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：Lab1

一、参考资料

https://github.com/gfzheng/MOSAD

https://space.bilibili.com/18340402/#/channel/detail?cid=1436

https://blog.csdn.net/nomasp/article/details/44926753

https://www.cnblogs.com/wpinfo/p/6437487.html

https://www.cnblogs.com/yanxiaodi/p/4923128.html

二、实验步骤

为了方便起见，这里按每周的任务来写实验步骤：

Week 1：现在看来，Week 1 的任务量相比于其他两周，实在是少的可怜。现在来一个个看：

1. 在 MainPage 上放置 CheckBox 和 line 两个控件，当 CheckBox 被勾选时 line 出现，取消勾选则 line 消失。

这个实现起来很简单，首先打开 MainPage.xaml，点击左边的工具箱，选择 XAML 控件 CheckBox，拖拉到可视化界面。如果位置不对，可以用鼠标在界面里面进行拖拉。Line 控件相对 CheckBox 而言比较麻烦，因为其在工具箱里无法找到，所有没法在 XAML 控件中直接拖拉。因此最后通过直接修改 XAML 代码来声明 Line。

代码如下：

```
<CheckBox x:Name="CheckBox" HorizontalAlignment="Left" VerticalAlignment="Top" Click="CheckBox_Click" G
<Line x:Name="Line" X1="230" X2="310" Y1="120" Y2="120" Stroke="Black" StrokeThickness="2" Visibility="
```

按照题目要求，我们需要设置 Line 默认不可见，并且勾选 CheckBox 后 Line 出现。所以 Line 控件上添加了属性 Visibility="Collapsed"，CheckBox 控件添加 Click="CheckBox_Click"。然后添加相应事件：

```
private void CheckBox_Click(object sender, RoutedEventArgs e)
{
    if (Line.Visibility == Visibility.Collapsed)
    {
        Line.Visibility = Visibility.Visible;
    }
    else
    {
        Line.Visibility = Visibility.Collapsed;
    }
}
```

这个功能就基本实现了。具体效果如下：

2．下一个功能：点击 create 按钮时，检查 Title、Description 是否为空，DueDate 是否正确 (是否大于等于当前日期)。如果不正确，弹出对话框，显示错误信息。点击 Cancel 按钮时，Title、Description 置空，DueDate 置为当前日期。

新建一个 NewPage 添加如下代码：

```
<TextBox x:Name="title" HorizontalAlignment="Left" Text="" VerticalAlignment="Top" Width="360" PlaceholderTe
<TextBox x:Name="description" HorizontalAlignment="Left" Text="" VerticalAlignment="Top" Width="360" Height=
<DatePicker x:Name="dueDate" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="214,423,0,0" Header=
<Button Content="Create" HorizontalAlignment="Left" VerticalAlignment="Top" Width="145" Margin="214,509,0,0"
<Button Content="Cancel" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="428,509,0,0" Width="146"
```

然后添加事件实现：

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    string output;
    if (title.Text.Length == 0)
    {
        output = "Title不能为空";
    }
    else if (description.Text.Length == 0)
    {
        output = "Description不能为空";
    }
    else if (dueDate.Date < DateTime.Now.Date)
    {
        output = "dueDate不合法";
    }
    else
    {
        output = "创建TODO成功";
    }
    MessageDialog msg = new MessageDialog(output);
    msg.ShowAsync();
}
```

```
private void Button_Click_1(object sender, RoutedEventArgs e)
{
    title.Text = String.Empty;
    description.Text = String.Empty;
    dueDate.Date = DateTime.Now.Date;
}
```

效果如下：

Week 2：·

界面宽度发生改变时，界面整体始终居中；

界面右侧需有滚动条。

·尽可能使得界面简介、美观，例如为界面添加背景。

背景一直都没有添加，因为找不到什么觉得好的背景……

言归正传，界面居中的话，其实完全可以和自适应界面一起叙述（因为居中也属于自适应的一种形式），这里需要用到各种各样的 Panel 指令。比如 StackPanel 会指定子节点顺序排放（垂直或水平）。RelativePanel 是一种布局样式，可以定义各界面元素之间的关系，当屏幕分辨率发生变化时，界面元素会做出相应的调整来适应。

滚动条：工具箱里的 ScrollViewer 控件，这个没什么。

（注：最开始做 Week1 的作业实现 CheckBox 和 Line 的绑定时，采用的是在 CheckBox 控件上添加文本，结果到了 Week2 要在文本和 CheckBox 之间插入图片时费了很多功夫。其实只要去掉 CheckBox 的文本，多添加一个 TextBlock 控件就可以了

具体代码如下：

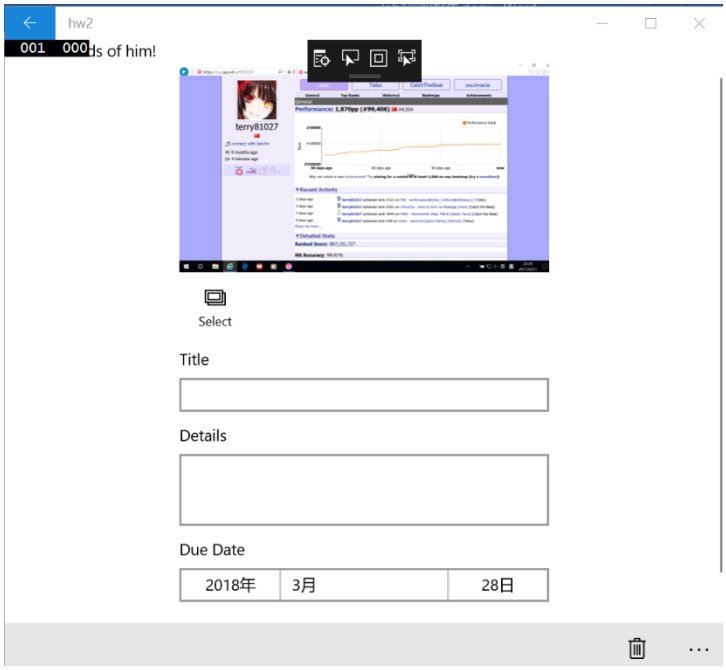
```
<Grid Grid.Row="1">
    <ScrollViewer ViewChanged="ScrollVier_ViewChanged">
        <StackPanel>
            <Image x:Name="background" Source="Assets/2017-06-21.png" Width="350" Height="200" Stre
            <RelativePanel Width="350">
                <AppBarButton x:Name="SelectPictureButton" Icon="Pictures" Label="Select" RelativeP
            </RelativePanel>
            <TextBox x:Name="Title" Header="Title" Width="350" Margin="0,12,0,0"/>
            <TextBox x:Name="Details" Header="Details" Width="350" Margin="0,12,0,0" Height="96"/>
            <DatePicker x:Name="DatePicker1" Width="350" Margin="0,12,0,0" Header="Due Date" Horizo
            <RelativePanel Width="350" Margin="0,24,0,0">
                <Button x:Name="createButton" Content="Create" Click="createButton_Click"/>
                <Button x:Name="cancelButton" Content="Cancel" RelativePanel.AlignRightWithPanel="T
            </RelativePanel>
        </StackPanel>
    </ScrollViewer>
</Grid>
```

MainPage、NewPage 两个界面之间的 Navigation：这个我是直接照着 TA 的代码和网上的资料搬过来的，具体原理我也不是很清楚。

放一部分代码吧，基本上都是照搬的：

```
            // 确保当前窗口处于活动状态
            Window.Current.Activate();
        }

        SystemNavigationManager.GetForCurrentView().BackRequested += App_BackRequested;

        SystemNavigationManager.GetForCurrentView().AppViewBackButtonVisibility = rootFrame.CanGoBack ?
        rootFrame.Navigated += OnNavigated;
    }

    private void App_BackRequested(object sender, Windows.UI.Core.BackRequestedEventArgs e)
    {
        Frame rootFrame = Window.Current.Content as Frame;
        if (rootFrame == null)
            return;

        // If we can go back and the event has not already been handled, do so.
        if (rootFrame.CanGoBack && e.Handled == false)
        {
            e.Handled = true;
            rootFrame.GoBack();
        }
    }

    private void OnNavigated(object sender, NavigationEventArgs e)
    {
        SystemNavigationManager.GetForCurrentView().AppViewBackButtonVisibility = ((Frame)sender).CanGoB
    }
```

大概效果是这样：



Week 3：Adaptive UI（自适应 UI），Data Binding。

这个相对与前面几周的任务来说，是最麻烦的。（太抽象的东西不是很懂）TA 的代码我就直接搬过来用了：

```
<VisualStateManager.VisualStateGroups>
    <VisualStateGroup x:Name="VisualStateGroup">
        <VisualState x:Name="VisualStateMin0">
            <VisualState.Setters>
                <Setter Target="InlineToDoItemViewGrid.(UIElement.Visibility)" Value="Collapsed"/>
                <Setter Target="ToDoListView.(Grid.ColumnSpan)" Value="2"/>
            </VisualState.Setters>
            <VisualState.StateTriggers>
                <AdaptiveTrigger MinWindowWidth="1"/>
            </VisualState.StateTriggers>
        </VisualState>
        <VisualState x:Name="VisualStateMin800">
            <VisualState.StateTriggers>
                <AdaptiveTrigger MinWindowWidth="800"/>
            </VisualState.StateTriggers>
        </VisualState>
    </VisualStateGroup>
</VisualStateManager.VisualStateGroups>
```

Data Binding：非常麻烦，Model 类参考 ppt 代码进行小修改：

```
class TodoItem
{
    public string id { get; set; }
    public string title { get; set; }
    public string description { get; set; }
    public DateTime date { get; set; }
    public ImageSource img { set; get; }
    public bool completed { get; set; }

    public TodoItem(string title, string description, DateTime date, ImageSource img)
    {
        this.id = Guid.NewGuid().ToString();
        this.title = title;
        this.description = description;
        this.date = date;
        this.img = img;
        this.completed = false;
```

ListItemViewModel 类中处理对 ObservableCollection 的增删改

```
public void AddTodoItem(string title, string description, DateTime date,ImageSource img)
{
    this.allItems.Add(new Models.TodoItem(title, description, date,img));
}

public void RemoveTodoItem(string id)
{
    for (int i = 0; i < allItems.Count; i++)
    {
        if (allItems[i].id == id)
        {
            this.allItems.RemoveAt(i);
        }
    }
    this.selectedItem = null;
}

public void UpdateTodoItem(string id, string title, string description, DateTime date,ImageSource img)
{
    selectedItem.id = id;
    selectedItem.title = title;
    selectedItem.description = description;
    selectedItem.date = date;
    selectedItem.img = img;
    this.selectedItem = null;
```

Ppt 中的 DIY 部分自己做了（但是貌似 RemoveTodoItem 还是有问题？）

ListView 在 DataBindng 中需要使用。ItemTemplate 用于数据绑定，数据绑定的模板一般是手写完成，用 Blend 也是可以创建数据绑定模板的。

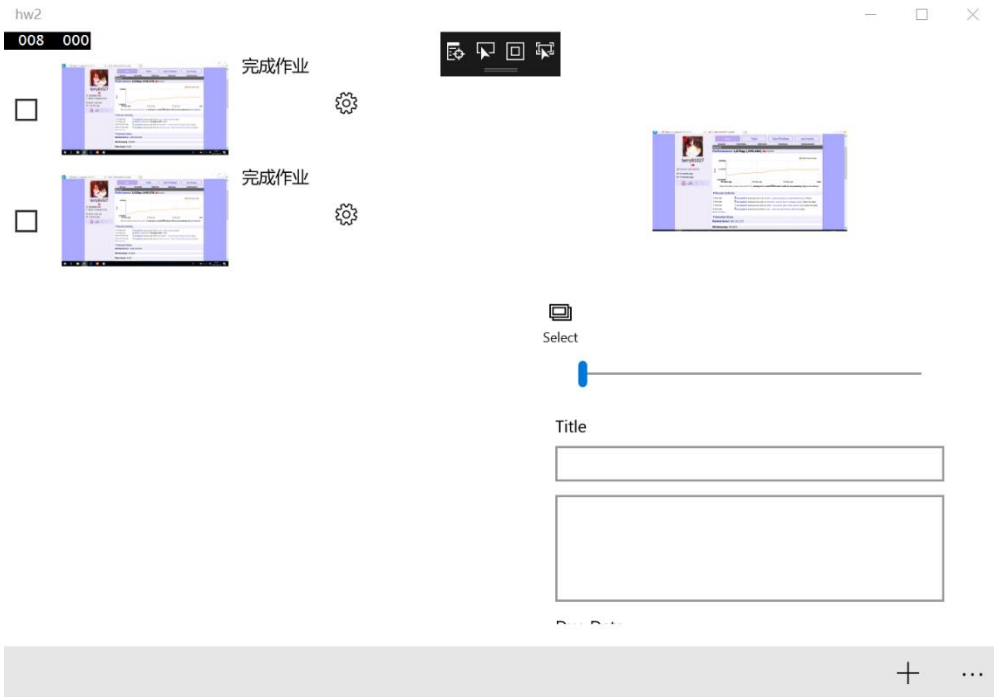
{x:Bind}扩展标可以与控件属性绑定

```
<ListView x:Name="ToDoListView" IsItemClickEnabled="True" ItemClick="TodoItem_ItemClicked"
          ItemsSource="{x:Bind ViewModel.AllItems}">
    <ListView.ItemTemplate>
        <DataTemplate x:DataType="md:TodoItem">
            <UserControl>
                <Grid Height="100">
                    <!--500-->
                    <VisualStateManager.VisualStateGroups>
                        <VisualStateGroup x:Name="image_auot_hidden">
                            <VisualState x:Name="gt800">
                                <VisualState.Setters>
                                    <Setter Target="img.(UIElement.Visibility)" Value="Vis
                                </VisualState.Setters>
                                <VisualState.StateTriggers>
                                    <AdaptiveTrigger MinWindowWidth="600"></AdaptiveTrigge
                                </VisualState.StateTriggers>
                            </VisualState>

                            <VisualState x:Name="lt800">
```

```
                <CheckBox x:Name="checkBox1" Grid.Column="0" VerticalAlignment="Center" I
                <Image x:Name="img" Source="{x:Bind img}" Grid.Column="1" Height="90" Wi
                <TextBlock Grid.Column="2" Text="{x:Bind title}" VerticalAlignment="Top"
                <Line x:Name="line1" Grid.Column="2" Stretch="Fill" Stroke="Yellow" Strol
                      Visibility="{Binding Path=IsChecked,ElementName=checkBox1, Mode=On
                <AppBarButton Grid.Column="3" Icon="Setting" IsCompact="True" VerticalAl
                    <AppBarButton.Flyout>
                        <MenuFlyout>
                            <MenuFlyoutItem  Text="Edit" Tag="Edit" />
                            <MenuFlyoutItem Text="Delete" Tag="Delete" Click="OnItemClic
                        </MenuFlyout>
                    </AppBarButton.Flyout>
                </AppBarButton>
            </Grid>
        </UserControl>

    </DataTemplate>
</ListView.ItemTemplate>
```

代码太长，就不全放出来了。

大概效果就是这样（随便截的）：

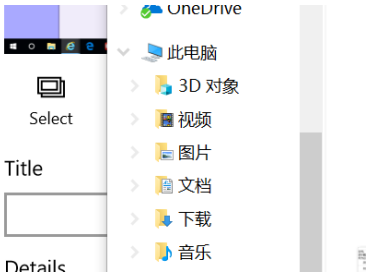


三、亮点与改进

在此基础上，增加了选择背景图片的方法（一些操作说明写在备注里）：

```
public async Task<BitmapImage> GetImageAsync(StorageFile storageFile)
{
    BitmapImage bitmapImage = new BitmapImage();
    FileRandomAccessStream stream = (FileRandomAccessStream)await storageFile.OpenAsync(FileAccessMode.Read);
    bitmapImage.SetSource(stream);
    return bitmapImage;
}

private async void SelectButton_Click(object sender, RoutedEventArgs e)
{
    FileOpenPicker picker = new FileOpenPicker();
    picker.ViewMode = PickerViewMode.List;  //设置文件的现实方式，这里选择的是图标
    picker.SuggestedStartLocation = PickerLocationId.PicturesLibrary; //设置打开时的默认路径，这里选择的是图片库
    picker.FileTypeFilter.Add(".png");
    picker.FileTypeFilter.Add(".jpg");//添加可选择的文件类型，这个必须要设置
    StorageFile file = await picker.PickSingleFileAsync();    //只能选择一个文件

    if (file != null)
    {
        this.img.ImageSource = await GetImageAsync(file);
```

通过 Select 按钮来选择本地背景图片。



四、遇到的问题

1．最开始做 Week1 的作业实现 CheckBox 和 Line 的绑定时，采用的是在 CheckBox 控件上添加文本，结果到了 Week2 要在文本和 CheckBox 之间插入图片时费了很多功夫。其实只要去掉 CheckBox 的文本，多添加一个 TextBlock 控件就可以了。

2．一开始只是单纯的使用 Margin 进行布局，然而问题非常多（比如一些控件的位置无法确定），后来改用了 Grid 布局：

```
<Grid.ColumnDefinitions>
    <ColumnDefinition Width="42"/>
    <ColumnDefinition Width="Auto"/>
    <ColumnDefinition Width="*"/>
    <ColumnDefinition Width="100"/>
</Grid.ColumnDefinitions>
```

```
<Grid.RowDefinitions>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="*"/>
</Grid.RowDefinitions>
```

这是一个栅格布局。利用 ColumnDefinitions 和 RowDefinition 来定义每一行和每一列的属性，其中有具体数字、Auto、*三种定义方式。具体数字就是像素大小，不需要具体说明；Auto 表示其值跟随行(或列)的长宽所决定；*表示比例。比如下列例子，表示按 5:3 来分割成两列，如果是 1*，可以省略成*。

3．Edit 和 Delete：这里目前为止还没有解决。已经通过 AppBarButton 在 Setting 创建了两个选项：

```
<AppBarButton Grid.Column="3" Icon="Setting" IsCompact="True" VerticalAlignment="Center" Cl
    <AppBarButton.Flyout>
        <MenuFlyout>
            <MenuFlyoutItem  Text="Edit" Tag="Edit" />
            <MenuFlyoutItem Text="Delete" Tag="Delete" Click="OnItemClick"/>
        </MenuFlyout>
    </AppBarButton.Flyout>
</AppBarButton>
```

同时创建了 MenuFlyoutItem_Click 事件，但是两个不同的按钮有完全不同的用处，共用一个事件，感觉上不行。于是单独创建 Click 事件，但是不知道为什么，一直报空指针的错误（不知道原因），因此暂时还没解决。

4．在做 Week3 的任务时，出现了一些奇怪的 bug（比如 Line 控件与文字突然分开了等等，就是之前已经完成的东西又出了问题）。

五、思考与总结

这次的实验作业对我来说非！常！困！难！完全零基础的情况下，通过自己摸索来实现功能还是一件很困难的事。虽然确实学了不少东西，也会解决不少问题，但这耗费了比专必课多很多的时间来弄明白。老师每周的进度也有点快。希望每周的作业难度不要跨度那么大，循序渐进才能有动力。

不过总而言之，收获还是不小的。