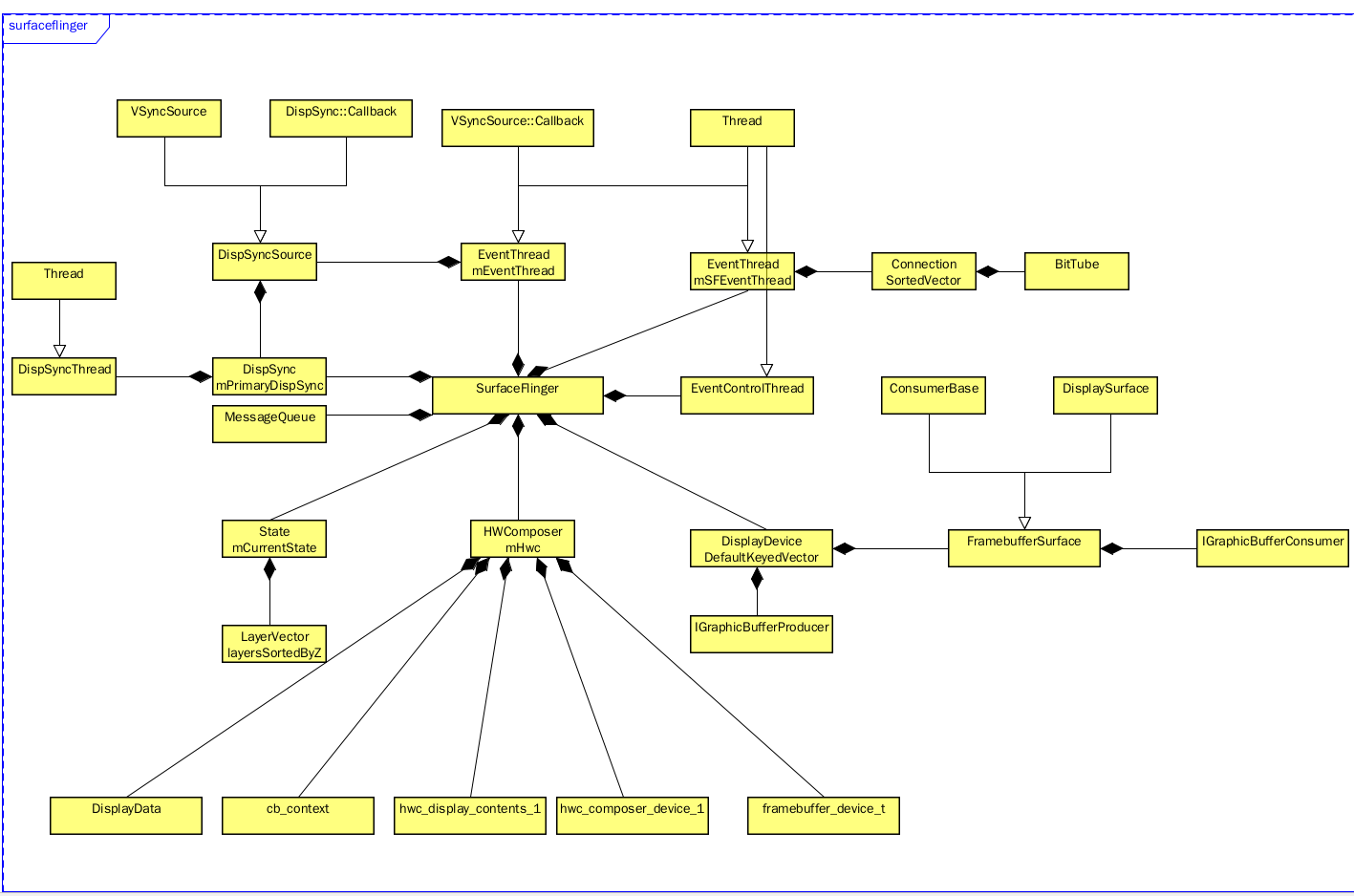
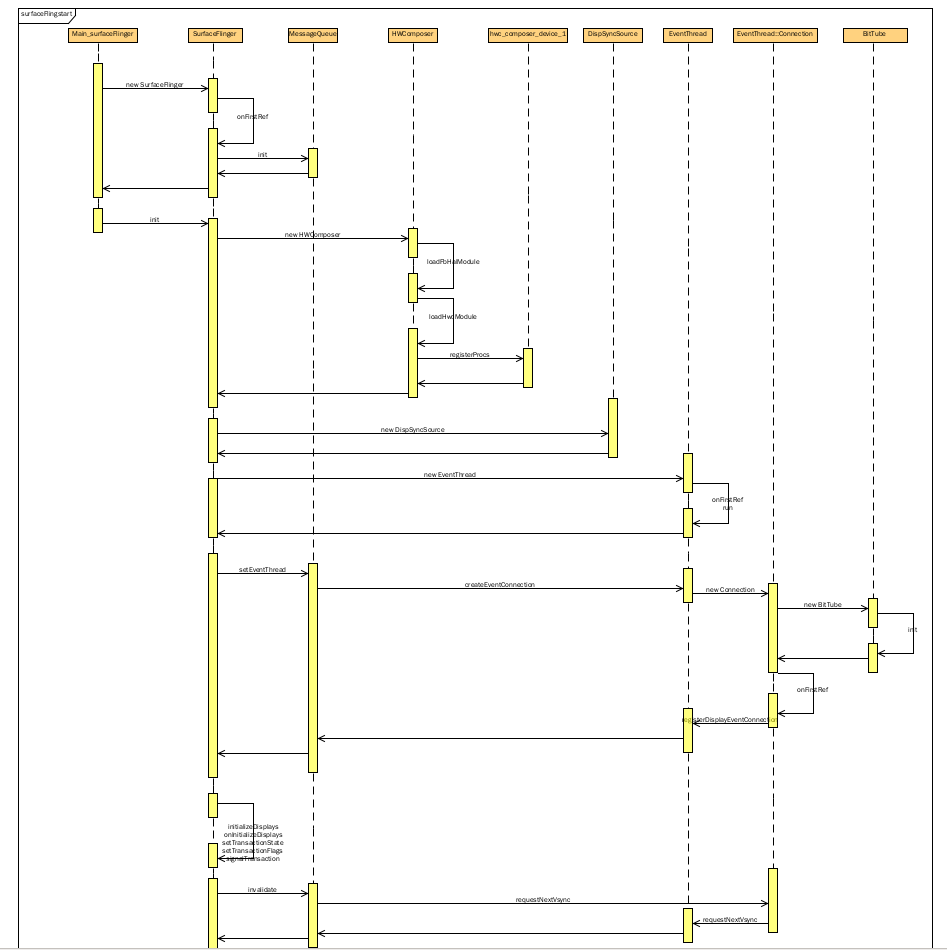
1. **surfaceflinger的啟動和初始化**

**surfaceFlinger结构：**

****

**surfaceFlinger的初始化：**

****

main\_surfaceflinger：

1.初始化MessageQueue。

2.Surfaceflinger init：

new HWComposer、new FramebufferSurface、new DisplayDevice、new DispSyncSource、

new EventThread、new EventControlThread；

DispSync對象中創建DispSyncThread。

EventThread線程啟動後會在mCondition.wait上等待，

**Wake up 1**:surfacefling主線程會調用messagequene的setEventThread函數創建一個EventConnection對象註冊到sf的EventThread線程中，同時喚醒wait的EventThread線程.

**Wake up 2:（initializeDisplays）** requestNextVsync將connection->count = 0 賦值為0，調用enableVSyncLocked函數，將EventThread對象註冊到DispSyncThread線程中开始接受VSync信号。

**Wake up 3: 接收从**DispSyncThread线程传送过来的VSync信号，onVSyncEvent將add的connection返回，進行postEvent操作。

创建EventControlThread 线程来设置HW VSYNC信号是否可用。

3.surfaceflinger中的启动后的線程：adb shell ps –t pid

system 317 1 261368 12544 ffffffff 7e9772bc S /system/bin/surfaceflinger

system 501 317 261368 12544 0037008c 7e97725c S Binder\_1

system 502 317 261368 12544 0037008c 7e97725c S Binder\_2

system 503 317 261368 12544 001100d8 7e92d794 S DispSync

system 526 317 261368 12544 001100d8 7e92d794 S ged-swd

system 601 317 261368 12544 001b252c 7e976dac S UEventThreadHWC

system 602 317 261368 12544 001100d8 7e92d794 S Dispatcher\_0

system 603 317 261368 12544 001100d8 7e92d794 S UICompThread\_0

system 619 317 261368 12544 001100d8 7e92d794 S MMCompThread\_0

system 620 317 261368 12544 001100d8 7e92d794 S VSyncThread\_0

system 626 317 261368 12544 000c4a80 7e9771cc S SFWatchDog

system 627 317 261368 12544 000b0eb0 7e976f2c S surfaceflinger

system 628 317 261368 12544 001100d8 7e92d794 S EventThread

system 629 317 261368 12544 000b0eb0 7e976f2c S surfaceflinger

system 630 317 261368 12544 001100d8 7e92d794 S EventThread

system 631 317 261368 12544 001100d8 7e92d794 S EventControl

system 760 317 261368 12544 0037008c 7e97725c S Binder\_3

system 880 317 261368 12544 0037008c 7e97725c S Binder\_4

system 2182 317 261368 12544 0037008c 7e97725c S Binder\_5

/system/lib64/hw/hwcomposer.mt6753.so 庫中的線程：

.UEventThreadHWC

.Dispatcher\_0

.UICompThread\_0

.MMCompThread\_0

.VSyncThread\_0

/system/lib64/libsurfaceflinger.so庫中的線程：

.surfaceflinger

.DispSync

.SFWatchDog

.EventThread

.EventThread

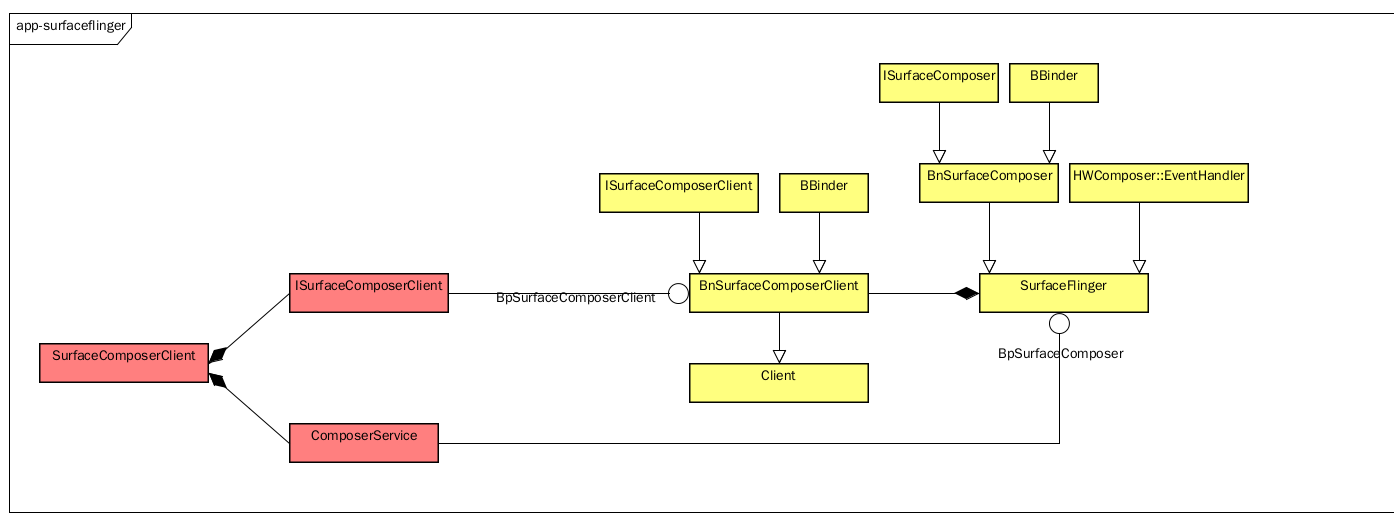
.EventControl

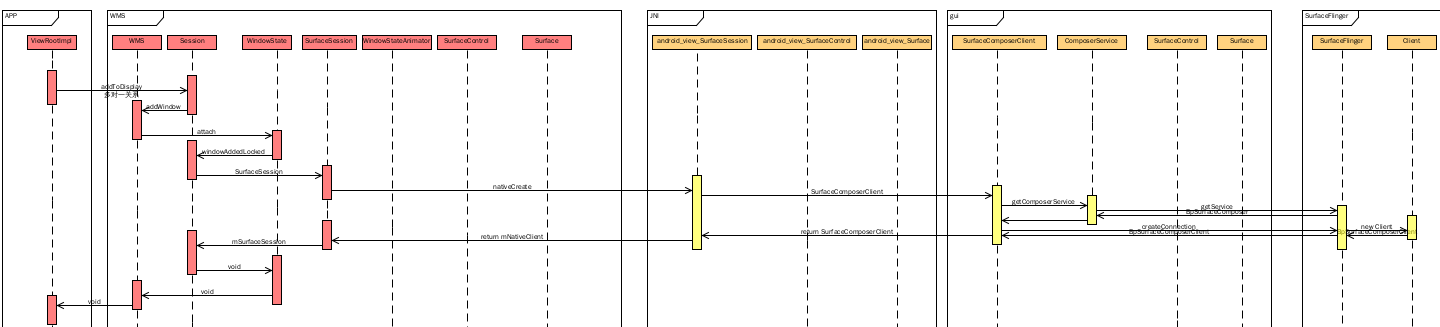
/system/lib64/egl/libGLES\_mali.so庫中的線程

5個binder線程

1. **App建立和surfaceflinger之間的聯繫**

**Surfacesession创建的流程：**

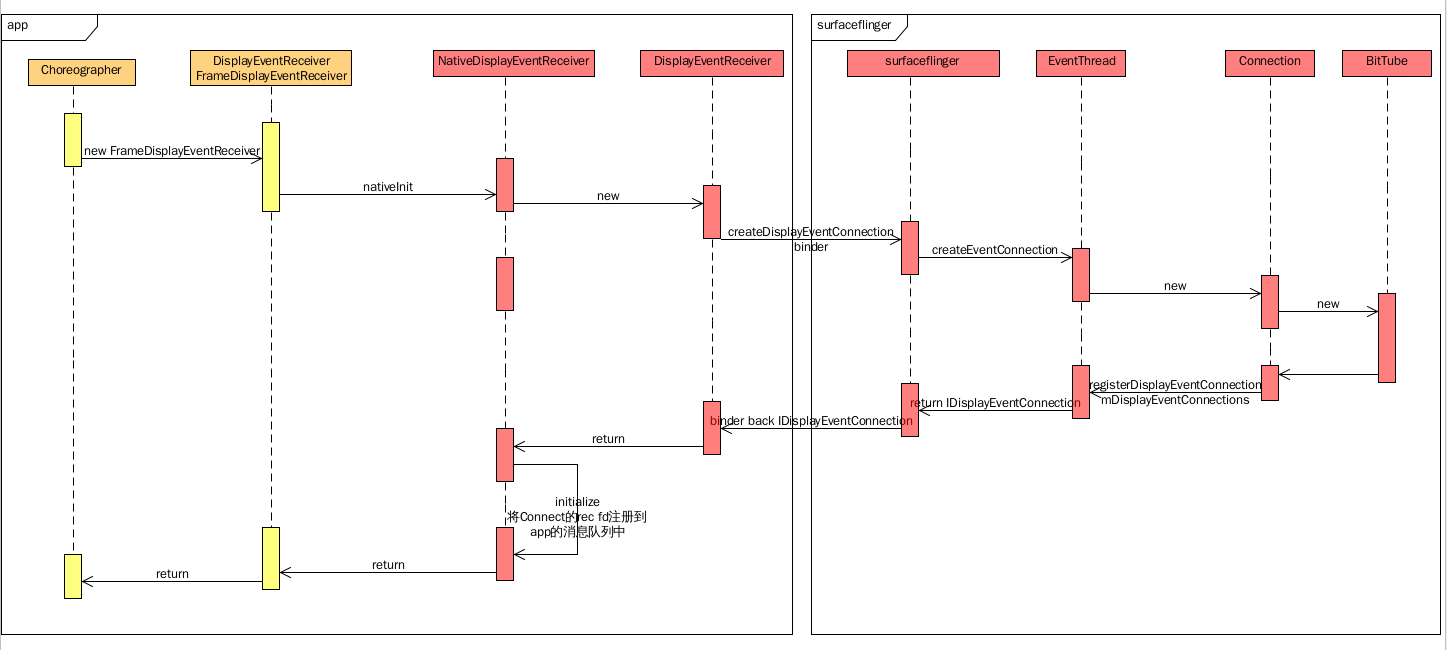




每個app對應一個Client。

每個進程在創建第一個窗口add的時候，在單实例的IWindowSession對象中會創建一個单实例SurfaceSession對象，這個對象就是一個進程和surfaceFlinger服務進行通信的代理對象。

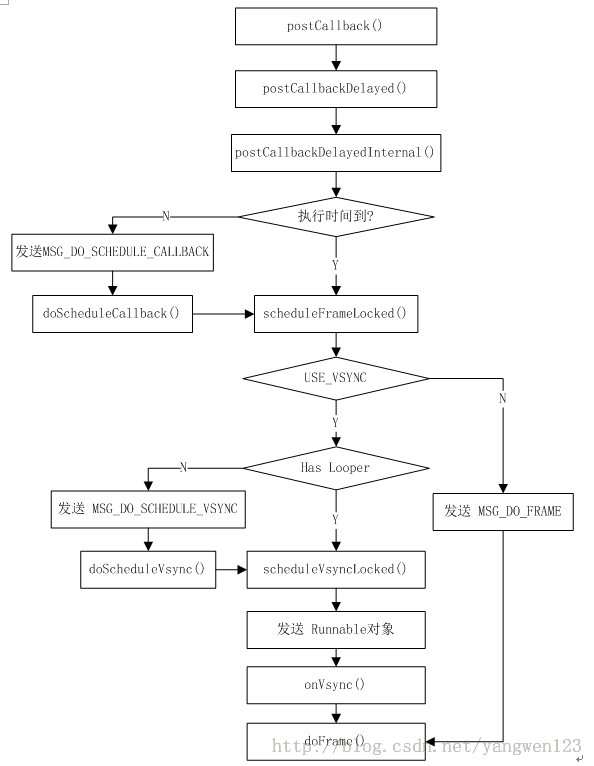
**App和surfaceflinger vsync信号通信机制：**



一个app对应一个Choreographer（TLS）对象，Choreographer直接和SurfaceFlinger进行通信，在EventThread中注册一个Connect 进行Vsync信号的传送。

Callback队列类型：CALLBACK\_INPUT、CALLBACK\_ANIMATION、CALLBACK\_TRAVERSAL。

DO\_FRAME 是帧刷新，SCHEDULE\_VSYNC是 垂直同步刷新。



在Choreographer定义数组队列new CallbackQueue[CALLBACK\_LAST + 1]给ViewRootImpl用来进行postCallback，将CallbackRecord加入到CallbackQueue队列。当Callback是立即执行的，调用scheduleFrameLocked函数，当需要延时处理的发送延时消息MSG\_DO\_SCHEDULE\_CALLBACK。在scheduleFrameLocked函数中检查是否使用VSync信号，使用的话检查是否在UI线程中，如果是的话直接调用scheduleVsyncLocked函数来请求VSync信号。

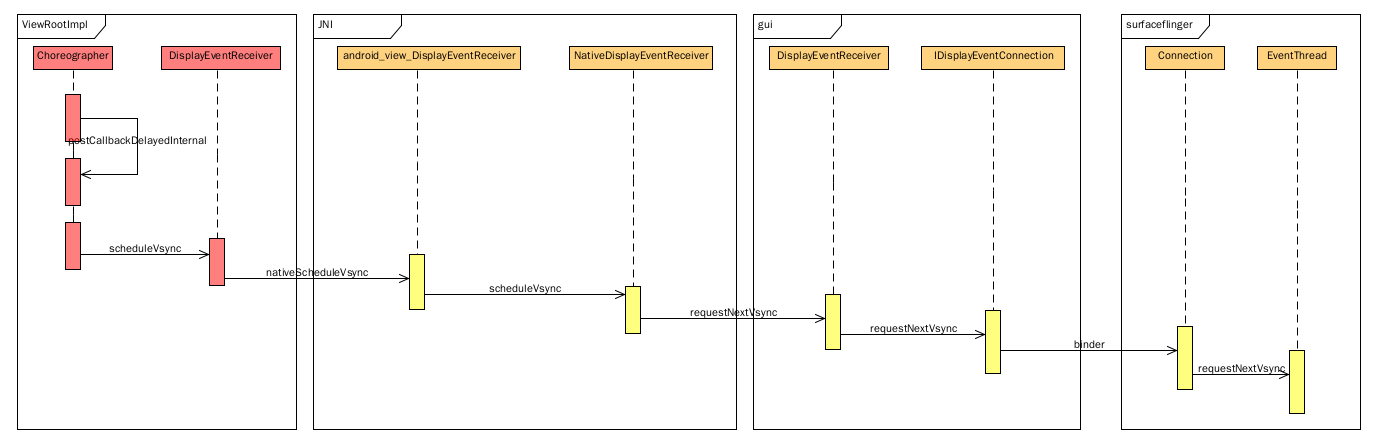
时间：VSync信号产生的时间、每一帧刷新频率的时间、最后一次Vsync处理时间.

postCallback 操作的场景：

invalidateChildInParent.

requestLayout.

invalidate



1. **第一個Vsync信號的處理進行layout**

ViewRootImpl.setView ->requestLayout->scheduleTraversals

请求第一个vsync信号

Window layout的時候會創建一個SurfaceControl，同時將IWindowSession中的SurfaceSession對象傳入到該對象，通過SurfaceSession對象來獲取surfaceflinger中的BufferQueueProducer buffer，同時用這個Producer創建SurfaceControl返回。

在WMS的relayoutWindow函數中通過SurfaceSession創建完成SurfaceControl後，通過Surface類的copyFrom函數調用native層的SurfaceControl來new一個Surface對象返回。最終該Surface對象保存在viewRootImpl中的mSurface對象中，WMS 中为每个窗口创建SurfaceControl。

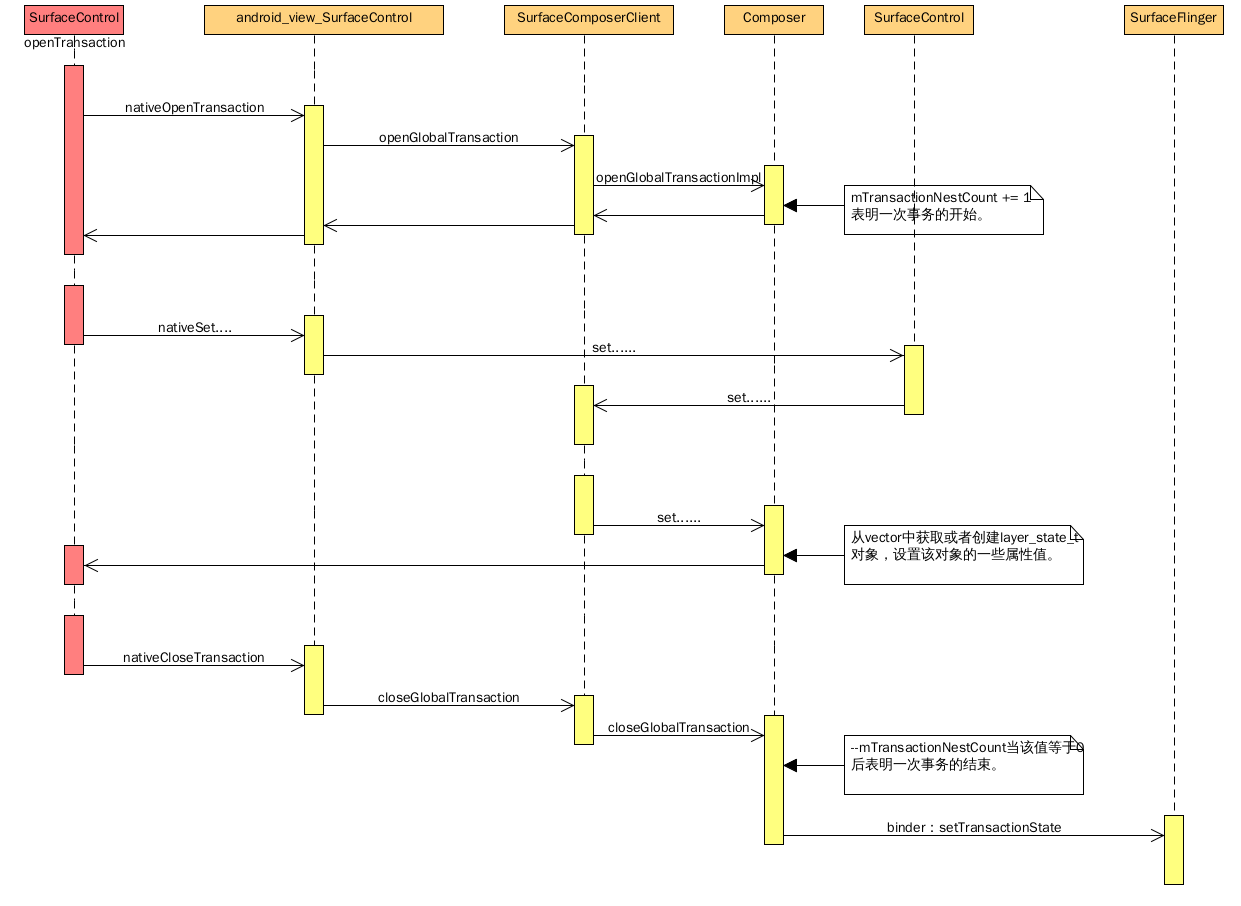
**创建的一些重要的对象：**

|  |  |  |  |
| --- | --- | --- | --- |
| 对象名称 | 创建者 | 类型 | 备注 |
| SurfaceSession | System\_server  （addWindow） | Java对象 | 每个进程在第一次创建窗口的时候请求WMS来创建，其在对应一个gui端对应一个SurfaceComposerClient对象。 |
| SurfaceComposerClient | System\_server  （addWindow） | Native对象 | 和SurfaceSession对象一一对应。 |
| SurfaceControl | System\_server  （relayoutWindow） | Java对象 | 每个窗口对应一个SurfaceControl对象，和gui中的SurfaceControl对象一一对应。 |
| SurfaceControl | System\_server  （relayoutWindow） | Native对象 | SurfaceComposerClient中创建，和java层的SurfaceControl对象一一对应。 |
| Composer | System\_server | Native 对象 | 是一个Singleton对象，在第一个SurfaceComposerClient对象创建的时候创建，除了开机动画，整个系统只会创建该对象一次。该对象中维护着ComposerState、DisplayState列表。 |

**WMS这时候的角色：**

通过surfacecontrol来设置surface的属性，统称为一次事务。

setLayer、setPosition、setSize、setFlags、setAlpha、setMatrix、setCrop、setLayerStack



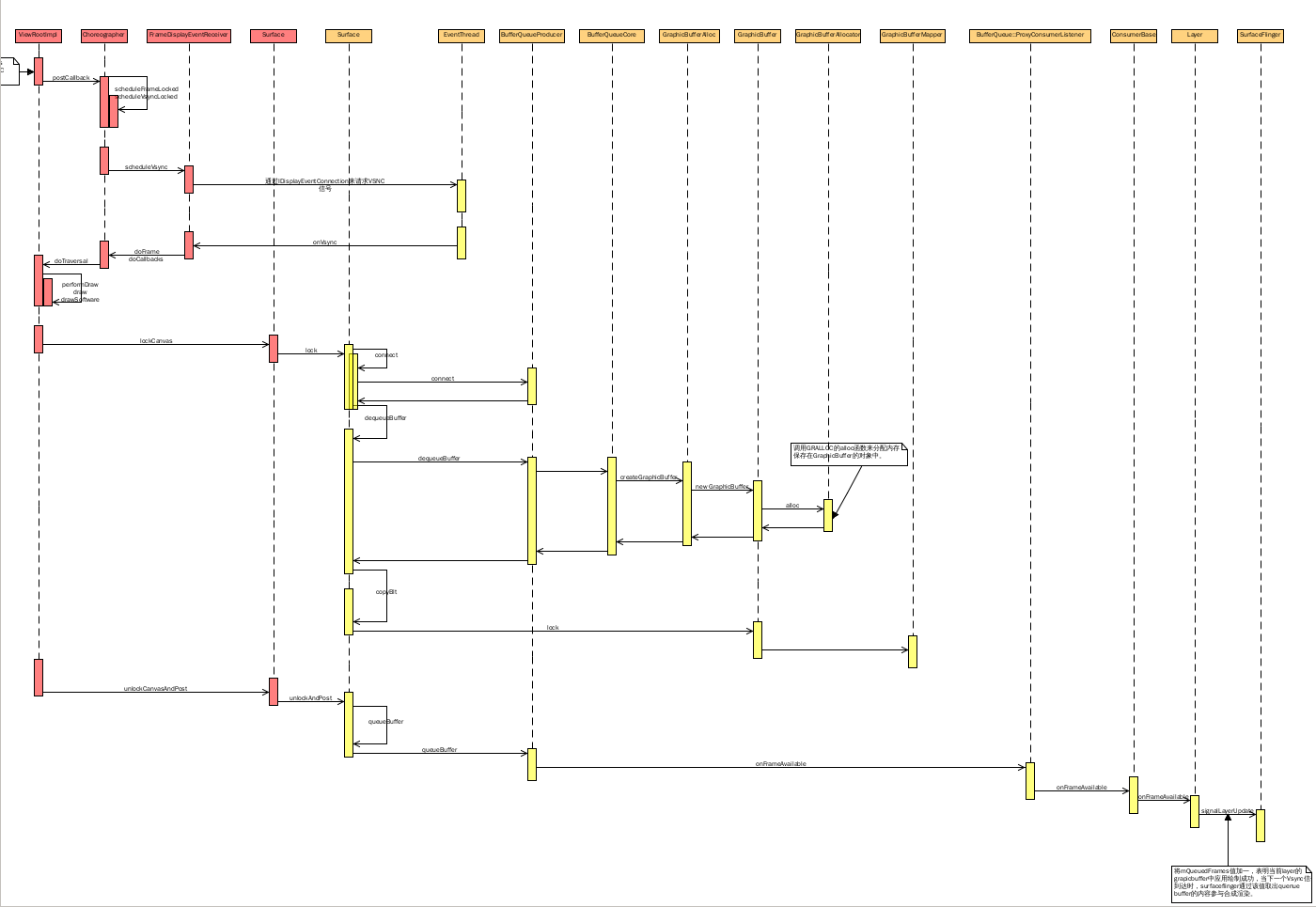
setTransactionState 详解：

遍历ComposerState容器，通过what标志位来设置每个layer相应改变的属性。

对每个layer处理完后，设置mTransactionFlags标志的值，表明有layer属性改变的事务请求，为合成渲染使用。

1. **後序的Vsync信號進行繪製和合成**

**1. Software Drawn**

****

typedef struct ANativeWindowBuffer ANativeWindowBuffer\_t；

typedef ANativeWindowBuffer\_t android\_native\_buffer\_t;

GraphicBuffer繼承與ANativeWindowBuffer；

每個窗口對應的為ANativeWindowBuffer；

ANativeWindow\_Buffer

dequeueBuffer：保證獲取一個free的mSlots，同時返回GraphicBuffer保存在surface中，調用GraphicBuffer的lockAsync方法返回bits。

**2. Hardware Drawn**

什么时候关闭硬件加速：

ActivityThread：

private void handleBindApplication(AppBindData data) {

if (data.persistent) {

// Persistent processes on low-memory devices do not get to

// use hardware accelerated drawing, since this can add too much

// overhead to the process.

if (!ActivityManager.isHighEndGfx()) {

HardwareRenderer.disable(false);

}

}

}

public static ActivityThread systemMain() {

// The system process on low-memory devices do not get to use hardware

// accelerated drawing, since this can add too much overhead to the

// process.

if (!ActivityManager.isHighEndGfx()) {

HardwareRenderer.disable(true);

} else {

HardwareRenderer.enableForegroundTrimming();

}

ActivityThread thread = new ActivityThread();

thread.attach(true);

return thread;

}

非persistent进程、system 进程而且PRIVATE\_FLAG\_FORCE\_HARDWARE\_ACCELERATED可以硬件加速。

1. ThreadedRenderer

2.绑定窗口到ThreadedRenderer

3. draw

RenderProxy:

CREATE\_BRIDGE 宏（声明Args 和Bridge\_）

SETUP\_TASK 宏（定义MethodInvokeRenderTask）

postAndWait 中task对象的销毁

**3．Composition and render**

1. **總結：**

new Choreographer ：創建窗口和surfaceflinger之間Vsync信號通信鏈接。

setview:創建整個進程與surfaceflinger之間通信的session。

Layout：為窗口創建一個surface app層的Surfac就是一個BufferQueueProducer對象，該對象和BufferQueueConsumer共享一塊內存，Producer返回給app，Consumer保存在layer中給surfacefling合成的時候調用。

1. **知識點：**

1.Gralloc

gralloc\_module\_t

2.Framebuffer

framebuffer\_device\_t

3.HWComposer

hwc\_composer\_device\_1

4.TLS、Singleton

5.Linux中的Condition(pthread\_cond\_init)、Mutex(pthread\_mutex\_init)。

6.linux epoll\_wait event 是否丢失

7.vsync信号处理者：app UIThread、RenderThread、surfaceflinger

8. SurfaceControl.openTransaction 、SurfaceControl.closeTransaction的作用：

static void SurfaceControl.openTransaction()

static void nativeOpenTransaction

static void SurfaceComposerClient::openGlobalTransaction()

static void Composer::openGlobalTransaction()

Composer::getInstance().openGlobalTransactionImpl()

9. WindowAnimator\WindowStateAnimator\AppWindowAnimator

10. addWindow:创建与input、surfaceflinger的通信通道。