**2.WindowManager 之add window**

1. activity window

@1.activity window 的创建：

为activity 窗口指定display：

private Activity performLaunchActivity(ActivityClientRecord r, Intent customIntent) {

if (activity != null) {

Context appContext = createBaseContextForActivity(r, activity);

......

}

private Context createBaseContextForActivity(ActivityClientRecord r, final Activity activity) {

int displayId = Display.DEFAULT\_DISPLAY;

try {

**displayId** = ActivityManagerNative.getDefault().getActivityDisplayId(r.token);

} catch (RemoteException e) {

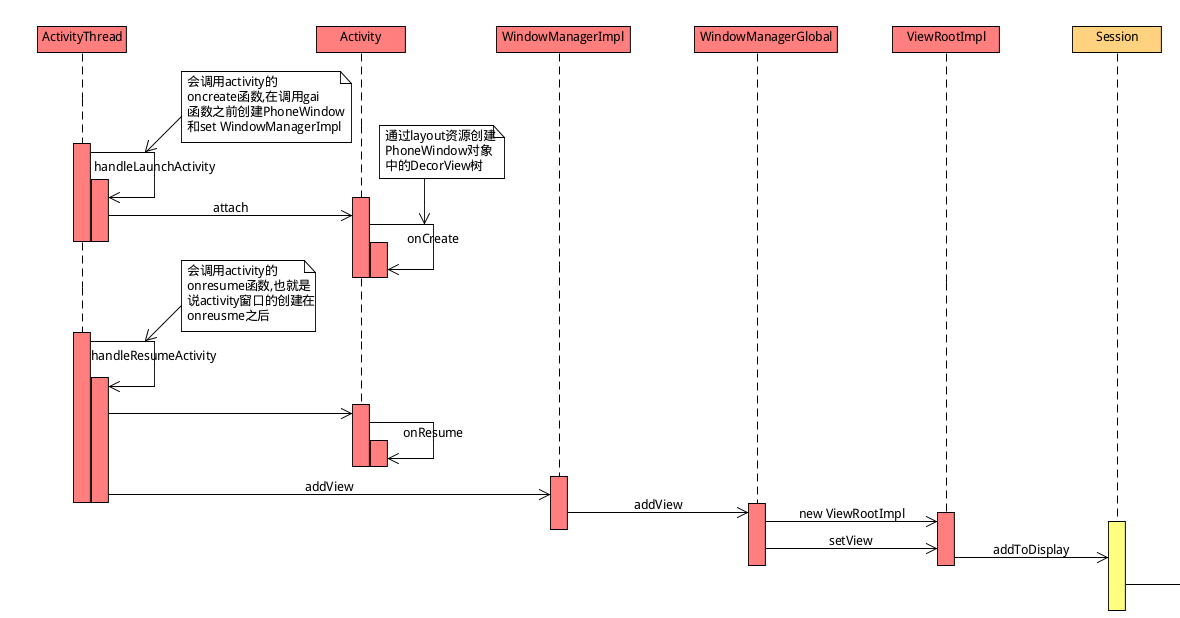
}

ContextImpl appContext = ContextImpl.createActivityContext(

this, r.packageInfo, displayId, r.overrideConfig);

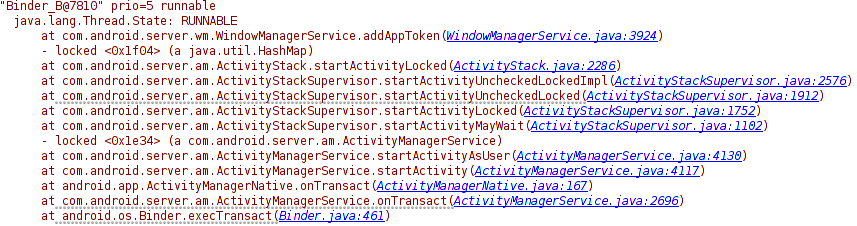
}

Activity 所属的display由AMS来指定，保存在activity的context中同时赋值给activity的WindowManagerImpl中的Display对象。



addWindow函数的解析：

1. 检查创建该窗口所需的display是否存在mDisplayContents容器中，存在返回，不存在创建该display。
2. 检查窗口是否已经添加到mWindowMap容器中。
3. 如果是sub window检查其parent window是否存在。
4. 检查WindowToken对象是否存在于mTokenMap容器中，WindowToken在startActivity的时候创建并且添加到mTokenMap容器中。



1. 创建WindowState对象。

通过windowTypeToLayerLw函数根据窗口的类型来分配mBaseLayer层级。

创建WindowStateAnimator对象，对SurfaceControl对象的管理。

创建InputWindowHandle对象。

@1：mBaseLayer的计算：

调用windowTypeToLayerLw进行计算，根据window type返回相应的值，比如TYPE\_STATUS\_BAR返回16，乘以10000加上1000 等于161000。

App都是21000.

@2：window的插入的位置顺序（生成window list）：

private void addWindowToListInOrderLocked(final WindowState win, boolean addToToken) {

if (win.mAttachedWindow == null) { //非子窗口

final WindowToken token = win.mToken;

int tokenWindowsPos = 0;

if (token.appWindowToken != null) {

tokenWindowsPos = addAppWindowToListLocked(win);//非系统窗口

} else {

addFreeWindowToListLocked(win); //系统窗口

}

if (addToToken) {

token.windows.add(tokenWindowsPos, win);

}

} else {

addAttachedWindowToListLocked(win, addToToken);

}

if (win.mAppToken != null && addToToken) {

win.mAppToken.allAppWindows.add(win);

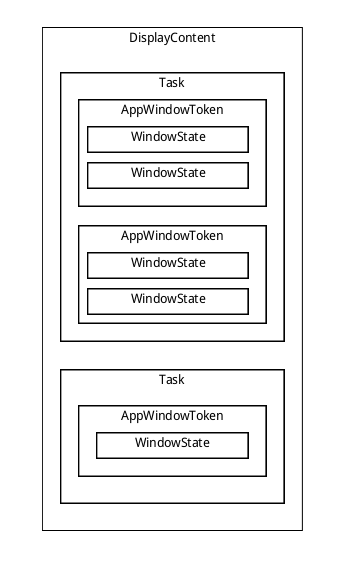
}

}

AppWindowToken继承于WindowToken；

addAppWindowToListLocked：

对app window进行添加到window list中.



private int addAppWindowToListLocked(final WindowState win) {

// Figure out where the window should go, based on the

// order of applications.

WindowState pos = null;

final ArrayList<Task> tasks = displayContent.getTasks();

int taskNdx;

int tokenNdx = -1;

for (taskNdx = tasks.size() - 1; taskNdx >= 0; --taskNdx) { //从最大的的task对其app token进行遍历

AppTokenList tokens = tasks.get(taskNdx).mAppTokens;

for (tokenNdx = tokens.size() - 1; tokenNdx >= 0; --tokenNdx) {//对task中取出的toke list进行遍历

final AppWindowToken t = tokens.get(tokenNdx);

if (t == token) { //如果token和添加的窗口的token 相同，则不需要进行遍历

--tokenNdx; //将查找获取的toke index减1从而获取其与其相邻的app window token

if (tokenNdx < 0) { //当查找的token之前没有相应窗口的token说明该窗口是该task第一次添加的token

--taskNdx; //那么将task index 减1，说明该添加窗口的token 需要加载与其相邻task的最后一个窗口之后。

if (taskNdx >= 0) {

tokenNdx = tasks.get(taskNdx).mAppTokens.size() - 1; //获取需要添加的token的index

}

}

break;

}

}

if (tokenNdx >= 0) {

// early exit

break;

}

}

if (pos != null) {

// Move behind any windows attached to this one.

WindowToken atoken = mTokenMap.get(pos.mClient.asBinder());

if (atoken != null) {

tokenWindowList =

getTokenWindowsOnDisplay(atoken, displayContent);

final int NC = tokenWindowList.size();

if (NC > 0) {

WindowState bottom = tokenWindowList.get(0);

if (bottom.mSubLayer < 0) {

pos = bottom;

}

}

}

placeWindowBefore(pos, win);

return tokenWindowsPos;

}

for ( ; taskNdx >= 0; --taskNdx) { //遍历之前获取的task 从最大task开始遍历

AppTokenList tokens = tasks.get(taskNdx).mAppTokens;

for ( ; tokenNdx >= 0; --tokenNdx) {

final AppWindowToken t = tokens.get(tokenNdx);//遍历之前计算出来的token

tokenWindowList = getTokenWindowsOnDisplay(t, displayContent);

final int NW = tokenWindowList.size();

if (NW > 0) {

pos = tokenWindowList.get(NW-1);//获取该token list中index值最大的一个window

break;

}

}

if (tokenNdx >= 0) {

// found

break;

}

}

if (pos != null) {

// Move in front of any windows attached to this

// one.

WindowToken atoken = mTokenMap.get(pos.mClient.asBinder());

if (atoken != null) {

final int NC = atoken.windows.size();

if (NC > 0) {

WindowState top = atoken.windows.get(NC-1);

if (top.mSubLayer >= 0) {

pos = top;

}

}

}

placeWindowAfter(pos, win); //将当前窗口插入到遍历得到的窗口之后

return tokenWindowsPos;

}

assignLayersLocked：通过之前add window list进行mLayer和mAnimLayer值得计算。

private final void assignLayersLocked(WindowList windows) {

int N = windows.size();

**int curBaseLayer = 0;**

**int curLayer = 0;**

int i;

boolean anyLayerChanged = false;

for (i=0; i<N; i++) {

final WindowState w = windows.get(i);

final WindowStateAnimator winAnimator = w.mWinAnimator;

boolean layerChanged = false;

int oldLayer = w.mLayer;

if (w.mBaseLayer == curBaseLayer || w.mIsImWindow

|| (i > 0 && w.mIsWallpaper)) {//如果窗口的base layer和curBaseLayer 一致增加curLayer 的值。

curLayer += WINDOW\_LAYER\_MULTIPLIER;

**w.mLayer = curLayer;**//使用curLayer 值来改变窗口的layer值

} else {

curBaseLayer = curLayer = w.mBaseLayer;//如果curBaseLayer 不等于该窗口的base layer

重新为curBaseLayer 赋值。说明该窗口时该类型的第一个窗口

w.mLayer = curLayer;

}

if (w.mLayer != oldLayer) {

layerChanged = true;

anyLayerChanged = true;

}

final AppWindowToken wtoken = w.mAppToken;

oldLayer = winAnimator.mAnimLayer;

if (w.mTargetAppToken != null) {

winAnimator.mAnimLayer =

w.mLayer + w.mTargetAppToken.mAppAnimator.animLayerAdjustment;

} else if (wtoken != null) {

winAnimator.mAnimLayer =

w.mLayer + wtoken.mAppAnimator.animLayerAdjustment;

} else {

winAnimator.mAnimLayer = w.mLayer;

}

if (w.mIsImWindow) {

winAnimator.mAnimLayer += mInputMethodAnimLayerAdjustment;

} else if (w.mIsWallpaper) {

winAnimator.mAnimLayer += mWallpaperAnimLayerAdjustment;

}

if (winAnimator.mAnimLayer != oldLayer) {

layerChanged = true;

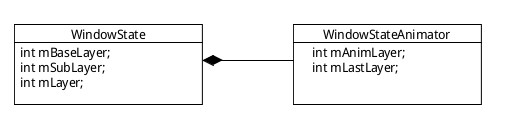
anyLayerChanged = true;

}

}

}

和layer相关的几个变量：



1. 创建InputChannel，将inputChannels[0]传送给InputManager，将inputChannels[1]传送给app。当inputChannels[0]上有input输入事件的时候直接将input event发送给app进行处理。

if (outInputChannel != null && (attrs.inputFeatures

& WindowManager.LayoutParams.INPUT\_FEATURE\_NO\_INPUT\_CHANNEL) == 0) {

String name = win.makeInputChannelName();

InputChannel[] inputChannels = InputChannel.openInputChannelPair(name);

win.setInputChannel(inputChannels[0]);

inputChannels[1].transferTo(outInputChannel);

mInputManager.registerInputChannel(win.mInputChannel, win.mInputWindowHandle);

}

7.调用WindowState的attach方法，创建SurfaceSession对象。

void attach() {

mSession.windowAddedLocked();

}

SurfaceSession主要用于和surfaceflinger进行通信的Bp端，一个app只会创建一个该对象。

SurfaceComposerClient::SurfaceComposerClient()

: mStatus(NO\_INIT), mComposer(Composer::getInstance())

{

}

void SurfaceComposerClient::onFirstRef() {

sp<ISurfaceComposer> sm(**ComposerService::getComposerService()**);

if (sm != 0) {

**sp<ISurfaceComposerClient> conn = sm->createConnection();**

if (conn != 0) {

mClient = conn;

mStatus = NO\_ERROR;

}

}

}

创建ComposerService对象调用其connectLocked函数在该函数中通过ServiceManager来获取SurfaceFlinger服务的BP端。

/\*static\*/ sp<ISurfaceComposer> ComposerService::getComposerService() {

ComposerService& instance = ComposerService::getInstance();

Mutex::Autolock \_l(instance.mLock);

if (instance.mComposerService == NULL) {

**ComposerService::getInstance().connectLocked();**

assert(instance.mComposerService != NULL);

ALOGD("ComposerService reconnected");

}

return instance.mComposerService;

}

通过获取的BP端对象调用surfaceflinger服务的createConnection函数来创建ISurfaceComposerClient对象。

sp<ISurfaceComposerClient> SurfaceFlinger::createConnection()

{

sp<ISurfaceComposerClient> bclient;

sp<Client> client(new Client(this));

status\_t err = client->initCheck();

if (err == NO\_ERROR) {

bclient = client;

}

return bclient;

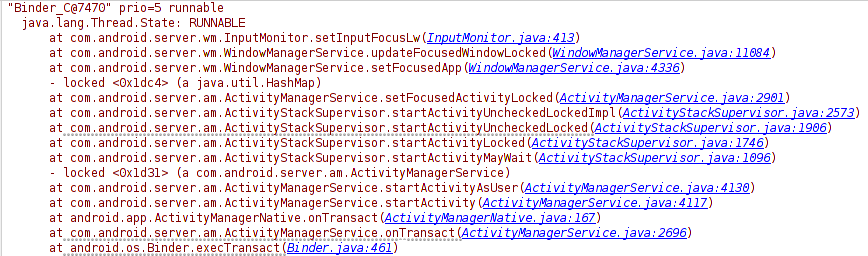
}

8.mWindowMap将WindowState加入到该容器中。

1. assignLayersLocked
2. mInputMonitor.updateInputWindowsLw
3. 几个重要的参数：

AppWindowToken：

WindowToken：



addAppToken：

对于activity 启动时候添加mTokenMap容器中的AppWindowToken。

addWindowToken：

对于系统窗口启动时候添加到mTokenMap容器中的WindowToken。

比如TYPE\_INPUT\_METHOD、TYPE\_WALLPAPER。