View 绘制原理

1. Window，Activity 和 DecorView 的关系

Window 是一个抽象类。

在 Activity 的 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 给 WindowManager。

```
@UnsupportedAppUsage(maxTargetSdk = Build.VERSION_CODES.R, trackingBug = 170729553)
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,
        Window window, ActivityConfigCallback activityConfigCallback, IBinder assistToken,
        IBinder shareableActivityToken) {
    attachBaseContext(context);

    mFragments.attachHost( parent: null /*parent*/);

    mWindow = new PhoneWindow(this, window, activityConfigCallback);
    mWindow.setWindowControllerCallback(mWindowControllerCallback);
    mWindow.setCallback(this);
    mWindow.setOnWindowDismissedCallback(this);
    mWindow.getLayoutInflater().setPrivateFactory(this);
```

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) findViewByld (R.android.id.content) 来得到 content 对应的 layout。

PhoneWindowWindow 中有几个视图相关的比较重要的成员变量如下所示:
1）mDecor：DecorView 的实例，标示 Window 内部的顶级视图
2）mContentParent：它是个 View，setContetView 所设置的布局文件就加到这个视图中
3）mContentRoot：它是个 ViewGroup，是 DecorView 的唯一子视图，内部包含 mContentParent,标题栏和状态栏

```
// This is the top-level view of the window, containing the window decor.
private DecorView mDecor;

// When we reuse decor views, we need to recreate the content root. This happens when the decor
// view is requested, so we need to force the recreating without introducing an infinite loop.
private boolean mForceDecorInstall = false;

// This is the view in which the window contents are placed. It is either
// mDecor itself, or a child of mDecor where the contents go.
ViewGroup mContentParent;
// Whether the client has explicitly set the content view. If false and mContentParent is not
// null, then the content parent was set due to window preservation.
private boolean mContentParentExplicitlySet = false;
```

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？

```
@UnsupportedAppUsage
private Window mWindow;

@UnsupportedAppUsage
private WindowManager mWindowManager;
/*package*/ View mDecor = null;
@UnsupportedAppUsage
/*package*/ boolean mWindowAdded = false;
/*package*/ boolean mVisibleFromServer = false;
@UnsupportedAppUsage(maxTargetSdk = Build.VERSION_CODES.P, trackingBug = 115609023)
```

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 PhoneWindow 的 setContentView()方法。

在 PhoneWindow 的 setContentView()方法中，调用了 installDecor()方法创建了一个 DecorView。然后通过 generateLayout()方法创建了 mContentParent 和 mContentRoot，mContentRoot 就是整个 DecorView 的布局对象，而 mContentParent 是的 R.id.content 的内容部分。

```java
@Override
public void setContentView(int layoutResID) {
    // Note: FEATURE_CONTENT_TRANSITIONS may be set in the process of installing the window
    // decor, when theme attributes and the like are crystalized. Do not check the feature
    // before this happens.
    if (mContentParent == null) {
        installDecor();
    } else if (!hasFeature(FEATURE_CONTENT_TRANSITIONS)) {
        mContentParent.removeAllViews();
    }

    if (hasFeature(FEATURE_CONTENT_TRANSITIONS)) {
        final Scene newScene = Scene.getSceneForLayout(mContentParent, layoutResID,
                getContext());
        transitionTo(newScene);
    } else {
        mLayoutInflater.inflate(layoutResID, mContentParent);
    }
    mContentParent.requestApplyInsets();
    final Callback cb = getCallback();
    if (cb != null && !isDestroyed()) {
        cb.onContentChanged();
    }
    mContentParentExplicitlySet = true;
}

private void installDecor() {
    mForceDecorInstall = false;
    if (mDecor == null) {
        mDecor = generateDecor( featureId: -1);
        mDecor.setDescendantFocusability(ViewGroup.FOCUS_AFTER_DESCENDANTS);
        mDecor.setIsRootNamespace(true);
        if (!mInvalidatePanelMenuPosted && mInvalidatePanelMenuFeatures != 0) {
            mDecor.postOnAnimation(mInvalidatePanelMenuRunnable);
        }
    } else {
        mDecor.setWindow(this);
    }
    if (mContentParent == null) {
        mContentParent = generateLayout(mDecor);
```

```
mDecor.startChanging();
mDecor.onResourcesLoaded(mLayoutInflater, layoutResource);

ViewGroup contentParent = (ViewGroup)findViewById(ID_ANDROID_CONTENT);
if (contentParent == null) {
    throw new RuntimeException("Window couldn't find content container view");
}
```

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。这个 a 就是 Activity 对象。

```
if (r.window == null && !a.mFinished && willBeVisible) {
    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;
    l.type = WindowManager.LayoutParams.TYPE_BASE_APPLICATION;
    l.softInputMode |= forwardBit;
```

2. Window，DecorView 和 ViewRoot 的关系

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 ActivityThread 的 handleResumeActivity 函数中找到了下面一段代码,它调用了 Activity 的 makeVisible()函数。

```
    r.activity.mVisibleFromServer = true;
    mNumVisibleActivities++;
    if (r.activity.mVisibleFromClient) {
        r.activity.makeVisible();
    }
}
```

最终通过 WindowManager 的 addView 函数添加了 DecorView 并显示出来。

```
public void setVisible(boolean visible) {
    if (mVisibleFromClient != visible) {
        mVisibleFromClient = visible;
        if (mVisibleFromServer) {
            if (visible) makeVisible();
            else mDecor.setVisibility(View.INVISIBLE);
        }
    }
}

void makeVisible() {
    if (!mWindowAdded) {
        ViewManager wm = getWindowManager();
        wm.addView(mDecor, getWindow().getAttributes());
        mWindowAdded = true;
    }
    mDecor.setVisibility(View.VISIBLE);
}
```

3. Activity、Window 和 WindowManager 的关系

1）一个 Activity 持有一个 PhoneWindow 的对象，而一个 PhoneWindow 对象持有一个 DecorView 的实例

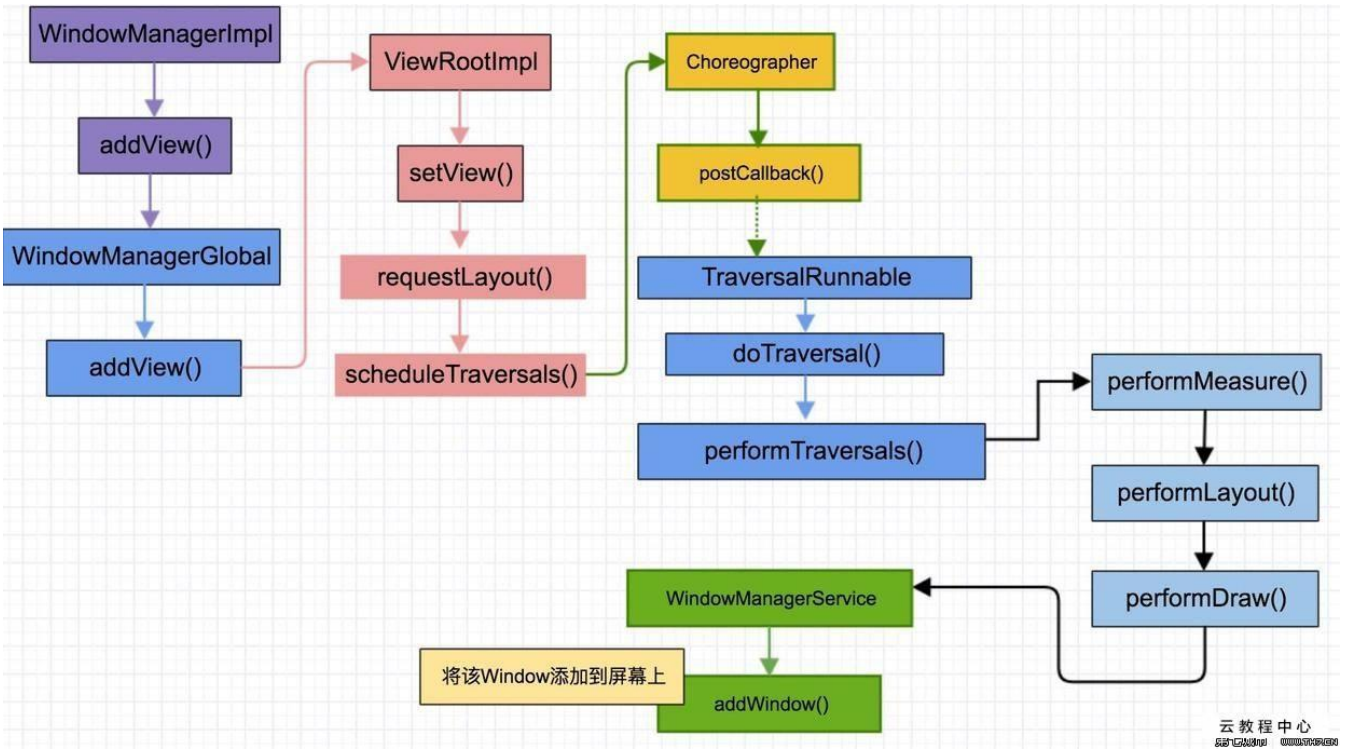
2）PhoneWindow 继承自 Window，一个 Window 对象内部持有 mWindowManager 的实例，通过调用 setWindowManager 方法与 WindowManager 关联在一起

3）WindowManager 继承自 ViewManager，WindowManagerImpl 是 WindowManager 接口的实现类，但是具体的功能都会委托给 WindowManagerGlobal 来实现

4）调用 WindowManager 的 addView 方法，实际上调用的是 WindowManagerImpl 的 addView 方法

4. Window 如何与 WindowManager 关联?

在 Activity 的 attach 方法中，调用 PhoneWindow 的 setWindowManager 方法，这个方法的具体实现发生在 Window 中，最终调用的是 Window 的 setWindowManager 方法，将 Window 和 WindowManager 关联在了一起。


WindowManagerImpl
addView()
WindowManagerGlobal
addView()
ViewRootImpl
setView()
requestLayout()
scheduleTraversals()
Choreographer
postCallback()
TraversalRunnable
doTraversal()
performTraversals()
performMeasure()
performLayout()
performDraw()
WindowManagerService
addWindow()
将该Window添加到屏幕上