View 的绘制流程是 measure -> layout -> draw。

不过由此引申的知识点还是有不少的：

1、首次 View 的绘制流程是在什么时候触发的？

在 ActivityThread.handleResumeActivity 里触发的。

ActivityThread.handleResumeActivity 里会调用 wm.addView 来添加 DecorView，wm 是 WindowManagerImpl。

最终通过 WindowManagerImpl.addView -> WindowManagerGlobal.addView -> ViewRootImpl.setView -> ViewRootImpl.requestLayout 就触发了第一次 View 的绘制。

1. ViewRootImpl 创建的时机？

ViewRootImpl 也是在 ActivityThread.handleResumeActivity 里创建的。具体在WindowManagerGlobal.addView方法中创建的。

1. ViewRootImpl 和 DecorView 的关系是什么？

在 ViewRootImpl.setView 里，通过 DecorView.assignParent 把 ViewRootImpl 设置为 DecorView 的 parent。

所以 ViewRootImpl 和 DecorView 的关系就是 ViewRootImpl 是 DecorView 的 parent。

因为 DecorView 是我们布局的顶层，现在我们就知道层层调用 requestLayout 等方法是怎么调用到 ViewRootImpl 里的了。

1. DecorView 的布局是什么样的？

对于 Activity 的层级，大家应该都看过一张图的描述，Activity -> PhoneWindow -> DecorView -> [title\_bar, content]，其中 DecorView 里包括了 title\_bar 和 content 两个 View，不过这个是默认的布局，实际上根据不同的主题样式，DecorView 对应有不同的布局。

图中所包含的 title\_bar 和 content 对应的是 R.layout.screen\_simple 布局。

具体的布局是在 PhoneWindow.installDecor -> generateLayout 中设置的。

1. DecorView 的创建时机？

调用链是 Activity.setContentView -> PhoneWindow.setContentView -> installDecor。

1. setContentView 的流程

setContentView 流程比较简单，会调用 PhoneWindow.setContentView。

其中做的事情是两个：

1.创建 DecorView

2.根据 layoutResId 创建 View 并添加到 DecorView 中

1. LayoutInflate 的流程

LayoutInflate.inflate 最终是调用 createViewFromTag 从 xml 生成 View 的，其实这里才是关键。

mFactory、mFactory2、mPrivateFactory 分别对应 Factory 和 Factory2 方法，对应的是两个 onCreateView 方法，Factory.onCreateView 没有传入 parent 参数，Factory2.onCreateView 传入了 parent 参数。

而 mFactory 和 mFactory2 我们是可以设置的，当然不能重复设置，重复设置会抛出异常。

如果已经有 mFactory 的值，则生成一个 FactoryMerger，这个也是继承了 Factory2，用来控制一下调用顺序。

然后我们再看 mPrivateFactory，看名称就知道是系统的隐藏方法。

调用时机是在 Activity.attach 中，Activity 其实是实现了 Factory2 的 onCreateView 方法，其中对 fragment 做了处理，如果是 fragment 标签，就调用 fragment 的 onCreateView，这里就不详细往下面看了，如果是非 fragment 的标签，就返回 null，走默认的创建 View 的方法。

所以上面的 Factory 和 Factory2，是系统留给我们的 hook View 创建流程的接口。

如果都没有设置，那就走到默认的创建 View 的方法。

默认创建 View 的方法比较简单，就是反射调用 View 的构造函数，然后做一个缓存，然后创建 View。

1. Activity、PhoneWindow、DecorView、ViewRootImpl 的关系？

PhoneWindow 其实是 Window 的唯一子类，是 Activity 和 View 交互系统的中间层，而 DecorView 是整个 View 层级的最顶层，ViewRootImpl 是 DecorView 的 parent，但是他并不是一个真正的 View，只是继承了 ViewParent 接口，用来掌管 View 的各种事件，包括 requestLayout、invalidate、dispatchInputEvent 等等。

1. PhoneWindow 的创建时机？

是在 Activity.attach 里创建的，而 Activity.attach 又是在 ActivityThread.performLaunchActivity 里创建的。

1. 如何触发重新绘制？

就是调用 requestLayout 和 invalidate。

1. requestLayout 和 invalidate 的流程

requestLayout 会一层层调用 parent 的 requestLayout，而上面的问题中我们也分析到了，DecorView 是整个 View 层级的最顶层，ViewRootImpl 又是 DecorView 的 parent，所以最终调用到 ViewRootImpl 的 requestLayout。

ViewRootImpl.requestLayout 调用 scheduleTraversals -> doTraversal -> performTraversals 开启绘制流程。

在 performTraversals 里，就是熟悉的 performMeasure -> performLayout -> performDraw 三个流程了。

先看 performMeasure，最终调用的是 View.measure。设置forceLayout为true，调用onMeasure方法。设置 LAYOUT\_REQUIRED flag，在 layout 中会用到。

再看 performLayout，performLayout 里一共执行了三件事：

1.执行 View.layout

2.执行调用过 requestLayout 的 View 的 measure 和 layout

3.将还没有执行的 requestLayout 加到队列中，下一次 frame 中进行执行

然后看 View.layout 的流程，最后就是 ViewRootImpl.performDraw -> draw 了。在 View 的绘制过程中，我们可以看到，只有 flag 被设置为 PFLAG\_DIRTY\_OPAQUE 才会进行绘制（这里划重点）。

这也就是大家经常说的 requestLayout 不会引发 draw。

invalidate -> invalidateInternal 的主要流程就是在设置 mPrivateFlags，invalidate 会调用 parent.invalidateChild。上面的 while 循环里，会层层计算 parent 的 dirty 区域，最终会调用到 ViewRootImpl.invalidateChildInParent -> invalidateRectOnScreen。最终调用 scheduleTraversals 去触发整个绘制流程，然后调用到 View.draw 方法，根据 PFLAG\_DIRTY\_OPAQUE flag 去决定是否重新绘制。

12、requestLayout 和 invalidate 的区别

requestLayout 和 invalidate 都会触发整个绘制流程。但是在 measure 和 layout 过程中，只会对 flag 设置为 FORCE\_LAYOUT 的情况进行重新测量和布局，而 draw 只会重绘 flag 为 dirty 的区域。

requestLayout 是用来设置 FORCE\_LAYOUT 标志，invalidate 用来设置 dirty 标志。所以 requestLayout 只会触发 measure 和 layout，invalidate 只会触发 draw。