

# Android Camera(流程架构介绍)

文档修订记录

版本编号	日期	变更人	简要说明
V1.0	2015-11-10	扶焰	初版整理
V1.1	2015-12-16	扶焰	部分内容更新

# 目录

前 言 .....	3
1. Camera 提供的业务.....	3
1.1 基本业务 .....	3
1.2 一般业务 .....	3
1.3 附加业务 .....	4
2. Android Camera 层次接口及代码路径 .....	4
2.1 层次接口 .....	4
2.2 代码路径.....	5
2.2.1 Camera 的 JAVA 应用程序代码路径.....	5
2.2.2 Camera 的 JAVA 框架代码路径 + .....	5
2.2.3 Camera 的 JAVA 本地调用部分 (JNI) * .....	6
2.2.4 Camera 本地框架 .....	6
2.2.5 Camera 服务部分 .....	6
2.2.6 MediaServer.....	6
2.2.7 Camera HAL.....	6
2.2.8 Camera tuning.....	7
2.2.9 Camera Driver.....	7
3. Android Camera Framework.....	7
3.1 Android Camera Java 框架 (应用层序接口/SDK api) .....	8
3.2 Camera 入口.....	9
3.2.1 Android Camera java.....	9
3.2.2 Android Camera JNI.....	10
3.3 Android Camera Native & CameraService .....	11
3.4 Camera 升级.....	13
4. Android Camera HardWare Layer .....	13
4.1 HAL 层的说明 .....	13
4.2 HAL 层的架构 .....	14
4.1.1 Qualcomm Camera HardWare (8X).....	15
4.1.2 MTK Camera HardWare (6580) .....	15
4.2 Camera moudle load .....	16
5. Linux Camera Driver.....	17
5.1 Camera Module 主要组成部分 .....	17
5.1.1 Sensor 感光原理.....	18
5.1.2 ISP (image signal processor) .....	18
5.1.3 Camera 硬件原理.....	19
5.2 Qualcomm 8X Driver (V4L2) .....	21
5.3 MTK Camera Driver.....	22
5.3.1 Driver 配置.....	22
5.3.2 Imgsensor 具体分析 .....	23
5.4 Camera HAL 业务实现 .....	24
5.4.1 Qualcomm (8X) .....	24

5.4.2 MTK 6580.....	26
6. 问题回顾.....	27

## 前 言

本文主要基于 Android 系统介绍 Camera 的基本架构以及业务实现,从 Camera 提供的业务到底层都有所涉及,而从 HAL 层开始还对 Qcom 平台和 MTK 平台进行了部分对比,希望能帮助刚接触 Camera 模块的朋友迅速掌握相关的技术。

## 1. Camera 提供的业务

### 1.1 基本业务

Camera 最基本的业务应该包含取景器 (Preview) 和拍摄照片 (Capture)。实际上业务是对开发者来讲的,对于不同的层次业务是不一样的,一般来讲底层提供的服务是为顶层服务的。因此各个层次提供的业务也是不一样的,比如说视屏通话 (App) 同样会用到 Camera Driver 的部分业务,但不一定会用到 HAL、Service 及以上的业务。同样也有可能 Camera Driver 提供的一些业务, Camera HAL 及以上也可能用不到,如为 Debug 提供的一些接口。

这里我们重点以 Camera App 出发介绍下 Android Camera 提供的业务。

### 1.2 一般业务

```
Open
release
getNumberOfCameras
getCameraInfo
setPreviewDisplay
startPreview
stopPrevie
autoFocus
takePicture
setParameters
getParameters
setShutterCallback
setPreviewCallback
setOneShotPreviewCallback
setPreviewCallbackWithBuffer
addCallbackBuffer
startFaceDetection
.....
```

```
interface
    AutoFocusCallback
    ShutterCallback
    PictureCallback
    PreviewCallback
    OnZoomChangeListener
    FaceDetectionListener
    ErrorCallback
    .....
```

android.hardware.Camera.Parameters 包装的对 para 的获取与设置  
eg: setPreviewSize 、 getPreviewSize

### 1.3 附加业务

系统开发者还会增加一些业务：  
如 NEC 加了

```
CameraDataCallback  
CameraMetaDataCallback  
setFaceDetectionCb  
.....  
等接口。
```

一般地这些厂商增加的接口都会是隐藏的接口。

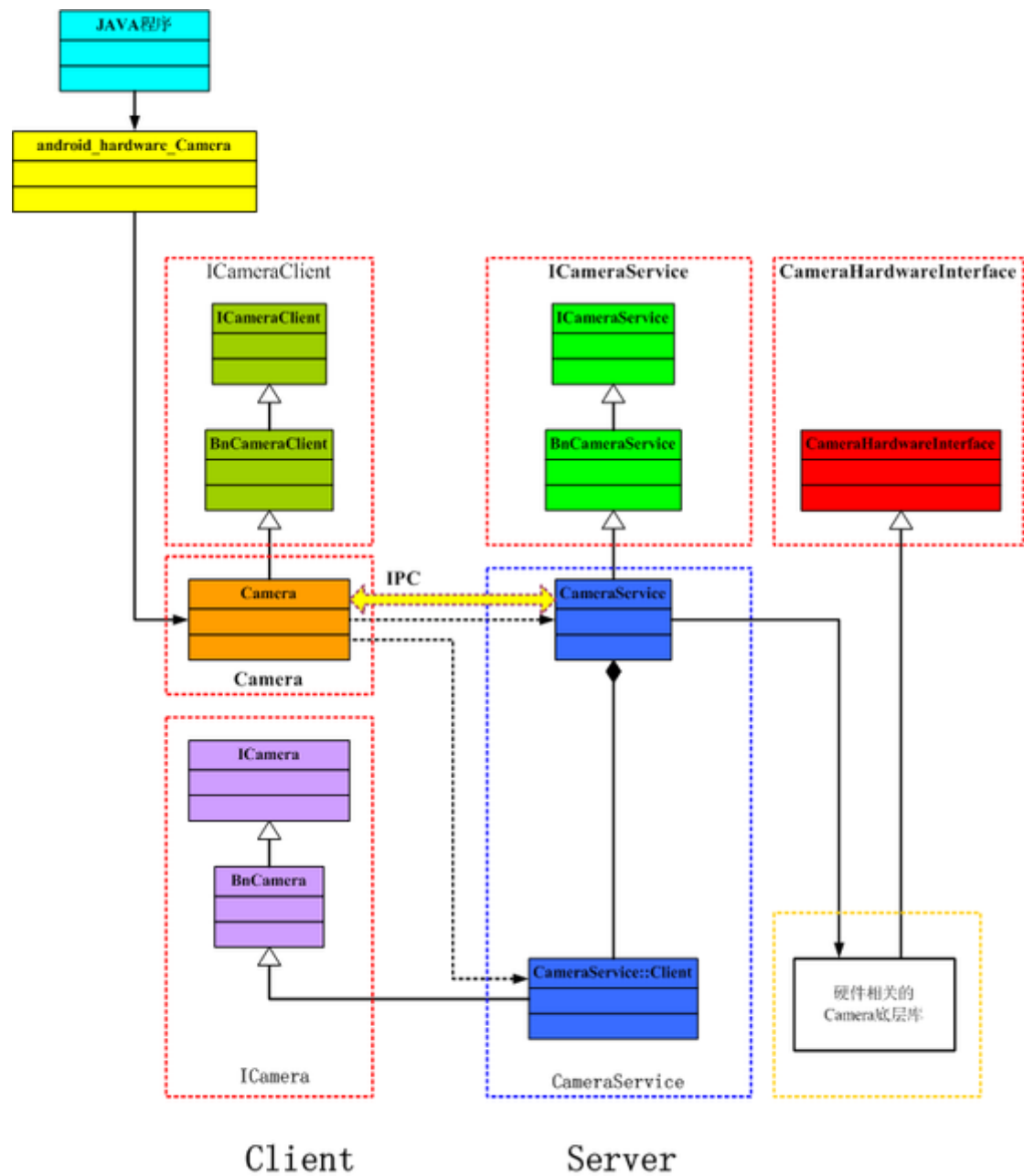
```
/** @hide  
    *  
*/
```

当然还有一些业务是附带在现有业务上实现的，如通过 parameter 传到一些指令，通过参数扩展传达一些指令等。

对于提供给应用开发者的一些业务，一部分在 Camera Native 就实现了，一部分在 service 实现，所以不一定以下每一层都会去实现这些业务。如 ShutterSound，在 CameraService 就实现了，当然也有底层的支持但不是 Camera 模块的。

## 2. Android Camera 层次接口及代码路径

### 2.1 层次接口



## 2.2 代码路径

### 2.2.1 Camera 的 JAVA 应用程序代码路径

Android/packages/apps/Camera2/src/com/android/camera/

Camera 是一个独立的 apk，即在该目录下有一个 Android.mk 文件编译成为目标是 xxx.apk

### 2.2.2 Camera 的 JAVA 框架代码路径 +

android/frameworks/base/core/java/android/hardware/Camera.java

+ >=Android5.0: android/frameworks/base/core/java/android/hardware/Camera2/..

编译成为目标是 java class (android.hardware), 打包在 framwwork.jar。

### 2.2.3 Camera 的 JAVA 本地调用部分 (JNI) \*

android/frameworks/base/core/jni/android\_hardware\_Camera.cpp

编译成为目标是 libandroid\_runtime.so, JNI 公用库。

\* >=Android 5.0, camera api v2 并不使用 JNI 部分。

### 2.2.4 Camera 本地框架

主要的头文件在以下的目录中:

android/frameworks/av/include/camera

Camera 本地框架实现文件在以下的目录中:

android/frameworks/av/camera

这部分的内容被编译目标是:

libcamera\_client.so

### 2.2.5 Camera 服务部分

android/frameworks/av/services/camera/libcameraservice

编译目标是: libcameraservice.so

另外 Framework、HAL 各层有一个公共引用的头文件:

android/system/core/include/system/camera.h

### 2.2.6 MediaServer

android/frameworks/av/media/mediaserver

编译目标是可执行程序即 bin: mediaserver

### 2.2.7 Camera HAL

实现 Hal 框架头文件在以下的目录中:

android/frameworks/av/services/camera/libcameraservice/CameraHardwareInterface.h

android/hardware/libhardware/include/hardware/camera/ camera\_common.h

android/hardware/libhardware/include/hardware/camera/camera.h

android/hardware/libhardware/include/hardware/hardware.h

Qualcomm 8x

android/hardware/qcom/camera/

MTK MT6580

android/vendor/mediatek/proprietary/hardware/mtkcam

## 2.2.8 Camera tuning

Qualcomm 8X

android/vendor/qcom/proprietary/mm-camera

MTK MT6580

android/vendor/mediatek/proprietary/custom/mt65xx/hal

## 2.2.9 Camera Driver

Qualcomm 8X

Kernel CAM 模块编译配置:

arch/arm/configs/msm8960\_defconfig

drivers/media/video/msm/sensors/Makefile

drivers/media/video/msm/kconfig

上电: android/kernel/arch/arm/mach-msm/board-8064-regulator-oem.h

android/kernel/arch/arm/mach-msm/board-8064-camera.c

时钟: android/kernel/arch/arm/mach-msm/clock8960.c

GPIO: android/kernel/arch/arm/mach-msm/board-8064-camera.c

时序:

vendor/qcom/proprietary/mm-camera/mm-camera2/media-controller/modules/sensors/sensor\_libs/xx/xx\_lib.c

I2C: android/kernel/drivers/media/video/msm/io/msm\_camera\_i2c.c

具体驱动:

android/kernel/drivers/media/video/msm/sensors/imx135\_v4l2.c

android/kernel/drivers/media/video/msm/sensors/mt9m113\_v4l2.c

MTK MT6580

Kernel CAM 模块编译配置:

arch/arm/configs/your\_project\_defconfig.mk

具体驱动:

android/kernel/drivers/misc/mediatek/imgsensor

android/kernel/drivers/misc/mediatek/lens

android/kernel/drivers/misc/mediatek/flashlight


android/kernel/drivers/misc/mediatek/cameraisp/src/mt6580/camera\_isp.c


## 3. Android Camera Framework

3.1 Android Camera Java 框架（应用层序接口/SDK api）

对应用开发者来说，这一层是非常重要的。Android 之初并没有很多功能，随着 Android 的不断升级，会有更多的功能出现，即会提供更多 api。

应用开发者要怎么才能跟上 Android 升级的节奏呢？请看 Google 官网 API。

<http://developer.android.com/guide/components/index.html>

 Developers

Design

**Develop**

Distribute

Training   API Guides   **Reference**   Tools   Google Services   Samples

Android APIs   API level: 23

android.databinding

android.drm

android.gesture

android.graphics

android.graphics.drawable

android.graphics.drawable.shapes

android.graphics.pdf

**android.hardware**

android.hardware.camera2

android.hardware.camera2.params

android.hardware.display

package  
**android.hardware**  
  
Provides support for hardware features, such as the camera. To use these hardware features, so you should declare hardware capabilities in your manifest.  
  
**Interfaces**  

<b>Camera.AutoFocusCallback</b>	This interface was deprecated in API level 21. Use Camera.AutoFocusChangeListener instead.
---------------------------------	--

Interfaces	
<b>Camera.AutoFocusCallback</b>	This interface was deprecated in API level 21. Use Camera.AutoFocusChangeListener instead.
<b>Camera.AutoFocusMoveCallback</b>	This interface was deprecated in API level 21. Use Camera.AutoFocusChangeListener instead.
<b>Camera.ErrorCallback</b>	This interface was deprecated in API level 21. Use Camera.OnErrorListener instead.
<b>Camera.FaceDetectionListener</b>	This interface was deprecated in API level 21. Use Camera.FaceDetectionListener2 instead.
<b>Camera.OnZoomChangeListener</b>	This interface was deprecated in API level 21. Use Camera.OnZoomChangeListener2 instead.
<b>Camera.PictureCallback</b>	This interface was deprecated in API level 21. Use Camera.PictureCallback2 instead.
<b>Camera.PreviewCallback</b>	This interface was deprecated in API level 21. Use Camera.PreviewCallback2 instead.
<b>Camera.ShutterCallback</b>	This interface was deprecated in API level 21. Use Camera.ShutterCallback2 instead.
<b>SensorEventListener</b>	Used for receiving notifications from the SensorManager.
<b>SensorEventListener2</b>	Used for receiving a notification when a flush() operation completes.
<b>SensorListener</b>	This interface was deprecated in API level 3. Use SensorEventListener instead.



# Classes

<a href="#">Camera</a>	<i>This class was deprecated in API level 21. Use <a href="#">Camera2</a> for new applications.</i>
<a href="#">Camera.Area</a>	<i>This class was deprecated in API level 21. Use <a href="#">Camera2</a> for new applications.</i>
<a href="#">Camera.CameraInfo</a>	<i>This class was deprecated in API level 21. Use <a href="#">Camera2</a> for new applications.</i>
<a href="#">Camera.Face</a>	<i>This class was deprecated in API level 21. Use <a href="#">Camera2</a> for new applications.</i>
<a href="#">Camera.Parameters</a>	<i>This class was deprecated in API level 21. Use <a href="#">Camera2</a> for new applications.</i>
<a href="#">Camera.Size</a>	<i>This class was deprecated in API level 21. Use <a href="#">Camera2</a> for new applications.</i>
<a href="#">ConsumerIrManager</a>	Class that operates consumer infrared on the device.
<a href="#">ConsumerIrManager.CarrierFrequencyRange</a>	Represents a range of carrier frequencies (in kHz) for consumer infrared.
<a href="#">GeomagneticField</a>	Estimates magnetic field at a given point on the Earth.
<a href="#">Sensor</a>	Class representing a sensor.
<a href="#">SensorEvent</a>	This class represents a <a href="#">Sensor</a> event and contains the sensor's <a href="#">data</a> .
<a href="#">SensorManager</a>	SensorManager lets you access the device's sensors.
<a href="#">TriggerEvent</a>	This class represents a Trigger Event - the event that triggers a TriggerEventListener.
<a href="#">TriggerEventListener</a>	This class is the listener used to handle TriggerEvents.

## 3.2 Camera 入口

### 3.2.1 Android Camera java

```
Looper looper;
if ((looper = Looper.myLooper()) != null) {
    mHandler = new EventHandler(this, looper);
} else if ((looper = Looper.getMainLooper()) != null) {
    mHandler = new EventHandler(this, looper);
} else {
```

```

        mEventHandler = null;
    }

private static void postEventFromNative(Object camera_ref,
                                       int what, int arg1, int arg2, Object obj)
{
    Camera c = (Camera)((WeakReference)camera_ref).get();
    if (c == null)
        return;
    if (c.mEventHandler != null) {
        Message m = c.mEventHandler.obtainMessage(what, arg1, arg2, obj);
        c.mEventHandler.sendMessage(m); //由应用层调用处理回调的数据
    }
}

```

### 3.2.2 Android Camera JNI

```

static JNINativeMethod camMethods[] = {
    {"native_setup", "(Ljava/lang/Object;)V", (void*)android_hardware_Camera_native_s
    etup },
    {"native_release", "()V", (void*)android_hardware_Camera_release },
    {"setPreviewDisplay", "(Landroid/view/Surface;)V",
    (void *)android_hardware_Camera_setPreviewDisplay },
    {"startPreview", "()V", (void *)android_hardware_Camera_startPreview },
    {"stopPreview", "()V", (void *)android_hardware_Camera_stopPreview },
    .....
};

```

```

int register_android_hardware_Camera(JNIEnv *env)
{
    .....
    // Register native functions
    return AndroidRuntime::registerNativeMethods(env,
    "android/hardware/Camera",

        camMethods, NELEM(camMethods));
    .....
}

```

什么时候注册的?

- ❖ Android Camera JNI 这里需要注意些什么?
- ❖ Java <> C/C++ 这些代码跑在哪个进程?

### 3.3 Android Camera Native & CameraService

- ❖ 为什么将 Camera Native 与 CameraService 一起讲？
- ❖ 服务进程是什么时候启动的？
- ❖ 应用进程与服务进程是怎样通信的？
- ❖ 哪些代码包含在服务进程中？
- ❖ mediaserver 包含哪些服务内容？

为什么将这两部分合在一起分析呢，因为这两部分的实现涉及到 Android 中的一个重要部分进程间通信--Binder。

服务端的启动：

```
android/system/core/rootdir/ init.rc
service media /system/bin/mediaserver
    class main
    user media
android/frameworks/av/media/mediaserver/ main_mediaserver.cpp
int main(int argc, char** argv)
{
    signal(SIGPIPE, SIG_IGN);
    sp<ProcessState> proc(ProcessState::self());
    sp<IServiceManager> sm = defaultServiceManager();
    ALOGI("ServiceManager: %p", sm.get());
    AudioFlinger::instantiate();
    MediaPlayerService::instantiate();
    CameraService::instantiate();
    AudioPolicyService::instantiate();
    ProcessState::self()->startThreadPool();
    IPCThreadState::self()->joinThreadPool();
}
```

- ❖ CameraService::instantiate() 在哪？

```
class CameraService :
{
public:
    BinderService<CameraService>,
    public BnCameraService
{
    class Client;
friend class BinderService<CameraService>;
.....
}

template<typename SERVICE>
```

```

class BinderService
{
public:
    static status_t publish() {
        sp<IServiceManager> sm(defaultServiceManager());
        return sm->addService(String16(SERVICE::getServiceName()), new
SERVICE());
    }
    .....
static void instantiate() { publish(); }
    .....
};

```

客户服务端联系的建立:

open->native\_setup->connect

```

sp<Camera> Camera::connect()
{
    LOGV("connect");
    sp<Camera> c = new Camera();
    const sp<ICameraService>& cs = getCameraService();
    if (cs != 0) {
        c->mCamera = cs->connect(c);
    }
    .....
}

const sp<ICameraService>& Camera::getCameraService()
{
    Mutex::Autolock _l(mLock);
    if (mCameraService.get() == 0) {
        sp<IServiceManager> sm = defaultServiceManager();    //获取服务管家代理

```

在 Camera 通过 CameraService 和 Client 建立联系后, Camera 就可以直接调用 Client 的函数来实现自己的功能了, 对于 Camera 来说, 就像自己带着参数进入服务器端线程运行, 并带回结果, 这就是 Binder 机制的特点——线程迁移。这些调用都是与 binder 分不开的, 而且流程都是一样的。

- ❖ Binder 通信是阻塞还是非阻塞?
- ❖ Camera 应用进程 (Native) 与服务进程通信时单向? 双向?
- ❖ Bp Bn? 哪些在一个进程?

摘一段韩超的《Android 系统原理及开发要点详解》中的一段话来概括 Android Camera 的 FrameWork 架构：

Camera 系统处理的宏观逻辑是：上层通过调用控制接口来控制下层，并设置回调函数；下层通过回调函数向上层传递各种类型的数据。

Note:

我认为快速系统学习 Camra 模块的方法也就是搞清楚其控制流和数据流。

### 3.4 Camera 升级

- Google Android 升级
- eg: 4.0 升到 4.2
- 一般升级框架 (JAVA 接口, Camera 本地, CameraService, 甚至 BSP), 即高版本会比低版本多一些接口以实现更丰富的功能。
- 不大可能出现丢失低版本的接口, 也就是说低版本的 App 一定能在高版本的系统上运行。
  
- 2. 定制厂商升级
- eg: NEC 将扩展的 Camra 升级
- 一般移植是 feature, 即将扩展的 feature 移植过来, 可能只是 App 升级
- , 如 UI 的改善; 也可能是从上到下都会升级, 如 NEC 的人脸识别, fastboot 等。
  
- 1 和 2 的混合
- 如 ESP 到 LEF
- 我们大多时候会遇到这种情况

## 4. Android Camera HardWare Layer

### 4.1 HAL 层的说明

这是最复杂的一层, 而且各个芯片厂商实现不一样, 各个产品的开发也可以在这里大做文章。

实际上 BSP 准确的说就是这一层。一些与芯片密切相关的策略可能会放在这里即常说的

游离在 kernel 之外的 driver。这样反而 kernel 简单了，一是没有大量策略相关的东西；二是做得相对标准了各个芯片厂商区别也不是很大了。当然这里我将 Qualcomm 的 QualcommCameraHAL, mm\_camera, MTK 的 3A HAL, mhal\_camera, mdphal, ... 都叫做 HAL 了，因此涉及的代码相对是各个层次最多的，而且这一层中也分很多层。

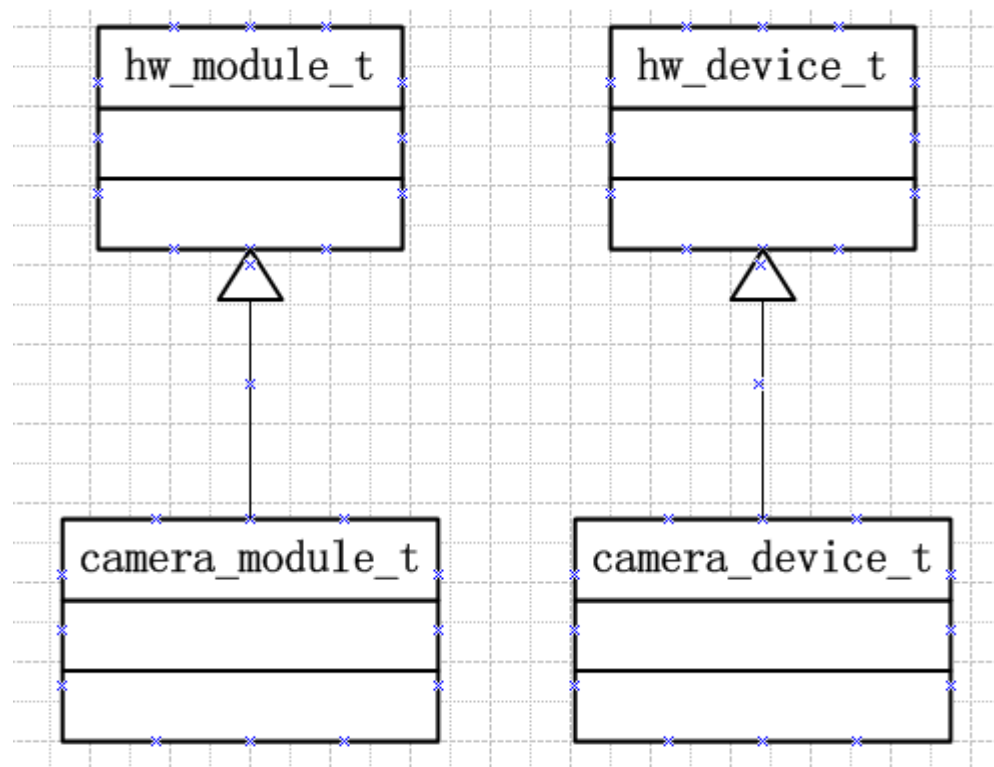
这一层大概会涉及与硬件相关的以下内容：

Imagesensor, isp, mdp, flashlight, lens...

这里多介绍 Android 框架的实现，对于具体的实现仅大概说明。

## 4.2 HAL 层的架构

- 新架构
- 新式的称之为 HAL stub；旧有的称之为 HAL module
- 实现步骤：
  - 继承 hw\_module\_t 与 hw\_device\_t
  - 定义 interface(stub operations)
  - 实现 interface
- Camera 现在也像其他模块一样采用新式的架构动态加载模块来实现，这个架构开始成为 Android 的标准了，现在几乎所有的与硬件相关的模块都通过这里连接，也就是说如果要再加一个硬件就要在这多加一个去实现 hardware stub。



#### 4.1.1 Qualcomm Camera HardWare( 8X)

```
typedef struct camera_module {
    hw_module_t common;
    int (*get_number_of_cameras)(void);
    int (*get_camera_info)(int camera_id, struct camera_info *info);
} camera_module_t;
```

```
typedef struct camera_device {
    hw_device_t common;
    camera_device_ops_t *ops;
    void *priv;
} camera_device_t;
```

```
typedef struct camera_device_ops {
    int (*set_preview_window)(struct camera_device *,
                             struct preview_stream_ops *window);
    .....
} camera_device_ops_t;
```

```
static hw_module_t camera_common = {
    .....
    id: CAMERA_HARDWARE_MODULE_ID,
    .....
};
camera_module_t HAL_MODULE_INFO_SYM = {
    .....
};
camera_device_ops_t camera_ops = {
    set_preview_window:    android::set_preview_window,
    .....
}
```

#### 4.1.2 MTK Camera HardWare( 6580)

```
mediatek/hardware/camera/device/CamDeviceManager/CamModule.cpp
static camera_module  instantiate_camera_module()
```

```

{
.....
    camera_module module = {
        common: {
            .....
            id:                CAMERA_HARDWARE_MODULE_ID,
            .....
            methods:           CamDeviceManager::get_module_methods(),
            .....
        },
        get_number_of_cameras: CamDeviceManager::get_number_of_cameras,
        get_camera_info:      CamDeviceManager::get_camera_info,
    };
    return module;
}

camera_module HAL_MODULE_INFO_SYM = instantiate_camera_module();

```

## 4.2 Camera module load

```

void CameraService::onFirstRef()
{
    BnCameraService::onFirstRef();
    if (hw_get_module(CAMERA_HARDWARE_MODULE_ID,
                     (const hw_module_t **)&mModule) < 0) {
        ALOGE("Could not load camera HAL module");
        mNumberOfCameras = 0;
    }
}

android/hardware/libhardware/ hardware.c
/** Base path of the hal modules */
#define HAL_LIBRARY_PATH1 "/system/lib/hw"
#define HAL_LIBRARY_PATH2 "/vendor/lib/hw"
#define HAL_LIBRARY_PATH3 "/system/lib"

static int load(const char *id, const char *path, const struct hw_module_t **pHmi)
{
    .....
    handle = dlopen(path, RTLD_NOW);
    .....
    hmi = (struct hw_module_t *)dlsym(handle, sym);
    .....
}

```



```
}
```

```
android/hardware/libhardware/include/hardware/camera/ camera_common.h  
#define CAMERA_HARDWARE_MODULE_ID "camera "  
int (*get_number_of_cameras)(void);  
int (*get_camera_info)(int camera_id, struct camera_info *info);
```

```
hardware/libhardware/include/hardware/camera/camera.h
```

其它方法需调用 module 的 open 后才能调用。

module 的 open 定义在 hardware/libhardware/include/hardware/hardware.h 中，对于 Qualcomm Camera 而言其实现是在 hardware/qcom/camera/QualcommCamera2.cpp 中，即实际上 QualcommCamera2.cpp 就是对上层的主要接口实现，CameraService 中调用 HAL 接口就直接在这个文件中去找对应关系

module open 的调用关系是：

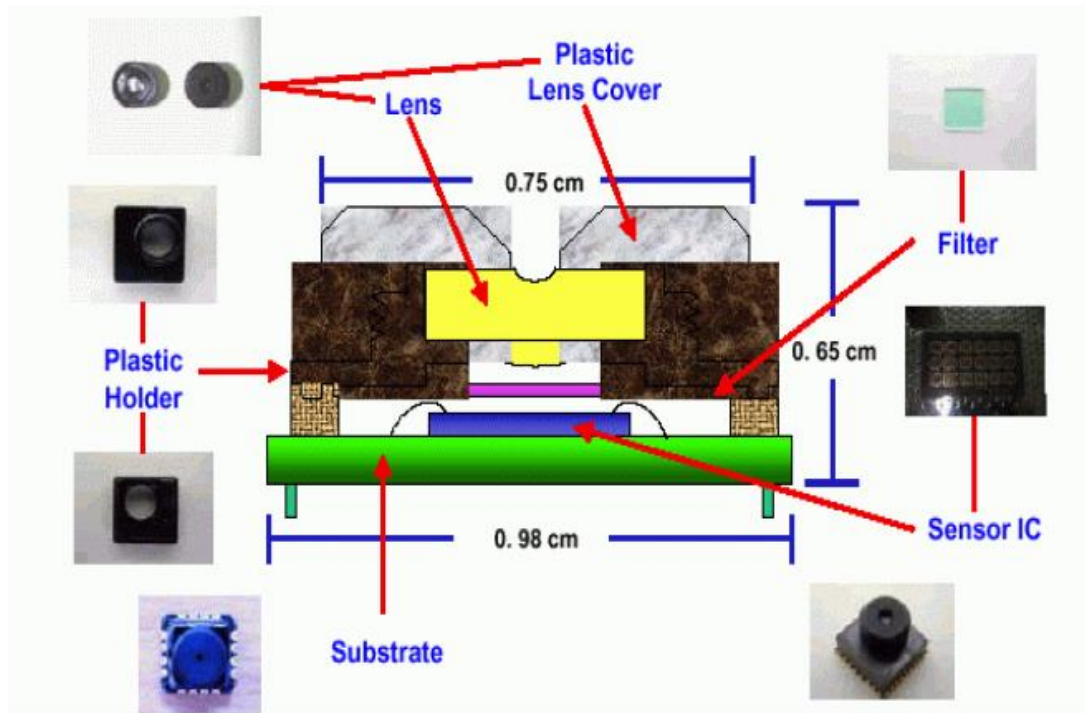
```
connect(cameraservice)->initialize(cameraclient)->initialize(camerahardwareinte  
rface)->open(QualcommCamera)
```

❖ Camera HAL 是在一个新的进程里吗？是，那是什么进程？不是，那它在哪个进程？

## 5. Linux Camera Driver

### 5.1 Camera Module 主要组成部分

Camera Module 主要的组成部分由：lens 和 Sensor IC，其中有些 Sensor IC 是集成了 DSP，有些是没有集成 DSP，没有集成 DSP 的 module 需要外部外挂 DSP。

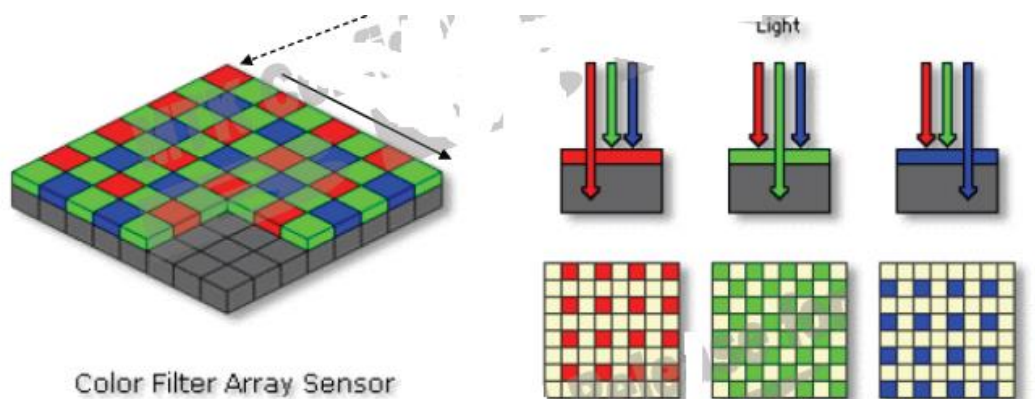


## Camera module structure anatomizing

### 5.1.1 Sensor 感光原理

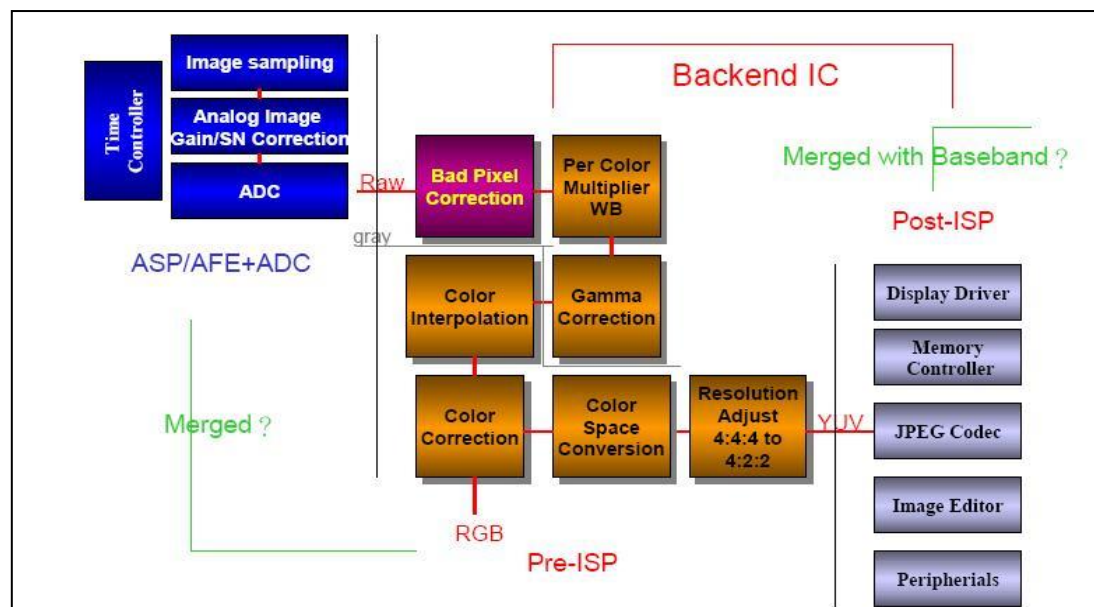
在摄像头的主要组件中，最重要的就是图像传感器了，因为感光器件对成像质量的重要性不言而喻。

Sensor 将从 lens 上传导过来的光线转换为电信号，再通过内部的 DA 转换为数字信号。由于 Sensor 的每个 pixel 只能感光 R 光或者 B 光或者 G 光，因此每个像素此时存贮的是单色的，我们称之为 RAW DATA 数据。



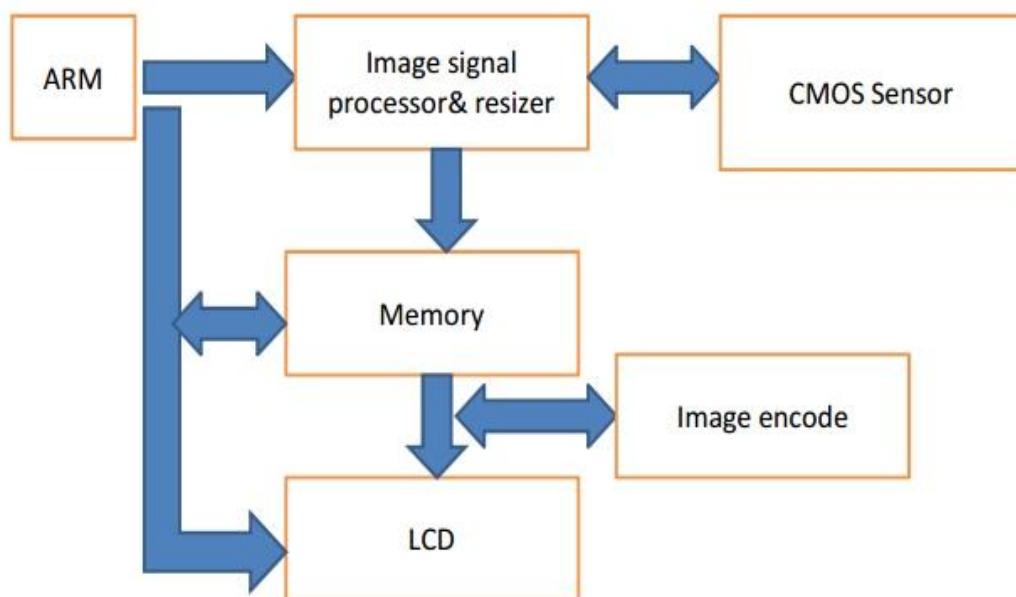
### 5.1.2 ISP (image signal processor)

如果集成了 DSP，则 RAW DATA 数据经过 AWB、color matrix、lens shading、gamma、sharpness、AE 和 de-noise 处理，后输出 YUV 或者 RGB 格式的数据；如果没有集成 DSP 则输出到后端处理。。

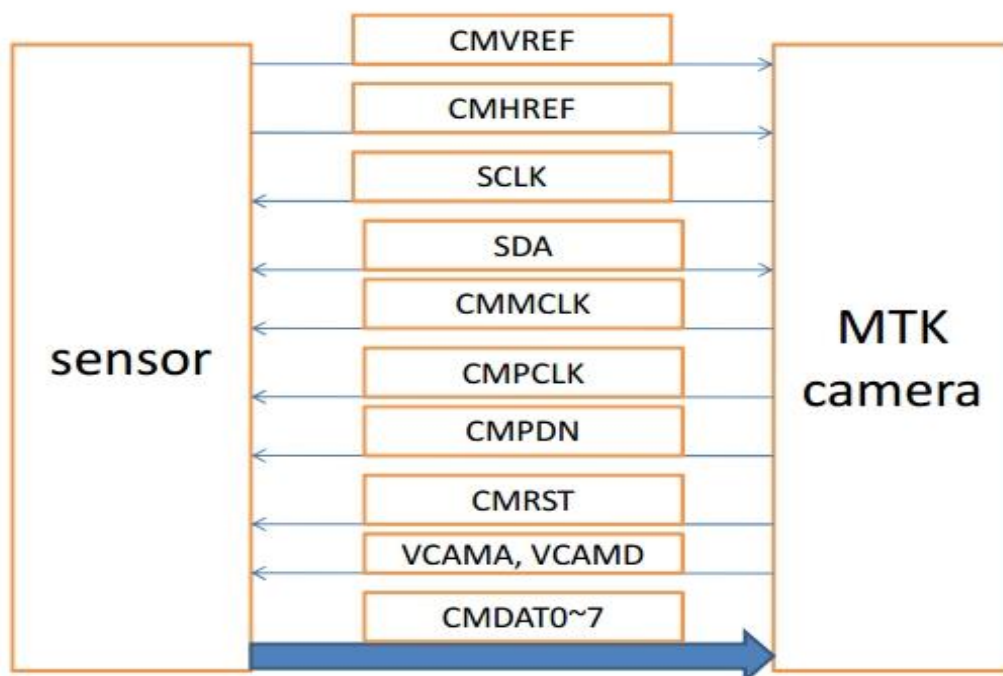


### 5.1.3 Camera 硬件原理

# Camera的硬件架构



## Camera 接口信号



Power

AVDD: 模拟电压

DOVDD:GPIO 数字电压

DVDD:核工作电压

OutPut (对 ap 而言)

PDN: power down

RESET: reset signal

XCLK: 工作时钟 (MCLK)

SCLK: IIC clock signal

SDA: IIC data signal

Input

DATA: 8-10 data bus

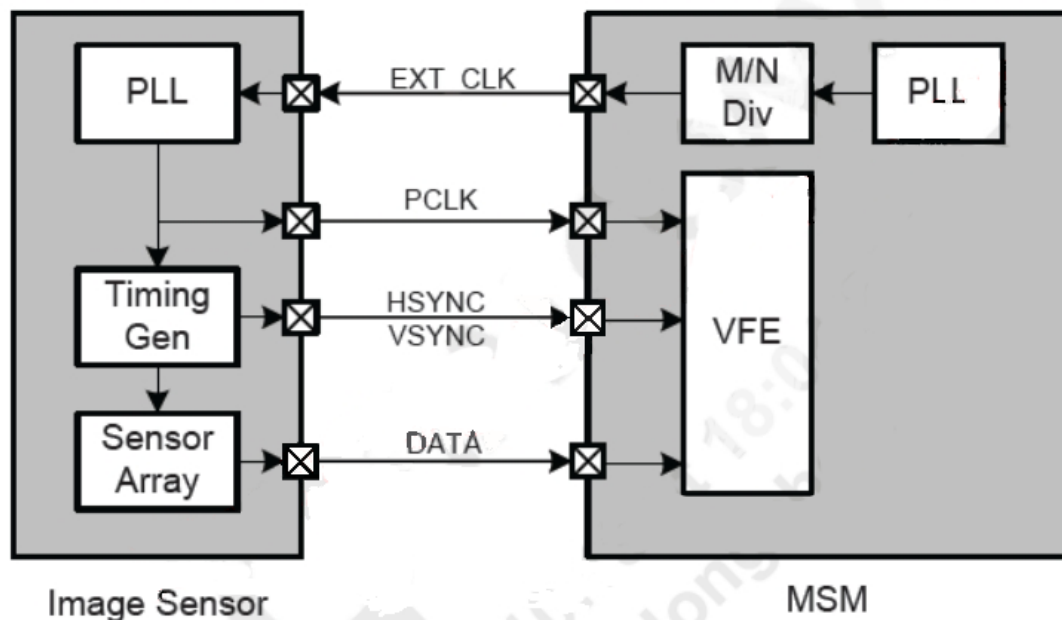
VSYNC: 帧同步 (VREF)

HSYNC: 行同步 (HREF)

PCLK: 像素同步

CSI: Camera Serial Interface DATA: CLK P/N 和 D0 P/N ~ D3 P/N

CPI: Camera Parallel Interface DATA:



## 5.2 Qualcomm 8X Driver(V4L2)

```
/kernel/drivers/media/video/msm/msm.c
/* register v4l2 video device to kernel as /dev/videoXX */
D("video_register_device\n");
rc = video_register_device(pvdev, VFL_TYPE_GRABBER,
msm_camera_v4l2_nr);
```

```

/* Now we really have to activate the camera */
D("%s: call mctl_open\n", __func__);
rc = pmctl->mctl_open(pmctl, MSM_APPS_ID_V4L2);

/* then sensor - move sub dev later */
rc = v4l2_subdev_call(p_mctl->sensor_sdev, core, s_power, 1);

android/kernel/drivers/media/video/msm/sensors/imx135_v4l2.c
static struct v4l2_subdev_core_ops imx135_subdev_core_ops = {
    .ioctl = msm_sensor_subdev_ioctl,
    .s_power = msm_sensor_power,
};

```

唤醒用户空间的 config

在 msm\_open 函数中最后会去

调用 msm\_send\_open\_server, 这个函数会去唤醒我们用户空间的 config 线程

```

if (pcam->use_count == 1) {
    rc = msm_send_open_server(pcam);
    .....

```

```

android/vendor/qcom/proprietary/mm-camera/server/core/qcamsvr.c
qcamsvr_process_server_node_event

```

❖ mm-camera 是新的进程吗?

```

/vendor/qcom/proprietary/mm-camera/apps/appslib
/system/bin/mm-qcamera-daemon

```

## 5.3 MTK Camera Driver

### 5.3.1 Driver 配置

#### 1. mkconfig

```

android/device/XXXX/XXXX/ProjectConfig.mk
CUSTOM_KERNEL_MAIN_IMGSENSOR = ov8825_mipi_raw
CUSTOM_KERNEL_SUB_IMGSENSOR = s5k8aayx_mipi_yuv

```

那 imagesensor 具体驱动就在

```

mediatek/custom/common/kernel/imgsensor/ov8825_mipi_raw
mediatek/custom/common/kernel/imgsensor/s5k8aayx_mipi_yuv

```

#### 2. 上电、时钟、GPIO、时序:

```

android/kernel/drivers/misc/mediatek/imgsensor/src/mt6580/camera_hw/kd_camera_h
w.h
//FIXME, should defined in DCT tool

```

```
//
#ifdef GPIO_CAMERA_LDO_EN_PIN
#define GPIO_CAMERA_LDO_EN_PIN GPIO94
#endif
//
#ifdef GPIO_CAMERA_CM_RST_PIN
#define GPIO_CAMERA_CM_RST_PIN GPIO211
#endif

android/kernel/drivers/misc/mediatek/imgsensor/src/mt6580/camera_hw/kd_camera_h
w.h
int kdCISModulePowerOn(CAMERA_DUAL_CAMERA_SENSOR_ENUM SensorIdx, char
*currSensorName, BOOL On, char* mode_name)
{
    ...
}

EXPORT_SYMBOL(kdCISModulePowerOn);
```

### 5.3.2 Imgsensor 具体分析

Imgsensor 作为一个内核的模块，是配置成启动时加载  
mediatek/custom/common/kernel/imgsensor/src/kd\_sensorlist.c

```
module_init(CAMERA_HW_i2C_init);
module_exit(CAMERA_HW_i2C_exit);

MODULE_DESCRIPTION("CAMERA_HW driver");
MODULE_AUTHOR("xxxx <xxx@xxx.com>");
MODULE_LICENSE("GPL");

static int __init CAMERA_HW_i2C_init(void)
{
    .....
    i2c_register_board_info(SUPPORT_I2C_BUS_NUM1, &i2c_devs1, 1);
    .....
    if(platform_driver_register(&g_stCAMERA_HW_Driver)) {
        PK_ERR("failed to register CAMERA_HW driver\n");
        return -ENODEV;
    }
}
```

```

.....
//Register proc file for main sensor register debug
prEntry = create_proc_entry("driver/camsensor", 0, NULL);
}

static int CAMERA_HW_probe(struct platform_device *pdev)
{
    return i2c_add_driver(&CAMERA_HW_i2c_driver);
}

static int CAMERA_HW_i2c_probe(struct i2c_client *client, const struct
i2c_device_id *id)
{
    .....
    //Register char driver
    i4RetVal = RegisterCAMERA_HWCharDrv();
    .....
}

static const struct file_operations g_stCAMERA_HW_fops =
{
    .owner = THIS_MODULE,
    .open = CAMERA_HW_Open,
    .release = CAMERA_HW_Release,
    .unlocked_ioctl = CAMERA_HW_Ioctl
};

```

Falshlight、lens 等驱动与 imagesensor 大致一样都是字符设备驱动。

## 5.4 Camera HAL 业务实现

这里有很多策略、算法实现，很多人也把这部分看做 Driver。

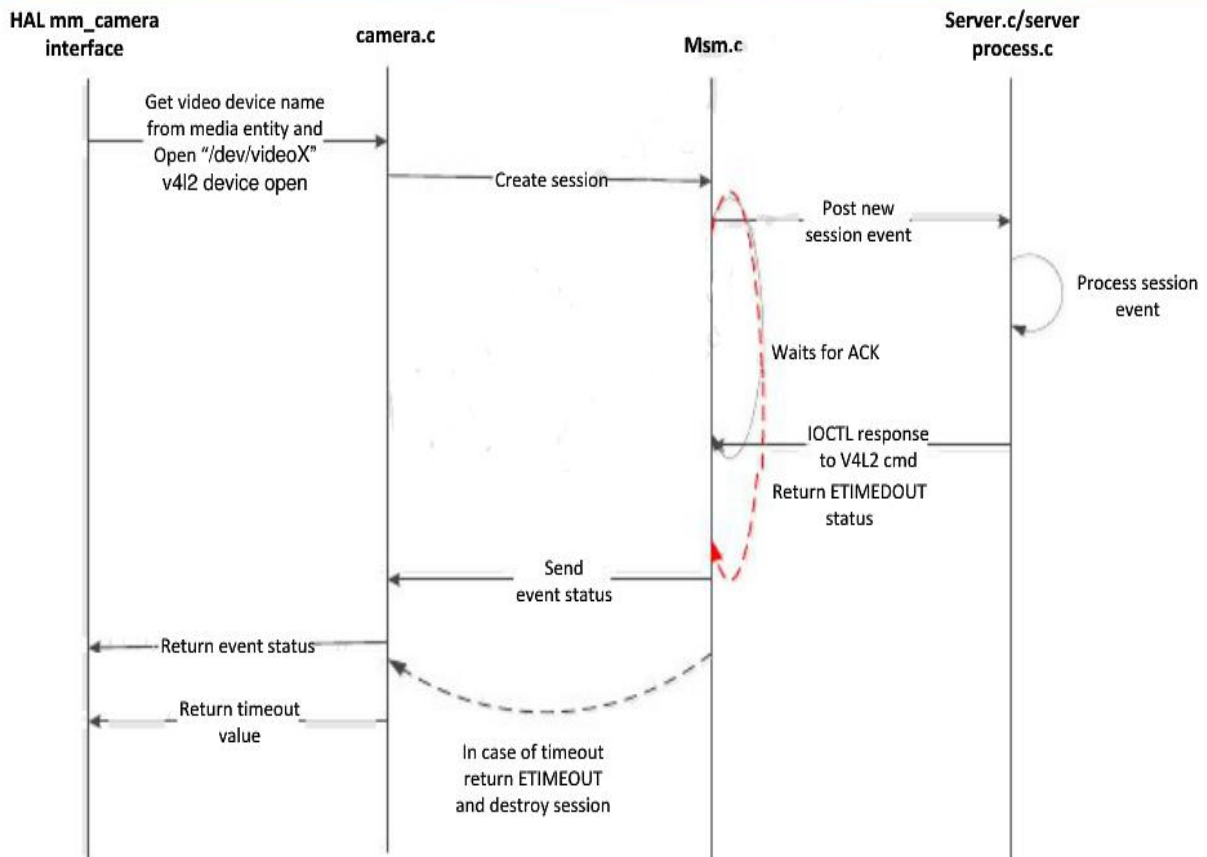
❖ Driver 不一定在 kernel 里， why?

### 5.4.1 Qualcomm (8X)

Hal 层以下 Camera Open 图：



## Call Flow – Open Device Node



初始化 (Open) :

```
/hardware/qcom/camera/ QualcommCamera2.cpp
new QCameraHardwareInterface(cameraId, mode);
```

```
/hardware/qcom/camera/QCameraHWI.cpp
调用对象 QCameraHardwareInterface 的构造函数
```

```
/* Open camera stack! */
mCameraHandle=camera_open(mCameraId, &mem_hooks);
```

```
android/hardware/qcom/camera/QCamera/stack/mm-camera-interface/src/
mm_camera_interface.c
rc = mm_camera_open(cam_obj);
```

```
android/hardware/qcom/camera/QCamera/stack/mm-camera-interface/src/mm_camera.c
my_obj->ctrl_fd = open(dev_name, O_RDWR | O_NONBLOCK);
```

Open 系统调用将打开驱动的设备节点。

到此为止，与 kernel 的 driver 已经打通了，即可以通过 ioctl 控制 Driver 了。

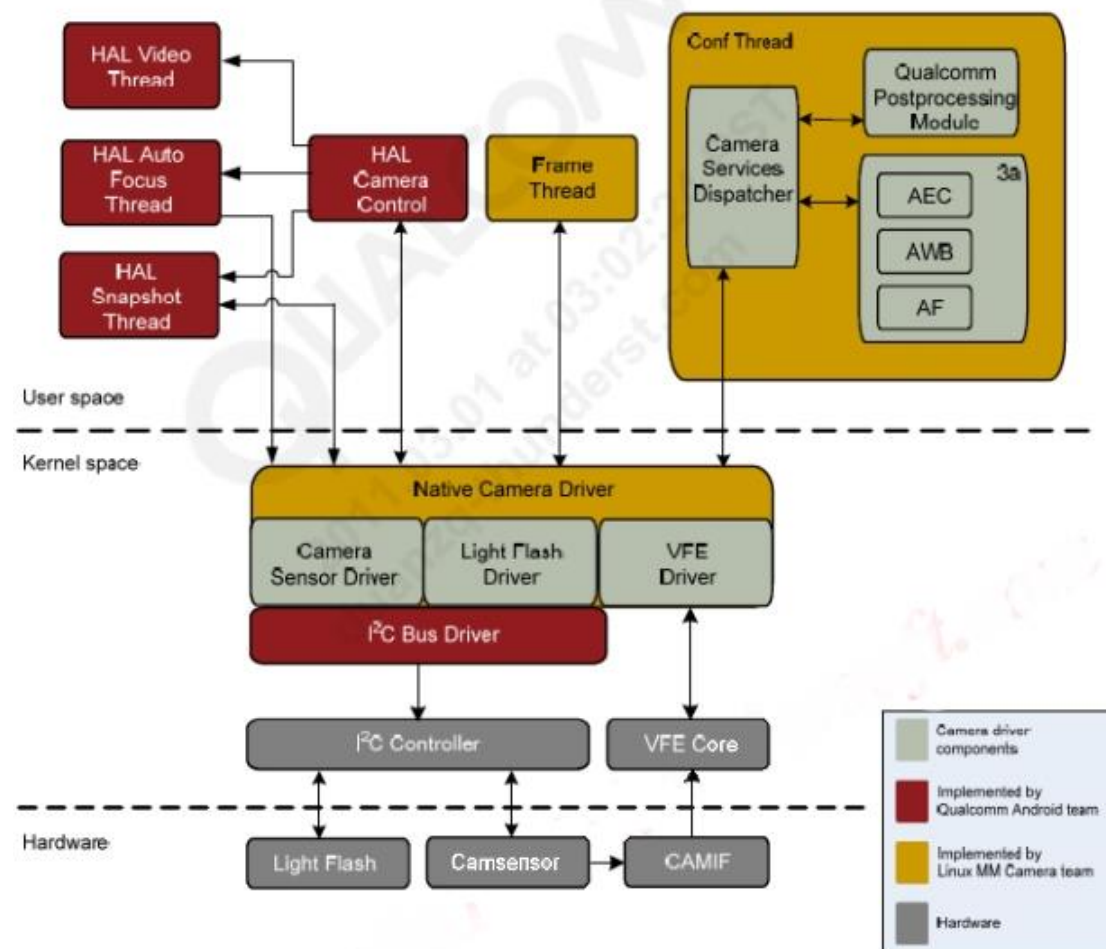
## 5.4.2 MTK 6580

vendor/mediatek/proprietary/hardware/mtkcam/legacy/platform/mt6580/hal/sensor/i  
mgsensor\_drv.cpp

MINT32

```
ImgSensorDrv::init(
    MINT32 sensorIdx
)
{
    .....
    if (mUsers == 0) {
        if (m_fdSensor == -1) {
            sprintf(cBuf, "/dev/%s", CAMERA_HW_DEVNAME);
            m_fdSensor = ::open(cBuf, O_RDWR);
            .....
        }
    }
```

其它 flashilight、lens、isp 等都是在这一层直接调用相关 HAL 控制驱动。  
与 Qualcomm 8X 不同的是 MTK 的 3A 与 HAL 跑在同一个进程。



## 6. 问题回顾

Binder 通信是阻塞还是非阻塞？

答：binder 是同步通信，需要等待应答。所以是阻塞的。

Driver 不一定在 kernel 里， why？

答： 在我们的印象中 driver 就是控制硬件模块， 让其能正常跑起来。

其实只要 kernel 中提供了接口， 我们也可以在用户空间实现 driver。

高通 camera 就是只需要在用户空间按照一定规范填写 bring up 时序和相应的寄存器值，当然 kernel 中也需要填写一下基本的配置信息。

这样就大幅减少了 kernel 中的代码量， 这样即方便了 driver 的调试， 也保护了核心代码

那么应用开发者要怎么才能跟上 Android 升级的节奏呢？

答： 实时更新 sdk 等。

Camera HAL 是在一个新的进程里吗？ 是， 那是什么进程？ 不是， 那它在哪个进程？

答： 不是一个新的进程， 就工作在 mediaserver 进程。

mm-camera 是新的进程吗？

答： 主要是 qcom 的一个守护进程。

Jni 什么时候注册？

答： Java 程序调用 System.loadLibrary() 时， JNI\_OnLoad() 函数被调用执行， 用于向 JavaVM 注册 JNI 函数等。 在本例中首先通过参数 JavaVM (Java 虚拟机指针) 获取当前应用程序所在的线程， 即： JNIEnv。 再通过调用 android::AndroidRuntime::registerNativeMethods() 注册 native 实现的函数指针。

CameraService::instantiate() 在哪？

答： mediaserver 启动时调用。

Camera 应用进程 (Native) 与服务进程通信时单向？ 双向？

答： 双向， 底层也可以向应用回调数据。

为什么将 Camera Native 与 CameraService 一起讲？

答： 因为 Camera Native 方法大部分都跟 CameraService 有关。

服务进程是什么时候启动的？

答： 开机启动 mediaserver 是启动。

应用进程与服务进程是怎样通信的？

答： 通过 binder 进行通信。



Thank you