链滴

Android 之自定义 View 的死亡三部曲之（Layout）

作者：angels

原文链接：https://ld246.com/article/1496383185342

来源网站：链滴

- 文章独家授权公众号：码个蛋
- 更多分享：　http://www.cherylgood.cn

前言

- 大家好！本次我们将继续学习Android之自定义View的死亡三部曲中的第二部：排兵布阵
- 我们在上一篇　Android之自定义View的死亡三部曲之（Measure）中分析了死亡三部曲的第一部也是三部中最复杂的一步：View的测量，想知道View的测量相关知识可以点进去查看哦！
- 通过第一部View的测量，我们就能拿到View的三围数据了（View的宽高）。
- 那么接下来我们要做的当然就是对测量好的View进行布局了。
- Ok，说干就干，这次，我们同样是从ViewRootImpl的performTraversals方法开始，还记得我们的erformTraversals方法体内部都有哪些内容么？我们再粘贴一下代码吧。

```
private void performTraversals() {
    ...
      if (!mStopped) {
    //1、获取顶层布局的childWidthMeasureSpec
      int childWidthMeasureSpec = getRootMeasureSpec(mWidth, lp.width);
    //2、获取顶层布局的childHeightMeasureSpec
      int childHeightMeasureSpec = getRootMeasureSpec(mHeight, lp.height);
      //3、测量开始测量
      performMeasure(childWidthMeasureSpec, childHeightMeasureSpec);
      }
    }

    if (didLayout) {
    //4、执行布局方法
      performLayout(lp, desiredWindowWidth, desiredWindowHeight);
      ...
    }
    if (!cancelDraw && !newSurface) {
     ...
    //5、开始绘制了哦
          performDraw();
      }
    }
  ...
}
```

- 我们上次分析测量是以performMeasure为入口进行分析的，那么本次分析到布局，当然是从performLayout作为起点了。
- Ok，那么我们就直接看performLayout方法体内部的源码吧

```
private void performLayout(WindowManager.LayoutParams lp, int desiredWindowWidth,
int desiredWindowHeight) {
    mLayoutRequested = false;
    mScrollMayChange = true;
```

```
    mInLayout = true;

    final View host = mView;
    if (DEBUG_ORIENTATION || DEBUG_LAYOUT) {
        Log.v(TAG, "Laying out " + host + " to (" +
                host.getMeasuredWidth() + ", " + host.getMeasuredHeight() + ")");
    }

    Trace.traceBegin(Trace.TRACE_TAG_VIEW, "layout");
    try {
    //1、调用了host.layout
      host.layout(0, 0, host.getMeasuredWidth(), host.getMeasuredHeight());
      mInLayout = false;


    .....
    } finally {
        Trace.traceEnd(Trace.TRACE_TAG_VIEW);
    }
    mInLayout = false;
 }
```

- 我们可以看到，在1处，直接调用了host.layout进行布局，而host是什么东东呢？其实host就是我的DecorView，还记得我们之前分析View的诞生之谜的时候，在创建ViewRootImpl时，直接把DecoView赋值给mView了。
- 那么也就是说其实是调用了DecorView的layout方法。我们再看下其传递的参数分别是0，0，host.etMeasuredWidth()，host.getMeasuredHeight()
- 而这四个参数按顺利所代码的含义分别是left，top，right，bottom，也就是左、上、右、下
- left、top当然是0了，为什么呢？难道你想手机屏幕显示一个画面是，左边和顶部不是刚好贴合的？显然不会希望这样，简直丑死啦。
- 宽就是我们DecorView测量后的宽度，高就是DecorView测量后的高度

- Ok，所有的控件当时都是继承自View了，那么我们看下View的layout方法

```
  public void layout(int l, int t, int r, int b) {
    if ((mPrivateFlags3 & PFLAG3_MEASURE_NEEDED_BEFORE_LAYOUT) != 0) {
        onMeasure(mOldWidthMeasureSpec, mOldHeightMeasureSpec);
        mPrivateFlags3 &= ~PFLAG3_MEASURE_NEEDED_BEFORE_LAYOUT;
        }

        int oldL = mLeft;
        int oldT = mTop;
        int oldB = mBottom;
        int oldR = mRight;
        //1、isLayoutModeOptical(mParent)判断是传统模式还是视觉模式，不懂的小伙伴可以百度下哦
        //然后对不同模式分别调用对象的方法，作用是设置View的四个点
        boolean changed = isLayoutModeOptical(mParent) ?
            setOpticalFrame(l, t, r, b) : setFrame(l, t, r, b);

        if (changed || (mPrivateFlags & PFLAG_LAYOUT_REQUIRED) == PFLAG_LAYOUT_REQUIR
```

```
D) {
        //2、直接调用onLayout方法进行布局
        onLayout(changed, l, t, r, b);

        if (shouldDrawRoundScrollbar()) {
            if(mRoundScrollbarRenderer == null) {
                mRoundScrollbarRenderer = new RoundScrollbarRenderer(this);
            }
        } else {
            mRoundScrollbarRenderer = null;
        }

        mPrivateFlags &= ~PFLAG_LAYOUT_REQUIRED;

        ListenerInfo li = mListenerInfo;
      if (li != null && li.mOnLayoutChangeListeners != null) {
          ArrayList listenersCopy =
                  (ArrayList)li.mOnLayoutChangeListeners.clone();
          int numListeners = listenersCopy.size();
          for (int i = 0; i < numListeners; ++i) {
              //3、如果设置了OnLayoutChangeListener，在layout之后就会回调告诉你了哦
              listenersCopy.get(i).onLayoutChange(this, l, t, r, b, oldL, oldT, oldR, oldB);
          }
      }
      }

      mPrivateFlags &= ~PFLAG_FORCE_LAYOUT;
      mPrivateFlags3 |= PFLAG3_IS_LAID_OUT;
  }
```

● 在1中针对不同的layoutMode调用了不同的方法，我们来看下一班的layoutMode模式下调用setFrame方法时，内部做了什么操作呢，

```
    protected boolean setFrame(int left, int top, int right, int bottom) {
        boolean changed = false;

        if (DBG) {
            Log.d("View", this + " View.setFrame(" + left + "," + top + ","
        + right + "," + bottom + ")");
        }
        //1、如果有一个值发生了改变，那么就需要重新调用onLayout方法了，后面会分析到
        if (mLeft != left || mRight != right || mTop != top || mBottom != bottom) {
            changed = true;

            // Remember our drawn bit
            int drawn = mPrivateFlags & PFLAG_DRAWN;

            //2、保存旧的宽和高
            int oldWidth = mRight - mLeft;
            int oldHeight = mBottom - mTop;
            //计算新的宽和高
            int newWidth = right - left;
```

```
        int newHeight = bottom - top;
        //3、判断宽高是否有分生变化
        boolean sizeChanged = (newWidth != oldWidth) || (newHeight != oldHeight);

        //Invalidate our old position
        //4、如果大小变化了，在已绘制了的情况下就请求重新绘制
        invalidate(sizeChanged);

        //5、存储新的值
        mLeft = left;
        mTop = top;
        mRight = right;
        mBottom = bottom;
        mRenderNode.setLeftTopRightBottom(mLeft, mTop, mRight, mBottom);

        mPrivateFlags |= PFLAG_HAS_BOUNDS;

        if (sizeChanged) {
            //6、大小变化时进行处理
            sizeChange(newWidth, newHeight, oldWidth, oldHeight);
         }

        if ((mViewFlags & VISIBILITY_MASK) == VISIBLE || mGhostView != null) {
             //7、如果此时View是可见状态下，立即执行绘制操作
              invalidate(sizeChanged);

            }

         mPrivateFlags |= drawn;

        mBackgroundSizeChanged = true;
        if (mForegroundInfo != null) {
            mForegroundInfo.mBoundsChanged = true;
          }

        notifySubtreeAccessibilityStateChangedIfNeeded();
    }
     return changed;
}
```

- 可以看到changed的值只与四个点是否发生了变化有关。
- 同时，我们还发现，如果你想获得某个view的top、left、right、bottom的值，在layout之后就可拿到了。
- 而从View.layout方法的2位置处我们知道，在执行了setFrame之后调用的是onLayout方法，所以就是说，我们可以在onLayout方法中获得四个位置点的值
- View类的成员变量mLeft、mRight、mTop和mBottom分别用来描述当前视图的左右上下四条边其父视图的左右上下四条边的距离，如果它们的值与参数left、right、top和bottom的值不相等，那就说明当前视图的大小或者位置发生变化了。这时候View类的成员函数setFrame就会将参数left、right、top和bottom的值分别记录在成员变量mLeft、mRight、mTop和mBottom中。

- 然后我们很开心的点开了View.onLayout方法，发现，居然是空的！～～空的！

```
protected void onLayout(boolean changed, int left, int top, int right, int bottom) {
}
```

```

● 没错，就是空的，一般该方法是用来确认childView的位置的，比如FrameLayout会调用onLayout法告知childView，你可以可以开始布局了哦。然后childView就会调用自身的layout方法完成自身的局工作，如果childView中还包含有childView，就会一直调用下去。

---

● 我们先来梳理下流程：

1、performTraversals内部调用performLayout开始执行布局工作

2、performLayout内部会调用layout开始进行布局

3、layout中会调用setFrame确定mTop，mLeft，mRight，mBottom的值以及判断是个点的值是发生了变化

4、最后调用onLayout方法通知下面的childView进行布局操作

● ok，那么我们就分析下FrameLayout的onLayout方法

---

```
@Override
protected void onLayout(boolean changed, int left, int top, int right, int bottom) {

    layoutChildren(left, top, right, bottom, false /* no force left gravity */);
}
```

● 从上面可以看到内部只是调用了layoutChildren方法，layoutChildren才是具体的实现

● 我们继续看下layoutChildren里面的代码：

---

```
void layoutChildren(int left, int top, int right, int bottom, boolean forceLeftGravity) {
    //1、获得子view的熟练
    final int count = getChildCount();
    //2、获得父view左面位置，getPaddingLeftWithForeground获得的是对应的内边距
    final int parentLeft = getPaddingLeftWithForeground();
    //3、获得父view右边位置
    final int parentRight = right - left - getPaddingRightWithForeground();
    //4、获得父view顶部位置
    final int parentTop = getPaddingTopWithForeground();
    //4、获得父view底部位置
    final int parentBottom = bottom - top - getPaddingBottomWithForeground();

    for (int i = 0; i < count; i++) {
        //5、遍历子view
        final View child = getChildAt(i);
        if (child.getVisibility() != GONE) {
```
```

```
            final LayoutParams lp = (LayoutParams) child.getLayoutParams();

            final int width = child.getMeasuredWidth();
            final int height = child.getMeasuredHeight();

            int childLeft;
            int childTop;

            int gravity = lp.gravity;
            if (gravity == -1) {
                gravity = DEFAULT_CHILD_GRAVITY;
            }

            final int layoutDirection = getLayoutDirection();
            final int absoluteGravity = Gravity.getAbsoluteGravity(gravity, layoutDirection);
            final int verticalGravity = gravity & Gravity.VERTICAL_GRAVITY_MASK;
             //6、针对不同的水平方向Gravity做处理
            switch (absoluteGravity & Gravity.HORIZONTAL_GRAVITY_MASK) {
                case Gravity.CENTER_HORIZONTAL:
                    childLeft = parentLeft + (parentRight - parentLeft - width) / 2 +
                    lp.leftMargin - lp.rightMargin;
                    break;
                case Gravity.RIGHT:
                    if (!forceLeftGravity) {
                        childLeft = parentRight - width - lp.rightMargin;
                        break;
                    }
                case Gravity.LEFT:
                default:
                    childLeft = parentLeft + lp.leftMargin;
            }
            //6、针对不同的垂直方向Gravity做处理
            switch (verticalGravity) {
                case Gravity.TOP:
                    childTop = parentTop + lp.topMargin;
                    break;
                case Gravity.CENTER_VERTICAL:
                     childTop = parentTop + (parentBottom - parentTop - height) / 2 +
                    lp.topMargin - lp.bottomMargin;
                    break;
                case Gravity.BOTTOM:
                    childTop = parentBottom - height - lp.bottomMargin;
                    break;
                default:
                    childTop = parentTop + lp.topMargin;
            }
            //7、调用child的layout方法，对child进行布局，前面我们分析了
            child.layout(childLeft, childTop, childLeft + width, childTop + height);
        }
}
}
```

- 知识点梳理：

1、获取父View的内边距padding的值

2、遍历子View，处理子View的layout_gravity属性、根据View测量后的宽和高、父View的paddin值、来确定子View的布局参数，

3、调用child.layout方法，对子View进行布局

对childView进行布局

- 从上面的分析我们的可以知道，如果子view属于FrameLayout这种布局类的View，里面就会重复面流程，如果不是，最终就会调用到View.onLayout,而这个方法是一个空的实现，所以我们在自定义View时，需要重新onLayout实现布局的操作

总结：

- 布局流程主要的操作就是确定View的四个点的数值，相对于之前的测量，是不是要简单一些呢？
-