现代操作系统应用开发实验报告

姓名：陈衍斌

学号：16340042

实验名称：Lab2

一、参考资料

https://blog.csdn.net/lindexi_gd/article/details/51242913

https://www.1keydata.com/cn/sql/

https://developer.microsoft.com/zh-cn/windows/apps

https://github.com/lizy14/Tsinghua-UWP

https://blog.csdn.net/NoMasp/article/details/47079387

二、实验步骤

- Week4 实验要求：
  - （Mainpage.xaml.cs）在 MainPage 中点击 checkbox 出现横线，输入数据（选择图片），挂起并关闭程序，重新启动时，程序显示在 Mainpage 界面，并且点击的 checkbox 与对应横线，数据与图片都存在。
  - （Newpage.xaml.cs）在 NewPage 中输入数据（或选择图片），挂起并关闭程序，重新启动时，程序显示在 Newpage 界面，数据与图片都存在。

（Bonus：保存图片状态）

（挂起并关闭，再运行后仍能回复下图中所有控件的内容和状态）

首先我们先明确一下 Suspending 的定义：

当用户挂起某个应用时，window 将暂停该应用。只要应用中不存在任何处于活动状态的执行，该应用将在出现锁屏界面时暂停。

当应用被挂起时，UWP 程序将会调用 Application.Suspending 事件。App.xaml.cs 中的 OnSuspending 事件将会提供处理程序来保存应用程序的状态。

OnSuspending 需要修改的地方如下：

```
private void OnSuspending(object sender, SuspendingEventArgs e)
{
    issuspend = true;
    var deferral = e.SuspendingOperation.GetDeferral();
    //TODO: 保存应用程序状态并停止任何后台活动
    Frame frame = Window.Current.Content as Frame;
    ApplicationData.Current.LocalSettings.Values["NavigationState"] = frame.GetNavigationState();
    deferral.Complete();
}
```

当应用程序在挂起时需要被激活时，它需要加载它保存的应用程序数据，恢复终止之前相同的状态。当用户重新加载挂起应用时，OnLaunched 方法会提供还原应用程序数据的处理程序。

Onlaunched 需要修改的地方如下：

```
rootFrame.NavigationFailed += OnNavigationFailed;

if (e.PreviousExecutionState == ApplicationExecutionState.Terminated)
{
    //TODO: 从之前挂起的应用程序加载状态
    if (ApplicationData.Current.LocalSettings.Values.ContainsKey("NavigationState"))
    {
        rootFrame.SetNavigationState((string)ApplicationData.Current.LocalSettings.Values["NavigationState"
    }
}
```

再来说一下 OnNavigatedFrom 和 OnNavigatedTo 两个函数：

OnNavigatedFrom：当界面不再是框架的活动界面时，这个方法将会被调用，主要作用时跳转到其他页面对 MainPage 和 NewPage 的的值进行保存，这里采用键值对存储，但是由于容量的问题，不能储存图片形式的值。因此通过下面这句将 file 储存起来：

```
ApplicationData.Current.LocalSettings.Values["Image"] = StorageApplicationPermissions.FutureAccessList.Add(file);
```

Date 项储存起来并不需要先转换成 string 再转换成 DateTimeOffset 类，可以如下存储：

```
NavigationCacheMode = NavigationCacheMode.Enabled;
```

代码如下:

```
protected override void OnNavigatedFrom(NavigationEventArgs e)
{
    bool suspending = ((App)App.Current).issuspend;
    if (suspending)
    {
        var composite = new ApplicationDataCompositeValue();
        composite["Title"] = TitleBlock.Text;
        composite["Details"] = DetailBlock.Text;
        composite["Date"] = Date.Date;
        composite["Visible"] = ((App)App.Current).ViewModel.AllItems[0].Completed;
        ApplicationData.Current.LocalSettings.Values["MainPage"] = composite;
    }
    DataTransferManager.GetForCurrentView().DataRequested -= OnShareDataRequested;
}
```

OnNavigatedTo：在挂起结束重新返回界面后，跳转到 MainPage 或 NewPage 并还原。

代码如下:

```
protected override async void OnNavigatedTo(NavigationEventArgs e)
{

    if (e.NavigationMode == NavigationMode.New)
    {
        ApplicationData.Current.LocalSettings.Values.Remove("MainPage");
        ApplicationData.Current.LocalSettings.Values["Image"] = null;
    }

    if (ApplicationData.Current.LocalSettings.Values["Image"] != null)
    {
        StorageFile tempimg;
        tempimg = await StorageApplicationPermissions.FutureAccessList.GetFileAsync((string)ApplicationData.Current.LocalSettings.Values["Image"]);
        IRandomAccessStream ir = await tempimg.OpenAsync(FileAccessMode.Read);
        BitmapImage bi = new BitmapImage();
        await bi.SetSourceAsync(ir);
        NewImage.Source = bi;
        ApplicationData.Current.LocalSettings.Values["Image"] = null;
    }
```

具体效果如下:

比如 MainPage，假设挂起之前的状态是这样的:

在“生命周期事件”处选择挂起并关闭，然后点击“本地计算机“重新打开，可以看到程序状态如下：

这样说明关闭之前的状态都被保存了下来。

NewPage 与 MainPage 基本一样，而且在实验课验收时也已经检查过了，这里就不再截图和累述了。

- Week5 实验要求：

  1：制作磁贴

  - 要求使用标准的处理 XML DOM 方式创建动态磁贴
  - 要求采用 Adaptive Tile （覆盖至少 small、medium、wide）
  - 实现效果：要求每添加一条项目，磁贴能进行更新，并且更新的内容循环展示（1-2-3-4-5-1-2-3-4……）
  - （Bonus：为磁贴添加背景图片）

2：App-to-App communication

在 MenuFlyoutItem 中增加 Share 选项，点击后相应条目能以邮件方式进行共享（不要求动态共享图片）。

首先，实现磁贴的话，我们先要选择磁贴想要显示的图片。双击项目列表中的 Package.appxmanifest 文件，然后将想要显示的图片放到相应的磁贴图标里即可。注意，

所选择的图片必须符合不同规格的磁贴所规定的分辨率（为了选择符合分辨率的想要的图片，还花了不少时间）：

首先我们先自定义磁贴的形式。这里使用 Notifications Visualizer 设计磁贴形式，即需要一个 xml 文件来编写磁贴。这里先介绍一下一些元素：

Binding：制定每一个磁贴都需要最外层包含一层 binding 元素，它是磁贴的框架。像这次实验需要至少小、中、宽三种不同大小的磁贴，因此 xml 文件中需要包含三层 binding 元素。

Image：存放图片，其中 placement 属性可以决定图片所放的位置。如 placement="background"，这就表示在磁贴的背景中应用这张图片，src 决定图片的位置，写上图片相应的路径即可。

text：文本内容，与绑定后的数据有关。其中 hint-align 属性决定标题和内容的位置。

有了这些，就可以先创建磁贴的形式：

```
<binding template="TileSmall" >
  <image placement="background" src="Assets/timg3.jpg" />
  <text></text>
  <text></text>
</binding>

<binding template="TileMedium">
  <image  placement="background" src="Assets/timg3.jpg"  />
  <group>
    <subgroup>
      <text  hint-style="caption" hint-align="center" >我是标题</text>
      <text  hint-style="captionsubtle" hint-align="center" hint-wrap="true
    </subgroup>
  </group>
</binding>
```

要想对磁贴的内容进行实时更新的话，先要访问 xml 文件中的所有 text 标签：

```
XmlDocument document = new XmlDocument();
document.LoadXml(System.IO.File.ReadAllText("Tile.xml"));
XmlNodeList textElements = document.GetElementsByTagName("text");
```

接下来编写 update()函数，引用定义的 xml 文件，通过对 DOM 元素操作的方法，取到所有 text 的标签，将更新的条目的 title 和 discription 作为文本节点插入 DOM 树中。在 create 按钮事件处理方法中调用 update()函数，就可以实现磁贴的同步更新。

实现磁贴内容的滚动循环的话，调用通知队列即可。

```
TileUpdateManager.CreateTileUpdaterForApplication().Update(tileNotification);
TileUpdateManager.CreateTileUpdaterForApplication().EnableNotificationQueue(true);
```

在建立循环队列前，先要将之前的内容清空：

```
TileUpdateManager.CreateTileUpdaterForApplication().Clear();
```

实现的效果如下：

小磁贴：

中磁贴：

宽磁贴：

接下来实现 App to App communication：

首先，在 MenuFlyoutItem 中增加 Share 选项：

```
<AppBarButton Grid.Column="3" Icon="Setting" IsCompact="True" VerticalAlignment="center">
    <AppBarButton.Flyout>
        <MenuFlyout>
            <MenuFlyoutItem Text="Share" Click="MenuFlyoutItem_Click"/>
        </MenuFlyout>
    </AppBarButton.Flyout>
</AppBarButton>
```

先实现已邮件形式共享。

在 NavigationedFrom 函数结尾加一句话：

```
DataTransferManager.GetForCurrentView().DataRequested -= OnShareDataRequested;
```

在 NavigationedTo 函数结尾加一句话：

```
DataTransferManager.GetForCurrentView().DataRequested += OnShareDataRequested;
```

这样就实现邮件共享。

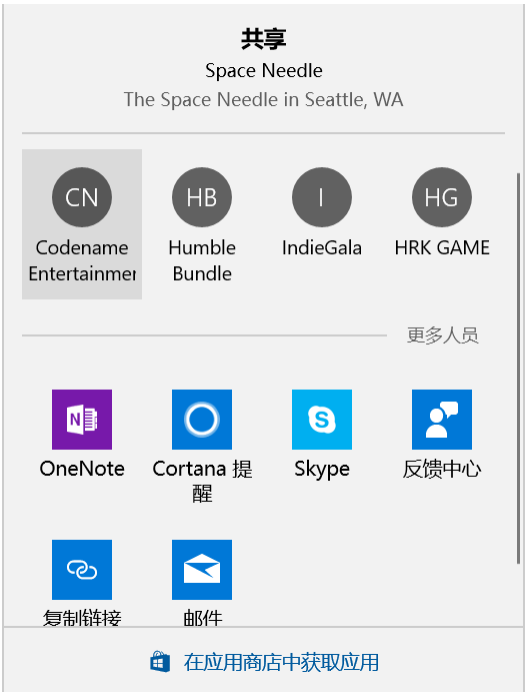
点击 share 时，会调用 MenuFlyoutItem_Click 函数，实现如下：

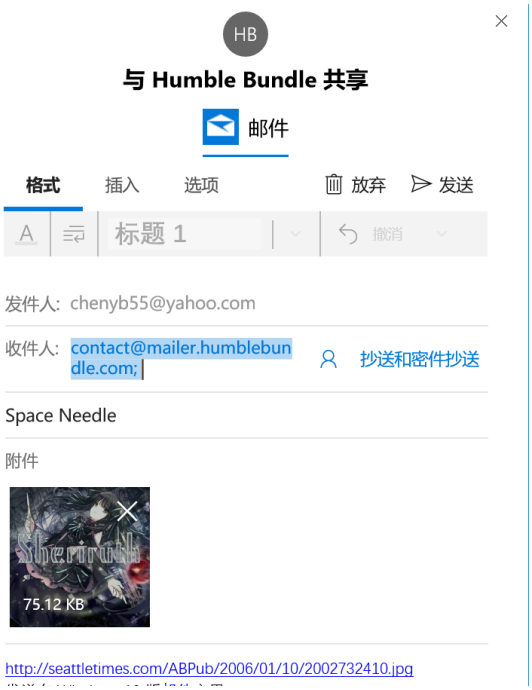
```
private void MenuFlyoutItem_Click(object sender, RoutedEventArgs e)
{
    var s = sender as FrameworkElement;
    var item = (Models.TodoItem)s.DataContext;
    App.title = item.title;
    App.details = item.description;
    App.path = item.path;
    DataTransferManager.ShowShareUI();
}
```

动态共享图片的话，ppt 里面给出方法，我就基本照着用了，代码如下：

```
async void OnShareDataRequested(DataTransferManager sender, DataRequestedEventArgs args)
{
    var dp = args.Request.Data;
    var deferral = args.Request.GetDeferral();
    var photoFile = await StorageFile.GetFileFromApplicationUriAsync(new Uri("ms-appx:///Assets/timg3.jpg"));
    dp.Properties.Title = "Space Needle";
    dp.Properties.Description = "The Space Needle in Seattle, WA";
    dp.SetStorageItems(new List<StorageFile> { photoFile });
    dp.SetWebLink(new Uri("http://seattletimes.com/ABPub/2006/01/10/2002732410.jpg"));
    deferral.Complete();
}
```

实验的最终效果如下：





- Week6 实验要求：

  SQLite 数据库本地存储：

  - 实现 todo 表项的增、删、改、查；并且能保存及恢复应用状态。
  - 需要保存：title，description，complete， date（年月日即可），image（Bonus 项）

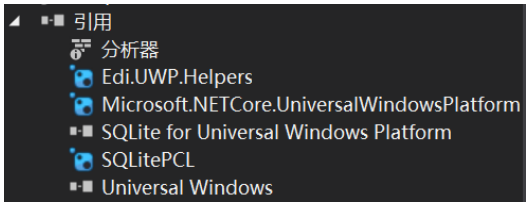
- 查询时为模糊查询，如下图，查询“现”即可显示日期为 title 或 description 或 date 中含有“现”的 item（查询到的 item 用字符串表示 title+description+date。若有多条，则每行一个 item）。

在进行这次作业之前，先需要做一些准备工作。

SQLite：在拓展里搜索 SQLite，安装 SQLite for Universal Windows Platform。虽然 VS 会提示不兼容，但是不影响实际使用。安装好后，在解决方案资源管理器中，右键“引用 “，选择”添加引用 “，选择 SQLite for Universal Windows Platform 添加即可。

SQLitePCL：右键“引用”选择“管理 NuGet 程序包”，搜索 SQLitePCL 然后安装。

准备工作做完后，引用里面的视图如下：



这表示准备工作已经完成。

接下来先设计搜索框的布局，代码如下：

```
<StackPanel Orientation="Horizontal" HorizontalAlignment="Right">
    <TextBox Name="SearchBox"  HorizontalAlignment="Right" Width="200" Height="30" Background="Gray" Opacity="0.2" PlaceholderText="Search"/>
    <AppBarButton Icon="Find" Margin="0,10,0,0" Click="AppBarButton_Click_1"/>
</StackPanel>
```

样式如下：

AppBarButton_Click_1 函数处理查询的结果，代码如下：

```
private void AppBarButton_Click_1(object sender, RoutedEventArgs e)
{
    String result = String.Empty;
    StringBuilder DataQuery = new StringBuilder("%%");
    DataQuery.Insert(1,SearchBox.Text);
    var db = App.conn;
    using (var statement = db.Prepare(App.SQL_SEARCH)) {
        statement.Bind(1,DataQuery.ToString());
        statement.Bind(2,DataQuery.ToString());
        statement.Bind(3,DataQuery.ToString());
        while (SQLiteResult.ROW == statement.Step()) {
            result += statement[0].ToString() + " ";
            result += statement[1].ToString() + " ";
            result += statement[2].ToString() + "\n";
        }
    }
    if (result == String.Empty)
    {
        var box1 = new MessageDialog("Not find").ShowAsync();
    }
    else
    {
        var box2 = new MessageDialog(result).ShowAsync();
    }
}
```

Z

StringBuilder 的作用：一般我们需要进行字符串的连接时，一般都是通过 String 对象相加实现。但是这种方式达到目的的效率比较低，每执行一次都会创建一个 String 对象。这样即耗时又浪费空间。而 StringBuilder 是一个可修改字符的缓冲器，当不直接进行字符串操作而应用 StringBuilder 时，与每次重复生成 String 对象相比，可以获得更好的性能。

```
StorageFolder localFolder = ApplicationData.Current.LocalFolder;
String path = localFolder.Path;
BitmapDecoder decoder = await BitmapDecoder.CreateAsync(fileStream);
SoftwareBitmap softwareBitmap = await decoder.GetSoftwareBitmapAsync();
String Item_id = null;
if (ViewModel.AllItems.Count == 0)
```

ppt 里提到了 LocalFolder 和 RoamingFolder，这里讲一下它们的作用：

LocalFolder：此存储位置是备份的，共用的，会自动备份，不会在系统维护时删除，在重新安装的时候可以还原。

RomingFolder：跨设备，会在多台设备之间同步。一些用户习惯的设置可以放在这里。

下面这个是数据库连接的全局变量：

```
static public SQLiteConnection conn;
```

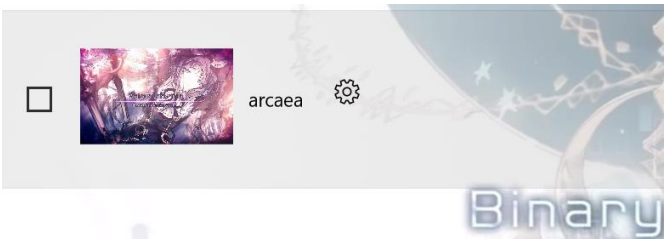
下面代码是数据库的构建，在不同的地方，每个变量都会随着数据的增删改来变化。例如，利用 create 按钮时，先判断是修改了原对象还是创建了新对象；如果是 create，则调用 SQL_INSERT；如果是 update，则调用 SQL_UPDATE：

```
public static String DB_NAME = "Todos.db";
public static String TABLE_NAME = "TodoItems";
public static String SQL_CREATE_TABLE = "CREATE TABLE IF NOT EXISTS " + TABLE_NAME
    + "(Id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,Title VARCHAR(100),Details VARCHAR(150),DueDate VARCHAR(150),
public static String SQL_QUERY_VALUE = "SELECT Title,Details,DueDate,Complete,Path FROM " + TABLE_NAME;
public static String SQL_INSERT = "INSERT INTO " + TABLE_NAME + "(Title,Details,DueDate,Complete,Path) VALUES(?,?,?
public static String SQL_UPDATE = "UPDATE " + TABLE_NAME + " SET Title = ?,Details = ?,DueDate = ?,Path=? WHERE Tit
public static String SQL_DELETE = "DELETE FROM " + TABLE_NAME + " WHERE Title = ? AND Details = ? AND DueDate = ?";
public static String SQL_SEARCH = "SELECT Title,Details,DueDate FROM " + TABLE_NAME + " WHERE Title LIKE ? OR Detai
public static String SQL_UPDATE_COMPLETE = "UPDATE " + TABLE_NAME + " SET Complete = ? WHERE Title = ?";
```

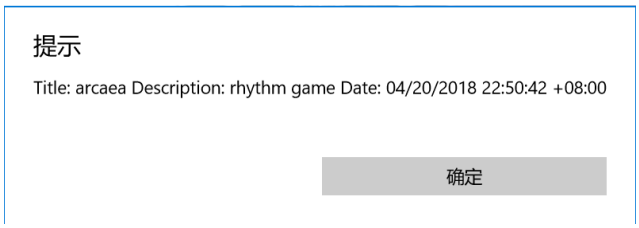
加载 DataBase：

```
    private void LoadDB() {
        conn = new SQLiteConnection(DB_NAME);
        using (var statement = conn.Prepare(SQL_CREATE_TABLE)) {
            statement.Step();
        }
    }
```

实验结果如下：



利用搜索框搜索 arcaea，得到的结果如下：



三、亮点与改进

emmm...主要还是按照作业要求来完成实验的，亮点并不多。

非要说的话，磁贴的封面背景和滚动时的封面背景是不同的图片。因为封面背景是在 Package.appxmanifest 里面设置的，而滚动的图片背景是在 xml 文件里，因此会有所不同。不过这个实现也不是什么困难的事。

改进的话，感觉单独不太好讲，与问题一起说吧。

四、遇到的问题

1. week3 的一些实现方式不满足 week4 的设计，挂起需要保存 MainPage 和 NewPage 的状态，而 week3 传递了自定义参数到 NewPage 中，无法进行序列化。解决方法是直接将变量写到 App.xaml.cs 中，这样无需在页面间传值。

2. 读取.xml 文件时无法找到文件：经实验发现是因为有些<Image>标签失效，删掉即可。

3. 上一次打开程序新建的通知仍然在磁贴上：这个纯粹是我脑抽，加上清空队列的代码即可。

4. File Management & SQLite Database 的图片处理：查找了网上一些方法，一种是转换为二进制，不过百度谷歌后依然不会。还有一种方法是存图片的名字，用字符串拼接的方法得到新的 BitmapImage。我觉得这是一种不错的方法，不过初始化时，老是报错（貌似是多线程冲突），也不知道为什么。最后还是采用最蠢的方法——存地址。（其实就

是没有实现，因为选择图片时根本得不到 BitmapImage 的 UriSource)。这个还有待解决。

五、思考与总结

相比与最开始三周的实验，这次的实验难度应该又有了一定程度的提升。不过因为有了前三周来说，做起来反而没那么困难。毕竟，最开始完全是在零基础的情况下学习，而 UWP 的课程节奏我觉得挺快的，每次课后都要花大量的功夫去研究，因为很多功能需要一些方法实现，但是很多方法并不清楚，所以需要花费很多功夫去查询。不过经过前三周的“洗礼”，现在也稍微有一点经验了。遇到问题也学会通过官方文档来解决。

尽管这六周涉及很多的内容并且基本实现了，但是我对这些知识掌握得可能并不牢固。毕竟，有些时候是对着一些模板照着套过去的。而且前面的知识，比如自适应 UI，现在可能也稍微忘记了一些了（再做一遍我可能还要查资料)。所以在学习新的知识时，要记得巩固以前学到的知识，没事时自己能动手运用一下，这样可以更加得心应手。

总之，通过这两次的实验，我学到了不少东西，受益匪浅。