


# 现代操作系统应用开发实验报告

姓名: 徐伟元 学号: 16340261 Blog: [My Blog](#) 

## 实验名称 : Lab2 MyList

### 参考资料

- [Microsoft Documents](#)
- [SQLite](#)
- CSDN
- MSDN
- CNBLOGS
- Stack Overflow

### 基础实验步骤

- Week4
  1. 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。
  2. 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。
- Week5
  1. 制作磁贴
    - 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
  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）
- Week6

SQLite 数据库本地存储：

  1. 实现 todo 表项的增、删、改、查；并且能保存及恢复应用状态。
  2. 需要保存：title，description，complete，date（年月日即可），image（Bonus项）
  3. 查询时为模糊查询，如：查询“现”即可显示日期为 title 或 description 或 date 中含有“现”的 item 。

### 关键基础代码

1. 挂起状态(Suspending state)。

首先，在 App.xmal.cs 代码中，已经有了关于挂起状态的函数。这是 windows 自己的考虑，所有应用都可以具备挂起状态，而且对其有需求。我们要做的事情，就是为其注册挂起事件，然后实现这个挂起事件。

1. 注册挂起事件

```
public App() {
    this.InitializeComponent();
    this.Suspending += OnSuspending;
}
```

## 2. 挂起事件

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

## 3. 在页面启动判断之前是否为挂起状态

```
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")) {
        string temp = (string)ApplicationData.Current.LocalSettings.Values["NavigationState"]
        rootFrame.SetNavigationState(temp);
    }
}
```

在 App 代码中为所有页面注册了挂起事件，接下来就是为不同页面实现各自的挂起需求，即存储哪些数据。首先，OnNavigatedFrom 是从当前页面跳转，那么这就是我们要进行判断是否进入挂起状态。

```
protected override void OnNavigatedFrom(NavigationEventArgs e) {
    bool suspending = ((App)App.Current).IsSuspending;
    if (suspending) {
        // Save volatile state in case we get terminated later on
        var composite = new ApplicationDataCompositeValue {
            ["title"] = Title.Text,
            ["description"] = Description.Text,
            ["date"] = Date.Date
            //...
        };
        ApplicationData.Current.LocalSettings.Values["Mainpage"] = composite;
    }
}
```

然后，OnNavigatedTo 是跳转到当前页面，在这里，我们要进行一个判断，当前页面是否是从挂起状态恢复，以此决定是否恢复数据。

```
protected async override void OnNavigatedTo(NavigationEventArgs e) {
    Frame rootFrame = Window.Current.Content as Frame;
    SystemNavigationManager.GetForCurrentView().AppViewBackButtonVisibility =
        AppViewBackButtonVisibility.Collapsed;
    if (e.NavigationMode == NavigationMode.New) {
        ApplicationData.Current.LocalSettings.Values.Remove("Mainpage");
    } else {
        // Try to restore state if any, in case we were terminated
        if (ApplicationData.Current.LocalSettings.Values.ContainsKey("Mainpage")) {
            var composite = ApplicationData.Current.LocalSettings.Values["Mainpage"] as
                ApplicationDataCompositeValue;
            Title.Text = (string)composite["title"];
            Description.Text = (string)composite["description"];
            Date.Date = (DateTimeOffset)composite["date"];
            //...
            ApplicationData.Current.LocalSettings.Values.Remove("Mainpage");
        }
    }
}
```

至此，简单的挂起关闭恢复数据就做好了。

## 2. 磁贴Tile

磁贴是 Windows 10 提出的新概念，应用可以在开始菜单拥有一个磁贴，用来展示一些概括内容或提供一个快速启动入口。主要利用两件事：一个 xml 文档构造磁贴，一个 cs 文件指定展示内容。对于磁贴的 xml 文档，在 **Notifications Visualizer** 软件有模板，这里我们就直接复制粘贴啦。

直接看代码还是很容易看懂的，不同的 template 绑定一个磁贴样式，然后其中的内容可以被 cs 代码更改。对于背景图和 item 图片可以看宽屏状态下的代码。我们可以指定图片为背景图，然后对于 item 图片，可以在 group 中划分一个 subgroup 来存放。

```
<?xml version="1.0" encoding="utf-8" ?>
<tile>
  <visual branding="nameAndLogo" displayName="MyList">

    <binding template="TileSmall" >
      <!--more-->
    </binding>

    <binding template="TileMedium">
      <!--more-->
    </binding>

    <binding template="TileWide" displayName="MyList">
      <image placement="background" src="Assets/background.jpg" />
      <group>
        <subgroup>
          <image hint-align="stretch" src="Assets/photo.jpg" />
        </subgroup>
        <subgroup>
          <text hint-style="caption" hint-align="center">我是标题</text>
          <text hint-style="captionsubtle" hint-align="center" hint-wrap="true">我是内容</text>
        </subgroup>
      </group>
    </binding>

    <binding template="TileLarge" displayName="MyList">
      <!--more-->
    </binding>

  </visual>
</tile>
```

cs 代码如下。这里处理的方式是很不优雅的。学过 web 的同学，肯定知道，这相当于代码之间的越界。其实可以考虑使用绑定的方法来实现。

```
private void Tile_Click(object sender, RoutedEventArgs e) {
    XmlDocument document = new XmlDocument();
    document.LoadXml(System.IO.File.ReadAllText("Tile.xml"));
    XmlNodeList textElements = document.GetElementsByTagName("text");
    var count = ViewModel.AllItems.Count;
    if (count == 0) return;
    textElements[0].InnerText = ViewModel.AllItems[count - 1].title;
    textElements[2].InnerText = ViewModel.AllItems[count - 1].title;
    textElements[3].InnerText = ViewModel.AllItems[count - 1].detail;
    textElements[4].InnerText = ViewModel.AllItems[count - 1].title;
    textElements[5].InnerText = ViewModel.AllItems[count - 1].detail;
    textElements[6].InnerText = ViewModel.AllItems[count - 1].title;
    textElements[7].InnerText = ViewModel.AllItems[count - 1].detail;
    var tileNotification = new TileNotification(document);
    TileUpdateManager.CreateTileUpdaterForApplication().Update(tileNotification);
    TileUpdateManager.CreateTileUpdaterForApplication().EnableNotificationQueue(true);
}
```

### 3. 分享Share

share功能，实现将我们的 item 通过 mail 发送出去。这个功能的实现还是很简单的。代码如下：

```
private void ShareClick(object sender, RoutedEventArgs e) {
    dynamic temp = e.OriginalSource;
    ViewModel.SelectedItem = (ListItem)(temp.DataContext);
    DataManager.ShowShareUI();
}

void OnShareDataRequested(DataTransferManager sender, DataRequestedEventArgs args) {
    DataRequest request = args.Request;
    request.Data.Properties.Title = ViewModel.SelectedItem.Title;
    request.Data.Properties.Description = ViewModel.SelectedItem.detail;
    request.Data.SetBitmap(RandomAccessStreamReference.CreateFromUri(new Uri("path")));
    DataRequestDeferral deferral = request.GetDeferral();
    request.Data.SetText(ViewModel.SelectedItem.detail);
    deferral.Complete();
}
```

### 4. 数据库SQLite

数据库重头戏。首先，在项目的解决方案中的 NuGet 包中，安装 SQLitePCL 和 Microsoft.Data.Sqlite 包。注意为项目中的所有小项目添加对他们的引用。

这里，我们采用数据库操作和我们的 App 分离的构架。这样的目的是为了便于管理，可以更快定位出错的地方。我们并不采用[官方文档](#)提供的 SQLite 使用方法，但使用它对于数据库层代码的管理方式，即在总项目中增添一个 DataAccess 小项目专门进行数据库操作（详细架构方法查看[官方文档](#)）。

采取这个架构方案，我明显感觉到代码分离的好处。在写 DataAccess 时只需要关注数据库本身，对于 App 的状况，是不需要关心的；在 App 中对数据库进行请求的时候，我知道有这个接口，至于怎么实现的，也不需要关心。

下面是 DataAccess 的代码。对于接口调用代码则不进行展示，具体见总项目代码。

```
public static class DataAccess {
    private static string create = @"CREATE TABLE IF NOT EXISTS MyList(Id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
                                                                            Title VARCHAR(140), " +
                                                                            "Detail VARCHAR(140), " +
                                                                            "Date VARCHAR(140), " +
                                                                            "Completed INTEGER, " +
                                                                            "Image BLOB)";

    private static string insert = @"INSERT INTO MyList(Id, Title, Detail, Date, Completed, Image) " +
                                    "VALUES (?, ?, ?, ?, ?, ?)";

    private static string delete = @"DELETE FROM MyList " +
                                    "WHERE Id = ?";

    private static string update = @"UPDATE MyList " +
                                    "SET Title = ?, Detail = ?, Date = ?, Completed = ?, Image = ? " +
                                    "WHERE Id = ?";

    private static string vagueQuery = @"SELECT * " +
                                        "FROM MyList " +
                                        "WHERE Title LIKE ? OR Detail LIKE ? OR Date LIKE ?";

    private static string titleQuery = @"SELECT * " +
                                        "FROM MyList " +
                                        "WHERE Title LIKE ?";

    private static string dateQuery = @"SELECT * " +
                                        "FROM MyList " +
                                        "WHERE Date LIKE ?";

    private static string seperate = new string('-', 80);

    public static SQLiteConnection connection;

    public static void InitializeDatabase() {
        connection = new SQLiteConnection("MyList.db");
        using (var statement = connection.Prepare(create)) {
            statement.Step();
        }
    }
}
```

```

public static void AddData(long id, string title, string detail, string date,
                           int completed, byte[] image) {
    using (var statement = connection.Prepare(insert)) {
        statement.Bind(1, id);
        statement.Bind(2, title);
        statement.Bind(3, detail);
        statement.Bind(4, date);
        statement.Bind(5, completed);
        statement.Bind(6, image);
        statement.Step();
    }
}

public static void DeleteData(long id) {
    using (var statement = connection.Prepare(delete)) {
        statement.Bind(1, id);
        statement.Step();
    }
}

public static void UpdateDate(long id, string title, string detail, string date,
                              int completed, byte[] image) {
    using (var statement = connection.Prepare(update)) {
        statement.Bind(1, title);
        statement.Bind(2, detail);
        statement.Bind(3, date);
        statement.Bind(4, completed);
        statement.Bind(5, image);
        statement.Bind(6, id);
        statement.Step();
    }
}

public static string VagueQueryData(string info) {
    string result = "";
    using (var statement = connection.Prepare(vagueQuery)) {
        statement.Bind(1, "%" + info + "%");
        statement.Bind(2, "%" + info + "%");
        statement.Bind(3, "%" + info + "%");
        while (statement.Step() == SQLiteResult.ROW) {
            string IsCompleted = (Int64)statement[4] == 1 ? "Yes" : "No";
            result += separate + "\n";
            result += "Id\t\t: " + statement[0] + "\n";
            result += "Title\t\t: " + statement[1] + "\n";
            result += "Detail\t\t: " + statement[2] + "\n";
            result += "Date\t\t: " + statement[3] + "\n";
            result += "Completed\t: " + IsCompleted + "\n";
            result += separate + "\n";
        }
    }
    if (result == "") result += "\tNo Matched Item!\n";
    return result;
}

public static string TitleQueryData(string info) {
    string result = "";
    using (var statement = connection.Prepare(titleQuery)) {
        statement.Bind(1, "%" + info + "%");
        while (statement.Step() == SQLiteResult.ROW) {
            string IsCompleted = (Int64)statement[4] == 1 ? "Yes" : "No";
            result += separate + "\n";
            result += "Id\t\t: " + statement[0] + "\n";
            result += "Title\t\t: " + statement[1] + "\n";
            result += "Detail\t\t: " + statement[2] + "\n";
            result += "Date\t\t: " + statement[3] + "\n";
        }
    }
}

```

```

        result += "Completed\t: " + IsCompleted + "\n";
        result += separate + "\n";
    }
}
if (result == "") result += "\tNo Matched Item!\n";
return result;
}

public static string DateQueryData(string info) {
    string result = "";
    using (var statement = connection.Prepare(dateQuery)) {
        statement.Bind(1, "%" + info + "%");
        while (statement.Step() == SQLiteResult.ROW) {
            string IsCompleted = (Int64)statement[4] == 1 ? "Yes" : "No";
            result += separate + "\n";
            result += "Id\t\t: " + statement[0] + "\n";
            result += "Title\t\t: " + statement[1] + "\n";
            result += "Detail\t\t: " + statement[2] + "\n";
            result += "Date\t\t: " + statement[3] + "\n";
            result += "Completed\t: " + IsCompleted + "\n";
            result += separate + "\n";
        }
    }
    if (result == "") result += "\tNo Matched Item!\n";
    return result;
}
}

```

还有一点就是分离 SQL 语句，这个是为了复用，我们这个是小型项目，所以体会不深，但是如果在大项目中，则是一件很棒的事情。

## 亮点与改进

### 1. 图片的恢复

在进行挂起并挂壁恢复数据的过程中，最让人头疼的莫过于，图片问题。以往也一定遇到了这个问题。这终归都是 UWP 的权限问题。UWP 规定应用程序的权限仅限于其安装目录路径，所以对于任意路径的文件获取，其实很难做到。在这里，我们采用规避路径权限问题的做法，实现图片的恢复。

在 Stack Overflow 查询到 UWP 中有 FutureAccessList 和 MostRecentlyUsedList 两个记录 StorageItems 的工具。

When you add an item to such list, you obtain a token and this is what you should remember in your LocalSettings (for example).  
Then you can reuse such token to access the file/folder

From [Stack Overflow](#)

下面是实现代码：

- 存储图片 Token

```

private async void Select_Photo(object sender, RoutedEventArgs e) {
    FileOpenPicker picker = new FileOpenPicker();

    // Initialize the picture file type to take

    StorageFile file = await picker.PickSingleFileAsync();

    if (file != null) {
        ApplicationData.Current.LocalSettings.Values["MyToken"] =
            StorageApplicationPermissions.FutureAccessList.Add(file);
        // Load the selected picture
    }
}

```



- 恢复图片文件

```
if (ApplicationData.Current.LocalSettings.Values.ContainsKey("MyToken")) {
    if ((string)ApplicationData.Current.LocalSettings.Values["MyToken"] != "") {
        StorageFile theFile = await StorageApplicationPermissions.FutureAccessList.GetFileAsync(
            (string)ApplicationData.Current.LocalSettings.Values["MyToken"]);
        if (theFile != null) {
            // Load the saved picture
        }
    }
}
```

## 2. 视图化数据库

安装 [SQLite Expert Personal](#)，可通过 GUI 界面对 database 进行查看和操作。

## 3. 多种匹配查询

可以选择查询匹配的要求。可以进行 **日期匹配**，**标题匹配**和**模糊全匹配**

## 4. 数据库图片存储

通过 [SQLit DataType](#) 介绍可以看到，SQLite 提供了一个 BLOB 类型来存储自定义或难以定义的类型。通俗来说，BLOB 可以做到，你存入什么，我就存什么。通过这一点，我们在数据库中加入一个 BLOB 类型的 Image 字段，来存储图片。这个字段究竟存储什么呢？图片文件是不太好的，所以我们将图片转成二进制数组进行存储。对于这个字段的处理，和普通字段没有任何区别，就是传入的参数为 byte[]。

这里利用 DataReader 来将 图片流转成二进制数组。

```
private async void Select_Photo(object sender, RoutedEventArgs e) {
    // open the image file
    if (file != null) {
        /// ...
        /// Load the selected picture
        var stream = await file.OpenReadAsync();
        /// image to byte
        using (var dataRender = new DataReader(stream)) {
            var imgBytes = new byte[stream.Size];
            await dataRender.LoadAsync((uint)stream.Size);
            dataRender.ReadBytes(imgBytes);
            /// Load image file into image
            /// save bytes
            imgData = imgBytes;
        }
    }
}
```

从 byte 数组恢复图片。这里利用 MemoryStream 的二进制构造函数来构造流，然后再利用 BitmapImage 的流构造函数构造图片。

```
private async Task<BitmapImage> BytesToBitmapImage(byte[] imgByte) {
    try {
        MemoryStream stream = new MemoryStream(imgByte);
        BitmapImage bitmap = new BitmapImage();
        await bitmap.SetSourceAsync(stream.AsRandomAccessStream());
        return bitmap;
    }
    catch (ArgumentNullException ex) {
        throw ex;
    }
}
```

## 遇到的问题

- 1. checkbox 和 line 的挂起关闭恢复  
*采用死绑定解决*
- 2. 图片路径权限问题  
*采用规避解决*
- 3. SQLite 对于图片的处理  
*查阅官档和技术博客解决*

## 思考与总结

对于挂起并关闭作业要求，和别人有过讨论。对于实现恢复 checkbox 和 line 的绑定 UI，这个需求有些异议。如果从恢复应用 UI 的角度来说，这是合理的。但是，从恢复输入现场来说，这个就是多余的。 基于其挂起存储空间的限制，我们认为，挂起恢复的目的，应该是保存用户的输入，避免崩溃或其他原因导致应用挂起然后用户再回到页面的时候，其输入数据丢失。这样需要用户重新输入，体验不佳。

所以，对于我们应该存储的，就是选中 Item 的 id 和对其进行的编辑。其余数据的丢失，数据库中依旧存有，并不应该都由挂起存储空间负责。我们的数据更新和 Item 状态，应该由用户点击保存数据到数据库来存储。我们的挂起存储的数据，应该只是输入数据，保证良好的用户体验。

基础任务完成的同时，会适当拓展一些功能，以此加深对 UWP 开发的认识和理解。对于开发过程中遇到的问题，很多时候依赖官网和技术博客，因此，个人的感悟和心路历程以[博客](#)的形式记录了下来，起备忘和给他人引用来使用。