## 【转帖】 C# 获取当前应用程序的名称和路径

原文请见：  
http://www.cnblogs.com/zyqduron/archive/2009/01/05/1369668.html  
  
string str1 =Process.GetCurrentProcess().MainModule.FileName;//可获得当前执行的exe的文件名。  
string str2=Environment.CurrentDirectory;//获取和设置当前目录（即该进程从中启动的目录）的完全限定路径。  
//备注按照定义，如果该进程在本地或网络驱动器的根目录中启动，则此属性的值为驱动器名称后跟一个尾部反斜杠（如“C:\”）。如果该进程在子目录中启动，则此属性的值为不带尾部反斜杠的驱动器和子目录路径（如“C:\mySubDirectory”）。  
string str3=Directory.GetCurrentDirectory();//获取应用程序的当前工作目录。  
string str4=AppDomain.CurrentDomain.BaseDirectory;//获取基目录，它由程序集冲突解决程序用来探测程序集。  
string str5=Application.StartupPath;//获取启动了应用程序的可执行文件的路径，不包括可执行文件的名称。  
string str6=Application.ExecutablePath;//获取启动了应用程序的可执行文件的路径，包括可执行文件的名称。  
string str7=AppDomain.CurrentDomain.SetupInformation.ApplicationBase;//获取或设置包含该应用程序的目录的名称。  
1.   System.Diagnostics.Process.GetCurrentProcess().MainModule.FileName  
   获取模块的完整路径。  
2.   System.Environment.CurrentDirectory  
     获取和设置当前目录(该进程从中启动的目录)的完全限定目录。  
3.   System.IO.Directory.GetCurrentDirectory()  
      获取应用程序的当前工作目录。这个不一定是程序从中启动的目录啊，有可能程序放在C:\www里,这个函数有可能返回C:\ Documents and Settings\ZYB\,或者C:\Program Files\Adobe\,有时不一定返回什么东东，我也搞不懂了。  
4.  System.AppDomain.CurrentDomain.BaseDirectory  
     获取程序的基目录。  
5.  System.AppDomain.CurrentDomain.SetupInformation.ApplicationBase  
获取和设置包括该应用程序的目录的名称。  
6.  System.Windows.Forms.Application.StartupPath  
     获取启动了应用程序的可执行文件的路径。效果和、一样。只是返回的字符串后面多了一个"\"而已  
7.  System.Windows.Forms.Application.ExecutablePath  
     获取启动了应用程序的可执行文件的路径及文件名，效果和一样。

[应用程序唯一运行实例C#代码实现（转）](http://www.cnblogs.com/mywebname/articles/1188479.html)

发布日期： 2006-06-30 | 更新日期： 2006-07-04  
作者：郑佐  
适用于： Windows 操作系统   
.NET Framework 1.x，2.0运行时环境   
.NET Windows开发  
  
**摘要：**本文阐述了在基于.NET平台的应用程序开发中如何实现唯一应用程序运行实例，对几种实现方式进行分析测试比较，从而寻找一种合适的处理方式。单击[此处](http://blog.csdn.net/zhzuo/archive/2006/07/04/874745.aspx" \t "_blank)才查看本文的示例代码。  
  
**内容索引**[概述](http://blog.csdn.net/zhzuo/archive/2006/06/30/857405.aspx#sec1)  
[进程匹配](http://blog.csdn.net/zhzuo/archive/2006/06/30/857405.aspx#sec2)  
[进程互斥](http://blog.csdn.net/zhzuo/archive/2006/06/30/857405.aspx#sec3)  
[运行标志](http://blog.csdn.net/zhzuo/archive/2006/06/30/857405.aspx#sec4)  
[功能测试](http://blog.csdn.net/zhzuo/archive/2006/06/30/857405.aspx#sec5)  
[更多资源](http://blog.csdn.net/zhzuo/archive/2006/06/30/857405.aspx#sec6)  
  
**概述**  
在开发一些应用系统的时候，由于程序内在的一些特征，系统的某些组成子程序只允许运行一个应用程序实例，以保证业务和数据处理安全。本文将从实际应用角度来分析其实现原理，对三种实现方式进行测试比较，从而确定一种合适的实现方法。文章的例子使用C#语言进行描述。   
  
**进程匹配**  
对于每一个应用程序运行实例都会包含该实例的一个或多个进程，而且在程序运行过程中可能会动态的创建或销毁进程，或者访问其他现有进程进行通信。不难发 现，在程序最先初始化的那一刻只有一个进程运行，而且应用程序进程生命周期最大进程名称集合是不变的。因此，在应用程序初始化的时候，可以根据进程关键信 息检查系统进程列表是否存在同当前初始化进程匹配的进程来确定是否已经运行进程实例。  
逻辑处理步骤如下，   
1．初始化应用程序，启动程序初始化进程；   
2．访问系统进程列表，根据初始化进程关键信息进行匹配查找；   
3．没有找到匹配进程（这一步是不会发生的，因为当前初始化进程也在列表中，不过还要看获取进程列表的实现代码怎么写），继续初始化进程，程序初始化完成运行。   
4．找到第一个匹配进程，判断找到的进程ID是否同初始化进程ID相同；   
5．如果第一个匹配进程ID同初始化进程ID相同，则为当前初始化进程，继续查找；   
6．没有找到第二个匹配进程，表明当前运行的是首个实例，继续初始化进程，程序初始化完成运行。   
7．找到第二个，表明已有一个实例在运行，停止当前程序初始化，提示已有应用程序运行。   
8．如果找到第一个匹配进程ID不同，表明已有一个实例在运行，停止当前程序初始化，提示已有应用程序运行。   
  
可见上面的逻辑实现中用于进程匹配的信息是关键，选择不当功能就无法实现。在这个实例中笔者使用了应用程序完全文件名称作为关键信息。  
在代码中首先需要引用下面命名空间，以调用WinAPI函数。   
using System.Runtime.InteropServices;   
把实现唯一运行实例功能的类名取为SingleInstance，在类前面加static关键字为C# 2.0新增的语言特征。   
public static class SingleInstance {}   
使用GetRunningInstance静态方法获取应用程序进程实例,如果没有匹配进程，返回Null值，   
public static Process GetRunningInstance()   
{   
        Process currentProcess = Process.GetCurrentProcess(); //获取当前进程   
        //获取当前运行程序完全限定名   
        string currentFileName = currentProcess.MainModule.FileName;   
        //获取进程名为ProcessName的Process数组。   
        Process[] processes = Process.GetProcessesByName(currentProcess.ProcessName);   
        //遍历有相同进程名称正在运行的进程   
        foreach (Process process in processes)   
        {   
                if (process.MainModule.FileName == currentFileName)   
                {   
                        if (process.Id != currentProcess.Id) //根据进程ID排除当前进程   
                                return process;//返回已运行的进程实例   
                }   
        }   
        return null;   
}   
  
接下来调用两个WinAPI，其功能将在包装方法中描述，   
[DllImport("User32.dll")]   
private static extern bool ShowWindowAsync(IntPtr hWnd, int cmdShow);   
[DllImport("User32.dll")]   
private static extern bool SetForegroundWindow(IntPtr hWnd);   
  
定义类成员辅助变量，   
private const int WS\_SHOWNORMAL = 1;   
  
以上的方法声明为私有，对其进一步包装，HandleRunningInstance静态方法为获取应用程序句柄，设置应用程序为前台运行，并返回bool值。   
public static bool HandleRunningInstance(Process instance)   
{   
        //确保窗口没有被最小化或最大化   
        ShowWindowAsync(instance.MainWindowHandle, WS\_SHOWNORMAL);   
        //设置为foreground window   
        return SetForegroundWindow(instance.MainWindowHandle);   
}   
  
对上面的方法创建一个重载版本，使调用代码更加简洁，   
public static bool HandleRunningInstance()   
{   
        Process p = GetRunningInstance();   
        if (p != null)   
        {   
                HandleRunningInstance(p);   
                return true;   
        }   
        return false;   
}   
  
上面的方法实现获取已经运行的进程实例的句柄，并获取其焦点显示到前台，这个很有用，在其他实现方式中也可以用到。  
  
在Main函数中调用下面代码实现单一应用程序实例，   
Process p = SingleInstance.GetRunningInstance();   
if (p != null) //已经有应用程序副本执行   
{   
        SingleInstance.HandleRunningInstance(p);   
}   
else //启动第一个应用程序   
{   
        Application.Run(new MainForm());   
}   
  
简洁的调用为，   
if (SingleInstance.HandleRunningInstance()== false)   
{   
        Application.Run(new MainForm());   
}   
  
可见，在上面的实现过程中，由于关键信息采用应用程序的完整文件名，因此在文件名称或路径名称修改后，以上实现就会失效。   
  
**进程互斥**  
在这个实现方式中需要定义一个进程同步基元，可以理解为临界资源，该资源只允许一个进程使用。根据这一点实现应用程序唯一运行实例就比较简单了。  
实现步骤如下，   
1．应用程序初始化访问该同步基元；   
2．可以访问，说明该同步基元未被使用，也就是说没有应用程序实例运行，使用同步基元，可以继续初始化成为第一个运行实例。   
3．不可以访问，说明该同步基元已被使用，也就是说已有应用程序实例运行，停止当前程序初始化，提示已有应用程序运行。   
4．应用程序实例退出释放同步基元占用。   
  
在代码中笔者使用System.Threading.Mutex类实现同步基元，实现应用程序实例之间互斥功能。Mutex默认名字取Assembly.GetEntryAssembly().FullName。  
在类成员中声明同步基元，   
private static Mutex mutex = null;   
  
CreateMutex静态方法创建应用程序进程Mutex，返回创建结果为true表示创建成功，false失败。   
public static bool CreateMutex()   
{   
        return CreateMutex(Assembly.GetEntryAssembly().FullName);   
}   
  
实现其重载方法，让用户可以自定义Mutex名字，   
public static bool CreateMutex(string name)   
{   
        bool result = false;   
        mutex = new Mutex(true, name, out result);   
        return result;   
}   
  
对应的释放Mutex资源方法为，   
public static void ReleaseMutex()   
{   
        if (mutex != null)   
        {   
                mutex.Close();   
        }   
}   
  
在Main函数中调用下面代码实现单一应用程序实例，   
if (SingleInstance.CreateMutex())   
{   
        Application.Run(new MainForm());   
        SingleInstance.ReleaseMutex();   
}   
else   
{   
        MessageBox.Show("程序已经运行！");   
}   
  
可见，在上面的实现过程中，Mutex名字是同步基元的唯一标识，如果刚好有不同的应用程序使用了相同名称的Mutex，那不同的应用程序实例也会出现互斥现象。   
  
**运行标志**  
使用应用程序运行标志简单来讲就是在程序初始化的时候设置一个标志表示程序已运行，在程序运行结束的时候删除该标志。  
基本步骤如下，   
1．应用程序初始化检查运行标志是否已经设置；   
2．发现已经设置，说明已有应用程序实例运行，停止当前程序初始化，提示已有应用程序运行。 3．发现没有设置，说明没有应用程序实例运行，继续当前程序初始化。   
4．退出应用程序时删除该运行标志。   
  
对于标志存储载体可以使用注册表、数据库或外部文件等，这里的代码使用外部文件实现。对存放标志的文件目录选择C:\Documents and Settings\All Users\Application Data，也可以是C:\Program Files\Common Files。  
声明类成员标志文件名称变量，   
private static string runFlagFullname = null;   
  
初始化程序运行标志，如果设置成功，返回true，已经设置返回false，设置失败将抛出异常，   
public static bool InitRunFlag()   
{   
        if (File.Exists(RunFlag))   
        {   
                return false;   
        }   
        using (FileStream fs = new FileStream(RunFlag, FileMode.Create))   
        {   
        }  
        return true;   
}   
  
释放初始化程序运行标志，如果释放失败将抛出异常，   
public static void DisposeRunFlag()   
{   
        if (File.Exists(RunFlag))   
        {   
                File.Delete(RunFlag);   
        }   
}   
  
获取或设置程序运行标志，必须符合Windows文件命名规范，   
public static string RunFlag   
{   
        get   
        {   
                if(runFlagFullname == null)   
                {   
                        string assemblyFullName = Assembly.GetEntryAssembly().FullName;   
                        string path = Environment.GetFolderPath(Environment.SpecialFolder.CommonApplicationData);   
                        runFlagFullname = Path.Combine(path, assemblyFullName);   
                }   
                return runFlagFullname;   
        }   
        set   
        {   
                runFlagFullname = value;   
        }   
}   
  
在Main函数中调用下面代码实现单一应用程序实例，   
if (SingleInstance.InitRunFlag())   
{   
        Application.Run(new MainForm());   
        SingleInstance.DisposeRunFlag();   
}   
else   
{   
        MessageBox.Show("程序已经运行！");   
}   
  
可见，在上面的实现过程中，需要访问文件IO，因此有可能会出现异常，对异常需要进行具体处理。如果不同应用程序使用了相同的运行标志，也会出现进程互斥 实现中存在的问题。由于运行标志存在外部载体中，如果笔者把启动的应用程序进程实例直接在Windows管理器进程列表中结束或使其产生异常，那设置的运 行标志就不会销毁，应用程序就没法再次运行。   
  
**功能测试**  
这一节对上面的三个功能进行测试，以分析之间的区别。功能测试类别包括下面五类，   
1．本地系统同一应用程序目录；   
2．本地系统同一应用程序修改运行文件名称使两次运行名称不同；   
3．本地系统两次运行程序目录不同，不修改文件名称；   
4．本地系统不同会话用户登录启动应用程序；   
5．远程计算机程序访问启动应用程序（一个程序在远程另一个在本地）。  
  
根据代码实现细节不同，对测试的结果可能会有所不同，这里的测试结果以笔者上面几节中实现的代码为准。为了测试简单化，通过给应用程序传入测试参数，决定使用哪种方式，入口函数调用代码为，   
[STAThread]   
static void Main(string[] args)   
{   
        if (args.Length == 0) //没有传送参数   
        {   
                Process p = SingleInstance.GetRunningInstance();   
                if (p != null) //已经有应用程序副本执行   
                        SingleInstance.HandleRunningInstance(p);   
                else //启动第一个应用程序   
                        Application.Run(new MainForm());   
        }   
        else //有多个参数   
        {   
                switch (args[0].ToLower())   
                {   
                        case "-api":   
                                if (SingleInstance.HandleRunningInstance() == false)   
                                        Application.Run(new MainForm());   
                                break;   
                        case "-mutex":   
                                if (args.Length >= 2) //参数中传入互斥体名称   
                                {   
                                        if ( SingleInstance.CreateMutex(args[1]) )   
                                        {   
                                                Application.Run(new MainForm());   
                                                SingleInstance.ReleaseMutex();   
                                        }   
                                        else   
                                                //调用SingleInstance.HandleRunningInstance()方法显示到前台。   
                                                MessageBox.Show("程序已经运行！");   
                                }   
                                else   
                                {   
                                        if (SingleInstance.CreateMutex())   
                                        {   
                                                Application.Run(new MainForm());   
                                                SingleInstance.ReleaseMutex();   
                                        }   
                                        else   
                                                //调用SingleInstance.HandleRunningInstance()方法显示到前台。   
                                                MessageBox.Show("程序已经运行！");   
                                }   
                                break;   
                        case "-flag"://使用该方式需要在程序退出时调用   
                                if (args.Length >= 2) //参数中传入运行标志文件名称   
                                        SingleInstance.RunFlag = args[1];   
                                try   
                                {  
                                        if (SingleInstance.InitRunFlag())   
                                        {   
                                                Application.Run(new MainForm());   
                                                SingleInstance.DisposeRunFlag();   
                                        }   
                                        else   
                                                //调用SingleInstance.HandleRunningInstance()方法显示到前台。   
                                                MessageBox.Show("程序已经运行！");   
                                }   
                                catch (Exception ex)   
                                {   
                                        MessageBox.Show(ex.ToString());   
                                }   
                                break;   
                        default:   
                                MessageBox.Show("应用程序参数设置失败。");   
                                break;   
                }   
        }   
}   
  
运行CMD命令行，  
第一种调用方式：  
 WindowsApplication1.exe –api 或 WindowsApplication1.exe   
第二种调用方式：  
 WindowsApplication1.exe –mutex 或WindowsApplication1.exe –mutex {F140AE26-626C-42f8-BD49-45025742205E}   
第三种调用方式：  
 WindowsApplication1.exe –flag 或WindowsApplication1.exe –flag c:\blog.csdn.net.zhzuo   
  
测试结果：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 匹配/互斥/标志 | 1同一目录 | 2修改名称 | 3不同目录 | 4不同用户 | 5远程访问 |
| 1同一目录 | O/O/O |  |  |  |  |
| 2修改名称 |  | X/O/O |  |  |  |
| 3不同目录 |  |  | X/O/O |  |  |
| 4不同用户 |  |  |  | #/X/O |  |
| 5远程访问 |  |  |  |  | X/O/O |

备注：O - 表示成功，X – 表示失败，# - 程序第二个运行没有反应   
  
针对远程访问的测试，需要在系统管理工具的.NET Framework 2.0 Configuration中进行设置授权该局域网路径允许访问，否则会抛出System.Security.SecurityException异常。 根据测试结果可见三种实现方式适用范围不同，理想的实现是结合他们的优点进行多点判断。

**主要代码**  
SingleInstance.cs文件，

using System;  
using System.IO;  
using System.Diagnostics;  
using System.Threading;  
using System.Reflection;  
using System.Runtime.InteropServices;  
/\*------------------------------------------------  
  郑佐 2006-07-01  http://blog.csdn.net/zhzuo        
--------------------------------------------------\*/  
namespace Zhengzuo.CSharpCode  
{  
    /// <summary>  
    /// 只启动一个应用程序实例控制类  
    /// </summary>  
    public static class SingleInstance  
    {  
        private const int WS\_SHOWNORMAL = 1;  
        [DllImport("User32.dll")]  
        private static extern bool ShowWindowAsync(IntPtr hWnd, int cmdShow);   
        [DllImport("User32.dll")]  
        private static extern bool SetForegroundWindow(IntPtr hWnd);  
        //标志文件名称  
        private static string runFlagFullname = null;  
        //声明同步基元  
        private static Mutex mutex = null;   
   
        /// <summary>  
        /// static Constructor  
        /// </summary>  
        static SingleInstance()  
        {  
        }

        #region api实现        
        /// <summary>  
        /// 获取应用程序进程实例,如果没有匹配进程，返回Null  
        /// </summary>  
        /// <returns>返回当前Process实例</returns>  
        public static Process GetRunningInstance()  
        {

            Process currentProcess = Process.GetCurrentProcess();//获取当前进程  
            //获取当前运行程序完全限定名  
            string currentFileName = currentProcess.MainModule.FileName;  
            //获取进程名为ProcessName的Process数组。  
            Process[] processes = Process.GetProcessesByName(currentProcess.ProcessName);  
            //遍历有相同进程名称正在运行的进程  
            foreach (Process process in processes)  
            {  
                if (process.MainModule.FileName == currentFileName)  
                {  
                    if (process.Id != currentProcess.Id)//根据进程ID排除当前进程  
                        return process;//返回已运行的进程实例  
                }  
            }  
            return null;  
        }

        /// <summary>  
        /// 获取应用程序句柄，设置应用程序前台运行，并返回bool值  
        /// </summary>  
        public static bool HandleRunningInstance(Process instance)  
        {  
            //确保窗口没有被最小化或最大化  
            ShowWindowAsync(instance.MainWindowHandle, WS\_SHOWNORMAL);  
            //设置真实例程为foreground window  
            return SetForegroundWindow(instance.MainWindowHandle);  
        }

        /// <summary>  
        /// 获取窗口句柄，设置应用程序前台运行，并返回bool值，重载方法  
        /// </summary>  
        /// <returns></returns>  
        public static bool HandleRunningInstance()  
        {  
            Process p = GetRunningInstance();  
            if (p != null)  
            {  
                HandleRunningInstance(p);  
                return true;  
            }  
            return false;  
        }

        #endregion

        #region Mutex实现  
        /// <summary>  
        /// 创建应用程序进程Mutex  
        /// </summary>  
        /// <returns>返回创建结果，true表示创建成功，false创建失败。</returns>  
        public static bool CreateMutex()  
        {  
            return CreateMutex(Assembly.GetEntryAssembly().FullName);  
        }

        /// <summary>  
        /// 创建应用程序进程Mutex  
        /// </summary>  
        /// <param name="name">Mutex名称</param>  
        /// <returns>返回创建结果，true表示创建成功，false创建失败。</returns>  
        public static bool CreateMutex(string name)  
        {  
            bool result = false;  
            mutex = new Mutex(true, name, out result);  
            return result;    
        }

        /// <summary>  
        /// 释放Mutex  
        /// </summary>  
        public static void ReleaseMutex()  
        {  
            if (mutex != null)  
            {  
                mutex.Close();  
            }  
        }

        #endregion

        #region 设置标志实现  
        /// <summary>  
        /// 初始化程序运行标志，如果设置成功，返回true，已经设置返回false，设置失败将抛出异常  
        /// </summary>  
        /// <returns>返回设置结果</returns>  
        public static bool InitRunFlag()  
        {  
            if (File.Exists(RunFlag))  
            {  
                return false;  
            }  
            using (FileStream fs = new FileStream(RunFlag, FileMode.Create))  
            {   
            }  
            return true;  
        }

        /// <summary>  
        /// 释放初始化程序运行标志，如果释放失败将抛出异常  
        /// </summary>  
        public static void DisposeRunFlag()  
        {  
            if (File.Exists(RunFlag))  
            {  
                File.Delete(RunFlag);  
            }  
        }

        /// <summary>  
        /// 获取或设置程序运行标志，必须符合Windows文件命名规范  
        /// 这里实现生成临时文件为依据，如果修改成设置注册表，那就不需要符合文件命名规范。  
        /// </summary>  
        public static string RunFlag  
        {

            get  
            {  
                if(runFlagFullname == null)  
                {  
                    string assemblyFullName = Assembly.GetEntryAssembly().FullName;  
                    //CommonApplicationData：//"C:\\Documents and Settings\\All Users\\Application Data"  
                    string path = Environment.GetFolderPath(Environment.SpecialFolder.CommonApplicationData);  
                    //"C:\\Program Files\\Common Files"  
                    //string path = Environment.GetFolderPath(Environment.SpecialFolder.CommonProgramFiles);  
                    runFlagFullname = Path.Combine(path, assemblyFullName);  
                }  
                return runFlagFullname;  
            }  
            set  
            {  
                runFlagFullname = value;  
            }  
        }  
        #endregion  
    }

}

Program.cs文件，

using System;  
using System.Windows.Forms;  
using System.Diagnostics;  
using Zhengzuo.CSharpCode;  
/\*------------------------------------------------  
  郑佐 2006-07-01  http://blog.csdn.net/zhzuo  
--------------------------------------------------\*/  
namespace Zhengzuo.Test.WinGui  
{  
    static class Program  
    {  
        [STAThread]  
        static void Main(string[] args)  
        {  
            if (args.Length == 0) //没有传送参数  
            {  
                Process p = SingleInstance.GetRunningInstance();  
                if (p != null) //已经有应用程序副本执行  
                {  
                    SingleInstance.HandleRunningInstance(p);  
                }  
                else //启动第一个应用程序  
                {  
                    RunApplication();  
                }  
            }  
            else //有多个参数  
            {  
                switch (args[0].ToLower())  
                {  
                    case "-api":  
                        if (SingleInstance.HandleRunningInstance() == false)  
                        {  
                            RunApplication();  
                        }  
                        break;  
                    case "-mutex":  
                        if (args.Length >= 2) //参数中传入互斥体名称  
                        {  
                            if ( SingleInstance.CreateMutex(args[1]) )  
                            {  
                                RunApplication();  
                                SingleInstance.ReleaseMutex();  
                            }  
                            else  
                            {  
                                //调用SingleInstance.HandleRunningInstance()方法显示到前台。  
                                MessageBox.Show("程序已经运行！");  
                            }  
                        }  
                        else  
                        {  
                            if (SingleInstance.CreateMutex())  
                            {  
                                RunApplication();  
                                SingleInstance.ReleaseMutex();  
                            }  
                            else  
                            {  
                                //调用SingleInstance.HandleRunningInstance()方法显示到前台。  
                                MessageBox.Show("程序已经运行！");  
                            }  
                        }  
                        break;  
                    case "-flag"://使用该方式需要在程序退出时调用  
                        if (args.Length >= 2) //参数中传入运行标志文件名称  
                        {  
                            SingleInstance.RunFlag = args[1];                            
                        }  
                        try  
                        {  
                            if (SingleInstance.InitRunFlag())  
                            {  
                                RunApplication();  
                                SingleInstance.DisposeRunFlag();  
                            }  
                            else  
                            {  
                                //调用SingleInstance.HandleRunningInstance()方法显示到前台。  
                                MessageBox.Show("程序已经运行！");  
                            }  
                        }  
                        catch (Exception ex)  
                        {  
                            MessageBox.Show(ex.ToString());  
                        }  
                        break;  
                    default:  
                        MessageBox.Show("应用程序参数设置失败。");  
                        break;  
                }  
            }  
        }

        //启动应用程序  
        static void RunApplication()  
        {  
            Application.EnableVisualStyles();  
            Application.Run(new MainForm());  
        }  
    }  
}

**功能测试**功能测试类别包括下面五类，  
1．本地系统同一应用程序目录；  
2．本地系统同一应用程序修改运行文件名称使两次运行名称不同；  
3．本地系统两次运行程序目录不同，不修改文件名称；  
4．本地系统不同会话用户登录启动应用程序；  
5．远程计算机程序访问启动应用程序（一个程序在远程另一个在本地）。

运行CMD命令行，  
第一种调用方式：  
WindowsApplication1.exe  
或 WindowsApplication1.exe –api  
第二种调用方式：  
WindowsApplication1.exe –mutex  
或WindowsApplication1.exe –mutex {F140AE26-626C-42f8-BD49-45025742205E}  
第三种调用方式：  
WindowsApplication1.exe –flag  
或WindowsApplication1.exe –flag c:\blog.csdn.net.zhzuo

测试结果,

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 匹配/互斥/标志 | 1同一目录 | 2修改名称 | 3不同目录 | 4不同用户 | 5远程访问 |
| 1同一目录 | O/O/O |  |  |  |  |
| 2修改名称 |  | X/O/O |  |  |  |
| 3不同目录 |  |  | X/O/O |  |  |
| 4不同用户 |  |  |  | #/X/O |  |
| 5远程访问 |  |  |  |  | X/O/O |

备注：O - 表示成功，X – 表示失败，# - 程序第二个运行没有反应

针对远程访问的测试，需要在系统管理工具的.NET Framework 2.0 Configuration中进行设置授权该局域网路径允许访问，否则会抛出System.Security.SecurityException异常。  
根据测试结果可见三种实现方式适用范围不同，理想的实现是结合他们的优点进行多点判断。

**使用GUID 的方法：**

public static class SingleInstance

{

[DllImport("User32.dll")]

private static extern bool ShowWindowAsync(IntPtr hWnd, int cmdShow);

[DllImport("User32.dll")]

private static extern bool SetForegroundWindow(IntPtr hWnd);

public static bool SingleCheck()

{

bool single\_flag = true;

Process curProcess = Process.GetCurrentProcess();

int cur\_pid = curProcess.Id;

string cur\_guid = getGUID(curProcess.MainModule.FileName);

// if you change assembly name, it will treat as other assembly.

// if you want to check in all list, just Process.GetProcessesByName()

foreach(Process pel in Process.GetProcessesByName(curProcess.ProcessName) ) {

if (pel.Id != cur\_pid)

{

try

{

string pel\_guid = getGUID(pel.MainModule.FileName);

if( cur\_guid != string.Empty && pel\_guid != string.Empty && cur\_guid == pel\_guid)

single\_flag = false;

}

catch (Exception err)

{

//throw new Exception(err.Message);

continue;

}

}

}

return single\_flag;

}

public static void SingleRunning()

{

Process curProcess = Process.GetCurrentProcess();

int cur\_pid = curProcess.Id;

string cur\_guid = getGUID(curProcess.MainModule.FileName);

// if you change assembly name, it will treat as other assembly.

// if you want to check in all list, just Process.GetProcessesByName()

foreach ( Process pel in Process.GetProcessesByName(curProcess.ProcessName) )

{

if (pel.Id != cur\_pid)

{

try

{

string pel\_guid = getGUID(pel.MainModule.FileName);

if (cur\_guid != string.Empty && pel\_guid != string.Empty && cur\_guid == pel\_guid)

{

ShowWindowAsync(pel.MainWindowHandle, 1);

SetForegroundWindow(pel.MainWindowHandle);

Environment.Exit(Environment.ExitCode);

break;

}

}

catch (Exception err)

{

continue;

}

}

}

}

public static string getGUID()

{

/\* other identity

// processName is not reliable if you change the assembly name "abc.exe" to "abcd.exe"

string process\_name = curProcess.ProcessName;

Assembly assembly = Assembly.LoadFile(curProcess.MainModule.FileName);

string guid1 = assembly.ManifestModule.ModuleVersionId.ToString(); // 只要每次编译（即使什么也没改），该值都会变

\*/

string guid = string.Empty;

try

{

Attribute guid\_attr = Attribute.GetCustomAttribute(Assembly.GetExecutingAssembly(), typeof(GuidAttribute));

guid = ((GuidAttribute)guid\_attr).Value; // GUID， 重新编译过，或者版本号变了， 该值都不变

}

catch (Exception err)

{

throw new Exception(err.Message);

}

return guid;

}

public static string getGUID(string assembly\_fileName)

{

string guid = string.Empty;

try

{

Assembly assem = getAssembly(assembly\_fileName);

if (assem != null)

{

Attribute guid\_attr = Attribute.GetCustomAttribute(assem, typeof(GuidAttribute));

guid = ((GuidAttribute)guid\_attr).Value;

}

}

catch (Exception err)

{

throw new Exception(err.Message);

}

return guid;

}

public static Assembly getAssembly(string fileName)

{

try

{

return Assembly.LoadFrom(fileName);

}

catch

{

return null;

}

}

}