Android 10 SDK 基础架构说明书

版本号: 1.0

发布日期: 2020.08.06

版本历史

版本号	日期	制/修订人	内容描述
1.0	2020.08.06	AW0681	初始版本文档

目　录

插　图

1 概述

Android 10 系统自下而上分别是 Linux 内核 (Linux Kernel)，系统库 (Libraries)，Android 运行时环境 (Android Runtime)，框架层 (Application Framework) 以及应用层 (Application)。本文主要是介绍基于 Android 10 系统的架构性设计，模块进行划分介绍，旨在让客户快速了解 Android 10 各模块基础结构与代码分布。

2 Android 10 模块架构介绍

以模块角度划分,Android 10 可划分为以下部分：

- 多媒体
- 显示
- 音频
- Camera
- OTA/Recovery
- 网络 Wifi
- 网络 Ethernet
- 蓝牙
- 多屏互动
- OMX

AndroidQ 支持完整的 Full Treble 特性, 关于 Treble 特性可参见谷歌官方链接 https://source.android.com/devices/architecture

Android 架构图先 show 一下

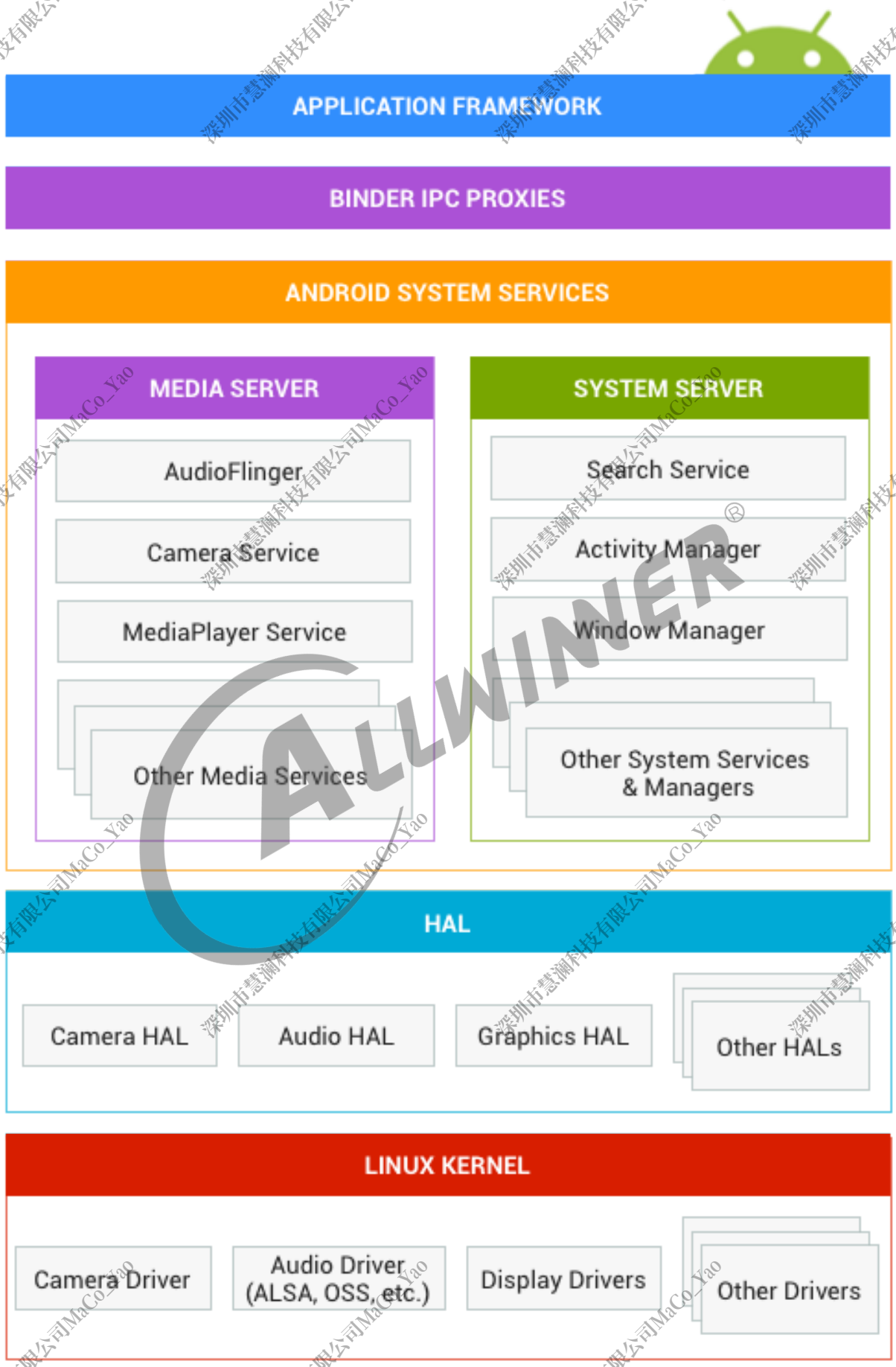

图 2-1: Android Q 架构图

3 多媒体

3.1 多媒体中间件 libcedarx

多媒体 cedarx 中间件是全志科技设计实现音视频编解码与播放控制的软件模块。主要模块包括接口层，流控模块，解封装模块，回放模块；

- 接口层：awplayer 是多媒体中间件 cedarx 在 android 平台上对接 mediaPlayer 的适配层。
- 流控模块：stream 模块将对本地或网络片源进行加载和处理，目前支持本地片源流以及通过 hls,http,rtsp 等网络协议传输的片源流。
- 解封装模块：parser 模块将对片源码流进行解封装操作，将片源码流分离为音频、视频、字幕流送给音频、视频、字幕解码库进行解码。
- 回放模块：playback 控制解码及音视频字幕的解码和送显。包括音频解码模块 audioDecComponent，视频解码模块 videoDecComponent，字幕解码模块 subtitleDecComponent；render 是渲染模块，音频渲染模块 audioRenderComponent，视频渲染模块 videoRenderComponent，字幕渲染模块 subtitleRenderComponent 六个模块。

3.2 中间件中各模块功能

- 接口层（android_adapter）：接口层中包括 awplayer，IPTV 的调用接口。接口层中包括 awplayer 播放器的接口文件，接收播放器 apk 发下来各种操作，并派发到 cedarx 的 demuxComponent 和 player 中进行处理。
- 中间件的播放器（xplayer）：xplayer 与接口层 awplayer 对接，响应播放操作的逻辑，控制 cedarx 的 demuxComponent 和 player 进行具体的播放逻辑。
- 流控模块（stream）：stream 模块即 cedarx 的流控模块，负责片源码流数据的加载和处理。stream 是播放的对象也是 parser 解封装的对象。
- 解封装模块（parser）：parser 模块即 cedarx 的接封装模块，parser 模块将对片源码流进行解封装操作，将片源码流分离为音频、视频、字幕流送给音频、视频、字幕解码库进行解码。
- 播控模块模块（playback）：player 是播控模块中的控制函数。实现了音视频字幕模块的加载与初始化，并控制码流的分发和调用音视频字幕解码模块进行解码，然后在音视频字幕解码接收模块做音视频同步处理并输出到屏幕与音响进行播放。主要包括音频解码模块包括音频解码模块 audioDecComponent，视频解码模块 videoDecComponent，字幕解码模块 subtitleDecComponent；音频解码渲染模块 audioRenderComponent，视频解码渲染模块 videoRenderComponent，字幕解码接渲染模块 subtitleRenderComponent 六个模块。

3.3 多媒体框架

我司多媒体框架如下图所示

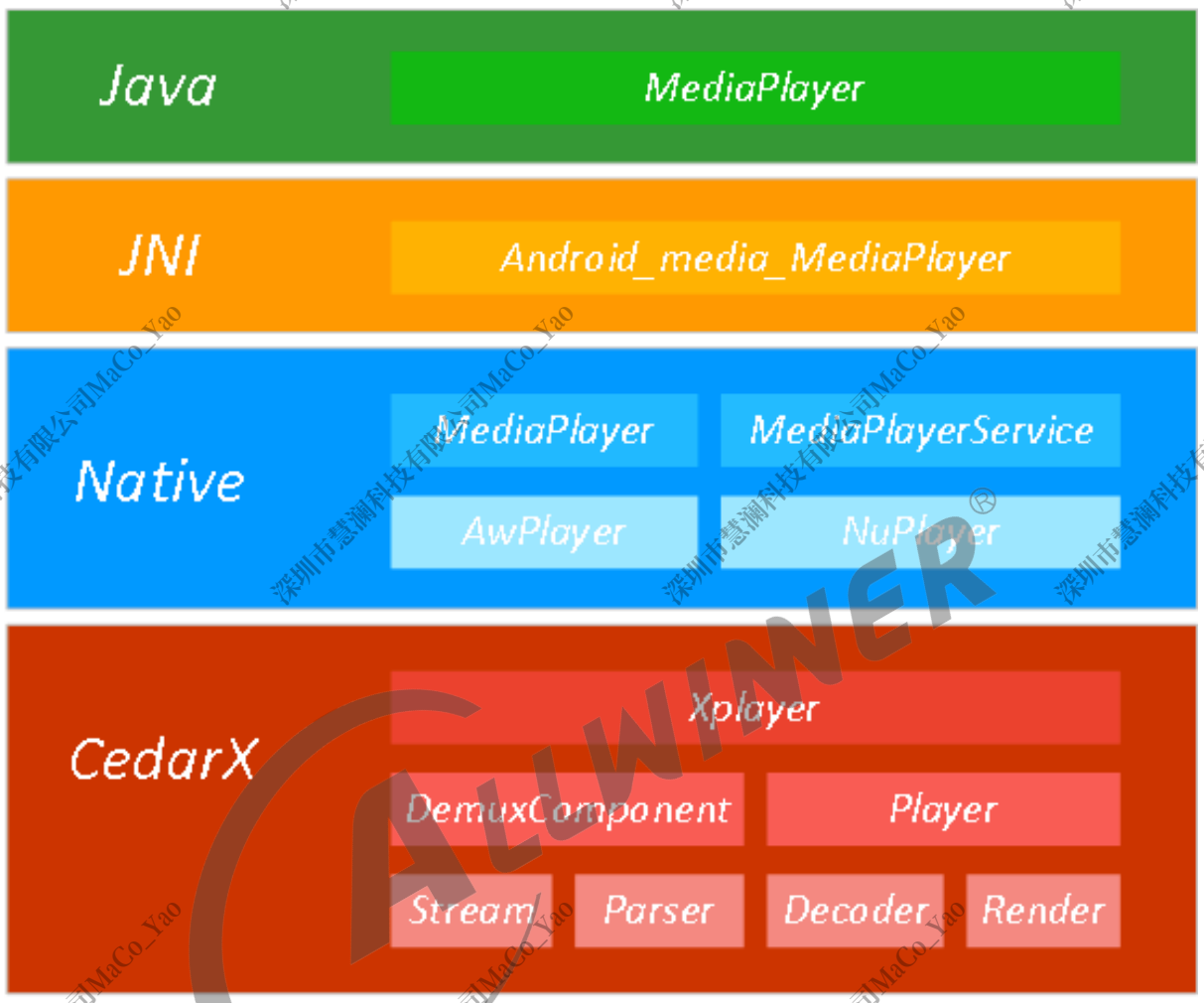

图 3-1：多媒体框架图

3.4 多媒体代码结构

1. Android 多媒体模块，java 层和 jni 层代码目录:

```
android/frameworks/base/media
├── java
├── jni
├── lib
├── mca
└── tests
```

2. Android 多媒体模块，Native 层代码目录:

```
android/frameworks/av/media
├── common_time
├── libcedarc
├── libcedarx
├── libcpustats
├── libeffects
├── libmedia
├── libmediaplayerservice
├── libnbaio
├── libstagefright
├── mediaserver
└── mtp
```

3. CedarX 多媒体中间件目录:

```
android/frameworks/av/media/libcedarx
├── android_adapter
│   ├── awplayer
│   ├── iptv
│   ├── metadataretriever
│   └── output
├── awrecorder
├── conf
├── config
├── demo
├── document
├── external
├── libcore
│   ├── playback
│   └── stream
├── xmetadataretriever
└── xplayer
```

4. CedarC 多媒体编解码库目录:

```
android/frameworks/av/media/libcedarc
├── base
├── conf
├── config
├── include
├── library
├── memory
├── openmax
│   ├── adec
│   ├── libstagefrighthw
│   ├── omxcore
│   ├── vdec
│   └── venc
└── vdecoder
```

4 显示

4.1 显示系统架构

我司显示系统架构图如下所示：

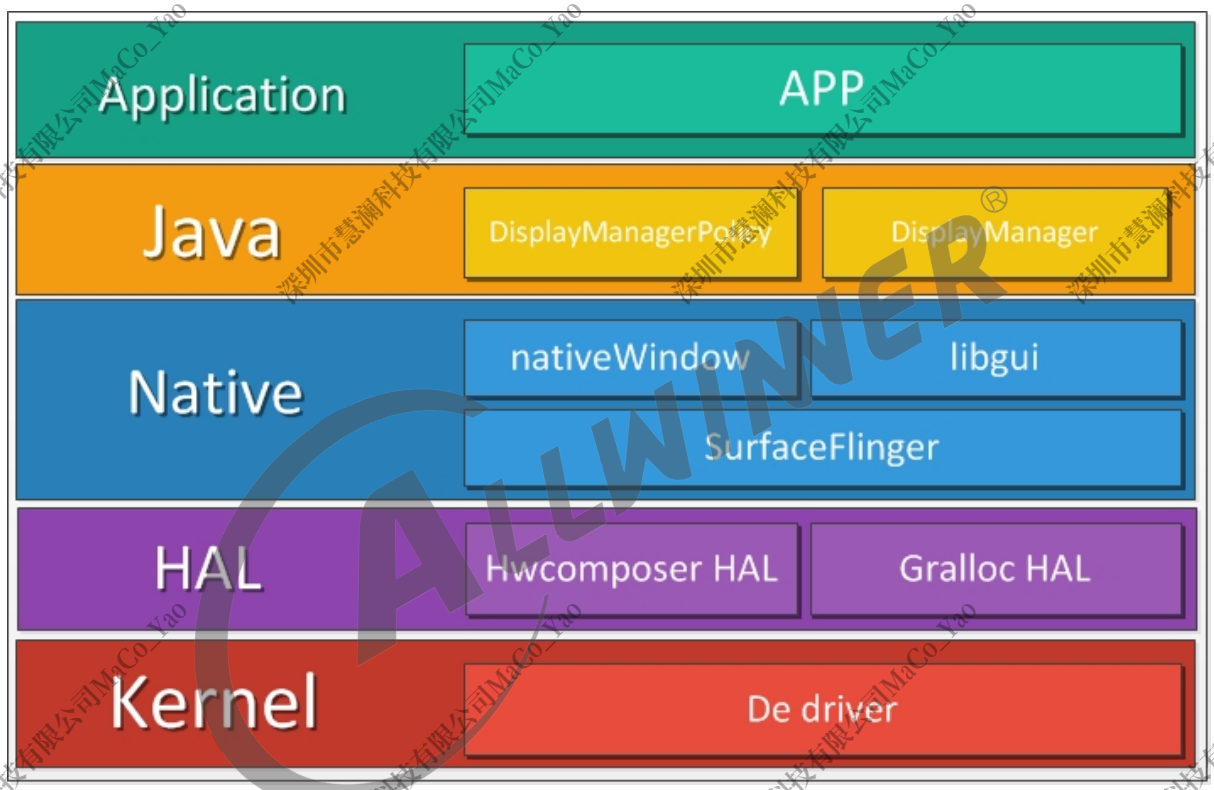

图 4-1：显示系统架构图

4.2 显示系统代码结构

```
vendor/aw/auto/framework/display/service/*
frameworks/native/libs/gui/
frameworks/native/server/surfaceflinger/
hardware/aw/display/hwc-hal/
hardware/aw/gpu/mail-midgard/gralloc
longan/kernel/linux-4.9/driver/video/fbdev/sunxi/disp2/
```

5 音频

5.1 音频系统架构

我司音频系统架构图如下所示：

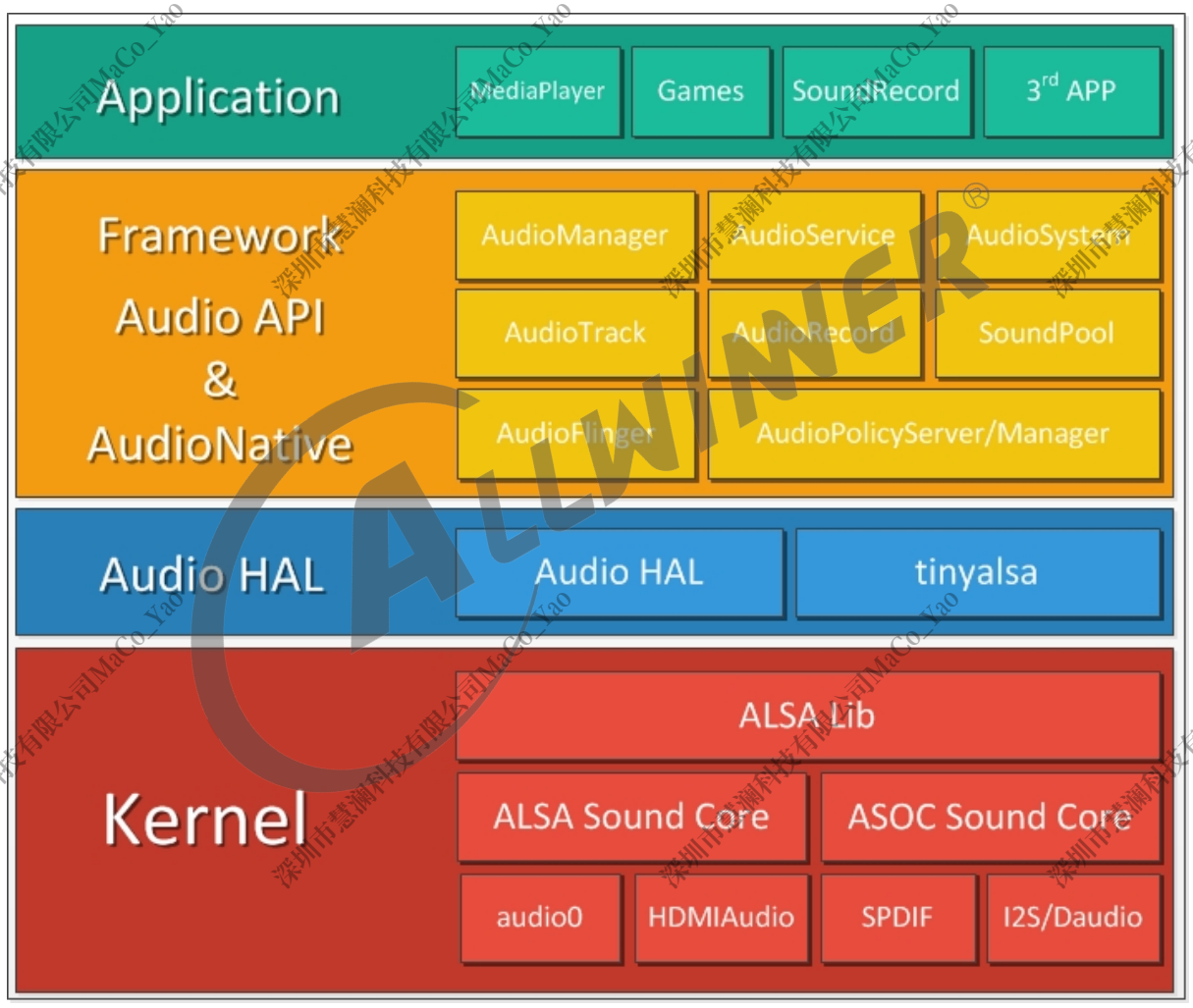

图 5-1：音频架构图

5.2 音频代码结构

```
framework:
android/frameworks/base
1.AudioManager.java    ---音频管理器，音量调节、音量UI、设置和获取参数等控制流的对外API
2.AudioService.java    ---音频系统服务（java层），音量调节、音量UI、音频设备插拔等控制流的具体实现
3.AudioSystem.java    ---音频控制的入口，是native层对上服务的接口
android/frameworks/av
1.AudioFlinger.cpp    ---音频系统的核心一，承担音频数据流AudioTrack和AudioRecord的混音、重采、输送等
    责任
2.AudioPolicyService.cpp    ---音频系统的核心二，负责音频策略，包含Audio HAL的加载，音频路由的选择等

HAL:
android/hardware/aw/audio/hal
audio_hw.c    ---AudioFlinger与音频驱动之间的对接层，匹配android系统与硬件的关键层
audio_data_dump.c    ---抓取hal音频输入输出数据
```

6 Camera

6.1 Camera 系统架构

我司 Camera 架构图如下图所示

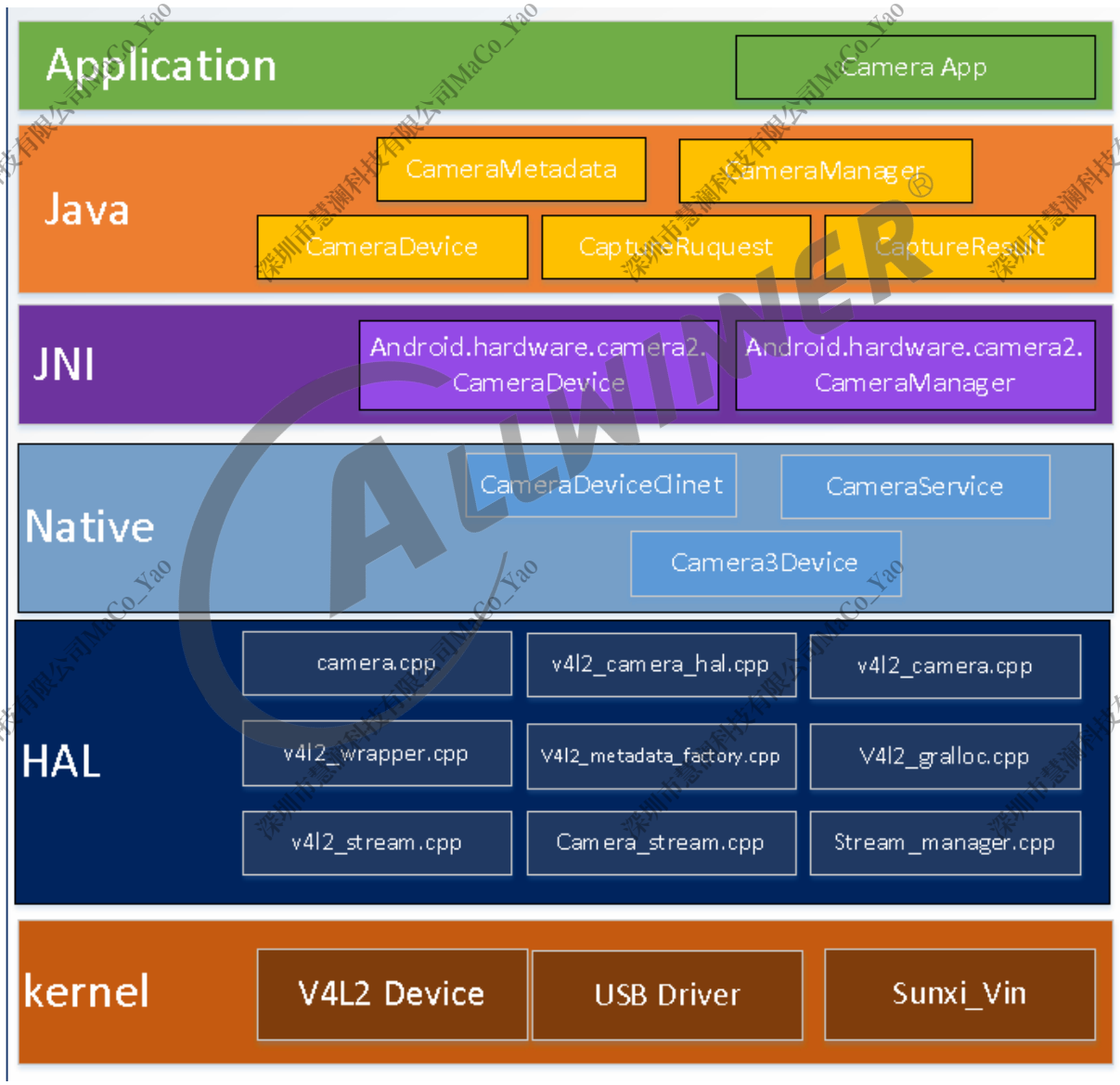

图 6-1: Camera 架构图

6.2 Camera 代码结构

6.2.1 JAVA 应用层

本架构在 JAVA 应用层运行的 apk，使用的是 Android 原生的 Camera，源码位于如下路径：

`packages/apps/Camera2`

它主要调用下面的原生接口进行实现：

```
Camera.java(android/frameworks/base/core/java/android/hardware)
```

6.2.2 Android 本地框架层

Android 本地框架层是 JAVA 层与硬件抽象层的桥梁，它提供了客户端与服务器端通过 IBinder 机制进行进程间通信的模型，极大的方便了上层与下层之间的沟通。它的主要代码文件夹为：

```
frameworks/av/camera/
frameworks/av/include/camera/
frameworks/av/services/camera/libcameraservice/
```

Android Camera 模块的本地框架的主要头文件包括：

```
1. Camera.h
2. CameraMetadata.h
3. CpatureResult.h
4. CaptureRequest.h
5. OutputConfiguration.h
6. ICamera.h
7. ICameraClient.h
8. ICameraService.h
9. CameraHardwareInterface.h
```

主要 cpp 文件包括：

```
1. Camera.cpp
2. CameraService.cpp
3. CameraClient.cpp
4. Camera2Client.cpp
5. CameraMetadata.cpp
6. CaptureResult.cpp
7. Camera3Device.cpp
8. Camera3Stream.cpp
```

Native framework 的定义位于 frameworks/av/camera/Camera.cpp 中，它提供了 android.hardware.Camera 类在本地的实现。这些类调用 IPC binder 来获得 camera service。

IPC binder 代理可以现实进程间通信。在 frameworks/av/camera 目录下有三个 camera binder 类的定义。ICameraService 是 cameraservice 的接口，ICamera 是被打开的 camera 设备的接口，ICameraClient 是 camera 设备返回给 application framework 层的接口。

6.2.3 Android 硬件抽象层接口

Android 的硬件抽象层接口在 framework/av/services/camera/device1 和 framework/av/services/camera/device3，它位于frameworks/av/services/camera/libcameraservice中，主要为上层提供了可以对硬件抽象层进行操作的接口。

6.2.4 Android 硬件抽象层

Camera 的 HAL 层路径为:

```
(android/hardware/aw/camera/)
```

7 OTA/Recovery

OTA 即空中下载技术 (over-the-air), 指 Android 系统提供的标准软件升级方式，即通过无线网络下载更新包并无损地升级系统，而无需通过有线方式进行连接。

7.1 OTA/Recovery 代码结构

7.1.1 Recovery Aosp 代码结构

```
├── bootable/recovery
└── recovery.cpp
    recovery进程入口
└── screen_ui.cpp
└── ui.cpp
    负责Recovery的显示功能
└── bootloader.cpp
    负责操作bootloader
└── updater
    └── install.cpp
        负责定义OTA升级中安装脚本的语句,厂商可在此扩展脚本接口。
└── applypatch
    └── applypatch.c
        负责在差分升级中通过打patch形式将旧文件升级到新文件
└── verifier.cpp
    负责提供校验OTA包的方法
└── roots.cpp
  提供挂载外部设备方法
└── default_device.cpp
  负责控制Recovery界面菜单
```

7.1.2 制作 OTA 升级包目录结构

```
└── build/tools/releasetools
    └── ota_from_target_files
          利用编译生成的targetfile文件生成OTA完全包或者差分包
    └── img_from_target_files
          利用编译生成的targetfiles生成包括system.img,boot.img,recovery.img等镜像
    └── sign_target_files_apk
          用于对targetfiles中的apk进行签名
```

8 网络 WIFI

8.1 网络 WIFI 框架

我司网络 WIFI 框架图如下所示

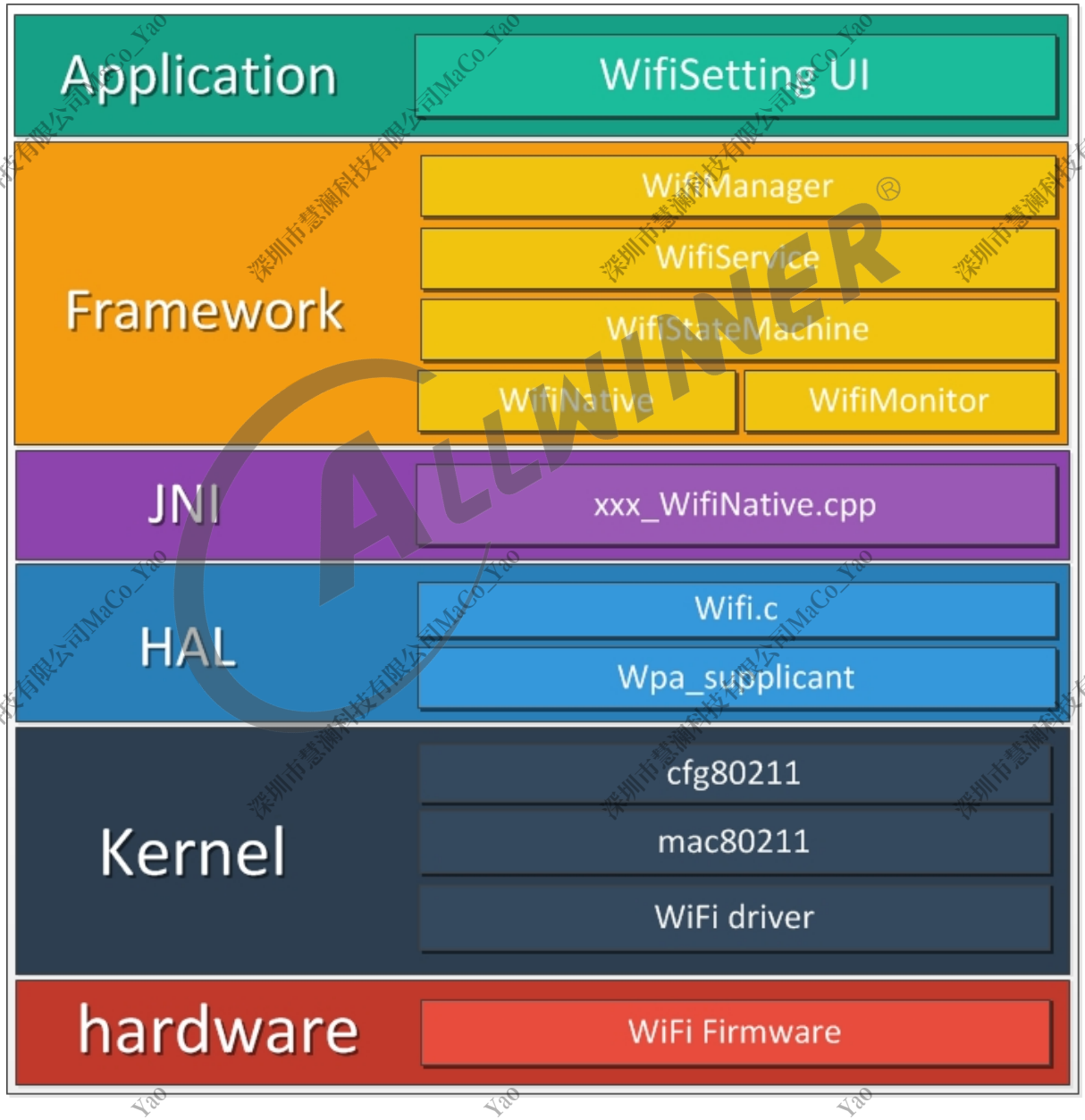

图 8-1: 网络 WIFI 框架图

1. WifiSetting ui 是 WiFi 的界面运用 APP，通过界面的操作，调用到 Framework 层的 API，

此 API 由 WifiManager 提供；

2. WifiManger 提供 API 给应用层用，然后接口函数通过异步通道 mAsyncChannel 发送信息给 WifiService 或是通过包含直接调用 WifiService 里面的方法；
3. WifiStateMachine 是 WiFi framework 的逻辑控制中心，管理 WiFi 的各种状态；
4. WifiNative 和 jni 是 frameworks 与 hal 层的沟通桥梁，将 WiFi 的操作命令传递给 wpa_supplicant；
5. WifiMonitor 通过阻塞等到 wpa_supplicant 反馈的命令；
6. wpa_supplicant 是 WiFi 驱动和 Android 层的中转站，wpa_supplicant 通过 nl80211 将命令传递给驱动，同时也负责对协议和加密认证的支持；

8.2 网络 WIFI 代码结构

层次	路径
APP	packages/apps/Settings/src/com/android/settings/wifi
framework	frameworks/base/wifi
jni	frameworks/base/core/jni/android_net_wifi_WifiNative.cpp
hal	hardware/libhardware_legacy/wifi
wpa_supplicant	external/wpa_supplicant_8
driver	linux-4.9/driver/net/wireless
firmware	hardware/broadcom/wlan/bcmdhd/firmware

9 蓝牙

9.1 蓝牙系统框架图

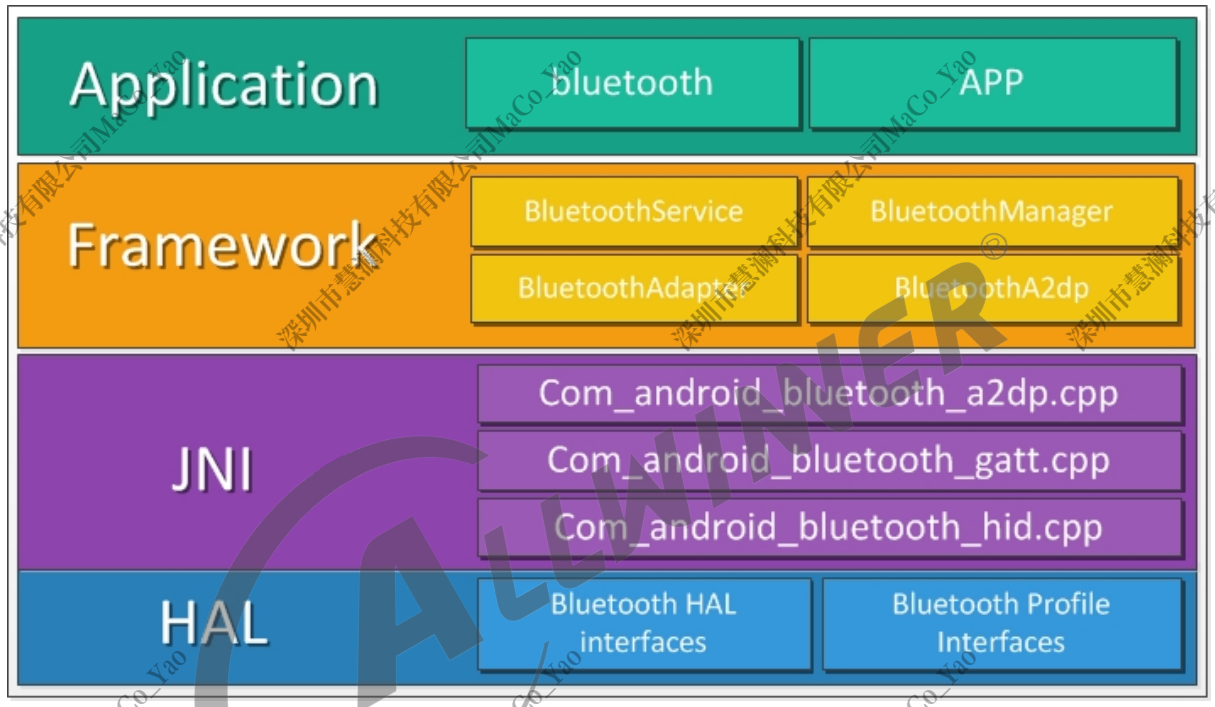

图 9-1: 蓝牙系统架构图

9.2 蓝牙模块代码目录

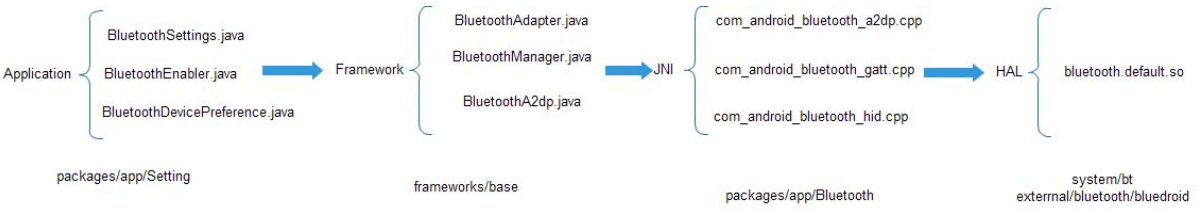

图 9-2: 蓝牙源码结构图

1. 模块涉及的 AOSP 原生代码目录

```
└── hardware/interface/bluetooth
    Broadcom蓝牙模组接口层
└── packages/apps/Bluetooth
    蓝牙进程，实现蓝牙服务和JNI部分，负责衔接frameworks和蓝牙协议栈。
└── frameworks/base/core/java/android/bluetooth
    蓝牙API。
```

2. 模块涉及的 SOC 厂家适配代码目录

```
├── hardware/broadcom/libbt                       # broadcom 接口库
├── hardware/broadcom/wlan/bcmdhd/firmware        # broadcom Wi-Fi/BT 固件
├── hardware/aw/wireless/partner/ampak/firmware # Ampak(broadcom chip) Wi-Fi/BT 固件
├── hardware/realtek/bluetooth/libbt-vendor       # realtek 接口库
├── hardware/realtek/bluetooth/firmware           # realtek BT 固件
├── hardware/xradio/bt/libbt-vendor               # xradio 接口库
└── hardware/xradio/bt/firmware                   # xradio BT 固件
```

在支持蓝牙语音通话功能时会涉及到音频模块，涉及到音频模块目录有：

```
android/hardware/aw/audio
```

10 多屏互动

10.1 多屏互动简介

多屏互动部分通过 MiracastReceiver 和 AllCast 应用来实现。MiracastReceiver 由全志科技开发实现，负责提供 Miracast 镜像接收端功能。目前以系统应用方式内置于 SDK 中。AllCast 由乐播科技开发实现，也叫乐播投屏，负责提供 DLNA 和 AirPlay 接收端功能，DLNA 接收端功能包括音乐、视频、图片推送，AirPlay 接收端功能包括音乐、视频、图片、镜像推送。目前以预装应用方式内置于 SDK 中，用户可以自由卸载和更新。

10.2 多屏互动目录结构

```
└── Android.mk
└── apk
    └── Android.mk
    └── MiracastReceiver.apk
    └── AllCast.apk
        └── allwinnertech
└── Android.mk
└── MiracastReceiver
    └── Android.mk
    └── MiracastReceiver.apk
    └── lib
```

11 OMX

11.1 OMX 简介

OpenMax 是一个多媒体应用程序的框架标准，分成三个层次分别是，OpenMax AL(应用层)，OpenMax IL(集成层) 和 OpenMax DL(开发层)。

第一层：OpenMax AL(Appliction Layer，应用层) 这一层是多媒体应用和多媒体中间层的标准接口，它使得多媒体应用在多媒体接口上具有可移植性。OpenMAX AL 层是 OpenMAX 规范 API 集得最上层接口，对于上层多媒体应用的开发只需要关注 OpenMax AL 层的接口，因此开发者只需要调用 AL 层的相应接口函数就可以完成对多媒体的开发。

第二层：OpenMax IL(Integration Layer，集成层) 这一层使得多媒体应用和多媒体框架可以以统一的方式访问多媒体编解码组件和底层的组件，多媒体编解码组件可以是硬件编解码和软件编解码。在架构底层上位多媒体的编解码和数据处理定义了一套统一的编程接口.OpenMax IL API，为用户屏蔽了底层的细节。IL 的主要目的是使用特征集合为编解码器提供一个系统抽象，为解决多个不同媒体系统之间轻便性的问题。

第三层：OpenMax DL(Development Layer，开发层) 这一层包含了视频、音频、图像编解码使用的函数集合，这些函数可以由芯片或硬件厂商对新处理器进行实现和优化，然后编解码供应商使用它来编写更广泛的编解码器功能。它包括音频信号的处理功能，如 FFT 和 filter，图像原始处理，如颜色空间转换、视频原始处理，以实现例如 MPEG-4、H.264、MP3、AAC 和 JPEG 等编解码器的优化。

11.2 OMX 框架

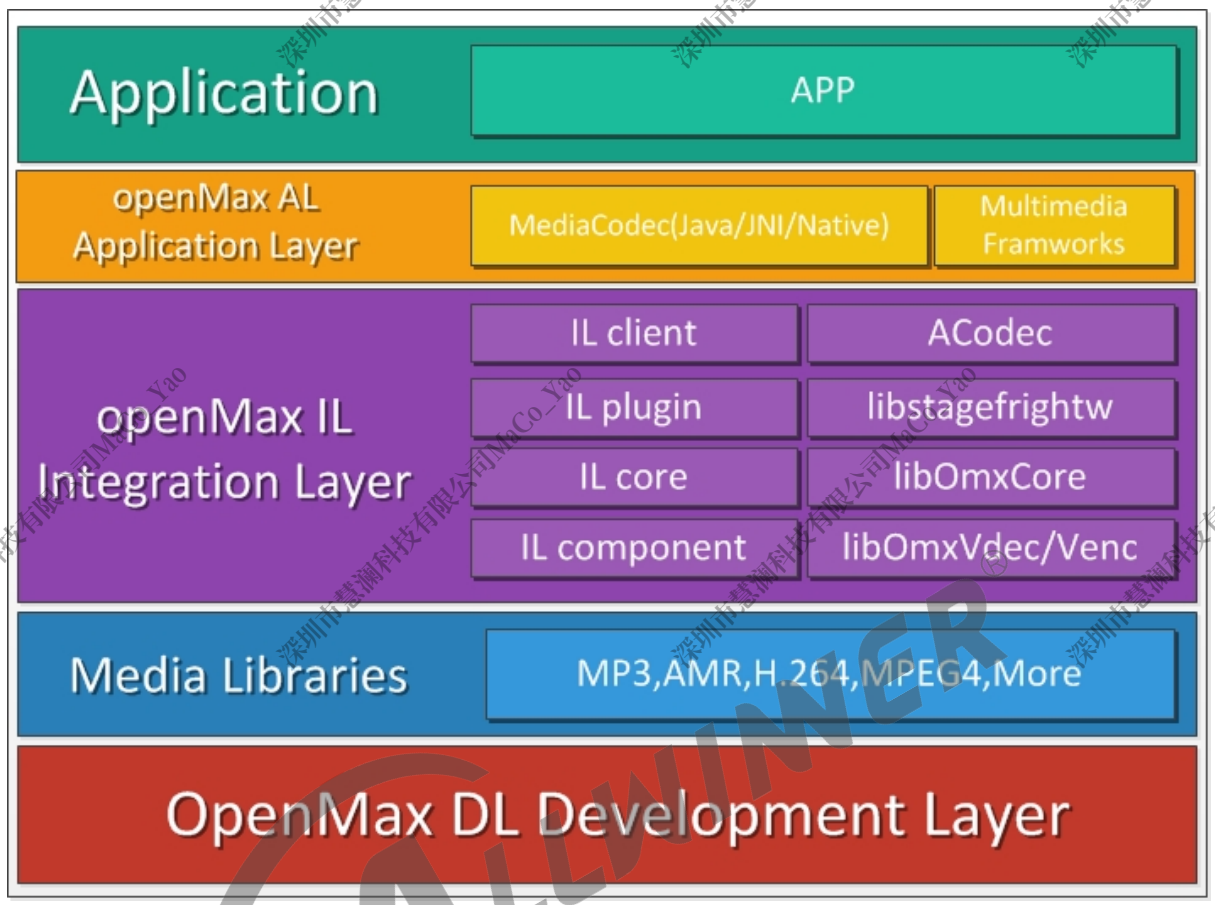

图 11-1: OMX 架构图

我司 Android OMX 框架如上图所示，基本使用的是标准 OpenMax IL 层的接口。

11.3 OMX 代码结构

11.3.1 Android 中 OpenMax 分层

MediaCodec 分为 3 部分：Java、JNI 和 Native。

1. Java 是上层 apk 使用的接口；
2. JNI 是 Java 访问 Native 的 JNI 方法；
3. Native 是 ACodec 的 wrapper；

- ACodec 是 OpenMAX IL 中的 OpenMAX IL client；

- Libstagefrighthw 是厂商的 OpenMAX IL plugin；
- libOMXCore 是 OpenMAX IL 中的 core；
- libOMXVdec 和 libOMXVenc 是 OpenMAX IL 中的 component。

11.3.2 Android 中 OpenMax 源码结构

```
- Api层（MediaCodec.java）
android/frameworks/base/media/java/android/media/MediaCodec.java
- JNI层（android_media_MediaCodec.cpp）
android/frameworks/base/media/jni/android_media_MediaCodec.cpp
- Native层（MediaCodec.cpp）
android/frameworks/av/media/libstagefright/MediaCodec.cpp
- libstagefrighthw实现
android/frameworks/av/media/libcedarc/openmax/libstagefrighthw
- libOMXCore实现
android/frameworks/av/media/libcedarc/openmax/omxcore
- libOmxVdec/libOmxVenc实现
android/frameworks/av/media/libcedarc/openmax/vdec
android/frameworks/av/media/libcedarc/openmax/venc
```

著作权声明

本文档及内容受著作权法保护，其著作权由珠海全志科技股份有限公司（“全志”）拥有并保留一切权利。

本文档是全志的原创作品和版权财产，未经全志书面许可，任何单位和个人不得擅自摘抄、复制、修改、发表或传播本文档内容的部分或全部，且不得以任何形式传播。

商标声明





（不完全列举）均为珠海全志科技股份有限公司的商标或者注册商标。在本文档描述的产品中出现的其它商标，产品名称，和服务名称，均由其各自所有人拥有。

免责声明

您购买的产品、服务或特性应受您与珠海全志科技股份有限公司（“全志”）之间签署的商业合同和条款的约束。本文档中描述的全部或部分产品、服务或特性可能不在您所购买或使用的范围内。使用前请认真阅读合同条款和相关说明，并严格遵循本文档的使用说明。您将自行承担任何不当使用行为（包括但不限于如超压，超频，超温使用）造成的不利后果，全志概不负责。

本文档作为使用指导仅供参考。由于产品版本升级或其他原因，本文档内容有可能修改，如有变更，恕不另行通知。全志尽全力在本文档中提供准确的信息，但并不确保内容完全没有错误，因使用本文档而发生损害（包括但不限于间接的、偶然的、特殊的损失）或发生侵犯第三方权利事件，全志概不负责。本文档中的所有陈述、信息和建议并不构成任何明示或暗示的保证或承诺。

本文档未以明示或暗示或其他方式授予全志的任何专利或知识产权。在您实施方案或使用产品的过程中，可能需要获得第三方的权利许可。请您自行向第三方权利人获取相关的许可。全志不承担也不代为支付任何关于获取第三方许可的许可费或版税（专利税）。全志不对您所使用的第三方许可技术做出任何保证、赔偿或承担其他义务。