全志科技
Allwinner Technology

H6 Android N

SDK 基础架构书

0.8

2017.3.31

文档履历

版本号	日期	制/修订人	内容描述
0.8	2017.3.31		正式版本

目录

1. H6 AndroidN 模块架构介绍

Android N 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)。

以模块角度划分,H6 Android N 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
- 产测工具

本说明书将会以模块进行划分介绍，旨在让客户快速了解 H6 AndroidN 各模块基础结构与代码分布。

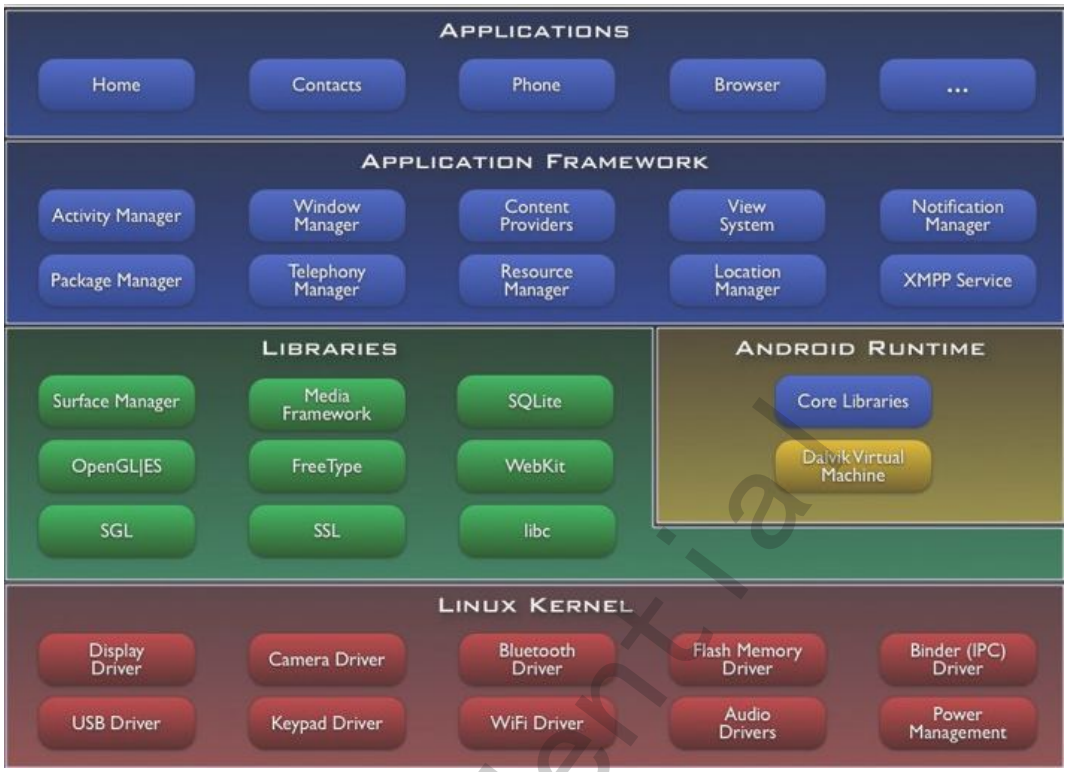

图 1: Android N 架构图

2. 多媒体

2.1 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：AW_PLAYER 是沟通上层播放的调用接口。
- 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,rtspd 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接封装操作，将片源码流分离为音频、视频、字幕流送给频、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 audioDecComp, 视频解码模块 VideoDecComp, 字幕解码模块 SubtitleDecComp；音频解码接收模块 audioRenderComp, 视频解码接收模块 VideoRenderComp, 字幕解码接收模块 SubtitleRenderComp 六个模块。

2.2 中间件中各模块功能

- 接口层（android_adapter）接口层中包括 AW_PLAYER,iptv 的调用接口。AW_PLAYER 为播放器的接口文件，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 AW_PLAYER 对接，响应播放操作的逻辑，控制 CedarX 的 demuxcomp 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接封装操作，将片源码流分离为音频、视频、字幕流送给频、视频、字幕解码库进行解码。
- 回放模块（playback）Player 是回放模块中的控制函数。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 audioDecComp, 视频解码模块 VideoDecComp, 字幕解码模块 SubtitleDecComp；音频解码接收模块 audioRenderComp, 视频解码接收模块 VideoRenderComp, 字幕解码接收模块 SubtitleRenderComp 六个模块。

2.3 多媒体框架

我司多媒体框架如下图所示

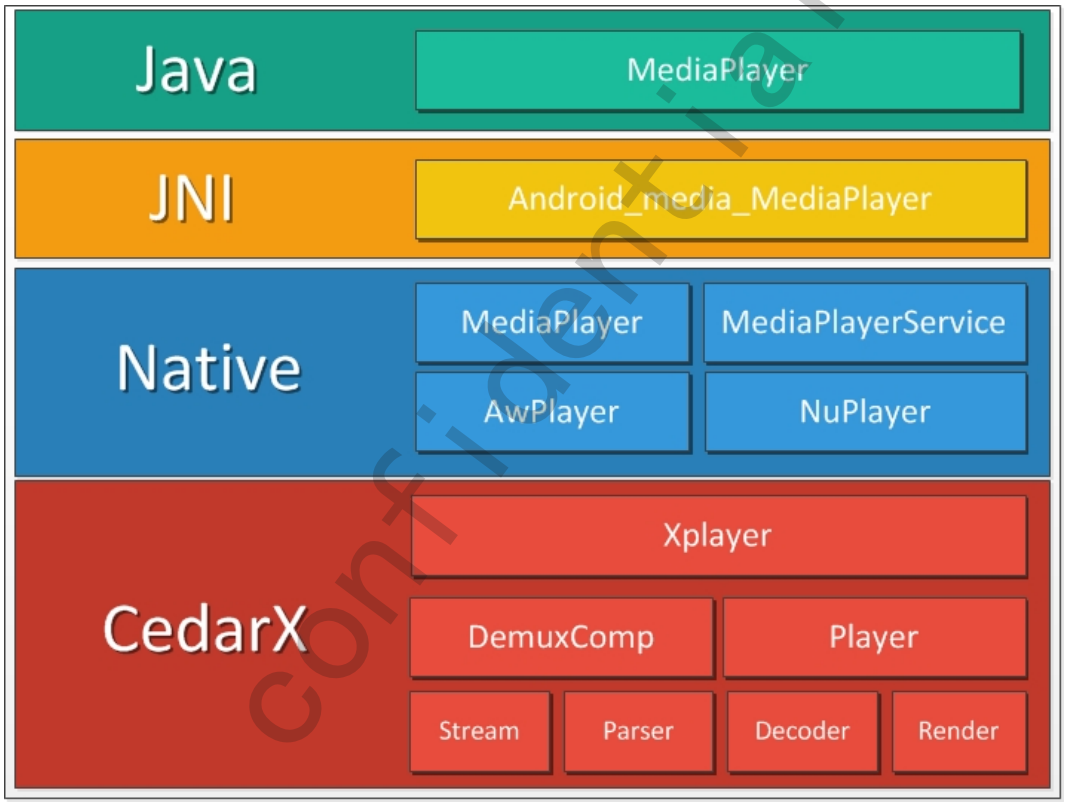

图 2: 多媒体框架图

2.4 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
├── 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
├── config
├── include
├── library
├── memory
├── openmax
│   ├── adec
│   ├── libstagefrighthw
│   ├── omxcore
│   ├── vdec
│   └── venc
└── vdecoder
```


3. 显示

3.1 显示系统架构

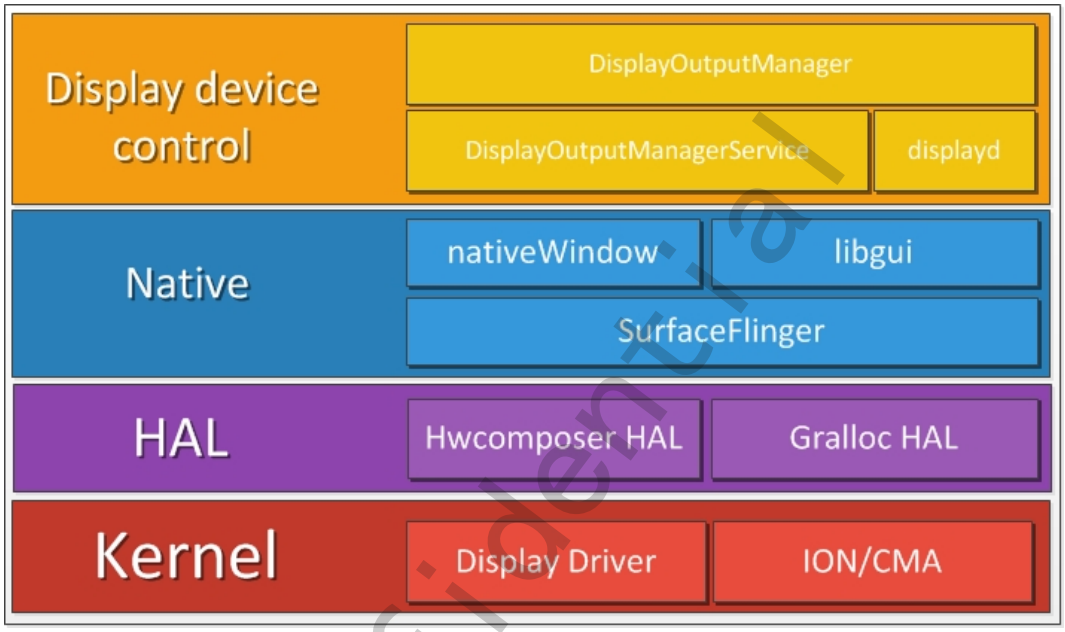

图 3

3.2 显示代码结构

1. display device control
frameworks/base/core/java/android/os/DisplayOutputManager.java
frameworks/base/core/java/android/os/IDisplayOutputManager.aidl
frameworks/base/services/core/java/com/android/server/Display-OutputManagerService.java
hardware/aw/displayd
2. native
frameworks/native/libs/gui/

frameworks/native/services/surfaceflinger/
3. hal
hardware/aw/hwc/
4. driver
lichee/linux-3.10/driver/video/sunxi/disp2/

4. 音频

4.1 音频系统架构

我司音频系统架构图如下所示：

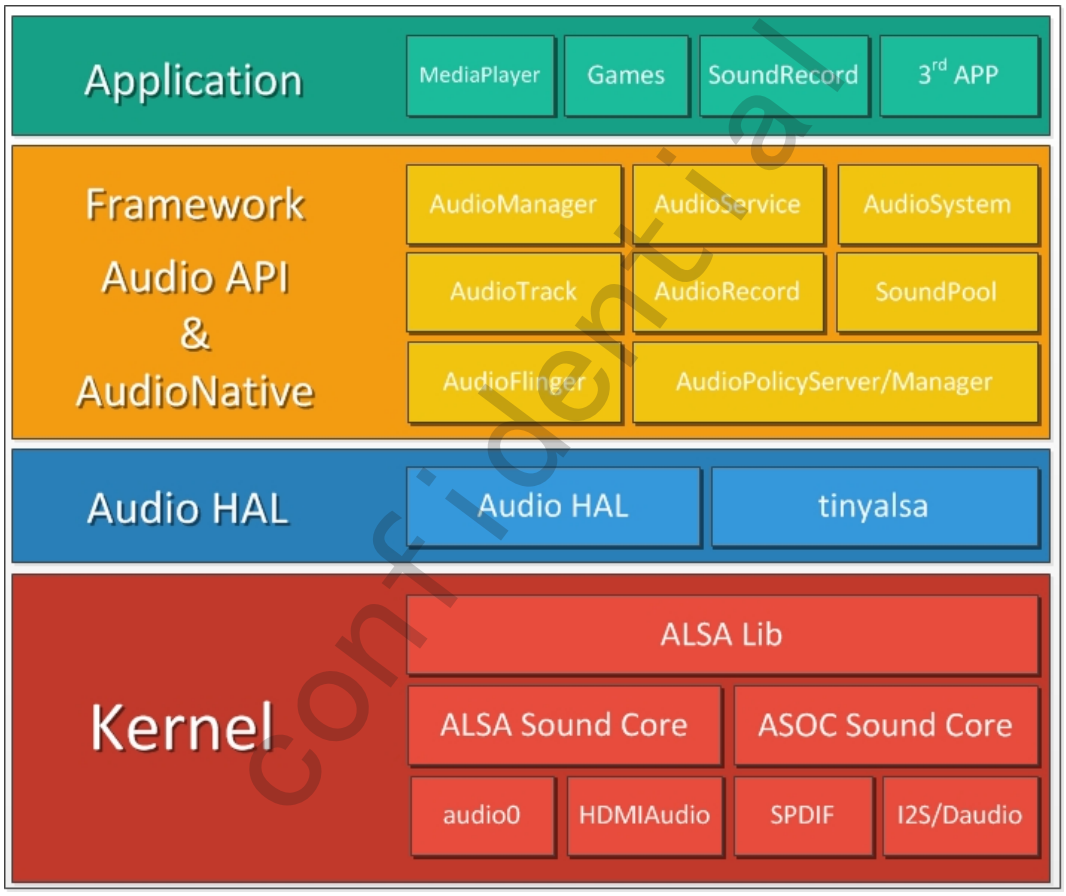

图 4: 音频架构图

4.2 音频代码结构

application:
android/packages/apps/settings/
1.SoundSettings.java ---音频系统设置
2.AudioChannelsSelect.java ---音频系统设置响应

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接口
2.AudioService.java ---音频系统服务，音量调节、音量UI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/device/softwinner/common/addons/frameworks/audio
4.AudioDeviceManagerObserver.java ---监听音频输入输出设备的热插拔
5.AudioManagerPolicy.java ---音频输入输出设备热插拔后的策略
6.AudioManagerEx.java ---AudioManager的扩展接口，实现透传、单、多通路输出等功能

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
android/hardware/aw/audio

5. Camera

5.1 Camera 系统架构

我司 Camera 架构图如下图所示

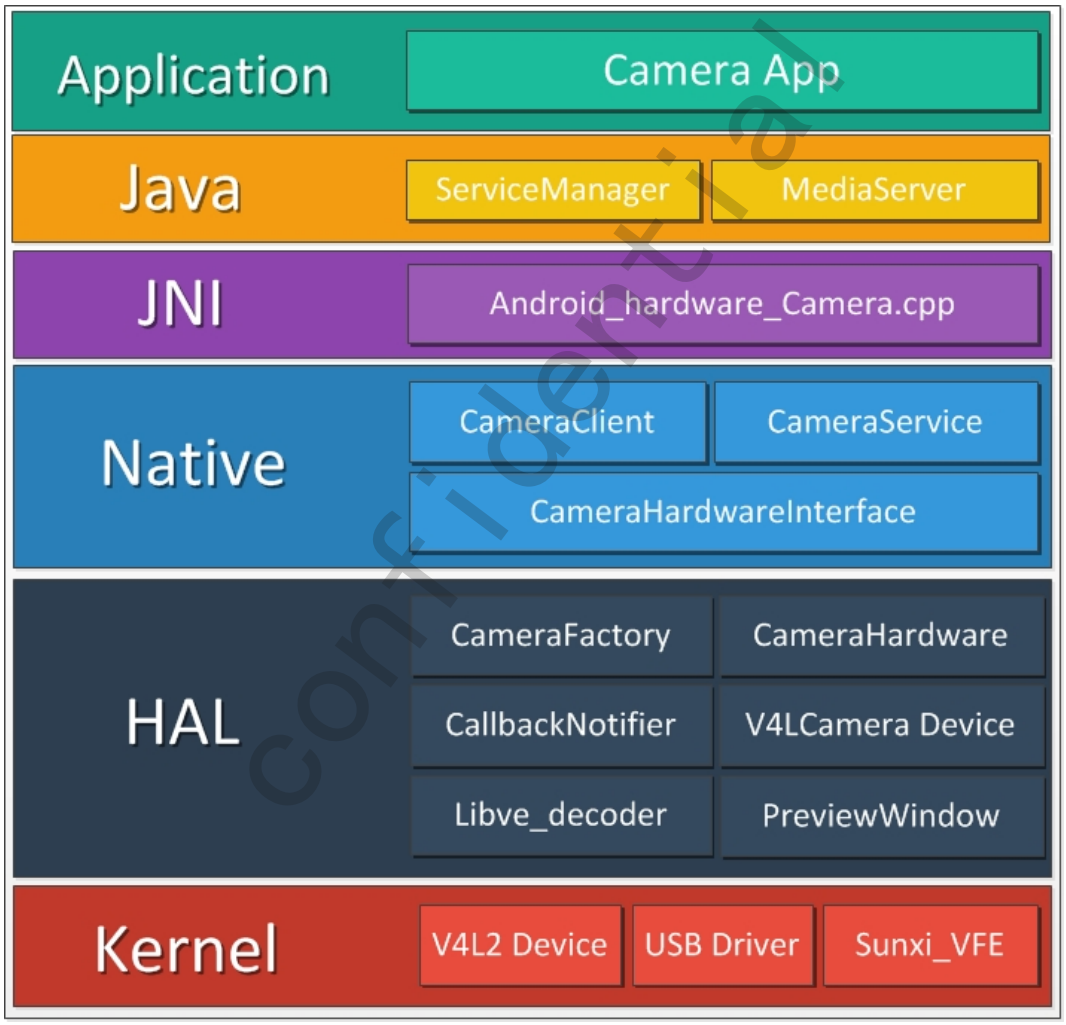

图 5: Camera 架构图

5.2 Camera 代码结构

5.2.1 JAVA 应用层

Android N 实现的 Camera 如下路径：android\packages\apps\Camera2\ 区别于 Android 4.4 平台，Android N 的 Camera Demo 使用的 API 接口是 Camera 2，原先的 Camera Java 层接口已经废除。Camera 2 接口在下面路径中完成定义：Android/frameworks/base/core/java/android/hardware/camera2/ 其中，主要使用的文件是 Android/frameworks/base/core/java/android/hardware/camera2/CameraManager.java

5.2.2 Android 本地框架层

在 Camera 2 中，我们仍然使用 Camera HAL 1.0，在代码位置以及具体使用的文件上和 Android 4.4 没有太大区别，具体是使用以下的文件：

```
android_hardware_Camera.cpp(android\frameworks\base\core\jni)
Camera.cpp (android\frameworks\av\camera)
CameraService.cpp(frameworks\av\services\camera\libcameraservice)
CameraClient.cpp(frameworks\av\services\camera\libcameraservice\api)
```

5.2.3 Android 硬件抽象层接口

同样的，对于硬件抽象层使用的接口文件依然不变：

```
CameraHardwareInterface.h(frameworks\av\services\camera\libcameraservice\device1)
```

5.2.4 Android 硬件抽象层

Camera HAL 层代码位置变更到以下路径，由于使用的是 Camera HAL 1.0，使用到的具体代码文件和 Android4.4 没有区别，代码路径如下：

(android\hardware\aw\camera\)

6. 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。

6.1 OTA/Recovery 代码结构

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
```

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

3. 厂商适配目录结构

└—— hardware/aw/lib/libboot
负责在OTA升级过程中烧写boot0,uboot分区。
└—— bootable/recovery/ir_keycode.cpp
负责在recovery模式支持遥控操作
└—— bootable/usb.cpp
└—— bootable/recovery/multi_device.cpp
负责在recovery模式进行OTA升级中支持从U盘等外部设备读取OTA更新包功能
└—— bootable/recovery/md5.cpp
提供计算md5值方法

7. 网络 WIFI

7.1 网络 WIFI 框架

我司网络 WIFI 框架图如下所示

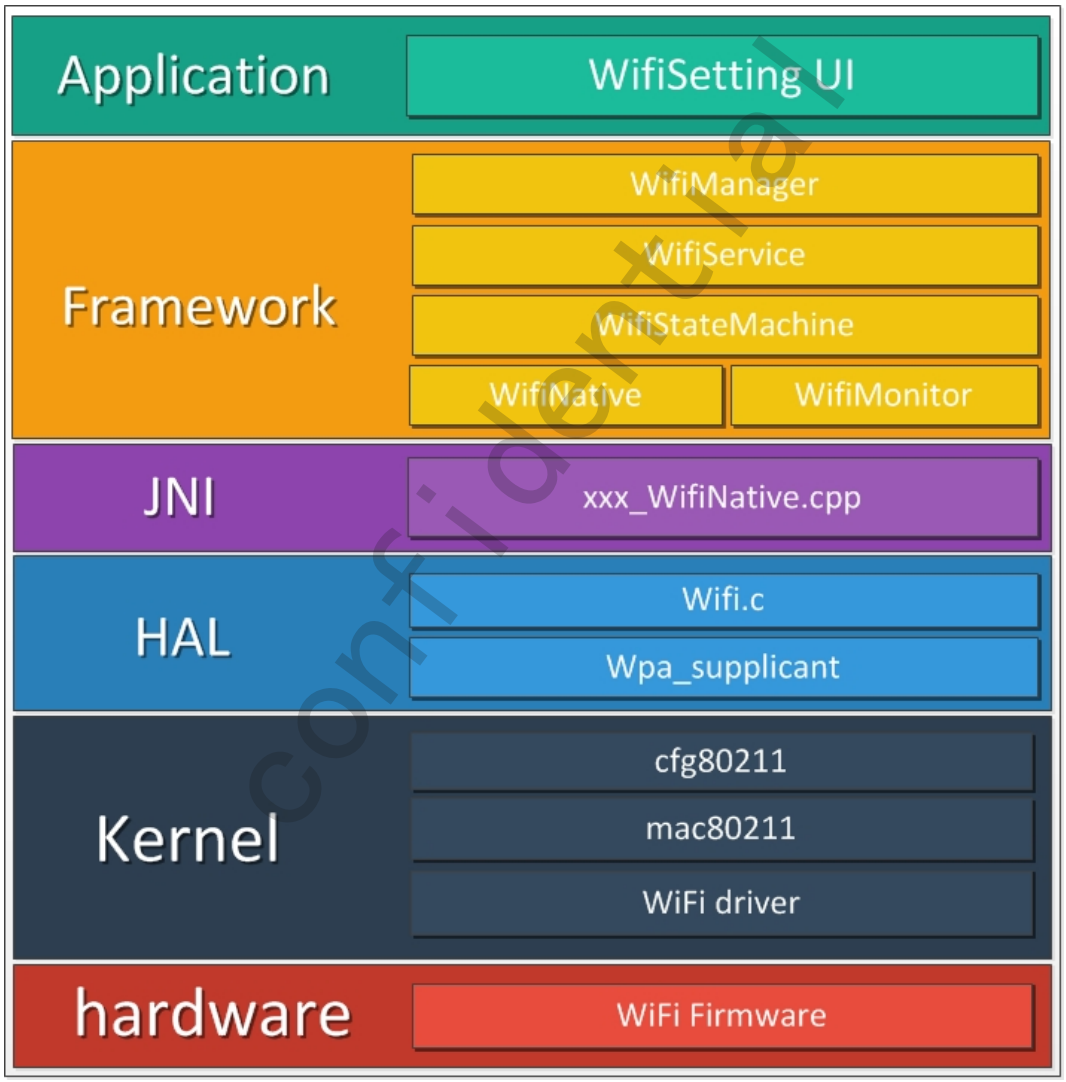

图 6: 网络 WIFI 框架图

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；

7.2 网络 WIFI 代码结构

层次	路径
APP	packages/apps/Settings/src/com/android/settings/wifi
framework	frameworks/base/wifi
framework	frameworks/base/opt/net/wifi
jni	frameworks/base/opt/net/wifi/service/jni/com_android_server_wifi_WifiNative.cpp
hal	hardware/libhardware_legacy/wifi
wpa_supplicant	external/wpa_supplicant_8
driver	linux-3.10/driver/net/wireless
firmware	hardware/aw/wlan/firmware

8. 网络 Ethernet

8.1 有线网络系统框架概述

有线网络上网方式包括 DHCP、静态 IP 以及 PPPoE，整体的框图如下：

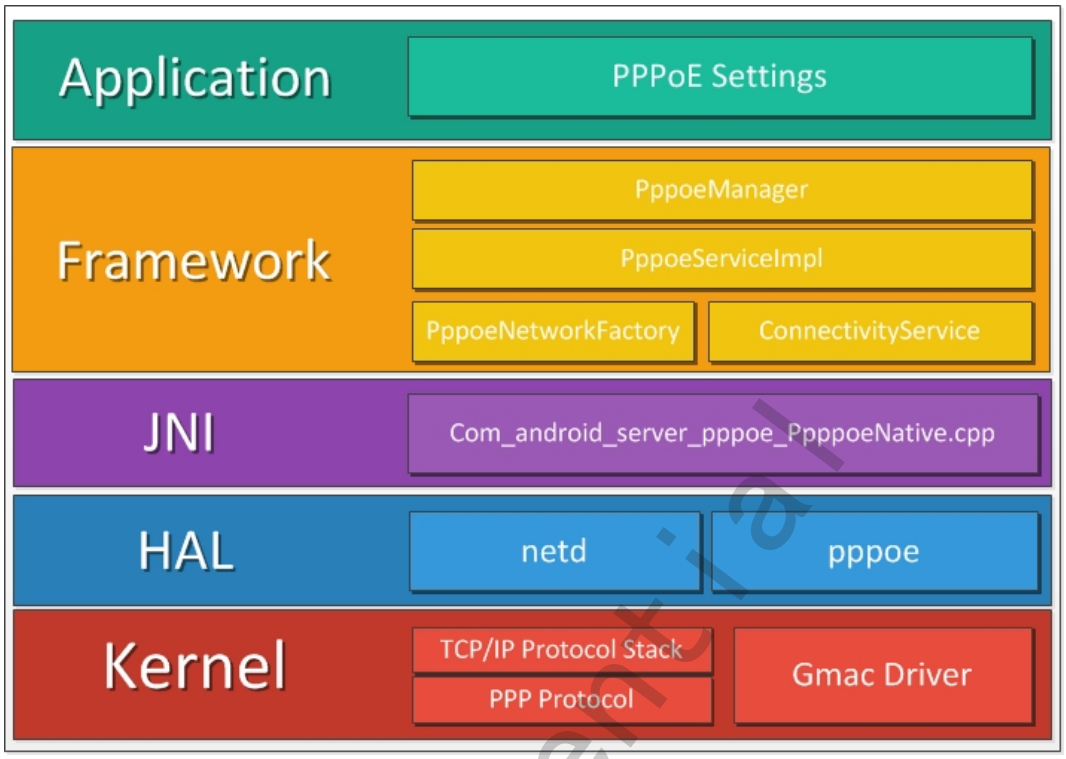

图 7: 有线网络框图

AndroidN PPPoE 从大体结构上可以分为六个层次：Application、Framework、JNI、HAL、Kernel、Hardware。

Application 包括 PPPoE 设置 apk(如 TvdSettings.apk)。

Framework 层中的 PppoeManager 类为应用层提供编程 API 接口，PppoeManager 通过 AIDL 与 PppoeService 关联，API 接口方法最终在 PppoeServiceImpl 类中实现；PPPoE 框架实现了一个 PppoeNetworkFactoy 类，该类是由 PppoeServiceImpl 类创建，负责 PPPoE 的打开关闭、处理网口插拔事件、向应用层发送广播通知 PPPoE 状态变化等，该类会向 ConnectivityService 注册 mNetworkAgent 来更新 dns、路由等信息；ConnectivityService 和 PppoeNetworkFactory 都会透过 NetworkManagerService 来对网口进行操作。

JNI 层主要是 com_android_server_pppoe_PppoeNative 类，该类与 HAL 层的 pppoe 模块交互。

HAL 层包括 netd、pppoe 两个部分。pppoe 模块主要负责 PPPoE 拨号、PPPoE 链路维护；netd 则直接透过 netlink 机制与内核交互，主要功能有：接收网口 up/down 消息、网线 link in/out 消息、网口 add/remove 消息、配置网口 IP 地址、更新网口 DNS、添加删除路由

表等。

Kernel 层中与网络相关的包括 TCP/IP 网络协议栈，Gmac 驱动；网络协议栈是网络报文接收与发送的必经之路，报文经过协议栈的传输后到达 Gmac 驱动，Gmac 驱动负责与具体的网口硬件交互。

Hardware 层包括以太网控制器和 PHY 两部分，负责在硬件上传输比特流。

8.2 有线网络模块设计

8.2.1 有线网络源码分布图

AndroidN 有线网络源码分布如下图所示：

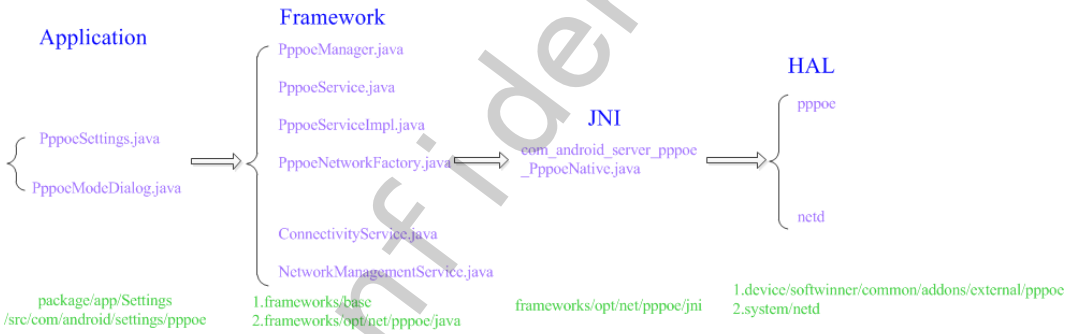

图 8: androidN 有线网络源码分布图

PPPoE 设置的源码位于 package/app/Settings/src/com/android/settings/pppoe 目录下，其中 PppoeModeDialog.java 负责 PPPoE 账号密码的输入，PppoeSettings.java 负责 PPPoE 的打开、关闭、以及网络信息的显示。

Framework 层的源码主要位于 frameworks/base 与 frameworks/opt/net/pppoe 目录中, 其中 PppoeManager.java 向应用层提供 API，具体 API 的实现由 PppoeServiceImpl 类完成，该类主要与 PppoeNetworkFactory 进行交互，而 PppoeNetworkFactory 是实现 Pppoe 拨号功能的核心类，其主要作用如下：

1、负责应用层网络连接/断开的具体实现；
2、其内部类InterfaceObserver继承自BaseNetworkObserver类，用以捕获网络状态的改变，比如网线的插拔、网口的生成等；
3、通过networkAgent机制通知ConnectivityService网络状态的变化；

JNI 层主要是 com_android_server_pppoe_PppoeNative 类，主要负责 PPPoE 的打开关闭，以及获取 PPPoE 状态。

HAL 层包括 netd、pppoe 两个部分。netd 透过 netlink 机制直接与 Kernel 交互，主要功能有：接收网口 up/down 消息、网线 link in/out 消息、网口 add/remove 消息、配置网口 IP 地址、更新网口 DNS、添加删除路由表等；pppoe 模块主要负责 PPPoE 拨号、PPPoE 链路维护。

9. 蓝牙

9.1 蓝牙系统框架图

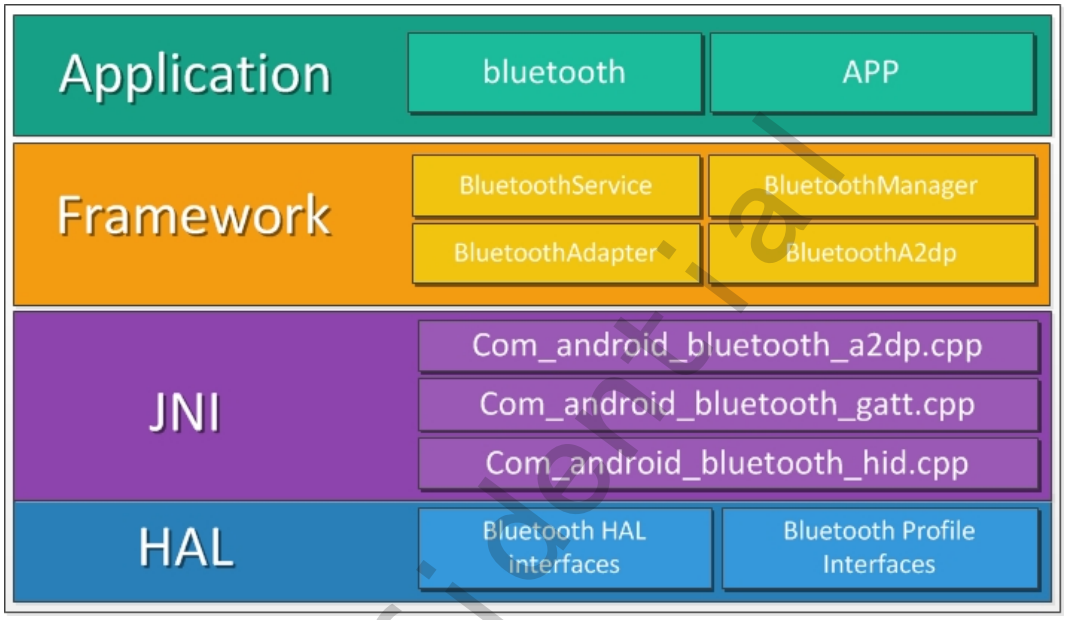

图 9: 蓝牙系统架构图

9.2 蓝牙模块代码目录

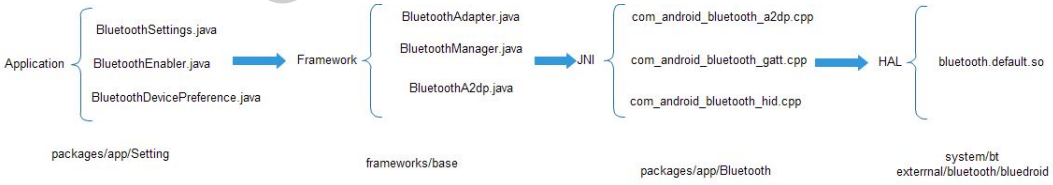

图 10: 蓝牙源码结构图

1. 模块涉及的 AOSP 原生代码目录

└—— system/bt
Android蓝牙协议栈。
└—— hardware/broadcom/libbt
Broadcom蓝牙模组vendor lib。
└—— 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

└—— system/bt
Android蓝牙协议栈，蓝牙协议的实现。
└—— hardware/broadcom/libbt
Broadcom蓝牙模组vendor lib，负责打开串口，模块上、下电，加载固件等。
└—— hardware/aw/bluetooth/libaw
蓝牙自适应库，定义支持的蓝牙模组型号。
└—— hardware/aw/bluetooth/bluedroid-usb
USB蓝牙dongle协议栈。
└—— hardware/aw/bluetooth/libbt-usb
USB蓝牙dongle vendor lib。
└—— device/softwinner/common/rtkbt
Realtek蓝牙模组支持代码，包括vendor lib和蓝牙协议栈。
└—— device/softwinner/petrel-p1
该目录里的init.rc文件定义了蓝牙模组在SOC上用到的串口号，比如ttyS1。

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 gg 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

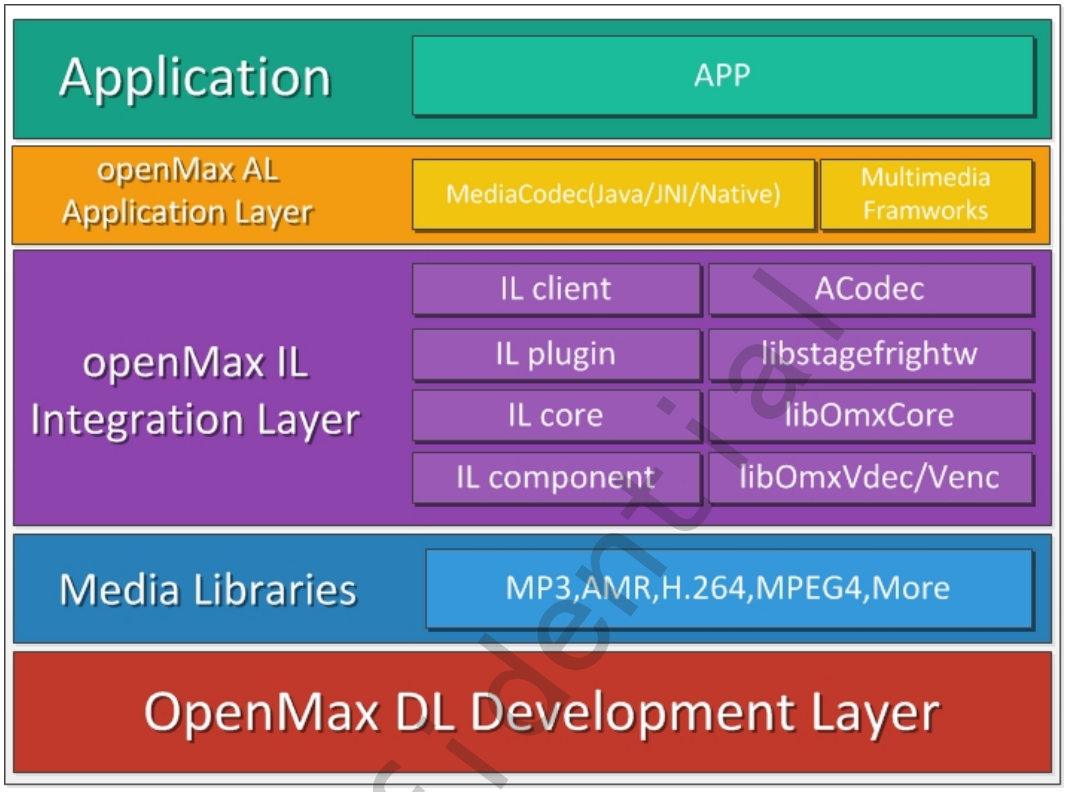

图 11: OMX 架构图

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
```


12. 产测工具

12.1 DragonBox

12.1.1 DragonBox 功能与工具介绍

该工具使用于工厂，用于测试机器是否能正常工作。当机器出厂前，都需要使用该工具测试运行，过滤明显的不良机器。应用可于二次开发扩展需要的测试项。

12.1.2 DragonBox 代码目录

```
├── assets：DragonBox的参考配置文件和的一些视频、声音文件。
├── platform：放置不同平台的功能接口代码，提高应用兼容性。
├── res：放置应用配置文件
└── src
    └── com
        └── softwinner
            └── dragonbox
                ├── config：解析配置文件，生成测试用例
                ├── entity：测试用例涉及的实体类
                ├── manager：封装系统接口，进行功能测试
                ├── platform：一些接口类
                ├── testcase：包含所有的测试用例
                ├── utils：使用到的工具类
                └── view：显示列表用到的数据适配器
```

12.2 DragonAging

12.2.1 DragonAging 代码目录

```
├── assets：参考用的配置文件和老化视频文件。
├── libs：用到的系统api静态库
├── res：资源文件
└── src
    └── com
        └── softwinner
            └── agingdragonbox：与应用整体设计框架相关的代码
                ├── engine：与测试用例相关的代码
                │   └── testcase：所有的测试用例
                └── xml：用于解析配置文件
```

12.3 DragonSN

12.3.1 DragonSN 简介

DragonSN 为私有数据烧写工具，该数据一般为厂商使用，存放固定内容，数据存放在存储器（nand 或 emmc）的 private 分区和 secure storage，机器出厂后用户无法修改该内容。主要的数据内容为 mac、sn、IMEI 等机器配对的信息。

存放内容为 key：value 即键值对，使用专用的接口可读写该内容。

本软件以 apk 应用形式来操作相应内容。因此，需要机器启动后才能使用。

本文中的默认需求为需要向机器写入 (SN,PN,IMEI,TID,EMAC,WMAC)，服务器中需要获取结果的字段为,CodeBurningResult，TestResult。

12.3.2 DragonSN 代码目录

```
├── assets：放置参考配置文件
├── DragonSNJni：往Secure Storage和private Storage烧入数据用到的JNI库
├── jtds-1.3.1.jar：数据库驱动
├── res：资源文件
└── src
    └── com
        └── allwinnertech
            └── dragonsn
                ├── BurnManager.java：从数据库获取数据进行烧录
                ├── Config.java：解析配置文件
                ├── DragonSNActivity.java：界面Activity
                ├── entity：与烧录信息相关的实体类
                ├── jni：提供调用jni库的接口
                ├── RemoteDBUtils.java：连接数据库
                └── view：用于显示列表的数据适配器
```

```
└── AllCast
    └── AllCast.apk
    └── Android.mk
        └── allwinnertech
└── Android.mk
└── MiracastReceiver
    └── Android.mk
    └── MiracastReceiver.apk
    └── lib
        └── arm
            └── libjni_WFDisplay.so
            └── libwfdmanager.so
            └── libwfdplayer.so
            └── libwfdrtsp.so
```



```
└—— libwfdutils.so
```


13. Declaration

This document is the original work and copyrighted property of Allwinner Technology (“Allwinner”). Reproduction in whole or in part must obtain the written approval of Allwinner and give clear acknowledgement to the copyright owner. The information furnished by Allwinner is believed to be accurate and reliable. Allwinner reserves the right to make changes in circuit design and/or specifications at any time without notice. Allwinner does not assume any responsibility and liability for its use. Nor for any infringements of patents or other rights of the third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Allwinner. This datasheet neither states nor implies warranty of any kind, including fitness for any particular application.tates nor implies warranty of any kind, including fitness for any particular application.

