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)

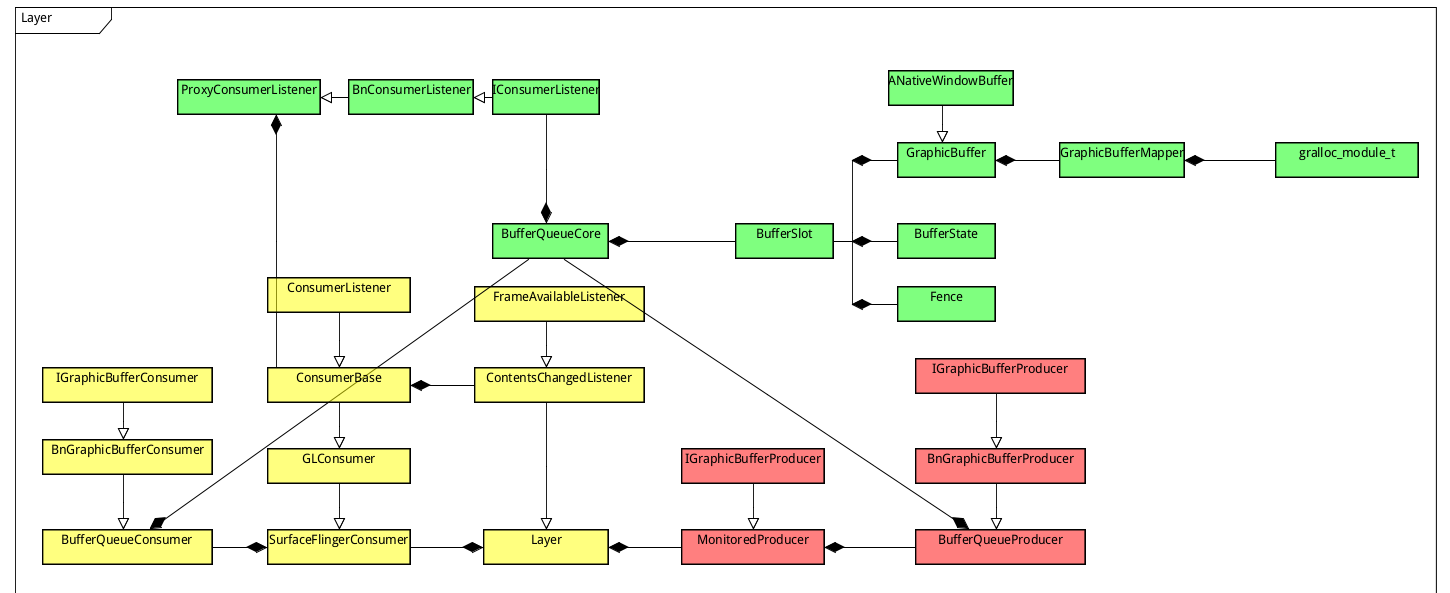
#12 0xb6dfa100 in \_\_pthread\_start (arg=0xb66f0930, arg@entry=<error reading variable: value has been

#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。



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;

State mDrawingState;

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

State mCurrentState;

State mDrawingState;

handleTransactionLocked详解：