Audio系统详解

目录

[Audio系统详解 1](#_Toc366832886)

[1 Audio的Framework层功能及用法大全 2](#_Toc366832887)

[1.1 多媒体的framework接口预览 2](#_Toc366832888)

[1.2 Audio相关的文件预览 5](#_Toc366832889)

[1.3 Audio核心 6](#_Toc366832890)

[1.3.1 AudioTrack 6](#_Toc366832891)

[1.3.2 AudioRecord 7](#_Toc366832892)

[1.3.3 AudioFormat 8](#_Toc366832893)

[1.3.4 AudioSystem 8](#_Toc366832894)

[1.4 Audio辅助 12](#_Toc366832895)

[1.4.1 AudioManager和AudioService 12](#_Toc366832896)

[1.4.2 RingToneManager和RingTone 12](#_Toc366832897)

[1.4.3 ToneGenerator 12](#_Toc366832898)

[1.4.4 SoundPool 13](#_Toc366832899)

[1.5 Audio路由 13](#_Toc366832900)

[1.5.1 MediaRounter 13](#_Toc366832901)

[1.6 Audio Framework层总结 13](#_Toc366832902)

[2 Audio系统的整体结构 13](#_Toc366832903)

[3 Audio系统的Native层 15](#_Toc366832904)

[3.1 Audio框架libmedia 16](#_Toc366832905)

[3.2 AudioFlinger 16](#_Toc366832906)

[3.3 AudioPolicy 16](#_Toc366832907)

[3.3.1 代码位置 16](#_Toc366832908)

[3.3.2 AudioPolicy的结构图 17](#_Toc366832909)

[3.3.3 AudioPolicyManagerBase 17](#_Toc366832910)

[3.3.4 AudioPolicyManagerBase的职责 17](#_Toc366832911)

[3.3.5 AudioPolicyManagerBase中的主要方法 18](#_Toc366832912)

[4 Audio系统的Hal层 20](#_Toc366832913)

[4.1 代码位置 20](#_Toc366832914)

# Audio的Framework层功能及用法大全

## 多媒体的framework接口预览

|  |  |  |  |
| --- | --- | --- | --- |
| **类型** | **文件名** | **描述** |  |
| 多媒体核心 | MediaPlayer.java |  |  |
| MediaRecorder.java |  |  |
| 应用层控制Extractor，解码 | DecoderCapabilities.java | 用来获得Android平台支持的解码器。 |  |
| EncoderCapabilities.java | 用来获得Android平台支持的编码器。 |  |
| MediaCodecInfo.java |  | Android4.1 |
| MediaCodec.java |  | Android4.1 |
| MediaCodecList.java |  | Android4.1 |
| MediaFormat.java |  | Android4.1 |
| MediaExtractor.java |  | Android4.1 |
| Utils | MediaCrypto.java | 为MediaCodec提供支持 | Android4.1 |
| MediaCryptoException.java |  | Android4.1 |
| MediaSyncEvent.java |  | Android4.1 |
| Audio核心 | AudioRecord.java |  |  |
| AudioTrack.java |  |  |
| AudioSystem.java |  |  |
| AudioFormat.java |  |  |
| 声音路由相关 | AudioRoutesInfo.java | 用来设置声音路由，现在用户可以自由设置AudioPolicy中的功能了。 | Android4.1 |
| MediaRouter.java |  | Android4.1 |
| Audio辅助及控制 | AsyncPlayer.java | 声音异步加载 |  |
| AudioManager.java | 管理音量、铃声 |  |
| AudioService.java | 声音管理服务， AudioManager中调用的是此中的实现 |  |
| RingtoneManager.java |  |  |
| Ringtone.java |  |  |
| ToneGenerator.java |  |  |
| SoundPool.java |  |  |
| MediaActionSound.java |  | Android4.1 |
| Camera | CamcorderProfile.java |  |  |
| CameraProfile.java |  |  |
| 扫描和Provider | MediaFile.java |  |  |
| MediaScannerClient.java |  |  |
| MediaScannerConnection.java |  |  |
| MediaScanner.java |  |  |
| MediaInserter.java |  |  |
| Metadata.java |  |  |
| TimedText.java |  |  |
| MediaMetadataRetriever.java |  |  |
| MiniThumbFile.java |  |  |
| ThumbnailUtils.java |  |  |
| 其它 | ExifInterface.java | 用来获得Jpeg图像的Exif信息 |  |
| FaceDetector.java | 人脸识别，有没有实现就不知道了,需要再研究。 |  |
| JetPlayer.java |  |  |
| RemoteControlClient.java |  |  |
| AmrInputStream.java |  |  |
| ResampleInputStream.java |  |  |

说明：在frameworks/base/media/java/android/media/ 路径下总共有57个文件：

文件夹: 2个

aidl文件: 10个

html文件: 1个

以上有说明的文件：44个, 相对Android4.0.3增加了11个。

下面是两个文件夹下的具体内容：

|  |  |
| --- | --- |
| VideoEditor | |-- AudioTrack.java |
| |-- EffectColor.java |
| |-- Effect.java |
| |-- EffectKenBurns.java |
| |-- ExtractAudioWaveformProgressListener.java |
| |-- MediaArtistNativeHelper.java |
| |-- MediaImageItem.java |
| |-- MediaItem.java |
| |-- MediaProperties.java |
| |-- MediaVideoItem.java |
| |-- OverlayFrame.java |
| |-- Overlay.java |
| |-- TransitionAlpha.java |
| |-- TransitionCrossfade.java |
| |-- TransitionFadeBlack.java |
| |-- Transition.java |
| |-- TransitionSliding.java |
| |-- VideoEditorFactory.java |
| |-- VideoEditorImpl.java |
| |-- VideoEditor.java |
| |-- VideoEditorProfile.java |
| `-- WaveformData.java |
| AudioFx | |-- AcousticEchoCanceler.java |
| |-- AudioEffect.java |
| |-- AutomaticGainControl.java |
| |-- BassBoost.java |
| |-- EnvironmentalReverb.java |
| |-- Equalizer.java |
| |-- NoiseSuppressor.java |
| |-- package.html |
| |-- PresetReverb.java |
| |-- Virtualizer.java |
| `-- Visualizer.java |

## Audio相关的文件预览

在整个Framework的Media部分与Audio相关的部分如下：

|  |  |  |  |
| --- | --- | --- | --- |
| **类型** | **文件名** | **描述** |  |
| Audio核心 | AudioRecord.java |  |  |
| AudioTrack.java |  |  |
| AudioSystem.java |  |  |
| AudioFormat.java |  |  |
| 声音路由相关 | AudioRoutesInfo.java | 用来设置声音路由，现在用户可以自由设置AudioPolicy中的功能了。 | Android4.1 |
| MediaRouter.java |  | Android4.1 |
| Audio辅助及控制 | AsyncPlayer.java | 声音异步加载 |  |
| AudioManager.java | 管理音量、铃声 |  |
| AudioService.java | 声音管理服务， AudioManager中调用的是此中的实现 |  |
| RingtoneManager.java |  |  |
| Ringtone.java |  |  |
| ToneGenerator.java |  |  |
| SoundPool.java |  |  |
| MediaActionSound.java |  | Android4.1 |

由上图可知：Framework层主要分为两个部分，Audio的核心层和Audio的辅助类。Audio核心层的类包括AudioTrack, AudioRecord, AudioSystem。Audio的辅助类主要有AudioManager，RingToneManager，ToneGenerator和SoundPool。

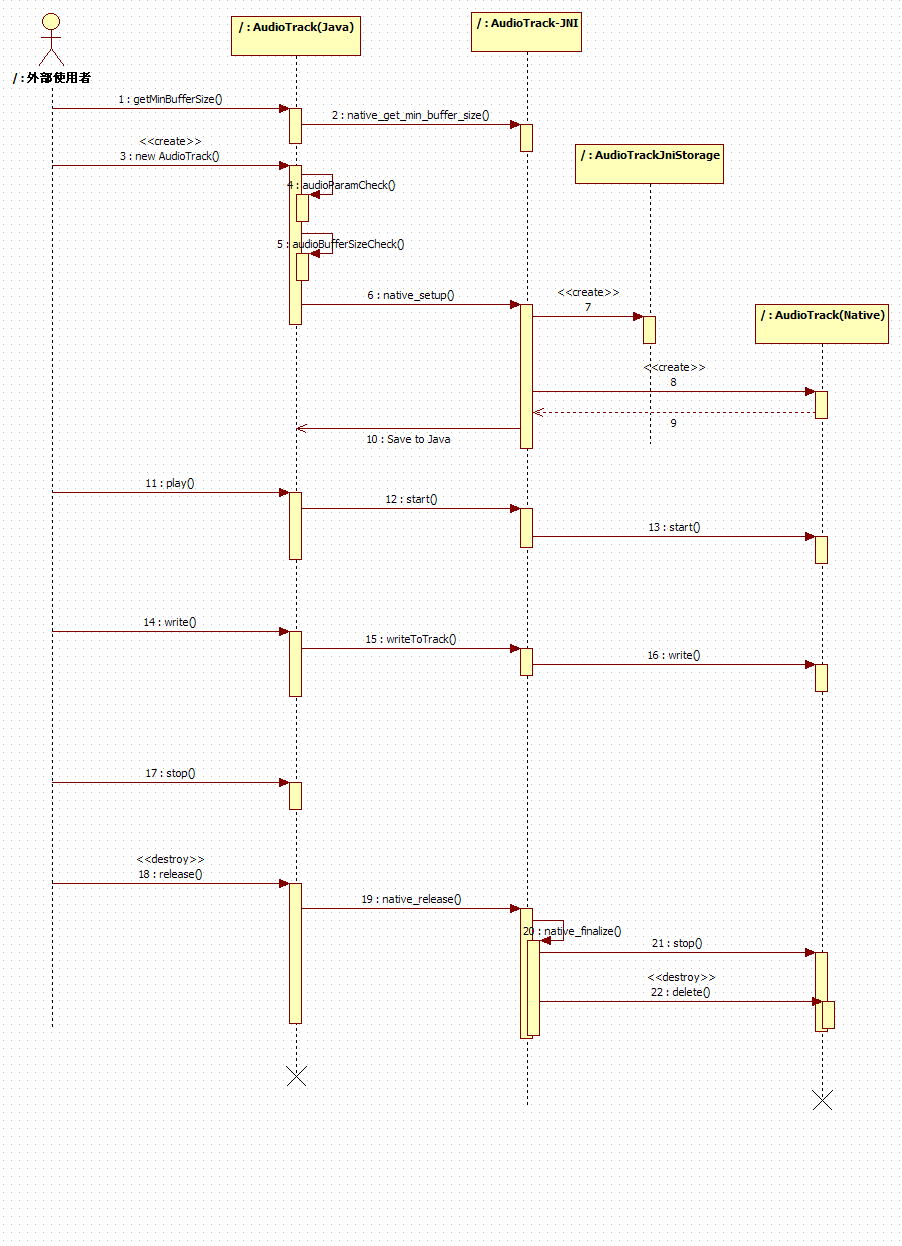
下面的章节节将分别介绍这些部分。

## Audio核心

### AudioTrack

AudioTrack是播放的API接口，具体用法是先new一个实例出来，然后再调用它里边的一个play, write, read等函数， 最后stop, release.

下图是一个建立及销毁的全过程：



### AudioRecord

AudioRecord是录音的API，提供接口给用户调用，并且含有大量的native函数，需要JNI调用Native的相关功能。

### AudioFormat

AudioFormat中定义了一些变量，这些变量在AudioTrack和AudioRecord中用到。主要有以下内容：

#### Audio data format

/\*\* Invalid audio data format \*/

public static final int ENCODING\_INVALID = 0;

/\*\* Default audio data format \*/

public static final int ENCODING\_DEFAULT = 1;

// These two values must be kept in sync with JNI code for AudioTrack, AudioRecord

/\*\* Audio data format: PCM 16 bit per sample. Guaranteed to be supported by devices. \*/

public static final int ENCODING\_PCM\_16BIT = 2;

/\*\* Audio data format: PCM 8 bit per sample. Not guaranteed to be supported by devices. \*/

public static final int ENCODING\_PCM\_8BIT = 3;

#### Chanel mask

包括chanel in 和 chanel out的很多掩码定义

### AudioSystem

此类包含了一些常量定义，设备定义，native的方法调用接口等。

以下是代码中的说明:

/\* These values must be kept in sync with AudioSystem.h \*/

/\*

\* If these are modified, please also update Settings.System.VOLUME\_SETTINGS

\* and attrs.xml and AudioManager.java.

\*/

/\* The audio stream for phone calls \*/

#### Audio stream相关

public static final int STREAM\_VOICE\_CALL = 0;

public static final int STREAM\_SYSTEM = 1;

public static final int STREAM\_RING = 2;

public static final int STREAM\_MUSIC = 3;

public static final int STREAM\_ALARM = 4;

public static final int STREAM\_NOTIFICATION = 5;

public static final int STREAM\_BLUETOOTH\_SCO = 6;

public static final int STREAM\_SYSTEM\_ENFORCED = 7;

public static final int STREAM\_DTMF = 8;

public static final int STREAM\_TTS = 9;

目前一共10种Type, 从0-9, 并且用以下的函数取得StreameType的Count.

// Expose only the getter method publicly so we can change it in the future

private static final int NUM\_STREAM\_TYPES = 10;

public static final int getNumStreamTypes() { return NUM\_STREAM\_TYPES; }

Audio Stream是一个重要的类型，在其它的很多函数中都需要用到它， 如下所示：

public static native boolean isStreamActive(int stream, int inPastMs);

public static native int initStreamVolume(int stream, int indexMin, int indexMax);

public static native int setStreamVolumeIndex(int stream, int index, int device);

public static native int getStreamVolumeIndex(int stream, int device);

public static native int getDevicesForStream(int stream);

#### Microphone是否静音

public static native int muteMicrophone(boolean on);

public static native boolean isMicrophoneMuted();

返回值是 AUDIO\_STATUS\_OK 或 AUDIO\_STATUS\_ERROR

#### setPhoneState()

public static native int setPhoneState(int state);

// phone state, match audio\_mode???

public static final int PHONE\_STATE\_OFFCALL = 0;

public static final int PHONE\_STATE\_RINGING = 1;

public static final int PHONE\_STATE\_INCALL = 2;

上边的被淘汰了，用下边的定义：

/\* modes for setPhoneState, must match AudioSystem.h audio\_mode \*/

public static final int MODE\_INVALID = -2;

public static final int MODE\_CURRENT = -1;

public static final int MODE\_NORMAL = 0;

public static final int MODE\_RINGTONE = 1;

public static final int MODE\_IN\_CALL = 2;

public static final int MODE\_IN\_COMMUNICATION = 3;

public static final int NUM\_MODES = 4;

#### setParameters() 和 getParameters

/\*

\* Sets a group generic audio configuration parameters. The use of these parameters

\* are platform dependent, see libaudio

\*

\* param keyValuePairs list of parameters key value pairs in the form:

\* key1=value1;key2=value2;...

\*/

public static native int setParameters(String keyValuePairs);

/\*

\* Gets a group generic audio configuration parameters. The use of these parameters

\* are platform dependent, see libaudio

\*

\* param keys list of parameters

\* return value: list of parameters key value pairs in the form:

\* key1=value1;key2=value2;...

\*/

public static native String getParameters(String keys);

#### DeviceConnectionState

public static native int setDeviceConnectionState(int device, int state, String device\_address);

public static native int getDeviceConnectionState(int device, String device\_address);

#### ForceUse

public static native int setForceUse(int usage, int config);

public static native int getForceUse(int usage);

// device categories config for setForceUse, must match AudioSystem::forced\_config

public static final int FORCE\_NONE = 0;

public static final int FORCE\_SPEAKER = 1;

public static final int FORCE\_HEADPHONES = 2;

public static final int FORCE\_BT\_SCO = 3;

public static final int FORCE\_BT\_A2DP = 4;

public static final int FORCE\_WIRED\_ACCESSORY = 5;

public static final int FORCE\_BT\_CAR\_DOCK = 6;

public static final int FORCE\_BT\_DESK\_DOCK = 7;

public static final int FORCE\_ANALOG\_DOCK = 8;

public static final int FORCE\_DIGITAL\_DOCK = 9;

public static final int FORCE\_NO\_BT\_A2DP = 10;

public static final int FORCE\_SYSTEM\_ENFORCED = 11;

private static final int NUM\_FORCE\_CONFIG = 12;

public static final int FORCE\_DEFAULT = FORCE\_NONE;

// usage for setForceUse, must match AudioSystem::force\_use

public static final int FOR\_COMMUNICATION = 0;

public static final int FOR\_MEDIA = 1;

public static final int FOR\_RECORD = 2;

public static final int FOR\_DOCK = 3;

public static final int FOR\_SYSTEM = 4;

private static final int NUM\_FORCE\_USE = 5;

#### StreamVolume

public static native int initStreamVolume(int stream, int indexMin, int indexMax);

public static native int setStreamVolumeIndex(int stream, int index, int device);

public static native int getStreamVolumeIndex(int stream, int device);

#### MasterVolume 和 MasterMute

public static native int setMasterVolume(float value);

public static native float getMasterVolume();

public static native int setMasterMute(boolean mute);

public static native boolean getMasterMute();

#### getDevicesForStream()

public static native int getDevicesForStream(int stream);

#### PrimaryOutput

public static native int getPrimaryOutputSamplingRate();

public static native int getPrimaryOutputFrameCount();

## Audio辅助

### AudioManager和AudioService

Android中AudioManager主管的是音量及一些铃音等。在代码中的典型用法如下：

// Request the audio focus so that other apps can pause playback.

final AudioManager audioManager = AudioManager)getSystemService(Context.AUDIO\_SERVICE);

audioManager.requestAudioFocus(null, AudioManager.STREAM\_MUSIC,

AudioManager.AUDIOFOCUS\_GAIN);

得到的AUDIO\_SERVICE是AudioManager类型，然后具体的工作再传给AudioService去做。

AudioManager提供的具体方法如下：

### RingToneManager和RingTone

RingToneManager类管理所有的RingTone, 其实际是从数据库中查找出都有哪些Ringtone给调用者，然后调用者可以指定创建哪个RongTone, RongToneManager就会new 相应的RingTone类实例。

RingTone分为三种类型：Ringtone, notifaction, alam.

从其存储的位置还可以分为三种： internel ringtone, media ringtone, drmringtone.

RingTone类其实是一个含有StreamType的简单播放器，调用系统的MediaPlayer.

首先尝试用LocalPlayer（类型为MediaPlayer）进行播放设置进来的URI,如果没有LocalPlayer 则用RemotePlayer来进行播放。RemotePlayer是从AudioManager中得来的，即是一个服务中的。

RingTone与Tone音的区别，参见ToneGenerator

### ToneGenerator

ToneGenerator的职责是控制简单声音的Tone声发音。 例如电话拨号按键时发出的声音。

ToneGenerator的API也很简单，只是一个构造，然后是StartTone, StopTone.

有一个ToneType的概念。 有很多种Tone类型。

这个ToneGenerator与RingTone不一样， RingTone只是一种概念，其实就是普通的声音文件，比如Mp3， 把它们存到特定的位置，即成为RingTone. 这个在扫描的代码中有具体的代码，应该是有三个文件夹的位置。

ToneGenerator有native层的实现，具体的实现均位于Native层。

### SoundPool

SoundPool是为了加快那些短小并且需要频繁播放的声音准备的，例如游戏中的一些发音“哈哈哈，嘿嘿嘿”。

从它的表面意思就能看出来，声音池，哈哈！！

什么？ 不懂？ 那你知道线程池不？ 也不知道， 那你就别混了……

## Audio路由

### MediaRounter

待研究…

## Audio Framework层总结

待研究…

# Audio系统的整体结构

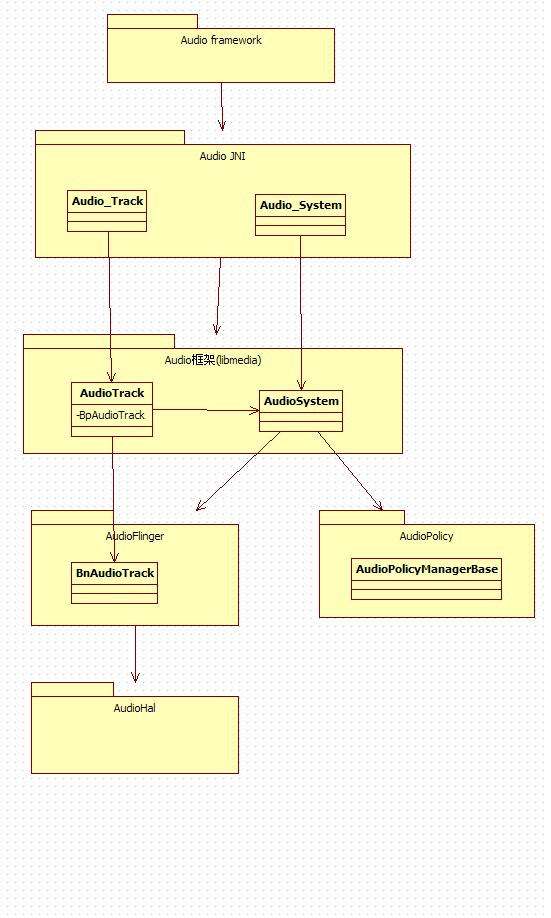
Audio从整个结构上来说分为Framework层，JNI 层，Native层，Hal及Driver层。

由上节可知，Framework层主要分为两个部分，Audio的核心层和Audio的辅助类。Audio核心层的类包括AudioTrack, AudioRecord, AudioSystem。Audio的辅助类主要有AudioManager，RingToneManager，ToneGenerator和SoundPool。

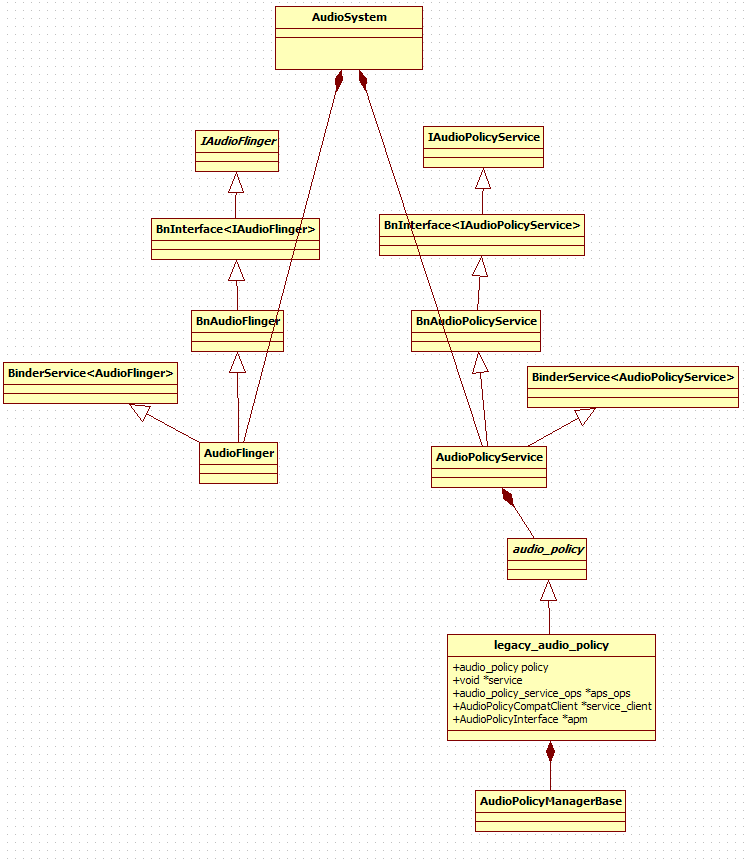
其中它们之中具有JNI的有：AudioRecorder, AudioTrack, AudioSystem, ToneGenerator, SoundPool。

再之下就是Native层的AudioPolicy和AudioFlinger了。这些内容将会在Native层的部分进行介绍。

下图是整个系统的框架图：



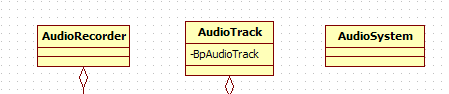
从Native层开始，它们的对外接口是libmedia中的AudioTrack, AudioRecorder, AudioSyste, 其中AudioSystem依赖于AudioFlinger和AudioPolicy这两个服务中提供的具体功能。如下图所示：



# Audio系统的Native层

如果按照Android的四层体系架构，Native层（包括）往下是一个完整的体系，Framework层（包括）往上是一个层次，它们分别有它们的对外接口。Framework层的对外接口，请参考第一章，Native层及以下的对外接口就是AudioRecorder, AudioTrack, AudioSyste这三个类。它们之间通过JNI进行联通。

对外接口的三个类：



## Audio框架libmedia

Audio的框架主要是几个对外接口类，及抽象接口定义，比如AudioTrack, AudioSyste, IAudioFlinger，以及AudioPolicyService类的定义。

未完待续…

## AudioFlinger

AudioFlinger是Android中音频的核心，主要负责音频的混音，并把完成的PCM数据送到Hal层的硬件去播放。

未完待续…

## AudioPolicy

### 代码位置

1. AudioPolicyService位置：

Frameworks/av/services/audioflinger/AudioService.h

Frameworks/av/services/audioflinger/AudioService.cpp

1. AudioPolicy的头文件定义：

hardware/libhardware/include/hardware/audio\_policy.h

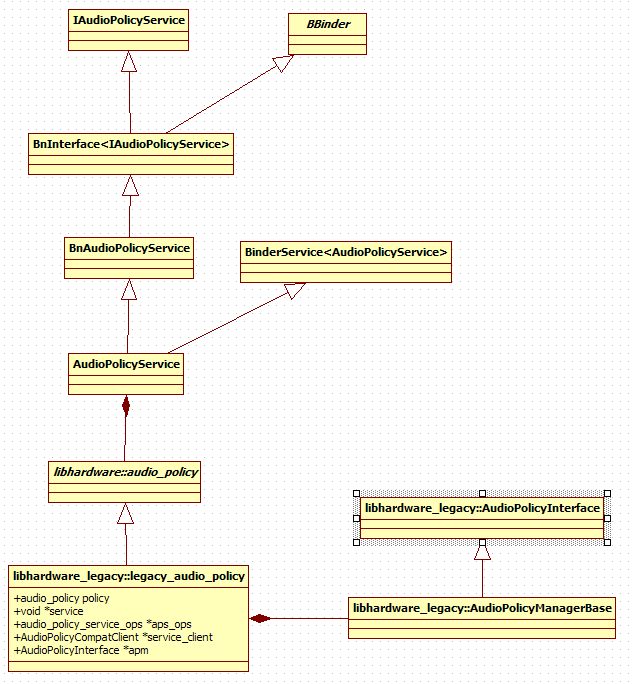
1. 实现位置

hardware/libhardware\_legacy/audio/audio\_policy\_hal.cpp

hardware/ libhardware\_legacy/audio/AudioPolicyManagerBase.cpp

1. 不同的厂商具有不同的实现从以上Android的实现继承下来。

### AudioPolicy的结构图



### AudioPolicyManagerBase

可以说AudioPolicyManagerBase是AudioPolicy实现的核心，这是Android的一个默认实现，对于每一个平台来说，产生可以从它继承一个子类，进行自己的定制化修改，但是AudioPolicyManagerBase已经完成了完整的逻辑，一般来说厂商的子类不会重载基类AudioPolicyManagerBase中的实现或者只重载一小部分的内容。

### AudioPolicyManagerBase的职责

设置电话的状态

设置Stream的音量，并计算音量，设置到AudioFlinger中。

StartOutput

Forceuse

根据电话状和Stream Type和ForceUse选择device

计算音量时，如果是插着耳机的话，为了保护听力，会降低音量。

### AudioPolicyManagerBase中的主要方法

virtual status\_t setDeviceConnectionState(audio\_devices\_t device,

AudioSystem::device\_connection\_state state,

const char \*device\_address) = 0;

// retrieve a device connection status

virtual AudioSystem::device\_connection\_state getDeviceConnectionState(audio\_devices\_t device,

const char \*device\_address) = 0;

// indicate a change in phone state. Valid phones states are defined by AudioSystem::audio\_mode

virtual void setPhoneState(int state) = 0;

// force using a specific device category for the specified usage

virtual void setForceUse(AudioSystem::force\_use usage, AudioSystem::forced\_config config) = 0;

// retrieve current device category forced for a given usage

virtual AudioSystem::forced\_config getForceUse(AudioSystem::force\_use usage) = 0;

// set a system property (e.g. camera sound always audible)

virtual void setSystemProperty(const char\* property, const char\* value) = 0;

// check proper initialization

virtual status\_t initCheck() = 0;

//

// Audio routing query functions

//

// request an output appropriate for playback of the supplied stream type and parameters

virtual audio\_io\_handle\_t getOutput(AudioSystem::stream\_type stream,

uint32\_t samplingRate = 0,

uint32\_t format = AudioSystem::FORMAT\_DEFAULT,

uint32\_t channels = 0,

AudioSystem::output\_flags flags = AudioSystem::OUTPUT\_FLAG\_INDIRECT) = 0;

// indicates to the audio policy manager that the output starts being used by corresponding stream.

virtual status\_t startOutput(audio\_io\_handle\_t output,

AudioSystem::stream\_type stream,

int session = 0) = 0;

// indicates to the audio policy manager that the output stops being used by corresponding stream.

virtual status\_t stopOutput(audio\_io\_handle\_t output,

AudioSystem::stream\_type stream,

int session = 0) = 0;

// releases the output.

virtual void releaseOutput(audio\_io\_handle\_t output) = 0;

// request an input appropriate for record from the supplied device with supplied parameters.

virtual audio\_io\_handle\_t getInput(int inputSource,

uint32\_t samplingRate = 0,

uint32\_t Format = AudioSystem::FORMAT\_DEFAULT,

uint32\_t channels = 0,

AudioSystem::audio\_in\_acoustics acoustics = (AudioSystem::audio\_in\_acoustics)0) = 0;

// indicates to the audio policy manager that the input starts being used.

virtual status\_t startInput(audio\_io\_handle\_t input) = 0;

// indicates to the audio policy manager that the input stops being used.

virtual status\_t stopInput(audio\_io\_handle\_t input) = 0;

// releases the input.

virtual void releaseInput(audio\_io\_handle\_t input) = 0;

//

// volume control functions

//

// initialises stream volume conversion parameters by specifying volume index range.

virtual void initStreamVolume(AudioSystem::stream\_type stream,

int indexMin,

int indexMax) = 0;

// sets the new stream volume at a level corresponding to the supplied index for the

// supplied device. By convention, specifying AUDIO\_DEVICE\_OUT\_DEFAULT means

// setting volume for all devices

virtual status\_t setStreamVolumeIndex(AudioSystem::stream\_type stream,

int index,

audio\_devices\_t device) = 0;

// retrieve current volume index for the specified stream and the

// specified device. By convention, specifying AUDIO\_DEVICE\_OUT\_DEFAULT means

// querying the volume of the active device.

virtual status\_t getStreamVolumeIndex(AudioSystem::stream\_type stream,

int \*index,

audio\_devices\_t device) = 0;

// return the strategy corresponding to a given stream type

virtual uint32\_t getStrategyForStream(AudioSystem::stream\_type stream) = 0;

// return the enabled output devices for the given stream type

virtual audio\_devices\_t getDevicesForStream(AudioSystem::stream\_type stream) = 0;

// Audio effect management

virtual audio\_io\_handle\_t getOutputForEffect(const effect\_descriptor\_t \*desc) = 0;

virtual status\_t registerEffect(const effect\_descriptor\_t \*desc,

audio\_io\_handle\_t io,

uint32\_t strategy,

int session,

int id) = 0;

virtual status\_t unregisterEffect(int id) = 0;

virtual status\_t setEffectEnabled(int id, bool enabled) = 0;

virtual bool isStreamActive(int stream, uint32\_t inPastMs = 0) const = 0;

virtual bool isStreamActiveRemotely(int stream, uint32\_t inPastMs = 0) const = 0;

virtual bool isSourceActive(audio\_source\_t source) const = 0;

# Audio系统的Hal层

## 代码位置

不同的厂商代码位置不同，但都位于hardware下，以下是intel平台CLV+的代码位置：

hardware/alsa\_sound/audio\_hw\_configurable

其中AudioStreamOutALSA.cpp类是音频输出的实现，如果要dump数据，直接在此类中把数据写到文件中即可， 采用的参数一般来说是， 44100, 16 bit, 双声道。