常用的属性如下。

- SelectedItems:获取 ListBox 控件的当前选定项的列表。
- SelectionMode:获取或设置 ListBox 控件的选择行为。
- DataContext:获取或设置 FrameworkElement 参与数据绑定时的数据上下文(继承自 FrameworkElement)。
- Items:获取用于生成控件内容的集合(继承自 ItemsControl)。
- ItemsPanel:获取或设置模版,它定义了控制项的布局的面板(继承自 ItemsControl)。
- ItemsSource:获取或设置用于生成 ItemsControl 的内容的集合(继承自 ItemsControl)。
- ItemTemplate:获取或设置用于显示每个项的 DataTemplate(继承自 ItemsControl)。

下面给出应用 ListBox 控件的示例代码。

源代码:chapter4\chapter4-7。

Order.cs 文件主要代码如下:

```
public class Order
{
    public String OrderNumber { get; set; }
    public String TicketName { get; set; }
    public String TicketType { get; set; }
    public decimal Price { get; set; }

    public Order(String orderNumber, String ticketName, String ticketType,
    decimal price)
    {
        this.OrderNumber=orderNumber;
        this.TicketName=ticketName;
        this.TicketType=ticketType;
        this.Price=price;
    }
}
```

Orders.cs 文件主要代码如下:

```
public class Orders:ObservableCollection<Order>
{
    public Orders()
    {
        Add(new Order("2012001","华山","学生票",250));
        Add(new Order("2012002", "水晶宫", "成人票", 200));
        //Add(new Order("2012001", "华清池", "儿童票", 100));
```

```
    }
}
```

ListBoxMain. xaml 文件主要代码如下：

```
xmlns:my="clr-namespace:ListBoxApp"

<Grid x:Name="LayoutRoot" Background="Transparent">
    <Grid.Resources>
        <my:Orders x:Key="orders"/>
    </Grid.Resources>
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>
    <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
      <TextBlock x:Name="PageTitle" Text="我的订单" Margin="9,-7,0,0"
          Style="{StaticResource PhoneTextTitle1Style}"/>
    </StackPanel>
    <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
        <ListBox ItemsSource="{StaticResource orders}" Margin="0,5,-12,10">
            <ListBox.ItemTemplate>
                <DataTemplate>
                    <StackPanel Orientation="Vertical">
                        <TextBlock Text="订单号：" Foreground="Blue"/>
                        <TextBlock Padding="5,0,5,0" Text="{Binding
                            OrderNumber}" />
                        <TextBlock Text="门票类型：" Foreground="Blue"/>
                        <TextBlock Text="{Binding TicketType}" />
                        <TextBlock Text="门票名称：" Foreground="Blue" />
                        <TextBlock Text="{Binding TicketName}" />
                        <TextBlock Text="单价：" Foreground="Blue" />
                        <TextBlock Text="{Binding Price}" />
                        <TextBlock Text="----------------------------" />
                    </StackPanel>
                </DataTemplate>
            </ListBox.ItemTemplate>
        </ListBox>
    </Grid>
</Grid>
```

运行模拟器，效果如图 4-6 所示。