原 LAV Filter 源代码分析 3： LAV Video （1）

2013年10月27日 20:58:00 阅读数：6519

LAV Video 是使用很广泛的DirectShow Filter。它封装了FFMPEG中的libavcodec，支持十分广泛的视频格式的解码。在这里对其源代码进行详细的分析。

LAV Video 工程代码的结构如下图所示

直接看LAV Video最主要的类CLAVVideo吧，它的定义位于LAVVideo.h中。

LAVVideo.h

```cpp
/* 雷霄骅
 * 中国传媒大学/数字电视技术
 * leixiaohua1020@126.com
 *
 */
/*
 *      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.
 */

#pragma once

#include "decoders/ILAVDecoder.h"
#include "DecodeThread.h"
#include "ILAVPinInfo.h"

#include "LAVPixFmtConverter.h"
#include "LAVVideoSettings.h"
#include "H264RandomAccess.h"
#include "FloatingAverage.h"

#include "ISpecifyPropertyPages2.h"
#include "SynchronizedQueue.h"

#include "subtitles/LAVSubtitleConsumer.h"
#include "subtitles/LAVVideoSubtitleInputPin.h"

#include "BaseTrayIcon.h"

#define LAVC_VIDEO_REGISTRY_KEY L"Software\\LAV\\Video"
#define LAVC_VIDEO_REGISTRY_KEY_FORMATS L"Software\\LAV\\Video\\Formats"
#define LAVC_VIDEO_REGISTRY_KEY_OUTPUT L"Software\\LAV\\Video\\Output"
#define LAVC_VIDEO_REGISTRY_KEY_HWACCEL L"Software\\LAV\\Video\\HWAccel"

#define LAVC_VIDEO_LOG_FILE     L"LAVVideo.txt"

#define DEBUG_FRAME_TIMINGS 0
#define DEBUG_PIXELCONV_TIMINGS 0

#define LAV_MT_FILTER_QUEUE_SIZE 4

typedef struct {
  REFERENCE_TIME rtStart;
  REFERENCE_TIME rtStop;
} TimingCache;
//解码核心类
//Transform Filter
[uuid("EE30215D-164F-4A92-A4EB-9D4C13390F9F")]
class CLAVVideo : public CTransformFilter, public ISpecifyPropertyPages2, public ILAVVideoSettings, public ILAVVideoStatus, public ILAVVideoCallback
{
public:
  CLAVVideo(LPUNKNOWN pUnk, HRESULT* phr);
  ~CLAVVideo();

```

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

    // IUnknown
    // 查找接口必须实现
    DECLARE_IUNKNOWN;
    STDMETHODIMP NonDelegatingQueryInterface(REFIID riid, void** ppv);

    // ISpecifyPropertyPages2
    // 属性页
    // 获取或者创建
    STDMETHODIMP GetPages(CAUUID *pPages);
    STDMETHODIMP CreatePage(const GUID& guid, IPropertyPage** ppPage);

    // ILAVVideoSettings

    STDMETHODIMP SetRuntimeConfig(BOOL bRuntimeConfig);
    STDMETHODIMP SetFormatConfiguration(LAVVideoCodec vCodec, BOOL bEnabled);
    STDMETHODIMP_(BOOL) GetFormatConfiguration(LAVVideoCodec vCodec);
    STDMETHODIMP SetNumThreads(DWORD dwNum);
    STDMETHODIMP_(DWORD) GetNumThreads();
    STDMETHODIMP SetStreamAR(DWORD bStreamAR);
    STDMETHODIMP_(DWORD) GetStreamAR();
    STDMETHODIMP SetPixelFormat(LAVOutPixFmts pixFmt, BOOL bEnabled);
    STDMETHODIMP_(BOOL) GetPixelFormat(LAVOutPixFmts pixFmt);
    STDMETHODIMP SetRGBOutputRange(DWORD dwRange);
    STDMETHODIMP_(DWORD) GetRGBOutputRange();

    STDMETHODIMP SetDeintFieldOrder(LAVDeintFieldOrder fieldOrder);
    STDMETHODIMP_(LAVDeintFieldOrder) GetDeintFieldOrder();
    STDMETHODIMP SetDeintForce(BOOL bForce);
    STDMETHODIMP_(BOOL) GetDeintForce();
    STDMETHODIMP SetDeintAggressive(BOOL bAggressive);
    STDMETHODIMP_(BOOL) GetDeintAggressive();

    STDMETHODIMP_(DWORD) CheckHWAccelSupport(LAVHWAccel hwAccel);
    STDMETHODIMP SetHWAccel(LAVHWAccel hwAccel);
    STDMETHODIMP_(LAVHWAccel) GetHWAccel();
    STDMETHODIMP SetHWAccelCodec(LAVVideoHWCodec hwAccelCodec, BOOL bEnabled);
    STDMETHODIMP_(BOOL) GetHWAccelCodec(LAVVideoHWCodec hwAccelCodec);
    STDMETHODIMP SetHWAccelDeintMode(LAVHWDeintModes deintMode);
    STDMETHODIMP_(LAVHWDeintModes) GetHWAccelDeintMode();
    STDMETHODIMP SetHWAccelDeintOutput(LAVDeintOutput deintOutput);
    STDMETHODIMP_(LAVDeintOutput) GetHWAccelDeintOutput();
    STDMETHODIMP SetHWAccelDeintHQ(BOOL bHQ);
    STDMETHODIMP_(BOOL) GetHWAccelDeintHQ();
    STDMETHODIMP SetSWDeintMode(LAVSWDeintModes deintMode);
    STDMETHODIMP_(LAVSWDeintModes) GetSWDeintMode();
    STDMETHODIMP SetSWDeintOutput(LAVDeintOutput deintOutput);
    STDMETHODIMP_(LAVDeintOutput) GetSWDeintOutput();

    STDMETHODIMP SetDeintTreatAsProgressive(BOOL bEnabled);
    STDMETHODIMP_(BOOL) GetDeintTreatAsProgressive();

    STDMETHODIMP SetDitherMode(LAVDitherMode ditherMode);
    STDMETHODIMP_(LAVDitherMode) GetDitherMode();

    STDMETHODIMP SetUseMSWMV9Decoder(BOOL bEnabled);
    STDMETHODIMP_(BOOL) GetUseMSWMV9Decoder();

    STDMETHODIMP SetDVDVideoSupport(BOOL bEnabled);
    STDMETHODIMP_(BOOL) GetDVDVideoSupport();

    STDMETHODIMP SetHWAccelResolutionFlags(DWORD dwResFlags);
    STDMETHODIMP_(DWORD) GetHWAccelResolutionFlags();

    STDMETHODIMP SetTrayIcon(BOOL bEnabled);
    STDMETHODIMP_(BOOL) GetTrayIcon();

    STDMETHODIMP SetDeinterlacingMode(LAVDeintMode deintMode);
    STDMETHODIMP_(LAVDeintMode) GetDeinterlacingMode();

    // ILAVVideoStatus
    STDMETHODIMP_(const WCHAR *) GetActiveDecoderName() { return m_Decoder.GetDecoderName(); }

    // CTransformFilter
    // 核心的
    HRESULT CheckInputType(const CMediaType* mtIn);
    HRESULT CheckTransform(const CMediaType* mtIn, const CMediaType* mtOut);
    HRESULT DecideBufferSize(IMemAllocator * pAllocator, ALLOCATOR_PROPERTIES *pprop);
    HRESULT GetMediaType(int iPosition, CMediaType *pMediaType);

    HRESULT SetMediaType(PIN_DIRECTION dir, const CMediaType *pmt);
    HRESULT EndOfStream();
    HRESULT BeginFlush();
    HRESULT EndFlush();
    HRESULT NewSegment(REFERENCE_TIME tStart, REFERENCE_TIME tStop, double dRate);
    //处理的核心
    //核心一般才有IMediaSample
    HRESULT Receive(IMediaSample *pIn);

    HRESULT CheckConnect(PIN_DIRECTION dir, IPin *pPin);
```

```
  HRESULT BreakConnect(PIN_DIRECTION dir);
  HRESULT CompleteConnect(PIN_DIRECTION dir, IPin *pReceivePin);

  int GetPinCount();
  CBasePin* GetPin(int n);

  STDMETHODIMP JoinFilterGraph(IFilterGraph * pGraph, LPCWSTR pName);

  // ILAVVideoCallback
  STDMETHODIMP AllocateFrame(LAVFrame **ppFrame);
  STDMETHODIMP ReleaseFrame(LAVFrame **ppFrame);
  STDMETHODIMP Deliver(LAVFrame *pFrame);
  STDMETHODIMP_(LPWSTR) GetFileExtension();
  STDMETHODIMP_(BOOL) FilterInGraph(PIN_DIRECTION dir, const GUID &clsid) { if (dir == PINDIR_INPUT) return FilterInGraphSafe(m_pInput, clsid); else return FilterInGraphSafe(m_pOutput, clsid); }
  STDMETHODIMP_(DWORD) GetDecodeFlags() { return m_dwDecodeFlags; }
  STDMETHODIMP_(CMediaType&) GetInputMediaType() { return m_pInput->CurrentMediaType(); }
  STDMETHODIMP GetLAVPinInfo(LAVPinInfo &info) { if (m_LAVPinInfoValid) { info = m_LAVPinInfo; return S_OK; } return E_FAIL; }
  STDMETHODIMP_(CBasePin*) GetOutputPin() { return m_pOutput; }
  STDMETHODIMP_(CMediaType&) GetOutputMediaType() { return m_pOutput->CurrentMediaType(); }
  STDMETHODIMP DVDStripPacket(BYTE*& p, long& len) { static_cast<CDeCSSTransformInputPin*>(m_pInput)->StripPacket(p, len); return S_OK; }
  STDMETHODIMP_(LAVFrame*) GetFlushFrame();
  STDMETHODIMP ReleaseAllDXVAResources() { ReleaseLastSequenceFrame(); return S_OK; }

public:
  // Pin Configuration
  const static AMOVIESETUP_MEDIATYPE    sudPinTypesIn[];
  const static int                      sudPinTypesInCount;
  const static AMOVIESETUP_MEDIATYPE    sudPinTypesOut[];
  const static int                      sudPinTypesOutCount;

private:
  HRESULT LoadDefaults();
  HRESULT ReadSettings(HKEY rootKey);
  HRESULT LoadSettings();
  HRESULT SaveSettings();

  HRESULT CreateTrayIcon();

  HRESULT CreateDecoder(const CMediaType *pmt);

  HRESULT GetDeliveryBuffer(IMediaSample** ppOut, int width, int height, AVRational ar, DXVA2_ExtendedFormat dxvaExtFormat, REFERENCE_TIME avgFrameDuration);
  HRESULT ReconnectOutput(int width, int height, AVRational ar, DXVA2_ExtendedFormat dxvaExtFlags, REFERENCE_TIME avgFrameDuration, BOOL bDXVA = FALSE);

  HRESULT SetFrameFlags(IMediaSample* pMS, LAVFrame *pFrame);

  HRESULT NegotiatePixelFormat(CMediaType &mt, int width, int height);
  BOOL IsInterlaced();


  HRESULT Filter(LAVFrame *pFrame);
  HRESULT DeliverToRenderer(LAVFrame *pFrame);

  HRESULT PerformFlush();
  HRESULT ReleaseLastSequenceFrame();

  HRESULT GetD3DBuffer(LAVFrame *pFrame);
  HRESULT RedrawStillImage();
  HRESULT SetInDVDMenu(bool menu) { m_bInDVDMenu = menu; return S_OK; }

  enum {CNTRL_EXIT, CNTRL_REDRAW};
  HRESULT ControlCmd(DWORD cmd) {
    return m_ControlThread->CallWorker(cmd);
  }

private:
  friend class CVideoOutputPin;
  friend class CDecodeThread;
  friend class CLAVControlThread;
  friend class CLAVSubtitleProvider;
  friend class CLAVSubtitleConsumer;
  //解码线程
  CDecodeThread        m_Decoder;
  CAMThread            *m_ControlThread;

  REFERENCE_TIME       m_rtPrevStart;
  REFERENCE_TIME       m_rtPrevStop;

  BOOL                 m_bForceInputAR;
  BOOL                 m_bSendMediaType;
  BOOL                 m_bFlushing;

  HRESULT              m_hrDeliver;

  CLAVPixFmtConverter  m_PixFmtConverter;
  std::wstring         m_strExtension;

  DWORD                m_bDXVAExtFormatSupport;
```

```
  DWORD                 m_bMadVR;
  DWORD                 m_bOverlayMixer;
  DWORD                 m_dwDecodeFlags;

  BOOL                  m_bInDVDMenu;

  AVFilterGraph         *m_pFilterGraph;
  AVFilterContext       *m_pFilterBufferSrc;
  AVFilterContext       *m_pFilterBufferSink;

  LAVPixelFormat        m_filterPixFmt;
  int                   m_filterWidth;
  int                   m_filterHeight;
  LAVFrame              m_FilterPrevFrame;

  BOOL                  m_LAVPinInfoValid;
  LAVPinInfo            m_LAVPinInfo;

  CLAVVideoSubtitleInputPin *m_pSubtitleInput;
  CLAVSubtitleConsumer *m_SubtitleConsumer;

  LAVFrame              *m_pLastSequenceFrame;

  AM_SimpleRateChange   m_DVDRate;

  BOOL                  m_bRuntimeConfig;
  struct VideoSettings {
    BOOL TrayIcon;
    DWORD StreamAR;
    DWORD NumThreads;
    BOOL bFormats[Codec_VideoNB];
    BOOL bMSWMV9DMO;
    BOOL bPixFmts[LAVOutPixFmt_NB];
    DWORD RGBRange;
    DWORD HWAccel;
    BOOL bHWFormats[HWCodec_NB];
    DWORD HWAccelResFlags;
    DWORD HWDeintMode;
    DWORD HWDeintOutput;
    BOOL HWDeintHQ;
    DWORD DeintFieldOrder;
    LAVDeintMode DeintMode;
    DWORD SWDeintMode;
    DWORD SWDeintOutput;
    DWORD DitherMode;
    BOOL bDVDVideo;
  } m_settings;

  CBaseTrayIcon *m_pTrayIcon;

#ifdef DEBUG
  FloatingAverage<double> m_pixFmtTimingAvg;
#endif
};
```

可见该类继承了CTransformFilter，其的功能真的是非常丰富的。在这里肯定无法对其进行一一分析，只能选择其中重点的函数进行一下分析。

该类中包含了解码线程类：CDecodeThread　　m_Decoder;，这里封装了解码功能。

同时该类中包含了函数Receive(IMediaSample *pIn);，是发挥解码功能的函数，其中pIn是输入的解码前的视频压缩编码数据。

下面来看看Receive()函数：

```cpp
//处理的核心
//核心一般才有IMediaSample
HRESULT CLAVVideo::Receive(IMediaSample *pIn)
{
  CAutoLock cAutoLock(&m_csReceive);
  HRESULT        hr = S_OK;

  AM_SAMPLE2_PROPERTIES const *pProps = m_pInput->SampleProps();
  if(pProps->dwStreamId != AM_STREAM_MEDIA) {
    return m_pOutput->Deliver(pIn);
  }

  AM_MEDIA_TYPE *pmt = NULL;
  //获取媒体类型等等
  if (SUCCEEDED(pIn->GetMediaType(&pmt)) && pmt) {
    CMediaType mt = *pmt;
    DeleteMediaType(pmt);
    if (mt != m_pInput->CurrentMediaType() || !(m_dwDecodeFlags & LAV_VIDEO_DEC_FLAG_DVD)) {
      DbgLog((LOG_TRACE, 10, L"::Receive(): Input sample contained media type, dynamic format change..."));
      m_Decoder.EndOfStream();
      hr = m_pInput->SetMediaType(&mt);
      if (FAILED(hr)) {
        DbgLog((LOG_ERROR, 10, L"::Receive(): Setting new media type failed..."));
        return hr;
      }
    }
  }

  m_hrDeliver = S_OK;

  // Skip over empty packets
  if (pIn->GetActualDataLength() == 0) {
    return S_OK;
  }
  //解码
  hr = m_Decoder.Decode(pIn);
  if (FAILED(hr))
    return hr;

  if (FAILED(m_hrDeliver))
    return m_hrDeliver;

  return S_OK;
}
```

由代码我们可以看出，实际发挥出解码功能的函数是hr = m_Decoder.Decode(pIn);。

下面我们来看看CDecodeThread类的Decode()方法：

```cpp
//解码线程的解码函数
STDMETHODIMP CDecodeThread::Decode(IMediaSample *pSample)
{
  CAutoLock decoderLock(this);

  if (!CAMThread::ThreadExists())
    return E_UNEXPECTED;

  // Wait until the queue is empty
  while(HasSample())
    Sleep(1);

  // Re-init the decoder, if requested
  // Doing this inside the worker thread alone causes problems
  // when switching from non-sync to sync, so ensure we're in sync.
  if (m_bDecoderNeedsReInit) {
    CAMThread::CallWorker(CMD_REINIT);
    while (!m_evEOSDone.Check()) {
      m_evSample.Wait();
      ProcessOutput();
    }
  }

  m_evDeliver.Reset();
  m_evSample.Reset();
  m_evDecodeDone.Reset();

  pSample->AddRef();

  // Send data to worker thread, and wake it (if it was waiting)
  PutSample(pSample);

  // If we don't have thread safe buffers, we need to synchronize
  // with the worker thread and deliver them when they are available
  // and then let it know that we did so
  if (m_bSyncToProcess) {
    while (!m_evDecodeDone.Check()) {
      m_evSample.Wait();
      ProcessOutput();
    }
  }

  ProcessOutput();

  return S_OK;
}
```

这个方法乍一看感觉很抽象，好像没看见直接调用任何解码的函数。如果LAVVideo的封装的ffmpeg的libavcodec的话，应该是最终调用avcodec_decode_video2()才对啊。。。先来看看CDecodeThread这个类的定义吧！

DecodeThread.h

```cpp
/* 雷霄骅
 * 中国传媒大学/数字电视技术
 * leixiaohua1020@126.com
 *
 */
/*
 *      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.
 */

#pragma once

#include "decoders/ILAVDecoder.h"
#include "SynchronizedQueue.h"

class CLAVVideo;

class CDecodeThread : public ILAVVideoCallback, protected CAMThread, protected CCritSec
{
public:
```

```
  CDecodeThread(CLAVVideo *pLAVVideo);
  ~CDecodeThread();

  // Parts of ILAVDecoder
  STDMETHODIMP_(const WCHAR*) GetDecoderName() { return m_pDecoder ? m_pDecoder->GetDecoderName() : NULL; }
  STDMETHODIMP_(long) GetBufferCount() { return m_pDecoder ? m_pDecoder->GetBufferCount() : 4; }
  STDMETHODIMP_(BOOL) IsInterlaced() { return m_pDecoder ? m_pDecoder->IsInterlaced() : TRUE; }
  STDMETHODIMP GetPixelFormat(LAVPixelFormat *pPix, int *pBpp) { ASSERT(m_pDecoder); return m_pDecoder->GetPixelFormat(pPix, pBpp); }
  STDMETHODIMP_(REFERENCE_TIME) GetFrameDuration() { ASSERT(m_pDecoder); return m_pDecoder->GetFrameDuration(); }
  STDMETHODIMP HasThreadSafeBuffers() { return m_pDecoder ? m_pDecoder->HasThreadSafeBuffers() : S_FALSE; }


  STDMETHODIMP CreateDecoder(const CMediaType *pmt, AVCodecID codec);
  STDMETHODIMP Close();
  //解码线程的解码函数
  STDMETHODIMP Decode(IMediaSample *pSample);
  STDMETHODIMP Flush();
  STDMETHODIMP EndOfStream();

  STDMETHODIMP InitAllocator(IMemAllocator **ppAlloc);
  STDMETHODIMP PostConnect(IPin *pPin);

  STDMETHODIMP_(BOOL) IsHWDecoderActive() { return m_bHWDecoder; }

  // ILAVVideoCallback
  STDMETHODIMP AllocateFrame(LAVFrame **ppFrame);
  STDMETHODIMP ReleaseFrame(LAVFrame **ppFrame);
  STDMETHODIMP Deliver(LAVFrame *pFrame);
  STDMETHODIMP_(LPWSTR) GetFileExtension();
  STDMETHODIMP_(BOOL) FilterInGraph(PIN_DIRECTION dir, const GUID &clsid);
  STDMETHODIMP_(DWORD) GetDecodeFlags();
  STDMETHODIMP_(CMediaType&) GetInputMediaType();
  STDMETHODIMP GetLAVPinInfo(LAVPinInfo &info);
  STDMETHODIMP_(CBasePin*) GetOutputPin();
  STDMETHODIMP_(CMediaType&) GetOutputMediaType();
  STDMETHODIMP DVDStripPacket(BYTE*& p, long& len);
  STDMETHODIMP_(LAVFrame*) GetFlushFrame();
  STDMETHODIMP ReleaseAllDXVAResources();

protected:
    //包含了对进程的各种操作，重要
  DWORD ThreadProc();

private:
  STDMETHODIMP CreateDecoderInternal(const CMediaType *pmt, AVCodecID codec);
  STDMETHODIMP PostConnectInternal(IPin *pPin);

  STDMETHODIMP DecodeInternal(IMediaSample *pSample);
  STDMETHODIMP ClearQueues();
  STDMETHODIMP ProcessOutput();

  bool HasSample();
  void PutSample(IMediaSample *pSample);
  IMediaSample* GetSample();
  void ReleaseSample();

  bool CheckForEndOfSequence(IMediaSample *pSample);

private:
//各种对进程进行的操作
  enum {CMD_CREATE_DECODER, CMD_CLOSE_DECODER, CMD_FLUSH, CMD_EOS, CMD_EXIT, CMD_INIT_ALLOCATOR, CMD_POST_CONNECT, CMD_REINIT};
  //注意DecodeThread像是一个处于中间位置的东西
  //连接了Filter核心类CLAVVideo和解码器的接口ILAVDecoder
  CLAVVideo    *m_pLAVVideo;
  ILAVDecoder  *m_pDecoder;

  AVCodecID    m_Codec;

  BOOL         m_bHWDecoder;
  BOOL         m_bHWDecoderFailed;

  BOOL         m_bSyncToProcess;
  BOOL         m_bDecoderNeedsReInit;
  CAMEvent     m_evInput;
  CAMEvent     m_evDeliver;
  CAMEvent     m_evSample;
  CAMEvent     m_evDecodeDone;
  CAMEvent     m_evEOSDone;

  CCritSec     m_ThreadCritSec;
  struct {
    const CMediaType *pmt;
    AVCodecID codec;
    IMemAllocator **allocator;
    IPin *pin;
  } m_ThreadCallContext;
  CSynchronizedQueue<LAVFrame *> m_Output;

  CCritSec     m_SampleCritSec;
  IMediaSample *m_NextSample;
```

```
125.
126.   IMediaSample *m_TempSample[2];
127.   IMediaSample *m_FailedSample;
128.
129.   std::wstring m_processName;
130. };
```

从名字上我们可以判断，这个类用于管理解码的线程。在这里我们关注该类里面的两个指针变量：

CLAVVideo *m_pLAVVideo;
ILAVDecoder *m_pDecoder;
其中第一个指针变量就是这个工程中最核心的类CLAVVideo，而第二个指针变量则是解码器的接口。通过这个接口就可以调用具体解码器的相应方法了。（注：在源代码中发现，解码器不光包含libavcodec，也可以是wmv9等等，换句话说，是可以扩展其他种类的解码器的。不过就目前的情况来看，lavvideo似乎不如ffdshow支持的解码器种类多）

该类里面还有一个函数：

ThreadProc()

该函数中包含了对线程的各种操作，其中包含调用了ILAVDecoder接口的各种方法：

```cpp
//包含了对进程的各种操作
DWORD CDecodeThread::ThreadProc()
{
  HRESULT hr;
  DWORD cmd;

  BOOL bEOS = FALSE;
  BOOL bReinit = FALSE;

  SetThreadName(-1, "LAVVideo Decode Thread");

  HANDLE hWaitEvents[2] = { GetRequestHandle(), m_evInput };
  //不停转圈，永不休止
  while(1) {
    if (!bEOS && !bReinit) {
      // Wait for either an input sample, or an request
      WaitForMultipleObjects(2, hWaitEvents, FALSE, INFINITE);
    }
    //根据操作命令的不同
    if (CheckRequest(&cmd)) {
      switch (cmd) {
          //创建解码器
      case CMD_CREATE_DECODER:
        {
          CAutoLock lock(&m_ThreadCritSec);
          //创建
          hr = CreateDecoderInternal(m_ThreadCallContext.pmt, m_ThreadCallContext.codec);
          Reply(hr);

          m_ThreadCallContext.pmt = NULL;
        }
        break;
      case CMD_CLOSE_DECODER:
        {
            //关闭
          ClearQueues();
          SAFE_DELETE(m_pDecoder);
          Reply(S_OK);
        }
        break;
      case CMD_FLUSH:
        {
            //清楚
          ClearQueues();
          m_pDecoder->Flush();
          Reply(S_OK);
        }
        break;
      case CMD_EOS:
        {
          bEOS = TRUE;
          m_evEOSDone.Reset();
          Reply(S_OK);
        }
        break;
      case CMD_EXIT:
        {
            //退出
          Reply(S_OK);
          return 0;
        }
        break;
      case CMD_INIT_ALLOCATOR:
        {
```

```
            CAutoLock lock(&m_ThreadCritSec);
            hr = m_pDecoder->InitAllocator(m_ThreadCallContext.allocator);
            Reply(hr);

            m_ThreadCallContext.allocator = NULL;
          }
          break;
        case CMD_POST_CONNECT:
          {
            CAutoLock lock(&m_ThreadCritSec);
            hr = PostConnectInternal(m_ThreadCallContext.pin);
            Reply(hr);

            m_ThreadCallContext.pin = NULL;
          }
          break;
        case CMD_REINIT:
          {
              //重启
            CMediaType &mt = m_pLAVVideo->GetInputMediaType();
            CreateDecoderInternal(&mt, m_Codec);
            m_TempSample[1] = m_NextSample;
            m_NextSample = m_FailedSample;
            m_FailedSample = NULL;
            bReinit = TRUE;
            m_evEOSDone.Reset();
            Reply(S_OK);
            m_bDecoderNeedsReInit = FALSE;
          }
          break;
        default:
          ASSERT(0);
        }
      }

      if (m_bDecoderNeedsReInit) {
        m_evInput.Reset();
        continue;
      }

      if (bReinit && !m_NextSample) {
        if (m_TempSample[0]) {
          m_NextSample = m_TempSample[0];
          m_TempSample[0] = NULL;
        } else if (m_TempSample[1]) {
          m_NextSample = m_TempSample[1];
          m_TempSample[1] = NULL;
        } else {
          bReinit = FALSE;
          m_evEOSDone.Set();
          m_evSample.Set();
          continue;
        }
      }
      //获得一份数据
      IMediaSample *pSample = GetSample();
      if (!pSample) {
        // Process the EOS now that the sample queue is empty
        if (bEOS) {
          bEOS = FALSE;
          m_pDecoder->EndOfStream();
          m_evEOSDone.Set();
          m_evSample.Set();
        }
        continue;
      }
      //解码
      DecodeInternal(pSample);

      // Release the sample
      //释放
      SafeRelease(&pSample);

      // Indicates we're done decoding this sample
      m_evDecodeDone.Set();

      // Set the Sample Event to unblock any waiting threads
      m_evSample.Set();
    }

    return 0;
  }
```

先分析到这里了，至于ILAVDecoder接口方面的东西下篇文章再写。

文章标签： LAVFilter 源代码 解码 ffmpeg directshow

个人分类： LAV Filter

所属专栏： 开源多媒体项目源代码分析