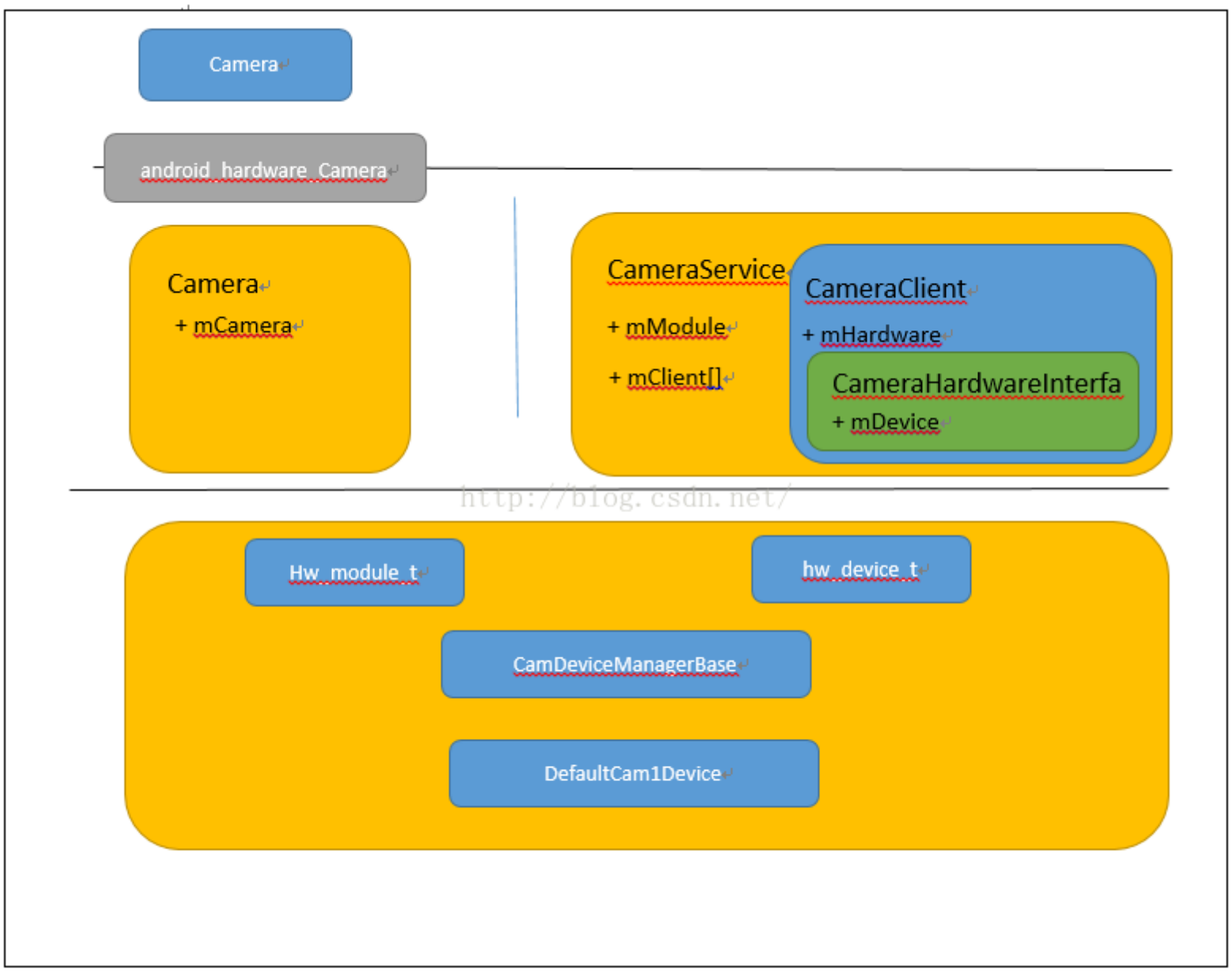
Android5.0 MTk Camera HAL 层代码分析

1. **Android** Camera 框架



如上图为 Camera 的主要框架，其中最上面的 Camera.**Java** 是应用的使用的接口，它处理维护一个在 java 层的状态外核心功能都是通过 Android_hardware_Camera 这个 JNI 调到 C++层实现的。其实他对应的是 C++层的一个同名的 Camera 类。

在 C++层的 Camera 类其实是 Binder 的 client 端，对应的 Service 端是 CameraService。每次请求 Camera 服务时会在 CameraService 端创建一个 CameraClient，并保存在 mClient 数组里面，同时返回给 Client 保存在 Camera 类的 mCamera 对象里面。这样子 CameraService 和 Client 端的 Camera.cpp 对象就彻底撒手。对后续的关于 Camera 操作都是 java 层调用 Camera.cpp,然后直接转交给 mCamera，也就是调用 CameraClient 的方法。CameraService 里面的 mClient 数组的最大长度就是 Camera 的个数。CameraService 在第一次 引用的时候会通过 HAL 标准规范获得一个 hw_module_t 对象 mModule。

CameraClient 在创建的时候会获得 mModule 对象，同时还会创建一个类型为 CameraHardwareInterface 的 mHardware 对象，并通过 mModule 的 open 函数获得一个 HAL 设备对象 hw_device_t mDevice，这个 mDevice 与 HAL 层里面的 DefaultCam1Device 里面的一个属性对应。CameraDeviceManagerBase 顾名思义就是管理 Cam1Device 的。如此 Camera 的操作就在 CameraClient 中交给 mHardWare，进一步交给 mDevice，再进一步到了 Cam1Device。

2. 关于 Camera 的主要代码目录：

l Java 层的：

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

l Jni

frameworks/base/core/jni/android_hardware_Camera.cpp

l Client 端

frameworks/av/camera/Camera.cpp

frameworks/av/camera/CameraBase.cpp

frameworks/av/camera/ICameraService.cpp

l Service 端

frameworks/av/services/camera/libcameraservice/CameraService.cpp

frameworks/av/services/camera/libcameraservice/api1/CameraClient.cpp

frameworks/av/services/camera/libcameraservice/device1/CameraHardwareInterface.cpp

l HAL 层

vendor\mediatek\proprietary\hardware\mtkcam\module_hal\module\module.h

vendor\mediatek\proprietary\hardware\mtkcam\module_hal\devicemgr\CamDeviceManagerBase.cpp

vendor\mediatek\proprietary\hardware\mtkcam\v1\device\Cam1DeviceBase.cpp

3. 主要的调用过程：

1. camera 的打开的过程：

Camera.open()---->native_setup()----> android_hardware_Camera_native_setup() ----> Camera::connect()---->CameraBase::connect() ----> CameraService::conect()---->CameraService::connectHelperLocked() ----> new CameraClient()。

2.设置 Preview 窗口的过程：

Camera.setPreviewDisplay() ----> setPreviewSurface----> android_hardware_Camera_setPreviewSurface() ---->Camera:: setPreviewTarget()----> c->setPreviewTarget() ---->CameraClient ----> CameraClient::setPreviewTarget()----> CameraClient::setPreviewWindow ----> CameraHardwareInterface::setPreviewWindow ----> mDevice->ops->set_preview_window (hw_device_t) ---->Cam1Device:: camera_set_preview_window() ----> Cam1Device:: setPreviewWindow()----> Cam1DeviceBase::setPreviewWindow() ----> Cam1DeviceBase:: initDisplayClient() ----> Cam1DeviceBase::initDisplayClient() ----> DisplayClient::setWindow() ----> DisplayClient:: set_preview_stream_ops() ----> mpStreamOps = window

3.数据被读取到图像缓冲区的过程：

Cam1DeviceBase::setPreviewWindow() ----> Cam1DeviceBase::initDisplayClient()---->IDisplayClient::createInstance() ----> DisplayClient::init()----> createDisplayThread()&& createImgBufQueue()---->Cam1DeviceBase::enableDisplayClient()---->DisplayClient::enableDisplay()---->DisplayClient::enableDisplay()---> mpDisplayThread-> postCommand(Command(Command::eID_WAKEUP)) ---->DisplayThread::threadLoop() ----> DisplayClient::onThreadLoop()---->DisplayClient::waitAndHandleReturnBuffers ---->rpBufQueue->dequeProcessor(vQueNode)----> DisplayClient::handleReturnBuffers() ----> enquePrvOps() ----> mpStreamOps->enqueue_buffer(mpStreamOps,rpImgBuf->getBufHndlPtr())

所以总体来说是在 DisplayClient 的 init 的时候创建了一个线程:

mpDisplayThread =IDisplayThread::createInstance(this)

一个队列：

mpImgBufQueue = newImgBufQueue(IImgBufProvider::eID_DISPLAY, "CameraDisplay@ImgBufQue");

然后 mpDisplayThread 等待 ImgBufQueue 有数据的时候通过 dequeProcessor 取到要渲染的数据，交给 mpStreamOps 也就是 preview_stream_ops 对象，其实 preview_stream_ops 只是对 ANativeWindow 的简单封装，最后调用的其实是 ANativeWindow 的 queueBuffer 函数，到这里一个流程基本结束，其实还是比较简单的，一个线程等待一个队列，有数据后把他塞到 ANativeWindow 中

4.在 preview 的时候数据的产生过程：

IImgBufQueue 继承于 IImgBufProvider 和 IImgBufProcessor 数据通过 IImgBufProvider 提供的接口写入，通过 IImgBufProcessor 的接口消费，ImgBufProvidersManager 是个大管家，管理着所有的 IImgBufProvider，可以通过 getDisplayPvdr(),getRecCBPvdr()等接口获取到对应的 IImgBufProvider，然后把数据塞给他，于是对应的饥肠辘辘的消费者就开始消费了，比如 DisplayClient。

IImgBufProvider 的设置过程：

DisplayClient::enableDisplay ()---->DisplayClient::setImgBufProviderClient (mpCamAdapter)----> IImgBufProviderClient::onImgBufProviderCreated(mpImgBufQueue) ---->BaseCamAdapter:: onImgBufProviderCreated(sp<IImgBufProvider>const&rpProvider) ----> ImgBufProvidersManager::setProvider()

Camera 的数据来源

DefaultCam1Device::onStartPreview()----> Cam1DeviceBase::initCameraAdapter() ----> CamAdapter::init() ---->IPreviewCmdQueThread::createInstance() ----> CamAdapter::startPreview() ---->StateIdle::onStartPreview() ----> CamAdapter::onHandleStartPreview() ---->mpPreviewCmdQueThread->postCommand(PrvCmdCookie::eUpdate, PrvCmdCookie::eSemBefore)----> PreviewCmdQueThread::threadLoop()----> PreviewCmdQueThread::update()----> PreviewCmdQueThread::updateOne() ----> PreviewBufMgr::enqueBuffer()----> IImgBufProvider:: enqueProvider()

5.Camera 回调的过程：

以 takepicture 为例，Callback 的设置过程

mDevice->ops->set_callbacks--> Cam1Device::camera_set_callbacks() --> Cam1DeviceBase::setCallbacks--> mpCamMsgCbInfo->mDataCb=data_cb

拍照的流程：

CamAdapter::takePicture()-> IState::onCapture() ->IStateHandler::onHandleCapture() ->CamAdapter::onHandleCapture()CamAdapter::onHandleCapture() ->CaptureCmdQueThread::onCapture() -> CaptureCmdQueThread::threadLoop() ->CamAdapter::onCaptureThreadLoop()->CapBufShot::sendCommand()->CapBufShot::onCmd_capture()-> SingleShot::startOne()在这里组装 IImgBuffer

回调的流程：

SingleShot::startOne() --> CamShotImp::handleDataCallback() --> CamShotImp::onDataCallback--> CapBufShot::fgCamShotDataCb --> CapBufShot::handleJpegData -->CamAdapter::onCB_RawImage -->mpCamMsgCbInfo->mDataCb

##注意两个两个标红的和标蓝的，他们这样就头尾对起来了。

==========分割线

1.所谓的 Camera 主要就是从摄像头取一点数据放到 LCD 上显示，所以打蛇打七寸，我们首先介绍 HAL 层的 Window 对象是 mpStrwamOps。而这个对象被设置的流程是：

Cam1DeviceBase::setPreviewWindow() ---> initDisplayClient() ---> DisplayClient::setWindow() ---> set_preview_stream_ops() ---> mpStreamOps = window

2.另一个面我们看下数据的获取流程，Camera HAL 层数据主要通过 ImgBufQueue 对象传递，而这个对象最开始的赋值过程如下：

CamAdapter::takePicture() -> IState::onCapture() ->IStateHandler::onHandleCapture() ->
CamAdapter::onHandleCapture()CamAdapter::onHandleCapture() -> CaptureCmdQueThread::onCapture() ->
CaptureCmdQueThread::threadLoop() ->
CamAdapter::onCaptureThreadLoop()->CapBufShot::sendCommand()->CapBufShot::onCmd_capture() ->
SingleShot::startOne()在这里组装 IImgBuffer

推荐文章：

http://www.itdadao.com/articles/c15a674054p0.html
http://blog.csdn.NET/shell812/article/details/49425763