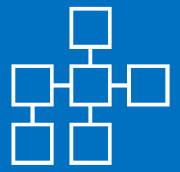
访问文件



大纲

1. 文本选择器访问文件
2. 编程方式访问文件
3. 读取和写入文件

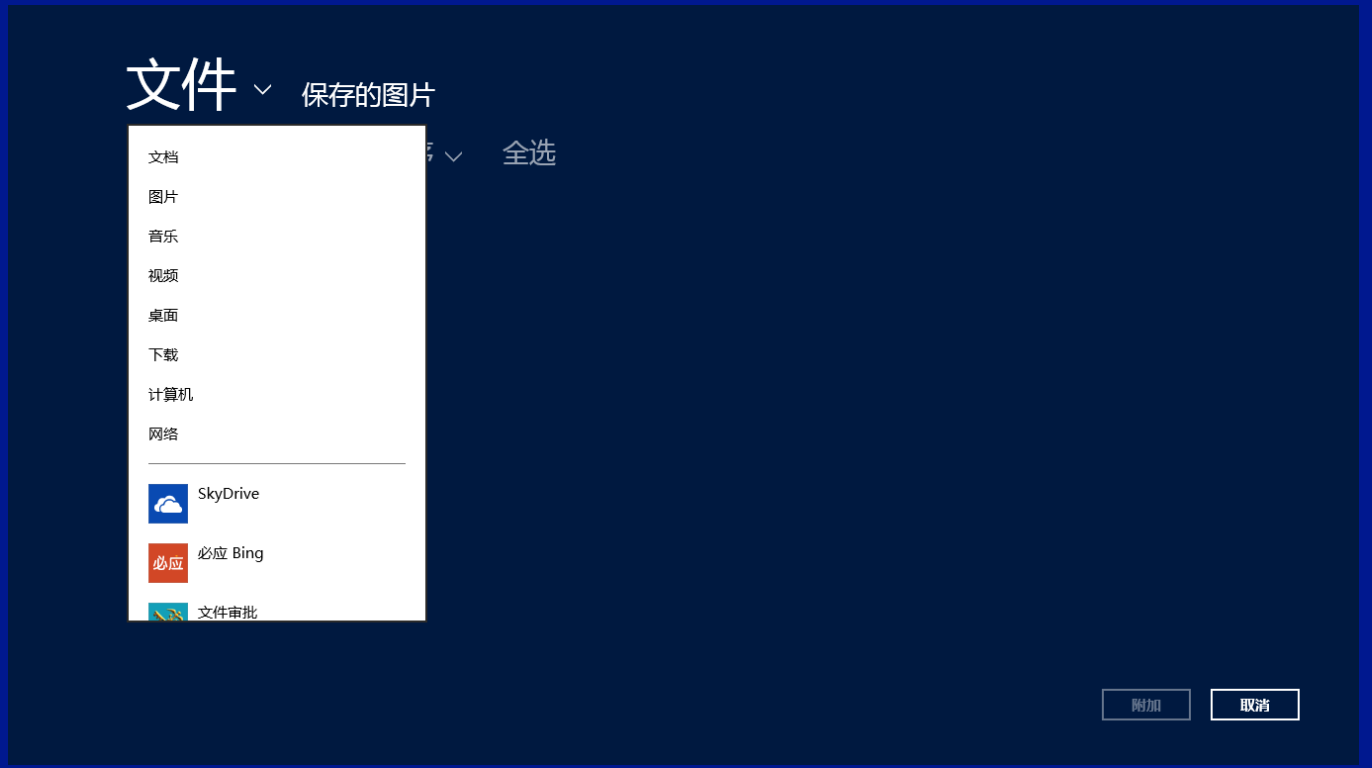
文本选择器访问文件

文件选择器

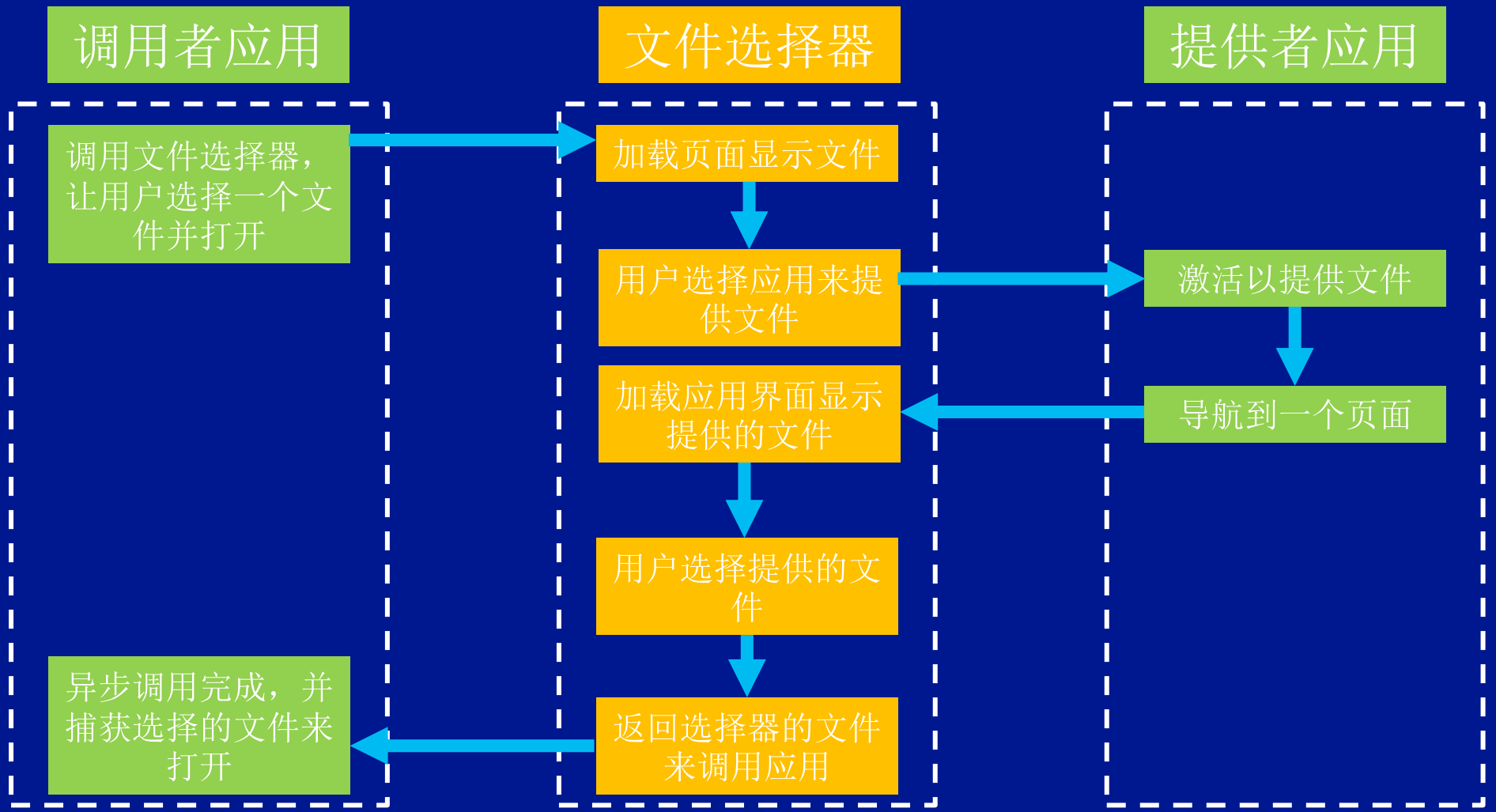
使用文件选取器通过让用户选取文件和文件夹来访问文件和文件夹

文件选取器在屏幕顶部和底部都有显示信息的区域，用于定向用户并在用户访问或保持文件时提供一致的体验。

显示的信息包括：
- 当前位置（位于左上角）
- 用户选择的项目栏（沿着底部）
- 用户可以浏览的位置的下拉列表（可以从左上角的下拉符号中选择）



文件选择器工作原理


调用者应用
文件选择器
提供者应用
调用文件选择器，让用户选择一个文件并打开
加载页面显示文件
用户选择应用来提供文件
激活以提供文件
加载应用界面显示提供的文件
导航到一个页面
用户选择提供的文件
异步调用完成，并捕获选择的文件来打开
返回选择器的文件来调用应用

核心API--FileOpenPicker类

FileOpenPicker：表示允许用户选择和打开文件的 UI 元素。

重要方法	描述
PickMultipleFilesAsync	显示文件选择器，用户可以选择多个文件。
PickSingleFileAsync	显示文件选择器，用户可以选择单个文件。

重要属性	访问类型	描述
CommitButtonText	读取/写入	获取/设置文件打开选择器提交按钮的标签文本
FileTypeFilter	只读	获取文件打开选择器显示的文件类型集合
SettingsIdentifier	读取/写入	获取/设置与文件打开选择器状态相关联的设置标识符
SuggestedStartLocation	读取/写入	获取/设置文件打开选择器寻找文件呈现给用户的初始位置
ViewMode	读取/写入	获取/设置文件打开选择器使用显示项的视图模型

核心API--FolderPicker类

FolderPicker：表示允许用户选择文件夹的 UI 元素。

重要方法	描述
PickSingleFolderAsync	显示 folderPicker 对象，以便用户可以选择文件夹。

重要属性	访问类型	描述
CommitButtonText	读取/写入	获取或设置文件夹选取器的提交按钮的标签文本
FileTypeFilter	只读	获取文件夹选择器显示的文档类型的集合
SettingsIdentifier	读取/写入	获取或设置与当前 FolderPicker 实例关联的设置标识符
SuggestedStartLocation	读取/写入	获取或设置文件夹选取器在其中查找要呈现给用户的文件夹的初始位置
ViewMode	读取/写入	获取或设置文件夹选择器用于显示项目的视图模式

核心API--StorageFile类

StorageFile：提供有关文件及其内容和操作它们的方式的信息。

重要方法	描述
OpenAsync	在文件中打开一个随机访问流
RenameAsync(String)	重命名当前文件
MoveAsync(IStorageFolder)	将当前文件移到指定文件夹
DeleteAsync()	删除当前文件

重要属性	访问类型	描述
Attributes	只读	获取文件特性
DateCreated	只读	获取创建当前文件的日期和时间
FileType	只读	获取文件的类型（文件名扩展）
Name	只读	获取文件的名称，包括文件扩展名
Properties	只读	获取一个对象，该对象提供对文件的相关内容属性的访问。

注意：

贴靠状态下显示文件选择器，文件选择器将不显示并引发异常。
在调用文件选择器前确保应用未进行贴靠或将其取消贴靠。

实现选取一个文件

```
FileOpenPicker openPicker = new FileOpenPicker();
openPicker.ViewMode = PickerViewMode.Thumbnail; // 设置文件选择器用于显示项目的视图模型
openPicker.SuggestedStartLocation = PickerLocationId.PicturesLibrary; // 呈现给用户文件初始化位置
openPicker.FileTypeFilter.Add(".jpg");
openPicker.FileTypeFilter.Add(".jpeg");
openPicker.FileTypeFilter.Add(".png");
StorageFile file = await openPicker.PickSingleFileAsync();
if (file != null) {
// 现在应用程序具有读取/写入选择的文件的权限
OutputTextBlock.Text = file.Name;
} else {
OutputTextBlock.Text = “操作取消"; }
```

实现选取一个文件夹

```
FolderPicker folderPicker = new FolderPicker();
folderPicker.SuggestedStartLocation = PickerLocationId.Desktop;
folderPicker.FileTypeFilter.Add(".docx");
folderPicker.FileTypeFilter.Add(".xlsx");
folderPicker.FileTypeFilter.Add(".pptx");
StorageFolder folder = await folderPicker.PickSingleFolderAsync();
if (folder != null) {
// 现在应用可以读取/写入选择的文件夹中所有内容（包括其他子文件夹内容）
StorageApplicationPermissions.FutureAccessList.AddOrReplace("PickedFolderToken", folder);
OutputTextBlock.Text = folder.Name;
} else {
OutputTextBlock.Text = “操作取消";}
```

实现保存文件

使用 FileSavePicker 让用户指定他们想保存应用内容的名称和位置。

步骤 1: 确保你可以调用文件选取器

在尝试创建和显示文件选取器之前检查应用的 ApplicationView.Value 并尝试取消贴靠。

```
internal bool EnsureUnsnapped()
{
// 应用在贴靠状态下FilePicker API不能正常运行
// 当贴靠时想显示FilePicker，必须首先调整到非贴靠视图
bool unsnapped = ((ApplicationView.Value != ApplicationViewState.Snapped) ||
ApplicationView.TryUnsnap());
if (!unsnapped)
 {
        NotifyUser("Cannot unsnap the sample.", NotifyType.StatusMessage); }
        return unsnapped;
}
```

实现保存文件

步骤 2:创建并自定义 FileSavePicker

FileSavePicker 方便用户指定保存文件的名称、文件类型以及位置。 可以通过在创建的 FileSavePicker 上设置属性来自定义文件选取器。

```
FileSavePicker savePicker = new FileSavePicker();
savePicker.SuggestedStartLocation =
PickerLocationId.DocumentsLibrary;
// 下拉文件类型，用户可以保存文件
savePicker.FileTypeChoices.Add("Plain Text", new List<string>()
{ ".txt" });
// 设置一个默认文件名称
savePicker.SuggestedFileName = "New Document";
```

实现保存文件

步骤 3:显示 FileSavePicker 以保存文件

在创建并自定义文件选取器之后，用户通过调用 savePicker.PickSaveFileAsync 来保存文件。

用户指定名称、文件类型和位置并确认保存文件之后，PickSaveFileAsync 返回一个表示已保存文件的 StorageFile 对象。可以通过使用 await 运算符来捕获和处理此文件。

```
StorageFile file = await savePicker.PickSaveFileAsync();
if (file != null)
{
 // 防止更新远程版本文件，直到完成改变并且调用CompleteUpdatesAsync.
CachedFileManager.DeferUpdates(file);
// 写入到文件
await FileIO.WriteTextAsync(file, file.Name);
// 让Windows知道已经完成了对文件的更新，其他应用可以更新文件的远程版本
 // 完成更新就可以让Windows要求用户输入了
FileUpdateStatus status = await CachedFileManager.CompleteUpdatesAsync(file);
if (status == FileUpdateStatus.Complete)
 {
OutputTextBlock.Text = "File " + file.Name + " was saved.";
}
else { OutputTextBlock.Text = "File " + file.Name + " couldn't be saved."; }
}
else { OutputTextBlock.Text = "Operation cancelled."; }
```

Demo

文本选择器访问文件

编程方式访问文件

编程方式访问某个特定位置的文件和文件夹，可枚举或访问该位置的多有文件。

枚举某位置所有文件和文件夹

```
StorageFolder picturesFolder = KnownFolders.PicturesLibrary;

StringBuilder outputText = new StringBuilder();

IReadOnlyList<StorageFile> fileList = await picturesFolder.GetFilesAsync();

outputText.AppendLine("Files:");

foreach (StorageFile file in fileList)

{

outputText.Append(file.Name + "\n");

}

IReadOnlyList<StorageFolder> folderList = await picturesFolder.GetFoldersAsync();

outputText.AppendLine("Folders:");

foreach (StorageFolder folder in folderList)

{

outputText.Append(folder.DisplayName + "\n");

}
```

查询某位置中文件并枚举匹配的文件

```
StorageFolder picturesFolder = KnownFolders.PicturesLibrary;

StorageFolderQueryResult queryResult = picturesFolder.CreateFolderQuery(CommonFolderQuery.GroupByMonth);

IReadOnlyList<StorageFolder> folderList = await queryResult.GetFoldersAsync();

StringBuilder outputText = new StringBuilder();

foreach (StorageFolder folder in folderList)

{

    IReadOnlyList<StorageFile> fileList = await folder.GetFilesAsync();

    // 打印该组中文件的月份和日期

    outputText.AppendLine(folder.Name + " (" + fileList.Count + ")");

    foreach (StorageFile file in fileList)

    { // 打印文件的名称

    outputText.AppendLine(" " + file.Name);

    }

}
```

Demo

编程方式访问文件

读取和写入文件

使用 StorageFile 对象读取和写入文件。

核心API--FileIO类

FileIO：为 IStorageFile 类型的对象表示的读取和写入文件提供帮助器方法。

重要方法	描述
ReadBufferAsync	读取指定文件的内容并返回缓冲区。
ReadTextAsync(IStorageFile)	读取指定文件的内容并返回文本。
WriteBufferAsync	从缓冲区将数据写入指定文件。
WriteBytesAsync	将字节数组写入指定文件。
WriteTextAsync(IStorageFile, String)	将文本写入指定文件。
AppendTextAsync(IStorageFile, String)	将文本追加到指定文件。

将文本写入文件

通过调用 FileIO 类的 WriteTextAsync 方法，将文本写入文件。

```
await Windows.Storage.FileIO.WriteTextAsync(sampleFile, "Swift as a shadow");
```

```
//
// 摘要:
//     将文本写入指定文件。
//
// 参数:
//   file:
//     要写入文本的文件。
//
//   contents:
//     要编写的文本。
//
// 返回结果:
//     此方法完成之际不返回任何对象或值。
[Overload("WriteTextAsync")]
public static IAsyncAction WriteTextAsync(IStorageFile file, string contents);
```

从文件读取文本

通过调用 FileIO 类的 ReadTextAsync 方法，从文件读取文本。

```
string text = await Windows.Storage.FileIO.ReadTextAsync(sampleFile) ;
```

```
//
// 摘要:
//     读取指定文件的内容并返回文本。
//
// 参数:
//   file:
//     要读取的文件。
//
// 返回结果:
//     当此方法成功完成时，它将返回作为文本字符串的文件的内容。
[Overload("ReadTextAsync")]
public static IAsyncOperation<string> ReadTextAsync(IStorageFile file);
```

使用缓冲区将字节写入文本

1.获取要写入文件的字节的缓冲区。

```
var buffer = Windows.Security.Cryptography.CryptographicBuffer.ConvertStringToBinary( "What fools these mortals be", Windows.Security.Cryptography.BinaryStringEncoding.Utf8);
```

调用 ConvertStringToBinary 以获取基于随机字符串的字节缓冲区

2.通过调用 FileIO 类的 WriteBufferAsync 方法，将字节从你的缓冲区写入文件。

```
await Windows.Storage.FileIO.WriteBufferAsync(sampleFile, buffer);
```

使用 WriteBufferAsync 将字节从缓冲区写入其 sampleFile

使用缓冲区从文件读取字节

1.通过调用 FileIO 类的 ReadBufferAsync 方法，将字节从你的缓冲区读入到文件。

```
var buffer = await Windows.Storage.FileIO.ReadBufferAsync(sampleFile);
```

调用 ReadBufferAsync 将字节从文件读入到缓冲区

2.使用 DataReader 对象读取 buffer 的长度，并读取缓冲区的内容。

```
DataReader dataReader =
Windows.Storage.Streams.DataReader.FromBuffer(buffer);
string text = dataReader.ReadString(buffer.Length);
```

使用流将文本写入文件

1. 通过调用 StorageFile.OpenAsync 方法，在你的文件上打开流。打开操作完成后，它将返回文件的内容流。

```
var stream = await sampleFile.OpenAsync(Windows.Storage.FileAccessMode.ReadWrite);
```

调用 StorageFile.OpenAsync 方法为文件 (sampleFile) 打开流

2. 通过从 stream 调用 GetOutputStreamAt 方法获取输出流。 将其放到 using 语句中以管理输出流的生存期。

```
using (var outputStream = stream.GetOutputStreamAt(0)) // GetOutputStream获取输出流
{
    // 将流写入文件
    DataWriter dataWriter = new DataWriter(outputStream);
    dataWriter.WriteString("The DataWriter provides method to write to various types, such as DataTimeOffset."); // 调用WriteString方法将文本写入到outputStream中
    // 关闭流
    await dataWriter.StoreAsync();
    await outputStream.FlushAsync();
}
```

使用流从文件读取文本

1.通过调用 StorageFile.OpenAsync 方法，在你的文件上打开流。打开操作完成后，它将返回文件的内容流，获取以后要使用的流的大小。

```
//调用 StorageFile.OpenAsync 方法打开到文件 (sampleFile) 的流
var stream = await sampleFile.OpenAsync(Windows.Storage.FileAccessMode.ReadWrite);
var size = stream.Size;
```

2. 通过调用 stream 的 GetInputStreamAt 方法获取输入流。 将其放到 using 语句中以管理输入流的生存期。

```
// 调用GetInputStream指定 0，以将 inputStream 的位置设置在流的开头
using (var inputStream = stream.GetInputStreamAt(0))
{
      // 使用流从文件读取文本
      DataReader dataReader = new DataReader(inputStream);
      // 读取文本
      uint numBytesLoaded = await dataReader.LoadAsync((uint)size);
      string text = dataReader.ReadString(numBytesLoaded);
 }
```

Demo

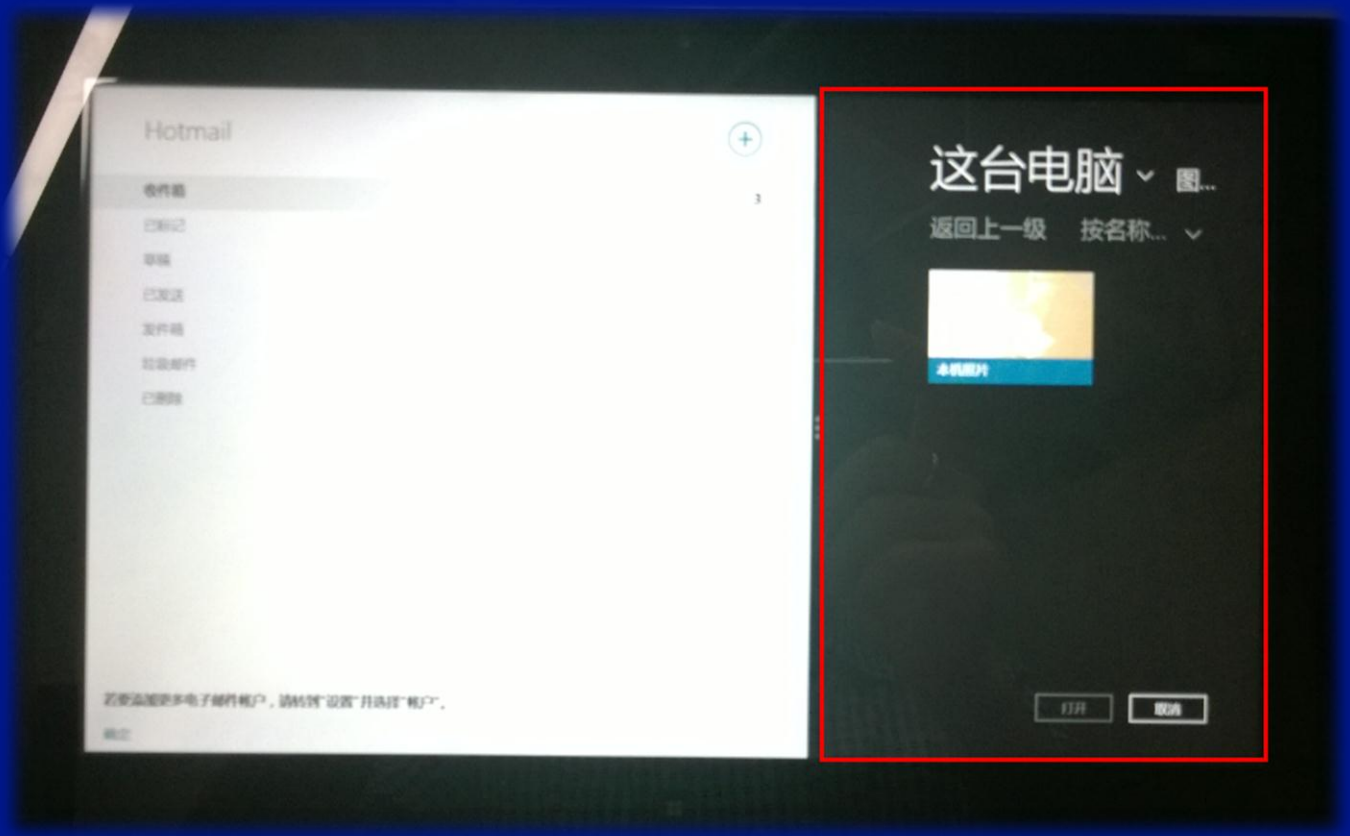
读取和写入文件

Windows 8.1 Pre中更新

文件选择器支持新的显示模式

Windows 8.1 Pre中支持应用程序在贴靠视图状态时直接调用文件选择器。

例如：同一个屏幕中同时使用QQ和邮件两个应用，Windows 8中邮件在贴靠视图是无法直接调用文件选择器，而Windows 8.1中是可以的。



其它的更新项

1	应用内管理用户库
2	添加到Windows索引
3	使用IsEqual进行文件间比较
4	检索父文件或文件夹

关于Windows 8.1 Pre中开发文件相关介绍详细参考：
http://msdn.microsoft.com/zh-cn/library/windows/apps/bg182881.aspx

总结…

1. 通过文件选择器可以选择单个/多个文件或者一个文件夹
2. 通过编程方式访问某特定位置的文件和文件夹
3. 可以将文本读取/写入到文件
4. 使用缓冲区将字节读取/写入到文件
5. 使用流将文本读取/写入到文件

资源

访问数据和文件:

http://msdn.microsoft.com/zh-cn/library/windows/apps/hh758319.aspx#

文件访问示例:

http://code.msdn.microsoft.com/windowsapps/File-access-sample-d723e597/

文件选取器示例:

http://code.msdn.microsoft.com/windowsapps/File-picker-sample-9f294cba/

Files：

http://msdn.microsoft.com/zh-cn/library/windows/apps/bg182881.aspx

Microsoft®