

Android 的 Audio 系统

Android 的 Audio 系统

- 第一部分 Audio 系统综述
- 第二部分 Audio 系统和上层接口
- 第三部分 Audio 的硬件抽象层

第一部分 Audio 系统综述

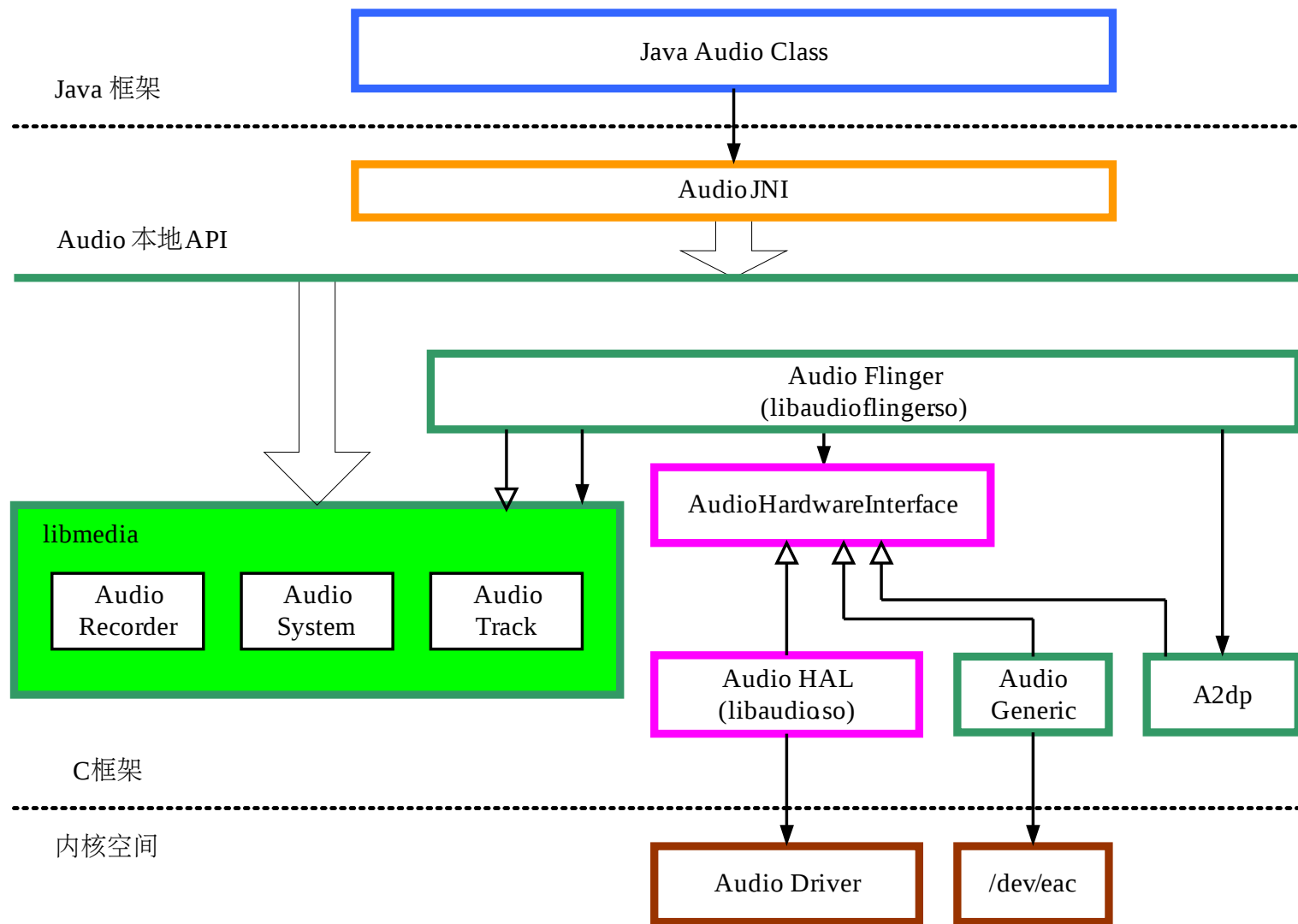
Audio 系统在 Android 中负责音频方面的数据流传输和控制功能，也负责音频设备的管理。

Audio 系统主要的分成几个层次：

1. media 中库提供的 Audio 系统的上层接口
2. AudioFlinger 作为 Audio 系统的中枢
3. Audio 库的硬件抽象层提供底层的支持
4. Audio 接口通过 JNI 和 Java 框架提供给上层

Audio 系统的上层接口主要提供了两方面的功能：放音（Track）和录音（Recorder）。

第一部分 Audio 系统综述



第一部分 Audio 系统综述

Media 库（libmedia.so）的 Audio 部分的目录中：
[frameworks/base/include/media/](#)
[frameworks/base/media/libmedia/](#)

这部分的内容被编译成库 **libmedia.so**，提供 Audio 部分的接口。

Audio Flinger (libaudioflinger.so) :
[frameworks/base/libs/audioflinger](#)

这部分内容被编译成库 **libaudioflinger.so**。

第一部分 Audio 系统综述

Audio 的 JNI 部分:

[frameworks/base/core/jni](#)

Audio 的 JAVA 部分:

[frameworks/base/media/java/android/media](#)

主要包含 AudioManager 和 Audio 系统的几个类。

Audio 硬件抽象层的接口:

[hardware/libhardware_legacy/include/hardware/](#)

第二部分 Audio 系统和上层接口

2.1 Audio 系统的各个层次

2.2 media 库中的 Audio 框架部分

2.3 AudioFlinger 本地代码

2.4 Audio 系统的 JNI 代码

2.5 Audio 系统的 Java 代码

2.1 Audio 系统的各个层次

Audio 系统的结构:

- ❑ libmedia.so 提供 Audio 接口，这些 Audio 接口既像上层开放，也向本地代码开发。
- ❑ libaudioflinger.so 提供 Audio 接口实现。
- ❑ Audio 硬件抽象层提供到硬件的接口，供 AudioFlinger 调用。
- ❑ Audio 使用 JNI 和 JAVA 对上层提供接口。

2.1 Audio 系统的各个层次

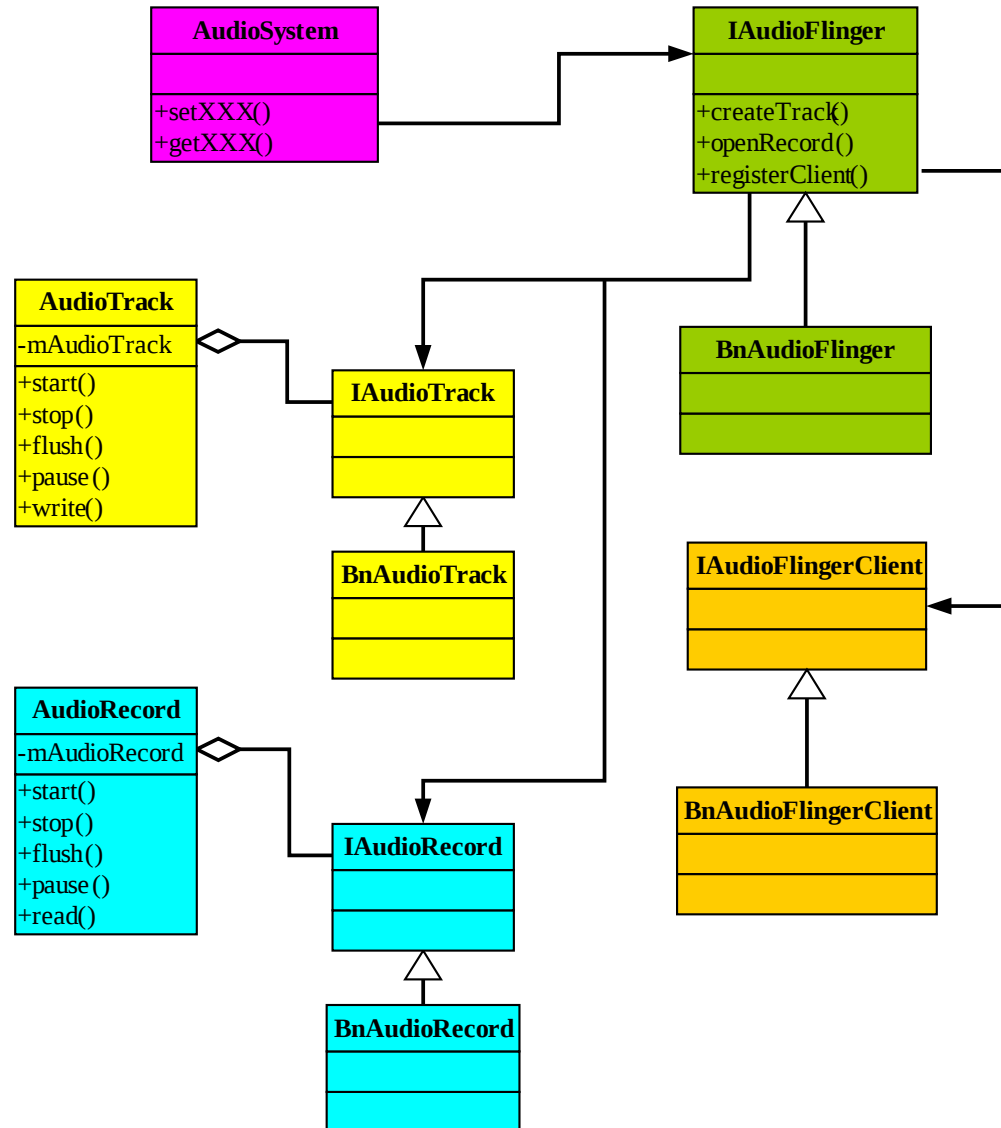
	Audio 管理环节	Audio 输出	Audio 输入
Java 层	AudioSystem	AudioTrack	AudioRecorder
本地框架层	AudioSystem	AudioTrack	AudioRecorder
AudioFlinger	IAudioFlinger	IAudioTrack	IAudioRecorder
硬件抽象层	AudioHardwareInterface	AudioStreamOut	AudioStreamIn

2.2 media 库中的 Audio 框架部分

Android 的 Audio 的核心框架在 media 库中提供，其中对上面主要实现 **AudioSystem**、**AudioTrack** 和 **AudioRecorder** 三个类。

提供了 **IAudioFlinger** 类接口，在这个类中，可以获得 **IAudioTrack** 和 **IAudioRecorder** 两个接口，分别用于声音的播放和录制。**AudioTrack** 和 **AudioRecorder** 分别通过调用 **IAudioTrack** 和 **IAudioRecorder** 来实现。

2.2 media 库中的 Audio 框架部分



2.2 media 库中的 Audio 框架部分

Audio 系统的头文件

(路径为: [frameworks/base/include/media/](#)) :

AudioSystem.h

IAudioFlinger.h

AudioTrack.h

IAudioTrack.h

AudioRecorder.h

IAudioRecorder.h

Ixxx 的接口通过 AudioFlinger 来实现, 其他接口通过 JNI 向上层提供接口。

2.2 media 库中的 Audio 框架部分

Audio 系统的头文件在 [frameworks/base/include/media/](#) 目录中，主要的头文件如下：

AudioSystem.h： media 库的 Audio 部分对上层的总管接口；

- ❑ IAudioFlinger.h： 需要下层实现的总管接口；
- ❑ AudioTrack.h： 放音部分对上接口；
- ❑ IAudioTrack.h： 放音部分需要下层实现的接口；
- ❑ AudioRecorder.h： 录音部分对上接口；
- ❑ IAudioRecorder.h： 录音部分需要下层实现的接口。

IAudioFlinger.h、IAudioTrack.h 和 IAudioRecorder.h 这三个接口通过下层的继承来实现（即 AudioFlinger）。

AudioFlinger.h， AudioTrack.h 和 AudioRecorder.h 是对上层提供的接口，它们既供本地程序调用（例如声音的播放器、录制器等），也可以通过 JNI 向 Java 层提供接口。

2.2 media 库中的 Audio 框架部分

AudioTrack 和 **AudioRecorder** 都具有 **start** , **stop** 和 **pause** 等接口。前者具有 **write** 接口, 用于声音的播放, 后者具有 **read** 接口, 用于声音的录制。

AudioSystem 用于 **Audio** 系统的控制工作, 主要包含一些 **set** 和 **get** 接口, 是一个对上层的类。

2.2 media 库中的 Audio 框架部分

AudioSystem.h :

```
class AudioSystem
{
public:
    enum stream_type {                // Audio 流的类型
        SYSTEM            = 1,
        RING              = 2,
        MUSIC             = 3,
        ALARM             = 4,
        NOTIFICATION      = 5,
        BLUETOOTH_SCO     = 6,
        ENFORCED_AUDIBLE  = 7,
        NUM_STREAM_TYPES
    };
    enum audio_output_type {          // Audio 数据输出类型
        // ..... 省略部分内容    };
    enum audio_format {              // Audio 数据格式
        FORMAT_DEFAULT = 0,
        PCM_16_BIT,
        PCM_8_BIT,
        INVALID_FORMAT
    };
};
```

2.2 media 库中的 Audio 框架部分

```
enum audio_mode {                // Audio 模式
    // ..... 省略部分内容    };
enum audio_routes {              // Audio 路径类型
    ROUTE_EARPIECE               = (1 << 0),
    ROUTE_SPEAKER                 = (1 << 1),
    ROUTE_BLUETOOTH_SCO           = (1 << 2),
    ROUTE_HEADSET                 = (1 << 3),
    ROUTE_BLUETOOTH_A2DP          = (1 << 4),
    ROUTE_ALL                     = -1UL,
};
static status_t setMasterVolume(float value);
static status_t setMasterMute(bool mute);
static status_t getMasterVolume(float* volume);
static status_t getMasterMute(bool* mute);
static status_t setStreamVolume(int stream, float value);
static status_t setStreamMute(int stream, bool mute);
static status_t getStreamVolume(int stream, float* volume);
static status_t getStreamMute(int stream, bool* mute);
static status_t setMode(int mode);
static status_t getMode(int* mode);
static status_t setRouting(int mode, uint32_t routes, uint32_t mask);
static status_t getRouting(int mode, uint32_t* routes);
// ..... 省略部分内容
};
```


2.3 AudioFlinger 本地代码

Audio 是 AudioFlinger 系统的中间层，其代码的路径为：

[frameworks/base/libs/audioflinger](#)

AudioFlinger 的核心文件是 AudioFlinger.h 和 AudioFlinger.cpp，提供了类 AudioFlinger，这个类是一个 IAudioFlinger 的实现。

2.3 AudioFlinger 本地代码

AudioFlinger 的实现:

```
class AudioFlinger : public BnAudioFlinger, public IBinder::DeathRecipient
{
public:    // ..... 省略部分内容
    virtual sp<IAudioTrack> createTrack(           // 获得音频输出接口 (Track)
        pid_t pid, int streamType, uint32_t sampleRate,
        int format, int channelCount, int frameCount,
        uint32_t flags, const sp<IMemory>& sharedBuffer,
        status_t *status);

    // ..... 省略部分内容
    virtual status_t setMasterVolume(float value);
    virtual status_t setMasterMute(bool muted);
    virtual status_t setStreamVolume(int stream, float value);
    virtual status_t setStreamMute(int stream, bool muted);
    virtual status_t setRouting(int mode, uint32_t routes, uint32_t mask);
    virtual uint32_t getRouting(int mode) const;
    virtual status_t setMode(int mode);
    virtual int      getMode() const;
    virtual sp<IAudioRecord> openRecord(           // 获得音频输出接口 (Record)
        pid_t pid, int streamType, uint32_t sampleRate,
        int format, int channelCount, int frameCount,
        uint32_t flags, status_t *status);
}
```

2.4 Audio 系统的 JNI 代码

Audio 的 JNI 部分的代码的路径为:

[frameworks/base/core/jni](#)

实现的几个文件为:

android_media_AudioSystem.cpp

android_media_AudioTrack.cpp

android_media_AudioRecord.cpp

主要提供三个类的支持:

android.media.AudioSystem

android.media.AudioTrack

android.media.AudioRecorder

2.5 Audio 系统的 JAVA 代码

Audio 的 JAVA 部分的代码的路径为:

[frameworks/base/media/java/android/media](#)

实现了以下的几个类:

`android.media.AudioSystem`

`android.media.AudioTrack`

`android.media.AudioRecorder`

`android.media.AudioFormat`

`android.media.AudioManager`

第三部分 Audio 的硬件抽象层

Audio 的硬件抽象层可以是 AudioFlinger 和 Audio 硬件的接口。可以基于 Linux 标准的 Alsa 或 OSS 实现，也可以基于私有的 Audio 驱动接口来实现。

Audio 的硬件抽象层的代码路径为：

[hardware/libhardware_legacy/include/hardware/](#)

其中主要的文件为：

AudioHardwareBase.h

AudioHardwareInterface.h

第三部分 Audio 的硬件抽象层

在 AudioHardwareInterface.h 中定义了类:

AudioStreamOut

AudioStreamIn

AudioHardwareInterface

在 AudioHardwareInterface.h 中定义了类:

AudioHardwareBase , 它继承 AudioHardwareInterface , 这是实现 Audio 硬件抽象层的主要接口。

第三部分 Audio 的硬件抽象层

Audio 的硬件抽象层 AudioStreamOut 和 AudioStreamIn 接口：

```
class AudioStreamOut {
public:
    virtual ~AudioStreamOut() = 0;
    virtual status_t setVolume(float volume) = 0;
    virtual ssize_t write(const void* buffer, size_t bytes) = 0;
    /*... */
};
class AudioStreamIn {
public:
    virtual ~AudioStreamIn() = 0;
    virtual status_t setGain(float gain) = 0;
    virtual ssize_t read(void* buffer, ssize_t bytes) = 0;
    /*... */
};
```

第三部分 Audio 的硬件抽象层

Audio 的硬件抽象层 AudioHardwareInterface 类 :

```
class AudioHardwareInterface
{
public:
    virtual status_t      initCheck() = 0;
    virtual status_t      setVoiceVolume(float volume) = 0;
    virtual status_t      setMasterVolume(float volume) = 0;
    virtual status_t      setRouting(int mode, uint32_t routes) = 0;
    virtual status_t      getRouting(int mode, uint32_t* routes) = 0;
    virtual status_t      setMode(int mode) = 0;
    virtual status_t      getMode(int* mode) = 0;
    /*... ..*/
    virtual AudioStreamOut* openOutputStream(
        int format=0,
        int channelCount=0,
        uint32_t sampleRate=0,
        status_t *status=0) = 0;
    virtual AudioStreamIn* openInputStream(
        int format,
        int channelCount,
        uint32_t sampleRate,
        status_t *status,
        AudioSystem::audio_in_acoustics acoustics) = 0;
    static AudioHardwareInterface* create();
};
```


第三部分 Audio 的硬件抽象层

在 **AudioFlinger** 的实现中，以下几个文件提供了 **Audio** 系统的通用实现：

- ❑ **AudioHardwareGeneric.cpp**
- ❑ **AudioHardwareStub.cpp**
- ❑ **AudioDumpInterface.cpp**

这些代码将编译成静态库 **libaudiointerface.so**，这作为 **Audio** 系统的通用实现来完成。

第三部分 Audio 的硬件抽象层

实际的 **Audio** 硬件抽象层，通常可以基于 **Linux** 中的 **OSS** 驱动程序和 **ALSA** 驱动程序来实现。

基于 **OSS** 的硬件抽象层的实现类似 **Audio Generic** 的实现，但是增加了控制接口。

基于 **ALSA** 的硬件抽象层的实现需要构建于用户空间的 **ALSA** 库上，目前在 **Android** 已经有了成熟的应用



谢谢！