Main\_surfaceFlinger

1. 向SFEventThread请求vsync:

1.1 surfaceflinger主线程和sfEventThread线程的关联

sp<VSyncSource> sfVsyncSrc = new DispSyncSource(&mPrimaryDispSync,

sfVsyncPhaseOffsetNs, true, "sf");

mSFEventThread = new EventThread(sfVsyncSrc);

mEventQueue.setEventThread(mSFEventThread);

使用sfEventThread创建Connection，将connection channel中的mReceiveFd加入到主线程中的wait\_poll池中，等待从sfEventThread线程中发送过来的vsync信号。

void MessageQueue::setEventThread(const sp<EventThread>& eventThread)

{

mEventThread = eventThread;

mEvents = eventThread->createEventConnection();

mEventTube = mEvents->getDataChannel();

mLooper->addFd(mEventTube->getFd(), 0, Looper::EVENT\_INPUT,

MessageQueue::cb\_eventReceiver, this);

}

1.2 请求vsync信号：

@1：WMS或者Display通过SurfaceControl的closeTransaction来提交一次Transaction。

通过binder线程调用到surfaceflinger中的**setTransactionState**函数。

该函数主要用来提交一些layer的ComposerState值，和Display改变DisplayState的值。

#0 android::MessageQueue::invalidate (this=<optimized out>)

#1 0xb6f393e4 in setTransactionFlags (this=0xb7c48df0, flags=<optimized out>)

#2 android::SurfaceFlinger::setTransactionState (this=0xb7c48df0, state=..., displays=..., flags=0)

#3 0xb6cd3dee in android::BnSurfaceComposer::onTransact (this=0xb7c48df0, code=<optimized out>,

#4 0xb6f3b0b8 in android::SurfaceFlinger::onTransact (this=0xb7c48df0, code=8, data=...,

#5 0xb6ee79ee in android::BBinder::transact (this=0xb7c48df4, code=8, data=..., reply=0xb66f0820, flags=16)

#6 0xb6eed02a in android::IPCThreadState::executeCommand (

#7 0xb6eed1ac in android::IPCThreadState::getAndExecuteCommand (this=this@entry=0xb7c4a720)

#8 0xb6eed224 in android::IPCThreadState::joinThreadPool (this=0xb7c4a720, isMain=<optimized out>)

#9 0xb6ef2194 in android::PoolThread::threadLoop (this=0xb7c4a6e8)

#10 0xb6ec5aee in android::Thread::\_threadLoop (user=0xb7c4a6e8)

#11 0xb6dfa100 in \_\_pthread\_start (arg=0xb66f0930, arg@entry=<error reading variable:

#12 0xb6dd10d4 in \_\_start\_thread (fn=<optimized out>, arg=<optimized out>) at bionic/libc/bionic/clone.cpp:41

#13 0x00000000 in ?? ()

调用setDisplayStateLocked和setClientStateLocked函数来设置display和Layer的属性。

当transactionFlags有改变的时候，调用setTransactionFlags请求一次vsync。

@2：app端完成surface中的绘制进行BufferQueueProducer的queueBuffer操作：

(gdb) bt

#0 android::MessageQueue::invalidate (this=<optimized out>)

#1 0xb6f2fd98 in android::Layer::onFrameAvailable (this=0xb7d558a0, item=...)

#2 0xb6cca2d0 in android::ConsumerBase::onFrameAvailable (this=<optimized out>, item=...)

#3 0xb6cc2244 in android::BufferQueue::ProxyConsumerListener::onFrameAvailable (this=<optimized out>,

#4 0xb6cc826c in android::BufferQueueProducer::queueBuffer (this=<optimized out>, slot=<optimized

#5 0xb6ccff8c in android::BnGraphicBufferProducer::onTransact (this=0xb7d6f8e8, code=<optimized out>,

#6 0xb6ee79ee in android::BBinder::transact (this=0xb7d6f8ec, code=7, data=..., reply=0xb66f082

#7 0xb6eed02a in android::IPCThreadState::executeCommand (this=this@entry=0xb7c4a720,

#8 0xb6eed1ac in android::IPCThreadState::getAndExecuteCommand (this=this@entry=0xb7c4a720)

#9 0xb6eed224 in android::IPCThreadState::joinThreadPool (this=0xb7c4a720, isMain=<optimized out>)

#10 0xb6ef2194 in android::PoolThread::threadLoop (this=0xb7c4a6e8)

#11 0xb6ec5aee in android::Thread::\_threadLoop (user=0xb7c4a6e8)

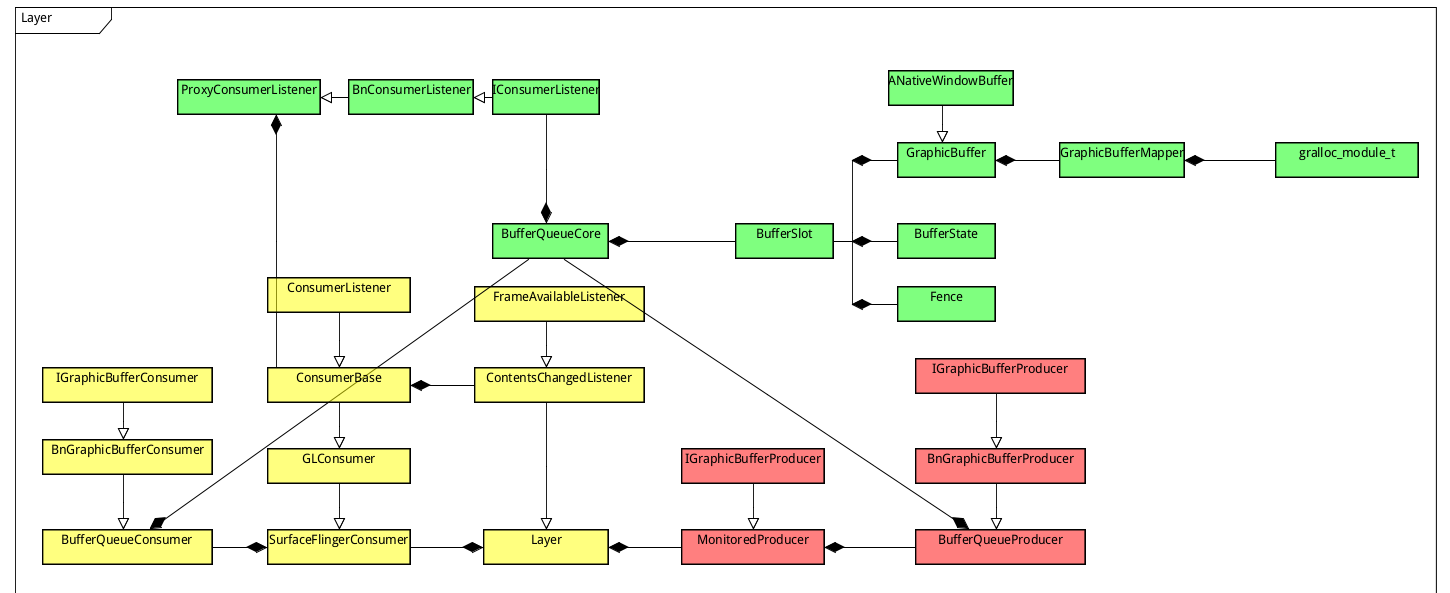
#12 0xb6dfa100 in \_\_pthread\_start (arg=0xb66f0930, arg@entry=<error reading variable: value has been

#13 0xb6dd10d4 in \_\_start\_thread (fn=<optimized out>, arg=<optimized out>)

#14 0x00000000 in ?? ()

MonitoredProducer最终传送给SurfaceControl用来创建给app进行绘制的Surface。

当app绘制完成后进行BufferQueueProducer::queueBuffer操作，最终调用到ConsumerBase::onFrameAvailable函数中后调用layer的onFrameAvailable函数，在该函数中调用signalLayerUpdate函数请求vsync信号。



BufferSlot的状态：

FREE

DEQUEUED

QUEUED

ACQUIRED

1. main\_surfaceFlinger对vsync的处理：

Surfaceflinger主线程在接收到vsync信号后发送MessageQueue::INVALIDATE消息到主线程。

void SurfaceFlinger::onMessageReceived(int32\_t what) {

switch (what) {

case MessageQueue::TRANSACTION: {

handleMessageTransaction();

break;

}

case MessageQueue::INVALIDATE: {

bool refreshNeeded = handleMessageTransaction();

refreshNeeded |= handleMessageInvalidate();

refreshNeeded |= mRepaintEverything;

if (refreshNeeded) {

signalRefresh();

}

break;

}

case MessageQueue::REFRESH: {

handleMessageRefresh();

break;

}

}

}

Layer的两个State：主要包括当前layer的一些属性

State mCurrentState：在setTransactionState函数中调用setClientStateLocked对每个layer进行修改mCurrentState的状态。

State mDrawingState：在doTransaction之后将mCurrentState赋值给mDrawingState，保存下一次setTransactionState之前的layer的状态。

这两个状态的作用主要是在doTransaction中使用mDrawingState和mCurrentState进行比较查看layer的属性是否发生了变化。

Surfaceflinger的两个State：主要包括所有displaydevice所有layer的集合。

State mCurrentState：

State mDrawingState;

双缓冲属性的作用：

**handleTransactionLocked详解：**

1. 遍历mCurrentState中所有的layer做doTransaction。

const LayerVector& currentLayers(mCurrentState.layersSortedByZ);

const size\_t count = currentLayers.size();

if (transactionFlags & eTraversalNeeded) {

for (size\_t i=0 ; i<count ; i++) {

const sp<Layer>& layer(currentLayers[i]);

uint32\_t trFlags = layer->getTransactionFlags(eTransactionNeeded);

if (!trFlags) continue;

**//检查layer是否发生了状态的变化**

const uint32\_t flags = layer->doTransaction(0);

if (flags & Layer::eVisibleRegion)

mVisibleRegionsDirty = true;

}

}

1. 当前DisplayDevice和之前DisplayDevice不一致时遍历mCurrentState和mDrawingState中所有的DisplayDevice进行比较操作：

2.1. remove DisplayDevice

2.2. update DisplayDevice

2.3. add DisplayDevice

1. 为所有的layer调用updateTransformHint

知识点：layer中的layerstack和DisplayDevice的layerstack的关系

if (transactionFlags & (eTraversalNeeded|eDisplayTransactionNeeded)) {

sp<const DisplayDevice> disp;

uint32\_t currentlayerStack = 0;

for (size\_t i=0; i<count; i++) {

const sp<Layer>& layer(currentLayers[i]);

uint32\_t layerStack = layer->getDrawingState().layerStack;

if (i==0 || currentlayerStack != layerStack) {

currentlayerStack = layerStack;

**//清除disp**

disp.clear();

for (size\_t dpy=0 ; dpy<mDisplays.size() ; dpy++) {

sp<const DisplayDevice> hw(mDisplays[dpy]);

if (**hw->getLayerStack() == currentlayerStack**) {

if (disp == NULL) {

**//找到了一个layerStacker相同的显示设备**

disp = hw;

} else {

**//如果有两个显示设备的layerStacker相同，都不用**

disp = NULL;

break;

}

}

}

}

if (disp == NULL) {

**// 没有找到具有相同layerStack的显示设备，使用缺省设备**

disp = getDefaultDisplayDevice();

}

**//设置Layer对象的TransformHint**

layer->updateTransformHint(disp);

}

}

1. 处理Layer增加情况和commitTransaction设置mDrawingState。

**handlePageFlip函数的详解：**

知识点：ANativeWindowBuffer 和ANativeWindow的关系

Vector<Layer\*> layersWithQueuedFrames;

for (size\_t i = 0, count = layers.size(); i<count ; i++) {

const sp<Layer>& layer(layers[i]);

if (layer->hasQueuedFrame()) {

frameQueued = true;

if (layer->shouldPresentNow(mPrimaryDispSync)) {

layersWithQueuedFrames.push\_back(layer.get());

} else {

layer->useEmptyDamage();

}

} else {

layer->useEmptyDamage();

}

}

for (size\_t i = 0, count = layersWithQueuedFrames.size() ; i<count ; i++) {

Layer\* layer = layersWithQueuedFrames[i];

const Region dirty(layer->latchBuffer(visibleRegions));

layer->useSurfaceDamage();

const Layer::State& s(layer->getDrawingState());

invalidateLayerStack(s.layerStack, dirty);

}