Android 10 Camera
开发指南

版本号: 1.1
发布日期: 2020.8.27

版本历史

版本号	日期	制/修订人	内容描述
1.1	2020.8.27	AW1052	初始版本

目　录

插　图

1 概述

1.1 编写目的

本文旨在全面介绍 Android 10 Camera 的开发要点，重点在 Framework，HAL 层以及常用的调试技巧，可作为快速学习 Camera 相关知识的重要学习资料。后续将会补充驱动等 Camera 知识。

1.2 适用范围

本文档适用于 Android 10 系统开发。

1.3 相关人员

Camera 相关的 HAL，驱动以及应用开发人员。

2 Camera 模块介绍

2.1 硬件结构

Camera 模块的硬件层次如下图所示，被拍摄的景物通过镜头 (lens) 将生成的光学图像投射到传感器 (sensor) 上。传感器的作用是将采集到的光信号转换为电信号，并通过模数转换 (A/D) 转化为数字信号。图像的数字信号通过 CSI 接口存储到 DRAM 中。然后 ISP(Image Signal Processor) 对图像进行包括噪音消除，暗黑补偿，3A 处理，色彩空间变换等一系列处理后生成 YUV 图像，最后可以送到 VE 编码器进行图像视频的编码，或者送到 DE 显示引擎进行显示。

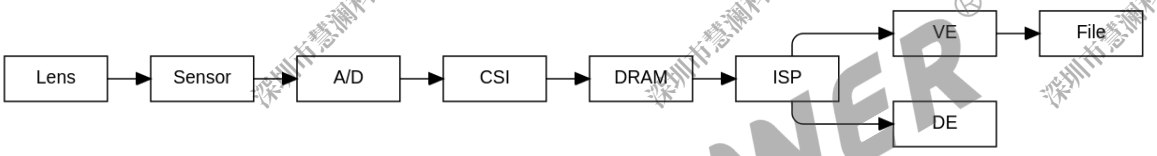

图 2-1: Camera 硬件结构图

2.2 软件层次

- 应用框架应用代码位于应用框架级别，它使用 Camera 2 API 与相机硬件进行交互。在内部，这些代码会调用相应的 Binder 接口，以访问与相机互动的 Native 代码。SDK 中 Camera2 代码位于 android/packages/apps/Camera2。
- Framework/AIDL 框架与 CameraService 关联的 Binder 接口可在 frameworks/av/camera/aidl/android/hardware 中找到。生成的代码会调用较低级别的 Native 代码以获取对实体相机的访问权限，并返回用于在 Framework 级别创建 CameraDevice，并最终创建 CameraCaptureSession 对象的数据。
- Native 框架此框架位于 frameworks/av/ 中，并提供相当于 CameraDevice 和 CameraCaptureSession 类的原生类。
- Binder IPC 接口 IPC binder 接口用于实现跨越进程边界的通信。调用相机服务的若干个相机 Binder 类位于 frameworks/av/camera/camera/aidl/android/hardware 目录中。ICameraService 是相机服务的接口；ICameraDeviceUser 是已打开的特定相机设备的接口；ICameraServiceListener 和 ICameraDeviceCallbacks 分别是对应用框架的 CameraService 和 CameraDevice 回调。
- 相机服务位于 frameworks/av/services/camera/libcameraservice/CameraService.cpp 下的相机服务是与 HAL 进行互动的实际代码。

- HAL 硬件抽象层定义了由相机服务调用、且必须实现以确保相机硬件正常运行的标准接口，SDK 中采用的是 HAL3。相机 HAL 的 HIDL 接口在 hardware/interfaces/camera 中定义，且典型的 Binderized HAL，必须要实现如下接口：
  - ICameraProvider：用于枚举单个设备并管理其状态。
  - ICameraDevice：相机设备接口。
  - ICameraDeviceSession：活跃的相机设备会话接口。

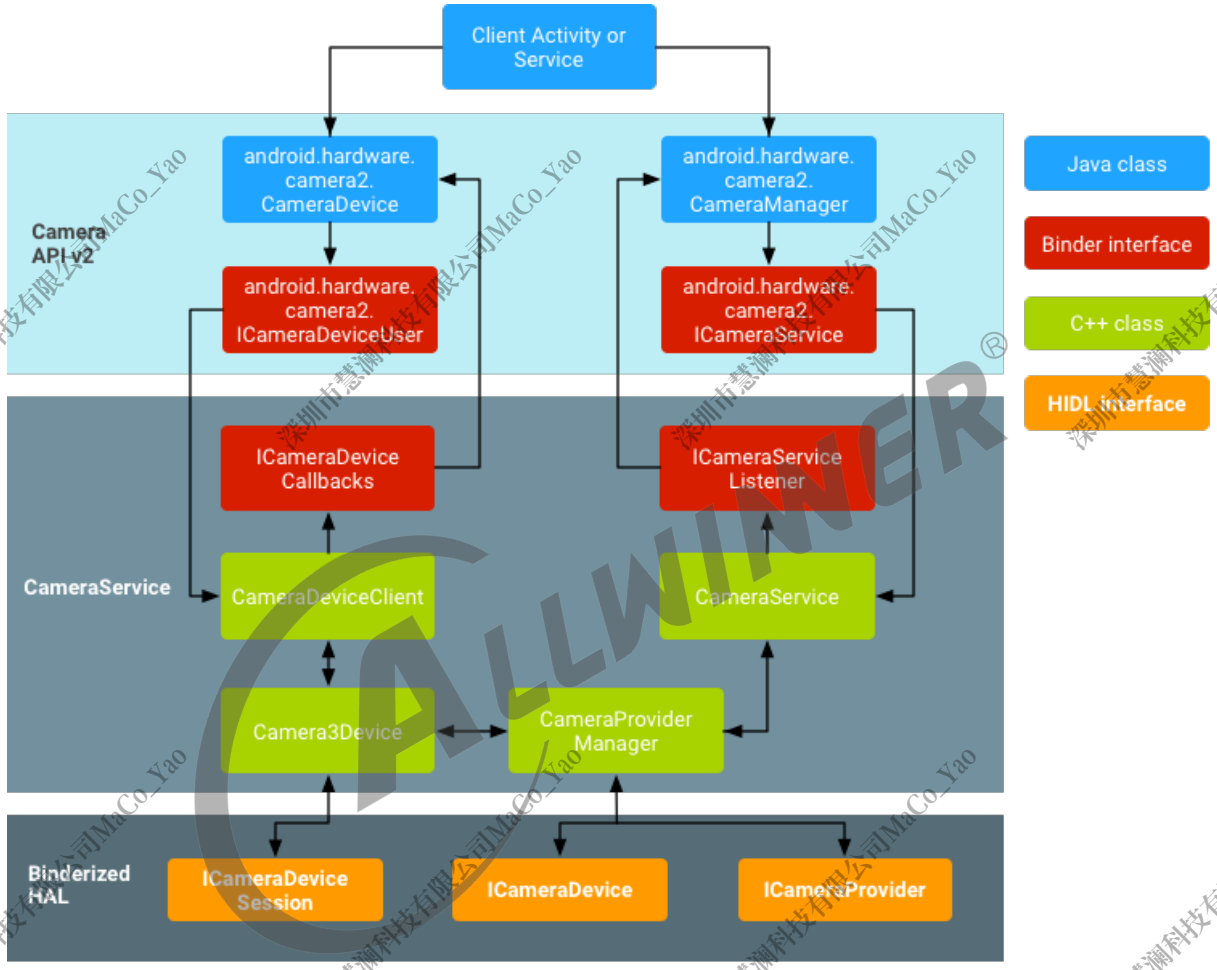

图 2-2: Camera 软件层次图

3 Camera 基本概念

3.1 分辨率

数字图像是由许多个点组成的，简称像素（Pixel）。图像分辨率，是指图像包含的像素数（我们通常说，这幅图像的分辨率为 1024×768，或者这幅图像的分辨率为 100 万像素）。它是衡量图像质量和细节表现能力的重要指标之一。多媒体领域常用的分辨率包括：

术语	分辨率大小
QCIF	176×144
QVGA	320×240
CIF	352×288
VGA	640×480
WVGA	800×480
D1	720×480/720×576
480i	720×480@30fps
576i	720×576@25fps
480p	720×480@60fps
576p	720×576@50fps
720p	1280×720@50fps or 60fps
1080i	1920×1080@25fps or 30fps
1080p	1920×1080@50fps or 60fps
130W 像素	1280×960
200W 像素	1600×1200
300W 像素	2048×1536
500W 像素	2592×1952
800W 像素	3264×2448
1300W 像素	4224×3136

3.2 YUV 颜色格式

Android 系统默认使用的颜色格式为 NV21（YUV420SP）。下面，对 YUV 数据格式做一个简单介绍。YUV，分为三个分量，Y 表示亮度，存储的是灰度信息。U 和 V 表示色度，存储的是色彩信息。主要用于电视领域和模拟视频领域，它的好处是：1. 减少了传送带宽，2. 解决了黑白电视和彩色电视的兼容问题。YUV 格式分为两大类：packed 和 planar。对于 planar 的 YUV 格

式，先连续存储所有像素点的 Y，紧接着存储所有像素点的 U 和 V（U 和 V 可能是交叉存储的(SP)，也可能是分别存储的（P））。对于 packed 的 YUV 格式，每个像素的 Y/U/V 是连续交错存储的。YUV 的存储格式与其采样方式密切相关，主要采样方式有三种：

1. YUV4:4:4：每一个 Y 对应一组 UV 分量。
2. YUV4:2:2：每两个 Y 共用一组 UV 分量。
3. YUV4:2:0：每四个 Y 共用一组 UV 分量。

YUV 常用存储格式包括：

- YUYV（YUV422）

Y00	U00	Y01	V00	Y02	U01	Y03	V01
Y10	U10	Y11	V10	Y12	U11	Y13	V11
Y20	U20	Y21	V20	Y22	U21	Y23	V21
Y30	U30	Y31	V30	Y32	U31	Y33	V31

图 3-1: YUYV 格式

- UYVY（YUV422）

U00	Y00	V00	Y01	U01	Y02	V01	Y03
U10	Y10	V10	Y11	U11	Y12	V11	Y13
U20	Y20	V20	Y21	U21	Y22	V21	Y23
U30	Y30	V30	Y31	U31	Y32	V31	Y33

图 3-2: UYVY 格式

- YUV422P

图 3-3: YUV422P 格式

- YV12/YU12（YUV420P）

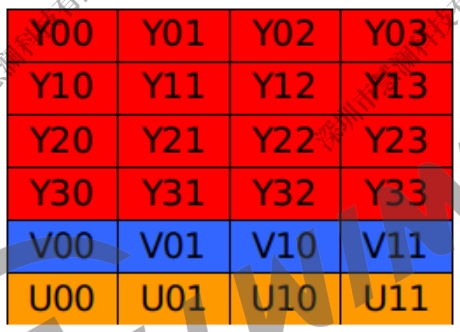

图 3-4: YV12 格式

- NV12/NV21(YUV420SP)

图 3-5: NV12 格式

3.3 JPEG 格式

JPEG 是一个压缩标准，是最常用的图像文件格式，一般 Camera 保存的图片格式为 JPEG 格式，后缀为 jpg 或者 jpeg。

JPEG 的性能，用质量与比特率之比来衡量，是相当优越的。它的优点包括:

- 它支持极高的压缩率，因此 JPEG 图像的下载速度大大加快。
- 它能够轻松地处理 16.8M 颜色，可以很好地再现全彩色的图像。
- 在对图像的压缩处理过程中，该图像格式可以允许自由地在最小文件尺寸（最低图像质量）和最大文件尺寸（最高图像质量）之间选择。
- 该格式的文件尺寸相对较小，下载速度快，有利于在带宽并不“富裕”的情况下传输。

JPEG 的缺点是:

- 并非所有的浏览器都支持将各种 JPEG 图像插入网页。
- 压缩时，可能使图像的质量受到损失，因此不适宜用该格式来显示高清晰度的图像。

JPEG 的压缩经过了如下四个步骤:

1. 颜色转换：在将彩色图像进行压缩之前，必须先对颜色模式进行数据转换。转换完成之后还需要进行数据采样。
2. CT 变换：是将图像信号在频率域上进行变换，分离出高频和低频信息的处理过程，然后再对图像的高频部分（即图像细节）进行压缩。首先以象素为单位将图像划分为多个 8×8 的矩阵，然后对每一个矩阵作 DCT 变换。把 8×8 的象素矩阵变成 8×8 的频率系数矩阵（所谓频率就是颜色改变的速度），频率系数都是浮点数。
3. 量化：由于下面第四步编码过程中使用的码本都是整数，因此要对频率系数进行量化，将之转换为整数。数据量化后，矩阵中的数据都是近似值，和原始图像数据之间有了差异，这一差异是造成图像压缩后失真的主要原因。这一过程中，质量因子的选取至为重要。值选得大，可以大幅度提高压缩比，但是图像质量就比较差，质量因子越小图像重建质量越好，但是压缩比越低。
4. 编码：编码是基于统计特性的方法。

JPEG 的基本类型分为段和图像数据，一般段的格式如下所示:

名称	字节数	数据	说明
段标识	1	FF	每个新段的开始标识
段类型	1		类型编码（称作“标记码”）
段长度	2		包括段内容和段长度本身，不包括段标识和段类型
段内容			≤65533 字节

JPEG 的常用段类型包括如下:

名称	标记码	说明
SOI	D8	文件头
EOI	D9	文件尾
SOF0	C0	帧开始（标准 JPEG）
SOF1	C1	同上
DHT	C4	定义 Huffman 表（霍夫曼表）
SOS	DA	扫描行开始
DQT	DB	定义量化表
DRI	DD	定义重新开始间隔
APP0	E0	定义交换格式和图像识别信息
COM	FE	注释

3.4 前后摄像头

前置后置摄像头最大的区别在于预览时的处理不同，后置摄像头直接显示采集到的视频帧，而前置摄像头必须要将视频进行左右镜像后显示，达到一种类似照镜子的效果。要注意的是，前置摄像头拍摄的照片并不是左右镜像的。

图 3-6: 前后摄像头示例

3.5 变焦

3.5.1 光学变焦

光学变焦是根据镜头的光学特性结构来实现变焦的，即通过旋转镜头来改变拍摄物品与镜头之间的距离，在画面上具有无损的画质。光学变焦的倍数越大，能拍摄的景物就越远。

3.5.2 数码变焦

数码变焦实质上是以视频帧中心按照一定比例截取一部分 buffer 进行放大显示和存储，它的目的是通过软件后期处理来模拟光学变焦的效果。通过数码变焦，拍摄的景物放大了，清晰度会有一定程度的下降，这一点与光学变焦是不能相比的，所以，数码变焦并没有太大的实际意义。数码变焦示意图如下所示：

图 3-7: 数码变焦示例图

3.5.3 混合变焦

混合变焦介于光学变焦以及数码变焦之间，利用多摄像头的信息以及图像处理算法，尽可能在保证画质的同时，保证放大变焦倍数，是大量手机厂商使用的变焦方式。

3.6 旋转

为了在 Android 设备中旋转正确，不管是通过软编还是硬编，都会检测当前的设备的旋转角度，并将角度信息记录到 JPEG 文件中。当在显示图像的软件中打开拍摄好的照片时，会先读取角度信息后旋转图像，并将旋转后的图像进行显示。

4 Camera Framework

4.1 Camera2 API

Android 10 推荐使用 Camera2 接口与相机进行交互，android.hardware.camera2 包提供了一个 interface 接口，用于多个独立的 Camera 设备与 Android 设备进行连接。Camera2 将 Camera 设备设计成一个流水线模型 (pipeline), 它通过输入请求 (input requests)，来拍摄单帧画面，并且每一个请求只拍摄一个图像。当处理完毕后，会对应输入请求随之输出一个拍摄结果 (capture result) 元数据包, 称之为 metadata packet 以及一系列的输出图像数据。这些请求是按输入的顺序进行排列的，多个请求发出时，可能会同时处于传输中，称为 in flight。考虑到 Camera 设备设计成一个具有多个阶段的流水线模型, 当有多个请求传输时，需要严格保证帧率达到最大。

当需要遍历，访问以及打开一个有效的 Camera 设备，用户需要获取一个 CameraManager 实例。每一个 Camera 设备都会提供一系列的静态属性用于描述该硬件设备，可用选项以及输出参数。这些参数可以通过 getCameraCharacteristics(String) 接口获取并将信息保存到 CameraCharacteristics 对象中。

当需要开始拍照时，应用必须要首先创建一个捕获会话 (capture session)，并且需要一系列的输出 Surfaces 提供给 Camera 设备，对应的方法是 createCaptureSession(SessionConfiguration)。每个 Surface 必须要经过预先配置，比如适当的大小和格式，并且要和 Camera 设备对应。目标 Surface 能够被多个类获取，包括 SurfaceView, SurfaceTexture, MediaCodec, MediaRecorder, Allocation 以及 ImageReader。

一般来说，Camera 的预览图像会被发送到 SurfaceView 或者 TextureView, 为 DngCreator 捕获的 Jpeg 图片或者 RAW 数据能够通过支持 Jpeg 或者 RAW Sensor 格式的 ImageReader 获得。对于 RenderScript,OpenGL Es, Native 代码，Camera 数据最好能够以 YUV 格式进行分配，而 SurfaceTexture 和 ImageReader 可以使用 YUV_420_888 格式保存。

应用此后需要创建一个 CaptureRequest，通过定义拍照参数来决定 Camera 设备的图片质量。CaptureRequest 还会列举本次拍照应当输出到的目标 Surface。CameraDevice 拥有一个工厂方法用以在特的情形下创建合适的 Request builder。当 CaptureRequest 成设置后，它就能够被传到只拍一张照片或者连续拍照 Capture Session 中处理，两种方法都能够接受一连串的 Reuqests，但连续的请求会比单个请求权限低。所以如果这两者同时发生时，单个拍照会在连续拍照之前进行处理。当处理完一个请求时，Camera 设备会创建一个 TotalCaptureResult 对象，它包括在拍照时刻 Camera 设备的状态信息以及最终使用的设置。如果参数之间需要舍入或者有冲突无法避免的时候，Camera 设备将发送一帧图像信息到输出 Surfaces，这些行为是异步的，有时候会延迟。

如下是有关 Camera2 的重要类介绍:

类名	解释
CameraDevice	CameraDevice 类代表了单个 Camera 设备连接到一个 Android 设备上，支持高帧率下的细精度的图像捕获控制以及后处理
CameraDevice.StateCallback	用于接收 Camera 设备的状态更新的回调
CameraCaptureSession	CameraDevice 对应的捕获会话 (capture session)，用于从 Camera 中捕获图像或者在最近相同的会话中处理刚获取的图像
CameraCaptureSession.CaptureCallback	用于追踪提交到 Camera 设备的 CaptureRequest 进度的回调对象
CameraCaptureSession.StateCallback	用于接收 Camera 捕获会话的更新状态的回调
CameraCharacteristics	用于描述 Camera 设备的属性
CameraCharacteristics.Key	键值用于 Camera 方便查找属性，可通过 get 获取 Camera 的 characteristics 属性
CameraDevice.StateCallback	用于接收 Camera 设备的状态更新的回调对象
CameraManager	用于检测，获取相机属性以及连接 CameraDevice 的系统服务管理器 (service Manager)
CameraManager.AvailabilityCallback	检测 Camera 设备能够打开或者不能打开的回调
CameraManager.TorchCallback	检测 Camera 的闪光模式无效，禁用或者可用的回调
CameraMetadata	用于描述 Camera 控制以及信息的基类
CameraOfflineSession	调用 CameraCaptureSession#switchToOffline 后，将 Capture Session 转为离线模式
CaptureFailure	从图像传感器中捕获单张图片发生错误时的报告
CaptureRequest	使用 Camera 设备拍照时需要的不可变的设置以及输出包
CaptureRequest.Builder	Capture Request 的 builder

类名	解释
CaptureRequest.Key	CaptureRequest 的键值，可以通过 CaptureResult#get 查询，也可以通过 CaptureRequest.Builder#set(Key, Object) 设置
CaptureResult	从图像传感器拍摄单张照片获取的分组结果
CaptureResult.Key	CaptureResult 的键值，通过 CaptureResult#get 查询
DngCreator	DngCreator 类提供了将原始像素数据写到 DNG 文件的方法
TotalCaptureResult	通过图像传感器拍摄单张图片获取的所有组装的结果
CameraAccessException	如果 Camera 设备不能通过 CameraManager 询问或者打开, 又或者对于一个打开的 CameraDevice 连接无效了，CameraAccessException 异常将会被抛出

4.2 Camera Framework 常用类

4.2.1 CameraManager

CameraManager 是全新的系统服务管理器，专门用于检测，连接，获取相机属性。相机应用可以通过 import android.hardware.camera2.CameraManager 直接调用 CameraManager 的接口。常用的 public 接口包括:

方法名	描述
getCameraCharacteristics	获取指定 cameraId 的相机设备属性
getCameraIdList	返回所有通过 ID 连接的相机设备
openCamera	连接指定 cameraId 的相机设备
registerAvailabilityCallback	注册当设备可用时的回调对象
registerTorchCallback	注册通知闪光灯模式状态的回调函数
setTorchMode	设定指定 ID 的闪光灯模式而无需打开相机设备
unregisterAvailabilityCallback	注销近期增加的回调，回调将不会接收连接或断开的通知
unregisterTorchCallback	注销近期增加的回调，回调对象将不会再接收闪光灯状态的通知

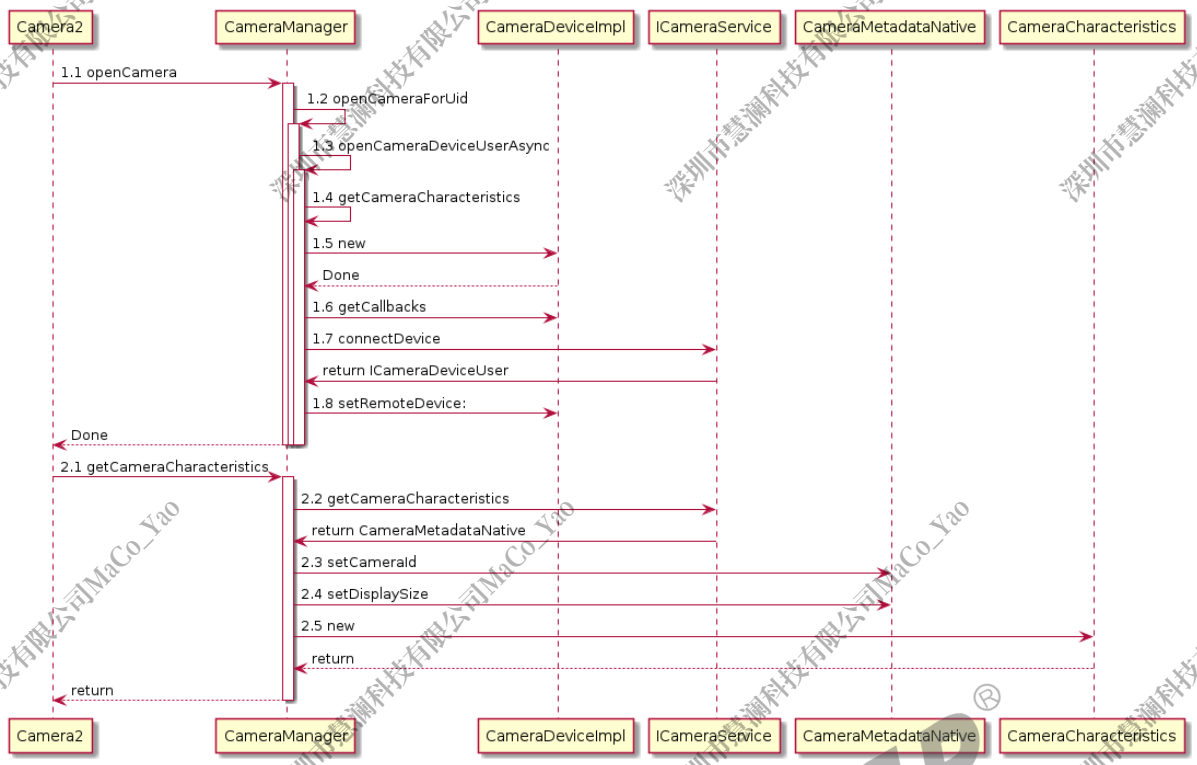

图 4-1: CamearManager 常用函数流程图

4.2.2 CameraCharacteristics

CameraCharacteristics 是用于描述相机特性的对象，它继承了基类 CameraMetadata，通过键值对 Map 存储属性，这些属性是不可以改变。Apk 可通过 CameraManager 获相机属性，具体的属性包括:

字段名	描述
COLOR_CORRECTION_AVAILABLE_ABERRATION--_MODES	相机设备支持的像差校正模式列表，用于 android.colorCorrection.--aberrationMode
CONTROL_AE_AVAILABLE_ANTIBANDING_MODES	相机设备支持的自动曝光反冲带模式列表，用于 android.control.--aeAntibandingMode
CONTROL_AE_AVAILABLE_MODES	相机设备支持的自动曝光模式列表，用于 android.control.aeMode
CONTROL_AE_AVAILABLE_TARGET_FPS_RANGES	相机设备支持的帧率范围，用于 android.control.--aeTargetFpsRange

字段名	描述
CONTROL_AE_COMPENSATION_RANGE	相机设备支持的曝光补偿范围，用于 android.control.--aeExposureCompensation
CONTROL_AE_COMPENSATION_STEP	曝光补偿能够改变的最小步长
CONTROL_AE_LOCK_AVAILABLE	相机设备是否支持 android.control.aeLock
CONTROL_AF_AVAILABLE_MODES	相机设备支持的自动聚焦模式列表，用于 android.control.afMode
CONTROL_AVAILABLE_EFFECTS	相机设备支持的颜色效果列表，用于 an-droid.control.effectMode
CONTROL_AVAILABLE_MODES	相机设备支持的控制模式列表，用于 android.control.mode
CONTROL_AVAILABLE_SCENE_MODES	相机设备支持的场景模式列表（如夜景、美食、烛光等），用于 an-droid.control.sceneMode
CONTROL_AVAILABLE_VIDEO_--STABILIZATION_MODES	相机设备支持的视频稳定模式列表，用于 android.control.--videoStabilizationMode
CONTROL_AWB_AVAILABLE_MODES	相机设备支持的自动白平衡模式列表，用于 android.control.awbMode
CONTROL_AWB_LOCK_AVAILABLE	相机设备是否支持 android.control.awbLock
CONTROL_MAX_REGIONS_AE	自动曝光可使用的最大测量区域数
CONTROL_MAX_REGIONS_AF	自动聚焦可使用的最大测量区域数
CONTROL_MAX_REGIONS_AWB	自动白平衡可使用的最大测量区域数
CONTROL_POST_RAW_SENSITIVITY_--BOOST_RANGE	相机设备支持的 boost 范围，用于 android.control.--postRawSensitivityBoost

字段名	描述
DEPTH_DEPTH_IS_EXCLUSIVE	指出一个捕捉请求是否能够同时支持深度输出（如 DEPTH16 / DEPTH_POINT_CLOUD）和普通颜色输出（如 YUV_420_888, JPEG 或 RAW
DISTORTION_CORRECTION_AVAILABLE_MODES	相机设备支持的畸变矫正模式列表，用于 android.- -distortionCorrection.mode
EDGE_AVAILABLE_EDGE_MODES	相机设备支持的边缘增强模式列表，用于 android.edge.mode
FLASH_INFO_AVAILABLE	相机设备是否有闪光灯部件
HOT_PIXEL_AVAILABLE_HOT_PIXEL_MODES	相机设备支持的像素矫正模式列表，用于 android.hotPixel.mode
INFO_SUPPORTED_HARDWARE_LEVEL	相机设备支持的硬件等级
INFO_VERSION	关于相机设备的制造商版本信息，例如网络服务硬件、传感器等
JPEG_AVAILABLE_THUMBNAIL_SIZES	相机设备支持的 JPEG 格式缩略图大小列表，用于 an-droid.jpeg.thumbnailSize
LENS_DISTORTION	矫正系数，以纠正相机设备的径向和切向的镜头畸变
LENS_FACING	相机设备相对于屏幕的方向，例如后置摄像头一般是 LENS_FACING_FRONT
LENS_INFO_AVAILABLE_APERTURES	相机设备支持的光圈大小列表，用于 android.lens.aperture
LENS_INFO_AVAILABLE_FILTER_DENSITIES	相机设备支持的中性密度滤波值列表，用于 android.lens.filterDensity
LENS_INFO_AVAILABLE_FOCAL_LENGTHS	相机设备支持的焦距列表，用于 android.lens.focalLength

字段名	描述
LENS_INFO_AVAILABLE_OPTICAL_STABILIZATION	相机设备支持的光学稳像（optical image stabilization，OIS）模式列表，用于 android.lens.--opticalStabilizationMode
LENS_INFO_FOCUS_DISTANCE_CALIBRATION	镜头焦距校准质量
LENS_INFO_HYPERFOCAL_DISTANCE	镜头的超焦距
LENS_INFO_MINIMUM_FOCUS_DISTANCE	能使镜头聚焦的最短距离
LENS_INTRINSIC_CALIBRATION	相机设备的内部标定参数（相机内参）
LENS_POSE_REFERENCE	镜头的引用位置，即用于 android.lens.poseTranslation 的起始坐标
LENS_POSE_ROTATION	相机设备相对于传感器坐标系的方向
LENS_POSE_TRANSLATION	相机光学中心位置
LENS_RADIAL_DISTORTION	镜头径向畸变，在 Android 28 上已弃用，使用 android.lens.distortion 代替
LOGICAL_MULTI_CAMERA_SENSOR_SYNC_TYPE	物理相机间帧时间戳同步的准确性，决定了物理相机同时开始曝光的能力
NOISE_REDUCTION_AVAILABLE_NOISE--_REDUCTION_MODES	相机设备支持的降噪模式列表，用于 android.noiseReduction.mode
REPROCESS_MAX_CAPTURE_STALL	重处理捕获请求时的最大失帧（以帧为单位）
REQUEST_AVAILABLE_CAPABILITIES	相机设备完全支持的功能列表
REQUEST_MAX_NUM_INPUT_STREAMS	相机设备能够同时支持的不同类型的输入流最大数量
REQUEST_MAX_NUM_OUTPUT_PROC	相机设备能够同时支持的不同处理格式的输出流最大数量（无停顿）
REQUEST_MAX_NUM_OUTPUT_PROC_STALLING	相机设备能够同时支持的不同处理格式的输出流最大数量（有停顿）
REQUEST_MAX_NUM_OUTPUT_RAW	对于任何 RAW 格式，相机设备能够同时支持的不同类型的输出流最大数量

字段名	描述
REQUEST_PARTIAL_RESULT_COUNT	定义一个结果将由多少个子结构组成
REQUEST_PIPELINE_MAX_DEPTH	指定一个帧从暴露到可使用，所经过的管道阶段数的最大值
SCALER_AVAILABLE_MAX_DIGITAL_ZOOM	active 区域宽高和裁剪区域宽度比值的最大值，用于 android.scaler.cropRegion
SCALER_CROPPING_TYPE	相机设备支持的裁剪类型，有 CENTER_ONLY 和 FREEFORM·两种
SCALER_STREAM_CONFIGURATION_MAP	相机设备支持的可用流的配置，包括最小帧间隔、不同格式、大小组合的帧时长
SENSOR_AVAILABLE_TEST_PATTERN_MODES	相机设备支持的测试模式列表，用于 android.sensor.testPatternMode
SENSOR_BLACK_LEVEL_PATTERN	每个彩色滤光片排列 (CFA) 镶嵌通道的固定黑色电平偏移量
SENSOR_CALIBRATION_TRANSFORM1	从参考传感器颜色空间映射到实际设备传感器颜色空间的每个设备校准转换矩阵
SENSOR_CALIBRATION_TRANSFORM2	从参考传感器颜色空间映射到实际设备传感器颜色空间 (这是原始缓冲区数据的颜色空间) 的每个设备校准转换矩阵
SENSOR_COLOR_TRANSFORM1	将颜色值从 CIE XYZ 颜色空间转换为参考传感器颜色空间的矩阵
SENSOR_COLOR_TRANSFORM2	将颜色值从 CIE XYZ 颜色空间转换为参考传感器颜色空间的矩阵
SENSOR_FORWARD_MATRIX1	将白平衡摄像机颜色从参考传感器颜色空间转换为带有 D50 白点的 CIE XYZ 颜色空间的矩阵
SENSOR_FORWARD_MATRIX2	将白平衡摄像机颜色从参考传感器颜色空间转换为带有 D50 白点的 CIE XYZ 颜色空间的矩阵

字段名	描述
SENSOR_INFO_ACTIVE_ARRAY_SIZE	经过几何畸变校正后，图像传感器对应于活动像素的面积
SENSOR_INFO_COLOR_FILTER_ARRANGEMENT	传感器上颜色滤波器的排列，按读取顺序，表示传感器左上 2x2 部分的颜色
SENSOR_INFO_EXPOSURE_TIME_RANGE	相机设备支持的图片曝光时间范围，用于 android.sensor.exposureTime
SENSOR_INFO_LENS_SHADING_APPLIED	相机设备输出的原始图像是否受镜头阴影校正影响
SENSOR_INFO_MAX_FRAME_DURATION	相机设备支持的最大的帧间隔时间，用于 android.sensor.frameDuration
SENSOR_INFO_PHYSICAL_SIZE	完整像素数列的物理尺寸
SENSOR_INFO_PIXEL_ARRAY_SIZE	完整像素数列的尺寸，可能包括黑色校准像素
SENSOR_INFO_PRE_--CORRECTION_ACTIVE_ARRAY_SIZE	在应用任何几何畸变校正之前，与活动像素相对应的图像传感器的面积
SENSOR_INFO_SENSITIVITY_RANGE	相机设备支持的敏感度范围，用于 android.sensor.sensitivity
SENSOR_INFO_TIMESTAMP_SOURCE	传感器开始捕捉的时间戳的时间基础
SENSOR_INFO_WHITE_LEVEL	传感器最大的 raw 值输出
SENSOR_MAX_ANALOG_SENSITIVITY	纯粹通过模拟增益实现的最大敏感度
SENSOR_OPTICAL_BLACK_REGIONS	表示传感器光学屏蔽的黑色像素区域的不相交矩形列表
SENSOR_ORIENTATION	使输出图像在设备屏幕上以本机方向垂直，顺时针方向旋转的角
SENSOR_REFERENCE_ILLUMINANT1	当计算 android.sensor.--colorTransform1, android.sensor.--calibrationTransform1 和 android.sensor.forwardMatrix1 矩阵时，作为场景光源的标准参考光源

字段名	描述
SENSOR_REFERENCE_ILLUMINANT2	当计算 android.sensor.colorTransform2, android.sensor.calibrationTransform2 和 android.sensor.forwardMatrix2 矩阵时，作为场景光源的标准参考光源
SHADING_AVAILABLE_MODES	相机设备支持的镜头应用模式列表，用于 android.shading.mode
STATISTICS_INFO_AVAILABLE_FACE_DETECT_MODES	相机设备支持的人脸检测模式列表，用于 android.statistics.faceDetectMode
STATISTICS_INFO_AVAILABLE_HOT_PIXEL_MAP_MODES	相机设备支持的热像素映射输出模式列表，用于 android.statistics.hotPixelMapMode
STATISTICS_INFO_AVAILABLE_LENS_SHADING_MAP_MODES	相机设备支持的镜头阴影映射输出模式列表，用于 android.statistics.lensShadingMapMode
STATISTICS_INFO_AVAILABLE_OIS_DATA_MODES	相机设备支持的 OIS 数据输出模式列表，用于 android.statistics.lensShadingMapMode
STATISTICS_INFO_MAX_FACE_COUNT	同时能够检测的人脸数的最大值
SYNC_MAX_LATENCY	请求 (与前一个请求不同) 提交后以及结果状态同步之前可能出现的最大帧数
TONEMAP_AVAILABLE_TONE_MAP_MODES	相机设备支持的图像增强映射模式列表，用于 android.tonemap.mode
TONEMAP_MAX_CURVE_POINTS	图像增强映射曲线支持的最大点的数量，用于 android.tonemap.curve

4.2.3 CameraDevice

CameraDevice 是用于形容连接到 Android 设备的 Camera 设备，通过 CameraDevice 可以对拍照图像进行精细的控制和后处理。当应用需要访问 Camera 设备时，首先需要在 AndroidManifest.xml 中定义相机权限: “android.permission.CAMERA”, 从而获取控制 Camera 的权限。此外，通过上述 CameraCharacteristics 获取的 INFO_SUPPORTED_HARDWARE_LEVEL 的值，还有如下区别:

1. CameraMetadata#INFO_SUPPORTED_HARDWARE_LEVEL_LEGACY: Camera 设备处于反向兼容模式，至少支持 camear2 API。
2. CameraMetadata#INFO_SUPPORTED_HARDWARE_LEVEL_LIMITED:Camera 设备几乎与老版 Camera 一致。
3. CameraMetadata#INFO_SUPPORTED_HARDWARE_LEVEL_EXTERNAL:Camera 设备为可拔插设备，且拥有更少的特性，和 CameraMetadata#INFO_SUPPORTED_HARDWARE_LEV 类似。
4. CameraMetadata#INFO_SUPPORTED_HARDWARE_LEVEL_FULL/CameraMetadata#INFO_SUPPORTED_HARDWARE_LEVEL_3:Camera 设备提供大幅度提升的性能。需要在 AndroidManifest.xml 中增加 “android.hardware.camera.level.full” 特性。

部分常用的 public 函数描述如下:

方法名	描述
close	尽快关闭相机设备的连接, 设备关闭后，StateCallback#onClosed 回调被调用。且相机设备此时能够再次被打开。
createCaptureRequest	创建 CaptureRequest.Builder，用于新的捕获请求。
createCaptureSession	创建 CameraCaptureSession，并通过 SessionConfiguration 聚集所有支持的参数
createReprocessCaptureRequest	创建 CaptureRequest.Builder，用于重处理 CaptureRquest。

4.2.4 CameraCaptureSession

CameraCaptureSession 被设计成一个会话，专门用于在相机设备中捕获图像以及重处理刚拍照后的图像。CameraCaptureSession 在创建时必须要提供目标输出的 surfaces 给 CameraDevice#createCaptureSession 接口，或者有必要提供一个 InputConfiguration 和目标输出 Surface 集合到 CameraDevice#createReprocessableCaptureSession。一旦新建了 CameraCaptureSession，它就会一直保持活跃，直到新的会话被创建或者该会话被关闭。

部分常用的 public 函数描述如下:

方法名	描述
capture	提交捕获一张图像的请求，输入参数 CaptureRequest 会定义图像参数，如传感器，镜头，闪光灯等
captureBurst	提交一系列请求用于捕获图像，称为“burst”，可用于连拍。和 captur 最大的区别是该方法能确保没有其他的 Request 夹杂其中
captureBurstRequests	与 captureBurst 功能类似
close	异步关闭 CaptureSession
prepare	为输出 surface 预先分配内存
setRepeatingBurst	通过该方法设置连拍
stopRepeating	停止连拍设置

4.2.5 CaptureRequest

CaptureRequest 包含硬件的配置，包括传感器，镜头属性，闪光灯等等，处理的流水线，控制算以及输出 buffer。CaptureRequest 通过 CameraDevice 的 createCaptureRequest 创建，并在 CameraCaptureSession 的 capture 以及 CameraCaptureSession 的 setRepeatingRequest 中作参数传入。CaptureRequest 的配置与 CameraCharacteristic 一一对应。

4.2.6 CaptureResult

CaptureResult 是单次捕获图片返回的结果分组。CaptureResult 也和 CaptureRequest 有类似的结构，包括硬件配置，处理流水线，控制算法以及输出 buffers。CaptureResult 是在 CameraDevic 处理完 CaptureRequest 后产生的。所有的 Capture 请求，都可以在 CaptureResult 中进行查询。

4.3 CameraService

CameraManager 通过 CameraService 进行一系列的操作如打开相机设备，获取相机属性等。CameraService 进程名为 camerserver，代码路径如下:

- frameworks/av/camera/cameraserver
- frameworks/av/services/camera/libcameraservice

Camera2 应用通过 CameraManager 与 CameraService 进行跨进程的 Binder 通信，CameraService 收到上层的请求后，再通过 HIDL 通信与 HAL 层进行通信。

下图为 CameraService 相关类图:

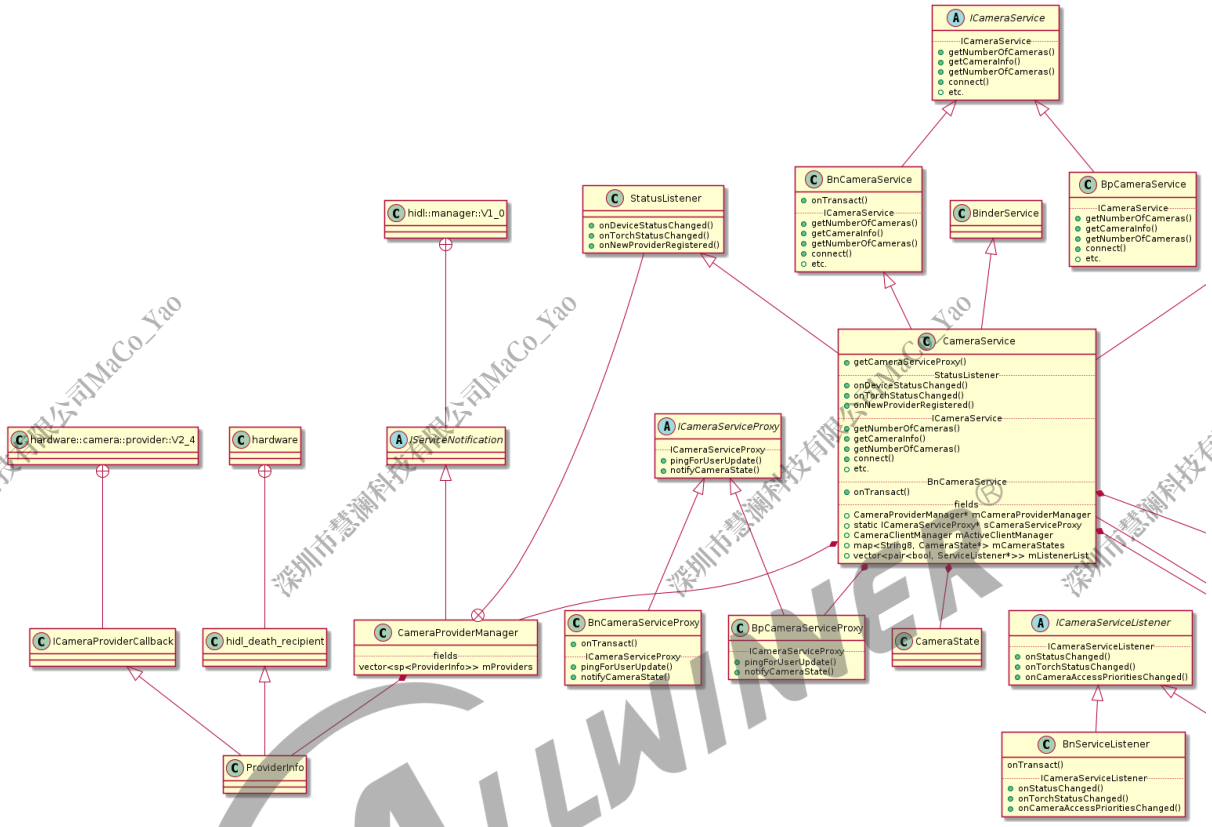

图 4-2: CameraService 类图

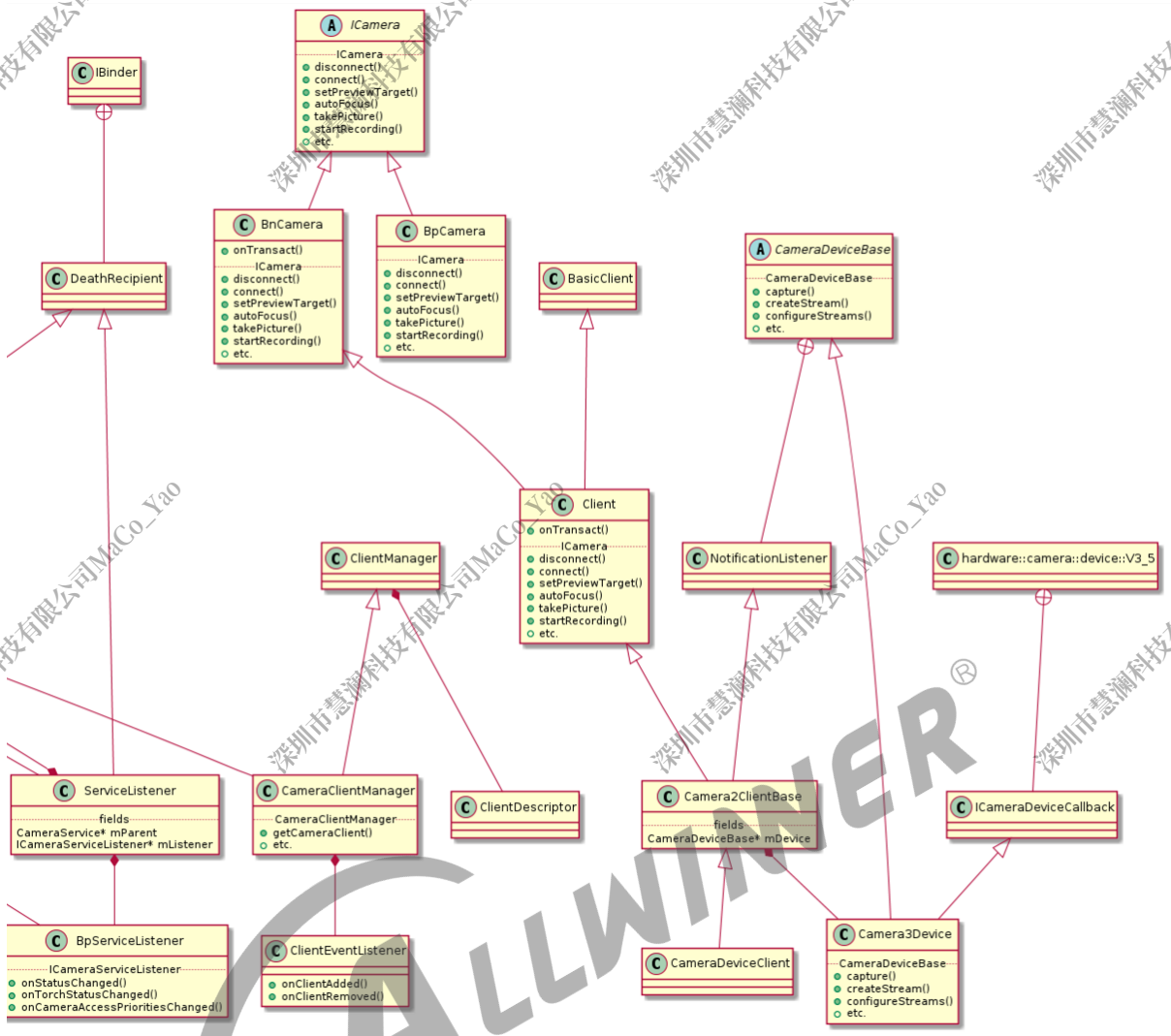

图 4-3: CameraService 类图

5 Camera HAL3 开发

5.1 Camera HAL3 功能概述

HAL3 的相机子系统将多个运行模式整合为一个统一的视图，用户可以使用这种视图实现之前的任何模式以及一些其他模式，例如连拍模式。这样一来，便可以提高用户对聚焦、曝光以及更多后期处理（例如降噪、对比度和锐化）效果的控制能力。此外，这种简化的视图还能够使应用开发者更轻松地使用相机的各种功能。

API 将相机子系统塑造为一个管道，该管道可按照 1:1 的基准将传入的帧捕获请求转化为帧。这些请求包含有关帧的捕获和处理的所有配置信息，其中包括分辨率和像素格式；手动传感器、镜头和闪光灯控件；3A 操作模式；RAW 到 YUV 处理控件；统计信息生成等等。

如下图所示，应用通过 Framework 层首先准备好 CaptureRequest，配置设置以及目标的 Surfaces，然后通过 capture 接口请求帧，经过 CameraDevice 时，将请求入列，会有一个等待请求队列的 FIFO 的队列。紧接着经过 Camera HAL3，进入 IN-FLIGHT 队列处理，处理结束后，将结果返回到输出流，并调用 onCaptureComplete() 回调表明拍照流程结束。此外，系统还会针对每组结果生成包含色彩空间和镜头遮蔽等信息的元数据。用户可以将相机 HAL3 看作相机 HAL1 的单向流管道。它会将每个捕获请求转化为传感器捕获的一张图像，这张图像将被处理成：

- 包含有关捕获的元数据的 Result 对象。
- 图像数据的 1 到 N 个缓冲区，每个缓冲区会进入自己的目标 Surface。

可能的输出 Surface 组经过预配置：

- 每个 Surface 都是一个固定分辨率的图像缓冲区流的目标位置。
- 一次只能将少量 Surface 配置为输出（约 3 个）。

一个请求中包含所需的全部捕获设置，以及要针对该请求将图像缓冲区（从总配置组）推送到其中的输出 Surface 的列表。请求可以只发生一次（使用 capture()），也可以无限重复（使用 setRepeatingRequest()）。捕获的优先级高于重复请求的优先级。

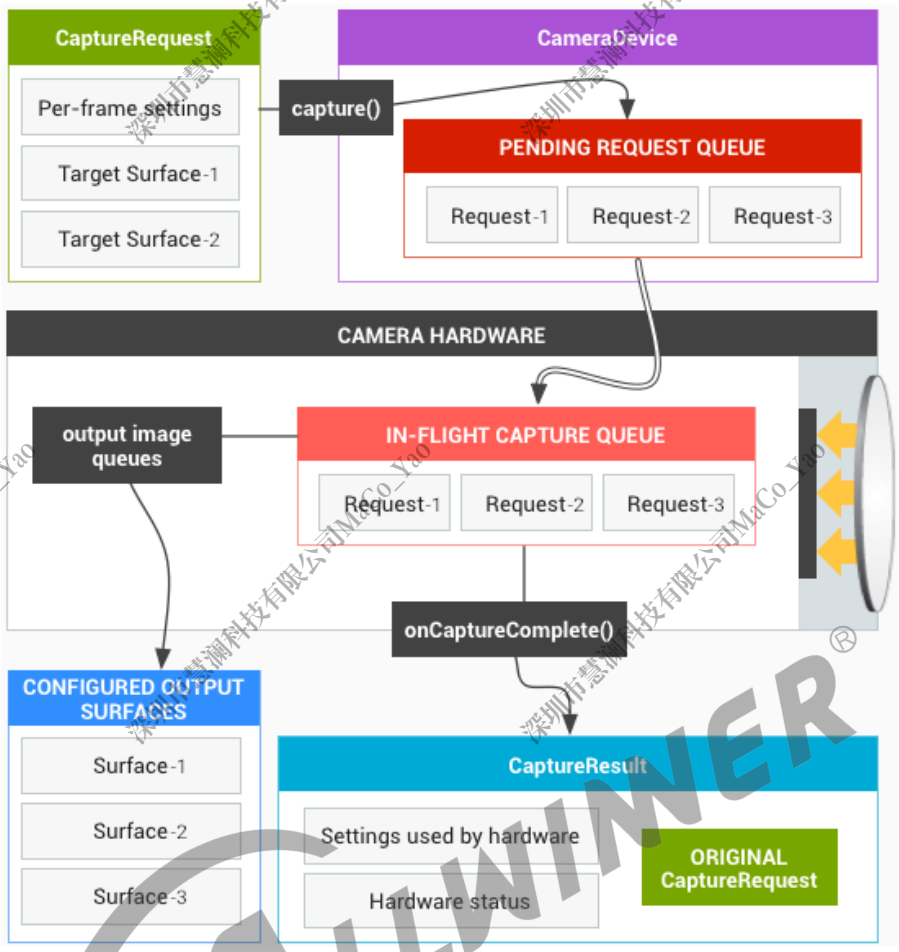

图 5-1：相机核心操作模型

应用框架会针对捕获的结果向相机子系统发出请求。一个 CaptureRequest 请求对应一组结果。请求包含有关捕获和处理这些结果的所有配置信息。其中包括分辨率和像素格式；手动传感器、镜头和闪光灯控件；3A 操作模式；RAW 到 YUV 处理控件；以及统计信息的生成等。这样一来，便可更好地控制结果的输出和处理。一次可发起多个请求，而且提交请求时不会出现阻塞。请求始终按照接收的顺序进行处理。

HAL 的操作摘要有如下几点：

- 捕获的异步请求来自于框架。
- HAL 设备必须按顺序处理请求。对于每个请求，均生成输出结果元数据以及一个或多个输出图像缓冲区。
- 请求和结果以及后续请求引用的信息流遵守先进先出规则。
- 指定请求的所有输出的时间戳必须完全相同，以便框架可以根据需要将它们匹配在一起。
- 所有捕获配置和状态（不包括 3A 例程）都包含在请求和结果中。

更详细的 HAL 数据流程图如下所示:

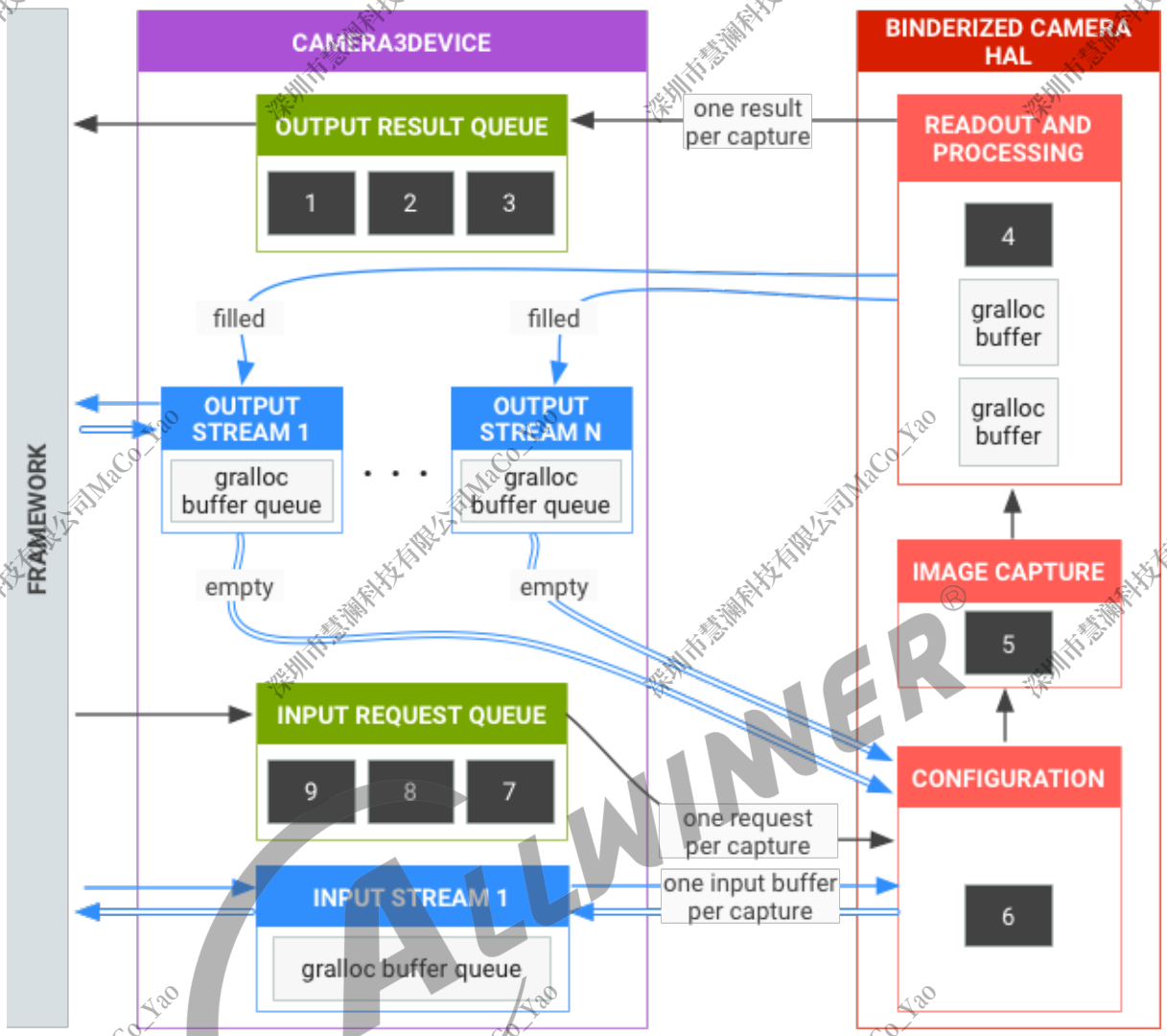

图 5-2: 相机 HAL 概览

5.2 Camera HAL3 初始化

Camera HAL3 的目录位于 android/hardware/aw/camera/3_4/, 在分析接口前，需要了解 V4L2CameraHAL 的初始化流程，如下图所示:

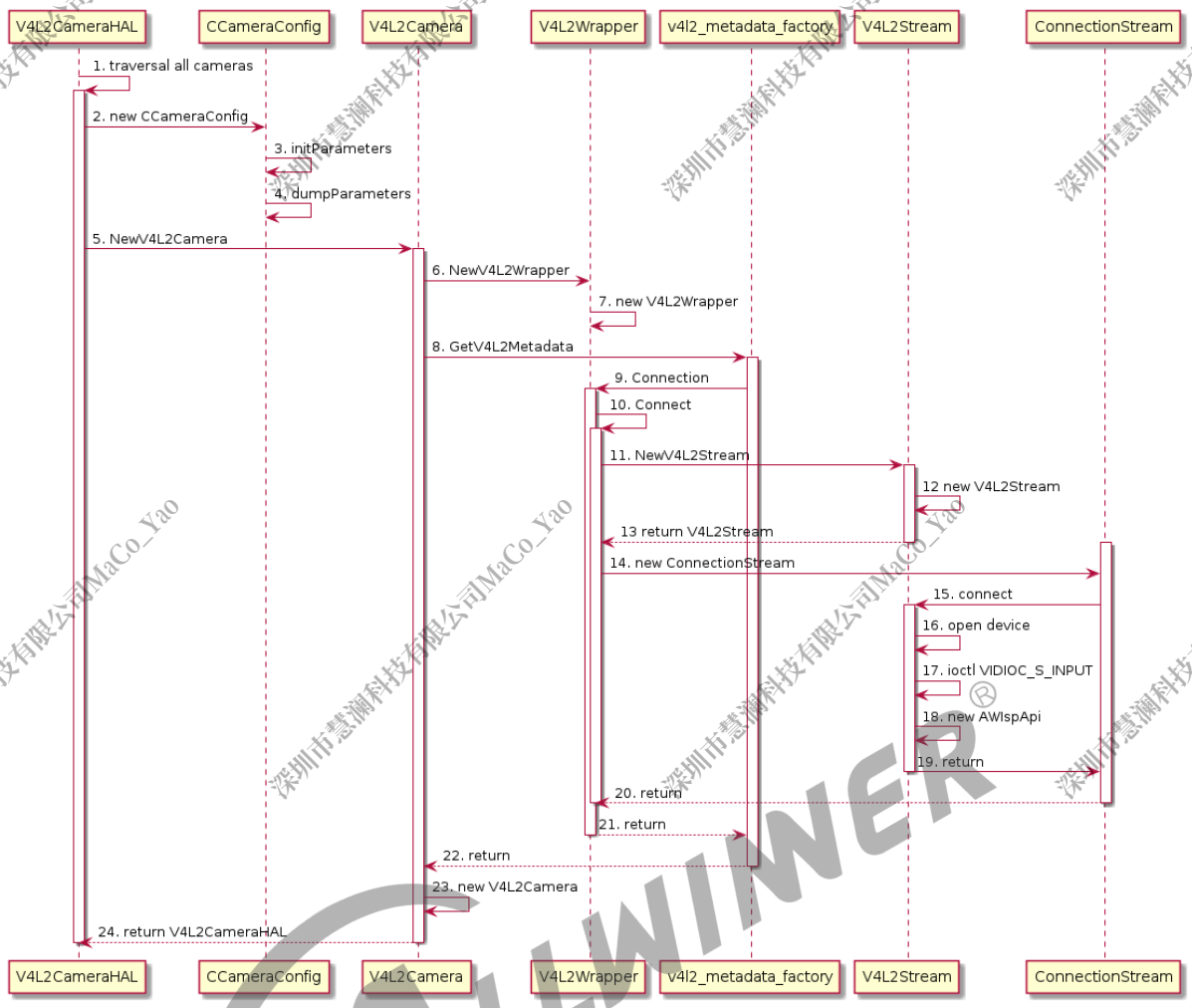

图 5-3: Camera HAL3 初始化时序图

5.3 Camera HAL3 接口

5.3.1 camera_module

camera module 的常规方法，必须是第一个成员变量，因为使用该数据结构的用户会将 hw_module_t 强制转化为 camera_module 指针。1. ENODEV: camera 设备不能够被打开因为内部错误 2. EINVAL: 输入参数是无效的。也就是说 id 是无效或者 module 是无效的。3. EBUSY: 对该 camera id，camera 设备已经调用了 open 4. EUSERS: 已经到达了最大能够打开 camera 设备的数目了。5. 其余 common.methods->open 都会返回 ENODEV. 6. 返回 0 时表示成功打开设备。

```
//hardware/aw/camera/3_4/v4l2_camera_hal.cpp
#define CAMERA_MODULE_API_VERSION_2_4 HARDWARE_MODULE_API_VERSION(2, 4)
#define HARDWARE_HAL_API_VERSION HARDWARE_MAKE_API_VERSION(1, 0)
#define CAMERA_HARDWARE_MODULE_ID "camera"
```

```
static hw_module_methods_t v4l2_module_methods = {
    .open = v4l2_camera_hal::open_dev};
...
camera_module_t HAL_MODULE_INFO_SYM __attribute__((visibility("default"))) = {
    .common =
        {
            .tag = HARDWARE_MODULE_TAG,
            .module_api_version = CAMERA_MODULE_API_VERSION_2_4,
            .hal_api_version = HARDWARE_HAL_API_VERSION,
            .id = CAMERA_HARDWARE_MODULE_ID,
            .name = "V4L2 Camera HAL v3",
            .author = "ZJW",
            .methods = &v4l2_module_methods,
            .dso = nullptr,
            .reserved = {0},
        },
    .get_number_of_cameras = v4l2_camera_hal::get_number_of_cameras,
    .get_camera_info = v4l2_camera_hal::get_camera_info,
    .set_callbacks = v4l2_camera_hal::set_callbacks,
    .get_vendor_tag_ops = v4l2_camera_hal::get_vendor_tag_ops,
    .open_legacy = v4l2_camera_hal::open_legacy,
    .set_torch_mode = v4l2_camera_hal::set_torch_mode,
    .init = nullptr,
    .reserved = {nullptr, nullptr}};
    }
```

V4L2CameraHAL 的接口流程图如下:

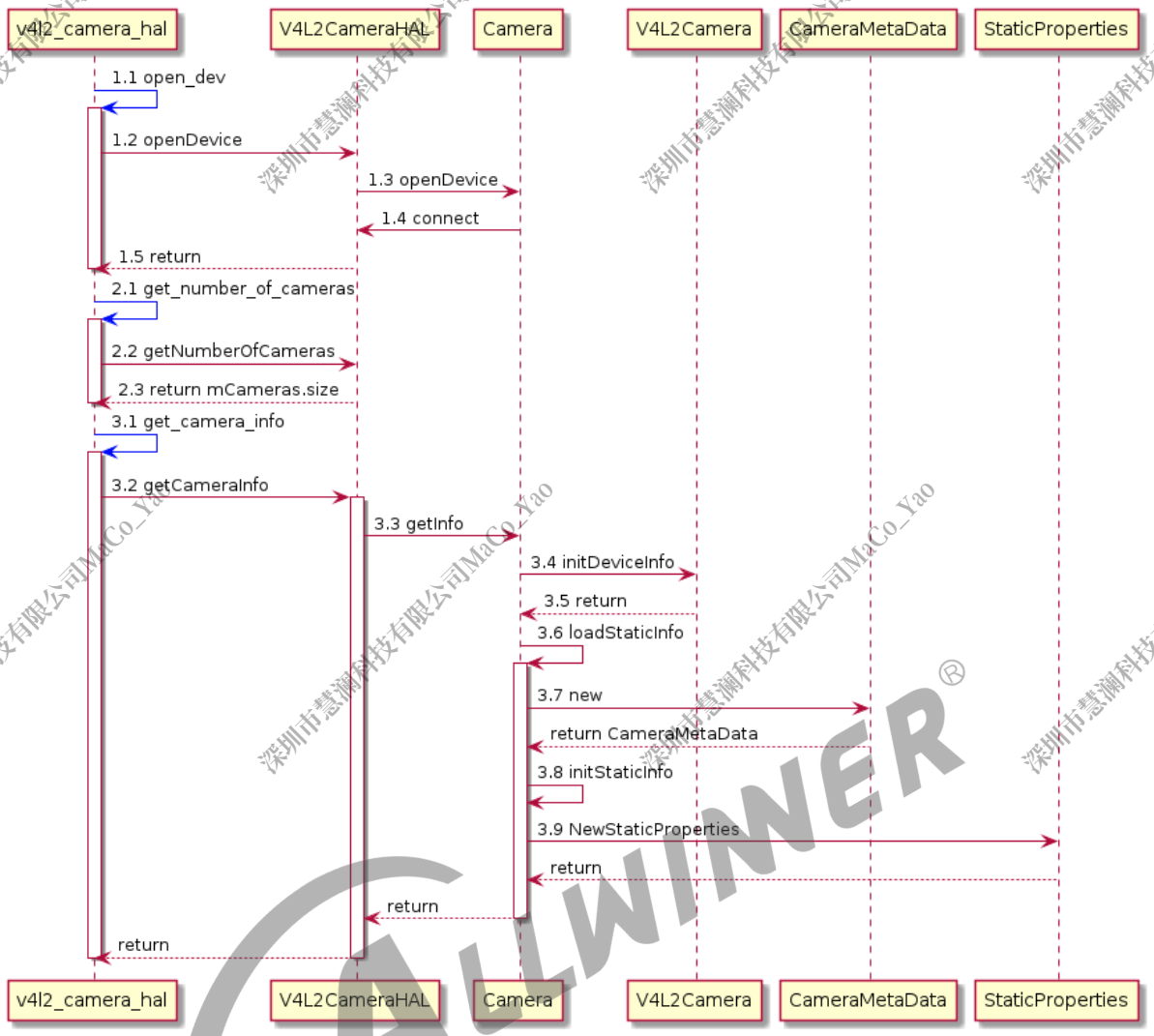

图 5-4: V4L2CameraHAL 接口流程图

5.3.2 camera3_device_ops_t

5.3.2.1 initialize

> initialize 本质是将 framework 的 callback 函数指针传到 HAL 层，该操作必须要在 open() 后且在其他所有函数之前调用。

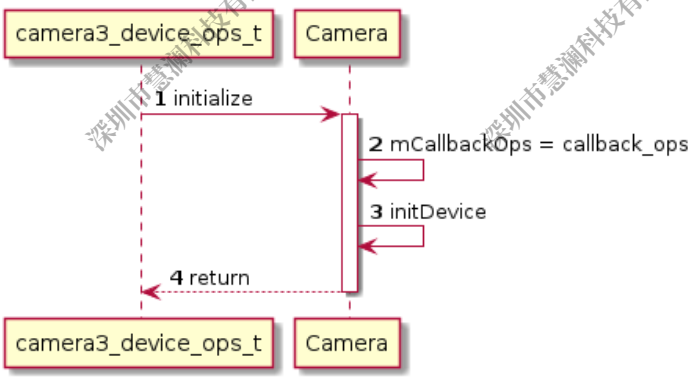

图 5-5: camera3_device_ops_t–initialize 流程图

Camera 的 initialize 主要设置了回调方法 callback_ops，具体回调函数后续展开分析。initDevice 实现暂时为空，为了初始化设备保留的接口。

5.3.2.2 configure_stream

- configure_stream 会重新设置 Camera HAL 设备的处理流水线 (pipeline) 并设置新的输入出流 (streams), 该调用将覆盖原有 stream_list 中的配置 (stream configuration), 该调用会在 initialize 调用后调用，并且在 processs_capture_requests 提交 request 前调用。
- configure_stream 的参数 stream_list 必须至少包含一个输出流 (output-capble stream), 且可能包含超过一个输入流 (input-capable stream)
- stream_list 可以包含当前正在使用的流 (通过最近调用 configure_streams 获得的)。这类流将保存有效的 usage 值，max_buffers 以及 private 指针 (camera3_stream_t)。
- 如果 HAL 需要为一个已有的流改变配置 (stream configuration)，它将根据这次 configure_stream 调用重新设置 usage 的值，或者 max_buffers 的值.
- framework 层将能够检测出这些改动，并在这次 request 使用这些 buffers 前，重新分配流的 buffers。
- 如果这次的传参 stream_list 并不包含当前使用的流，HAL 层能够安全的移除这些流的所有引用，它们将不会再被 framework 重用。并且所有的 gralloc buffers 将会在 configure_streams 调用返回前被释放。
- 参数 stream_list 是属于 framework 的，并在 configure_streams 完成后不能被访问。camera3_stream_t 的地址在 HAL 层将保持有效，直到第一次不包含 camera3_stream_t 参数的 configure_stream 接口被调用。除了 usage 以及 max_buffers 变量在 configure_streams 调用时能够改动，HAL 层不能修改其他 stream 结构体的值 (private pointer 除外)。
- 如果流是新设置的，结构体中的 max_buffer 以及 private pointer 将会设置为 0。usag 将会被设置为消费者 flags(consumer usage flags)。HAL 设备必须在 configure_streams

返回时设置这些变量。这些变量将会被 Framework 以及平台的 gralloc 模块用于为每一个流分配 gralloc 内存。

- 新分配的 buffers 可能会被 framework 包含在一个 capture request 中。一旦 gralloc buffers 通过回调方法 process_capture_result 返回到 framework,framework 就有可能在任何时间内立即释放内存或者重新使用它。

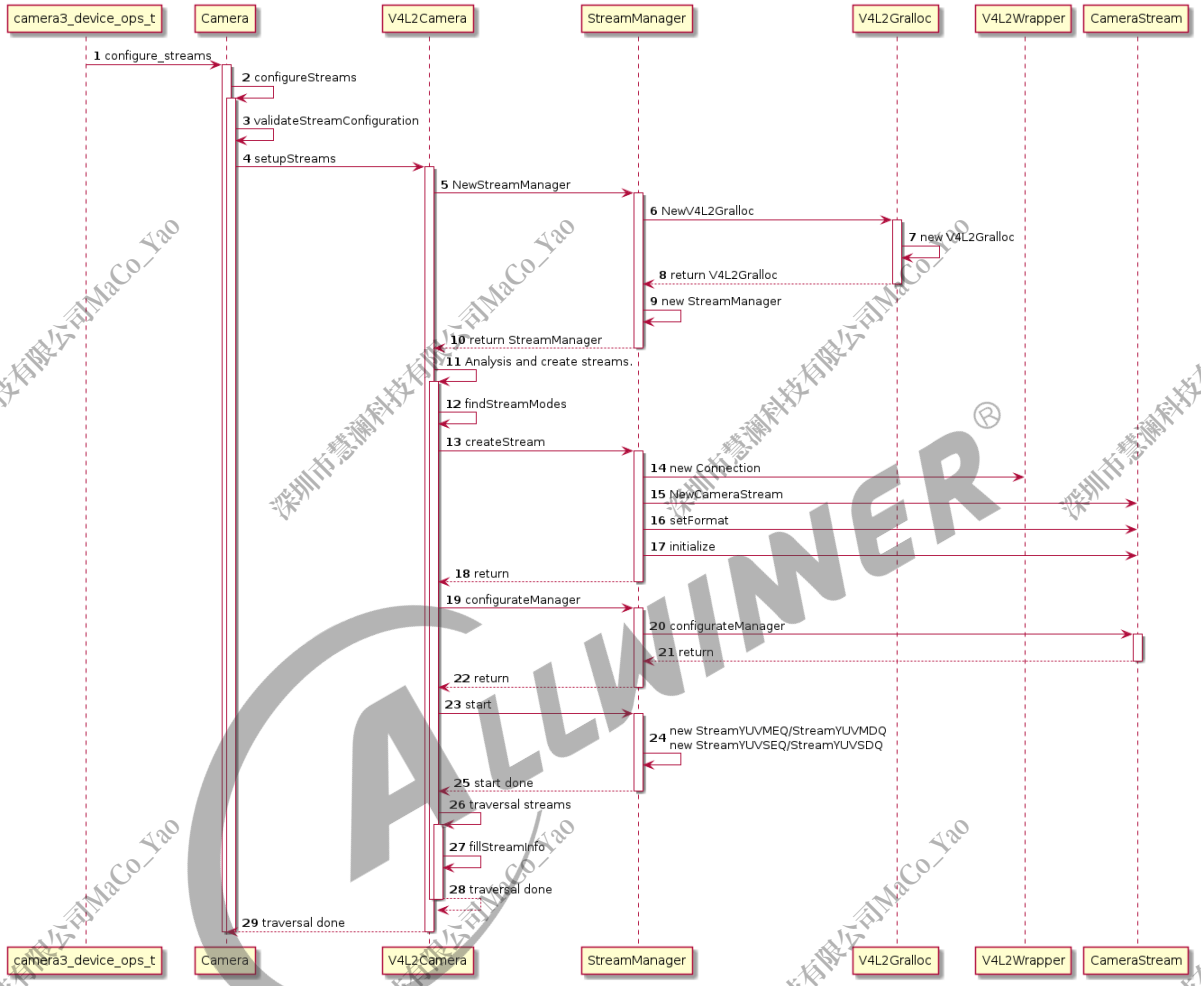

图 5-6: configure_stream

5.3.2.3 construct_default_request_settings

construct_default_request_settings 为通用的 Camera 创建 capture 设置，设备必须返回一个 settings buffer 满足配置要求，并且为 CAMERA3_TEMPLATE_* 的其中一项。HAL 层拥有该结构体，并且指向的指针必须在设备关闭前保持有效。framework 和 HAL 在 construct_default_request_settings 返回前可能不会进行修改。

5.4 数据流向

5.4.1 process_capture_request

process_capture_request 发送一个新的 capture 请求到 HAL。HAL 直到准备好接受下一个 capture 请求前，都不会返回该方法。framework 同一时内只会调用一次 process_capture_request，且调用只会来自同一个线程。下一次使用 process_capture_request 作为一次新的请求会在其对应的 buffers 准备好时调用。在预览的情境下，这意味着方法会被 framework 立即调用。

在整个 process_capture_request 的调用中，从 HAL 层获取 capture 结果的流程是异步的。该调用要求 metadata 必须是有效的，但输出的 buffer 可能会提供 sync fences 用于等待。多个的 request 期望能够同时传输，以保持完整的输出帧率。

framework 拥有 request 结构权。这只能够保证在本次调用中是该 request 请求是有效的。HAL 层设备必须拷贝这些需要的信息以用于 capture 的处理。HAL 层有责任去等待以及关闭 buffer 的 fences 并且返回 buffer 的 handle 到 framework。

假如 input_buffer 不为空，HAL 必须将输入 buffer 的 release sync fence 的文件的描述符写入到 input_buffer->release_fence. 假如 HAL 关于 input buffer 的 release sync fence 返回-1，framework 就能够立即重用这些 input buffer。否则 Framework 将会等待 sync fence 才会去重新填充或者重用输入 buffer。

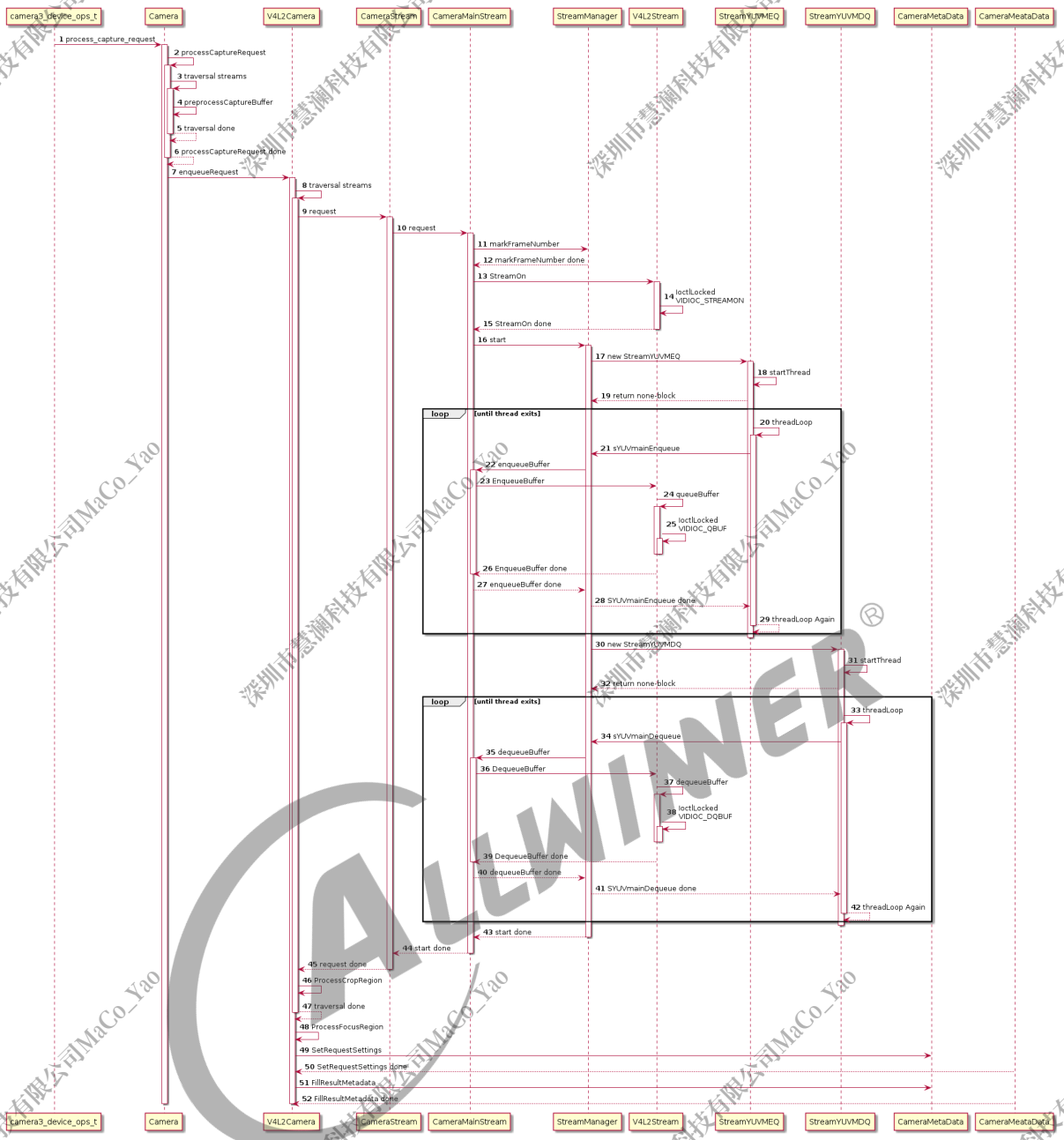

图 5-7: process_capture_request

5.4.2 process_capture_result

发送处理结果到 framework。单次的 capture 请求，可能会被 HAL 层调用多次 process_capture_result。这样的设计能够允许 metadata 以及低分辨率的 buffer 立刻能够被返回，后处理 (post-processed) 的 JPEG buffer 能够在后续准备好后进行调用。每次调用必须包括请求的帧序号以对应 metadata 或者 buffers。在 process_capture_result 调用时，可能只会包含一个 component(buffer 或者 metadata), 每一个 stream 的 buffer 以及 result metadata，在每次 process_capture_result 对应单个请求时，必须被 HAL 返回 (输出时产

生错误时也会返回结果)。不允许一个既不包含 output buffer 或者 result metadata 的 process_capture_result 调用被执行。在同一结果中 metadata 和 buffers 的顺序并不影响。但对于指的 stream 中，buffers 必须是遵循 FIFO 的顺序。stream A 的 request 5 的返回必须要先于 request 6。同理 metadata 也一样。然而不同的流之间独立的，所以 stream A 的 request 5 可能会比 stream B 的 request 6 返回得慢。metadata 同理也一样。HAL 层保留 result 结构的拥有权，只需要保证该数据在本次调用有效即可。framework 将会在调用返回前拷贝它所需要的内容。output buffers 不需要被填充。framework 会在等待流缓冲区的 release sync fence 前读取所有的缓冲区数据。因此该方法应该被 HAL 层尽可能快，尽管部分 output buffer 还可能仍在填充。HAL 必须包括有效的 release sync fences 填充到每个流的 output_buffers entry，或者返回-1 如果 stream buffer 已经被填满了。如果 result buffer 不能够根据一个 request 被构建，HAl 应该返回一个空的 metadata buffer，但仍需要提供输出 buffers 以及他们的 sync fences。并且 notify 方法必须发送 ERROR_RESULT 信息。假如 output buffer 不能够被填满，起 status 字段必须设置为 STATUS_ERROR。并且调用 notify 方法发送 ERROR_BUFFER 信息。如果整个 capture 失败了，那么该方法仍然需要被调用以返回 output buffers 到 framework。所有的 buffer 状态需要设置为 STATUS_ERROR，并且 result metadata 应该为 empty buffer。并且 notify 需要发送 ERROR_REQUEST 信息。在这种情境下，单独的 ERROR_RESULT/ERROR_BUFFER 不应该发送。

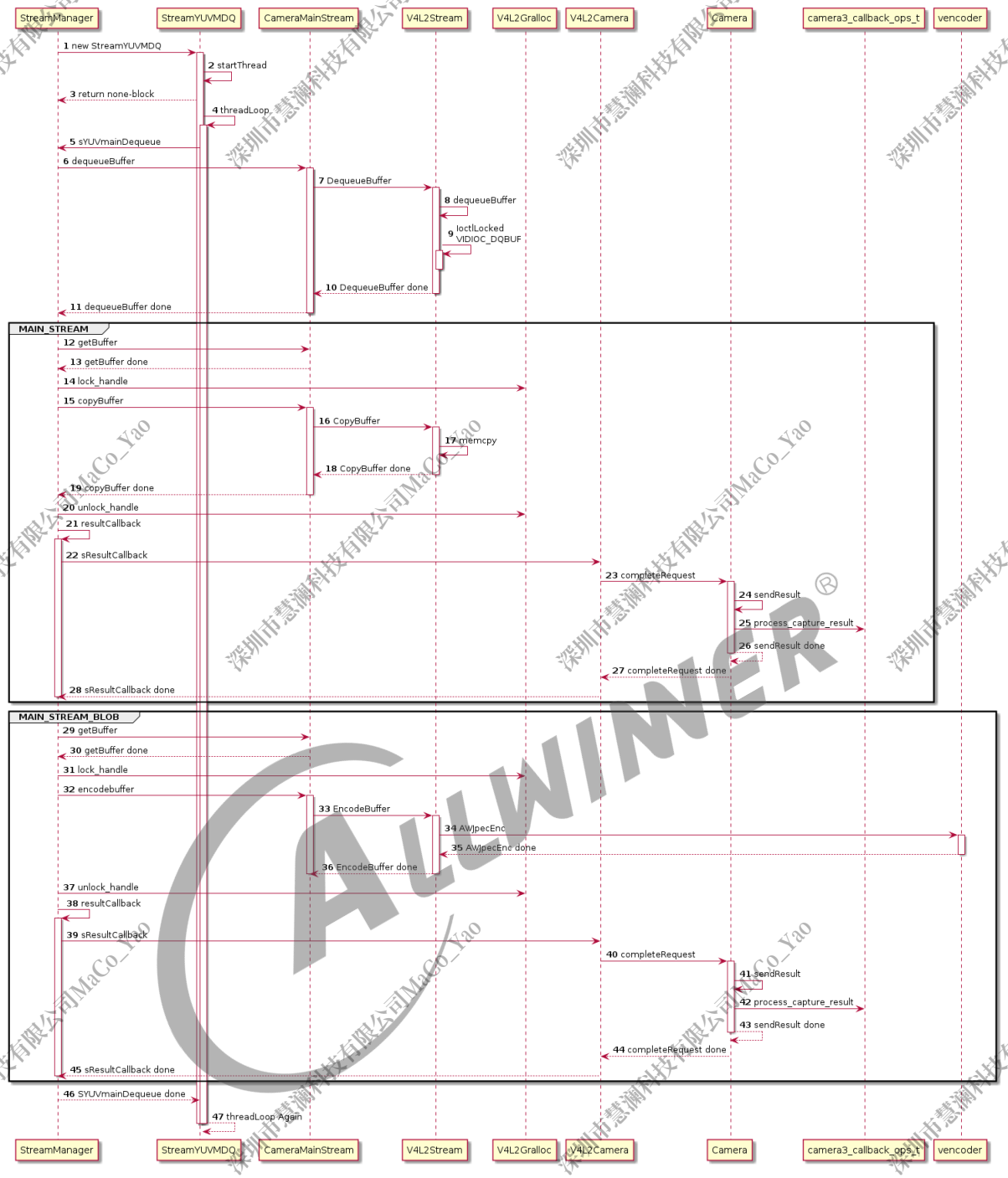

图 5-8: process_capture_result

6 Camera 配置

6.1 内核配置

VIN 驱动目前一定要选择成 ko 加载形式，而且需要依赖于 I2C support。Menuconfig 的配置如下：

```
1.Device Drivers -> <*>I2C support
2.Device Drivers -> <*>Multimedia support -> [*] Cameras/video grabbers support
3.Device Drivers -> <*>Multimedia support -> [*] Media Controller support
4.Device Drivers -> <*>Multimedia support -> [*] V4L2 sub-device userspace API
5.Device Drivers -> <*>Multimedia support -> [*] V4L platform devices -> <M> sunxi video
    input (camera csi/mipi isp vipp)driver
6.Device Drivers -> <*>Multimedia support -> [*] V4L platform devices -> <M> v4l2 new
    driver for SUNXI
7.Device Drivers -> <*>Multimedia support -> [*] V4L platform devices -> <M> use internal
    cci
8.Device Drivers -> <*>Multimedia support -> [*] V4L platform devices -> <M> use flash
    module
9.Device Drivers -> <*>Multimedia support -> [*] V4L platform devices -> <M> use actuator
    module
10.Device Drivers -> <*>Multimedia support -> [*] V4L platform devices -> <M> sensor list
    for adaptive
11.Device Drivers -> <*>Multimedia support -> [*] V4L platform devices -> <*> use vin log
    for debuf
12.Device Drivers -> <*>Multimedia support -> [*] V4L platform devices -> <*> use IOMMU for
     memery alloc
```

在 linux-4.9 下执行 make ARCH=arm64 menuconfig 后可进行配置。

6.2 Device Tree 配置

Soc 节点下需要添加 vind/sensor/actuator/flash/CCI/CSI/MIPI/ISP/VIPP(scaler) 子节点，用于配置内存资源和终端资源, 如下为 board.dts 示例:

```
vind0:vind@0 {
    vind0_clk = <336000000>;
    vind0_isp = <300000000>;
    status = "okay";

    actuator0:actuator@0 {
        device_type = "actuator0";
        actuator0_name = "dw9714_act";
        actuator0_slave = <0x18>;
        actuator0_af_pwdn = <>;
        actuator0_afvdd = "afvcc-csi";
```

```
        actuator0_afvdd_vol = <2800000>;
        status = "okay";
    };
    flash0:flash@0 {
        device_type = "flash0";
        flash0_type = <2>;
        flash0_en = <&r_pio PL 11 1 0 1 0>;
        flash0_mode = <>;
        flash0_flvdd = "";
        flash0_flvdd_vol = <>;
        device_id = <0>;
        status = "okay";
    };
    sensor0:sensor@0 {
        device_type = "sensor0";
        sensor0_mname = "gc5025_mipi";
        sensor0_twi_cci_id = <2>;
        sensor0_twi_addr = <0x6e>;
        sensor0_mclk_id = <0>;
        sensor0_pos = "rear";
        sensor0_isp_used = <1>;
        sensor0_fmt = <1>;
        sensor0_stby_mode = <0>;
        sensor0_vflip = <0>;
        sensor0_hflip = <0>;
        sensor0_cameravdd-supply = <&reg_dldo4>;
        sensor0_cameravdd_vol = <2800000>;
        sensor0_iovdd-supply = <&reg_dldo2>;
        sensor0_iovdd_vol = <1800000>;
        sensor0_avdd-supply = <&reg_dldo3>;
        sensor0_avdd_vol = <2800000>;
        sensor0_dvdd-supply = <&reg_eldo2>;
        sensor0_dvdd_vol = <1200000>;
        sensor0_power_en = <>;
        sensor0_reset = <&pio PE 9 1 0 1 0>;
        sensor0_pwdn = <&pio PE 8 1 0 1 0>;
        status  = "okay";
    };
    sensor1:sensor@1 {
        device_type = "sensor1";
        sensor1_mname = "gc2385_mipi";
        sensor1_twi_cci_id = <2>;
        sensor1_twi_addr = <0x6c>;
        sensor1_mclk_id = <0>;
        sensor1_pos = "front";
        sensor1_isp_used = <1>;
        sensor1_fmt = <1>;
        sensor1_stby_mode = <0>;
        sensor1_vflip = <0>;
        sensor1_hflip = <0>;
        sensor1_iovdd-supply = <&reg_dldo2>;
        sensor1_iovdd_vol = <1800000>;
        sensor1_avdd-supply = <&reg_dldo3>;
        sensor1_avdd_vol = <2800000>;
        sensor1_dvdd-supply = <&reg_eldo2>;
        sensor1_dvdd_vol = <1800000>;
        sensor0_power_en = <>;
        sensor1_reset = <&pio PE 7 1 0 1 0>;
        sensor1_pwdn = <&pio PE 6 1 0 1 0>;
        status  = "okay";
```

```
    };
    vinc0:vinc@0 {
        vinc0_csi_sel = <0>;
        vinc0_mipi_sel = <0>;
        vinc0_isp_sel = <0>;
        vinc0_isp_tx_ch = <0>;
        vinc0_tdm_rx_sel = <0xff>;
        vinc0_rear_sensor_sel = <0>;
        vinc0_front_sensor_sel = <1>;
        vinc0_sensor_list = <0>;
        status = "okay";
    };
    vinc1:vinc@1 {
        vinc1_csi_sel = <0>;
        vinc1_mipi_sel = <0>;
        vinc1_isp_sel = <0>;
        vinc1_isp_tx_ch = <0>;
        vinc1_tdm_rx_sel = <0xff>;
        vinc1_rear_sensor_sel = <0>;
        vinc1_front_sensor_sel = <1>;
        vinc1_sensor_list = <0>;
        status = "okay";
    };
    vinc2:vinc@2 {
        vinc2_csi_sel = <1>;
        vinc2_mipi_sel = <1>;
        vinc2_isp_sel = <0>;
        vinc2_isp_tx_ch = <0>;
        vinc2_tdm_rx_sel = <0xff>;
        vinc2_rear_sensor_sel = <0>;
        vinc2_front_sensor_sel = <1>;
        vinc2_sensor_list = <0>;
        status = "okay";
    };
    vinc3:vinc@3 {
        vinc3_csi_sel = <1>;
        vinc3_mipi_sel = <1>;
        vinc3_isp_sel = <0>;
        vinc3_isp_tx_ch = <0>;
        vinc3_tdm_rx_sel = <0xff>;
        vinc3_rear_sensor_sel = <0>;
        vinc3_front_sensor_sel = <1>;
        vinc3_sensor_list = <0>;
        status = "okay";
    };
}
```

下面对 board.dts 的 vind 节点的各项配置进行说明。

表 6-1: 对焦马达配置

配置	含义
device_type = “actuator0”;	设备类型以及编号
actuator0_name = “ad5820_act”;	马达名，需要和驱动定义的一致
actuator0_slave = <0x18>;	马达 I2C 地址，具体参考驱动或者 datasheet
actuator0_af_pwdn = <>;	马达控制引脚配置
actuator0_afvdd = “afvcc-csi”;	马达供电配置，现已弃用，改在 sensor 节点中配置

配置	含义
actuator0_afvdd_vol = <2800000>; status = “disabled”;	马达供电电压配置，现已弃用，改在 sensor 节点中配置 模块使能，“okay” 为使能，“disabled” 为不使用

对焦马达配置一般只需要配置 actuator0_name、actuator0_slave 即可，最后要把模块使能。

表 6-2: 闪光灯配置

配置	含义
device_type = “flash0”;	设备类型以及编号
flash0_type = <2>;	闪光灯类型，具体看注释 1.
flash0_en = <>;	闪光灯控制引脚，一般用作手电筒功能时会置起该引脚
flash0_mode = <>;	闪光灯控制引脚，一般用作闪光功能时会置起该引脚
flash0_flvdd = “”;	闪光灯供电，现已弃用，如要独立供电，参考对焦马达供电使用方法
flash0_flvdd_vol = <>;	闪光灯供电电压配置，现已弃用
device_id = <0>;	闪光灯编号

注释 1: 一般选择类型 “1” 或者 “2”；两者之间的区别就是类型 “2” 是虚拟闪光灯，即无论是选择闪光灯功能还是选择手电筒功能，都是置起 flash0_en 这个引脚。所以要注意类型和控制引脚之间的搭配。

表 6-3: 摄像头配置

配置	含义
device_type = “sensor0”;	设备类型以及编号
sensor0_mname = “imx278_mipi”;	摄像头名称、型号，需要和驱动定义的一致
sensor0_twi_cci_id = <2>;	摄像头通信的 TWI/CCI ID，如配置为 “2” 则表示使用 TWI2
sensor0_twi_addr = <0x20>;	摄像头 I2C 地址
sensor0_mclk_id = <0>;	mclk ID
sensor0_pos = “rear”;	摄像头位置，“rear”：后摄；“front”：前摄
sensor0_isp_used = <1>;	是否使用 ISP
sensor0_fmt = <1>;	模块使能，“okay” 为使能，“disabled” 为不使用
sensor0_stby_mode = <0>;	休眠模式，默认不使用此配置
sensor0_vflip = <0>;	垂直镜像使能
sensor0_hflip = <0>;	水平镜像使能
sensor0_cameravdd-supply = <>;	一般用作对焦马达供电
sensor0_cameravdd_vol = <>;	对焦马达供电电压
sensor0_iovdd-supply = <&reg_bldo1>;	对应 sensor datasheet 里面的 IOVDD
sensor0_iovdd_vol = <1800000>;	IOVDD 供电电压
sensor0_avdd-supply = <&reg_bldo4>;	对应 sensor datasheet 里面的 AVDD
sensor0_avdd_vol = <2800000>;	AVDD 供电电压

配置	含义
sensor0_dvdd-supply = <&reg_bldo2>;	对应 sensor datasheet 里面的 DVDD
sensor0_dvdd_vol = <1200000>;	DVDD 供电电压
sensor0_power_en = <>;	sensor 控制 IO，可以为空，配置过程看注释 4
sensor0_reset = <&pio PE 7 1 0 1 0>;	sensor 控制 IO，可以为空
sensor0_pwdn = <&pio PE 6 1 0 1 0>;	sensor 控制 IO，可以为空

6.3 驱动加载

Camera 驱动加载定义在 android/device/softwinner/[方案]/init.device.rc 中实现，加载顺序如下，且需保证 vin_v4l2.ko 在最后进加载。假如没有对焦马达，请不要加载 actuator.ko 以及对应的马达型号 ko, 此处使用的是 dw9714 型号。此外 sensor 驱动也需要使用对应的，本例中使用的是 gc2385 和 gc5025。

```
on late-fs
### csi module
    insmod /vendor/modules/videobuf2-core.ko
    insmod /vendor/modules/videobuf2-memops.ko
    insmod /vendor/modules/videobuf2-dma-contig.ko
    insmod /vendor/modules/videobuf2-v4l2.ko
    insmod /vendor/modules/vin_io.ko
    insmod /vendor/modules/gc2385_mipi.ko
    insmod /vendor/modules/gc5025_mipi.ko
    insmod /vendor/modules/actuator.ko
    insmod /vendor/modules/dw9714_act.ko
    insmod /vendor/modules/vin_v4l2.ko
```

6.4 Camera.cfg

Camera.cfg 用于对 Camera 进行配置，路径位于 SDK 的 device/softwinner/[方案]/configs 下。也可以直接通过 adb push 到设备目路/vendor/etc/中，重启即可生效。通用的 Camera.cfg 模板如下:

```
;------------------------------------------------------------------------------
; 用于camera的配置
;
; 采用格式:
; key = key_value
; 注意: 每个key需要顶格写;
;       key_value紧跟着key后面的等号后面, 位于同一行中;
;       key_value限制大小为256字节以内;
;
;------------------------------------------------------------------------------

;------------------------------------------------------------------------------
; exif information of "make" and "model"
;------------------------------------------------------------------------------
```

```
key_camera_exif_make = MAKE_AllWinner
key_camera_exif_model = PRODUCT_BOARD

;------------------------------------------------------------------------------
; 1 for single camera, 2 for double camera
;------------------------------------------------------------------------------
number_of_camera = 2

;------------------------------------------------------------------------------
; 0 for no support for Time-division multiplexing
; 1 for support which means you can open all camera at the same time
;------------------------------------------------------------------------------
use_camera_multiplexing = 0

;------------------------------------------------------------------------------
; CAMERA_FACING_BACK
; gc2355
;------------------------------------------------------------------------------
camera_id = 0

;------------------------------------------------------------------------------
; 1 for CAMERA_FACING_FRONT
; 0 for CAMERA_FACING_BACK
;------------------------------------------------------------------------------
camera_facing = 0

;------------------------------------------------------------------------------
; 1 for camera without isp(using built-in isp of Axx)
; 0 for camera with isp
;------------------------------------------------------------------------------
use_builtin_isp = 0

;------------------------------------------------------------------------------
; camera orientation (0, 90, 180, 270)
;------------------------------------------------------------------------------
camera_orientation = 0

;------------------------------------------------------------------------------
; driver device name
;------------------------------------------------------------------------------
camera_device = /dev/video0

;------------------------------------------------------------------------------
; device id
; for two camera devices with one CSI
;------------------------------------------------------------------------------
device_id = 0

used_preview_size = 1
key_support_preview_size = 800x600,640x480,320x240,176x144
key_default_preview_size = 800x600

used_picture_size = 1
key_support_picture_size = 4224x3136,2592x1944,1600x1200,1280x720,640x480
key_default_picture_size = 4224x3136

used_interpolation_size = 1
key_support_src_interpolation_size = 2592x1944
key_default_dst_interpolation_size = 4224x3136
```

```
used_flash_mode = 0
key_support_flash_mode = on,off,auto
key_default_flash_mode = off

used_color_effect=1
key_support_color_effect = none,mono,negative,sepia,aqua
key_default_color_effect = none

used_frame_rate = 1
key_support_frame_rate = 30
key_default_frame_rate = 30

used_focus_mode = 1
key_support_focus_mode = auto,infinity,macro,fixed,continuous-video,continuous-picture
key_default_focus_mode = auto

used_scene_mode = 0
key_support_scene_mode = auto,portrait,landscape,night,night-portrait,theatre,beach,snow,
    sunset,steadyphoto,fireworks,sports,party,candlelight,barcode
key_default_scene_mode = auto

used_white_balance = 1
key_support_white_balance = auto,incandescent,fluorescent,warm-fluorescent,daylight,cloudy-
    daylight
key_default_white_balance = auto

used_exposure_compensation = 1
key_max_exposure_compensation = 4
key_min_exposure_compensation = -4
key_step_exposure_compensation = 1
key_default_exposure_compensation = 0

used_zoom = 1
key_zoom_supported = true
key_smooth_zoom_supported = false
key_zoom_ratios = 100,120,150,200,230,250,300
key_max_zoom = 2
key_default_zoom = 0
key_horizonal_view_angle = 48.6
key_vertical_view_angle = 37.0

;------------------------------------------------------------------------------
; CAMERA_FACING_FRONT
; gc0310
;------------------------------------------------------------------------------
camera_id = 1

;------------------------------------------------------------------------------
; 1 for camera without isp(using built-in isp of Axx)
; 0 for camera with isp
;------------------------------------------------------------------------------
use_builtin_isp = 0

;------------------------------------------------------------------------------
; 1 for CAMERA_FACING_FRONT
; 0 for CAMERA_FACING_BACK
;------------------------------------------------------------------------------
camera_facing = 1

;------------------------------------------------------------------------------
```

```
; camera orientation (0, 90, 180, 270)
;------------------------------------------------------------------------------
camera_orientation = 0

;------------------------------------------------------------------------------
; driver device name
;------------------------------------------------------------------------------
camera_device = /dev/video1

;------------------------------------------------------------------------------
; device id
; for two camera devices with one CSI
;------------------------------------------------------------------------------
device_id = 1

used_preview_size = 1
key_support_preview_size = 640x480,320x240
key_default_preview_size = 640x480

used_picture_size = 1
key_support_picture_size = 2592x1944,1600x1200,1280x720,640x480,320x240
key_default_picture_size = 2592x1944

used_interpolation_size = 1
key_support_src_interpolation_size = 1600x1200
key_default_dst_interpolation_size = 2592x1944

used_flash_mode = 0
key_support_flash_mode = on,off,auto
key_default_flash_mode = on

used_color_effect= 1
key_support_color_effect = none,mono,negative,sepia,aqua
key_default_color_effect = none

used_frame_rate = 1
key_support_frame_rate = 30
key_default_frame_rate = 30

used_focus_mode = 0
key_support_focus_mode = auto,infinity,macro,fixed
key_default_focus_mode = auto

used_scene_mode = 0
key_support_scene_mode = auto,portrait,landscape,night,night-portrait,theatre,beach,snow,
    sunset,steadyphoto,fireworks,sports,party,candlelight,barcode
key_default_scene_mode = auto

used_white_balance = 0
key_support_white_balance = auto,incandescent,fluorescent,warm-fluorescent,daylight,cloudy-
    daylight
key_default_white_balance = auto

used_exposure_compensation = 1
key_max_exposure_compensation = 3
key_min_exposure_compensation = -3
key_step_exposure_compensation = 1
key_default_exposure_compensation = 0

used_zoom = 0
```

```
key_zoom_supported = true
key_smooth_zoom_supported = false
key_zoom_ratios = 100,120,150,200,230,250,300
key_max_zoom = 2
key_default_zoom = 0
key_horizonal_view_angle = 44.3
key_vertical_view_angle = 33.9
```

6.5 media_profiles.xml

media_profiles.xml 主要用于保存 Camera 支持摄像相关参数，包括摄像质量，音视频编码格式，帧率，比特率等，该参数主要有摄像头厂商提供，如下为两个摄像头的配置：

```
# device/softwinner/[方案]/configs/media_profiles.xml

 <CamcorderProfiles cameraId="0" >
        <EncoderProfile quality="480p" fileFormat="mp4" duration="30">
            <Video codec="h264"
                   bitRate="1500000"
                   width="640"
                   height="480"
                   frameRate="30" />

            <Audio codec="aac"
                   bitRate="12200"
                   sampleRate="8000"
                   channels="1" />
        </EncoderProfile>
        <EncoderProfile quality="timelapse480p" fileFormat="mp4" duration="30">
            <Video codec="h264"
                   bitRate="1500000"
                   width="640"
                   height="480"
                   frameRate="30" />

            <Audio codec="aac"
                   bitRate="12200"
                   sampleRate="8000"
                   channels="1" />
        </EncoderProfile>


        <ImageEncoding quality="90" />
        <ImageEncoding quality="80" />
        <ImageEncoding quality="70" />
        <ImageDecoding memCap="20000000" />


  </CamcorderProfiles>
  <CamcorderProfiles cameraId="1">

        <EncoderProfile quality="480p" fileFormat="mp4" duration="30">
            <Video codec="h264"
                   bitRate="1500000"
                   width="640"
                   height="480"
```

```
                frameRate="30" />

            <Audio codec="aac"
                   bitRate="12200"
                   sampleRate="8000"
                   channels="1" />
        </EncoderProfile>

        <EncoderProfile quality="timelapse480p" fileFormat="mp4" duration="30">
            <Video codec="h264"
                   bitRate="1500000"
                   width="640"
                   height="480"
                   frameRate="30" />

            <Audio codec="aac"
                   bitRate="12200"
                   sampleRate="8000"
                   channels="1" />
        </EncoderProfile>

        <ImageEncoding quality="90" />
        <ImageEncoding quality="80" />
        <ImageEncoding quality="70" />
        <ImageDecoding memCap="20000000" />

  </CamcorderProfiles>
```

6.6 手电筒配置

如果机器带有闪光灯，要想使用通知栏设置里面的手电筒，需要到如下目录下的文件中，增加如下内容：

```
#android/device/softwinner/common/config/tablet_core_hardware.xml
<feature name="android.hardware.camera.flash" />
# camera.cfg
used_flash_mode = 1
```

6.7 USB Camera 配置

6.7.1 内核配置

如果需要设置 USB Camera，首先需要在内核进行配置，可直接修改 lichee/linux-4.9/arch/arm64/configs/[芯片平台]_android_defconfig

```
CONFIG_MEDIA_USB_SUPPORT=y
CONFIG_USB_VIDEO_CLASS=m
```

也可以通过 make menuconfig arch=arm64 进行设置，路径为:

```
Symbol: USB_VIDEO_CLASS [=n]
  | Type  : tristate
  | Prompt: USB Video Class (UVC)
  |   Location:
  |     -> Device Drivers
  |       -> Multimedia support (MEDIA_SUPPORT [=y])
  | (1)     -> Media USB Adapters (MEDIA_USB_SUPPORT [=n])
```

```
--- Multimedia support
      *** Multimedia core support ***
[*]   Cameras/video grabbers support
[ ]   Analog TV support
[ ]   Digital TV support
[ ]   AM/FM radio receivers/transmitters support
[ ]   Software defined radio support
[*]   Remote Controller support
[*]   Media Controller API
[ ]     Enable Media controller for DVB (EXPERIMENTAL)
[*]   V4L2 sub-device userspace API
[ ]   Enable advanced debug functionality on V4L2 drivers
[ ]   Enable old-style fixed minor ranges on drivers/video devices
      *** Media drivers ***
<*>   Compile Remote Controller keymap modules
[ ]   Remote controller decoders  ----
[*]   Remote Controller devices  --->
[*]   Media USB Adapters  --->
[*]   V4L platform devices  --->
[ ]   Memory-to-memory multimedia devices  ----
[ ]   Media test drivers  ----
      *** Supported MMC/SDIO adapters ***
< >   Cypress firmware helper routines
      *** Media ancillary drivers (tuners, sensors, i2c, spi, frontends) ***
[*]   Autoselect ancillary drivers (tuners, sensors, i2c, spi, frontends)
      Sensors used on soc_camera driver  ----
<*>   sunxi video encoder and decoder support
< >   google video vp9 decoder support
< >   AW tsc module support
```

图 6-1: CONFIG_MEDIA_USB_SUPPORT

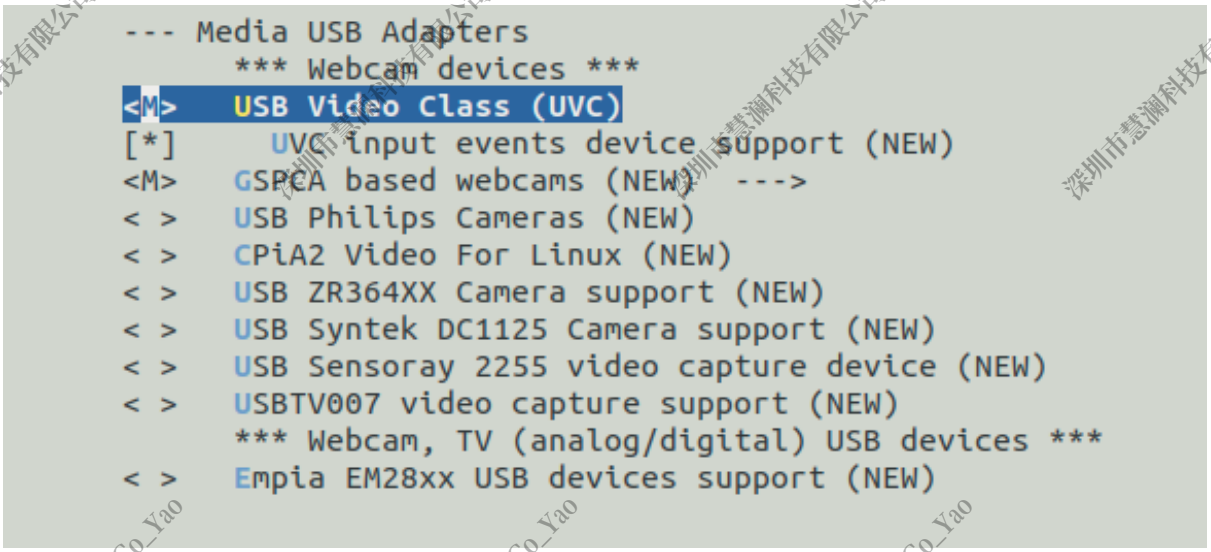

图 6-2: CONFIG_USB_VIDEO_CLASS

6.7.2 方案配置

配置了内核后，需要在启动时加载 ko 文件，可在方案目录 device/softwinner/[方案]/init.device.rc 中增加 uvcvideo.ko 以及 videobuf2-vmalloc.ko。

```
on late-fs
### csi&uvc module
    insmod /vendor/modules/videobuf2-core.ko
    insmod /vendor/modules/videobuf2-memops.ko
    insmod /vendor/modules/videobuf2-dma-contig.ko
    insmod /vendor/modules/videobuf2-vmalloc.ko
    insmod /vendor/modules/videobuf2-v4l2.ko
    insmod /vendor/modules/vin_io.ko
    insmod /vendor/modules/gc2385_mipi.ko
    insmod /vendor/modules/gc030a_mipi.ko
    insmod /vendor/modules/imx278_mipi.ko
    insmod /vendor/modules/imx386_2lane_mipi.ko
    insmod /vendor/modules/vin_v4l2.ko
    insmod /vendor/modules/uvcvideo.ko
```

同时，需要修改节点权限，目录为 device/softwinner/[方案]-common/ueventd.[芯片平台].rc

```
/dev/video*                0777    camera      camera
```

6.7.3 Selinux 配置

为了使 USB Camera 应用正常运行，需要增加 Selinux 规则，如下:

路径: device/softwinner/common/sepolicy/vendor/genfs_contexts

```
genfscon sysfs /devices/platform/soc/5200000.ehci1-controller u:object_r:sysfs_usb:s0
genfscon sysfs /bus/usb u:object_r:sysfs_usb:s0
```

路径: device/softwinner/pluto-demo / sepolicy/public/untrusted_app.te

```
allow untrusted_app video_device:chr_file { getattr ioctl open read write };
```

分别在 system/sepolicy/public/app.te 以及/system/sepolicy/prebuilts/api/29.0/public/app.te 去掉 video_device。

```
# Access to any of the following character devices.
neverallow appdomain {
    audio_device
    camera_device
    dm_device
    radio_device
    rpmsg_device
    #video_device
}:chr_file { read write };
```

在 system/sepolicy/public/device.te 以及/system/sepolicy/prebuilts/api/29.0/public/device.te 增加如下规则:

```
type video_device, dev_type, mlstrustedobject;
```

7 编译

7.1 HAL3 编译流程

编译前需要检查如下配置:

1.hal.mk

```
# CAMERA
PRODUCT_PACKAGES += \
    camera.device@3.2-impl \
    android.hardware.camera.provider@2.4-service-lazy \
    android.hardware.camera.provider@2.4-impl \
    libcamera \
    camera.[方案]
```

2.[方案].mk

```
# device/softwinner/[方案]/[方案].mk
USE_CAMERA_HAL_3_4 := true
```

3.manifest.xml

```
#device/softwinner/[方案]/configs/manifest.xml
<hal format="hidl">
    <name>android.hardware.camera.provider</name>
    <transport>hwbinder</transport>
    <version>2.4</version>
    <interface>
        <name>ICameraProvider</name>
        <instance>legacy/0</instance>
    </interface>
</hal>
```

保证上述修改后，可以生成对应的库文件了，方法如下:

```
source build/enveseupt.sh
lunch [方案]
cd /hardware/aw/camera/3_4
mm
```

生成的 HAL3 库位置在 out/target/product/[方案]/vendor/lib/hw/camera.[方案].so

可通过将库推到 Android 设备进行调试:

```
adb root
adb remount
adb push camera.[方案].so /vendor/lib/hw/
adb reboot
```

8 调试

8.1 HAL3 打印调试开关

在 HAL3 中为了调试方便，可以通过打开调试开打印调试信息，可修改文件 android/hardware/aw/camera/3_4/common.h，将打印等级调至 4, 如下所示:

```
#define LOG_LEVEL                       4
#define HAL_LOGE(fmt, args...) if(LOG_LEVEL >= 0) ALOGE("%s:%d: " fmt, __func__, __LINE__,
    ##args)
#define HAL_LOGW(fmt, args...) if(LOG_LEVEL > 1) ALOGW("%s:%d: " fmt, __func__, __LINE__,
    ##args)
#define HAL_LOGI(fmt, args...) if(LOG_LEVEL > 2) ALOGI("%s:%d: " fmt, __func__, __LINE__,
    ##args)
#define HAL_LOGD(fmt, args...) if(LOG_LEVEL > 3) ALOGD("%s:%d: " fmt, __func__, __LINE__,
    ##args)
#define HAL_LOGV(fmt, args...) if(LOG_LEVEL > 4) ALOGV("%s:%d: " fmt, __func__, __LINE__,
    ##args)
#define HAL_LOGE_IF(cond, fmt, args...) if(LOG_LEVEL == 0) \
    ALOGE_IF(cond, "%s:%d: " fmt, __func__, __LINE__, ##args)
#define HAL_LOGW_IF(cond, fmt, args...) if(LOG_LEVEL > 0) \
    ALOGW_IF(cond, "%s:%d: " fmt, __func__, __LINE__, ##args)
#define HAL_LOGI_IF(cond, fmt, args...) if(LOG_LEVEL > 1) \
    ALOGI_IF(cond, "%s:%d: " fmt, __func__, __LINE__, ##args)
```

8.2 保存帧数据

在 Camera 的调试中，为了界定问题，可使用函数 saveBuffers 进行单帧数据或者多帧数据的保存。其定义如下, 其中 str 为帧数据将要保存的路径名，p 为指向数据地址的指针，length 为数据的长度，is_oneframe 指的是保存为单帧或者多帧数据。

```
//hardware/aw/camera/3_4/common.cpp
bool saveBuffers(char *str,void *p, unsigned int length,bool is_oneframe)
{
    int fd;
    HAL_LOGD("Debug to save a frame!");
    if((access(str,0) != -1)&&(is_oneframe)) {
        HAL_LOGD("File %s is exists!!!\n",str);
    }
    if(is_oneframe)
        fd = open(str,O_CREAT|O_RDWR|O_TRUNC,0777);        //save one frame data
    else
        fd = open(str,O_CREAT|O_RDWR|O_APPEND,0777);       //save more frames
    if(!fd) {
        HAL_LOGE("Open file error");
        return false;
```

```
    }
    if(write(fd,p,length)){
        HAL_LOGD("Write file successfully");
        close(fd);
        return true;
    }
    else {
        HAL_LOGE("Write file fail");
        close(fd);
        return false;
    }
}
```

saveBuffers 可用于进行硬编码前后，从而界定是 Camera 驱动输出的 yuv 数据有异常，还是经过了 VE 硬编码的 JPEG 数据异常，如:

```
//将硬编码前的yuv数据保存至/data/camera/yuv.bin,且为单帧数据。
saveBuffers("/data/camera/yuv.bin",src_addr,jpeg_enc.src_w * jpeg_enc.src_h*3/2,true);
int ret = AWJpecEnc(&sjpegInfo, &exifInfo, (void *)jpeg_buf, &bufSize);
//将硬编码后的jpeg数据保存在/data/camera/test.jpg，且为单帧数据。
saveBuffers("/data/camera/test.jpg",(void *)jpeg_buf,bufSize,true);
```

Android 设备也需要进行如下操作:

1. 创建调试目录，上述例子需要创建/data/camera, 可运行命令 mkdir /data/camera。
2. 修改调试目录权限，chmod 777 /data/camera。
3. 关闭 selinux 权限，setenforce 0。

生成的 yuv 数据可以通过 RawViewer 进行查看，可查看单帧或多帧数据。

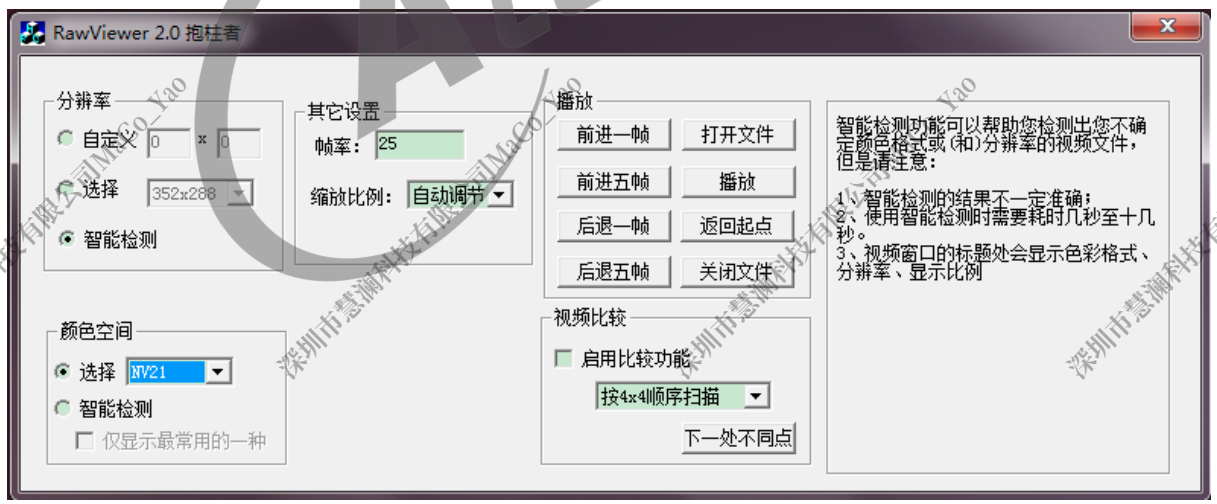

图 8-1: RawViewer 示例图

8.3 检查驱动节点

当拍照异常时，可先查看/dev/video*，如果 dev 目录下没有生成 video 节点，即为注册 video 节点失败。此时有两种方式检查节点生成失败的原因。

1. 检查模块加载顺序是否确 lsmod 看一下模块是否加载正确如果报的错误是 [VIN_ERR]registering gc2355_mipi, No such device! 则表明 sensor 模块 gc2355_mipi 没有加载。
2. 检查 board.dts 文件配置是否配置了 vind0, 且 status 为 okay。

8.4 Camera 闪退异常

Camera 闪退异常一般是在 camera hal3 初始化阶段出现了问题，主要关注如下四个文件：

1. v4l2_camera_hal.cpp
2. v4l2_wrapper.cpp
3. board.dts
4. camera.cfg

当 Camera 闪退异常发生时，可先使用 9.1.3 的方法检查驱动节点是否存在，再检查 camera.cfg 前后摄像头配置是否匹配，最后检查上报的 camera_info 信息 (如摄像头的数量) 是否一致。

8.5 Camera 预览黑屏

该问题通常是在参数配置阶段，或者获取和设置 v4l2 驱动层数据的时候出现了问题。主要关注如下几个文件：

1. v4l2_metadata_factory.cpp
2. v4l2_stream.cpp

8.6 Camera 应用预览画面倒置或者翻转

该问题一般是设置给 framework 层的 orientation 不对，或者是驱动层的水平和垂直的镜像翻转没有设置对。

可通过修改 camera.cfg 修正方向, 但需要注意对应的 sensor，前摄还后摄。

```
;-------------------------------------------------------------------------------
; camera orientation (0, 90, 180, 270)
;-------------------------------------------------------------------------------
camera_orientation = 0
```

驱动层的水平和垂直镜像翻转可以通过修改 sensor_list_cfg.ini, 路径为 android/device/softwinner/[方案]/hawkview/sensor_list_cfg.ini:

```
sensor_hflip0           = 1
sensor_vflip0           = 1
```

8.7 Camera 录像异常

打开 camera app 时，预览正常，但录像异常，该问题一般是配置的分辨率或者帧率有问题导致的，主要关注如下几个文件：

1. media_profiles.xml
2. camera.cfg

```
# media_profiles.xml
<EncoderProfile quality="480p" fileFormat="mp4" duration="30">
            <Video codec="h264"
                   bitRate="1500000"
                   width="640"
                   height="480"
                   frameRate="30" />
</EncoderProfile>
```

```
# camera.cfg
used_frame_rate = 1
key_support_frame_rate = 30
key_default_frame_rate = 30
```

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


、 
、全志科技、  （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。