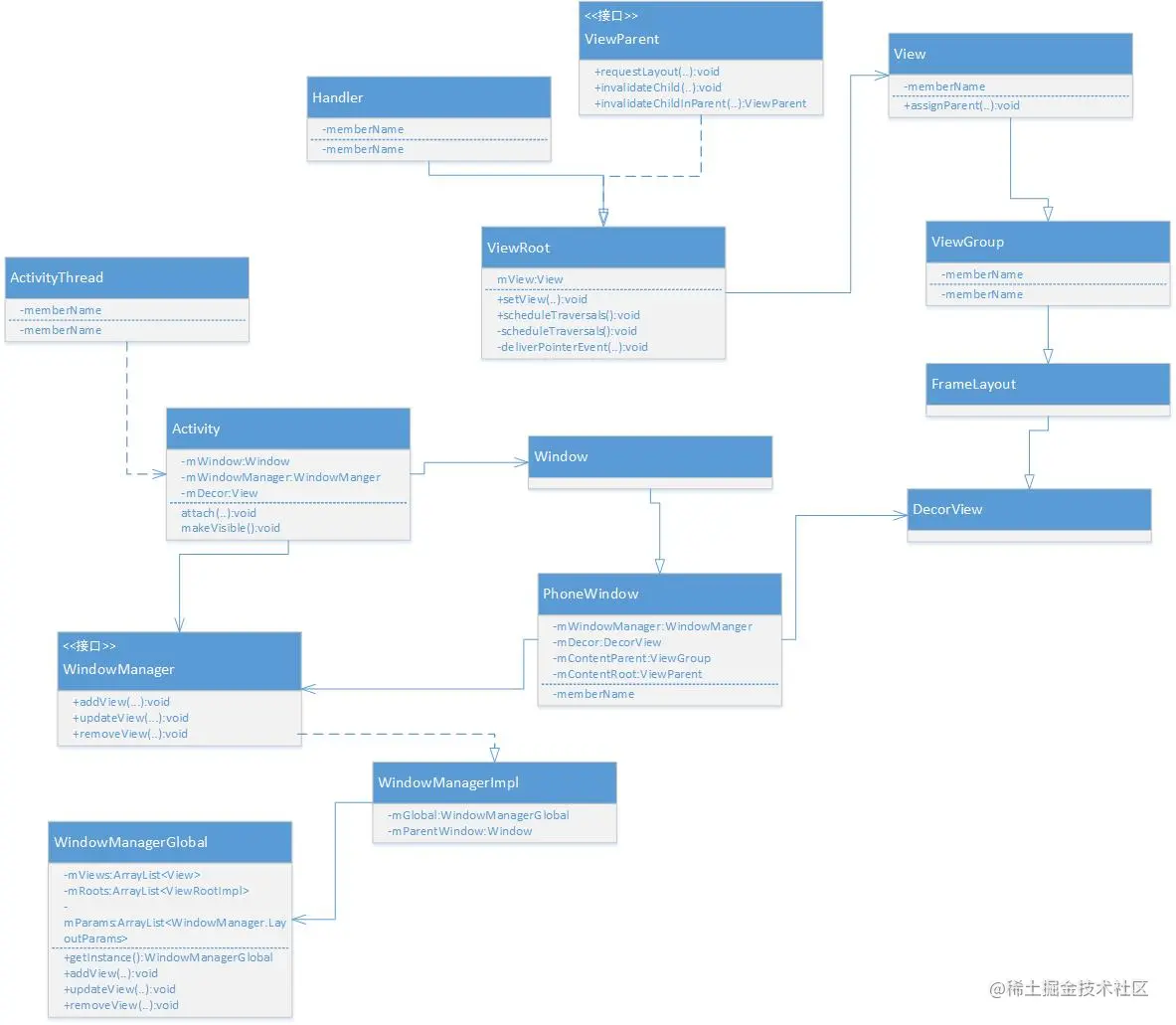
**Android 视图架构详解**

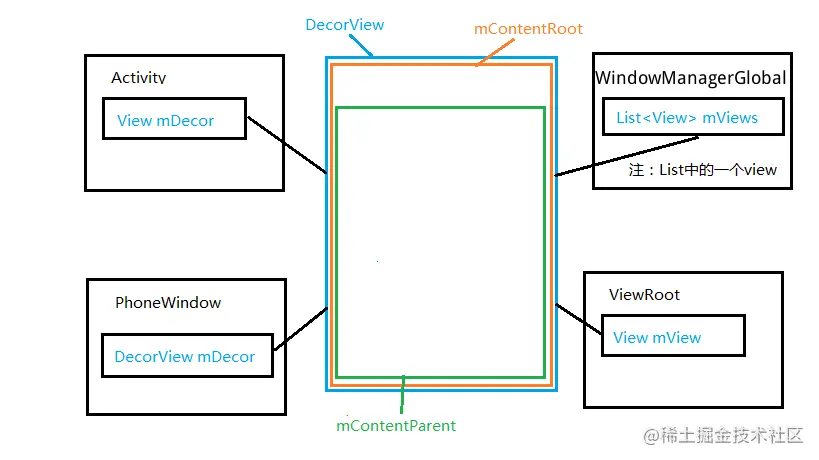
 最近一直在研究View的绘制相关的机制，发现需要补充一下Android View Architecture的相关知识，所以就特地研究了一下这方面的代码，写成本篇文章  
 为了节约你的时间，本篇文章内容大致如下:

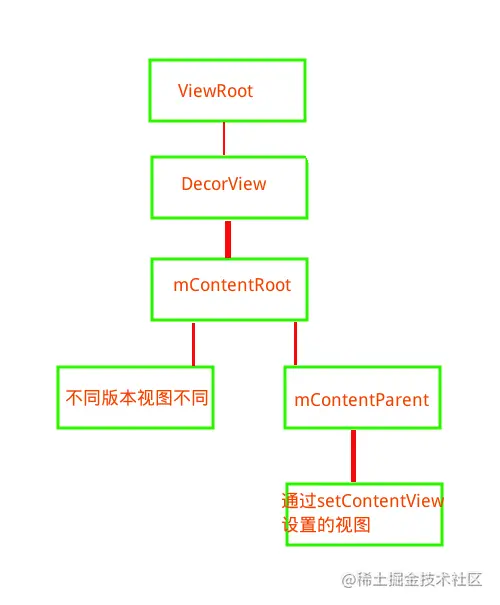
* Activity，DecorView，PhoneWindow和ViewRoot的作用和相关关系

**Android View Architecture**

 先来几张图，大致展现一下Android 视图架构的大概。







**Activity和Window**

 总所周知,Activity并不负责视图控制，它只是控制生命周期和处理事件，真正控制视图的是Window。一个Activity包含了一个Window，Window才是真正代表一个窗口，也就是说Activity可以没有Window，那就相当于是Service了。在ActivityThread中也有控制Service的相关函数或许正好印证了这一点。  
 Activity和Window的第一次邂逅是在ActivityThread调用Activity的attach()函数时。

//[window]:通过PolicyManager创建window,实现callback函数,所以,当window接收到

//外界状态改变时,会调用activity的方法,

final void attach(Context context, ActivityThread aThread,

Instrumentation instr, IBinder token, int ident,

Application application, Intent intent, ActivityInfo info,

CharSequence title, Activity parent, String id,

NonConfigurationInstances lastNonConfigurationInstances,

Configuration config, String referrer, IVoiceInteractor voiceInteractor) {

....

mWindow = PolicyManager.makeNewWindow(this);

//当window接收系统发送给它的IO输入事件时,例如键盘和触摸屏事件,就可以转发给相应的Activity

mWindow.setCallback(this);

.....

//设置本地窗口管理器

mWindow.setWindowManager(

(WindowManager)context.getSystemService(Context.WINDOW\_SERVICE),

mToken, mComponent.flattenToString(),

(info.flags & ActivityInfo.FLAG\_HARDWARE\_ACCELERATED) != 0);

.....

}

 在attach()中，新建一个Window实例作为自己的成员变量，它的类型为PhoneWindow,这是抽象类Window的一个子类。然后设置mWindow的WindowManager。

**Window,Activity和DecorView**

 DecorView是FrameLayout的子类，它可以被认为是Android视图树的根节点视图。DecorView作为顶级View，一般情况下它内部包含一个竖直方向的LinearLayout，在这个LinearLayout里面有上下两个部分（具体情况和Android版本及主体有关），上面的是标题栏，下面的是内容栏。在Activity中通过setContentView所设置的布局文件其实就是被加到内容栏之中的，而内容栏的id是content，在代码中可以通过ViewGroup content = （ViewGroup)findViewById(R.android.id.content)来得到content对应的layout。  
 Window中有几个视图相关的比较重要的成员变量如下所示:

* mDecor:DecorView的实例，标示Window内部的顶级视图
* mContentParent:setContetView所设置的布局文件就加到这个视图中
* mContentRoot:是DecorView的唯一子视图，内部包含mContentParent,标题栏和状态栏。

 Activity中不仅持有一个Window实例，还有一个类型为View的mDecor实例。这个实例和Window中的mDecor实例有什么关系呢？它又是什么时候被创建的呢？  
 二者其实指向同一个对象，这个对象是在Activity调用setContentView时创建的。我们都知道Activity的setContentView实际上是调用了Window的setContentView方法。

@Override

public void setContentView(int layoutResID) {

if (mContentParent == null) { //[window]如何没有DecorView,那么就新建一个

installDecor(); //[window]

} else if (!hasFeature(FEATURE\_CONTENT\_TRANSITIONS)) {

mContentParent.removeAllViews();

}

....

//[window]第二步,将layout添加到mContentParent

mLayoutInflater.inflate(layoutResID, mContentParent);

.....

}

 代码很清楚的显示了布局文件的视图是添加到mContentParent中，而且Window通过installDecor来新建DecorView。

//[window]创建一个decorView

private void installDecor() {

if (mDecor == null) {

mDecor = generateDecor(); //直接new出一个DecorView返回

....

}

if (mContentParent == null) {

//[window] 这一步也是很重要的.

mContentParent = generateLayout(mDecor); //mContentParent是setContentVIew的关键啊

.....

}

....

}

protected ViewGroup generateLayout(DecorView decor) {

// Apply data from current theme.

.......

//[window] 根据不同的style生成不同的decorview啊

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

// 加入到deco中,所以应该是其第一个child

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

mContentRoot = (ViewGroup) in; //给DecorView的第一个child是mContentView

// 这是获得所谓的content

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

}

.....

return contentParent;

}

 从上述的代码中，我们可以清楚的看到mDecor和mContentParent和mContentRoot的关系。  
 那么，Activity中的mDecor是何时被赋值的？我们如何确定它和Widnow中的mDecor指向同一个对象呢？我们可以查看ActivityThread的handleResumeActivity函数，它负责处理Activity的resume阶段。在这个函数中，Android直接将Window中的DecorView实例赋值给Activity。

final Activity a = r.activity;

r.window = r.activity.getWindow();

View decor = r.window.getDecorView();

decor.setVisibility(View.INVISIBLE);

ViewManager wm = a.getWindowManager();

WindowManager.LayoutParams l = r.window.getAttributes();

a.mDecor = decor;

**Window，DecorView 和 ViewRoot**

 ViewRoot对应ViewRootImpl类，它是连接WindowManagerService和DecorView的纽带，View的三大流程(测量（measure），布局（layout），绘制（draw）)均通过ViewRoot来完成。ViewRoot并不属于View树的一份子。从源码实现上来看，它既非View的子类，也非View的父类，但是，它实现了ViewParent接口，这让它可以作为View的名义上的父视图。RootView继承了Handler类，可以接收事件并分发，Android的所有触屏事件、按键事件、界面刷新等事件都是通过ViewRoot进行分发的。**ViewRoot可以被理解为“View树的管理者”——它有一个mView成员变量，它指向的对象和上文中Window和Activity的mDecor指向的对象是同一个对象**。

 我们来先看一下ViewRoot的创建过程。由于ViewRoot作为WindowMangerService和DecorView的纽带，只有在WindowManager将持有DecorView的Window添加进窗口管理器才创建。我们可以查看WindowMangerGlobal中的addView函数。对WindowManager不太熟悉的同学可以参考[《Window和WindowManager解析》](https://link.juejin.cn/?target=http%3A%2F%2Fztelur.github.io%2F2015%2F10%2F28%2FWindow%25E5%2592%258CWindowManager%25E8%25A7%25A3%25E6%259E%2590%2F)

public void addView(View view, ViewGroup.LayoutParams params,

Display display, Window parentWindow) {

// 创建ViewRootImpl,然后将下述对象添加到列表中

....

root = new ViewRootImpl(view.getContext(), display);

view.setLayoutParams(wparams);

mViews.add(view);

mRoots.add(root);

mParams.add(wparams);

....

try {

// 添加啦!!!!!!!!这是通过ViewRootImpl的setView来完成，这个View就是DecorView实例

root.setView(view, wparams, panelParentView);

} catch (RuntimeException e) {

....

}

....

}

 那么，Window是什么时候被添加到WindowManager中的呢？我们回到ActivityThread的handleResumeActivity函数。我们都知道Activity的resume阶段就是要显示到屏幕上的阶段，在Activity也就是DecorView将要显示到屏幕时，系统才会调用addView方法。  
 我们在handleResumeActivity函数中找到了下面一段代码,它调用了Activity的makeVisible()函数。

// ActivityThread

r.activity.makeVisible();

//Activity

//[windows] DecorView正式添加并显示

void makeVisible() {

if (!mWindowAdded) {

ViewManager wm = getWindowManager();

wm.addView(mDecor, getWindow().getAttributes());

mWindowAdded = true;

}

mDecor.setVisibility(View.VISIBLE);

}

 我们通过源代码发现，正式在makeVisible函数中，系统进行了Window的添加。

引用  
[wiki.jikexueyuan.com/project/dee…](https://link.juejin.cn/?target=http%3A%2F%2Fwiki.jikexueyuan.com%2Fproject%2Fdeep-android-v1%2Fsurface.html)  
[blog.csdn.net/guxiao1201/…](https://link.juejin.cn/?target=http%3A%2F%2Fblog.csdn.net%2Fguxiao1201%2Farticle%2Fdetails%2F41744107)  
[forlan.iteye.com/blog/226938…](https://link.juejin.cn/?target=http%3A%2F%2Fforlan.iteye.com%2Fblog%2F2269381)

目录

* [Android View Architecture](https://juejin.cn/post/6844903430381633544#heading-0)
* [Activity和Window](https://juejin.cn/post/6844903430381633544#heading-1)
* [Window,Activity和DecorView](https://juejin.cn/post/6844903430381633544#heading-2)
* [Window，DecorView 和 ViewRoot](https://juejin.cn/post/6844903430381633544#heading-3)