Android应用视图的载体View-概述

〇、概述

View 是屏幕上的一块矩形区域，负责界面的绘制和触摸事件的处理，它是一中界面层的抽象，所有的控件都继承自View

View 是Android中显示框架中较为复杂的一环，首先是它的生命周期会随着Activity的生命周期进行变化，掌握View的生命周期对我们自定义View有着重要的意义。另一方面View从ViewRootImpl.performTraversals()开始，经历measure、layout、draw三个流程最终显示在用户面前，用户在点击屏幕时，点击事件随着Activity传入Window，最终由ViewGroup/View进行分发处理。

注：

ViewRootImpl.performTraversals()方法的调用起点是DecorView.requestLayout()，因为是DecorView是view的最终ParentView.

在ViewGroup调用addViewInLayout(View child, int index, LayoutParams params)这个方法是设定了ParentView是ViewGroup自身，而ParentView递归到最后是DecorView

DecorView中有一个子View叫mContentRoot，默认条件下是由R.layout.screen_simple这个资源填充而成，而这个资源文件文件内有一个“@android:id/content”对应的FrameLayout，而这个FrameLayout则是mContentParent。

整体View处理流程：

1. Activity创建及启动 在ActivityThread启动Activity时（ActivityThread.performLunchActivity()方法），调用Activity的attach方法。在Activity的attach方法中创建Window和Window的DecorView，并同时给Window设置WindowManager。而这个WindowManager方法则创建的是一个WindowManagerImpl，是WindowManager的具体实现。在Activity创建及启动的过程中实现了这个多东西。

```
ActivityThread.java

package android.app;
...
private Activity performLaunchActivity(ActivityClientRecord r,
Intent customIntent) {
    ...
    Activity activity = null;
    try {
        java.lang.ClassLoader cl = r.packageInfo.getClassLoader();
        activity = mInstrumentation.newActivity(
                cl, component.getClassName(), r.intent);
        ...
    } catch (Exception e) {
        ...
    }

    try {
        Application app = r.packageInfo.makeApplication(false,
mInstrumentation);
        ...
        if (activity != null) {
            Context appContext = createBaseContextForActivity(r,
activity);
            CharSequence title =
r.activityInfo.loadLabel(appContext.getPackageManager());
            ...
            //向Activity附加东西
            activity.attach(appContext, this, getInstrumentation(),
r.token,
                    r.ident, app, r.intent, r.activityInfo, title,
r.parent,
                    r.embeddedID, r.lastNonConfigurationInstances,
config,
                    r.referrer, r.voiceInteractor);

            //表示是否是启动过
            activity.mStartedActivity = false;
            ...
        }
    }
    ...
}
...
```

```
Activity.java
```

```java
package android.app;
...
public class Activity extends ContextThemeWrapper
        implements LayoutInflater.Factory2,
        Window.Callback, KeyEvent.Callback,
        OnCreateContextMenuListener, ComponentCallbacks2,
        Window.OnWindowDismissedCallback {
    ...
    //字面意思，从是否从客户端显示，个人理解是是否是显示
    /*package*/ boolean mVisibleFromClient = true;
    ...
    final void attach(....){
        ...
        //创建Activity中的Window
        mWindow = new PhoneWindow(this);
        ...
        //设置Window管理器
        mWindow.setWindowManager(

(WindowManager)context.getSystemService(Context.WINDOW_SERVICE),
            mToken, mComponent.flattenToString(),
            (info.flags & ActivityInfo.FLAG_HARDWARE_ACCELERATED)
!= 0);
        ...
        mWindowManager = mWindow.getWindowManager();
        ...
    }

    /**
     * 控制Activity的主Window是否显示。
     * Control whether this activity's main window is visible.
This is intended
     * only for the special case of an activity that is not going
to show a
     * UI itself, but can't just finish prior to onResume() because
it needs
     * to wait for a service binding or such.  Setting this to
false allows
     * you to prevent your UI from being shown during that time.
     *
     * <p>The default value for this is taken from the
     * {@link android.R.attr#windowNoDisplay} attribute of the
activity's theme.
     */
    public void setVisible(boolean visible) {
        if (mVisibleFromClient != visible) {
            mVisibleFromClient = visible;
            if (mVisibleFromServer) {
                if (visible) makeVisible();
                else mDecor.setVisibility(View.INVISIBLE);
            }
        }
    }
    ...
}
...
```

```
Window.java

package android.view;
...
public void setWindowManager(WindowManager wm, IBinder appToken,
String appName,
            boolean hardwareAccelerated) {
        ...
        if (wm == null) {
            wm =
(WindowManager)mContext.getSystemService(Context.WINDOW_SERVICE);
        }
        mWindowManager =
((WindowManagerImpl)wm).createLocalWindowManager(this);
        ...
    }
...
```

```
WindowManagerImpl.java

package android.view;

public final class WindowManagerImpl implements WindowManager {

    private final WindowManagerGlobal mGlobal =
WindowManagerGlobal.getInstance();
...
    public WindowManagerImpl createLocalWindowManager(Window
parentWindow) {
        return new WindowManagerImpl(mDisplay, parentWindow);
    }
...
}
```

2. 在ActivityResume展示的时候，调用handleResumeActivity()方法，并且在Activity的main Window 需要显示的时候将view加入Window，然后调用WindowManagerImpl中的addView方法，最终调用WindowManagerGlobal的addView方法

```
ActivityThread.java

package android.app;

public final class ActivityThread {
    ...

    final void handleResumeActivity(IBinder token,
            boolean clearHide, boolean isForward, boolean
reallyResume) {

        if (r != null) {
            final Activity a = r.activity;

            ...

            if (r.window == null && !a.mFinished && willBeVisible)
{
                r.window = r.activity.getWindow();
                View decor = r.window.getDecorView();
                decor.setVisibility(View.INVISIBLE);
                //WindowManger继承ViewManager
                ViewManager wm = a.getWindowManager();
                //布局方式
                WindowManager.LayoutParams l =
r.window.getAttributes();
                a.mDecor = decor;
                l.type =
WindowManager.LayoutParams.TYPE_BASE_APPLICATION;
                l.softInputMode |= forwardBit;
                // a.mViwsiableFromClient 默认为true，在上方代码中有
                if (a.mVisibleFromClient) {
                    a.mWindowAdded = true;
                    // 其中vm在上一步骤中提到其实是WindowManagerImpl
                    wm.addView(decor, l);
                }
    }
    ...
}
```

```java
Activity.java


package android.app;
...
public class Activity extends ContextThemeWrapper
        implements LayoutInflater.Factory2,
        Window.Callback, KeyEvent.Callback,
        OnCreateContextMenuListener, ComponentCallbacks2,
        Window.OnWindowDismissedCallback {
    ...
    //字面意思，从是否从客户端显示，个人理解是是否是显示
    /*package*/ boolean mVisibleFromClient = true;

    /** Retrieve the window manager for showing custom windows. */
    public WindowManager getWindowManager() {
        return mWindowManager;
    }

}
```

```java
WindowManagerImpl.java


package android.view;
...
public final class WindowManagerImpl implements WindowManager {
    //单例模式获取WindowManagerGlobal的唯一值
    private final WindowManagerGlobal mGlobal =
WindowManagerGlobal.getInstance();
    ...

    @Override
    public void addView(@NonNull View view, @NonNull
ViewGroup.LayoutParams params) {
        applyDefaultToken(params);
        mGlobal.addView(view, params, mDisplay, mParentWindow);
    }
    ...
}
```

```java
WindowManagerGlobal.java
```

```java
package android.view;
...
public final class WindowManagerGlobal {
    ...
    private static WindowManagerGlobal sDefaultWindowManager;
    ...

    private WindowManagerGlobal() {
    }
    ...

    public static WindowManagerGlobal getInstance() {
        synchronized (WindowManagerGlobal.class) {
            if (sDefaultWindowManager == null) {
                sDefaultWindowManager = new WindowManagerGlobal();
            }
            return sDefaultWindowManager;
        }
    }

    ...

    public void addView(View view, ViewGroup.LayoutParams params,
            Display display, Window parentWindow) {
        ...
        ViewRootImpl root;
        View panelParentView = null;

        synchronized (mLock) {
            int index = findViewLocked(view, false);
            if (index >= 0) {
                if (mDyingViews.contains(view)) {
                    // Don't wait for MSG_DIE to make it's way
through root's queue.
                    mRoots.get(index).doDie();
                } else {
                    throw new IllegalStateException("View " + view
                            + " has already been added to the
window manager.");
                }
                    // The previous removeView() had not completed
executing. Now it has.
                }

                // If this is a panel window, then find the window
it is being
                // attached to for future reference.
                if (wparams.type >=
WindowManager.LayoutParams.FIRST_SUB_WINDOW &&
                        wparams.type <=
WindowManager.LayoutParams.LAST_SUB_WINDOW) {
                    final int count = mViews.size();
                    for (int i = 0; i < count; i++) {
                        if (mRoots.get(i).mWindow.asBinder() ==
wparams.token) {
                            panelParentView = mViews.get(i);
                        }
                    }
                }
                //创建根节点
                root = new ViewRootImpl(view.getContext(),
display);
                //设置布局参数
                view.setLayoutParams(wparams);
                //将View加入集合中
                mViews.add(view);
                //将根节点加入集合中
                mRoots.add(root);
                //将View的布局参数加入集合中
                mParams.add(wparams);
        }

        // do this last because it fires off messages to start
doing things
        try {
            //将根节点的View设置为DecorView
            root.setView(view, wparams, panelParentView);
        } catch (RuntimeException e) {
            throw e;
        }
    }
    ...
}
```

```
ViewRootImpl.java

package android.view;
public final class ViewRootImpl implements ViewParent,
        View.AttachInfo.Callbacks,
HardwareRenderer.HardwareDrawCallbacks {

    ...
    public void setView(View view, WindowManager.LayoutParams
attrs, View panelParentView) {
        ...
        // Schedule the first layout -before- adding to the window
        // manager, to make sure we do the relayout before
receiving
        // any other events from the system.
        requestLayout();
        ...
        //将view的父设为自己，也就是将DecorView的父设为自己
        view.assignParent(this);
        ...
    }
    ...
}
```

3. 经过上面的代码分析，ViewRootImpl和DecorView已经有了父子关系。那么最终怎么调用了ViewRootImpl的performTraversals()的方法呢?

这个是Activity.setContentView()方法，调用getWindow().setContentView()方法就是PhoneWindow.setContentView()方法，然后初始化DecorView和内容父View(ContentParent)，然后调用ContentParent.requestApplyInsets()方法，再调用requestFitSystemWindows，直到调用最终父View的requestFitSystemWindows，而最终的父View是ViewRootImpl。然后调用scheduleTraversals，剩余参考代码

```
Activity.java

package android.app;
public class Activity extends ContextThemeWrapper
        implements LayoutInflater.Factory2,
        Window.Callback, KeyEvent.Callback,
        OnCreateContextMenuListener, ComponentCallbacks2,
        Window.OnWindowDismissedCallback, WindowControllerCallback
{
    /**
     * Set the activity content from a layout resource.  The
resource will be
     * inflated, adding all top-level views to the activity.
     *
     * @param layoutResID Resource ID to be inflated.
     *
     * @see #setContentView(android.view.View)
     * @see #setContentView(android.view.View,
android.view.ViewGroup.LayoutParams)
     */
    public void setContentView(@LayoutRes int layoutResID) {
        getWindow().setContentView(layoutResID);
        initWindowDecorActionBar();
    }

    /**
     * Retrieve the current {@link android.view.Window} for the
activity.
     * This can be used to directly access parts of the Window API
that
     * are not available through Activity/Screen.
     *
     * @return Window The current window, or null if the activity
is not
     *         visual.
     */
    public Window getWindow() {
        return mWindow;
    }
}
```

```
View.java

package android.view;
public class View implements Drawable.Callback, KeyEvent.Callback,
        AccessibilityEventSource {
    ...
    /*
     * Caller is responsible for calling requestLayout if
necessary.
     * (This allows addViewInLayout to not request a new layout.)
     */
    void assignParent(ViewParent parent) {
        if (mParent == null) {
            mParent = parent;
        } else if (parent == null) {
            mParent = null;
        } else {
            throw new RuntimeException("view " + this + " being
added, but"
                    + " it already has a parent");
        }
    }
    ...

    /**
```

```
     * Ask that a new dispatch of {@link #fitSystemWindows(Rect)}
be performed.
     * @deprecated Use {@link #requestApplyInsets()} for newer
platform versions.
     */
    public void requestFitSystemWindows() {
        if (mParent != null) {
            mParent.requestFitSystemWindows();
        }
    }

    /**
     * Ask that a new dispatch of {@link
#onApplyWindowInsets(WindowInsets)} be performed.
     */
    public void requestApplyInsets() {
        requestFitSystemWindows();
    }

    ...
```

```
ViewRootImpl.java

package android.view;
public final class ViewRootImpl implements ViewParent,
        View.AttachInfo.Callbacks,
HardwareRenderer.HardwareDrawCallbacks {
    ...
    @Override
    public void requestFitSystemWindows() {
        checkThread();
        mApplyInsetsRequested = true;
        scheduleTraversals();
    }
    ...

    void scheduleTraversals() {
        if (!mTraversalScheduled) {
            mTraversalScheduled = true;
            mTraversalBarrier =
mHandler.getLooper().getQueue().postSyncBarrier();
            //加入子线程,最终执行mTraversalRunnable的run方法
            mChoreographer.postCallback(
                    Choreographer.CALLBACK_TRAVERSAL,
mTraversalRunnable, null);
```

```
            if (!mUnbufferedInputDispatch) {
                scheduleConsumeBatchedInput();
            }
            notifyRendererOfFramePending();
            pokeDrawLockIfNeeded();
        }
    }

    final class TraversalRunnable implements Runnable {
        @Override
        public void run() {
            doTraversal();
        }
    }
    final TraversalRunnable mTraversalRunnable = new
TraversalRunnable();

    void doTraversal() {
        if (mTraversalScheduled) {
            mTraversalScheduled = false;

mHandler.getLooper().getQueue().removeSyncBarrier(mTraversalBarrier
);

            if (mProfile) {
                Debug.startMethodTracing("ViewAncestor");
            }
            //目标方法
            performTraversals();

            if (mProfile) {
                Debug.stopMethodTracing();
                mProfile = false;
            }
        }
    }

    private void performTraversals() {
        ...

    }

}
```