

WP7有约

# 一个应用的破蛋过程

WP7有约  
9:13 上课  
主楼 520



下一节课



李永伦

献给所有喜欢 WP7 的童鞋！



[cnblogs.com/allenlooplee](http://cnblogs.com/allenlooplee)

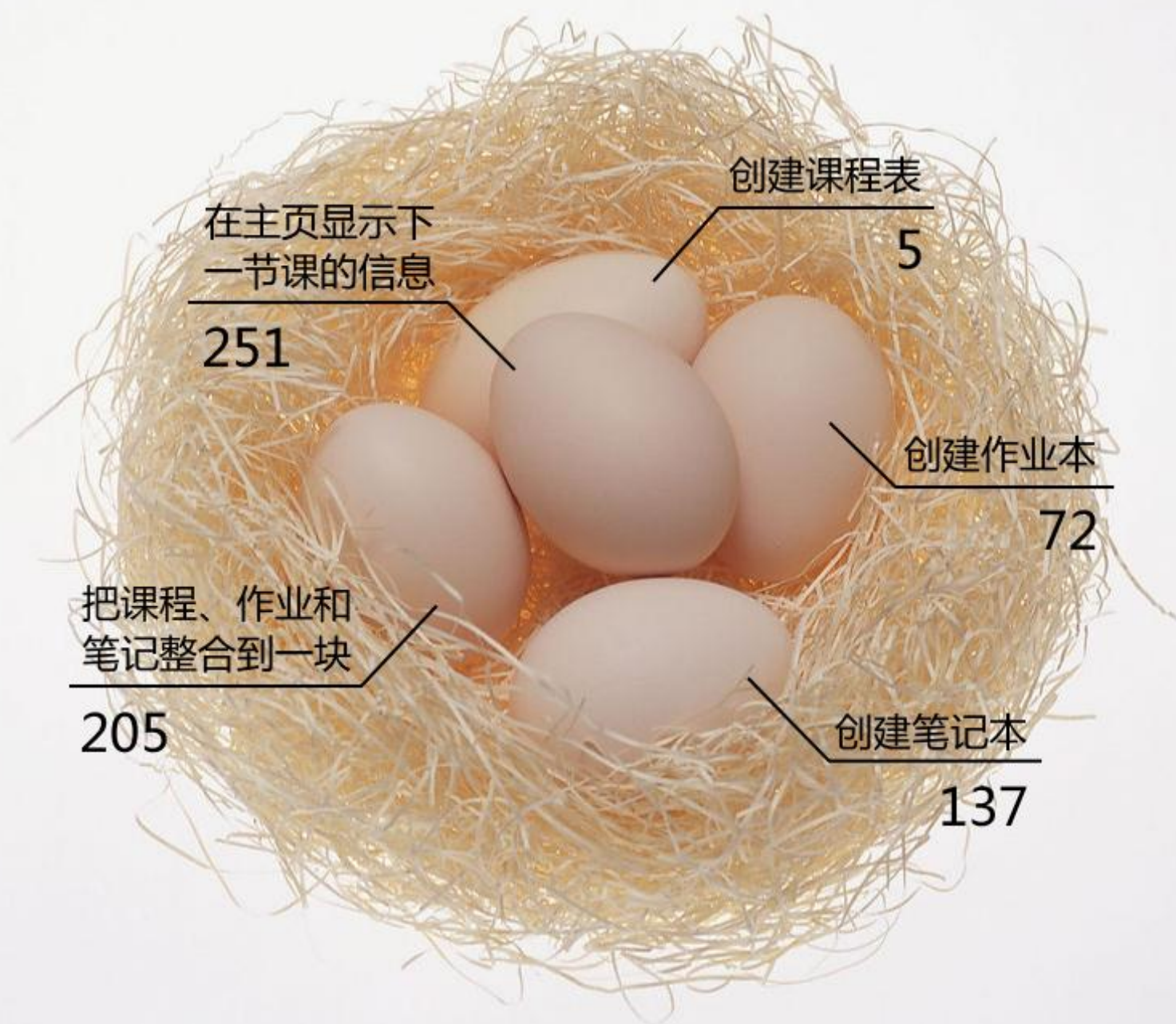


[allenlooplee@yahoo.com.cn](mailto:allenlooplee@yahoo.com.cn)



[iridescent.codeplex.com](http://iridescent.codeplex.com)

# 目录



准备上课 ➔

## 第1课

# 创建课程表

你好，老七！

上课啦！

编辑课程表

Blend如何提供设计时数据的支持？

连接前端和后端

操作课程表

菜单•样品菜色



## 1.1 你好，老七！

### 1.1.1 准备工作

[WP7](#) 终于发布了，到目前为止，有关它的新闻和介绍我相信你已经看过不少了，所以这里将会直接跳过，不过在开始之前，我认为还是有必要提醒你做好相关的准备：

- ✓ [Expression Blend 4 for Windows Phone](#) 和 [Visual Studio 2010 Express for Windows Phone](#)，你并不需要完整的 Expression Studio 4 Ultimate 和 Visual Studio 2010 Ultimate，不过如果你有的话\*可能\*会更好。
- ✓ 白开水，大量白开水，接下来你将会与我一起进行大量脑力活动，你需要补充足够的水分才能让大脑更好地工作。
- ✓ 零食，最好是坚果类，薯片也可以，人无法长时间集中精力，也不该迫使自己长时间集中精力，当你感到注意力开始涣散时，不妨抓一把零食放到嘴里嚼，注意别弄到键盘上哦。
- ✓ 最后，也是最重要的，你，没错，是你，仅当你准备好接受新的知识时，你的大脑才会对它们进行积极的处理，否则就会把它们挡在外面。

那么，你准备好了吗？

### 1.1.2 创建你的第一个 WP7 应用

首先，打开 Expression Blend，创建一个 Windows Phone Panorama Application 项目，如图 1-1 所示：

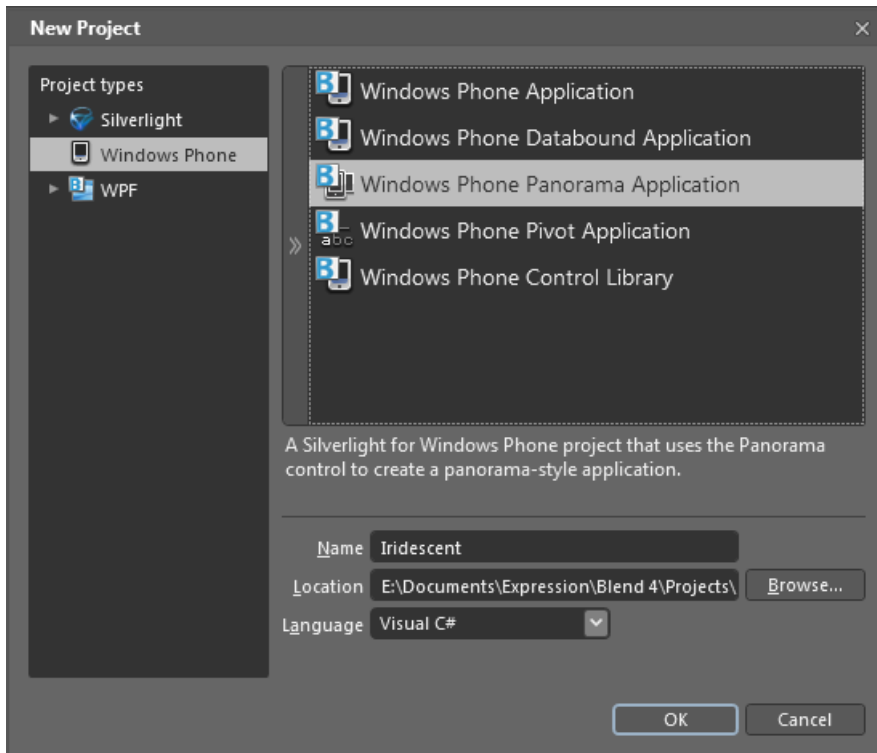


图 1-1 创建你的第一个 WP7 应用

项目创建好之后，你会看到一个充满整个页面的 Panorama 控件，里面有两个 Panorama 项，每个 Panorama 项里面有一个 ListBox，而 ListBox 里也有了示例数据。你可以调整 [Artboard](#) 的缩放比例，如图 1-2 所示，以便显示整个 UI：

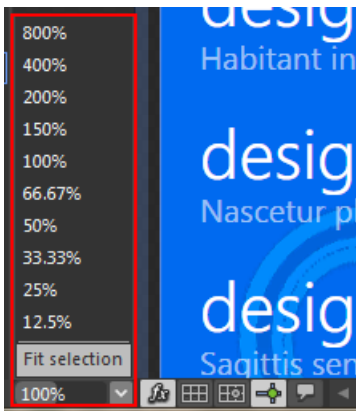


图 1-2 调整 Artboard 的缩放比例

注意，这里所说的整个 UI 是指手机屏幕所能显示的部分，而 Panorama 控件具有延伸到屏幕以外区域的特性，所以我们无法一次过把整个 Panorama 控件尽收眼底，这确实是一件憾事。

接着，我们来看看 Panorama 控件，如果你对它的效果没有感性认识，不妨到先看看 [WP7 的 6 个内置 Hub](#)。认识 Panorama 控件的最简单方法是结合 [Objects and Timeline 面板](#) 和 Artboard 来体验一下：

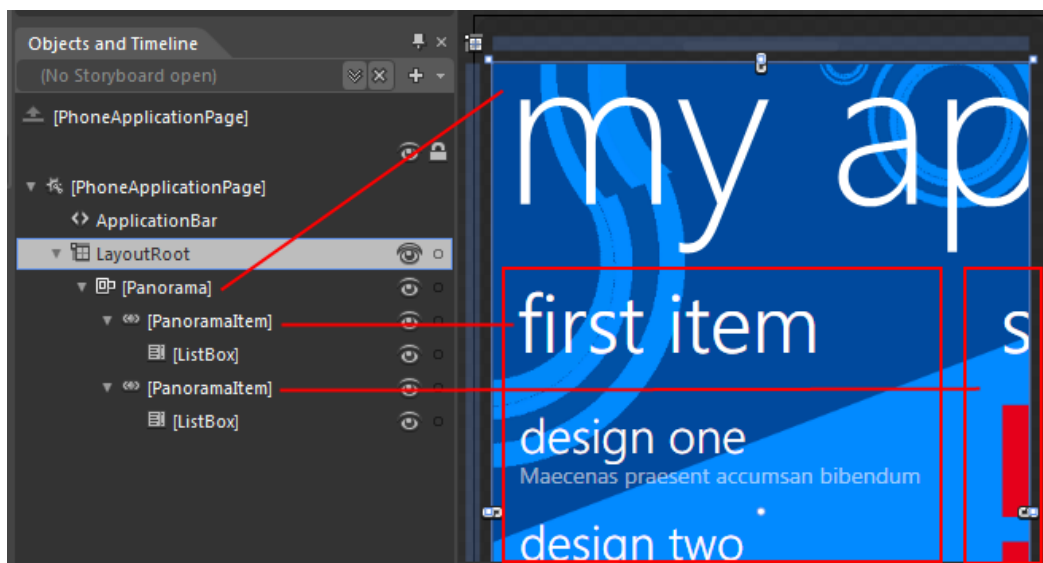


图 1-3 认识 Panorama 控件

如上图所示，每个 Panorama 控件都是由一个标题和若干 Panorama 项构成的，而每个 Panorama 项又会包含一个标题和一些内容，在这里，这些内容是通过 ListBox 来展示的，你可以根据实际的需要把它换成任何其它控件。此外，需要说明的是，Panorama 控件和 Panorama 项的标题都已经内化成自身的属性，只需通过 [Properties 面板](#) 设置就可以了，无需额外添加 TextBlock 或者其它控件。

目前，我们的 Panorama 控件包含了两个 Panorama 项，但从上图可以看到，只有第一个能完全显示出来（由于截图的关系，Artboard 的一部分隐藏在滚动条下面），而第二个只能看到一小部分，那么，如何才能显示第二个 Panorama 项，以便操作上面的控件呢？答案非常简单，只需在 Objects and Timeline 面板上单击第二个 Panorama 项，如图 1-4 所示，就可以了：



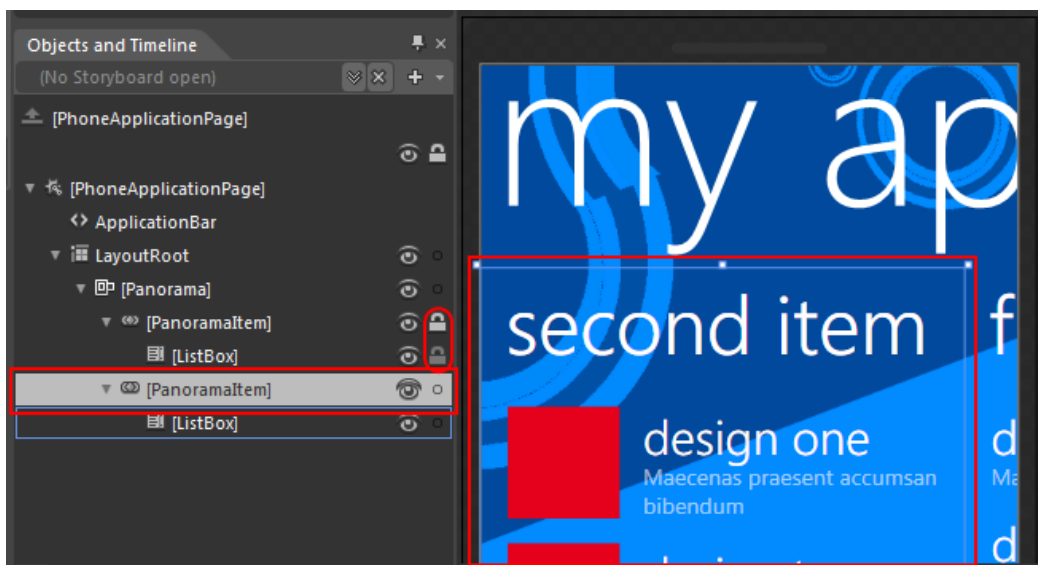


图 1-4 选择第二个 Panorama 项

值得提醒的是，为了在操作时不影响其它 Panorama 项，我们还可以通过 Objects and Timeline 面板把其它 Panorama 项锁定，正如上图所示的那样。

在继续阅读下面的内容之前，我强烈建议你稍稍暂停一下，把注意力集中在 Objects and Timeline 面板上，熟悉一下各个对象之间的关系，试着单击每个对象，然后看看它对应了 Artboard 上的哪个对象。如果你已经迫不及待想要亲自体验一下 Panorama 控件的效果，你现在可以按 F5 了。

### 1.1.3 修改 Panorama 控件

接下来，我们要执行以下任务：

- ✓ 修改 Panorama 控件的标题
- ✓ 去掉 Panorama 控件的背景
- ✓ 删除现有的两个 Panorama 项
- ✓ 添加一个新的 Panorama 项

第一个任务非常简单，确保 Objects and Timeline 面板上的 Panorama 控件处于选中状态，在 Properties 面板上的搜索框里输入 Title，第一个搜索结果就是我们要找的属性了，修改这个属性的值，如图 1-5 所示，然后按回车：

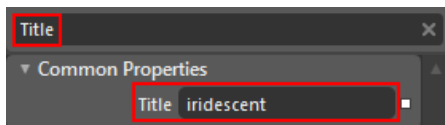


图 1-5 修改 Panorama 控件的 Title 属性

第二个任务也挺简单，在 Properties 面板上的搜索框里输入 Back，然后选择 No brush，如图 1-6 所示，就可以了：

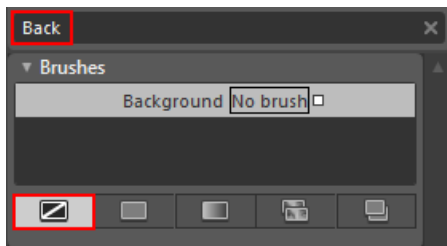


图 1-6 去掉 Panorama 控件的背景

第三个任务更简单，按下 Ctrl 键，依次选中两个 Panorama 项，然后按 Del 键就可以了。

最后一个任务是添加新的 Panorama 项，打开 [Assets 面板](#)，在搜索框里输入 Pan，如图 1-7 所示：

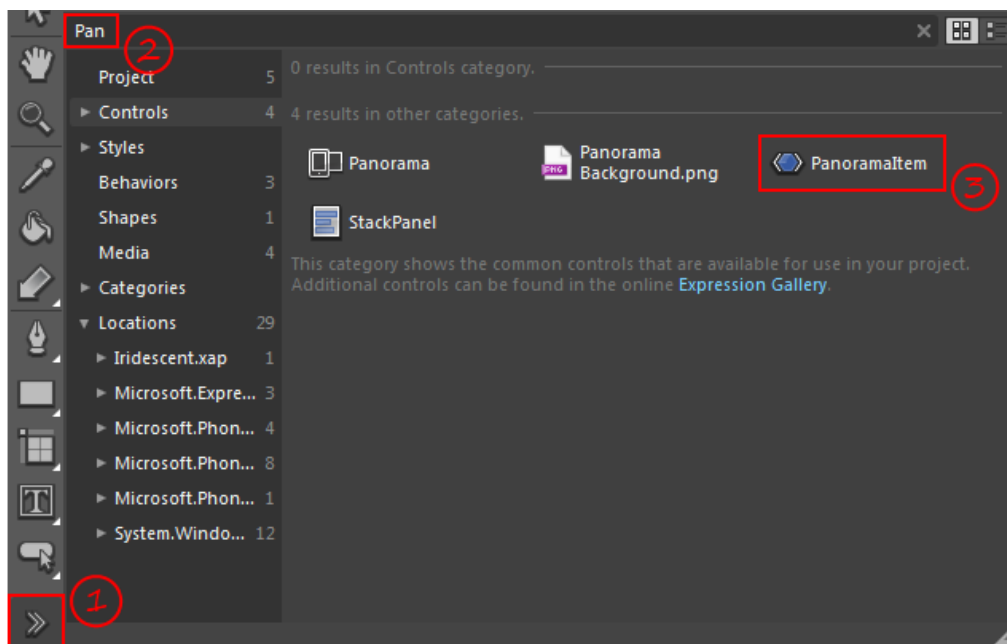


图 1-7 通过 Assets 面板添加 Panorama 项

然后把 Panoramaltem 拖到 Panorama 控件上就可以了。

注意，你可以把 Panoramaltem 拖到 Objects and Timeline 面板的 Panorama 控件上，也可以拖到 Artboard 的 Panorama 控件上，如果 Artboard 上的控件比较多，并且把 Panorama 控件挡住了，那么当你把 Panoramaltem 拖到 Artboard 上时，有可能会把它误加到其它控件上。这是添加控件的一般方法，针对添加 Panoramaltem，我们还有更简单的

方法，那就是右击 Panorama 控件，然后选择 Add Panoramaltem，如图 1-8 所示，就可以了：

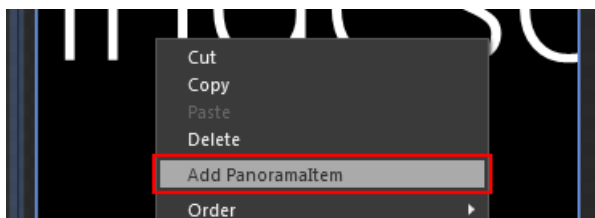


图 1-8 通过上下文菜单添加 Panorama 项

#### 1.1.4 测试应用

现在，向 Panorama 项添加一个 TextBlock，内容随你，调整一下位置和大小，然后按 F5：

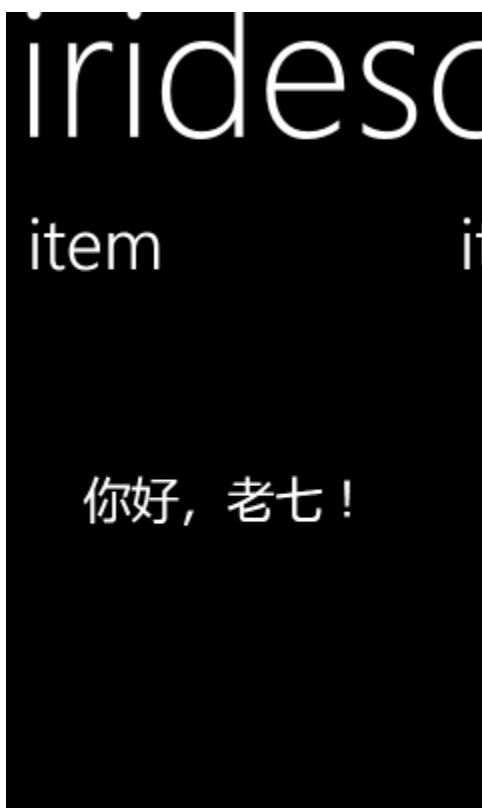


图 1-9 运行你的第一个 WP7 应用

一般地，Panorama 控件至少包含两个 Panorama 项，而这里只有一个，属于边界情况，细心观察上图，表面上，右边好像还有一个 Panorama 项，但当你从屏幕上向左滑动时，你会发现这其实是同一个 Panorama 项。那么向右滑动呢？情况一样。利用这个特点，我们可以创建一个简易计数器，把 Panorama 项的 TextBlock 绑定到一个计数变量上，当我们向左滑动时，计数变量加 1，向右时则减 1，其效果就像我们拥有一个无限延伸的 Panorama 控件，而边界情况就是这个计数变量的最大值和最小值，尽管如此，我们也无

需太过担心，假设计算变量的类型是 `Int32`，我相信没有人会向左或者向右滑动超过 20 亿次吧？如果你有兴趣的话，不妨把它当做课后练习。现在，按 **Back** 退出应用程序。

## 1.2 上课啦！

### 1.2.1 创建课程表页面

上课啦？什么课？哪里上？看到这些问题，有没有一种亲切的感觉？说不定你今天就问了我这些问题哦，那时你是不是在找课程表呢？如果课程表就在手机里该多好啊！事不宜迟了，我们自己弄一个吧。

右击 [Projects 面板](#) 里的项目节点，选择 **Add New Item**，如图 1-10 所示：

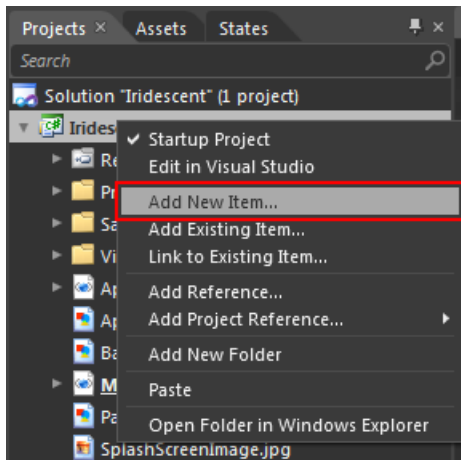


图 1-10 添加新项

在弹出的 **New Item** 对话框里选择 **Windows Phone Pivot Page**，输入页面的名字，如图 1-11 所示，然后按 **OK**：

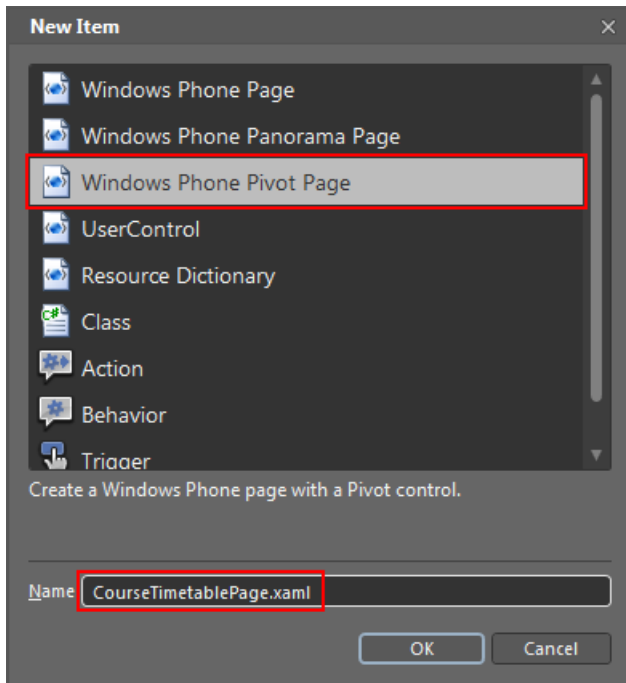


图 1-11 创建课程表页面

和 Panorama 页一样，Pivot 页也有一个充满整个页面的 Pivot 控件，刚创建好的 Pivot 控件默认附带两个 Pivot 项，我们可以把它们分别用于星期一和星期二。

确保 Pivot 控件处于选中状态，在 Properties 面板上寻找 Title 属性，并把它值改为“课程表”，如图 1-12 所示：

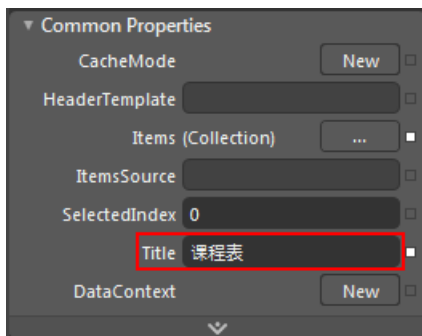


图 1-12 修改 Pivot 控件的 Title 属性

接着在 Objects and Timeline 面板上选中第一个 Pivot 项，在 Properties 面板上寻找 Header 属性，并把它值改为“星期一”，如图 1-13 所示：

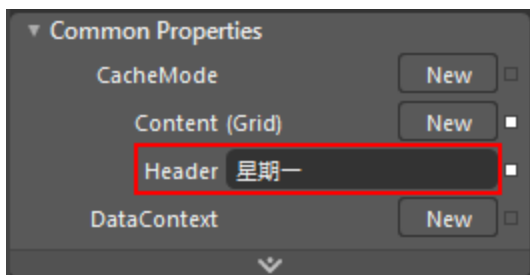


图 1-13 修改 Pivot 项的 Header 属性

完了之后把第二个 Pivot 项的 Header 属性值改为“星期二”。此时，你的 Pivot 控件应该有这样的：



图 1-14 课程表页面

嗯，看起来像个样了，然而，标题下面那么大的一块空位应该怎么办呢？毫无疑问，以列表的方式呈现一天的课程是比较适合的，但是，我希望列表的每一项除了显示课程名称之外，还能显示上课时间和上课地点。

### 1.2.2 导入示例数据

在继续设计 UI 之前，我们需要导入一些示例数据，以便在设计时就能看到最终效果。当然，你也可以让 Expression Blend 为你生成这些数据，不过，它无法为我们生成课程名称以及适合的上课地点，这样，当你在设计时调整控件外观时就会感到缺了点儿什么，而这正是使用真实数据的好处。

假设我们要导入下面这个 XML 文件的数据：

```
<?xml version="1.0" encoding="utf-8" ?>
<courses>
  <course name="公司法" starttime="8:10"
    endtime="10:00" location="综合楼802" />
  <course name="刑事诉讼法" starttime="10:10"
    endtime="11:50" location="综合楼802" />
  <course name="合同法" starttime="14:30"
    endtime="16:00" location="教学楼101" />
</courses>
```

代码 1-1 课程示例数据

我们可以在 Data 面板上单击 Create sample data 按钮，然后选择 Import Sample Data from XML，如图 1-15 所示：

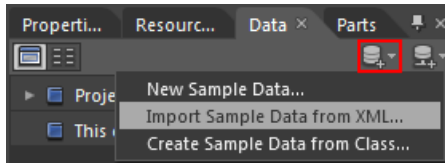


图 1-15 从 XML 文件导入示例数据

在弹出的 [Import Sample Data from XML 对话框](#)里，单击 Browse 按钮浏览并指定数据文件，如图 1-16 所示，然后按 OK 关闭对话框：

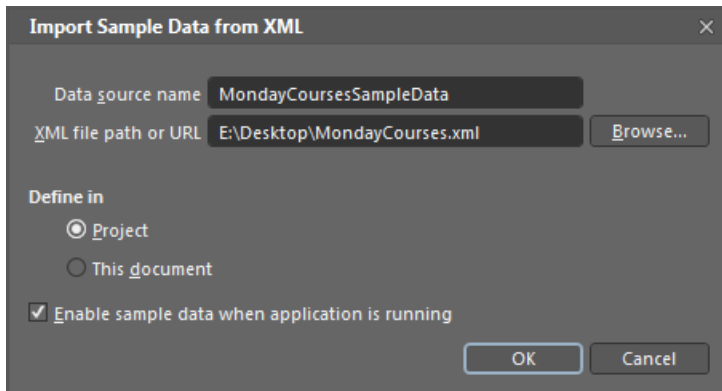


图 1-16 选择 XML 文件

此时，Expression Blend 会很努力地在后台帮你生成一大堆东西，等它做完之后，你会看到 Data 面板上多了一堆东西，现在，确保 Data 面板上的 List Mode 按钮处于按下状态，然后把 courseCollection 拖到 Pivot 项标题下面的空白处，如图 1-17 所示：

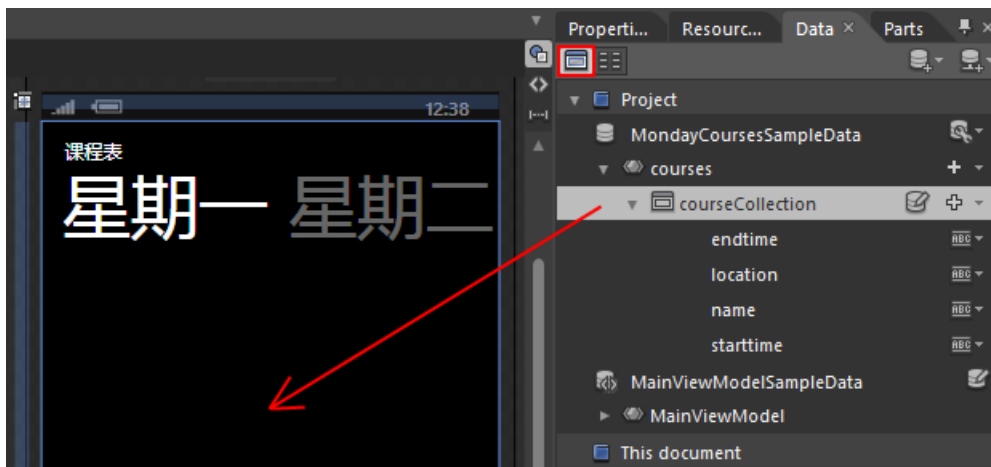


图 1-17 通过 Data 面板创建课程列表

此时，Expression Blend 会为你创建一个 ListBox，并把它的 ItemsSource 属性绑定到 courseCollection 上，现在，右击 ListBox 里的任何地方，然后选择 Auto Size\Fill，如图 1-18 所示，以便让 ListBox 充满整个 Grid（Pivot 项默认有一个 Grid 子元素）：

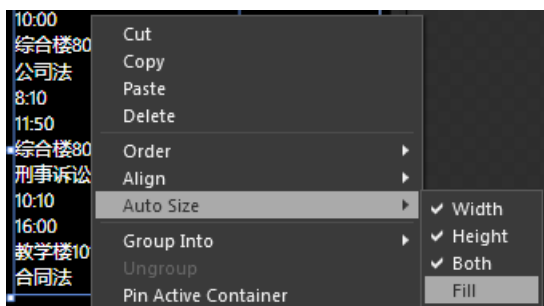


图 1-18 调整课程列表的大小

### 1.2.3 修改列表项的模板

嗯，不错，每个列表项都包含了课程名称、上课时间、下课时间以及上课地点，可是，这些内容各占一行，字体大小也是一样，每个列表项之间又没有明显的间距，显然不是那么好看，下面我们给它调整一下，右击 ListBox 里的任何地方，然后选择 Edit Additional Templates\Edit Generated Items (ItemTemplate)\Edit Current 进入列表项模板的编辑状态，如图 1-19 所示：

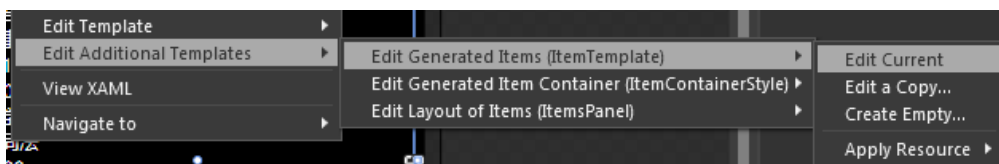


图 1-19 修改列表项的模板

此时，Objects and Timeline 面板会发生变化，如图 1-20 所示，上面的对象不再是我们之前看到的那些，而变成列表项里的对象：



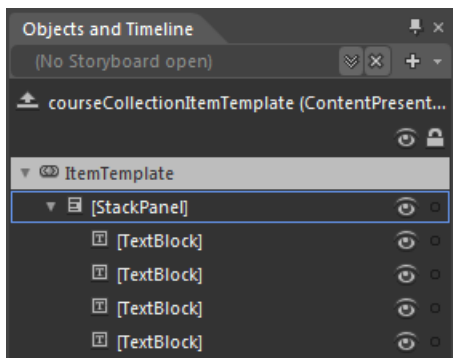


图 1-20 在 Objects and Timeline 面板上查看列表项的模板

从上图不难看出，每个列表项都包含了四个 TextBlock，这些 TextBlock 是用一个 StackPanel 装着的。现在，你可以发挥你的创造力，把它调整成你喜欢的样子，下面是我的调整结果：



图 1-21 修改后的模板

看到这里，你可能会问，怎么让上课时间和下课时间水平排列呢？很简单，你可以把它们放在一个 StackPanel 里，然后把 StackPanel 的 Orientation 属性的值设为 Horizontal 就行了。此时，Objects and Timeline 面板上面的对象应该是这样的：

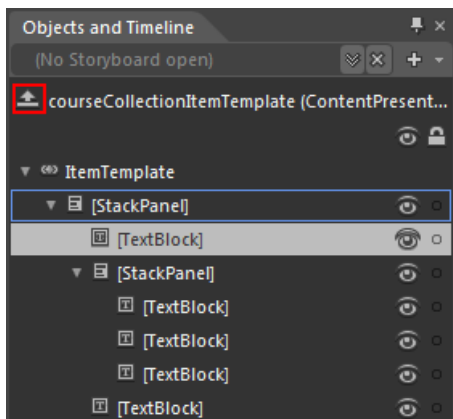


图 1-22 在 Objects and Timeline 面板上查看列表项的模板

单击上图红框那个箭头退出列表项模板的编辑状态。此时，Objects and Timeline 面板恢复“原状”了，你可以在上面看到 Pivot 控件和 Pivot 项。

#### 1.2.4 应用列表项的模板

选中第二个 Pivot 项，按照上面的步骤把星期二的课程数据导入，并把它拖到 Pivot 项上（注意，是第二个 Pivot 项哦），然后调整 ListBox 的大小，使之充满整个 Pivot 项。那么，列表项的显示方式怎么办？要重复编辑一次吗？当然不用！我们只需应用刚才那个就可以了。右击 ListBox 里的任何地方，然后选择 Edit Additional Templates\Edit Generated Items (ItemTemplate)\Apply Resource\courseCollectionItemTemplate，如图 1-23 所示：

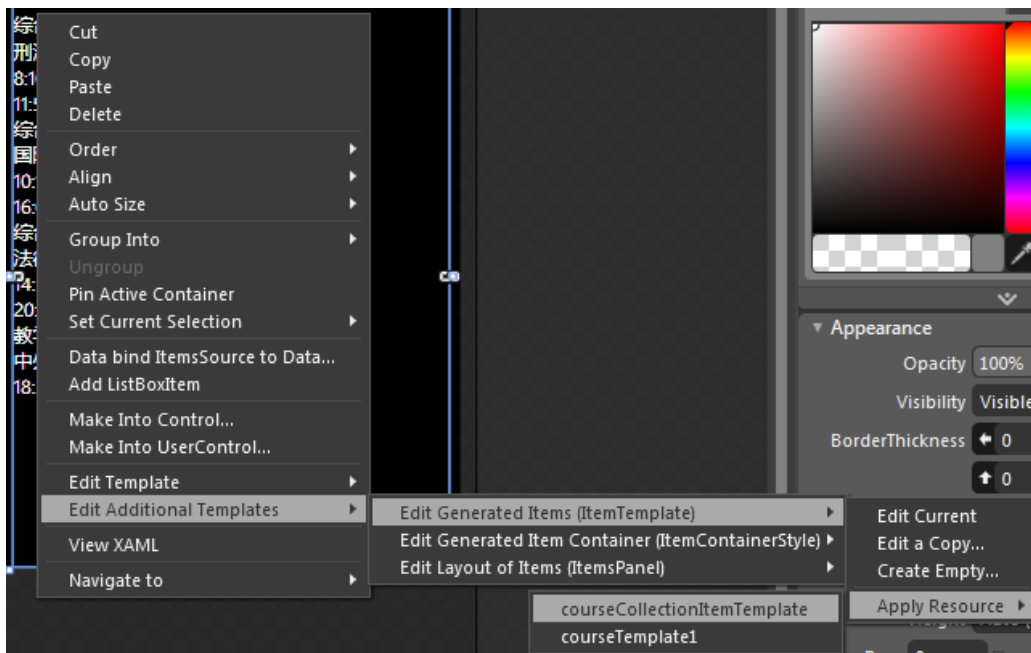


图 1-23 应用列表项的模板

此时，你会发现列表项的风格已经变成和前面的一样了：

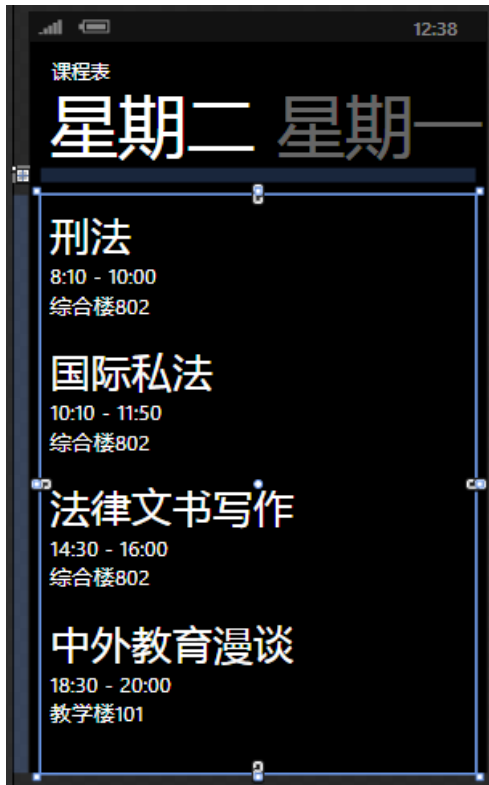


图 1-24 课程表页面

好了，我们现在可以把其它几天的课程都加上去，然后在主页（即第一个页面）添加一个按钮打开这个页面就……慢着！我怎么编辑课程表？

## 1.3 编辑课程表

### 1.3.1 添加 Application Bar

显然，如果这个课程表不能编辑，那么它就等同花瓶了，所以我们要为它增加编辑功能，包括新建、编辑和删除，我们可以把这些功能放在 Application Bar 上。

在 Expression Blend 里添加 Application Bar 非常简单，右击 Objects and Timeline 面板上的 PhoneApplicationPage，然后选择 Add ApplicationBar，如图 1-25 所示：

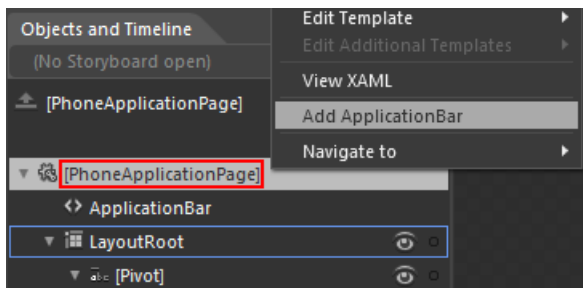


图 1-25 添加 Application Bar

接着右击 ApplicationBar，然后选择 Add ApplicationBarIconButton，如图 1-26 所示：

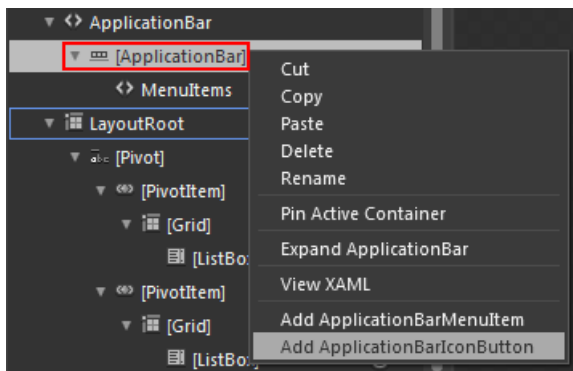


图 1-26 添加 Application Bar 按钮

确保刚才添加的 Application Bar 按钮处于选中状态，在 Properties 面板上把它的 IconUri 属性值改为 New，并把它的 Text 属性值改为“新建”，如图 1-27 所示：

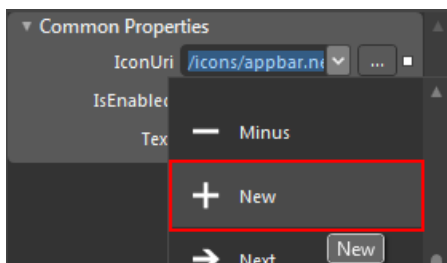


图 1-27 修改 Application Bar 按钮的属性

按照上面的步骤添加另外两个按钮，完成之后你的课程表页面应该是这样的：



图 1-28 课程表页面

删除功能只需获取选中的课程并把它删除就可以了，新建和编辑则不同，它们都需要另一个页面来处理，我们知道，新建功能和编辑功能在用户界面上的最大区别是前者的页面有内容而后者的没有，所以我们可以为它们创建一个共用页面。

### 1.3.2 添加新建/编辑课程页面

创建一个 Windows Phone Page，并把它命名为 `NewOrEditCoursePage.xaml`，如图 1-29 所示：

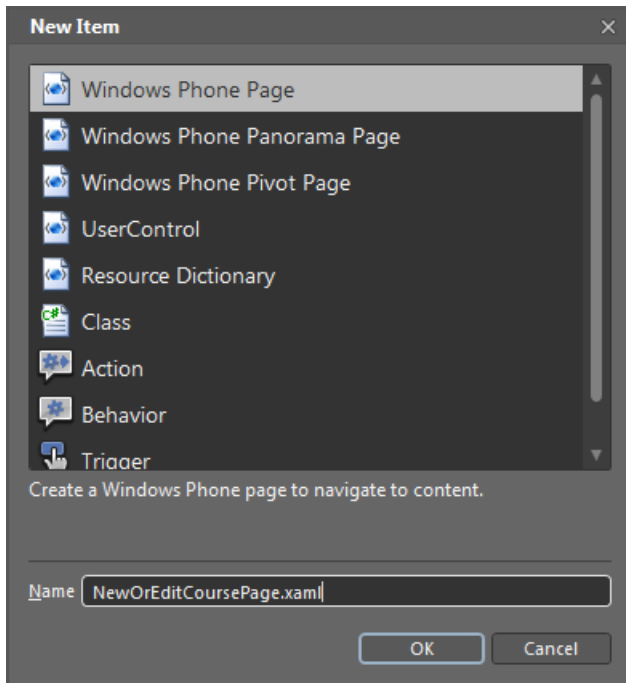


图 1-29 添加新建/编辑课程页面

完了之后把 ApplicationTitle 的 Text 属性值改为“课程表”，但 PageTitle 保留原样，如图 1-30 所示：



图 1-30 新建/编辑课程页面

因为新建功能和编辑功能共用同一个页面，所以 PageTitle 的 Text 属性值可能是新建课程或者编辑课程，这将会在打开此页面时通过传入参数设置。标题下面那块空地将会放置四个控件，分别对应课程名称、上课时间、下课时间和上课地点，首尾两个将会是 TextBox 控件，而中间两个将会是 [Silverlight for Windows Phone Toolkit](#) 的 TimePicker 控件。

右击 Projects 面板上的 References 节点，选择 Add Reference，如图 1-31 所示：

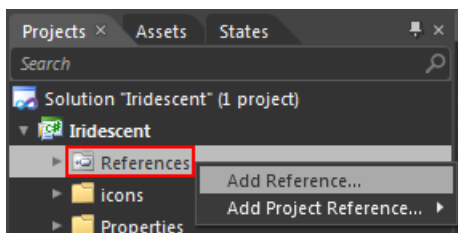


图 1-31 添加引用

在弹出的 Add Reference 对话框里把 C:\Program Files\Microsoft SDKs\Windows Phone\v7.0\Toolkit\Sep10\Bin\Microsoft.Phone.Controls.Toolkit.dll 引用进来。

添加完引用之后就可以把控件添加到页面了，完成之后的页面如图 1-32 所示：



图 1-32 新建/编辑课程页面

需要说明的是，TimePicker 控件已经自带标题功能，你只需设置它的 Header 属性就可以了，而普通的 TextBox 没有标题功能，只能自行添加 TextBlock 来模拟，为了使它的颜色和

TimePicker 控件的标题的颜色一样，我们需要把它的 Foreground 属性值改为 PhoneSubtleBrush。

此时，我的 Objects and Timeline 面板是这样的：

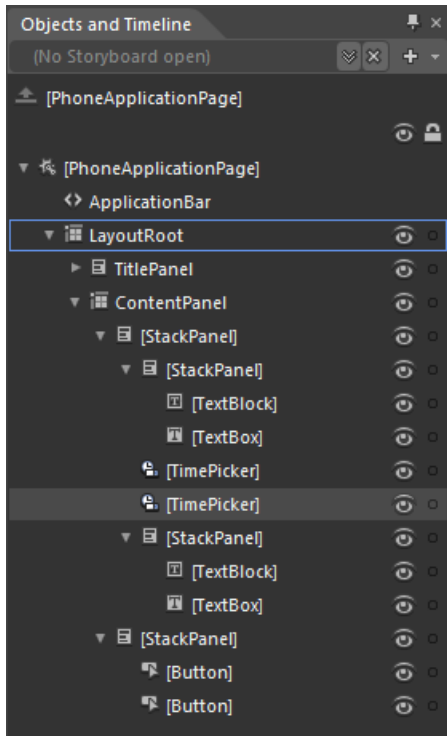


图 1-33 从 Objects and Timeline 面板上查看页面

好了，课程表的用户界面已经设计完了，是不是很想看看运行效果呢？没问题！

### 1.3.3 添加打开页面的代码

回到 CourseTimetablePage 页，在 Objects and Timeline 面板上选中第一个 Application Bar 按钮，然后在 Properties 面板上单击 Events，并双击 Click 旁边的编辑框，如图 1-34 所示：

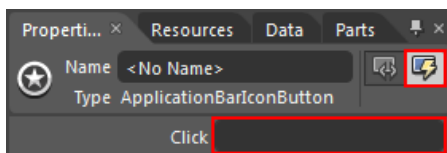


图 1-34 双击 Click 旁边的编辑框

此时，Expression Blend 会打开 CourseTimetablePage 页的代码隐藏文件，并为 Application Bar 按钮添加一个事件处理程序方法，我们只需在 TODO 下面加上一句，如代码 1-2 所示就可以了：



```
private void AppBarIconButton_Click(object sender, System.EventArgs e)
{
    // TODO: Add event handler implementation here.
    NavigationService.Navigate(
        new Uri("/NewOrEditCoursePage.xaml", UriKind.RelativeOrAbsolute));
}
```

代码 1-2 添加打开页面的代码

接着，回到 MainPage 页，把上面的 TextBlock 去掉，拖一个 Button 到中间，然后右击这个 Button，选择 Navigate To\CourseTimetablePage，如图 1-35 所示：



图 1-35 为按钮添加导航

### 1.3.4 测试应用

好了，按 F5 吧：



图 1-36 运行应用

当你单击屏幕中间那个按钮时，课程表就会显示：



图 1-37 课程表页面

向左或者向右滑动屏幕可以在不同的 Pivot 项之间来回切换，而向上或者向下滑动屏幕则可以查看当天课程。

当你单击 Application Bar 上的新建按钮时，新建课程表的界面将会显示：

课程表

page name

课程名称

上课时间

12:14 PM

下课时间

12:14 PM

上课地点

确定 取消

图 1-38 新建/编辑课程页面

你可以输入课程名称和上课地点。

当你单击上课时间或者下课时间下面那个 `TimePicker` 控件时，设置时间的界面将会显示：



图 1-39 选择时间

你可以通过滑动设置时间，这个界面下面有两个 Application Bar 按钮，左边那个是确定，右边那个是取消，但为什么这两个图标是一样的？

其实它是找不到图标才这样的，如果你下载了它的代码，你会在 [TimePickerPage.xaml](#) 里看到它已经把图标位置硬性规定为 `/Toolkit.Content/ApplicationBar.Check.png` 和 `/Toolkit.Content/ApplicationBar.Cancel.png` 了，你可以在 `PhoneToolkitSample\Toolkit.Content` 文件夹里找到这两个图标，在项目里创建一个 `Toolkit.Content` 文件夹，把它们复制进去，并把它们的 Build Action 设置为 Content，重新运行就能看到了。

到目前为止，我们只写了一行代码（其实这行代码也可以省掉的），应用程序的功能和操作就基本上体现出来了，此时，你可能会问，在我们设计用户界面的过程里，Expression Blend 到底在背后为我们做了什么呢？

#### 1.4 Expression Blend 如何提供设计时数据的支持？

首先，我们来看看 Expression Blend 为我们生成了哪些文件：

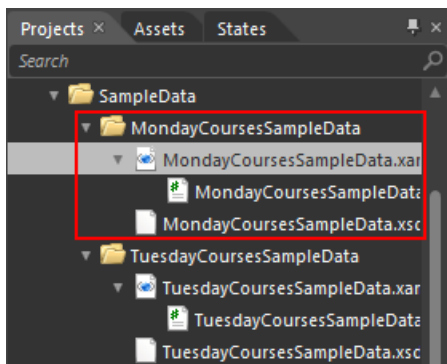


图 1-40 示例数据文件

如上图所示，由于我们的数据是从 XML 文件导入的，所以你会看到一个 XSD 文件，这是从我们那个 XML 文件生成的 XML Schema，这个 XSD 文件将会用来生成相关的类，这些类都放在对应的 C# 文件里，而我们的数据最终是以 XAML 的形式存在的。

当我们导入 XML 数据时，Expression Blend 不但为我们生成这些文件，还在 App.xaml 的 Application.Resource 里添加了相应的对象，如代码 1-3 所示：

```
<Application.Resources>
    <SampleData1:courses x:Key="TuesdayCoursesSampleData" d:IsDataSource="True"/>
    <SampleData:courses x:Key="MondayCoursesSampleData" d:IsDataSource="True"/>
</Application.Resources>
```

代码 1-3 示例数据对象

为什么添加到 App.xaml 而不是某个页面的 XAML 文件里呢？这是因为我们在 Import Sample Data from XML 对话框里选择了 Define in Project（参见图 16），如果我们当时选择 Define in This document，它就会添加到某个页面的 XAML 文件里。

那么，Expression Blend 又是如何得知我们的数据保存在哪个 XAML 文件里呢？答案就在 courses 类的构造函数里，如代码 1-4 所示：

```

public courses()
{
    try
    {
        System.Uri resourceUri =
            new System.Uri(
                "/Iridescent;component/SampleData/" +
                "MondayCoursesSampleData/MondayCoursesSampleData.xaml",
                System.UriKind.Relative);
        if (System.Windows.Application.GetResourceStream(resourceUri) != null)
        {
            System.Windows.Application.LoadComponent(this, resourceUri);
        }
    }
    catch (System.Exception)
    {
    }
}

```

代码 1-4 获取示例数据文件

这个 URI 看起来有点古怪，如果你用 Visual Studio 打开这个项目，你将会看到这个 XAML 文件的 Build Action 是 Page，事实上，它会被编译成 BAML，并且嵌到 Iridescent 程序集里，这种古怪的 URI 就是引用程序集内嵌资源的表示方式。

接着，当我们从 Data 面板把 courseCollection 拖到 Pivot 项上时（参见图 17），Expression Blend 会把它所属的 MondayCoursesSampleData 绑到 Pivot 控件的父容器的 DataContext 属性上，在当前 Pivot 项的子容器里创建一个 ListBox，并把 courseCollection 绑到它的 ItemsSource 属性上，如代码 1-5 所示：

```

<Grid x:Name="LayoutRoot" Background="Transparent"
    DataContext="{Binding Source={StaticResource MondayCoursesSampleData}}">
    <!--Pivot Control-->
    <controls:Pivot Title="课程表">
        <!--Pivot item one-->
        <controls:PivotItem Header="星期一">
            <Grid>
                <ListBox
                    ItemTemplate="{StaticResource courseCollectionItemTemplate}"
                    ItemsSource="{Binding courseCollection}"/>
            </Grid>
        </controls:PivotItem>
    </controls:Pivot>

```

代码 1-5 绑定示例数据

还记得 Import Sample Data from XML 对话框里有个选项是 Enable sample data when application is running 吗？当时它是选中的，如果我们把这个选项去掉，DataContext 属性前面就会多一个“d:”，如代码 1-6 所示：

```
<Grid x:Name="LayoutRoot" Background="Transparent" d:DataContext="{Binding Sou
```

#### 代码 1-6 设计时绑定

这个前缀告诉编译器在生成最终程序集时忽略这个属性，这样你就不会在程序运行的时候看到这些数据了。

如果你细心观察 Projects 面板，可能会发现几个我们从未提及过的文件，如图 1-41 所示：

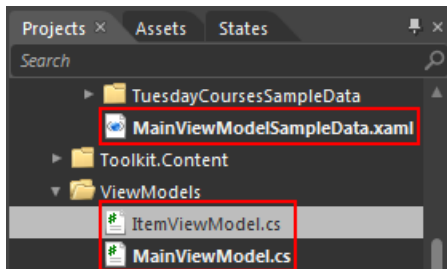


图 1-41 未提及过的示例数据文件

它们是干嘛的呢？事实上，这些文件在我们刚创建完项目时就存在了，还记得最初的 Panorama 页吗（参见图 1-3 和图 1-4），里面的 Panorama 项是有数据的，而这些数据就是来自 MainViewModelSampleData.xaml 文件的。那么，这些数据又是如何关联到 Panorama 控件的呢？打开 MainPage.xaml，在文件的顶部，你会发现它的蛛丝马迹，如代码 1-7 所示：

```
<phone:PhoneApplicationPage xmlns="http://schemas.microsoft.com/winfx/2006,
    x:Class="Iridescent.MainPage"
    d:DataContext="{d:DesignData SampleData/MainViewModelSampleData.xaml}"
    FontFamily="{StaticResource PhoneFontFamilyNormal}"
```

#### 代码 1-7 绑定示例数据文件

这里使用的不是我们常见的 Binding，而是 d:DesignData，正如你所看到的，它用来指定数据文件的位置，但这些数据是在设计时使用的，所以你会看到 DataContext 前面有个“d:”，事实上，我们把这些带有“d:”前缀的属性称为[设计时属性](#)。

和这些数据相关的类分别定义在 MainViewModel.cs 和 ItemViewModel.cs 文件里。如果你打开 MainViewModel.cs，你会发现里面的 LoadData 方法包含了另一份不同的数据，为什么，它们分别用来干嘛的？我给你截两个图，看你能否找到什么蛛丝马迹：

```
<local:MainViewModel.Items>
    <local:ItemViewModel LineOne="design one" L:
    <local:ItemViewModel LineOne="design two" L:
    <local:ItemViewModel LineOne="design three"
```

#### 代码 1-8 设计时示例数据

```
Items.Add(new ItemViewModel() { LineOne = "runtime one", Li
Items.Add(new ItemViewModel() { LineOne = "runtime two", Li
Items.Add(new ItemViewModel() { LineOne = "runtime three",
```

代码 1-9 运行时示例数据

看到了吗？XAML 文件里每个 ItemViewModel 对象的 LineOne 属性值都是“design XXX”，而 C#文件的则是“runtime XXX”，事实上，这已经道出它们的用途了，XAML 文件里的数据是设计时使用的，而 C#文件里的则是运行时使用的。

但是，MainPage.xaml 的根元素的 DataContext 属性前面有“d:”前缀啊，之前不是说编译器会在生成程序集的时候忽略它吗，那应用程序又如何找到运行时使用的数据呢？答案就在 MainPage.cs 里，如代码 1-10 所示：

```
// Constructor
public MainPage()
{
    InitializeComponent();

    // Set the data context of the listbox control to the sample data
    DataContext = App.ViewModel;
    this.Loaded +=new RoutedEventHandler(MainPage_Loaded);
}

// Load data for the ViewModel Items
private void MainPage_Loaded(object sender, RoutedEventArgs e)
{
    if (!App.ViewModel.IsDataLoaded)
    {
        App.ViewModel.LoadData();
    }
}
```

代码 1-10 创建 ViewModel 对象并加载数据

App 类的 ViewModel 是一个静态属性，它的任务只是创建一个 MainViewModel 对象，当 MainPage 的构造函数被调用时，会把这个 MainViewModel 对象绑到自己的 DataContext 属性上，当 Loaded 事件触发时，如果数据还没装载，就调用 LoadData 方法装载数据。

从上面的讨论不难看出，Expression Blend 的确为我们做了很多，而这些知识将会协助我们完成后面的开发任务。

## 保存课程表



### 1.4.1 创建 Course 类

说了那么多前端的东西，是时候看看后端了。课程表软件说到底就是一个管理课程表数据的软件，所以数据存储是一个非常重要的环节。如果说用户界面的设计是 Expression Blend 的强项，那么业务逻辑的开发就是 Visual Studio 的地盘，既然接下来的着眼点是后端，当然要切换到 Visual Studio 啦。

右击 Projects 面板里的解决方案节点，选择 Edit in Visual Studio，如图 1-42 所示：

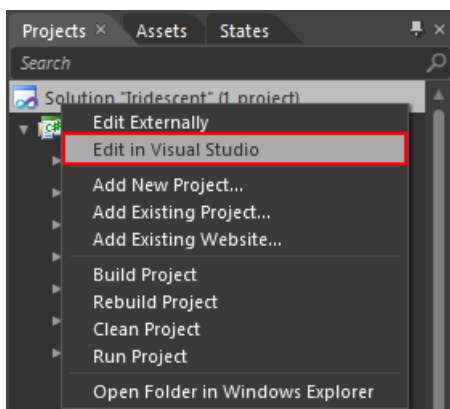


图 1-42 在 Expression Blend 里打开 Visual Studio

此时，Visual Studio 会打开，接下来，我们将会 Visual Studio 里完成后端部分的开发。

首先，创建一个 Models 文件夹，在里面添加一个 Course 类，并让它实现 INotifyPropertyChanged 接口，如代码 1-11 所示：

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

    private void OnPropertyChanged(string propertyName)
    {
        if (PropertyChanged != null)
        {
            PropertyChanged(this,
                new PropertyChangedEventArgs(propertyName));
        }
    }
}
```

代码 1-11 Course 类

接着，我们需要为它添加如下属性：

属性名字	属性类型	备注
Name	string	课程名称
Day	string	星期几
StartTime	DateTime	上课时间
EndTime	DateTime	下课时间
Location	string	上课地点

表 1-1 Course 类的属性

这些属性实现起来并不难，但我们不能使用 C# 3.0 的自动属性来实现，因为我们需要在属性的 set 访问器里调用 OnPropertyChanged 方法，下面我们选择 Name 属性来示范一下：

```
private string _name;
public string Name
{
    get { return _name; }
    set
    {
        _name = value;
        OnPropertyChanged("Name");
    }
}
```

代码 1-12 Name 属性

如果你觉得在创建这些属性时手动输入所有代码太麻烦，你也可以[创建一个代码段](#)，然后通过代码段来添加属性。

### 1.4.2 创建 JsonCourseStore 类

接下来我们需要考虑一下以什么方式来存储课程表的数据，如果不考虑使用第三方类库的话，我们至少有 JSON 序列化、XML 序列化以及 LINQ to XML 等几种常见的方式，而无论我们今天的选择是什么，将来都有可能换用别的方式，既然我们已经预见了变化，就应该通过抽象把它们隔离开来。

创建一个 Services 文件夹，在里面创建一个 ICourseStore 接口，那么，我们又应该如何设计这个接口呢？毫无疑问，它应该以某种形式满足 CRUD 四种需求，那么，这种形式又是怎样的呢？回忆一下上一节的内容以及前面的用户界面，你觉得我们会对这个接口有什么期望？显然，如果我们能够获取一组 Course 对象，并把它们绑到 Pivot 项里的 ListBox 就好了。这个好办，我们可以在 ICourseStore 接口里声明一个 Courses 属性，类型是 ObservableCollection<Course>，这样，数据绑定可以协助我们满足“R”的需求，当我们从用户界面上选中一个课程，单击 Application Bar 上的删除按钮，我们可以从 ListBox 上获取当前选中的 Course 对象，并从 Courses 属性里删除，这样，“D”的需求也得到满足了，剩下的就是“C”和“U”了，它们有一个共同的地方，就是都要打开一个新的页面，不同的是前者对应一个新的 Course 对象，而后者则对应当前选中的 Course 对象，当用户单击确定之后，这些改动就会提交到 Courses 属性，换句话说，四种需求都得到满足了。但是，

这里好像没有提到把数据保存到独立存储区啊？没错，所以 `ICourseStore` 接口还需要提供一个 `Commit` 方法，把数据提交到独立存储区。此外，我们还可以提供一个 `Rollback` 方法，把 `Courses` 属性上的数据还原为独立存储区上的数据，这样，如果用户不小心把课程表改乱了，也无需通过重启应用程序来重新加载独立存储区上的数据。

好了，`ICourseStore` 接口确定下来了，如代码 1-13 所示：

```
public interface ICourseStore
{
    ObservableCollection<Course> Courses { get; }

    void Rollback();

    void Commit();
}
```

代码 1-13 `ICourseStore` 接口

那么，接口的实现呢？刚才我们提到了三种常见的存储方式，我个人比较喜欢 LINQ to XML，不过今天我想试一下 JSON 序列化。

首先，在 `Services` 文件夹里创建一个 `JsonCourseStore` 类，并让它实现 `ICourseStore` 接口，如代码 1-14 所示：

```
public class JsonCourseStore : ICourseStore
{
    public ObservableCollection<Course> Courses
    {
        get;
        private set;
    }

    public void Rollback()
    {
    }

    public void Commit()
    {
    }
}
```

代码 1-14 `JsonCourseStore` 类

接着，引用以下类库和命名空间：

类库/命名空间
System.ServiceModel.Web.dll
System.IO
System.IO.IsolatedStorage
System.Runtime.Serialization.Json

表 1-2 引用类库和命名空间

好了，现在可以开始实现 `JsonCourseStore` 类了。当应用程序启动时，会调用 `JsonCourseStore` 类的构造函数，从独立存储区把课程表的数据反序列化到 `Courses` 属性里；而当用户执行回滚操作时，会调用 `Rollback` 方法，从独立存储区把课程表的数据反序列化出来，替换 `Courses` 属性的现有数据。感觉上，这两个方法所做的事情不尽相同，然而，当你开始实际的编码时，就会发现它们的代码其实是一样的，按照惯例，我们会把这部分代码提取到一个辅助方法里。

在 `JsonCourseStore` 类里创建一个 `LoadCoursesFromIsolatedStorage` 方法，具体实现如代码 1-15 所示：

```
private void LoadCoursesFromIsolatedStorage()
{
    using (var fileStore =
        IsolatedStorageFile.GetUserStoreForApplication())
    {
        if (fileStore.FileExists("Courses.json"))
        {
            using (var fileStream =
                fileStore.OpenFile("Courses.json", FileMode.Open))
            {
                var jsonSerializer =
                    new DataContractJsonSerializer(
                        typeof(ObservableCollection<Course>));
                Courses = (ObservableCollection<Course>)
                    jsonSerializer.ReadObject(fileStream);
            }
        }
        else
        {
            Courses = new ObservableCollection<Course>();
        }
    }
}
```

代码 1-15 LoadCourseFromIsolatedStorage 方法

这样，构造函数和 Rollback 方法只需调用这个辅助方法就行了。值得提醒的是，在应用程序首次启动时，课程表的数据文件还不存在，此时，我们只需为 Courses 属性创建一个空的集合就行了。

当用户执行提交操作时，会调用 Commit 方法，把 Courses 属性序列化到独立存储区，如代码 1-16 所示：

```
public void Commit()
{
    using (var fileStore =
        IsolatedStorageFile.GetUserStoreForApplication())
    using (var fileStream =
        fileStore.OpenFile("Courses.json", FileMode.Create))
    {
        var jsonSerializer =
            new DataContractJsonSerializer(
                typeof(ObservableCollection<Course>));
        jsonSerializer.WriteObject(fileStream, Courses);
    }
}
```

代码 1-16 Commit 方法

好了，数据存储的开发也完成了，但是，我们怎么把它和前面设计的用户界面关联起来呢？这正是接下来要讲的。

## 1.5 连接前端和后端

### 1.5.1 弄清页面和抽象模型之间的关系

还记得 Expression Blend 是怎么做的吗？它为 MainPage 页创建一个与之对应的 MainViewModel 类，并在代码隐藏文件里把后者的实例绑到前者的 DataContext 属性上，而剩下的事情就交给数据绑定来处理。接下来，我们将会模范这种做法，把前端和后端连接起来。

我们知道，一个课程表包含若干列，每列都包含了一个标题和一组当天的课程，整个课程表对应于 CourseTimetablePage 页，里面的每列对应于一个 Pivot 项，其中，列的标题将会作为 Pivot 项的标题显示，而列所包含的那组当天的课程将会在 Pivot 项所包含的 ListBox 里显示。为了方便理解，我们把它们之间的映射关系制成下表：

页面	页面的抽象模型
CourseTimetablePage 页	CourseTimetableViewModel 类
Pivot 项	CourseTimetableColumnViewModel 类
Header 属性	Header 属性
ListBox 控件	Courses 属性

表 1-3 页面和抽象模型之间的映射关系

### 1.5.2 创建 CourseTimetableColumnViewModel 类

我们先来看看 CourseTimetableColumnViewModel 类。在 ViewModels 文件夹里创建一个 CourseTimetableColumnViewModel 类，并让它实现 INotifyPropertyChanged 接口。然后为它添加一个 Header 属性，如代码 1-17 所示：

```
private string _header;
public string Header
{
    get { return _header; }
    set
    {
        _header = value;
        OnPropertyChanged("Header");
    }
}
```

代码 1-17 Header 属性

这个属性将会在构造函数里初始化，如代码 1-18 所示：

```
public CourseTimetableColumnViewModel(string header)
{
    _header = header;
}
```

代码 1-18 在构造函数里初始化 \_header 字段

这些都不难，稍微有点麻烦的是 Courses 属性，我们该如何实现这个属性呢？

目前，JsonCourseStore 类的 Courses 属性返回的是全部课程，而 CourseTimetableColumnViewModel 类的 Courses 属性仅返回当天的课程，显然，我们要做过滤，此外，由于 JsonCourseStore 类返回的课程没有固定顺序，我们还要排序。一个可能的方案是通过 LINQ 从源集合筛选当天课程，并根据时间进行排序，然后添加到目标集合，我们还需要监听源集合的相关事件，以便在源集合的内容发生更改时把更改反映到目标集合。感觉上这个方案需要不少代码，还有没有别的方案呢？

有！我们可以使用 CollectionViewSource 类，它能根据我们指定的过滤和排序条件提供集合视图，我们可以通过它的 View 属性访问这个视图，如代码 1-19 所示：

```
private CollectionViewSource _courses;
public ICollectionView Courses
{
    get { return _courses.View; }
}
```

代码 1-19 Courses 属性

\_courses 的初始化也是在构造函数里进行，如代码 1-20 所示：

```
_courses = new CollectionViewSource();
_courses.Source = ?
_courses.Filter +=
    (o, e) => e.Accepted = ((Course)e.Item).Day == header;
_courses.SortDescriptions.Add(
    new SortDescription("StartTime", ListSortDirection.Ascending));
```

代码 1-20 在构造函数里初始化 \_courses 字段

现在，请思考一下，我们是不是直接给 Source 属性创建一个 JsonCourseStore 对象呢？回忆一下 JsonCourseStore 类的设计思路，CRUD 四个操作都是直接在它的 Courses 属性上进行的，而 C 和 U 这两个操作是发生在另一个页面的（NewOrEditCoursePage 页），这意味着我们需要一个全局的 JsonCourseStore 对象。

解决方案有很多，你可以把 JsonCourseStore 类设计成单例模式，也可以使用依赖注入容器，而最简单的做法莫过于在 App 类里添加一个静态属性，如代码 1-21 所示：

```
private static ICourseStore _courseStore = null;
public static ICourseStore CourseStore
{
    get
    {
        if (_courseStore == null)
        {
            _courseStore = new JsonCourseStore();
        }

        return _courseStore;
    }
}
```

代码 1-21 CourseStore 静态属性

这样，Source 属性的初始化就可以通过下面这句来完成了：

```
_courses.Source = App.CourseStore.Courses;
```

代码 1-22 设置 `_courses` 字段的 `Source` 属性

诚然，这并不是什么好的做法，你可能会坚持使用依赖注入容器，因为这样能更好的降低对象之间的耦合度，如果你已经知道怎么做，请不要犹豫，立即行动，鉴于这篇文章的内容已经很多了，所以我想把这个内容留到后面的文章。

### 1.5.3 创建 `CourseTimetableViewModel` 类

接着来看 `CourseTimetableViewModel` 类，在 `ViewModels` 文件夹里创建一个 `CourseTimetableViewModel` 类，并让它实现 `INotifyPropertyChanged` 接口。毫无疑问，它应该包含一组 `CourseTimetableColumnViewModel` 对象，我们可以创建一个 `Columns` 属性来存放它们，如代码 1-23 所示：

```
public ObservableCollection<CourseTimetableColumnViewModel> Columns
{
    get;
    private set;
}
```

代码 1-23 `Columns` 属性

此外，我们还需要一个属性来跟踪和当前显示的 `Pivot` 项绑定的是哪个 `CourseTimetableColumnViewModel` 对象，我们可以创建一个 `SelectedColumnIndex` 属性来负责这项工作，如代码 1-24 所示：

```
private int _selectedColumnIndex;
public int SelectedColumnIndex
{
    get { return _selectedColumnIndex; }
    set
    {
        _selectedColumnIndex = value;
        OnPropertyChanged("SelectedColumnIndex");
    }
}
```

代码 1-24 `SelectedColumnIndex` 属性

这个属性将会和 `Pivot` 控件的 `SelectedIndex` 属性进行双向绑定。这两个属性都会在构造函数里初始化，如代码 1-25 所示：



```

public CourseTimetableViewModel()
{
    Columns = new ObservableCollection<CourseTimetableColumnViewModel>
    {
        new CourseTimetableColumnViewModel("星期日"),
        new CourseTimetableColumnViewModel("星期一"),
        new CourseTimetableColumnViewModel("星期二"),
        new CourseTimetableColumnViewModel("星期三"),
        new CourseTimetableColumnViewModel("星期四"),
        new CourseTimetableColumnViewModel("星期五"),
        new CourseTimetableColumnViewModel("星期六"),
    };

    SelectedColumnIndex = (int)DateTime.Today.DayOfWeek;
}

```

代码 1-25 在构造函数里初始化 Columns 和 SelectedColumnIndex 两个属性

我们知道，用户体验很重要，我们应该尽可能减少用户获取所需信息的步骤，当用户打开课程表时最想看到的应该是今天的课程，这正是为什么我要把 SelectedColumnIndex 属性的值初始化为 DateTime.Today.DayOfWeek 属性的值。然而，当一种观点产生的时候，反面观点也会随之而来，有人可能建议每次打开课程表都能看到相同的东西，比如说，显示一周第一天的课程，也有人建议和上次离开该页面时的东西保持一致。事实上，这些观点没有谁对谁错，它们的目的是为了改善用户体验，你可以按照你认为正确的去做，或许，我们也可以考虑在后续的版本里通过选项设置让用户选择。

#### 1.5.4 把 ViewModel 类关联到对应的页面

现在，我们可以着手处理绑定了。打开 CourseTimetablePage.xaml，切换到 XAML 模式，把现有的两个 Pivot 项删掉或者注释掉，因为我们不再单独处理个别 Pivot 项，而是把 CourseTimetableViewModel 类的 Columns 属性绑到 Pivot 控件的 ItemsSource 属性上，由 Pivot 控件动态创建 Pivot 项。此外，我们还要把 CourseTimetableViewModel 类的 SelectedColumnIndex 属性绑到 Pivot 控件的 SelectedIndex 属性上，并设置为双向绑定。设置好绑定后，Pivot 控件的 XAML 应该是这样的：

```

<controls:Pivot Title="课程表" ItemsSource="{Binding Columns}"
    SelectedIndex="{Binding SelectedColumnIndex, Mode=TwoWay}">
</controls:Pivot>

```

代码 1-26 绑定 Columns 和 SelectedColumnIndex 两个属性

但是，Pivot 控件又从何得知应该创建怎样的 Pivot 项呢？事实上，它无从知晓，也不知道该如何使用 CourseTimetableColumnViewModel 类 Columns 属性，所以我们必须通过某种方式告诉它我们希望它怎么做，而这种方式就是数据模板。我们需要创建两个数据模板，

一个用于 Pivot 项的标题，另一个用于 Pivot 项的内容，这两个数据模板将会放在页面的资源字典里。

第一个数据模板很简单，只需创建一个 TextBlock，并把 Header 属性绑到它的 Text 属性上，如代码 1-27 所示：

```
<DataTemplate x:Key="pivotItemHeaderTemplate">
    <Grid>
        <TextBlock Text="{Binding Header}" />
    </Grid>
</DataTemplate>
```

代码 1-27 Pivot 项的标题模板

另一个数据模板也不难，你可以把其中一个 Pivot 项的 XAML 代码复制过来，并修改 ItemsSource 属性的绑定，如代码 1-28 所示：

```
<DataTemplate x:Key="pivotItemContentTemplate">
    <Grid>
        <ListBox
            ItemTemplate="{StaticResource courseCollectionItemTemplate}"
            ItemsSource="{Binding Courses}"/>
    </Grid>
</DataTemplate>
```

代码 1-28 Pivot 项的内容模板

之前 Expression Blend 为我们生成的 Course 类的属性采用全小写命名方式，而现在已经改为 Pascal 命名方式，所以 courseCollectionItemTemplate 数据模板的相关绑定也要改过来，如代码 1-29 所示：

```

<DataTemplate x:Key="courseCollectionItemTemplate">
    <StackPanel Margin="10,12,0,12">
        <TextBlock Text="{Binding Name}"
            FontSize="{StaticResource PhoneFontSizeExtraLarge}"/>
        <StackPanel Orientation="Horizontal">
            <TextBlock Text="{Binding StartTime}"
                d:LayoutOverrides="Width"
                FontSize="{StaticResource PhoneFontSizeMedium}"/>
            <TextBlock TextWrapping="Wrap"
                Text="-"
                FontSize="{StaticResource PhoneFontSizeMedium}"
                Margin="5,0"/>
            <TextBlock Text="{Binding EndTime}"
                d:LayoutOverrides="Width"
                FontSize="{StaticResource PhoneFontSizeMedium}"/>
        </StackPanel>
        <TextBlock Text="{Binding Location}"
            FontSize="{StaticResource PhoneFontSizeMedium}"/>
    </StackPanel>
</DataTemplate>

```

代码 1-29 修正 Course 类的属性的绑定

数据模板创建好之后就可以应用到 Pivot 控件了，如代码 1-30 所示：

```

<controls:Pivot
    Title="课程表" ItemsSource="{Binding Columns}"
    SelectedIndex="{Binding SelectedColumnIndex, Mode=TwoWay}"
    HeaderTemplate="{StaticResource pivotItemHeaderTemplate}"
    ItemTemplate="{StaticResource pivotItemContentTemplate}" />

```

代码 1-30 在 Pivot 控件上应用模板

由于我们不再使用 Expression Blend 生成的示例数据了，所以你可以把包含 Pivot 控件的 Grid 的 DataContext 属性去掉，然后在代码隐藏文件的构造函数里设置 DataContext 属性，如代码 1-31 所示：

```

public CourseTimetablePage()
{
    InitializeComponent();

    DataContext = new CourseTimetableViewModel();
}

```

代码 1-31 初始化 DataContext 属性

### 1.5.5 测试应用

好了，我知道你很心急了，按 F5 吧，然后单击中间的按钮……噢！出错了！

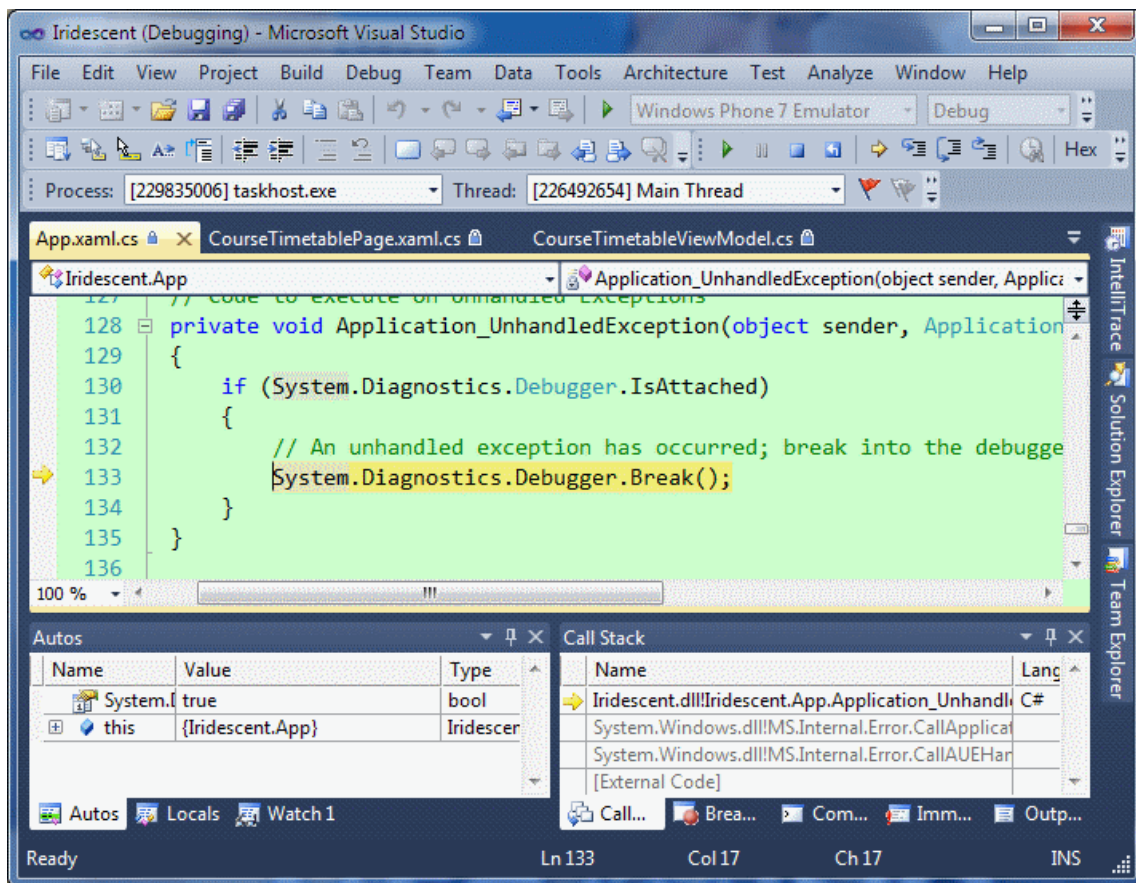


图 1-43 应用异常中止

怎么回事呢？经过一番调查，我发现问题出在代码 25 的设置 SelectedColumnIndex 属性那行，如果你去掉这行，你会发现什么问题也没有，如果你尝试为它硬编码一个值，你会发现同一个值有时候会抛异常，有时候不会，非常不稳定，为什么呢？原来，当我们设置 SelectedColumnIndex 属性的时候，它会把我们给它设的值同步到 Pivot 控件的 SelectedIndex 属性，但此时 Pivot 控件有可能还没完全构建好，于是就出错了。

解决方法是把代码 25 出错的那行删掉，在代码 31 设置 DataContext 属性的那行后面加上这句，如代码 1-32 所示：

```
Loaded +=  
    (o, e) =>  
        ((CourseT timetableViewModel)DataContext).SelectedColumnIndex =  
            (int)DateTime.Today.DayOfWeek;
```

代码 1-32 在 Loaded 事件处理程序里初始化 SelectedColumnIndex 属性

值得提醒的是，通过 `Loaded` 事件来设置 `SelectedColumnIndex` 属性是必须的，因为此时页面及其包含的控件已经构建好了，如果你只是把设置 `SelectedColumnIndex` 属性的代码从代码 25 复制粘贴到代码 31，那么问题依旧存在。

现在，当你打开课程表时，你会看到 `Pivot` 控件快速滑到今天的课程，如图 1-44 所示：

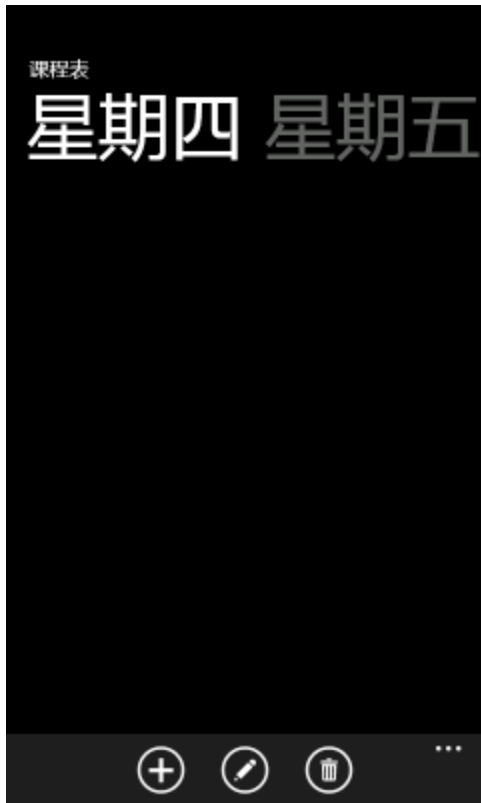


图 1-44 课程表页面

哎哟，课程表空荡荡的，怎么检查课程的显示格式是否正确？添加几个看看吧。但是，添加课程的功能还没有……

## 1.6 操作课程表

### 1.6.1 创建 `NewOrEditCourseViewModel` 抽象类

我们知道，新增和修改这两个操作是共用同一个页面的，但它们的内部逻辑又稍微有点不同，你可以：

- ✓ 分别为它们创建两个不同的 `ViewModel` 类，针对不同的操作为页面创建不同的 `ViewModel` 对象。
- ✓ 也可以在同一个 `ViewModel` 类里通过标记变量和条件语句区分两种不同的逻辑。

- ✓ 你还可以像我这样，在 ViewModels 文件夹里创建一个 NewOrEditCourseViewModel 抽象类，让它实现 INotifyPropertyChanged 接口，并创建 Title 和 Course 两个属性以及 Submit 和 Discard 两个抽象方法，如代码 1-33 所示：

```
public abstract class NewOrEditCourseViewModel
    : INotifyPropertyChanged
{
    private string _title;
    public string Title...

    private Course _course;
    public Course Course...

    public abstract void Submit();

    public abstract void Discard();

    public event PropertyChangedEventHandler PropertyChanged;

    private void OnPropertyChanged(string propertyName)...
}
```

代码 1-33 NewOrEditCourseViewModel 抽象类

接着，在 NewOrEditCourseViewModel 类里创建 NewCourseViewModel 和 EditCourseViewModel 两个私有类，并让它们继承 NewOrEditCourseViewModel 类。

### 1.6.2 创建 NewCourseViewModel 类

课程表上的每个课程都会关联到一周的某天，但 NewOrEditCoursePage 页上却没有地方设置星期几（参见图 1-32），这是为什么呢？Pivot 控件的一个特点是每次只能显示一个 Pivot 项，这意味着整个课程表每次只能显示一天的课程，如果把这天看做上下文，当用户单击 Application Bar 上的新增按钮时，应用程序就可以从上下文获知应该把课程添加到哪一天，从而为用户省下设置星期几这个步骤。

我们可以通过参数传递这个数据，然后在构造函数里使用它创建 Course 对象，如代码 1-34 所示：

```
public NewCourseViewModel(string day)
{
    Title = "新建课程";
    Course = new Course
    {
        Day = day,
    };
}
```

代码 1-34 NewCourseViewModel 类的构造函数

当用户单击确定时，就会把课程添加到 JsonCourseStore，而单击取消的话就什么都不做，如代码 1-35 所示：

```
public override void Submit()
{
    App.CourseStore.Courses.Add(Course);
}

public override void Discard()
{
}
```

代码 1-35 Submit 和 Discard 方法

### 1.6.3 创建 EditCourseViewModel 类

那么，EditCourseViewModel 类呢？当用户单击 Application Bar 上的编辑按钮时，它需要的不是今天星期几，而是用户当前选中的课程是什么，我们又该如何告诉它呢？我们知道，同一天的课程在时间上是互斥的，因为同一时间你不可能在不同教室上课（除非你懂影分身术），换句话说，Course 类的 Day 和 StartTime 这两个属性组合起来可以成为唯一标识。

通过这个唯一标识，我们可以从 JsonCourseStore 里获取用户当前选中的课程，但是，我们是否把获取到的课程直接赋给 EditCourseViewModel 的 Course 属性呢？想想看，Course 属性将会和 NewOrEditCoursePage 页上的控件进行双向绑定，当用户编辑控件的内容时，数据会直接反映在 Course 属性上，如果 Course 属性就是从 JsonCourseStore 里获取到的课程，那么数据就会直接提交到 JsonCourseStore。

如果你选择这样做，你得先做个备份，假如用户单击取消，你就可以把备份的数据还原回去。另一个做法是从 JsonCourseStore 里获取用户当前选中的课程，然后克隆一份赋给 Course 属性，当用户单击确定时，就把 Course 属性的数据更新过去，而单击取消的话也是什么都不做。

这里选择后面那种做法，如代码 1-36 所示：

```

public EditCourseViewModel(string id)
{
    Title = "编辑课程";
    _source = App.CourseStore.Courses.First(
        c => c.Day + c.StartTime.ToShortTimeString() == id);
    Course = new Course
    {
        Name = _source.Name,
        Day = _source.Day,
        StartTime = _source.StartTime,
        EndTime = _source.EndTime,
        Location = _source.Location,
    };
}

private Course _source;

public override void Submit()
{
    _source.Name = Course.Name;
    _source.Day = Course.Day;
    _source.StartTime = Course.StartTime;
    _source.EndTime = Course.EndTime;
    _source.Location = Course.Location;
}

```

代码 1-36 EditCourseViewModel 类的实现

创建好 NewCourseViewModel 和 EditCourseViewModel 两个私有类之后，我们需要考虑一下如何访问它们的实例，办法可能有很多，其中最简单的是在 NewOrEditCourseViewModel 类里创建两个静态方法，分别用于创建这两个类的实例：

```

public static NewOrEditCourseViewModel
    CreateNewCourseViewModel(string day)
{
    return new NewCourseViewModel(day);
}

public static NewOrEditCourseViewModel
    CreateEditCourseViewModel(string id)
{
    return new EditCourseViewModel(id);
}

```

代码 1-37 创建对应 ViewModel 对象的方法



1.6.4 把 ViewModel 类关联到对应的页面

有了实例之后，我们就要考虑数据绑定的问题了，这个不难处理，我们总共也只有五个绑定需要创建，你可以参照下表修改 NewOrEditCoursePage.xaml 的内容：

描述	类型	属性	绑定表达式
页面标题	TextBlock	Text	{Binding Title}
课程名称	TextBox	Text	{Binding Course.Name, Mode=TwoWay}
上课时间	TimePicker	Value	{Binding Course.StartTime, Mode=TwoWay}
下课时间	TimePicker	Value	{Binding Course.EndTime, Mode=TwoWay}
上课地点	TextBox	Text	{Binding Course.Location, Mode=TwoWay}

表 1-4

由于页面之间的切换是通过 `Navigate` 方法来实现的（参见代码 2），而它又只接受 `Uri` 对象作为参数，于是查询字符串就自然而然地成为页面之间传递数据的主要途径了。我们需要的参数有三个：`action`、`day` 和 `id`，其中 `action` 的值有 `new` 和 `edit` 两种，分别对应新建和编辑两种操作，`action` 可以和 `day` 或 `id` 搭配使用，但不会同时使用三个参数。

我们可以根据这些参数的值创建对应的 `ViewModel` 对象，然后把它赋给页面的 `DataContext` 属性，如代码 1-38 所示：

```
var action = NavigationContext.QueryString["action"];
if (action == "new")
{
    DataContext =
        NewOrEditCourseViewModel.CreateNewCourseViewModel(
            NavigationContext.QueryString["day"]);
}
else
{
    DataContext =
        NewOrEditCourseViewModel.CreateEditCourseViewModel(
            NavigationContext.QueryString["id"]);
}
```

代码 1-38 把 `DataContext` 属性初始化为对应的 `ViewModel` 对象

现在的问题是，这段代码应该放在哪里？构造函数？还是别的什么地方？这取决于 `NavigationCacheMode` 属性的值是什么，如果它的值是 `Disabled`，这意味着页面不会缓存，每次都会创建新的实例，那么我们可以把这段代码放在页面的构造函数里，否则，我们应该把它放在 `OnNavigatedTo` 方法里，事实上，任何时候我们来到一个页面，该页面的 `OnNavigatedTo` 方法都会被调用，所以放在这里比较保险。

你可能会问，为什么 `CourseTimetablePage` 页的 `DataContext` 属性是在构造函数里设置的？好问题！那是因为 `CourseTimetablePage` 页的 `ViewModel` 对象由始至终都未曾改变，

而 NewOrEditCoursePage 页的 ViewModel 对象及与之绑定的 Course 对象每次都不同，页面的缓存可能会导致用户看到“过期”的信息。

当用户单击确定时，将会调用 ViewModel 对象的 Submit 方法，然后调用 NavigationService 对象的 GoBack 方法返回课程表；而单击取消的话就直接返回，如代码 1-39 所示：

```
private void OKButton_Click(object sender, RoutedEventArgs e)
{
    ((NewOrEditCourseViewModel)DataContext).Submit();
    NavigationService.GoBack();
}

private void CancelButton_Click(object sender, RoutedEventArgs e)
{
    NavigationService.GoBack();
}
```

代码 1-39 确定和取消按钮的实现

#### 1.6.5 为课程表页面上的 Application Bar 按钮添加 Click 事件处理程序

现在，让我们回到 CourseTimetablePage.cs，把新建按钮的 Click 事件处理程序改成下面这样：

```
private void ApplicationBarNewIconButton_Click(
    object sender, EventArgs e)
{
    var timetable = (CourseTimetableViewModel)DataContext;
    var day =
        timetable.Columns[timetable.SelectedColumnIndex].Header;
    NavigationService.Navigate(
        new Uri("/NewOrEditCoursePage.xaml?action=new&day=" + day,
            UriKind.RelativeOrAbsolute));
}
```

代码 1-40 新建按钮的事件处理程序

需要说明的是，Pivot 项的标题是“星期 X”，对应于 Course 对象的 Day 属性，因此我们可以把它作为参数通过查询字符串传给 NewOrEditCoursePage 页。

接着，为编辑按钮创建一个 Click 事件处理程序：

```
private void ApplicationBarEditIconButton_Click(
    object sender, EventArgs e)
{
    var timetable = (CourseTimetableViewModel)DataContext;
    var selectedColumn =
        timetable.Columns[timetable.SelectedColumnIndex];
    var selectedCourse =
        (Iridescent.Models.Course)selectedColumn.Courses.CurrentItem;
    var id =
        selectedCourse.Day +
        selectedCourse.StartTime.ToShortTimeString();
    NavigationService.Navigate(
        new Uri("/NewOrEditCoursePage.xaml?action=edit&id=" + id,
            UriKind.RelativeOrAbsolute));
}
```

代码 1-41 编辑按钮的事件处理程序

最后是删除按钮的 Click 事件处理程序：

```
private void ApplicationBarDeleteIconButton_Click(
    object sender, EventArgs e)
{
    var timetable = (CourseTimetableViewModel)DataContext;
    var selectedColumn =
        timetable.Columns[timetable.SelectedColumnIndex];
    var selectedCourse =
        (Iridescent.Models.Course)selectedColumn.Courses.CurrentItem;
    App.CourseStore.Courses.Remove(selectedCourse);
}
```

代码 1-42 删除按钮的事件处理程序

你可能会问，为什么执行删除之前不给用户提示一下？嗯，这是个值得考虑的问题，有时候你不提示用户会质问你万一删错了怎么办，有时候你提示了用户也会抱怨这样做很烦，一般而言，仅仅在执行操作之前给用户提示一下是远远不够的，你还需要让用户可以设置以后不再提示，以及告诉用户随时可以在哪里开启提示。不过，你可以尝试实现一个简单的提示，至于可以设置的提示将会在以后的文章里探讨。

### 1.6.6 为保存和撤销添加 Application Bar 菜单项

现在还缺什么吗？噢，对了，还缺两个菜单项，打开 CourseTimetablePage.xaml，切换到 XAML 模式，在<shell:ApplicationBar></shell:ApplicationBar>里添加两个 Application Bar 菜单项，如代码 1-43 所示：

```

<shell:ApplicationBar.MenuItems>
    <shell:ApplicationBarMenuItem
        Text="保存所有更改"
        Click="ApplicationBarCommitMenuItem_Click"/>
    <shell:ApplicationBarMenuItem
        Text="撤销所有更改"
        Click="ApplicationBarRollbackMenuItem_Click"/>
</shell:ApplicationBar.MenuItems>

```

代码 1-43 保存和撤销的 Application Bar 菜单项

它们的 Click 事件处理程序比较简单，只是分别调用 JsonCourseStore 的 Commit 方法和 Rollback 方法，如代码 1-44 所示：

```

private void ApplicationBarCommitMenuItem_Click(
    object sender, EventArgs e)
{
    App.CourseStore.Commit();
}

private void ApplicationBarRollbackMenuItem_Click(
    object sender, EventArgs e)
{
    App.CourseStore.Rollback();
}

```

代码 1-44 保存和撤销菜单项的事件处理程序

### 1.6.7 测试应用

我已经等不及要按 F5 了，单击主页中间的按钮打开课程表，单击新建按钮，输入课程名称，如图 1-45 所示：

课程表

# 新建课程

课程名称

上课时间

下课时间

q	w	e	r	t	y	u	i	o	p
a	s	d	f	g	h	j	k	l	
↑	z	x	c	v	b	n	m	<x>	
&123	,	space	En	.	←				

图 1-45 输入课程名称

修改上课时间：

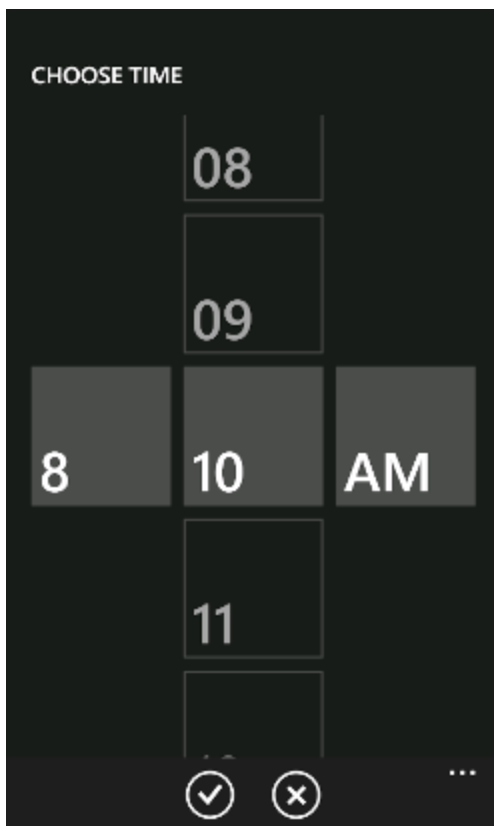


图 1-46 修改上课时间

单击确定返回：

课程表

# 新建课程

课程名称

上课时间

12:00 AM

下课时间

12:00 AM

上课地点

确定 取消

图 1-47 新建课程页面的控件被重置了

奇怪了！我刚才明明输入了课程名称，为什么现在没了呢？而且上课时间也没改过来！究竟发生了什么事？

我查看了 Silverlight for Windows Phone Toolkit 的代码，在 [DateTimePickerPageBase 类](#) 的 `HandleClosedStoryboardCompleted` 方法里可以看到，当我们设好时间并单击确定时，它会调用 `NavigationService` 对象的 `GoBack` 方法返回。前面我们提到，任何时候当我们来到一个页面时，该页面的 `OnNavigatedTo` 方法就会被调用，现在，我们在 `NewOrEditCoursePage.cs` 的 `OnNavigatedTo` 方法里设置一个断点，如图 1-48 所示，然后从设置时间的页面返回，看看会发生什么事：





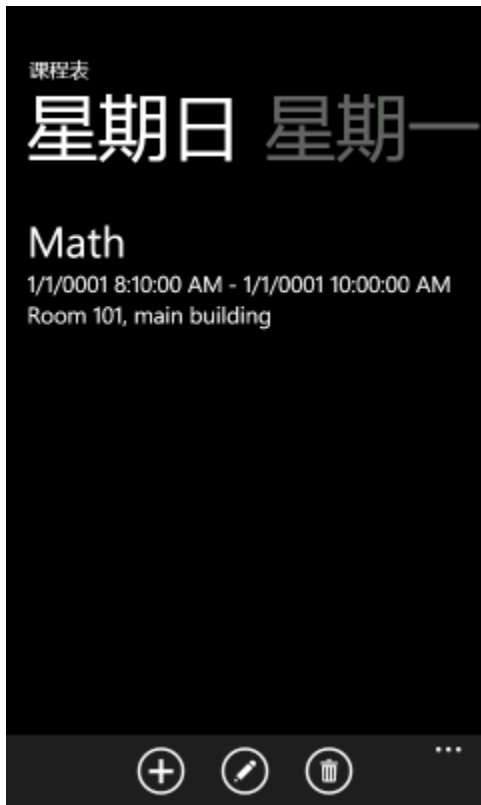


图 1-49 课程表页面

慢着！时间的显示格式有问题，我期望它显示为 8:10 – 10:00 而不是现在这样，怎么办？这个时候就轮到转换器出场了。

### 1.6.8 创建时间转换器

首先，创建一个 Utils 文件夹，在里面添加一个 TimeConverter 类，并让它实现 IValueConverter 接口，实现这个接口只需实现两个方法，一个是 Convert 方法，用于把 Course 对象的 StartTime 属性和 EndTime 属性的值转换为显示在 UI 上的字符串，另一个是 ConvertBack 方法，这个方法只在双向绑定时才会派上用场，而这里是单向绑定，所以我们不必为它提供实现，如代码 1-45 所示：

```

public class TimeConverter : IValueConverter
{
    object IValueConverter.Convert(
        object value, Type targetType,
        object parameter, CultureInfo culture)
    {
        return ((DateTime)value).ToShortTimeString();
    }

    object IValueConverter.ConvertBack(
        object value, Type targetType,
        object parameter, CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}

```

代码 1-45 TimeConverter 类

接着在 CourseTimetablePage.xaml 的资源字典里添加一个 TimeConverter 对象，如代码 1-46 所示：

```

<util:TimeConverter
    xmlns:util="clr-namespace:Iridescent.Utils"
    x:Key="timeConverter"/>

```

代码 1-46 在 XAML 里创建 TimeConverter 对象

然后把那两个 TextBlock 的 Text 属性的绑定表达式分别改为 “{Binding StartTime, Converter={StaticResource timeConverter}}” 和 “{Binding EndTime, Converter={StaticResource timeConverter}}”。

### 1.6.9 测试应用

现在按 F5 重新运行应用程序，并新建一个课程，这次时间的显示格式就没问题了：



图 1-50 课程表正确显示时间格式

选中这个课程，并单击编辑按钮：

课程表

编辑课程

课程名称

Math

上课时间

8:10 AM

下课时间

10:00 AM

上课地点

Room 101, main building

确定

取消

图 1-51 编辑课程

嗯，很好，页面标题和课程信息都正确显示了，如图 1-52 所示，修改一下并按确定返回：

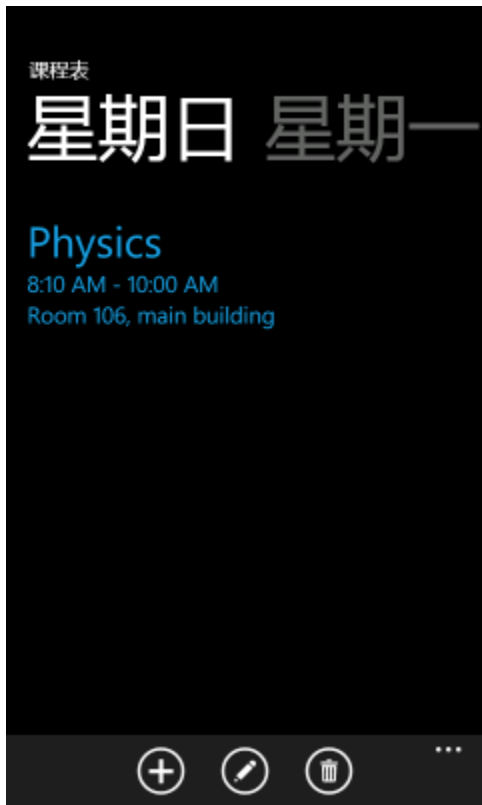


图 1-52 修改后的课程表

课程信息的更改也正确反映到课程表了。

现在，确保课程处于选中状态，单击删除按钮，噢，出错了：

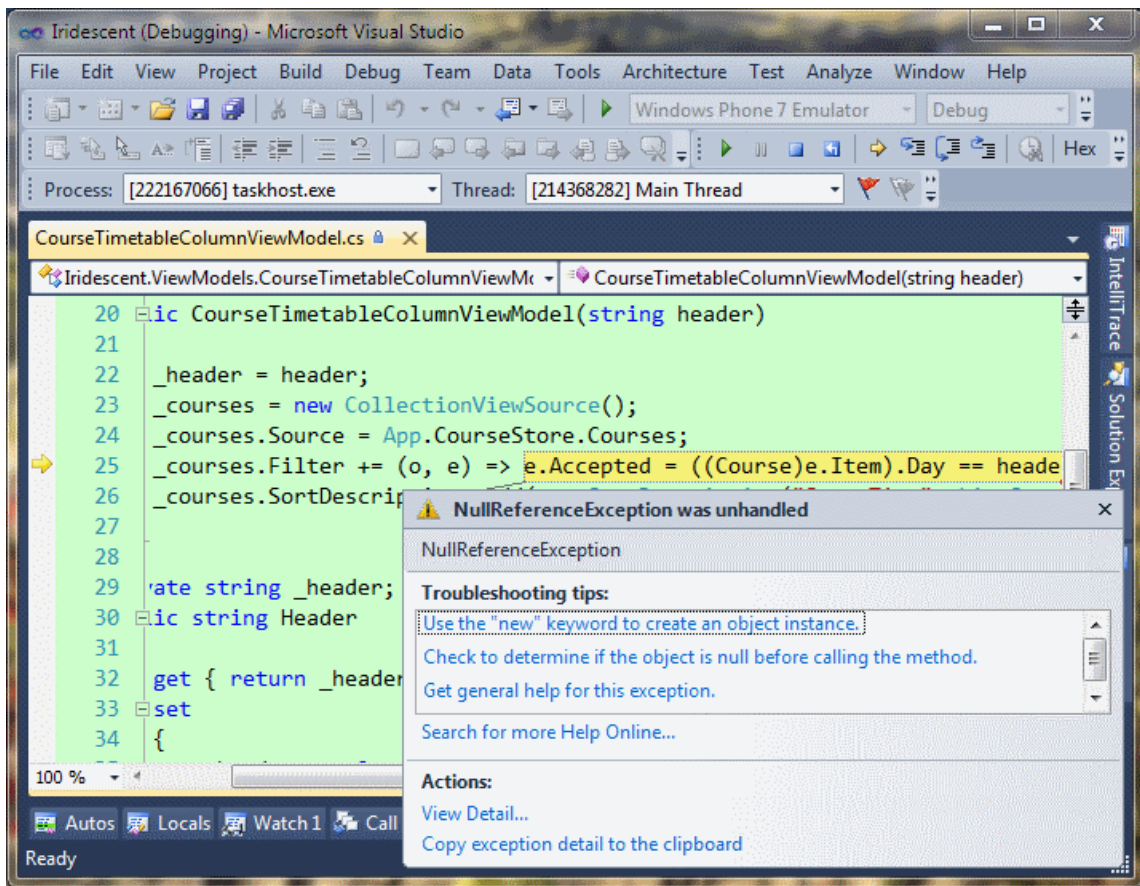


图 1-53 应用异常中止

这个问题好解决，我们只需在使用 `e.Item` 之前判断一下它是否为 `null` 就行了：

```
_courses.Filter += (o, e) => e.Accepted =  
    e.Item != null && ((Course)e.Item).Day == header;
```

代码 1-47 判断对象是否为 `null`

现在按 **F5** 重新运行应用程序，并新建一个课程，先别删除这个课程，我们需要用它来执行以下测试：

1. 单击 **Application Bar** 右上角的省略号。此时，**Application Bar** 的菜单项会显示出来。
2. 单击“保存所有更改”菜单项，并按两次 **Back** 键退出应用程序。
3. 从应用程序列表启动应用程序，并进入课程表。此时，刚才新建的课程应该显示在课程表里。
4. 选中这个课程，单击编辑按钮，修改课程信息，并按确定返回。此时，课程信息的更改应该反映在课程表上。
5. 单击 **Application Bar** 右上角的省略号，并单击“撤销所有更改”菜单项。此时，课程表上的课程信息应该还原为更改之前。但是，没有还原！！

6. 确保这个课程处于选中状态，单击编辑按钮。此时，我们可以看到课程信息确实已经还原了！！
7. 修改课程信息，并按确定返回。此时，我们可以看到新的更改没有反映在课程表上！！
8. 确保这个课程处于选中状态，单击编辑按钮。此时，我们可以看到课程信息确实更改了！！
9. 单击新建按钮，输入课程信息，并按确定返回。此时，我们在课程表上并没看到刚刚新建的课程！！

为什么会这样？从上面的测试不难看出，所有怪事都是单击“撤销所有更改”菜单项之后发生的，而后面五条测试的结果显然在告诉我们课程表页面和 `JsonCourseStore` 已经脱节了。沿着这条线索，我们打开 `JsonCourseStore.cs` 文件，仔细阅读里面的代码，`Rollback` 方法是直接调用 `LoadCoursesFromIsolatedStorage` 方法的，当我们单击“撤销所有更改”菜单项时，它会从独立存储区读取 `Courses.json` 文件，并把里面的数据反序列化到 `Courses` 属性。慢着！把数据反序列化到 `Courses` 属性？这会改变整个集合的引用！换句话说，每次调用 `Rollback` 方法时都会重新创建一个集合，而课程表却一直和“过期”的集合关联着，难怪后面几条测试给人的感觉是课程表页面和 `JsonCourseStore` 脱节了。

知道症结所在，剩下的事情就好办了，你可以分开实现 `JsonCourseStore` 的构造函数和 `Rollback` 方法，也可以像我这样，修改 `LoadCoursesFromIsolatedStorage` 方法，如代码 1-48 所示：

```

private void LoadCoursesFromIsolatedStorage()
{
    using (var fileStore =
        IsolatedStorageFile.GetUserStoreForApplication())
    {
        if (fileStore.FileExists("Courses.json"))
        {
            using (var fileStream =
                fileStore.OpenFile("Courses.json", FileMode.Open))
            {
                Courses.Clear();

                var jsonSerializer =
                    new DataContractJsonSerializer(
                        typeof(ObservableCollection<Course>));
                var courses = (ObservableCollection<Course>)
                    jsonSerializer.ReadObject(fileStream);
                courses.ForEach(Courses.Add);
            }
        }
    }
}

```

代码 1-48 修改 LoadCoursesFromIsolatedStorage 方法

新版的 LoadCoursesFromIsolatedStorage 方法不再把数据直接反序列化到 Courses 属性，而是把 Courses 属性清空，再把数据逐条添加进去。至于 Courses 属性的初始化则挪到构造函数了。需要说明的是，ObservableCollection 类没有 ForEach 方法，这是我自己创建的扩展方法，你也可以直接使用 foreach 语句实现。

现在按 F5 重新运行应用程序，并重新执行一次上面的测试，嗯，这次没问题了，最后，选中这个课程，单击删除按钮，好了，课程已经删除了。在整个操作的过程中，还有两个地方我认为需要改善一下的。第一个是当用户单击“保存所有更改”菜单项时，应用程序应该在完成操作之后提示一下，否则用户可能会有点茫然。另一个是从 NewOrEditCoursePage 页返回 CourseTimetablePage 页时总是看到今天的课程，当我们新建或编辑的课程不是今天的课程时可能引起不必要的疑惑。试想一下，我在新建星期二的课程，当我输入完课程信息并按确定返回时，我看到星期一的课程，因为今天是星期一，这时我的感觉会是怪怪的，比较符合直觉的做法应该是返回时显示星期二的课程，而不是盲目地显示今天的课程。这两个改进的实现就当课后作业留给你吧，对于第二个问题，我可以给个提示，研究一下 OnNavigatedTo 方法和 OnNavigatingFrom 方法，应该难不了你吧？



## 1.7 菜单·样品菜色

### 1.7.1 添加菜单

还差什么呢？噢，对了，目前我们是通过主页中间的按钮打开课程表的，这显然不好意思拿出来见人，我们还是创建一个正式的主菜单吧。

现在，让我们切换到 Expression Blend，如果你的 Expression Blend 已经关闭了，你可以右击 Solution Explorer 的 MainPage.xaml，然后选择 Open in Expression Blend，如图 1-54 所示：

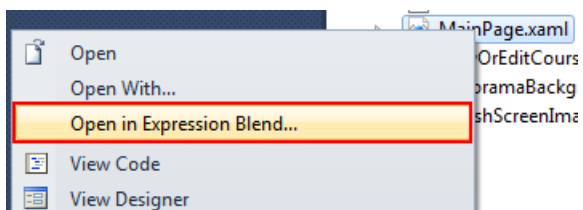


图 1-54 在 Visual Studio 里打开 Expression Blend

菜单的制作方式有很多种，这里选用 ListBox，如图 1-55 所示：



图 1-55 主页上的菜单

菜单创建好后，右击里面的“课程表”，然后选择 Navigate To\CourseTimetablePage。

此时，Expression Blend 会为 TextBlock 添加一个 NavigateToPageAction 行为，但这个行为默认是关联到 MouseLeftButtonDown 事件的，这样，当用户按下“课程表”还没松手就打开课程表了，而我们的习惯是松手之后才执行相关的操作，为了实现这个效果，我们可以把关联事件改为 MouseLeftButtonUp，如图 1-56 所示：

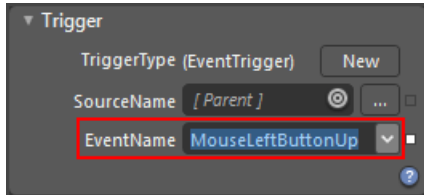


图 1-56 把触发时间设为 MouseLeftButtonUp

### 1.7.2 重新添加示例数据

菜单有了，但样品菜色却被我们弄没了，如果你现在打开 CourseTimetablePage.xaml，你将会看到此番情景：

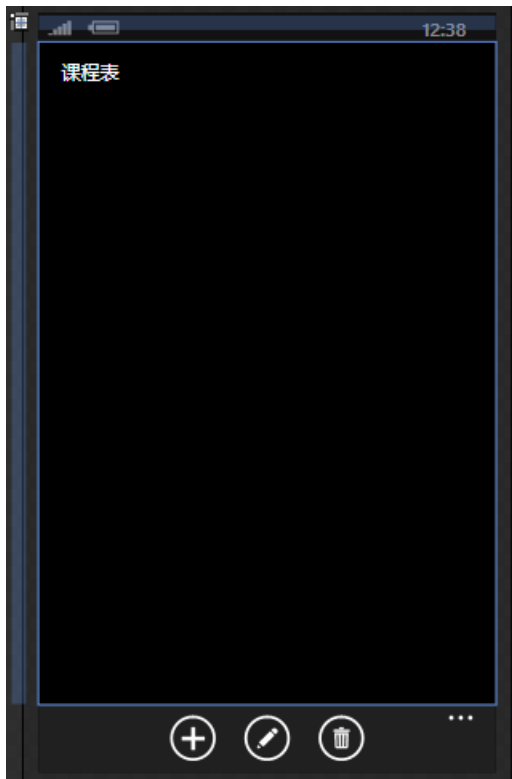


图 1-57 没有示例数据的课程表页面

之前我们手动创建两个 Pivot 项，然后把它们分别绑到两个示例数据源，但现在 Pivot 项是通过数据绑定动态创建的，之前那些示例数据源就排不上用场了，既然用不了，那就删了吧。

打开 Data 面板，右击数据源，然后选择 Delete data source，如图 1-58 所示：

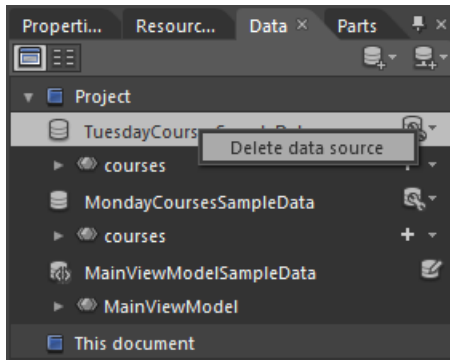


图 1-58 删除不用的示例数据

此时，Expression Blend 会把 SampleData 文件夹里的相关文件一并删除。

在 Expression Blend 里，没有设计时数据会为调整控件模板和样式带来很大不便，下次你去找设计师调整一下用户界面，他们很可能会对你大骂一顿，那么，有没有办法让它再次显示设计时数据呢？答案是有的，而且不止一种，下面来看其中一种。我们知道，CourseTimetablePage 页使用了数据绑定，而绑定表达式只提及了绑定的属性，并未提及属性所在的类型，换句话说，只要属性名字能够匹配，示例数据就应该绑得上去。

首先，准备一份 XML，注意匹配绑定表达式的属性名字，如代码 1-49 所示：

```
<?xml version="1.0" encoding="utf-8"?>
<Timetable SelectedColumnIndex="0">
  <Column Header="星期一">
    <Course Name="公司法" Day="星期一" StartTime="8:10"
      EndTime="10:00" Location="综合楼802" />
    <Course Name="刑事诉讼法" Day="星期一" StartTime="10:10"
      EndTime="11:50" Location="综合楼802" />
    <Course Name="合同法" Day="星期一" StartTime="14:30"
      EndTime="16:00" Location="教学楼101" />
  </Column>
  <Column Header="星期二" />
</Timetable>
```

代码 1-49 包含示例数据的 XML 文件

接着，在 Data 面板上把它导入（参见图 1-15），注意，这次要把 Enable sample data when application is running 选项去掉，如图 1-59 所示：

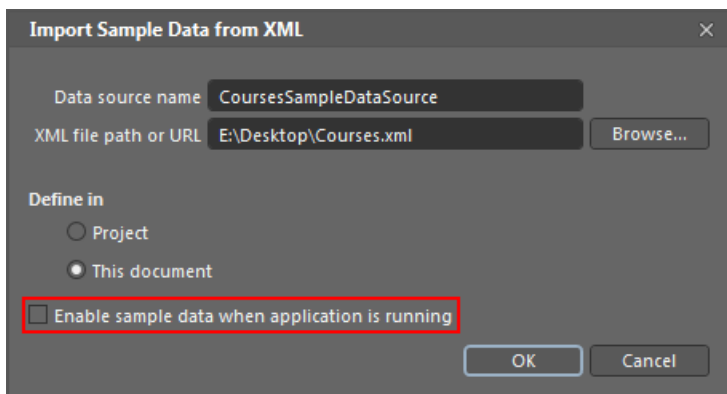


图 1-59 设置导入的示例数据

导入之后在 Data 面板上把自动生成的 ColumnCollection 和 CourseCollection 分别改为 Columns 和 Courses，以便匹配绑定表达式的属性名字，如图 1-60 所示：

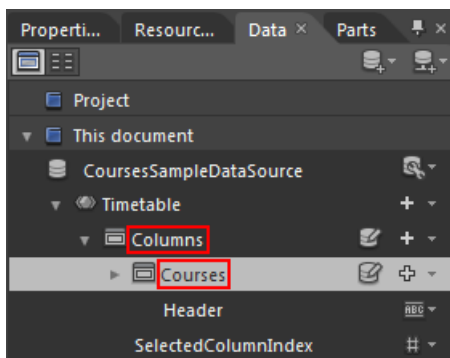


图 1-60 修改示例数据的属性名字

现在，把 Timetable 拖到 Pivot 控件上，此时你会看到鼠标下方有个小提示，告诉你 Expression Blend 将会把 Pivot 控件的 DataContext 属性绑到 CoursesSampleDataSource，如图 1-61 所示：

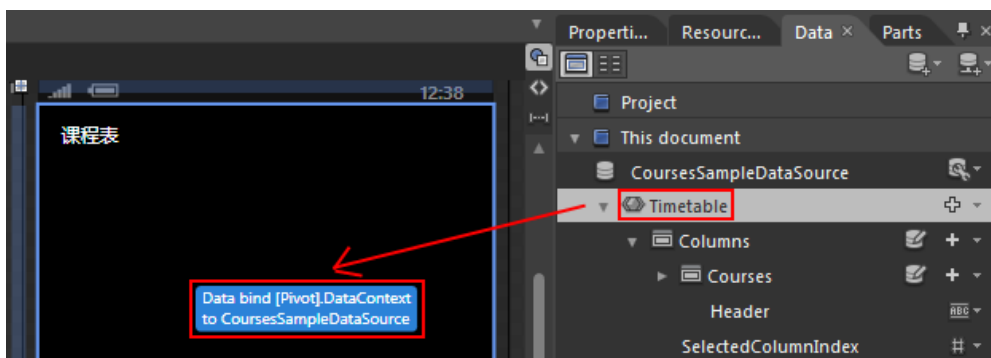


图 1-61 通过 Data 面板创建课程列表

当你松开鼠标时，就会看到课程名称了，但上课时间、下课时间和上课地点却显示不出来，如图 1-62 所示：

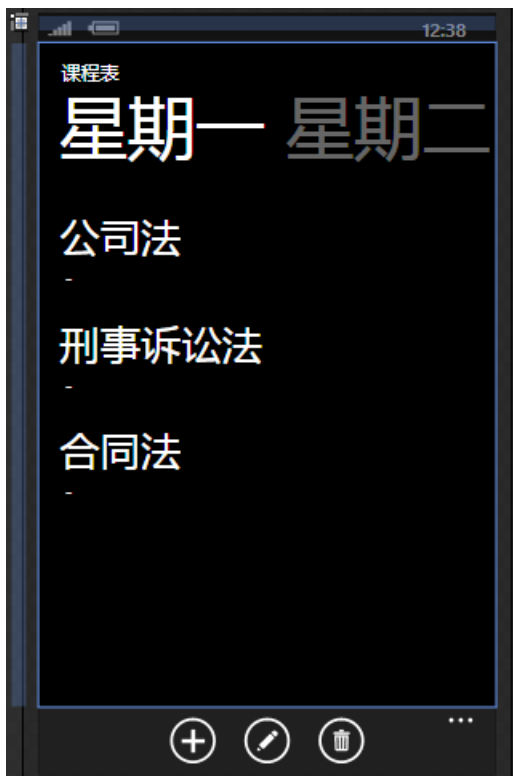


图 1-62 只显示课程名称的课程列表

这是为什么呢？如果你把 `CourseTimetablePage` 页面关闭，然后重新打开它，你就会看到问题所在了：

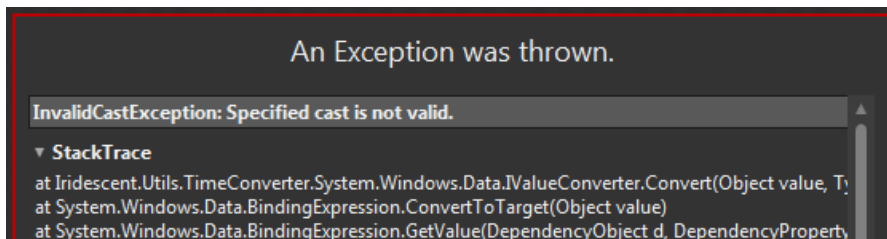


图 1-63 异常信息

这个异常明确地告诉我们调用转换器时抛出 `InvalidCastException`。还记得吗，我们的转换器会把传入对象强制转换成 `DateTime` 对象，然后调用它的 `ToShortTimeString` 方法，但 `Expression Blend` 为示例数据生成的 `StartTime` 属性和 `EndTime` 属性是字符串类型的！

明白为什么会这样，问题就不难解决了，把转换器的 `Convert` 方法改成代码 1-50 所示的那样：

```

object IValueConverter.Convert(
    object value, Type targetType,
    object parameter, CultureInfo culture)
{
    if (value.GetType() == typeof(DateTime))
    {
        return ((DateTime)value).ToShortTimeString();
    }
    else
    {
        return value.ToString();
    }
}

```

代码 1-50 修改 Convert 方法

重新编译项目，然后重新打开 CourseTimetablePage 页，你就会看到设计时数据了：

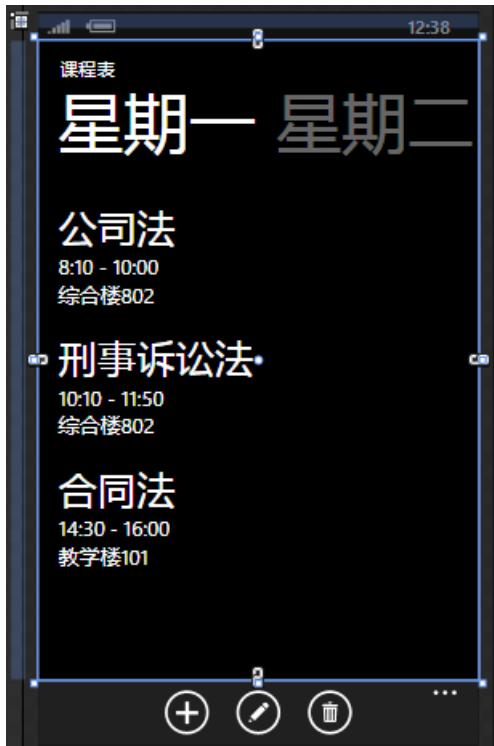


图 1-64 正常显示示例数据的课程表

好了，总算对设计师有个交代了。

## 1.8 下课了……



第2课

# 创建作业本

- 作业本
- 保存作业本
- 原型
- 定制数据模板
- 插曲 #1
- 连接前端和后端
- 编辑作业本
- 插曲 #2
- 编辑作业本•续





## 2.1 作业本

### 2.1.1 调查需求

[上节课](#)布置的作业有做吗？没人吭声啊，看来大家都忘了哦，没事，我们这次弄个作业本出来，大家就有地方记作业了。在开始设计应用程序之前，我们先来看看通常的作业本是怎样记作业的：

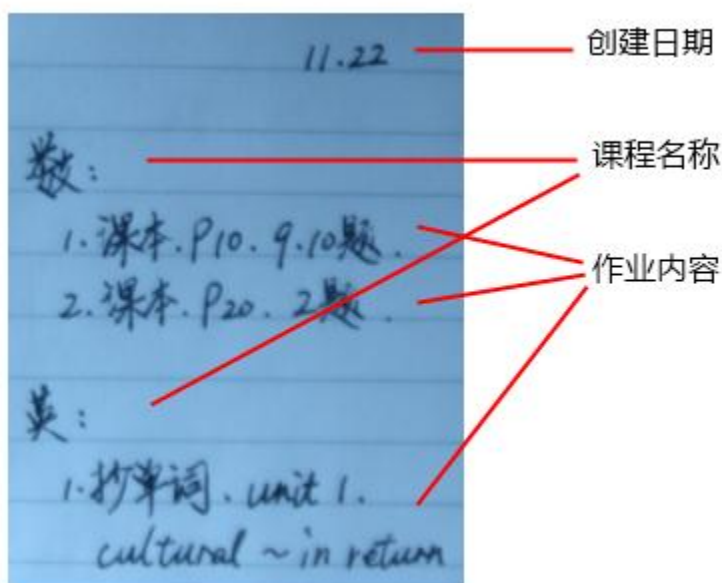


图 2-1 传统笔记本

从上图可以看到，作业本有点像日记本，每次记录时都会写下当天的日期，每天的作业又会根据课程进行归类。

慢着！我怎么知道这些作业什么时候交？一般情况下，中小学生的作业都是第二天上课时交的，但大学生就不同了，他们的作业可能第二天交，也可能一周之后交，有时甚至几周之后才交，更重要的是，不同的作业可能在不同的时间交。换句话说，我们的应用程序还需要支持记录交作业的时间。此外，每当完成一项作业，我们可以在旁边做个记号，这样，当我们打开作业本时，即使作业再多也能马上知道哪些还没做完。

### 2.1.2 创建 Assignment 类

现在，用 Visual Studio 打开项目，在 Models 文件夹里创建一个 Assignment 类，和上节课的 Course 类一样，它也需要实现 INotifyPropertyChanged 接口。由于我们有很多类都需要实现 INotifyPropertyChanged 接口，为了避免不必要的重复，你可以考虑创建一个类专门实现这个接口，然后让有需要的类继承这个类。这个需求似乎比较常见，因此 [Prism](#) 提供了一个 NotificationObject 类，我们只需继承它就行了，如代码 2-1 所示。

```
public class Assignment : NotificationObject
{
}

```

代码 2-1 Assignment 类

继承之前别忘了引用 Bin\Phone\Microsoft.Practices.Prism.dll 类库和 Microsoft.Practices.Prism.ViewModel 命名空间哦。根据前面的讨论，Assignment 类应该包含以下属性：

属性名字	属性类型	备注
Id	Guid	唯一标识
CourseName	string	课程名称
StartDate	DateTime	创建日期
DueDate	DateTime	截止日期
Content	string	作业内容
IsCompleted	bool	完成状态

表 2-1 Assignment 类的属性

我们知道，Id 属性作为唯一标识，其值一旦生成就不会改变，因此我们只需在构造函数里初始化它就行了，如代码 2-2 所示：

```
public Assignment()
{
    Id = Guid.NewGuid();
}

public Guid Id { get; private set; }

```

代码 2-2 初始化 Id 属性

而其它属性则需要在它们的 set 访问器里调用从 NotificationObject 类继承过来的 RaisePropertyChanged 方法，比如说，我们可以这样实现 IsCompleted 属性：

```

private bool _isCompleted;
public bool IsCompleted
{
    get { return _isCompleted; }
    set
    {
        _isCompleted = value;
        RaisePropertyChanged("IsCompleted");
    }
}

```

代码 2-3 IsCompleted 属性

看到这里，你可能会说，作业的状态应该不止“已完成”和“未完成”两种啊，比如说，当老师刚把作业布置下来时，它应该是“未开始”；当我们开始做某项作业时，它应该是“进行中”；有时候准备工作还没好，我们不得不把作业推迟，此时它应该是“已推迟”；有时候老师可能大发慈悲说某些作业不用做了，此时它应该是“已取消”，等等等等。照这样说，我们是否也该考虑把现在的两个日期细化为“计划开始日期”、“计划结束日期”、“实际开始日期”和“实际结束日期”，然后加上一个“作业进度”什么的？千万不要这样，没有学生愿意采用这么细致的作业管理方案，再说这样做也会分散他们的注意、加重他们的负担，作业本的主要目的只有一个，就是让学生对要做哪些作业一目了然，所有功能的设计都应该围绕这点展开，所有功能的取舍也应该以此为标准。

## 2.2 保存作业本

数据存储方面，我打算仿效课程表的做法，通过 JSON 序列化把作业本的数据保存到独立存储区，实现这个并不难，你可以照搬课程表的做法，创建一个 IAssignmentStore 接口和一个 JsonAssignmentStore 类。当你实现完 JsonAssignmentStore 类之后，你将会发现它和 JsonCourseStore 类有 99.9%的代码是相同的，事实上，你可以把 JsonCourseStore.cs 文件复制一份，并重命名为 JsonAssignmentStore.cs，然后把里面的“Course”字眼都替换成“Assignment”就可以了。不过，这种重复着实让人不爽啊，看来是时候重构一下了。

ICourseStore 接口和 IAssignmentStore 接口的区别只在于集合元素的类型和集合属性的名字，前者可以通过泛型统一起来，至于后者，我们可以把属性的名字统一为 Items，这样，两个接口就能统一起来了，如代码 2-4 所示：

```
public interface IDataStore<T>
{
    ObservableCollection<T> Items { get; }

    void Rollback();

    void Commit();
}
```

代码 2-4 IDataStore 接口

而实现方面，我们可以创建一个 JsonDataStore<T>类，并让它实现 IDataStore<T>接口，如代码 2-5 所示：

```
public class JsonDataStore<T> : IDataStore<T>
{
    public JsonDataStore(string fileName)
    {
        _fileName = fileName;
        Items = new ObservableCollection<T>();
        LoadItemsFromIsolatedStorage();
    }

    private string _fileName;

    // ...
}
```

代码 2-5 JsonDataStore 类

需要说明的是，之前我们把文件名硬编码在 JsonXXXStore 类里，那是因为它对于 JsonXXXStore 类来说是固定的、一对一的，而现在的 JsonDataStore<T>类不再仅仅对应一个文件，因此我们把它保存在一个私有字段里。其它的和 JsonAssignmentStore 类没有太大出入。

看到这里，有些同学可能会问，ICourseStore 接口和 JsonCourseStore 类已经投入使用了，现在换用 IDataStore<T>接口和 JsonDataStore<T>类会不会造成很大影响？这个问题问得好，如果你确实不想修改其它代码，那你可以把 JsonCourseStore 类改造成 JsonDataStore<Course>类的“马甲”，如代码 2-6 所示：

```

public class JsonCourseStore : JsonDataStore<Course>, ICourseStore
{
    public JsonCourseStore()
        : base("Courses.json")
    { }

    public ObservableCollection<Course> Courses
    {
        get { return Items; }
    }
}

```

代码 2-6 JsonDataStore<Course>类的马甲

需要说明的是，我们通过继承 JsonDataStore<Course>类获得 Rollback 和 Commit 两个方法的实现，此外，由于其它代码是通过 ICourseStore 接口间接使用 JsonCourseStore 类的实例的，于是我们保留了 ICourseStore 接口，并把 Courses 属性重定向到 Items 属性。

不过，就项目现在的规模而言，我们可以把重构做的更彻底一些，我们可以把 ICourseStore.cs 和 JsonCourseStore.cs 两个文件删除，如果你想保险一点，可以先把它们从项目排除出去，然后重新编译，此时 Visual Studio 会告诉你找不到 ICourseStore 接口和 JsonCourseStore 类，分别把它们替换成 IDataStore<Course>接口和 JsonDataStore<Course>类，调用后者的构造函数时记得提供文件名，即 Courses.json，重新编译，此时 Visual Studio 会显示一堆错误，全部都是说找不到 Courses 属性的，把它们都替换成 Items 属性，重新编译，好了，如果那两个文件还没删除的话，现在可以安全删除了。

## 2.3 原型

### 2.3.1 设计原型

现在是时候考虑一下用户界面了，仔细观察我们的作业本（图 2-1），是否觉得这种布局方式有种似曾相识的感觉？如果你一直关注 WP7 的相关消息，你可能已经看过类似的用户界面了——People Hub 的联系人列表。下面我们把它们两个放在一起看看：

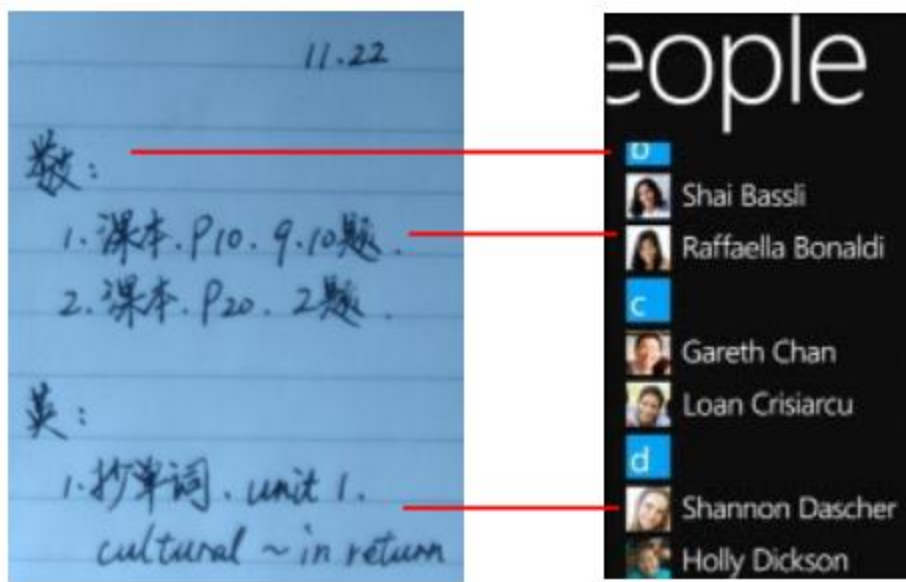


图 2-2 对比传统作业本和 People Hub 的联系人列表

从上图可以看到，作业列表和联系人列表刚好能够对应起来，课程名称对应姓氏首字母，作为分组标题，而作业内容则对应联系人，作为分组内容。

看到这里，你可能会问，WP7 的 Silverlight 貌似没有这样的控件啊，难道要我们自己动手弄一个？原本是没有的，不过十一月发布的 [SL for WP Toolkit](#) 已经增加了这个控件，名字叫做 LongListSelector。上节课我们使用了 Silverlight for Windows Phone Toolkit 的 TimePicker 控件，当时引用的是九月份发布的版本，现在你可以下载新的版本，然后重新引用一下。

仔细观察图 2-2，你会发现作业列表上面有个日期没法对应到联系人列表，我们该怎么处理这个日期呢？这个问题问得好，事实上，这正是作业列表和联系人列表的最大区别，我们知道，联系人列表只有一份，但作业列表却会有很多份，每份都会有一个不同的日期，这些作业列表共同组成了一本作业本。如果把每份作业列表看作一个由标题和 LongListSelector 控件组成的页面，那么整个作业本就可以看作由 N 个这样的页面组成的应用程序了，但我们不必真的创建 N 个这样的页面，我们可以仿效课程表的做法，利用 Pivot 控件的特点，让每个 Pivot 项显示一份作业列表，这样 Pivot 项的标题可以用来显示作业列表上面的日期，而标题下面则通过 LongListSelector 控件显示每个课程的作业。不过，这样的设计是否真的妥当呢？

试想一下，如果我们首先通过日期来划分 Pivot 项，接着通过课程来划分作业，那么每次我们要新建作业的时候，我们可能得先创建一个 Pivot 项，如果对应今天的 Pivot 项还没有的话，接着指定作业所属的课程，最后才填写和作业相关的信息，这个过程显然有点繁琐，我们应该尽可能简化其中的步骤。说到这里，有些同学可能会建议，不如让应用程

序自动创建今天的 **Pivot** 项，这样至少可以省掉一个步骤。嗯，这个主意值得考虑，不过，并非每天都会有作业，比如说，今天是星期天，我进入作业本只是看一下这个周末有哪些作业，但应用程序却自动为我创建了今天的 **Pivot** 项，而这并非我想要的，这意味着应用程序不得不在退出的时候把这个空的 **Pivot** 项删除。事实上，对于大学生来说，尤其是大三、大四的，今天有课明天没有是很常见的，难道要让用户设置哪天有课哪天没课，或者干脆直接解释课程表的数据，看看哪天有课哪天没课？

从上面讨论不难看出，日期这个因素很不稳定，不太适合用来划分 **Pivot** 项，但课程就不同了，一旦课程表创建好了，作业本上会有哪些课程的作业也就定下来了，既然如此，何不把分组的顺序换一下？如果我们通过课程来划分 **Pivot** 项，那就不用考虑 **Pivot** 项的创建和删除了，因为用户在访问作业本的过程中会涉及到哪些课程是确定的，此外，当用户新建作业时也无需额外的步骤来指定今天的日期，因为这可以从 **DateTime** 的 **Today** 属性获取，这样我们就为用户省下两个步骤了。

从这里我们可以看到，应用程序的设计绝对不是把控件堆砌起来显示数据就完事了，它包含的是一组完整的用户体验，而不同的组织方式可能会产生完全不一样的用户体验，有时候多一两个步骤好像没什么大不了，但假如这一两个步骤要重复十次的话，用户就要额外执行十几二十个这样的步骤了，要么你为用户省下这些步骤，要么你让竞争对手为用户服务。

### 2.3.2 创建作业本页面

现在让我们切换回 Expression Blend，创建一个 Windows Phone **Pivot Page**，并把它命名为 **AssignmentBookPage.xaml**。完成之后把 **Pivot** 控件的 **Title** 属性设为“作业本”，把两个 **Pivot** 项的 **Header** 属性分别设为“数学”和“英语”，最后，把一个 **LongListSelector** 控件拖到第一个 **Pivot** 项里，如图 2-3 所示。

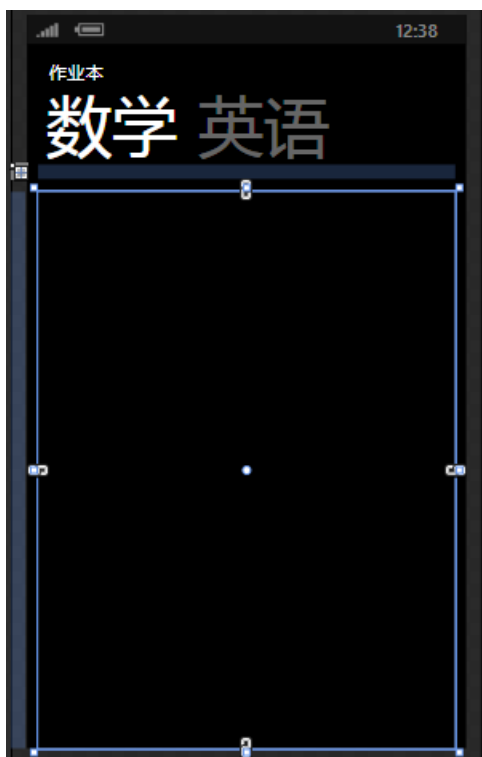


图 2-3 作业本页面

接下来我们要为 LongListSelector 控件定制作业的显示方式，而执行这个任务的最佳场所是 Expression Blend，但要发挥 Expression Blend 的潜能，我们需要准备一些示例数据，那么我们是否可以像上节课那样导入一些 XML 数据，然后把它们拖到 LongListSelector 控件上呢？很遗憾，不行，因为 LongListSelector 控件对于需要进行分组显示的数据源有特别要求。你可能以为我们只需把一个作业集合赋给 ItemsSource 属性，然后指定集合元素的某个属性作为分组依据，LongListSelector 控件就会自动为我们分组，但事实并非如此，LongListSelector 控件要求我们先把数据分组好，然后把这些分组凑成一个集合赋给 ItemsSource 属性，而且硬性规定每个分组至少实现 IEnumerable 接口，否则初始化时将会因为转换失败而抛出 InvalidCastException 异常。

此外，为了便于显示分组标题，每个分组最好有个属性保存标题的内容，那么我们如何创建这样的数据源？其实创建这样的数据源并不难，LINQ 的 group XXX by YYY 完全可以胜任这项任务，难处在于我们还想让它在 Expression Blend 的设计器上显示，所以我们得费一点儿周折了。

首先，切换到 Visual Studio，在 ViewModels 文件夹里创建一个 AssignmentListViewModel 类，并让它继承 NotificationObject 类，如代码 2-7 所示。



```
public class AssignmentListViewModel : NotificationObject
{
}

```

代码 2-7 AssignmentListViewModel 类

接着，创建一个 GetAssignments 方法，返回一些 Assignment 对象，如代码 2-8 所示。

```
private List<Assignment> GetAssignments()
{
    return new List<Assignment>
    {
        new Assignment
        {
            CourseName = "数学",
            StartDate = new DateTime(2010, 11, 21),
            DueDate = new DateTime(2010, 11, 22),
            Content = "课本. P9. 4题",
        },
        // ...
    };
}

```

代码 2-8 GetAssignments 方法

然后，再创建一个 AssignmentGroups 属性，通过 LINQ 选取全部数学作业并根据创建日期进行分组，如代码 2-9 所示：

```
public IEnumerable<IGrouping<DateTime, Assignment>> AssignmentGroups
{
    get
    {
        return from a in GetAssignments()
               where a.CourseName == "数学"
               group a by a.StartDate;
    }
}

```

代码 2-9 AssignmentGroups 属性

做好这些准备工作之后，我们就可以着手把示例数据关联到用户界面上了。

打开 AssignmentBookPage.xaml 文件，创建一个资源字典，并在里面创建一个 AssignmentListViewModel 对象，如代码 2-10 所示。

```

<phone:PhoneApplicationPage.Resources>
    <vm:AssignmentListViewModel
        xmlns:vm="clr-namespace:Iridescent.ViewModels"
        x:Key="assignmentListViewModel"/>
</phone:PhoneApplicationPage.Resources>

```

代码 2-10 在资源字典里创建 AssignmentListViewModel 对象

好了之后就把第一个 Pivot 项的 DataContext 属性设为上面创建的 AssignmentListViewModel 对象，并把 LongListSelector 控件的 ItemsSource 属性绑到这个对象的 AssignmentGroups 属性，如代码 2-11 所示。

```

<controls:PivotItem Header="数学"
    DataContext="{StaticResource assignmentListViewModel}">
    <Grid>
        <toolkit:LongListSelector
            ItemsSource="{Binding AssignmentGroups}"/>
    </Grid>
</controls:PivotItem>

```

代码 2-11 设置 Pivot 项的数据绑定

此时，如果你切换到 Expression Blend，它会提示你重新加载文件，因为刚才我们在 Visual Studio 里做了修改。加载完毕之后，你会看到 LongListSelector 控件里多了一些东西，如图 2-4 所示。

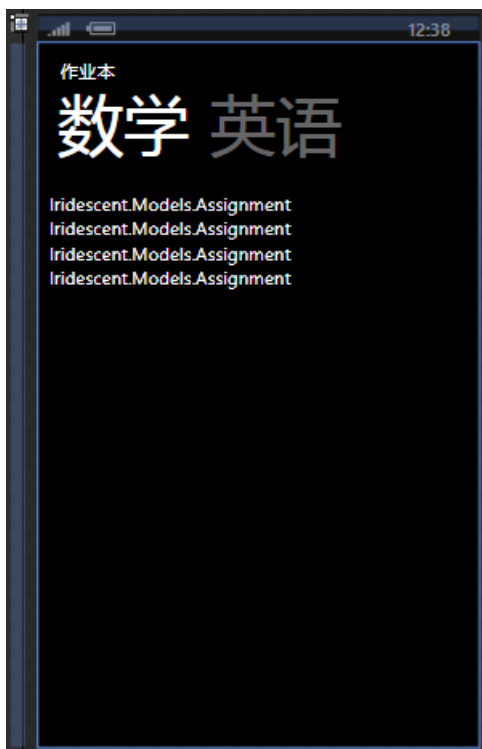


图 2-4 作业列表里多了一些东西

从上图可以看出，示例数据已经绑上去了，但为什么显示出来的是“`Iridescent.Models.Assignment`”，而且每个都是一样？这是因为 `LongListSelector` 控件并不知道如何显示 `Assignment` 对象，所以直接调用它们的 `ToString` 方法获取可以显示的内容，而我们在创建 `Assignment` 类的时候并未重写 `ToString` 方法，所以 `LongListSelector` 控件调用的是从 `Object` 类继承下来的版本，这个版本返回的是对象的类型的完全限定名，也就是我们刚才看到的“`Iridescent.Models.Assignment`”。

那么分组标题又哪去了？事实上，分组标题并未显示出来，因为 `LongListSelector` 控件并不知道分组的哪个属性表示分组标题。换句话说，`LongListSelector` 控件压根不知道如何使用我们提供的数据，而把使用方法告诉它正是我们的责任。

## 2.4 定制数据模板

### 2.4.1 定制分组标题的数据模板

首先是定制分组标题的数据模板，右击 `LongListSelector` 控件里的任何地方，选择 `Edit Additional Templates\Edit GroupHeaderTemplate\Create Empty`，如图 2-5 所示。

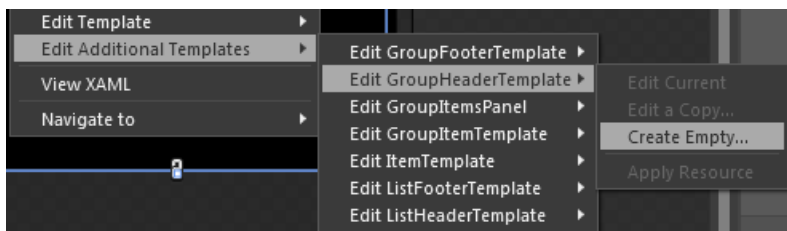


图 2-5 定制分组标题的数据模板

在弹出的 Create DataTemplate Resource 对话框里输入模板名字，如图 2-6 所示，然后按 OK 关闭对话框。

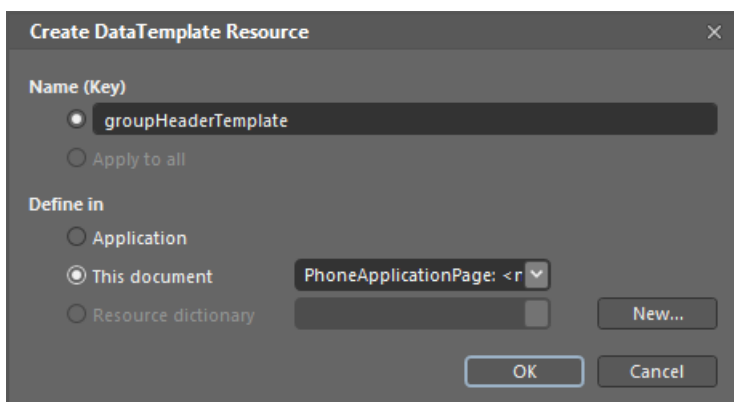


图 2-6 创建数据模板

进入模板的编辑状态之后，你会看到一个空的 Grid。从 [Tools 面板](#) 把一个 TextBlock 拖到 Grid 里，确保 TextBlock 处于选中状态（而不是编辑状态），单击 Text 属性右边的小正方形，并选择 Data Binding，如图 2-7 所示。

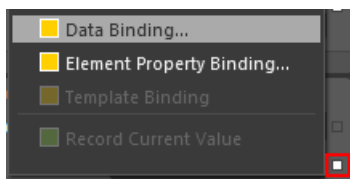


图 2-7 设置数据绑定

在弹出的 Create Data Binding 对话框里选中 Use a custom path expression，并在旁边的编辑框里输入 Key，如图 2-8 所示：

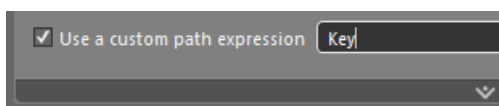


图 2-8 设置自定义路径表达式

为什么输入 Key 呢？因为通过 LINQ 的 group XXX by YYY 创建的分组对象实现了 IGrouping<TKey, TElement>接口，而这个接口有个 Key 属性保存了分组的依据——创建日期，也就是这里需要的分组标题了。

当你按 OK 关闭对话框之后，你将会看到：

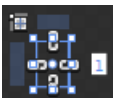


图 2-9 分组标题的数据模板

奇怪了！我们明明提供了示例数据啊，而且数据绑定也没弄错啊，为什么 TextBlock 没有任何显示？仔细观察 Text 属性下面的 DataContext 属性，如图 2-10 所示：

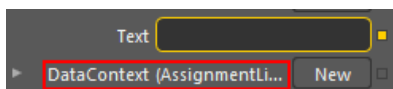


图 2-10 DataContext 属性出了问题

此时的值应该是分组对象而不是 AssignmentListViewModel 对象啊！我怀疑 LongListSelector 控件没有正确处理 DataContext 在设计时的传递（bug？），导致 Expression Blend 无法获取正确的数据。

既然如此，我们只好再弄点示例数据了，单击 Text 属性右边的编辑框，选择 Reset，然后把 Text 属性的值改为“2010/11/29”。接着，在 Objects and Timeline 面板上选中 Grid，单击 Background 属性右边的小正方形，并选择 System Resource\PhoneAccentBrush，如图 2-11 所示：

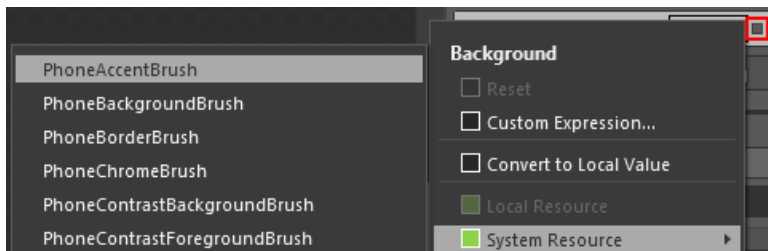


图 2-11 设置背景颜色

此时，你的 [Artboard](#) 应该是这样的：

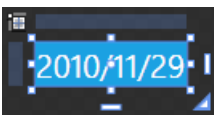


图 2-12 分组标题

退出模板的编辑状态，保存所有修改，然后重新编译项目，好了之后就能看到分组标题了，如图 2-13 所示：

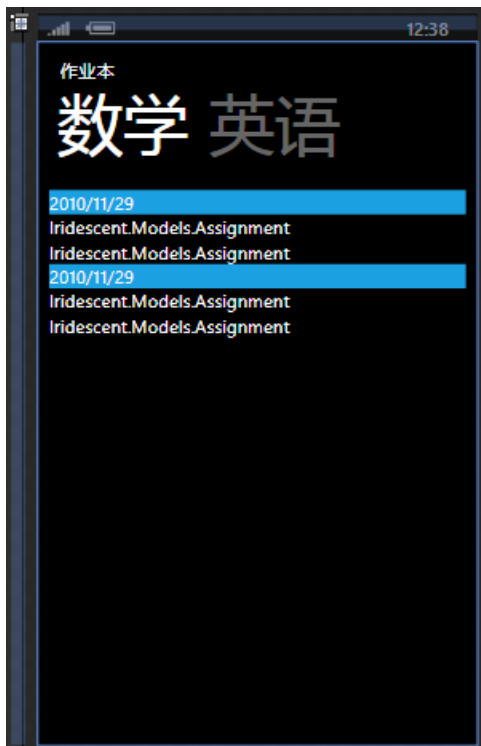


图 2-13 作业本页面

不要奇怪分组标题都是“2010/11/29”，这是我们刚才为了编辑的方便硬编码上去的结果，暂时忍耐一下吧。

#### 2.4.2 定制列表项的数据模板

接下来是列表项的数据模板，右击 LongListSelector 控件里的任何地方，选择 Edit Additional Templates\Edit ItemTemplate\Create Empty，在弹出的 Create DataTemplate Resource 对话框里输入模板名字（itemTemplate），然后按 OK 关闭对话框。

现在，我们要思考的问题是，如何更好地显示作业数据呢？回顾表 2-1，Id 属性为了便于应用程序搜索 Assignment 对象而创建的，用户并不需要知晓它的存在，所以我们不必把它呈现在用户面前，Pivot 项的标题已经显示了 CourseName 属性，分组标题也显示了 StartDate 属性，剩下的就是 DueDate、Content 和 IsCompleted 三个属性了，那么我们应该如何显示这三个属性？

此时，我的脑子里浮现出的第一个想法是这样的：

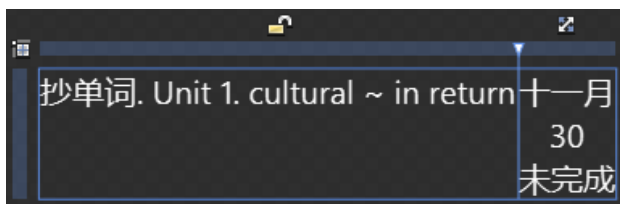


图 2-14 列表项的数据模板

整个 Grid 分为两个 Column，左边是作业内容，自动换行，右边从上到下分别是截止日期的月、日和完成状态。一般情况下，创建日期和截止日期的年份都是一样的，所以我们没有必要提供重复的信息，即使碰到跨年的情况，用户也不会因为缺少年份而感到疑惑，除非有个老师布置了一个跨越两年或以上的作业。

想到这里，我的脑子里突然闪出一个问题，表示完成状态的 TextBlock 能否去掉，并以其它方式表达这个信息呢？此时，我的脑子里迅速浮现出各种各样的图标，但是，还有更好的方式吗？颜色，突然这个词儿从我的脑子里掠过，一般而言，与文字相比，我们的大脑对颜色的反应更快更准。有鉴于此，我把列表项的模板改成这样：

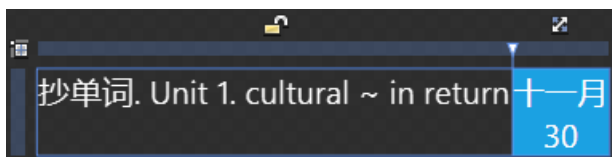


图 2-15 改进后的数据模板

右边部分将会根据作业的不同状态显示不同底色。

退出模板的编辑状态，保存所有修改，然后重新编译项目，好了之后就能看到效果了：



图 2-16 作业本页面

显然，字体的大小、控件之间的间距还不能让人满意，我们需要调整一下，这个过程可能有点反复和枯燥，但这却是我们体贴用户的重要途径，我们不但要让用户的眼睛感到满意，还要让用户的手指感到满意（别忘记我们开发的是触屏应用程序哦），下面是我调整之后的效果：





图 2-17 调整后的作业本页面

现在，我们可以再次进入模板的编辑状态，为对应的控件设置数据绑定了，做法和前面为分组标题设置数据绑定的一样（图 2-7 和图 2-8），各个控件对应的自定义路径表达式如下图所示：

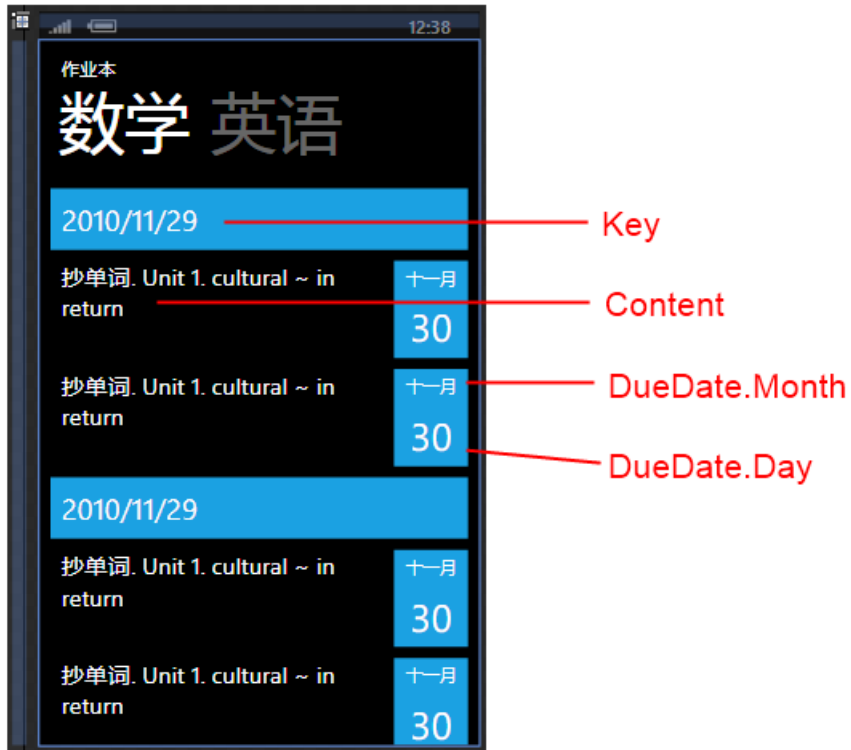


图 2-18 页面上的控件和 Assignment 类的属性之间的对应关系

好了之后就可以看到我们前面准备的示例数据了：



图 2-19 设置好数据绑定之后的作业本页面

噢，分组标题！我希望只显示日期，而且是符合中国区域设置的短日期格式，还有月份的显示，我希望是“十一月”而不是“11”。

### 2.4.3 创建日期和月份转换器

这个时候又轮到转换器出场了。首先，切换到 Visual Studio，在 Utils 文件夹里创建下面两个类，如代码 2-12 和代码 2-13 所示：

```

public class DateConverter : IValueConverter
{
    object IValueConverter.Convert(
        object value, Type targetType,
        object parameter, CultureInfo culture)
    {
        return ((DateTime)value).ToString("d", culture);
    }

    object IValueConverter.ConvertBack(
        object value, Type targetType,
        object parameter, CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}

```

代码 2-12 DataConverter 类

```

public class MonthNameConverter : IValueConverter
{
    object IValueConverter.Convert(
        object value, Type targetType,
        object parameter, CultureInfo culture)
    {
        return
            culture.DateTimeFormat.GetAbbreviatedMonthName((int)value);
    }

    object IValueConverter.ConvertBack(
        object value, Type targetType,
        object parameter, CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}

```

代码 2-13 MonthNameConverter 类

需要说明的是，因为我们的绑定是单向的，所以没有必要实现 `ConvertBack` 方法。

接着，在 `AssignmentBookPage.xaml` 的资源字典里创建它们的实例，如代码 2-14 所示：

```

<utils:DateConverter
    xmlns:utils="clr-namespace:Iridescent.Uutils"
    x:Key="dateConverter"/>
<utils:MonthNameConverter
    xmlns:utils="clr-namespace:Iridescent.Uutils"
    x:Key="monthNameConverter"/>

```

代码 2-14 创建转换器对象

看到这里，你可能会问，这两个转换器的Convert方法都使用了culture这个参数，但我们没有直接调用Convert方法啊，那我们怎么把这个参数传给它？这可以通过设置绑定表达式的ConverterCulture属性做到，现在，把那两个TextBlock的Text属性的绑定表达式改为

“{Binding Key, Converter={StaticResource dateConverter}, ConverterCulture=zh-CN}”和  
 “{Binding DueDate.Month, Converter={StaticResource monthNameConverter}, ConverterCulture=zh-CN}”。

#### 2.4.4 创建底色转换器

剩下的就是截止日期的底色了，既然转换器可以把DateTime对象转换成字符串，它也应该可以把Assignment对象转换成SolidColorBrush对象，不过，在创建这个转换器之前，我们得先弄清楚什么状态对应什么底色。

前面我们说过，作业本的主要目的是让学生对要做哪些作业一目了然，而“未完成”的作业里可能存在一些已经过了截止日期的，这类作业需要马上处理，所以我们应该单独为这类作业设置一种底色，以使用户及时知晓并采取行动。假设这三种状态及其对应的底色如下表所示（你也可以换成其它底色）：

状态	底色
已逾期	Red
未完成	#FF1BA1E2
已完成	Green

表 2-2 各种作业状态对应的底色

那么转换器的Convert方法可以这样实现：

```

object IValueConverter.Convert(
    object value, Type targetType,
    object parameter, CultureInfo culture)
{
    var assignment = (Assignment)value;

    if (assignment.IsCompleted)
    {
        return new SolidColorBrush(Colors.Green);
    }
    else
    {
        if (assignment.DueDate < DateTime.Today)
        {
            return new SolidColorBrush(Colors.Red);
        }
        else
        {
            return
                new SolidColorBrush(Color.FromArgb(255, 27, 161, 226));
        }
    }
}

```

代码 2-15 底色转换器的 Convert 方法

接着，在AssignmentBookPage.xaml的资源字典里创建它的实例（参考代码2-14），并把那个StackPanel的Background属性的绑定表达式改为“{Binding Converter={StaticResource assignmentToBrushConverter}}”。

好了之后就编译一下，没问题的话就可以看到效果了（你也可以在 Visual Studio 里看）：



图 2-20 作业本页面根据作业的完成状态显示底色

看到这里，你可能会问，“未完成”的底色和分组标题的底色是一样的，为什么不直接使用 PhoneAccentBrush 这个系统资源呢？这是因为用户有可能在手机的 Settings 里把 Accent Color 设成和其它状态一样的颜色，这会导致两种不同的状态应用相同的底色，而用户也有可能因此获得错误的信息。

现在，如果你试图编辑列表项的数据模板，你将会看到此番情景：

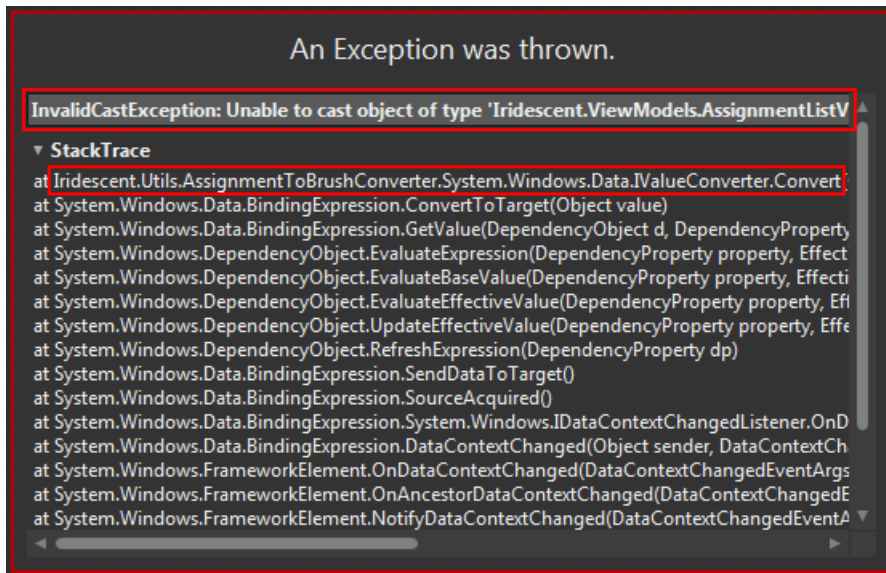


图 2-21 数据模板抛出异常

怎么回事？！

从上图不难看出，底色转换器的 Convert 方法抛出一个 InvalidCastException 异常，而异常的信息也明确告诉我们无法把对象从 AssignmentListViewModel 类型转换成 Assignment 类型（把滚动条向右拖动就可以看到了）。在 Convert 方法里，我们只做过一次转换，就是在开始的时候把 value 参数转换成 Assignment 类型（参见代码 2-15），因为此时的 DataContext 应该是 Assignment 对象，但上面这个异常却告诉我们 value 参数不是 Assignment 对象！为什么会这样？还记得前面编辑分组标题的数据模板时，即时我们设置好数据绑定也看不到示例数据（参见图 2-9），当时我们猜测 LongListSelector 控件没有正确处理 DataContext 在设计时的传递（参见图 2-10），而这个猜测在这里得到了证实。

明白为什么会这样，问题就不难解决了，把 Convert 方法开始那行强制转换换成下面这段代码：

```
var assignment = value as Assignment;

if (assignment == null)
{
    return
        (SolidColorBrush)Application.Current.Resources[
            "PhoneAccentBrush"];
}
```

代码 2-16 处理转换不成功的情况

编译一下。现在，如果你尝试编辑列表项的数据模板，你不会再看到上面的异常了。



#### 2.4.5 测试应用

接下来干嘛？你懂的！打开 MainPage.xaml，添加一个菜单项，并让它导航至 AssignmentBookPage 页，如图 2-22 所示：

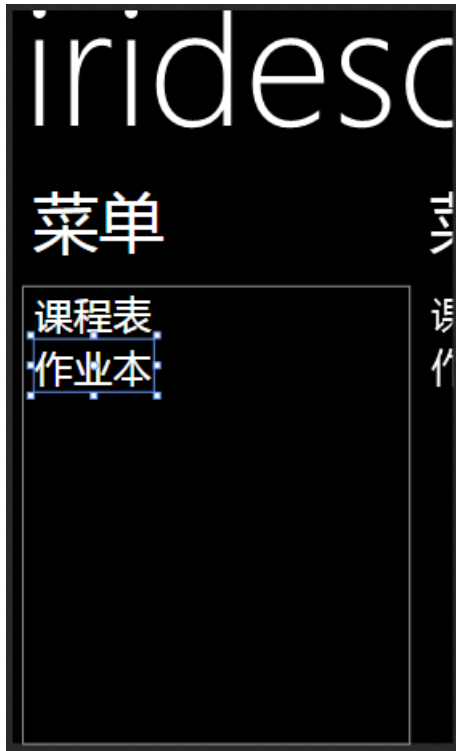


图 2-22 添加作业本菜单项

好了，按 F5 吧……

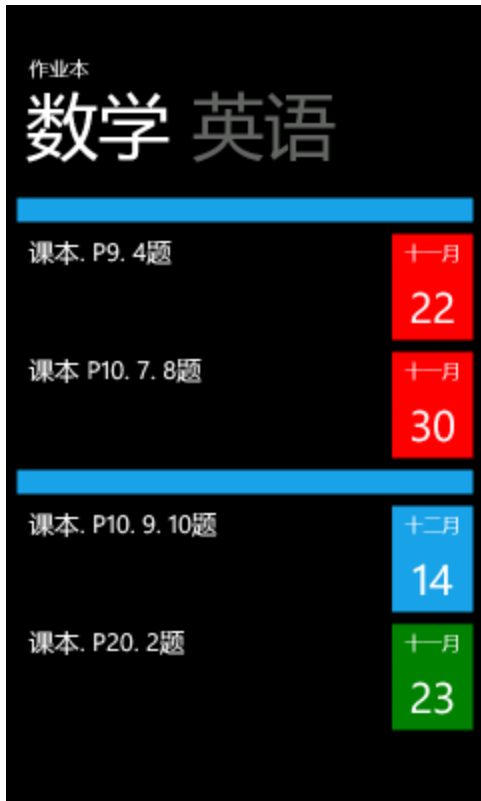


图 2-23 运行作业本

啊？！分组标题哪去啦？？

## 2.5 插曲 #1

究竟发生了什么事？示例数据和绑定表达式应该都没问题啊，否则 Expression Blend 和 Visual Studio 的设计器也不会正常显示，那么问题到底出在哪里呢？突然，一个想法在我的脑子里闪过，如果我在 `DateConverter` 类的 `Convert` 方法里设个断点，你觉得会怎么样？试一下吧……结果是，没有到达这个断点，换句话说，`Convert` 方法根本没被调用！这种情况有点像数据绑定找不到分组对象的 `Key` 属性，比如说，我故意把绑定表达式的 `Key` 改为 `Key1`，结果 Expression Blend 的设计器就变成这样了：

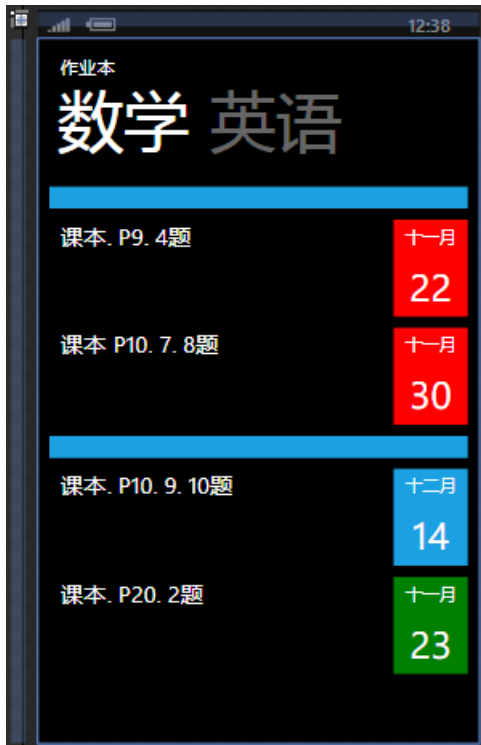


图 2-24 分组标题没有显示

我们知道，分组对象实现了 `IGrouping<TKey, IElement>` 接口，因此 `Key` 属性肯定存在，否则编译器会报错，那么，什么情况下这个属性是不可见的，或者说，有什么办法可以让它不可见？想到这里，一个词儿突然在我的脑子里冒出来——显式接口实现！如果 `Key` 属性是显式实现的，仅当变量的类型是 `IGrouping<TKey, IElement>` 时 `Key` 属性才是可见的。看到这里，你可能会说，Silverlight 不可能直接调用分组对象的 `Key` 属性，它应该是通过反射获取这个属性的。没错，当我们在绑定表达式里以字符串的形式给出属性路径，`PropertyPathConverter` 对象将会把这个字符串转换成 `PropertyPath` 对象，那么，`PropertyPath` 对象又是如何找到对应的属性呢？在[微软公开的.NET Framework 4.0 源代码](#)里，我找到了 `PropertyPath` 类的实现，里面有个 `GetPropertyHelper` 方法负责获取指定的属性，如代码 2-17 所示：

```
private PropertyInfo GetPropertyHelper(Type ownerType)
{
    PropertyInfo result;
    try
    {
        result = ownerType.GetProperty(propertyName);
    }
    catch (AmbiguousMatchException)
```

代码 2-17 `GetPropertyHelper` 方法

如果 Key 属性是显式实现的话，GetProperty 方法就会返回 null！换句话说，数据绑定和显式实现的属性一起工作的话会出问题。那么，group XXX by YYY 返回的分组对象是不是显示实现 Key 属性的呢？我们知道，使用 group XXX by YYY 实质上就是调用 Enumerable 类的 GroupBy 方法，经过一番查找，我发现它返回的分组对象就是 Lookup 类内部的 Grouping 类的实例，但 Grouping 类的 Key 属性是隐式实现的，有趣的是，Key 属性上方有一段注释，如代码 2-18 所示：

```
// DDB195907: implement IGrouping<>.Key implicitly
// so that WPF binding works on this property.
public TKey Key {
    get { return key; }
}
```

代码 2-18 Key 属性

除了 Key 属性之外，Grouping 类的其它属性都是显式实现的，我猜 Key 属性原来也是显式实现的，后来由于数据绑定的问题才改为隐式实现。

这些代码是 WPF 4.0 的，而 Key 属性上面的注释也明确提到了 WPF，这是不是说 Key 属性的值在 WPF 里可以正确显示？我们可以设计一个简单的实验来验证一下：

1. 创建一个 ListBox。
2. 定制 ListBox 的 ItemTemplate，里面只放一个 TextBlock。
3. 把 TextBlock 的 Text 属性设为“{Binding Key}”。
4. 通过 GroupBy 方法创建分组对象的集合，并把它绑到 ListBox 的 ItemsSource 属性。
5. 按 F5。

我分别在 WPF 4.0、SL 4.0 和 SL for WP7 上执行这个实验，发现只有 WPF 4.0 能够正确显示 Key 属性的值，其它两个的 ListBox 是一片空白的。我怀疑 SL 的分支是在这个问题得到修复之前创建的，但我没有代码证实这个猜想。

还有一个问题我没弄明白的，为什么设计器能够正确显示而程序真正运行的时候却不能？难道设计器对显式实现的属性有什么特别的照顾？为了验证这个猜想，我又做了一个实验，我不直接返回分组对象，而是通过下面这个 Grouping 类包装一下再返回：

```

public class Grouping : IGrouping<DateTime, Assignment>
{
    public Grouping(IGrouping<DateTime, Assignment> g)
    {
        _grouping = g;
    }

    private IGrouping<DateTime, Assignment> _grouping;

    DateTime IGrouping<DateTime, Assignment>.Key
    {
        get { return _grouping.Key; }
    }

    // ...
}

```

代码 2-19 Grouping 类

结果，设计器也不显示了……我不知道为什么设计器能够正确显示 **GroupBy** 方法返回的分组对象的 **Key** 属性，这里面肯定有些东西是我不知道的，如果你知道原因，或者先我一步找到原因，那你一定要告诉我哦！

## 2.6 连接前端和后端

### 2.6.1 创建 AssignmentGroupViewModel 类

既然显式实现的属性会对数据绑定造成不良影响，那我们就换成隐式实现吧。首先，在 **ViewModels** 文件夹里创建 **AssignmentGroupViewModel** 类，并让它继承 **ObservableCollection<Assignment>** 类，如代码 2-20 所示。

```

public class AssignmentGroupViewModel :
    ObservableCollection<Assignment>
{
}

```

代码 2-20 AssignmentGroupViewModel 类

为什么要继承 **ObservableCollection<Assignment>** 类呢？前面说过，**LongListSelector** 控件硬性规定分组对象至少实现 **IEnumerable** 接口，不过，要想获得更好的效果，仅仅实现 **IEnumerable** 接口是不够的，**LongListSelector** 控件通过内部的 **GetItemsInGroup** 方法来获取分组内容，如代码 2-21 所示：

```

private static IList GetItemsInGroup(object group)
{
    IList itemsList = group as IList;
    if (group != null)
    {
        return itemsList;
    }

    List<object> items = new List<object>();

    IEnumerator itemEnum = ((IEnumerable)group).GetEnumerator();
    bool hasItems = itemEnum.MoveNext();

    while (hasItems)
    {
        items.Add(itemEnum.Current);
        hasItems = itemEnum.MoveNext();
    }

    return items;
}

```

代码 2-21 GetItemsInGroup 方法

从上面代码不难看出，如果分组对象实现了 `IList` 接口，那么每次获取分组内容时都会免掉一次遍历。此外，我们还希望当分组内容发生改变时，比如新建/删除一项作业，分组对象能够自动通知 `LongListSelector` 控件做出相应的更新，为了实现这个效果，分组对象需要实现 `INotifyCollectionChanged` 接口。毫无疑问，能够一次过满足我们所有要求的最简单做法就是继承 `ObservableCollection<Assignment>` 类了。

看到这里，你可能会问，`IGrouping<TKey, TElement>` 接口不用实现吗？不用，`LongListSelector` 控件没有规定分组对象必须实现这个接口，我们只需简单地创建一个 `Key` 属性，如代码 2-22 所示，配合绑定表达式里的属性路径就行了：

```
private DateTime _Key;
public DateTime Key
{
    get { return _Key; }
    set
    {
        _Key = value;
        OnPropertyChanged(
            new PropertyChangedEventArgs("Key"));
    }
}
```

代码 2-22 Key 属性

需要说明的是，ObservableCollection<Assignment>类也实现了 INotifyPropertyChanged 接口，所以我们可以直接使用它的 OnPropertyChanged 方法。

### 2.6.2 初始化 AssignmentGroupViewModel 对象

接下来是分组对象的初始化，这个过程的主要任务有两个：

1. 查询数据源，把满足条件的作业内容添加到自身。
2. 监听数据源，把满足条件的内容更改反映到自身。

执行这两个任务的前提是有个可用的数据源，我们可以仿效课程表的做法，在 App 类里通过静态属性提供 JsonDataStore<Assignment>对象，如代码 2-23 所示：

```
private static IDataStore<Assignment> _assignmentStore = null;
public static IDataStore<Assignment> AssignmentStore
{
    get
    {
        if (_assignmentStore == null)
        {
            _assignmentStore =
                new JsonDataStore<Assignment>("Assignments.json");
        }

        return _assignmentStore;
    }
}
```

代码 2-23 AssignmentStore 静态属性

有了数据源我们就可以着手执行第一个任务了，如代码 2-24 所示：

```

public AssignmentGroupViewModel(string courseName, DateTime startDate)
{
    _Key = startDate;

    Func<Assignment, bool> predicate =
        a => a.CourseName == courseName &&
            a.StartDate == startDate;

    App.AssignmentStore.Items.Where(predicate).ForEach(Add);
}

```

代码 2-24 查询作业本数据源

需要说明的是，这里把判断条件单独提取出来了，因为执行第二个任务时还要用到，如代码 2-25 所示：

```

App.AssignmentStore.Items.CollectionChanged +=
    (o, e) =>
    {
        if (e.Action == NotifyCollectionChangedEventArgs.Add)
        {
            var assignment = (Assignment)e.NewItems[0];

            if (predicate(assignment))
            {
                Add(assignment);
            }
        }
        else if (e.Action == NotifyCollectionChangedEventArgs.Remove)
        {
            var assignment = (Assignment)e.OldItems[0];

            if (Contains(assignment))
            {
                Remove(assignment);
            }
        }
    };
}

```

代码 2-25 监听作业本数据源

需要说明的是，e 参数的 NewItems 和 OldItems 两个属性看起来好像可能包含多个元素，但事实上它们只会包含一个，因为 NotifyCollectionChangedEventArgs 类的构造函数限制了这个可能，不过这个限制仅存在于 Silverlight 的现有版本（SL3、SL4、SL for WP7）。



另外，这里使用了 Lambda 语句来创建 `CollectionChanged` 事件的处理程序，虽然你也可以通过一个单独的方法做到，但使用 Lambda 语句可以利用闭包的特点重用前面的判断条件，当然，使用匿名方法的语法也是可以的。

还差什么呢？噢，对了，`LongListSelector` 控件内部会调用分组对象的 `Equals` 方法进行判等，我们可以重写 `AssignmentGroupViewModel` 类的 `Equals` 和 `GetHashCode` 两个方法，使之根据 `Key` 属性来判等以及获取哈希值。这个任务留给你当课后作业吧。

### 2.6.3 修改 `AssignmentListViewModel` 类

既然分组对象的类型改了，那 `AssignmentListViewModel` 类的 `AssignmentGroups` 属性也得做出相应的调整吧：

```
public ObservableCollection<AssignmentGroupViewModel> AssignmentGroups
{
    get;
    private set;
}
```

代码 2-26 调整后的 `AssignmentGroups` 属性

由于 `AssignmentListViewModel` 类对应用户界面上的 `Pivot` 项，我们还需要给它创建一个 `Title` 属性，如代码 2-27 所示：

```
private string _title;
public string Title
{
    get { return _title; }
    set
    {
        _title = value;
        RaisePropertyChanged("Title");
    }
}
```

代码 2-27 `Title` 属性

有了这些准备，我们就可以着手实现 `AssignmentListViewModel` 类的构造函数了，如代码 2-28 所示：

```

public AssignmentListViewModel(string courseName)
{
    _title = courseName;

    AssignmentGroups =
        new ObservableCollection<AssignmentGroupViewModel>();

    App.AssignmentStore.Items
        .Where(a => a.CourseName == courseName)
        .Select(a => a.StartDate)
        .Distinct()
        .Select(d => new AssignmentGroupViewModel(courseName, d))
        .ForEach(AssignmentGroups.Add);
}

```

代码 2-28 初始化 `AssignmentGroups` 属性

看到这里，你可能会说，这条 LINQ 语句看起来有点复杂嘛！其实不然，想想看，我们的最终目的是什么？创建分组对象并把它们添加到 `AssignmentGroups` 属性。那创建分组对象需要什么条件？课程名称和创建日期。课程名称已经有了，创建日期来自哪里？来自数据源。那我们对创建日期有些什么要求？我们只要和指定课程相关的，而且不要重复的。现在，你再看看上面这条 LINQ 语句，从上往下看，有没有觉得它像下面这条“流水线”？

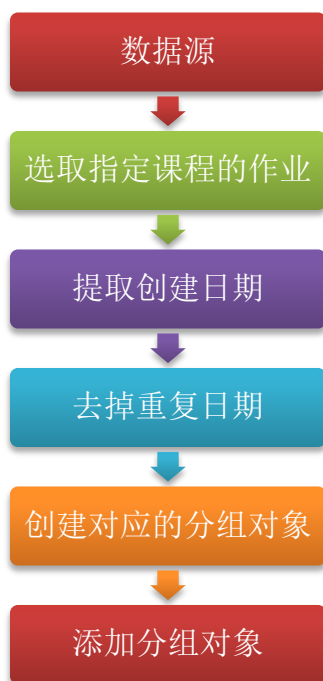


图 2-25 创建 `AssignmentGroups` 对象的过程

前面我们说过，当用户新建一项作业时，它会自动添加到“今天”的分组里，但如果“今天”的分组还没创建出来呢？那 `AssignmentListViewModel` 类就应该为这项新的作业创建“今天”的分组，并把它添加到 `AssignmentGroups` 属性，如代码 2-29 所示：

```
App.AssignmentStore.Items.CollectionChanged +=
    (o, e) =>
    {
        if (e.Action == NotifyCollectionChangedAction.Add)
        {
            var startDate = ((Assignment)e.NewItems[0]).StartDate;

            if (AssignmentGroups.All(g => g.Key != startDate))
            {
                AssignmentGroups.Add(
                    new AssignmentGroupViewModel(
                        courseName, startDate));
            }
        }
        else if (e.Action == NotifyCollectionChangedAction.Reset)
        {
            AssignmentGroups.Clear();
        }
    };
}
```

代码 2-29 监听作业本数据源

当用户删除一项作业时，如果这项作业是所属分组的唯一一项作业，`LongListSelector` 控件会自动隐藏这个分组。而当用户撤销所有更改时，`AssignmentListViewModel` 类得把 `AssignmentGroups` 属性清空。

#### 2.6.4 创建 `AssignmentBookViewModel` 类

到目前为止，`AssignmentBookPage` 页里的每个组成部分都有对应的 `ViewModel` 类了，现在是时候为它创建一个了。在 `ViewModels` 文件夹里创建一个 `AssignmentBookViewModel` 类，并创建一个 `AssignmentLists` 属性，如代码 2-30 所示。

```
public ObservableCollection<AssignmentListViewModel> AssignmentLists
{
    get;
    private set;
}
```

代码 2-30 `AssignmentLists` 属性

AssignmentBookViewModel 类的任务是读取课程表的数据，然后创建对应的 AssignmentListViewModel 对象，如代码 2-31 所示。

```
public AssignmentBookViewModel()
{
    AssignmentLists =
        new ObservableCollection<AssignmentListViewModel>();

    App.CourseStore.Items
        .Select(c => c.Name)
        .Distinct()
        .Select(n => new AssignmentListViewModel(n))
        .ForEach(AssignmentLists.Add);
}
```

代码 2-31 AssignmentBookViewModel 类的构造函数

看到这里，你可能会问，为什么这里不用监听数据源的更改？如果你要编辑课程表，一定要进入课程表的用户界面，一旦离开课程表的用户界面，课程表的数据就会冻结下来，换句话说，在 AssignmentBookViewModel 对象的整个生命周期里，课程表的数据是稳定的。

### 2.6.5 设置数据绑定

现在，我们可以着手处理数据绑定了。打开 AssignmentBookPage.xaml 文件，切换到 XAML 模式，在页面的资源字典里添加两个数据模板，如代码 2-32 所示。

```
<DataTemplate x:Key="pivotHeaderTemplate">
    <Grid>
        <TextBlock Text="{Binding Title}"/>
    </Grid>
</DataTemplate>
<DataTemplate x:Key="pivotItemTemplate">
    <Grid>
        <toolkit:LongListSelector
            ItemsSource="{Binding AssignmentGroups}"
            GroupHeaderTemplate="{StaticResource groupHeaderTemplate}"
            ItemTemplate="{StaticResource itemTemplate}"/>
    </Grid>
</DataTemplate>
```

代码 2-32 在资源字典里添加数据模板

接着，把现有的 Pivot 项删除，并在 Pivot 控件上设置数据模板和数据绑定，如代码 2-33 所示。

```
<controls:Pivot
    Title="作业本"
    ItemsSource="{Binding AssignmentLists}"
    HeaderTemplate="{StaticResource pivotHeaderTemplate}"
    ItemTemplate="{StaticResource pivotItemTemplate}"/>
```

代码 2-33 设置数据绑定

最后在 AssignmentBookPage 的构造函数里创建一个 AssignmentBookViewModel 对象，并把它赋给 DataContext 属性，如代码 2-34 所示。

```
public AssignmentBookPage()
{
    InitializeComponent();

    DataContext = new AssignmentBookViewModel();
}
```

代码 2-34 初始化 DataContext 属性

#### 2.6.6 测试应用

好了，不知不觉又到看效果的时候了！按 F5 运行应用程序：



图 2-26 主菜单

单击“课程表”菜单项进入课程表，新建两个课程，如图 2-27 所示，保存，然后按 Back 键返回主菜单：



图 2-27 课程表页面

在主菜单里单击“作业本”菜单项进入作业本，此时，你会看到作业本已经为刚才创建的两个课程准备了两个 Pivot 项，如图 2-28 所示：

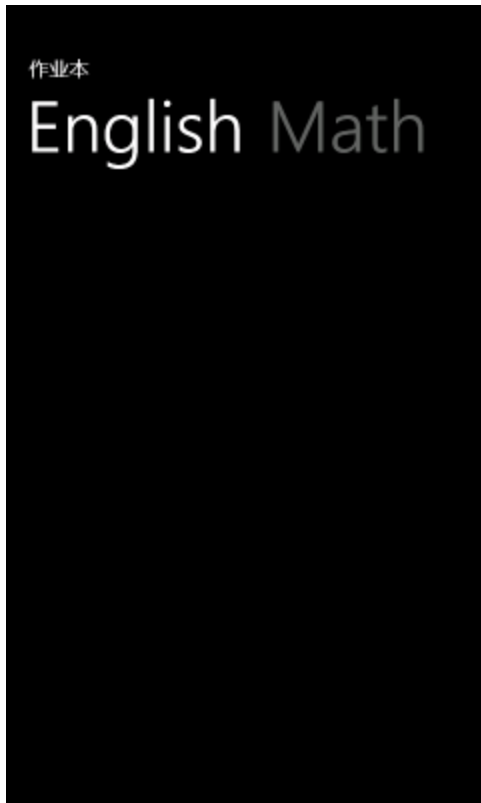


图 2-28 作业本页面

只是作业本上没有任何内容，也没有任何途径可以添加内容……

## 2.7 编辑作业本

### 2.7.1 创建 Application Bar

作业本支持的操作和课程表一样，包括新建、编辑、删除、保存所有更改和撤销所有更改，其中，新建和保存以 `ApplicationBarIconButton` 的方式放在 `Application Bar` 上，撤销所有更改以 `ApplicationBarMenuItem` 的方式放在 `Application Bar` 上，而编辑和删除则放在上下文菜单里，如图 2-29 所示。

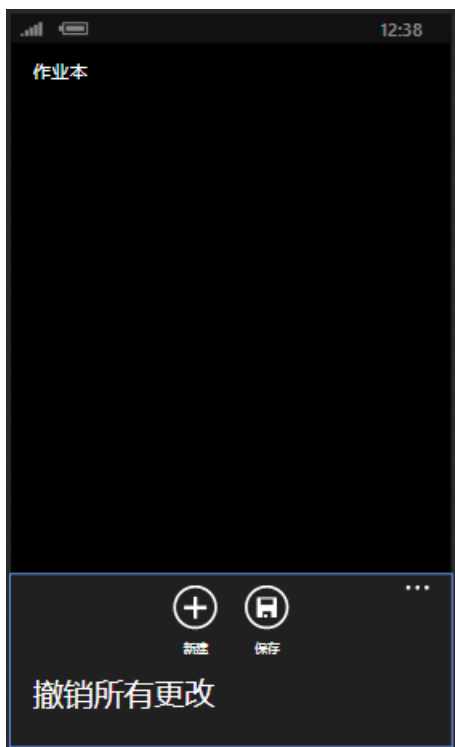


图 2-29 Application Bar

为什么这样安排？当老师布置作业时，我们会掏出作业本记下作业，下课之后，当我们要做作业时，我们会掏出作业本看看要做哪些作业，换句话说，新建、保存和显示作业内容这三个功能已经可以满足用户绝大多数的需求了。新建和保存作为最常用的两个操作自然应该放在最显眼的位置，删除和撤销所有更改这两个操作基本上不会用到，至于编辑，一般情况下我们只是用来修改作业的完成状态，由于编辑和删除是针对特定作业的，我们把它放在上下文菜单里，当用户长按某项作业时将会显示出来，而撤销所有更改则隐藏在 Application Bar 的菜单里。

### 2.7.2 创建新建/编辑作业页面

接着，创建一个 Windows Phone Page，并把它命名为 `NewOrEditAssignmentPage.xaml`，这个页面会在用户单击 Application Bar 上的新建按钮或者上下文菜单上的编辑菜单项时显示。完了之后把 `ApplicationTitle` 的 `Text` 属性值改为“作业本”，但 `PageTitle` 保留原样，如图 2-30 所示。



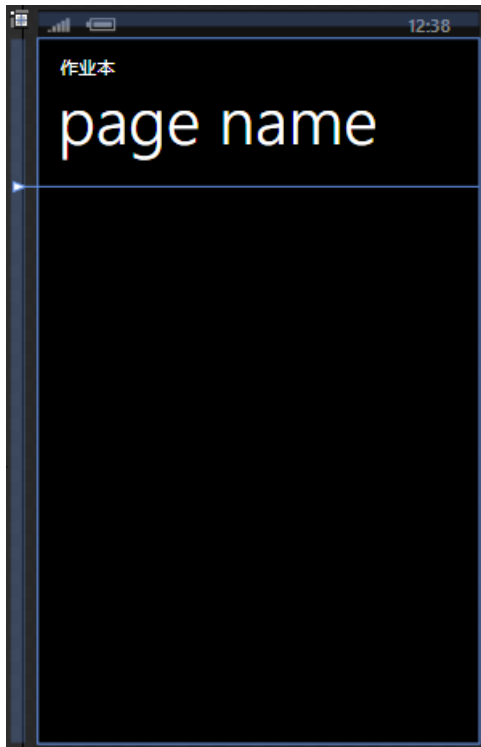


图 2-30 空的新建/编辑作业页面

那么，这个页面应该放些什么控件呢？想想看，创建一个完整的 **Assignment** 对象需要哪些数据？**Id** 是自动生成的，课程名称可以从上下文获取，创建日期可以从 **DateTime** 的 **Today** 属性获取，剩下的就是截止日期、作业内容和完成状态了。截止日期可以使用 **SL for WP Toolkit** 的 **DatePicker** 控件，作业内容可以使用 **TextBox** 控件（上面的标题需要额外添置 **TextBlock** 控件），而完成状态则可以使用 **CheckBox** 控件，如图 2-31 所示。



图 2-31 新建/编辑作业页面

看到这里，你可能会问，为什么不把其它信息也显示出来呢？你可以这样做，但是，请注意，这个页面的主要目的是收集而不是显示信息，我们应该尽可能简化用户的输入过程，在这里放置控件显示其它信息，尤其是可编辑的控件，可能会耗费用户额外的注意力，比如说，有些用户会下意识地检查所有数据是否输入正确。创建作业的过程应该是既简单又快速的，而我们也希望用户能有这样的感受，但耗费用户额外的注意力意味着增加整个操作过程的时间，从而可能导致用户的感受和我们期望的刚好相反，这是我们不希望看到的。

### 2.7.3 创建 ViewModel 类

ViewModel 类方面，我们将会仿效课程表的做法，创建 `NewOrEditAssignmentViewModel`、`NewAssignmentViewModel` 和 `EditAssignmentViewModel` 三个类，如图 2-32 所示。

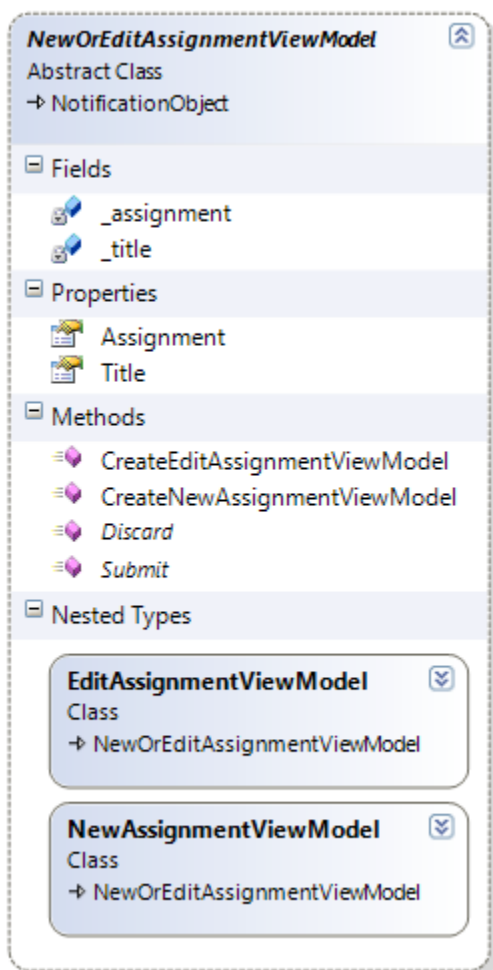


图 2-32 ViewModel 类的类图

我们知道，**NewOrEditAssignmentPage** 页有两个模式，一个是新建模式，另一个是编辑模式，前者对应 **NewAssignmentViewModel** 类，而后者则对应 **EditAssignmentViewModel** 类。当用户新建一项作业时，**NewAssignmentViewModel** 类可以从 **DateTime** 的 **Today** 属性获取创建日期，但它没法获取课程名称，所以我们需要通过参数传给它，如代码 2-35 所示。

```

public NewAssignmentViewModel(string courseName)
{
    Title = "新建作业";
    Assignment = new Assignment
    {
        CourseName = courseName,
        StartDate = DateTime.Today,
        DueDate = DateTime.Today.AddDays(1),
    };
}

```

代码 2-35 NewAssignmentViewModel 类的构造函数

为什么 DueDate 属性也要设置呢？想想看，如果我们不给它设置一个值，由于 DateTime 是值类型，将被自动初始化为“1/1/0001”，当用户看到页面上的 DatePicker 控件显示这样一个日期可能会感到不友好，再者，老师布置下来的作业一般不会当天交（课堂作业除外），而第二天交的情况则比较常见（当然，计算下一个“上课日”可能更加合理）。

而当用户编辑一项作业时，EditAssignmentViewModel 类将会从数据源里查找这项作业的数据，但前提是我们把作业的 Id 告诉它，如代码 2-36 所示：

```

public EditAssignmentViewModel(Guid id)
{
    Title = "编辑作业";
    _source = App.AssignmentStore.Items.First(
        a => a.Id == id);
    Assignment = new Assignment(id)
    {
        CourseName = _source.CourseName,
        StartDate = _source.StartDate,
        DueDate = _source.DueDate,
        Content = _source.Content,
        IsCompleted = _source.IsCompleted,
    };
}

```

代码 2-36 EditAssignmentViewModel 类的构造函数

需要说明的是，Assignment 类的 Id 属性是只读的，而 Assignment 类原来的构造函数会在每次调用时创建一个新的 Id，这导致了无法使用现有的 Id，所以我们需要在 Assignment 类里添加下面这个构造函数，如代码 2-37 所示：

```
public Assignment(Guid id)
{
    Id = id;
}
```

代码 2-37 为 Assignment 类添加一个接受现有 Id 的构造函数

2.7.4 关联 ViewModel 类和对应的页面

创建好 ViewModel 类之后，我们就可以着手处理它们和 NewOrEditAssignmentPage 页之间的关联了。首先是设置数据绑定，需要设置的控件以及对应的绑定表达式如下表所示：

描述	类型	属性	绑定表达式
页面标题	TextBlock	Text	{Binding Title}
截止日期	DatePicker	Value	{Binding Assignment.DueDate, Mode=TwoWay}
作业内容	TextBox	Text	{Binding Assignment.Content, Mode=TwoWay}
完成状态	CheckBox	IsChecked	{Binding Assignment.IsCompleted, Mode=TwoWay}

表 2-3 各个空间的绑定表达式

接着，重写 OnNavigatedTo 方法：

```
base.OnNavigatedTo(e);

if (DataContext == null)
{
    var action = NavigationContext.QueryString["action"];
    if (action == "new")
    {
        DataContext =
            NewOrEditAssignmentViewModel.CreateNewAssignmentViewModel(
                NavigationContext.QueryString["coursename"]);
    }
    else
    {
        DataContext =
            NewOrEditAssignmentViewModel.CreateEditAssignmentViewModel(
                new Guid(NavigationContext.QueryString["id"]));
    }
}
```

代码 2-38 重写 OnNavigatedTo 方法

最后，为两个按钮创建事件处理程序，如代码 2-39 所示：

```
private void OKButton_Click(object sender, RoutedEventArgs e)
{
    ((NewOrEditAssignmentViewModel)DataContext).Submit();
    NavigationService.GoBack();
}

private void CancelButton_Click(object sender, RoutedEventArgs e)
{
    NavigationService.GoBack();
}
```

代码 2-39 确定/取消按钮的事件处理程序

由于这部分内容和上节课的大同小异，这里就不详细解释了。

### 2.7.5 实现新建和保存操作

接下来是实现前面提到的五个操作。首先是最常用的新建和保存。保存操作非常简单，我们只需为它创建一个事件处理程序就行了，如代码 2-40 所示：

```
private void ApplicationBarCommitIconButton_Click(
    object sender, EventArgs e)
{
    App.AssignmentStore.Commit();
}
```

代码 2-40 保存按钮的 Click 事件处理程序

而新建则有点难度，我们需要获取课程名称，怎么获取？我们知道，课程名称实际上就是 Pivot 项的标题，也就是 AssignmentListViewModel 的 Title 属性，只要我们知道当前显示的是哪个 AssignmentListViewModel 对象就可以了。为此，我们需要在 AssignmentBookViewModel 类里添加一个 SelectedListIndex 属性，如代码 2-41 所示：

```
private int _selectedListIndex;
public int SelectedListIndex
{
    get { return _selectedListIndex; }
    set
    {
        _selectedListIndex = value;
        RaisePropertyChanged("SelectedListIndex");
    }
}
```

代码 2-41 SelectedListIndex 属性

并为它和 Pivot 控件的 SelectedIndex 属性设置双向绑定。此外，需要说明的是，为了使用 RaisePropertyChanged 方法，我们需要让 AssignmentBookViewModel 类继承 NotificationObject 类。

有了这些准备，我们就可以创建事件处理程序了，如代码 2-42 所示：

```
private void ApplicationBarNewIconButton_Click(  
    object sender, EventArgs e)  
{  
    var book = (AssignmentBookViewModel)DataContext;  
    if (book.AssignmentLists.Count > 0)  
    {  
        var courseName =  
            book.AssignmentLists[book.SelectedIndex].Title;  
        NavigationService.Navigate(  
            new Uri(  
                "/NewOrEditAssignmentPage.xaml?action=new&" +  
                "coursename=" + courseName,  
                UriKind.RelativeOrAbsolute));  
    }  
    else  
    {  
        MessageBox.Show("课程表还没创建。");  
    }  
}
```

代码 2-42 新建按钮的 Click 事件处理程序

需要说明的是，如果课程表里面没有课程，Pivot 控件就不会创建 Pivot 项，所以在做进一步处理之前，我们需要判断 AssignmentLists 里面有没有东西。

### 2.7.6 测试应用

好了，又到看效果的时候了！按 F5 运行应用程序，在主菜单里单击“作业本”菜单项进入作业本，如图 2-33 所示：

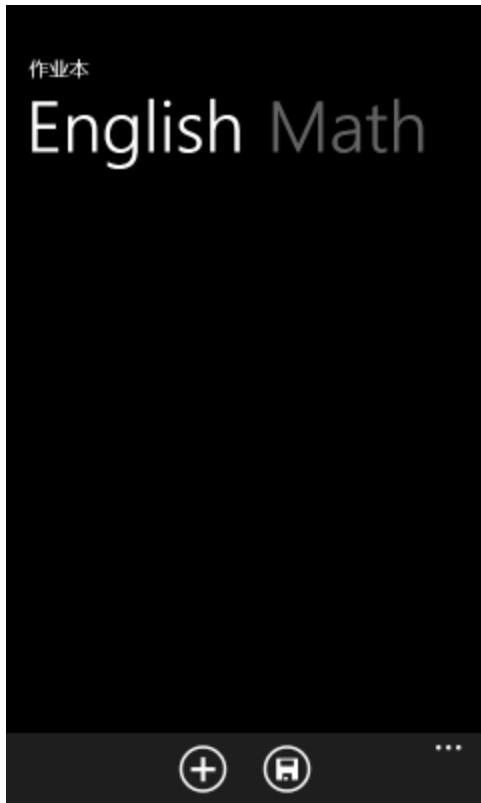


图 2-33 作业本页面

接着，单击 Application Bar 上的新建按钮创建一项作业，如图 2-34 所示：



作业本

# 新建作业

截止日期

12/9/2010

作业内容

words of unit 1, cultural - in return

☐ 已完成

确定 取消

图 2-34 新建作业

然后，单击确定返回：

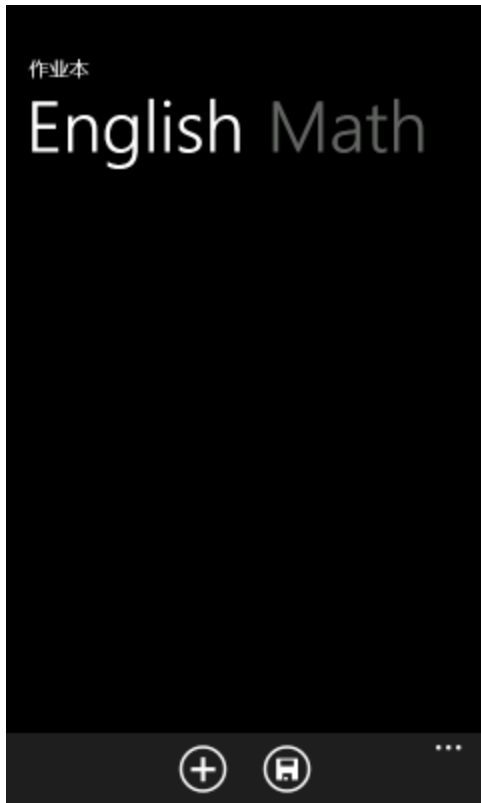


图 2-35 刚才新建的作业没有显示出来

Oh, My Lady Gaga! 我的作业呢??

## 2.8 插曲 #2

### 2.8.1 调查问题

究竟发生了什么事？是数据没有添加进去？是事件通知没有发出？还是出现线程安全的问题？我调试了一下，数据已经正确添加进去了，事件通知也正确发出去了，所有操作都在 UI 线程里执行，而且没有出现并发问题，那么问题到底出在哪里呢？

带着这个疑问，我从 [codeplex.com](http://codeplex.com) 上下载了 [SL for WP Toolkit 的最新代码 \(Change Set 57505\)](#)，然后调试进去看看。在调试的过程中，我发现每次从 `NewOrEditAssignmentPage` 页返回 `AssignmentBookPage` 页时，`LongListSelector` 控件都会调用 `Balance` 方法，但每次都会“跳过”本应执行的大部分代码，一开始我没怎么留意，觉得这个方法一下子就返回实在太神奇了，仔细观察，原来它是通过第一个 `if` 里的 `return` 悄悄返回的，如代码 2-43 所示。

```

private void Balance()
{
    if (!IsReady() || _flattenedItems.Count == 0)
    {
        // See comment in the call below. Necessary
        CollapseRecycledElements();
        _resolvedFirstIndex = _resolvedCount = _s
        return;
    }
}

```

代码 2-43 Balance 方法

难怪 LongListSelector 控件什么也没显示，因为 Balance 方法后面那些负责调整显示的代码一句都没执行。为什么会这样？关键在于 IsReady 方法，因为它每次都返回 false。当我单步进入 IsReady 方法时，发现 \_itemsPanel 和 ItemsSource 都不为 null，但 ActualHeight 的值却为 0.0，从而导致 IsReady 方法返回 false，如代码 2-44 所示。

```

private bool IsReady()
{
    return _itemsPanel != null && ItemsSource != null &&
        ActualHeight > 0;
}

```

代码 2-44 IsReady 方法

为什么会这样？这是因为，当我们打开 NewOrEditAssignmentPage 页时，由于 AssignmentBookPage 页暂时无需显示，Silverlight 会把它从主对象树移除，于是 ActualHeight 会被“清零”，当我们从 NewOrEditAssignmentPage 页返回时，Silverlight 需要重新测量每个控件的大小（包括页面本身），并安排它们的位置，ActualHeight 的值为 0.0 意味着 Silverlight 还没完成布局处理的工作，换句话说，LongListSelector 控件还没准备好，IsReady 方法返回 false 是正确的。

奇怪的是，每次我们从 NewOrEditAssignmentPage 页返回时，Balance 方法里的 IsReady 方法没有一次返回 true 的，这可能意味着 Balance 方法的调用时机不对，那什么时候调用才对呢？控件加载完毕的时候，即 Loaded 事件触发的时候，那么，LongListSelector 控件在 Loaded 事件触发的时候做了些啥呢？其实没什么，只是简单地把 \_isLoading 设为 true，然后调用 EnsureData 方法，如代码 2-45 所示。

```
void LongListSelector_Loaded(object sender, RoutedEventArgs e)
{
    _isLoading = true;

    EnsureData();
}
```

代码 2-45 Loaded 事件处理程序

这么看来，问题的关键就在于 EnsureData 方法有没有正确调用 Balance 方法了。我们来看看 EnsureData 方法的代码：

```
private void EnsureData()
{
    if (_flattenedItems == null || _flattenedItems.Count == 0)
    {
        FlattenData();
        Balance();
    }
}
```

代码 2-46 EnsureData 方法

FlattenData 和 Balance 是两个很重要的方法，前者负责从 ItemsSource 把数据初始化到 \_flattenedItems，而后者则负责确定哪些数据需要显示以及如何显示。显然，当我们从 NewOrEditAssignmentPage 页返回时，如果我们创建了作业，if 里面的语句是不可能执行的，因为 \_flattenedItems 里面包含了我们的作业！？这听起来很别扭，不是吗？毫无疑问，LongListSelector 控件没有考虑我们的情况，即打开一个另一个页面操作数据源，这是不应该的，你不可能指望我们把所有事情都放在同一个页面里处理吧？

### 2.8.2 修复问题

既然知道了原因，问题就不难解决了，把 LongListSelector 控件的 Loaded 事件处理程序改成下面这样：

```

void LongListSelector_Loaded(object sender, RoutedEventArgs e)
{
    _isLoading = true;

    //EnsureData();

    if (_flattenedItems == null || _flattenedItems.Count == 0)
    {
        FlattenData();
    }

    Balance();

    _isLoadingRaisedBefore = true;
}

private bool _isLoadingRaisedBefore;

```

代码 2-47 Loaded 事件处理程序

看到这里，你可能会问，\_isLoadingRaisedBefore 是干嘛的？我们知道，第一次进入 AssignmentBookPage 页和从 NewOrEditAssignmentPage 页返回时都会触发 Loaded 事件，这是两种需要区别处理的情况，因为 Balance 方法里包含了重设 \_resolvedFirstIndex 和 \_resolvedCount 的代码（参见代码 2-43），如果我们在后面那种情况下执行这行代码，LongListSelector 控件的显示就会乱掉，因为它计算不出正确的显示索引，\_isLoadingRaisedBefore 的存在就是为了防止这种情况的发生。

接着，在 Balance 方法里用 if 把重设 \_resolvedFirstIndex 和 \_resolvedCount 的那行代码包围起来，如代码 2-48 所示：

```

private void Balance()
{
    if (!IsReady() || _flattenedItems.Count == 0)
    {
        // See comment in the call below. Necessary here
        CollapseRecycledElements();

        if (!_isLoadedRaisedBefore)
        {
            _resolvedFirstIndex = _resolvedCount = _source.Count;
        }

        return;
    }
}

```

代码 2-48 修改后的 Balance 方法

值得提醒的是，每次调用 FlattenData 方法都会重设 \_flattenedItems，这对于从 NewOrEditAssignmentPage 页返回的情况来说是没有必要的，所以 Loaded 事件处理程序里的 FlattenData 方法需要放在 if 里，否则，使用 ObservableCollection 就会变得毫无意义了。

改好之后，编译一下。注意，如果你是通过 MSI 安装 SL for WP7 Toolkit 的话，你需要先在项目属性里修改一下版本再编译，否则待会重新添加引用的时候 Visual Studio 会自作聪明的引用原来那个 dll 文件，因为 MSI [在注册表里做了手脚](#)。

### 2.8.3 测试应用

一切准备就绪之后就可以按 F5 了。单击 Application Bar 上的新建按钮打开 NewOrEditAssignmentPage 页，如图 2-36 所示：

作业本

新建作业

截止日期

12/13/2010

作业内容

☐ 已完成

确定

取消

图 2-36 新建作业

输入作业内容，然后按确定返回：

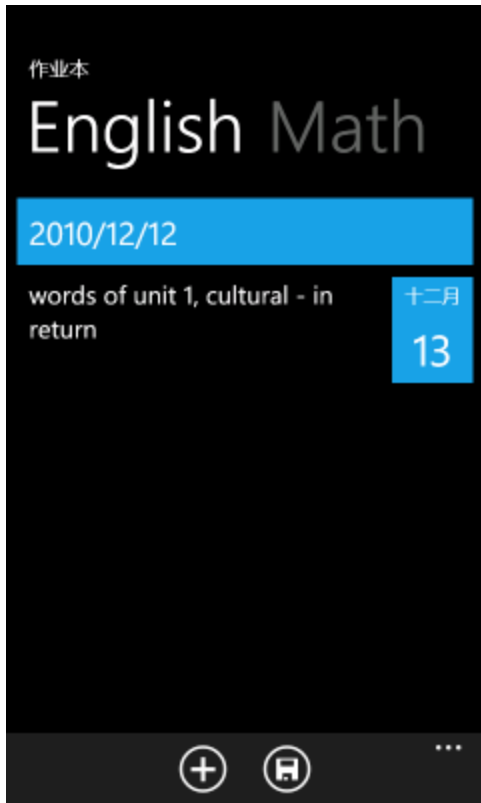


图 2-37 作业本可以显示刚才创建的作业了

噢，终于看到我的作业啦！

## 2.9 编辑作业本 · 续

### 2.9.1 实现编辑和删除操作

回到作业本的操作，接下来我们要实现编辑和删除两个操作。前面提到，我打算把它们放在上下文菜单里，那么，如何创建上下文菜单？非常简单，我们可以使用 SL for WP Toolkit 的 ContextMenu 控件，如代码 2-49 所示：



```

<DataTemplate x:Key="itemTemplate">
    <Grid Margin="0,0,0,12">
        <toolkit:ContextMenuService.ContextMenu>
            <toolkit:ContextMenu>
                <toolkit:MenuItem Header="编辑"
                                Click="EditMenuItem_Click"/>
                <toolkit:MenuItem Header="删除"
                                Click="DeleteMenuItem_Click"/>
            </toolkit:ContextMenu>
        </toolkit:ContextMenuService.ContextMenu>
    </Grid>
</DataTemplate>

```

代码 2-49 添加上下文菜单

正如你所看到的，ContextMenu 控件只需嵌入目标对象就能工作了，非常方便。

接下来的问题是如何实现它们的事件处理程序。我们知道，这两个操作有一个共同点，就是要获取用户当前选中的作业，怎么获取呢？有些同学可能会建议，在 AssignmentListViewModel 类里添加一个 SelectedAssignment 属性，并为它和 LongListSelector 控件的 SelectedItem 属性设置双向绑定，这样，一旦用户选中某项作业，我们就可以通过 SelectedAssignment 属性获取作业的 Id 了。你可以这样做，不过，这个做法会带来一个小小的问题，就是用户在长按某项作业之前得先单击一下。什么意思？我们知道，手机没有鼠标右击的概念，我们是通过长按（Touch and Hold）打开上下文菜单的，但从触摸手势的角度来看，长按和单击（Tap）是两个不同的触摸手势。LonglistSelector 控件只会在单击的时候设置 SelectedItem 属性，它不处理长按，所以当我们通过长按打开上下文菜单时，SelectedItem 属性可能为 null 或者之前选中的其它作业，前者会引发异常，而后者则会为用户带来困扰。为了避免这些问题，要么我们再次修改 LongListSelector 控件的代码，要么用户不得不执行一步额外的操作，显然，这都不是什么好办法，还有没有别的选择？

当然有！你知道吗，DataContext 属性是一个很特别的属性，子元素可以从父元素那里继承这个属性的值，对照代码 49 来看，这意味着 MenuItem 的 DataContext 和 Grid 的有着相同的值，而这个值正是我们苦苦寻找的作业！换句话说，只要我们获取到用户单击的 MenuItem 对象，就可以通过它的 DataContext 属性获取用户想要操作的作业。我们知道，事件处理程序的第一个参数就是引发该事件的对象，于是我们可以通过这个参数来访问 MenuItem 对象，如代码 2-50 所示：

```

private void EditMenuItem_Click(object sender, RoutedEventArgs e)
{
    var menuItem = (MenuItem)sender;
    var assignment =
        (Iridescent.Models.Assignment)menuItem.DataContext;
    NavigationService.Navigate(
        new Uri("/NewOrEditAssignmentPage.xaml?action=edit&id=" +
            assignment.Id.ToString(), UriKind.RelativeOrAbsolute));
}

private void DeleteMenuItem_Click(object sender, RoutedEventArgs e)
{
    var menuItem = (MenuItem)sender;
    var assignment =
        (Iridescent.Models.Assignment)menuItem.DataContext;
    App.AssignmentStore.Items.Remove(assignment);
}

```

代码 2-50 编辑和删除菜单项的 Click 事件处理程序

这样，我们既不需要在 AssignmentListViewModel 类里添加一个 SelectedAssignment 属性，也不需要修改 LongListSelector 控件的代码，更不需要委屈用户执行额外的操作，真是一举三得啊！

现在只剩一个操作了——撤销所有更改，我相信这对于你来说不是问题，所以我决定把它留给你当课后作业。

### 2.9.2 测试应用

好了，又到看效果的时候了！按 F5 运行应用程序，新建三项作业，如图 2-38 所示：

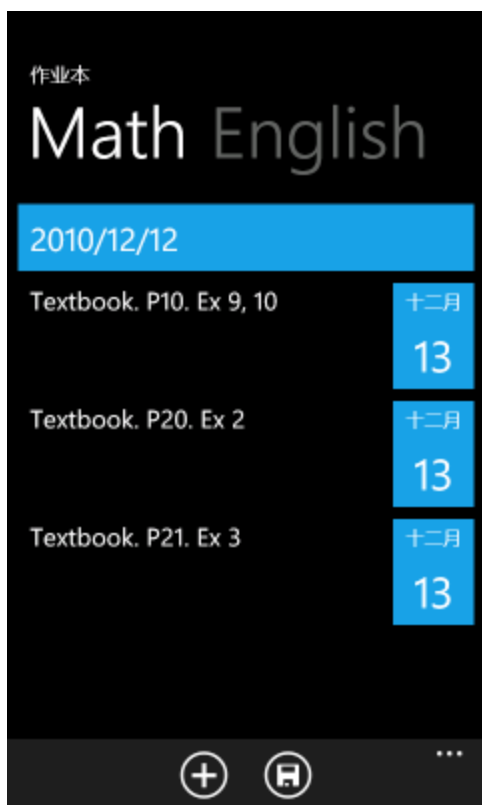


图 2-38 在作业本里新建三项作业

长按第三项作业，你会看到这项作业以外的所有东西都缩小了，给人一种向后移动的感觉，这个动画生动地突出了正在操作的作业以及上下文菜单，不过，不知道是不是动画的 bug，第三项作业的截止日期上面有个瑕疵（试了几次都是这样），如图 2-39 所示：



图 2-39 上下文菜单

单击编辑将会打开 NewOrEditAssignmentPage 页，修改一下截止日期，如图 2-40 所示：

作业本

编辑作业

截止日期

12/14/2010

作业内容

Textbook. P21. Ex 3

☐ 已完成

确定

取消

图 2-40 编辑作业

然后按确定返回，你会看到刚才修改的截止日期，如图 2-41 所示：

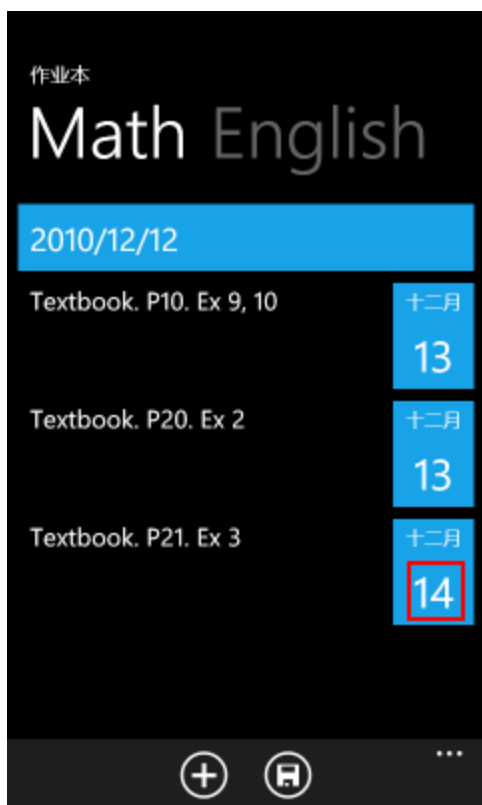


图 2-41 作业本正确显示刚才的修改

接着，长按第二项作业（Textbook. P20. Ex 2），并选择删除：

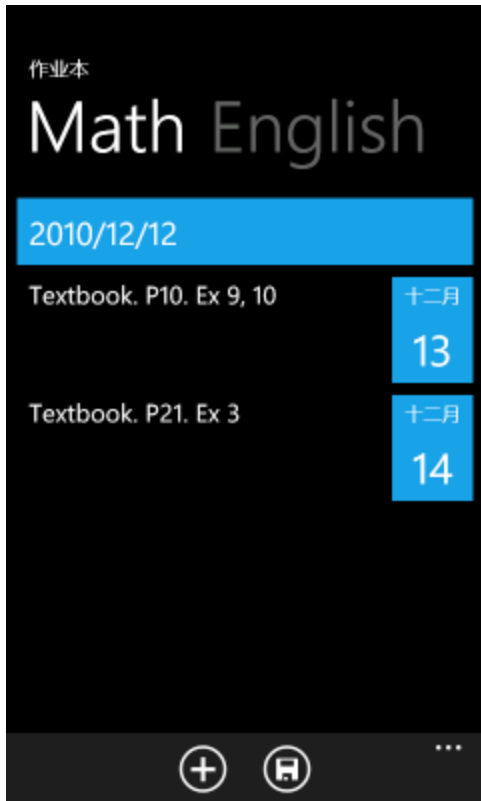


图 2-42

作业成功删除，如图 2-42 所示。

但是，如果你尝试删除（剩下的）第二项作业，你会发现它还在那里！为什么！？我调试了一下，发现此时 `MenuItem` 对象的 `DataContext` 属性的值居然是已故的前任第二项（`Textbook. P20. Ex 2`），而不是我们期望的现任第二项（`Textbook. P21. Ex 3`）！因为前任第二项已被删除，所以 `Remove` 方法不会触发 `CollectionChanged` 事件，`LongListSelector` 控件自然不会更新显示。如果你现在尝试删除第一项作业（既是前任也是现任），你会成功的，但是，在删除之后，如果你再次尝试删除剩下的唯一一项作业，你会发现此时 `MenuItem` 对象的 `DataContext` 属性的值变成刚故的前任第一项（`Textbook. P10. Ex 9, 10`）！从此以后，剩下的唯一一项作业就再也删除不了了，除非你返回 `MainPage` 页重新打开 `AssignmentBookPage` 页。由于编辑操作采用了相同的实现思路，如果一项作业删除不了，那么它也编辑不了。

究竟发生了什么事？是 `ContextMenu` 控件的 bug 吗？我另外创建了一个新的项目，在同等条件下，分别在 `ListBox` 和 `LongListSelector` 上测试了 `ContextMenu`，结果，`ListBox` 一方表现正常，而 `LongListSelector` 一方问题依旧，有趣的是，即使不用打开新的页面，结果还是一样。这让我不得不再一次怀疑是 `LongListSelector` 控件的问题。

我重新运行应用程序，然后单步执行第一次删除的整个过程。在这个过程中，我发现一个很奇怪的事情，当我删除第二项时，`LongListSelector` 控件先把第三项的

ContentPresenter 和 Assignment 分离开来，并把分离出来的 ContentPresenter 推入内部的 \_recycledItems（类型为 Stack<ContentPresenter>），接着对第二项做相同的事，然后把第二项从 \_flattenedItems 里删除，最后重新关联第三项的 ContentPresenter 和 Assignment，问题就出现在最后一步，它居然直接使用 \_recycledItems 顶部的 ContentPresenter，换句话说，它把第二项的 ContentPresenter 和第三项的 Assignment 关联了！见鬼！此时，如果我删除第一项的话，它会把第一项的 ContentPresenter 和第三项的 Assignment 关联！从这里不难看出，它应该在关联之前把 \_recycledItems 顶部那个垃圾扔掉！

既然知道了原因，问题就不难解决了，在 OnRemove 方法的相应地方加上红框里面那句：

```
int removeCount = resolvedLastIndex - startingIndex + 1;

while (removeCount-- > 0)
{
    RecycleLast();
}

_recycledItems.Pop();
```

代码 2-51 修改后的 OnRemove 方法代码

重新编译所有东西，然后运行应用程序，这次没问题了。

写完这篇文章之后，我的第一感觉是 LongListSelector 控件远未达到产品级别的质量，它的问题导致我无法专注于应用程序本身的功能设计和实现，如果你是本着学习和研究的态度去用它，那没问题，如果你想用它来做产品，那你就要做好心理准备了。不管怎样，这次我还是学到了不少东西。LongListSelector 控件的补丁我已经提交到 [codeplex.com](http://codeplex.com) 了，在官方发布修正版本之前，我只能使用自己修改的版本了→\_→



## 2.10 下课了……



## 第3课

# 创建笔记本

记笔记

设计用户体验

连接前端和后端

标签

命令与行为

笔记本

## 销售心理学 行

目标客户并不在乎你的产品是什么，他只在乎你的产品或服务能为他做什么。（P59）

导致人们购买或不买的两个主要动因：希望获益和害怕损失。（P60；案例：P61）

对于任何给定时刻产生的各种情感，其中最强烈的那一种将决定一个人在那一刻的决策和行动。（P67）

人们既想要更好的东西，同时又安于现状。（P75；“晚期接受者”定义：P75）

客户说想考虑一段时间再做决定背后的心理活动。（P68）



## 3.1 记笔记

### 3.1.1 调查需求

俗话说：好记性不如烂笔头。当然，这并不是说我们的脑子不好使，也不是叫我们不要用脑子记东西，而是提醒我们解放脑力，让大脑从事更有价值的思考。因此，这节课我们将会创建一个笔记本，用来记录课堂重点，但是，我们需要什么样的笔记本呢？我曾经在[《你的灯亮着吗？》](#)里读到这样一句话：如果某人能够解决这个问题，但是他本人却不会遇到这一问题时，那么你们首先要做的就是让他也感受到这个问题。最近公司来了一批校招招生，我找了个机会混进去听了一节入职前的技术培训，我想知道在课堂上把手机掏出来记笔记是一种什么样的感觉。

在课堂上，每当我想记点什么时，就会不自觉地拿起纸笔而不是手机，而且，用手机记笔记远没用纸笔来得随意自如。随后，我找了一些大学生和中学生，分别了解一下他们记笔记的情况，结果发现，他们记笔记的方式真是多种多样，有的直接记在书上，有的记在专门的笔记本上，有的记在练习册上，有的记在卷子上，有的甚至用手机把老师的板书直接拍下来……不难看出，他们的做法是怎么方便就怎么记，就目前而言，企图用一个手机应用来取代他们现有的做法显然是不现实的，也没必要，用户有权选择他们认为适合的做法，而我们的职责只是提供必要的帮助和支持。

那么，我们可以提供什么样的帮助和支持呢？想想看，现有的自由零散的做法会导致什么问题呢？最直接的影响是很难快速找到想要的内容，因为它们可能遍布各处，这种时候要是有个索引或者目录什么的就好了……Bingo！我们可以创建一个应用，帮助用户建立这个索引，虽然用户也可以另外找本小册子建立索引，但我们可以通过一个标签系统帮助用户快速找到相关的内容。这样，用户既可以保留现有的自由的记笔记习惯，又可以获得新的有序的管理效果。那么，用户应该在何时以及如何建立这个索引呢？

当然是越早越好！比如说，用户可以在每晚做完作业之后稍稍整理一下笔记，然后为它们创建一些条目并贴上标签。用户不必为所有笔记创建条目，可以挑选重要的来创建，这个过程本身就可以加深对知识的理解和巩固对知识的记忆。至于条目的内容，用户可以引用课本或者老师板书的原话，也可以用自己的话来概括复述，还可以直接引用课本或者练习册的页码和位置（段落、题号或者标记）等等，这个过程可以帮助用户熟悉如何根据条目的内容找到对应的笔记。

### 3.1.2 创建 Note 类

现在，用 Visual Studio 打开项目，在 Models 文件夹里创建一个 Note 类，并让它继承 NotificationObject 类，如代码 3-1 所示。

```
public class Note : NotificationObject
{
}

```

代码 3-1 Note 类

根据前面的讨论，Note 类应该包含以下四个属性：

属性名字	属性类型	属性描述
Id	Guid	唯一标识
Course	string	课程名称
Content	string	笔记内容
Tags	string	笔记标签

表 3-1 Note 类的属性

其中，Id 是只读属性，根据[上节课](#)的经验，我们需要给它创建以下两个构造函数：

```
public Note()
    : this(Guid.NewGuid())
{ }

public Note(Guid id)
{
    Id = id;
}

```

代码 3-2 Note 类的构造函数

它们分别用于新建和编辑两种情景。Course、Content 和 Tags 三个属性的 set 访问器均需调用 RaisePropertyChanged 方法，此方法是从 NotificationObject 类继承过来的。Tags 属性可以包含一个或多个标签，当 Tags 属性包含多个标签时，标签与标签之间将会使用逗号 (,) 进行分隔。

### 3.1.3 创建 NoteStore 静态属性

数据存储方面，我们将会直接使用上节课重构的结果——JsonDataStore 类。打开 App.xaml.cs 文件，在 App 类里创建一个 NoteStore 静态属性，如代码 3-3 所示：

```

private static IDataStore<Note> _noteStore = null;
public static IDataStore<Note> NoteStore
{
    get
    {
        if (_noteStore == null)
        {
            _noteStore = new JsonDataStore<Note>("Notes.json");
        }

        return _noteStore;
    }
}

```

代码 3-3 NoteStore 静态属性

接下来，是时候考虑一下用户体验了。

## 3.2 设计用户体验

### 3.2.1 创建笔记本页面

我们知道，每门课都有自己的笔记，就像每门课都有自己的作业一样，所以这里将会仿效作业本的做法，利用 Pivot 控件的特点，让每个 Pivot 项显示一门课程的笔记。现在，切换到 Expression Blend，创建一个 Windows Phone Pivot Page，并把它命名为 NoteBookPage.xaml，完了之后把 Pivot 控件的 Title 属性设为“笔记本”，把两个 Pivot 项的 Header 属性分别设为“销售心理学”和“行为金融学”，如图 3-1 所示：

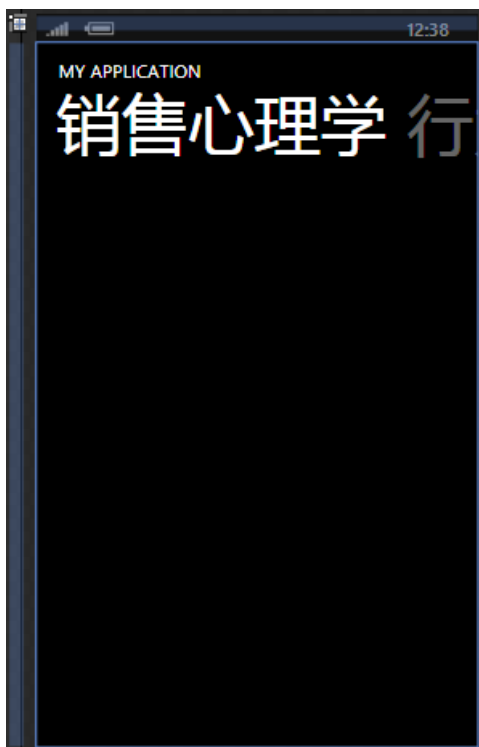


图 3-1 空的笔记本页面

那么，标题下面这么大的一块空位应该怎么安排呢？

首先，有两点我们是明确的，第一，我们需要列出笔记，第二，我们需要提供某种方式让用户切换标签。根据以往的经验，`ListBox` 最适合用来列出笔记，至于切换标签的方式，我觉得 `SL for WP Toolkit` 的 `ListPicker` 也是一个不错的选择。此时，我的脑子里浮现出第一个设计：



图 3-2 笔记本页面

然而，很遗憾，我对这个设计并未感到满意。第一，在手机屏幕这样的有限空间里，我们应该始终坚持把尽可能多的空间留给最重要的内容，ListPicker 作为一个辅助元素应该只在用户需要切换标签的时候才显示，否则把空间腾出来，这样可以显示更多笔记。第二，笔记本和作业本、课程表有着类似的布局设计，但 ListPicker 的存在破坏了布局设计的一致性，这点我实在无法容忍。那么，怎样才能让 ListPicker “呼之则来挥之则去”？

我们可以调整 ListBox 的大小，使之充满整个 Pivot 项，并把 ListPicker 藏在屏幕下方外面，然后在 Application Bar 上放置一个按钮，当用户单击这个按钮时，ListPicker 将会从屏幕下方外面向上平移，直至屏幕底部，如图 3-3 所示：



图 3-3 改进的标签控件

这样，用户可以通过 `ListPicker` 切换标签了。需要说明的是，`Application Bar` 上的按钮没有设置图标，这是因为我没有找到合适的，下次有时间我自己设计一个放上去。此时，我的脑子里冒出一个问题，为什么不直接以这种方式显示标签列表，而是大费周章地通过 `ListPicker` 打开一个新的页面呢？显然，我找不到合适的理由说服自己把 `ListPicker` 留下，同时，这个问题也给我带来了新的灵感。我们可以让显示标签的 `ListBox` 取代 `ListPicker`，当用户单击 `Application Bar` 上的按钮时，`ListBox` 将会从屏幕下方外面向上平移，直至覆盖显示笔记的 `ListBox` 为止，当用户选好标签之后，显示标签的 `ListBox` 将会向下平移，直至屏幕下方外面为止。

现在，把 `ListPicker` 删除，然后把一个 `ListBox` 添加到 `LayoutRoot`，并调整它的大小，使之充满整个屏幕（覆盖后面的 `Pivot` 控件）。此时，`Objects and Timeline` 面板上面的对象应该是这样的：



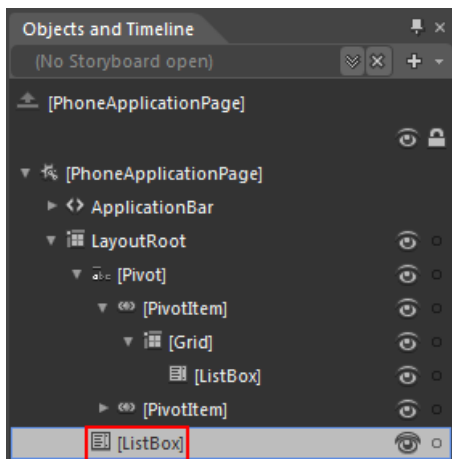


图 3-4 在 Objects and Timeline 面板上查看页面结构

接着，把它向下平移，直至它的顶部和 Pivot 控件的底部重叠为止，如图 3-5 所示：

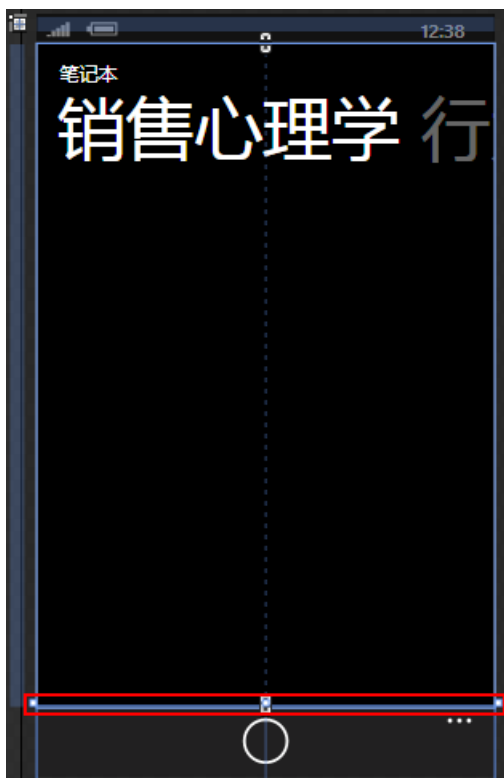


图 3-5 调整标签列表的位置

此时，它的 Height、VerticalAlignment 和 Margin 三个属性都会自动做出相应的调整，如图 3-6 所示：

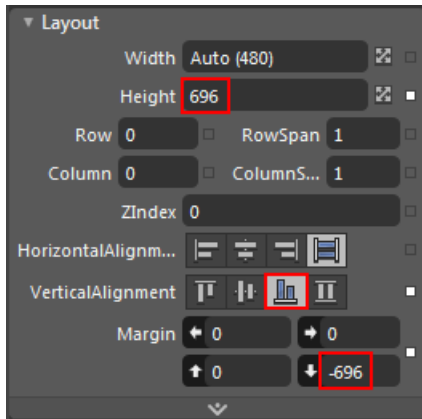


图 3-6 标签列表的属性

接下来，我们将会为它创建动画。

### 3.2.2 为标签列表创建动画

单击 Objects and Timeline 面板上的+按钮，如图 3-7 所示：

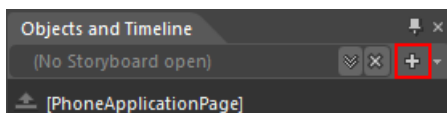


图 3-7 创建动画

在弹出的 Create Storyboard Resource 对话框里输入一个名字，如图 3-8 所示，然后按 OK 关闭对话框：

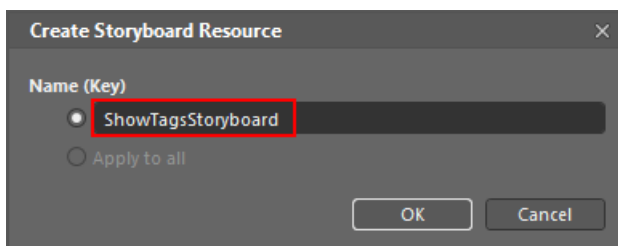


图 3-8 输入动画名字

此时，Objects and Timeline 面板将会变成这样：

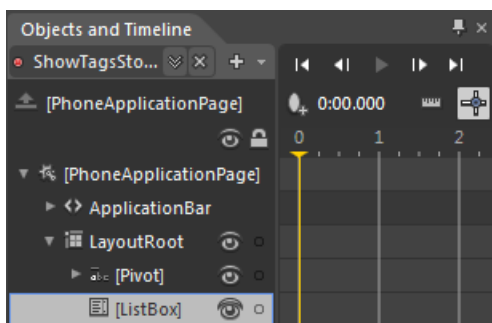


图 3-9 开启动画模式的 Objects and Timeline 面板

确保 ListBox 处于选中状态，把播放指针拖到 0.5 秒的为止，然后把 ListBox 向上平移，直至和 Pivot 控件完全重叠为止。此时，Objects and Timeline 面板将会变成这样：

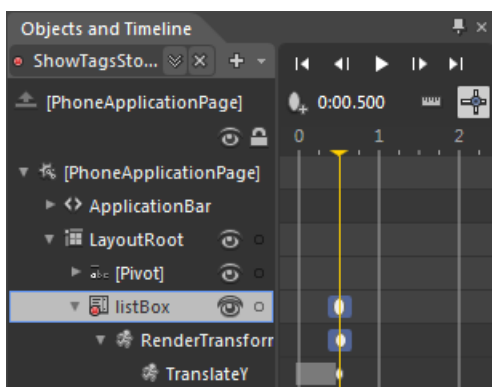


图 3-10 在 Objects and Timeline 面板上查看动画

看到这里，你可能会问，为什么只有结束时间的关键帧？你也可以手动设置开始时间的关键帧，如果没有设置，那么对象原本的状态将会默认为开始时间的关键帧。

当用户选好标签之后，我们需要把 ListBox 隐藏起来，不难想象，隐藏动画其实是显示动画的反转，那么，如何创建反转动画呢？非常简单，单击+按钮旁边的箭头，然后选择 Duplicate，如图 3-11 所示：

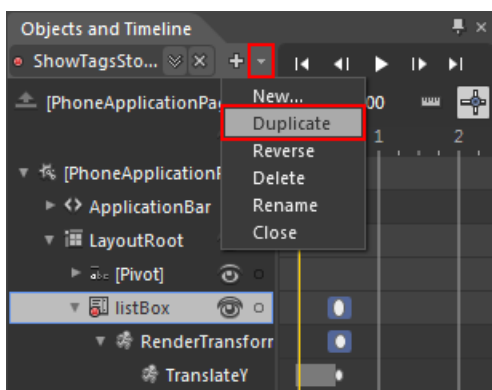


图 3-11 复制动画

此时，Expression Blend 会为你创建一个 ShowTagsStoryboard\_Copy1 动画。单击+按钮旁边的箭头，然后选择 Rename 把它重命名为 HideTagsStoryboard。接着，再次单击+按钮旁边的箭头，然后选择 Reverse 反转动画。好了之后单击+按钮旁边的 X 按钮关闭 Storyboard。

那么，我们如何触发这些动画？前面说过，当用户单击 Application Bar 上的按钮时，ListBox 将会显示，这个比较简单，只需在按钮的事件处理程序里调用 ShowTagsStoryboard 的 Begin 方法就可以了，如代码 3-4 所示：

```
private void ApplicationBarShowTagsIconButton_Click(  
    object sender, EventArgs e)  
{  
    ShowTagsStoryboard.Begin();  
}
```

代码 3-4 显示标签按钮的 Click 事件处理程序

而当用户选好标签之后，ListBox 将会隐藏，这个可以通过 Expression Blend 提供的 ControlStoryboardAction 来实现。打开 Assets 面板，选择 Behaviors，然后把 ControlStoryboardAction 拖到 Objects and Timeline 面板的 ListBox 上，如图 3-12 所示：

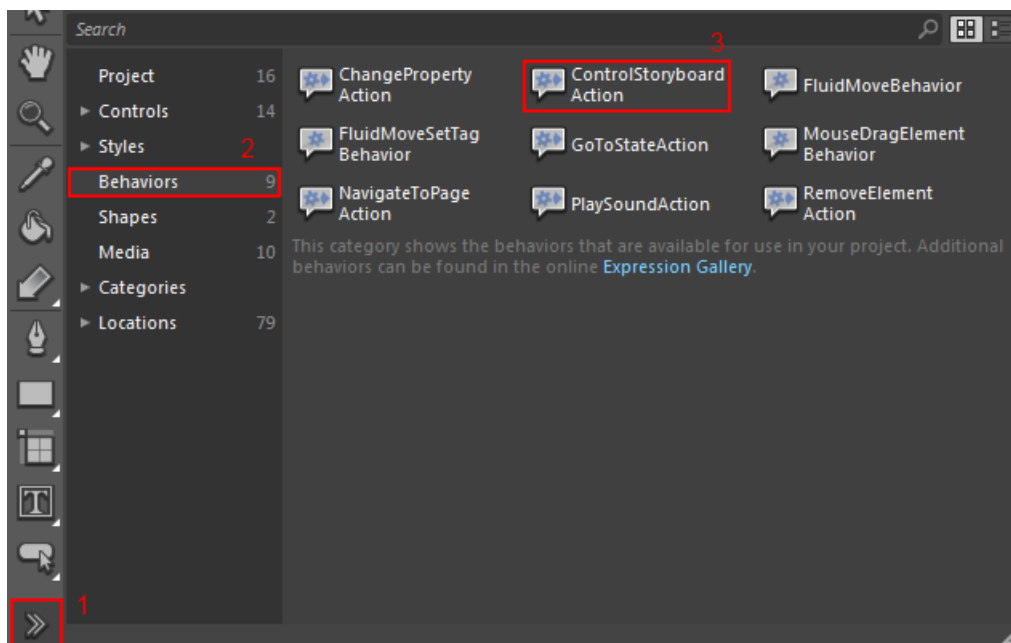


图 3-12 在 Assets 面板上找到 ControlStoryboard

此时，Objects and Timeline 面板将会变成这样：

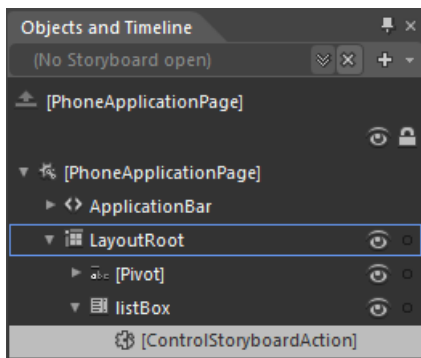


图 3-13 在 Objects and Timeline 面板查看页面结构

确保 ControlStoryboardAction 处于选中状态，在 Properties 面板上把 EventName 和 Storyboard 两个属性的值分别改为 MouseLeftButtonUp 和 HideTagsStoryboard，如图 3-14 所示：

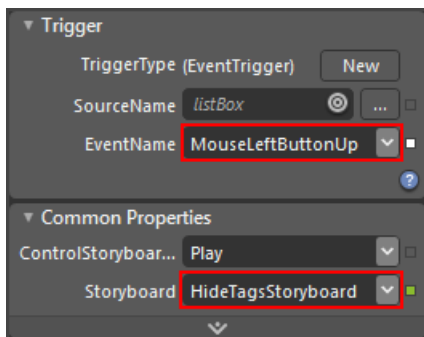


图 3-14 设置 ControlStoryboardAction

这样，当用户在 ListBox 里选好标签并松开手时就会触发 HideTagsStoryboard。

看到这里，你可能会问，为什么前面不直接在 Application Bar 的按钮上使用 ControlStoryboardAction？这是因为 Application Bar 并非 Silverlight 的一部分，你不可以把它和我们平时接触到的 Silverlight 对象等同起来。事实上，如果你试图把 ControlStoryboardAction 拖到 ApplicationBarIconButton 上，Expression Blend 会提示你 “Not a valid drop target”，如图 3-15 所示：

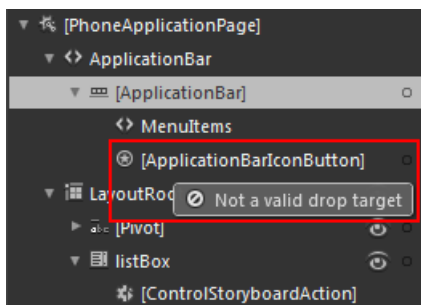


图 3-15 ControlStoryboardAction 无法应用到 Application Bar 按钮上

接下来，我们将会为两个 ListBox 定制数据模板。

### 3.2.3 添加示例数据

首先，通过 Data 面板导入以下两个 XML 文件：

```
<?xml version="1.0" encoding="utf-8" ?>
<Notes>
  <Note Content="目标">目标</Note>
  <Note Content="导致">导致</Note>
  <Note Content="对于">对于</Note>
  <Note Content="人们">人们</Note>
  <Note Content="客户">客户</Note>
</Notes>
```

代码 3-5 笔记示例数据

```
<?xml version="1.0" encoding="utf-8" ?>
<Tags>
  <Tag>购买行为</Tag>
  <Tag>案例</Tag>
  <Tag>概念定义</Tag>
  <Tag>问题</Tag>
  <Tag>练习</Tag>
</Tags>
```

代码 3-6 标签示例数据

此时，Data 面板将会变成这样：

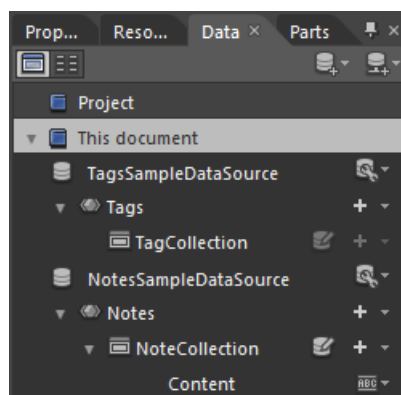


图 3-16 在 Data 面板上查看示例数据

把 Data 面板上的 NoteCollection 拖到显示笔记的 ListBox 上，然后进入列表项模板的编辑状态，把 StackPanel 的 Margin 属性、TextBlock 的 FontSize 属性和 TextBlock 的 TextWrapping 属性分别设为 PhoneTouchTargetOverhang、PhoneFontSizeNormal 和 Wrap。此时，你的笔记列表应该是这样的：

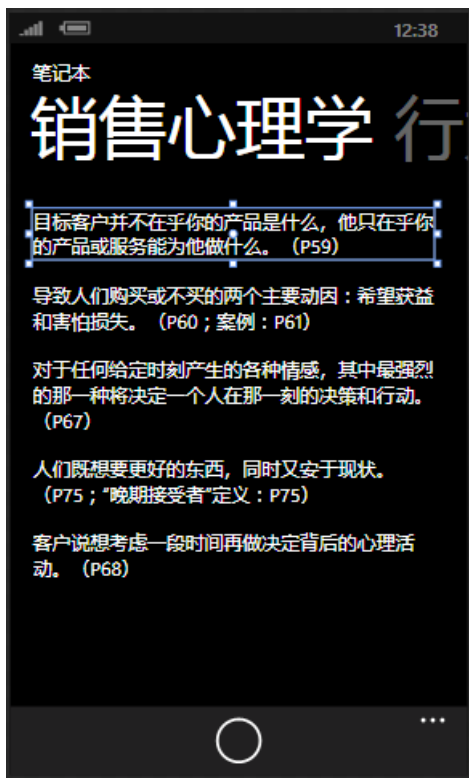


图 3-17 调整好模板之后的笔记列表

好了之后退出列表项模板的编辑状态。

接着，把 TagCollection 拖到显示标签的 ListBox 上，然后进入列表项模板的编辑状态，把 StackPanel 的 Margin 属性、TextBlock 的 FontSize 属性和 TextBlock 的 TextWrapping 属性分别设为 PhoneTouchTargetOverhang、PhoneFontSizeLarge 和 Wrap。此时，你的标签列表应该是这样的：

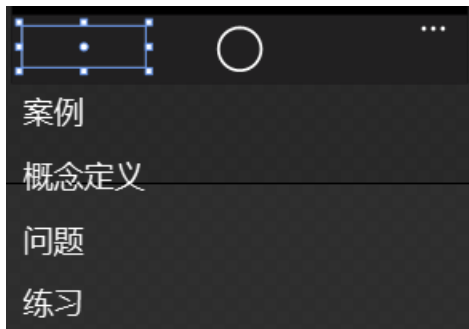


图 3-18 调整好模板之后的标签列表

接下来干嘛？你懂的！

### 3.2.4 测试应用

打开 MainPage.xaml, 添加一个菜单项, 并让它导航至 NotebookPage 页, 如图 3-19 所示:



图 3-19 在主菜单上添加笔记本菜单项

好了, 按 F5 吧:



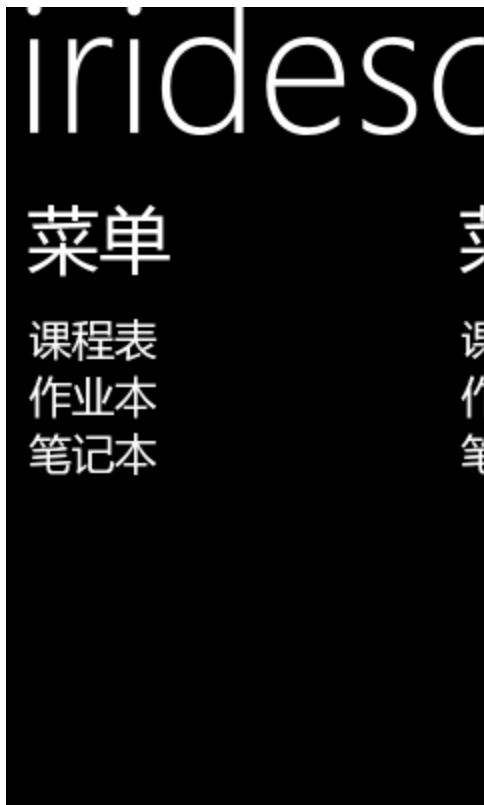


图 3-20 主菜单

单击笔记本：

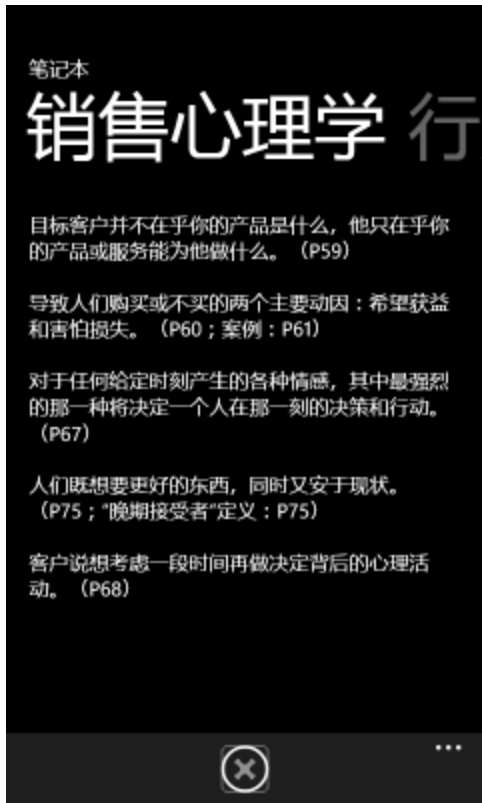


图 3-21 笔记本页面

单击 Application Bar 上的按钮：

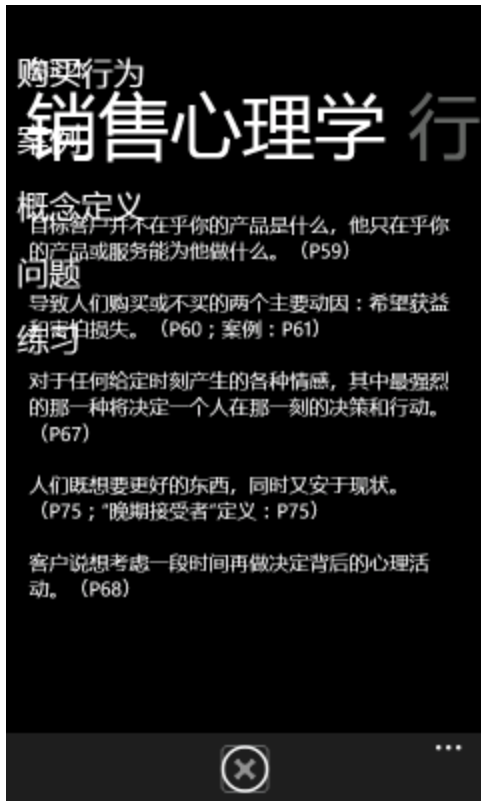


图 3-22 显示标签列表

啊！忘记把 ListBox 的 Background 设成不透明了！

### 3.2.5 改进标签列表的动画

退出应用程序。把 ListBox 的 Background 改为 PhoneBackgroundBrush。而动画方面，0.5 秒感觉有点长，我们可以把它改为 0.25 秒：

```
<Storyboard x:Name="ShowTagsStoryboard">  
    <DoubleAnimation Duration="0:0:0.25" To="-696"
```

#### 代码 3-7 修改动画时间

此外，还有一个地方值得改善的，我希望 ListBox 出来的时候能够有一种逐渐慢下来的感觉，而离开的时候则相反，逐渐快起来。这可以通过缓动函数（easing function）来实现。

单击 Objects and Timeline 面板上的 Open a Storyboard 按钮，然后选择 ShowTagsStoryboard，如图 3-23 所示：

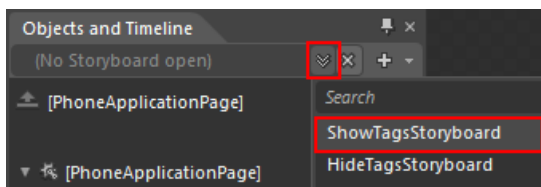


图 3-23 打开 ShowTagsStoryboard 动画

展开 ListBox 节点，确保下面的 RenderTransform 处于选中状态，如图 3-24 所示：

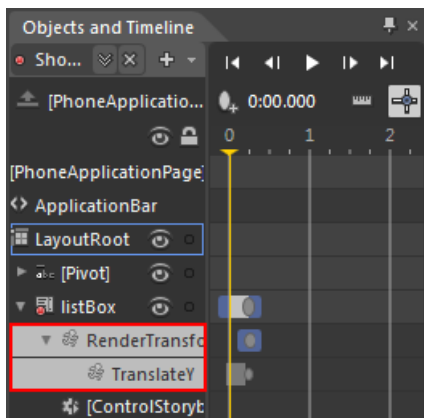


图 3-24 在 Objects and Timeline 面板上选中 RenderTransform

然后在 Properties 面板上把 EasingFunction 设为 Cubic Out，如图 3-25 所示：

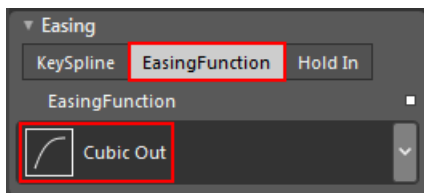


图 3-25 设置缓动函数

好了之后仿照上述步骤把 HideTagsStoryboard 的 EasingFunction 设为 Cubic In。现在，你可以按 F5 看看修改后的效果了。

### 3.2.6 设计笔记本的操作

接下来，是时候考虑一下笔记本的相关操作了。我们知道，课程表通常都是整个创建的，而作业通常也一次过把一门课当天要做的都记下来，对于这种集中式批量操作来说，提供保存所有更改和撤销所有更改是有必要的，但笔记本就不同了，里面的内容很可能分散在不同的时间点记录，没有太明显的集中式批量操作，如果我们遵循课程表和作业本的套路，要么用户不得不在每次记笔记时都额外执行一次保存操作，要么用户不得不承担最后可能忘记统一保存而丢失数据的风险。

因此，我打算把五项操作简化为新建、编辑和删除三项，并在用户执行每项操作之后自动保存数据，其中，新建将会以 `ApplicationBarIconButton` 的方式放在 `Application Bar` 上，如图 3-26 所示：

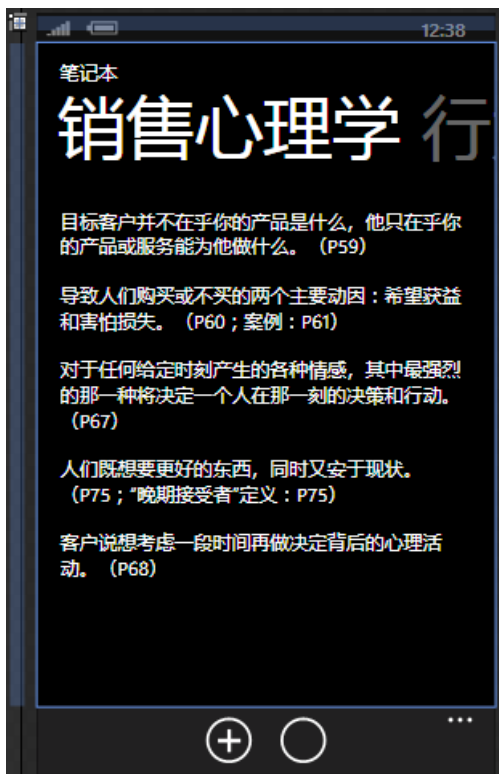


图 3-26 为笔记本添加 `Application Bar`

而编辑和删除则放在上下文菜单里，如代码 3-8 所示：

```
<DataTemplate x:Key="NoteTemplate">
    <StackPanel Margin="{StaticResource PhoneTouchTargetOverlap}">
        <TextBlock Text="{Binding Content}" FontSize="{Binding FontSize}">
            <toolkit:ContextMenuService.ContextMenu>
                <toolkit:ContextMenu>
                    <toolkit:MenuItem Header="编辑"/>
                    <toolkit:MenuItem Header="删除"/>
                </toolkit:ContextMenu>
            </toolkit:ContextMenuService.ContextMenu>
        </TextBlock>
    </StackPanel>
</DataTemplate>
```

代码 3-8 上下文菜单

根据之前的经验，我们需要一个新的页面来处理新建和编辑操作，如图 3-27 所示：

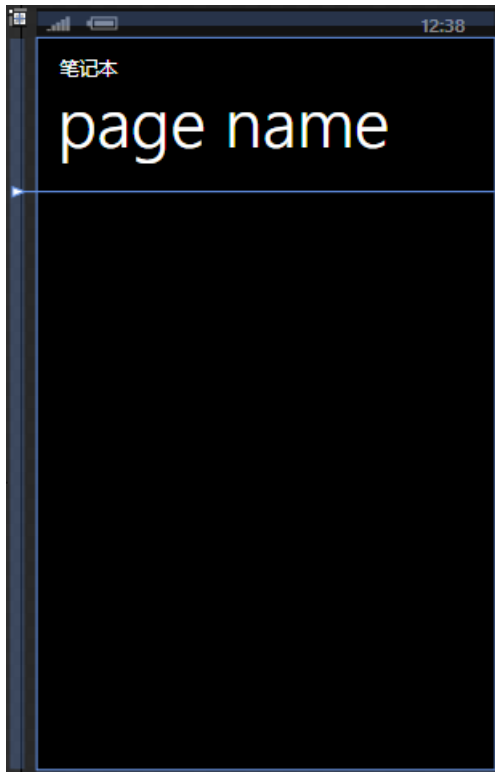


图 3-27 空的新建/编辑笔记页面

那么，我们应该如何设计这个页面呢？

想想看，Note 类的哪些属性和用户无关？Id。哪些属性无需劳烦用户处理？CourseName。那么剩下的 Content 和 Tags 两个属性就应该出现在这里了。毫无疑问，TextBox 完全能够胜任显示和编辑这两个属性的工作，如图 3-28 所示：



图 3-28 新建/编辑页面

值得注意的是，这里不再通过普通的 **Button** 控件来提供“确定”和“取消”两项功能，而是通过 **Application Bar** 上的按钮来提供，为什么呢？当用户输入完毕之后，软键盘可能处于开启状态，它会遮盖普通的 **Button** 控件，这意味着用户就不得不先单击一下页面上的空白处关闭软键盘，再单击 **Button** 控件关闭页面，而 **Application Bar** 不会被软键盘遮盖，这意味着用户可以在输入完毕之后直接单击上面的按钮关闭页面。你知道吗，这个小小的简化可以极大地提升用户体验，之前测试课程表和作业本的时候，这个不必要的步骤曾多次让我误触 **TimePicker/DatePicker** 控件的有效范围，导致新的页面被打开，我对此深感厌恶，你知道，当用户对软件的操作感到厌恶时，后果将会很严重！

### 3.2.7 测试应用

现在，回到 **NoteBookPage** 页，为 **Application Bar** 上的新建按钮添加一个事件处理程序，如代码 3-9 所示：

```
private void ApplicationBarNewIconButton_Click(  
    object sender, System.EventArgs e)  
{  
    NavigationService.Navigate(  
        new Uri("/NewOrEditNotePage.xaml", UriKind.RelativeOrAbsolute));  
}
```

代码 3-9 新建按钮的 Click 事件处理程序

然后按 F5:



图 3-29 运行结果

从上图可以看出，软键盘和 Application Bar 是并列一起的，所以我可以在任何时候单击 Application Bar 上的按钮。此外，当我在 TextBox 里输入较长的内容时，TextBox 还会自动调整自身的高度，以便完整显示我输入的内容。当我单击第二个 TextBox 进行输入时，整个页面将会稍稍向上平移，这样做的好处非常明显——避免软键盘遮盖 TextBox！从这里不难看出，WP7 在用户体验上的设计确实很体贴！

### 3.3 连接前端和后端

#### 3.3.1 创建笔记本的 ViewModel 类

接下来，我们要为这些用户界面创建对应的 ViewModel 类。我们知道，整个笔记本就是一个 Pivot 控件，而每门课程的笔记则对应一个 Pivot 项，这个结构和上节课的作业本类似，于是，我们可以仿效上节课的做法，分别创建 NotebookViewModel 和 NoteListViewModel 两个类，如代码 3-10 和 3-11 所示。



```
public class NotebookViewModel : NotificationObject
{
}

```

代码 3-10 NotebookViewModel 类

```
public class NoteListViewModel : NotificationObject
{
}

```

代码 3-11 NoteListViewModel 类

由于每个 Pivot 项都包含了一个标题和一组笔记，NoteListViewModel 类自然需要提供两个对应的属性，如代码 3-12 所示。

```
private string _header;
public string Header
{
    get { return _header; }
    set
    {
        _header = value;
        RaisePropertyChanged("Header");
    }
}

private CollectionViewSource _notes;
public ICollectionView Notes
{
    get { return _notes.View; }
}

```

代码 3-12 Header 和 Notes 属性

值得注意的是，这里不直接使用 ObservableCollection 集合，而是和第一节课的课程表一样，通过 CollectionViewSource 来提供集合视图，这样做的好处是我们只需指定过滤条件，剩下的事情 CollectionViewSource 会代为处理，而无需我们亲自出手，如代码 3-13 所示。

```

public NoteListViewModel(string courseName)
{
    _header = courseName;

    _notes = new CollectionViewSource();
    _notes.Source = App.NoteStore.Items;
    _notes.Filter += (o, e) => e.Accepted =
        e.Item != null && ((Note)e.Item).CourseName == courseName;
}

```

代码 3-13 初始化 \_header 和 \_notes 字段

而 NoteBookViewModel 类则和上节课的作业本一样，直接使用 ObservableCollection 集合，如代码 3-14 所示：

```

public ObservableCollection<NoteListViewModel> NoteLists
{
    get;
    private set;
}

```

代码 3-14 NoteLists 属性

并根据课程表里的数据创建对应的 NoteListViewModel 对象，如代码 3-15 所示：

```

public NoteBookViewModel()
{
    NoteLists = new ObservableCollection<NoteListViewModel>();

    App.CourseStore.Items
        .Select(c => c.Name)
        .Distinct()
        .Select(n => new NoteListViewModel(n))
        .ForEach(NoteLists.Add);
}

```

代码 3-15 初始化 NoteLists 属性

有了 ViewModel 类，我们就可以着手处理数据绑定了。

### 3.3.2 设置笔记本的数据绑定

现在，打开 NoteBookPage.xaml 文件，切换到 XAML 模式，在页面的资源字典里添加两个数据模板，如代码 3-16 所示：

```

<DataTemplate x:Key="pivotHeaderTemplate">
    <Grid>
        <TextBlock Text="{Binding Header}"/>
    </Grid>
</DataTemplate>
<DataTemplate x:Key="pivotItemTemplate">
    <Grid>
        <ListBox ItemTemplate="{StaticResource NoteTemplate}"
            ItemsSource="{Binding Notes}"/>
    </Grid>
</DataTemplate>

```

代码 3-16 Pivot 项的标题和内容模板

接着，把它们应用到 Pivot 控件上，如代码 3-17 所示：

```

<controls:Pivot
    Title="笔记本"
    HeaderTemplate="{StaticResource pivotHeaderTemplate}"
    ItemTemplate="{StaticResource pivotItemTemplate}"
    ItemsSource="{Binding NoteLists}"/>

```

代码 3-17 应用模板

然后，把 LayoutRoot 的 DataContext 属性去掉，然后在代码隐藏文件的构造函数里设置 DataContext 属性，如代码 3-18 所示：

```

public NoteBookPage()
{
    InitializeComponent();

    DataContext = new NoteBookViewModel();
}

```

代码 3-18 初始化 DataContext 属性

### 3.3.3 测试应用

好了，按 F5 吧。在打开笔记本之前，我们得先新建一些课程，如图 3-30 所示，否则 Pivot 控件不会创建任何 Pivot 项的。



图 3-30 课程表页面

好了之后就可以打开笔记本了：

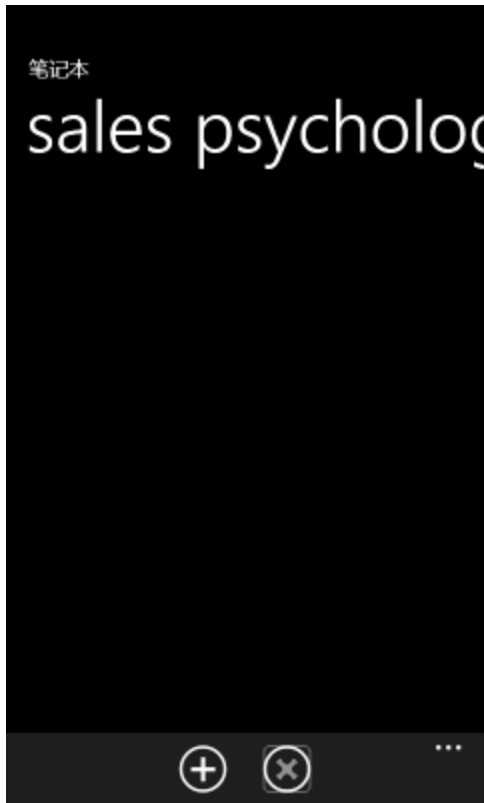


图 3-31 空的作业本页面

当然，现在的笔记本既没有笔记也不能创建笔记，因为这部分功能还没实现呢！

### 3.3.4 创建新建/编辑页面的 ViewModel 类

新建和编辑笔记的工作是由 `NewOrEditNotePage` 页负责的，根据上两节课的经验，我们需要创建 `NewOrEditNoteViewModel`、`NewNoteViewModel` 和 `EditNoteViewModel` 三个类，但我已经厌倦了每次都要手工创建这么多类，而且还有这么多重复的代码，所以这次我要对这部分进行重构。

在 `ViewModels` 文件夹里创建一个 `NewOrEditItemViewModel` 泛型类，并让它继承 `NotificationObject` 类，如代码 3-19 所示：

```
public class NewOrEditItemViewModel<T> : NotificationObject
{
}

```

代码 3-19 `NewOrEditItemViewModel` 类

仔细观察现有的 `NewOrEditCourseViewModel` 和 `NewOrEditAssignmentViewModel` 两个抽象类，不难发现，它们的有效成分是页面标题、`Model` 类的实例和提交数据的方法。页面标题很好处理，一个普通的类型为 `string` 的 `Title` 属性就可以了，如代码 3-20 所示：

```
private string _title;
public string Title
{
    get { return _title; }
    set
    {
        _title = value;
        RaisePropertyChanged("Title");
    }
}
```

代码 3-20 Title 属性

Model 类的实例有点棘手，因为我们有 3 个不同的 Model 类，怎么处理？我们有两个选择，一个是把属性的类型声明为 object，另一个就是这里采用的做法——泛型，如代码 3-21 所示：

```
private T _item;
public T Item
{
    get { return _item; }
    set
    {
        _item = value;
        RaisePropertyChanged("Item");
    }
}
```

代码 3-21 Item 属性

这也正是我把 NewOrEditItemViewModel 类声明为泛型类的缘由。

我们知道，每个 Model 类的数据都会提交到不同的地方，我们显然不能把这部分代码固化在 NewOrEditItemViewModel 类里，所以我决定通过委托把代码外包出去，如代码 3-22 所示：

```
private Action<T> _submit;
public void Submit()
{
    _submit(_item);
}
```

代码 3-22 Submit 方法

最后，我们需要在构造函数里初始化这三个成分，如代码 3-23 所示：

```
public NewOrEditItemViewModel(string title, T item, Action<T> submit)
{
    _title = title;
    _item = item;
    _submit = submit;
}
```

代码 3-23 NewOrEditItemViewModel 类的构造函数

那么，我们如何使用这个类呢？

打开 NewOrEditNotePage.xaml.cs 文件，重写 OnNavigatedTo 方法，如代码 3-24 所示：

```
protected override void OnNavigatedTo(NavigationEventArgs e)
{
    base.OnNavigatedTo(e);

    if (DataContext == null)
    {
        var action = NavigationContext.QueryString["action"];
        if (action == "new")
        {
        }
        else
        {
        }
    }
}
```

代码 3-24 重写 OnNavigatedTo 方法

这个是我们根据查询字符串初始化 DataContext 属性的基本套路。

当 action 的值是 new 时，初始化 DataContext 属性的代码应该是这样的：

```
DataContext = new NewOrEditItemViewModel<Note>(
    "新建笔记",
    new Note
    {
        CourseName = NavigationContext.QueryString["coursename"],
    },
    n =>
    {
        App.NoteStore.Items.Add(n);
        App.NoteStore.Commit();
    });
```

代码 3-25 新建笔记时的 DataContext 属性

而当 action 的值是 edit 时，初始化 DataContext 属性的代码应该是这样的：

```
var id = new Guid(NavigationContext.QueryString["id"]);
var originalNote = App.NoteStore.Items.First(n => n.Id == id);
DataContext = new NewOrEditItemViewModel<Note>(
    "编辑笔记",
    new Note(id)
    {
        CourseName = originalNote.CourseName,
        Content = originalNote.Content,
        Tags = originalNote.Tags,
    },
    n =>
    {
        originalNote.Content = n.Content;
        originalNote.Tags = n.Tags;
        App.NoteStore.Commit();
    });
```

代码 3-26 编辑笔记时的 DataContext 属性

这样，下次我们再有新的 Model 类就可以直接创建 ViewModel 类的实例了。

看到这里，你可能会问，代码 24 那个套路每次都一样的，应该可以处理一下吧？嗯，可以的。我们有两种处理方案，第一种方案是创建一个 `NewOrEditItemViewModelFactoryBase` 抽象类，并在里面使用模板方法模式（Template Method Pattern）处理那个套路，这样做的代价是我们需要为每个 Model 类创建一个对应的工厂类。如果你不喜欢这种做法，你可以选择第二种方案，创建一个 `NewOrEditItemPage` 类，按照代码 24 重写 `OnNavigatedTo` 方法，然后应用模板方法模式，这样做的代价是我们需要让每个新建/编辑页面继承这个类。无论我们选择哪种方案，有一点是可以肯定的，那就是 `NewOrEditItemViewModel` 类的三个成分似乎无法避免，因为



这些工作始终要做，而我们从一种方案改成另一种方案只不过是把这些工作从一个地方挪到另一个地方罢了。

如果你确实希望减轻这些工作，（理论上）也不是不可能，不过你得做好心理准备，因为你需要创建一个足够灵活的子系统，并且提供充足的元数据，这些数据包括每个 **Model** 类在不同状态下分别对应的页面标题、新建 **Model** 类的实例需要初始化哪些属性以及这些属性的数据来自哪里或者如何计算、克隆现有 **Model** 类的实例的方法是哪个、数据提交到哪里以及调用哪个方法、提交数据的是否需要同时保存到独立存储区等等等等。当我写到这里的时候，我已经隐约感觉得出这将是個很复杂的子系统，如果你真的打算实现这个子系统，那么请你先把右手抬起来，捂在左边胸口，然后问问自己：

- 1. 会有人愿意负责提供这些数据吗？如果有，会是谁呢？
- 2. 会有人愿意负责维护这个子系统吗？如果有，会是谁呢？
- 3. 当你的程序用上这个子系统之后，你能得到什么实质的好处？
- 4. 这些好处能否抵消提供数据和维护子系统的付出？

如果你没有自欺，你的心将会告诉你这个决定是否值得。

3.3.5 关联新建/编辑页面和对应的 **ViewModel** 类

创建好 **ViewModel** 类之后，我们就可以着手处理它和页面之间的关联了。首先是设置数据绑定，需要设置的控件以及对应的绑定表达式如下表所示：

描述	类型	属性	绑定表达式
页面标题	TextBlock	Text	{Binding Title}
笔记内容	TextBox	Text	{Binding Item.Content, Mode=TwoWay}
笔记标签	TextBox	Text	{Binding Item.Tags, Mode=TwoWay}

表 3-2 各个控件的绑定表达式

接着，为 **Application Bar** 上的两个按钮创建事件处理程序，如代码 3-27 所示：

```
private void ApplicationBarOKIconButton_Click(
    object sender, EventArgs e)
{
    ((NewOrEditItemViewModel<Note>)DataContext).Submit();
    NavigationService.GoBack();
}

private void ApplicationBarCancelIconButton_Click(
    object sender, EventArgs e)
{
    NavigationService.GoBack();
}
```

代码 3-27 确定/取消按钮的 **Click** 事件处理程序

页面的功能有了，可打开页面的功能还没好呢！

我们知道，NewOrEditNotePage 页的入口点有两个，而且都在 NoteBookPage 页上，一个是 Application Bar 上的新建按钮，另一个是上下文菜单里的编辑菜单项。当用户单击新建按钮时，我们需要告诉 NewOrEditNotePage 页当前的课程是什么，但是，我们从哪里获取这个信息？办法有很多种，其中一种是在 NoteBookViewModel 类里创建一个 SelectedNoteList 属性，如代码 3-28 所示：

```
private NoteListViewModel _selectedNoteList;
public NoteListViewModel SelectedNoteList
{
    get { return _selectedNoteList; }
    set
    {
        _selectedNoteList = value;
        RaisePropertyChanged("SelectedNoteList");
    }
}
```

代码 3-28 SelectedNoteList 属性

并把它绑到 Pivot 控件的 SelectedItem 属性，如代码 3-29 所示：

```
<controls:Pivot
    Title="笔记本"
    HeaderTemplate="{StaticResource pivotHeaderTemplate}"
    ItemTemplate="{StaticResource pivotItemTemplate}"
    ItemsSource="{Binding NoteLists}"
    SelectedItem="{Binding SelectedNoteList, Mode=TwoWay}"/>
```

代码 3-29 绑定 SelectedNoteList 属性

然后在新建按钮的事件处理程序里通过 SelectedNoteList 的 Header 属性获取课程名称，如代码 3-30 所示：

```

private void ApplicationBarNewIconButton_Click(
    object sender, System.EventArgs e)
{
    var book = (NoteBookViewModel)DataContext;
    if (book.NoteLists.Count > 0)
    {
        var courseName = book.SelectedNoteList.Header;
        NavigationService.Navigate(
            new Uri(
                "/NewOrEditNotePage.xaml?action=new&" +
                "courseName=" + courseName,
                UriKind.RelativeOrAbsolute));
    }
    else
    {
        MessageBox.Show("课程表还没创建。");
    }
}

```

代码 3-30 新建按钮的 Click 事件处理程序

当用户单击编辑菜单项时，我们需要告诉 NewOrEditNotePage 页笔记的 Id 是什么。我们知道，上下文菜单是嵌在列表项里的，这意味着它能从列表项那里继承 DataContext 属性的值，而这个值正是当前选中的笔记，所以我们可以从菜单项的 DataContext 属性获取笔记的 Id，如代码 3-31 所示：

```

private void EditMenuItem_Click(object sender, RoutedEventArgs e)
{
    var menuItem = (MenuItem)sender;
    var note = (Note)menuItem.DataContext;
    NavigationService.Navigate(
        new Uri("/NewOrEditNotePage.xaml?action=edit&id=" +
            note.Id.ToString(), UriKind.RelativeOrAbsolute));
}

```

代码 3-31 编辑菜单项的 Click 事件处理程序

类似地，删除操作也是通过相同的方式获取当前选中的笔记，然后把它删除，如代码 3-32 所示：

```
private void DeleteMenuItem_Click(object sender, RoutedEventArgs e)
{
    var menuItem = (MenuItem)sender;
    var note = (Note)menuItem.DataContext;
    App.NoteStore.Items.Remove(note);
    App.NoteStore.Commit();
}
```

代码 3-32 删除菜单项的 Click 事件处理程序

### 3.3.6 测试应用

好了，不知不觉又到看效果的时候了！打开笔记本：

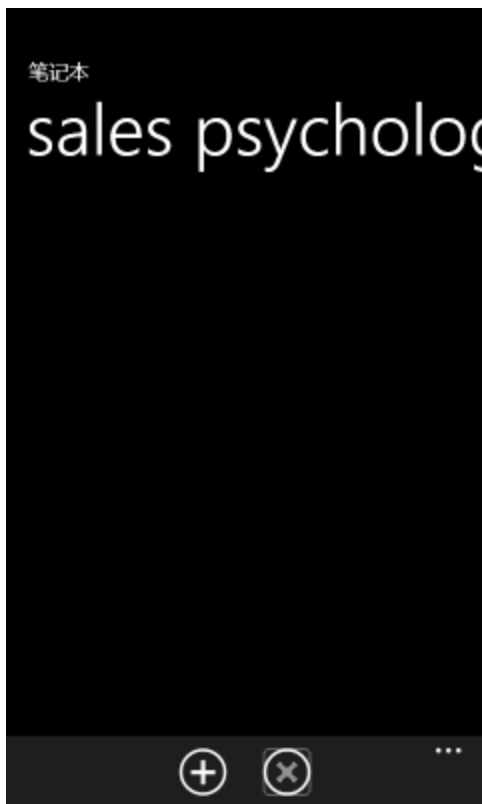


图 3-32 空的笔记本页面

单击新建按钮：



图 3-33 新建笔记

输入笔记内容和笔记标签，然后单击确定返回：



图 3-34 在笔记本上查看刚才新建的笔记

接着，长按笔记打开上下文菜单：



图 3-35 打开上下文菜单

选择编辑:

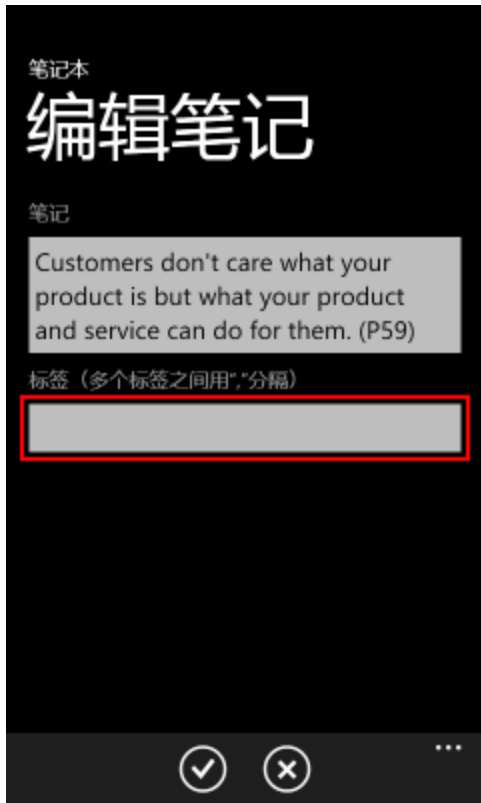


图 3-36 编辑笔记（没有标签）

嗯？我刚才没有输入标签？还是我现在眼花看错？为什么标签没有保存？现在，重新输入标签，单击页面上的空白处，单击确定返回，然后再次编辑笔记：



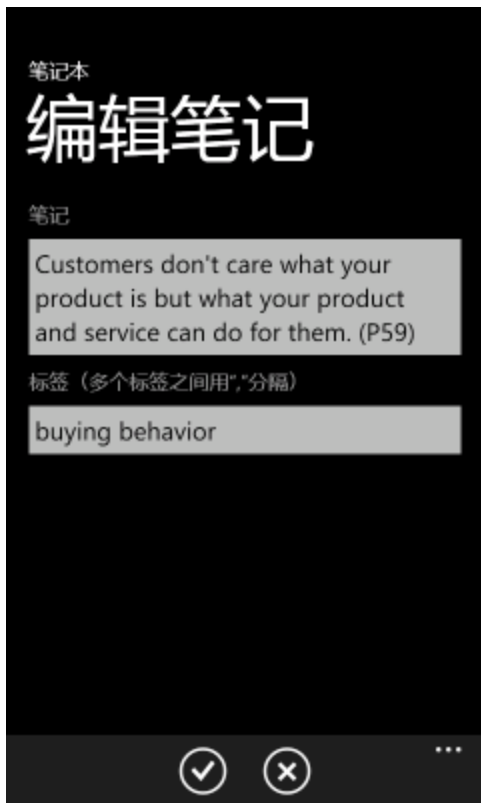


图 3-37 编辑笔记（有标签）

哈！这次有了！怎么回事？！

### 3.3.7 修复数据绑定没有提交的问题

这个时候，我的脑子里突然蹦出一个问题，当 `TextBox` 处于编辑状态时，单击确定按钮会不会触发 `TextBox` 的 `LostFocus` 事件？为了回答这个问题，我做了一个试验，结果发现，当 `TextBox` 处于编辑状态时，单击页面上的按钮或者空白处都能触发 `TextBox` 的 `LostFocus` 事件，而单击 `Application Bar` 上的按钮却不会，在这种情况下，`TextBox` 的内容不会提交！这个问题不难解决，我们只需在调用 `Submit` 方法之前让 `TextBox` 失去焦点就行了，要实现这个效果，最简单的办法是让页面获得焦点，如代码 3-33 所示：

```
private void ApplicationBarOKIconButton_Click(  
    object sender, EventArgs e)  
{  
    Focus();  
    ((NewOrEditItemViewModel<Note>)DataContext).Submit();  
    NavigationService.GoBack();  
}
```

代码 3-33 使页面获得焦点

这等效于单击页面上的空白处。不幸的是，这个办法只在新建笔记时有效，我不知道为什么，看来我们只能走别的路子了。

在 Silverlight 里，`TextBox` 的 `Text` 属性在两种情况下会更新绑定源，第一种情况我们刚才试过了，结果你也知道了；第二种是手动更新，这里的手动更新并不是指把数据从 `Text` 属性手动复制到 `Note` 对象的对应属性，而是告诉 `Text` 属性的绑定表达式把当前数据更新回绑定源。我们知道，当用户单击 `Application Bar` 上的确定按钮时，最多只有一个 `TextBox` 获得焦点，其它 `TextBox` 会因为失去焦点自动更新，因而没有必要对每个 `TextBox` 进行手动更新。那么，如何才能得到获得焦点的控件？`FocusManager` 类提供了一个 `GetFocusedElement` 静态方法，它返回获得焦点的控件，如果这个控件是 `TextBox`，我们就告诉 `Text` 属性的绑定表达式把当前数据更新回绑定源，如代码 3-34 所示：

```
private void ApplicationBarOKIconButton_Click(
    object sender, EventArgs e)
{
    var focusedTextBox =
        FocusManager.GetFocusedElement() as TextBox;
    if (focusedTextBox != null)
    {
        focusedTextBox
            .GetBindingExpression(TextBox.TextProperty)
            .UpdateSource();
    }

    ((NewOrEditItemViewModel<Note>)DataContext).Submit();
    NavigationService.GoBack();
}
```

代码 3-34 更新绑定源

重新运行应用程序，这次没问题了。虽然这个问题我们自己也可以解决，但我个人认为这是系统应该考虑到的问题，即使普通按钮和 `Application Bar` 上的按钮有着本质的区别，容许这种行为上的不一致会为开发者带来不便和困惑。

## 3.4 标签

### 3.4.1 根据用户选中的标签筛选笔记

到目前为止，我们的笔记本只不过是一个很普通的笔记本，虽然我们提供了与标签相关的用户界面，但这部分功能还没真正实现出来。那么，如何实现这部分功能？

我们知道，当用户单击 `Application Bar` 上的显示标签按钮时，显示标签的 `ListBox` 将会滑出来，里面列出当前课程的标签，用户可以从选择一个标签，此时，`ListBox` 将会滑出

去，笔记本的内容也将根据选中的标签进行筛选。从这里不难看出，我们需要在 `NoteListViewModel` 类里添加两个属性，一个用于存放当前课程的标签，另一个用于存放当前选中的标签，如代码 3-35 所示：

```
public ObservableCollection<string> Tags { get; private set; }

private string _selectedTag;
public string SelectedTag
{
    get { return _selectedTag; }
    set
    {
        _selectedTag = value;
        RaisePropertyChanged("SelectedTag");
    }
}
```

代码 3-35 Tags 和 SelectedTag 属性

需要说明的是，在使用 `Tags` 属性之前，我们需要在构造函数里初始化它，即创建一个空的集合对象。

接着，把它们分别绑到 `ListBox` 的 `ItemsSource` 和 `SelectedItem` 两个属性，如代码 3-36 所示：

```
ItemsSource="{Binding SelectedNoteList.Tags}" Background="{StaticR
SelectedItem="{Binding SelectedNoteList.SelectedTag, Mode=TwoWay}"
```

代码 3-36 设置数据绑定

那么，当用户选好标签之后，我们如何筛选笔记？还记得我们是如何根据课程名称筛选笔记的吗？我们直接把课程筛选条件告诉 `CollectionViewSource`，当数据源发生改变时，`CollectionViewSource` 将会自动筛选，因此，我们不妨考虑把标签筛选条件整合进去，让 `CollectionViewSource` 一并处理，如代码 3-37 所示：

```

_notes.Filter +=
    (o, e) =>
    {
        bool accepted = false;
        var note = e.Item as Note;
        if (e.Item != null && note.CourseName == courseName)
        {
            if (String.IsNullOrEmpty(SelectedTag))
            {
                accepted = true;
            }
            else
            {
                accepted = note.Tags.Contains(SelectedTag);
            }
        }
        e.Accepted = accepted;
    };

```

代码 3-37 筛选笔记

需要说明的是，当用户还没选择任何标签时，SelectedTag 属性的值为 null，此时，我们应该按照不做标签筛选的情况处理，否则看看笔记的标签是否包含用户选中的标签。

看到这里，你可能会问，当用户选择一个标签时，数据源并未发生任何改变啊，CollectionViewSource 应该不会自动筛选吧？这个问题问得好！事实上，它不会自动筛选，我们需要手动刷新一下它生成的视图，如代码 3-38 所示：

```

public string SelectedTag
{
    get { return _selectedTag; }
    set
    {
        _selectedTag = value;
        Notes.Refresh();
        RaisePropertyChanged("SelectedTag");
    }
}

```

代码 3-38 刷新笔记列表

那么，如何计算标签列表？

### 3.4.2 计算标签列表

想想看，什么时候需要计算标签列表？每次打开笔记本的时候肯定需要计算标签列表，但除此之外呢？当用户新建或编辑笔记时，可能引入新的标签；当用户编辑或删除笔记时，

可能删除现有标签，这些都会导致重新计算标签列表。既然计算标签列表的代码需要在这么多地方使用，我们当然应该把它提取到一个单独的方法里，如代码 3-39 所示：

```
public void ComputeTags()
{
    Tags.Clear();
    Tags.Add("(全部)");
    App.NoteStore.Items
        .Where(n => n.CourseName == Header &&
                !String.IsNullOrEmpty(n.Tags))
        .SelectMany(n => n.Tags.Split(',').Select(t => t.Trim()))
        .Where(t => !String.IsNullOrEmpty(t))
        .Distinct()
        .ForEach(Tags.Add);
}
```

代码 3-39 计算标签

计算标签列表的思路非常简单，首先，选取当前课程的笔记（忽略没有标签的），接着，从中提取标签，为了避免前/后空格的影响，这里使用 Trim 方法做了处理，然后，去掉空字符串以及重复的标签，最后，把标签添加到 Tags 属性。

现在，请思考一下，我们应该在哪调用 ComputeTags 方法？有些同学可能会说，这还不简单，分别在 NoteListViewModel 类的构造函数和三个操作的事件处理程序里调用不就行了？在 NoteListViewModel 类的构造函数和删除操作的事件处理程序里调用是没问题的，但在新建和编辑两个操作的事件处理程序里调用就有问题了，为什么呢？举个例子吧：

```
private void EditMenuItem_Click(object sender, RoutedEventArgs e)
{
    var menuItem = (MenuItem)sender;
    var note = (Note)menuItem.DataContext;
    NavigationService.Navigate(
        new Uri("/NewOrEditNotePage.xaml?action=edit&id=" +
                note.Id.ToString(), UriKind.RelativeOrAbsolute));
    ((NoteBookViewModel)DataContext).SelectedNoteList.ComputeTags();
}
```

代码 3-40 编辑菜单项的 Click 事件处理程序

你觉得上面代码的最后一行是在 NewOrEditNotePage 页显示之前还是之后执行呢？答案是之前，这意味着标签列表的计算在用户编辑笔记之前就完成了，这显然不是我们期望的效果。怎么办？

想想看，每次从 NewOrEditNotePage 页返回都会发生什么事呢？触发 NoteBookPage 页的 Loaded 事件！于是，我们可以把代码 3-39 里的最后一行放在 Loaded 事件处理程序

里，不过，这样做会导致一个问题，每次从主菜单打开 **NoteBookPage** 页时，当前课程的标签列表会被计算两次，第一次是在 **NoteListViewModel** 类的构造函数里，第二次是在 **Loaded** 事件处理程序里，因为 **NoteBookPage** 页每次显示都会触发 **Loaded** 事件。怎么处理这个问题？最简单的办法是通过一个 **bool** 变量区分 **NoteBookPage** 页是否已经打开过。不过，既然我们的目的只是为了在标签发生改变时做些事情，为什么不直接监听 **Note** 对象的 **PropertyChanged** 事件呢？要实现这个效果，有三个事儿需要我们做的：

1. 监听现有 **Note** 对象的 **PropertyChanged** 事件。
2. 监听 **App.NoteStore.Items** 的 **CollectionChanged** 事件，一旦有新的 **Note** 对象添加进来就监听它的 **PropertyChanged** 事件。
3. 监听 **App.NoteStore.Items** 的 **CollectionChanged** 事件，一旦现有的 **Note** 对象被删除就移除 **PropertyChanged** 事件的监听。

我们可以在 **PropertyChanged** 事件处理程序里调用 **ComputeTags** 方法。

虽然这种做法听起来有点复杂，但它避免了第一种做法的问题。本质上，这两种做法是等效的，只是一个在前台处理，另一个在后台处理，而正是这个角度的转变让我们对它们有了更进一步的了解。想想看，用户并非每次新建/编辑笔记之后都会单击 **Application Bar** 上的显示标签按钮，一个比较常见的使用情景用户把当天的笔记都输入了，然后通过标签的筛选来复习特定的内容，这样的话，在用户每次新建/编辑笔记之后重新计算标签列表显然没有必要。事实上，如果用户没有单击 **Application Bar** 上的显示标签按钮，我们根本没有必要计算标签列表，换句话说，我们可以把代码 3-39 里的最后一行放在显示标签按钮的 **Click** 事件处理程序里，从而实现按需计算，如代码 3-41 所示：

```
private void ApplicationBarShowTagsIconButton_Click(  
    object sender, EventArgs e)  
{  
    var book = (NoteBookViewModel)DataContext;  
    if (book.SelectedNoteList != null)  
    {  
        book.SelectedNoteList.ComputeTags();  
    }  
    ShowTagsStoryboard.Begin();  
}
```

代码 3-41

需要注意的是，当用户单击 **Application Bar** 上的显示标签按钮时，如果用户还没创建任何课程，直接调用 **ComputeTags** 方法将会引发异常，所以我们需要在调用之前判断一下。不过，这种做法也有个问题，试想一下，如果用户多次单击 **Application Bar** 上的显示标签按钮，其间没有新建/编辑任何笔记，那么，除了第一次计算标签内容是必要的，后面几次都是多余的。那么，如何才能避免多余的计算？看到这里，你可能会说，为什么不把后两种做法结合起来试一下呢，比如说，我们可以通过一个 **bool** 变量标识是否需要计算，

然后在 `PropertyChanged` 事件处理程序里把它的值设为 `true`，而在 `ComputeTags` 方法里，仅当这个变量的值为 `true` 时才执行计算，执行完毕之后把它的值设为 `false`。嗯，这个主意不错，我就把它留给你当课后作业吧。

### 3.4.3 测试应用

好了，不知不觉又到看效果的时候了！按 `F5` 运行应用程序，新建一条笔记，如图 3-38 所示：



图 3-38 笔记本页面

单击 `Application Bar` 上的显示标签按钮，如图 3-39 所示：

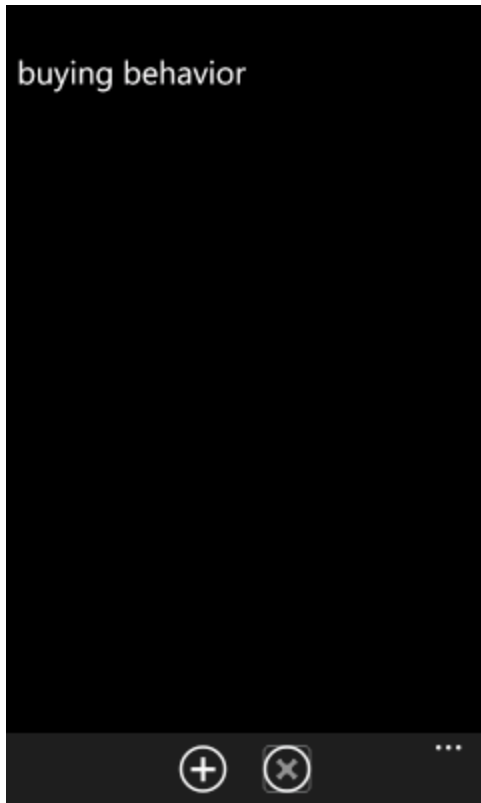


图 3-39 标签列表

单击页面空白处收回标签列表。再新建一条笔记，如图 3-40 所示：



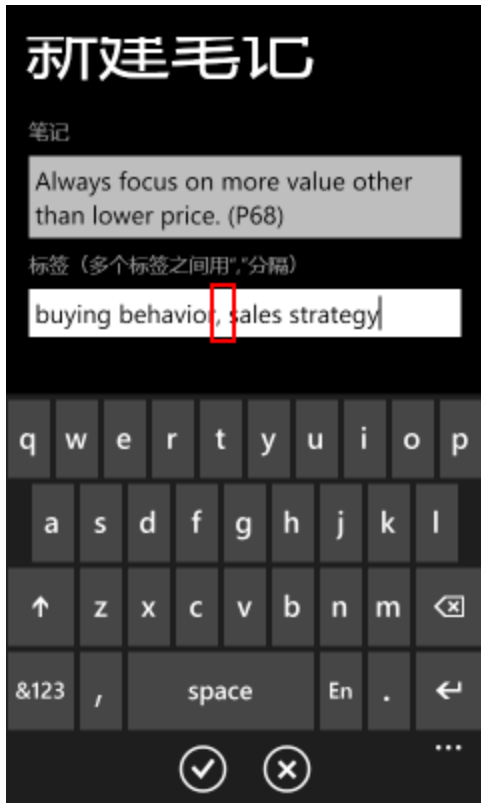


图 3-40

这次，我们给它两个标签，其中一个标签是现有的，另一个是新的，并且分隔符后面有个空格。

单击确定返回，然后单击 Application Bar 上的显示标签按钮，如图 3-41 所示：

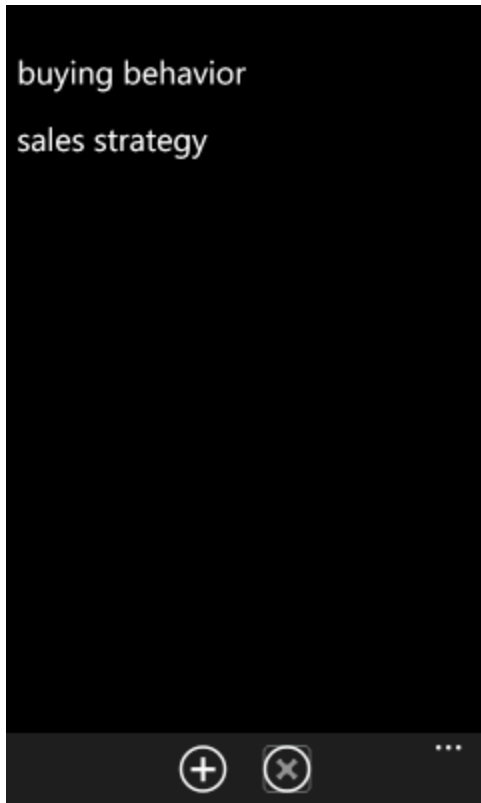


图 3-41 标签列表

再新建一条笔记，如图 3-42 所示：



图 3-42 笔记本页面

现在，单击 Application Bar 上的显示标签按钮，如图 3-43 所示：



图 3-43 标签列表

选择 buying behavior，如图 3-44 所示：



图 3-44 筛选后的笔记本

再次单击 Application Bar 上的显示标签按钮，选择 sales strategy，如图 3-45 所示：

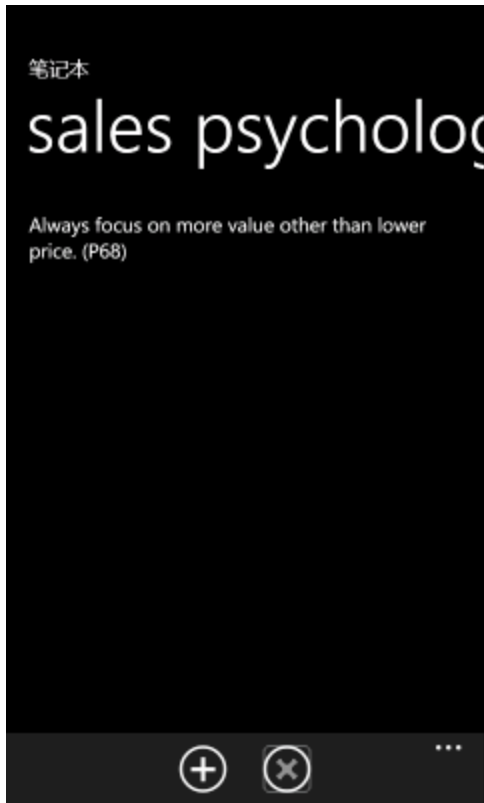


图 3-45 筛选后的笔记本

非常好！不过，现在有个问题，我想显示所有笔记怎么办？

#### 3.4.4 添加“全部”标签

没问题，我们可以在计算标签列表的时候加插一个“特殊”的标签，如代码 3-42 所示：

```
public void ComputeTags()
{
    Tags.Clear();
    Tags.Add("(全部)");
    App.NoteStore.Items
        .Where(n => n.CourseName == Header)
        .SelectMany(n => n.Tags.Split(',').Select(t => t.Trim()))
        .Distinct()
        .ForEach(Tags.Add);
}
```

代码 3-42 添加“全部”标签

并在筛选的时候进行“特殊”处理，如代码 3-43 所示：

```

if (String.IsNullOrEmpty(SelectedTag) || SelectedTag == "(全部)")
{
    accepted = true;
}
else
{
    accepted = note.Tags.Contains(SelectedTag);
}

```

代码 3-43 处理“全部”标签

### 3.4.5 测试应用

好了，重新运行应用程序，分别为两个课程新建一些笔记，如图 3-46 所示：

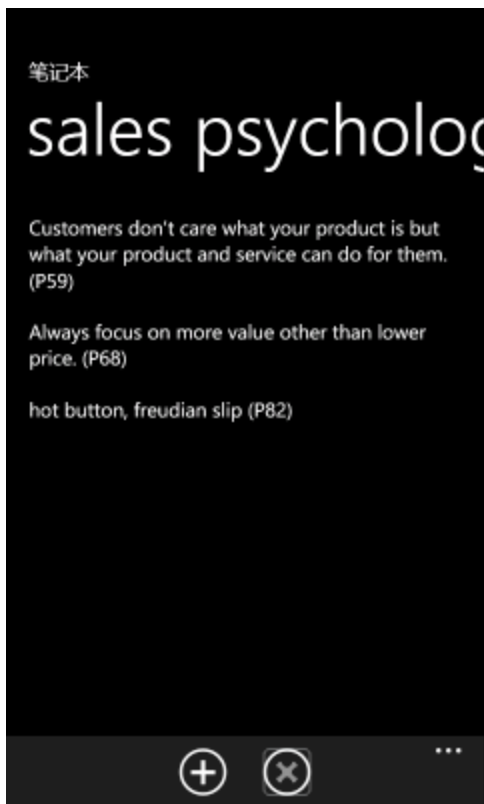


图 3-46 笔记本页面

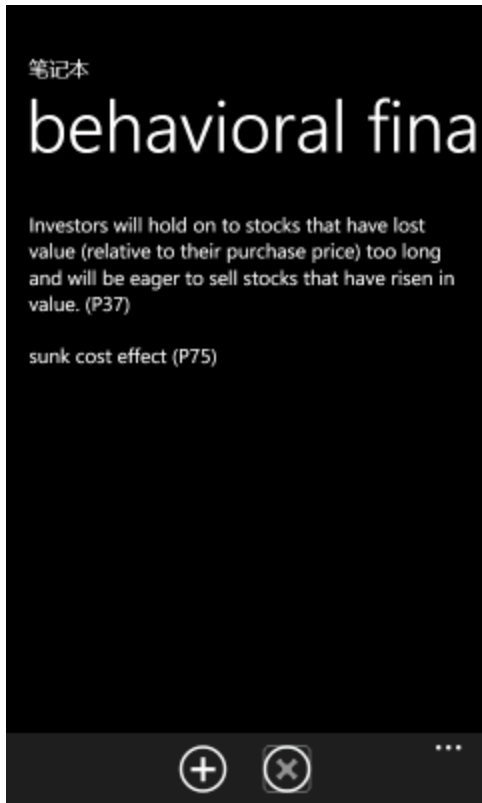


图 3-47 另一门课程的笔记

现在，单击 Application Bar 上的显示标签按钮，如图 3-48 所示：





图 3-48 标签列表

选择 disposition effect，如图 3-49 所示：

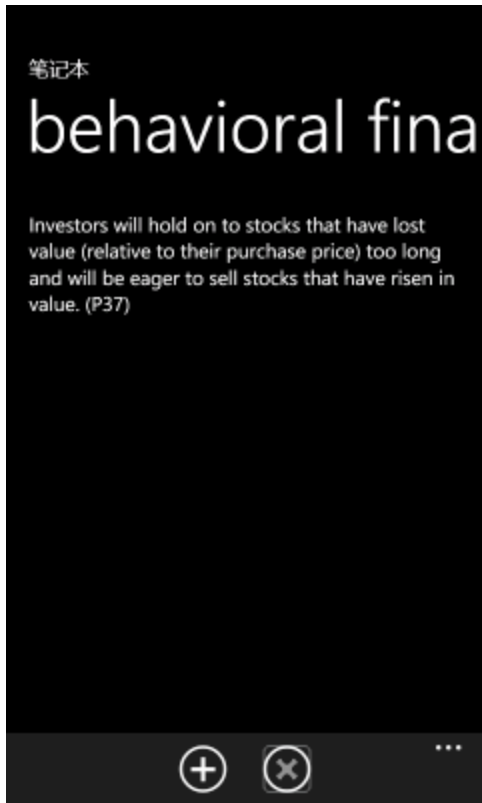


图 3-49 筛选后的笔记

现在，切换到 sales psychology 课程，单击 Application Bar 上的显示标签按钮，选择 sales strategy，如图 3-50 所示：

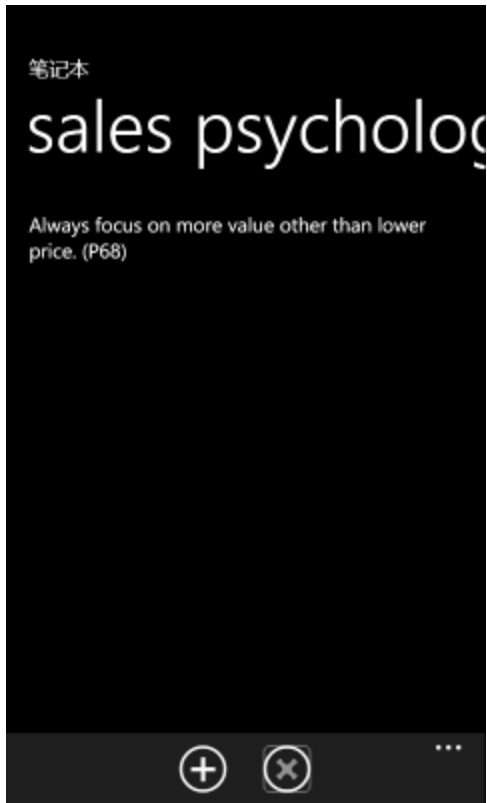


图 3-50 筛选后的笔记

再次单击 Application Bar 上的显示标签按钮，选择（全部），如图 3-51 所示：

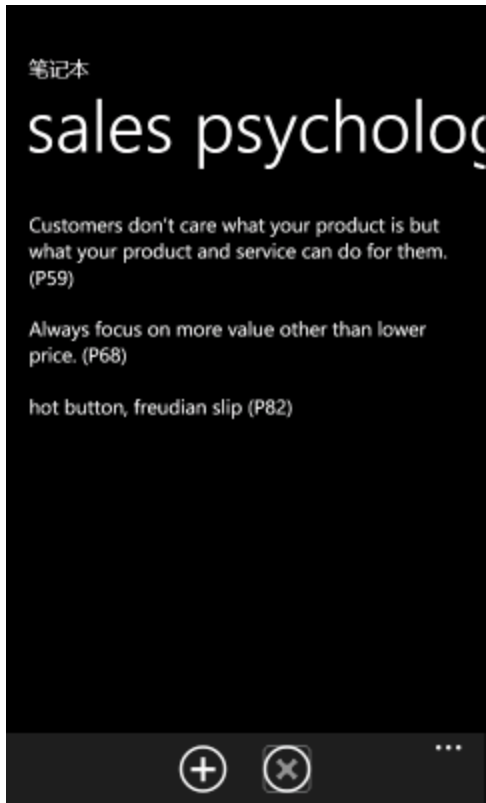


图 3-51 显示全部笔记

现在，切换回 behavioral finance 课程，如图 3-52 所示：

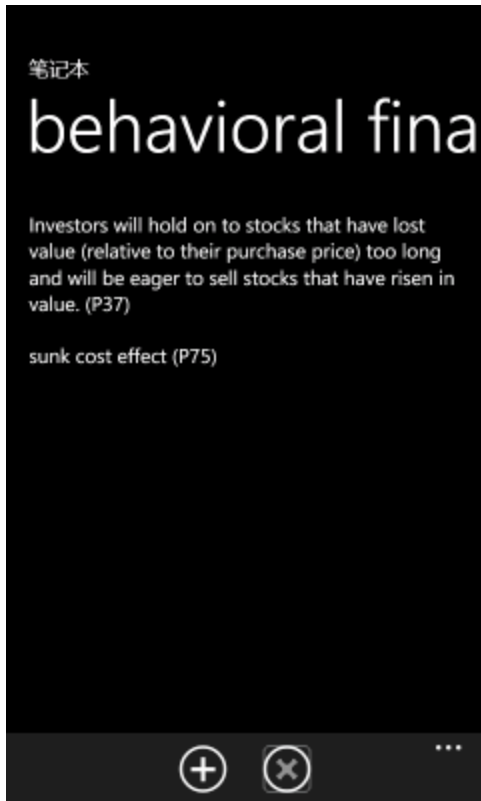


图 3-52 切换到另一门课程

怎么回事？！我们刚才已经做了筛选啊！

#### 3.4.6 修复切换课程时重设当前选中的列表项的问题

原来，当我们切换课程时，ListBox 的 `ItemsSource` 属性发生改变，导致 ListBox 的 `SelectedItem` 属性被重设为 `null`，而 `NoteListViewModel` 对象的 `SelectedTag` 属性和 ListBox 的 `SelectedItem` 属性是双向绑定的，因而也被重设为 `null` 了。ListBox 的 `SelectedItem` 属性被重设为 `null` 是对的，因为新的数据源不一定包含 `SelectedItems` 属性的值，但把 `NoteListViewModel` 对象的 `SelectedTag` 属性也重设为 `null` 就不对了，因为同一个 `NoteListViewModel` 对象的 `Tags` 属性肯定包含 `SelectedTags` 属性的值，因此，`SelectedTag` 属性的 `set` 访问器应该忽略这个重设，如代码 3-44 所示：

```

set
{
    if (_selectedTag != null && value == null)
    {
        return;
    }

    _selectedTag = value;
    Notes.Refresh();
    RaisePropertyChanged("SelectedTag");
}

```

代码 3-44 SelectedTag 属性忽略选中标签的重设

重新运行应用程序，重新执行一次上面的测试，嗯，这次没问题了。

## 3.5 命令与行为

### 3.5.1 创建命令对象

我们知道，WPF 和最新的 Silverlight 4 都支持命令绑定，比如说，Button 控件有一个 Command 属性和一个 CommandParameter 属性，前者用于绑定实现 ICommand 接口的对象，后者用于绑定传给前者的参数，但 SL for WP 却只有一个 ICommand 接口，这意味着我们无法为按钮设置命令，而 Application Bar 上的按钮这种异类就更不用说了。幸亏 [Prism](#) 为我们带来了 ApplicationBarButtonCommand（使用之前请先引用 Microsoft.Practices.Prism.Interactivity.dll 类库），它能让我们为 Application Bar 上的按钮设置命令。下面，我们拿 NewOrEditNotePage 页来示范它的用法。

在设置命令之前，我们得先有个命令，而命令通常是由 ViewModel 类提供的。打开 NewOrEditItemViewModel.cs，在 NewOrEditItemViewModel 类里添加一个 SubmitCommand 属性，如代码 3-45 所示：

```

public ICommand SubmitCommand { get; private set; }

```

代码 3-45 SubmitCommand 属性

那么，我们应该如何初始化它？一般的做法是创建一个 SubmitCommand 类，并让它实现 ICommand 接口，然后在 NewOrEditItemViewModel 类的构造函数里把 SubmitCommand 类的实例赋给 SubmitCommand 属性。如果你不嫌麻烦的话，你可以这样做，不过，由于创建命令对象的需求非常普遍，Prism 为我们带来了 DelegateCommand 泛型类，我们只需把提交数据的代码传给它的构造函数就可以了，如代码 3-46 所示：

```
SubmitCommand = new DelegateCommand<T>(submit);
```

代码 3-46 初始化 SubmitCommand 属性

接着，把 \_submit 私有字段以及在构造函数里初始化它的代码删除，因为我们不再需要它了。删除之后，把 Submit 方法改成这样：

```
public void Submit()
{
    SubmitCommand.Execute(Item);
}
```

代码 3-47 Submit 方法

换句话说，原来的 \_submit 私有字段被现在的 SubmitCommand 属性取代了。

### 3.5.2 设置确定按钮的命令

现在，打开 NewOrEditNotePage 页，从 Assets 面板上把 ApplicationBarButtonCommand 拖到 Objects and Timeline 面板的 PhoneApplicationPage 上，如图 3-53 所示：

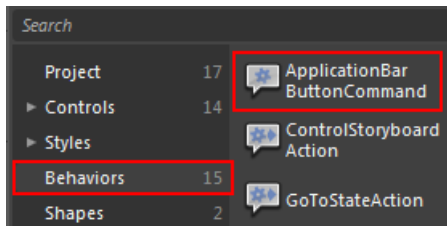


图 3-53 ApplicationBarButtonCommand

此时，Objects and Timeline 面板将会变成这样：

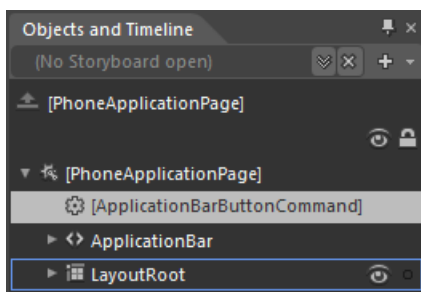


图 3-54 在 Objects and Timeline 面板查看页面结构

接着，在 Properties 面板上把 ButtonText 属性的值设为“确定”，如图 3-55 所示：

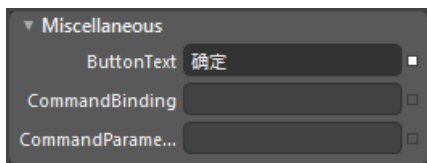


图 3-55 设置 ButtonText 属性

单击 CommandBinding 右边的小正方形，并选择 Custom Expression，如图 3-56 所示：

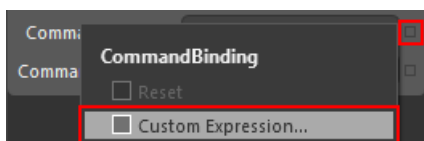


图 3-56 设置自定义表达式

在弹出的 Custom expression 对话框里输入 “{Binding SubmitCommand}” 并按回车，如图 3-57 所示：

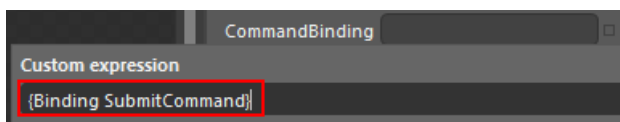


图 3-57 输入自定义表达式

用同样的办法把 CommandParameterBinding 设为 “{Binding Item}”。

那么，用户提交数据之后如何返回？这个时候就轮到 ApplicationBarButtonNavigation 上场了。从 Assets 面板上把 ApplicationBarButtonCommand 拖到 Objects and Timeline 面板的 PhoneApplicationPage 上，此时，Objects and Timeline 面板将会变成这样：

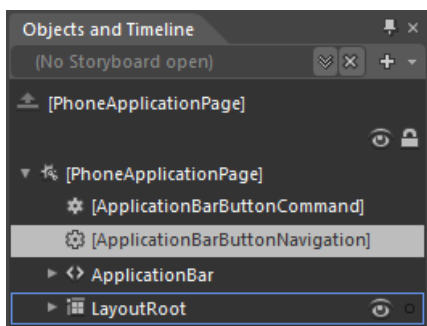


图 3-58 在 Objects and Timeline 面板查看页面结构

接着，在 Properties 面板上把 ButtonText 和 NavigateTo 两个属性的值分别设为 “确定” 和 “#GoBack”，如图 3-59 所示：



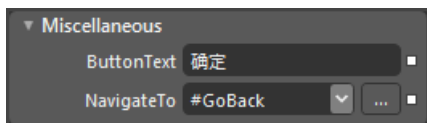


图 3-59 设置 ButtonText 和 NavigatedTo 属性

需要说明的是，“#GoBack”是一个硬性规定的特殊值，当我们把 NavigateTo 属性的值设为“#GoBack”时，ApplicationBarButtonNavigation 会调用 NavigationService.GoBack 方法返回，而当我们把 NavigateTo 属性设为 XXX.xaml 时，它会调用 NavigationService.Navigate 方法导航至对应的页面。

### 3.5.3 为更新 TextBox 控件的绑定源创建行为

那么，Text 属性更新绑定源的问题呢？难道 Prism 也提供了相应的组件？没有，这次我们得亲自出手了。右击 Utils 文件夹，然后选择 Add New Items，在弹出的 New Item 对话框里选择 Behavior，并把它命名为 AppBarButtonUpdateSource，如图 3-60 所示：

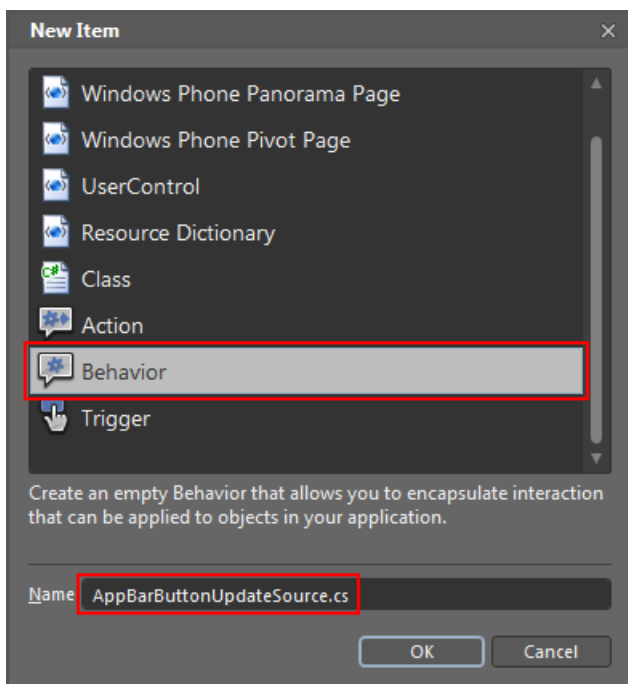


图 3-60 新建 Behavior

我们知道，Application Bar 上的按钮并非 Silverlight 的一部分，因此 Behavior 无法直接作用于它，而 Silverlight 里只有 PhoneApplicationPage 类提供了访问 Application Bar 的方法，因此我们需要把 AppBarButtonUpdateSource 的目标类型改为 PhoneApplicationPage，如代码 3-48 所示：

```
public class AppBarButtonUpdateSource :
    Behavior<PhoneApplicationPage>
{
```

代码 3-48 AppBarButtonUpdateSource 类

这也正是我们把 AppBarButtonCommand 和 AppBarButtonNavigation 拖到 PhoneApplicationPage 上的原因。

那么，如何才能找到 Application Bar 上的按钮？Prism 为我们带来了 FindButton 扩展方法，可以通过按钮的文字来查找，因此，我们需要创建一个 ButtonText 属性和一个 \_button 私有字段，前者用于指定待查找按钮的文字，后者用于保存找到的按钮，如代码 3-49 所示：

```
public string ButtonText { get; set; }

private ApplicationBarIconButton _button;
```

代码 3-49 与按钮相关的属性/字段

FindButton 扩展方法需要一个实现 IApplicationBar 接口的对象，而能够提供这个对象的只有 PhoneApplicationPage 对象，后者可以通过 Behavior 的 AssociatedObject 属性访问。于是，我们可以在 OnAttached 方法里初始化 \_button 私有字段，如代码 3-50 所示：

```
protected override void OnAttached()
{
    base.OnAttached();

    // Insert code that you would want run when the Behavior is attached

    _button = AssociatedObject.ApplicationBar.FindButton(ButtonText);
    if (_button != null)
    {
        _button.Click += ButtonClick;
    }
}
```

代码 3-50 重写 OnAttached 方法

找到按钮之后，我们需要把更新绑定源的代码关联到按钮上，而做到这点的唯一办法就是为按钮创建一个 Click 事件处理程序，如代码 3-51 所示：

```
private void ButtonClick(object sender, EventArgs e)
{
    var focusedTextBox = FocusManager.GetFocusedElement() as TextBox;
    if (focusedTextBox != null)
    {
        focusedTextBox
            .GetBindingExpression(TextBox.TextProperty)
            .UpdateSource();
    }
}
```

代码 3-51 Click 事件处理程序

最后，在 OnDetaching 方法里解除它们之间的关联，如代码 3-52 所示：

```
protected override void OnDetaching()
{
    base.OnDetaching();

    // Insert code that you would want to run here

    if (_button != null)
    {
        _button.Click -= ButtonClick;
    }
}
```

代码 3-52 重写 OnDetaching 方法

现在，重新编译项目，然后从 Assets 面板上把 AppBarButtonUpdateSource 拖到 Objects and Timeline 面板的 PhoneApplicationPage 上，并把 ButtonText 属性的值设为“确定”。此时，Objects and Timeline 面板将会变成这样，如图 3-61 所示：

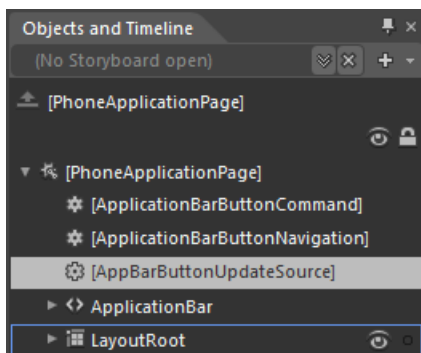


图 3-61 从 Objects and Timeline 面板查看页面结构

不过，这个顺序是不对的，`AppBarButtonUpdateSource` 应该排在其它两个的前面，但这个顺序在 Expression Blend 里无法调整，因此我们需要手动修改 XAML。

### 3.5.4 设置取消按钮的命令

至于取消按钮，由于它只是简单地返回，我们只需为它添置一个 `AppBarButtonNavigation` 就行了。添置好后，Objects and Timeline 面板将会变成这样：

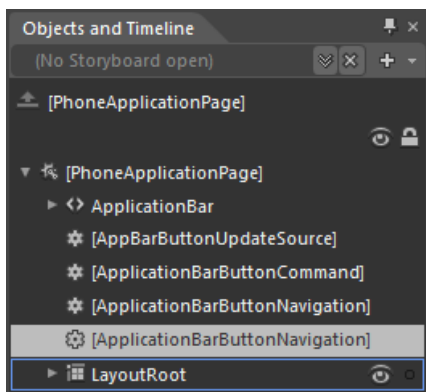


图 3-62 从 Objects and Timeline 面板查看页面结构

现在，我们可以把这两个按钮的 Click 事件处理程序去掉了。

命令绑定是 MVVM 模式的重要组成部分，它不但可以进一步降低 View 和 ViewModel 之间的耦合度，还可以简化单元测试的工作。目前我们通过 Behavior 来实现命令绑定只是权宜之计，希望微软可以在将来的版本里把这部分功能补完了。还有的就是希望微软能够进一步完善 Application Bar，包括处理焦点问题以及提供更丰富的访问方式。

### 3.6 下课了……



## 第4课

# 把课程、做也和笔记 整合到一块

化零为整

连接前端和后端

打开 CourseHubPage 页

动起来



## 4.1 化零为整

### 4.1.1 创建 Course Hub 页面

前面三节课我们分别实现了[课程表](#)、[作业本](#)和[笔记本](#)三个主要功能，然而，它们的内容分散在三个不同的页面，试想一下，如果我想查看某门课下一次什么时候上课、今天有哪些作业要做以及今天记了哪些笔记，我就不得不在多个不同的页面之间来回切换了，这显然降低了应用程序的可用性（usability）！因此，这节课的任务是化零为整，把相关内容整合起来。

在 WP7 里，内容的整合一般是通过“Hub”来实现的，比如说，People Hub、Pictures Hub、Music + Video Hub、Office Hub、Games Hub 以及 Marketplace Hub 等都是典型的代表。那么，如何创建这样的页面？非常简单！右击 Projects 面板里的项目节点，选择 Add New Item，如图 4-1 所示：

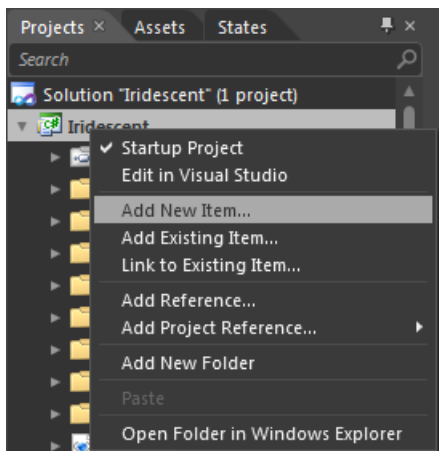


图 4-1 新建页面

在弹出的 New Item 对话框里选择 Windows Phone Panorama Page，输入页面的名字，如图 4-2 所示，然后按 OK：

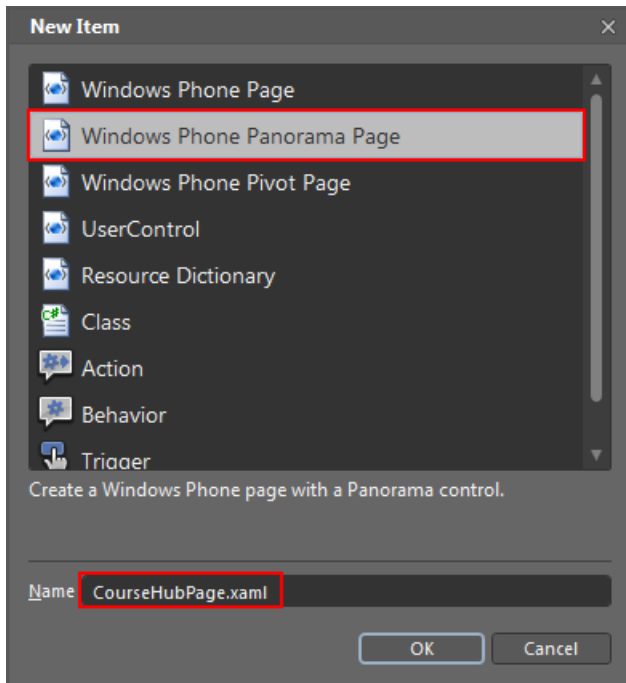


图 4-2 输入页面名字

此时，Expression Blend 会为你创建一个 Panorama 页，里面包含了两个 Panorama 项。在 Properties 面板上把 Panorama 控件的 Title 属性设为“组织行为学”，接着，添加一个 Panorama 项，然后，把页面上的三个 Panorama 项的 Header 属性分别设为“课程概况”、“今天笔记”和“今天作业”，如图 4-3 所示：



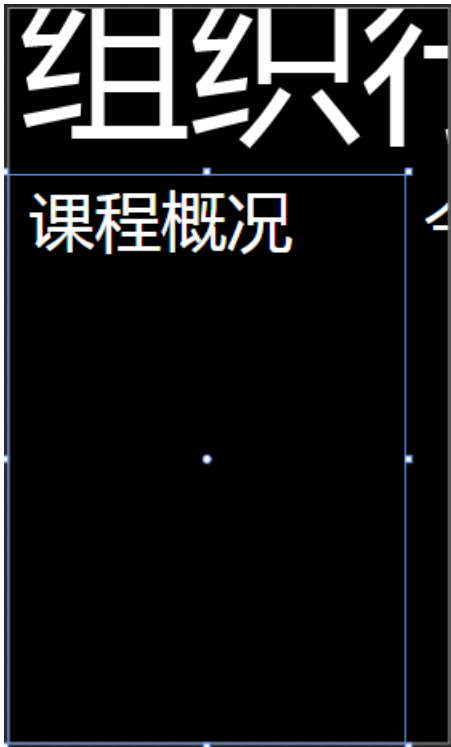


图 4-3 Course Hub 页面

这三个 Panorama 项分别用于显示当前课程的基本信息、今天记录的笔记以及今天要做的作业。接下来，我们将会详细探讨每个 Panorama 项的设计。

#### 4.1.2 设计课程概况

首先是“课程概况”，我希望它能告诉我每周星期几有课以及下节课的相关信息，如图 4-4 所示：



图 4-4 课程概况

看到这里，你可能会问，这是怎么弄的？你可以在 Panorama 项里放置两个 TextBlock，也可以放置一个 ListBox，你的决定将会影响到后面的实现，这里没有谁对谁错，你要做的只是权衡利弊，做出决定，然后承担责任。这里选择了后一种做法，主要是考虑到将来添加其它信息时可以更加方便。

#### 4.1.3 设计今天笔记

接着是“今天笔记”，要显示今天的笔记并不难，也就是一个 ListBox 的功夫，如图 4-5 所示：

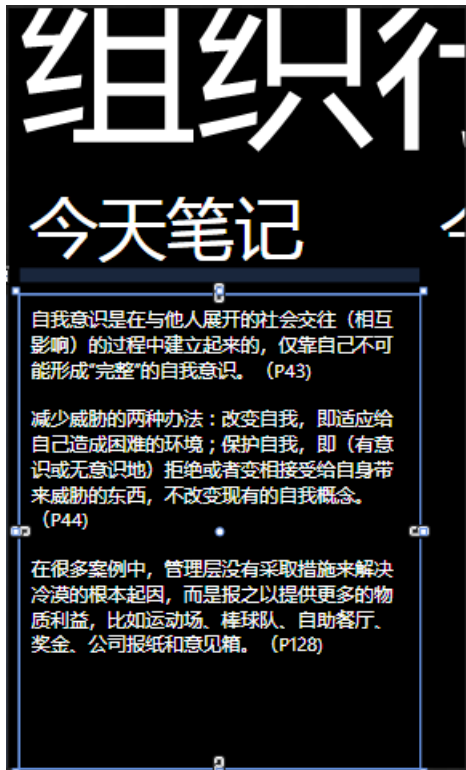


图 4-5 今天笔记

然而，仅仅这样就够了吗？我们知道，新建和查看是最常用的两项操作，可以满足用户绝大多数时候的需求，我们应该尽可能让用户最快地接触到常用的功能，这意味着我们可以考虑把新建操作集成到这里，但完整的新建操作需要打开 `NewOrEditNotePage` 页才能完成啊，而 `CourseHubPage` 页的设计原则是尽可能让涉及到的操作“就地完成”，并且用户可以马上看到结果，怎么处理？

要回答这个问题，我们得先搞清楚用户在什么情况下会通过 `CourseHubPage` 页来新建笔记。我们知道，用户在新建笔记的时候需要提供笔记内容和笔记标签，然而，只有笔记内容是必须提供的，笔记标签的提供可以延后，如果用户要在极短时间内新建笔记，比如说在课堂上，那么暂时忽略笔记标签，只输入笔记内容显然会极大地加速整个操作过程，换句话说，用户只需一个 `TextBox` 和一个 `Button` 就可以完成新建操作，如图 4-6 所示：

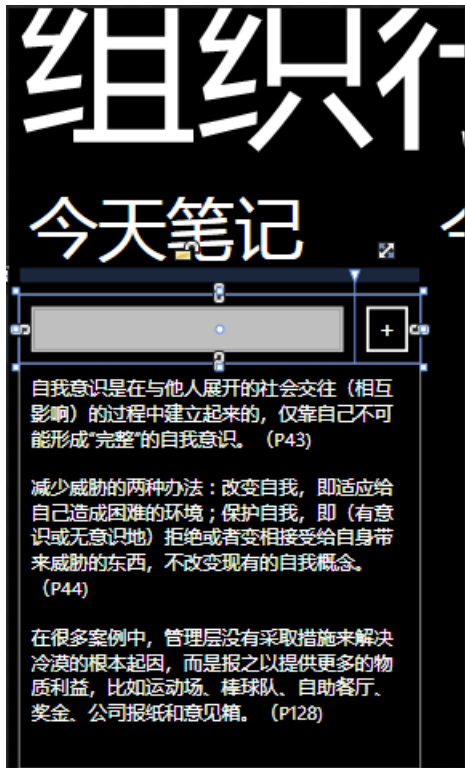


图 4-6 修改后的今天笔记

需要注意的是，TextBox 的 TextWrapping 属性的值默认是 Wrap，这是为了让 TextBox 能够根据内容的长度自动调整自身的高度，然而，这个好处在这里会导致下面的 ListBox 被挤压，从而使得笔记内容的显示空间减少，这显然不是我们希望看到的，因此我们需要把 TextBox 的 TextWrapping 属性的值设为 NoWrap。

#### 4.1.4 设计今天作业

最后是“今天作业”，出于相同的理由，我也为它配备了新建功能，如图 4-7 所示：

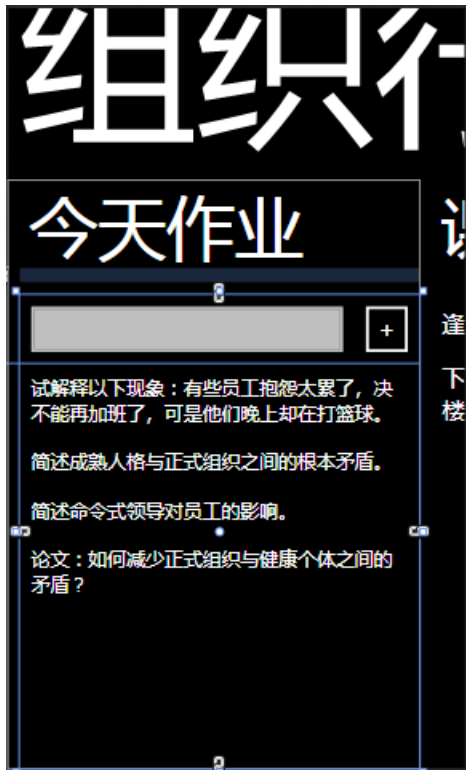


图 4-7 今天作业

然而，仅仅这样就够了吗？当然不够，首先，用户无从知晓哪些作业还没完成，其次，当用户完成一项作业时，很自然地想把它标记为已完成，然而，就目前的设计而言，用户得先退回主菜单，然后进入作业本找到并编辑这项作业，这个繁琐的过程无疑降低了应用程序的可用性。有没有办法可以简化这个过程？

我们知道，作业的完成状态是通过 `IsCompleted` 属性来标识的，这个属性的类型是 `bool`，一般而言，如果我们要在用户界面上表达这个类型的数据，我们会选择 `CheckBox`，因此，我们不妨为每项作业配备一个 `CheckBox`，如图 4-8 所示：

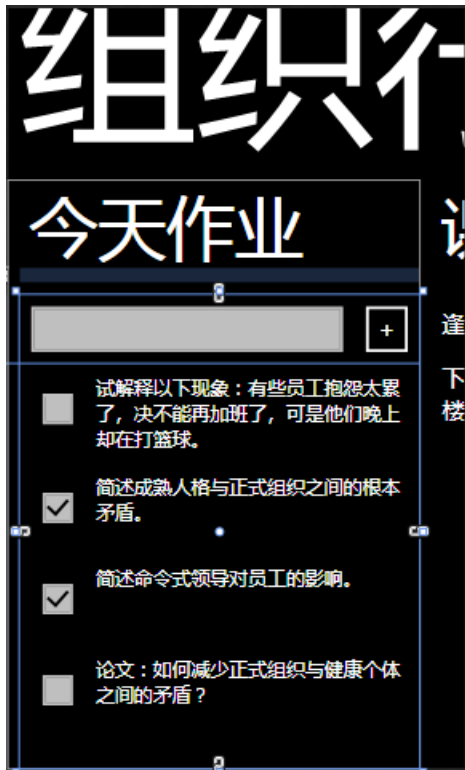


图 4-8 修改后的今天作业

这样，用户既可以直观地了解作业的完成状态，又可以轻易地更改作业的完成状态，而我们唯一要做的只是创建一个双向数据绑定！

## 4.2 连接前端和后端

### 4.2.1 创建 CourseHubViewModel 类

有了用户界面，接下来就是为它创建对应的 ViewModel 类了。首先，在 ViewModel 文件夹里创建一个 CourseHubViewModel 类，并让它继承 NotificationObject 类，如代码 4-1 所示：

```
public class CourseHubViewModel : NotificationObject
{
}

```

代码 4-1 CourseHubViewModel 类

接着，创建以下三个属性：

```

public ObservableCollection<string> Overview
{
    get;
    private set;
}

public ObservableCollection<Note> Notes
{
    get;
    private set;
}

public ObservableCollection<Assignment> Assignments
{
    get;
    private set;
}

```

代码 4-2 CourseHubViewModel 类的属性

它们将会分别绑到课程概况、今天笔记和今天作业上的 `ListBox`。那么，我们应该如何初始化它们呢？

#### 4.2.2 初始化 Assignments 和 Notes 属性

`Assignments` 属性的初始化最简单，因为 `Assignment` 类里有个 `StartDate` 属性，我们可以根据这个属性在 `CouseHubViewModel` 类的构造函数里筛选出这个课程今天的作业，如代码 4-3 所示：

```

Assignments = new ObservableCollection<Assignment>();
App.AssignmentStore.Items
    .Where(a =>
        a.CourseName == courseName &&
        a.StartDate == DateTime.Today)
    .ForEach(Assignments.Add);

```

代码 4-3 初始化 Assignments 属性

但是，`Note` 类没有类似的属性啊，怎么办？创建一个吧，如代码 4-4 所示：

```
private DateTime _createdOn;
public DateTime CreatedOn
{
    get { return _createdOn; }
    set
    {
        _createdOn = value;
        RaisePropertyChanged("CreatedOn");
    }
}
```

代码 4-4 CreatedOn 属性

这样我们就可以像筛选作业那样筛选笔记了，如代码 4-5 所示：

```
Notes = new ObservableCollection<Note>();
App.NoteStore.Items
    .Where(n =>
        n.CourseName == courseName &&
        n.CreatedOn == DateTime.Today)
    .ForEach(Notes.Add);
```

代码 4-5 初始化 Notes 属性

#### 4.2.3 初始化 Overview 属性

最麻烦的是 Overview 属性，它需要我们重新组织/聚合课程的信息，比如图 4-4 的第一条信息——“逢星期一、星期二、星期五有课”，其中“逢 XXX 有课”是固定部分，“星期一、星期二、星期五”是可变部分，这部分信息保存在 Course 对象的 Day 属性里，我们可以通过 LINQ 查询课程表里特定课程的所有 Course 对象，然后提取它们的 Day 属性，并按照我们期望的格式聚合起来，如代码 4-6 所示：

```
public CourseHubViewModel(string courseName)
{
    Overview = new ObservableCollection<string>();
    Overview.Add(
        String.Format(
            "逢{0}有课。",
            App.CourseStore.Items
                .Where(c => c.Name == courseName)
                .Select(c => c.Day)
                .Aggregate((x, y) => x + "、" + y)));
```

代码 4-6 计算星期几有课

至于第二条信息，我们得先找到下节课的 Course 对象，怎么找？想想看，如果我们手头上有一份课程表，我们会怎么找呢？我们会先看看明天有没有这节课，如果有，那就



是它了，如果没有，看看后天有没有，如此类推，下星期的今天为止。我们可以模拟这个过程查找下节课的 Course 对象，如代码 4-7 所示：

```
var date = DateTime.Today;
Course course = null;
do
{
    date = date.AddDays(1);
    var chineseDayName = GetChineseDayName(date.DayOfWeek);
    course =
        App.CourseStore.Items
            .Where(c => c.Day == chineseDayName)
            .FirstOrDefault(c => c.Name == courseName);
    if (course != null)
    {
        break;
    }
} while (date.DayOfWeek != DateTime.Today.DayOfWeek);
```

代码 4-7 计算下节课的日期

需要说明的是，某个课程（Course 对象）星期几有课是以字符串的形式表示的，而某天（DateTime 对象）星期几却是以 DayOfWeek 枚举的形式表示的，因此我们需要一个 GetChineseDayName 方法把 DayOfWeek 枚举转换成对应的中文字符串，如代码 4-8 所示：

```
private string GetChineseDayName(DayOfWeek dayOfWeek)
{
    switch (dayOfWeek)
    {
        case DayOfWeek.Friday:
            return "星期五";
        case DayOfWeek.Monday:
            return "星期一";
        case DayOfWeek.Saturday:
            return "星期六";
        case DayOfWeek.Sunday:
            return "星期日";
        case DayOfWeek.Thursday:
            return "星期四";
        case DayOfWeek.Tuesday:
            return "星期二";
        case DayOfWeek.Wednesday:
            return "星期三";
        default:
            throw new ArgumentException();
    }
}
```

代码 4-8 GetChineseDayName 方法

现在，请思考一下，GetChineseDayName 方法有没有可能抛出 ArgumentException 异常？如果有，什么情况下会抛出这个异常？如果没有，为什么？

好了，找到下节课之后，我们就可以着手相关信息的聚合了，如代码 4-9 所示：

```
Overview.Add(
    String.Format(
        "下节课：{0}月{1}日，{2} - {3}，{4}。",
        date.Month,
        date.Day,
        course.StartTime.ToShortTimeString(),
        course.EndTime.ToShortTimeString(),
        course.Location));
```

代码 4-9 显示下节课的信息

值得提醒的是，这里的做法是不具备多语言扩展的，不过，就目前而言，这已足够了。

#### 4.2.4 实现新建笔记操作

接下来，我们将会实现“今天笔记”的新建操作，从用户界面上看，这个功能是由一个 TextBox 和一个 Button 组成的，前者只需配备一个对应的字符串属性，如代码 4-10 所示：

```
private string _noteContent;
public string NoteContent
{
    get { return _noteContent; }
    set
    {
        _noteContent = value;
        RaisePropertyChanged("NoteContent");
    }
}
```

代码 4-10 NoteContent 属性

至于后者，根据上节课的经验，我们需要为它创建一个 NewNoteCommand 属性，如代码 4-11 所示：

```
public ICommand NewNoteCommand { get; private set; }
```

代码 4-11 NewNoteCommand 属性

然后在构造函数里把它初始化为一个 DelegateCommand 对象，如代码 4-12 所示：

```
NewNoteCommand = new DelegateCommand(
    () =>
    {

    },
    () =>
    {

    });
```

代码 4-12 初始化 NewNoteCommand 属性

DelegateCommand 类的构造函数接受两个参数，第一个是执行这个命令时将会调用的代码，第二个是判断这个命令能否调用的代码。当用户单击按钮时，我们需要根据当前课程、笔记内容以及今天日期等信息创建一个 Note 对象，然后保存这个 Note 对象。现在的问题是，如何通知页面的 ListBox 更新？办法其实有很多，比如说，我们可以学[第二节课](#)那样，手动监听 CollectionChanged 事件，也可以学[上节课](#)那样，通过 CollectionViewSource

间接监听 `CollectionChanged` 事件，不过，这次我打算采用更加直接的办法，在保存 `Note` 对象之后手动把它添加到 `Notes` 属性，如代码 4-13 所示：

```
var note = new Note
{
    CourseName = courseName,
    Content = NoteContent,
    CreatedOn = DateTime.Today,
};
App.NoteStore.Items.Add(note);
App.NoteStore.Commit();
Notes.Insert(0, note);
NoteContent = String.Empty;
```

代码 4-13 新建笔记的代码

需要说明的是，这里不是通过 `Add` 方法把 `Note` 对象添加到 `Notes` 属性的末尾，而是通过 `Insert` 方法把 `Note` 对象添加到 `Notes` 属性的首位，为什么这样呢？想想看，当用户单击按钮时，其视线将会落在按钮及其附近，如果新建的笔记显示在 `ListBox` 的首位，那就正好落在用户的视线范围里面，这样用户就无需挪动视线寻找并确认新建的笔记了。完成这些操作之后，我们需要把 `TextBox` 清空，为下次输入做好准备，因为 `TextBox` 是绑到 `NoteContent` 属性的，所以我们只需把 `NoteContent` 属性的值重设为空字符串就行了。

至于 `DelegateCommand` 类的构造函数的第二个参数，即判断这个命令能否调用的代码，也很简单，只有当 `TextBox` 里有内容，那么单击按钮才会执行这个命令，如代码 4-14 所示：

```
NewNoteCommand = new DelegateCommand(
    () =>
    {
        var note = new Note
        {
            CourseName = courseName,
            Content = NoteContent,
            CreatedOn = DateTime.Today,
        };
        App.NoteStore.Items.Add(note);
        App.NoteStore.Commit();
        Notes.Insert(0, note);
        NoteContent = String.Empty;
    },
    () => !String.IsNullOrEmpty(NoteContent));
```

代码 4-14 判断命令是否可以执行

“今天作业”的新建操作的实现方式和这里的一样，我打算把它留给你当今天作业，值得提醒的是，在创建 Assignment 对象时，你需要把相关属性初始化为恰当的值，这可以参考[第二节课](#)的做法。

#### 4.2.5 创建 CourseName 属性

最后，页面的课程名称也需要一个对应的属性，如代码 4-15 所示：

```
public string CourseName { get; private set; }
```

代码 4-15 CourseName 属性

看到这里，你可能会问，为什么这里直接使用自动属性，而不像代码 10 的 NoteContent 属性那样在 set 访问器里调用 RaisePropertyChanged 方法？这是因为，在用户访问 CourseHubPage 页期间，当前课程是不变的，即 CourseName 属性的值不会发生改变，因此简单的自动属性已经足够了。CourseName 属性的初始化也非常简单，只需把传给页面的课程名称赋给它就行了，如代码 4-16 所示：

```
public CourseHubViewModel(string courseName)
{
    CourseName = courseName;
}
```

代码 4-16 初始化 CourseName 属性

#### 4.2.6 设置数据绑定

创建好 ViewModel 类之后，我们就可以着手处理它和 CourseHubPage 页之间的关联了。首先是设置数据绑定，需要设置的控件以及对应的绑定表达式如下表所示：

描述	类型	属性	绑定表达式
页面标题	TextBlock	Text	{Binding CourseName}
课程概况	ListBox	ItemsSource	{Binding Overview}
今日笔记	ListBox	ItemsSource	{Binding Notes}
笔记内容	TextBox	Text	{Binding NoteContent, Mode=TwoWay}
今日作业	ListBox	ItemsSource	{Binding Assignments}
作业内容	TextBox	Text	{Binding AssignmentContent, Mode=TwoWay}

表 4-1 各个控件的绑定表达式

看到这里，你可能会问，还有两个按钮呢？上节课我们曾经说过，SL for WP 的 Button 控件没有 Command 属性，不能直接绑定命令对象，我们需要通过 Behavior 间接实现 Button 控件和命令对象之间的绑定，但 Prism 没有为 Button 控件提供现成的 Behavior，怎么办？我们可以仿照 Prism 的 [ApplicationBarButtonCommand](#) 创建一个适用于 Button 控件的 ButtonCommand，也可以直接使用 [Windows Phone 7 Developer Guide – Code Samples](#) 里面提供的 ButtonCommand，还可以使用 [MVVM Light Toolkit](#) 的 EventToCommand。毫无疑问，第二种方案是最简单的，而第三种方案则是最强大的，它不但可以把任意事件映射到

任意命令对象，最新版本还支持把事件处理程序的参数传给命令对象。下面将会示范如何通过 EventToCommand 给 Button 控件绑定命令对象。

首先，引用 GalaSoft.MvvmLight.Extras.WP7.dll 类库，接着，打开 Assets 面板，选择 Behaviors，然后把 EventToCommand 拖到 Objects and Timeline 面板的 Button 上，如图 4-9 所示：

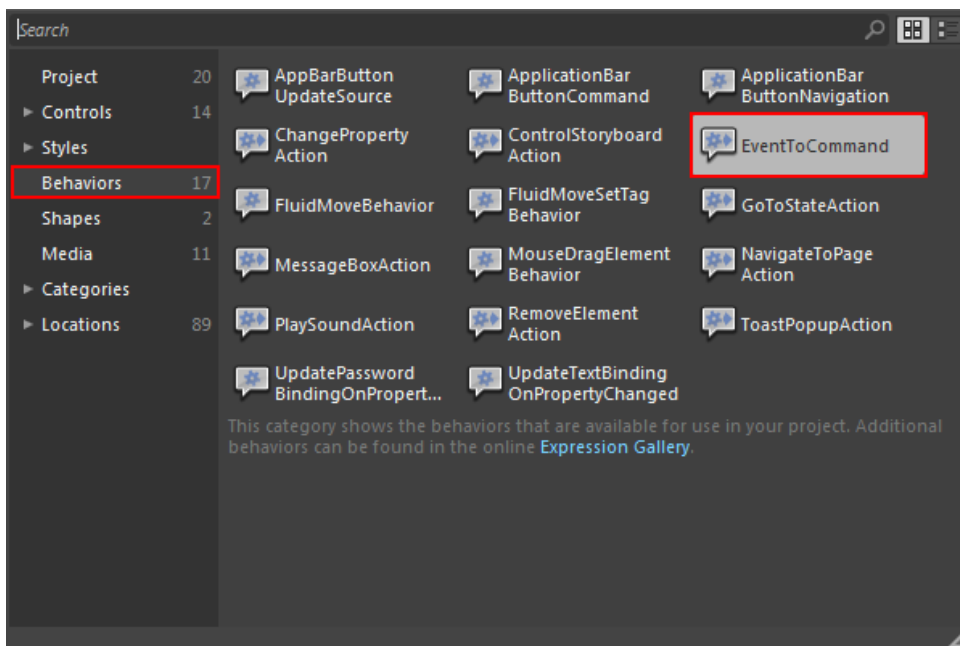


图 4-9 EventToCommand

此时，Objects and Timeline 面板将会变成这样：

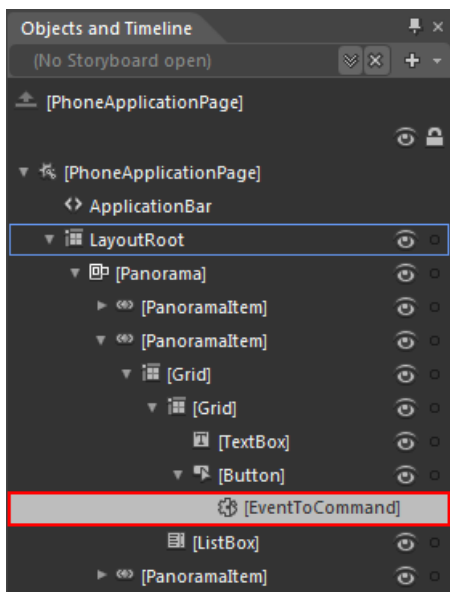


图 4-10 从 Objects and Timeline 面板上查看页面结构

确保 EventToCommand 处于选中状态，在 Properties 面板上把 Command 属性的值设为 “{Binding NewNoteCommand}”，如图 4-11 所示：



图 4-11 设置 Command 的绑定表达式

看到这里，你可能会问，不用设置事件吗？如果你查看 Properties 面板上的 EventName 属性，你会发现它已经设为 Click 了，因为 Button 控件的默认事件就是 Click。是不是很简单呢，剩下那个 Button 控件也是这样处理哦。

#### 4.2.7 把 DataContext 属性初始化为 CourseHubViewModel 对象

现在，万事俱备只欠……嗯，创建一个 CourseHubViewModel 对象，并把它赋给 CourseHubPage 页的 DataContext 属性，但是，我们从哪获取课程名称呢？我们知道，查询字符串是页面之间传递信息的一个主要途径，我们可以假设从某个页面来到 CourseHubPage 页时，查询字符串里包含了一个名为 “coursename” 的参数，然后在 OnNavigatedTo 方法里通过 NavigationContext.QueryString 来访问，如代码 4-17 所示：

```
protected override void OnNavigatedTo(NavigationEventArgs e)
{
    base.OnNavigatedTo(e);

    DataContext = new CourseHubViewModel(
        NavigationContext.QueryString["coursename"]);
}
```

代码 4-17 重写 OnNavigatedTo 方法

那么，用户如何打开这个页面？

### 4.3 打开 CourseHubPage 页

#### 4.3.1 调整课程表的用户体验

什么地方最适合用来打开 CourseHubPage 页呢？我们知道，这个页面不能直接打开，因为在打开之前我们必须提供一个课程名称，因此，仿效前三节课在主菜单里通过菜单项打开页面的做法是行不通的。如果换了传统的桌面应用程序，我们可能会使用 ComboBox 和 Button 这个组合，在 ComboBox 里列出所有课程名称让用户选择，选好之后单击按钮打开 CourseHubPage 页，可是，这里是手机应用啊，还有没有更好的方式呢？

毫无疑问，CourseHubPage 页需要一个课程上下文，哪里可以提供这个上下文呢？课程表！那么，如何设计打开 CourseHubPage 页的操作呢？有两种可能的方案，第一种是在课程表的 Application Bar 上添加一个“打开”按钮，如图 4-12 所示：



图 4-12 第一种方案

这样，用户可以通过单击选中某个课程，然后单击“打开”按钮打开 CourseHubPage 页。

第二种是把编辑和删除两个操作放在课程的上下文菜单里，把保存操作改成 Application Bar 的按钮，如图 4-13 所示：





图 4-13 第二种方案

这样，用户可以通过单击课程打开 `CourseHubPage` 页，通过长按课程打开上下文菜单，然后执行编辑或删除操作。这里选择第二种方案，这样课程表的页面设计和操作体验可以与作业本、笔记本的保持一致。

#### 4.3.2 创建保存按钮和上下文菜单

打开 `CourseTimetablePage.xaml.cs` 文件，把 `ApplicationBarCommitMenuItem_Click` 方法重命名为 `ApplicationBarCommitIconButton_Click`，然后把“保存”按钮的 Click 事件处理程序设为 `ApplicationBarCommitIconButton_Click`，如代码 4-18 所示：

```
<shell:ApplicationBarIconButton
    IconUri="/icons/appbar.save.rest.png"
    Text="保存" Click="ApplicationBarCommitIconButton_Click"/>
```

代码 4-18 添加保存 `Application Bar` 按钮

接着，打开 `CourseTimetablePage.xaml` 文件，在 `courseCollectionItemTemplate` 数据模板里添加上下文菜单，并把两个菜单项的 Click 事件处理程序分别设为 `EditMenuItem_Click` 和 `DeleteMenuItem_Click`，如代码 4-19 所示：

```

<DataTemplate x:Key="courseCollectionItemTemplate">
    <StackPanel Margin="10,12,0,12">
        <toolkit:ContextMenuService.ContextMenu>
            <toolkit:ContextMenu>
                <toolkit:MenuItem Header="编辑"
                                Click="EditMenuItem_Click"/>
                <toolkit:MenuItem Header="删除"
                                Click="DeleteMenuItem_Click"/>
            </toolkit:ContextMenu>
        </toolkit:ContextMenuService.ContextMenu>
    </StackPanel>
</DataTemplate>

```

代码 4-19 添加上下文菜单

那么，这次是否也学前面那样，把 ApplicationBarEditIconButton\_Click 和 ApplicationBarDeleteIconButton\_Click 两个方法分别重命名为 EditMenuItem\_Click 和 DeleteMenuItem\_Click 呢？不行哦，因为前后两种操作方式背后的实现原理是不一样的，之前的操作方式是单击选中课程然后单击 Application Bar 上的按钮执行相应操作，因此我们可以通过 ICollectionView.CurrentItem 来获取当前选中的课程，而现在的操作方式是长按课程打开上下文菜单然后单击菜单项执行相应操作，执行操作的时候目标课程不会变成选中状态，因此通过 ICollectionView.CurrentItem 来获取目标课程是行不通的，怎么办？

上节课我们曾经说过，上下文菜单能从列表项那里继承 DataContext 属性的值，而这个值正是目标课程，因此我们可以通过菜单项的 DataContext 属性获取目标课程的相关信息，如代码 4-20 所示：

```

private void EditMenuItem_Click(object sender, RoutedEventArgs e)
{
    var menuItem = (MenuItem)sender;
    var course = (Course)menuItem.DataContext;
    var id = course.Day + course.StartTime.ToShortTimeString();
    NavigationService.Navigate(
        new Uri("/NewOrEditCoursePage.xaml?action=edit&id=" +
            id, UriKind.RelativeOrAbsolute));
}

private void DeleteMenuItem_Click(object sender, RoutedEventArgs e)
{
    var menuItem = (MenuItem)sender;
    var course = (Course)menuItem.DataContext;
    App.CourseStore.Items.Remove(course);
}

```

代码 4-20 编辑/删除菜单项的 Click 事件处理程序

这样，编辑、删除和保存三个操作就改造完毕了，那么，打开 CourseHubPage 页的操作呢？

### 4.3.3 实现单击操作

一般情况下，我们通过 Click 事件处理程序来实现单击操作的，然而，正如代码 24 所示的那样，列表项的最外层容器是 StackPanel，它没有 Click 事件，怎么办？有三种可能的方案，第一种是通过 StackPanel 的 MouseLeftButtonUp 事件模拟单击操作，这种方案最简单，但不够精确，因为 Click 事件本身是由 MouseLeftButtonDown 和 MouseLeftButtonUp 两个事件组成的，如果你想获得更加精确的效果，可以使用 [Windows Phone 7 Developer Guide – Code Samples](#) 里面提供的 FrameworkElementClickCommand，它不但正确处理了 MouseLeftButtonDown 和 MouseLeftButtonUp 两个事件，还提供了命令对象的绑定，最后一种是使用 SL for WP Toolkit 的 GestureService/GestureListener 组件，由于 Click 事件本质上是 Tap 手势操作，我们可以通过 GestureService/GestureListener 组件监听并处理这个手势操作。下面将会示范如何通过 GestureService/GestureListener 组件处理 Tap 手势操作。

首先，打开 CourseTimestablePage.xaml 文件，在 courseCollectionItemTemplate 数据模板里添加 GestureService/GestureListener 组件，并把 GestureListener 的 Tap 事件处理程序设为 CourseItem\_Tap，如代码 4-21 所示：

```
<DataTemplate x:Key="courseCollectionItemTemplate">
    <StackPanel Margin="10,12,0,12">
        <toolkit:GestureService.GestureListener>
            <toolkit:GestureListener Tap="CourseItem_Tap"/>
        </toolkit:GestureService.GestureListener>
    </StackPanel>
</DataTemplate>
```

代码 4-21 添加单击鼠标手势

接着，在 CourseTimestablePage.xaml.cs 文件里创建这个事件处理程序，如代码 4-22 所示：

```
private void CourseItem_Tap(object sender, GestureEventArgs e)
{
    var stackPanel = (StackPanel)sender;
    var course = (Course)stackPanel.DataContext;
    NavigationService.Navigate(
        new Uri("/CourseHubPage.xaml?courseName=" +
            course.Name, UriKind.RelativeOrAbsolute));
}
```

代码 4-22 Tap 事件处理程序

需要说明的是，sender 参数指向的并非 GestureListener 对象本身，而是包含它的 StackPanel，我们可以通过 StackPanel 的 DataContext 属性获取目标课程的课程名称，以便作为查询字符串的 courseName 参数的值传给 CourseHubPage 页。

### 4.3.4 测试应用

好了，终于可以按 F5 了：

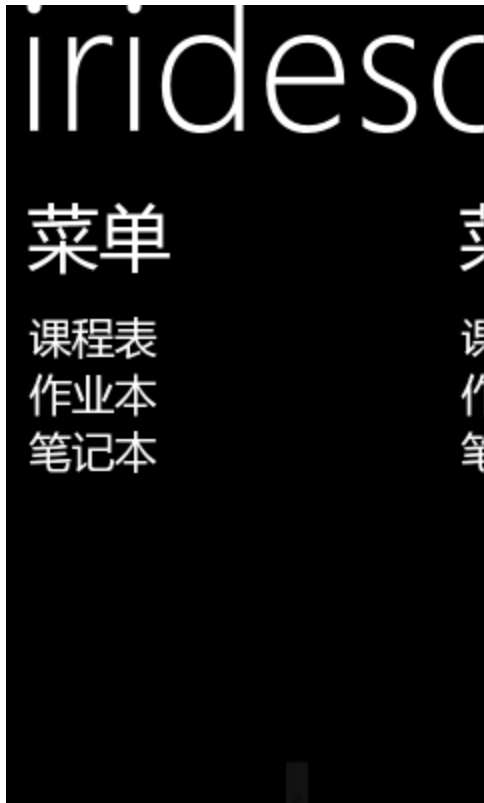


图 4-14 主菜单

单击主菜单上的“课程表”进入课程表，在星期一、星期二和星期五这三天里分别创建一节“organizational behavior”课程，如图 4-15 所示：



图 4-15 新建课程

值得注意的是，当你单击“上课地点”下面的编辑框，使之变成编辑状态时，整个页面会稍稍向上平移，这样做是为了避免软键盘打开时把编辑框挡住，这个特性是系统自带的，我们不必做任何事情就能拥有。不过，由于“确定”和“取消”两个功能还沿用着旧的设计，即通过传统的 **Button** 控件来实现，因此无可避免地被软键盘挡住了，我们应该改用上节课的新设计，即通过 **Application Bar** 按钮来实现，以便用户在输入完毕之后随时可以关闭页面，而不用先单击页面空白处关闭软键盘再单击确定按钮关闭页面，嗯，改造页面的工作就留给你当课后作业吧……

创建好课程之后，单击课程表上的任意一个课程打开 **CourseHubPage** 页，如图 4-16 所示：



图 4-16 Course Hub 页面

从上图可以看出，哪天有课已经正确显示了，下节课的上课日期也正确计算出来了。

接着，我们来看看“今天笔记”：



图 4-17 今天笔记

在编辑框里输入一条笔记，然后单击旁边的新建按钮，如图 4-18 所示：

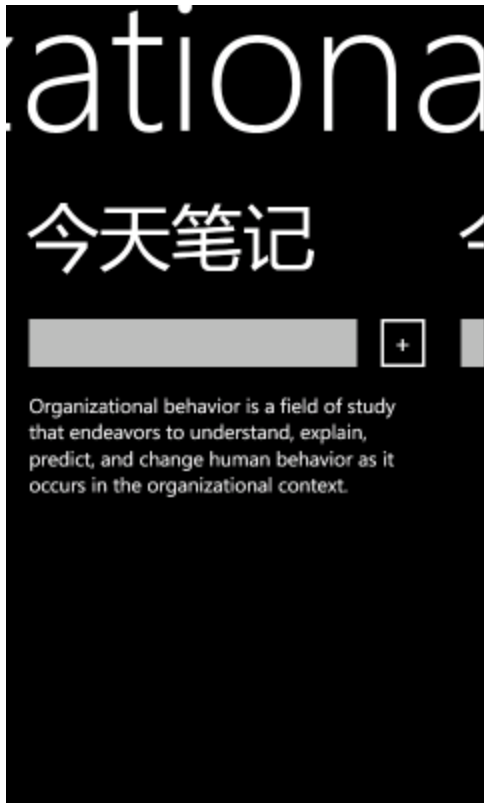


图 4-18 新建笔记

再新建两条看看：





图 4-19 今天笔记

嗯，很好，前面创建的笔记在下面，后面创建的笔记在上面。

接下来，我们看看“今天作业”：



图 4-20 今天作业

创建三项作业，并通过旁边的 CheckBox 控件把其中一项标记为已完成，如图 4-21 所示：

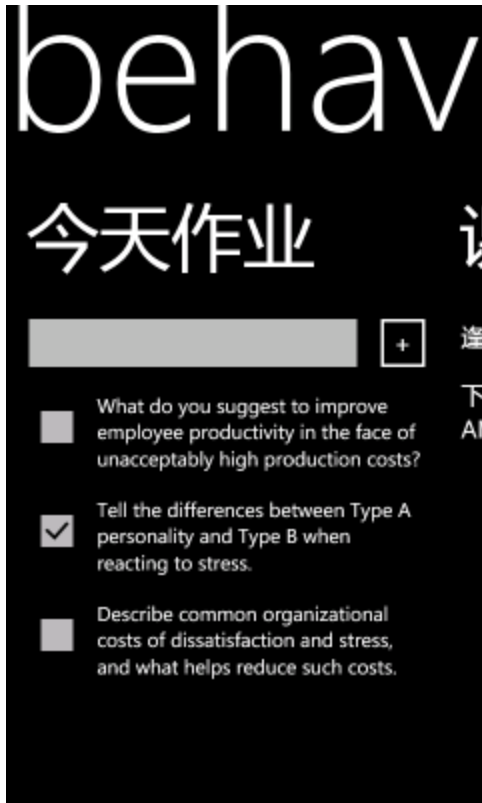


图 4-21 今天作业

现在，按两次 Back 键回到主菜单，然后单击“笔记本”进入笔记本，如图 4-22 所示：

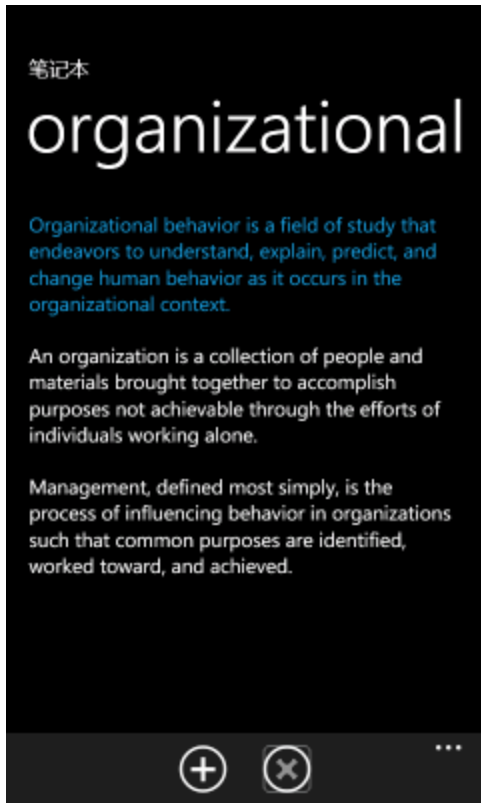


图 4-22 笔记本页面

长按第二条笔记打开上下文菜单，然后单击编辑菜单项打开编辑笔记的页面，如图 4-23 所示：

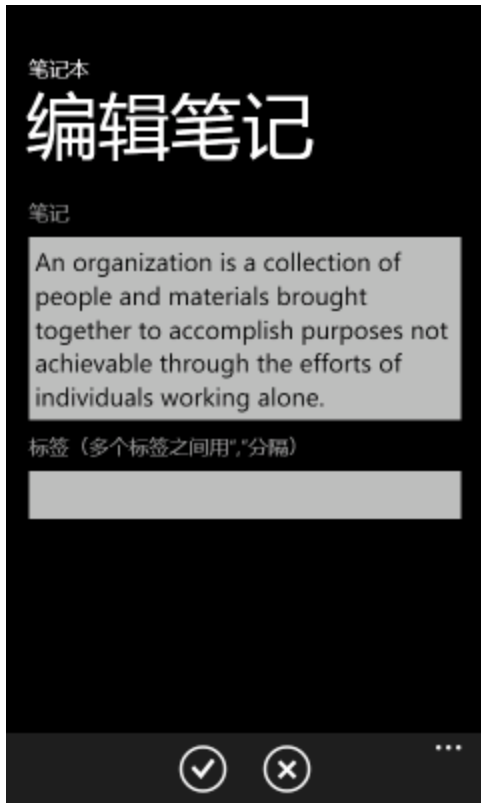


图 4-23 编辑笔记

修改一下笔记内容，如图 4-24 所示，然后按确定保存并关闭页面：

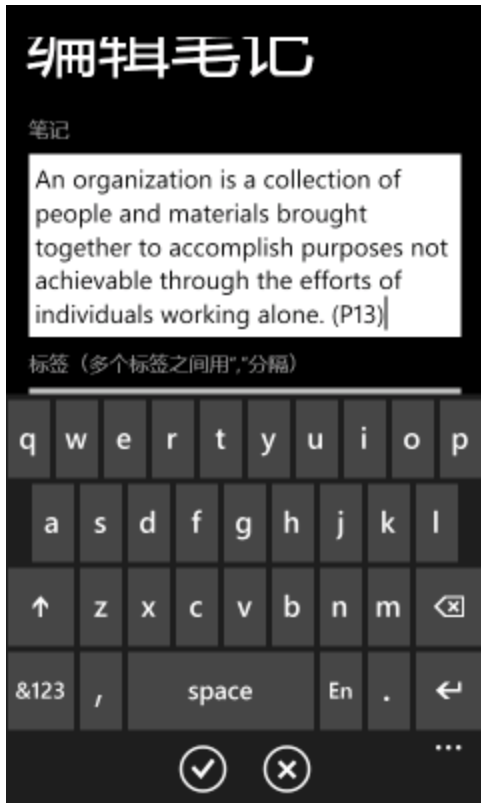


图 4-24 修改笔记内容

好了之后按 **Back** 键回到主菜单，然后单击“作业本”进入作业本，如图 4-25 所示：

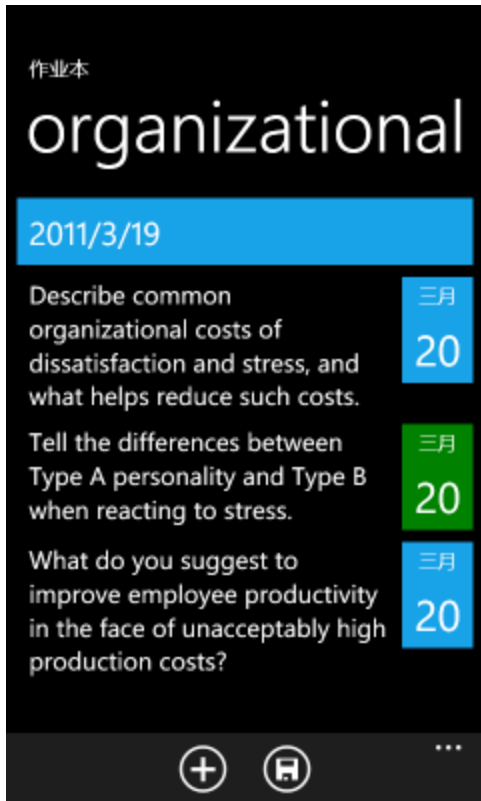


图 4-25 作业本页面

从上图可以看出，作业的起止时间、完成状态都正确设置了。

现在，长按第一项作业打开上下文菜单，然后单击编辑菜单项打开编辑作业的页面，把这项作业标记为已完成，如图 4-26 所示，并按确定关闭页面：

作业本

编辑作业

截止日期

3/20/2011

作业内容

Describe common organizational costs of dissatisfaction and stress, and what helps reduce such costs.

☒ 已完成

确定

取消

图 4-26 编辑作业

好了之后按 **Back** 键回到主菜单，然后单击“课程表”进入课程表，接着单击课程表上的任意一个课程打开 **CourseHubPage** 页，嗯，笔记内容和作业的完整状态都正确地更新了，如图 4-27 所示：



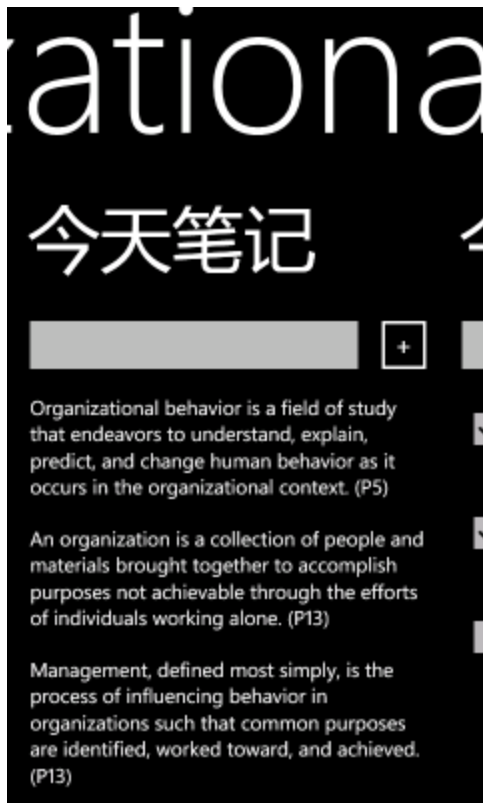


图 4-27 今天笔记

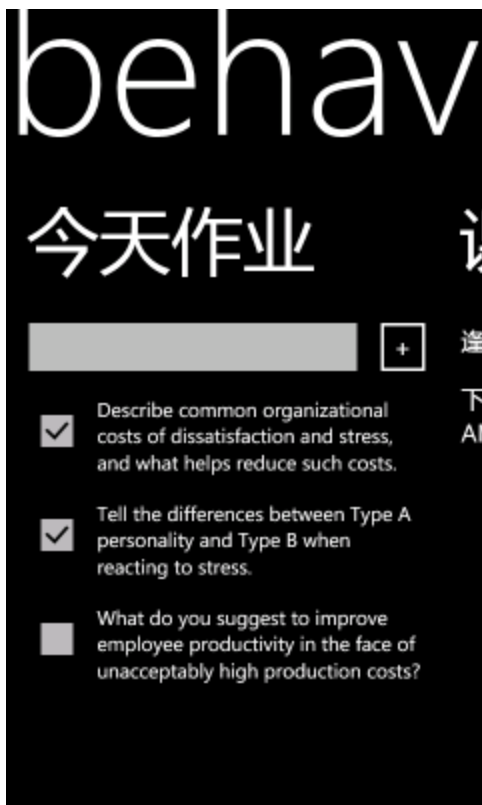


图 4-28 今天作业

嗯，非常好，只是还有一个小小的地方需要改进的，我希望软键盘能够提供词条联想，比如说，当我输入“Org”三个字母时，它可以提示“Organization”给我选择，怎样才能做到呢？很简单，把编辑框的 `InputScope` 属性设为 `Text` 就可以了，如图 4-29 所示：

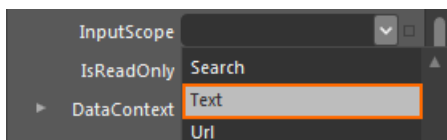


图 4-29 设置 `TextBox` 的 `InputScope` 属性

这样，当你在编辑框里输入数据时，软键盘上方就会显示相关的词条联想了，如图 4-30 所示：



图 4-30 词条联想

至此，Course Hub 已经实现完毕了，但是，你有没有觉得它有点儿单调？

## 4.4 动起来

### 4.4.1 添加 TiltEffect 效果

单调吗？那就来点儿动感吧！首先是给课程表的课程列表添加 [Tilt Effect](#)，它可以为控件交互产生一种特别的视觉反馈，感觉上就像控件后面垫了一层柔软的海绵，当你单击控件时，看起来就像你把控件放后面推了一下，当你释放控件时，它会自动复原。MSDN 提供了 Tilt Effect 的[示范代码](#)以及把它移植到你的应用的[教程](#)，不过，如果你和我一样嫌这麻烦（说到底就是懒），可以直接使用 [SL for WP Toolkit Feb 2011](#) 提供的 Tilt Effect 组件，不过，它还没修复我在[第二节课](#)里提到的 bug，因此在使用之前我们需要按照那节课提到的办法修改一下 SL for WP Toolkit 的源代码。一切准备就绪之后我们就可以应用 Tilt Effect 了。打开 CourseT timetablePage.xaml 文件，找到 pivotItemContentTemplate 数据模板，然后在 ListBox 里添加 TiltEffect.IsTiltEnabled 附加属性，如代码 4-23 所示：

```
<ListBox
    ItemTemplate="{StaticResource courseCollectionItemTemplate}"
    ItemsSource="{Binding Courses}"
    toolkit:TiltEffect.IsTiltEnabled="True"/>
```

代码 4-23 启用 TiltEffect 效果

这样就行了，是不是很简单呢，如果你希望把 Tilt Effect 应用到页面上的所有控件，你只需在 PhoneApplicationPage 里设置这个属性就可以了。

#### 4.4.2 添加 Page Transitions 效果

接下来是 Page Transitions，如果你有留意 Windows Phone 7 的相关视频，那么你应该已经看过 Windows Phone 7 的翻页效果，这是其中一种 Page Transitions，怎么把它应用到我们的页面呢？非常简单，在你想应用翻页效果的页面根元素里添加如下代码：

```
<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>
```

代码 4-24 添加 Page Transitions 效果

我们可以把这段代码分别添加到 CourseTimetablePage.xaml 和 CourseHubPage.xaml 两个页面里。

不过，仅仅这样还看不到效果，因为应用程序的 RootFrame 还只是一个普通的 PhoneApplicationFrame，我们需要把它换成 TransitionFrame。打开 App.xaml.cs 文件，在 InitializePhoneApplication 方法里把 RootFrame 初始化为 TransitionFrame，如代码 4-25 所示：

```
private void InitializePhoneApplication()
{
    if (phoneApplicationInitialized)
        return;

    // Create the frame but don't set it
    // screen to remain active until the
    RootFrame = new TransitionFrame();
}
```

代码 4-25 把 RootFrame 改为 TransitionFrame

这样，当用户单击课程表上的某个课程打开 CourseHubPage 页时，就会看到翻页效果了。除了翻页效果（Turnstile），SL for WP Toolkit 还提供了滚动（roll）、旋转（rotate）、滑动（slide）和回旋（swivel）等四种效果，你可以通过 SL for WP Toolkit 附带的示范程序看看这些效果应用到页面时会是怎样的。

#### 4.4.3 添加 FluidMoveBehavior 动画效果

最后是 [FluidMoveBehavior](#)，你知道吗，[我第一次看到它产生的效果时就被它深深地吸引住了](#)，我想，用“一见钟情”来描述这种感觉一点儿都不过分！你玩过新浪微博吗，当你发一条新的微博时，现有的微博会向下平移，与此同时，新的微博会在最上面逐渐显现出来，我希望为“今天笔记”实现这样的效果。

右击“今天笔记”上的 ListBox，选择 Edit Additional Templates\Edit Layout of Items (ItemsPanel)\Create Empty，如图 4-31 所示：

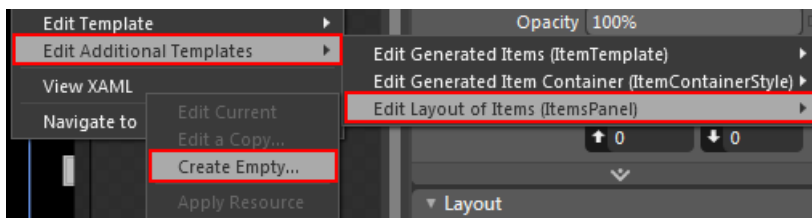


图 4-31 编辑布局模板

在弹出的 Create ItemsPanelTemplate Resource 对话框里输入模板名字，如图 4-32 所示，然后按 OK 关闭对话框：

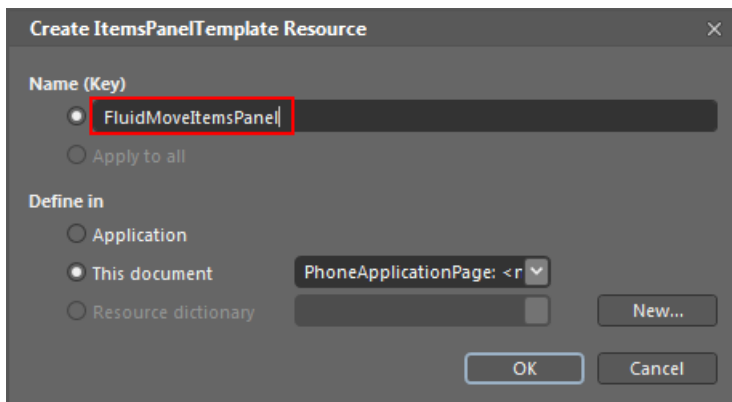


图 4-32 输入模板名字

进入模板的编辑状态之后，你会看到一个空的 `StackPanel`，从 `Assets` 面板把 `FluidMoveBehavior` 拖到 `StackPanel` 里，确保 `FluidMoveBehavior` 处于选中状态，在 `Properties` 面板上把 `AppliesTo` 和 `Duration` 两个属性分别设为“`Children`”和“`0.5`”，如图 4-33 所示：

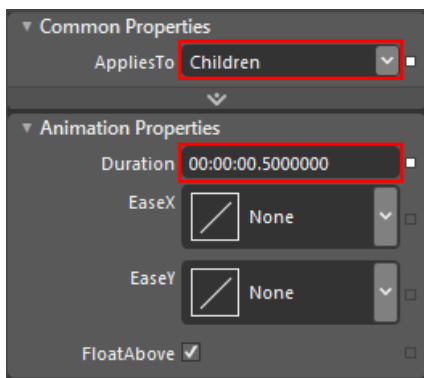


图 4-33 设置 `AppliesTo` 和 `Duration` 属性

这样就行了，是不是很简单呢，如果你希望为“今天作业”实现同样的效果，你不必重新创建一个模板，直接应用刚才那个就行了，如图 4-34 所示：

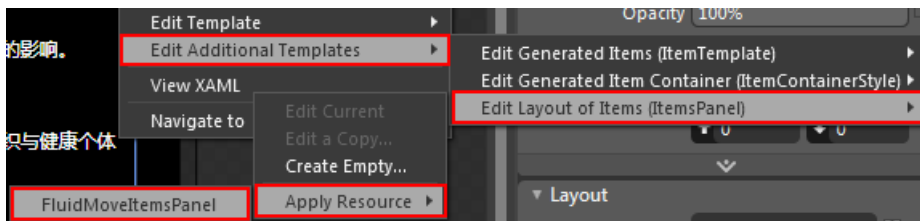


图 4-34 应用布局模板

那么，如何让新的笔记会在最上面逐渐显现出来呢？如果你看过[刚才那个视频](#)，你可能会说，用最后介绍那个办法不就行了？非常遗憾，不行哦，`LayoutStates` 这组可视状态

是 Silverlight 4 的新特性，而 Silverlight for Windows Phone 7 是基于 Silverlight 3 的，尚未支持它们，因此无法通过它们实现我们想要的效果，怎么办？

想想看，这个效果的本质是什么？一个动画！这个动画只做一件事情，使新的笔记逐渐呈现出来，换句话说，使它的 **Opacity** 属性的值从 0 逐渐变成 1。现在，右击“今天笔记”上的 **ListBox**，选择 **Edit Additional Templates\Edit Generated Items (ItemTemplate)\Edit Current**，如图 4-35 所示：

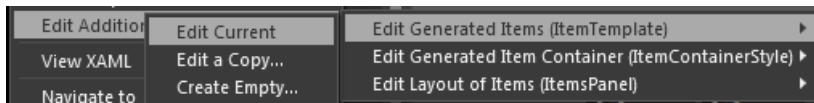


图 4-35 编辑列表项模板

在 **Objects and Timeline** 面板上选中 **StackPanel**，并在 **Properties** 面板上把它的 **Opacity** 属性的值设为 0，因为新的笔记最初是不显示的。接着，在 **Objects and Timeline** 面板上创建一个名为 **FadeIn** 的动画，并把播放指针拖到 0.5 秒处，如图 4-36 所示：

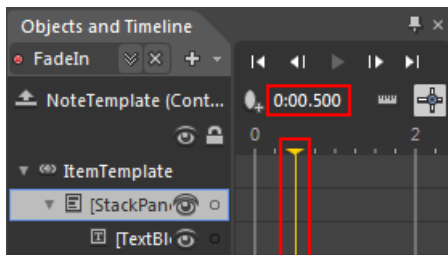


图 4-36 把播放指针拖到 0.5 秒处

然后在 **Properties** 面板上把 **StackPanel** 的 **Opacity** 属性的值设为 100，如图 4-37 所示：

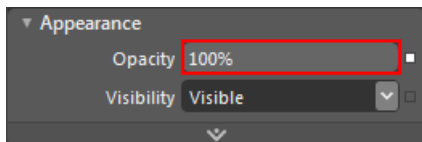


图 4-37 设置 **Opacity** 属性

需要说明的是，**Opacity** 属性的类型是 **double**，**Expression Blend** 为了免除输入小数点的麻烦在界面上把这个值扩大了 100 倍，换句话说，在 **Properties** 面板上设 100 相当于在 **XAML** 里设 1。

好了之后关闭动画，然后在 **Assets** 面板上把 **ControlStoryboardAction** 拖到 **Objects and Timeline** 面板的 **StackPanel** 上，如图 4-38 所示：

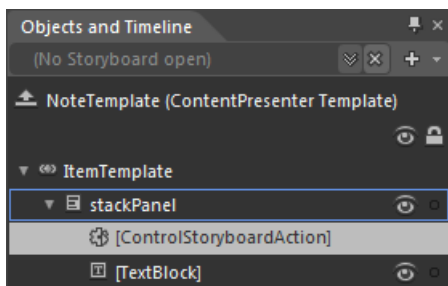


图 4-38 添加 ControlStoryboardAction

在 Properties 面板上把 EventName 和 Storyboard 两个属性的值分别设为 Loaded 和 FadeIn，如图 4-39 所示：

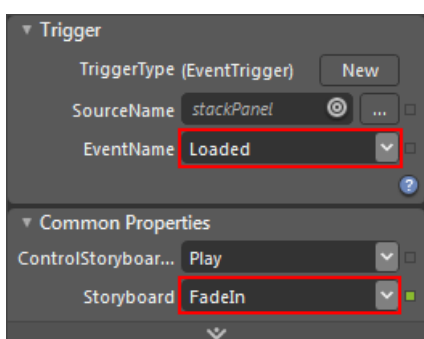


图 4-39 设置 ControlStoryboardAction 的属性

这样就大功告成了。虽然这可以实现我们想要的效果，却无可避免地带来了一个新的问题：我怎样把这个效果应用到“今天作业”呢？

#### 4.4.4 创建 FadeInWhenLoading

最直接的办法是在“今天作业”上重做一遍上面的步骤，额，这显然不是一个办法，我不想重复劳动！这个时候，我们可以创建一个 Behavior，把上面的逻辑封装起来，然后直接应用到 StackPanel 上，从而实现重用。

现在，右击 Utils 文件夹，然后选择 Add New Items，在弹出的 New Item 对话框里选择 Behavior，并把它命名为 FadeInWhenLoading，如图 4-40 所示：



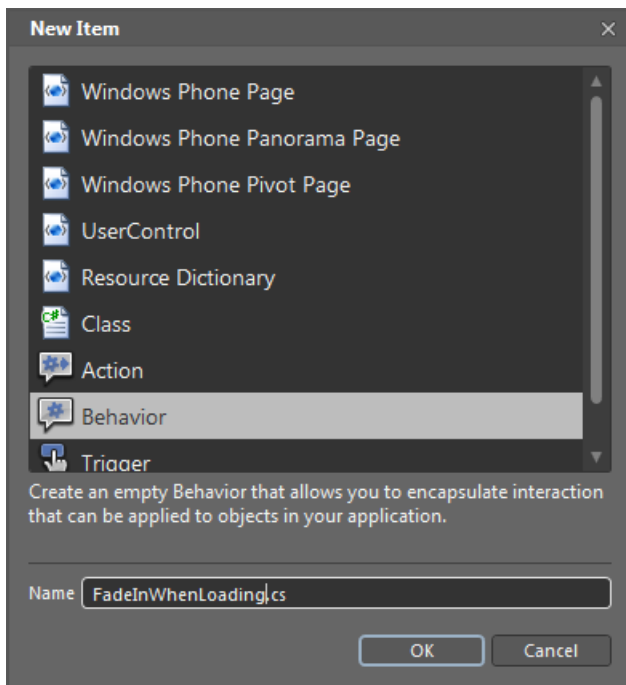


图 4-40 创建 FadeInWhenLoading

我们知道，FrameworkElement 类是整个继承体系里最先同时拥有 Opacity 属性和 Loaded 事件的，因此，我们需要把 FadeInWhenLoading 类的声明改成这样：

```
public class FadeInWhenLoading : Behavior<FrameworkElement>
{
    public FadeInWhenLoading()
    {
```

代码 4-26 修改 FadeInWhenLoading 的基类

接着，在 OnAttached 方法里把 Opacity 属性的值设为 0，并订阅 Loaded 事件，如代码 4-27 所示：

```
protected override void OnAttached()
{
    base.OnAttached();

    // Insert code that you would want run when the Behavior is attached
    AssociatedObject.Opacity = 0;
    AssociatedObject.Loaded +=
        new RoutedEventHandler(AssociatedObject_Loaded);
}
```

代码 4-27 重写 OnAttached 方法

而 `AssociatedObject_Loaded` 方法所做的事情仅仅是创建并播放使 `Opacity` 属性的值从 0 逐渐变成 1 的动画，如代码 4-28 所示：

```
private void AssociatedObject_Loaded(object sender, RoutedEventArgs e)
{
    Storyboard storyboard = new Storyboard();
    DoubleAnimation animation = new DoubleAnimation
    {
        Duration = TimeSpan.FromMilliseconds(500),
        To = 1.0,
    };
    storyboard.SetTargetProperty(
        animation, new PropertyPath(UIElement.OpacityProperty));
    storyboard.Children.Add(animation);
    storyboard.SetTarget(storyboard, AssociatedObject);
    storyboard.Begin();
}
```

代码 4-28 Loaded 事件处理程序

最后，在 `OnDetaching` 方法里取消订阅 `Loaded` 事件，如代码 4-29 所示：

```
protected override void OnDetaching()
{
    base.OnDetaching();

    // Insert code that you would want run when the Beha
    AssociatedObject.Loaded -=
        new RoutedEventHandler(AssociatedObject_Loaded);
}
```

代码 4-29 重写 `OnDetaching` 方法

好了之后，重新编译一下，然后右击“今天笔记”上的 `ListBox`，选择 `Edit Additional Templates\Edit Generated Items (ItemTemplate)\Edit Current` 进入模板编辑状态，撤销前面所做的操作（包括设置 `Opacity` 属性的值、创建动画以及应用 `ControlStoryboardAction`），然后在 `Assets` 面板上把 `FadeInWhenLoading` 拖到 `Objects and Timeline` 面板的 `StackPanel` 上，如图 4-41 所示：

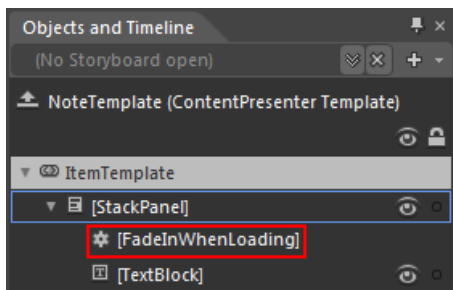


图 4-41 使用 FadeInWhenLoading

大功告成！现在，你可以把 FadeInWhenLoading 直接应用到“今天作业”上了。

动画效果是 Windows Phone 7 用户体验的一个重要组成部分，它使得应用程序不再单纯地从某个状态突然跳到另一个状态，比如 Tilt Effect，它使得控件不再只有按下和释放两种状态，当你单击控件时，它会向你单击的那个地方倾斜，非常有趣；又比如 Page Transitions，前进到一个页面和返回相同页面会产生不同的动画效果，想象你看书的时候翻到下一页和回到上一页的情景，这使你不会在这个过程中迷失；再比如 FluidMoveBehavior，它使你清楚的了解到当前查看的内容是如何变化的，所有这些，目的只有一个，提高用户体验，当然，前提是运用得当。

## 4.5 下课了……



## 第5课

# 在主页显示下一节课的信息

最重要的问题

创建用户界面

实现内部逻辑

把 ViewModel 连接到 View 上

最近的作业完成得怎样？

哪些内容是重点？



## 5.1 最重要的问题

还记得当初我们开发这个应用的目的吗？是为了随时随地可以查看课程信息。但你会闲来无事打开课程表查看课程信息吗？我就不会了！那用户要课程表来干嘛？一般情况下，用户查看课程表是为了回答以下问题：

1. 下一节课是什么？
2. 今天要上哪些课？
3. 明天要上哪些课？

我希望在主页上提供直接获取上述问题的答案的途径，比如说，我们可以在主页上添加下面这个格子（Tile）：



图 5-1 下一节课

它可以清楚地告诉用户下一节课是什么，几点开始，在哪里上。至于第二、三个问题，我们可以在主页上添加下面两个格子：



图 5-2 今天课程和明天课程

值得注意的是，这两个格子的右上角都有一个箭头记号，这是为了提醒用户单击它们可以获取相关信息。我们不可能在主页上罗列今天和明天的所有课程，这样会导致信息泛滥，同时也会加重用户识别有用信息的负担，因此我们在这里提供一些捷径，以便用户更快到达包含所需信息的地方。此外，用户最常打开的页面应该是当前课程的 **Course Hub** 了，在那里用户可以完成当前课程最常用的操作，因此我们可以在主页上再添加一个格子，帮助用户直接到达那个页面：



图 5-3 当前课程

值得注意的是，与几点上课相比，用户更加关心几点下课，因此当前课程显示的是下课时间而不是上课时间。

我们应该把这些格子放在主页上，以使用户随时了解最重要的信息：



图 5-4 主页上的课程信息

那么，怎样创建这样的用户界面呢？其实不难，上面的刷新按钮可以直接使用 [Coding4Fun Windows Phone Toolkit](#) 的 `RoundButton` 控件，而下面的四个格子由于内容结构基本相同，很容易让人想到把它们放到一个集合型的控件里，这里将会选择 `ItemsControl` 作为它们的容器。

## 5.2 创建用户界面

### 5.2.1 在主页上添加课程板块

首先，在主页上添加一个 Panorama 项，并把标题设为“课程”，如图 5-5 所示：

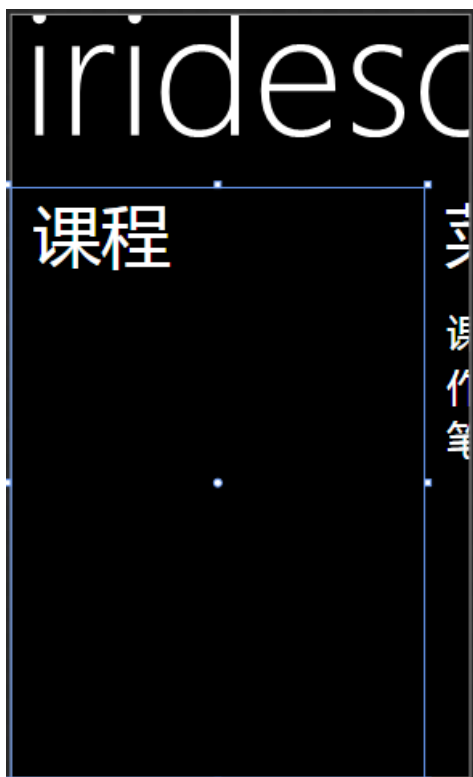


图 5-5 课程板块

新创建出来的 Panorama 项默认包含一个 Grid，我们可以通过 Properties 面板上的 RowDefinitions 属性把它分成上下两行，上面那行用来放置刷新按钮，高度设为自动适应，而下面那行用来放置 ItemsControl，高度使用默认设置。

### 5.2.2 在课程板块上添加刷新按钮

好了之后，从 Assets 面板上把 RoundButton 控件拖到 Grid 里，默认自动放在 Grid 的第一行，如图 5-6 所示（注意，在使用 RoundButton 控件之前请先引用 Coding4Fun.Phone.Control.dll 文件）：

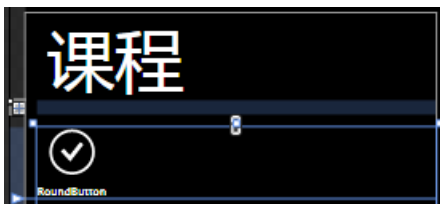


图 5-6 添加刷新按钮

设置 ImageSource 属性，把图标换成图 5-4 里面那个，然后把 Content 属性的值设为“刷新”，如图 5-7 所示：

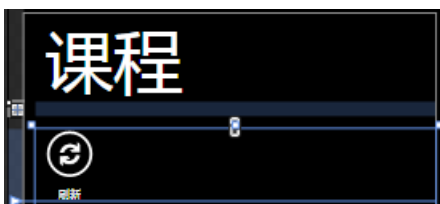


图 5-7 修改 RoundButton 按钮的图片和文字

默认情况下，RoundButton 控件的文字是位于图标下方的，而且无法直接通过设置某个属性使它的文字放在图标右边，想要达到图 5-4 的效果，我们需要修改它的样式。右击 RoundButton 控件，选择 Edit Template\Edit a Copy 进入模板编辑状态，此时，你的 Objects and Timeline 面板将会变成这样：

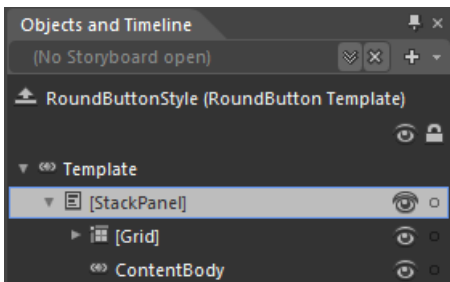


图 5-8 在 Objects and Timeline 面板上查看模板结构

确保 StackPanel 处于选中状态，如上图所示，然后在 Properties 面板上把 Orientation 属性改为 Horizontal。接着，选中 StackPanel 下面的 ContentBody，并在 Properties 面板上把字体大小设为 PhoneFontSizeMedium。

好了之后退出模板编辑状态，此时，你的 RoundButton 控件应该是这样的：

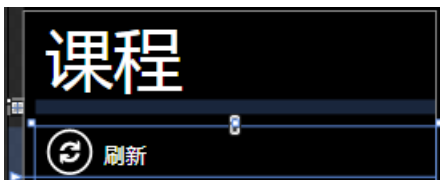


图 5-9 修改后的 RoundButton 按钮



看到这里，你可能会问，为什么要在模板编辑状态里设置字体大小呢？这是因为直接修改 RoundButton 控件的字体大小不起作用，我认为这是一个 bug。我在 CodePlex 上[提了这个问题](#)，第二天他们就发布了[修正的版本](#)，动作真的好快！

### 5.2.3 在课程板块里添加格子列表

接着，在里面添加一个 ItemsControl，在 Properties 面板上把它的 Row 属性设为 1，并使它充满 Grid 的第二行。由于 ItemsControl 默认使用 StackPanel 来排列它的子元素，这会导致格子从上到下排成一列，要使格子按照图 5-4 所示的那样排列，我们需要把 StackPanel 替换成 WrapPanel。

右击 ItemsControl，选择 Edit Additional Templates\Edit Layout of Items (ItemsPanel)\Create Empty，如图 5-10 所示：

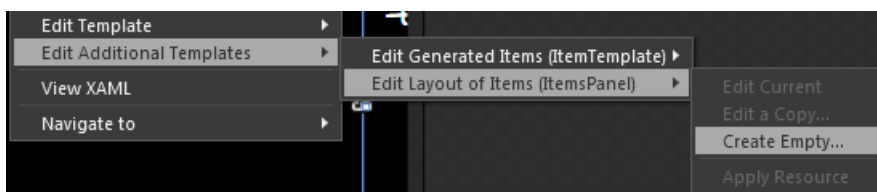


图 5-10 创建空的布局模板

在弹出的 Create ItemsPanelTemplate Resource 对话框里输入模板名字，如图 5-11 所示，然后单击 OK 关闭对话框：

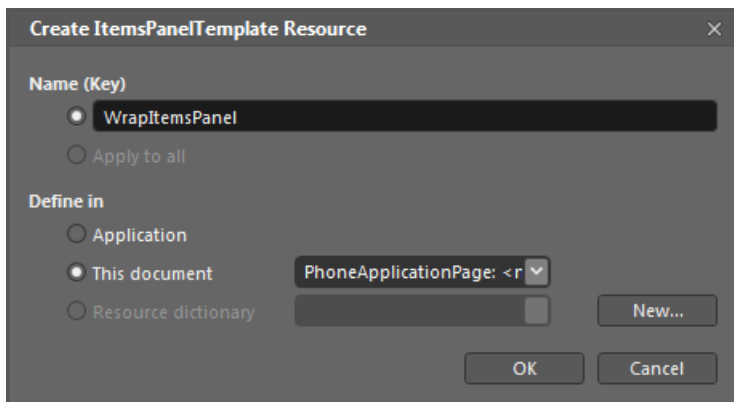


图 5-11 输入模板名字

此时，你的 Objects and Timeline 面板将会变成这样：

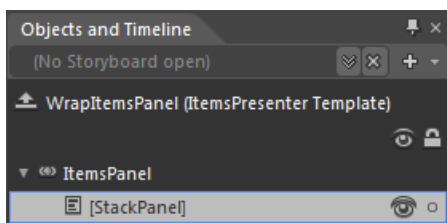


图 5-12 在 Objects and Timeline 面板上查看模板结构

把里面的 StackPanel 删掉，然后从 Assets 面板上把 WrapPanel 拖到里面，如图 5-13 所示：

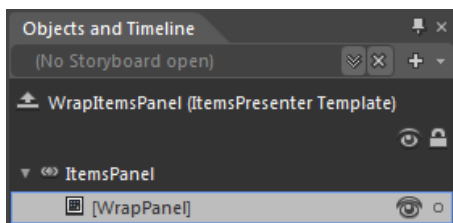


图 5-13 把 StackPanel 替换为 WrapPanel

好了之后退出模板编辑状态。

#### 5.2.4 定制格子的模板

接下来，我们将会定制每个格子的内容结构。再次右击 ItemsControl，选择 Edit Additional Templates\Edit Generated Items (ItemTemplate)\Create Empty，然后在弹出的 Create DataTemplate Resource 对话框里输入模板名字，如图 5-14 所示，然后单击 OK 关闭对话框：

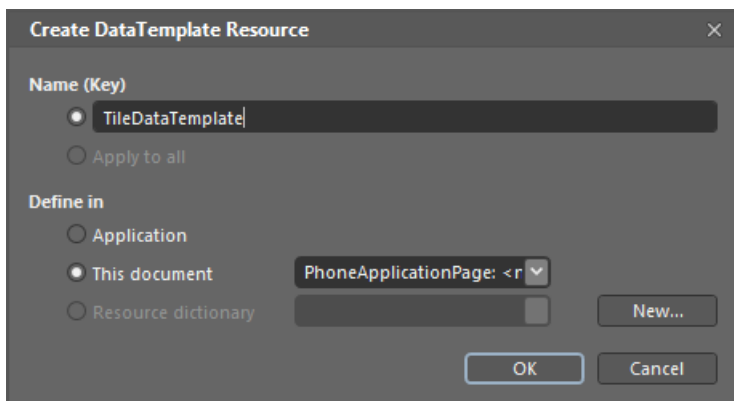


图 5-14 输入模板名字

此时，你会看到屏幕中间有个空白的 Grid，如图 5-15 所示：



图 5-15 空白的 Grid

由于这次我们没有使用设计时数据，数据模板的设计将会比较考究想象力，不过我相信你能跟得上的。

确保 Grid 出于选中状态，把它的 Width 和 Height 两个属性都设为 190（这个数值是经过多次微调得到的最佳结果），并把它的背景色设为 PhoneAccentBrush，如图 5-16 所示：

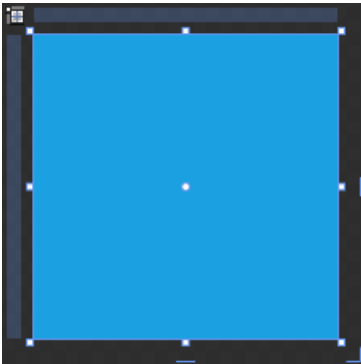


图 5-16 设置 Grid 的背景色

然后把 Grid 的 Margin 设成这样：



图 5-17 调整 Grid 的 Margin 属性

接着，在 Grid 里添加相应的控件，使它变成这样：

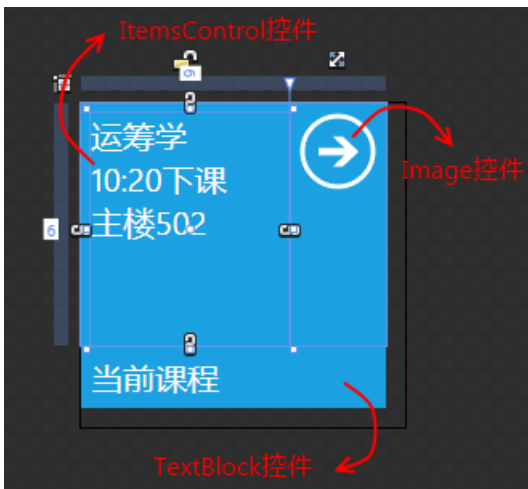


图 5-18 在 Grid 里添加相应的控件

此时，你的 Objects and Timeline 面板应该是这样的：

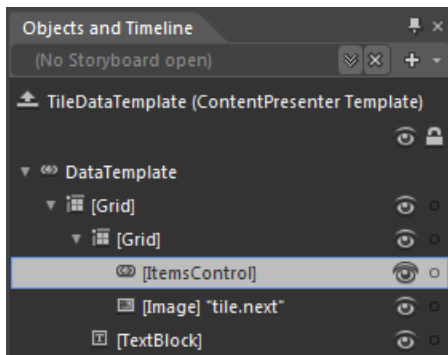


图 5-19 在 Objects and Timeline 面板上查看模板结构

值得提醒的是，在调整控件的大小和位置时，你可以暂时为控件硬编码一些内容，比如上图的“当前课程”和课程信息。好了之后退出模板编辑状态。

## 5.3 实现内部逻辑

### 5.3.1 调查需求

现在是时候为那些格子创建对应的 **ViewModel** 类了，不过，在开始之前我们需要搞清楚这些格子的工作方式：

1. 每个格子都有一个标题，标题内容不会改变。
2. 每个格子都有一组信息，用户可以通过单击刷新按钮更新这些信息。
3. 如果用户单击格子时将会执行某个操作，那么格子右上角需要显示箭头图标，否则不显示该图标。
4. “明天课程”有两个状态：明天有课和明天没课，如果明天有课，则显示课程总数，并显示箭头图标，单击将会打开课程表并显示明天的课程；否则显示“明天没课”，且不显示箭头图标，单击也不执行任何操作。
5. “今天课程”有三个状态：今天没课、今天有课和已经放学，如果今天没课，则显示“今天没课”，且不显示箭头图标，单击不执行任何操作；如果今天有课，并且还有课程剩余，则显示剩余课程的数目，否则显示“放学了”，这两种状态都会显示箭头图标，单击将会打开课程表并显示今天课程。
6. “下一节课”也有三个状态：今天没课、今天有课和已经放学，如果今天没课，则显示“今天没课”；如果今天有课，并且还有课程剩余，则显示下一节课，否则显示“放学了”，这个格子不显示箭头图标，单击也不执行任何操作。
7. “当前课程”有五个状态：今天没课、还没上课、正在上课、课间休息和已经放学，仅当它处于正在上课状态时才显示箭头图标，此时单击将会打开当前课程的 **Course Hub**，处于其它状态时只显示状态名称，且单击不执行任何操作。

噢，看起来有点复杂哦，不过不用担心，我们一起有步骤地地把它们实现了。

### 5.3.2 创建 TileViewModelBase 类

首先，在 ViewModels 文件夹里创建一个 TileViewModelBase 抽象类，并让它继承 NotificationObject 类，如代码 5-1 所示：

```
public abstract class TileViewModelBase : NotificationObject
{
}

```

代码 5-1 TileViewModelBase 类

这里将会放置被四个格子的 ViewModel 类共用的代码，那么，哪些代码是共用的呢？

根据上面的讨论，每个格子都需要提供一组信息，里面可能包含课程信息或者格子的状态信息，我们可以把这些信息放在一个集合里，并在构造函数里初始化这个集合，如代码 5-2 所示：

```
public TileViewModelBase()
{
    Contents = new ObservableCollection<string>();
}

public ObservableCollection<string> Contents { get; private set; }

```

代码 5-2 创建并初始化 Contents 属性

其次，每个格子都有一个标题，但这个标题的内容不会改变，因此我们可以通过自动属性来实现，如代码 5-3 所示：

```
public string Title { get; protected set; }

```

代码 5-3 Title 属性

这个属性的初始化需要在每个格子的 ViewModel 类的构造函数里进行，因为每个格子的标题都不同。

至于格子右上角的图标，我们只提供一个布尔型属性，指示是否显示这个图标就行了，不过，由于这个图标的显示和格子的状态紧密相连，我们需要为这个布尔型属性提供通知机制，以便在格子的状态发生改变时可以通知用户界面做出相应的调整，如代码 5-4 所示：

```
private bool _showNextIcon;
public bool ShowNextIcon
{
    get { return _showNextIcon; }
    set
    {
        _showNextIcon = value;
        RaisePropertyChanged("ShowNextIcon");
    }
}
```

代码 5-4 ShowNextIcon 属性

此外，格子还需要支持单击操作和刷新操作，如代码 5-5 所示：

```
public ICommand TileCommand { get; protected set; }
public ICommand RefreshCommand { get; protected set; }
```

代码 5-5 单击和刷新命令

由于每个格子在计算应该显示的信息时都要获取今天或者明天的全部课程，我们可以把这部分代码放在基类里，如代码 5-6 所示：

```
protected IEnumerable<Course> GetCourses(string day)
{
    return App.CourseStore.Items.Where(c => c.Day == day);
}
```

代码 5-6 获取某天的课程

有了这个基类，我们就可以分别为那些格子创建对应的 ViewModel 类了。

### 5.3.3 实现“明天课程”

我们先挑一个简单的来试一下，根据前面的讨论，“明天课程”最简单，因此我们先为它创建 ViewModel 类。在 ViewModels 文件夹里创建一个 TomorrowCoursesTileViewModel 类，并让它继承 TileViewModelBase 抽象类，如代码 5-7 所示：

```
public class TomorrowCoursesTileViewModel : TileViewModelBase
{
    private string _day;
}
```

代码 5-7 TomorrowCoursesTileViewModel 类

当用户单击这个格子将会打开课程表并显示明天的课程，为了执行这个操作我们需要知道明天是星期几，因此我在里面添加了一个 \_day 私有字段，用来存放这个信息。

接着，我们需要在构造函数里初始化相关的属性，如代码 5-8 所示：

```
public TomorrowCoursesTileViewModel()
{
    Title = "明天课程";
    TileCommand = new DelegateCommand(
        () =>
        {
            1
        },
        () => ShowNextIcon);
    RefreshCommand = new DelegateCommand(
        () =>
        {
            2
        });
}
```

代码 5-8 初始化相关属性

其中，打开课程表的代码将会放在（1）里，而初始化格子的代码则会放在（2）里。

看到这里，你可能会问，如何在 ViewModel 类里打开一个页面？一直以来，我们都是在一个页面里通过 NavigationService 的 Navigate 方法打开另一个页面，而在 ViewModel 类里，NavigationService 是访问不了的，怎么办？要解决这个问题，我们得先搞清楚 Windows Phone 的页面导航模型是怎样的，Windows Phone 的导航模型是由一个 PhoneApplicationFrame 和一个或多个 PhoneApplicationPage 共同组成的，前者提供与导航相关的事件和 Navigate 方法，而后者则包含应用的内容，它们共享着同一个 NavigationService。虽然在 ViewModel 类里无法访问页面的资源，但是我们可以通过 App 类获取当前应用的 PhoneApplicationFrame 对象，继而调用它的 Navigate 方法，如代码 5-9 所示：

```
((App)App.Current).RootFrame.Navigate(
    new Uri("/CourseTimetablePage.xaml?day=" + _day,
        UriKind.RelativeOrAbsolute));
```

代码 5-9 调用 Navigate 方法

事实上，MVVM 模式认为 ViewModel 层不应该依赖于 View 层的任何东西，更好的做法应该是创建一个 INavigationService 接口把导航服务隔离开来，然后在 ViewModel 类里通过依赖注入的方式来获取和访问导航服务，如果你打算为你的 ViewModel 类创建单元测试，那么这种隔离就更加有必要了。不过，我不想在这里把事情弄得太复杂，因此采用代码 5-9 这种简单直接的做法。此外，需要说明的是，仅当格子右上角显示箭头图标时单击格子

才会执行相应操作，因此在初始化 `TileCommand` 属性的时候我们需要告诉它根据 `ShowNextIcon` 属性的值来判断是否允许执行相应的操作。

而刷新方面，我们只需要看看明天是否有课，然后根据情况更新相关的信息就行了，如代码 5-10 所示：

```
Contents.Clear();

var tomorrow = DateTime.Today.AddDays(1);
Contents.Add(String.Format("{0}月{1}日", tomorrow.Month, tomorrow.Day));
_day = tomorrow.DayOfWeek.GetChineseDayName();
Contents.Add(_day);

var count = GetCourses(_day).Count();
if (count > 0)
{
    Contents.Add(String.Format("一共{0}节课", count.ToString()));
    ShowNextIcon = true;
}
else
{
    Contents.Add("明天没课");
    ShowNextIcon = false;
}
```

代码 5-10 刷新格子

需要说明的是，`GetChineseDayName` 方法是一个扩展方法，用来生成与 `DayOfWeek` 枚举值对应的中文星期名称，事实上，它是从上节课的 `CourseHubViewModel` 类里提取出来的。

#### 5.3.4 实现“今天课程”

接着，我们来看看“今天课程”，它和“明天课程”没有太大区别，我们只是需要多处理一个状态，以及把显示课程总数改为显示剩余课程的数目。在 `ViewModels` 文件夹里创建一个 `TodayCoursesTileViewModel` 类，并让它继承 `TileViewModelBase` 抽象类，如代码 5-11 所示：



```

public class TodayCoursesTileViewModel : TileViewModelBase
{
    public TodayCoursesTileViewModel()
    {
        Title = "今天课程";
        TileCommand = new DelegateCommand(
            () =>
            {
                ((App)App.Current).RootFrame.Navigate(
                    new Uri("/CourseTimetablePage.xaml?day=" + _day,
                        UriKind.RelativeOrAbsolute));
            },
            () => ShowNextIcon);
        RefreshCommand = new DelegateCommand(
            () =>
            {
                });
    }

    private string _day;
}

```

代码 5-11 TodayCoursesTileViewModel 类

刷新的时候，如果今天的课程数目为 0，当然是显示“今天没课”了，如代码 5-12 所示：

```

Contents.Clear();

var today = DateTime.Today;
var thisTime = new DateTime(
    1, 1, 1, DateTime.Now.Hour, DateTime.Now.Minute, 0);
Contents.Add(String.Format("{0}月{1}日", today.Month, today.Day));
_day = today.DayOfWeek.GetChineseDayName();
Contents.Add(_day);

var count = GetCourses(_day).Count();
if (count == 0)
{
    Contents.Add("今天没课");
    ShowNextIcon = false;
}

```

代码 5-12 刷新格子（1）

否则，看看是否所有课程都结束了，如果是就显示“放学了”，如代码 5-13 所示：

```
else if (GetCourses(_day).All(c => thisTime > c.EndTime))
{
    Contents.Add("放学了");
    ShowNextIcon = true;
}
```

代码 5-13 刷新格子 (2)

否则，显示剩余课程的数目，如代码 5-14 所示：

```
else
{
    var remainingCount =
        GetCourses(_day).Count(c => c.StartTime > thisTime);
    Contents.Add(
        String.Format("剩余{0}节课", remainingCount.ToString()));
    ShowNextIcon = true;
}
```

代码 5-14 刷新格子 (3)

需要说明的是，这里把剩余课程定义为还没开始的课程，已经开始但还没下课的不包含在里面。此外，Course 类的 StartTime 和 EndTime 两个属性的值是通过 SL for WP Toolkit 的 TimePicker 设置的，由于它只关心时间部分，因此把年、月和日的值都设为 1 了，而 DateTime.Now 包含了今天的年、月和日，会对后面的比较造成影响，所以我另外创建了一个 thisTime。

不过，要让打开的课程表正确显示我们想要的课程，还得修改一下课程表的代码。打开 CourseTimetablePage.xaml.cs 文件，在构造函数里把 Loaded 事件处理程序的代码改成下面这样：

```

Loaded +=
    (o, e) =>
    {
        int selectedColumnIndex = (int)DateTime.Today.DayOfWeek;

        if (NavigationContext.QueryString.ContainsKey("day"))
        {
            selectedColumnIndex =
                (int)NavigationContext.QueryString["day"]
                .GetDayOfWeekValue();
        }

        ((CourseTimetableViewModel)DataContext).SelectedColumnIndex =
            selectedColumnIndex;
    };

```

代码 5-15 修改课程表页面的 Loaded 事件处理程序

当用户打开课程表时，默认显示今天的课程，如果查询字符串里面包含了 day 参数，则显示该参数指定的那天课程。需要说明的是，GetDayOfWeekValue 是一个扩展方法，用来生成与中文星期名称对应的 DayOfWeek 枚举值，本质上它是 GetChineseDayName 方法的反向实现。

### 5.3.5 实现“下一节课”

接着，我们来看看“下一节课”。在 ViewModels 文件夹里创建一个 NextCourseTileViewModel 类，并让它继承 TileViewModelBase 抽象类，如代码 5-16 所示：

```

public class NextCourseTileViewModel : TileViewModelBase
{
    public NextCourseTileViewModel()
    {
        Title = "下一节课";
        ShowNextIcon = false;
        TileCommand = new DelegateCommand(
            () => { },
            () => ShowNextIcon);
        RefreshCommand = new DelegateCommand(
            () =>
            {
                // ...
            });
    }
}

```

代码 5-16 NextCourseTileViewModel 类

根据前面的讨论，这个格子不显示箭头图标，单击也不执行任何操作，因此我们把 `ShowNextIcon` 属性的值设为 `false`，把 `TileCommand` 属性初始化为“空”命令。看到这里，你可能会问，为什么要这样初始化 `TileCommand` 属性呢？因为将来设置数据绑定的时候，我们是在格子的模板里统一设置的，如果某个格子的 `TileCommand` 属性为 `null`，那么将来用户单击这个格子时可能会抛出 `NullReferenceException`。

刷新的时候，如果今天的课程数目为 0 就显示“今天没课”了，如果所有课程都结束了就显示“放学了”，如代码 5-17 所示：

```
Contents.Clear();

var day = DateTime.Today.DayOfWeek.GetChineseDayName();
var thisTime = new DateTime(
    1, 1, 1, DateTime.Now.Hour, DateTime.Now.Minute, 0);

var count = GetCourses(day).Count();
if (count == 0)
{
    Contents.Add("今天没课");
}
else if (GetCourses(day).All(c => thisTime > c.EndTime))
{
    Contents.Add("放学了");
}
```

代码 5-17 刷新格子 (1)

否则，我们需要从剩余课程里找出下一节课。如果此时正在上最后一节课呢？前面我们把剩余课程定义为还没开始的课程，那么现在的状态既没有剩余课程又不是已经放学，怎么办？这是一个临界状态，我们可以增加一个状态来描述这种情况，如果现在是最后一节课，那么我们可以显示“最后一节了”，否则显示下一节课的相关信息，如代码 5-18 所示：

```

else
{
    var nextCourse =
        GetCourses(day).FirstOrDefault(c => c.StartTime > thisTime);
    if (nextCourse == null)
    {
        Contents.Add("最后一节了");
    }
    else
    {
        Contents.Add(nextCourse.Name);
        Contents.Add(nextCourse.StartTime.ToShortTimeString() + "上课");
        Contents.Add(nextCourse.Location);
    }
}

```

代码 5-18 刷新格子 (2)

### 5.3.6 实现“当前课程”

最后，我们来看看“当前课程”，这是四个格子里面最麻烦的，也是状态最多的。在 ViewModels 文件夹里创建一个 CurrentCourseTileViewModel 类，并让它继承 TileViewModelBase 抽象类，如代码 5-19 所示：

```

public class CurrentCourseTileViewModel : TileViewModelBase
{
    public CurrentCourseTileViewModel()
    {
        Title = "当前课程";
        TileCommand = new DelegateCommand(
            () =>
            {
                ((App)App.Current).RootFrame.Navigate(
                    new Uri("/CourseHubPage.xaml?courseName=" +
                        _courseName,
                        UriKind.RelativeOrAbsolute));
            },
            () => ShowNextIcon);
        RefreshCommand = new DelegateCommand(
            () =>
            {
            });
    }

    private string _courseName;
}

```

代码 5-19 CurrentCourseTileViewModel 类

刷新的时候，如果今天的课程数目为 0 就显示“今天没课”了，如果所有课程都没开始就显示“还没上课”，如果所有课程都结束了就显示“放学了”，如代码 5-20 所示：

```

Contents.Clear();

var day = DateTime.Today.DayOfWeek.GetChineseDayName();
var thisTime = new DateTime(
    1, 1, 1, DateTime.Now.Hour, DateTime.Now.Minute, 0);

var count = GetCourses(day).Count();
if (count == 0)
{
    Contents.Add("今天没课");
    ShowNextIcon = false;
}
else if (GetCourses(day).All(c => c.StartTime > thisTime))
{
    Contents.Add("还没上课");
    ShowNextIcon = false;
}
else if (GetCourses(day).All(c => thisTime > c.EndTime))
{
    Contents.Add("放学了");
    ShowNextIcon = false;
}

```

代码 5-20 刷新格子 (1)

否则，看看此时是不是正在上课，如果是就显示课程信息，如果不是就显示“课间休息”，如代码 5-21 所示：

```

else
{
    var currentCourse =
        App.CourseStore.Items.FirstOrDefault(
            c => c.StartTime <= thisTime && thisTime <= c.EndTime);
    if (currentCourse == null)
    {
        Contents.Add("课间休息");
        ShowNextIcon = false;
    }
    else
    {
        Contents.Add(currentCourse.Name);
        Contents.Add(
            currentCourse.EndTime.ToShortTimeString() + "下课");
        Contents.Add(currentCourse.Location);
        ShowNextIcon = true;
        _courseName = currentCourse.Name;
    }
}
}

```

代码 5-21 刷新格子 (2)

看到这里，你可能会问，每次刷新格子都要通过 LINQ 做很多次遍历，这样会不会降低应用的效率？嗯，是的，目前的做法不是很好，比较好的做法是获取某天的课程，然后对它们进行排序，并缓存到一个集合里，接着从前往后做一次遍历，看看当前时间在哪个位置，从而判断格子处于什么状态。但即使这样，每次刷新还是需要一次遍历，更好的做法应该是每次找到当前时间的位置，就用一个变量做个记号，有点像书签一样，下次刷新时就从这个位置开始往后遍历，这样的话，理想状态下一天就只需做一次遍历了。不过，所有这些都是以一个完整且稳定的课程表为前提的，如果你打算让你的课程表支持临时调课、插课或者和远程服务器进行同步，那么这些优化措施可能会让事情变得糟糕。我不建议太早进行性能优化，因为这样可能会导致设计僵化，我无法告诉你什么时候适合优化性能，你必须根据自己的设计和计划作出决定。但有一点我是可以肯定的，如果性能问题超出了用户的容忍程度，那么无论如何我们都要采取行动了，因为用户无法直接看到你的设计有多灵活，却非常清楚他们内心此刻的负面感受。

### 5.3.7 创建并初始化 TileViewModels 和 RefreshTilesCommand 属性

现在，打开 MainViewModel.cs 文件，它是我们创建这个项目的时候 Expression Blend 自动为主页创建的 ViewModel 类，不过里面有很多我们不需要的代码，在继续之前先把它们清理掉，但保留 INotifyPropertyChanged 接口的实现代码。好了之后，在里面创建下面两个属性，如代码 5-22 所示：



```

public ObservableCollection<TileViewModelBase> TileViewModels
{
    get;
    private set;
}

public ICommand RefreshTilesCommand { get; private set; }

```

代码 5-22 TileViewModels 和 RefreshTilesCommand 属性

接着，在构造函数里初始化这两个属性，如代码 5-23 所示：

```

public MainViewModel()
{
    TileViewModels = new ObservableCollection<TileViewModelBase>
    {
        new CurrentCourseTileViewModel(),
        new NextCourseTileViewModel(),
        new TodayCoursesTileViewModel(),
        new TomorrowCoursesTileViewModel(),
    };

    RefreshTilesCommand = new DelegateCommand(
        () =>
        {
            TileViewModels.ForEach(
                t => t.RefreshCommand.Execute(null));
        });

    RefreshTilesCommand.Execute(null);
}

```

代码 5-23 初始化 TileViewModels 和 RefreshTilesCommand 属性

初始化完这两个属性之后别忘了“刷新”一下哦，否则主页上的那些格子不会显示任何东西的。至此，ViewModel 类全部创建完毕了，接下来，我们将会把它们关联到对应的 View 上。

## 5.4 把 ViewModel 连接到 View 上

### 5.4.1 设置数据绑定

如果你是在 Visual Studio 里写代码的，那么现在是时候回到 Expression Blend 了。确保主页上的 ItemsControl 处于选中状态，在 Properties 面板上单击 ItemsSource 属性右边的小

正方形，选择 Custom Expression，然后在弹出的 Custom Expression 对话框里输入 “{Binding TileViewModels}”。

接着，右击 ItemsControl，选择 Edit Additional Templates\Edit Generated Items (ItemTemplate)\Edit Current 进入模板编辑状态，根据下表用同样的办法设置 TextBlock 和 ItemsControl 的数据绑定：

控件	属性	绑定表达式
TextBlock	Text	{Binding Title}
ItemsControl	ItemsSource	{Binding Contents}

表 5-1 各个控件的绑定表达式

至于格子右上角的箭头图标，我们需要把它的 Visibility 属性绑到 ShowNextIcon 属性，不过，它们的类型并不一样，因此我们需要使用转换器。在 Properties 面板上单击 Visibility 属性右边的小正方形，选择 Data Binding，在弹出的 Create Data Binding 对话框里选中 Data Context 选项卡，然后勾选 Use a custom path expression，并在旁边输入 ShowNextIcon，如图 5-20 所示：

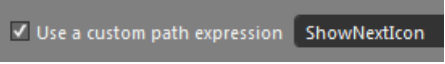


图 5-20 设置自定义路径表达式

接着，单击 Value converter 右边的省略号按钮，如图 5-21 所示：

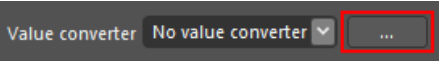


图 5-21 设置值转换器

然后在弹出的 Add Value Converter 对话框里选择 Coding4Fun.Phone.Controls.Converters 下面的 BooleanToVisibilityConverter，如图 5-22 所示：

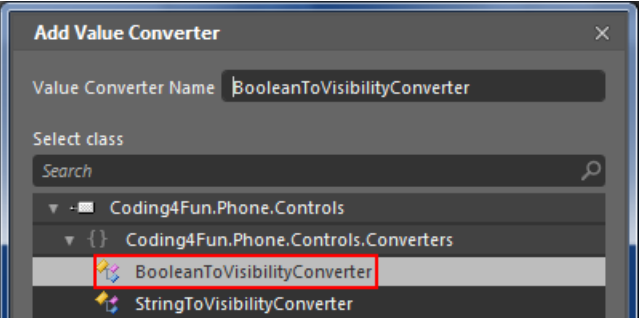


图 5-22 选择 BooleanToVisibilityConverter

好了之后单击 OK 关闭对话框。

### 5.4.2 设置命令绑定

接下来，我们将会把命令对象绑定到相关控件上。首先是格子的单击操作，每个格子最外层是一个 Grid，它没有 Click 事件，因此我选择了 `MouseLeftButtonUp` 事件来代替，我希望把它关联到 `TileViewModelBase` 的 `TileCommand` 属性，怎么关联？还记得[上节课](#)介绍的 `EventToCommand` 吗？现在又轮到它出场了！

从 Assets 面板上把 `EventToCommand` 拖到 Grid 上，此时，你的 Objects and Timeline 面板将会变成这样：

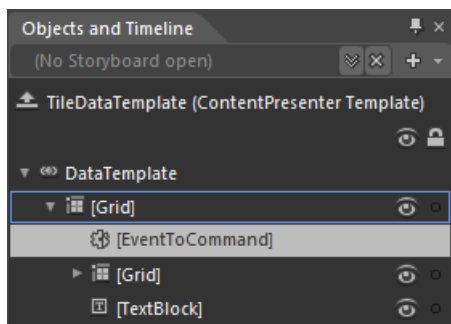


图 5-23 在 Objects and Timeline 面板上查看模板结构

确保 `EventToCommand` 处于选中状态，在 Properties 面板上把 `EventName` 属性设为 `MouseLeftButtonUp`，并把 `Command` 属性的绑定表达式设为 “`{Binding TileCommand}`”。好了之后退出模板编辑状态，用同样的办法给 `RoundButton` 控件添加 `EventToCommand`，然后把 `Command` 属性的绑定表达式设为 “`{Binding RefreshTilesCommand}`”。当我们把 `EventToCommand` 拖到一个控件上时，`EventName` 属性会被自动设为目标控件的默认事件，因为 Click 事件刚好是 `RoundButton` 控件的默认事件，所以我们无需对这个属性做任何修改。

### 5.4.3 测试应用

终于可以看效果了，哎呀，我等这个时候等到花儿也谢了，事不宜迟了，按 F5 吧：



图 5-24 主页上的课程信息

从上图可以看到，今天是星期天，没课，明天是星期一，有三节课。

单击“明天课程”将会打开课程表，并显示星期一的课程，如图 5-25 所示：



图 5-25 星期一的课程

到了星期一早上，如果我们在所有课程开始之前打开应用，主页上的格子就会变成这样：



图 5-26 第二天的课程信息

过了一会，在第一节课开始之后刷新一下，主页上的格子就会变成这样：



图 5-27 刷新之后的课程信息

单击“当前课程”将会打开 Course Hub，并显示当前课程的相关信息，包括课程概况、今天笔记和今天作业，如图 5-28 所示：



图 5-28 Course Hub 页面

到了下午放学之后再打开应用，主页上的格子就会变成这样：





图 5-29 放学后的课程信息

非常好！不过，有一个小小的地方还需要改进一下，目前每个格子里的内容和标题的字体大小是一样的，这意味着它们是同等重要的，但事实上用户更加关注内容而不是标题，因此我们应该增加内容的字体大小，从而更加突出内容的重要性，事实上，通过不同的字体大小区分不同内容的重要性是 Metro 的设计原则之一。嗯，这个改进的实现就留给你当课后作业吧。

## 5.5 最近的作业完成得怎样？

### 5.5.1 调查需求

通过作业本，用户可以清楚地看到每天有什么作业，哪些已经完成，哪些还没完成，但如果用户希望直观地了解每天的作业量有多少，已经完成的占多少，还没完成的又占多少，每天的作业量是否均匀、能否跟得上呢？前面那组是微观层面的问题，而后面那组是宏观层面的问题，要回答宏观层面的问题，我们需要借助统计工具，比如说，我们可以通过下面这个图表回顾过去五天的作业情况：

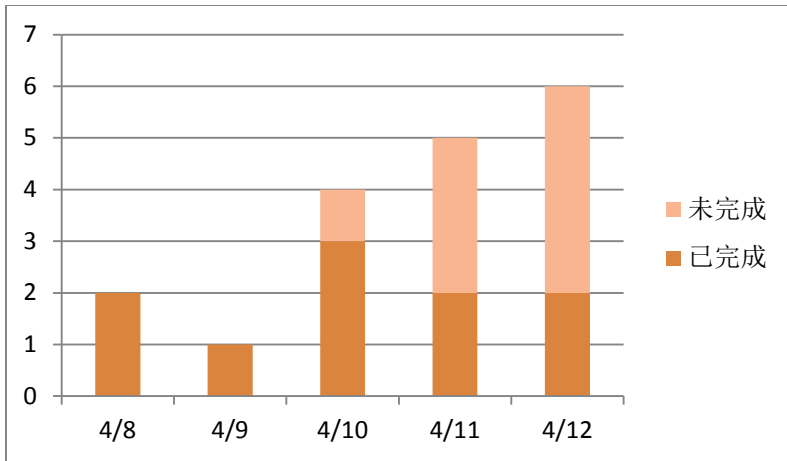


图 5-30 作业状况统计图

从上图不难看出，最近作业量增大了，还没完成的作业比重也在增大，是时候找找原因了。不过，查明这个原因不是我们的任务，实现这个功能才是我们的任务。那么，如何实现这个功能？

### 5.5.2 统计作业状况

实现这个功能需要从两个方面入手，第一，实现一个算法计算统计结果，第二，使用一个控件显示统计结果。我们先来看看第一个方面，这个算法的基本思路是先找出过去五天的作业，然后把它们分成已完成和未完成两组，接着分别对每组进行统计，计算每天的作业有多少。换句话说，我们的统计结果将会产生两组二维数据，我们可以通过两个集合来保存它们，如代码 5-24 所示：

```

private IList _finishedStatistic;
public IList FinishedStatistic
{
    get { return _finishedStatistic; }
    set
    {
        _finishedStatistic = value;
        NotifyPropertyChanged("FinishedStatistic");
    }
}

private IList _unfinishedStatistic;
public IList UnfinishedStatistic
{
    get { return _unfinishedStatistic; }
    set
    {
        _unfinishedStatistic = value;
        NotifyPropertyChanged("UnfinishedStatistic");
    }
}

```

代码 5-24 用于保存统计结果的属性

这两组数据的计算过程基本上是一样的，如图 5-31 所示：



图 5-31 计算过程

唯一的区别在于一个要筛选出已完成的作业，另一个要筛选出未完成的作业，这个“变数”可以通过方法的参数隔离开来。

根据图 5-31，我们可以创建一个 `ComputeStatistic` 方法，如代码 5-25 所示：

```
private IList ComputeStatistic(bool isCompleted)
{
    var pastFiveDays = new List<DateTime>
    {
        DateTime.Today,
        DateTime.Today.AddDays(-1),
        DateTime.Today.AddDays(-2),
        DateTime.Today.AddDays(-3),
        DateTime.Today.AddDays(-4),
    };
}
```

代码 5-25 计算统计结果 (1)

我们可以通过方法的参数指定计算哪组数据，此外，在执行任何具体计算代码之前，我们需要知道过去五天的日期是什么。

看到这里，你可能会问，每次调用这个方法都要创建这样一个集合会不会很低效？是的，你可以把它保存在一个属性里，然后在构造函数里对它进行初始化，这样每次调用这个方法时只需访问那个属性就行了，不过，如果你决定这样做，你需要额外考虑一个问题：当用户在使用过程中时间跨越了零点，应用需要重新根据新的“今天”重新创建这个集合，诚然，这种情况发生的几率不大，但若放任不管，它就会成为一个潜在的 bug。

接下来，执行图 5-31 所示的五个计算步骤，如代码 5-26 所示：

```

var q1 = from a in App.AssignmentStore.Items
        where pastFiveDays.Contains(a.StartDate)
        select a;

var q2 = from a in q1
        where a.IsCompleted == isCompleted
        select a;

var q3 = from a in q2
        group a by a.StartDate;

var q4 = from g in q3
        orderby g.Key
        select g;

var q5 = from g in q4
        select new KeyValuePair<string, int>(
            g.Key.ToString("M/d"),
            g.Count());

var statistic = q5.ToList();

```

代码 5-26 计算统计结果 (2)

看到这里，你可能会问，如果现有的作业不足五天呢，比如说一开始没有任何作业？这种情况下，我们应该为没有作业的那些天按顺序“补零”，以便横坐标可以正常显示过去五天，如代码 5-27 所示：

```

int startIndex = statistic.Count;
for (int i = startIndex; i < 5; i++)
{
    statistic.Insert(
        0, new KeyValuePair<string, int>(
            pastFiveDays[i].ToString("M/d"),
            0));
}

return statistic;
}

```

代码 5-27 计算统计结果 (3)

有了 ComputeStatistic 方法，我们就可以实际计算代码 24 那两个属性的值了，如代码 5-28 所示：

```
public void RefreshStatistic()
{
    FinishedStatistic = ComputeStatistic(true);
    UnfinishedStatistic = ComputeStatistic(false);
}
```

代码 5-28 RefreshStatistic 方法

那么，什么时候调用这个方法呢？我们知道，作业的统计数据不会因为时间的流逝而自动改变，因为作业的新建和编辑都需要用户手动完成，而主页上面也没有提供任何直接操作的途径，所以我们无需在主页上为这个统计图表提供一个单独的刷新按钮，只需在页面加载的时候刷新一下就行了，如代码 5-29 所示：

```
public MainPage()
{
    InitializeComponent();

    DataContext = App.ViewModel;

    Loaded +=
        (o, e) =>
            ((MainViewModel)DataContext).RefreshStatistic();
}
```

代码 5-29

接下来，我们将会把统计结果以统计图表的方式呈现出来。

### 5.5.3 显示统计结果

这里选择 [Mindscape Phone Elements 的 Stacked Bar Chart](#)，当然，你也可以使用其它熟悉的图标控件。[下载](#)并添加 Mindscape Phone Elements 的类库的引用，然后在主页上添加一个 Panorama 项，并把标题设为“作业”，如图 5-32 所示：

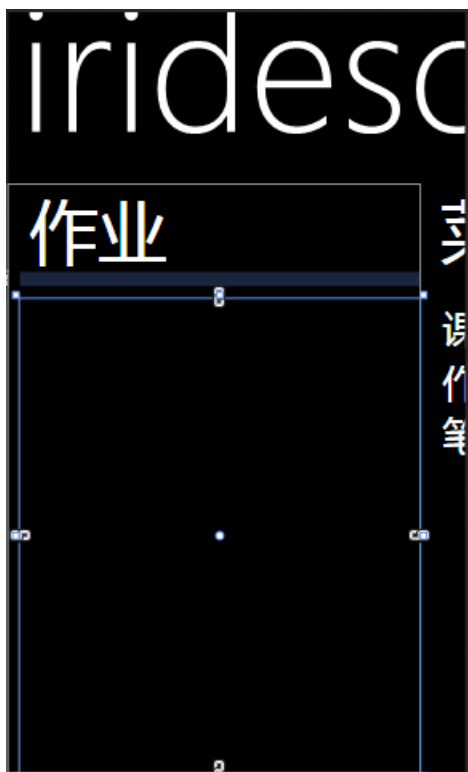


图 5-32 主页上的作业板块

接着，从 Assets 面板上把 Chart 控件拖到 Grid 里，并使之充满整个 Grid，如图 5-33 所示：

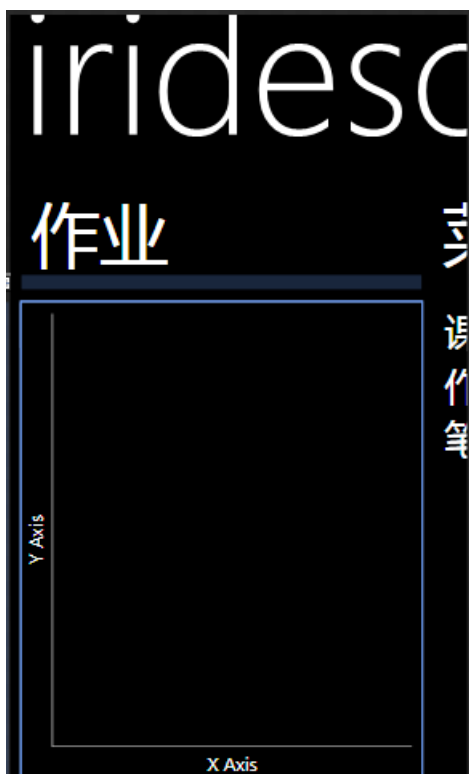


图 5-33 添加图表控件

我们的统计图表包含两组统计数据，因此我们需要在 Chart 控件里添加两个 StackedBarSeries 控件，此时，你的 Objects and Timeline 面板应该是这样的：

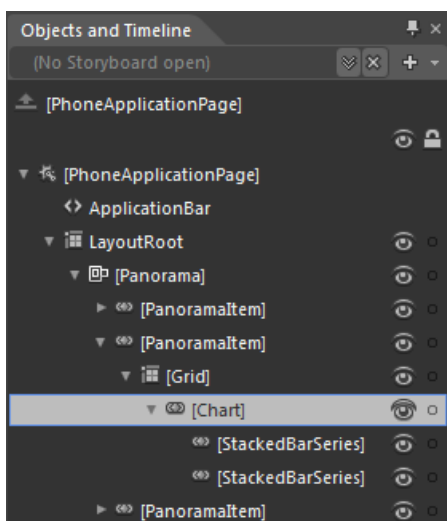


图 5-34 在 Objects and Timeline 面板上查看页面结构

这两个 StackedBarSeries 控件将会分别绑到 FinishedStatistic 和 UnfinishedStatistic 两个属性。怎么绑呢？选中第一个 StackedBarSeries 控件，在 Properties 面板上按照下表设置相关属性的绑定表达式/值：



属性	绑定表达式
ItemsSource	{Binding FinishedStatistic}
XBinding	{Binding Key}
YBinding	{Binding Value}
SeriesBrush	#FFDB843D

表 5-2 图表控件相关属性的绑定表达式

需要注意的是，ItemsSource 属性的类型是 IList，这正是为什么我把代码 5-24 的两个属性的类型声明为 IList，当然，你也可以把它们声明为 List<KeyValuePair<string, int>>。至于第二个 StackedBarSeries 控件，你只需把 ItemsSource 的绑定表达式设为 “{Binding UnfinishedStatistic}”，并为 SeriesBrush 属性换一种颜色就行了。

5.5.4 测试应用

现在，按 F5 看看效果：

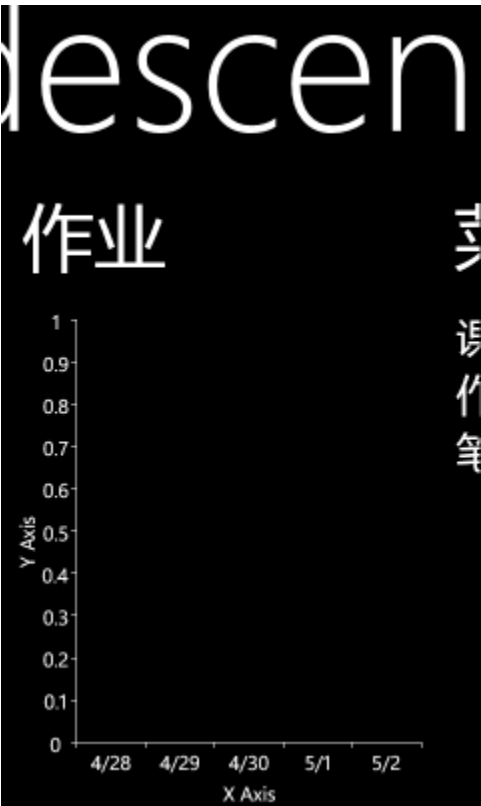


图 5-35 空的图表

进入作业本添加一些作业，然后回到主页看看：

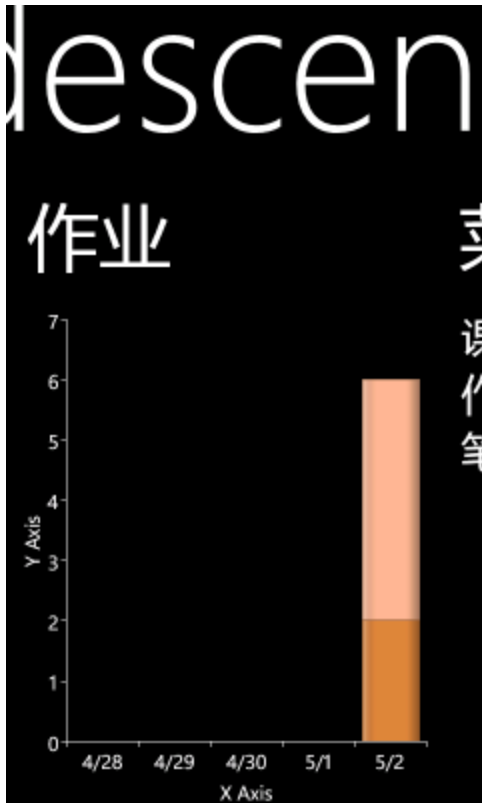


图 5-36 今天的统计结果

如果作业本里保存了过去五天的作业数据，那么统计图表将会显示成这样：

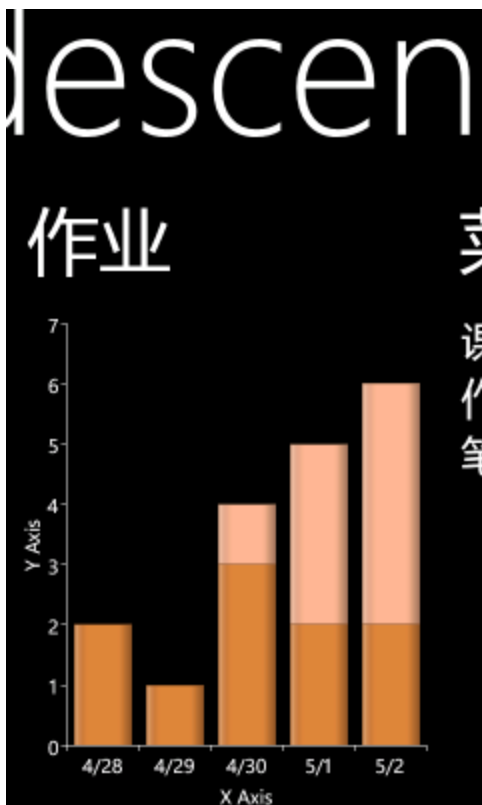


图 5-37 过去五天的统计结果

目前，我们是硬性规定显示过去五天的统计结果，这个设定显然无法满足所有用户，比如说，有些用户希望查看过去七天甚至过去十天的统计结果，这个时候可以考虑在应用设置里给出选项。如果你决定给予用户这个选择权，那么你就需要考虑在保证统计图表清晰可辨的情况下最多能够显示多少天的统计结果，你需要给定一个选择上限，同时向用户解释这样做的原因。

## 5.6 哪些内容是重点？

### 5.6.1 调查需求

教育的一个重要方面是不断强调，这主要表现为重要的内容经常重复出现。当我们的笔记本记满了笔记，并且通过标签做好分类，我们很自然就会问，哪些标签的出现频率最高，因为这些拥有这些标签的笔记通常都是课堂重点/考试要点。这个时候又轮到统计工具出场了，我们可以通过饼图列出七个出现频率最高的标签，剩下的统一归入“其它”类别，像这样：

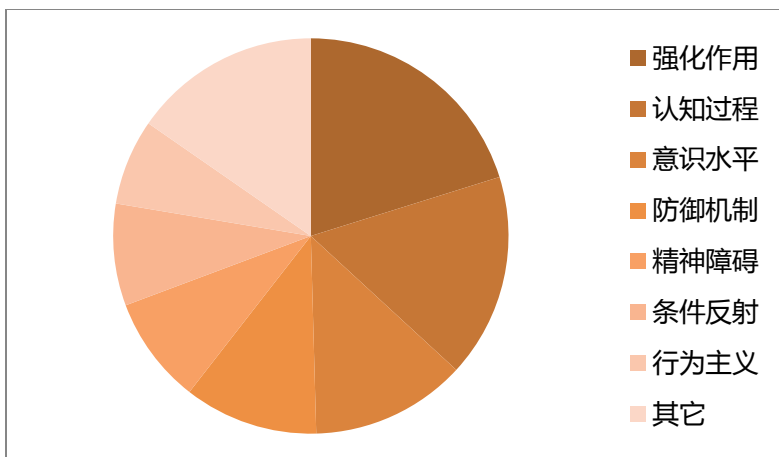


图 5-38 笔记状况统计图

为什么是七个呢？这和大脑的短期记忆容量有关，长期研究表明，这个容量是  $7 \pm 2$ 。事实上，当我们在设计 Windows Phone 应用的用户界面时，我们应该把短期记忆容量考虑进去，不要在上面放置太多东西，以免用户产生心理饱和。

### 5.6.2 创建统计图页面

事不宜迟了，我们动手吧！创建一个 Windows Phone Page，并把它命名为 TagStatisticPage.xaml，如图 5-39 所示：

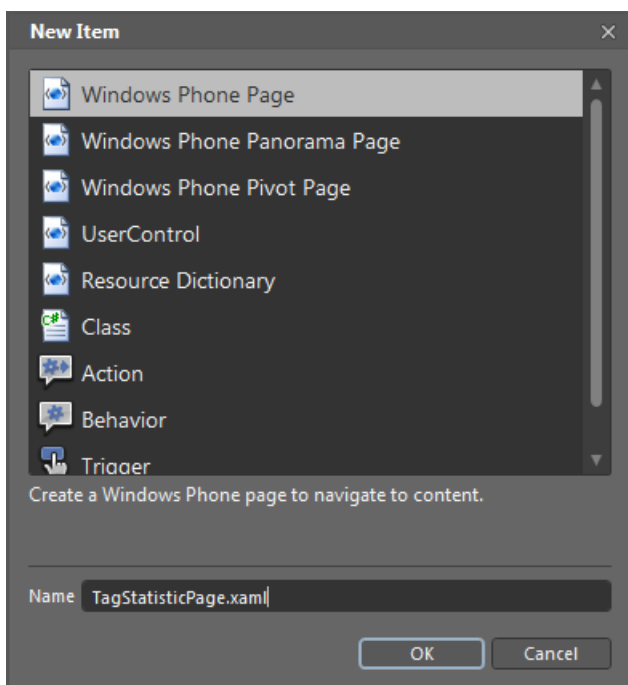


图 5-39 创建统计图页面

然后把应用程序标题和页面标题分别改为“笔记本”和“标签统计”，如图 5-40 所示：

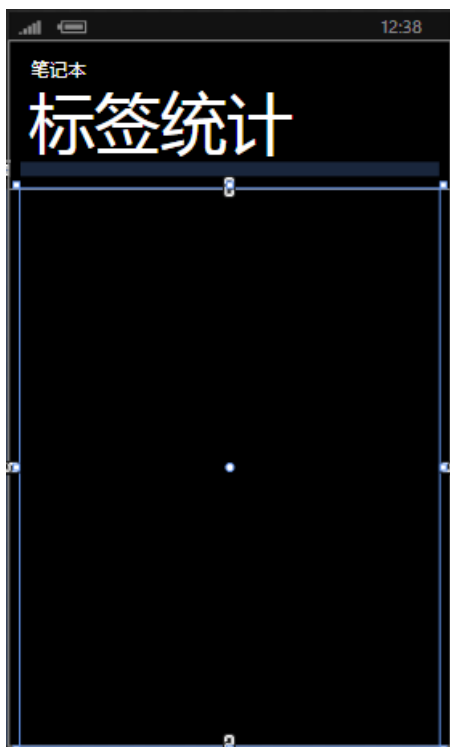


图 5-40 设置程序标题和页面标题

接着，从 Assets 面板上把 PieChart 控件拖到页面的 ContentPanel 里，并使之充满整个 ContentPanel，如图 5-41 所示：

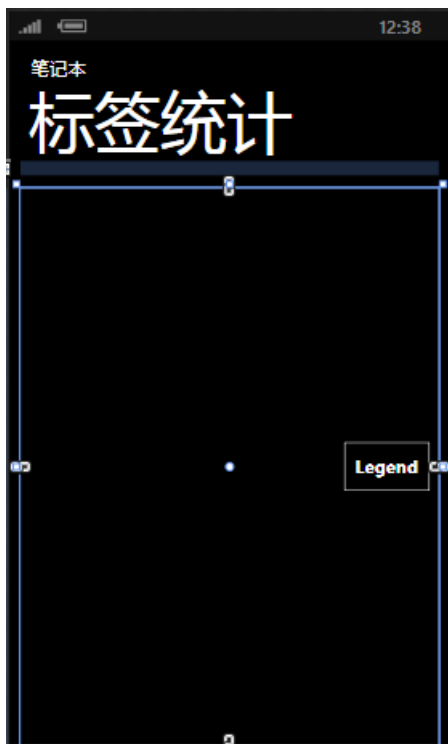


图 5-41 添加图表控件

我们的饼图包含了一组统计数据，因此我们需要在 `PieChart` 控件里添加一个 `PieSeries` 控件，此时，你的 `Objects and Timeline` 面板应该是这样的：

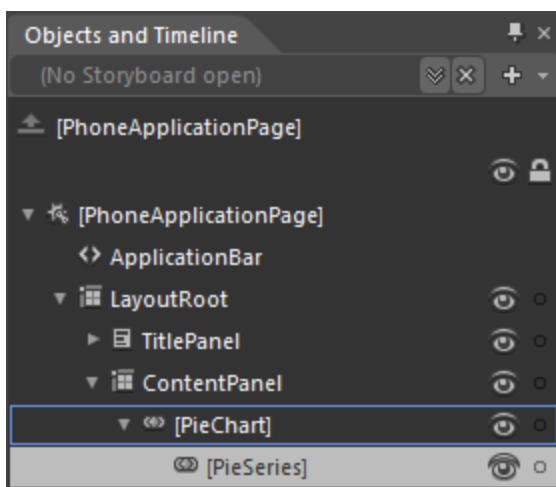


图 5-42 在 `Objects and Timeline` 面板上查看页面结构

接下来，我们需要为这个页面创建一个 `ViewModel` 类。

### 5.6.3 计算统计结果

在 `ViewModels` 文件夹里创建一个 `TagStatisticViewModel` 类，并在里面创建一个 `TagStatistic` 属性，如代码 5-30 所示：

```
public class TagStatisticViewModel
{
    public IList TagStatistic { get; private set; }
}
```

代码 5-30 TagStatisticViewModel 类

我们将会在构造函数里初始化这个属性，由于标签统计是针对每个课程而不是所有课程来做的，因此我们需要通过构造函数的参数传递课程名称，如代码 5-31 所示：

```
public TagStatisticViewModel(string courseName)
{
}
}
```

代码 5-31 TagStatisticViewModel 类的构造函数

这个属性的计算过程并不复杂，如图 5-43 所示：



图 5-43 计算过程

前五个计算步骤可以通过 LINQ 轻松实现，如代码 5-32 所示：

```

var q1 = from n in App.NoteStore.Items
        where n.CourseName == courseName
        select n;

var q2 = from n in q1
        where !String.IsNullOrEmpty(n.Tags)
        select n;

var q3 = from n in q2
        from t in n.Tags.Split(',')
        select t.Trim();

var q4 = from t in q3
        where !String.IsNullOrEmpty(t)
        select t;

var q5 = from t in q4
        group t by t into g
        orderby g.Count() descending
        select new KeyValuePair<string, int>(
            g.Key, g.Count());

```

代码 5-32 统计笔记

至于最后一步，我们需要分情况处理，如果统计结果包含的分组在七个或者以内，那么我们只需直接显示统计结果；如果超过七个，那么我们需要把剩余的分组聚合到“其它”分类：

```

var statistic = new List<KeyValuePair<string, int>>();
statistic.AddRange(q5.Take(7));
if (q5.Count() > 7)
{
    statistic.Add(
        new KeyValuePair<string, int>(
            "其它",
            q5.Skip(7).Select(p => p.Value).Sum()));
}

TagStatistic = statistic;

```

代码 5-33 把超出部分归入“其它”类别

创建好 ViewModel 类之后，我们就可以把它关联到 TagStatisticPage 页了。



5.6.4 连接 ViewModel 和 View

首先，确保 PieSeries 控件处于选中状态，在 Properties 面板上按照下表设置相关属性的绑定表达式/值：

属性	绑定表达式
ItemsSource	{Binding TagStatistic}
DataBinding	{Binding Value}
TitleBinding	{Binding Key}

表 5-3 图表控件相关属性的绑定表达式

接着，打开 TagStatisticPage.xaml.cs 文件，在里面创建一个 OnNavigatedTo 方法，如代码 5-34 所示：

```
protected override void OnNavigatedTo(NavigationEventArgs e)
{
    base.OnNavigatedTo(e);

    DataContext = new TagStatisticViewModel(
        NavigationContext.QueryString["coursename"]);
}
```

代码 5-34 重写 OnNavigatedTo 方法

这里假设查询字符串里面包含了一个名为 coursename 的参数。

最后，我们需要在 NoteBookPage 页里添加一个 Application Bar 菜单项，用于打开 TagStatisticPage 页，如图 5-44 所示：



图 5-44 添加标签统计菜单项

并在它的 Click 事件处理程序里添加相关代码，如代码 5-35 所示：

```

private void ApplicationBarTagStatisticMenuItem_Click(
    object sender, System.EventArgs e)
{
    var book = (NoteBookViewModel)DataContext;
    if (book.NoteLists.Count > 0)
    {
        var courseName = book.SelectedNoteList.Header;
        NavigationService.Navigate(
            new Uri(
                "/TagStatisticPage.xaml?" +
                "courseName=" + courseName,
                UriKind.RelativeOrAbsolute));
    }
    else
    {
        MessageBox.Show("课程表还没创建。");
    }
}

```

代码 5-35 标签统计菜单项的 Click 事件处理程序

值得注意的是，如果课程表没有任何课程，那么我们就应该打开 TagStatisticPage 页了，不过，我们需要通过消息框向用户说明一下。

### 5.6.5 测试应用

现在，按 F5，然后进入笔记本，打开 TagStatisticPage 页，如图 5-45 所示：

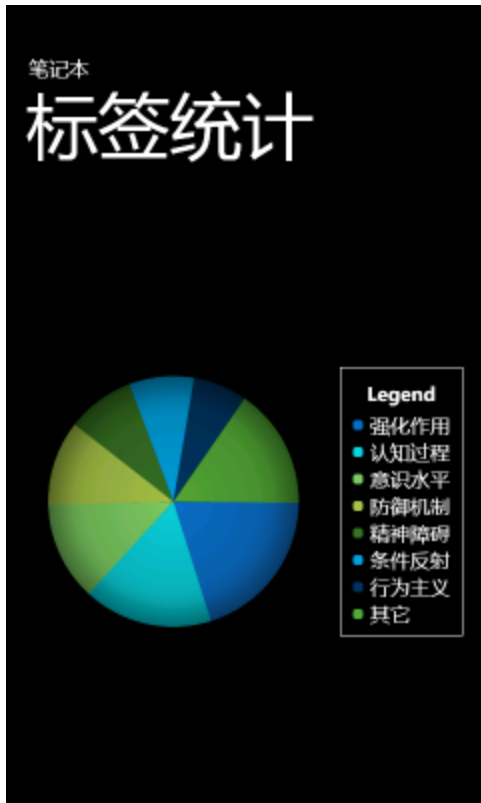


图 5-45 标签统计页面

嗯，很好！不过，有几个地方需要调整一下：

1. 我希望统计图表上方能够显示课程名称；
2. 目前统计图表是从三点位置顺时针展开的，我希望改成从零点位置顺时针展开；
3. 我希望统计图表的配色方案和图 38 的一样。

第一个问题很容易解决，我们只需在 `TagStatisticViewModel` 类里创建一个 `Title` 属性，并在构造函数里把它初始化为课程名称，然后把它绑到 `PieChart` 控件的 `Title` 属性就行了。第二个问题也很好解决，我们只需把 `PieSeries` 控件的 `StartAngle` 属性的值改为 270 就行了。至于第三个问题，我们需要设置 `PieSeries` 控件的 `Brushes` 属性，按顺序添加相应颜色的 `SolidColorBrush` 对象，你可以直接写 XAML，也可以通过 Blend 的 `Brush Collection Editor` 来设置，如图 5-46 所示：

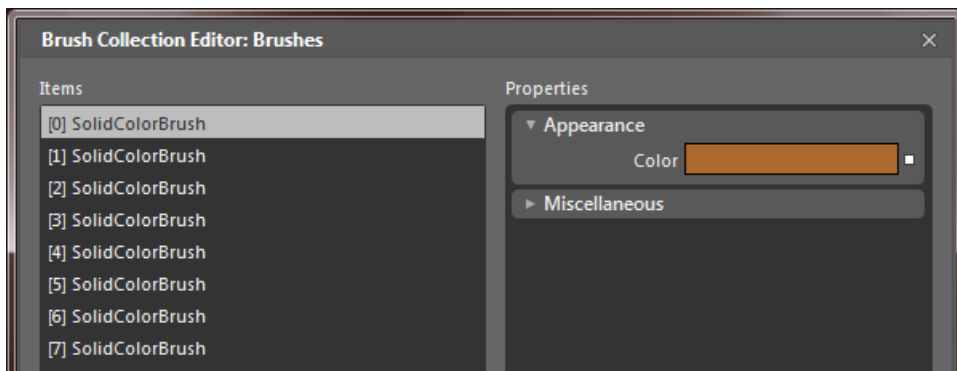


图 5-46 Brush Collection Editor

我采用的是第二种做法，因为它允许我通过取色器直接从图 5-38 上获取颜色，当然，如果你有自己的配色方案，并且知道那些颜色的 HTML 代码，那么直接写 XAML 也很方便。

改好之后重新运行，然后打开 TagStatisticPage 页：

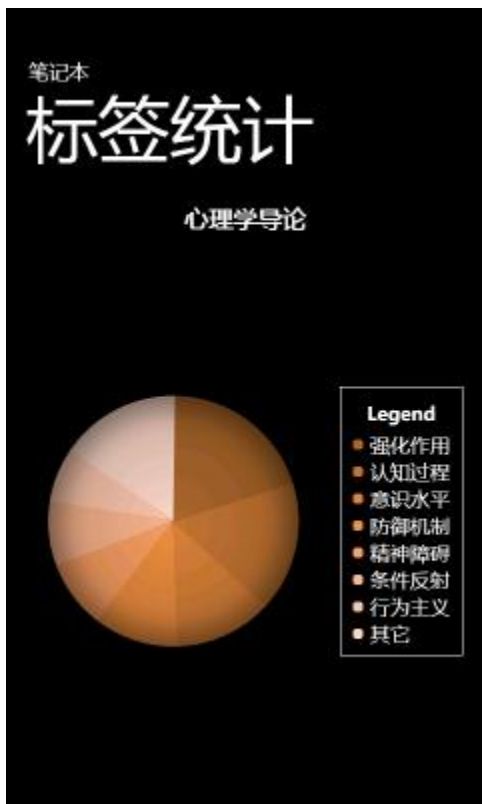


图 5-47

对比图 5-45，图 5-47 给人一种相容有序的感觉，这在很大程度上得益于配色方案的选择，这些颜色都是基于同一色相的，因此能够给人一种相容的感觉，同时，这些颜色根据明度（明亮程度）从小到大顺时针排列，因此给人一种有序的感觉，此外，不同明度能使颜色产生不同的“重量”，明度较低的颜色给人感觉较重，因此用在出现频率较高的标

签上，而明度较高的颜色给人感觉较轻，因此用在出现频率较低的标签上，这样用户可以直观地感知内容的重要程度。

一个好的应用除了能够帮助用户解决相关的问题，还应该照顾到用户的无意识需求，这不但对于用户来说是一件好事，对于开发者来说也是一件好事，最自然的操作是凭直觉的操作，最自然的选择是符合直觉的选择，这是一种无需经过思考的条件反射，当用户的无意识需求被照顾到了，他们也会不经意地选择你的应用，他们甚至很难解释清楚为什么就要选择你的应用。

5.7 下课了……



what's next?

try "mango" →