Android RelativeLayout和LinearLayout性能分析

RelativeLayout和LinearLayout是Android中常用的布局，两者的使用会极大的影响程序生成每一帧的性能，因此，正确的使用它们是提升程序性能的重要工作。下面将通过分析它们的源码来探讨其View绘制性能，并得出其正确的使用方法。

RelativeLayout和LinearLayout是如何进行measure的？

通过官方文档我们知道View的绘制进行measure, layout, draw，分别对应onMeasure(), onLayout, onDraw()，而他们的性能差异主要在onMeasure()上。首先是RelativeLayout：

```java
@Override
protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
......
View[] views = mSortedHorizontalChildren;
int count = views.length;

for (int i = 0; i < count; i++) {
    View child = views[i];
    if (child.getVisibility() != GONE) {
        LayoutParams params = (LayoutParams) child.getLayoutParams();
        int[] rules = params.getRules(layoutDirection);

        applyHorizontalSizeRules(params, myWidth, rules);
        measureChildHorizontal(child, params, myWidth, myHeight);

        if (positionChildHorizontal(child, params, myWidth, isWrapContentWidth)) {
            offsetHorizontalAxis = true;
        }
    }
}

views = mSortedVerticalChildren;
count = views.length;
final int targetSdkVersion = getContext().getApplicationInfo().targetSdkVersion;

for (int i = 0; i < count; i++) {
    View child = views[i];
    if (child.getVisibility() != GONE) {
        LayoutParams params = (LayoutParams) child.getLayoutParams();

        applyVerticalSizeRules(params, myHeight);
        measureChild(child, params, myWidth, myHeight);
        if (positionChildVertical(child, params, myHeight, isWrapContentHeight)) {
            offsetVerticalAxis = true;
        }

        if (isWrapContentWidth) {
            if (isLayoutRtl()) {
                if (targetSdkVersion < Build.VERSION_CODES.KITKAT) {
                    width = Math.max(width, myWidth - params.mLeft);
                } else {
                    width = Math.max(width, myWidth - params.mLeft - params.leftMargin);
                }
            } else {
                if (targetSdkVersion < Build.VERSION_CODES.KITKAT) {
                    width = Math.max(width, params.mRight);
                } else {
                    width = Math.max(width, params.mRight + params.rightMargin);
                }
            }
        }

        if (isWrapContentHeight) {
            if (targetSdkVersion < Build.VERSION_CODES.KITKAT) {
                height = Math.max(height, params.mBottom);
            } else {
                height = Math.max(height, params.mBottom + params.bottomMargin);
            }
        }

        if (child != ignore || verticalGravity) {
            left = Math.min(left, params.mLeft - params.leftMargin);
            top = Math.min(top, params.mTop - params.topMargin);
        }

        if (child != ignore || horizontalGravity) {
            right = Math.max(right, params.mRight + params.rightMargin);
            bottom = Math.max(bottom, params.mBottom + params.bottomMargin);
        }
    }
}
......
}
```

根据上述关键代码，RelativeLayout分别对所有子View进行两次measure，横向纵向分别进行一次。
而LinearLayout：

```
@Override
protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
    if (mOrientation == VERTICAL) {
        measureVertical(widthMeasureSpec, heightMeasureSpec);
    } else {
        measureHorizontal(widthMeasureSpec, heightMeasureSpec);
    }
}
```

根据线性布局方向，执行不同的方法，这里分析measureVertical方法。

```
void measureVertical(int widthMeasureSpec, int heightMeasureSpec) {
......
for (int i = 0; i < count; ++i) {
    ......

    LinearLayout.LayoutParams lp = (LinearLayout.LayoutParams) child.getLayoutParams();

    totalWeight += lp.weight;

    if (heightMode == MeasureSpec.EXACTLY && lp.height == 0 && lp.weight > 0) {
        // Optimization: don't bother measuring children who are going to use
        // leftover space. These views will get measured again down below if
        // there is any leftover space.
        final int totalLength = mTotalLength;
        mTotalLength = Math.max(totalLength, totalLength + lp.topMargin + lp.bottomMargin);
        skippedMeasure = true;
    } else {
        int oldHeight = Integer.MIN_VALUE;

        if (lp.height == 0 && lp.weight > 0) {
            // heightMode is either UNSPECIFIED or AT_MOST, and this
            // child wanted to stretch to fill available space.
            // Translate that to WRAP_CONTENT so that it does not end up
            // with a height of 0
            oldHeight = 0;
            lp.height = LayoutParams.WRAP_CONTENT;
        }

        // Determine how big this child would like to be. If this or
        // previous children have given a weight, then we allow it to
        // use all available space (and we will shrink things later
        // if needed).
        measureChildBeforeLayout(
               child, i, widthMeasureSpec, 0, heightMeasureSpec,
               totalWeight == 0 ? mTotalLength : 0);

        if (oldHeight != Integer.MIN_VALUE) {
           lp.height = oldHeight;
        }

        final int childHeight = child.getMeasuredHeight();
        final int totalLength = mTotalLength;
        mTotalLength = Math.max(totalLength, totalLength + childHeight + lp.topMargin +
               lp.bottomMargin + getNextLocationOffset(child));

        if (useLargestChild) {
            largestChildHeight = Math.max(childHeight, largestChildHeight);
        }
    }
    ......
```

LinearLayout首先会对所有的子View进行measure，并计算totalWeight(所有子View的weight属性之和)，然后判断子View的weight属性是否为最大，如为最大则将剩余的空间分配给它。如果不使用weight属性进行布局，则不进行第二次measure。

```
// Either expand children with weight to take up available space or
// shrink them if they extend beyond our current bounds. If we skipped
// measurement on any children, we need to measure them now.
int delta = heightSize - mTotalLength;
if (skippedMeasure || delta != 0 && totalWeight > 0.0f) {
    float weightSum = mWeightSum > 0.0f ? mWeightSum : totalWeight;

    mTotalLength = 0;

    for (int i = 0; i < count; ++i) {
        final View child = getVirtualChildAt(i);

        if (child.getVisibility() == View.GONE) {
            continue;
        }

        LinearLayout.LayoutParams lp = (LinearLayout.LayoutParams) child.getLayoutParams();

        float childExtra = lp.weight;
        if (childExtra > 0) {
            // Child said it could absorb extra space -- give him his share
            int share = (int) (childExtra * delta / weightSum);
            weightSum -= childExtra;
            delta -= share;

            final int childWidthMeasureSpec = getChildMeasureSpec(widthMeasureSpec,
                    mPaddingLeft + mPaddingRight +
                            lp.leftMargin + lp.rightMargin, lp.width);

            // TODO: Use a field like lp.isMeasured to figure out if this
            // child has been previously measured
            if ((lp.height != 0) || (heightMode != MeasureSpec.EXACTLY)) {
                // child was measured once already above...
                // base new measurement on stored values
                int childHeight = child.getMeasuredHeight() + share;
                if (childHeight < 0) {
                    childHeight = 0;
                }

                child.measure(childWidthMeasureSpec,
                        MeasureSpec.makeMeasureSpec(childHeight, MeasureSpec.EXACTLY));
            } else {
                // child was skipped in the loop above.
                // Measure for this first time here
                child.measure(childWidthMeasureSpec,
                        MeasureSpec.makeMeasureSpec(share > 0 ? share : 0,
                                MeasureSpec.EXACTLY));
            }

            // Child may now not fit in vertical dimension.
            childState = combineMeasuredStates(childState, child.getMeasuredState()
                    & (MEASURED_STATE_MASK>>MEASURED_HEIGHT_STATE_SHIFT));
        }

        ......
    }
     ......
} else {
    alternativeMaxWidth = Math.max(alternativeMaxWidth,
                                   weightedMaxWidth);


    // We have no limit, so make all weighted views as tall as the largest child.
    // Children will have already been measured once.
    if (useLargestChild && heightMode != MeasureSpec.EXACTLY) {
        for (int i = 0; i < count; i++) {
            final View child = getVirtualChildAt(i);

            if (child == null || child.getVisibility() == View.GONE) {
                continue;
            }

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

            float childExtra = lp.weight;
            if (childExtra > 0) {
                child.measure(
                        MeasureSpec.makeMeasureSpec(child.getMeasuredWidth(),
                                MeasureSpec.EXACTLY),
                        MeasureSpec.makeMeasureSpec(largestChildHeight,
                                MeasureSpec.EXACTLY));
            }
        }
    }
}
......
}
```

提高绘制性能的使用方式

根据上面源码的分析，RelativeLayout将对所有的子View进行两次measure，而LinearLayout在使用weight属性进行布局时也会对子View进行两次measure，如果他们位于整个View树的顶端时并可能进行多层的嵌套时，位于底层的View将会进行大量的measure操作，大大降低程序性能。因此，应尽量将RelativeLayout和LinearLayout置于View树的底层，并减少嵌套。