Windows Phone Mango编程实践

***Windows Phone Mango Programming Practice***

第二篇 Silverlight交互篇

# 应用程序生命周期(Application Lifecycle)

正如孙子兵法中所讲：“故不尽知用兵之害者，则不能尽知用兵之利也。”同样的道理，虽然逻辑删除对于移动终端的设备性能可以得到提升，但是对于开发者而言，必须了解逻辑删除的负面作用，就是需要开发者手工编写代码存储应用程序的状态信息，而且要了解临时数据保存和永久数据保存的区别和方法。这样才能开发出用户粘度强的应用程序。

10秒钟原则正是软件开发中针对“用兵之害”所应采取的规避。故曰：知彼知己者，百战不殆；不知彼而知己，一胜一负；不知彼不知己，每战必殆。

代码之美的实践就是不断设计更优的算法，不断提升代码质量，不断创造改变世界的软件。

## 应用程序生命周期

### 概述

Windows Phone执行模型的设计初衷是提供一个快速反应的用户体验，为此，To achieve this, Windows Phone allows only one application to run in the foreground at any given time.Windows Phone仅仅允许在前台运行一个应用程序，即与用户进行交互的可见的当前运行的应用程序。如果应用程序不在前台运行，那么The operating system puts an application in a dormant state when it is no longer in the foreground.操作系统将其休眠。 If the device memory available for the foreground application is insufficient for a good user experience, then the operating system will begin to terminate dormant applications, with the least recently used applications terminated first.如果操作系统为前台应用程序提供系统资源不足，那么操作系统将终止被休眠的应用，终止的顺序按照休眠的先后顺序。 因此，应用程序必须在停用（deactivated）和激活（reactivated）时管理应用程序的运行状态，This helps to create a user experience in which applications appear to the user to maintain a single instance, even when they have been terminated and reactivated.这有助于为用户提供良好的使用体验。

Windows PhoneThe execution model also provides users with a consistent navigation experience between applications.执行模型还设计了应用程序之间的导航体验。On Windows Phone, users navigate forward by launching applications from the installed applications list or from a Tile on Start in addition to other means, such as tapping on a toast notification associated with an application.在Windows Phone，用户可以从软件安装列表中启动应用程序，也可以在启动屏幕（Start）通过Tile启动应用程序，比如手指触控Toast通知则启动与此关联的应用程序。在应用程序中提供的后退按钮可以返回前一页面，也可以通过硬件的后退按键（Back）实现向后导航。Windows Phone Mango还提供按住硬件的后退按键（Back）实现应用程序快速切换。

众所周知，Windows Phone的开发可以采用Silverlight和XNA框架，或者Silverlight和XNA混合模式。在开发XNA游戏时，操作系统对应用程序的中断有特殊的处理。具体的内容我们在本章的XNA Game Studio逻辑删除中讲解。

### 应用程序生命周期相关的术语

本章所涉及的应用程序生命周期相关的术语如下。

|  |  |
| --- | --- |
| **名称** | **详细说明** |
| 应用程序状态  (Application State) | 应用程序状态数据是在应用中的所有页面中使用的数据。 比如从Web服务获得的数据结构， 在每个页面上所显示的数据内容或者形式也许不同，但应用程序状态数据是属于应用程序级的数据，是应用程序整体中的一部分。 |
| 页面状态  (Page State) | 应用程序页面当前的可视状态。比如用户通过搜索键离开应用程序，而在应用程序的页面中包含用户输入的数据或者字符串，当用户通过后退键重新回到应用程序时，用户最期望的是页面显示的数据与离开前保持一致。应用程序保存页面状态，并在页面加载时恢复保存的数据，使用户体验到始终一致的应用程序。 |
| 应用程序事件  (Application Events) | 关于应用程序状态管理的四个主要事件：启动（Launching）、停用（Deactivated）、激活（Activated）和关闭（Closing）。这些事件的处理程序是包含在应用程序对象中，是作为Windows Phone应用程序的Visual Studio项目模板的一部分。开发者需要做的就是在事件处理函数中编写代码来管理应用程序的状态。 |
| 页面事件  (Page Events) | 重载Windows Phone中的PhoneApplicationPage对象的 OnNavigatedTo（NavigationEventArgs）和OnNavigatedFrom（NavigationEventArgs）方法，管理页面状态。 |
| 逻辑删除  (Tombstoning) | 应用程序被终止的过程，但应用程序的状态数据却被保留。保存的数据包括应用程序当前页面和历史页面的堆栈。如果用户导航回一个逻辑删除的应用程序，应用程序将被重新激活，恢复当前页面和历史页面的数据。 |
| 状态字典  (State Dictionaries) | Windows Phone应用程序提供字典（Dictionary）对象，以键和值成对的形式保存和查询数据。应用程序逻辑删除时，在字典中保存数据，当应用程序重新激活时，查询字典恢复原先的状态。请注意，字典的所有数据必须是可序列化的。 |

表13-1 生命周期相关术语

### 应用程序生命周期模型

本节讨论Windows Phone应用程序生命周期的所有元素，并重点讲述在生命周期的每一过程应采取的行动。 This section provides background on what the operating system and user do to trigger the various changes in application state.本节提供有关操作系统和用户触发应用程序状态变化的背景，以及应用程序应执行的事件处理方法。

有关The execution model is designed to provide end users with a fast, responsive experience at all times.应用程序生命周期模型如下图13-1 应用程序生命周期所示。

This topic provides a detailed explanation of the lifecycle of Windows Phone applications and gives you an overview of the steps needed to develop applications that are responsive and provide a consistent navigation experience. 

图13-1 应用程序生命周期

1. 启动事件

The [Launching](http://translate.googleusercontent.com/translate_c?hl=zh-CN&ie=UTF8&prev=_t&rurl=translate.google.co.jp&sl=en&tl=zh-CN&u=http://msdn.microsoft.com/en-us/library/microsoft.phone.shell.phoneapplicationservice.launching(v%3DVS.92).aspx&usg=ALkJrhhtIUeiOUTZlD10e4i5ATB5iwxTpA) event is raised when a new application instance is launched by the user from the installed applications list or from a Tile on Start in addition to other means, such as tapping on a toast notification associated with an application or selecting an application from the Photos Extras menu.启动事件是应用程序从安装列表或者从启动屏幕（Start）的Tile启动，比如手指触控与应用程序相关联的toast通知而创建新的实例时运行的事件。当执行启动事件，应用程序将创建新的实例，而不是延续以前的实例。为了确保应用程序快速加载，在此事件的处理函数中应尽可能减少代码和处理逻辑，尤其是文件和网络操作等的资源操作密集型任务。为了应用程序获得最佳的用户体验，应在后台线程上执行这些任务。

1. 运行

应用程序启动后进入运行状态，直到应用程序终止在前台的运行的激活状态。当手机自动锁屏时，应用程序也会离开运行状态，除非在应用程序设置应用程序空闲监测无效禁止自动锁屏。

1. OnNavigatedFrom方法

当离开应用程序时[OnNavigatedFrom (NavigationEventArgs)](http://translate.googleusercontent.com/translate_c?hl=zh-CN&ie=UTF8&prev=_t&rurl=translate.google.co.jp&sl=en&tl=zh-CN&u=http://msdn.microsoft.com/en-us/library/system.windows.controls.page.onnavigatedfrom(v%3DVS.92).aspx&usg=ALkJrhhTnFNMeMdTx8fVirIkHE5ZPCdfSw)方法将被调用。当应用程序停用（deactivated）时也会调用[OnNavigatedFrom (NavigationEventArgs)](http://translate.googleusercontent.com/translate_c?hl=zh-CN&ie=UTF8&prev=_t&rurl=translate.google.co.jp&sl=en&tl=zh-CN&u=http://msdn.microsoft.com/en-us/library/system.windows.controls.page.onnavigatedfrom(v%3DVS.92).aspx&usg=ALkJrhhTnFNMeMdTx8fVirIkHE5ZPCdfSw)方法保存页面状态，以便在返回该页面时恢复原先的状态。唯一的例外是向后导航，NavigationMode属性可以判断出是否是向后导航，如果是则没有必要保存状态，因为下次访问时页面会被重建。

1. 停用事件

当用户向前导航离开应用程序或者按下“启动”的物理键启动另一个应用程序时，停用（Deactivated）事件将被调用。当应用程序启动选择器任务时或者手机自动锁屏，停用（Deactivated）事件也会被调用。如果禁止应用程序的空闲监测，那么手机自动锁屏将不会执行。

在处理停用事件时，应用程序应保存所有的运行状态数据，以便随后的恢复时使用。Windows Phone应用程序提供State对象，在状态字典里保存应用程序的状态。当应用程序逻辑删除后被重新激活时，状态字典中的数据将被赋值。因为状态字典里的数据是保存在内存中，所以保存的状态不应包含资源密集的文件操作。

有可能应用程序在停用后被彻底终止，如果应用程序终止，状态字典将不会被保留。鉴于此，应在停用事件中将所有的状态数据保存到独立存储空间。

1. 休眠

当应用程序向前导航离开应用程序，在停用事件被调用后，应用程序将进入休眠状态。在这种状态下，应用程序的所有线程被停止，但是应用程序仍然完整的保留在内存中。如果应用程序被重新激活， 应用程序不需要重新创建任何状态，因为所有的状态一直保留在内存中。

如果新的应用程序启动后需要更多的内存资源，以保证良好的用户体验，操作系统将逻辑删除休眠的应用程序释放内存。

1. 逻辑删除

逻辑删除的应用程序被终止，但是有关导航状态和状态字典在停用事件的处理过程中可以被保留。智能设备同时管理5个最新的逻辑删除信息。如果用户导航进入逻辑删除的应用程序，应用程序将重新启动并且使用保存的数据和状态信息。除非应用程序被简单终止没有任何数据被保存下来。

1. 激活事件

当应用程序从休眠或者逻辑删除状态返回时，将调用激活事件（Activated Event）。操作系统检查应用程序的IsApplicationInstancePreserved属性，以确定应用程序从休眠状态还是从逻辑删除状态返回。如果IsApplicationInstancePreserved为true表示应用程序处于休眠状态，操作系统从内存中恢复应用程序的状态。如果IsApplicationInstancePreserved为false表示应用程序处于逻辑删除状态，应用程序将使用状态字典恢复逻辑删除前的状态。请注意不要在激活事件中执行资源密集型的任务，比如从独立存储空间或者网络加载资源，这些无疑都增加了应用程序重新恢复所消耗的时间；最佳的做法是在应用程序启动后在后台线程上执行这类操作。

1. OnNavigatedTo方法

当用户导航到页面时调用页面的OnNavigatedTo（NavigationEventArgs）方法，包括应用程序第一次启动，或者应用程序从休眠、逻辑删除状态被重新加载。在此方法中，应检查页面是否是新的实例，如果不是则需重新加载状态数据。如果是新的实例，则从页面的状态字典中读取数据，恢复页面的UI状态。

1. 关闭事件

当用户向后导航直至离开应用程序的第一个页面时，关闭事件（Closing Event）将被调用，随后应用程序将被终止。在关闭事件的处理函数中，应保存应用程序中实例的数据。在保存所有应用程序数据和页面导航事件时，有一个10秒钟限制。如果保存时间超出10秒，应用程序将被强制终止。为此，软件设计时应考虑在整个生命周期中随时保存重要的数据，避免在关闭事件中处理大量的文件I/O操作。

## 逻辑删除

When the user switches away from the foreground application, or when an external event causes another application to become the foreground application, the original application is deactivated in a process referred to as "tombstoning."当应用程序被用户切换由前台切换至后台，或者应用程序受到外部事入侵时，应用程序的状态将由运行状态改变为休眠状态，并有可能导致应用程序处于逻辑删除（tombstoning）状态。导致应用程序进入逻辑删除的外部事件包括电话呼叫事件、屏幕锁定事件等。When the user switches back to the application, it is reactivated.当用户切换回应用程序，它将被重新激活。这样做就解除了用户在其移动终端设备后台上运行多个应用程序而导致的程序间竞争有限的系统资源，而使用户的移动终端设备性能和电量得到优化。

In addition, an application that is tombstoned may be completely removed from memory if, for example, the operating system detects that the device is running low on resources.此外，应用程序即进入停用状态后，有可能会从内存中完全删除，比如操作系统检测到设备运行的资源不足时。 In this scenario, when the user goes back to the application, it will restart.在这种情况下，当用户返回到应用程序，它会重新启动。 Therefore, you must make sure that you have preserved all persistent state to storage whenever your application is deactivated.因此，当应用程序被停用时，必须确保已经保留了所有的状态信息。This means that if you want to preserve state (including page and control state) when your application stops being the foreground application, you must handle the events exposed by the operating system to save any state you require to implement a seamless resume experience (such as setting the contents of text boxes, a selected date, and so on).这意味着，如果你想应用程序重新激活时保持退出前的状态（包括页面和控件状态），您必须自行处理数据的保存，如设置的文本框内容，选定的日期等。

正如孙子兵法中所讲：“故不尽知用兵之害者，则不能尽知用兵之利也。”同样的道理，虽然逻辑删除对于移动终端的设备性能可以得到提升，但是对于开发者而言，必须了解逻辑删除的负面作用，就是需要开发者手工编写代码存储应用程序的状态信息，而且要了解临时数据保存和永久数据保存的区别和方法。这样才能开发出用户粘度强的应用程序。

应用程序逻辑删除后，用户可能不会返回到应用程序中。因此，应用程序在停用事件和关闭事件处理程序中将永久状态数据保存到独立存储空间。为避免代码重复，可能创建单独的存储方法，将永久数据保存到独立存储空间，并在这两个事件处理程序调用此方法。请注意在存储数据至独立存储空间的耗时控制在10秒钟之内，建议在运行过程中随时保存重要数据，在停用和关闭事件处理程序中将保存的数量降至最低。

如果应用程序依赖于独立存储空间的数据，不应该在启动事件或者激活事件处理程序中加载数据，最佳的做法是在应用程序启动后，数据从独立存储空间异步加载的方法。因为在应用程序启动时，读取独立存储空间存储的数据需要等待较长的时间。

**注意：**

不能保证所有的应用程序被停用（即Deactivated）后都能被重新激活（即Activated），因此应该保存重要的状态信息于独立存储空间或远程服务之中。

### 导致逻辑删除发生的操作

本节列出所有可能导致应用程序进入逻辑删除处于停用状态的操作。

1. 用户操作

下列用户操作可能导致应用程序逻辑删除。

* 用户按下开始按钮（Start）。
* 用户停止与应用程序交互，手机自动进入锁屏状态。当然应用程序也可以设计为当手机锁屏状态时仍然在后台运行，比如导航程序就需要具备此功能。

1. 事件和任务

为了提高性能，以下的启动器和选择器通常不会导致应用程序进入逻辑删除状态。但是，也不能排除没有这种可能。因此应用程序应设计处理发生的可能。

* PhotoChooserTask
* CameraCaptureTask
* MediaPlayerLauncher
* EmailAddressChooserTask
* PhoneNumberChooserTask
* Multiplayer Game Invite (XNA)
* Gamer You Card (XNA)

**注意**：

在一般情况下调用应用程序并不会被逻辑删除，但是当操作系统开启一个体验的时候发现需要比当前可用资源更多的资源时，那么应用程序就可能被逻辑删除。

### 10秒钟原则

尤其需要注意的是这四个事件的处理时间不能超过10秒，即10秒原则。10秒钟原则正是软件开发中针对“用兵之害”所应采取的规避。故曰：知彼知己者，百战不殆；不知彼而知己，一胜一负；不知彼不知己，每战必殆。

|  |  |  |
| --- | --- | --- |
| **操作** | **限制** | **详情** |
| 启动(Startup) | 10秒 | 应用程序启动，Page的OnNavigatedTo事件处理完毕 |
| 退出(Exit) | 10秒 | 用户按下后退键，Closing事件处理完毕 |
| 返回应用程序(Return to application) | 10秒 | 返回到先前启动应用程序[如按下后退键]，页面的OnNavigatedTo事件处理完毕 |
| 离开应用程序(Leaving an application) | 10秒 | 应用程序退出前台[例如：按下启动键，屏幕锁定事件]，Deactivated事件处理完毕 |

表13-2 10秒钟原则

## XNA Game Studio逻辑删除

### 概述

为了提供一个流畅的用户体验和良好的性能，同时节省宝贵的电池电量，Windows Phone中在前台的应用程序是完全运行，而其他应用程序将被关闭或停用。正因为如此，了解哪些类型的事件可能会导致你的游戏逻辑删除或者关闭，以及如何保存和载入游戏状态是非常重要的。只有这样，才能为游戏者提供一个无缝的和令人愉快的游戏体验。

本节重点介绍了可能会导致游戏中断的条件，以及如何检测和响应中断事件。

Windows Phone的游戏可中断主要有两种方式：

1. 游戏被停用（*deactivated*），随后进入逻辑删除状态，在游戏运行过程中信号暂时中断。造成停用的原因可能是来自系统顶级的通信中断，比如接收到电话呼叫请求、短信、警告和日历事件。也可能是以下几个原因：
   * 在玩游戏时用户按下搜索键；
   * 屏幕被锁定时系统超时；
   * 用户按下并释放电源键。
2. 游戏终止（*closed*）。在游戏运行过程中信号完全中断，发生的条件是：
   * 用户结束游戏；
   * 游戏处于停用状态时间超时；
   * Windows Phone关机或者重新启动；
   * 系统释放内存，停止游戏的执行。

### 与Silverligh逻辑删除的区别

XNA Game Studio和Windows Phone提供的多种方法检测和响应停用、激活、关闭和启动事件。

* XNA Game Studio提供**Game** 类的**OnDeactivation**、 **OnActivation**、**OnExiting**和**Initialize** 方法管理应用程序生命周期的事件；Windows Phone 的Silverlight API提供**PhoneApplicationService** 类的**Deactivated**、**Activated**、**Closing**和**Launching**事件。

虽然XNA的**OnDeactivation**和**OnActivation**方法与Windows Phone的**Deactivated**和**Activated** 方法类似，但是二者还是存在明显的区别。如下表 Xna.Framework.Game方法和表  Microsoft.Phone.Shell.PhoneApplicationService事件所示。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Initialize | OnActivation | OnDeactivation | Exiting |
| Game launches | X | X |  |  |
| Game is deactivated (tombstoned) |  |  | X |  |
| Game is reactivated | X | X |  |  |
| Guide dialog is up |  |  | X |  |
| Guide dialog is dismissed |  | X |  |  |
| Game shuts down |  |  | X | X |

表13-3 Xna.Framework.Game方法

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Launching | Activated | Deactivated | Closing |
| Game launches | X |  |  |  |
| Game is deactivated (tombstoned) |  |  | X |  |
| Game is reactivated |  | X |  |  |
| Guide dialog is up |  |  |  |  |
| Guide dialog is dismissed |  |  |  |  |
| Game shuts down |  |  |  | X |

表13-4  Microsoft.Phone.Shell.PhoneApplicationService事件

由于游戏启动时就调用**Game.OnActivation**和**OnDeactivation**方法，因此这些方法本身对于游戏逻辑删除后重新激活时恢复数据是没有使用价值的。此时，调用**PhoneApplicationService**类恢复数据是有必要的。

**注意：**

在使用**PhoneApplicationService**类的方法之前，需要在XNA Game Studio中添加Microsoft.Phone工具集。操作方法是在Visual Studio的**[Solution Explorer]—[References]**点击右键，选择**[Add Reference…]**，选中Microsoft.Phone添加入工程中。并在代码文件的头部，增加Microsoft.Phone.Shell的引用。

### 判断重新激活的方法

逻辑删除发生之后检测应用程序重新激活的方法之一就是使用**PhoneApplicationService.StartupMode**属性判断。**PhoneApplicationService.StartupMode**属性在游戏创建前被赋值，所以可以通过检测该属性值判断应用程序是启动还是从逻辑删除状态恢复。当应用程序从逻辑删除状态恢复时**StartupMode**属性被赋值为**Activate**；当应用程序启动时**StartupMode**属性被赋值为**Launch**。

protected override void Initialize()

{

if (PhoneApplicationService.Current.StartupMode == StartupMode.Activate)

{

// game is resuming from tombstoning. Restore any transient data that was saved.

}

}

**注意：**

该技术不能应用于检测游戏停用的问题，无法判断是游戏推出还是逻辑删除事件。要解决此问题需要使用**PhoneApplicationService.Deactivated**的事件处理程序。

### 区别游戏停用或者重新激活的事件

 使用**PhoneApplicationService.Deactivated**和**Activated**事件处理区别游戏停用还是从逻辑删除状态重新激活，避免使用前面提及的**Game.OnActivation**和**OnDeactivation**处理逻辑删除事件。

使用**PhoneApplicationService**事件响应逻辑删除的步骤：

1. 在游戏类的构造函数中创建 **PhoneApplicationService.Deactivated**和**PhoneApplicationService.Activated**事件处理程序；
2. 使用事件处理程序来存储和恢复的瞬态和持久的数据，以便逻辑删除状态返回时重新加载瞬态和永久数据。.

### 保存和加载瞬态数据

游戏应用程序停用时**PhoneApplicationService.Current**对象管理瞬态数据。正因如此，**PhoneApplicationService.State**属性被用来保存游戏的瞬态数据，该属性是**System.Collections.Generic.IDictionary**对象，关键词和值是一一对应的。

比如，在**PhoneApplicationService.Deactivated**事件处理程序中，保存瞬态数据的方法如下代码所示。

void GameDeactivated(object sender, DeactivatedEventArgs e)

{

PhoneApplicationService.Current.State["BugPos"] = bug.Position;

PhoneApplicationService.Current.State["BugRot"] = bug.Rotation;

PhoneApplicationService.Current.State["BugTarget"] = bug.Target;

PhoneApplicationService.Current.State["BugMoving"] = bug.Moving;

PhoneApplicationService.Current.State["foodLocations"] = foodLocations;

}

在**PhoneApplicationService.Activated**事件处理程序中，瞬态数据被重新加载。

void GameActivated(object sender, ActivatedEventArgs e)

{

bug.Position = (Vector2)(PhoneApplicationService.Current.State["BugPos"]);

bug.Rotation = (float)(PhoneApplicationService.Current.State["BugRot"]);

bug.Target = (Vector2)(PhoneApplicationService.Current.State["BugTarget"]);

bug.Moving = (bool)(PhoneApplicationService.Current.State["BugMoving"]);

foodLocations =

(List<Vector2>)(PhoneApplicationService.Current.State["foodLocations"]);

gameIsPaused = true;

}

**注意：**

**PhoneApplicationService.State**的状态字典在停用事件之后和激活事件之前将不能被修改，因此在使用激活和停用的事件处理程序来修改游戏的状态数据是可行的方法。

### 保存和加载持久数据

当游戏结束时，**PhoneApplicationService**对象将被删除，所以不能使用其保存和任何持久的数据。相反，应使用游戏的独立存储空间保存持久数据，比如游戏的最高分和玩家游戏记录。不同应用程序的独立存储空间是不能互相访问的，所以独立存储空间最好的保存持久数据的地方。

使用独立存储空间，需要在代码文件的头部添加**System.IO.IsolatedStorage**的引用，调用独立存储空间的步骤如下：

1. 调用**IsolatedStorageFile.GetUserStoreForApplication**方法获得 **IsolatedStorageFile** 对象；
2. 使用**IsolatedStorageFile.OpenFile**打开文件，模式为**FileMode.Create**，应用程序将返回 **IsolatedStorageFileStream**对象；
3. 使用**IsolatedStorageFileStream.Write**方法向文件中写入数据。

// Save the game state (in this case, the high score).

IsolatedStorageFile savegameStorage = IsolatedStorageFile.GetUserStoreForApplication();

// open isolated storage, and write to the file.

IsolatedStorageFileStream fs = null;

using (fs = savegameStorage.CreateFile(SAVEFILENAME))

{

if (fs != null)

{

// just overwrite the existing info for this example.

byte[] bytes = System.BitConverter.GetBytes(highScore);

fs.Write(bytes, 0, bytes.Length);

}

}

当应用程序重新启动，加载数据得步骤如下：

1. 调用**IsolatedStorageFile.GetUserStoreForApplication**方法获得 **IsolatedStorageFile**对象；
2. 使用**IsolatedStorageFile.OpenFile** 打开文件，模式为**FileMode.Open**，应用程序将返回 **IsolatedStorageFileStream**对象；
3. 使用**IsolatedStorageFileStream.Read**方法读取文件中的数据。

using (IsolatedStorageFile savegameStorage =

IsolatedStorageFile.GetUserStoreForApplication())

{

if (savegameStorage.FileExists(SAVEFILENAME))

{

using (IsolatedStorageFileStream fs =

savegameStorage.OpenFile(SAVEFILENAME, System.IO.FileMode.Open))

{

if (fs != null)

{

// Reload the saved high-score data.

byte[] saveBytes = new byte[4];

int count = fs.Read(saveBytes, 0, 4);

if (count > 0)

{

highScore = System.BitConverter.ToInt32(saveBytes, 0);

}

}

}

}

}

### 在Windows Phone模拟器中调试

测试游戏的启动、停用、关闭和重新激活。

1. 在 Visual Studio工具的**XNA Game Studio Deployment Device**选项中选择Windows Phone Emulator；



图13-2 Start Debugging

1. 启动模拟器和游戏；
2. 点击“**Home**” 按钮(**Start**键)；

此时游戏将被停止，模拟器中显示的手机的启动屏幕。

1. 点击后退键（**Back**键）；

游戏被重新激活。

1. 在Visual Studio中选择停止调试。

模拟游戏关闭事。

1. 在Visual Studio中选择重新启动游戏应用程序的调试。

模拟游戏启动。

## 动手实践——快速应用切换

快速应用切换（Fast Application Switching）是Windows PhoneMango的新功能，允许应用程序在停用后在内存中休眠。通过判断应用程序是否从休眠而不是逻辑删除状态被激活，可以优化应用程序的恢复时间。本节重点内容是如何使用应用程序生命周期事件ActivatedEventArgs参数的测试，实现快速应用切换的功能。

Windows Phone Mango中一旦应用程序被操作系统的后台执行，应用程序的镜像其实仍然保存在内存中，只要不损害当前活动的应用程序的性能，或者当前活动的应用程序不需要更多的系统资源。如果重新启动应用程序，用户将体验到应用程序镜像从内存中瞬间恢复的快速操作。这种新机制被称为快速应用切换（Fast Application Switching –FAS）。



### 检测FAS

从内存还原镜像的方法很简单，Windows Phone Mango的**ActivatedEventArgs**事件中包含**IsApplicationInstancePreserved**属性，由此属性的值可以知道应用程序的镜像是否保存在内存中。**IsApplicationInstancePreserved**属性值为true表示应用程序的镜像可以从内存中读取，如果值为false，则表示逻辑删除已经发生。

打开Part 2\chapter 13\ExecutionModelSample\ExecutionModelSample.sln，在ExecutionModelSample工程中打开**App.xaml.cs**文件。

Silverlight Project: ExecutionModelSample File: App.xaml.cs

// Code to execute when the application is activated (brought to foreground)

// This code will not execute when the application is first launched

private void Application\_Activated(object sender, ActivatedEventArgs e)

{

if (e.IsApplicationInstancePreserved)

{

WasTombstoned = false;

ApplicationDataStatus = "application instance preserved.";

return;

}

else

{

WasTombstoned = true;

}

// Check to see if the key for the application state data is in the State dictionary.

if (PhoneApplicationService.Current.State.ContainsKey("ApplicationDataObject"))

{

// If it exists, assign the data to the application member variable.

ApplicationDataStatus = "data from preserved state.";

ApplicationDataObject = PhoneApplicationService.Current.State["ApplicationDataObject"] as string;

}

}

**注意：**

上面的代码中，应用程序从内存中恢复保留的镜像，并没有执行进一步的初始化操作。但并非总是如此，比如应用程序使用套接字（Widows Phone Mango的新功能），不管逻辑删除是否发生，在应用程序休眠时套接字连接都会终止。一旦应用程序被重新激活，套接字的连接需要重新终止的连接。

### 强制逻辑删除

Windows ® Phone Mango的应用程序生命周期管理器默认在内存中保留应用程序的镜像。假定应用程序实现逻辑删除过程过于复杂使得调试起来很麻烦，比如要使系统内存不足才能让操作系统删除应用程序的镜像，这样调试时才会进入逻辑删除逻辑代码。幸运的是，新的开发工具，使我们能够强制应用程序执行逻辑删除。现在，我们将了解如何强制操作系统执行应用程序逻辑删除。

打开**ExecutionModelSample**项目的属性，然后导航到**[Debug]**选项卡。选中**[Tombstone upon deactivation while debugging]**选项。在调试期间应用程序停用后将强制执行逻辑删除。清除该选项，应用程序将执行快速应用切换（FAS）。

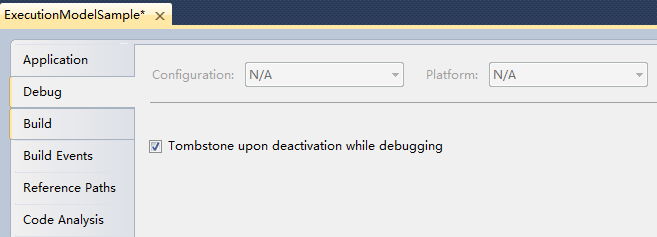


图13-3 **ExecutionModelSample**项目的属性

## **动手实践——应用程序 “足迹”**

应用程序“足迹”借鉴于Windows Phone开发培训包[Silverlight实验室](http://msdn.microsoft.com/zh-cn/windowsphone/gg675037.aspx)的**应用程序生命周期的代码，在此基础上将日期选择控件更改为**Silverlight for Windows Phone Toolkit的**DatePicker**控件，并使用Page Transitions组件实现日期选择的动画效果。Silverlight for Windows Phone Toolkit（<http://silverlight.codeplex.com/>）上面有很多有趣的控件，比如GestureService/GestureListener、Page Transitions、PerformanceProgressBar，这些组件会让我们的应用程序体验出Windows Phone独特的Metro风格。

### 前提

安装[**Silverlight for Windows Phone Toolkit**](http://silverlight.codeplex.com/)，下载地址<http://silverlight.codeplex.com/> ，如图13-4 Silverlight for Windows Phone Toolkit。





图13-4 Silverlight for Windows Phone Toolkit

### 创建用户界面

创建足迹的Windows Phone 7应用程序，通过这个简单的应用程序，理解Windows Phone应用程序生命周期和逻辑删除。

打开**Visual Studio 2010**或者**Visual Studio 2010 Express**，新建**Silverlight for Windows Phone**工程，类型为**Windows Phone Application**，工程名称为[**ApplicationLifecycle**]，如图13-5 新建Windows Phone应用程序。

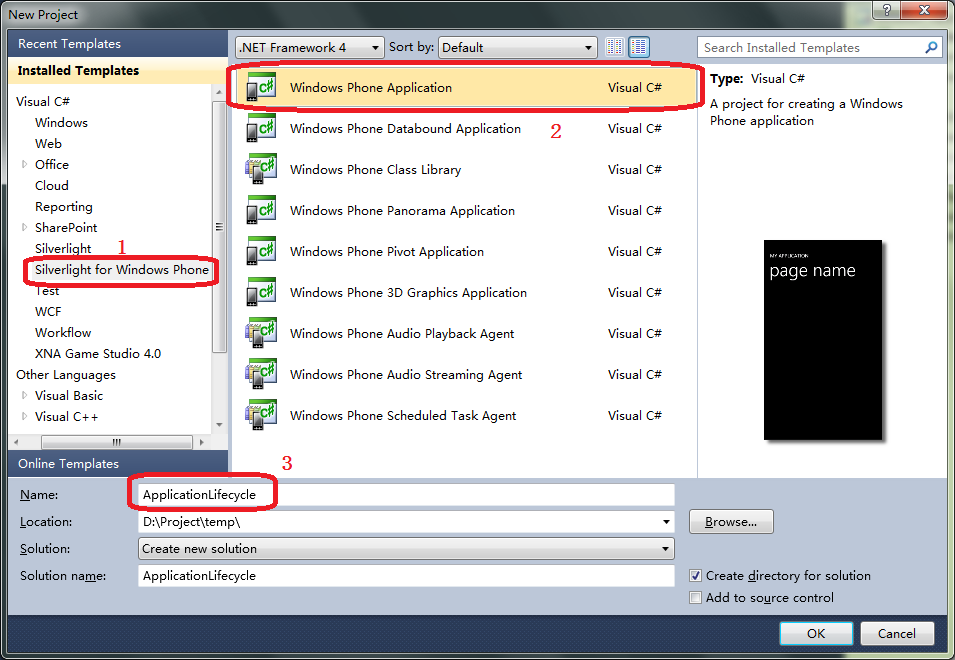


图13-5 新建Windows Phone应用程序

为创建的工程添加一个新的工程文件夹并命名为**Misc**。在工程名称（**ApplicationLifecycle**）上右键单击，选择**Add**然后选择**New Folder**，如图13-6 创建一个新的工程文件夹。

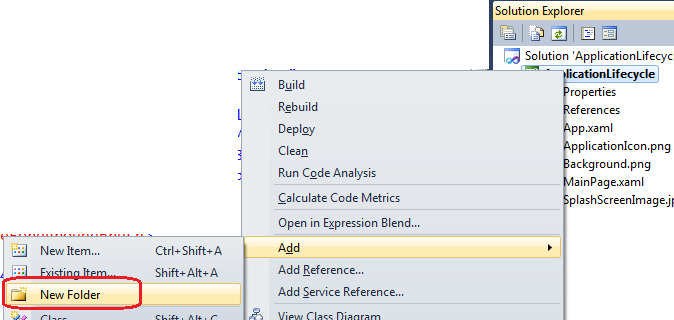


图13-6 创建一个新的工程文件夹

把**\Asset\Misc**文件夹中系统提供的辅助类**TravelReportInfo**和**Utils**添加到**Misc**文件夹。为此，需要在**Misc**上右键单击，选择**Add**然后从右面扩展菜单中选择**Existing Item**。

* + 1. 
    2. 图13-7 添加Misc文件夹中的类

在“**Add Existing Item**”对话框中，转到**\Asset\Misc\**文件夹的所在路径，然后选择所有的源文件**TravelReportInfo.cs**和**Utils.cs**，点击**Add**。

* **TravelReportInfo**是一个模型类，用来表示一个旅程；它包含了描述旅行的数据域。
* **Utils**正如它的命名一样，是一个泛型类，它被整个工程中各种各样的函数广泛使用。

在工程的引用中添加一个针对**System.Xml.Serialization**的引用程序集和针对**Microsoft.Controls.Toolkit**的应用程序集。在工程**References**文件夹上右键单击选择**Add Reference**，如图13-8 右键选择添加引用。

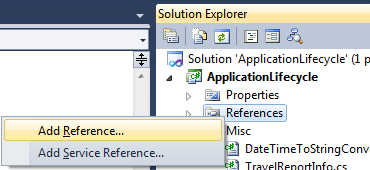
* + 1. 

图13-8 右键选择添加引用

在**Add Reference**对话框中的组件列表中选择**System.Xml.Serialization**，然后单击**OK**，如图13-9 添加引用。

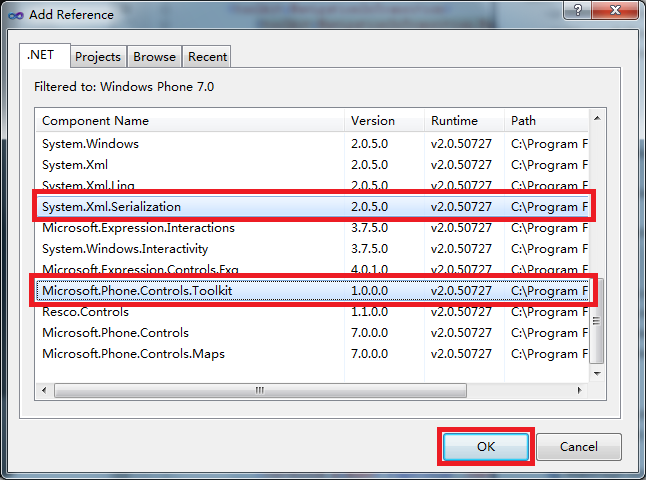


图13-9 添加引用

设计MainPage的显示样式为“目的地”和“心情”栏目使用控件**TextBox**，“起始”和“结束”时间使用控件**DatePicker**，如图 MainPage样式。



图13-10 MainPage样式

Silverlight Project: ApplicationLifecycle File: MainPage.xaml

<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="254\*" />

<ColumnDefinition Width="202\*" />

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="80"/>

<RowDefinition Height="80"/>

<RowDefinition Height="80"/>

<RowDefinition/>

<RowDefinition/>

</Grid.RowDefinitions>

<StackPanel Orientation="Horizontal" Grid.ColumnSpan="2">

<TextBlock Text="目的地" Style="{StaticResource PhoneTextLargeStyle}" VerticalAlignment="Center" Width="100" />

<TextBox x:Name="txtDestination" Text="{Binding Destination, Mode=TwoWay}" Width="350" InputScope="AddressCity"/>

</StackPanel>

<StackPanel Grid.Row="1" Orientation="Horizontal" Grid.ColumnSpan="2">

<TextBlock Text="起始" Style="{StaticResource PhoneTextLargeStyle}" VerticalAlignment="Center" Width="100" />

<toolkit:DatePicker x:Name="txtFromDate" Width="350" DataContext="{Binding ElementName=txtFromDate}" ValueChanged="txtFromDate\_ValueChanged" />

</StackPanel>

<StackPanel Grid.Row="2" Orientation="Horizontal" Grid.ColumnSpan="2">

<TextBlock Text="结束" Style="{StaticResource PhoneTextLargeStyle}" VerticalAlignment="Center" Width="100" />

<toolkit:DatePicker x:Name="txtToDate" Width="350" DataContext="{Binding ElementName=txtToDate}" ValueChanged="txtToDate\_ValueChanged" />

</StackPanel>

<StackPanel Grid.Row="3" Grid.ColumnSpan="2">

<TextBlock Text="心情" Style="{StaticResource PhoneTextLargeStyle}"/>

<TextBox x:Name="txtJustification" Text="{Binding Justification, Mode=TwoWay}" AcceptsReturn="True" Height="200" VerticalScrollBarVisibility="Auto"/>

</StackPanel>

</Grid>

</Grid>

在MainPage.xaml的开始位置加入以下的代码。

Silverlight Project: ApplicationLifecycle File: MainPage.xaml

xmlns:toolkit="clr-namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone.Controls.Toolkit"

添加调用DatePicker日期选择控件的动画效果。

Silverlight Project: ApplicationLifecycle File: MainPage.xaml

<toolkit:TransitionService.NavigationInTransition>

<toolkit:NavigationInTransition>

<toolkit:NavigationInTransition.Backward>

<toolkit:TurnstileTransition Mode="BackwardIn"/>

</toolkit:NavigationInTransition.Backward>

<toolkit:NavigationInTransition.Forward>

<toolkit:TurnstileTransition Mode="ForwardIn"/>

</toolkit:NavigationInTransition.Forward>

</toolkit:NavigationInTransition>

</toolkit:TransitionService.NavigationInTransition>

<toolkit:TransitionService.NavigationOutTransition>

<toolkit:NavigationOutTransition>

<toolkit:NavigationOutTransition.Backward>

<toolkit:TurnstileTransition Mode="BackwardOut"/>

</toolkit:NavigationOutTransition.Backward>

<toolkit:NavigationOutTransition.Forward>

<toolkit:TurnstileTransition Mode="ForwardOut"/>

</toolkit:NavigationOutTransition.Forward>

</toolkit:NavigationOutTransition>

</toolkit:TransitionService.NavigationOutTransition>

添加调用DatePicker日期选择控件的事件响应函数。在本例中DatePicker日期选择控件的数据绑定的处理方法与其他**TextBox**控件的处理方法不同，在下一节的内容将详细讲述在逻辑删除中DatePicker日期选择控件的数据绑定处理方法。

Silverlight Project: ApplicationLifecycle File: MainPage.xaml.cs

private void txtToDate\_ValueChanged(object sender, DateTimeValueChangedEventArgs e)

{

}

private void txtFromDate\_ValueChanged(object sender, DateTimeValueChangedEventArgs e)

{

}

添加应用程序栏ApplicationBar，设置三个按钮“Next”、“Save”和“Cancel”。

Silverlight Project: ApplicationLifecycle File: MainPage.xaml

<phone:PhoneApplicationPage.ApplicationBar>

<shell:ApplicationBar IsVisible="True" IsMenuEnabled="False" Opacity="1">

<shell:ApplicationBarIconButton IconUri="/Toolkit.Content/appbar.next.rest.png" Text="Next" Click="ApplicationBarNext\_Click" />

<shell:ApplicationBarIconButton IconUri="/Toolkit.Content/ApplicationBar.Check.png" Text="Save" Click="ApplicationBarSave\_Click" />

<shell:ApplicationBarIconButton IconUri="/Toolkit.Content/ApplicationBar.Cancel.png" Text="Cancel" Click="ApplicationBarCancel\_Click" />

</shell:ApplicationBar>

</phone:PhoneApplicationPage.ApplicationBar>

添加应用程序栏的事件响应代码。

Silverlight Project: ApplicationLifecycle File: MainPage.xaml.cs

private void ApplicationBarCancel\_Click(object sender, EventArgs e)

{

Utils.ClearTravelReport(((App.Current.RootVisual as PhoneApplicationFrame).DataContext as TravelReportInfo));

}

private void ApplicationBarNext\_Click(object sender, EventArgs e)

{

if (NavigationService.CanGoForward)

{

NavigationService.GoForward();

}

else

{

//Navigate to second page

NavigationService.Navigate(new Uri("/SecondPage.xaml", UriKind.Relative));

}

}

private void ApplicationBarSave\_Click(object sender, EventArgs e)

{

Utils.SaveTravelReport(

(App.Current.RootVisual as PhoneApplicationFrame).DataContext as TravelReportInfo,

"TravelReportInfo.dat",

false);

}

重载**OnBackKeypress**方法，当用户正要通过导航键离开当前应用程序被问及是否要保存当前的旅行报告数据。

Silverlight Project: ApplicationLifecycle File: MainPage.xaml.cs

protected override void OnBackKeyPress(System.ComponentModel.CancelEventArgs e)

{

base.OnBackKeyPress(e);

//Ask userto preserve data in persistent store

MessageBoxResult res = MessageBox.Show("Do you want to save your work before?", "You are exiting the application", MessageBoxButton.OKCancel);

if (res == MessageBoxResult.OK)

Utils.SaveTravelReport((App.Current.RootVisual as PhoneApplicationFrame).DataContext as TravelReportInfo,

"TravelReportInfo.dat", true);

else

Utils.ClearTravelReport((App.Current.RootVisual as PhoneApplicationFrame).DataContext as TravelReportInfo);

}

创建第二个显示画面SecondPage.xaml，显示效果如图 SecondPage样式。

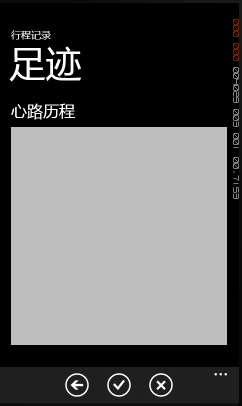


图13-11 SecondPage样式

在应用程序名字上右键单击，选择**Add**并在扩展菜单列表中选择**New Item**，选择 *Windows Phone Portrait Page*, 命名为 **SecondPage** 然后点击 ***Add***，如图 添加SecondPage。

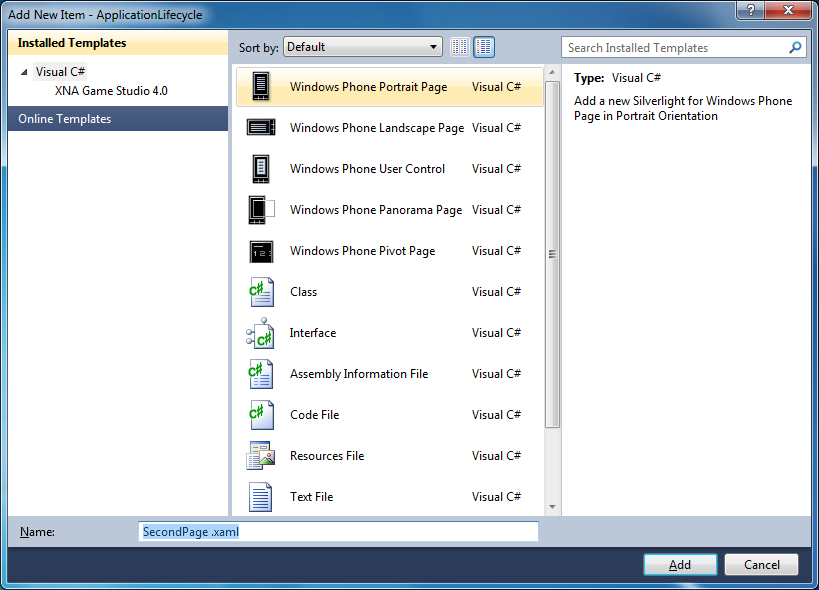


图 添加SecondPage

SecondPage显示的样式代码。

Silverlight Project: ApplicationLifecycle File: SecondPage.xaml

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

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition Height="\*"/>

</Grid.RowDefinitions>

<!--TitlePanel contains the name of the application and page title-->

<StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">

<TextBlock x:Name="ApplicationTitle" Text="行程记录" Style="{StaticResource PhoneTextNormalStyle}"/>

<TextBlock x:Name="PageTitle" Text="足迹" Margin="9,-7,0,0" Style="{StaticResource PhoneTextTitle1Style}"/>

</StackPanel>

<!--ContentPanel - place additional content here-->

<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">

<Grid.RowDefinitions>

<RowDefinition Height="\*"/>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<StackPanel VerticalAlignment="Top">

<TextBlock Text="心路历程" Style="{StaticResource PhoneTextLargeStyle}"/>

<TextBox x:Name="txtSummary" Text="{Binding Summary, Mode=TwoWay}" AcceptsReturn="True" Height="460" VerticalScrollBarVisibility="Auto"/>

</StackPanel>

</Grid>

</Grid>

添加应用程序栏的三个按钮，分别是“Back”回退按钮、“Save”保存按钮和“Cancel”清除数据按钮。

Silverlight Project: ApplicationLifecycle File: SecondPage.xaml

<phone:PhoneApplicationPage.ApplicationBar>

<shell:ApplicationBar IsVisible="True" IsMenuEnabled="False" Opacity="1">

<shell:ApplicationBarIconButton IconUri="/Toolkit.Content/appbar.back.rest.png" Text="Back" Click="ApplicationBarGoBack\_Click" />

<shell:ApplicationBarIconButton IconUri="/Toolkit.Content/ApplicationBar.Check.png" Text="Save" Click="ApplicationBarSave\_Click" />

<shell:ApplicationBarIconButton IconUri="/Toolkit.Content/ApplicationBar.Cancel.png" Text="Cancel" Click="ApplicationBarCancel\_Click" />

</shell:ApplicationBar>

</phone:PhoneApplicationPage.ApplicationBar>

添加应用程序栏按钮的单击处理函数，分别实现回退导航，保存和清除数据的功能。

Silverlight Project: ApplicationLifecycle File: SecondPage.xaml.cs

private void ApplicationBarGoBack\_Click(object sender, EventArgs e)

{

//Navigate go back

if (NavigationService.CanGoBack)

{

NavigationService.GoBack();

}

else

{

NavigationService.Navigate(new Uri("/MainPage.xaml", UriKind.Relative));

}

}

private void ApplicationBarSave\_Click(object sender, EventArgs e)

{

Utils.SaveTravelReport(

(App.Current.RootVisual as PhoneApplicationFrame).DataContext as TravelReportInfo,

"TravelReportInfo.dat",

false);

}

private void ApplicationBarCancel\_Click(object sender, EventArgs e)

{

Utils.ClearTravelReport(((App.Current.RootVisual as PhoneApplicationFrame).DataContext as TravelReportInfo));

}

### 创建应用程序执行逻辑

开发人员利用逻辑删除（tombstone）事件来保存**应用程序状态**和**页面状态。**利用这些状态，开发人员可以把应用程序恢复到最后一个正确的状态。

* **应用程序状态** 是应用程序的一种状态且并不与任何特定页面有关联。应用程序状态是在*PhoneApplicationService*类公开的事件中管理的。
* **页面状态**是一种应用程序页面可见的状态。它包含了诸如ScrollViewer 控件中滚轴的位置和TextBox控件中的内容等信息。页面的状态管理应该由OnNavigatedTo 和 OnNavigatedFrom事件处理程序来处理。

本节将通过模拟一个Windows Phone应用程序的整个生命周期，包括对用户操作而导致一个应用程序的状态改变的描述。 在本节中所有和生命周期相关的事件（启动、运行、关闭、禁止和激活）都是Microsoft.Phone.Shell命名空间下PhoneApplicationService类的成员。

当用户操作离开某些Windows Phone应用程序时，这些应用程序将会处于无效的禁止状态或者是进入逻辑删除状态。本节重点突出事件处理的执行模式的一些最佳实践。

**注意：** Windows Phone工程模板提供了Windows Phone开发者工具，该工具包含存根于工程下的App.xaml.cs文件内的事件处理程序。

打开App.xaml.cs，在文件的开头部分替换using声明代码段，如下：

Silverlight Project: ApplicationLifecycle File: App.xaml.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Animation;

using System.Windows.Navigation;

using System.Windows.Shapes;

using Microsoft.Phone.Controls;

using Microsoft.Phone.Shell;

using System.IO.IsolatedStorage;

using System.Xml.Serialization;

启动应用应用程序（Application\_Launching）的事件处理程序的功能：应用程序将会试图从独立的存储控件中加载以前被保存的数据（持久化数据）。 如果没有数据，创建一个空的数据对象。

Silverlight Project: ApplicationLifecycle File: App.xaml.cs

// Code to execute when the application is launching (eg, from Start)

// This code will not execute when the application is reactivated

private void Application\_Launching(object sender, LaunchingEventArgs e)

{

//Trace the event for debug purposes

Utils.Trace("Application Launching");

//Create new data object variable

TravelReportInfo travelReportInfo = null;

//Try to load previously saved data from IsolatedStorage

using (IsolatedStorageFile isf = IsolatedStorageFile.GetUserStoreForApplication())

{

//Check if file exits

if (isf.FileExists("TravelReportInfo.dat"))

{

using (IsolatedStorageFileStream fs = isf.OpenFile("TravelReportInfo.dat", System.IO.FileMode.Open))

{

//Read the file contents and try to deserialize it back to data object

XmlSerializer ser = new XmlSerializer(typeof(TravelReportInfo));

object obj = ser.Deserialize(fs);

//If successfully deserialized, initialize data object variable with it

if (null != obj && obj is TravelReportInfo)

travelReportInfo = obj as TravelReportInfo;

else

travelReportInfo = new TravelReportInfo();

}

}

else

//If previous data not found, create new instance

travelReportInfo = new TravelReportInfo();

}

//Set data variable (either recovered or new) as a DataContext for all the pages of the application

RootFrame.DataContext = travelReportInfo;

}

激活应用程序（Application\_Activated）的事件处理程序函数。

Silverlight Project: ApplicationLifecycle File: App.xaml.cs

// Code to execute when the application is activated (brought to foreground)

// This code will not execute when the application is first launched

private void Application\_Activated(object sender, ActivatedEventArgs e)

{

//Trace the event for debug purposes

Utils.Trace("Application Activated");

//Create new data object variable

TravelReportInfo travelReportInfo = null;

//Try to locate previous data in transient state of the application

if (PhoneApplicationService.Current.State.ContainsKey("UnsavedTravelReportInfo"))

{

//If found, initialize the data variable and remove in from application's state

travelReportInfo = PhoneApplicationService.Current.State["UnsavedTravelReportInfo"] as TravelReportInfo;

PhoneApplicationService.Current.State.Remove("UnsavedTravelReportInfo");

}

//If found set it as a DataContext for all the pages of the application

//An application is not guaranteed to be activated after it has been tombstoned,

//thus if not found create new data object

if (null != travelReportInfo)

RootFrame.DataContext = travelReportInfo;

else

RootFrame.DataContext = new TravelReportInfo();

}

禁止应用程序（Application\_Deactivated）的事件处理程序函数。

Silverlight Project: ApplicationLifecycle File: App.xaml.cs

// Code to execute when the application is deactivated (sent to background)

// This code will not execute when the application is closing

private void Application\_Deactivated(object sender, DeactivatedEventArgs e)

{

//Trace the event for debug purposes

Utils.Trace("Application Deactivated");

//Add current data object to Application state

PhoneApplicationService.Current.State.Add("UnsavedTravelReportInfo", RootFrame.DataContext as TravelReportInfo);

}

关闭应用程序（Application\_Closing）的事件处理程序函数。

Silverlight Project: ApplicationLifecycle File: App.xaml.cs

// Code to execute when the application is closing (eg, user hit Back)

// This code will not execute when the application is deactivated

private void Application\_Closing(object sender, ClosingEventArgs e)

{

//Trace the event for debug purposes

Utils.Trace("Application Closing");

}

在一些情况下，开发人员在应用程序被禁止或转为无效时力图为用户提供一个准确的用户界面。为此，设计的基础应当关注应用程序何时被恢复。对于这样的情况，使用OnNavigatedTo和OnNavigatedFrom事件来保存页面状态中所需要的数据。

在OnNavigatedFrom事件处理函数中保存MainPage的页面状态，并记录被聚焦（Focused）的控件。

Silverlight Project: ApplicationLifecycle File: MainPage.xaml.cs

protected override void OnNavigatedFrom(System.Windows.Navigation.NavigationEventArgs e)

{

//Trace the event for debug purposes

Utils.Trace("Navigated From MainPage");

//Remove focused element from previous time if any

if (State.ContainsKey("FocusedElement"))

{

State.Remove("FocusedElement");

}

//If some input control is in focus, save it to the page state

object obj = FocusManager.GetFocusedElement();

if (null != obj)

{

string focusedControl = (obj as FrameworkElement).Name;

State.Add("FocusedElement", focusedControl);

}

if (State.ContainsKey("txtDestination"))

State.Remove("txtDestination");

State.Add("txtDestination", txtDestination.Text);

if (State.ContainsKey("txtJustification"))

State.Remove("txtJustification");

State.Add("txtJustification", txtJustification.Text);

base.OnNavigatedFrom(e);

}

在OnNavigatedFrom事件处理函数中恢复MainPage的页面状态。

Silverlight Project: ApplicationLifecycle File: MainPage.xaml.cs

protected override void OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e)

{

//Trace the event for debug purposes

Utils.Trace("Navigated To MainPage");

//Check if page state has saved focus and apply it back

if (State.ContainsKey("FocusedElement"))

{

focusedElement = this.FindName(State["FocusedElement"] as string) as Control;

bFocused = true;

}

else

{

bFocused = false;

}

TravelReportInfo travelReportInfo = ((App.Current.RootVisual as PhoneApplicationFrame).DataContext as TravelReportInfo);

if (State.ContainsKey("txtDestination"))

travelReportInfo.Destination = State["txtDestination"] as string;

if (State.ContainsKey("txtJustification"))

{

travelReportInfo.Justification = State["txtJustification"] as string;

}

base.OnNavigatedTo(e);

}

上面的代码中，我们增加了两个变量focusedElement、bFocused来记录聚焦（focused）控件。focusedElement变量记录聚焦的UI，bFocused布尔型变量记录是否聚焦。由于在执行OnNavigatedTo事件处理函数时，MainPage还未生成，因此恢复控件的聚焦操作需要放在PhoneApplicationPage\_LayoutUpdated事件处理中。

在MainPage.xaml文件的开始部分增加如下代码。

Silverlight Project: ApplicationLifecycle File: MainPage.xaml

LayoutUpdated="PhoneApplicationPage\_LayoutUpdated"

在MainPage.xaml.cs增加PhoneApplicationPage\_LayoutUpdated事件的处理函数。

Silverlight Project: ApplicationLifecycle File: MainPage.xaml.cs

private void PhoneApplicationPage\_LayoutUpdated(object sender, EventArgs e)

{

try

{

if (bFocused && (null != focusedElement))

{

bFocused = false;

focusedElement.Focus();

focusedElement = null;

}

}

catch (Exception ex)

{

//Trace the exception for debug purposes

Utils.Trace(String.Format("Exception = {0}.", ex.GetType()) + "\n");

}

}

在SecondPage中增加页面恢复时的控件聚焦操作，即PhoneApplicationPage\_LayoutUpdated事件的处理函数。

Silverlight Project: ApplicationLifecycle File: SecondPage.xaml

LayoutUpdated="PhoneApplicationPage\_LayoutUpdated"

Silverlight Project: ApplicationLifecycle File: SecondPage.xaml.cs

Control focusedElement;

bool bFocused = false;

private void PhoneApplicationPage\_LayoutUpdated(object sender, EventArgs e)

{

try

{

if (bFocused && (null != focusedElement))

{

bFocused = false;

focusedElement.Focus();

}

}

catch (Exception ex)

{

//Trace the exception for debug purposes

Utils.Trace(String.Format("LayoutUpdated occur exception = {0}.", ex.GetType()) + "\n");

}

}

设定日期选择控件DatePicker的数据更新处理。TextBox通过数据绑定实现了在页面恢复时自动将更新保存的数据。但是对于日期选择控件DatePicker，当用户操作DatePicker选择日期会打开日期选择的页面，同样会离开MainPage的OnNavigatedFrom，选择完毕后返回MainPage更新DatePicker控件的值也会执行MainPage的OnNavigatedTo事件处理函数。为此，需要修改MainPage的OnNavigatedTo和OnNavigatedFrom事件处理函数。

设定变量判断日期选择发生的页面切换还是由于逻辑删除发生的页面切换。在MainPage类中声明两个变量。

Silverlight Project: ApplicationLifecycle File: MainPage.xaml.cs

bool bToDateChangedNavigateTo = false;

bool bFromDateChangedNavigateTo = false;

实现DatePicker控件的数据发生改变时的事件处理。

Silverlight Project: ApplicationLifecycle File: MainPage.xaml.cs

private void txtToDate\_ValueChanged(object sender, DateTimeValueChangedEventArgs e)

{

TravelReportInfo travelReportInfo = ((App.Current.RootVisual as PhoneApplicationFrame).DataContext as TravelReportInfo);

travelReportInfo.LastDay = DateTime.Parse(txtToDate.ValueString);

bToDateChangedNavigateTo = true;

}

private void txtFromDate\_ValueChanged(object sender, DateTimeValueChangedEventArgs e)

{

TravelReportInfo travelReportInfo = ((App.Current.RootVisual as PhoneApplicationFrame).DataContext as TravelReportInfo);

travelReportInfo.FirstDay = DateTime.Parse(txtFromDate.ValueString);

bFromDateChangedNavigateTo = true;

}

修改OnNavigatedFrom事件处理函数，保存日期选择控件的瞬态值。

Silverlight Project: ApplicationLifecycle File: MainPage.xaml.cs

if (State.ContainsKey("txtFromDate"))

State.Remove("txtFromDate");

State.Add("txtFromDate", txtFromDate.ValueString);

if (State.ContainsKey("txtToDate"))

State.Remove("txtToDate");

State.Add("txtToDate", txtToDate.ValueString);

修改OnNavigatedTo事件处理函数，恢复日期选择控件的值。

Silverlight Project: ApplicationLifecycle File: MainPage.xaml.cs

if (State.ContainsKey("txtToDate"))

{

if (!bToDateChangedNavigateTo)

{

travelReportInfo.LastDay = DateTime.Parse(State["txtToDate"] as string);

txtToDate.Value = travelReportInfo.LastDay;//this operation call txtFromDate\_ValueChanged event

bToDateChangedNavigateTo = false;

}

else

{

bToDateChangedNavigateTo = false;

}

}

else

{

//format todata

txtToDate.Value = travelReportInfo.LastDay;

bToDateChangedNavigateTo = false;

}

if (State.ContainsKey("txtFromDate"))

{

if (!bFromDateChangedNavigateTo)

{

travelReportInfo.FirstDay = DateTime.Parse(State["txtFromDate"] as string);

txtFromDate.Value = travelReportInfo.FirstDay;//this operation call txtFromDate\_ValueChanged event

bFromDateChangedNavigateTo = false;

}

else

{

bFromDateChangedNavigateTo = false;

}

}

else

{

// format fromdata

txtFromDate.Value = travelReportInfo.FirstDay;

bFromDateChangedNavigateTo = false;

}

修改PhoneApplicationPage\_LayoutUpdated事件处理函数，与OnNavigatedTo和OnNavigatedFrom事件处理函数共同组成了日期选择控件数据更新和绑定的逻辑。

Silverlight Project: ApplicationLifecycle File: MainPage.xaml.cs

TravelReportInfo travelReportInfo = ((App.Current.RootVisual as PhoneApplicationFrame).DataContext as TravelReportInfo);

txtFromDate.Value = travelReportInfo.FirstDay;

bFromDateChangedNavigateTo = false;

txtToDate.Value = travelReportInfo.LastDay;

bToDateChangedNavigateTo = false;

### 逻辑删除处理流程

设置调试时强制执行逻辑删除。打开**ApplicationLifecycle**项目的属性，然后导航到**[Debug]**选项卡。选中**[Tombstone upon deactivation while debugging]**选项，在调试时应用程序停用后将强制执行逻辑删除。

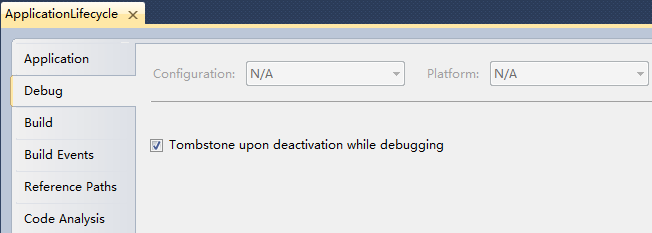


图13-12 **ApplicationLifecycle**项目的属性

编译并运行程序，在输出（Output）窗口查看启动事件。如果输出窗口不可见，通过在**VisualStudio**菜单上点击**View|Output** 或者通过按下**CTRL + W,O**快捷键来打开它，如图 打开输出（**Output**）窗口。

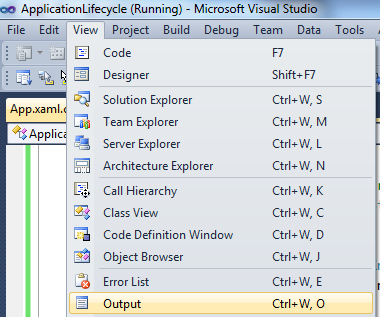


图13-13 打开输出（Output）窗口

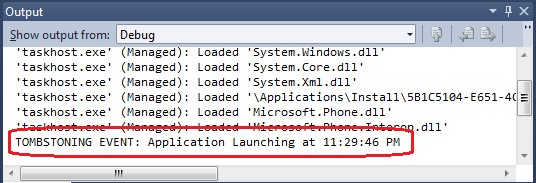


图13-14 输出窗口中的启动（Launching）事件

1. 在第一个迭代过程中，应用程序选择“创建新的数据对象”代码路径，应用程序的日期选择控件只显示默认值，如图13-15 启动画面。



图13-15 启动画面

1. 更改一些值，并点击返回（Back）键，应用程序关闭过程中，用户被问及是否保存程序内容。点击OK来保存数据，如图13-16 关闭应用程序。

图13-16 关闭应用程序

1. 查看输出窗口中的关闭事件信息，如图13-17 输出窗口中的关闭（Closing）事件。

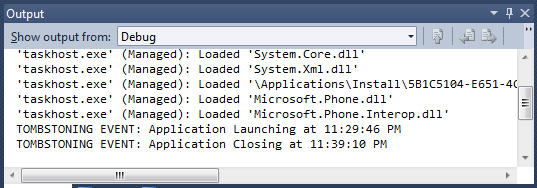


图13-17 输出窗口中的关闭（Closing）事件

1. 用户从程序列表中重新选择ApplicationCycle程序运行时，查看输出（Output）窗口的事件信息，保存在独立存储空间的数据被恢复，如图13-18 输出窗口中的启动（Launching）事件。

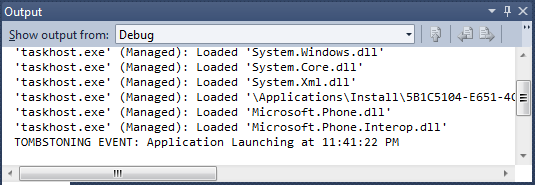


图13-18 输出窗口中的启动（Launching）事件

重新启动后的画面中恢复了保存在独立存储空间中的数据。



图13-19 重新启动后的画面

1. 点击应用程序栏的Next按钮，跳转到第二个页面。聚焦“心路历程”控件，然后点击启动（Start）键来停用（deactivate）当前应用程序，如图13-20 停用（Deactivating） 当前应用程序。



图13-20 停用（Deactivating） 当前应用程序

1. 点击返回（Back）按钮返回应用程序，出现带有“Resuming”和加载动画的页面。如图13-21 应用程序恢复画面。

图13-21 应用程序恢复画面

1. 为了看到不同的行为，再次点击启动（Start）键并通过点击开始页面上的右箭头从主屏幕页面跳转至应用程序列表，重新启动应用程序。此时，原先进入逻辑删除状态的应用程序将被Kill，系统会重新启动该应用程序的一个新的实例，所有的瞬态数据将被清空，保存在独立存储空间中的数据将被重新加载。如图13-22 应用程序列表启动应用程序。本例中的第一个页面保存的数据被加载，而第二个页面的瞬态数据就被丢失无法加载。

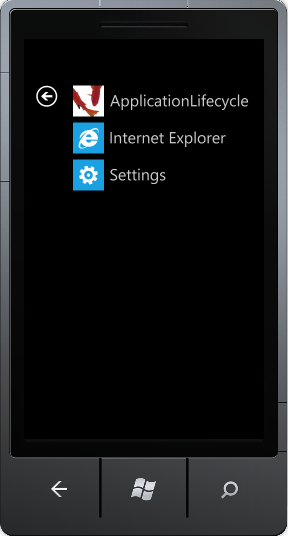
  

图13-22 应用程序列表启动应用程序

通过上面简单的例子，我们理解的应用程序生命周期的逻辑删除。下面为大家提供更加实用的，代码更优化，更能体现代码之美的保存和恢复页面瞬态数据的方法，请参考MSDN的How to: Preserve and Restore Page State for Windows Phone，其提供的保存和恢复TextBox、CheckBox、Slider、RadioButton等控件状态和聚焦状态的方法可以直接在我们的代码中使用。仔细阅读并应用，那会让您的代码结构更清晰。正如论语中所讲：“见贤思齐焉”。

代码之美的实践就是不断提升代码质量，不断设计更优的算法，不断创造改变世界的软件。

## Windows Phone OS 7.0应用程序的生命周期

下面的关系图说明了Windows Phone OS 7.0应用程序的生命周期。



图13-23 Windows Phone OS 7.0 应用程序生命周期