软件设计文档

1.技术选型理由：

（1）在 Windows 集成应用商店之前的版本，我们一般是下载程序进行安装，这些程序一般指的就是后缀为“.exe”的文件，英文称之为 Program，大多安装在“C:\Program Files”下。UWP 是 Universal Windows Platform 简称，即 Windows 通用应用平台，可以在 Windows 10 Mobile/Surface/PC/Xbox/HoloLens 等平台上运行，它并不是为某一个终端而设计，而是可以在所有 Windows10 设备上运行一种全平台应用，一般安装在“C:\Program Files\windowsapp”下

（2）传统 EXE 程序启动时经常会同时加载多个进程，并且一些进程还互相守护，比较占用系统资源，各个后台进程系统几乎没有任何控制权限。UWP 应用则受到系统的完全控制。比如 UWP 切换到后台运行时，Windows 会暂停你不用的 UWP 来节省资源或者电量，比如打开任务管理器，可以看到后台运行的“应用商店”就几乎不占用系统资源

（3）Win32 转制 UWP 应用好处或新增特性：

• 动态磁贴

• 消息通知

• 干净和安全的应用安装及卸载

• 应用商店监测、审核和监督

• 应用手动或自动升级

• 应用货币化，购买方便

• 应用被广泛搜索和推广

• 用户评价和评分作为下载参考

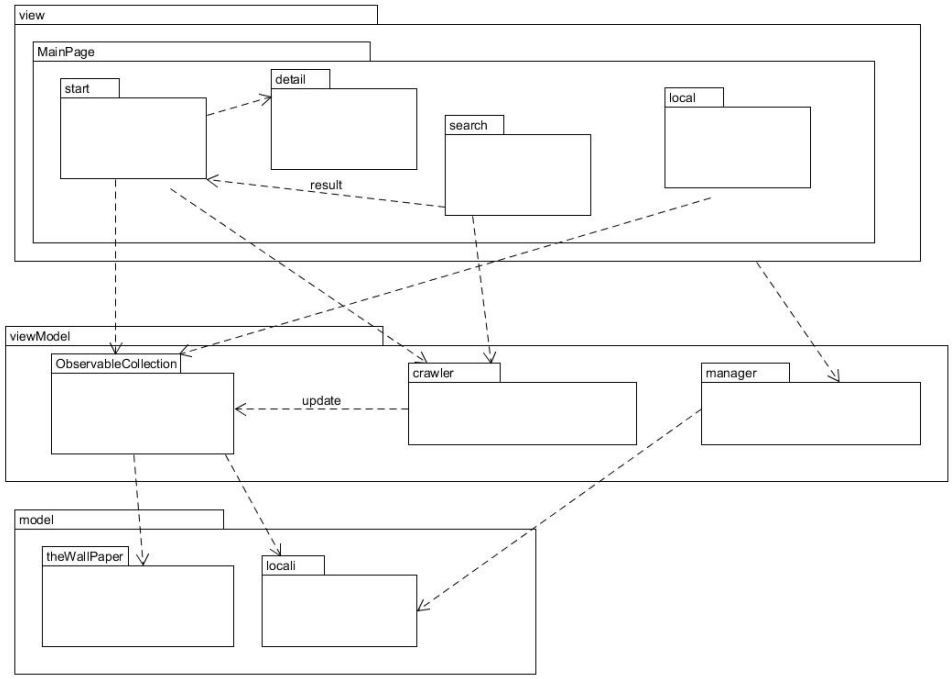
（4） 本次开发基于 VS 2015 IDE，编程语言为 C#，和 java 相似，易入门，同时微

软官网有比较详细的教程：https://developer.microsoft.com/zh-cn/windows/apps

2.架构设计：

采用 MVVM 架构，在 xaml.cs 里面写逻辑，MVVM 是 View、Model、ViewModel 合起来的称呼。

画出 UML 图如下：



mainpage 主 ui 界面，包含菜单等。内含 frame 组件用于展示其他页面，类似于 spa，所以添加页面非常便捷。各个页面与 observaleCollection 绑定，使用自适应的 gridview 展示壁纸列表。壁纸从一个国外壁纸网站上爬取来，因为一开始没有发现有 api，使用 crawler 类用于获取网页并解析出图片 url，类别目录等。manager 类用于下载到本地和设置为壁纸等功能。

search 页面方面，首先爬去国外壁纸网站的索引目录，获取其中的 tag 标签以及对应链接，然后根据输入的文本和索引中的 tag 是否匹配，如果匹配则跳向相应链接，search 内容结果用的是 start 页面，只是替换了图片内容。同时，提示下拉框根据用户输入，匹配索引项，并返回结果，字符串匹配用 indexof，包含索引内容即返回。

获取索引 tag:

```
private async Task addTags()
{
    pre.IsActive = true;
    var crawler = new Utils.Crawler();
    await Task.Run(() => crawler.grabHtml(website));
    tags  = crawler.parserTag();
    links = tags.Where((c, i) => i % 2 == 0).ToList();
    tags = tags.Where((c, i) => i % 2 != 0).ToList();
    pre.IsActive = false;
}
```

判断是否匹配：

```
private List<string> getMatchs(string text)
{
    List<string> matchs = new List<string>();
    foreach (var tag in tags)
    {
        bool contains = tag.IndexOf(text, StringComparison.OrdinalIgnoreCase) >= 0;
        //if (tag.Contains(sender.Text))
        if (contains)
            matchs.Add(tag);
    }
    return matchs;
}
```

爬取的国外壁纸网站链接为：

https://wall.alphacoders.com/by_category.php?id=3&page=

关于爬取壁纸网页方面：

其实就是通过 dom 方式解析 html 页面，然后获取 div 区域里的 image 的 url：

```
public async Task grabHtml(string website)
{
    HttpWebRequest request = (HttpWebRequest)WebRequest.Create(website);
    request.Method = "POST";
    request.ContentType = "application/x-www-form-urlencoded";
    Stream myRequestStream = await request.GetRequestStreamAsync();
    byte[] bs = Encoding.ASCII.GetBytes(postDataStr);
    myRequestStream.Write(bs, 0, bs.Length);

    WebResponse response = await request.GetResponseAsync();
    Stream stream = response.GetResponseStream();
    var result = "";
    using (StreamReader sr = new StreamReader(stream))
    {
        result = sr.ReadToEnd();
    }
    //HtmlDocument htmlDoc = new HtmlDocument();
    htmlDoc.LoadHtml(result);
}
```

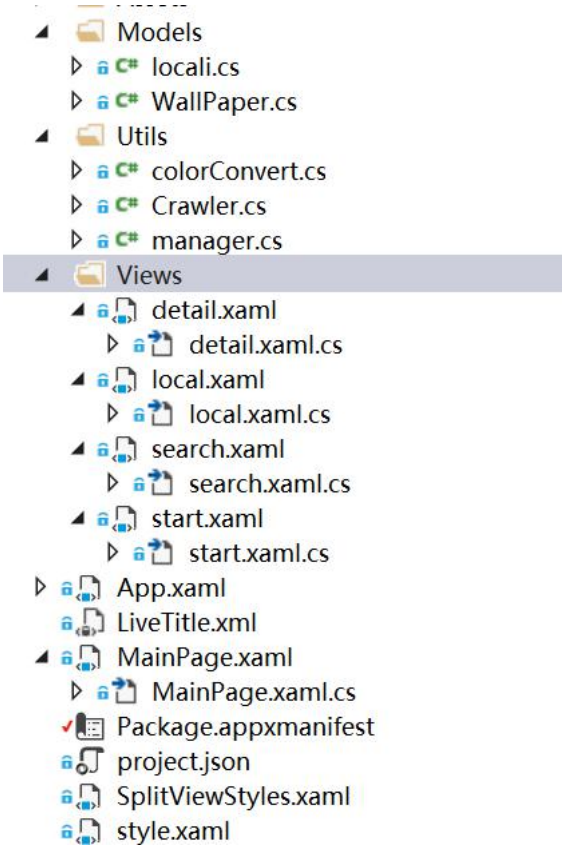
```
public int parser(ObservableCollection<theWallPaper> wallpapers)
{
    HtmlNode rootnode = htmlDoc.DocumentNode;
    //string xpathstring = "//div[@class='boxgrid']/a/img";
    //HtmlNodeCollection aa = rootnode.SelectNodes(xpathstring);
    List<HtmlNode> images = rootnode.Descendants().Where
        (x => (x.Name == "div" && x.Attributes["class"] != null && x.Attributes["class"].Value.Contains("boxgrid"))).ToLis
    List<string> uris = new List<string>();
    foreach (var image in images)
    {
        var temp = image.Descendants("a").ToList()[0].Descendants("img").ToList()[0].GetAttributeValue("src", null);

        var newone = new theWallPaper("test", new Thumbnail(temp, temp));
        wallpapers.Add(newone);

        uris.Add(temp);
    }

    //string num = rootnode.Descendants().Where
    //    (x => (x.Name == "div" && x.Attributes["class"] != null && x.Attributes["class"].Value.Contains("header-title"))
    string num = rootnode.Descendants().Where
        (x => (x.Name == "h1" && x.Attributes["class"] != null && x.Attributes["class"].Value.Contains("center title"))).T
    Regex regex = new Regex(@"\d+");
    string res = regex.Match(num).Value;
    return Convert.ToInt32(res) ;
}
```

文件目录如下：



- view-xaml 文件：用来呈现用户界面
- view-cs 文件：用来处理 View 的常规事件，负责管理 View
- models：数据模板传递的数据模型

数据绑定：
在 xaml 文件中绑定 view model：

```
<GridView x:Name="wplist" ItemsSource="{x:Bind theWallPapers}" ItemClick="GridView_ItemClick" IsItemClickEnable
    <GridView.ItemsPanel>
        <ItemsPanelTemplate>
            <ItemsWrapGrid Orientation="Horizontal"/>
        </ItemsPanelTemplate>
    </GridView.ItemsPanel>
    <GridView.ItemTemplate>
        <DataTemplate x:DataType="data:theWallPaper">
```

发现页面的缩略图：

```
<StackPanel Margin="0 0 0 10">
    <Image Source="{x:Bind thumbnail.small}" Width="350" Height="219" ></Image>
</StackPanel>
```

数据 model 如下：

```
namespace WallPaper.Models
{
    public class theWallPaper
    {
        public string name { get; set; }
        public Thumbnail thumbnail { get; set; }
        public theWallPaper(string name_, Thumbnail thumbnail_)
        {
            name = name_;
            thumbnail = thumbnail_;
        }
    }

    public class Thumbnail
    {
        public string small { get; set; }
        public string large { get; set; }
        public Thumbnail(string s, string l)
        {
            small = s;
            //large = l;
            //var aStringBuilder = new StringBuilder(s);
            //aStringBuilder.Remove(41, 3).Insert(41, "1920");
            large = Regex.Replace(s, @"thumb-350", "thumb-1920");
        }
    }

}
```

其中，s 是缩略图的 url，l 是大图的 url。

在 xaml.cs 中为数据模板赋值：

```
public ObservableCollection<theWallPaper> theWallPapers { get; set; }
```

```
private async Task init()
{
    MainProgressRing.IsActive = true;
    await addwallpaper();
    MainProgressRing.IsActive = false;
}

private async Task addwallpaper()
{
    var crawler = new Utils.Crawler();
    await Task.Run(() => crawler.grabHtml(website + page.ToString()));
    crawler.parser(theWallPapers);
    page++;
    MainPage.page = page;
}
```

3.模块划分：

Models 部分：

类名/布局文件	用途
locali.cs	下载后本地图片数据 **model**，以 **BitmapImage** 方式记录
WallPaper.cs	主页显示的图片 **model**，以 **string** 方式记录

Utils 工具类：

类名/布局文件	用途
colorConvert.cs	设置 **UWP** 界面颜色，这里和 **WIN10** 系统自身的颜色一致，并且随着系统颜色改变而改变。
Crawler.cs	获取网页上的图片，并添加到 **view models** 的集合中
manager.cs	下载图片，设置壁纸，以及获取本地图片

View 部分：

类名/布局文件	用途

detail.xaml	单独点击图片后，显示每一张图片的页面
detail.xaml.cs	加载时添加高斯模糊效果，并且提供下载，设置壁纸操作
local.xaml	显示本地下载图片页面
local.xaml.cs	获取本地图片数据并传递，提供右键点击图片设置壁纸和删除操作
search.xaml	搜索页面
search.xaml.cs	完成搜索操作，匹配输入关键字
start.xaml	发现页面，显示网上的壁纸
start.xaml.cs	加载图片，并且可下滑更新，同时右键可下载壁纸，设置壁纸

磁贴：

类名/布局文件	用途
LiveTitle.xml	为大，中，小磁贴设置样式

MainPage:

类名/布局文件	用途
MainPage.xaml	整个页面框架设计
MainPage.xaml.cs	页面按钮的逻辑控制，跳转

样式：

类名/布局文件	用途
style.xaml	页面样式设置

4.软件设计技术：

（1）面向对象编程：

①设置壁纸 url 对象：

```
public class theWallPaper
{
    public string name { get; set; }
    public Thumbnail thumbnail { get; set; }
    public theWallPaper(string name_, Thumbnail thumbnail_)
    {
        name = name_;
        thumbnail = thumbnail_;
    }
}
```

```
public class Thumbnail
{
    public string small { get; set; }
    public string large { get; set; }
    public Thumbnail(string s, string l)
    {
        small = s;
        //large = l;
        //var aStringBuilder = new StringBuilder(s);
        //aStringBuilder.Remove(41, 3).Insert(41, "1920");
        large = Regex.Replace(s, @"thumb-350", "thumb-1920");
    }
}
```

②设置加载本地图片的对象：

```
public class locali
{
    public string name;
    public BitmapImage bitmap;
    public locali(string name_, BitmapImage bitmap_)
    {
        name = name_;
        bitmap = bitmap_;
    }
}
```

（2）异步编程：

```
public async Task download(string url, int tf)
{
    string filename = url.Substring(url.Length - 10, 10);

    StorageFolder fold = Windows.Storage.ApplicationData.Current.LocalFolder;

    var item = await ApplicationData.Current.LocalFolder.TryGetItemAsync(filename);
    if (item == null)
    {
        StorageFile file = await fold.CreateFileAsync(filename, CreationCollisionOption.ReplaceExisting);
        HttpClient client = new HttpClient();

        var buffer = await client.GetBufferAsync(new Uri(url));
        await Windows.Storage.FileIO.WriteBufferAsync(file, buffer);
        if (tf == 1)
        {
            manager.setWallpaper(filename);
        }
    }


}
```

```
public static async Task<bool> setWallpaper(string filename)
{
    StorageFolder fold = Windows.Storage.ApplicationData.Current.LocalFolder;
    var file = await fold.GetFileAsync(filename);
    UserProfilePersonalizationSettings profileSettings = UserProfilePersonalizationSettings.Current;
    //bool success = await profileSettings.TrySetLockScreenImageAsync(file);
    bool success = await profileSettings.TrySetWallpaperImageAsync(file);

    var timer = new Windows.UI.Xaml.DispatcherTimer { Interval = TimeSpan.FromSeconds(0.5) };
    timer.Tick += (sender, args) =>
    {
        MainPage.initTitlebar();
        timer.Stop();
    };

    timer.Start();
    return success;
}
```

在下载完成图片后，才能对该图片进行壁纸设置。

（3）模块化编程：

将系统颜色变换，获取网页上的图片，以及壁纸的下载，设置等功能封装成一个类，方便维护以及使用。

获取网页图片：

```
public async Task grabHtml(string website)
{
    HttpWebRequest request = (HttpWebRequest)WebRequest.Create(website);
    request.Method = "POST";
    request.ContentType = "application/x-www-form-urlencoded";
    Stream myRequestStream = await request.GetRequestStreamAsync();
    byte[] bs = Encoding.ASCII.GetBytes(postDataStr);
    myRequestStream.Write(bs, 0, bs.Length);

    WebResponse response = await request.GetResponseAsync();
    Stream stream = response.GetResponseStream();
    var result = "";
    using (StreamReader sr = new StreamReader(stream))
    {
        result = sr.ReadToEnd();
    }
    //HtmlDocument htmlDoc = new HtmlDocument();
    htmlDoc.LoadHtml(result);
}
```

下载：

```
public async Task download(string url, int tf)
{
    string filename = url.Substring(url.Length - 10, 10);

    StorageFolder fold = Windows.Storage.ApplicationData.Current.LocalFolder;

    var item = await ApplicationData.Current.LocalFolder.TryGetItemAsync(filename);
    if (item == null)
    {
        StorageFile file = await fold.CreateFileAsync(filename, CreationCollisionOption.ReplaceExisting);
        HttpClient client = new HttpClient();

        var buffer = await client.GetBufferAsync(new Uri(url));
        await Windows.Storage.FileIO.WriteBufferAsync(file, buffer);
        if (tf == 1)
        {
            manager.setWallpaper(filename);
        }
    }
```

设置：

```
public static async Task<bool> setWallpaper(string filename)
{
    StorageFolder fold = Windows.Storage.ApplicationData.Current.LocalFolder;
    var file = await fold.GetFileAsync(filename);
    UserProfilePersonalizationSettings profileSettings = UserProfilePersonalizationSettings.Current;
    //bool success = await profileSettings.TrySetLockScreenImageAsync(file);
    bool success = await profileSettings.TrySetWallpaperImageAsync(file);

    var timer = new Windows.UI.Xaml.DispatcherTimer { Interval = TimeSpan.FromSeconds(0.5) };
    timer.Tick += (sender, args) =>
    {
        MainPage.initTitlebar();
        timer.Stop();
    };

    timer.Start();
    return success;
}
```

删除：

```
public static async Task deleteFile(string filename)
{
    StorageFolder fold = Windows.Storage.ApplicationData.Current.LocalFolder;
    var file = await fold.GetFileAsync(filename);
    await file.DeleteAsync();
}
```

5.设计模式：

MVVM 模式。