Introduction for Camera

Heaton

Contents

- **Camera Framework**
  - Camera Architecture
  - From java to hardware: Camera.open flow
  - startPreview()
  - From hardware to Java: Jpeg Callback flow

Camera Framework

Camera Architecture

Camera JNI :

frameworks/base/core/jni/android_hardware_Camera.cpp

(libandroid_runtime.so)

Camera client:

Header files:

frameworks\av\include\camera

CPP files:

frameworks\av\camera

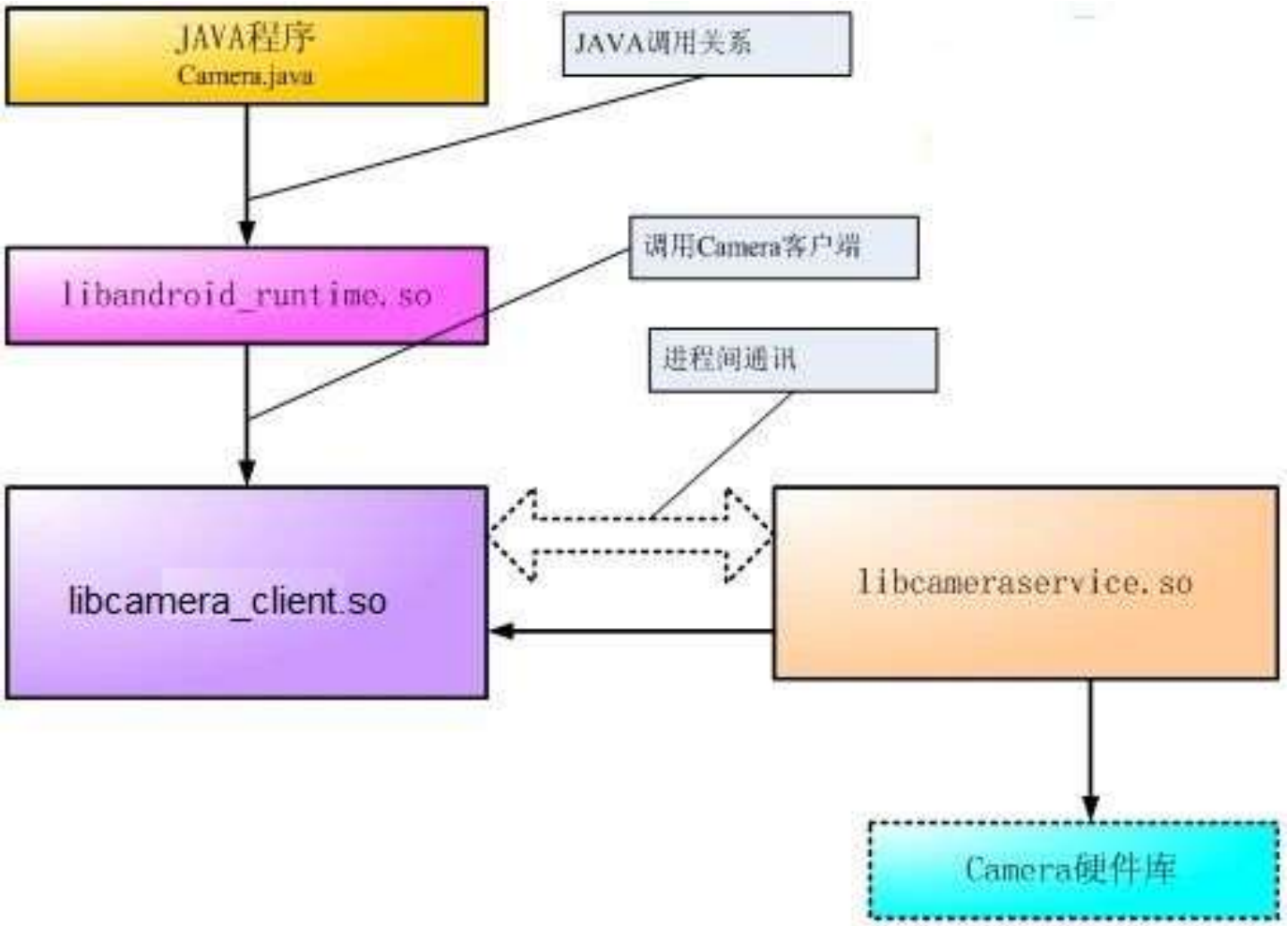
(libcamera_client.so)

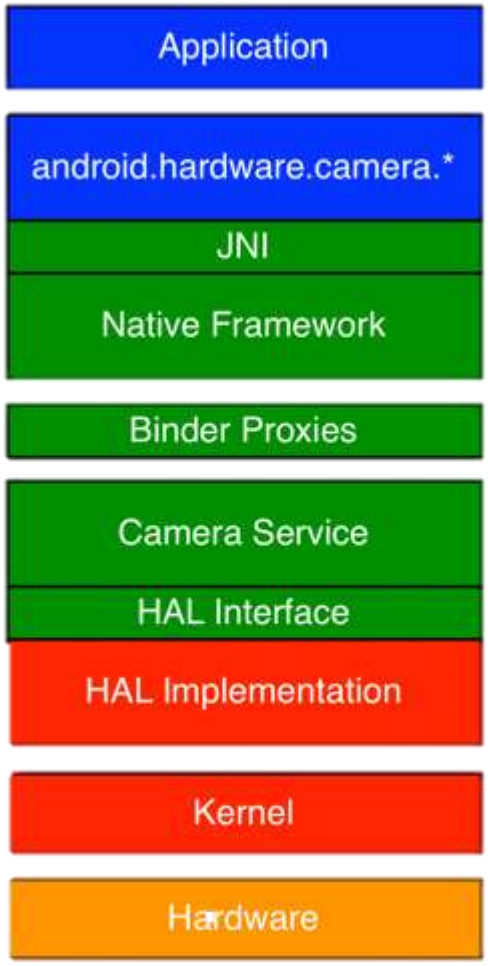
Camera Service:

frameworks\av\services\camera\libcameraservice\

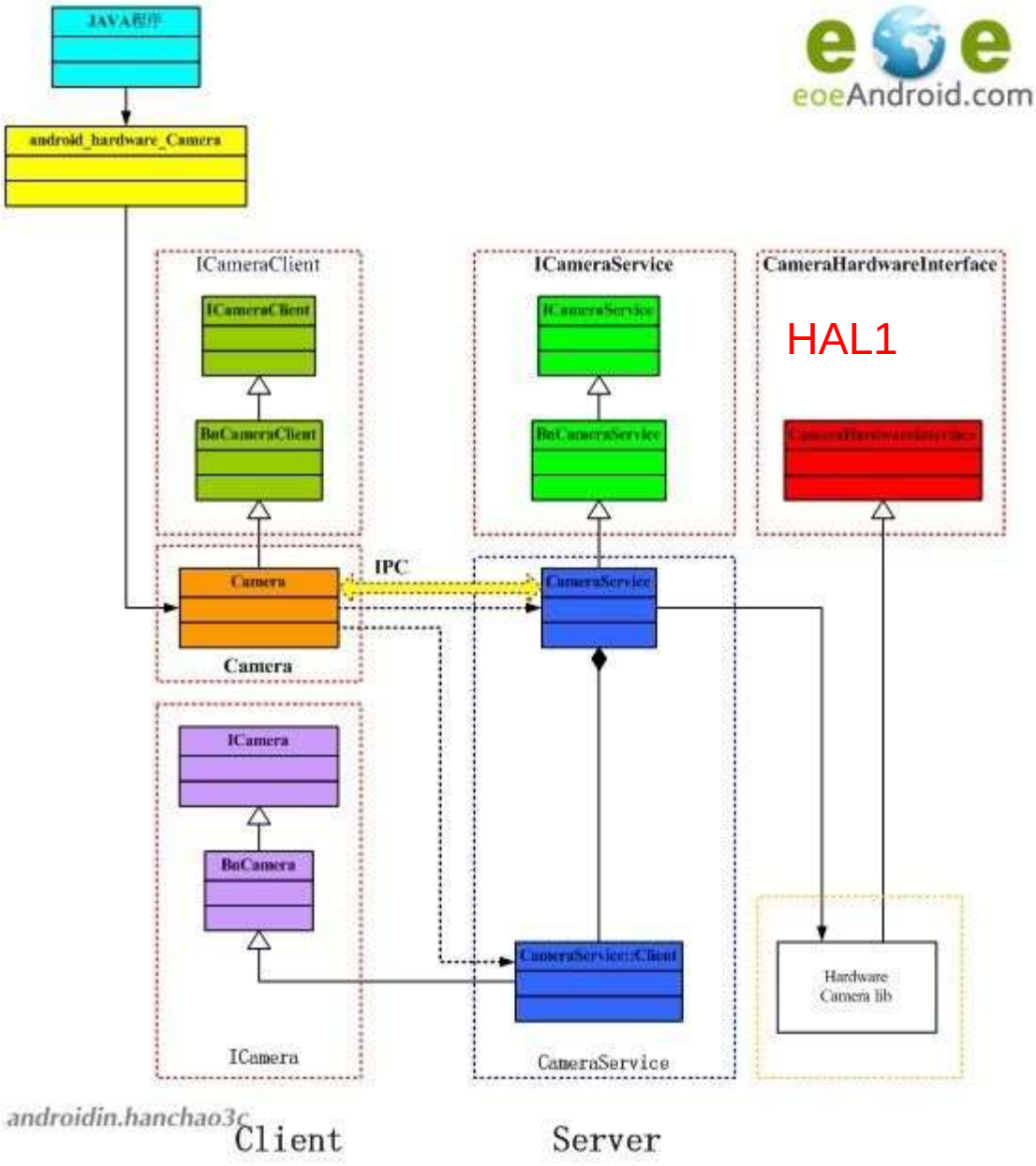
(libcameraservice.so)

Architecture





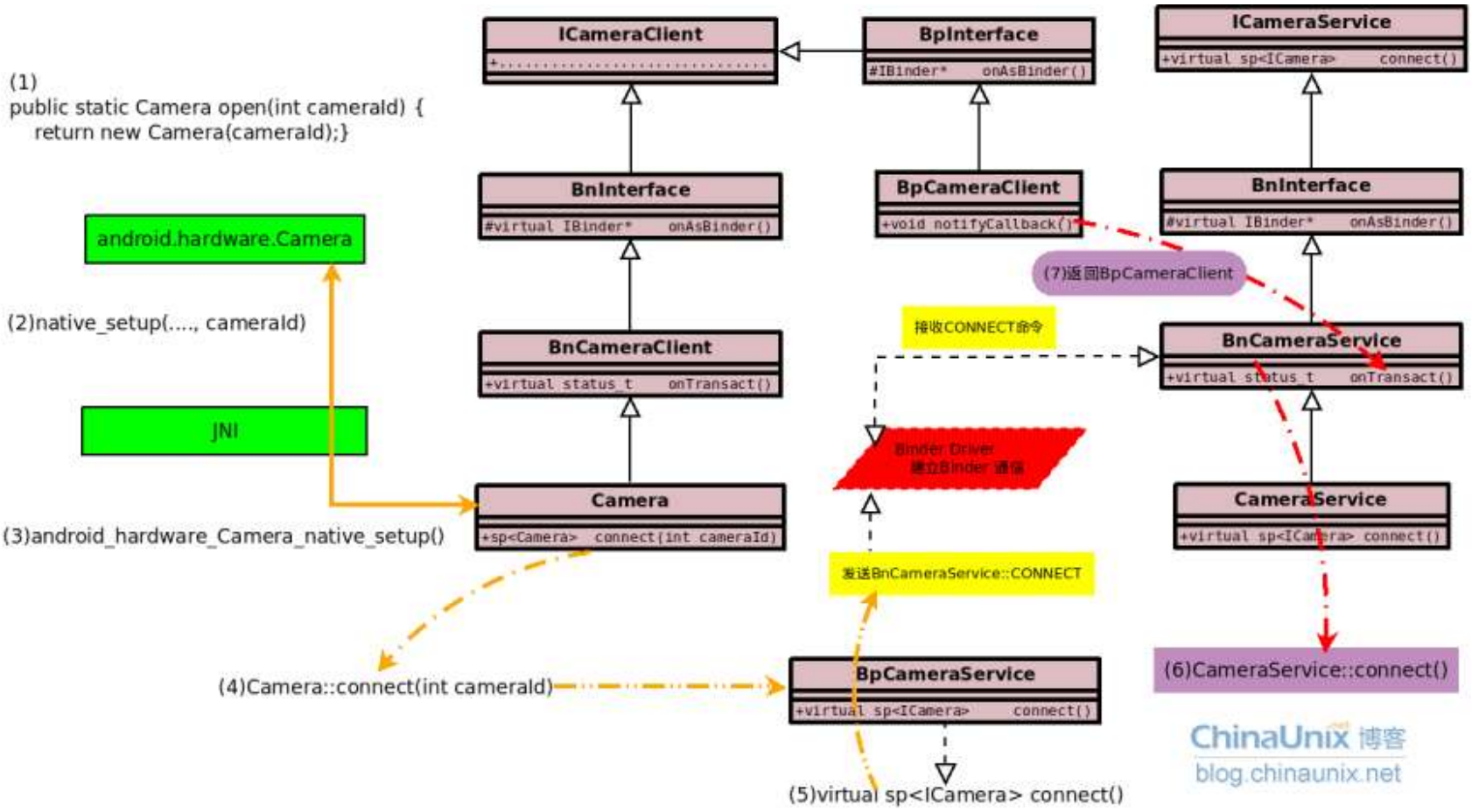
Application Framework/SDK	Java Classes Exposed to the application for interacting with the hardware	frameworks/base/core/java/android/hardware
JNI	Glue Code between Java Classes and Native Classes	frameworks/base/core/jni/android_hardware_Camera.cpp
Native Framework	Native counterpart of the Java Camera Classes. Manages all Binder Interactions	frameworks/av/camera/Camera.cpp
IPC Binder	3 Binder Interfaces ICameraService and Icamera from Application to Framework. ICameraClient for callbacks into the Application	frameworks/av/camera
Camera Service	The Camera Service that manages permissions and lifecycle of the Camera Devices	frameworks/av/services/camera/libcameraservice/CameraService.cpp
HAL Interface	Hardware Interface.	platform/hardware/libhardware/include
HAL Implementation	Hardware specific implementations. Depends on host processor, ISP and Sensor	platform/hardware/<vendor>/<platform>
Kernel drivers		


JAVA程序
android_hardware_Camera
ICameraClient
ICameraClient
BnCameraClient
ICameraService
ICameraService
BnCameraService
CameraHardwareInterface
HAL1
Camera
Camera
IPC
CameraService
ICamera
BnCamera
ICamera
CameraService::Client
CameraService
Hardware
Camera lib
androidin.hanchao3c
Client
Server

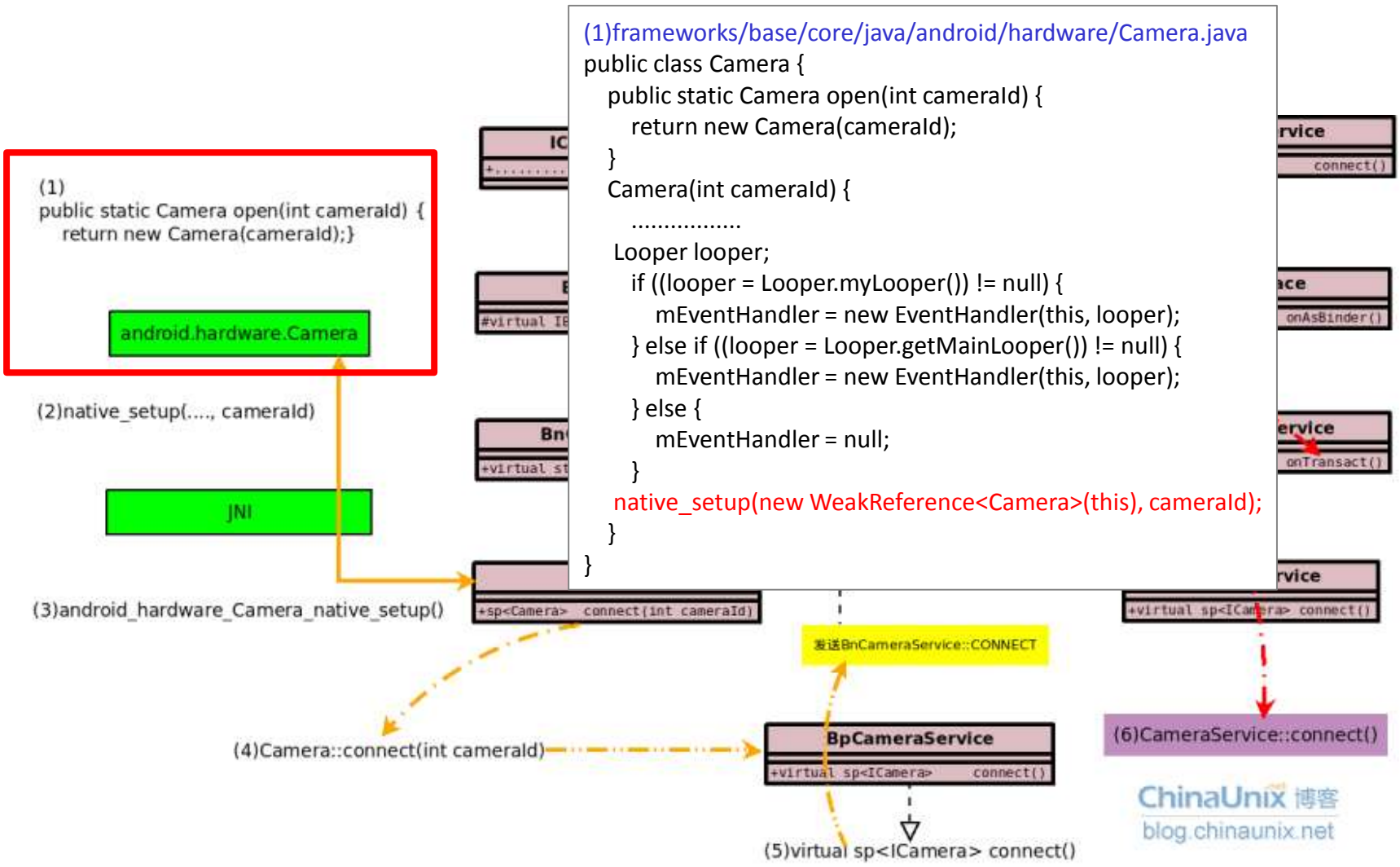
Camera keynote

- Camera uses binderized classes
  - ICamera -- proxy to camera hardware
  - ICameraClient -- receives callbacks
  - ICameraService -- creates and controls Icamera
- ICameraService, like ICamera and ICameraClient, are binder interfaces(proxy)
- Defined in frameworks/av/:
  - include/camera/ICamera*.h
  - camera/ICamera*.cpp
  - service/camera/libcameraservice/CameraService.cpp
- Camera class:
  - is a BnCameraClient
  - contains an Icamera

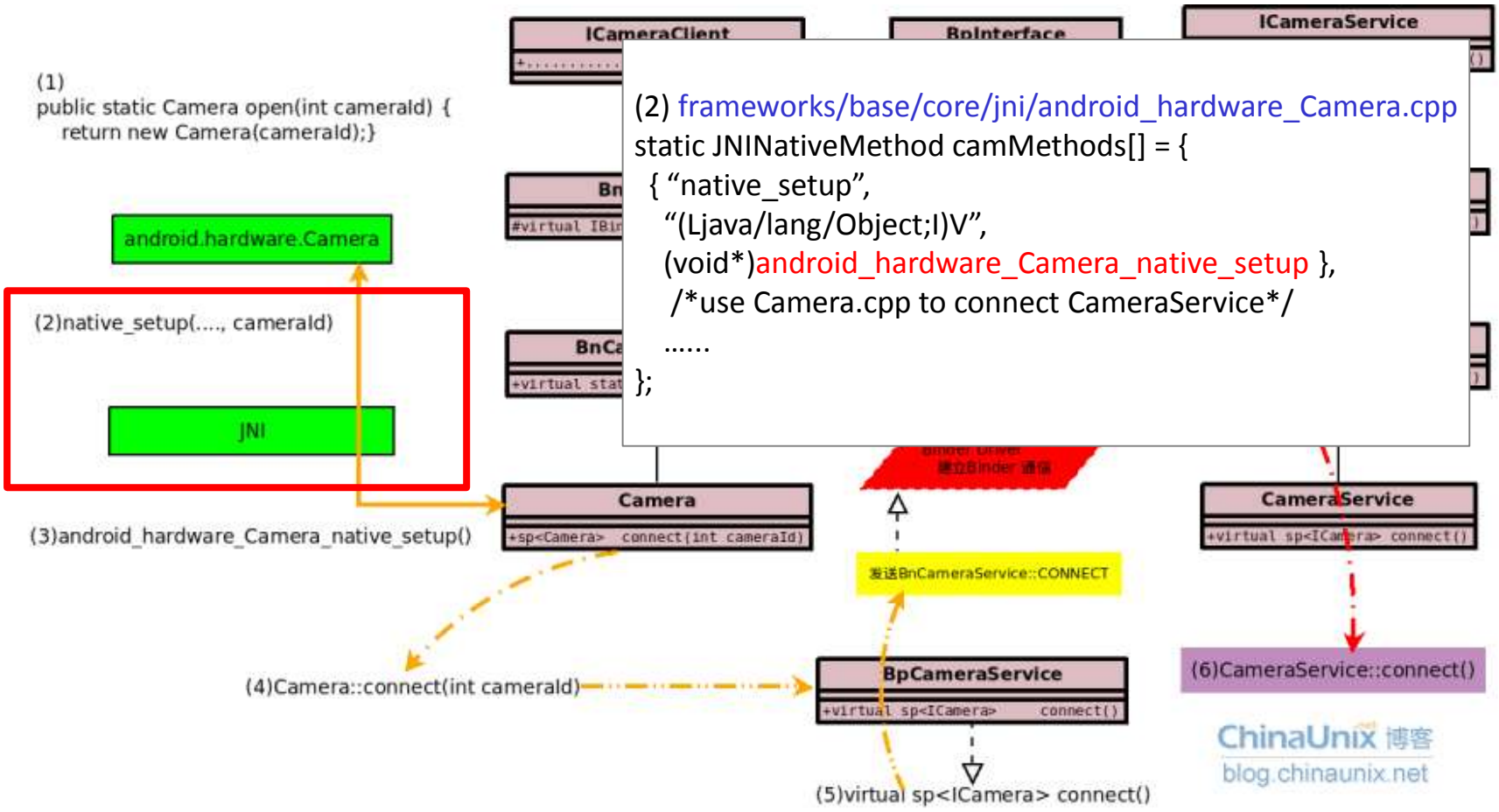
From top to bottom flow: Camera open case



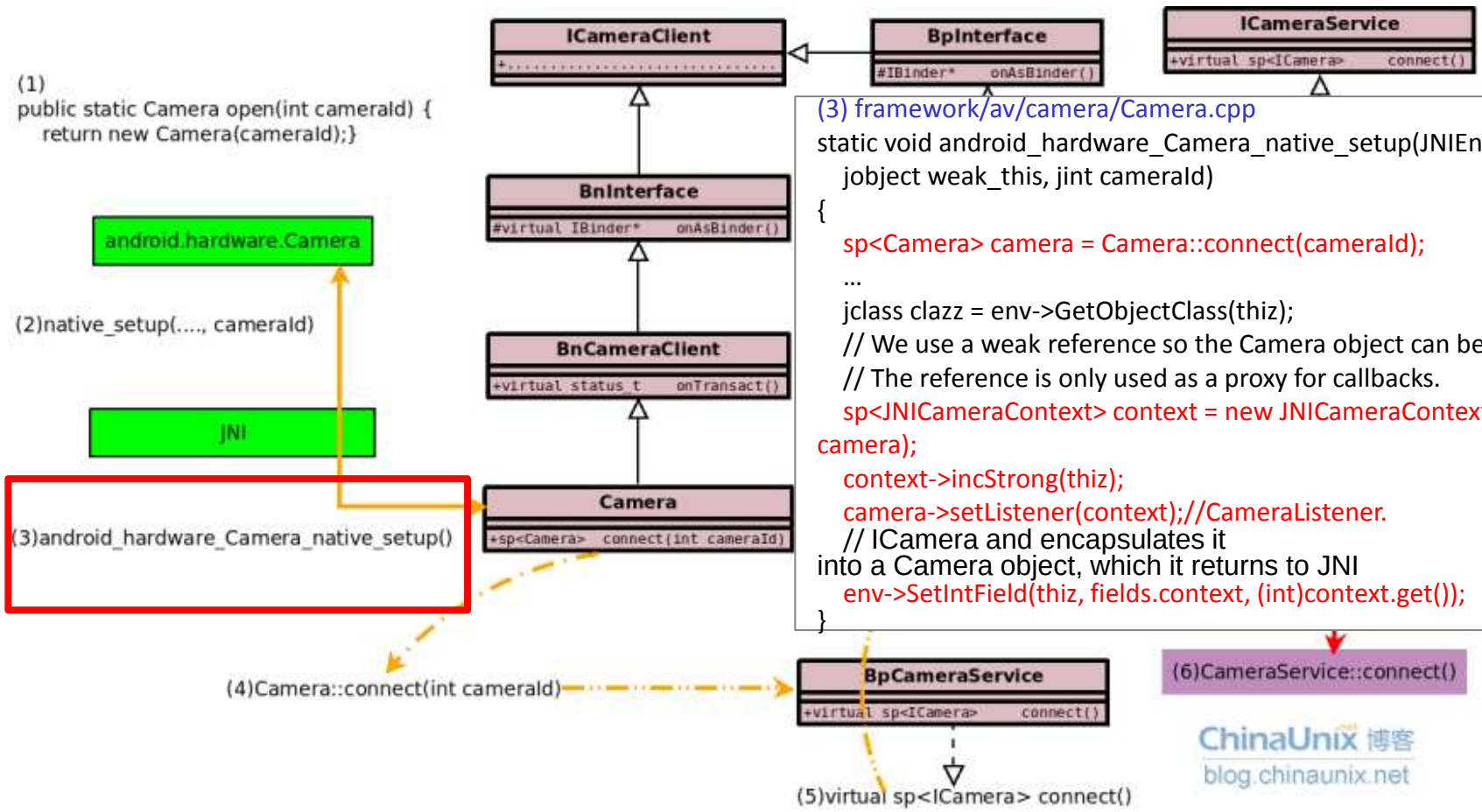
Step 1: open()



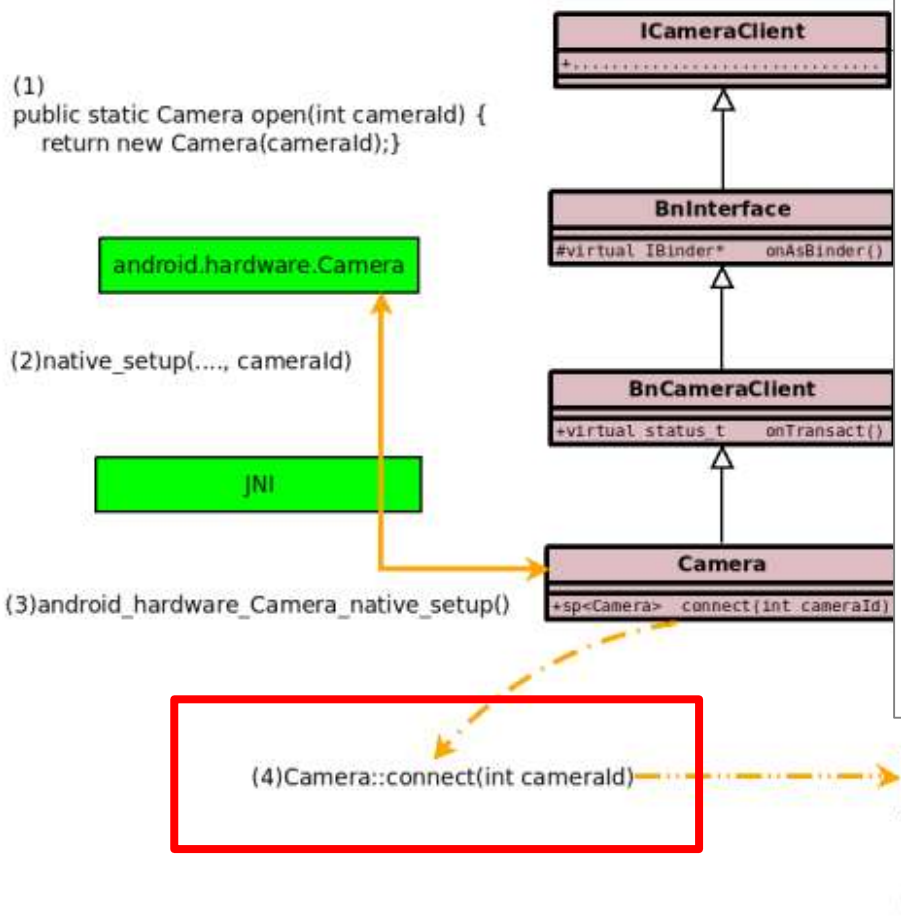
Step 2:Use JNI to call Native function



Step 3: call connect , set call back listener



Step 4: get Camera from BpCameraService



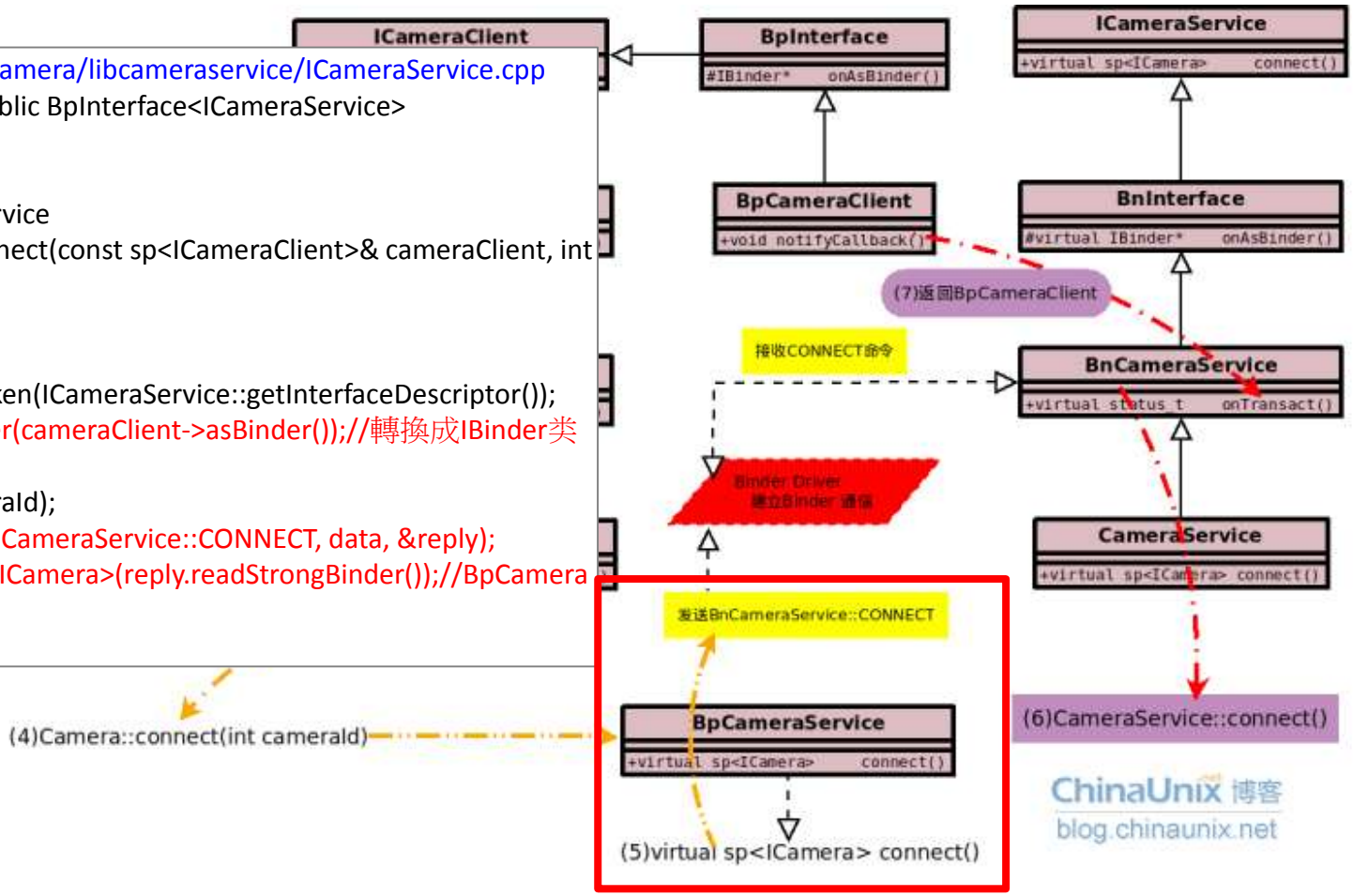
(4) framework/av/camera/CameraBase.cpp

```
sp<Camera> Camera::connect(int cameraId)
{
    ALOGV("connect");
    sp<Camera> c = new Camera();//BnCameraClient
    const sp<ICameraService>& cs =
getCameraService();//return BpCameraService
    if (cs != 0) {//Used for processing all kinds of events
        c->mCamera = cs->connect(c, cameraId);//return
BpCamera
    }
    if (c->mCamera != 0) {
        c->mCamera->asBinder()->linkToDeath(c);
        c->mStatus = NO_ERROR;
    } else {
        c.clear();
    }
    return c;
}
```

Step 5: Use IPC binder between BnCameraService and BpCameraService

(5)framework/av/service/camera/libcameraservice/ICameraService.cpp

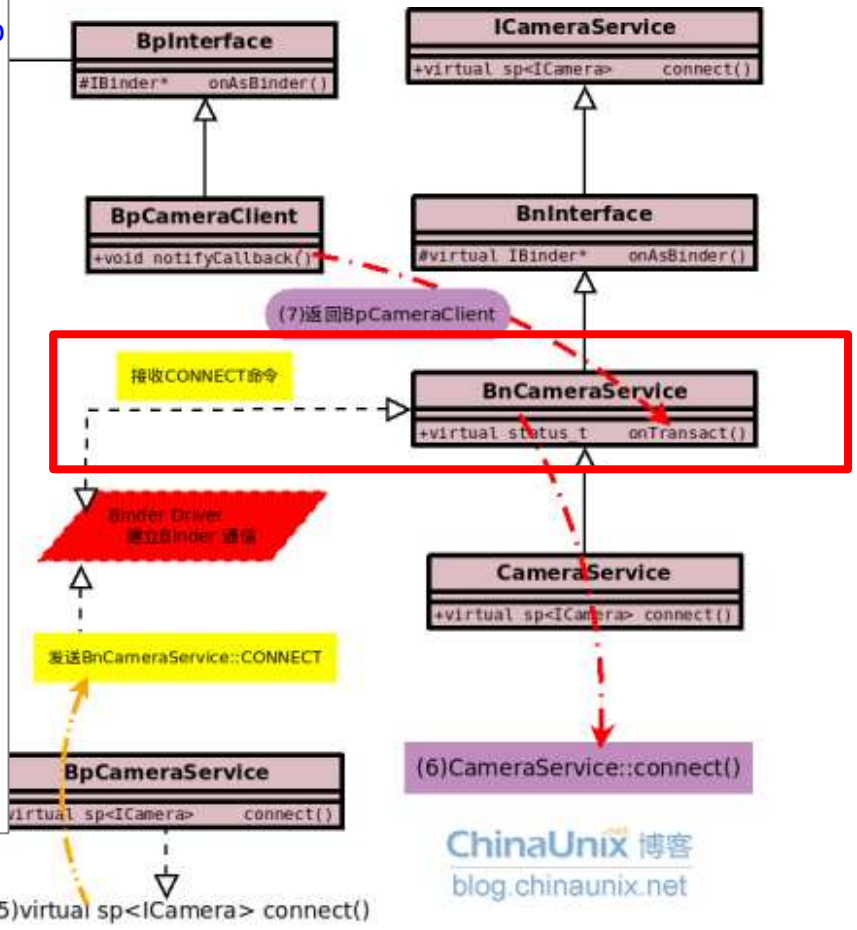
```
class BpCameraService: public BpInterface<ICameraService>
{
public:
    // connect to camera service
    virtual sp<ICamera> connect(const sp<ICameraClient>& cameraClient, int cameraId)
    {
        Parcel data, reply;
        data.writeInterfaceToken(ICameraService::getInterfaceDescriptor());
        data.writeStrongBinder(cameraClient->asBinder());//轉換成IBinder类型BpCameraClient
        data.writeInt32(cameraId);
        remote()->transact(BnCameraService::CONNECT, data, &reply);
        return interface_cast<ICamera>(reply.readStrongBinder());//BpCamera
    }
};
```



Step 5: Use IPC binder between BnCameraService and BpCameraService

(5)framework/av/service/camera/libcameraservice/ICameraService.cpp

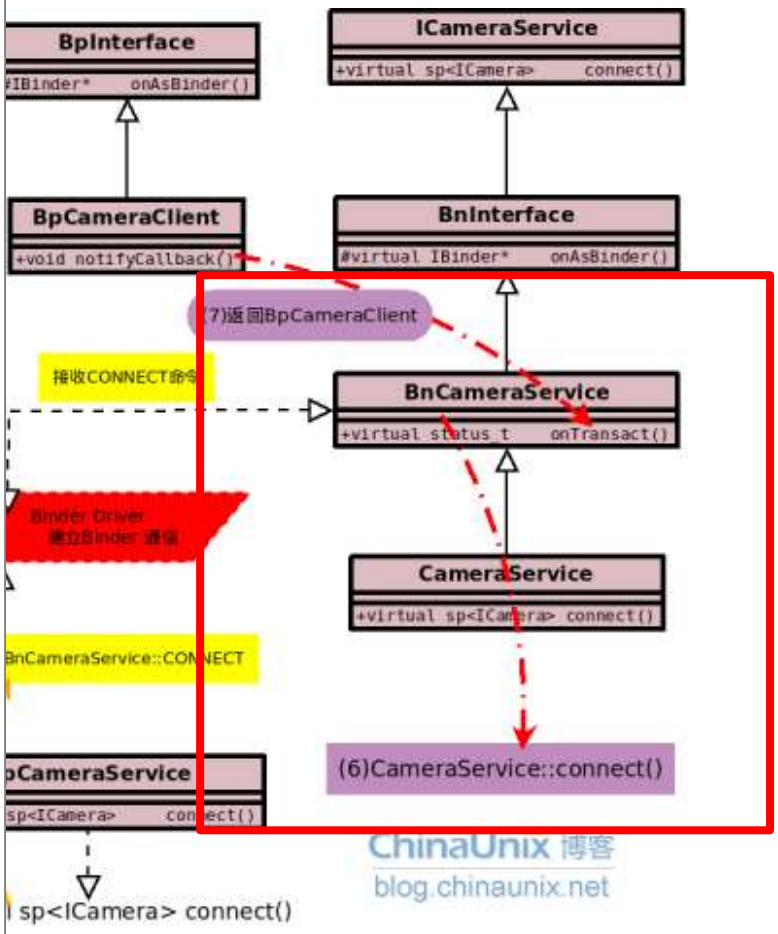
```
status_t BnCameraService::onTransact(
    uint32_t code, const Parcel& data, Parcel* reply, uint32_t flags)
{
    switch(code) {
        case CONNECT: {
            CHECK_INTERFACE(ICameraService, data, reply);//
            sp<ICameraClient> cameraClient =
                interface_cast<ICameraClient>(data.readStrongBinder());
            //get BpCameraClient
            sp<ICamera> camera = connect(cameraClient,
data.readInt32());//
             //return Client 繼承BnCamera
            reply->writeStrongBinder(camera->asBinder());
             //轉成BpCamera 寫回reply
            return NO_ERROR;
        } break;
    }
}
```



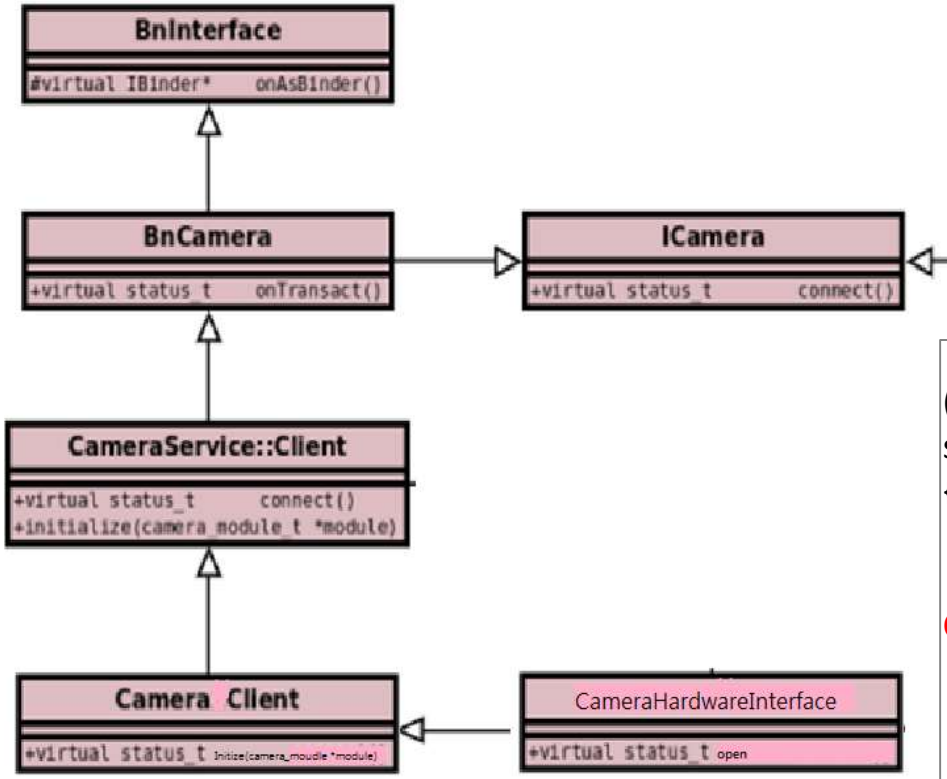
Step 6: instance CameraClient

(6)framerwork/av/service/camera/libcameraservice/CameraService.cpp

```
sp<ICamera> CameraService::connect(
        const sp<ICameraClient>& cameraClient, int cameraId) {
    .....
    int deviceVersion;
    if (mModule->common.module_api_version == CAMERA_MODULE_API_VERSION_2_0) {
        deviceVersion = info.device_version;
    } else {
        deviceVersion = CAMERA_DEVICE_API_VERSION_1_0;
    }
    /*根據HAL不同API創建不同的Client instance after 4.2 version*/
    switch(deviceVersion) {
      case CAMERA_DEVICE_API_VERSION_1_0:
            client = new CameraClient(this, cameraClient,
                    clientPackageName, cameraId,
                    facing, callingPid, clientUid, getpid());
            break;
     case CAMERA_DEVICE_API_VERSION_2_0:
     case CAMERA_DEVICE_API_VERSION_2_1:
     case CAMERA_DEVICE_API_VERSION_3_0:
            client = new Camera2Client(this, cameraClient,
                    clientPackageName, cameraId,
                    facing, callingPid, clientUid, getpid(),
                    deviceVersion);
            break;
    }
    /*初始化camera_module_t *module*/
    if (client->initialize(mModule) != OK) {
        return NULL;
    }
    .......
    return client;/*最后返回*/
}
```



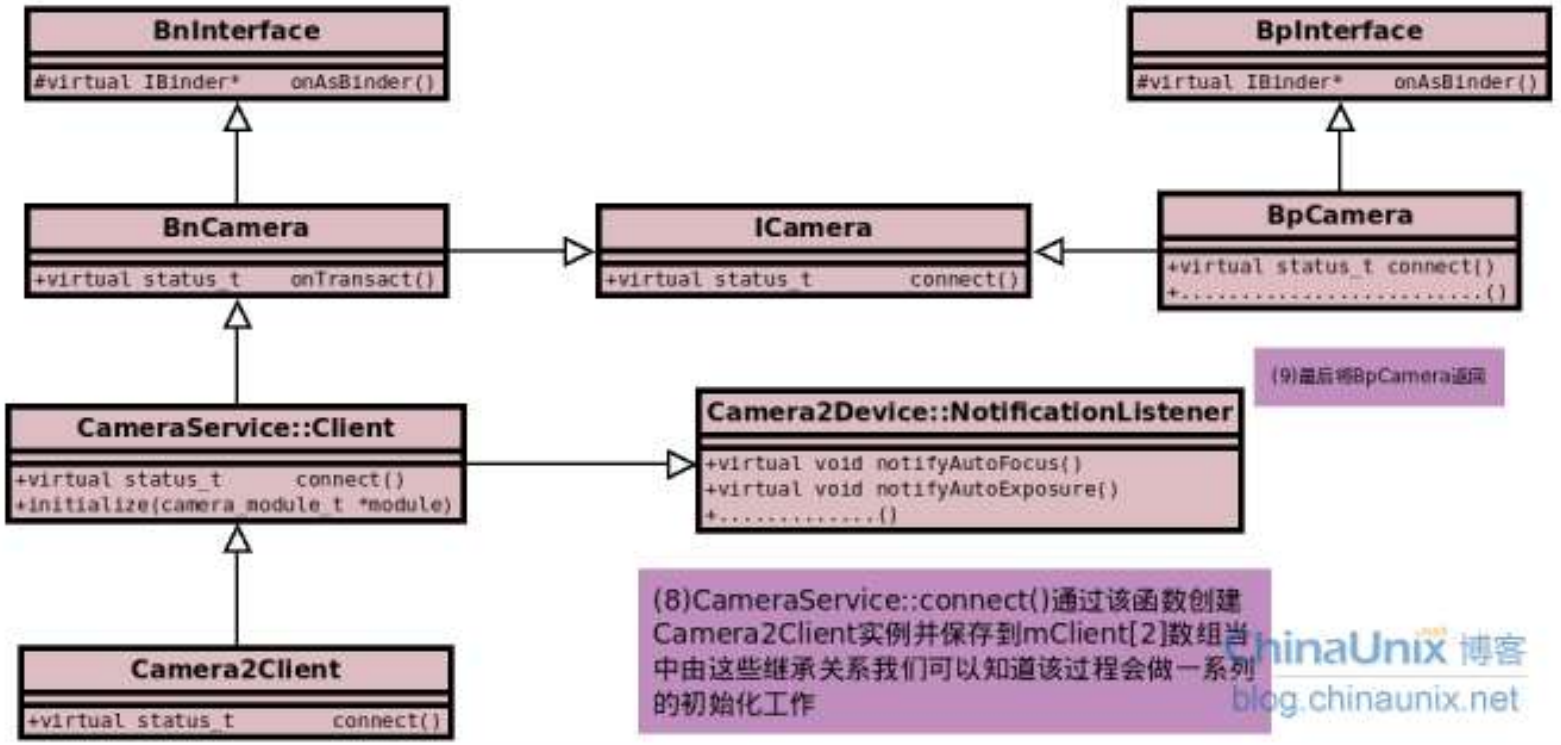
Initialize CameraClient



(8)HAL 1 Case..CameraClient

```
status_t CameraClient::initialize(camera_module_t *modul
{
   .......
   mHardware = new
CameraHardwareInterface(camera_device_name);
   res = mHardware->initialize(&module->common);
   .......
   mHardware->setCallbacks(notifyCallback,
         dataCallback,
         dataCallbackTimestamp,
         (void *)mCameraId);
    .....
}
```

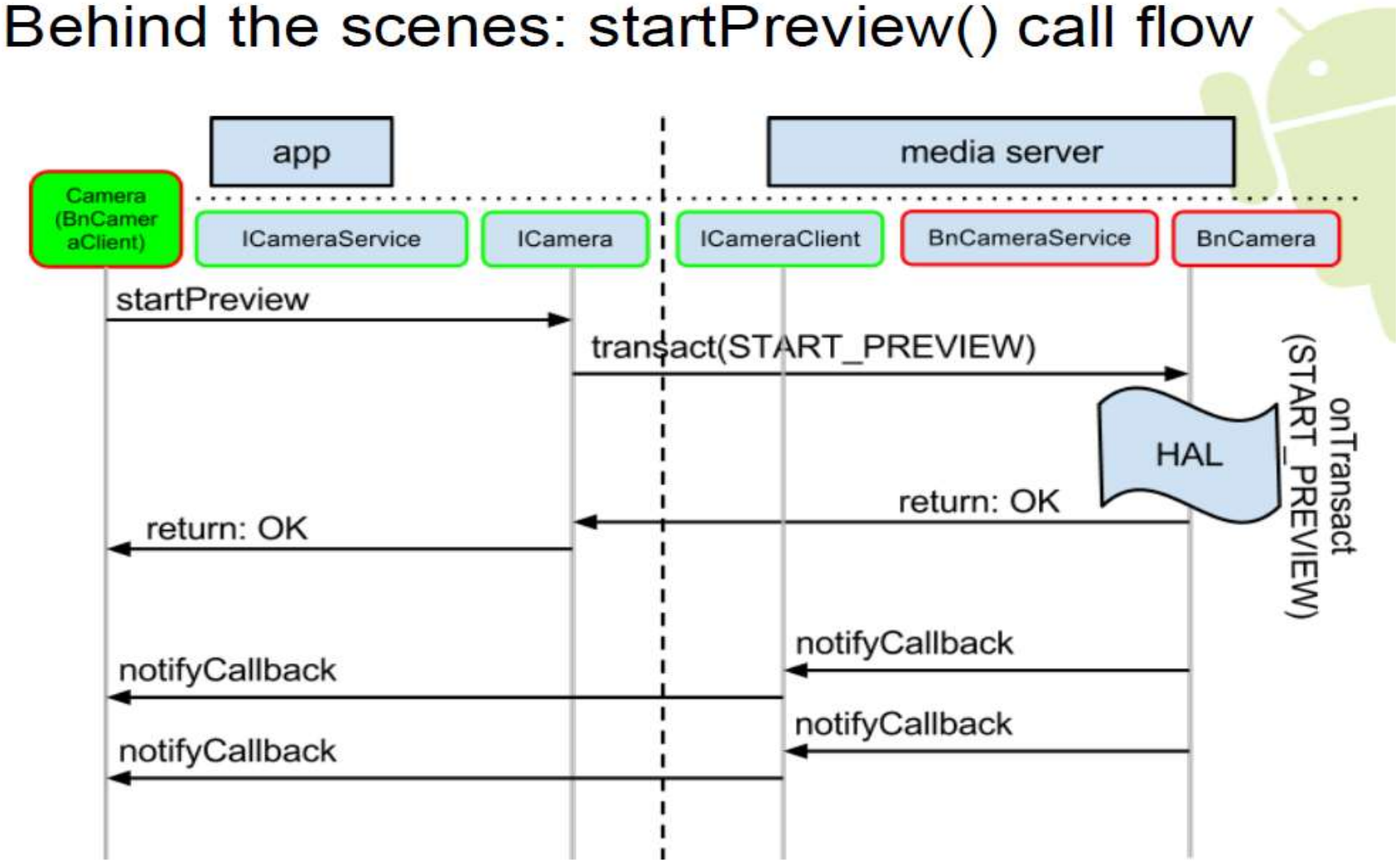
Initialize Camera2Client



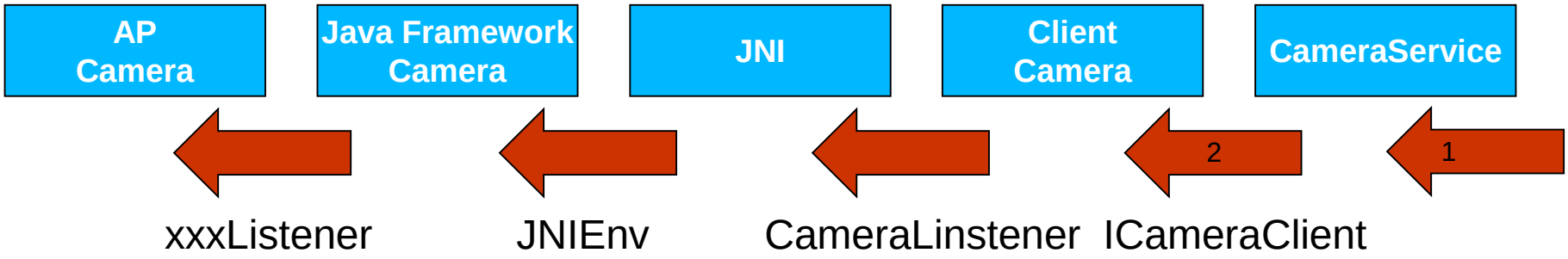

```
(8)HAL 2or3 Case..Camera2Client
status_t Camera2Client::initialize(camera_module_t *module) {
   mDevice = CameraDeviceFactory::createDevice(cameraId); //Camera2Device
   ……
   mDevice->initialize(module);
   …..
   mDevice->setNotifyCallback(this);

   return OK;
}
```

You can call medial server method by IPC



Callback Interfaces



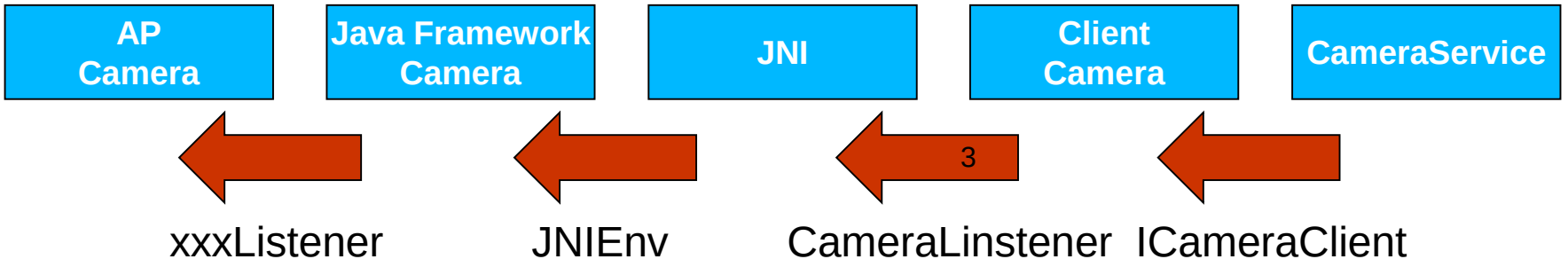
1. call back from hardward , flow start from camera client ex:jpeg call back

```
void CameraClient::dataCallback(int32_t msgType,
        const sp<IMemory>& dataPtr, camera_frame_metadata_t *metadata, void* user) {
CameraClient* client =
            static_cast<CameraClient*>(getClientFromCookie(user));
    switch (msgType & ~CAMERA_MSG_PREVIEW_METADATA) {
        …….
        case CAMERA_MSG_COMPRESSED_IMAGE:
            client->handleCompressedPicture(dataPtr);
            Break;
        ……..
    }
}
```

2. ***use ipc binder "bpcameraclient , bncameraclient in IcameraClient to send data , and implement in Camera.cpp***

```
void CameraClient::handleCompressedPicture(const sp<IMemory>& mem) {
    sp<ICameraClient> c = mRemoteCallback;
    mLock.unlock();
    if (c != 0) {
        c->dataCallback(CAMERA_MSG_COMPRESSED_IMAGE, mem, NULL);
    }
```

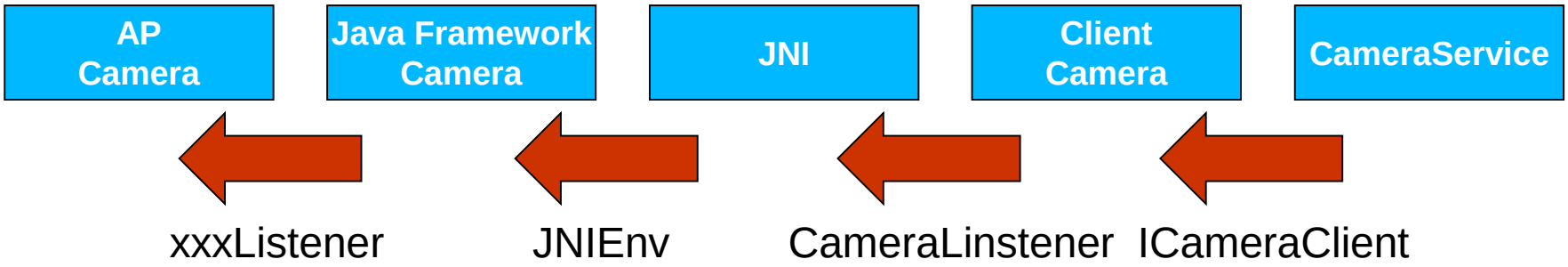
Callback Interfaces



3. use ipc binder "bpcameraclient , bncameraclient in IcameraClient to send data , and implement in Camera.cpp

```
void Camera::dataCallback(int32_t msgType, const sp<IMemory>& dataPtr,
                          camera_frame_metadata_t *metadata)
{
    sp<CameraListener> listener;
    {
        Mutex::Autolock _l(mLock);
        listener = mListener;
    }
    if (listener != NULL) {
        listener->postData(msgType, dataPtr, metadata);
    }
}
```

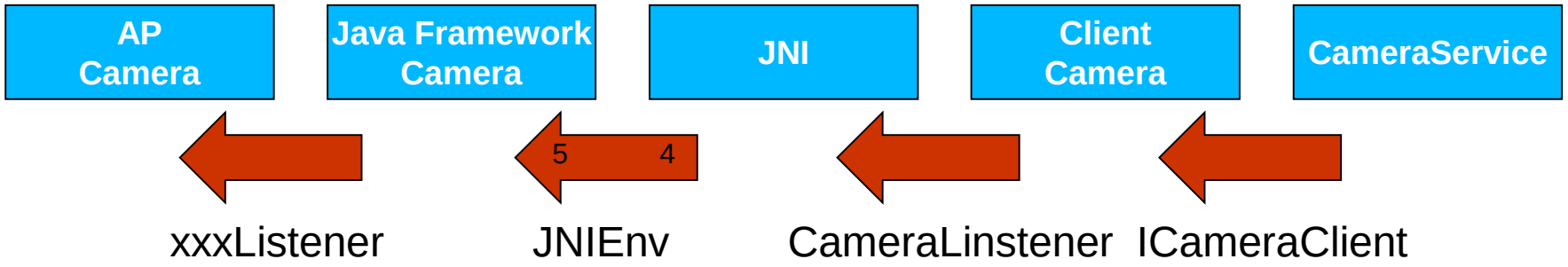
Callback Interfaces




```
class CameraListener: virtual public RefBase
{
public:
    virtual void notify(int32_t msgType, int32_t ext1, int32_t ext2) = 0;
    virtual void postData(int32_t msgType, const sp<IMemory>& dataPtr,
                          camera_frame_metadata_t *metadata) = 0;
    virtual void postDataTimestamp(nsecs_t timestamp, int32_t msgType,
                                   const sp<IMemory>& dataPtr) = 0;
};
```

Note:
msgType enum values and data structures, ex: “camera_frame_metadata_t”
re defined in “system\core\include\system\camera.h”.

Callback Interfaces



4.Implement cameraListener post data in android_haradware_camera.cpp

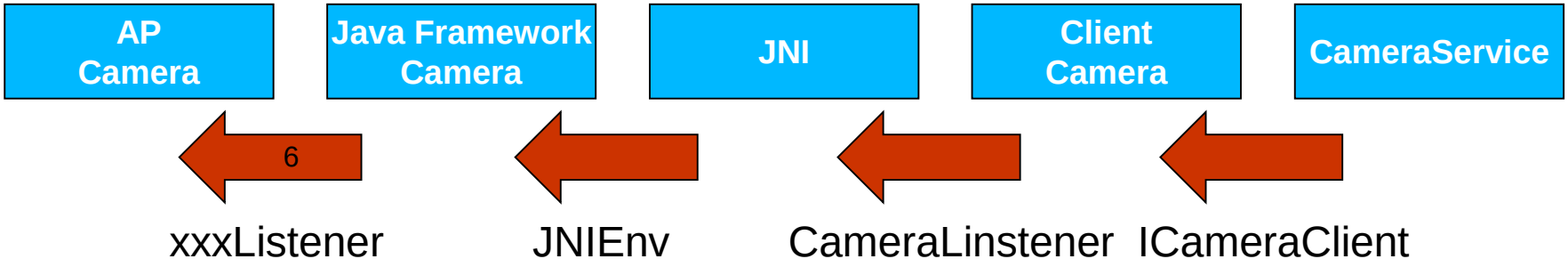
```
void JNICameraContext::postData(int32_t msgType, const sp<IMemory>& dataPtr, camera_frame_metadata_t *metadata)
{
    ……..
    JNIEnv *env = AndroidRuntime::getJNIEnv();
     .....
    int32_t dataMsgType = msgType
       & ~CAMERA_MSG_PREVIEW_METADATA;
    switch (dataMsgType) {
          …….
        case CAMERA_MSG_RAW_IMAGE:
            ALOGV("rawCallback");
            if (mRawImageCallbackBuffers.isEmpty()) {
                env->CallStaticVoidMethod(mCameraJClass,
                fields.post_event,
                        mCameraJObjectWeak, dataMsgType, 0, 0, NULL);
            } else {
                copyAndPost(env, dataPtr, dataMsgType);
            }
            Break;
    }
```

5.post image data to Java by JNI

```
void JNICameraContext::copyAndPost(JNIEnv* env, const
sp<IMemory>& dataPtr, int msgType)
{
      obj = env->NewByteArray(size);
       env->CallStaticVoidMethod(mCameraJClass,
fields.post_event,
            mCameraJObjectWeak, msgType, 0, 0, obj);

}
//jclass clazz = env->FindClass("android/hardware/Camera");
    fields.post_event = env->GetStaticMethodID(clazz,
"postEventFromNative",
"(Ljava/lang/Object;IIILjava/lang/Object;)V");
```

Callback Interfaces



5. Call back to ap level :Framework/base/core/java/android/hardware/camera.java

```
private class EventHandler extends Handler
   {
@Override
       public void handleMessage(Message msg) {
          switch(msg.what) {
         case CAMERA_MSG_COMPRESSED_IMAGE:
                  if (mJpegCallback != null) {
                           mJpegCallback.onPictureTaken((byte[])msg.obj, mCamera);
                  }
              Return;
   }
```