**Android HAL**

Android Hal 是用来隔离linux内核和Android用户空间的一个中间库，它主要是为了让一些硬件厂商不用根据linux内核的gpl协议放出全部的linux驱动代码。有了hal硬件厂商就可以只在内核驱动中实现基本的操作而把硬件操作的逻辑移到hal层来实现，hal层是apache协议的，所以厂商可以只提供二进制库。

**0.HAL Arch**

hal的调用流程如下：

app -> framework -> jni -> runtime lib -> libhardware -> hal stub -> hal imp -> kernel driver

**1.HAL implementations**

Android中HAL的实现有两种方式：

* framework -> jni -> device driver，直接通过jni操作硬件驱动
* framework -> jni -> libhardware -> hal stub -> device driver，通过hal stub操作硬件驱动

第一种方式通过jni直接操作硬件驱动，但这样做可移植性很差，因为没有一个统一的中间api，framework调用的接口和硬件驱动接口都是可变的。

第二种方式则通过hal stub来给上层提供一个统一的调用接口，只要在hal中实现这些接口，用户层的代码就可以不用做太大的改变。

**2.HAL api**

新的stubbed hal的实现和它提供的api(hardware/libhardware)

**2.0.头文件**

include/hardware

**2.1.结构体**

* struct hw\_module\_t;从libhardware加载的动态库中获取的模块结构体
* struct hw\_module\_methods\_t;模块操作方法，打开模块，获取设备结构体
* struct hw\_device\_t; 设备以及操作方法

**2.2.主要接口**

int hw\_get\_module(const char \*id, const struct hw\_module\_t \*\*module);

* id: HAL module id
* \*module: HAL module operations

**2.3.实现**

这个接口的主要工作是通过id和系统的设置获取到硬件共享库的路径，然后通过dlopen打开共享库并通过一个固定的符号找到struct hw\_module\_t在内存中的地址，然后再通过该结构体提供的方法来打开设备并操作设备。

**3.Lights service分析**

light service是使用hal stub的一个典型例子，它的java service通过jni调用hal stub来操作驱动给用户空间提供的接口来实现硬件操作的。

**3.0.lights service的实现**

com\_android\_server\_LightsService.cpp

* static light\_device\_t\* get\_device(hw\_module\_t\* module, char const\* name);打开设备模块
* static jint init\_native(JNIEnv \*env, jobject clazz);获取模块
* static void setLight\_native(JNIEnv \*env, jobject clazz, int ptr, int light, int colorARGB, int flashMode, int onMS, int offMS, int brightnessMode) 设置light的颜色和亮度等参数

hardware/lights.h

* struct light\_device\_t;light设备结构体，其中有设置light的函数指针
* int (\*set\_light)(struct light\_device\_t\* dev, struct light\_state\_t const\* state); 设置light的函数指针

device/htc/passion-common/liblights/lights.c

* struct led\_prop; led属性, filename指led属性在sysfs中的路径，fd指向打开的文件描述符 struct led\_prop { const char \*filename; int fd;};
* static int open\_lights(const struct hw\_module\_t\* module, char const\* name, struct hw\_device\_t\*\* device) lights模块的打开函数实现，使用名字来设置不同类型的light

<https://blog.csdn.net/Luoshengyang/article/details/6567257>

一. 在Android内核源代码工程中编写硬件驱动程序。

二. 在Android系统中增加C可执行程序来访问硬件驱动程序。

三. 在Android硬件抽象层增加接口模块访问硬件驱动程序。

四. 在Android系统中编写JNI方法在应用程序框架层提供Java接口访问硬件。

五. 在Android系统的应用程序框架层增加硬件服务接口。

六. 在Android系统中编写APP通过应用程序框架层访问硬件服务。

framework目录



**android10.0的init执行顺序**

on early-init

on init

on late-init

on early-fs

on fs

on post-fs

on late-fs

class\_start early\_hal

on post-fs-data

load\_persist\_props\_action

zygote-start

firmware\_mounts\_complete

on early-boot

on boot

class\_start hal

class\_start core

on trigger\_xxxx

class\_start main

class\_start late\_start

on trigger\_restart\_framework

class\_start\_post\_data hal

class\_start\_post\_data core

class\_start main

class\_start late\_start

on trigger\_shutdown\_framework

class\_reset late\_start

class\_reset main

class\_reset\_post\_data core

class\_reset\_post\_data hal