## View的加载过程

ActivityThread. performLaunchActivity

activity = mInstrumentation.newActivity(cl, component.getClassName(), r.intent);

Application app = r.packageInfo.makeApplication(false, mInstrumentation);

Context appContext = createBaseContextForActivity(r, activity);

activity.**attach**(appContext, this, getInstrumentation(), r.token,

r.ident, app, r.intent, r.activityInfo, title, r.parent,

r.embeddedID, r.lastNonConfigurationInstances, config,

r.voiceInteractor);

attachBaseContext(context);

mFragments.attachActivity(this, mContainer, null);

mWindow = PolicyManager.**makeNewWindow**(this);

mWindow.setCallback(this);

activity.setTheme(theme);

mInstrumentation.**callActivityOnCreate**(activity, r.state);

Activity. onCreate

setContentView(R.layout.incall\_screen);

getWindow().setContentView(layoutResID);

installDecor();

mDecor = generateDecor();

new DecorView(getContext(), -1);

mContentParent = generateLayout(mDecor);

View in = mLayoutInflater.inflate(layoutResource, null);

decor.addView(in, new ViewGroup.LayoutParams(MATCH\_PARENT, MATCH\_PARENT));

ViewGroup contentParent = (ViewGroup)findViewById(ID\_ANDROID\_CONTENT);

mLayoutInflater.inflate(layoutResID, mContentParent);

ActivityThread. handleResumeActivity

performResumeActivity

ActivityClientRecord r = mActivities.get(token);

r.activity.performResume();

mInstrumentation.callActivityOnResume(this);

wm.**addView**(decor, l);

mGlobal.addView(view, params, mDisplay, mParentWindow);

root = new ViewRootImpl(view.getContext(), display); // 保存在WindowManagerGlobal内

mWindowSession = WindowManagerGlobal.getWindowSession(); //WMS的本地功能代理

mWidth = -1;

mHeight = -1;

mDirty = new Rect();

mWinFrame = new Rect();

mWindow = new W(this); //用于接收WMS发过来的控制信息，并送到ViewRootHandler队列中处理

mAttachInfo = new View.AttachInfo(mWindowSession, mWindow, display, this, mHandler, this);

view.setLayoutParams(wparams);

mViews.add(view);

mRoots.add(root);

mParams.add(wparams);

root.**setView**(view, wparams, panelParentView);

ViewRootImpl. setView

mView = view;

mWindowAttributes.copyFrom(attrs);

**requestLayout**();

scheduleTraversals(); //启动view树的遍历

//将mWindow传给WMS，添加之后WMS对mWindowAttributes等属性参数进行赋值

res = **mWindowSession**.**addToDisplay**(**mWindow**, mSeq, **mWindowAttributes**,

getHostVisibility(), mDisplay.getDisplayId(),

mAttachInfo.mContentInsets, mInputChannel);

// WindowManagerService

mService.**addWindow**(this, window, seq, attrs, viewVisibility, displayId,

outContentInsets, outInputChannel);

token = new WindowToken(this, attrs.token, -1, false);

1、View添加到Windows流程，创建surface

2、windows参数设置，view参数设置

3、window token， app token， window id

4、WindowState

## View的invalidate刷新过程

1. 如何只刷新有变化的区域

2、刷新的遍历流程

**View的绘制，在addWindow的时候，就拿到了一个surface，以后更新画布那些都在这上面操作？**

## ImageView的大小计算

<RelativeLayout

android:layout\_width=*"match\_parent"*

android:layout\_height=*"match\_parent"*

<ImageView

android:id=*"@+id/view"*

android:layout\_width=*"match\_parent"*

android:layout\_height=*"wrap\_content"*

android:scaleType=*"centerCrop""* />

图片大小2400 1500

屏幕大小 1200 1920

实际大小 w1200 h1500

<ImageView

android:id=*"@+id/view"*

android:layout\_width=*"1500px"*

android:layout\_height=*"wrap\_content"*

android:background=*"#555555"*

android:scaleType=*"centerInside"*

实际大小 1200 1500

<ImageView

android:id=*"@+id/view"*

android:layout\_width=*"1500px"*

android:layout\_height=*"2000px"*

android:background=*"#555555"*

android:scaleType=*"centerInside"*

实际大小 1200 1664

这里非1500 2000是因为外面包裹的是一个RelativeLayout的同时，里面还有其他的View

改为FrameLayout就好了。

从源码分析，只要子View指定了大小，子View的测量规格都是

resultSize = childDimension;

resultMode = MeasureSpec.EXACTLY;

所以，顶多是显示不全而已。

如果把高变成wrap\_content。则View的大小为1500 1500。View的测量规格为：

resultSize = size; //整屏

resultMode = MeasureSpec.AT\_MOST;

这个时候，就需要看ImageView的onMeasure方法了。

**首先是decorView的测量规格：**

private static int getRootMeasureSpec(int windowSize, int rootDimension) {

int measureSpec;

switch (rootDimension) {

case **ViewGroup.LayoutParams.MATCH\_PARENT**:

// Window can't resize. Force root view to be windowSize.

measureSpec = MeasureSpec.makeMeasureSpec(**windowSize, MeasureSpec.EXACTLY**);

break;

case ViewGroup.LayoutParams.WRAP\_CONTENT:

// Window can resize. Set max size for root view.

measureSpec = MeasureSpec.makeMeasureSpec(windowSize, MeasureSpec.AT\_MOST);

break;

**然后是Content**

<com.android.internal.widget.**ActionBarOverlayLayout**

android:layout\_width="**match\_parent**"

android:layout\_height="**match\_parent**"

android:theme="?attr/actionBarTheme">

<**FrameLayout** android:id="@android:id/content"

android:layout\_width="**match\_parent**"

android:layout\_height="**match\_parent**" />

如果指定的都是**match\_parent，那么在measureChildWithMargins计算子类测量规格的时候：**

父View的测量规格是整屏大小和**EXACTLY，子View的是：**

public static int getChildMeasureSpec(int spec, int padding, int childDimension) {

switch (specMode) {

// Parent has imposed an exact size on us

case MeasureSpec.EXACTLY:

if (childDimension >= 0) { 手动指定的情况

resultSize = childDimension;

resultMode = MeasureSpec.EXACTLY;

} else if (childDimension == LayoutParams.**MATCH\_PARENT**) {

// Child wants to be our size. So be it.

resultSize = **size**; //传递父view sise

resultMode = MeasureSpec.**EXACTLY**;

} else if (childDimension == LayoutParams.WRAP\_CONTENT) {

// Child wants to determine its own size. It can't be

// bigger than us.

resultSize = size;

resultMode = MeasureSpec.AT\_MOST;

}

break;

getWidth() 是实际显示的宽度。

getMeasureWidth() 是测量宽度，在布局之前计算出来的。

getIntrinsicWidth() 是原有宽度，有时候原有宽度可能很大，但是实际上空间不够，所有效果上并没有那么大，这个方法可以获得原有宽度，可以辅助测量的时候选择合适的展示宽度。

getMinimumWidth() 是最小宽度，是XML参数定义里的 minWidth，也是一个辅助测量展示的参数。

Gradle

##The SDK platform-tools version (24.0.1) is too old to check APIs compiled with API 25; please update

升级sdk搞定

## shrinkResources true

> File 'D:\git\_workspace\cgeo-appcompat\main\build\intermediates\res\resources-basic-debug-stripped.ap\_' specified for property 'resourceFile' does not exist.

改为false