第 3 章 页面导航

通过上一章的学习，读者已经了解到应用程序的基本结构，以及如何对应用程序生命周期内各个阶段进行处理，本章将向读者讲述与页面导航相关的内容。

与网站应用程序相似，UAP 应用程序也会划分为多个页面，不同的页面将向用户呈现不同的内容。例如，一个闹钟应用程序可能存在以下页面：浏览页面用于列出用户已经创建的闹钟清单以及每条闹钟信息的状态；编辑页面可用于填写新闹钟信息或修改现有闹钟的信息；设置页面允许用户对应用程序进行个性化选择，如自定义闹钟铃声等。

可见，合理使用页面导航，能够增强应用程序的层次性和逻辑表现能力。本章内容包括：

- 使用 Frame 类实现页面导航；
- 导航并传递参数；
- 管理导航记录；
- 处理手机上的“回退”按钮事件；
- 理解页面缓存；
- 管理页面的导航状态。

3.1 实现导航

Frame 类派生自 ContentControl 类，这说明它是一个内容控件。与其他内容控件不同的是，Frame 具有导航功能，因此，Frame 类的内容属性所承载的通常是应用程序页面的实例，而实现页面导航是通过调用 Navigate 方法来完成的，该方法有多个重载，比较常用的有以下两个重载版本：

```
public bool Navigate(System.Type sourcePageType);
public bool Navigate(System.Type sourcePageType, object parameter);
```

其中，sourcePageType 参数是必须的，类型为 Type，即要导航的目标页面的类型。第二个重载版本有一个 parameter 参数，可为任意类型，表示导航参数，也就是在导航到目标

页面时传递的附加信息，目标页面可以根据 parameter 参数来做出相应处理，本章后面会向读者详细说明。

每个应用程序页面由 Page 类表示，但通常不会直接使用 Page 类，在向应用程序项目添加新页面时都会从 Page 类派生出一个子类，即自定义页面类，Navigate 方法中的 sourcePageType 参数则指定为自定义页面类的 Type。如果导航顺利完成，Navigate 方法返回 True，否则返回 False。如果导航失败，会引发 NavigationFailed 事件。

下面示例将演示如何通过 Frame 对象来实现页面导航。

应用程序项目包含两个页面：FirstPage 和 SecondPage。应用程序运行后将导航到 FirstPage 页，接着在 FirstPage 页上单击页面下方的命令按钮进入 SecondPage 页。

首先将 App 类的代码做如下修改：

```
public sealed partial class App : Application
{
    public App ()
    {
        this.InitializeComponent();
    }

    /// <summary>
    /// 用于导航的 Frame 对象
    /// </summary>
    public Frame RootFrame { get; private set; }

    protected override void OnLaunched ( LaunchActivatedEventArgs args )
    {
        // 实例化 Frame 对象
        RootFrame = new Frame();
        // 将当前 Frame 作为窗口的内容
        Window.Current.Content = RootFrame;
        // 导航到页面一
        RootFrame.Navigate(typeof(FirstPage));
        // 激活当前窗口
        Window.Current.Activate();
    }
}
```

代码先实例化一个 Frame 对象，然后将它作为当前窗口的内容，紧接着调用 Navigate 方法导航到 FirstPage 页。

在 FirstPage 中，通过处理命令按钮事件导航到 SecondPage 页，代码如下：

```
private void OnNext ( object sender, RoutedEventArgs e )
{
    this.Frame.Navigate(typeof(SecondPage));
}
```

由于 Page 类公开了一个 Frame 属性,可以获取负责导航的 Frame 实例的引用,因此上面代码通过 Frame 属性得到参与导航的 Frame 对象,然后调用其 Navigate 方法导航到 SecondPage 页。

应用程序的运行结果如图 3-1 所示。

图 3-1 使用 Frame 类进行页面导航

完整的示例代码请参考\第 3 章\Example_1。

3.2 导航事件

在页面导航过程中,应用程序会引发一系列事件,在需要的时候,开发者应该处理这些事件。

对 Frame 而言,在导航到目标页面之前会引发 Navigating 事件,此时可以做必要的检查,如果希望取消导航,可以设置事件参数 NavigatingCancelEventArgs 的 Cancel 属性为 True 来阻止导航;当 Frame 对象顺利导航到目标页面后,会发生 Navigated 事件。通过事件参数对象的 SourcePageType 属性可以获得目标页面的类型,即传递给 Navigate 方法的 sourcePageType 参数的值。

对于页面来说,当导航进入页面后,会调用 OnNavigatedTo 方法;当导航即将离开页面时,OnNavigatingFrom 方法会被调用,同样,可以通过设置方法参数 e 的 Cancel 属性为 True 来取消导航;当导航已经离开当前页面后会调用 OnNavigatedFrom 方法。以上三个方法都是虚方法,开发者在派生类中可以重写它们并加入自定义处理代码。

那么,如何知道上面所述的几个事件的发生顺序呢?接下来将通过一个示例来向读者展示各个导航事件的引发顺序。

在应用程序项目中添加三个页面,分别命名为 Page1、Page2 和 Page3。三个页面的结构相似,所以此处只介绍 Page1 页面的结构,其描述界面布局的 XAML 代码如下:

```
<Grid>
    <TextBlock Text = "Page 1" FontSize = "50"
               VerticalAlignment = "Center"
               HorizontalAlignment = "Center"/>
</Grid>
```

页面中仅用一个 TextBlock 元素来显示标识文本，以方便在运行阶段进行观察。

本示例程序主窗口的内容对象并非 Frame 实例，而是 MainPage 页面，然后将负责导航的 Frame 对象放置在 MainPage 页面中。MainPage 页面的 XAML 如下：

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height = "Auto"/>
        <RowDefinition/>
    </Grid.RowDefinitions>
    <ListView Grid.Row = "0" Margin = "2,18,2,5" ItemClick = "OnItemClick" IsItemClickEnabled
 = "True">
        <ListViewItem Grid.Column = "0"> Page 1 </ListViewItem>
        <ListViewItem Grid.Column = "1"> Page 2 </ListViewItem>
        <ListViewItem Grid.Column = "2"> Page 3 </ListViewItem>
        <ListView.ItemsPanel>
            <ItemsPanelTemplate>
                <Grid>
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition/>
                        <ColumnDefinition/>
                        <ColumnDefinition/>
                    </Grid.ColumnDefinitions>
                </Grid>
            </ItemsPanelTemplate>
        </ListView.ItemsPanel>
        <ListView.ItemContainerStyle>
            <Style TargetType = "ListViewItem">
                <Setter Property = "Height" Value = "38"/>
                <Setter Property = "FontSize" Value = "17"/>
                <Setter Property = "Padding" Value = "17,2"/>
                <Setter Property = "Background" Value = "DarkBlue"/>
            </Style>
        </ListView.ItemContainerStyle>
    </ListView>

    <Frame Grid.Row = "1" x: Name = "frame" Background = "Transparent" Margin = "5"
           Navigating = "OnFrameNavigating"
           Navigated = "OnFrameNavigated"></Frame>
</Grid>
```

通过 ListView 控件中的三个子项来控制 Page1、Page2 和 Page3 三个页面的导航。处

理 Frame 对象的 Navigating 和 Navigated 事件,并在事件处理代码中通过 Debug 类输出调试信息。

```
private void OnFrameNavigating ( object sender, NavigatingCancelEventArgs e )
{
    string msg = "Frame 的 Navigating 事件发生,SourcePageType = {0}";
    System.Diagnostics.Debug.WriteLine(msg, e.SourcePageType);
}

private void OnFrameNavigated ( object sender, NavigationEventArgs e )
{
    string msg = "Frame 的 Navigated 事件发生,SourcePageType = {0}";
    System.Diagnostics.Debug.WriteLine(msg, e.SourcePageType);
}
```

然后打开 Page1、Page2 和 Page3 三个页面的代码文件,分别重写它们的 OnNavigatedTo、OnNavigatingFrom 及 OnNavigatedFrom 方法,同样使用 Debug 类来输出调试信息。下面代码以 Page1 为例,其他两个页面相似。

```
protected override void OnNavigatedTo ( NavigationEventArgs e )
{
    Debug.WriteLine("Page 1: OnNavigatedTo 方法被调用.");
}

protected override void OnNavigatingFrom ( NavigatingCancelEventArgs e )
{
    Debug.WriteLine("Page 1: OnNavigatingFrom 方法被调用.");
}

protected override void OnNavigatedFrom ( NavigationEventArgs e )
{
    Debug.WriteLine("Page 1: OnNavigatedFrom 方法被调用.");
}
```

最后,打开 App 类的代码文件,将代码修改为:

```
public sealed partial class App : Application
{
    public App()
    {
        this.InitializeComponent();
    }

    protected override void OnLaunched(LaunchActivatedEventArgs e)
    {
        // 实例化 MainPage 类
```

```
            MainPage main = new MainPage();
            // 直接将 MainPage 页作为主窗口的内容对象
            Window.Current.Content = main;
            // 确保当前窗口处于活动状态
            Window.Current.Activate();
        }
    }
```

运行应用程序，如果未看见“输出”窗口，可以在“快速启动”中输入“输出”，并在下拉列表中选择“视图”→“输出”，或者“调试”→“窗口”→“输出”命令调出“输出”窗口。

单击页面顶部的“Page 1”子项，Frame 对象便导航到 Page1 页面，“输出”窗口输出的调试如下：

```
Frame 的 Navigating 事件发生，SourcePageType = Example_2.Page1
Frame 的 Navigated 事件发生，SourcePageType = Example_2.Page1
Page 1: OnNavigatedTo 方法被调用。
```

从上面信息可以看到，Frame 类的 Navigating 事件首先被触发，然后发生 Navigated 事件，当导航进入 Page1 页面后，页面实例的 OnNavigatedTo 方法被调用。

接下来，单击应用程序页面上的“Page 2”，从 Page1 页导航到 Page2 页，输出的调试信息如下：

```
Frame 的 Navigating 事件发生，SourcePageType = Example_2.Page2
Page 1: OnNavigatingFrom 方法被调用。
Frame 的 Navigated 事件发生，SourcePageType = Example_2.Page2
Page 1: OnNavigatedFrom 方法被调用。
Page 2: OnNavigatedTo 方法被调用。
```

因为此次是从 Page1 页向 Page2 页导航，故 Page1 页的 OnNavigatingFrom 方法被调用，然后引发 Frame 对象的 Navigated 事件，离开 Page1 页面后会调用 OnNavigatedFrom 方法。当导航进入 Page2 后，就会调用 Page2 实例的 OnNavigatedTo 方法。

读者只需一边切换页面一边观察“输出”窗口中的内容，多重复几遍就会找到规律了。也可以通过图 3-2 来模拟各个导航事件的引发顺序。

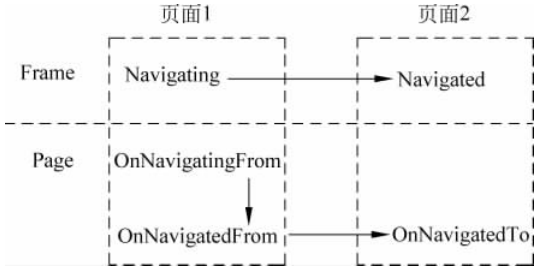

图 3-2　导航事件发生顺序示意图

完整的示例代码请参考\第 3 章\Example_2。

3.3 传递导航参数

前面提到过,Navigate 方法提供以下重载版本:

```
public bool Navigate(System.Type sourcePageType, object parameter);
```

其中,parameter 参数可以传递任意对象的引用,当导航到目标页面后,可以在处理 OnNavigatedTo 方法时,通过方法参数的 Parameter 属性来获取传递的对象。

下面示例将帮助读者理解如何在导航时传递参数。示例应用程序包含两个页面,第一个页面允许用户输入员工信息,输入完成后,单击页面下方的命令按钮,跳转到第二个页面,并把输入的员工信息传递给第二个页面并显示出来。

用于输入员工信息的页面的 XAML 如下:

```
<Grid>
    …
    <TextBlock Text="输入员工信息" Style="{StaticResource HeaderTextBlockStyle}" Margin
="0,6,0,12"/>

    <StackPanel Grid.Row="1" Margin="5">
        <TextBox x:Name="txtID" Header="员工编号:"/>
        <TextBox x:Name="txtName" Header="员工姓名:"/>
        <TextBox x:Name="txtAge" Header="员工年龄:">
            <TextBox.InputScope>
                <InputScope>
                    <InputScope.Names>
                        <InputScopeName NameValue="Number"/>
                    </InputScope.Names>
                </InputScope>
            </TextBox.InputScope>
        </TextBox>
    </StackPanel>
</Grid>

<Page.BottomAppBar>
    <AppBar>
        <AppBarButton Icon="Accept" Label="确定" Click="OnClick"/>
    </AppBar>
</Page.BottomAppBar>
```

BottomAppBar 属性表示页面底部的命令按钮栏,可以在其中放置带小图标和文本的按钮。在本例中,当用户单击“确定”命令按钮后导航到显示员工信息的页面,并向目标页面

传递员工信息，代码如下：

```
private void OnClick ( object sender, RoutedEventArgs e )
{
    if (txtID.Text == "" || txtName.Text == "" || txtAge.Text == "")
    {
        return;
    }
    // 收集输入信息
    Dictionary<string, string> data = new Dictionary<string, string>();
    data.Add("id", txtID.Text);
    data.Add("name", txtName.Text);
    data.Add("age", txtAge.Text);
    // 导航并传递参数
    this.Frame.Navigate(typeof(DisplayPage), data);
}
```

上面代码中，使用 Dictionary<string, string>（字典）对象作为参数传递给 Navigate 方法，导航到目标页面后，即可以在 OnNavigatedTo 方法中获取该数据对象。

描述显示员工信息页面的 XAML 代码如下：

```
<Grid>
    …
    <TextBlock Text="显示员工信息" Style="{StaticResource HeaderTextBlockStyle}" Margin
="0,8,0,13"/>

    <Grid Grid.Row="1" Margin="6">
        …
        <TextBlock Text="员工编号:" Style="{StaticResource tbdisplayStyle}" />
        <TextBlock Text="员工姓名:" Style="{StaticResource tbdisplayStyle}" Grid.Row="
1"/>
        <TextBlock Text="员工年龄:" Style="{StaticResource tbdisplayStyle}" Grid.Row="
2"/>
          <TextBlock x:Name="tbID" Grid.Column="1" Style="{StaticResource
tbdisplayStyle}"/>
          <TextBlock x:Name="tbName" Grid.Row="1" Grid.Column="1" Style="
{StaticResource tbdisplayStyle}"/>
        <TextBlock x:Name="tbAge" Grid.Column="1" Grid.Row="2" Style="{StaticResource
tbdisplayStyle}"/>
    </Grid>
</Grid>
```

前三个 TextBlock 控件用于显示标识文本，后三个 TextBlock 控件用来显示页面从导航结果中接收的员工信息数据。处理代码如下：

```
protected override void OnNavigatedTo ( NavigationEventArgs e )
{
    if (e.Parameter != null && e.Parameter is Dictionary<string, string>)
    {
        // 获取导航参数
        Dictionary<string, string> getdata = (Dictionary<string, string>)e.Parameter;
        // 显示内容
        tbID.Text = getdata["id"];
        tbName.Text = getdata["name"];
        tbAge.Text = getdata["age"];
    }
}
```

由于导航到显示员工信息页面时传递的数据是 Dictionary<string, string>类型，因此在取出数据时也要转化为匹配的类型才能读取到正确的数据。

示例运行结果如图 3-3 所示。

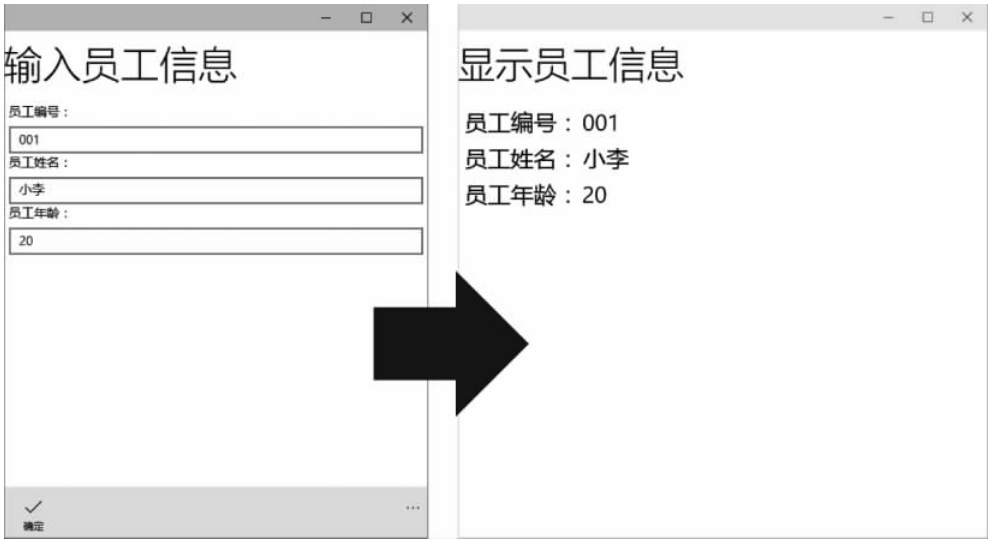

图 3-3 传递导航参数到目标页面

完整的示例代码请参考\第 3 章\Example_3。

3.4 管理导航记录

与网页浏览器相似，Frame 类在完成页面导航后会动态维护两个导航记录堆栈：一个是 BackStack，包含可以向后回退的页面记录；另一个是 ForwardStack，包含可以向前导航的页面记录。可以用图 3-4 来模拟导航记录堆栈，图中以页面 C 为参照项，页面 A、B 为向

后导航的记录项，页面 D、E 则为向前导航的记录项。

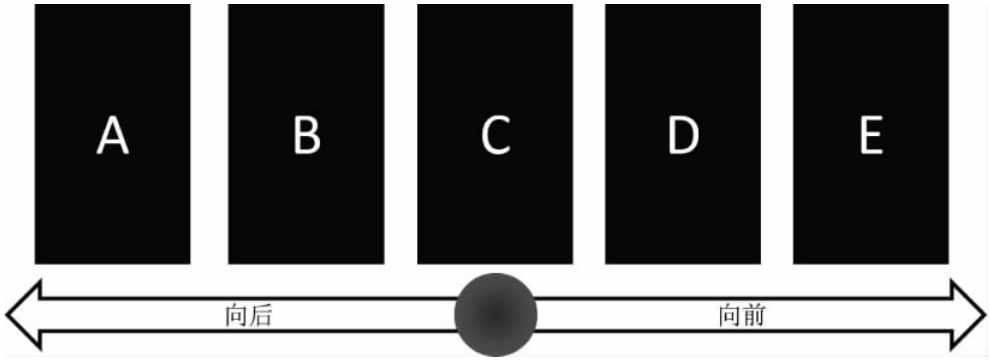

图 3-4　导航记录示意图

不管是向后堆栈还是向前堆栈，每一条页面导航记录都用 PageStackEntry 类（位于 Windows. UI. Xaml. Navigation 命名空间）来表示，该类的 SourcePageType 属性会返回一个 Type 信息，表示当前记录中页面的类型。

通常，开发者并不关心导航堆栈中的导航记录，仅仅是通过 Frame 类的 GoBack 方法进行向后导航，或者调用 GoForward 方法向前导航。在调用 GoBack 方法之前应当检查一下 CanGoBack 属性是否为 True，如果属性返回 False，表明不能向后导航。同理，在使用 GoForward 方法之前也应当检查一下 CanGoForward 属性是否允许向前导航。

下面示例将演示如何向前/向后导航。示例项目包括三个页面，分别命名为 Page_A、Page_B、Page_C，程序运行后默认进入 Page_A 页。三个页面的结构相同，此处以 Page_A 页面为例进行讲述，界面 XAML 如下：

```
<StackPanel>
    <TextBlock  Text="Page A" HorizontalAlignment="Center" FontSize="60"/>
    <TextBlock Name="tbInfo" FontSize="28"/>
</StackPanel>

<Page.BottomAppBar>
    <AppBar>
        <StackPanel Orientation="Horizontal">
            <AppBarButton Click="OnBack" Label="上一页" Icon="Back"/>
            <AppBarButton Click="OnForward" Label="下一页" Icon="Forward"/>
        </StackPanel>
    </AppBar>
</Page.BottomAppBar>
```

页面底部的命令栏中的两个 AppBarButton 按钮用于执行向后或向前导航操作，实现代码如下：

```
private void OnBack ( object sender, RoutedEventArgs e )
{
    // 如果可以后退,便后退一页
    if (this.Frame.CanGoBack)
    {
        this.Frame.GoBack();
    }
}

private void OnForward ( object sender, RoutedEventArgs e )
{
    // 如果可以向前,则往前一页
    if (this.Frame.CanGoForward)
    {
        this.Frame.GoForward();
    }
    else
    {
        this.Frame.Navigate(typeof(Page_B));
    }
}
```

页面类的 Frame 将引用负责导航的 Frame 控件实例,并调用其 GoBack 和 GoForward 方法来向后或向前导航。

当导航进入页面后会执行 OnNavigatedTo 方法,分析 NavigationEventArgs 参数的 NavigationMode 属性可以知道页面的导航模式,由 NavigationMode 枚举来描述。如果在应用程序运行后首次导航到页面,或直接调用 Frame 类的 Navigate 方法完成导航,那么 NavigationMode 的值为 New,即全新导航;如果导航是通过回退记录后退到当前页面的,NavigationMode 的值为 Back;如果是通过前进记录向前导航到当前页面,则 NavigationMode 的值为 Forward。为了让读者能够看到 NavigationMode 的值,在页面的 OnNavigatedTo 方法中加入以下代码:

```
protected override void OnNavigatedTo ( NavigationEventArgs e )
{
    this.tbInfo.Text = "导航模式: " + e.NavigationMode.ToString();
}
```

如此一来,NavigationMode 的值会显示在页面上。

另外,为了让读者能够实时看到 Frame 中 BackStack 堆栈和 ForwardStack 堆栈中的导航记录的变化,项目还使用了一个用户控件,里面包含两个 ListView 控件,分别用于显示 BackStack 和 ForwardStack 中的记录项,其主要的 XAML 代码如下:

```
<Grid Background = "LightGray" x: Name = "root">
    …
    <TextBlock Text = "回退记录" Style = "{StaticResource center}"/>
    <TextBlock Text = "前进记录" Style = "{StaticResource center}" Grid.Column = "2"/>

    <ListView x: Name = "lvBack"    Grid.Row = "1"/>
    <ListView x: Name = "lvForward"    Grid.Row = "1" Grid.Column = "2"/>

    <Rectangle Fill = "LightGray" Width = "3"
               Grid.Row = "0" Grid.RowSpan = "2" Grid.Column = "1"/>
</Grid>
```

当 Frame 对象的 Navigated 事件发生时，将导航堆栈中的记录分别添加到两个 ListView 控件中。

```
void m_frame_Navigated ( object sender, NavigationEventArgs e )
{
    lvBack.Items.Clear();
    foreach (PageStackEntry item in m_frame.BackStack)
    {
        lvBack.Items.Add(item.SourcePageType.Name);
    }
    lvForward.Items.Clear();
    foreach (PageStackEntry item in m_frame.ForwardStack)
    {
        lvForward.Items.Add(item.SourcePageType.Name);
    }
}
```

在 App 类中定义以下方法，用于显示导航记录的用户控件放到一个 Popup 控件中，并弹出到屏幕上。

```
public void ShowNavStack ( Frame frame )
{
    PopupControl pc = new PopupControl(frame);
    Popup pop = new Popup();
    pop.Child = pc;
    pc.Height = 300;
    pc.Width = 600;
    pop.VerticalOffset = Window.Current.CoreWindow.Bounds.Height - pc.Height - 60;
    pop.IsOpen = true;
}
```

设置 IsOpen 为 True 后 Popup 控件就会再弹出，Popup 总是在应用程序窗口的前面，

不会被页面挡住,也不受页面之间导航的影响,这样做能够更好地观看导航记录的变化。随后在 OnLaunched 中调用该方法。

```
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    …
    ShowNavStack(rootFrame);

    // 确保当前窗口处于活动状态
    Window.Current.Activate();
}
```

运行示例程序后,首先进入 Page_A 页面,由于是首次导航到页面,因此导航模式为 New,此时向后和向前导航记录堆栈没有任何记录,如图 3-5 所示。

图 3-5 未产生导航记录

当离开 Page_A 页面,导航到 Page_B 页,再从 Page_B 页导航到 Page_C 页面,当进入 Page_C 页面后,Page_A 和 Page_B 页两个页面就被放到回退堆栈中,此时 Frame 对象如果向后导航,会回到 Page_B 页,再向后就回到 Page_A 页面。每调用一次 GoBack 方法只能后退一条记录,要回退两条记录,则需要调用两次 GoBack 方法。如图 3-6 所示。

当从 Page_C 中执行 GoBack 操作后,导航回退到 Page_B 页面,因为是通过向后导航到 Page_B 页面的,故导航模式为 Back。此时,后退堆栈只剩下 Page_A 页面了,同时前进堆栈中记录了 Page_C 页面。如图 3-7 所示。

有时候,开发者希望删除一条或多条导航记录,例如从 Page_A 导航到 Page_B,再导航到 Page_C,但是从 Page_C 返回时,用户不希望返回 Page_B,而是直接回到 Page_A。在 Page_C 页的 OnNavigatedTo 方法中加入以下代码:

图 3-6　已产生回退记录

图 3-7　产生向前导航记录

```
// 查找与页面 B 相关的记录
var r = this.Frame.BackStack.FirstOrDefault(p => p.SourcePageType == typeof(Page_B));
// 删除一条回退记录
if (r != null)
{
    Frame.BackStack.Remove(r);
}
```

修改后,当导航进入 Page_C 页后将回退堆栈中与 Page_B 页相关的记录删除,当从 Page_C 向后导航时,就会跳过 Page_B 页而直接返回到 Page_A 页。

有关本示例的完整代码请参考\第 3 章\Example_4。

3.5 处理手机上的“后退”键

在 PC 和平板电脑上是没有“后退”键的,但是,手机设备上有一个“后退”键,因此在默认情况下,只要点击手机上的“后退”键就会返回到“开始”屏幕,而不管 Frame 中有没有导航历史记录,而用户更习惯于在应用程序中一页一页地后退,直到后退的导航记录数为 0 时才回到“开始”屏幕。

要改变“回退”按键的默认行为,开发者需要处理相应的物理按键事件。在 Windows.Phone.UI.Input 命名空间下,有一个名为 HardwareButtons 的静态类,该类包含四个事件,当手机上的物理按键被按下时,开发者可以进行一些自定义处理。有关这四个事件的说明,请参考表 3-1。

表 3-1 HardwareButtons 类的事件列表

事 件	说 明
BackPressed	当手机上的“后退”键被按下时发生
CameraHalfPressed	当手机上的摄像头键处于半按下状态时发生(拍照前进行对焦,通常只按下一半)
CameraPressed	当摄像头键被完全按下时发生
CameraReleased	当用户松开摄像头键时发生

要实现在用户按下“后退”键时进行额外处理,程序代码应该响应 BackPressed 事件。不过,Windows.Phone.UI.Input 命名空间下的类型都位于 PhoneContract 下的 API 子集中,属于 Extension SDKs 子集,因此在使用 HardwareButtons 类之前要在项目中引用 Extensoin SDKs 子集,而后在代码中还需要使用 ApiInformation 类的 IsTypePresent 方法进行 API 可用性的检查。这是因为 UAP 应用程序是通用于所有设备平台的,如果应用程序运行在 PC 上,则 HardwareButtons 类是不可见的,只有当应用程序在手机设备上时才可以访问,为了避免发生异常,必须进行 API 有效性的检查。

下面将通过一个简单的示例来帮助读者理解如何处理 BackPressed 事件。如图 3-8 所示,示例程序包含三个页面,第一个页面为应用主页,页面上有两个链接,点击“我的音乐”链接后会进入“我的音乐”页面,点击“我的视频”链接后会进入“我的视频”页面。而当用户按下手机上的“后退”键时,应用程序会先退回到应用主页,再次按“后退”键就回到“开始”屏幕。

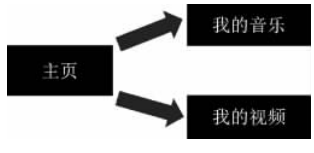

图 3-8 示例应用的页面结构

打开“解决方案资源管理器”窗口,右击引用节点,添加新引用。在“引用管理”窗口的左侧选择 Universal App Platform→Extensions 节点,并在右边的列表中勾选 Windows Mobile Extension SDK 项,最后单击“确定”按钮完成引用添加。如图 3-9 所示。

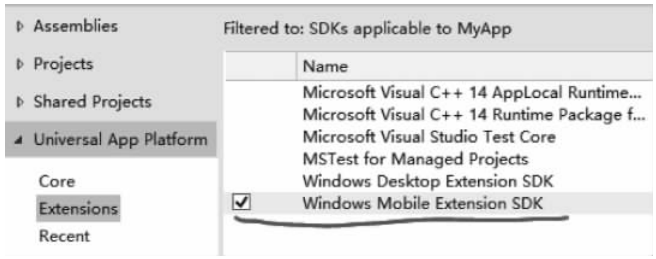

图 3-9　添加 Windows Mobile Extension SDK 引用

在 App 类的 OnLaunch 方法中加入以下代码：

```
if (Windows.Foundation.Metadata.ApiInformation.IsTypePresent("Windows.Phone.UI.Input.
HardwareButtons"))
{
    Windows.Phone.UI.Input.HardwareButtons.BackPressed += OnBackPressed;
}
```

处理 BackPressed 事件的方法名为 OnBackPressed，实现代码如下：

```
void OnBackPressed ( object sender, Windows.Phone.UI.Input.BackPressedEventArgs e )
{
    // 获取 Frame 对象实例
    Frame rootFrame = Window.Current.Content as Frame;
    if (rootFrame != null)
    {
        // 如果 Frame 对象可以进行后退操作
        // 则应该阻止事件传播到系统
        // 并调用 GoBack 方法
        if (rootFrame.CanGoBack)
        {
            e.Handled = true;
            rootFrame.GoBack();
        }
    }
}
```

项目模板生成的默认代码是将 Frame 实例作为当前窗口的内容数据的，因此，通过 Window.Current.Content 可以获取到 Frame 实例，但 Content 属性是 Object 类型，所以要使用 as 关键字进行引用转换。接着，通过 Frame 实例的 CanGoBack 属性判断导航是否还能后退，如果可以，则将事件参数的 Handled 属性设置为 True，可以阻止 BackPressed 事件继续向下传播，操作系统接收不到该按键的消息，就不会执行默认行为，即不会立刻返回“开始”屏幕。随后还要调用 GoBack 方法使 Frame 对象向后导航。

运行应用程序,在主页上点击“我的音乐”链接,此时会进入“我的音乐”页面,然后点击手机上的“后退”键,便不再返回“开始”屏幕,而是回到主页;再次点击“后退”按键,由于导航的后退堆栈已无记录,此时就会返回“开始”屏幕。如图 3-10 所示。

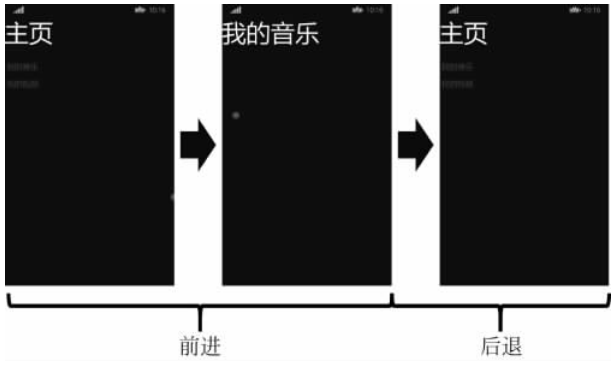

图 3-10 自定义处理 BackPressed 事件

完整的示例代码请参考\第 3 章\Example_5。

3.6 缓存页面实例

Frame 类的 CacheSize 属性表示在应用程序周期内应该缓存的页面的实例数。一定要注意,这里所说的页面缓存不是导航记录,很多初学者容易混淆,页面缓存是指在应用程序运行期间,页面的实例是否被持久保持。如果页面没有被缓存,则每次导航到页面时都会调用页面类的构造函数重新实例化页面类;如果页面已缓存,则当导航到页面时不会重新实例化,而是使用现有的页面实例。

可能很多读者并没有注意到,每次导航进入页面都会调用页面类的构造函数,不妨通过示例来验证一下。

向应用程序项目添加一个新页面,可以随意命名,本例将其命名为 NewPage。在新页面中输入以下 XAML 代码:

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition/>
    </Grid.RowDefinitions>
    <TextBlock Text="第二页" Style="{StaticResource HeaderTextBlockStyle}" />

    <Grid Grid.Row="1" Margin="3,21,3,5">
        <Grid.RowDefinitions>
```

```
            <RowDefinition Height = "Auto"/>
            <RowDefinition Height = "Auto"/>
        </Grid.RowDefinitions>
        <Button Content = "返回"
                HorizontalAlignment = "Left"
                VerticalAlignment = "Top"
                Margin = "10,15,0,0"
                Click = "OnBack"/>

        <TextBox Header = "请输入数字: " Name = "txtInput" Grid.Row = "1" Margin = "0,10,0,0">
            <TextBox.InputScope>
                <InputScope>
                    <InputScope.Names>
                        <InputScopeName NameValue = "Number"/>
                    </InputScope.Names>
                </InputScope>
            </TextBox.InputScope>
        </TextBox>
    </Grid>

</Grid>
```

Button 控件被单击后会向后导航回到主页面(MainPage),TextBox 控件可以用来验证构造函数是否被调用。思路是:在 NewPage 页的构造函数中将 TextBox 中的文本设置为“0”,当导航进入 NewPage 页面后,可以在 TextBox 中随机输入一些数字,然后退回主页面,当再次导航到 NewPage 后,如果 TextBox 中的文本又被初始化为“0”,这表明构造函数被调用,否则会保留前面输入的数字。

在 NewPage 类的构造函数中加入以下代码:

```
public NewPage ()
{
    this.InitializeComponent();
    txtInput.Text = "0";
}
```

运行示例应用程序,单击主页上的“转到第二页”按钮,导航到 NewPage 页,然后在 TextBox 中输入数字“5000”,如图 3-11 所示。

随后单击“返回”按钮回到主页,然后再单击“转到第二页”按钮再次导航到 NewPage 页,如图 3-12 所示,此时读者会看到,TextBox 中的文本又变回了“0”,这说明 NewPage 类的构造函数被调用,即 NewPage 页已重新实例化。

图 3-11　输入数字 5000

图 3-12　TextBox 中的文本又变回“0”

该试验表明在页面没有被缓存的情况下，每次导航到页面时都会重新实例化，接下来对示例程序进行修改，使其支持页面缓存。

在 App 类的 OnLaunch 方法中，将 CacheSize 改为 2，表示缓存两个页面实例(MainPage 页和 NewPage 页)。

```
rootFrame.CacheSize = 2;
```

修改 CacheSize 的值并不能使页面支持缓存，还必须将要进行缓存的页面类的 NavigationCacheMode 属性设置为 Enabled(启用缓存)。如果将页面的 NavigationCacheMode 属性设置为 Required，则表示强制缓存当前页面，即使已缓存的页面实例数量已经超出 CacheSize 的值也会将页面实例缓存；若设置为 Enabled 值，则当缓存的页面实例个数超出 CacheSize 属性的值后就不再缓存当前页面的实例；若设置为 Disabled 值，则表明当前页面禁用缓存。

回到 NewPage 类的构造函数，将页面的 NavigationCacheMode 属性设置为 Enabled。

```
public NewPage ()
{
    this.InitializeComponent();
    // 启用缓存
    this.NavigationCacheMode = Windows.UI.Xaml.Navigation.NavigationCacheMode.Enabled;
    txtInput.Text = "0";
}
```

现在，读者可以再次运行示例程序，转到 NewPage 页，然后在 TextBox 中输入数字 85000，然后返回到主页。接着再次转到 NewPage 页，可以看到，刚才输入的 85000 仍然保留，说明页面实例已被缓存，导航进入页面后使用的还是以前的实例。如图 3-13 所示。

图 3-13　前面输入的文本已被保留

完整的示例代码请参考\第 3 章\Example_6。

3.7 保存和恢复导航状态

有时候，开发者希望在用户关闭(或被操作系统结束)应用程序时将当前应用的导航状态保存起来，当用户下一次启动应用程序时，恢复导航状态，从而恢复用户上次关闭应用时的页面。

Frame 类公开了两个方法可以用于读写页面的导航状态。调用 GetNavigationState 方法可以得到 Frame 对象当前的导航状态，类型为字符串；当需要恢复导航状态时只需调用 SetNavigationState 方法，并把通过 GetNavigationState 方法得到的导航状态作为参数传递即可。

下面示例将演示如何保存和恢复导航状态。示例包含主页和另一个页面，应用程序首次运行时会默认进入主页，用户可以通过页面上的按钮进入第二个页面。当应用程序被挂起时，将当前 Frame 的导航状态保存到本地设置中。当用户再次运行应用时，从本地设置中读出导航状态并应用到 Frame 对象上，以恢复应用关闭前所处的页面。

在 App 类中处理 Suspending 事件的方法中加入以下代码，以便在应用程序被挂起时保存导航状态。

```
private void OnSuspending(object sender, SuspendingEventArgs e)
{
    var deferral = e.SuspendingOperation.GetDeferral();

    Frame rootFrame = Window.Current.Content as Frame;
    // 获取导航状态
    string navstate = rootFrame.GetNavigationState();
    // 在调试信息中输出该状态内容
    System.Diagnostics.Debug.WriteLine("导航状态: " + navstate);
    // 将导航状态保存到本地设置中
    var localSettings = ApplicationData.Current.LocalSettings;
    localSettings.Values["nav"] = navstate;

    deferral.Complete();
}
```

将 OnLaunched 方法改为以下代码：

```
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    Frame rootFrame = Window.Current.Content as Frame;

    if (rootFrame == null)
    {
        rootFrame = new Frame();
        rootFrame.Language = "zh-cn";      //设置默认语言
        Window.Current.Content = rootFrame;
```

```
            // 如果应用程序之前是被用户关闭的,或者被操作系统结束的
            // 就要考虑对导航状态进行恢复
              if (e.PreviousExecutionState == ApplicationExecutionState.Terminated || e.
    PreviousExecutionState == ApplicationExecutionState.ClosedByUser)
            {
                // 如果本地设置中包含导航状态,则进行恢复
                // 否则进入默认页面
                object value;
                var localSettings = ApplicationData.Current.LocalSettings;
                if (localSettings.Values.TryGetValue("nav", out value))
                {
                    rootFrame.SetNavigationState(value as string);
                }
                else
                {
                    rootFrame.Navigate(typeof(MainPage));
                }
            }
            else //否则直接导航到主页面
            {
                rootFrame.Navigate(typeof(MainPage));
            }
        }

        Window.Current.Activate();
    }
```

当用户通过常规操作关闭应用程序后,再次运行时 PreviousExecutionState 的值会变为 ClosedByUser,因此不仅要检查 Terminated,还要检查应用程序的前一个状态是否为 ClosedByUser。如果条件成立,就从本地设置中将导航状态读出来,并调用 SetNavigationState 方法来恢复导航状态。

运行应用程序,并单击页面上的按钮进入第二个页面,如图 3-14 所示。

此时关闭应用程序,然后重新启动应用程序,会发现应用程序会自动导航到第二个页面了,如图 3-15 所示。

图 3-14 准备进入第二个页面

图 3-15 再次启动应用程序

完整的示例代码请参考\第 3 章\Example_7。