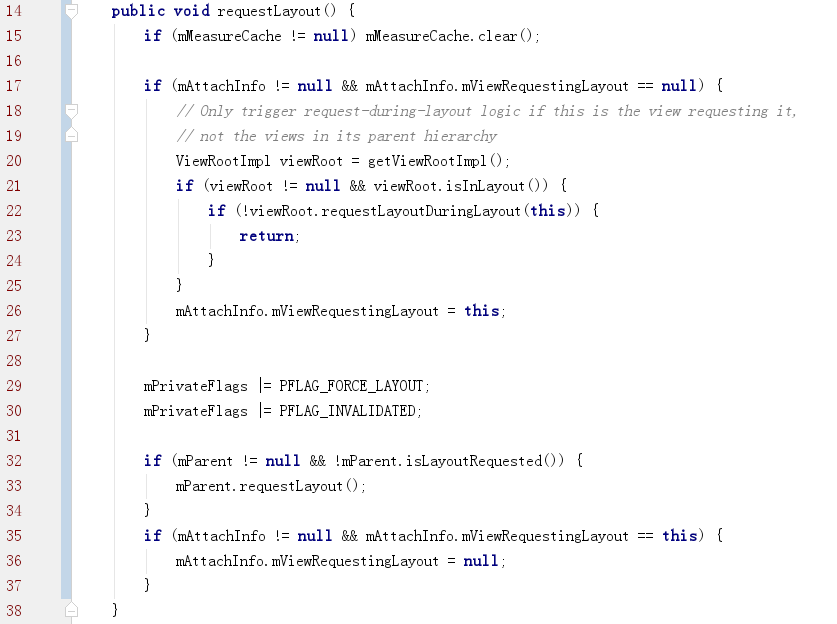
View解读

* **方法解读**
* **requestLayout()**

首先某个页面内的一个子View调用requestLayout()方法，它会使当前页面所有的View（即View树）进行测量、布局流程，至于绘制流程的执行：只有View树里的大小或位置发生变化的子View才会执行绘制流程。

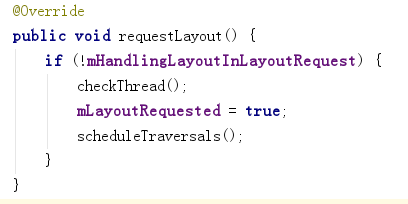


看32行判断，如果当前View在请求布局的时候，View树正在进行布局流程的话，该请求会延迟到布局流程完成后或者绘制流程完成且下一次布局发现的时候再执行。33行执行mParent的requestLayout()方法。

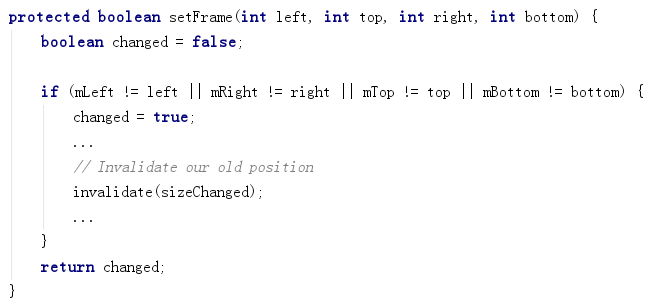
说明：mParent是ViewParent接口类型，在这里是当前View的父布局即ViewGroup类型的容器类，ViewGroup实现了ViewParent接口。当ViewGroup类型容器添加子View的时候，mParent会被赋值指向这个ViewGroup。

所以这里最终调用的是父布局的requestLayout()方法。而ViewGroup没有重写requestLayout()方法，所以ViewGroup又会调用它的父类View的requestLayout()方法。通过这种方式一层一层的向上调用，直到DecorView，即根View，而根View又会传递给ViewRootImpl，也即是说子View的requestLayout事件，最终会被ViewRootImpl接收并得到处理。

那么直接看ViewRootImpl的requestLayout()方法：



在这里，调用了scheduleTraversals方法，这个方法是一个异步方法，最终会调用到ViewRootImpl的performTraversals方法，这也是View工作流程的核心方法，在这个方法内部，分别调用perforMeasure、perforLayout、perforDraw方法来进行View的三大工作流程,这三个方法也正是一个页面绘制的起点。这里注意调用requestLayout()方法并不一定会引起页面的onDraw的调用，这是因为在performDraw内部draw的过程中发现mDirty为空，所以onDraw不会被调用，不重绘。 但是在perforLayout方法调用中如果在layout过程中发现l,t,r,b和以前不一样，那就会触发一次invalidate，具体会在setFrame方法中判断：



所以requestLayout有可能会导致onDraw被调用，也可能不导致onDraw被调用，取决于view的l,t,r,b是否改变。

* **invalidate()**

参考;https://github.com/Idtk/Blog/blob/master/Blog/9%E3%80%81Invalidate.md

它是View的一个方法不能被子类重写，调用这个方法会使当前View被重绘刷新布局（不包含测量和布局过程）：

说明：View中有invalidate重载的方法：

invalidate(Rect dirty)；

invalidate(int l, int t, int r, int b)；

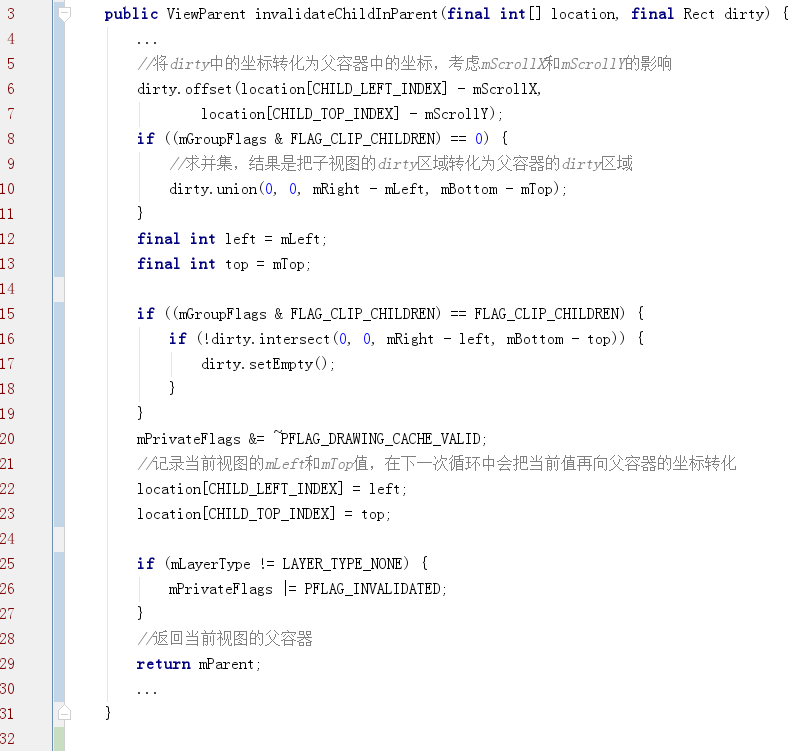
这些方法是可以设定重绘的区域的,包括invalidate()方法最后都要调用invalidateInternal方法。



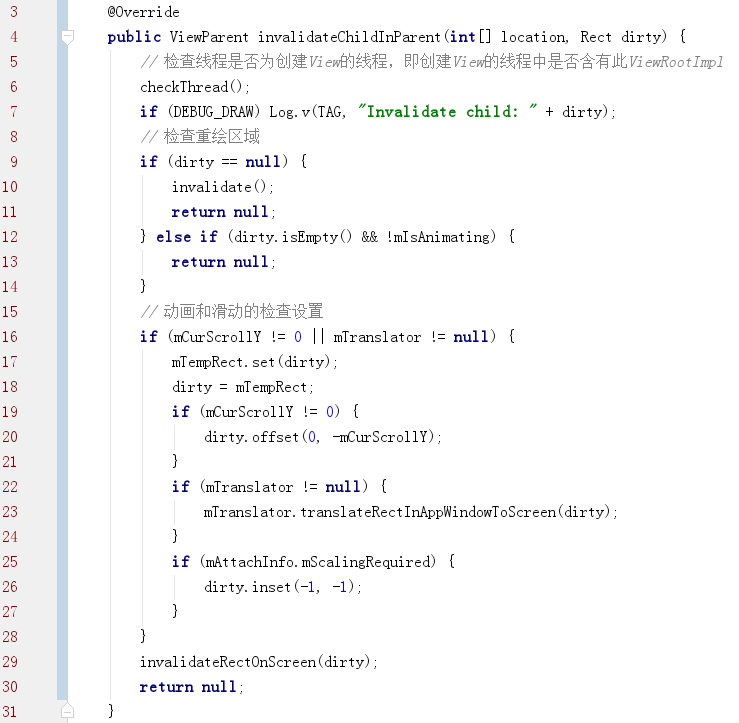
35行通过父View的invalidateChild(this, damage)方法，将需要重绘的区域传递给父View（在分析requestLayout的时候知道mParent是ViewGroup类型，所以调用的是ViewGroup中的invalidateChild方法）：



设置了需要重绘的区域dirty。之后再do...while方法中，反复的调用parent = parent.invalidateChildInParent(location, dirty)方法，来调用父类的invalidateChildInParent对View的重绘请求进行传递。由于不断向上调用父容器的方法，到最后会调用到ViewRootImpl的invalidateChildInParent方法，在调用到ViewRootImpl之前会调用ViewGroup的invalidateChildInParent：



调用invalidateChildInParent会传进去一个Rect叫dirty，代表子窗口需要刷新的rect，父窗口会根据这个rect和父窗口本身做union，从而得到父窗口需要刷新的rect区域，然后再传给父窗口的父窗口，一直递归直到ViewRootImpl的invalidateChildInParent方法：



该方法所做的工作与上面的差不多，都进行了offset和union对坐标的调整，然后把dirty区域的信息保存在mDirty中，最后29行调用invalidateRectOnScreen方法，继续调用scheduleTraversals方法，之后使用handler.postCallback一个mTraversalRunnable对象，从而调用doTraversal方法，最后调用performTraversals()触发View的工作流程，由于没有添加measure和layout的标记位，因此measure、layout流程不会执行，而是直接从performDraw流程开始。

总结：当View调用invalidate后，会为该View添加一个标记位，同时不断向父容器请求刷新，父容器通过计算得出自身需要重绘的区域，最后传递到ViewRootImpl中，最终触发performTraversals方法，进行开始View树重绘流程(只绘制需要重绘的视图，即设置了标志位的子View才会被重绘)。

**另外;ViewGroup的invalidate就是对子view进行重绘。**

* **postInvalidate()(源码就不分析了)**

这个方法与invalidate方法的作用是一样的，都是使View树重绘，但两者的使用条件不同，postInvalidate是在非UI线程中调用，invalidate则是在UI线程中调用。

* **LayoutParams**



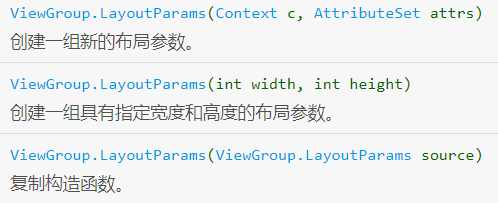
当我们使用java代码动态创建或者改变一个视图的时候需要用到LayoutParams类。LayoutParams是ViewGroup的内部类，所以ViewGroup.LayoutParams是一个基类，它主要用于包装一个View在父布局中宽，高的信息，也就是告诉它的父布局的它的布局方式。它的宽高有三种方式表述：

1. MATCH\_PARENT，这意味着该视图要与其父级（减去填充）一样大。

2、WRAP\_CONTENT，包裹住控件就好。

3、一个确切的数字。

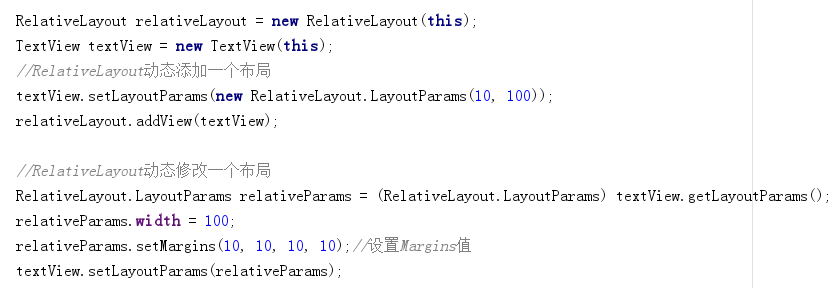
**构造方法：**



它的三个构造方法都是初始化LayoutParams的width和height即视图的宽和高参数。其中第一个是通过提取attrs中（即布局文件中）的layout\_width和layout\_height两个参数来初始化。

**使用：**

1. 并不是创建所有的视图都使用ViewGroup.LayoutParams，一般使用它的子类。当我们将一个视图添加到一个ViewGroup类型父视图的时候，这个父视图是什么类型，这个子视图就要创建什么类型的LayoutParams。比如：一个LinearLayout要动态添加一个TextView，那么这个TextView要设置一个LinearLayout.LayoutParams：



1. 每个ViewGroup类型父视图绘制子View时都有自己的规则，所以每个子View被动态添加到父容器时都要创建它对应父容器的LayoutParams。从源码的角度看就是：原因在于RelayoutLayout（或其他继承自ViewGroup的layout，如：LinearLayout）在进行递归布局的时候，RelayoutLayout会获取子View的LayoutParams，并强制转换成RelayoutLayout.LayoutParams，如:

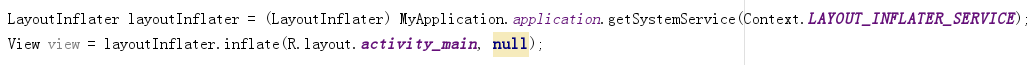
**RelayoutLayout.LayoutParams lp = (RelayoutLayout.LayoutParams) child.getLayoutParams();**

* **LayoutInflater布局加载器**

主要用于将xml布局加载成一个View对象，被称为布局加载器。

**初始化方式**

第一种：



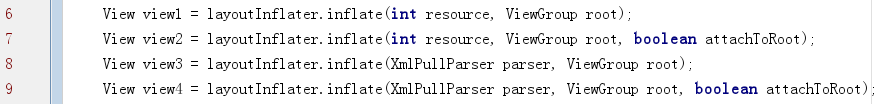
第二种：

1505877942(1)

其实第二种最终也是通过第一种方式初始化，只不过是LayoutInfkater给封装了一下。这里注意：初始化LayoutInflater使用的Context，使用Activity或者Application的实例都可以初始化。

**加载布局**

LayoutInflater通过inflate方法加载布局：



上面四个方法其中L6的inflate方法内部调用的是L7的方法，假如L6中root不为空则attachToRoot为true。但是所有的inflate方法最终调用的还是LayoutInflater当中的inflate(XmlPullParser parser,ViewGroup root, boolean attachToRoot)方法，这里我们只分析这个方法的执行过程，就能知道它是怎么加载布局的：



通过上面的分析我们知道了加载布局的大体流程，也明确的知道了L7的inflate方法参数的意义：

resource：是要加载的布局id。

root：它必须是一个ViewGroup类型的View。如果你传递了root，他有三个作用：

1、**root一定会生成一个LyaoutParams**。并且设置给新View（这一步跟attachToRoot的值无关）。

2、**将新View添加进root**。这取决于attachToRoot的值，为true则添加，false不添加。

3、**一旦root添加了这个新View，那么一定会将root返回，否则返回这个新View**。

4、如果根布局是<merge/>标签，则将布局的view解析出来添加进root中，并返回root。

attachToRoot：如果root为空可以认定它没有任何作用，如果root不为空，则它决定是否将新View添加到root当中，为true则添加，false不添加。

**另外**，inflate这个方法的返回值也要注意：假如root添加了新View或者解析<merge/>标签布局则返回root对象，如果没添加则返回这个新View。

**布局加载原理**

参考：<http://allenfeng.com/2017/02/24/how-android-layout-inflater-work/>

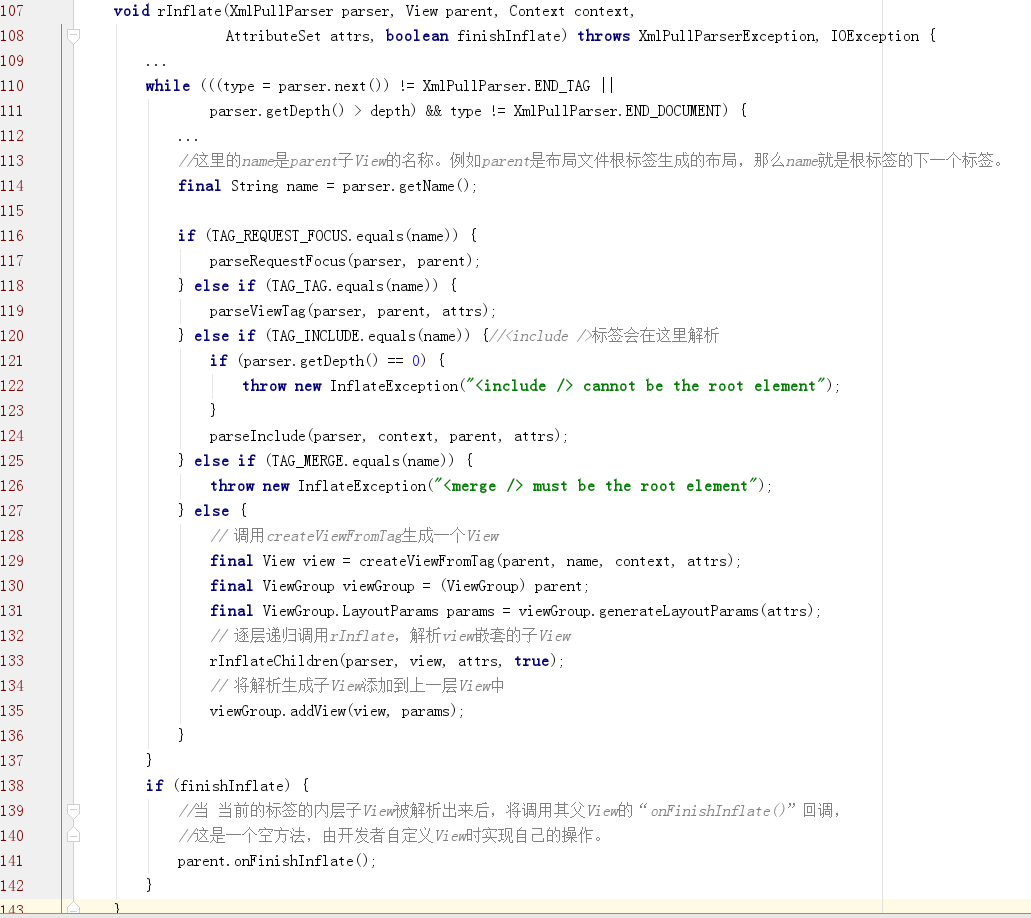
布局加载的具体流程就是布局加载原理，从上图的代码可以捋清楚布局加载的大体流程：

首先看L13，判断如果加载的布局的根节点是<merge/>标签那么调用rInflate方法继续解析里面的子布局然后添加到root中，最后将root返回。

继续看L22，如果不是<merge/>标签，会调用createViewFromTag根据根标签生成一个View，然后在L36行调用rInflateChildren（内部也是调用rInflate方法）将xml里根布局中的子标签递归解析，并添加到View中，继续在L40行如果传进来的root不为空并且attchToRoot为true则将新生成的View添加进root最后返回root。如果在L40行没有将View添加进root，则返回这个View。

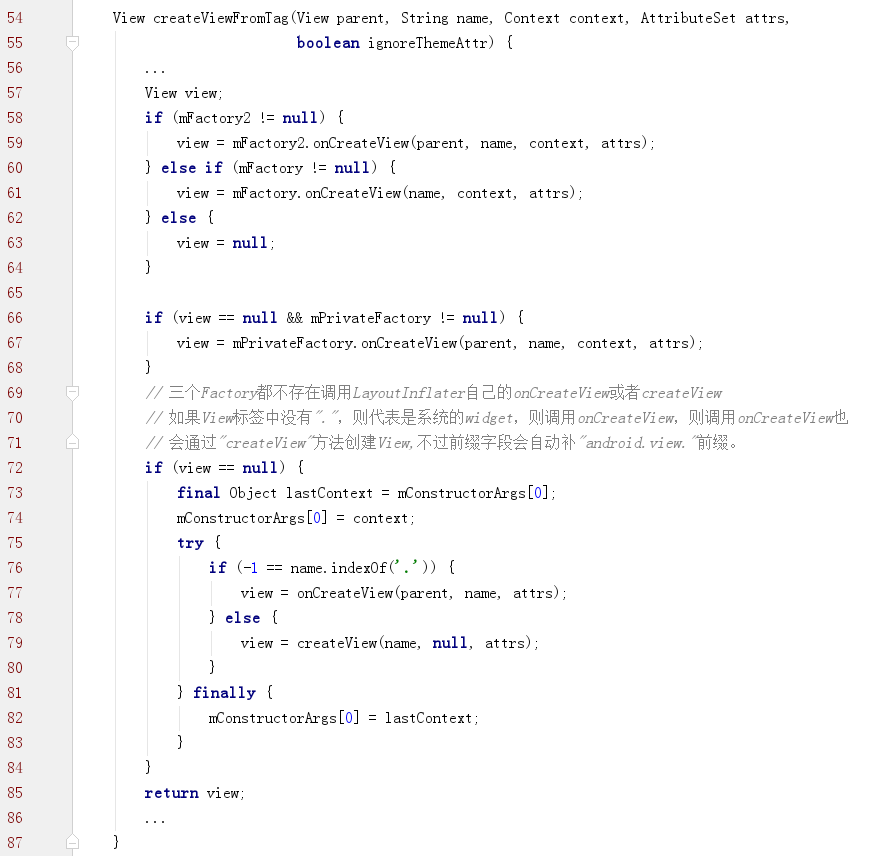
上面就是一个布局加载一个布局的流程，但是最核心的原理是如何解析布局中的所有的标签，并根据这些标签创建一个View，这里主要涉及到两个方法，rInflate方法和createViewFromTag方法。

rInflate方法主要是从根标签的下一个标签开始递归解析xml文件里的标签，每解析一个标签就调用createViewFromTag方法创建一个View，直到解析完最后一个标签：



原来，rInflate主要是调用了createViewFromTag生成当前解析到的View节点，并递归调用rInflate逐层生成子View，添加到生成的子View的父View中。当某个节点下面的所有子节点View解析生成完成后，才会调起onFinishInflate回调。

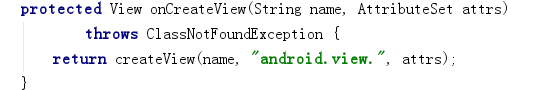
所以createViewFromTag才是真正生成View的地方：



从L58行到L68行会依次调用mFactory2、mFactory和mPrivateFactory三者之一的onCreateView方法去创建一个View。如果这几个Factory都为null，会调用LayoutInflater自己的onCreateView或者createView来实例化View。

注意：通常情况下，自定义工厂mFactory2、mFactory和私有工厂mPrivateFactory是空的，当Activity继承自AppCompatActivity时，才会存在自定义Factory。

看来最终生成View在onCreateView或者createView中，onCreateView调用的其实是createView，即View的节点名称没有.时，将自动补上android.view.前缀（即完整类名）：



所以最终生成View是在createView方法中：



L152通过反射机制生成一个实例，这个实例就是我们需要的View了。

到此布局的加载机制就全部完成！