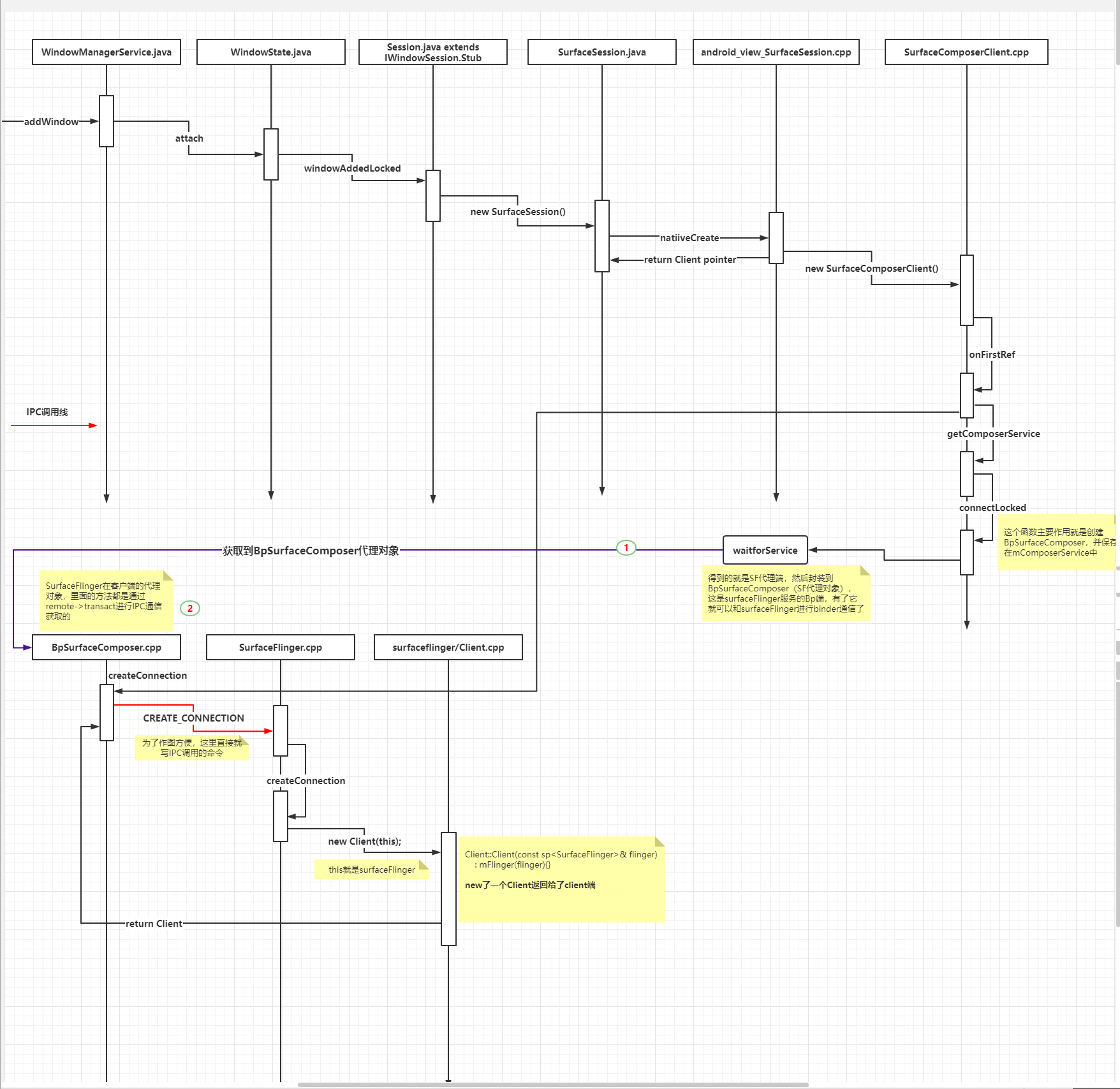
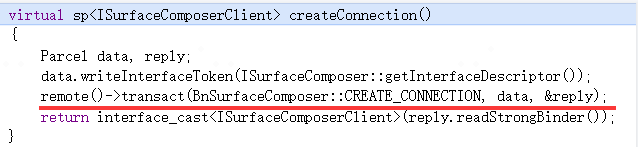
Window和SurfaceFlinger建立连接分析

流程图：

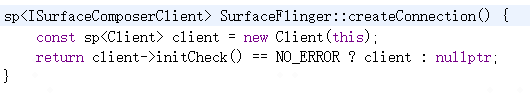




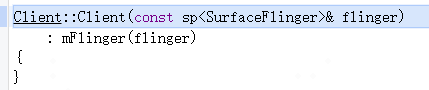
**通过IPC调用，然后强转为SUrfaceComposerClient**



**将this作为参数，new Client**



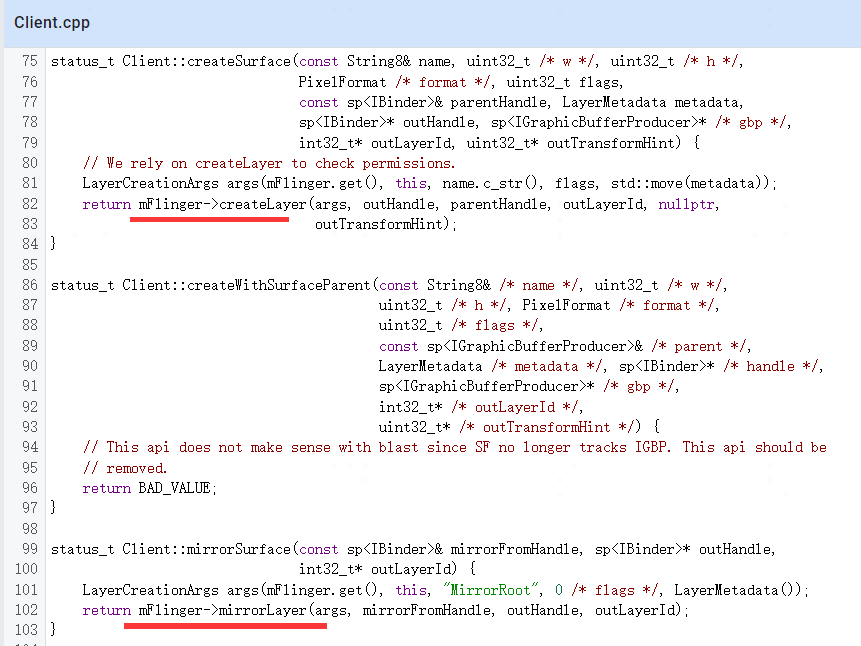
**this就是SurfaceFlinger对象**



**Client的构造函数很简单，什么也没做，就是保存了SurfaceFlinger的引用，Client继承BnSurfaceComposerClient**

clipboard.png

**作为Binder服务端，可以想到Client做的事情还是需要SurfaceFlinger来做的，去看Client.cpp源码发现，所有方法中其实还是调用mFlinger->XXX，Client只是一层封装：**



**步骤2分析完了，主要就是在SurfaceFlinger进程创建了一个Client对象，并返回，返回到SurfaceComposerClient这边再通过interface\_cast转换为了BpSurfaceComposerClient，保存在了SurfaceComposerClient的mClient变量中，自此window和surfaceFlinger的连接就建立了**

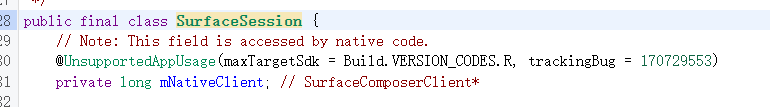
**其实步骤1和步骤2创建了两个SurfaceFlinger进程的Bp端，一个是BpSurfaceComposer，这个Bp端是直接调用SurfaceFlinger的函数，还有一个BpSurfaceComposerClient，这个Bp端是通过SurfaceFlinger进程的Client对象间接调用SurfaceFlinger的函数**

**总结：**

**1、每一个window最终都会调用WMS的addWindow方法进行添加，并且创建一个WindowState来描述窗口信息**

**2、接着调用WindowState的attach方法，通过Session的windowAddedLocked方法创建SurfaceSession对象**

**3、SurfaceSession对象构造方法中调用nativeCreate方法创建native层的SurfaceComposerClient对象，并保存其指针在自己的mNativeClient中：**



**4、SurfaceComposerClient对象首次创建时通过初始化函数onFirstRef分两步建立与surfaceflinger进程的连接**

**5、步骤1，创建surfaceflinger的Bp端BpSurfaceComposer，并保存在SurfaceComposerClient的mComposerService变量中；**

**步骤2，通过BpSurfaceComposer调用surfaceflinger的createConnection函数在surfaceflinger进程中创建Bn端Client对象，并返回，SurfaceComposerClient中保存的是Client的Bp端BpSurfaceComposerClient，并保存在SurfaceComposerClient的mClient变量中。**