**2.Application类**

每个运行的WPF应用程序都由System.Windows.Application类的一个实例来表示。该类可以跟踪在应用程序中打开的所有窗口，决定何时关闭应用程序，并引发可执行初始化和清除操作的应用程序事件。

# Application有哪些高级应用？

* 控制WPF应用程序的启动
* 控制WPF应用程序的关闭
* 显示初始化界面
* 处理命令行参数
* 实现窗口的访问和交互
* 制作单实例应用程序

# 控制WPF应用程序的启动

## 通过XAML指定程序启动的主窗口

通过Visual Studio创建新的WPF应用程序项目时，VS会自动创建默认名称为App.xaml的XMAL文件,如图所示：



App.xaml文件是WPF应用程序启动时创建Application实例的默认文件，因此该文件不能随意更改文件名，需默认为App.xaml。它的XAML文本如下（VS自动创建，可修改内容）：

<Application x:Class="MyWpfApp.App"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:local="clr-namespace:MyWpfApp"

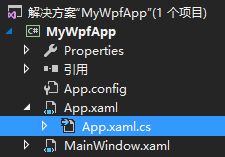
StartupUri="MainWindow.xaml">

<Application.Resources>

</Application.Resources>

</Application>

由此可知，App.xaml文件声明了创建Application实例的类型，如例子中的类型MyWpfApp.App类，其中MyWpfApp是类App的命名空间，一般是应用程序的项目名。而类App的分部类隐藏代码在文件App.xmal.cs中定义：



它的代码很简单：

namespace MyWpfApp

{

/// <summary>

/// App.xaml 的交互逻辑

/// </summary>

public partial class App : Application

{

}

}

即App类直接继承基类Application，无需添加其他属性和方法。

除此之外，App.xaml文件的另一个重要的作用是通过特性StartupUri指定WPF应用程序的主窗口：

StartupUri="MainWindow.xaml"

上面的StartupUri特性指明MainWindow类是主窗口的实例类型。最后，App.xaml文件也可以将WPF应用程序的全局资源存放在<Application.Resources>元素内，如样式和模板。

## 通过代码指定程序启动的主窗口

详见本章最后一节“制作单实例应用程序”。

# 控制WPF应用程序的关闭

WPF应用程序的关闭需要调用方法Application.Shutdown()，它将强制使Main函数里面的Application.Run()方法返回。当然，这不意味应用程序立即关闭，因为可以在Run()方法后面执行其他代码，如Application.Exit事件。WPF程序关闭模式有三种：

* OnLastWindowClose

这是WPF的默认关闭模式，除非关闭所有窗口，不然WPF应用程序将不会隐式自动调用Application.Shutdown()方法关闭程序。若主窗口率先关闭，则Application.MainWindow属性将引用已关闭的主窗口对象，除非重新引用其他窗体对象作为主窗口。

* OnMainWindowClose

这是传统的关闭模式，只要主窗口关闭，Application.Shutdown()方法将被隐式自动调用。在返回Run()方法之前程序会自动关闭剩余的窗口。

* OnExplicitShutdown

该模式需要在代码中显示调用Application.Shutdown()方法才能关闭程序，即使所有的窗体都已经关闭。（如果应用程序是长期运行的后台任务的前端，或者只是希望使用更复杂的逻辑来决定程序何时关闭，该模式或许有意义）。在返回Run()方法之前程序会自动关闭剩余的窗口。

## 通过XAML指定程序关闭的模式

在App.xaml文件中指定：

<Application …

ShutdownMode="OnMainWindowClose">

</Application>

## 通过代码指定程序关闭的模式

详见本章最后一节“制作单实例应用程序”。

# 显示初始化界面

WPF应用程序在启动时，会有延迟现象，这是由CLR初始化.NET环境导致。如果需要在出现主窗体前的延迟阶段显示初始化界面，可以按如下两个简单步骤实现:

## 为项目添加图像文件

通常图像文件格式为.bmp、.png或者.jpg。

## 修改图像文件的Build Action属性

在Solution Explorer中将图像文件的Build Action属性设置为SplashScreen。

当然，这种简单的配置无法实现对初始化界面的大小设置和边框设置，也无法修改初始化界面的褪去时间（默认为500毫秒）。

# 处理命令行参数

为了能够处理命令行参数，需要响应Application.Startup事件。命令行参数是由事件参数StartupEventArgs的属性Args提供。

## 修改App.xaml文件

关联Startup事件的后台代码：

<Application … Startup="Application\_Startup">

## 修改App.xaml.cs文件

在StartupEventArgs事件委托的方法中操作命令行参数：

private void Application\_Startup(object sender, StartupEventArgs e)

{

//读取命令行参数的字符串数组

string[] args = e.Args;

}

# 实现窗口的访问和交互

WPF通过全局的静态对象Application引用Application实例对象。可以通过操作Application实例对象访问任意窗口：

## 访问主窗口

通过Application.Current.MainWindow属性访问主窗口，该主窗口可以是已关闭的窗口。

## 访问当前仍然处于打开状态的所有窗口

通过Application.Current.Windows属性访问所有打开的窗口。

# 制作单实例应用程序

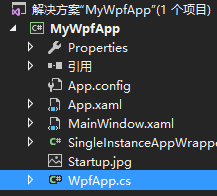
单实例应用程序是指只能有一个Application实例对象的应用程序。默认的WPF应用程序不是单实例的应用程序。多次点击非单实例的应用程序的可执行文件，将创建多个相互独立的WPF应用程序，即多个Application对象。而单实例应用程序会检测Application对象是否存在，若不存在，则创建新的Application对象；否则，跳过创建Application对象的步骤。对于基于文档的应用程序，创建单实例应用程序是非常合理的，如Word应用程序。下面介绍创建单实例的应用程序步骤：

## 添加Microsoft.VisualBasic.dll程序集引用

因为WPF本身没有创建单实例应用程序的功能集，因此通过包含该功能的VB旧版本程序集Microsoft.VisualBasic.dll实现这一目的。

## 自定义Application的单实例类型

单实例应用程序创建Application实例的简单方案是自定义一个继承自Application的类型，而不再使用XAML文件里面定义的Application。



本例创建一个名为WpfApp.cs的文件，代码如下：

namespace MyWpfApp

{

public class WpfApp : System.Windows.Application

{

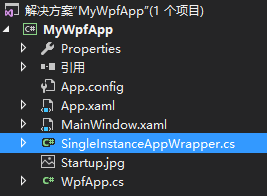
}

}

同样，也可以直接利用VS自定义的App.xaml.cs文件中定义的类型。

## 创建单实例应用程序的包装器

自定义单实例包装器类型，该类型必须继承自位于Miscrosoft.VisualBasic.ApplicationServices命名空间下的WindowsFormsApplicationBase基类。



本例创建一个名为SingleInstanceAppWrapper.cs的文件，代码如下：

public class SingleInstanceAppWrapper : WindowsFormsApplicationBase

{

//Application句柄

private WpfApp app = null;

public SingleInstanceAppWrapper()

{

//使能单实例模式

this.IsSingleInstance = true;

}

//重写多实例构造函数

protected override void

OnStartupNextInstance(StartupNextInstanceEventArgs eventArgs)

{

//base.OnStartupNextInstance(eventArgs);

//获取焦点

app.MainWindow.WindowState = System.Windows.WindowState.Normal;

app.MainWindow.Focus();

}

//重写启动函数

protected override bool OnStartup(StartupEventArgs eventArgs)

{

// return base.OnStartup(eventArgs);

//创建Application实例

app = new WpfApp();

//指定应用程序关闭模式

app.ShutdownMode =

System.Windows.ShutdownMode.OnMainWindowClose;

//创建主窗体

MainWindow mainWindow = new MainWindow();

mainWindow.WindowStartupLocation =

System.Windows.WindowStartupLocation.CenterScreen;

app.Run(mainWindow);

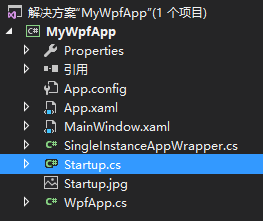
return false;

}

}

## 自定义Startup启动类型

因为应用程序需要在创建Application实例之前创建包装器SingleInstanceAppWrapper的实例，因此只能通过传统的Main函数入口启动。



本例创建一个名为Startup.cs的启动文件，代码如下：

public class Startup

{

[STAThread]

public static void Main(string[] args)

{

SingleInstanceAppWrapper wrapper =

new SingleInstanceAppWrapper();

wrapper.Run(args);

}

}

最后，修改项目的启动入口点：

