第十八期：Android音视频的编解码

eoe特刊

android

eoeAndroid音视频的编解码

eoeAndroid audio and video codec

android eoe 开发者门户

优亿市场 Android应用发布与分享平台

目录

【Android 音频入门介绍】

1.1 android 音频介绍

这个世界音频设备千变万化，Android 也不可能为每种设备都提供支持。Android 定义了一个框架，这个框架来适配底层的音频设备。该适配层的定义位于：

hardware/libhardware_legacy/include/hardware_legacy/AudioHardwareInterface.h

要想视频底层的音频设备必须要继承该文件中定义的 AudioStreamOut，AudioStreamIn，AudioHardwareInterface 等类，并实现 createAudioHardware 函数。

下面我们看一下 Android 创建音频设备的代码，代码位于：

frameworks/base/libs/audioflinger/AudioHardwareInterface.cpp

该文件有如下代码：

```
AudioHardwareInterface* AudioHardwareInterface::create()
{
/*
* FIXME: This code needs to instantiate the correct audio device
* interface. For now - we use compile-time switches.
*/
AudioHardwareInterface* hw = 0;
char value[PROPERTY_VALUE_MAX];
#ifdef GENERIC_AUDIO
hw = new AudioHardwareGeneric();
#else
// if running in emulation - use the emulator driver
if (property_get("ro.kernel.qemu", value, 0)) {
LOGD("Running in emulation - using generic audio driver");
hw = new AudioHardwareGeneric();
}
else {
LOGV("Creating Vendor Specific AudioHardware");
hw = createAudioHardware();
}
#endif
if (hw->initCheck() != NO_ERROR) {
LOGW("Using stubbed audio hardware. No sound will be produced.");
delete hw;
hw = new AudioHardwareStub();
}
#ifdef WITH_A2DP
hw = new A2dpAudioInterface(hw);
#endif
#ifdef ENABLE_AUDIO_DUMP
// This code adds a record of buffers in a file to write calls made by AudioFlinger.
// It replaces the current AudioHardwareInterface object by an intermediate
```

```
one which
    // will record buffers in a file (after sending them to hardware) for testing purpose.
    // This feature is enabled by defining symbol ENABLE_AUDIO_DUMP.
    // The output file is set with setParameters("test_cmd_file_opening PCM dump interface");
    hw = new AudioDumpInterface(hw); // replace interface
    #endif
    return hw;
    }
```

从代码中我们可以看出如果定义了 GENERIC_AUDIO 的宏，则会创建 AudioHardwareGeneric，如果是模拟器的话，AudioHardwareGeneric 会不能初始化，进而创建 AudioHardwareStub。这两个类都是 Audio 设备的适配层，是 Android 默认提供的。模拟器都是用 AudioHardwareStub，不会有声音输出。设备都是用 AudioHardwareGeneric，因为默认 GENERIC_AUDIO 是设置的。

一般我们只关心 AudioHardwareGeneric 实现，谁会去给模拟器去调试声音呢，反正我没这个闲心。首先说明一下这个音频适配层是 Android 自带的，可以保证你的音频设备正常运行，但是不能发挥设备的最佳性能。通过后面的描述你将会了解。AudioHardwareGeneric 的定义位于：

frameworks/base/libs/audioflinger/AudioHardwareGeneric.cpp

查看源码你会发现这个适配层需要实现设备/dev/eac，并且该设备只输出 44.1khz 采样率的音频数据给 /dev/eac 设备，如果不是 44.1khz 的采样率的数据，AudioHardwareGeneric 会经过 Resample 过程把它转换成 44.1kHZ 的音频数据，然后再输出给音频设备。44.1kHZ 音频数据是最普遍的音频采样率，大部分 Mp3 都是以这个采样率压缩的，所以选择这个采样率做为默认采样率还是有一定的合理性的。AudioHardwareGeneric 是软件实现 Resample 过程是，效率会比较低。很多音频设备支持不同采样率的数据，可以理解成硬件实现 Resample 过程。

通过上面的描述我们可以知道这个通用音频适配层只是让你的设备可以用而已，不能发挥设备的性能优势，如果你的设备对音频质量有更高的要求，必须要自己实现音频适配层。谷歌只能保证你的音频可以播放，但是不能保证效率(他也没有办法保证效率)。

1.2 所支持的音频格式

对于播放，Android 支持各种各样的音频文件格式和编解码。对于录音的支持少一些，以后我们学到录音部分将会讨论这点。

AAC：

高级音频编码(以及其扩展：HE AAC)编解码,.m4a，.3gp 文件.AAC 是一个流行的标准，IPOD 和其他便携式媒体播放器都使用它。Android 在 MPEG4 音频文件和 3GP 文件内（都是基于 MPEG4 格式）支持这种音频格式。最近 AAC 的附加规范 HE AAC 也被支持了。

MP3:

MPGE-1 音频层 3,.mp3 文件。Android 支持 MP3，MP3 可能是使用最广泛的音频编解码，这允许 Android 通过各种网站和音乐商店来使用大部分在线音频。

AMR:

自适应多速率编解码(AMR-NB,AMR-WB),.3gp,.amr 文件。AMR 音频编解码已经被标准化了，主要被 3GPP（第三代合作伙伴项 目)用于语音音频编解码。3GPP 是一个为其合作伙伴创建规范的电信行业机构。换句话说，AMR 编解码主要用于现代移动电话的语音呼叫程序,并且手机厂商 和手机携带者普遍都支持这个格式。AMR 这格式一般对语音编码很有用，但对更复杂的类型表现的不够好，比如音乐。

Ogg:

Ogg Vorbis,.ogg 文件。Ogg Vorbis 是个开源的，无专利费的音频编解码。其品质可媲美商业性的，需缴纳专利费的编解码比如 MP3，AAC。它由一群自愿者开发，当前由 Xiph.Org 基金会负责维护。

PCM:

脉冲编码调制通常被用在 WAVE,WAV 文件，.wav 文件。PCM 这技术主要用于音频在电脑和其他电子音频设备上的存储。它通常是个未压缩的音频 文件，其数据代表随着时间流逝一段音频的振幅。“采样率”是多长时间一次一个振幅读取被存储起来。“位深度”是指多少位被用来代表一个单独的样本。一段 16KHZ 采样率，32 位位深度的音频数据是指它包含每秒钟 16000 个的 32 位的数据用来表示音频振幅。采样率和位深度越高，数字化音频越精准。采样率 和位深度也决定了音频文件的大小。Android 在 WAV 文件内支持 PCM 音频数据。WAV 是 PC 上的一个长期存在的标准音频格式。

1.3 通过 Intent 使用内建的音频播放器

正如使用摄像头，在一个应用程序里提供播放音频文件的能力，最容易的方法就是使用内建的“音乐”程序的功能。这个程序有个用户熟悉的界面，能播放所有 Android 支持的格式，并且能通过一个 intent，被触发去播放一个指定的文件。

普通的 android.content.Intent.ACTION_VIEW intent，其数据设置为一个音频文件的 Uri，并指定其 MIME 类型，这样 Android 会自动选择一个合适的应用程序来播放。这个程序应该是内建的 音乐播放程序，但用户可能被提供其他的选项，如果他/她安装了其他的音频播放软件。

```
1.  Intent intent = new Intent(android.content.Intent.ACTION_VIEW);
2.  intent.setDataAndType(audioFileUri, "audio/mp3");
3.  startActivity(intent);
```

注解：MIME 全称是 Multipurpose Internet Mail Extensions（多用途互联网邮件扩展）。它起初专门用来帮助电子邮件客户端发送和接收附件。但它的使用范围从电子邮件极大地扩展到其他的通讯协 议，包括 HTTP，标准万维网服务。Android 使用 MIME 类型来解析 intent，并且用它来决定应该选择哪个应用程序来处理 intent.

每个文件类型都有特定的（有时候不止一个）MIME 类型。MIME 类型通过至少由 2 部分组成，由斜杠分开的字符来指定。第一部分是更通用的类型，比如 “audio”.第二部分是更具体的类型，比如"mpeg".一个通用的类型"audio"和一个更具体的类型"mpeg"将产生一个 “audio/mpeg"MIME 类型字符,这个 MIME 类型通常用于 MP3 文件。

1.4 Android 音频实例分析

这里有个通过一个 intent 触发内建的音频播放程序的完整例子。

```
1.  package com.apress.proandroidmedia.ch5.intentaudioplayer;
2.  import java.io.File;
3.  import android.app.Activity;
4.  import android.content.Intent;
5.  import android.net.Uri;
6.  import android.os.Bundle;
7.  import android.os.Environment;
8.  import android.view.View;
9.  import android.view.View.OnClickListener;
10. import android.widget.Button;
```

我们的 activity 在触发音频播放之前会一直监听一个 Botton 是否被按下。

```
1.  public class AudioPlayer extends Activity implements OnClickListener {
2.  Button playButton;
3.  @Override
4.  public void onCreate(Bundle savedInstanceState) {
5.  super.onCreate(savedInstanceState);
6.  setContentView(R.layout.main);
```


我们将 content view 设置为我们的 XML 后，我们能得到一个 Button 的引用，并将我们的 activity(this)设为 OnClickListener.

```
1.  playButton = (Button) this.findViewById(R.id.Button01);
2.  playButton.setOnClickListener(this);
3.  }
```

当我们的 Button 被点击，OnClick 方法会被调用。在这个方法里，我们用一个普通的 android.content.Intent.ACTION_VIEW 来构建 intent,然后创建一个文件对象，这个对象是 SD 卡上已经存在的音频文件的索引。这种情况下，这个音频文件被手动放置到 SD 卡的“Music" 目录下，这个目录通常放置和音乐相关的音频文件。

```
1.  public void onClick(View v) {
2.  Intent intent = new Intent(android.content.Intent.ACTION_VIEW);
3.  File sdcard = Environment.getExternalStorageDirectory();
4.  File audioFile = new File(sdcard.getPath() + "/Music/goodmorningandroid.mp3");
```

接下来，我们设置 intent 的数据为来源于音频文件的 Uri 并将其类型设置为 MIME 类

型，audio/mp3.最后传递我们的 intent 给 startActivity 来触发内建的音乐播放程序。

```
1.      intent.setDataAndType(Uri.fromFile(audioFile), "audio/mp3");
2.      startActivity(intent);
3.  }
```

下面是一个简单的 XML 布局文件，其中 Button 的文本为"Play Audio"，前面所述的 activity 会用到这个 Button.

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.      android:orientation="vertical"
4.      android:layout_width="fill_parent"
5.      android:layout_height="fill_parent"
6.      >
7.      <Button android:text="Play Audio" android:id="@+id/Button01"
8.      android:layout_width="wrap_content" android:layout_height="wrap_content"
></Button>
9.  </LinearLayout>
```

1.5 Android 支持的视频格式

先简要说明下 Android 支持的视频格式，如下：

Type	Format	Encoder	Decoder	File Type(s) Supported
Audio	AAC LC/LTP		X	3GPP (.3gp) and MPEG-4 (.mp4, .m4a). No support for raw AAC (.aac)
	HE-AACv1 (AAC+)		X	
	HE-AACv2 (enhanced AAC+)		X	
	AMR-NB	X	X	3GPP (.3gp)
	AMR-WB		X	3GPP (.3gp)
	MP3		X	MP3 (.mp3)
	MIDI		X	Type 0 and 1 (.mid, .xmf, .mxmf). Also RTTTL/RTX (.rttl, .rtx), OTA (.ota), and iMelody (.imy)
	Ogg Vorbis		X	Ogg (.ogg)
	PCM/WAVE		X	WAVE (.wav)
Image	JPEG	X	X	JPEG (.jpg)
	GIF		X	GIF (.gif)
	PNG		X	PNG (.png)
	BMP		X	BMP (.bmp)
Video	H.263	X	X	3GPP (.3gp) and MPEG-4 (.mp4)
	H.264 AVC		X	3GPP (.3gp) and MPEG-4 (.mp4)
	MPEG-4 SP		X	3GPP (.3gp)

由于硬件的不同，不同的手机支持的编解码方式有些不一样，在 T-Mobile G1 实际设备中增加了对 WMA，WMV，H.264 AVC 格式解码的支持。Android 支持的音/视频编码方式仅包括：AMR-NB，H.263， 输出的视频格式也只*.3gp 或者*.mp4，这点在以后的

开发中需要注意。

对 Android 的编解码有了一些了解后，我们再去研究下如何在 Android 上来播放/录制视频，打开 Android SDK 中关于媒体方面的说明，摘要其主要部分如下：

MediaPlayer	MediaPlayer class can be used to control playback of audio/video files and streams.
MediaRecorder	Used to record audio and video.
MediaRecorder.AudioEncoder	Defines the audio encoding.
MediaRecorder.AudioSource	Defines the audio source.
MediaRecorder.OutputFormat	Defines the output format.
MediaRecorder.VideoEncoder	Defines the video encoding.
MediaRecorder.VideoSource	Defines the video source.

首先注意的就是：MediaPlayer，MediaRecoder，主要是用来播放视频与录制视频的类；由于这 2 个类都比较复杂，本篇在后面会详细介绍如何使用 MediaRecoder，MediaPlayer 将在以后篇幅中再详细介绍。

1.6 Android 支持的编码方式:

其他的几个类，定义 Android 支持的编码方式，如下：

类	说明	编码方式定义
MediaRecorder.AudioEncoder	音频编码当时	AMR_NB：AMR NB 编码
MediaRecorder.AudioSource	声音采样设备	MIC：麦克风
MediaRecorder.OutputFormat	录制输出格式	MPEG_4：*.mp4 RAW_AMR：*.amr THREE_GPP：*.3gp
MediaRecorder.VideoEncoder	视频编码方式	H263：H.263 编码 H264：H.264 编码 MPEG_4_SP：mp4 编码
MediaRecorder.VideoSource	视频采样设备	CAMERA：数码相机

下面我们来看看如何使用 MediaRecorder 录制声音:

先对声音录制有个大致的了解，需要设置声音数据的来源、输出编码方式、输出文件路径、输出文件格式等。有一点需要说明的是：输出文件格式就相当于一个容器，具体采用什么编码需要指定编码格式。编码一样可能输出格式不一样，输出格式一样其编码方式可能不一样。

android.media 包含与媒体子系统交互的类。使用 android.media.MediaRecorder 类进行媒体采样，包括音频和视频。MediaRecorder 作为状态机运行。需要设置不同的参数，比如源设备和格式。设置后，可执行任何时间长度的录制，直到用户停止。
录制音频主要片段如下：

```
MediaRecorder mrec ;
File audiofile = null;
private static final String TAG=”SoundRecordingDemo”;
protected void startRecording () throws IOException
{
mrec.setAudioSource(MediaRecorder.AudioSource.MIC);
mrec.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
mrec.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);
if (mSampleFile == null)
{
File sampleDir = Environment.getExternalStorageDirectory();
try
{
audiofile = File.createTempFile(”ibm”, “.3gp”, sampleDir);
}catch (IOException e)
{
Log.e(TAG,”sdcard access error”);
return;
}
}
mrec.setOutputFile(audiofile.getAbsolutePath());
mrec.prepare();
mrec.start();
}
protected void stopRecording ()
{
mrec.stop();
mrec.release();
processaudiofile(audiofile.getAbsolutePath());
}
protected void processAudiofile ()
{
ContentValues values = new ContentValues(3);
```

```
long current = System.currentTimeMillis();
values.put(MediaStore.Audio.Media.TITLE, "audio" + audiofile.getName());
values.put(MediaStore.Audio.Media.DATE_ADDED, (int) (current / 1000));
values.put(MediaStore.Audio.Media.MIME_TYPE, "audio/3gpp");
values.put(MediaStore.Audio.Media.DATA, audiofile.getAbsolutePath());
ContentResolver contentResolver = getContentResolver();
Uri base = MediaStore.Audio.Media.EXTERNAL_CONTENT_URI;
Uri newUri = contentResolver.insert(base, values);
sendBroadcast(new Intent(Intent.ACTION_MEDIA_SCANNER_SCAN_FILE, newUri));
}
```

对以上代码解析如下：

1. 在 startRecording 方法中，实例化并初始化 MediaRecorder 的实例。
2. 输入源被设置为麦克风（MIC）。
3. 输出格式被设置为 3GPP（*.3gp 文件），这是移动设备专用的媒体格式。
4. 编码器被设置为 AMR_NB，这是音频格式，采样率为 8 KHz。NB 表示窄频。SDK 文档 解释了不同的数据格式和可用的编码器。
5. 音 频文件存储在存储卡而不是内存中。External.getExternalStorageDirectory() 返回存储卡位置的名称，在该目录中将创建一个临时文件名。然后，通过调用 setOutputFile 方法将文件关联到 MediaRecorder 实例。音频数据将存储到该文件中。
6. 调用 prepare 方法完成 MediaRecorder 的初始化。准备开始录制流程时，将调用 start 方法。在调用 stop 方法之前，将对存储卡上的文件进行录制。release 方法将释放分配给 MediaRecorder 实例的资源。
7. 音频采样完成之后， 向设备的媒体库添加该音频。在该代码样例中，processAudiofile 方法将音频添加到媒体库。使用 Intent 通知设备上的媒体应用程序有新内容可用。
8. 最后还需要注意的是，录制声音需要一定的权限，需要向 AndroidManifest.xml 添加权限声明：

```
<uses-permission
android:name="android.permission.RECORD_AUDIO"></uses-permission>
```

总结说明：

1. Android SDK 中对视频开发方面提供了很大的方便，但是在使用的过程中发现，这些类封装的都很高层，很多地方估计也无法满足开发人员的需求，主要体现在缺少对文件流操作等底层接口，我们只能通过控制状态来控制文件流，缺少这方面操作增加了 Android 视频开发的难度，比如说增加其他编码器等。
2. Android T-Mobile G1 可以在线看 Youtube 视频，然而我们在开发中却发现，其无法播放 FLV 格式的视频，这就有些不明白了？估计在未来应该会支持的，FLV 是当前最流 行的流媒体格式，如果不支持对用户在 Android 上直接看这些 FLV 视频会有很大的影响。

【Android 音频视频开发】

视频录制功能正在走来,在 Android sdk 中有与之相关的类:android.media.MediaRecorder 当然，因为模拟器上没有提供必要的硬件设施，所以在学习过程中并不能实现。Media 能够播放来自任何地方的文件：一个实际的文件资源、系统中的一个文件或者是一个可用的网络链接。

2.1 如何播放 media 音频

1、将文件放到你的工程的 res/raw 文件夹中，在这个文件夹中，Eclipse 插件将会找到它，同时，会将这个资源与你的 R

2、创建一个 MediaPlayer,并使用 MediaPlayer.create 与资源相关联起来，之后在实例中使用 start()方法。

例如：MediaPlayer mp=MediaPlayer.create(context,R.raw.sound_file_1);
mp.start();

如果要想停止播放，使用 stop()方法。如果你想稍后重新播放这段 media，你必须在再次使用 start()方法之前使用 reset()方法和 prepare()方法来操作 MediaPlayer 对象。（create()第一次调用 prepare()）如果想暂停播放，可以使用 pause()方 法。在你暂停的地方恢复播放功能使用 start()方法即可实现。

下面介绍如何播放一个文件：

1、用 new 创建一个 MediaPlayer 实例；

2、调用 setDataSource()方法，这个方法有一个 String 类型的参数，这个 String 类型的参数包含了你所要播放的文件的路径－本地文件系统或者是 URL；

3、之后，先调用 prepare()方法，然后才是 start()方法。

Java 代码：
复制到剪贴板 Java 代码

```
1. MediaPlayer mp=new MediaPlayer();
2. mp.setDataSource(PATH_TO_FILE);
3. mp.prepare();
4. mp.start();
```

5. 需要注意的一点是：如果你传递的是一个 URL 方式的文件，那么这个文件必须是可以下载的，并且是不间断的，简单地说就是在播放的同时进行着下载。

2.2 如何录制 media 音频资源

1、使用 new 创建一个实例 android.media.MediaRecorder；

2、创建一个 android.content.ContentValues 实例并设置一些标准的属性，像 TITLE，TIMESTAMP，最重要的是 MIME_TYPE;

3、创建一个要放置的文件的路径，这可以通过 android.content.ContentResolver 在内容数据库中来创建一个入口，并且自动地标志一个取得这个文件的路径。

4、使用 MediaRecorder.setAudioSource()方法来设置音频资源；这将会很可能使用到 MediaRecorder.AudioSource.MIC;

5、使用 MediaRecorder.setOutputFormat()方法设置输出文件格式；

6、用 MediaRecorder.setAudioEncoder()方法来设置音频编码；

7、最后，prepare()和 start()所录制的音频，stop()和 release()在要结束的时候调用。

2.3 实例分析

Java 代码:
复制到剪贴板 Java 代码

```
1.  recorder=new MediaRecorder();
2.  ContentValues values=new ContentValues(3);
3.  values.put(MediaStore.MediaColumns.TITLE,SOME_NAME_HERE);
4.  values.put(MediaStore.MediaColumns.TIMESTAMP,System.currentTimeMillis());
5.  values.put(MediaStore.MediaColumns.MIME_TYPE,recorder.getMimeContentType();

6.  ContentResolver contentResolver=new ContentResolver();
7.  Uri base=MediaStore.Audio.INTERNAL_CONTENT_URI;
8.  Uri newUri=contentResolver.insert(base,values);
9.  //在所给定的 URL 中向一个表格插入一列数据
10. //函数原型：final Uri insert(Url,ContentValues values);
11. if(newUri==null){
12. //这里需要异常处理，我们在这里不能创建一个新的内容入口
13. }
14. String path=contentResolver.getDataFilePath(newUri);
15. //可以使用 setPreviewDisplay()来陈列一个 preview 来使 View 适合
16. recorder.setAudioSource(MediaRecorder.AudioSource.MIC);
17. recorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
18. recorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);
19. recorder.setOutputFile(path);
20. recorder.prepare();
21. recorder.start();
```

停止录制：

Java 代码：

复制到剪贴板 C/C++代码

```
1. recorder.stop();
2. recorder.release();
```

在录制音频资源的过程中，使用到了 ContentValues 这个类，下面来解说这个类。
ContentValues 这个类是用来存储一系列的值的，这些值要求 ContentResolver 能够 process 的。
ContentValues(int size)构造函数使用所给定的初始值创建一个空系列的值。
ContentValues(ContentValues from)这个构造函数创建一个从给定的 ContentValues 中来进行复制所产生的值。

这个类有如下的常用方法:

void clear() 删除所有的值
boolean containsKey(String key) 如果这个对象有已命名的值就返回真
int describeContents() 描述值类型
Object get(String key) 获得值
void put(String key,Integer value)增加一个值到对应的 set 中
还有一个类就是 ContentResolver，这个类想内容模型提供了应用程序数据

【Android 多媒体相关】

3.1 多媒体系统的结构和层次介绍

3.1.1 多媒体系统的结构介绍

Android的多媒体部分的框架涉及到应用层、JAVA框架、C语言框架、硬件抽象层等环节。多媒体主要包括两方面的内容：输入输出环节（音频视频的输入输出）中间处理环节（编解码环节）其中，输入输出环节由其他方面的硬件抽象
层实现，中间处理环节主要由PacketVideo实现，可以使用硬件加速。

Android的多媒体应用业务：

- Music Player
- Video Player
- Camera
- Sound Recorder
- Camcorder
- Video Telephone

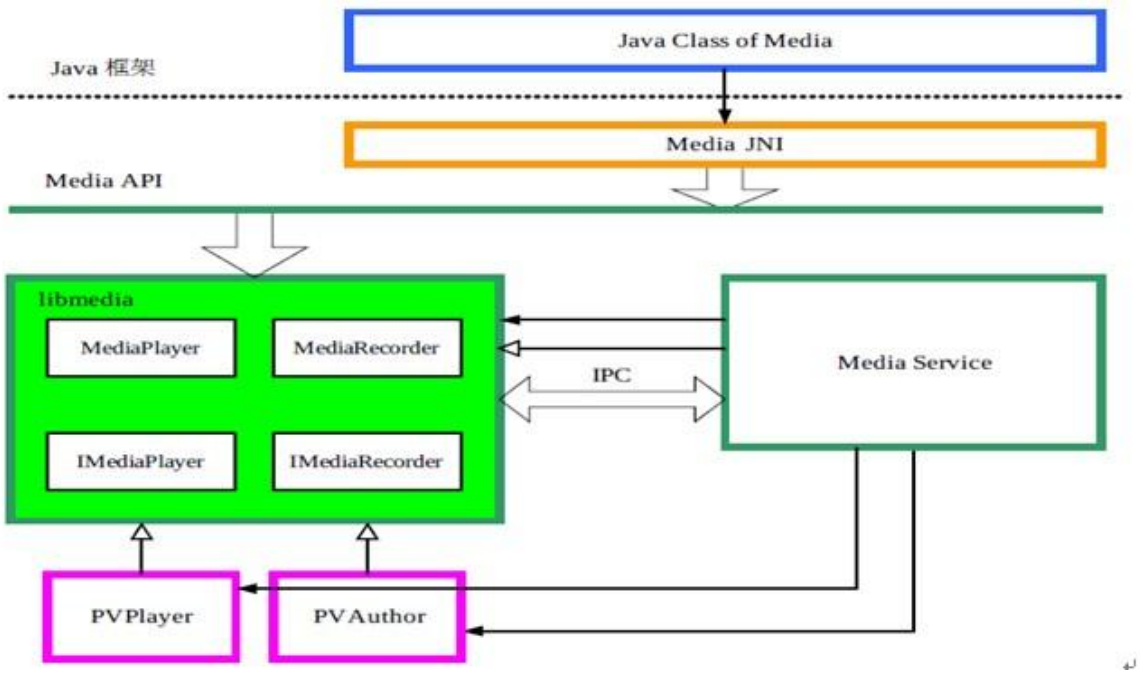
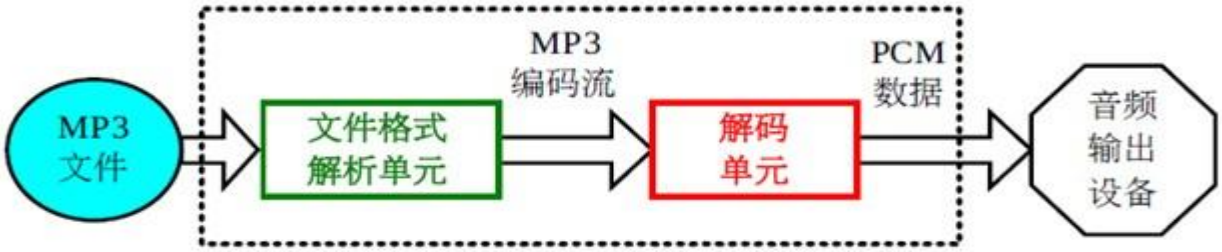


图1 多媒体系统体系结构

从多媒体应用实现的角度，主要包括两方面的内容：

- 输入输出环节（音频、视频纯数据流的输入输出系统）
- 中间处理环节（文件格式处理环节和编解码环节）

以一个MP3播放器为例，从功能的角度就是将一个mp3格式的文件作为播放器的输入，将声音从播放设备输出。从实现的角度，MP3播放器经过了一下的阶段：MP3格式的文件解析、MP3编码流的解码、PCM输出的播放。



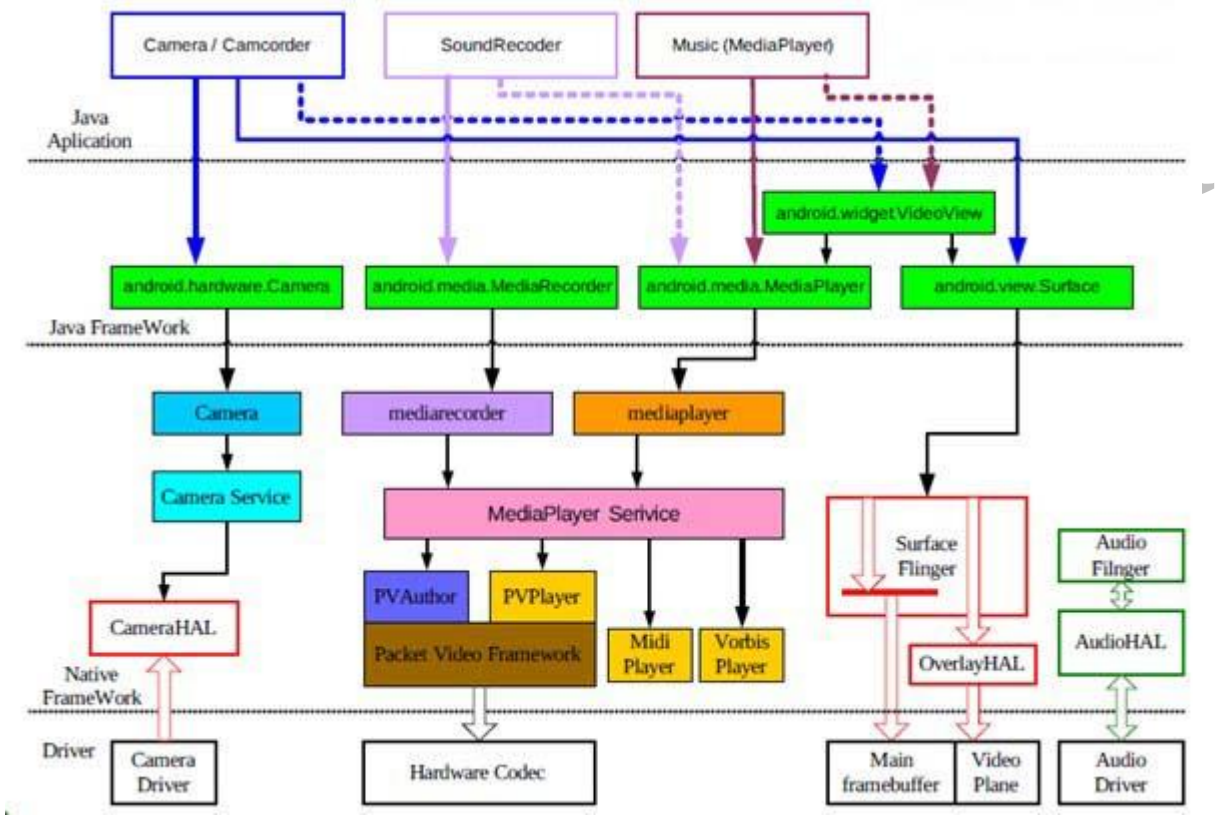


图2 多媒体系统结构

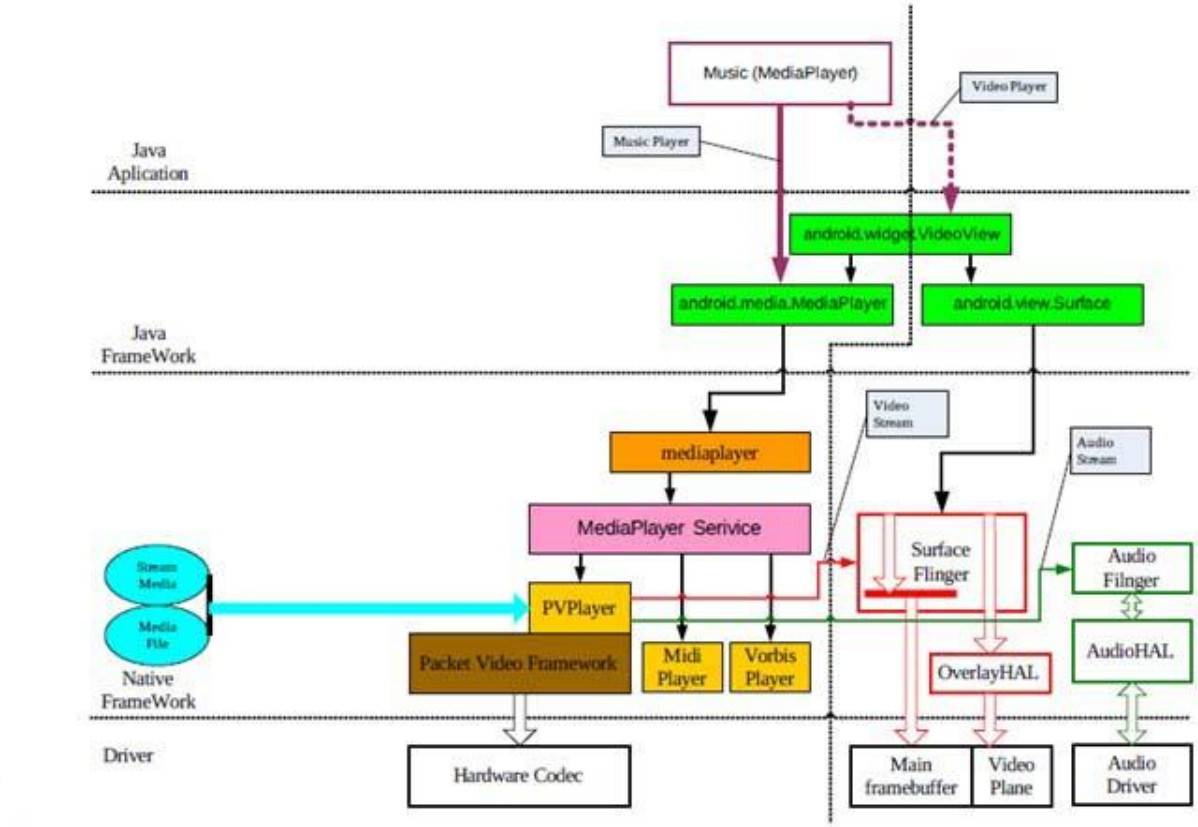


图3 音频视频播放器

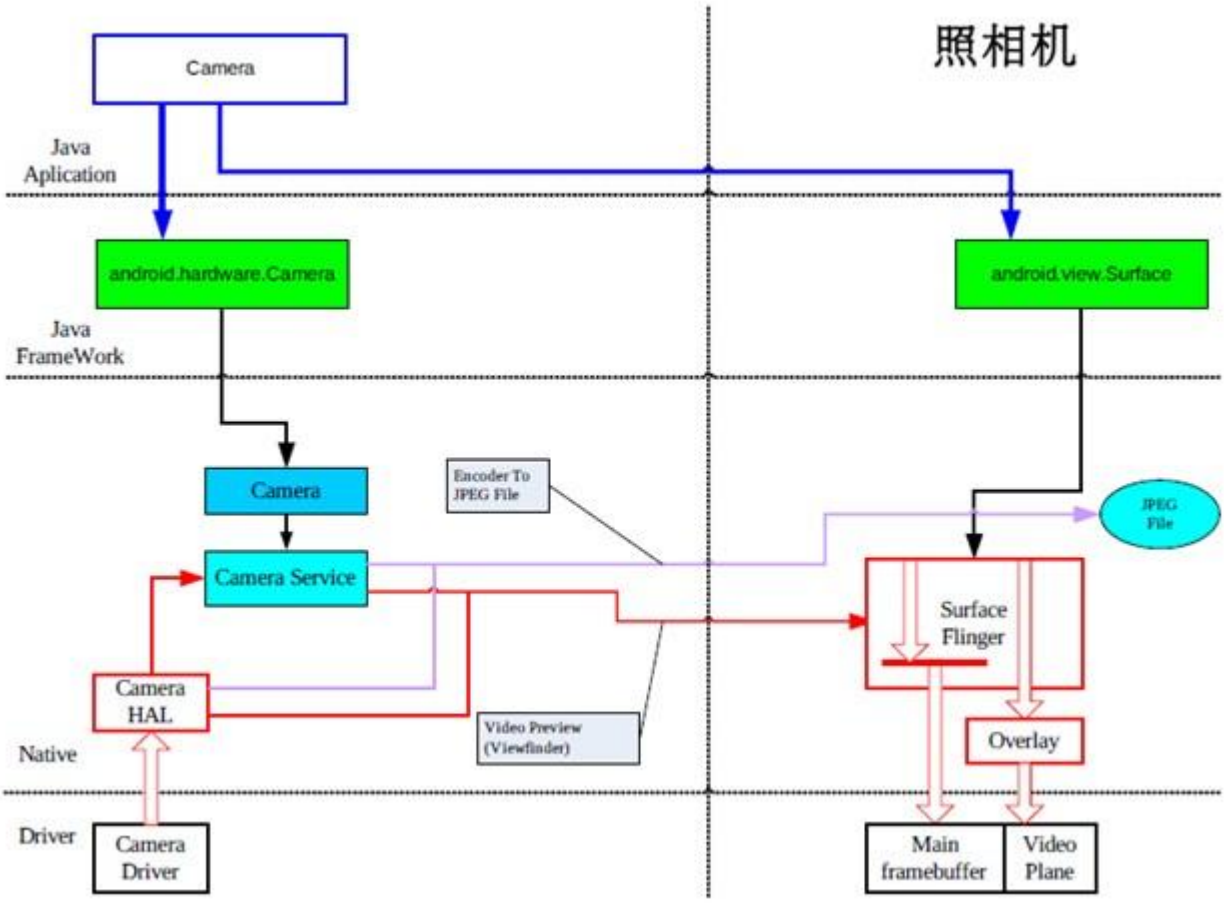


图4 Camera 结构图

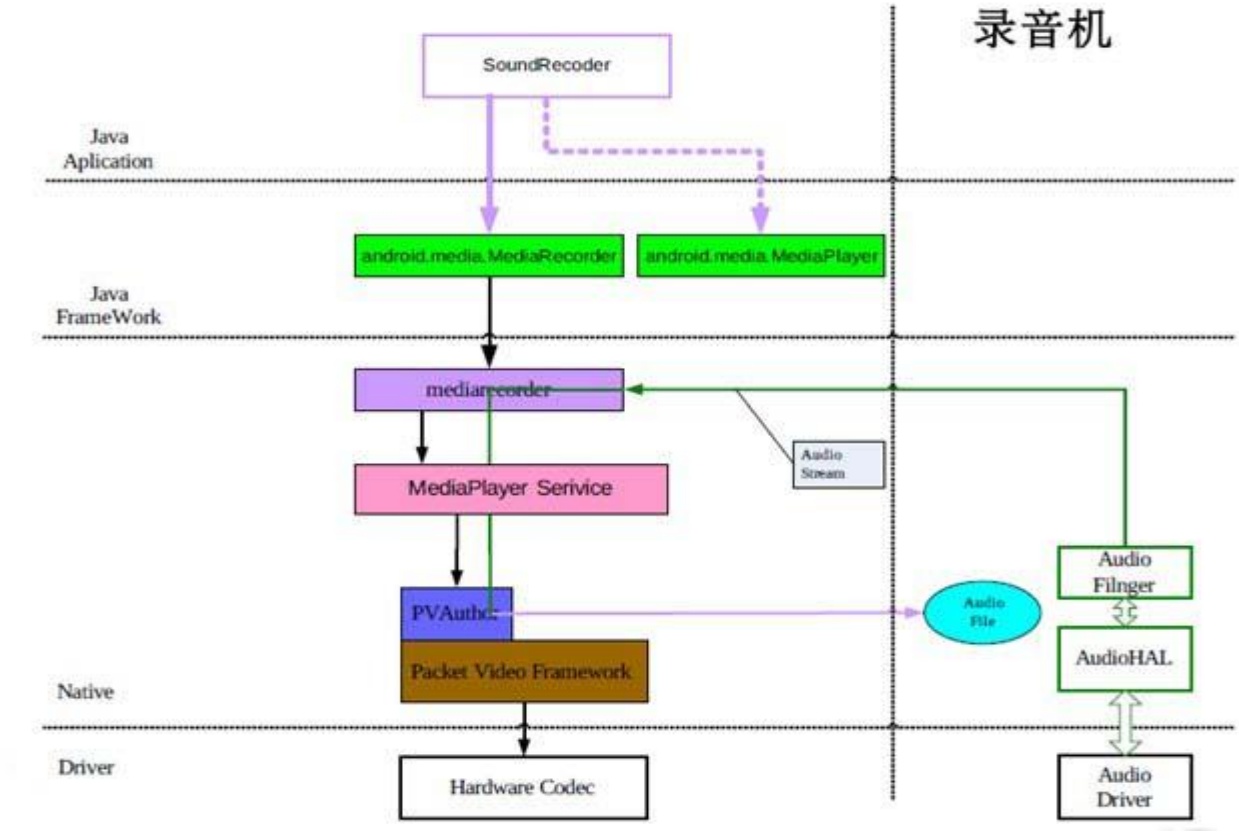


图5 录音机结构图

3.1.2 多媒体的各个层次

- ✧ libmedia的框架部分
- ✧ 多媒体服务
- ✧ 多媒体部分的JNI代码
- ✧ 多媒体部分的JAVA 框架代码
- ✧ 类android.widget.VideoView
- ✧

Android多媒体部分的C语言部分的核心是media库，它主要记录了媒体播放器和媒体记录器的框架。media库向上层通过JNI提供接口，下层通过Packet Video等实现。

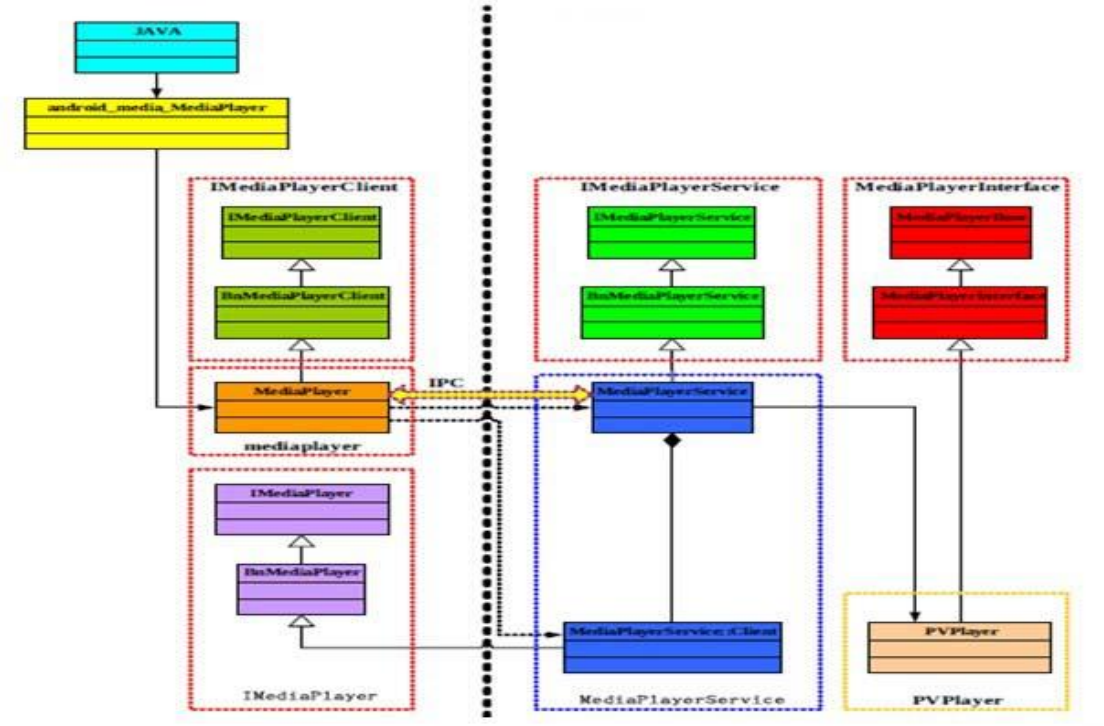


图6 libmedia框架部分

多媒体服务的守护进程的代frameworks/base/media/mediaserver/
其中只有一个源文件main_mediaserver.cpp，将被编译成为一个可执行程序mediaserver。
它负责启动了多媒体服务、照相机服务，音频服务这三个服务service media
/system/bin/mediaserver　user media
group system audio camera graphics inet net_bt net_bt_admin
media的JAVA本地调用部分（JNI）：　frameworks/base/media/*
这部分内容编译成为目标是libmedia_jni.so.主要文件是：
android_media_MediaPlayer.cpp
android_media_MediaRecorder.cpp
VideoView是一个media集成的高层的JAVA类，这个类的文件是：
frameworks/base/core/java/android/widget/文件的类型是VideoView.java类的路径是
android.widget.javaVideoView是一个集成了MediaPlayer和SurfaceView的类，可以作为一个UI元素（View）直接放置在JAVA应用的界面中，用于视频的播放。

3.2 多媒体实现的核心部分 OpenCore

OpenCore 概述：

OpenCore 的另外一个常用的称呼是 PacketVideo，它是 Android 的多媒体核心。事实上，PacketVideo 是一家公司的名称，而 OpenCore 是这套多媒体框架的软件层的名称。在 Android 的开发者中间，二者的含义基本相同。对比 Android 的其它程序库，OpenCore 的代码非常庞大，它是一个基于 C++的实现，定义了全功能的操作系统移植层，各种基本的功能均被封装成类的形式，各层次之间的接口多使用继承等方式。

OpenCore 是一个多媒体的框架，从宏观上来看，它主要包含了两大方面的内容：

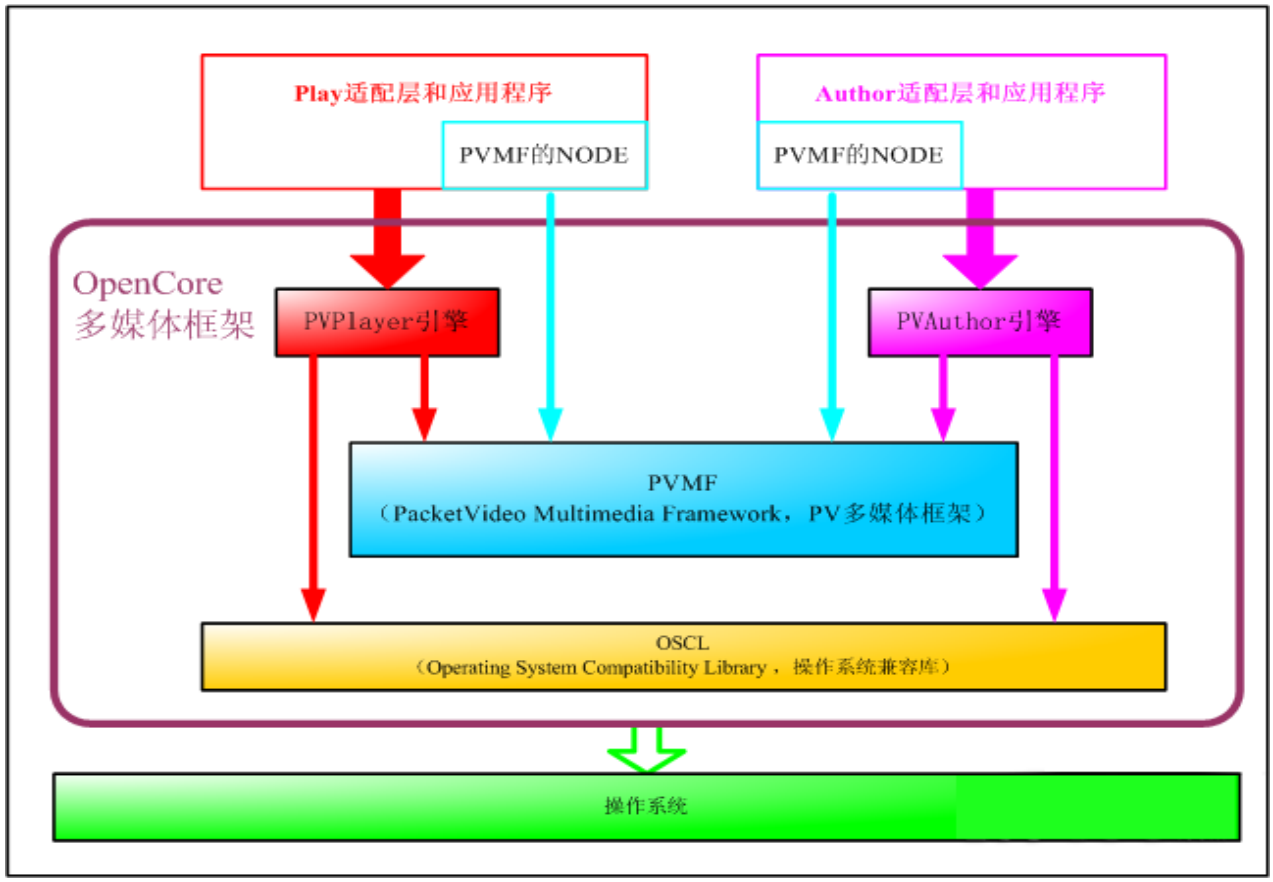
1) **PVPlayer**：提供媒体播放器的功能，完成各种音频(Audio)、视频(Video)流的回放

(Playback)功能

2) **PVAuthor**：提供媒体流记录的功能，完成各种音频(Audio)、视频(Video)流的以及静态图像捕获功能

PVPlayer 和 PVAuthor 以 SDK 的形式提供给开发者，可以在这个 SDK 之上构建多种应用程序和服务。在移动终端中常常使用的多媒体应用程序，例如媒体播放器、照相机、录像机、录音机等等。

为了更好的组织整体的架构，OpenCore 在软件层次在宏观上分成几个层次：



1) **OSCL**：Operating System Compatibility Library (操作系统兼容库)，包含了一些操作系统底层的操作，为了更好地在不同操作系统移植。包含了基本数据类型、配置、字符串工具、IO、错误处理、线程等内 容，类似一个基础的 C++库。

2) **PVMF**：PacketVideo Multimedia Framework (PV 多媒体框架)，在框架内实现一个文件解析(parser)和组成(composer)、编解码的 NODE，也可以继承其通用的接口，在用户层实现一些 NODE。

3) **PVPlayer Engine**：PVPlayer 引擎。

4) **PVAuthor Engine**：PVAuthor 引擎。

事实上，OpenCore 中包含的内容非常多：从播放的角度，PVPlayer 的输入的(Source)是文件或者网络媒体流，输出(Sink)是音频视频的输出设备，其基本功能包含了媒体流控

制、文件解析、音频视频流的解码(Decode)等方面的内容。除了从文件中播放 媒体文件之外，还包含了与网络相关的 RTSP 流(Real Time Stream Protocol, 实时流协议)。在媒体流记录的方面，PVAuthor 的输入的(Source)是照相机、麦克风等设备，输出(Sink)是各种文件， 包含了流的同步、音频视频流的编码(Encode)以及文件的写入等功能。

在使用 OpenCore 的 SDK 的时候，有可能需要在应用程序层实现一个适配器(Adaptor)，然后在适配器之上实现具体的功能，对于 PVMF 的 NODE 也可以基于通用的接口，在上层实现，以插件的形式使用。

3.3 Opencore 的代码结构

3.3.1 代码结构

以开源 Android 的代码为例，OpenCore 的代码在以下目录中：external/opencore/。

这个目录是 OpenCore 的根目录，其中包含的子目录如下所示：

1) **android**：这里面是一个上层的库，它基于 PVPlayer 和 PVAuthor 的 SDK 实现了一个为 Android 使用的 Player 和 Author。

2) **baselibs**：包含数据结构和线程安全等内容的底层库

3) **codecs_v2**：这是一个内容较多的库，主要包含编解码的实现，以及一个 OpenMAX 的实现

4) **engines**：包含 PVPlayer 和 PVAuthor 引擎的实现

5) **extern_libs_v2**：包含了 khronos 的 OpenMAX 的头文件

6) **fileformats**：文件格式的解析(parser)工具

7) **nodes**：提供一些 PVMF 的 NODE，主要是编解码和文件解析方面的。

8) **oscl**：操作系统兼容库

9) **pvmi**： 输入输出控制的抽象接口

10) **protocols**：主要是与网络相关的 RTSP、RTP、HTTP 等协议的相关内容

11) **pvcommon**：pvcommon 库文件的 Android.mk 文件，没有源文件。

12) **pvplayer**：pvplayer 库文件的 Android.mk 文件，没有源文件。

13) **pvauthor**：pvauthor 库文件的 Android.mk 文件，没有源文件。

14) **tools_v2**：编译工具以及一些可注册的模块。

在 external/opencore/目录中还有2个文件，如下所示：

```
Android.mk：全局的编译文件

pvplayer.conf：配置文件
```

在 external/opencore/的各个子文件夹中包含了众多的 Android.mk 文件，它们之间还存在着"递归"的关系。例如根目录下的 Android.mk，就包含了如下的内容片断：

```
include $(PV_TOP)/pvcommon/Android.mk

include $(PV_TOP)/pvplayer/Android.mk

include $(PV_TOP)/pvauthor/Android.mk
```

这表示了要引用 pvcommon，pvplayer 和 pvauthor 等文件夹下面的 Android.mk 文件。

external/opencore/的各个 Android.mk 文件可以按照排列组合进行使用，将几个 Android.mk 内容合并在一个库当中。

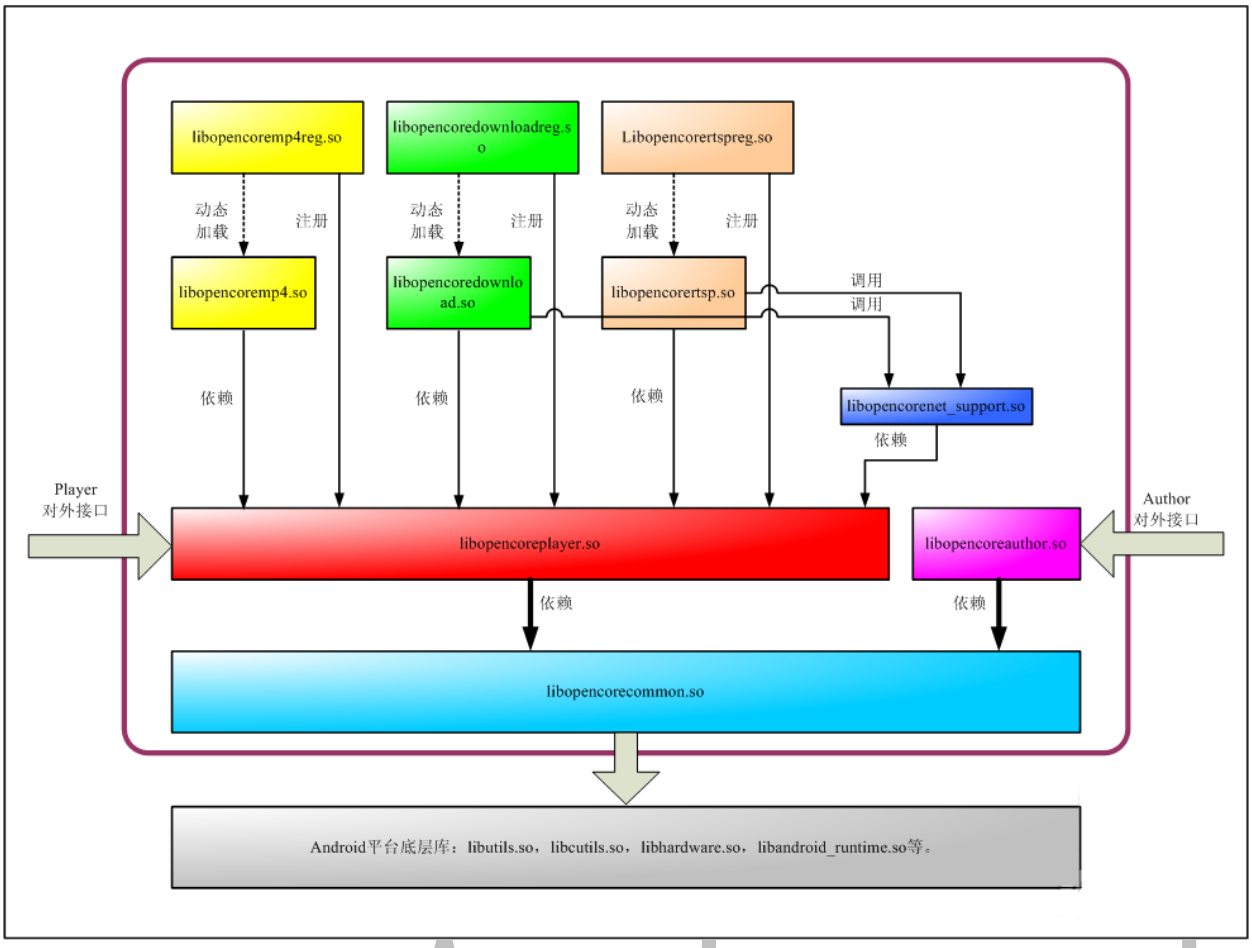
3.3.2 编译结构

库的层次关系

在 Android 的开源版本中编译出来的内容，OpenCore 编译出来的各个库如下所示：

1) libopencoreauthor.so：OpenCore 的 Author 库
2) libopencorecommon.so：OpenCore 底层的公共库
3) libopencoredownloadreg.so ：下载注册库
4) libopencoredownload.so：下载功能实现库
5) libopencoremp4reg.so：MP4 注册库
6) libopencoremp4.so：MP4 功能实现库
7) libopencorenet_support.so：网络支持库
8) libopencoreplayer.so：OpenCore 的 Player 库
9) libopencorertspreg.so：RTSP 注册库
10) libopencorertsp.so：RTSP 功能实现库

这些库的层次关系如下图所示：



OpenCore 的各个库之间具有如下的关系：

libopencorecommon.so 是所有的库的依赖库，提供了公共的功能;

libopencoreplayer.so 和 libopencoreauthor.so 是两个并立的库，分别用于回放和记录，而且这两个库是 OpenCore 对外的接口库;

libopencorenet_support.so 提供网络支持的功能;

一些功能以插件(Plug-In)的方式放入 Player 中使用，每个功能使用两个库，一个实现具体功能，一个用于注册。

3.3.3 libopencorecommon.so 库的结构

libopencorecommon.so 是整个 OpenCore 的核心库，它的编译控制的文件的路径为 pvcommon/Android.mk，这个文件使用递归的方式寻找子文件：

```
include $(BUILD_SHARED_LIBRARY)

include $(PV_TOP)//oscl/oscl/osclbase/Android.mk

include $(PV_TOP)//oscl/oscl/osclerror/Android.mk
```

```
include $(PV_TOP)//oscl/oscl/osclmemory/Android.mk

include $(PV_TOP)//oscl/oscl/osclutil/Android.mk

include $(PV_TOP)//oscl/pvlogger/Android.mk

include $(PV_TOP)//oscl/oscl/osclproc/Android.mk

include $(PV_TOP)//oscl/oscl/osclio/Android.mk

include $(PV_TOP)//oscl/oscl/osclregcli/Android.mk

include $(PV_TOP)//oscl/oscl/osclregserv/Android.mk

include $(PV_TOP)//oscl/unit_test/Android.mk

include $(PV_TOP)//oscl/oscl/osclib/Android.mk

include $(PV_TOP)//pvmi/pvmf/Android.mk

include $(PV_TOP)//baselibs/pv_mime_utils/Android.mk

include $(PV_TOP)//nodes/pvfileoutputnode/Android.mk

include $(PV_TOP)//baselibs/media_data_structures/Android.mk

include $(PV_TOP)//baselibs/threadsafe_callback_ao/Android.mk

include $(PV_TOP)//codecs_v2/utilities/colorconvert/Android.mk

include
$(PV_TOP)//codecs_v2/audio/gsm_amr/amr_nb/common/Android.mk

include $(PV_TOP)//codecs_v2/video/avc_h264/common/Android.mk
```

这些被包含的 Android.mk 文件真正指定需要编译的文件，这些文件在 Android.mk 的目录及其子目录中。事实上，在 libopencorecommon.so 库中包含了以下内容：

1) OSCL 的所有内容

2) Pvmf 框架部分的内容（pvmi/pvmf/Android.mk）

3) 基础库中的一些内容（baselibs）

4) 编解码的一些内容

5) 文件输出的 node（nodes/pvfileoutputnode/Android.mk）

从库的结构中可以看出，最终生成库的结构与 OpenCore 的层次关系并非完全重合。libopencorecommon.so 库中就包含了底层的 OSCL 的内容、PVMF 的框架以及 Node 和编

解码的工具。

3.3.4 libopencoreplayer.so 库的结构

libopencoreplayer.so 是用于播放的功能库，它的编译控制的文件的路径为 pvplayer/Android.mk，它包含了以下的内容：

```
include $(BUILD_SHARED_LIBRARY)

include $(PV_TOP)//engines/player/Android.mk

include
$(PV_TOP)//codecs_v2/audio/aac/dec/util/getactualaacconfig/
Android.mk

include
$(PV_TOP)//codecs_v2/video/avc_h264/dec/Android.mk

include $(PV_TOP)//codecs_v2/audio/aac/dec/Android.mk

include
$(PV_TOP)//codecs_v2/audio/gsm_amr/amr_nb/dec/Android.m
k

include
$(PV_TOP)//codecs_v2/audio/gsm_amr/amr_wb/dec/Android.
mk

include
$(PV_TOP)//codecs_v2/audio/gsm_amr/common/dec/Android.
mk

include $(PV_TOP)//codecs_v2/audio/mp3/dec/Android.mk

include
$(PV_TOP)//codecs_v2/utilities/m4v_config_parser/Android.mk

include
$(PV_TOP)//codecs_v2/utilities/pv_video_config_parser/Androi
d.mk

include $(PV_TOP)//codecs_v2/omx/omx_common/Android.mk

include $(PV_TOP)//codecs_v2/omx/omx_queue/Android.mk

include $(PV_TOP)//codecs_v2/omx/omx_h264/Android.mk

include $(PV_TOP)//codecs_v2/omx/omx_aac/Android.mk
```

```
include $(PV_TOP)//codecs_v2/omx/omx_amr/Android.mk

include $(PV_TOP)//codecs_v2/omx/omx_mp3/Android.mk

include
$(PV_TOP)//codecs_v2/omx/factories/omx_m4v_factory/Androi
d.mk

include $(PV_TOP)//codecs_v2/omx/omx_proxy/Android.mk

include $(PV_TOP)//nodes/common/Android.mk

include $(PV_TOP)//pvmi/content_policy_manager/Android.mk

include
$(PV_TOP)//pvmi/content_policy_manager/plugins/oma1/passt
hru/Android.mk

include
$(PV_TOP)//pvmi/content_policy_manager/plugins/common/A
ndroid.mk

include
$(PV_TOP)//pvmi/media_io/pvmiofileoutput/Android.mk

include $(PV_TOP)//fileformats/common/parser/Android.mk

include $(PV_TOP)//fileformats/id3parcom/Android.mk

include $(PV_TOP)//fileformats/rawgsmamr/parser/Android.mk

include $(PV_TOP)//fileformats/mp3/parser/Android.mk

include $(PV_TOP)//fileformats/mp4/parser/Android.mk

include $(PV_TOP)//fileformats/rawaac/parser/Android.mk

include $(PV_TOP)//fileformats/wav/parser/Android.mk

include $(PV_TOP)//nodes/pvaacffparsernode/Android.mk

include $(PV_TOP)//nodes/pvmp3ffparsernode/Android.mk

include $(PV_TOP)//nodes/pvamrffparsernode/Android.mk

include $(PV_TOP)//nodes/pvmediaoutputnode/Android.mk

include $(PV_TOP)//nodes/pvomxvideodecnode/Android.mk
```

```
include $(PV_TOP)//nodes/pvomxaudiodecnode/Android.mk

include $(PV_TOP)//nodes/pvwavffparsernode/Android.mk

include $(PV_TOP)//pvmi/recognizer/Android.mk

include
$(PV_TOP)//pvmi/recognizer/plugins/pvamrffrecognizer/Androi
d.mk

include
$(PV_TOP)//pvmi/recognizer/plugins/pvmp3ffrecognizer/Androi
d.mk

include
$(PV_TOP)//pvmi/recognizer/plugins/pvwavffrecognizer/Androi
d.mk

include $(PV_TOP)//engines/common/Android.mk

include
$(PV_TOP)//engines/adapters/player/framemetadatautility/And
roid.mk

include
$(PV_TOP)//protocols/rtp_payload_parser/util/Android.mk

include $(PV_TOP)//android/Android.mk

include $(PV_TOP)//android/drm/oma1/Android.mk

include
$(PV_TOP)//tools_v2/build/modules/linux_rtsp/core/Android.m
k

include
$(PV_TOP)//tools_v2/build/modules/linux_rtsp/node_registry/A
ndroid.mk

include
$(PV_TOP)//tools_v2/build/modules/linux_net_support/core/An
droid.mk

include
$(PV_TOP)//tools_v2/build/modules/linux_download/core/Andr
oid.mk

include
$(PV_TOP)//tools_v2/build/modules/linux_download/node_regi
```

```
stry/Android.mk

include
$(PV_TOP)//tools_v2/build/modules/linux_mp4/core/Android.m
k

include
$(PV_TOP)//tools_v2/build/modules/linux_mp4/node_registry/
Android.mk
```

libopencoreplayer.so 中包含了以下内容：

1) 一些解码工具

2) 文件的解析器（mp4）

3) 解码工具对应的 Node

4) player 的引擎部分（engines/player/Android.mk）

5) 为 Android 的 player 适配器（android/Android.mk）

6) 识别工具（pvmi/recognizer）

7) 编解码工具中的 OpenMax 部分(codecs_v2/omx)

8) 对应几个插件 Node 的注册

libopencoreplayer.so 中的内容较多，其中主要为各个文件解析器和解码器，PVPlayer 的核心功能在 engines/player/Android.mk 当中，而 android/Android.mk 的内容比较特殊，它是在 PVPlayer 之上构建的一个为 Android 使用的播放器。

3.3.5 libopencoreauthor.so 库的结构

libopencoreauthor.so 是用于媒体流记录的功能库，它的编译控制的文件的路径为 pvauthor/Android.mk，它包含了以下的内容：

```
include $(BUILD_SHARED_LIBRARY)

include $(PV_TOP)//engines/author/Android.mk

include
$(PV_TOP)//codecs_v2/video/m4v_h263/enc/Android.mk

include
$(PV_TOP)//codecs_v2/audio/gsm_amr/amr_nb/enc/Android.mk
```

```
include $(PV_TOP)//codecs_v2/video/avc_h264/enc/Android.mk

include $(PV_TOP)//fileformats/mp4/composer/Android.mk

include $(PV_TOP)//nodes/pvamrencnode/Android.mk

include $(PV_TOP)//nodes/pvmp4ffcomposernode/Android.mk

include $(PV_TOP)//nodes/pvvideoencnode/Android.mk

include $(PV_TOP)//nodes/pvavcencnode/Android.mk

include $(PV_TOP)//nodes/pvmediainputnode/Android.mk

include $(PV_TOP)//android/author/Android.mk
```

libopencoreauthor.so 中包含了以下内容：

1) 一些编码工具（视频流 H263、H264，音频流 Amr）

2) 文件的组成器（mp4）

3) 编码工具对应的 Node

4) 表示媒体输入的 Node（nodes/pvmediainputnode/Android.m）

5) author 的引擎部分（engines/author/Android.mk）

6) 为 Android 的 author 适配器（android/author/Android.mk）

libopencoreauthor.so 中主要为各个文件编码器和文件组成器，PVAuthor 的核心功能在 engines/author/Android.mk 当中，而 android/author/Android.mk 是在 PVAuthor 之上构建的一个为 Android 使用的媒体记录器。

3.3.6 其他库

另外的几个库的 Android.mk 文件的路径如下所示：

网络支持库 libopencorenet_support.so：

```
tools_v2/build/modules/linux_net_support/core/Android.m
k
```

MP4 功能实现库 libopencoremp4.so 和注册库 libopencoremp4reg.so：

```
tools_v2/build/modules/linux_mp4/core/Android.mk
```

```
tools_v2/build/modules/linux_mp4/node_registry/Android.m
k
```

RTSP 功能实现库 libopencorertsp.so 和注册库 libopencorertspreg.so：

```
tools_v2/build/modules/linux_rtsp/core/Android.mk

tools_v2/build/modules/linux_rtsp/node_registry/Android.m
k
```

下载功能实现库 libopencoredownload.so 和注册库 libopencoredownloadreg.so：

```
tools_v2/build/modules/linux_download/core/Android.mk

tools_v2/build/modules/linux_download/node_registry/Android.m
k
```

3.4 OpenCore OSCL 简介

OSCL，全称为 Operating System Compatibility Library (操作系统兼容库)，它包含了一些在不同操作系统中移植层的功能，其代码结构如下所示：

```
oscl/oscl

|-- config                    ：配置的宏

|-- makefile

|-- makefile.pv

|-- osclbase                  ：包含基本类型、
宏以及一些 STL 容器类似的功能

|-- osclerror                 ：错误处理的功能

|-- osclio                    ：文件 IO 和
Socket 等功能

|-- osscllib                  ：动态库接口等功
能

|-- osclmemory ：

内存管理、自动指针等功能

|-- osclproc                  ：线程、多任务通
讯等功能
```

```
|-- osclregcli          ：注册客户端的功
能

|-- osclregserv       ：注册

服务器的功能

`-- osclutil              ：字符串等基本功
能
```

在 oscl 的目录中，一般每一个目录表示一个模块。OSCL 对应的功能是非常细致的，几乎对 C 语言中每一个细节的功能都进行封装，并使用了 C++的接口提供给上层使用。事实上，OperCore 中的 PVMF、Engine 部分都在使用 OSCL，而整个 OperCore 的调用者也需要使用 OSCL。

在 OSCL 的实现中，很多典型的 C 语言函数被进行了简单的封装，例如：osclutil 中与数学相关的功能在 oscl_math.inl 中被定义成为了内嵌（inline）的函数：

```
OSCL_COND_EXPORT_REF
OSCL_INLINE double oscl_log(double
value)

{

        return (double) log(value);

}

OSCL_COND_EXPORT_REF
OSCL_INLINE double
oscl_log10(double value)

{

        return (double) log10(value);

}

OSCL_COND_EXPORT_REF
OSCL_INLINE double oscl_sqrt(double
value)

{

        return (double) sqrt(value);
```

oscl_math.inl 文件又被 oscl_math.h 所包含，因此其结果是 oscl_log()等功能的使用等价于原始的 log()等函数。

很多 C 语言标准库的句柄都被定义成为了 C++类的形式，实现由一些繁琐，但是复杂性都不是很高。以 oscllib 为例，其代码结构如下所示：

```
oscl/oscl/oscllib/

|-- Android.mk

|-- build

|    `-- make

|         `-- makefile

`-- src

     |-- oscl_library_common.h

     |-- oscl_library_list.cpp

     |-- oscl_library_list.h

     |-- oscl_shared_lib_interface.h

     |-- oscl_shared_library.cpp

     `-- oscl_shared_library.h
```


oscl_shared_library.h 是提供给上层使用的动态库的接口功能，它定义的接口如下所示：

```
class OsclSharedLibrary

{

public:

          OSCL_IMPORT_REF
OsclSharedLibrary();

          OSCL_IMPORT_REF
OsclSharedLibrary(const          OSCL_String&
aPath);

          OSCL_IMPORT_REF
~OsclSharedLibrary();

          OSCL_IMPORT_REF     OsclLibStatus
LoadLib(const OSCL_String& aPath);

          OSCL_IMPORT_REF     OsclLibStatus
```

```
        LoadLib();

                OSCL_IMPORT_REF                 void
        SetLibPath(const OSCL_String& aPath);

                OSCL_IMPORT_REF     OsclLibStatus
        QueryInterface(const OsclUuid& aInterfaceId,
        OsclAny*& aInterfacePtr);

                OSCL_IMPORT_REF     OsclLibStatus
        Close();

                OSCL_IMPORT_REF void AddRef();

                OSCL_IMPORT_REF                 void
        RemoveRef();

        }
```

这些接口显然都是与库的加载有关系的，而在 oscl_shared_library.cpp 中其具体的功能是使用 dlopen()等函数来实现的。

3.5 文件格式处理和编解码部分简介

在多媒体方面，文件格式的处理和编解码(Codec)是很基础的两个方面的内容。多媒体应用的两个方面是媒体的播放(PlayBack)和媒体的记录(Recording)。

在媒体的播放过程中，通常情况是从对媒体文件的播放，必要的两个步骤为文件的解析和媒体流的解码。例如对于一个 mp4 的文件，其中可能包括 AMR 和 AAC 的音频流，H263、MPEG4 以及 AVC(H264)的视频流，这些流被封装在 3GP 的包当中，媒体播放器要做的就是从文件中将这些流解析出来，然后对媒体流进行解码，解码后的数据才可以播放。

在媒体的记录过程中，通过涉及到视频、音频、图像的捕获功能。对于将视频加音频录制成文件功能，其过程与播放刚好相反，首先从硬件设备得到视频和音频的媒体流，然后对其进行编码，编码号的流还需要被分层次写入到文件之中，最终得到组成好的文件。

OpenCore 有关文件格式处理和编解码部分两部分的内容，分别在目录 fileformats 和 codecs_v2 当中。这两部分都属于基础性的功能，不涉及具体的逻辑，因此它们被别的模块调用来使用，例如：构建各种 Node。

3.5.1 文件格式的处理

由于同时涉及播放文件和记录文件两种功能，因此 OpenCore 中的文件格式处理有两种类型，一种是 parser(解析器)，另一种是 composer(组成器)。

fileformats 的目录结构如下所示：fileformats

```
|-- avi
|   `-- parser
|-- common
|   `-- parser
|-- id3parcom
|   |-- Android.mk
|   |-- build
|   |-- include
|   `-- src
|-- mp3
|   `-- parser
|-- mp4
|   |-- composer
|   `-- parser
|-- rawaac
|   `-- parser
|-- rawgsmamr
|   `-- parser
`-- wav
    `-- parser
```

目录包含各个子目录中，它们对应的是不同的文件格式，例如 mp3、mp4 和 wav 等。

3.5.2 编解码

编解码部分主要针对 Audio 和 Video，codecs_v2 的目录结构如下所示：

```
|-- avi
|    `-- parser
|-- common
|    `-- parser
|-- id3parcom
|    |-- Android.mk
|    |-- build
|    |-- include
|    `-- src
|-- mp3
|    `-- parser
|-- mp4
|    |-- composer
|    `-- parser
|-- rawaac
|    `-- parser
|-- rawgsmamr
|    `-- parser
`-- wav
     `-- parser
```

在 audio 和 video 目录中，对应了针对各种流的子目录，其中可能包含 dec 和 enc 两个目录，分别对应解码和编码。video 目录展开后的内容如下所示：

```
-- video
     |-- avc_h264
     |    |-- common
```

```
|     |-- dec
|     |-- enc
|     `-- patent_disclaimer.txt
`-- m4v_h263
      |-- dec
      |-- enc
      `-- patent_disclaimer.txt
```

codecs_v2 目录的子目录 omx 实现了一个 khronos

OpenMAX 的功能。OpenMAX 是一个多媒体应用程序的框架标准，由 NVIDIA 公司和 Khronos 在 2006 年推出。OpenMAX IL 1.0（集成层）技术规格定义了媒体组件接口，以便在嵌入式器件的流媒体框架中快速集成加速式编解码器。

OpenMAX 的设计实现可以让具有硬件编辑码功能的平台提供统一的接口和框架，在 OpenMAX 中可以直接使用硬件加速的进行编解码乃至输出的功能，对外保持统一的接口。但是在此处的 OpenMAX 则是一个纯软件的实现。

【Android 音频视频 实例教程】

4.1 音频视频编解码格式

4.1.1 音频解码格式

*MPEG Audio Layer 1/2
*MPEG Audio Layer 3(MP3)
*MPEG2 AAC
*MPEG4 AAC
*Windows Media audeo v1/v2/7/8/9
*RealAudio cook/sipro(real media series)
*RealAudio AAC/AACPlus(real media series)
*QDesign Music 2(apple series)
*Apple MPEG-4 AAC(apple series)
*ogg(ogg vorbis 音频)
*AC3(DVD 专用音频编码)
*DTS(DVD 专用音频编码)
*APE(monkey’s 音频)
*AU(sun 格式)
*FLAC(fress lossless 音频)
*M4A(mpeg-4 音频)（苹果改用的名字，可以改成.mp4）
*MP2(mpeg audio layer2 音频)
*MWA

4.1.2 视频编解码格式

*MPEG1(VCD)
*MPEG2(DVD)
*MPEG4(divx,xvid)
*MPEG4 AVC/h.264
*h.261
*h.262
*h.263
*h.263+
*h.263++
*MPEG-4 v1/v2/v3(微软 windows media 系列)
*Windows Media Video 7/8/9/10
*Sorenson Video 3（用于 QT5，成标准了）(apple series)
*RealVideo G2(real media series)
*RealVideo 8/9/10(real media series)
*Apple MPEG-4(apple series)
*Apple H.264(apple series)
*flash video

4.1.3 音、视频文件格式

*说明：首先要分清楚媒体文件和编码的区别：文件是既包括视频又包括音频、甚至还带有脚本的一个集合，也可以叫容器；文件当中的视频和音频的压缩算法才是具体的编码。

*AVI
音视频交互存储，最常见的音频视频容器。支持的视频音频编码也是最多的

*MPG
MPEG 编码采用的音频视频容器，具有流的特性。里面又分为 PS，TS 等，PS 主要用于 DVD 存储，TS 主要用于 HDTV。

*VOB
DVD 采用的音频视频容器格式（即视频 MPEG-2，音频用 AC3 或者 DTS），支持多视频多音轨多字幕章节等。

*MP4
MPEG-4 编码采用的音频视频容器，基于 QuickTime MOV 开发，具有许多先进特性。

*3GP
3GPP 视频采用的格式，主要用于流媒体传送。

*ASF
Windows Media 采用的音频视频容器，能够用于流传送，还能包容脚本等。

*RM
RealMedia 采用的音频视频容器，用于流传送。

*MOV
QuickTime 的音频视频容器，恐怕也是现今最强大的容器，甚至支持虚拟现实技术，Java 等，它的变种 MP4,3GP 都没有这么厉害。

*MKV
MKV 它能把 Windows Media Video，RealVideo，MPEG-4 等视频音频融为一个文件，而且支持多音轨，支持章节字幕等。

*WAV
一种音频容器（注意：只是音频），大家常说的 WAV 就是没有压缩的 PCM 编码，其实 WAV 里面还可以包括 MP3 等其他 ACM 压缩编码。
音、视频技术

Audio CD
*标准 CD 格式也就是 44.1K 的采样频率，速率 88K/秒，16 位量化位数
**.cda 格式，这就是 CD 音轨了，一个 CD 音频文件是一个*.cda 文件，这只是一个索引信息，并不是真正的包含声音信息，所以不论 CD 音乐的长短，在电脑上看到的“*.cda 文件”都是 44 字节长

MP3
*MPEG 音频文件的压缩是一种有损压缩，MPEG3 音频编码具有 10：1~12：1 的高压缩率，同时基本保持低音频部分不失真，但是牺牲了声音文件中 12KHz 到 16KHz 高音频这部分

的质量来换取文件的尺寸，相同长度的音乐文件，用＊.mp3 格式来储存，一般只有＊.wav 文件的 1/10，而音质要 次于 CD 格式或 WAV 格式的声音文件

*MP3 格式压缩音乐的采样频率有很多种，可以用 64Kbps 或更低的采样频率节省空间，也可以用 320Kbps 的标准达到极高的音质

*每分钟音乐的 MP3 格式只有 1MB 左右大小

MIDI： 经常玩音乐的人应该常听到 MIDI（Musical Instrument Digital Interface）这个词，MIDI 允许数字合成器和其他设备交换数据。MID 文件格式由 MIDI 继承而来。MID 文件并不是一段录制好的声音，而是记 录声音的信息，然后在告诉声卡如何再现音乐的一组指令。这样一个 MIDI 文件每存 1 分钟的音乐只用大约 5～10KB。今天，MID 文件主要用于原始乐器作 品，流行歌曲的业余表演，游戏音轨以及电子贺卡等。＊.mid 文件重放的效果完全依赖声卡的档次。＊.mid 格式的最大用处是在电脑作曲领域。＊.mid 文件可以用作曲软件写出，也可以通过声卡的 MIDI 口把外接音序器演奏的乐曲输入电脑里，制成＊.mid 文件。

WMA:
*WMA 的压缩率一般都可以达到 1：18 左右，WMA 的另一个优点是内容提供商可以通过 DRM（Digital Rights Management）方案如 Windows Media Rights Manager 7 加入防拷贝保护。这种内置了版权保护技术可以限制播放时间和播放次数甚至于播放的机器等等，这对被盗版搅得焦头乱额的音乐公司来说可是一个福音，另外 WMA 还支持音频流(Stream)技术，适合在网络上在线播放

* WMA 这种格式在录制时可以对音质进行调节。同一格式，音质好的可与 CD 媲美，压缩率较高的可用于网络广播
以文件名标识识别音频编码格式
*.aac
音频编码：aac
*.ac3
音频编码：ac3
*.ape
*.au
音频编码：pcm_s16be
*.m4a
音频编码：mpeg4 aac
*.mp2
*.mp3
*.ogg
音频编码：vorbis
*.wav
音频编码：pcm_s16le
*.flav
*.wma
音频编码：wma7x

以文件名标识识别音频编码格式 ：

1．*.MP4 (MP4 MPEG-4 视频)
视频编码：mpeg4
音频编码：mpeg4 aac

2．*.3gp (3GPP 第三代合作项目)
视频编码：mpeg4
音频编码：amr_nb((mono, 8000 Hz, Sample Depth 16 bit, bitrate 12 kbps)

3．*.3g2 (3GPP 第三代合作项目 2)
视频编码：mpeg4
音频编码：mpeg4 aac
4．*.asf (ASF 高级流格式)
视频编码：msmpeg4
音频编码：mp3

5．*.avi （AVI 音视频交错格式）
视频编码：mpeg4
音频编码：pcm_s161e

6．*.avi （divx 影片）
视频编码：mpeg4
音频编码：mp3

7．*.avi（xvid 视频）
视频编码：Xvid
音频编码：mp3

8．*.vob （DVD）
视频编码：mpeg2 video
音频编码：ac3

9．*.flv (flash 视频格式)
视频编码：
音频编码：mp3

10．*.mp4 (iPod 320*240 MPEG-4 视频格式)
视频编码：mpeg4
音频编码：mpeg4 aac

11．*.mp4(iPod video2 640*480 MPEG-4 视频格式)
视频编码：mpeg4
音频编码：mpeg4 aac

12．*.mov (MOV 苹果 quicktime 格式)
视频编码：mpeg4_qt
音频编码：mpeg4 aac_qt

13．*.mpg （mpeg1 影片）

视频编码：mpeg1 video
音频编码：mp2

14．*.mpg （mpeg2 影片）
视频编码：mpeg2 video
音频编码：mp2

15．*.mp4 （mpeg4 avc 视频格式）
视频编码：h.264
音频编码：mpeg4 aac

16．*.mp4 （PSP mpeg4 影片）
视频编码：Xvid
音频编码：mpeg4 aac

17．*.mp4 （PSP AVC 视频格式）
视频编码：h.264
音频编码：mpeg4 aac

18．*.rm （RM realvideo）
视频编码：rv10
音频编码：ac3

19．*.mpg (超级 VCD)
视频编码：mpeg2 video
音频编码：mp2

20．*.swf (SWF 格式)
视频编码：
音频编码：mp3

21．*.mpg （video CD 格式）
视频编码：mpeg1 video
音频编码：mp2

22．*.vob （mpeg2 ps 格式）
视频编码：mpeg2 video
音频编码：ac3

23．*.wmv（windows 视频格式）
视频编码：wmv3x
音频编码：wma7x

4.2 Android 视录视频示例

Android 视频通话功能，从最简单的视频录制开始, Activity 类:VideoActivity

```
package com.media.Media;


import java.io.File;

import java.io.IOException;


import android.app.Activity;

import android.media.MediaRecorder;

import android.os.Bundle;

import android.os.Environment;

import android.view.SurfaceHolder;

import android.view.SurfaceView;

import android.view.View;

import android.view.View.OnClickListener;

import android.widget.Button;


public class VideoActivity extends Activity {



    private File myRecAudioFile;

    private SurfaceView mSurfaceView;

    private SurfaceHolder mSurfaceHolder;

    private Button buttonStart;

    private Button buttonStop;

    private File dir;

    private MediaRecorder recorder;
```

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.video);
        mSurfaceView = (SurfaceView) findViewById(R.id.videoView);
        mSurfaceHolder = mSurfaceView.getHolder();

mSurfaceHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
        buttonStart=(Button)findViewById(R.id.start);
        buttonStop=(Button)findViewById(R.id.stop);
        File defaultDir = Environment.getExternalStorageDirectory();
        String path =
defaultDir.getAbsolutePath()+File.separator+"V"+File.separator;//创
建文件夹存放视频
        dir = new File(path);
        if(!dir.exists()){
            dir.mkdir();
        }
        recorder = new MediaRecorder();

        buttonStart.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                recorder();
```

```
            }
        });

        buttonStop.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                recorder.stop();
                recorder.reset();
                recorder.release();
                recorder=null;
            }
        });
    }

    public void recorder() {
        try {
            myRecAudioFile = File.createTempFile("video",
".3gp",dir);//创建临时文件

recorder.setPreviewDisplay(mSurfaceHolder.getSurface());//预览

recorder.setVideoSource(MediaRecorder.VideoSource.CAMERA);//视频源
            recorder.setAudioSource(MediaRecorder.AudioSource.MIC);
//录音源为麦克风

recorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);//输
```

```
出格式为 3gp

            recorder.setVideoSize(800, 480);//视频尺寸

            recorder.setVideoFrameRate(15);//视频帧频率


recorder.setVideoEncoder(MediaRecorder.VideoEncoder.H263);//视频编码


recorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);//音频编
码

            recorder.setMaxDuration(10000);//最大期限


recorder.setOutputFile(myRecAudioFile.getAbsolutePath());//保存路径

            recorder.prepare();

            recorder.start();

        } catch (IOException e) {

            e.printStackTrace();

        }

    }
```

界面:video.xml

```
    <?xml version="1.0" encoding="utf-8"?>
    <LinearLayout

xmlns:android="http://schemas.android.com/apk/res/android"
            android:orientation="vertical"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent">
            <SurfaceView android:id="@+id/videoView"
                android:visibility="visible"
                android:layout_width="320px"
                android:layout_height="240px">
            </SurfaceView>

            <RelativeLayout
                android:layout_width="fill_parent"
                android:layout_height="wrap_content">
                <Button
```

```
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="录制"
                android:id="@+id/start"/>
            <Button
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_toRightOf="@id/start"
                android:text="停止"
                android:id="@+id/stop"/>
        </RelativeLayout>
    </LinearLayout>
```

权限配置:AndroidManifest.xml

```
    <?xml version="1.0" encoding="utf-8"?>
    <manifest
xmlns:android="http://schemas.android.com/apk/res/android"
            package="com.media.Media"
            android:versionCode="1"
            android:versionName="1.0">
        <application                android:icon="@drawable/rabbit"
android:label="@string/app_name">
            <activity android:name=".VideoActivity"
                            android:label="@string/app_name">
                <intent-filter>
                    <action
android:name="android.intent.action.MAIN" />
                    <category
android:name="android.intent.category.LAUNCHER" />
                </intent-filter>
            </activity>
        </application>

        <uses-sdk android:minSdkVersion="7" />
        <uses-permission
android:name="android.permission.CAMERA"/>
        <uses-permission
android:name="android.permission.RECORD_AUDIO"/>
        <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    </manifest>
```

4.3 利用 ffmpeg 框架扩展 android 平台解码器

Andorid 平台本身的多媒体框架 opencore 支持音频格式有：3GPP (.3gp)、MPEG-4 (.mp4, .m4a)、mp3、Type 0 and 1 (.mid, .xmf, .mxmf). Also RTTTL/RTX (.rtttl, .rtx), OTA (.ota), and iMelody (.imy)、Ogg (.ogg) （普遍用的比较多的一种） 、WAVE (.wav)，支持的视频

格式只有两种：3gp、MP4两种；很显然支持的视频格式太少，如果我们要做一个通用的多媒体播放器的话是远远不够的。像 rm,rmvb,avi 等这些格式都是最常见的视频格式。我们可以通过开源框架 ffmpeg 通过 ndk 框架的 jni 的编程方式来实现以上三种视频的解码。FFmpeg 是一个集录制、转换、音/视频编码解码功能为一体的完整的开源解决方案。FFmpeg 的开发是基于 Linux 操作系统，但是可以在大多数操作系统中编译和使用。FFmpeg 支持 MPEG、DivX、MPEG4、AC3、DV、FLV 等40多种编码，AVI、MPEG、OGG、Matroska、ASF 等90多种解码.TCPMP, VLC, MPlayer 等开源播放器都用到了 FFmpeg。下面开始我们的学习：

第一步：我们要下载 ffmpeg 源码并利用 andorid ndk 框架对 ffmpeg 编译为 ffmpeg.so 类库。源码下载地址大家可以查看 http://www.ffmpeg.org/download.html 有 git 和 svn 等几种下载方式。推荐大家用 git 方式下载，svn （svn://svn.ffmpeg.org/ffmpeg/trunk）方式版本已经截止到**2011-01-19**已经不再更新。Git 下载地址（git://git.videolan.org/ffmpeg.git）现在已经更新至2011-04-27 。编译 ffmpeg 参考文章 ：www.cnblogs.com/scottwong/archive/2010/12/17/1909455.html 编译完成后会生成一个 libffmpeg.so 文件该动态链接库文件就是我们后边要用到的 ffmpeg 框架；这个 so 里的是 jni 方法，可以由 java 层调用的，而这些 jni 方法里用到的函数则就是来至 libffmpeg.so

第二步：利用 andorid 提供的 ndk 框架 编写自己用于解码的 c 文件具体操作步骤如下。拿一个标准的 ndk 例子来做的测试就是 ndk samples 文件夹里的 hello-jni 工程。进入该工程的 jni 目录，将 ffmpeg 的源代码拷到该目录下，做这部的原因是你要编译的 so 文件里需 要调用 ffmpeg 的方法，自然要引用 ffmpeg 里的 h 文件，然后将 libffmpeg.so 文件拷到 ndk 目录下的 platforms/android-5/arch-arm/usr/lib 目录下因为等等系统编译的时候要用。接下来就编辑 android.mk 和 hello-jni.c 文件

android.mk

```
# Copyright (C) 2009 The Android Open Source Project
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
# http://www.apache.org/licenses/LICENSE-2.0
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
PATH_TO_FFMPEG_SOURCE:=$(LOCAL_PATH)/ffmpeg
LOCAL_C_INCLUDES += $(PATH_TO_FFMPEG_SOURCE)
LOCAL_LDLIBS := -lffmpeg
LOCAL_MODULE        := hello-jni
LOCAL_SRC_FILES := hello-jni.c
include $(BUILD_SHARED_LIBRARY)
PATH_TO_FFMPEG_SOURCE:=$(LOCAL_PATH)/ffmpeg
```

这行是定义一个变量，也就是 ffmpeg 源码的路径

```
LOCAL_C_INCLUDES += $(PATH_TO_FFMPEG_SOURCE)
```

这行是指定源代码的路径，也就是刚才拷过去的 ffmpeg 源码，$(LOCAL_PATH)是根目录，如果没有加这行那么引入 ffmpeg 库中的 h 文件编译就会出错说找不到该 h 文件。

LOCAL_LDLIBS := -lffmpeg

这行很重要，这是表示你这个 so 运行的时候依赖于 libffmpeg.so 这个库， 再举个例子：如果你要编译的 so 不仅要用到 libffmpeg.so 这个库还要用的 libopencv.so 这个库的话，你这个参数就应该写成

LOCAL_LDLIBS := -lffmpeg -lopencv

hello-jni.c

```
/*
 * Copyright (C) 2009 The Android Open Source Project
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 *
 */
#include <string.h>
#include <stdio.h>
#include <android/log.h>
#include <stdlib.h>
#include <jni.h>
#include <ffmpeg/libavcodec/avcodec.h>
/* This is a trivial JNI example where we use a native method
 * to return a new VM String. See the corresponding Java source
 * file located at:
 *
 *   apps/samples/hello-jni/project/src/com/example/HelloJni/HelloJni.java
 */
jstring
Java_com_example_hellojni_HelloJni_stringFromJNI( JNIEnv* env,
                                                  jobject thiz )
{
    char str[25];
    sprintf(str, "%d", avcodec_version());

    return (*env)->NewStringUTF(env, str);
}
```

#include <ffmpeg/libavcodec/avcodec.h>

这行是因为下面要用到 avcodec_version()这个函数。

改完这两个文件以后就可以编译了~~用 ndk-build 命令编译完后在工程的 libs/armeabi 目录底下就会有一个 libhello-jni.so 文件了！（两行眼泪啊~终于编译成功了）

编译完成后就可以进行测试了，记得将 libffmpeg.so 也拷到 armeabi 目录底下，并在 java 代码中写上

```
static {
        System.loadLibrary("ffmpeg");
         System.loadLibrary("hello-jni");
    }
```

HelloJni.java

```
package com.example.hellojni;
import android.app.Activity;
import android.widget.TextView;
import android.os.Bundle;

public class HelloJni extends Activity
{
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);

        /* Create a TextView and set its content.
         * the text is retrieved by calling a native
         * function.
         */
        TextView  tv = new TextView(this);
        tv.setText( "1111" );
        //System.out.println();
        setContentView(tv);
        tv.setText(String.valueOf(stringFromJNI()));
    }

    /* A native method that is implemented by the
     * 'hello-jni' native library, which is packaged
     * with this application.
     */
    public native String  stringFromJNI();

    /* This is another native method declaration that is *not*
     * implemented by 'hello-jni'. This is simply to show that
     * you can declare as many native methods in your Java code
     * as you want, their implementation is searched in the
     * currently loaded native libraries only the first time
     * you call them.
     * Trying to call this function will result in a
     * java.lang.UnsatisfiedLinkError exception !
     */
    public native String  unimplementedStringFromJNI();
    /* this is used to load the 'hello-jni' library on application
     * startup. The library has already been unpacked into
     * /data/data/com.example.HelloJni/lib/libhello-jni.so at
     * installation time by the package manager.
     */
    static {
          System.loadLibrary("ffmpeg");
         System.loadLibrary("hello-jni");
    }
```

到此就完成了，将程序装到手机可看到打印出“3426306”，google 搜索“ffmpeg 3426306”得知果然是 ffmpeg 的东西，证明成功的调用了 libffmpeg.so 库里的方法了。

本例子只是一个测试的 ffmpeg 框架的测试 demo ，以后大家要扩展自己的解码器必须查看与 ffmpeg 的开发文档。在后续的文章中会提供某几种格式的代码实例。

【其他】

5.1 提交BUG

如果你发现文档中有不妥的地方，请发邮件至eoeandroid@eoemobile.com 进行反馈，我们会定期更新、发布更新后的版本。

5.2 关于eoeAndroid

eoeAndroid 是国内成立最早，规模最大的Android 开发者社区，拥有海量的Android 学习资料。分享、互助的氛围，让Android 开发者迅速成长，逐步从懵懂到了解，从入门到开发出属于自己的应用，eoeAndroid 为广大Android 开发者奠定坚实的技术基础。从初级到高级，从环境搭建到底层架构，eoeAndroid 社区为开发者精挑细选了丰富的学习资料和实例代码。让Android 开发者在社区中迅速的成长，并在此基础上开发出更多优秀的Android 应用。

北京易联致远无限技术有限公司

责任编辑： QUMIN

美术支持： 阿彦　　**技术支持：** gaotong86

中国最大的Android 开发者社区：www.eoeandroid.com

中国本土的Android 软件下载平台：www.eoemarket.com