# PreviewWindow初始化

当app打开相机的时候，会设置预览窗口到hal中，hal才能往窗口送对应的数据，那么整个过程是什么呢。

在app层是surface

在Hal层是preview\_stream\_ops\_t，关键变量是mANativeWindow。

首先通过app->camera\_server，调用camera.so,触发

## setPreviewWindow起点

这个是hal的框架发起的。

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

status\_t setPreviewWindow(**const** sp<ANativeWindow>& buf)  
{  
 ALOGV(**"%s(%s) buf %p"**, \_\_FUNCTION\_\_, mName.string(), buf.get());  
 mPreviewWindow = buf;  
 mHalPreviewWindow.user = **this**;  
 **return** mDevice->ops->set\_preview\_window(mDevice,  
 buf.get() ? &mHalPreviewWindow.nw : 0);  
}

mHalPreviewWindow.nw定义

**struct** camera\_preview\_window {  
 **struct** preview\_stream\_ops nw;  
 **void** \*user;  
};

mHalPreviewWindow

同时定义了函数指针

**void** initHalPreviewWindow()  
{  
 mHalPreviewWindow.nw.cancel\_buffer = \_\_cancel\_buffer;  
 mHalPreviewWindow.nw.lock\_buffer = \_\_lock\_buffer;  
 mHalPreviewWindow.nw.dequeue\_buffer = \_\_dequeue\_buffer;  
 mHalPreviewWindow.nw.enqueue\_buffer = **\_\_enqueue\_buffer;**  
 mHalPreviewWindow.nw.set\_buffer\_count = \_\_set\_buffer\_count;  
 mHalPreviewWindow.nw.set\_buffers\_geometry = \_\_set\_buffers\_geometry;  
 mHalPreviewWindow.nw.set\_crop = \_\_set\_crop;  
 mHalPreviewWindow.nw.set\_timestamp = \_\_set\_timestamp;  
 mHalPreviewWindow.nw.set\_usage = \_\_set\_usage;  
 mHalPreviewWindow.nw.set\_swap\_interval = \_\_set\_swap\_interval;  
  
 mHalPreviewWindow.nw.get\_min\_undequeued\_buffer\_count =  
 \_\_get\_min\_undequeued\_buffer\_count;  
}

然后在接口的头文件camera.h定义：hardware/libhardware/include/hardware/camera.h

typedef struct preview\_stream\_ops {

int (\***dequeue\_buffer**)(struct preview\_stream\_ops\* w,

buffer\_handle\_t\*\* buffer, int \*stride);

int (\***enqueue\_buffer**)(struct preview\_stream\_ops\* w,

buffer\_handle\_t\* buffer);

int (\*cancel\_buffer)(struct preview\_stream\_ops\* w,

buffer\_handle\_t\* buffer);

int (\*set\_buffer\_count)(struct preview\_stream\_ops\* w, int count);

int (\*set\_buffers\_geometry)(struct preview\_stream\_ops\* pw,

int w, int h, int format);

int (\*set\_crop)(struct preview\_stream\_ops \*w,

int left, int top, int right, int bottom);

int (\*set\_usage)(struct preview\_stream\_ops\* w, int usage);

int (\*set\_swap\_interval)(struct preview\_stream\_ops \*w, int interval);

int (\*get\_min\_undequeued\_buffer\_count)(const struct preview\_stream\_ops \*w,

int \*count);

int (\*lock\_buffer)(struct preview\_stream\_ops\* w,

buffer\_handle\_t\* buffer);

int (\*set\_timestamp)(struct preview\_stream\_ops \*w, int64\_t timestamp);

} preview\_stream\_ops\_t;

其中 enqueue\_buffer和dequeue\_buffer是最重要的方法

另外：mPreviewWindow和mHalPreviewWindow的区别和联系：mHalPreviewWindow的user字段就是mPreviewWindow！

**static** ANativeWindow \*\_\_to\_anw(**void** \*user)  
 {  
 CameraHardwareInterface \*\_\_this =  
 **reinterpret\_cast**<CameraHardwareInterface \*>(user);  
 **return** \_\_this->mPreviewWindow.get();  
 }  
#define anw(n) \_\_to\_anw(((**struct** camera\_preview\_window \*)n)->user)

## CameraHal::setPreviewWindow

camera\_device\_open指定set\_preview\_window的实际是调用地址为camera\_set\_preview\_window，从而调用了实现层的

由函数指针关系，我们知道最终是调用的camera\_set\_preview\_window方法。

CameraHal\_Module.cpp

**int** camera\_device\_open(**const** hw\_module\_t\* module, **const char**\* name,  
 hw\_device\_t\*\* device)  
{

camera\_ops->set\_preview\_window = camera\_set\_preview\_window;

}

**int** camera\_set\_preview\_window(**struct** camera\_device \* device,  
 **struct** preview\_stream\_ops \*window)  
{  
 rv = gCameraHals[rk\_dev->cameraid]->setPreviewWindow(window);  
 **return** rv;  
}

gCameraHals

打开camera过程中，会去调用setPreviewWindow

**int** CameraHal::setPreviewWindow(**struct** preview\_stream\_ops \*window)  
{  
 msg.command = CMD\_SET\_PREVIEW\_WINDOW;  
 sem.Create();  
 msg.arg1 = (**void**\*)(&sem);  
 msg.arg2 = (**void**\*)window;  
 setCamStatus(CMD\_SET\_PREVIEW\_WINDOW\_PREPARE, 1);  
 commandThreadCommandQ.put(&msg);  
}

这里是采用消息机制，为了避免频繁打开？

**void** CameraHal::commandThread()

**case** CMD\_SET\_PREVIEW\_WINDOW:  
 {  
 LOGD(**"%s(%d):receive CMD\_SET\_PREVIEW\_WINDOW"**,\_\_FUNCTION\_\_,\_\_LINE\_\_);  
 mParameters.getPreviewSize(&app\_previw\_w,&app\_preview\_h);  
 mDisplayAdapter->setPreviewWindow((**struct** preview\_stream\_ops \*)msg.arg2);  
 prevStatus = mCameraAdapter->getCurPreviewState(&drv\_w,&drv\_h);  
 **if** ((mDisplayAdapter->getPreviewWindow()) && prevStatus) {  
 err=mDisplayAdapter->startDisplay(app\_previw\_w, app\_preview\_h);

}

## DisplayAdapter.setPreviewWindow

最后终于调用了DisplayAdapter的适配，完成了window的赋值

**int** DisplayAdapter::setPreviewWindow(**struct** preview\_stream\_ops\* window)  
{  
 **if**(mANativeWindow){  
 pauseDisplay();  
 }  
 mANativeWindow = window;  
 **return** 0;  
}

## 小结

设置完之后， mANativeWindow等价于Surface==

# enqueue\_buffer时序

从hal层请求enqueue\_buffer！

err = mANativeWindow->enqueue\_buffer(mANativeWindow,

## enqueue\_buffer

mANativeWindow的类型是preview\_stream\_ops\_t，定义在hardware/libhardware/include/hardware/camera.h

**static int** \_\_enqueue\_buffer(**struct** preview\_stream\_ops\* w,  
 buffer\_handle\_t\* buffer)  
{  
 ANativeWindow \*a = anw(w);  
 **return** a->queueBuffer(a,  
 container\_of(buffer, ANativeWindowBuffer, handle), -1);  
}

## libgui

frameworks/native/libs/gui/include/gui/Surface.h

**class** Surface  
 : **public** ANativeObjectBase<ANativeWindow, Surface, RefBase>

实现，再次通过函数指针，完成具体的实现

Surface::Surface(**const** sp<IGraphicBufferProducer>& bufferProducer, **bool** controlledByApp)

mGraphicBufferProducer(bufferProducer)

ANativeWindow::queueBuffer = hook\_queueBuffer;

从而调用

## nativewindow

其中**struct** ANativeWindow的定义

frameworks/native/libs/nativewindow/include/system/window.h

**int** (\*queueBuffer)(**struct** ANativeWindow\* window,  
 **struct** ANativeWindowBuffer\* buffer, **int** fenceFd);

实现

frameworks/native/libs/nativewindow/ANativeWindow.cpp

**int** ANativeWindow\_queueBuffer(ANativeWindow\* window, ANativeWindowBuffer\* buffer, **int** fenceFd) {  
 **return** window->queueBuffer(window, buffer, fenceFd);  
}

\frameworks\av\services\camera\libcameraservice\device1\CameraHardwareInterface.h

CameraHardwareInterface中定义了ANativeWindow这个native层的窗口

### hook\_queueBuffer

**int** Surface::hook\_queueBuffer(ANativeWindow\* window,  
 ANativeWindowBuffer\* buffer, **int** fenceFd) {  
 Surface\* c = getSelf(window);  
 **return** c->queueBuffer(buffer, fenceFd);  
}

然后最终调用

### Surface::queueBuffer

**int** Surface::queueBuffer(android\_native\_buffer\_t\* buffer, **int** fenceFd) {

**int** i = getSlotFromBufferLocked(buffer);  
**if** (i < 0) {  
 **if** (fenceFd >= 0) {  
 close(fenceFd);  
 }  
 **return** i;  
}  
**if** (mSharedBufferSlot == i && mSharedBufferHasBeenQueued) {  
 **if** (fenceFd >= 0) {  
 close(fenceFd);  
 }  
 **return** OK;  
}

sp<Fence> fence(fenceFd >= 0 ? **new** Fence(fenceFd) : Fence::NO\_FENCE);  
IGraphicBufferProducer::QueueBufferOutput output;  
IGraphicBufferProducer::QueueBufferInput input(timestamp, isAutoTimestamp,  
 **static\_cast**<android\_dataspace>(mDataSpace), crop, mScalingMode,  
 mTransform ^ mStickyTransform, fence, mStickyTransform,  
 mEnableFrameTimestamps);

status\_t err = mGraphicBufferProducer->queueBuffer(i, input, &output);

mQueueBufferCondition.broadcast();

### BufferQueueProducer::queueBuffer

status\_t BufferQueueProducer::queueBuffer(**int** slot,  
 **const** QueueBufferInput &input, QueueBufferOutput \*output) {  
 ATRACE\_CALL();  
 ATRACE\_BUFFER\_INDEX(slot);

# displayThreadCommandQ.get(&msg);

取出一帧，填充数据信息

FramInfo\_s\* frame = (FramInfo\_s\*)msg.arg2;  
frame\_used\_flag = (**long**)msg.arg3;  
  
queue\_buf\_index = (**long**)msg.arg1;   
queue\_display\_index = CONFIG\_CAMERA\_DISPLAY\_BUF\_CNT;

# 找出queue\_display\_index

queue\_display\_index = CONFIG\_CAMERA\_DISPLAY\_BUF\_CNT;  
*//get a free buffer* **for** (i=0; i<CONFIG\_CAMERA\_DISPLAY\_BUF\_CNT; i++) {  
 **if** (mDisplayBufInfo && mDisplayBufInfo[i].buf\_state == 0)  
 **break**;  
 }  
 **if** (i<CONFIG\_CAMERA\_DISPLAY\_BUF\_CNT) {  
 queue\_display\_index = i;  
 }

# 格式转化

# setBufferState

标记为可用状态

setBufferState(queue\_display\_index, 1);  
mapper.unlock((buffer\_handle\_t)mDisplayBufInfo[queue\_display\_index].priv\_hnd);

# 送入窗口

err = mANativeWindow->enqueue\_buffer(mANativeWindow, (buffer\_handle\_t\*)mDisplayBufInfo[queue\_display\_index].buffer\_hnd);

# returnFrame

为啥要还？、

**if**(mFrameProvider)  
 mFrameProvider->returnFrame(frame->frame\_index,frame\_used\_flag);

# 取出一个显示buffer

queue\_cnt = 0;  
 **for** (i=0; i<mDislayBufNum; i++) {  
 **if** (mDisplayBufInfo[i].buf\_state == 1)   
 queue\_cnt++;// 统计目前用了多少个buffer  
 }  
  
 **if** (queue\_cnt > mDispBufUndqueueMin) {//高于了最小空闲  
err = mANativeWindow->dequeue\_buffer(mANativeWindow, (buffer\_handle\_t\*\*)&hnd, &stride);  
**if** (err == 0) {   
 *// lock the initial queueable buffers* bounds.left = 0;  
 bounds.top = 0;  
 bounds.right = mDisplayWidth;  
 bounds.bottom = mDisplayHeight;  
 mANativeWindow->lock\_buffer(mANativeWindow, (buffer\_handle\_t\*)hnd);  
 mapper.lock((buffer\_handle\_t)(\*hnd), CAMHAL\_GRALLOC\_USAGE, bounds, y\_uv);  
  
 phnd = (NATIVE\_HANDLE\_TYPE\*)\*hnd;  
 **for** (i=0; i<mDislayBufNum; i++) {  
 **if** (phnd == mDisplayBufInfo[i].priv\_hnd) {  
 dequeue\_buf\_index = i;  
 **break**;  
 }  
 }  
   
 **if** (i >= mDislayBufNum) {   
 LOGE(**"%s(%d): dequeue buffer(0x%x ) don't find in mDisplayBufferMap"**, \_\_FUNCTION\_\_,\_\_LINE\_\_,(**long**)phnd);   
 **continue**;  
 } **else** {  
 setBufferState(dequeue\_buf\_index, 0);//标记为空闲buffer  
 }  
   
 }