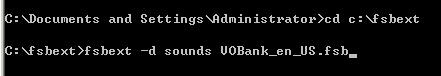
**24小时极限挑战WPF：LOLVoiceExtractor（WPF/C++DLL）实战**

**--Zephyroal**

##### 楔子：

游戏入迷太多终究不是件好事，技术同样有趣，可千万不能荒废，在每日闲余撸一把的时候，突然想过一个念头，游戏里人物的语音如此有趣，何必把他们提取出来做出手机铃声之类的逗逗去呢？

作为一个懒人，第一时间想到的是搜索工具，很快在[坛子](http://bbs.178.com/thread-1302716-1-1.html)找到一个好东西：**fsbext，**



用着还算不错，一部成功，但缺点也很大，命令行，看可怜的回帖数就知道对于普通玩家造成的困扰了，但很惊喜的发现了fsbext目录里还有若干.h和.C文件。。。

难不成是源代码？！

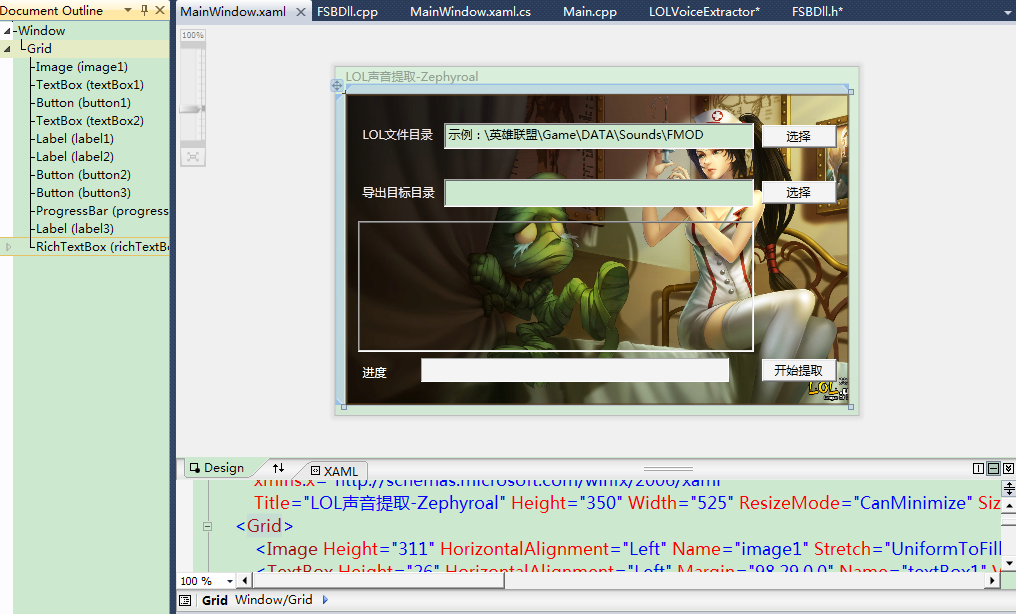
初看了下，fsb文件应该是FMOD的打包格式，如果代码可用，这样可以搞定非常多游戏的音效文件，何不做一个工具出来方便广大玩家玩乐？当然，特先申明，作此篇绝无任何破解破坏目的，毕竟，一个玩家愿意花时间精力去提取游戏中的元素，那可是建立在对这游戏的喜爱之上的，对不？

还好，这几日工作压力也正好不大，说干就干。

首先是工具用什么做，神马,MFC?no，I need some fresh air!既然是休闲目的,那就索性使用一些新鲜的技术，QT? C++/CLI? wxWidget? 额，还记得[为你解惑之WPF经典9问详解](http://www.cnblogs.com/hmxp8/archive/2011/10/12/2208357.html)，虽然对C#/ WPF从未有过接触，但其天生基于DX的特点博得了偶的好感,觉得就去过把摸石头过河尝鲜的瘾。

IDE也干脆搬上一直独守空房的VS2010，将工程命名为LOLVoiceExtractor，开工，特严正申明，由于是一切从零开始，记录下这一切只是作为一个学习的复习，也顺便记录一些要点，没准可以方便后面的菜鸟解惑，专业人士路过还望假装什么都没看见，免得贻笑大方，呵呵~

##### 1,设计WPF窗体



这部分很简单，拖一拖，拉一拉即可，跟MFC貌似也没啥差别，值得注意的是编辑器有时候点选Window范围之外的控件会有些麻烦，作为一个暴力流派：推荐直接手写XAML是最好的选择。

##### 2，控件响应

有了界面之后，就要开始添加逻辑了，如果没用VS08及后续版本IDE的可能会有些困惑，快速添加逻辑响应的面板到哪里去了？！稍安勿躁，打开属性面板，点击想要添加消息后边的空白处即可。

这里是一个目录打开的按钮，google一番即可，这里做了玩家是否是刚打开面板的判断，因为在设计窗体时添加了一个默认的参考目录，暗色显示，用于提示玩家。

private void button2\_Click(object sender, RoutedEventArgs e)

{

System.Windows.Forms.FolderBrowserDialog fbd = new System.Windows.Forms.FolderBrowserDialog();

fbd.ShowDialog();

if(m\_bFirstSeclectFolder)

{

m\_bFirstSeclectFolder=false;

Brush brush = new SolidColorBrush(Color.FromArgb(0xFF, 0x00, 0x00, 0x00));

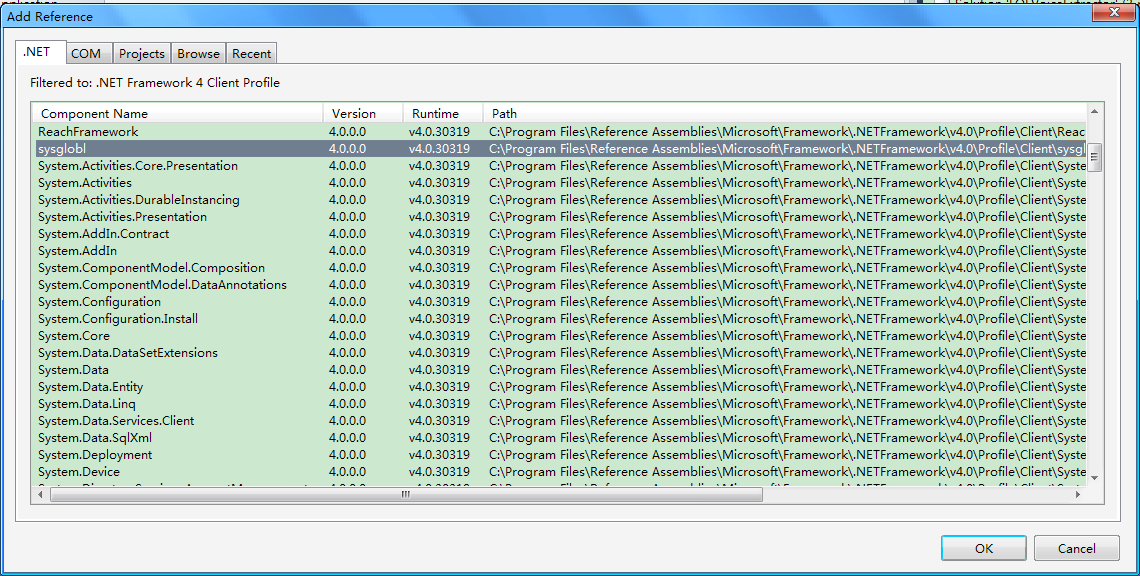
this.textBox1.Foreground = brush;

}

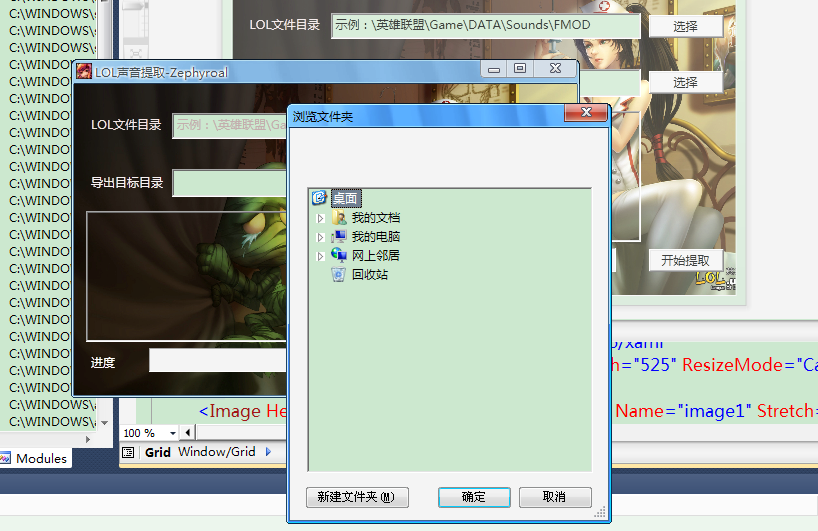
this.textBox1.Text = fbd.SelectedPath;

}

F5试试效果，但悲剧发现编译不过，查看原因，在项目解决方案中的Reference添加相应包即通过。



Ok，很可惜没写Concole的HelloWorld，就开始了C#的如下这货了，是不是很简单？



##### 3，C代码->C++

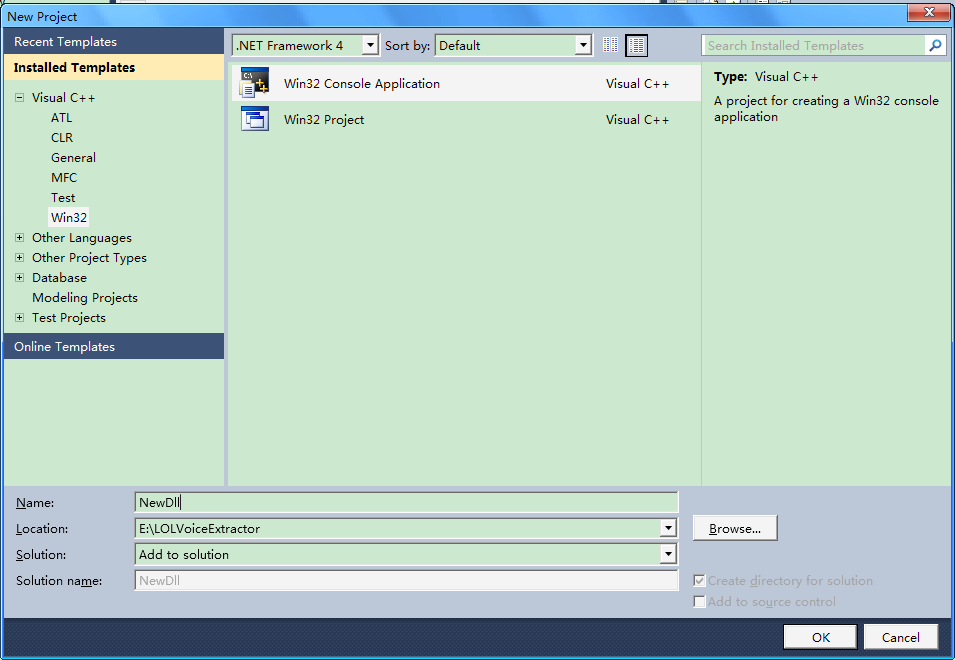
完成了其它几个控件逻辑之后，需要从UI获取的元素就都获得了：

源文件，输出目录，开始按钮，输出状态。

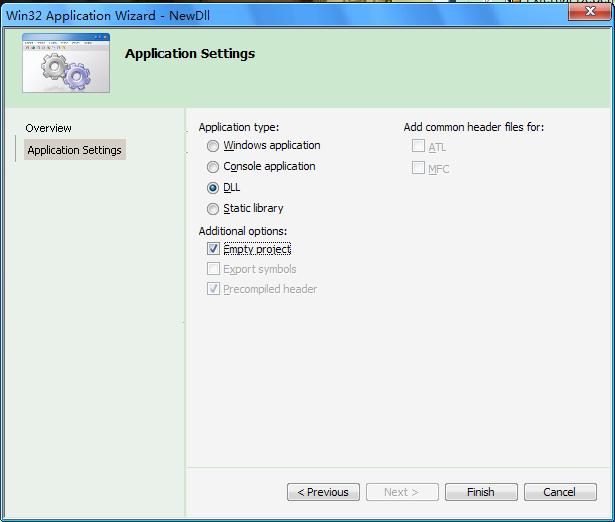
下面就可以开始核心的文件处理过程了，打算先跑起fsbext，熟悉了再转换成C#代码，首先建立了另一个C++项目（这引发了一个餐具，后面再提），很快发现，不妙。直接编译，几百个错。。。

估计写着东西的老外是用GNU之类的编译体系，似乎把bug越改越多不淡定了，即使成功改成C++之后，弄成C#估计又有得受了，而且在各处blog寻找WPF控件处理时也瞥到了对于托管代码效率的诟病，想想也是，WPF的初衷就是为了显示与逻辑的分离，那为何不干脆分离彻底？！直接让WPF处理界面逻辑，核心处理放在C++写出的DLL中谨供调用，虽然自己从未尝试，但既然都走到这里了，那就继续摸石头摸下去，毕竟自己听到很多人讲起过这套机制的优势，平时总借口忙，那择日不如撞日，现在就予一试！

首先建立一个DLL工程，

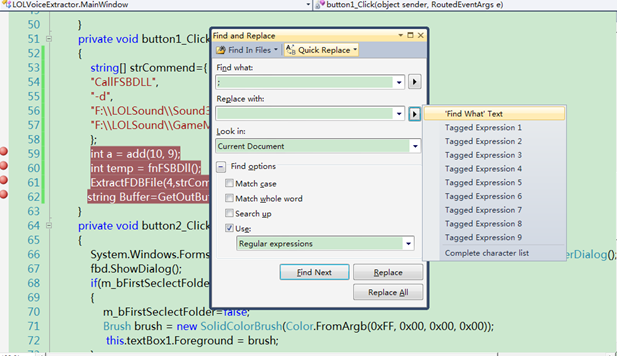


拷入fsbext的代码，一切从头开始：



##### 4, 代码修改，正则表达式

先静下来，认真看了下，大概错误有三类，分别是宏，typedef,及一些struct的使用，冷静，冷静，一个一个来，心中不断默念给自己加油，入定之后错误很快被一一搞定，这其中用了下正则表达式，现学现卖，对一些恶心的大批量处理真可谓神器，帮上了很大忙，顺便发现VS全部替换中也可以使用正则表达式，用\0来表达“你所找到的内容”，It’s awesome!



完成编译之后，再写一个DLLCaller，测试通过，

Ps：中间纠结忘记了console的传参中argv的第0个参数是程序本身名字，好吧，中了一坑。。。

##### 5， C#调用DLL

重新回到WPF的怀抱，现在轮到C#来调用DLL了，翻看了下资料，竟然到出库都不需要。

在DLL工程中，将函数声明如下：

#ifdef FSBDLL\_EXPORTS

#define FSBDLL\_API extern "C" \_\_declspec(dllexport)

#else

#define FSBDLL\_API extern "C" \_\_declspec(dllimport)

#endif

FSBDLL\_API int ExtractFDBFile(int argc, char \*argv[]);

FSBDLL\_API const char\* GetOutBuffer();

在C#中，按如下导入：

[DllImport("FSBDll.dll", EntryPoint = "ExtractFDBFile")]

public static extern int ExtractFDBFile(int argc, string[] argv);

##### 6，调用参数问题

由于一处想返回DLL中产生的字符串，网上搜了一堆文章，头头是道，可没一个讲通的，多谢一位老兄，[**C#托管代码与C++非托管代码互相调用一（C#调用C++代码&.net 代码安全）**](http://www.cnblogs.com/Jianchidaodi/archive/2009/03/09/1407270.html)

一个sample茅塞顿开，原来char什么的直接用一个string去接即可~~

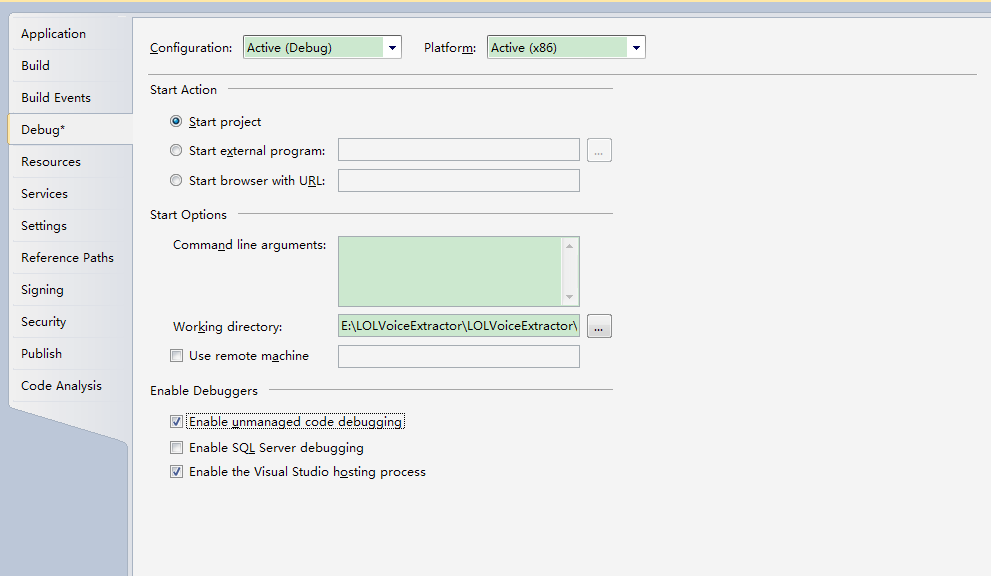
[DllImport("FSBDll.dll", EntryPoint = "GetOutBuffer")]

public static extern string GetOutBuffer();

一步到位调用成功，实在是灰常Easy, KISS，继续旅程。

##### 7，C#中调试C++代码

完成C#对DLL的调用后，直接将C++工程添加到了项目中，这里可以开启非托管代码调试，唯一不爽的是这样就不能再runtime时修改代码了（两者不都是支持的么？一起就不行了，诶。。。）



打开Enable unmanaged code debuggin

##### 8，OutBuffer处理

DLL返回字符串，传出当前文件的处理进度（文件非常多。。。），这里直接用上一个RichTextBox，并将其HorizontalScrollBarVisibility属性设置为"Auto"

并在添加新字符串后处理

if (newStr != "")

{

this.richTextBox1.AppendText(newStr);

this.richTextBox1.ScrollToEnd();

}

这样就可以搞定一个可以拖动的输出消息面板了（实际RichText可以设置数据源，但偷懒了，这里直接简单处理）。。。

要清空当前输出面板话

this.richTextBox1.Document.Blocks.Clear();

##### 9，Timer

要不断的接受DLL输出的消息，可以使用一个定时器来实现相应功能：

//构造一个DispatcherTimer类实例

m\_Timer= new System.Windows.Threading.DispatcherTimer();

//设置事件处理函数

m\_Timer.Tick += new EventHandler(WriteBuffer);

//定时器时间间隔

m\_Timer.Interval = new System.TimeSpan(0, 0, 0,0,10);

其中WirteBuffer即时处理函数，TimeSpan的格式分别为天，时，分，秒，毫秒。

##### 10，指针 壁纸 Icon

完成了主题功能之后，接下来便主要是美化工作了，

程序的壁纸可以通过Window的background属性设置，也可以通过一张Image实现，同样，为了实现最大化的膜拜精神，特地把鼠标指针也提取了出来



设置如下，这里使用了stream，且放后边再提。

T his.Cursor = new System.Windows.Input.Cursor(stream3);

程序的Icon图标及运行窗口的Icon则分别可以在Window的属性面板和项目工程的属性中设置，这里不再赘述~

##### 11,注册表，记录打开文件夹位置

平时工作习惯了，查完bug后会尽力从玩家角度考虑作品的易用性（大家无视笔者，这家伙又开始给自己抹了，哈哈哈~~），体验了几把软件，发现每次需要重选打开文件目录，目标目录十分麻烦，怎么办？可以写Ini文件，也可以写注册表，出于最易携带性考虑，直接写入注册表；

private void GetDefaultDir()

{

try

{

RegistryKey testKey = Registry.CurrentUser.OpenSubKey("LolSoundsExtractor");

if (testKey == null)

{

testKey = Registry.CurrentUser.CreateSubKey("LolSoundsExtractor");

testKey.SetValue("OpenFolderPath", "");

testKey.Close();

Registry.CurrentUser.Close();

}

else

{

m\_sDefaultFilePath = testKey.GetValue("OpenFolderPath").ToString();

testKey.Close();

Registry.CurrentUser.Close();

}

}

catch (Exception e)

{

}

}

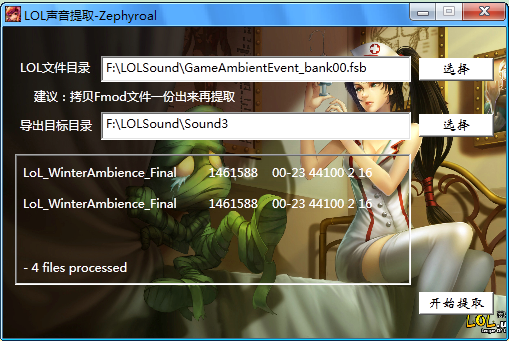
简单实例如上，实际中还可以做许多有趣的东西，比如玩家在第一回打开软件时系统会想起FirstBlood的声音，而以后就是琴女mm的悠悠弹琴声了~~

##### 12，返回string

DLL向C#中返回字符是个比较难缠的问题，调试过程中又出现过若干次bug，下面的帖子可以找到一种解决思路：

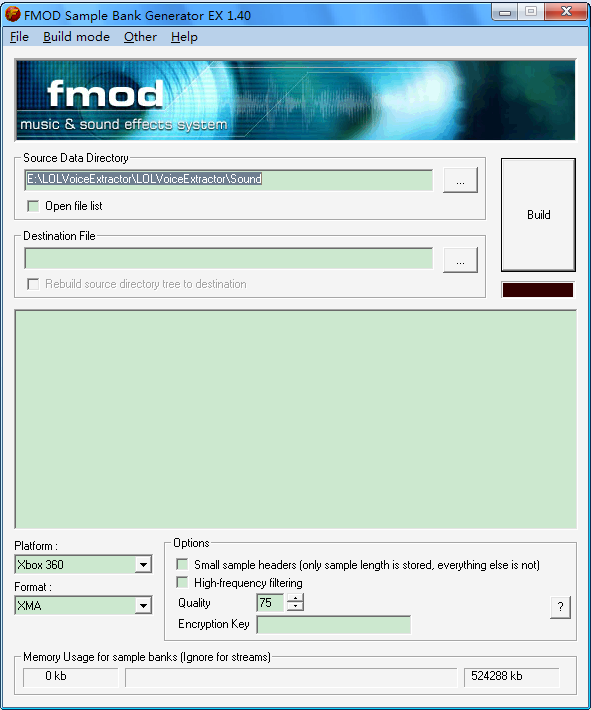
<http://topic.csdn.net/u/20091116/00/d3fc1021-d233-4100-a83a-21aea321d131.html>

但万恶，万恶的是，我自己都忘了是怎么做的，说到这里，不得不提前面埋的一个伏笔，那就是我发现C++和C#厮混得很融洽之后就干了件无比扯淡的事情，将原来的C++DLL工程整体ShiftDelete了，（先前将DLL工程复制入WPF工程中，但导入并一直修改的竟然tmd还是原来C++中的，活见鬼了。。。）恢复工作也做了许久，但一直未能救回代码，代码已经遗失，自己之前做的方法记不起来了，后边再提重写之后的方法。



先立一个毒誓，我，再也不用Shift+Del键了

##### 13，导出的文件格式

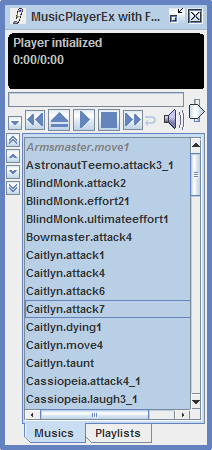
 

导出的wav文件据说比较可恶，但我用的是Duomi，一开始没发现，但开始尝试在程序中播放导出的wav文件的时候斯巴达了，原来导出的wav文件并非常规格式，似乎被FMOD处理过。

尝试了n款号称万能的转换工具，都苦苦难寻一果，就差想不开要去捣鼓wav格式了的时候，灵光一现，

还记得一开始的那个帖子么？

别小瞧那个Java做的播放器，人家才是万能播放器，竟然还带转换功能，爱死它了！



##### 14，开启线程，分离处理过程

这里由于需要做一个解压，并同时监控解压进度的功能，如同游戏里的资源加载，最好的方法就是建立一个线程，独立处理解压功能：

m\_Thread = new Thread(new ThreadStart(ExtractFile));

m\_Thread.Start();

当然了，比起游戏里，这算是非常简单的使用了~

##### 15，计算当前进度

关于读取进度，直接使用了两个值，需要处理的和已经处理的，从DLL中导出，并未做时间估算：

FSBDLL\_API int GetTotalNum()

{

return g\_iTotalNum;

}

FSBDLL\_API int GetNowNum()

{

return g\_iNowNum;

}

##### 16，SplashScreen欢迎界面

无意中发现了Page的选项还有个SplashScreen功能，手痒，直接P了一张，俺得让自己的程序看着有模有样的，对吧？

（虽然很囧的加载时间都几乎不需要）



除了BuildAction，当然还可以手动建立Spalash

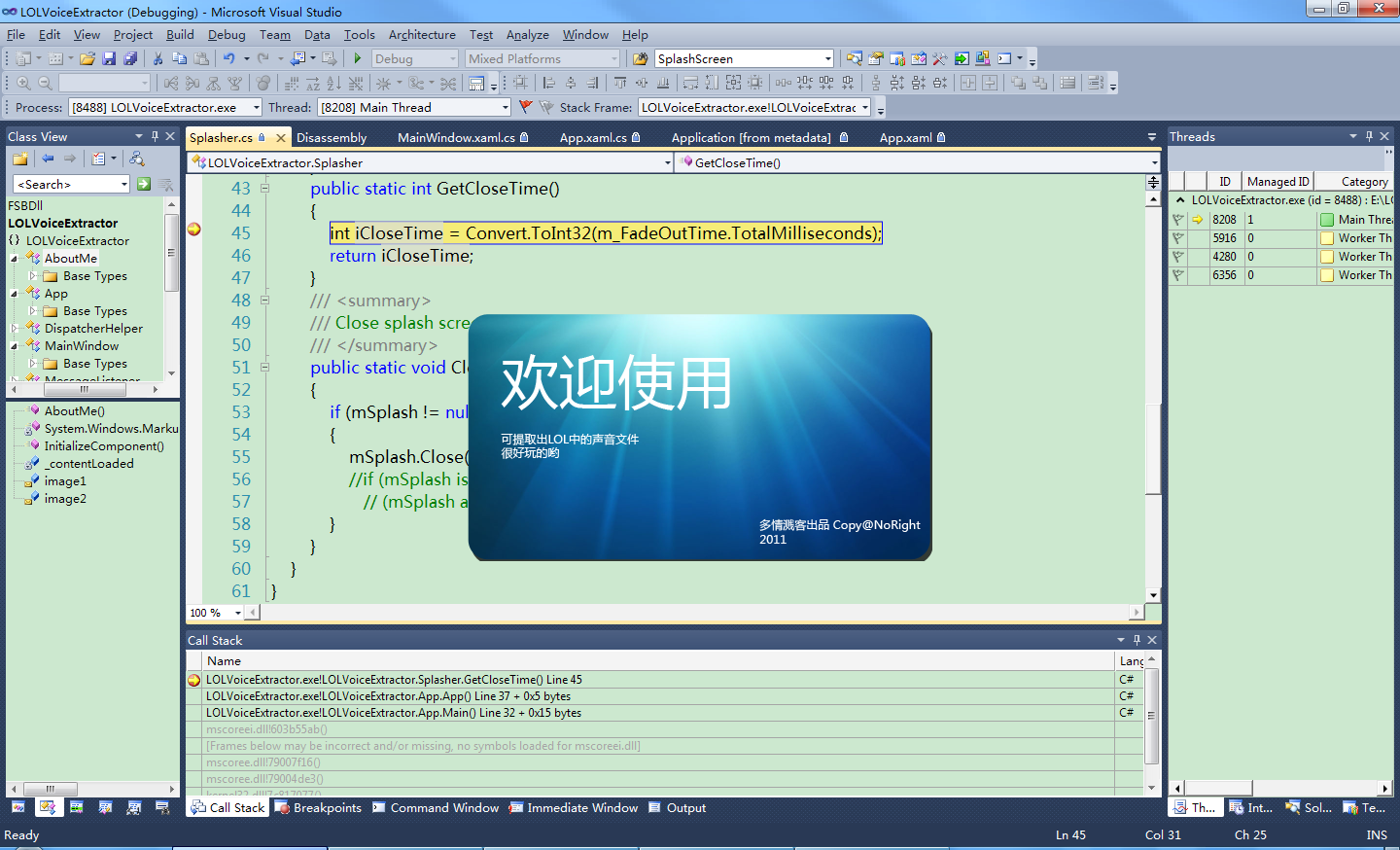
<http://www.codeproject.com/KB/WPF/WPFsplashscreen.aspx>

这里用到了手动定义Main函数，遇错

C:\DOCUME~1\hemu\LOCALS~1\Temp\07VINTJ8C6_8OU9F}$5L_R1.jpg

<http://www.cnblogs.com/yigedaizi/archive/2011/06/16/WPFapp_g_cs.html>

解决方法是取消BuildAction中的Application选项，这样VS就不会再为你自作主张地写Main函数了~



##### 17，Wave文件

关于音频播放，WPF本身的COM载入后只能实现WAV文件的播放，但鉴于敝人做大做强的习惯作风，我想要的是 Wav/Mp3/Ogg诸多格式的全能播放，正好前面在试图转换去加载FMOD发现了它有C#组件，并且有完整的samples，好，解铃还须系铃人，强大的FMOD正式进入清场。

所需要的FMOD组件主要是：

fmod.cs / fmod\_dsp.cs / fmod\_errors.cs

外加一个fmodex.dll。

导入之后，建立一个FMOD的实现类，完成初始化，加载，及Update即可，

这里提供一个简单的Demo：

private void IniSystem()

{

uint version = 0;

FMOD.RESULT result;

/\*

Create a System object and initialize.

\*/

result = FMOD.Factory.System\_Create(ref mSystem);

ERRCHECK(result);

result = mSystem.getVersion(ref version);

ERRCHECK(result);

if (version < FMOD.VERSION.number)

{

System.Windows.MessageBox.Show("Error! You are using an old version of FMOD " + version.ToString("X") + ". This program requires " + FMOD.VERSION.number.ToString("X") + ".");

return;

}

result = mSystem.init(32, FMOD.INITFLAG.NORMAL, (IntPtr)null);

ERRCHECK(result);

}

private void Load()

{

FMOD.RESULT result;

//LoadVoice

m\_sVoiceSource = new string[(int)eVoice.VoiceNum];

mVoice = new FMOD.Sound[(int)eVoice.VoiceNum];

m\_sVoiceSource[(int)eVoice.VoiceAnni] = "AnnieFun.mp3";

for (int i = 0; i < (int)eVoice.VoiceNum; ++i)

{

result = mSystem.createSound(mVoiceDir+m\_sVoiceSource[i], FMOD.MODE.HARDWARE, ref mVoice[i]);

ERRCHECK(result);

}

}

注意，还需要建立一个Timer不断去更新FMOD的系统，否则，多放几个文件你就懂了，嘿嘿：

private void timer\_Tick(object sender, System.EventArgs e)

{

FMOD.RESULT result;

if (channel != null)

{

FMOD.Sound currentsound = null;

result = channel.isPlaying(ref playing);

if ((result != FMOD.RESULT.OK) && (result != FMOD.RESULT.ERR\_INVALID\_HANDLE) && (result != FMOD.RESULT.ERR\_CHANNEL\_STOLEN))

{

ERRCHECK(result);

}

result = channel.getPaused(ref paused);

if ((result != FMOD.RESULT.OK) && (result != FMOD.RESULT.ERR\_INVALID\_HANDLE) && (result != FMOD.RESULT.ERR\_CHANNEL\_STOLEN))

{

ERRCHECK(result);

}

result = channel.getPosition(ref ms, FMOD.TIMEUNIT.MS);

if ((result != FMOD.RESULT.OK) && (result != FMOD.RESULT.ERR\_INVALID\_HANDLE) && (result != FMOD.RESULT.ERR\_CHANNEL\_STOLEN))

{

ERRCHECK(result);

}

channel.getCurrentSound(ref currentsound);

if (currentsound != null)

{

result = currentsound.getLength(ref lenms, FMOD.TIMEUNIT.MS);

if ((result != FMOD.RESULT.OK) && (result != FMOD.RESULT.ERR\_INVALID\_HANDLE) && (result != FMOD.RESULT.ERR\_CHANNEL\_STOLEN))

{

ERRCHECK(result);

}

}

}

##### ,18，枚举文件

除了程序的一些指定音效，我还加了一个好玩的彩蛋，鼠标滑过一个人物头像图标，他的眼睛会发光，并且随机响起游戏人物的声音，这里没有指定加载列表，而是直接枚举了指定目录下的所有MP3文件，建立列表，并在播放时随机提取~

C#下并不能调用熟悉的底层Win32API，这点很郁闷，又是打开了好基友Google，一番狂点，发现msdn上有个现成的例子，但用的是LinQ，这货真的很强，看着很爽但真用起来不爽，几个东西不好实现，最后还是选择了古老办法，查到了DirectoryInfo这个组件，然后直接枚举~

//LoadSounds

DirectoryInfo dir = new DirectoryInfo(mSoundsDir);

//不是目录?

if (dir == null)

return;

FileSystemInfo[] files = dir.GetFiles("\*.mp3", SearchOption.AllDirectories);

m\_iSoundCount = (int)files.Length;

mSound = new FMOD.Sound[m\_iSoundCount];

for (int i = 0; i < files.Length; i++)

{

FileInfo file = files[i] as FileInfo;

//是文件t

if (file != null)

{

result = mSystem.createSound(file.FullName,FMOD.MODE.HARDWARE, ref mSound[i]);

ERRCHECK(result);

}

}

##### 19，C#中的Enum

可能你注意到了，前满在加载Music使用了Enum，并且在主模块也使用了Enum来标识状态，据说C#的Enum是很强大的，但与C++它至少有以下两点不同：

1. Enum值并非能像C++中那样直接当成宏来使用，Enum的名字差不多算是一个独立的命名空间；
2. Enum值的转换需要手动进行，包括一些int型的转换，这点郁闷；

##### 20,EmbadedResources

图片资源多了，程序不免显得臃肿，最好的方法是将相关的图素直接压入程序Exe中，查看了一下，把资源打包成EmbadedResource即可；

CodeProject上有一份详细的说明， Demo如下： <http://www.codeproject.com/KB/dotnet/embeddedresources.aspx>

大家都是这么说的，ms很简单，可实际使用中可使头疼了好一阵子，怎么也不成，最后试验出用Stream读取是最好的方法：

System.Reflection.Assembly tManifestAssembly = System.Reflection.Assembly.GetExecutingAssembly();

System.IO.Stream stream0 = tManifestAssembly.GetManifestResourceStream(typeof(App),"Images.SplashScreen1.png");

System.IO.Stream stream1 = tManifestAssembly.GetManifestResourceStream("LOLVoiceExtractor.Images.SplashScreen1.png");

这两种方式皆可，

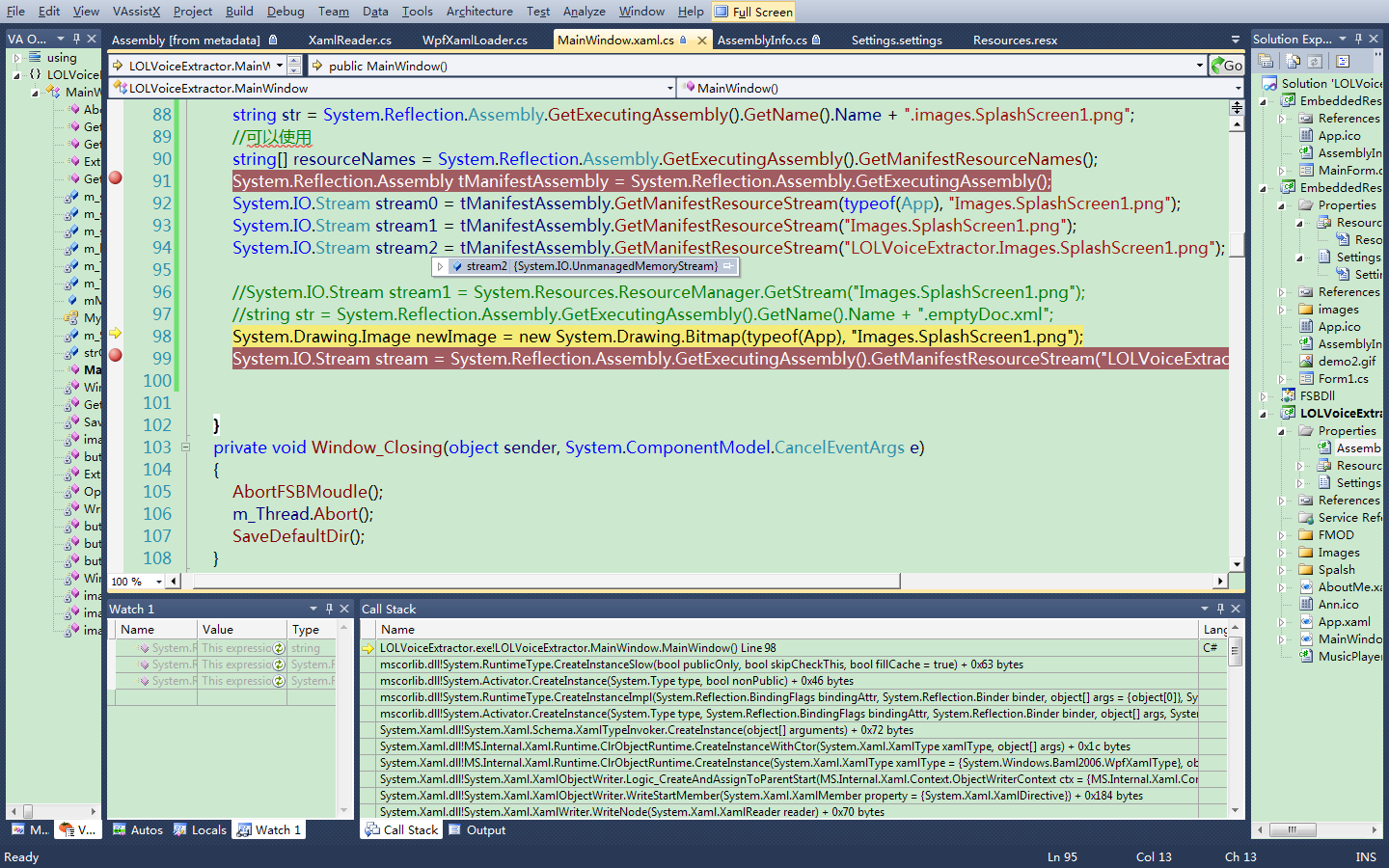
但读取出的资源显示是System.IO.UnmanagedMemoryStream，找了下资料：

<http://stackoverflow.com/questions/917589/why-am-i-getting-stream-as-system-io-unmanagedmemorystream>

原因如下：

Resources get compiled as part of the assembly (EXE or DLL), which means they gets loaded into unmanaged memory when the OS starts the process. This is the reason why any stream returned byGetManifestResourceStream must therefore be unmanaged (of type UnmanagedMemoryStream).

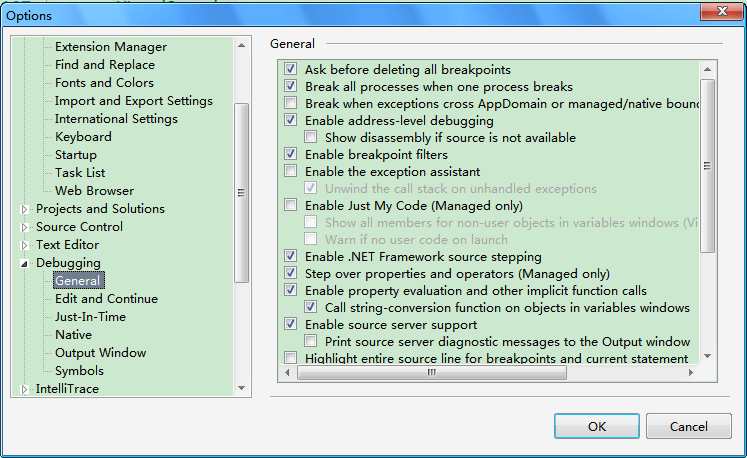
What's the problem with this, anyway? The interface of MemoryStream and UnmanagedMemoryStream are basically identical, and it's only the (hidden) functionality that differs, which shouldn't be of any consequence to you.



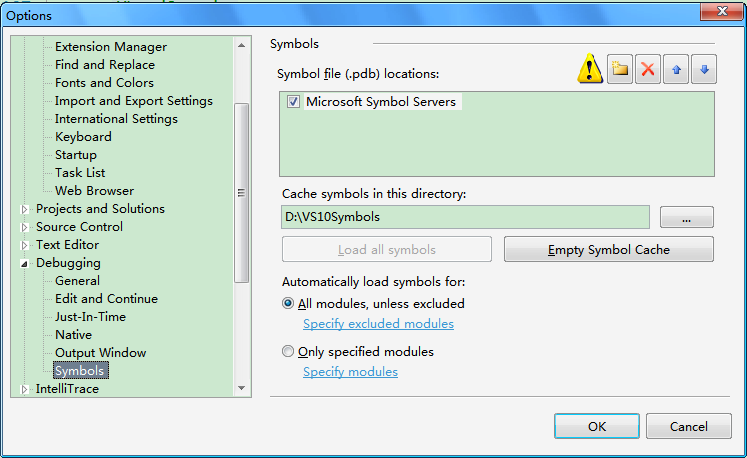
其它的都好办，唯有Spalash，代码也没，资源无法载入，最后只好自己写了一个，放弃了WPF自带的SpalashBuild，这样也好，再建立若干线程，Timer之后，虽然麻烦写，尤其是将C#中一个对象整成全局变量花了一番力气，但架构已经可以用于实际的大型程序读取资源、初始化的Spalash界面。Ok，继续~

##### 21，源代码调试

在测试的时候，常常程序挂了，只得到一个Exception，很是郁闷，其实，经查，MS还是挺大方的，因为.Net库已经大部分开源，并且在VS中原生支持，打开VS的Opiontion，只需简单两步：



1，在Debug面板中找到Enable .Net Framework source steppin ，勾选，并确认弹出窗口



,2，然后再到Symbol中将Microsoft Symbol Servers选上，定个目录，确认，稍等即可

完成之后，长长的CallStack就不会再那么恶心了，全是清清爽爽的代码了^\_^~

##### 22，StringBuilder

还记得前面提过关于DLL的string传参问题不？在试图加入暂停/继续功能后，彻底抓狂了，C++直接返回char并不可行，最后几经折腾，发现了StringBuilder 这个好东西，

在C++中：

FSBDLL\_API void SetBuffer(char \*ViewBuffer)

{

sprintf(ViewBuffer,"%s",g\_sOutBuffer.c\_str());

g\_sOutBuffer.clear();

g\_sOutBuffer="";

return;

}

在C#中：

[DllImport("FSBDll.dll", EntryPoint = "SetBuffer")]

public static extern void SetBuffer(StringBuilder abuf);

申明后建立一个StringBuilder，由于buffer会很大，之力不管三七二十一，直接给了个很大初始值，实际运用中可以优化~

mStringBuilder = new StringBuilder();

mStringBuilder.Capacity = 20480;//设置字符串最大长度

直接调用如下

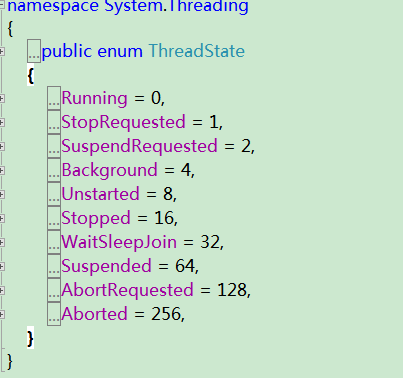
SetBuffer(mStringBuilder);

Ini = mStringBuilder.ToString(); ;

最后，如果中途断出，Runtime还是会报错，相关信息很少，做到这里，痛苦不言而喻，几近放弃，最后只在国外的一个论坛中找到了类似情况，有关砖家淡定地云云这好像是.Net4.0的原因，好吧，我就是4.0的，转成Net3.5结果问题真的解决了，囧~

##### 23,线程的管理

关于中途断出的问题，其中还是大有门道的，总结一下，Thread直接Abort并不能立即结束线程，先看下 Thread的几个状态：



在线程开始继续的时候必须先要明确判断线程所处的状态，否则极易出错！

m\_State = MyState.State\_Extracting;

if (m\_Thread.ThreadState == ThreadState.Unstarted)

{

m\_Thread.Start();

}

else if (m\_Thread.ThreadState == ThreadState.Suspended || m\_Thread.ThreadState == ThreadState.SuspendRequested)

{

m\_Thread.Resume();

}

else

{

m\_Thread = new Thread(new ThreadStart(ExtractFile));

m\_Thread.Start();

}

其中，new出线程的默认状态是Unstarted这时候直接Start既可，但如果是中途挂起 ，线程可能处在Suspended或者SuspendRequested状态，要恢复则应Resume，最后，如果上一个任务已经完成，则直接重建一个线程。

但是，中途如果Abort一个线程，则会出现莫名奇妙的冲突，即新线程会与这个被Abort的线程冲突，数据发生交叉，这里还有待解决~

##### 24,总结

一个念头，突发奇想，没想到最后还真坚持下来了，这点颇感安慰，程序的主题框架在第一天就完成，8个小时左右，上回学习Lua半天，正则表达式半天，虽然C#类似于Java

，但毕竟.Net是一大块领域，这个速度，还是不错的，记得某某说过一个优秀的程序员的指标是一周之内能掌握一门语言，当时还是受到了很大鼓舞的，回家时风大无比，但一路鸡冻，还特地给地铁站拉二胡的老头投了几个硬币~

到了后边，手头事情多起来了，每天只能抽些时间解决细节问题，花了差不多一周的时间解决细节问题，（估算了下，差不多花了24小时吧，所以就标题党一回，呵呵）。。。其中，像DLL交互，内嵌资源，自建Spalash，线程管理的问题还是结结实实摔了几个跟头的，托管太多，细节被影藏，对于习惯了控件渲染都由自己来的真的很不自在，但总的来说WPF还是相当易用上手的工具，其许多高级特性，SilverLight，ManagedDX调用，数据反射都还未体验，等待下篇吧~