一、参考资料

https://docs.microsoft.com/zh-cn/windows/uwp/ ---- 微软官方 UWP 开发文档

https://docs.microsoft.com/zh-cn/windows/uwp/data-binding/data-binding-quickstart ---- 数据绑定概述（微软官方文档）

https://docs.microsoft.com/zh-cn/windows/uwp/data-binding/data-binding-in-depth ---- 深入了解数据绑定（微软官方文档）

http://www.cnblogs.com/cjw1115/p/4888348.html 数值绑定用法相关博客

http://www.ruanyifeng.com/blog/2015/02/mvcmvp_mvvm.html MVVM 的了解学习

二、实验步骤

第一周的作业比较简单，就是完成两个界面和一些简单逻辑；

一、 界面编写：界面基本上就是 xaml 的编写，了解了 button、textbox、textblock、daatepicker、image 等控件就可以完成编写；

二、 创建新事件的逻辑：为 button 添加 click 函数即可，函数获取到控件里面填写的相关内容然后判断是否为空，都不为空才允许创建事件；相关逻辑代码如下：

```
private async void CreateButton_Click(object sender, RoutedEventArgs e)
{
    eMessage = "";

    if (title.Text == "")
    {
        eMessage += "title should not be empty!\n";
    }

    if (detail.Text == "")
    {
        eMessage += "detail should not be empty!\n";
    }

    if (DateTime.Now.Year == date.Date.DateTime.Year && DateTime.Now.Month == date.Date.DateTime.Month && DateTime.Now.Day == date.Date.DateTime.Day)
    {
    }
    else if (DateTime.Compare(DateTime.Now, date.Date.DateTime) > 0)
    {
        eMessage += "the due date has passed!\n";
    }

    if(importance.SelectedIndex == -1)
    {
        eMessage += "the importance should not be empty";
    }

    if (eMessage == "")
    {
        eMessage = "you have created a new task successfully!";
    }

    // Create the message dialog and set its content
    var messageDialog = new MessageDialog(eMessage);

    if (eMessage == "you have created a new task successfully!")
    {
        messageDialog.Commands.Add(new UICommand(
                    "OK",
                    new UICommandInvokedHandler(this.CommandInvokedHandler)));

        // Set the command that will be invoked by default
        messageDialog.DefaultCommandIndex = 0;
        ViewModel.AddTaskItem(title.Text, detail.Text, importance.SelectedIndex, date.Date.DateTime);
        MyListView.SelectedIndex = -1;
        // MyListView.SelectedIndex = ViewModel.TaskItems.Count();
    }
    else
    {
        messageDialog.Commands.Add(new UICommand(
            "Try again",
            new UICommandInvokedHandler(this.CommandInvokedHandler)));
        messageDialog.Commands.Add(new UICommand(
            "Close",
            new UICommandInvokedHandler(this.CommandInvokedHandler)));
        messageDialog.DefaultCommandIndex = 0;
        messageDialog.CancelCommandIndex = 1;
    }
    await messageDialog.ShowAsync();
}
```

三、

除此之外还需要把 checkbox 和 line 绑定起来，因为到这里为止还没有学到数据绑定所以就是简单的为 checkbox 写了两个函数 ischecked 和 isunchecked，前者把 line 的 visibility 写为 visible，表示 checkbox 被勾选以后就把 line 显示出来，后面一个函数把 visibility 写为 collapse，表示勾选被取消就把 line 隐藏；这个代码相当简单就不贴出来了。到这里就基本完成第一周的任务了。

第二周也比较简单，需要完成布局（主要是 mianpage 的自适应和 newpage 的自动居中）

以及页面间导航；这一周需要用到布局相关的知识，了解 listview、grid 布局，学习 frame 类及其函数基本上就差不多够了，然后开始编写布局和页面跳转：

一、 Mainpag 编写：这周的主界面是只需要完成一个列表，列表里面有许多的事件 item，考虑到 item 的个数不定，随着后续添加而增多，所以可以使用 listView 布局，每一个 item 都是一个 listViewitem；然后每一个 listViewitem 都用一个 grid 打包起来（这样就方便布局每个 item 中的 title、checkbox、image、line 等），所以我们使用 grid，画出方格，分出四列，第一列放 checkbox，第二列放图片，第三列放 line 和 textblock，第四列放一个表示重要性的符号，这里面前三个元素居左，最后一个居右，所以在第三列给出 grid.columndefinition =“*”，这样全部四列占满屏幕，多余部分由第三列占取；这样就基本完成界面布局了，然后由于 item 增多，可能在单个界面无法完成完全的显示，所以需要显示滚动条，这里的 listView 是自带滚动条的，所以无需多做操作，最终相关 xaml 代码如下：

二、 Newpage 编写： newpage 基本内容在上周已经完成了，这种只需要布局一下，然后调整居中即可。整个信息列表（title、detail、date 等）使用一个 grid 包含，设置出每行的行高即可，然后将这个 grid 的 verticalalign 置为 center 就可以完成整体的居中（因为这个 grid 包含在页面 grid 之下)。最后，需要显示滚动条，直接加上 scrollView（需要了解 scrollview)；这样 newpage 布局就基本上完成了。cs 代码如

下：

```
<ScrollViewer Grid.Row="1" VerticalAlignment="Center">
  <StackPanel>
    <Grid Width="440">
      <Grid.RowDefinitions>
        <RowDefinition Height="460"/>
        <RowDefinition Height="50"/>
        <RowDefinition Height="40"/>
        <RowDefinition Height="80"/>
        <RowDefinition Height="40"/>
        <RowDefinition Height="80"/>
        <RowDefinition Height="40"/>
        <RowDefinition Height="240"/>
        <RowDefinition Height="40"/>
        <RowDefinition Height="80"/>
        <RowDefinition Height="50"/>
      </Grid.RowDefinitions>
      <Image x:Name="photo" Grid.Row="0" Source="Assets/Logo.png" Height="440" Width="440"/>
      <AppBarButton Grid.Row="1" Width="60" Height="40" Icon="BrowsePhotos" HorizontalAlignment="Right" VerticalAlignment="Top" Click="ChooseImageButton_Click" FontSize="24"/>
      <TextBlock Grid.Row="2" HorizontalAlignment="Left" Text="Title" TextWrapping="Wrap" VerticalAlignment="Top" FontSize="24" FontWeight="Bold"/>
      <TextBox Grid.Row="3" x:Name="title" HorizontalAlignment="Left" Text="" VerticalAlignment="Top" Height="40" Width="440" TextChanged="title_TextChanged"/>
      <TextBlock Grid.Row="4" HorizontalAlignment="Left" Text="Importance" TextWrapping="Wrap" VerticalAlignment="Top" FontSize="24" FontWeight="Bold"/>
      <ComboBox Grid.Row="5" x:Name="importance" SelectedIndex="0" Width="440" Height="40" SelectionChanged="ComboBox_SelectionChanged">
        <x:String>red ---- very important</x:String>
        <x:String>orange ---- important</x:String>
        <x:String>green ---- not so important</x:String>
        <x:String>blue---- secondery</x:String>
      </ComboBox>
      <TextBlock Grid.Row="6" HorizontalAlignment="Left" Text="Detail" TextWrapping="Wrap" VerticalAlignment="Top" FontSize="24" FontWeight="Bold"/>
      <TextBox Grid.Row="7" x:Name="detail" HorizontalAlignment="Left" Text="" VerticalAlignment="Top" Height="200" Width="440" AcceptsReturn="True" TextWrapping="Wrap" TextChanged="detail_TextChanged"/>
      <TextBlock Grid.Row="8" HorizontalAlignment="Left" Text="Due" TextWrapping="Wrap" VerticalAlignment="Top" FontSize="24" FontWeight="Bold"/>
      <DatePicker Grid.Row="9" x:Name="date" HorizontalAlignment="Left" VerticalAlignment="Top" MinWidth="440" FontSize="20"/>
      <Grid Grid.Row="10">
        <Grid.ColumnDefinitions>
          <ColumnDefinition Width="220"/>
          <ColumnDefinition Width="220"/>
        </Grid.ColumnDefinitions>
        <Button Grid.Column="0" x:Name="CreateButton" Content="create" HorizontalAlignment="Left" VerticalAlignment="Top" Click="CreateButton_Click" FontSize="24"/>
        <Button Grid.Column="1" x:Name="CancelButton" Content="cancel" HorizontalAlignment="Right" VerticalAlignment="Top" Click="CancelButton_Click" FontSize="24"/>
      </Grid>
    </Grid>
  </StackPanel>
```

三、

页面间导航，使用 navigated 函数和 goback 函数即可完成，难度不大，代码如下：

```
Frame.Navigate(typeof(BlankPage1), index);
```

```
Frame.GoBack();
```

四、

关于返回按钮和新建按钮的制作：

返回按钮：（使用 app 自带的 appviewbackbutton 即可，在 app.xaml.cs 中插入如下代码来显示和隐藏这个按钮）

```
rootFrame.Navigated += OnNavigated;
SystemNavigationManager.GetForCurrentView().BackRequested += OnBackRequested;
```

```
private void OnNavigated(object sender, NavigationEventArgs e)
{
    SystemNavigationManager.GetForCurrentView().AppViewBackButtonVisibility = ((Frame)sender).CanGoBack ?
        AppViewBackButtonVisibility.Visible :
        AppViewBackButtonVisibility.Collapsed;
}

private void OnBackRequested(object sender, BackRequestedEventArgs e)
{
    Frame rootFrame = Window.Current.Content as Frame;
    if (rootFrame == null)
    {
        return;
    }
    if (rootFrame.CanGoBack && e.Handled == false)
    {
        e.Handled = true;
        rootFrame.GoBack();
    }
}
```

这样就可以在 newpage 显示出返回按钮，在 mainpage 隐藏返回按钮；新建按钮：直接导向新页面

Xaml：

```
<Page.BottomAppBar>
    <CommandBar>
        <AppBarButton x:Name="DeleteButton"  Icon="Delete" Label="Add" Click="DeleteButton_Click"/>
        <AppBarButton x:Name="appBarButton"  Icon="Add" Label="Add" Click="NavigateButton_Click"/>
    </CommandBar>
</Page.BottomAppBar>
```

Cs 代码：

```
private void NavigateButton_Click(object sender, RoutedEventArgs e)
{
    int index = -1;
    Frame.Navigate(typeof(BlankPage1), index);
}
```

第三周就很爆炸了，仅仅是课件和文档的学习我就花了整整两天，，，，需要了解的东西是真的多，首先是 adaptive UI（关于 visualstateGroup 和 visualstate 的使用来实现自适应 UI），然后是数据绑定（x：bind 和 binding），虽然只要学两个东西但是难度大啊！

这里考虑使用 MVVM 模式，这个模式比较好用，所以这里可以了解一下 MVC，MVP 和 MVVM（参考之前提到的博客）

然后逐个解释 MVVM：

一、 首先是 Model，这个概念对应有一个 TaskItem 的类，它的每个实例对应 mainpage 的每一个 item，然后它的成员变量对应了每一个 item 要绑定的信息，具体实现如下：

```
public class TaskItem: INotifyPropertyChanged
{
    private bool isChecked { get; set; }
    private string title { get; set; }
    private string detail { get; set; }
    private int importance { get; set; }
    private DateTimeOffset dueTime { get; set; }

    public bool IsChecked
    {
        get { return this.isChecked; }
        set {
            isChecked = value;
            OnPropertyChanged();
        }
    }

    public string Title
    {
        get { return this.title; }
        set
        {
            title= value;
            OnPropertyChanged();
        }
    }

    public int Importance
    {
        get { return importance; }
        set
        {
            importance = value;
            OnPropertyChanged();
        }
    }

    public string Detail
    {
        get { return this.detail; }
        set
        {
            detail = value;
            OnPropertyChanged();
        }
    }

    public DateTimeOffset DueTime
    {
        get { return dueTime; }
        set
        {
            dueTime = value;
            OnPropertyChanged();
        }
    }
```

```
public TaskItem()
{
    this.isChecked = false;
    this.title = "";
    this.detail = "";
    this.importance = 0;
    this.dueTime = new DateTime(2018, 3, 30);
}

public TaskItem(string Title, string Detail, int Importance, DateTime DueTime)
{
    this.isChecked = false;
    this.title = Title;
    this.detail = Detail;
    this.importance = Importance;
    this.dueTime = DueTime;
}
public event PropertyChangedEventHandler PropertyChanged;

public void OnPropertyChanged([CallerMemberName]string propertyName = "")
{
    if (PropertyChanged != null)
    {
        PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
    }
}
}
```

说明一下：ischecked 对应 checkbox 的是否勾选；title 对应 title 信息；detail 对应 detail 信息；importance 对应重要性信息；dueTime 对应 duedate 信息；

然后必须要把这个类集成到一个 INotifyPropertyChanged 的接口，否则无法响应到 item 内容的更改，通俗点说就是更新 item 的信息以后无法在页面实时更新，这是一个坑，也是弄了很久才解决

二、 ViewModel，这个概念有对应类 TaskListViewModel，用来处理对 View 的响应，当 View 提出某个请求，先交到 ViweModel，然后由 ViewModel 来调用 Model 的接口进行处理，所以最后 Model 和 View 都只知道 ViewModel 而不知道对方，这就是 MVVM。这个类里面写了对 item 的增删改，当 mainpage 提出要增删改某个 item 就告诉 ViewModel，由它处理，具体实现如下：

```
public class TaskListViewModel
{
    private ObservableCollection<TaskItem> taskItems = new ObservableCollection<TaskItem>();
    public ObservableCollection<TaskItem> TaskItems { get { return this.taskItems; } }
    public TaskListViewModel()
    {
        this.taskItems.Add(new TaskItem("Watch movie", "Watch movie with roommate", 1, new DateTime(2018, 3, 30)));
        this.taskItems.Add(new TaskItem("Play game", "Play computer games with roommate", 2, new DateTime(2018, 3, 31)));
    }
    public void AddTaskItem(string Title, string Detail, int Importance, DateTime DueTime)
    {
        this.taskItems.Add(new TaskItem(Title, Detail, Importance, DueTime));
    }
    public void DeleteTaskItem(int index)
    {
        this.taskItems.RemoveAt(index);

    }
    public void UpdateList(int index, string Title, string Detail, int Importance, DateTime DueTime)
    {
        taskItems[index].Title = Title;
        taskItems[index].Detail = Detail;
        taskItems[index].DueTime = DueTime;
        taskItems[index].Importance = Importance;
    }
}
```

这个比较简单然后也很好理解所以就不再说明三、 View，对应于页面显示（mainpage 和 newpage），这里面需要将 mainpage 的相关控件绑定对应的 taskItem 的成员变量，以完成信息绑定，具体做法如下：

```
<ListView x:Name="MyListView" IsItemClickEnabled="True"
        ItemsSource="{x:Bind ViewModel.TaskItems}"
        Grid.Row="1" Grid.Column="0"
        ScrollViewer.VerticalScrollBarVisibility="Visible" HorizontalContentAlignment="Stretch" ItemClick="Update_Click">
    <ListView.ItemContainerStyle>
        <Style TargetType="ListViewItem">
            <Setter Property="Margin" Value="0,0,0,0"/>
            <!--<Setter Property="bo"-->
            <Setter Property="HorizontalContentAlignment" Value="Stretch"></Setter>
            <Setter Property="Height" Value="150"/>
        </Style>
    </ListView.ItemContainerStyle>
    <ListView.ItemTemplate>
        <DataTemplate x:DataType="local:TaskItem">
            <Grid>
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="60"/>
                    <ColumnDefinition Width="auto"/>
                    <ColumnDefinition Width="*"/>
                    <ColumnDefinition Width="70"/>
                </Grid.ColumnDefinitions>
                <CheckBox IsChecked="{x:Bind IsChecked, Converter={StaticResource BoolToBool}, Mode=TwoWay}" Grid.Column="0" Content="" HorizontalAlignment="Left" VerticalAlignment="Center"/>
                <Image Visibility="{Binding ElementName=width, Path=Width, Converter={StaticResource WidthToVisibleConverter}, Mode=OneWay}" Grid.Column="1" Source="Assets/lake.png" Height="40" Width="40" Margin="0,0,40,0" HorizontalAlignment="Left"/>
                <Rectangle Grid.Column="2" HorizontalAlignment="Left" VerticalAlignment="Center" Height="3" Width="200" Fill="Black" Visibility="{x:Bind IsChecked, Converter={StaticResource BoolToVisibleConverter}, Mode=OneWay}"/>
                <TextBlock Text="{x:Bind Title, Mode=OneWay}" Grid.Column="2" HorizontalAlignment="Left" VerticalAlignment="Center" Height="34" Width="300" FontSize="24"/>
                <FontIcon Grid.Column="3" Glyph="&#xED81;" Foreground="{x:Bind Importance, Converter={StaticResource ImportanceToColor}, Mode=OneWay}"/>
            </Grid>
        </DataTemplate>
    </ListView.ItemTemplate>
</ListView>
```

说明：使用 ListView.ItemTemplate 提供模板容器，DataTemplate 定义数据模板，然后在 ItemsSource 指出模板数据来源： ItemsSource="{x:Bind ViewModel.TaskItems}

四、 Adaptive UI，为什么这个最后说呢，因为如果先做好自适应再做数据绑定后面还是还要更改的（因为 line 的显示和隐藏是需要绑定的，而不是仅仅靠 visualstate 就

能够完成)。这里可以考虑采用 blend 设计，但是作为程序员还是要了解编程细节，

代码如下：

```
<VisualStateManager.VisualStateGroups>
    <VisualStateGroup>
        <VisualState x:Name="MinWidth0">
            <VisualState.StateTriggers>
                <AdaptiveTrigger MinWindowWidth="1"/>
            </VisualState.StateTriggers>
            <VisualState.Setters>
                <Setter Target="grid.(UIElement.Visibility)" Value="Collapsed"/>
            </VisualState.Setters>
        </VisualState>
        <VisualState x:Name="MinWidth600">
            <VisualState.StateTriggers>
                <AdaptiveTrigger MinWindowWidth="600"/>
            </VisualState.StateTriggers>
            <VisualState.Setters>
                <Setter Target="grid.(UIElement.Visibility)" Value="Collapsed"/>
            </VisualState.Setters>
        </VisualState>
        <VisualState x:Name="MinWidth800">
            <VisualState.StateTriggers>
                <AdaptiveTrigger MinWindowWidth="1000"/>
            </VisualState.StateTriggers>
            <VisualState.Setters>
                <Setter Target="appBarButton.(Control.IsEnabled)" Value="False"/>
            </VisualState.Setters>
        </VisualState>
    </VisualStateGroup>
</VisualStateManager.VisualStateGroups>
```

这里设置当页面宽度小于 600 和 1000 的时候会隐藏 newpage 的内容（新建和更新 item 界面)，大于 1000 就要显示出来；

而图片的显示和隐藏是通过获取屏幕宽度并判断宽度是否大于 600 来实现的：

Xaml：

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}" Padding="20,20,20,20" SizeChanged="Page_SizeChanged">
```

Cs：

```
public MainPage()
{
    this.InitializeComponent();
    this.ViewModel = App.viewModel;
    if (Window.Current.Bounds.Width > 600)
    {
        width.Width = 700;
    }
    else
    {
        width.Width = 500;
    }
}
private void Page_SizeChanged(object sender, SizeChangedEventArgs e)
{
    if (Window.Current.Bounds.Width > 600)
    {
        width.Width = 700;
    }
    else
    {
        width.Width = 500;
    }
}
```

页面 grid 绑定 sizechanged 函数，根据函数改变一个“桥”“width”的宽度，然后让 image 的可见性绑定到这个“桥”，当页面宽度大于 600，这个桥的宽度是 700，根据值转换器 visibility 转化为 1，图片可见，当页面宽度小于 600，桥的宽度是 500，根据值转换器 visibility 转化为 0，图片不可见；代码如下

Xaml：

```
            <Image Visibility="{Binding ElementName=width, Path=Width, Converter={StaticResource WidthToVisibleConverter}, Mode=OneWay}"
            <Rectangle Grid.Column="2" HorizontalAlignment="Left" VerticalAlignment="Center" Height="3" Width="200" Fill="Black" Visibility="{
            <TextBlock Text="{x:Bind Title, Mode=OneWay}" Grid.Column="2" HorizontalAlignment="Left" VerticalAlignment="Center" Height="34"
            <FontIcon Grid.Column="3" Glyph="&#xEDB1;" Foreground="{x:Bind Importance, Converter={StaticResource ImportanceToColor}, Mode
        </Grid>
    </DataTemplate>
</ListView.ItemTemplate>
</ListView>

<Border x:Name="width"/>
```

值转换器 Cs：

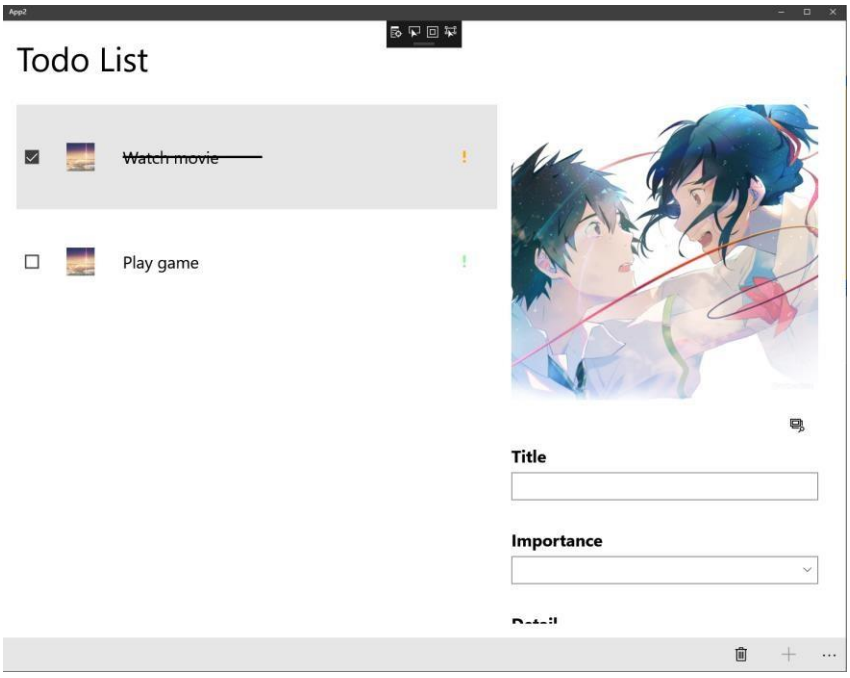
```
public class widthToVisibleConverter : Windows.UI.Xaml.Data.IValueConverter
{
    public object Convert(object value, Type targetType,
    object parameter, string language)
    {
        Windows.UI.Xaml.Visibility visibility = Windows.UI.Xaml.Visibility.Collapsed;
        // value is the data from the source object.
        double thisbool = (double)value;
        if (thisbool > 600)
        {
            visibility = Windows.UI.Xaml.Visibility.Visible;
            return visibility;
        }
        // Return the value to pass to the target.
        return visibility;
    }
```

至此，1000 以上到 1000-600；1000-600 到 600 一下的过度完成。

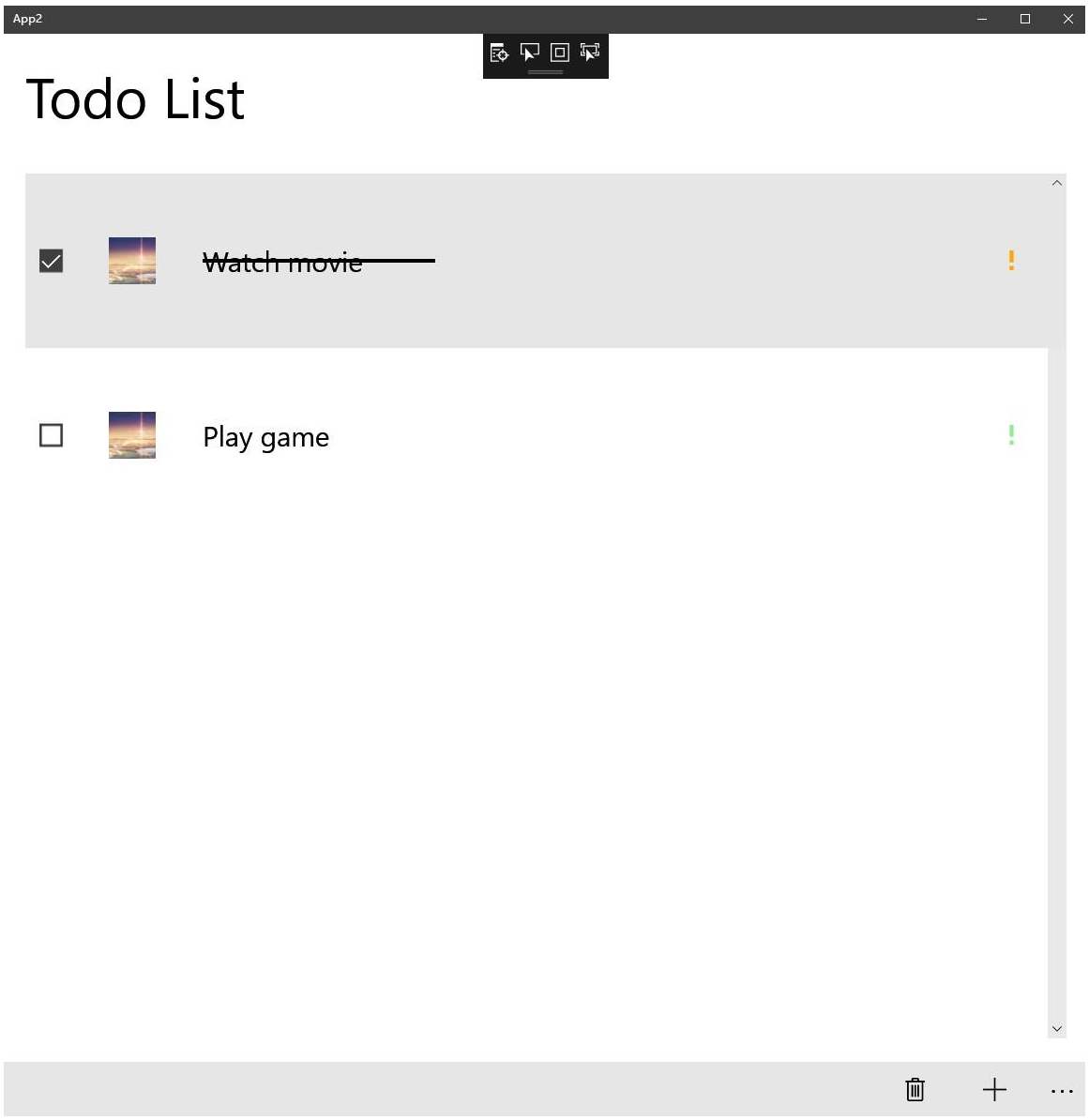
到这里实验总算是艰难完成，期间困难数不尽数，，，，这在后面再说，，，，尤其是第三周，敢觉一周完成了三周的工作量，，，orz

三、关键步骤截图

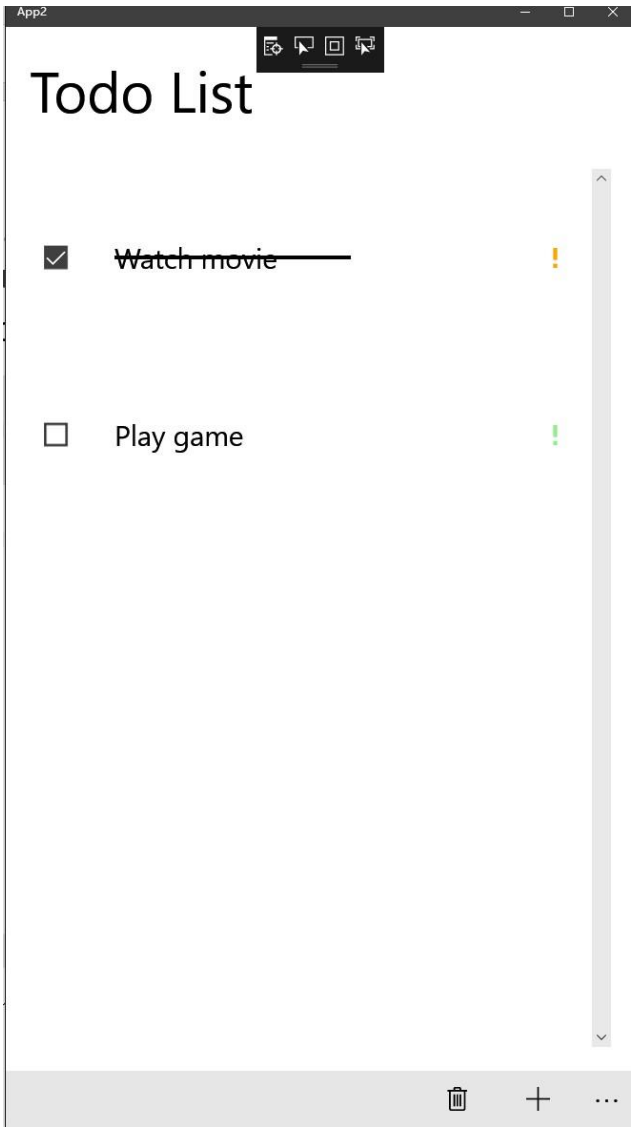
1000 宽度以上界面：



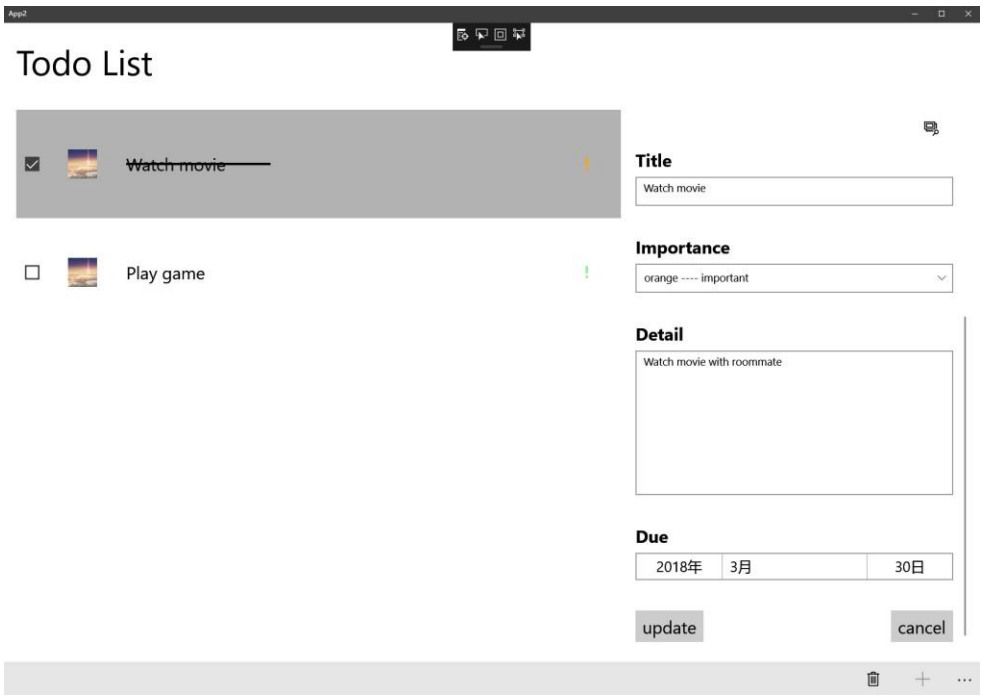
宽度 600-1000 界面：



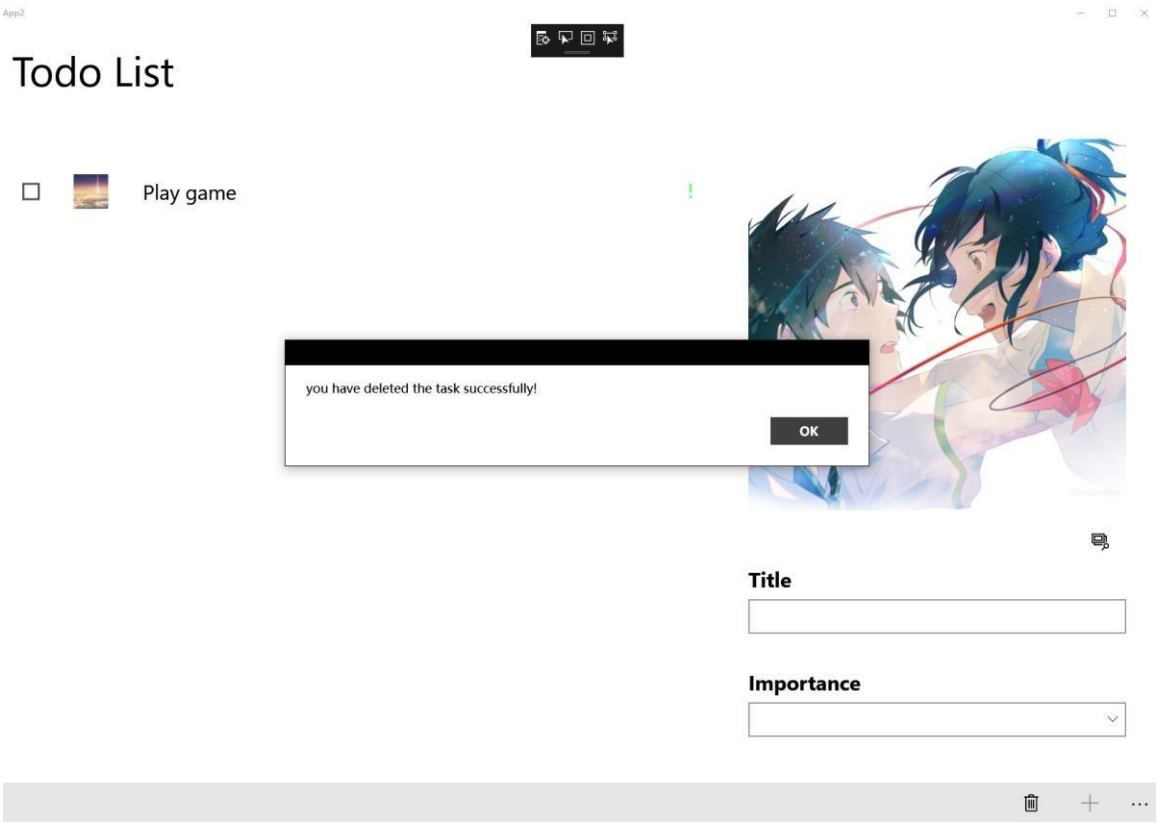
600 以下界面：



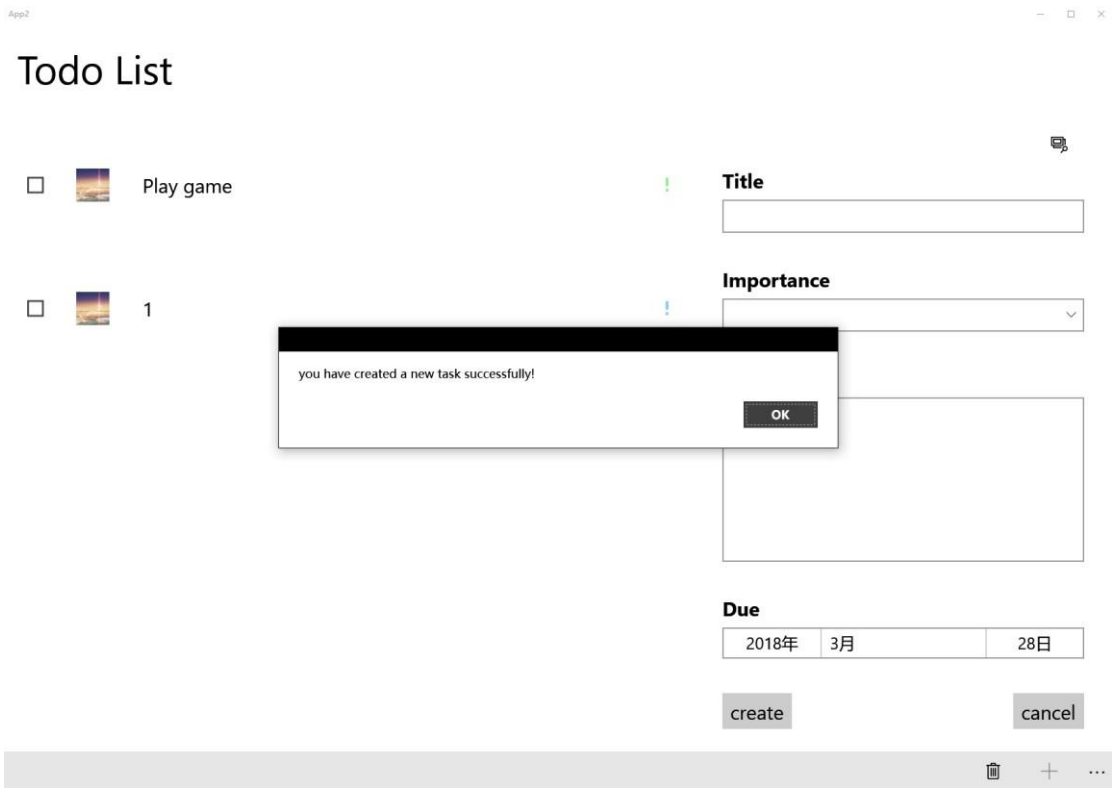
宽屏下点击显示信息：


App2
Todo List
Watch movie
Play game
Title
Watch movie
Importance
orange ---- important
Detail
Watch movie with roommate
Due
2018年
3月
30日
update
cancel

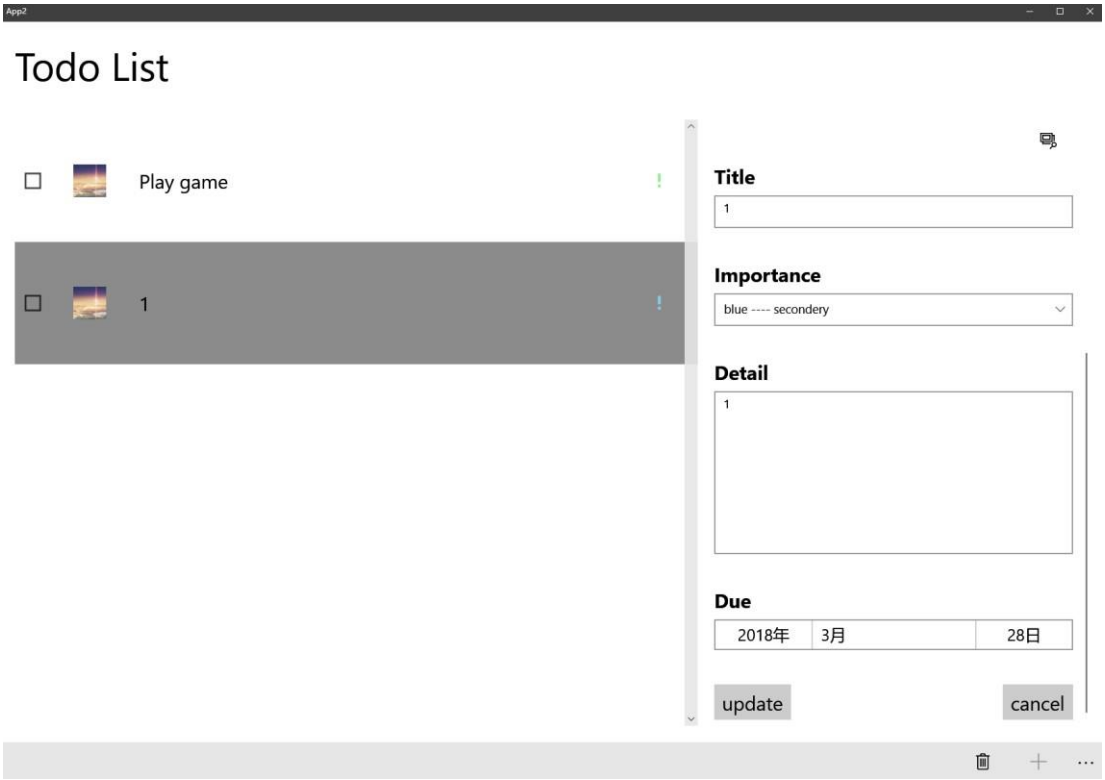
删除操作：


App2
Todo List
Play game
you have deleted the task successfully!
OK
Title
Importance

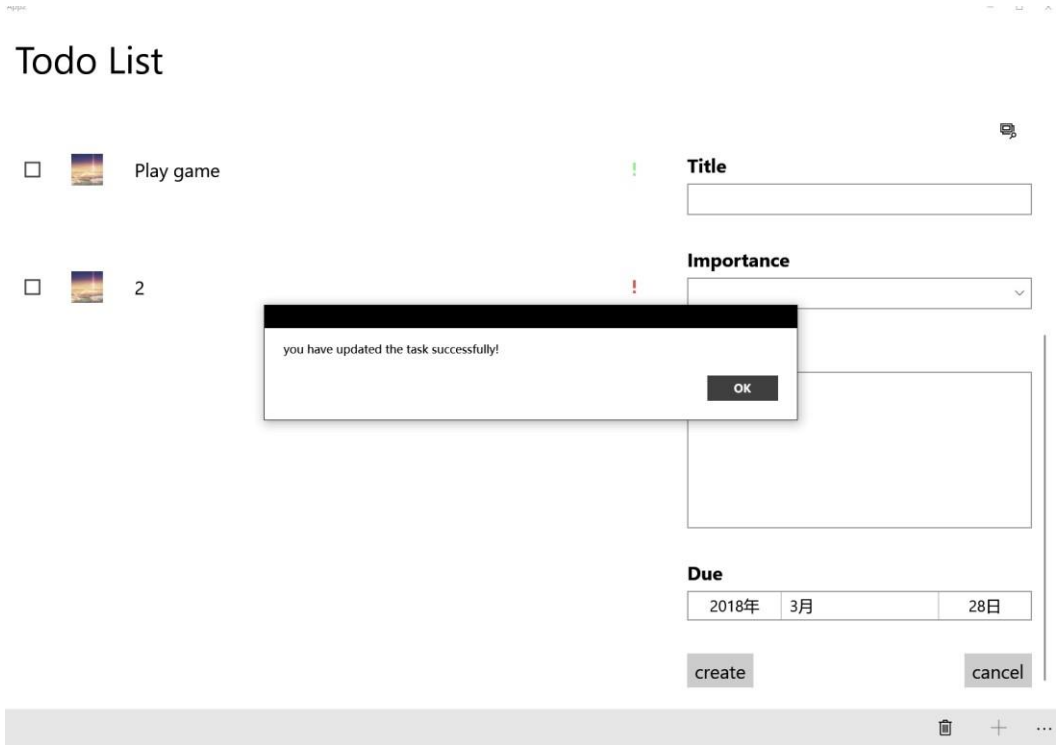
添加操作：



选中以后信息显示：



更新操作：



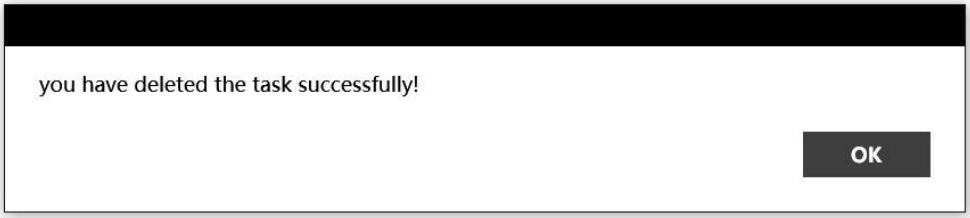
点击显示信息：

窄屏下操作：

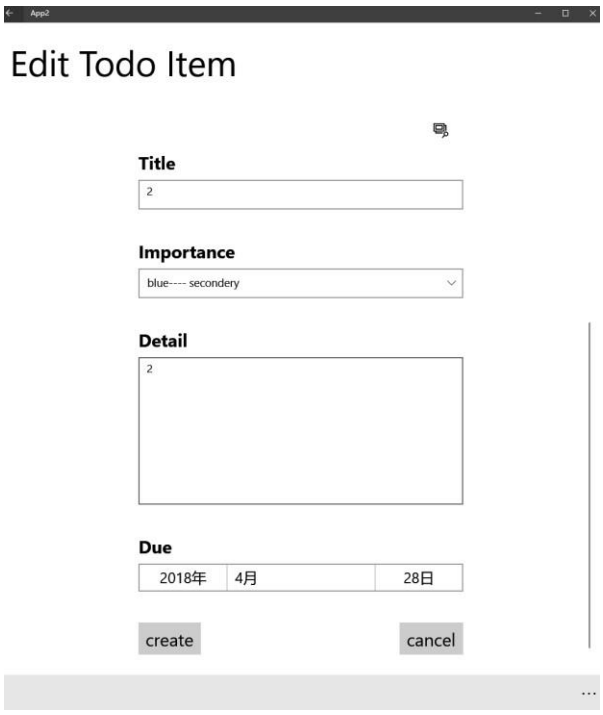
删除：

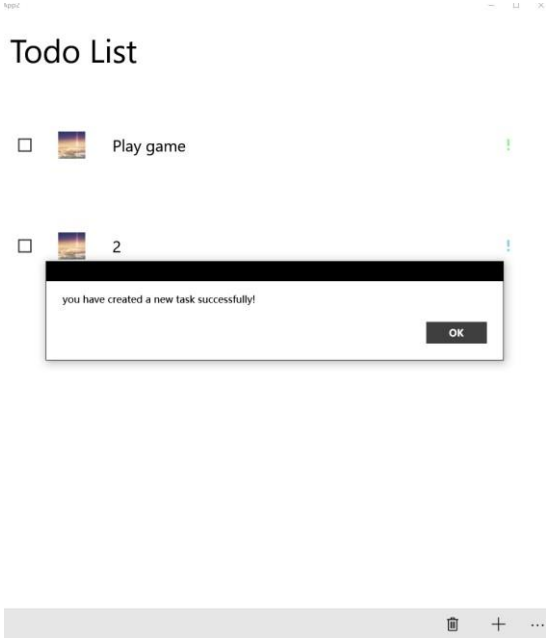
App2

Todo List

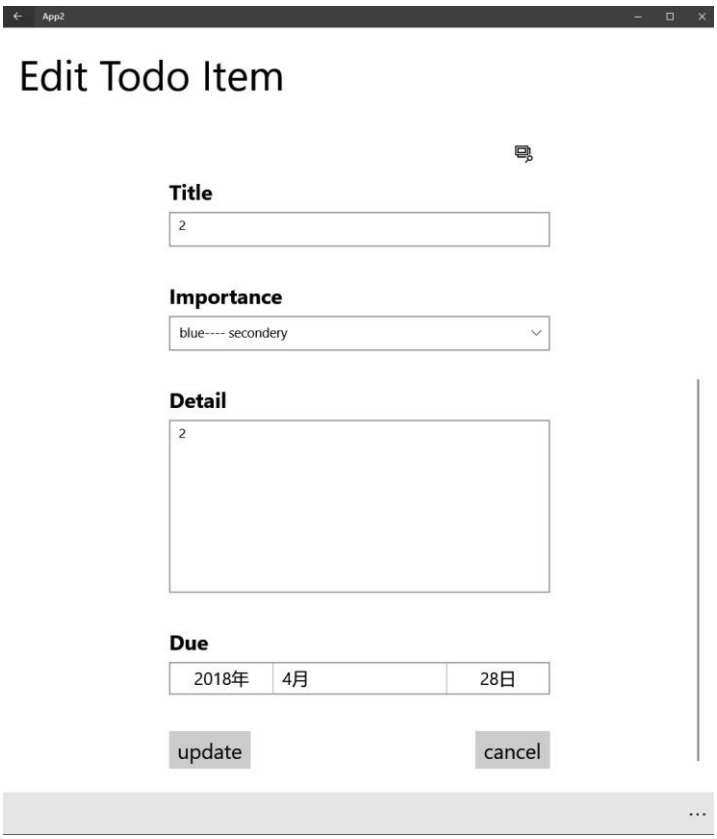


创建：

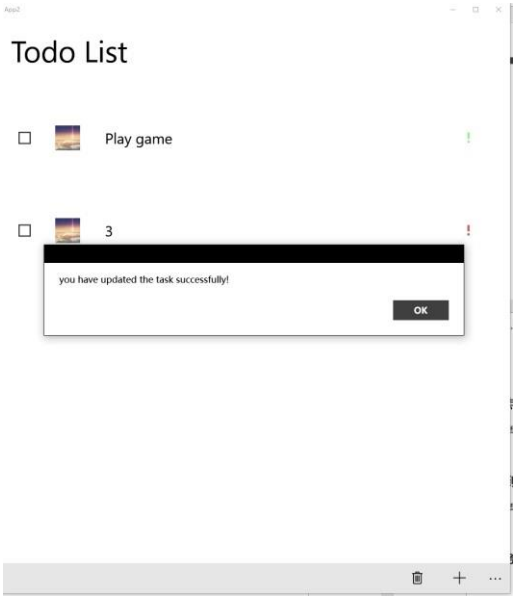

App2
Edit Todo Item
Title
2
Importance
blue---- secondery
Detail
2
Due
2018年
4月
28日
create
cancel


Todo List
Play game
2
you have created a new task successfully!
OK

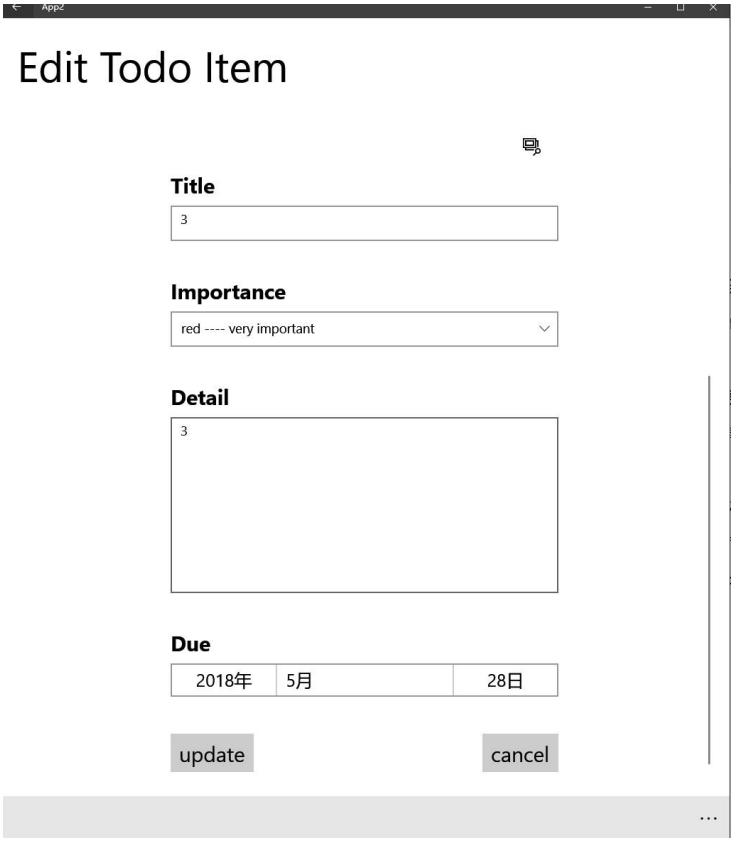
点击后显示信息：


App2
Edit Todo Item
Title
2
Importance
blue---- secondery
Detail
2
Due
2018年
4月
28日
update
cancel
...

更新:

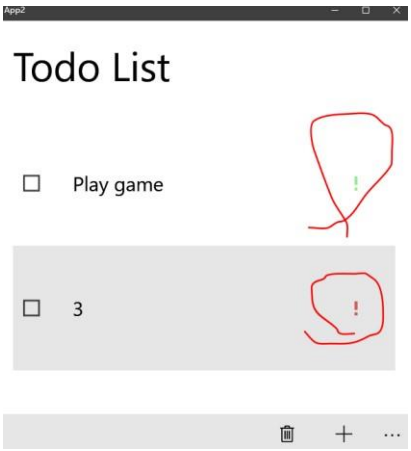


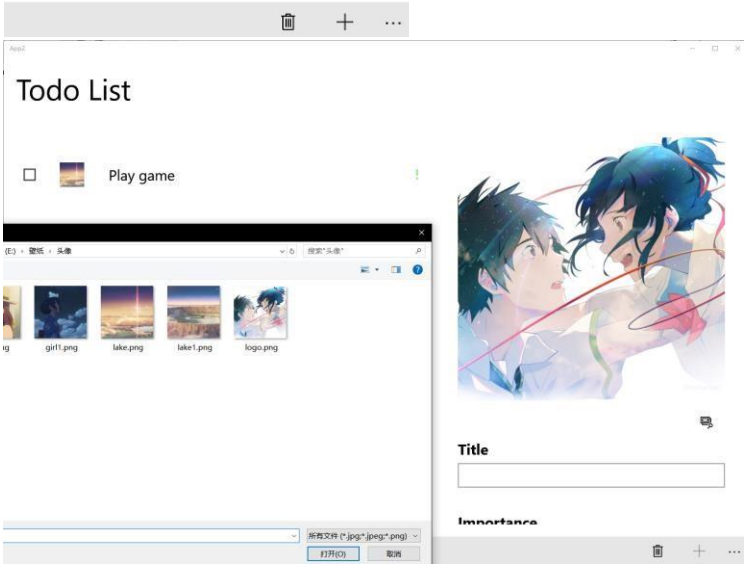
点击后显示：

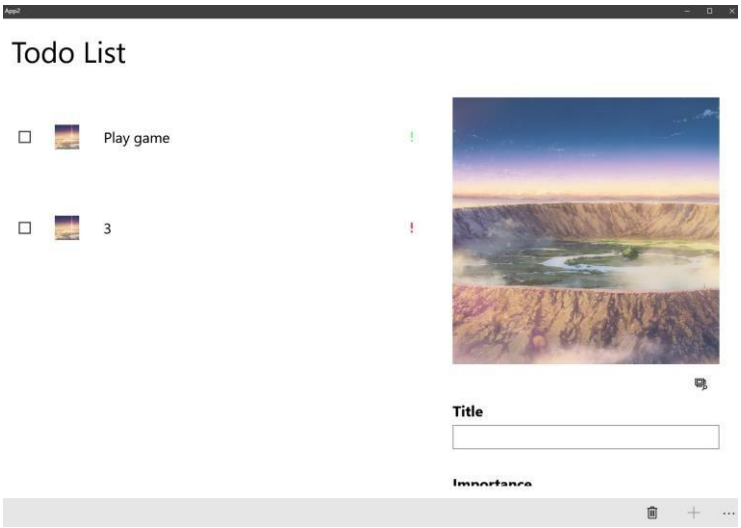


更改之后的却改了，对应信息是对的。

四、亮点与改进（可选）亮点：主要是增加重要性等级、增加图片选择：







重要性等级是通过一个 combobox，可以选择四个等级，红色对应很重要，橘色对应重要，绿色对应不那么重要，蓝色对应不重要：

Xaml：

复选框：

```
<ComboBox Grid.Row="5" x:Name="importance" SelectedIndex="-1" Width="440" Height="40" SelectionChanged="ComboBox_SelectionChanged">
    <x:String>red ---- very important</x:String>
    <x:String>orange ---- important</x:String>
    <x:String>green ---- not so important</x:String>
    <x:String>blue ---- secondery</x:String>
</ComboBox>
```

感叹号：

```
<FontIcon Gr                    #xEDB1;" Foreground="{x:Bind Importance, Converter={StaticResource ImportanceToColor}, Mode=OneWay}"/>
Grid>
```

TextBlock.Text

Cs 代码：

Importance 接口

```
public class TaskItem: INotifyPropertyChanged
{
    private bool isChecked { get; set; }
    private string title { get; set; }
    private string detail { get; set; }
    private int importance { get; set; }
    private DateTimeOffset dueTime { get; set; }

    public bool IsChecked
    {
        get { return this.isChecked; }
        set {
            isChecked = value;
            OnPropertyChanged();
        }
    }

    public string Title
    {
        get { return this.title; }
        set
        {
            title= value;
            OnPropertyChanged();
        }
    }

    public int Importance
    {
        get { return importance; }
        set
        {
            importance = value;
            OnPropertyChanged();
        }
    }
```

值转换器：

```
public class importanceToColor : Windows.UI.Xaml.Data.IValueConverter
{
public object Convert(object value, Type targetType,
object parameter, string language)
{
   // value is the data from the source object.
   int color = (int)value;
   SolidColorBrush red = new SolidColorBrush(Colors.IndianRed);
   SolidColorBrush orange = new SolidColorBrush(Colors.Orange);
   SolidColorBrush green = new SolidColorBrush(Colors.LightGreen);
   SolidColorBrush blue = new SolidColorBrush(Colors.SkyBlue);
   if (color == 0)
   {
      return red;
   }
   else if (color ==1)
   {
      return orange;
   } else if (color ==2)
   {
      return green;
   }
   else
   {
      return blue;
   }
}
```

图片选择实现：

主要是按钮绑定事件，然后事件中打开资源管理器，进行文件选择，可以使用 jpg、jepg、png 文件，选择其他文件无效：

```
private async void ChooseImageButton_Click(object sender, RoutedEventArgs e)
{
   var picker = new Windows.Storage.Pickers.FileOpenPicker();
   picker.ViewMode = Windows.Storage.Pickers.PickerViewMode.Thumbnail;
   picker.SuggestedStartLocation =
      Windows.Storage.Pickers.PickerLocationId.ComputerFolder;
   picker.FileTypeFilter.Add(".jpg");
   picker.FileTypeFilter.Add(".jpeg");
   picker.FileTypeFilter.Add(".png");

   Windows.Storage.StorageFile file = await picker.PickSingleFileAsync();
   if (file != null)
   {
      using (IRandomAccessStream fileStream = await file.OpenAsync(Windows.Storage.FileAccessMode.Read))
      {
         // Set the image source to the selected bitmap
         BitmapImage bitmapImage = new BitmapImage();
         await bitmapImage.SetSourceAsync(fileStream);
         photo.Source = bitmapImage;
      }
   }
}
```

五、遇到的问题

这里自己要就讲一下自己这次实验遇到的比较大的几个问题：

1. update 更新操作以后无法实时显示

这个其实是因为自己没有给 tastItem 类添加 INotifyPropertyChanged 接口，这个接口是用来检测这个类的成员变量是否被更改，如果有更改就通知客户端（View 界面）做出更新，没有这个类的话，在 update 函数中的更改没对 ObservableCollection（这个类可以检测添加删除移动等操作并且通知客户端做出更新）的某个元素进行增加或者删除或者移动，所以最后没办法向客户端做出通知，但是如果借助 INotifyPropertyChanged 接口，那么仅仅是改变 ObservableCollection 实例的元素的属性就可以做出相应通知了。

2. 有时候 vs 总是报错：无法识别某个名称：这时候首先右键项目、清理、部署，如果这时候不报错就是 vs 的问题，否则是自己的问题

六、思考与总结

写多了自己恐怕也记不住把，就写一两点，然后争取每次实验写出不同的感想，得到不同的收获吧：

初入 uwp 的坑，自己的**第一个**收获：官方文档永远是最好用的，没有之一，好好看 uwp 微软官方文档，上面的 demo 很多，例子很多，讲解很到位，好好学习是可以很有收获的，这次的作业自己的数据绑定、值转换器、各种各样 api 的使用都是在官网上学习的。

第二个收获：Github 上有很多很好的示例，自己可以好好学习一下

然后就是：学习还是要有耐心，有 bug 有问题第一时间静下心来想为什么，而不是抱怨、问别人，拒绝做伸手党！