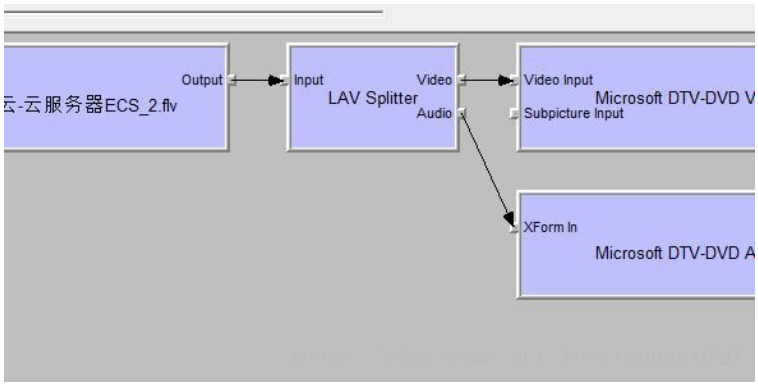
原 LAV Filter 源代码分析 2： LAV Splitter

2013年10月14日 21:06:30 阅读数：9422

LAV Filter 中最著名的就是 LAV Splitter，支持Matroska /WebM，MPEG-TS/PS，MP4/MOV，FLV，OGM / OGG，AVI等其他格式，广泛存在于各种视频播放器（暴风影音这类的）之中。

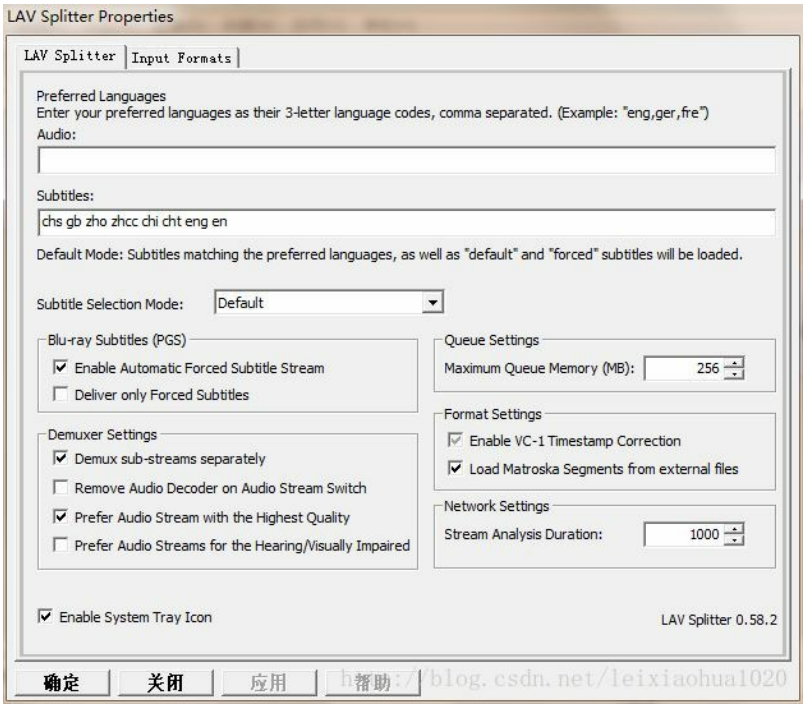
本文分析一下它的源代码。在分析之前，先看看它是什么样的。

使用GraphEdit随便打开一个视频文件，就可以看见LAV Filter：

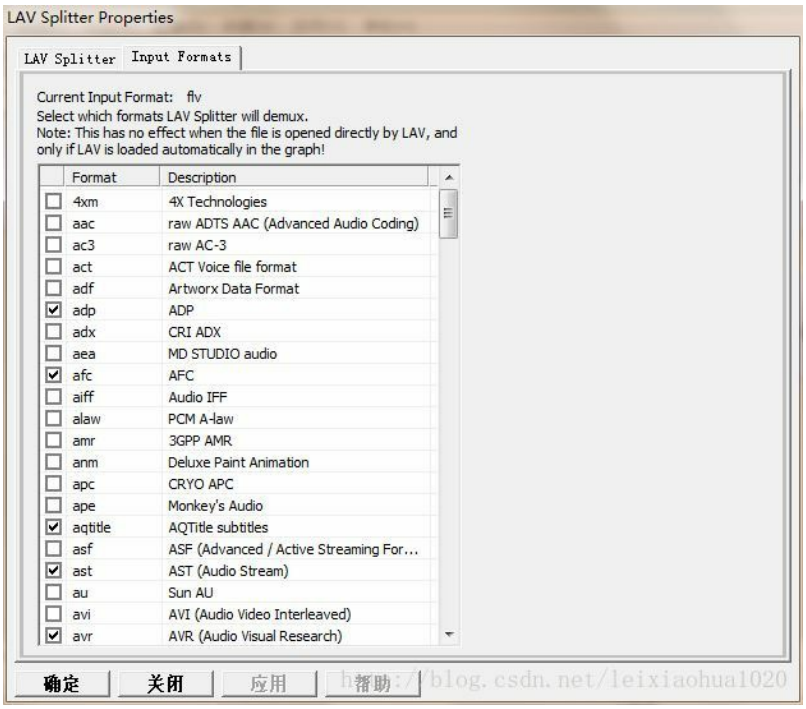


可以右键点击这个Filter看一下它的属性页面，如图所示：

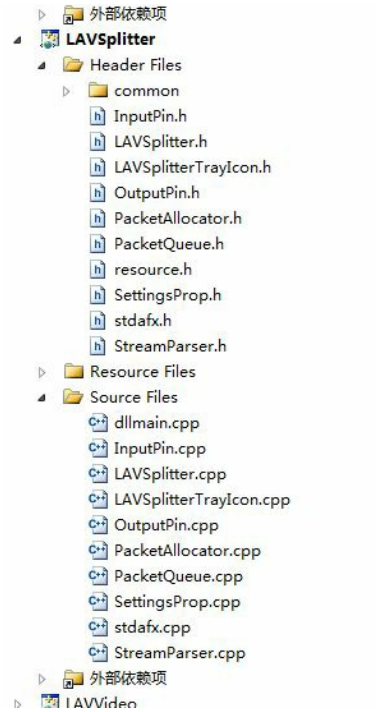
属性设置页面：



支持输入格式：



下面我们在 VC 2010 中看一下它的源代码：



从何看起呢？就先从directshow的注册函数看起吧，位于dllmain.cpp之中。部分代码的含义已经用注释标注上了。从代码可以看出，和普通的DirectShow Filter没什么区别。

dllmain.cpp

```cpp
/*
 *      Copyright (C) 2010-2013 Hendrik Leppkes
 *      http://www.1f0.de
 *
 *  This program is free software; you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation; either version 2 of the License, or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License along
 *  with this program; if not, write to the Free Software Foundation, Inc.,
 *  51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
```

```
 */

// Based on the SampleParser Template by GDCL
// ------------------------------------------------------------------------
// Copyright (c) GDCL 2004. All Rights Reserved.
// You are free to re-use this as the basis for your own filter development,
// provided you retain this copyright notice in the source.
// http://www.gdcl.co.uk
// ------------------------------------------------------------------------
//各种定义。。。
#include "stdafx.h"

// Initialize the GUIDs
#include <InitGuid.h>

#include <qnetwork.h>
#include "LAVSplitter.h"
#include "moreuuids.h"

#include "registry.h"
#include "IGraphRebuildDelegate.h"

// The GUID we use to register the splitter media types
DEFINE_GUID(MEDIATYPE_LAVSplitter,
  0x9c53931c, 0x7d5a, 0x4a75, 0xb2, 0x6f, 0x4e, 0x51, 0x65, 0x4d, 0xb2, 0xc0);

// --- COM factory table and registration code --------------
//注册时候的信息
const AMOVIESETUP_MEDIATYPE
  sudMediaTypes[] = {
    { &MEDIATYPE_Stream, &MEDIASUBTYPE_NULL },
};
//注册时候的信息（PIN）
const AMOVIESETUP_PIN sudOutputPins[] =
{
  {
    L"Output",            // pin name
      FALSE,              // is rendered?
      TRUE,               // is output?
      FALSE,              // zero instances allowed?
      TRUE,               // many instances allowed?
      &CLSID_NULL,        // connects to filter (for bridge pins)
      NULL,               // connects to pin (for bridge pins)
      0,                  // count of registered media types
      NULL                // list of registered media types
  },
  {
    L"Input",             // pin name
      FALSE,              // is rendered?
      FALSE,              // is output?
      FALSE,              // zero instances allowed?
      FALSE,              // many instances allowed?
      &CLSID_NULL,        // connects to filter (for bridge pins)
      NULL,               // connects to pin (for bridge pins)
      1,                  // count of registered media types
      &sudMediaTypes[0]   // list of registered media types
  }
};
//注册时候的信息（名称等）
//CLAVSplitter
const AMOVIESETUP_FILTER sudFilterReg =
{
  &__uuidof(CLAVSplitter),        // filter clsid
  L"LAV Splitter",                // filter name
  MERIT_PREFERRED + 4,            // merit
  2,                              // count of registered pins
  sudOutputPins,                  // list of pins to register
  CLSID_LegacyAmFilterCategory
};
//注册时候的信息（名称等）
//CLAVSplitterSource
const AMOVIESETUP_FILTER sudFilterRegSource =
{
  &__uuidof(CLAVSplitterSource),  // filter clsid
  L"LAV Splitter Source",         // filter name
  MERIT_PREFERRED + 4,            // merit
  1,                              // count of registered pins
  sudOutputPins,                  // list of pins to register
  CLSID_LegacyAmFilterCategory
};

// --- COM factory table and registration code --------------

// DirectShow base class COM factory requires this table,
// declaring all the COM objects in this DLL
// 注意g_Templates名称是固定的
CFactoryTemplate g_Templates[] = {
  // one entry for each CoCreate-able object
  {
    sudFilterReg.strName,
      sudFilterReg.clsID,
      CreateInstance<CLAVSplitter>,
```

```
      CreateInstance<CLAVSplitter>,
      CLAVSplitter::StaticInit,
      &sudFilterReg
  },
  {
    sudFilterRegSource.strName,
      sudFilterRegSource.clsID,
      CreateInstance<CLAVSplitterSource>,
      NULL,
      &sudFilterRegSource
  },
  // This entry is for the property page.
  // 属性页
  {
      L"LAV Splitter Properties",
      &CLSID_LAVSplitterSettingsProp,
      CreateInstance<CLAVSplitterSettingsProp>,
      NULL, NULL
  },
  {
      L"LAV Splitter Input Formats",
      &CLSID_LAVSplitterFormatsProp,
      CreateInstance<CLAVSplitterFormatsProp>,
      NULL, NULL
  }
};
int g_cTemplates = sizeof(g_Templates) / sizeof(g_Templates[0]);

// self-registration entrypoint
STDAPI DllRegisterServer()
{
  std::list<LPCWSTR> chkbytes;

  // BluRay
  chkbytes.clear();
  chkbytes.push_back(L"0,4,,494E4458"); // INDX (index.bdmv)
  chkbytes.push_back(L"0,4,,4D4F424A"); // MOBJ (MovieObject.bdmv)
  chkbytes.push_back(L"0,4,,4D504C53"); // MPLS
  RegisterSourceFilter(__uuidof(CLAVSplitterSource),
    MEDIASUBTYPE_LAVBluRay, chkbytes, NULL);

  // base classes will handle registration using the factory template table
  return AMovieDllRegisterServer2(true);
}

STDAPI DllUnregisterServer()
{
  UnRegisterSourceFilter(MEDIASUBTYPE_LAVBluRay);

  // base classes will handle de-registration using the factory template table
  return AMovieDllRegisterServer2(false);
}

// if we declare the correct C runtime entrypoint and then forward it to the DShow base
// classes we will be sure that both the C/C++ runtimes and the base classes are initialized
// correctly
extern "C" BOOL WINAPI DllEntryPoint(HINSTANCE, ULONG, LPVOID);
BOOL WINAPI DllMain(HANDLE hDllHandle, DWORD dwReason, LPVOID lpReserved)
{
  return DllEntryPoint(reinterpret_cast<HINSTANCE>(hDllHandle), dwReason, lpReserved);
}

void CALLBACK OpenConfiguration(HWND hwnd, HINSTANCE hinst, LPSTR lpszCmdLine, int nCmdShow)
{
  HRESULT hr = S_OK;
  CUnknown *pInstance = CreateInstance<CLAVSplitter>(NULL, &hr);
  IBaseFilter *pFilter = NULL;
  pInstance->NonDelegatingQueryInterface(IID_IBaseFilter, (void **)&pFilter);
  if (pFilter) {
    pFilter->AddRef();
    CBaseDSPropPage::ShowPropPageDialog(pFilter);
  }
  delete pInstance;
}
```

接下来就要进入正题了，看一看核心的分离器（解封装器）的类CLAVSplitter的定义文件LAVSplitter.h。乍一看这个类确实了得，居然继承了那么多的父类，实在是碉堡了。先不管那么多，看看里面都有什么函数吧。主要的函数上面都加了注释。注意还有一个类CLAVSplitterSource继承了CLAVSplitter。

LAVSplitter.h

```
[cpp]
/*
 *      Copyright (C) 2010-2013 Hendrik Leppkes
 *      http://www.1f0.de
 *
 *  This program is free software; you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation; either version 2 of the License, or
 *  (at your option) any later version.
```

```
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License along
 *  with this program; if not, write to the Free Software Foundation, Inc.,
 *  51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 *  Initial design and concept by Gabest and the MPC-HC Team, copyright under GPLv2
 *  Contributions by Ti-BEN from the XBMC DSPlayer Project, also under GPLv2
 */

#pragma once

#include <string>
#include <list>
#include <set>
#include <vector>
#include <map>
#include "PacketQueue.h"

#include "BaseDemuxer.h"

#include "LAVSplitterSettingsInternal.h"
#include "SettingsProp.h"
#include "IBufferInfo.h"

#include "ISpecifyPropertyPages2.h"

#include "LAVSplitterTrayIcon.h"

#define LAVF_REGISTRY_KEY L"Software\\LAV\\Splitter"
#define LAVF_REGISTRY_KEY_FORMATS LAVF_REGISTRY_KEY L"\\Formats"
#define LAVF_LOG_FILE     L"LAVSplitter.txt"

#define MAX_PTS_SHIFT 50000000i64

class CLAVOutputPin;
class CLAVInputPin;

#ifdef  _MSC_VER
#pragma warning(disable: 4355)
#endif
//核心解复用（分离器）
//暴漏的接口在ILAVFSettings中
[uuid("171252A0-8820-4AFE-9DF8-5C92B2D66B04")]
class CLAVSplitter
  : public CBaseFilter
  , public CCritSec
  , protected CAMThread
  , public IFileSourceFilter
  , public IMediaSeeking
  , public IAMStreamSelect
  , public IAMOpenProgress
  , public ILAVFSettingsInternal
  , public ISpecifyPropertyPages2
  , public IObjectWithSite
  , public IBufferInfo
{
public:
  CLAVSplitter(LPUNKNOWN pUnk, HRESULT* phr);
  virtual ~CLAVSplitter();

  static void CALLBACK StaticInit(BOOL bLoading, const CLSID *clsid);

  // IUnknown
  //
  DECLARE_IUNKNOWN;
  //暴露接口，使外部程序可以QueryInterface，关键！
  //翻译（“没有代表的方式查询接口”）
  STDMETHODIMP NonDelegatingQueryInterface(REFIID riid, void** ppv);

  // CBaseFilter methods
  //输入是一个，输出就不一定了！
  int GetPinCount();
  CBasePin *GetPin(int n);
  STDMETHODIMP GetClassID(CLSID* pClsID);

  STDMETHODIMP Stop();
  STDMETHODIMP Pause();
  STDMETHODIMP Run(REFERENCE_TIME tStart);

  STDMETHODIMP JoinFilterGraph(IFilterGraph * pGraph, LPCWSTR pName);

  // IFileSourceFilter
  // 源Filter的接口方法
  STDMETHODIMP Load(LPCOLESTR pszFileName, const AM_MEDIA_TYPE * pmt);
  STDMETHODIMP GetCurFile(LPOLESTR *ppszFileName, AM_MEDIA_TYPE *pmt);

```

```
  // IMediaSeeking
  STDMETHODIMP GetCapabilities(DWORD* pCapabilities);
  STDMETHODIMP CheckCapabilities(DWORD* pCapabilities);
  STDMETHODIMP IsFormatSupported(const GUID* pFormat);
  STDMETHODIMP QueryPreferredFormat(GUID* pFormat);
  STDMETHODIMP GetTimeFormat(GUID* pFormat);
  STDMETHODIMP IsUsingTimeFormat(const GUID* pFormat);
  STDMETHODIMP SetTimeFormat(const GUID* pFormat);
  STDMETHODIMP GetDuration(LONGLONG* pDuration);
  STDMETHODIMP GetStopPosition(LONGLONG* pStop);
  STDMETHODIMP GetCurrentPosition(LONGLONG* pCurrent);
  STDMETHODIMP ConvertTimeFormat(LONGLONG* pTarget, const GUID* pTargetFormat, LONGLONG Source, const GUID* pSourceFormat);
  STDMETHODIMP SetPositions(LONGLONG* pCurrent, DWORD dwCurrentFlags, LONGLONG* pStop, DWORD dwStopFlags);
  STDMETHODIMP GetPositions(LONGLONG* pCurrent, LONGLONG* pStop);
  STDMETHODIMP GetAvailable(LONGLONG* pEarliest, LONGLONG* pLatest);
  STDMETHODIMP SetRate(double dRate);
  STDMETHODIMP GetRate(double* pdRate);
  STDMETHODIMP GetPreroll(LONGLONG* pllPreroll);

  // IAMStreamSelect
  STDMETHODIMP Count(DWORD *pcStreams);
  STDMETHODIMP Enable(long lIndex, DWORD dwFlags);
  STDMETHODIMP Info(long lIndex, AM_MEDIA_TYPE **ppmt, DWORD *pdwFlags, LCID *plcid, DWORD *pdwGroup, WCHAR **ppszName, IUnknown **pp
Object, IUnknown **ppUnk);

  // IAMOpenProgress
  STDMETHODIMP QueryProgress(LONGLONG *pllTotal, LONGLONG *pllCurrent);
  STDMETHODIMP AbortOperation();

  // ISpecifyPropertyPages2
  STDMETHODIMP GetPages(CAUUID *pPages);
  STDMETHODIMP CreatePage(const GUID& guid, IPropertyPage** ppPage);

  // IObjectWithSite
  STDMETHODIMP SetSite(IUnknown *pUnkSite);
  STDMETHODIMP GetSite(REFIID riid, void **ppvSite);

  // IBufferInfo
  STDMETHODIMP_(int) GetCount();
  STDMETHODIMP GetStatus(int i, int& samples, int& size);
  STDMETHODIMP_(DWORD) GetPriority();

  // ILAVFSettings
  STDMETHODIMP SetRuntimeConfig(BOOL bRuntimeConfig);
  STDMETHODIMP GetPreferredLanguages(LPWSTR *ppLanguages);
  STDMETHODIMP SetPreferredLanguages(LPCWSTR pLanguages);
  STDMETHODIMP GetPreferredSubtitleLanguages(LPWSTR *ppLanguages);
  STDMETHODIMP SetPreferredSubtitleLanguages(LPCWSTR pLanguages);
  STDMETHODIMP_(LAVSubtitleMode) GetSubtitleMode();
  STDMETHODIMP SetSubtitleMode(LAVSubtitleMode mode);
  STDMETHODIMP_(BOOL) GetSubtitleMatchingLanguage();
  STDMETHODIMP SetSubtitleMatchingLanguage(BOOL dwMode);
  STDMETHODIMP_(BOOL) GetPGSForcedStream();
  STDMETHODIMP SetPGSForcedStream(BOOL bFlag);
  STDMETHODIMP_(BOOL) GetPGSOnlyForced();
  STDMETHODIMP SetPGSOnlyForced(BOOL bForced);
  STDMETHODIMP_(int) GetVC1TimestampMode();
  STDMETHODIMP SetVC1TimestampMode(int iMode);
  STDMETHODIMP SetSubstreamsEnabled(BOOL bSubStreams);
  STDMETHODIMP_(BOOL) GetSubstreamsEnabled();
  STDMETHODIMP SetVideoParsingEnabled(BOOL bEnabled);
  STDMETHODIMP_(BOOL) GetVideoParsingEnabled();
  STDMETHODIMP SetFixBrokenHDPVR(BOOL bEnabled);
  STDMETHODIMP_(BOOL) GetFixBrokenHDPVR();
  STDMETHODIMP_(HRESULT) SetFormatEnabled(LPCSTR strFormat, BOOL bEnabled);
  STDMETHODIMP_(BOOL) IsFormatEnabled(LPCSTR strFormat);
  STDMETHODIMP SetStreamSwitchRemoveAudio(BOOL bEnabled);
  STDMETHODIMP_(BOOL) GetStreamSwitchRemoveAudio();
  STDMETHODIMP GetAdvancedSubtitleConfig(LPWSTR *ppAdvancedConfig);
  STDMETHODIMP SetAdvancedSubtitleConfig(LPCWSTR pAdvancedConfig);
  STDMETHODIMP SetUseAudioForHearingVisuallyImpaired(BOOL bEnabled);
  STDMETHODIMP_(BOOL) GetUseAudioForHearingVisuallyImpaired();
  STDMETHODIMP SetMaxQueueMemSize(DWORD dwMaxSize);
  STDMETHODIMP_(DWORD) GetMaxQueueMemSize();
  STDMETHODIMP SetTrayIcon(BOOL bEnabled);
  STDMETHODIMP_(BOOL) GetTrayIcon();
  STDMETHODIMP SetPreferHighQualityAudioStreams(BOOL bEnabled);
  STDMETHODIMP_(BOOL) GetPreferHighQualityAudioStreams();
  STDMETHODIMP SetLoadMatroskaExternalSegments(BOOL bEnabled);
  STDMETHODIMP_(BOOL) GetLoadMatroskaExternalSegments();
  STDMETHODIMP GetFormats(LPSTR** formats, UINT* nFormats);
  STDMETHODIMP SetNetworkStreamAnalysisDuration(DWORD dwDuration);
  STDMETHODIMP_(DWORD) GetNetworkStreamAnalysisDuration();

  // ILAVSplitterSettingsInternal
  STDMETHODIMP_(LPCSTR) GetInputFormat() { if (m_pDemuxer) return m_pDemuxer->GetContainerFormat(); return NULL; }
  STDMETHODIMP_(std::set<FormatInfo>&) GetInputFormats();
  STDMETHODIMP_(BOOL) IsVC1CorrectionRequired();
  STDMETHODIMP_(CMediaType *) GetOutputMediatype(int stream);
  STDMETHODIMP_(IFilterGraph *) GetFilterGraph() { if (m_pGraph) { m_pGraph->AddRef(); return m_pGraph; } return NULL; }

```

```
  STDMETHODIMP_(DWORD) GetStreamFlags(DWORD dwStream) { if (m_pDemuxer) return m_pDemuxer->GetStreamFlags(dwStream); return 0; }
  STDMETHODIMP_(int) GetPixelFormat(DWORD dwStream) { if (m_pDemuxer) return m_pDemuxer->GetPixelFormat(dwStream); return AV_PIX_FMT_NONE; }
  STDMETHODIMP_(int) GetHasBFrames(DWORD dwStream){ if (m_pDemuxer) return m_pDemuxer->GetHasBFrames(dwStream); return -1; }

  // Settings helper
  std::list<std::string> GetPreferredAudioLanguageList();
  std::list<CSubtitleSelector> GetSubtitleSelectors();

  bool IsAnyPinDrying();
  void SetFakeASFReader(BOOL bFlag) { m_bFakeASFReader = bFlag; }
protected:
  // CAMThread
  enum {CMD_EXIT, CMD_SEEK};
  DWORD ThreadProc();

  HRESULT DemuxSeek(REFERENCE_TIME rtStart);
  HRESULT DemuxNextPacket();
  HRESULT DeliverPacket(Packet *pPacket);

  void DeliverBeginFlush();
  void DeliverEndFlush();

  STDMETHODIMP Close();
  STDMETHODIMP DeleteOutputs();
  //初始化解复用器
  STDMETHODIMP InitDemuxer();

  friend class CLAVOutputPin;
  STDMETHODIMP SetPositionsInternal(void *caller, LONGLONG* pCurrent, DWORD dwCurrentFlags, LONGLONG* pStop, DWORD dwStopFlags);

public:
  CLAVOutputPin *GetOutputPin(DWORD streamId, BOOL bActiveOnly = FALSE);
  STDMETHODIMP RenameOutputPin(DWORD TrackNumSrc, DWORD TrackNumDst, std::vector<CMediaType> pmts);
  STDMETHODIMP UpdateForcedSubtitleMediaType();

  STDMETHODIMP CompleteInputConnection();
  STDMETHODIMP BreakInputConnection();

protected:
    //相关的参数设置
  STDMETHODIMP LoadDefaults();
  STDMETHODIMP ReadSettings(HKEY rootKey);
  STDMETHODIMP LoadSettings();
  STDMETHODIMP SaveSettings();
  //创建图标
  STDMETHODIMP CreateTrayIcon();

protected:
  CLAVInputPin *m_pInput;

private:
  CCritSec m_csPins;
  //用vector存储输出PIN（解复用的时候是不确定的）
  std::vector<CLAVOutputPin *> m_pPins;
  //活动的
  std::vector<CLAVOutputPin *> m_pActivePins;
  //不用的
  std::vector<CLAVOutputPin *> m_pRetiredPins;
  std::set<DWORD> m_bDiscontinuitySent;

  std::wstring m_fileName;
  std::wstring m_processName;
  //有很多纯虚函数的基本解复用类
  //注意：绝大部分信息都是从这获得的
  //这里的信息是由其派生类从FFMPEG中获取到的
  CBaseDemuxer *m_pDemuxer;

  BOOL m_bPlaybackStarted;
  BOOL m_bFakeASFReader;

  // Times
  REFERENCE_TIME m_rtStart, m_rtStop, m_rtCurrent, m_rtNewStart, m_rtNewStop;
  REFERENCE_TIME m_rtOffset;
  double m_dRate;
  BOOL m_bStopValid;

  // Seeking
  REFERENCE_TIME m_rtLastStart, m_rtLastStop;
  std::set<void *> m_LastSeekers;

  // flushing
  bool m_fFlushing;
  CAMEvent m_eEndFlush;

  std::set<FormatInfo> m_InputFormats;

  // Settings
  //设置
  struct Settings {
    BOOL TrayIcon;
    std::wstring prefAudioLangs;
```

```
    std::wstring prefAudioLangs;
    std::wstring prefSubLangs;
    std::wstring subtitleAdvanced;
    LAVSubtitleMode subtitleMode;
    BOOL PGSForcedStream;
    BOOL PGSOnlyForced;
    int vc1Mode;
    BOOL substreams;

    BOOL MatroskaExternalSegments;

    BOOL StreamSwitchRemoveAudio;
    BOOL ImpairedAudio;
    BOOL PreferHighQualityAudio;
    DWORD QueueMaxSize;
    DWORD NetworkAnalysisDuration;

    std::map<std::string, BOOL> formats;
  } m_settings;

  BOOL m_bRuntimeConfig;

  IUnknown *m_pSite;

  CBaseTrayIcon *m_pTrayIcon;
};

[uuid("B98D13E7-55DB-4385-A33D-09FD1BA26338")]
class CLAVSplitterSource : public CLAVSplitter
{
public:
  // construct only via class factory
  CLAVSplitterSource(LPUNKNOWN pUnk, HRESULT* phr);
  virtual ~CLAVSplitterSource();

  // IUnknown
  DECLARE_IUNKNOWN;
  //暴露接口，使外部程序可以QueryInterface，关键！
  //翻译（“没有代表的方式查询接口”）
  STDMETHODIMP NonDelegatingQueryInterface(REFIID riid, void** ppv);
};
```

先来看一下查询接口的函数NonDelegatingQueryInterface()吧

```cpp
//暴露接口，使外部程序可以QueryInterface，关键！
STDMETHODIMP CLAVSplitter::NonDelegatingQueryInterface(REFIID riid, void** ppv)
{
  CheckPointer(ppv, E_POINTER);

  *ppv = NULL;

  if (m_pDemuxer && (riid == __uuidof(IKeyFrameInfo) || riid == __uuidof(ITrackInfo) || riid == IID_IAMExtendedSeeking || riid == IID_IAMMediaContent)) {
    return m_pDemuxer->QueryInterface(riid, ppv);
  }
  //写法好特别啊，意思是一样的
  return
    QI(IMediaSeeking)
    QI(IAMStreamSelect)
    QI(ISpecifyPropertyPages)
    QI(ISpecifyPropertyPages2)
    QI2(ILAVFSettings)
    QI2(ILAVFSettingsInternal)
    QI(IObjectWithSite)
    QI(IBufferInfo)
    __super::NonDelegatingQueryInterface(riid, ppv);
}
```

这个NonDelegatingQueryInterface()的写法确实够特别的，不过其作用还是一样的：根据不同的REFIID，获得不同的接口指针。在这里就不多说了。

再看一下Load()函数

```cpp
// IFileSourceFilter
// 打开
STDMETHODIMP CLAVSplitter::Load(LPCOLESTR pszFileName, const AM_MEDIA_TYPE * pmt)
{
  CheckPointer(pszFileName, E_POINTER);

  m_bPlaybackStarted = FALSE;

  m_fileName = std::wstring(pszFileName);

  HRESULT hr = S_OK;
  SAFE_DELETE(m_pDemuxer);
  LPWSTR extension = PathFindExtensionW(pszFileName);

  DbgLog((LOG_TRACE, 10, L"::Load(): Opening file '%s' (extension: %s)", pszFileName, extension));

  // BDMV uses the BD demuxer, everything else LAVF
  if (_wcsicmp(extension, L".bdmv") == 0 || _wcsicmp(extension, L".mpls") == 0) {
    m_pDemuxer = new CBDDemuxer(this, this);
  } else {
    m_pDemuxer = new CLAVFDemuxer(this, this);
  }
  //打开
  if(FAILED(hr = m_pDemuxer->Open(pszFileName))) {
    SAFE_DELETE(m_pDemuxer);
    return hr;
  }
  m_pDemuxer->AddRef();

  return InitDemuxer();
}
```

在这里我们要注意CLAVSplitter的一个变量：m_pDemuxer。这是一个指向 CBaseDemuxer的指针。因此在这里CLAVSplitter实际上调用了 CBaseDemuxer中的方法。

从代码中的逻辑我们可以看出：

1.寻找文件后缀

2.当文件后缀是：".bdmv"或者".mpls"的时候，m_pDemuxer指向一个CBDDemuxer（我推测这代表目标文件是蓝光文件什么的），其他情况下m_pDemuxer指向一个CLAVFDemuxer。

3.然后m_pDemuxer会调用Open()方法。

4.最后会调用一个InitDemuxer()方法。

在这里我们应该看看m_pDemuxer->Open()这个方法里面有什么。我们先考虑m_pDemuxer指向CLAVFDemuxer的情况。

```cpp
// Demuxer Functions
// 打开（就是一个封装）
STDMETHODIMP CLAVFDemuxer::Open(LPCOLESTR pszFileName)
{
  return OpenInputStream(NULL, pszFileName, NULL, TRUE);
}
```

发现是一层封装，于是果断决定层层深入。

```cpp
//实际的打开,使用FFMPEG
STDMETHODIMP CLAVFDemuxer::OpenInputStream(AVIOContext *byteContext, LPCOLESTR pszFileName, const char *format, BOOL bForce)
{
  CAutoLock lock(m_pLock);
  HRESULT hr = S_OK;

  int ret; // return code from avformat functions

  // Convert the filename from wchar to char for avformat
  char fileName[4100] = {0};
  if (pszFileName) {
    ret = WideCharToMultiByte(CP_UTF8, 0, pszFileName, -1, fileName, 4096, NULL, NULL);
  }

  if (_strnicmp("mms:", fileName, 4) == 0) {
    memmove(fileName+1, fileName, strlen(fileName));
    memcpy(fileName, "mmsh", 4);
  }

  AVIOInterruptCB cb = {avio_interrupt_cb, this};

trynoformat:
  // Create the avformat_context
  // FFMPEG中的函数
  m_avFormat = avformat_alloc_context();
  m_avFormat->pb = byteContext;
  m_avFormat->interrupt_callback = cb;

  if (m_avFormat->pb)
    m_avFormat->flags |= AVFMT_FLAG_CUSTOM_IO;

  LPWSTR extension = pszFileName ? PathFindExtensionW(pszFileName) : NULL;

  AVInputFormat *inputFormat = NULL;
  //如果指定了格式
  if (format) {
      //查查有木有
    inputFormat = av_find_input_format(format);
  } else if (pszFileName) {
    LPWSTR extension = PathFindExtensionW(pszFileName);
    for (int i = 0; i < countof(wszImageExtensions); i++) {
      if (_wcsicmp(extension, wszImageExtensions[i]) == 0) {
        if (byteContext) {
          inputFormat = av_find_input_format("image2pipe");
        } else {
          inputFormat = av_find_input_format("image2");
        }
        break;
      }
    }
    for (int i = 0; i < countof(wszBlockedExtensions); i++) {
      if (_wcsicmp(extension, wszBlockedExtensions[i]) == 0) {
        goto done;
      }
    }
  }

  // Disable loading of external mkv segments, if required
  if (!m_pSettings->GetLoadMatroskaExternalSegments())
    m_avFormat->flags |= AVFMT_FLAG_NOEXTERNAL;

  m_timeOpening = time(NULL);
  //实际的打开
  ret = avformat_open_input(&m_avFormat, fileName, inputFormat, NULL);
  //出错了
  if (ret < 0) {
    DbgLog((LOG_ERROR, 0, TEXT("::OpenInputStream(): avformat_open_input failed (%d)"), ret));
    if (format) {
      DbgLog((LOG_ERROR, 0, TEXT(" -> trying again without specific format")));
      format = NULL;
      //实际的关闭
      avformat_close_input(&m_avFormat);
      goto trynoformat;
    }
    goto done;
  }
  DbgLog((LOG_TRACE, 10, TEXT("::OpenInputStream(): avformat_open_input opened file of type '%S' (took %I64d seconds)"), m_avFormat->iformat->name, time(NULL) - m_timeOpening));
  m_timeOpening = 0;
  //初始化AVFormat
  CHECK_HR(hr = InitAVFormat(pszFileName, bForce));

  return S_OK;
done:
  CleanupAVFormat();
  return E_FAIL;
}
```

看到这个函数，立马感受到了一种“拨云见日”的感觉。看到了很多FFMPEG的API函数。最重要的依据当属avformat_open_input()了，通过这个函数，打开了实际的文件。如果出现错误，则调用avformat_close_input()进行清理。

最后，还调用了InitAVFormat()函数：

```cpp
//初始化AVFormat
STDMETHODIMP CLAVFDemuxer::InitAVFormat(LPCOLESTR pszFileName, BOOL bForce)
{
  HRESULT hr = S_OK;
  const char *format = NULL;
  //获取InputFormat信息（，短名称，长名称）
  lavf_get_iformat_infos(m_avFormat->iformat, &format, NULL);
  if (!bForce && (!format || !m_pSettings->IsFormatEnabled(format))) {
    DbgLog((LOG_TRACE, 20, L"::InitAVFormat() - format of type '%S' disabled, failing", format ? format : m_avFormat->iformat->name)
);
    return E_FAIL;
  }

  m_pszInputFormat = format ? format : m_avFormat->iformat->name;

  m_bVC1SeenTimestamp = FALSE;

  LPWSTR extension = pszFileName ? PathFindExtensionW(pszFileName) : NULL;

  m_bMatroska = (_strnicmp(m_pszInputFormat, "matroska", 8) == 0);
  m_bOgg = (_strnicmp(m_pszInputFormat, "ogg", 3) == 0);
  m_bAVI = (_strnicmp(m_pszInputFormat, "avi", 3) == 0);
  m_bMPEGTS = (_strnicmp(m_pszInputFormat, "mpegts", 6) == 0);
  m_bMPEGPS = (_stricmp(m_pszInputFormat, "mpeg") == 0);
  m_bRM = (_stricmp(m_pszInputFormat, "rm") == 0);
  m_bPMP = (_stricmp(m_pszInputFormat, "pmp") == 0);
  m_bMP4 = (_stricmp(m_pszInputFormat, "mp4") == 0);

  m_bTSDiscont = m_avFormat->iformat->flags & AVFMT_TS_DISCONT;

  WCHAR szProt[24] = L"file";
  if (pszFileName) {
    DWORD dwNumChars = 24;
    hr = UrlGetPart(pszFileName, szProt, &dwNumChars, URL_PART_SCHEME, 0);
    if (SUCCEEDED(hr) && dwNumChars && (_wcsicmp(szProt, L"file") != 0)) {
      m_avFormat->flags |= AVFMT_FLAG_NETWORK;
      DbgLog((LOG_TRACE, 10, TEXT("::InitAVFormat(): detected network protocol: %s"), szProt));
    }
  }

  // TODO: make both durations below configurable
  // decrease analyze duration for network streams
  if (m_avFormat->flags & AVFMT_FLAG_NETWORK || (m_avFormat->flags & AVFMT_FLAG_CUSTOM_IO && !m_avFormat->pb->seekable)) {
    // require at least 0.2 seconds
    m_avFormat->max_analyze_duration = max(m_pSettings->GetNetworkStreamAnalysisDuration() * 1000, 200000);
  } else {
    // And increase it for mpeg-ts/ps files
    if (m_bMPEGTS || m_bMPEGPS)
      m_avFormat->max_analyze_duration = 10000000;
  }

  av_opt_set_int(m_avFormat, "correct_ts_overflow", !m_pBluRay, 0);

  if (m_bMatroska)
    m_avFormat->flags |= AVFMT_FLAG_KEEP_SIDE_DATA;

  m_timeOpening = time(NULL);
  //获取媒体流信息
  int ret = avformat_find_stream_info(m_avFormat, NULL);
  if (ret < 0) {
    DbgLog((LOG_ERROR, 0, TEXT("::InitAVFormat(): av_find_stream_info failed (%d)"), ret));
    goto done;
  }
  DbgLog((LOG_TRACE, 10, TEXT("::InitAVFormat(): avformat_find_stream_info finished, took %I64d seconds"), time(NULL) - m_timeOpening
));
  m_timeOpening = 0;

  // Check if this is a m2ts in a BD structure, and if it is, read some extra stream properties out of the CLPI files
  if (m_pBluRay) {
    m_pBluRay->ProcessClipLanguages();
  } else if (pszFileName && m_bMPEGTS) {
    CheckBDM2TSCPLI(pszFileName);
  }

  SAFE_CO_FREE(m_stOrigParser);
  m_stOrigParser = (enum AVStreamParseType *)CoTaskMemAlloc(m_avFormat->nb_streams * sizeof(enum AVStreamParseType));
  if (!m_stOrigParser)
    return E_OUTOFMEMORY;

  for(unsigned int idx = 0; idx < m_avFormat->nb_streams; ++idx) {
    AVStream *st = m_avFormat->streams[idx];

    // Disable full stream parsing for these formats
    if (st->need_parsing == AVSTREAM_PARSE_FULL) {
```

```
      if (st->codec->codec_id == AV_CODEC_ID_DVB_SUBTITLE) {
        st->need_parsing = AVSTREAM_PARSE_NONE;
      }
    }

    if (m_bOgg && st->codec->codec_id == AV_CODEC_ID_H264) {
      st->need_parsing = AVSTREAM_PARSE_FULL;
    }

    // Create the parsers with the appropriate flags
    init_parser(m_avFormat, st);
    UpdateParserFlags(st);

#ifdef DEBUG
    AVProgram *streamProg = av_find_program_from_stream(m_avFormat, NULL, idx);
    DbgLog((LOG_TRACE, 30, L"Stream %d (pid %d) - program: %d, codec: %S; parsing: %S;", idx, st->id, streamProg ? streamProg->pmt_pid : -1, avcodec_get_name(st->codec->codec_id), lavf_get_parsing_string(st->need_parsing)));
#endif
    m_stOrigParser[idx] = st->need_parsing;

    if ((st->codec->codec_id == AV_CODEC_ID_DTS && st->codec->codec_tag == 0xA2)
     || (st->codec->codec_id == AV_CODEC_ID_EAC3 && st->codec->codec_tag == 0xA1))
      st->disposition |= LAVF_DISPOSITION_SECONDARY_AUDIO;

    UpdateSubStreams();

    if (st->codec->codec_type == AVMEDIA_TYPE_ATTACHMENT && (st->codec->codec_id == AV_CODEC_ID_TTF || st->codec->codec_id == AV_CODEC_ID_OTF)) {
      if (!m_pFontInstaller) {
        m_pFontInstaller = new CFontInstaller();
      }
      m_pFontInstaller->InstallFont(st->codec->extradata, st->codec->extradata_size);
    }
  }

  CHECK_HR(hr = CreateStreams());

  return S_OK;
done:
  //关闭输入
  CleanupAVFormat();
  return E_FAIL;
}
```

该函数通过avformat_find_stream_info()等获取到流信息，这里就不多说了。

文章标签： LAVFilter LAVSplitter 源代码 分析

个人分类： LAV Filter

所属专栏： 开源多媒体项目源代码分析