**现代操作系统应用开发实验报告**

——刘亚辉 16340157

UWP第二周实验报告

1. 参考资料
2. <https://blogs.msdn.microsoft.com/tiles_and_toasts/2015/06/30/adaptive-tile-templates-schema-and-documentation/>（动态磁贴）
3. <https://docs.microsoft.com/en-us/uwp/api/windows.storage.applicationdatacompositevalue> (composite用法)
4. <https://blog.csdn.net/lindexi_gd/article/details/51242913> (XDocument生成动态磁贴、Toast)
5. <https://www.c-sharpcorner.com/article/live-tiles-in-universal-windows-apps/> (动态磁贴)
6. <https://github.com/oysteinkrog/SQLite.Net-PCL> (SQLite数据库)
7. <https://www.cnblogs.com/yanxiaodi/p/4941312.html> (SQLite.Net-PCL数据库使用)
8. <http://www.cnblogs.com/ms-uap/p/4798269.html> (SQLitePCL)
9. <http://www.cnblogs.com/h82258652/p/4802076.html> (SQLite.Net-PCL)
10. <http://www.sqliteexpert.com/> (SQLiteExpert可视化工具)
11. PPT、课件
12. 实验步骤

**·**需求及流程：

1. week-4：实现挂起并关闭时保存当前的状态

当应用启动时会调用onLaunched方法，并向该方法传递LaunchActivatedEventArgs参数，从LaunchActivatedEventArgs. PreviousExecutionState获取应用之前的状态，这将返回ApplicationExecutionState对象。

当应用暂停时，系统会尝试将应用及其数据保留在内存中。 但是，如果系统没有资源将应用保留在内存中，它将终止应用。 应用不会收到它们被终止的通知，所以只能**将应用数据保存在 OnSuspension 事件处理程序中**，或者**在 EnteredBackground 处理程序中以异步方式保存。**

当应用确定它在终止后被激活时，它应该加载它保存的应用程序数据，以使应用处于与其终止之前相同的状态。 当用户**切换回已终止的暂停应用时**，该应用应该在其 **OnLaunched 方法中还原其应用程序数据。** 当终止应用时系统不会通知应用，因此**在暂停应用之前，应用必须保存其应用程序数据**并**释放独占资源和文件句柄，并且当在终止后又激活应用时还原这些内容。**

1. week-5：

实现动态磁贴循环显示以及共享条目的功能。

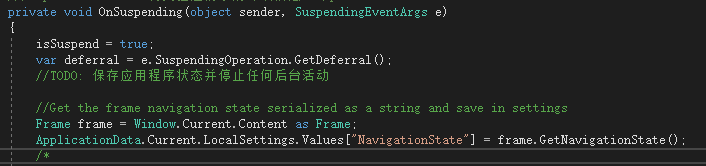
1. week-6：

利用SQLite数据库实现数据持久化。

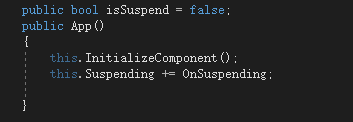
**·**详细实现过程：

1. 实现挂起关闭并恢复数据的功能：

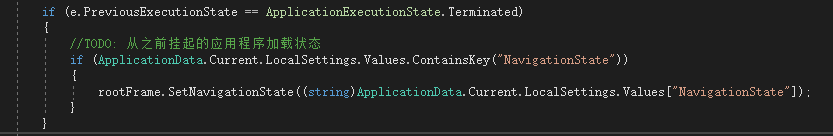
首先需要在App.xaml.cs中构建onSuspending函数，表示挂起过程中需要进程的一些操作，保存当前的状态：



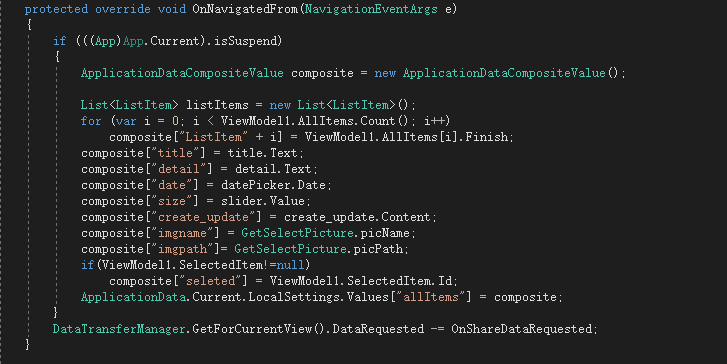
并进行事件委托：



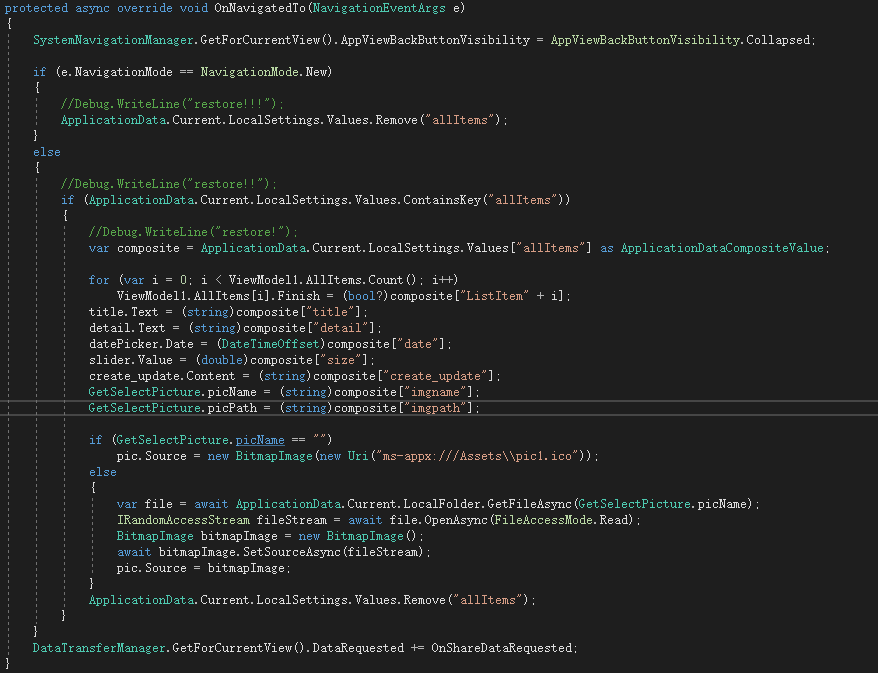
并在重新打开，OnLaunched函数中利用NavigationState实现加载到之前的几面：



之后利用MainPage和NewPage中的OnNavigatedFrom和OnNavigatedTo两个函数，来进行保存数据与恢复数据：



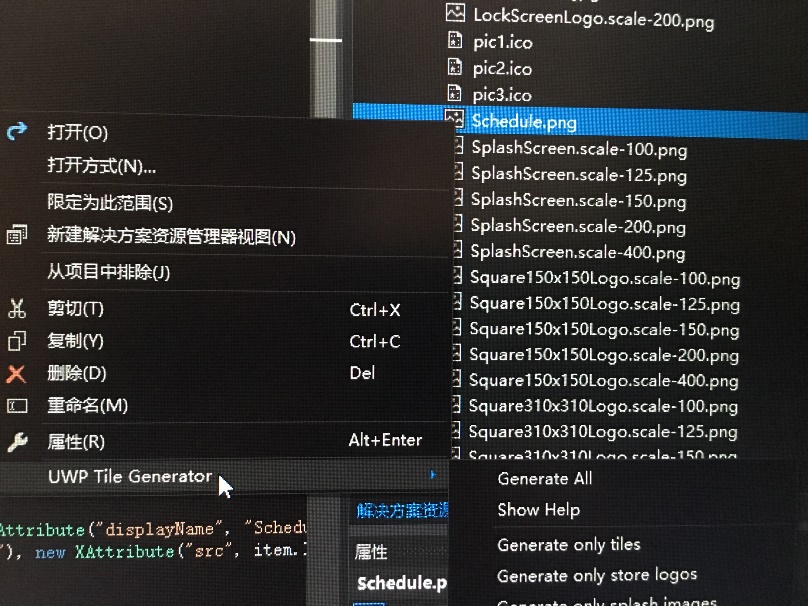
上图是MainPage中的挂起时保存数据的函数，通过ApplicationDataCompositeValue对象实例和键值对的方法来保存数据。其中图片的保存只是保存了path和name。（A**pplicationDataCompositeValue**：一组必须按原子方式序列化和反序列化的相关应用设置。 使用**复合设置**可轻松处理相互依赖的设置的原子更新。 系统会在并发访问和漫游时确保复合设置的完整性。 复合设置针对少量数据进行了优化，如果将它们用于大型数据集，性能可能很差。在此只有少量数据并未考虑性能情况。）



上图是MainPage中的恢复数据的函数，在该函数中，判断是否是挂起后再次打开，然后给之前界面上的元素进行赋值，图片利用ApplicationData.Current.LocalFolder.GetFileAsync函数来得到。在读取或编写本地应用数据前，必须**检索本地应用数据存储。** 若要检索本地应用数据存储，需要使用 **ApplicationData.Current.LocalSettings.Value[KEY] 方法**获取应用**作为 ApplicationDataContainer 对象（在此处为ApplicationDataCompositeValue对象）的本地设置。 使用 ApplicationData.Current.LocalFolder 属性**可以获取**StorageFolder 对象中的文件。**使用**ApplicationData.Current.LocalCacheFolder**属性来获取保存不包括在备份和还原中的文件的本地应用数据存储中的文件夹。

NewPage中处理方法也是如此。

1. 动态磁贴及共享条目写法：

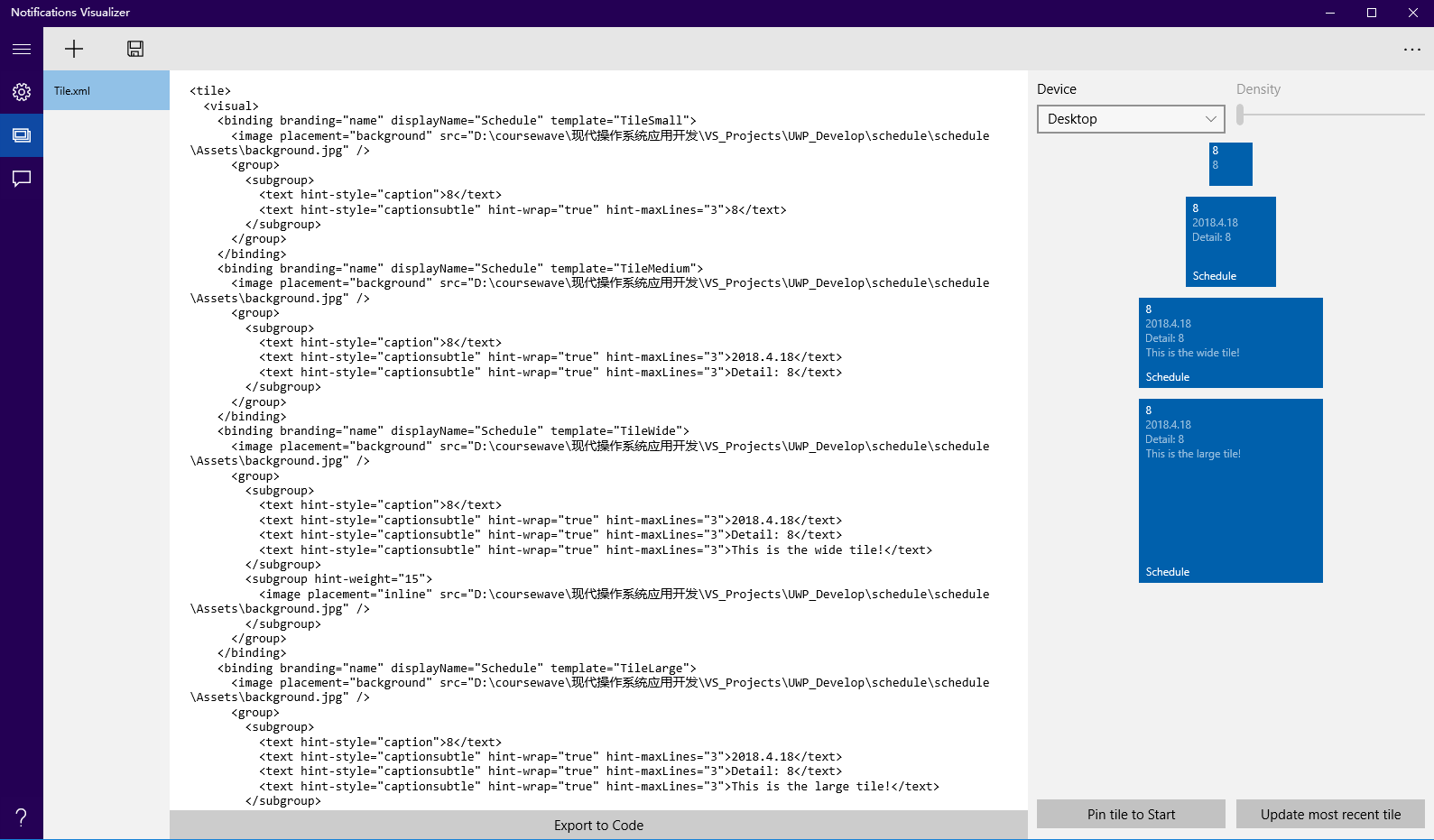
首先，利用下图UWPTileGenerator这款插件来生成所需的各种磁贴的图片，效果如下：  
  
 选择图片之后点击Generate All即可生成所需的各种图片。

之后开始书写生成动态磁贴的代码，这里我使用XDocument来实现xml的生成：

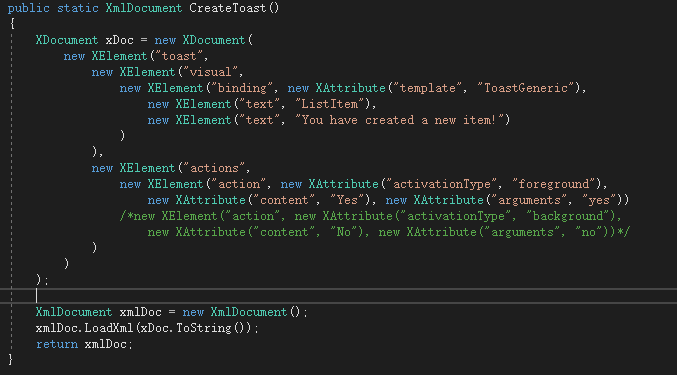


以上代码实现的是小磁贴的样式设计，通过XAttribute，XElement等嵌套使用，即可像xml文件那样利用C#实现后台动态磁贴的生成。其中visual表示可视化的元素设计，binding表示绑定该磁贴类型（TileSmall, TileMidium,TileWide或者TileLarge）；branding表示该磁贴的现实的标识，有三个属性（name, logo, nameAndlogo）；displayName可以向branding那样动态更改磁贴左下角的标识；hint-wrap表示当前文本是否适应当前磁贴宽度自动换行，true表示自动换行，false表示不换行，超过磁贴宽度不显示。

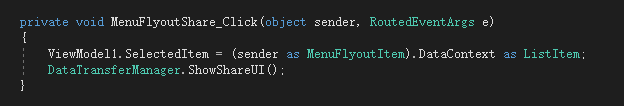
另外，也可以是用可视化工具Notifications Visualizer来帮助实现C#动态磁贴代码的生成：在左侧xml文本框内书写磁贴xml代码，样式效果如右图所示，之后点击Export to Code即可。



之后，实现Toast，方法与Tile相同，也可以利用可视化工具来生成C#代码。



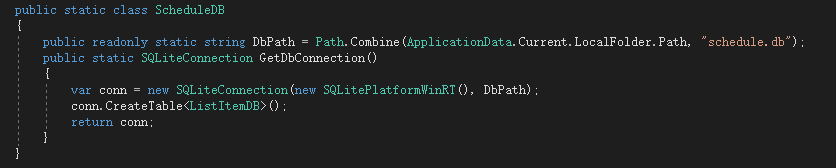
最后，实现条目共享功能：先在MenuFlyout中新加一个元素，作为share点击的按钮，并注册点击事件：



DataTransferManager是一个静态类，用来初始化共享条目，先调用Get GetForCurrentView方法来得到当前活动界面，然后给DataRequested对象增加事件监听器。在离开和进入MainPage界面中分别取消事件委托或者增加事件委托。DataTransferManager.GetForCurrentView().DataRequested -=/+= OnShareDataRequested;

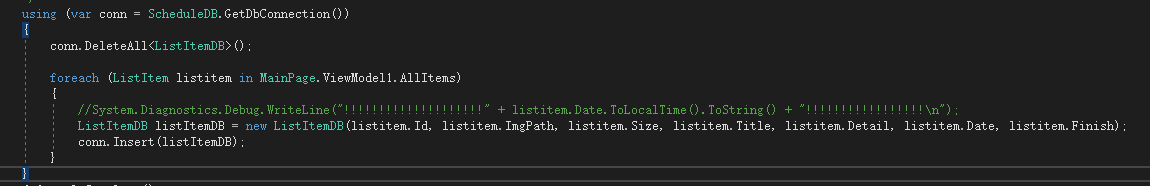
1. SQLite数据库使用：

首先，安装SQLite引用依赖和SQLite.Net-PCL API的包之后。新建ScheduleDB.cs文件，添加ScheduleDB类，在类内部生成新的数据库。

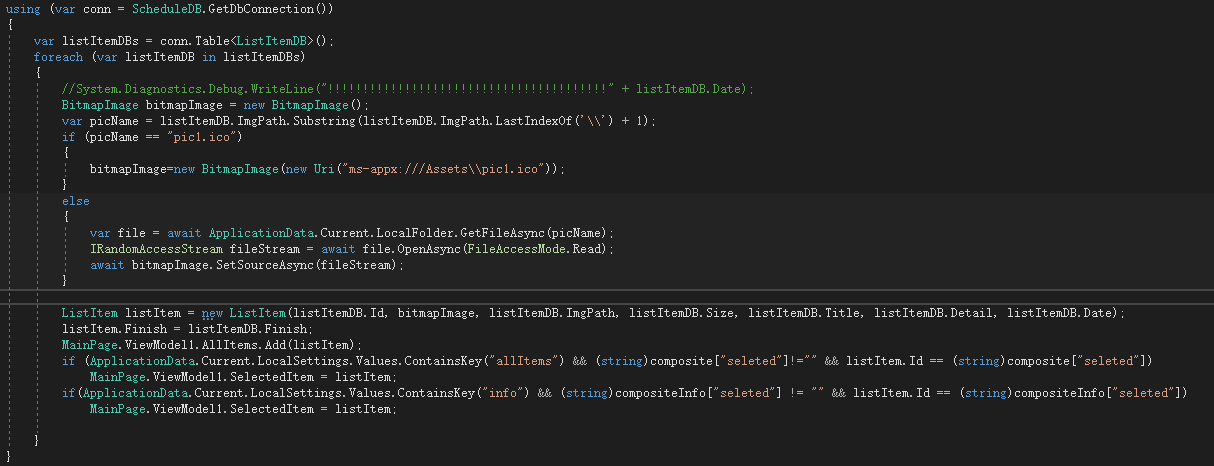


每次使用数据库进行增删改查操作是，调用SQLiteConnection方法，得到一个数据库的连接变量，上述方法，只有在第一次调用时才会生成一个新的表，之后调用便会忽略创建表的操作。在进行数据持久化的设计中，我在打开应用的时候，连接数据库来实现数据恢复，在关闭应用的时候实现数据保存。

数据保存如下：首先清空所有的数据，并且将当前ListItem中的数据在插入一遍：



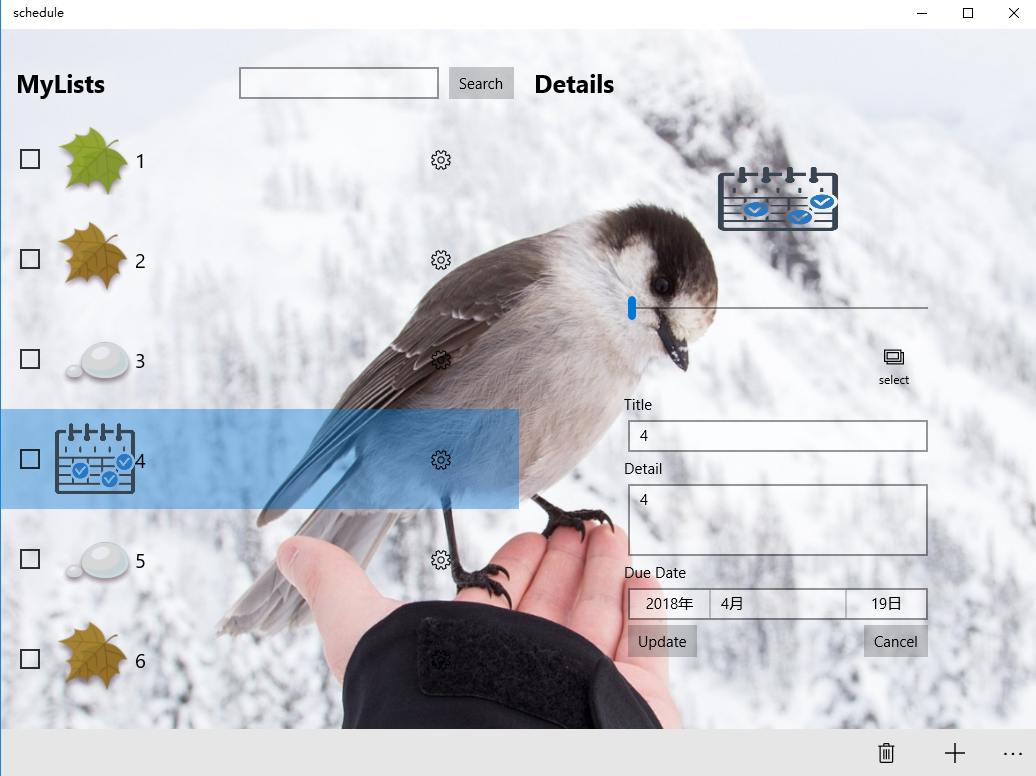
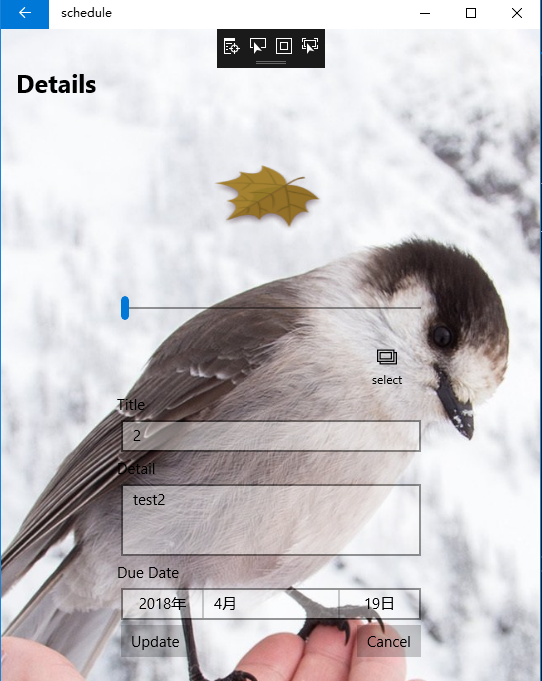
数据读取如下：using的语法相当于一个块作用域，当前的打括号内的变量在执行完之后即被销毁：



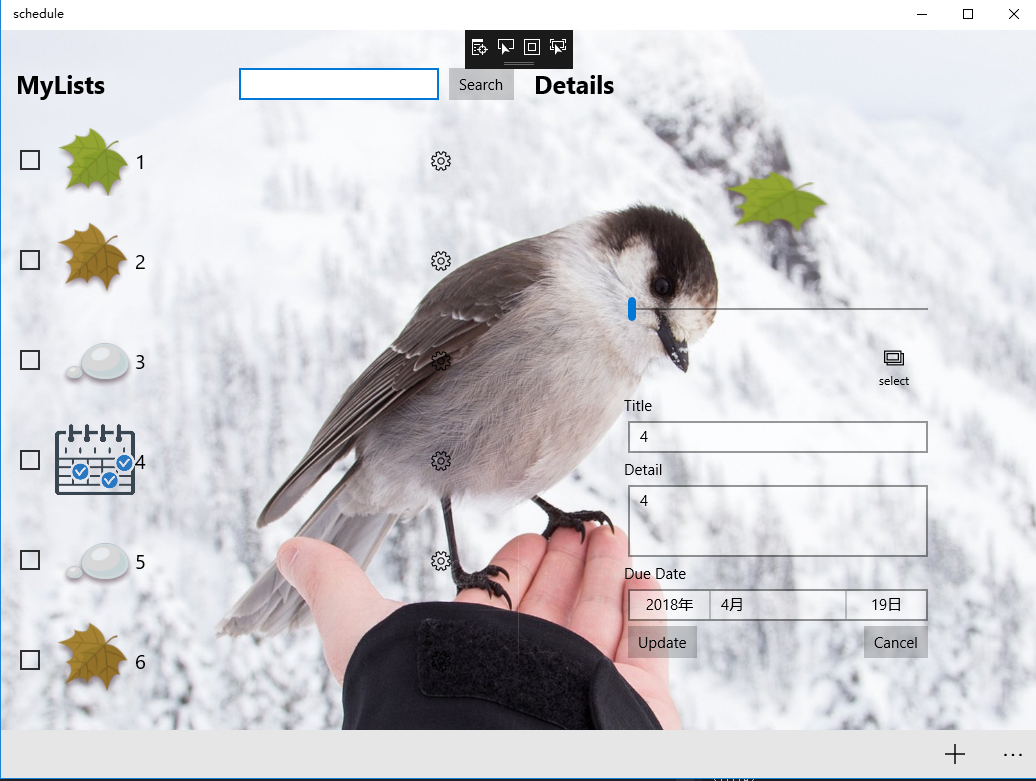
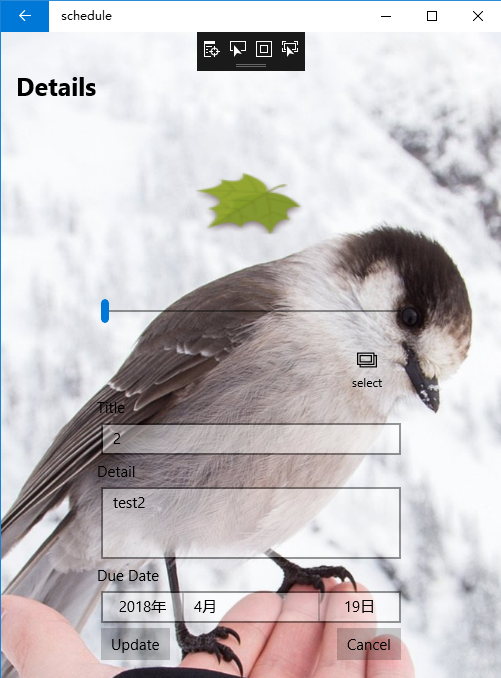
上述代码实现了数据恢复，并且恢复了挂起之前的selected的item。

1. 关键步骤截图
2. 挂起并关闭保存状态：

MainPage： NewPage：

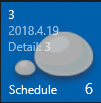
再次打开：

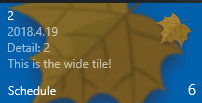
在上述图片中，实现挂起关闭然后再次打开并恢复状态；并且，Create和Update按钮也是有效的，依然可以更新之前的seletedItem或者创建一个新的item。

1. 动态磁贴以及共享：

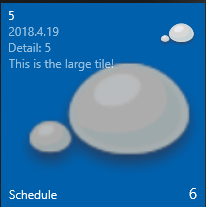
 小磁贴中，显示Title和Detail，并且没有Detail关键字；然后增加了Badge和动态背景图片。



中磁贴中，显示Title、Date和Detail，并且有Detail关键字；然后增加了Branding，displayName属性、Badge和动态背景图片。

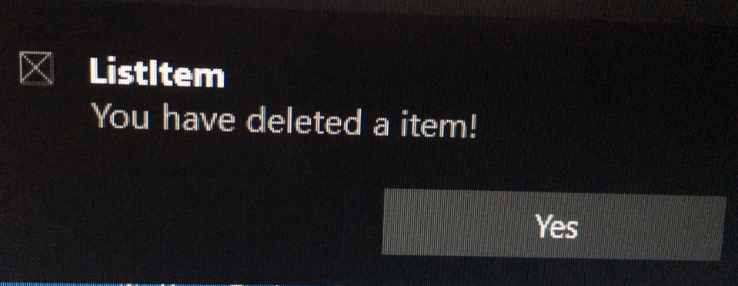


宽磁贴和中磁贴相同，另外增加了右上角的image图片和一句话。

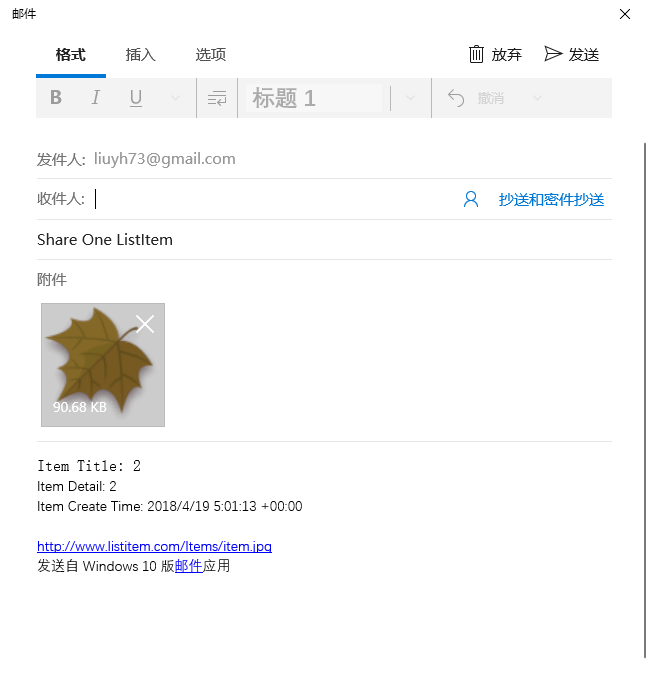


大磁贴和宽磁贴实现效果相同。

另外，每当创建更新删除一个item的时候都会弹出一个Toast，实现效果如下：

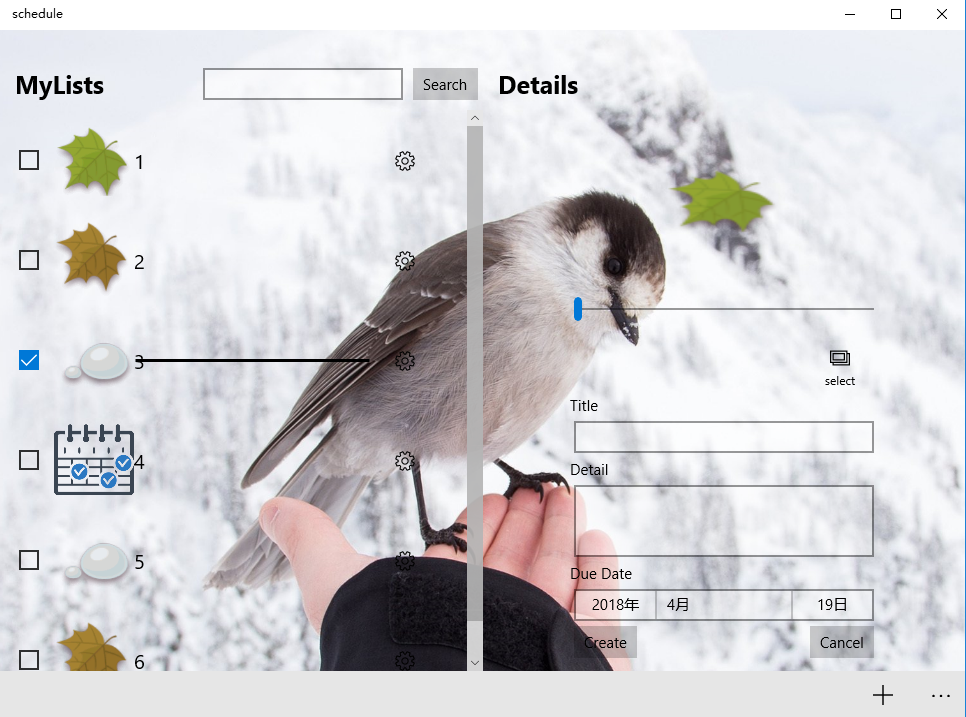


共享条目的实现截图：



在上面分享界面中，分享了TiTle，Detail，Date等信息，也实现动态分享图片。

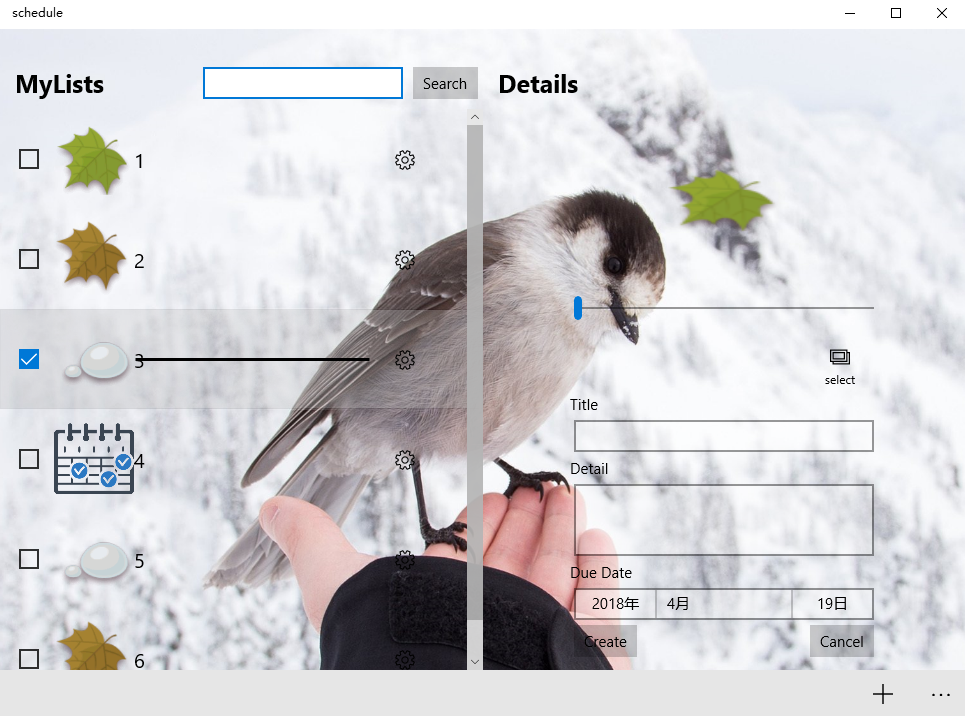
1. 数据持久化：



当前MyLists中共有六个item，其中第三个CheckBox为选中状态，然后点击关闭：



由数据库上图可以看出，第三个item的Finish状态为1，再次打开之后如下：



由上图可以看出，已正常实现数据持久化 。

1. 亮点与改进（可选）

在挂起与关闭中完成了图片的保存与回复，并且保存了selectedItem，使得恢复之后可以进行Create和Update功能。

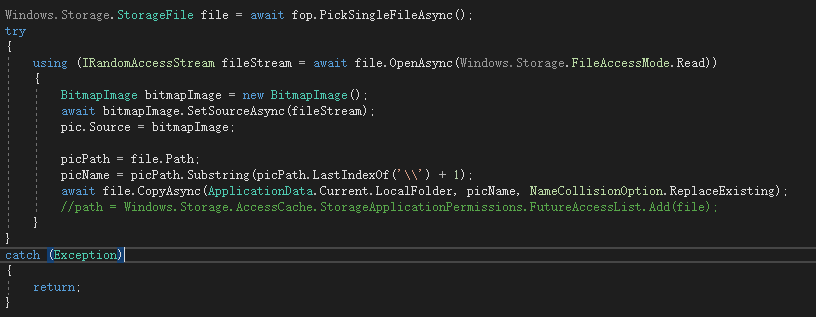
在动态磁贴中，利用UWP Tile Generater插件生成各种不同类型的磁贴图片；学习利用可视化工具Notifications Visualizer，通过xml来生成C#代码；实现了磁贴的动态背景图片，和宽磁贴、大磁贴右上角的动态图片标识。另外实现了Toast的通知功能。

在数据库实现数据持久化中，实现了图片的保存与读取，所使用的API为SQLite.Net-PCL，这个API的插入，更新，删除更加简单一些。

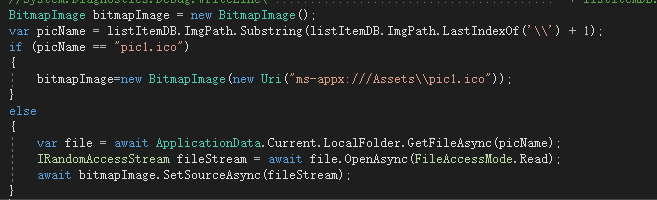
1. 遇到的问题
2. 图片的存储和读取（包括挂起并关闭的图片保存，分享动态图片，磁贴动态背景，数据库存储）：

在整个实现过程中，图片的保存与读取是最为麻烦的，下面来说一下我的解决方法：我保存的是图片的路径，这样的话在图片不删除的情况下，通过再次访问图片所在路径将图片读取出来即可。（否则存储图片可能需要用到序列化，二进制数据等等，另外我也尝试了使用byte数组，但是没有成功）在实际实现的过程中，主要是在选择图片的过程中做一些处理，选择图片的时候将选择的图片复制一个备份到ApplicationData.Current.LocalFolder下，并且保存图片的完整路径（主要保存名字即可），所以下次读取该图片时，从LocalFolder读取即可：

具体解决代码如下：



Try-Catch中实现了将用户选取的图片保存到LocalFolder文件夹下；那么这样必定会有一个问题，那就是如果不点击选择图片，那么作为默认显示的图片pic1.ico则不会保存到LocalFolder下；下面的代码解决了这个问题：



在上述代码中，我将数据库中所保存的每个item的ImgPath截取得到picName，然后判断当前的图片是否为每次应用默认的图片pic1.ico，如果判断为真的话，那么需要从Assets文件夹下读取该图片，如果判断为假的话，利用ApplicationData.Current.LocalFolder的GetAsync方法将图片读取出来。

其他保存图片并读取的方法同上。

1. 时间的问题：

每次将时间保存到数据库下面，然后在读取出来之后时间就会出现错误，比之前添加到数据库下少了8-9个小时，经过查阅资料，发现是因为DateTimeOffset存储的是世界时间，每当读取时间的时候就不是当地时间了。通过加上ToLocalTime()的方法即可正确得到时间。

1. 动态磁贴的C#实现：

最初的想法是书写xml代码来时先磁贴，但是发现这样的方法不易于动态绑定属性，之后采用XDocument类来实现该功能。其中遇到的一个问题就是branding的属性值，由于branding只能有三个值：name，logo，nameAndLogo，所以填写其他任何属性值都将导致磁贴无法显示。

1. 引用包的问题：

当自己的代码拷贝到另一个设备上时，有时候会出现缺失引用包的问题，一直编译失败，各种文件库都不存在，没有什么好的解决方案，通过多次尝试生成/部署解决方案，然后重启VS解决了这个问题，具体原因尚不清楚。

1. 思考与总结

在这次UWP的开发过程中，主要实现的就是数据持久化的问题，而其中最重要的势必就是图片的读取与保存，下面主要写一下几个重要的关键词LocalFolder和RoamingFolder的作用：LocalFolder为StorageFolder类型，利用Windows.Storage.StorageFolder localFolder = Windows.Storage.ApplicationData.Current.LocalFolder;即可获取到一个localFolder对象，并将文件图片保存在C:\Users\liuyh73 \Appdata\Local\Packages\ 63c44838-72f3-4577-b55b-2ea17f8a96ee\_06an8vaprdhge\LocalState文件夹下，是备份的共用的，对象使得C#可以访问该文件夹下的文件，比如图片等等。并且也可以使得xaml语法可以使用 "ms-appdata:///local/" 协议，可以访问本地应用程序数据存储区的文件。 RoamingFolder作用是跨设备漫游，会在多设备之间同步，使用 "ms-appdata:///roaming/" 协议，可以访问漫游应用程序数据存储区的文件，文件保存在C:\Users\liuyh73\AppData\Local\Packages\63c44838-72f3-4577-b55b-2ea17f8a96ee\_06an8vaprdhge\RoamingState文件夹下。在整个开发过程中也用到了StringBuilder，C#原生的string 在每次调用它的+=的时候 都会创建一个新的对象，把原来的对象覆盖了，如果一直用string开销会很大。所以使用StringBuilder对象的Append方法即可解决该问题，提高效率。

总体来说，这次开发过程收获了很多，对于文件访问权限的了解更进一步，对于数据持久化、数据库的使用可以更加的熟练。另外对于代码的每一部分更加清晰，所有功能可以很好地联系在一起，尽管有些代码有一些冗余，相信在之后可以将代码维护的更加精炼。还有利用MVVM框架可以是自己的思虑更加清晰，另外可以将代码模块化，利用单例模式，观察者模式等设计模式来设计程序，提高自己的开发效率，相信在不断的学习过程中可以不断提升自己的能力。