1. 从整体看

1. 从整体看分俩个流程

- 测量流程:从根View递归调用每一级子View的measure()方法,对他们进行测量
- 布局流程:从根View递归调用每一级子View的layout()方法,把测量过程得出的子View的位置和尺寸传给子View,子View保存
- 尺寸是子View算的,位置是父View算的,通过layout传递给子View

2. 为什么分俩个流程

- 整个流程比较复杂,可能会测量多个子View后,放在一起才会得到最后的尺寸和位置

2. 个体流程(每个View)

- 运行前,开发者在xml文件里写入对View的布局要求layout_xxx
- 父View在自己的onMeasure()中,根据开发者在xml中写的对子View的要求,和自己的可用空间,得出对子View的具体尺寸要求
- 子View在自己的onMeasure中,根据自己的特性算出自己的期望尺寸
- 如果是ViewGroup,还会在这里调用每个子View的measure进行测量
- 父View在子View计算出期望尺寸后,得出子View的实际尺寸和位置
- 子View在自己的layout()方法中,将父View传进来的自己的世纪尺寸和位置保存
- 如果是ViewGroup,还会在onLayout()里调用每个子View的layout()把他们的尺寸位置传给他们

onLayout方法

- draw()是绘制的总调度,内部调用了onDraw()绘制方法
- measure()是测量的总调度,内部调用了onMeasure()测量方法
- layout()不止调度,也负责把父View传过来的参数存储起来
- onLayout()的工作是负责对子View进行布局的(自定义ViewGroup需要重写)

getMeasureWidth/getWidth

- 在layout调用之前,onMeasure调用之后只能拿到MeasureWidth和MeasureHeight
- getMeasureWidth是测量出来的宽高,不一定是最后的宽高

onMeasure完全自己计算

```
switch(specWidthMode){
    case MeasureSpec.EXACTLY:
        width = specWidthSize;
        break;
    case MeasureSpec.AT_MOST:
        if (width>specWidthSize){
            width = specWidthSize;
        }
        break;
}
setMeasureDimension(width,height)
```

- 上面是计算宽度的代码,高度同理,最后保存最后的测量接口
- Android提供了上面代码的方法

```
参数1:计算出来的宽度(demo中(PADDING+RADIUS)*2)
参数2:父View要求的宽度
width = resolveSize(width,widthMeasureSpec)
```

- 高度同上
- 还有一个方法

```
// 最后一个参数默认填0即可
resolveSizeAndState(height,heightMeasureSpec,0)
```

- resolveSizeAndState比resolveSize多存一个记录View是否被强制压小了
- 可以使用getMeasureState获取View是否被强制压小了

```
(getMeasureState()&MEASURED_STATE_TOO_SMALL)!=0就表示图片被压小了
```

- 因为许多原生的组件并不遵循resolveSizeAndState,所以其实用处不大

TagLayout自定义

- widthMeasureSpec和heightMeasureSpec是包含padding大小的
- 子View的margin属性是由ViewGroup处理的,ViewGroup在onMeasure和onLayout时一定要考虑ViewGroup自己的padding和子View的margin的影响
- measureChild减去了ViewGroup的padding保证child最大可用空间
- measureChildWithMargins减去了ViewGroup的padding和子View的margin和已经使用了的宽高,保证child最大可用空间
- 使用了measureChildWithMargins后报异常,需要重写generateDefaultLayoutParams

```
    @Override
    public LayoutParams generateLayoutParams(AttributeSet attrs) {
        return new MarginLayoutParams(getContext(),attrs);
    }
```

```
    private static final String TAG = "TagLayout";
    List<Rect> childrenBounds = new ArrayList<>();

    public TagLayout(Context context) {
        super(context);
    }

    public TagLayout(Context context, AttributeSet attrs) {
        super(context, attrs);
    }

    public TagLayout(Context context, AttributeSet attrs, int
defStyleAttr) {
        super(context, attrs, defStyleAttr);
    }

    @Override
    protected void onMeasure(int widthMeasureSpec, int
heightMeasureSpec) {
        super.onMeasure(widthMeasureSpec, heightMeasureSpec);
        int widthUsed = 0;
        int heightUsed = 0;
        int maxLineHeight = 0;
        int childCount = getChildCount();
        for (int i = 0; i < childCount; i++) {
            View child = getChildAt(i);
```

```
            measureChildWithMargins(child, widthMeasureSpec,
widthUsed, heightMeasureSpec, heightUsed);
            Rect rect;
            if (childrenBounds.size() <= i) {
                childrenBounds.add(new Rect());
                rect = childrenBounds.get(i);
            } else {
                rect = childrenBounds.get(i);
            }
            if (child.getMeasuredHeight() > maxLineHeight) {
                maxLineHeight = child.getMeasuredHeight();
            }
            if (widthUsed + child.getMeasuredWidth() >
getMeasuredWidth()) {
                //换行
                heightUsed += maxLineHeight;
                maxLineHeight = 0;
                widthUsed = 0;
                measureChildWithMargins(child, widthMeasureSpec,
widthUsed, heightMeasureSpec, heightUsed);
            }
            rect.set(widthUsed, heightUsed, widthUsed +
child.getMeasuredWidth(), child.getMeasuredHeight() + heightUsed);
            widthUsed += child.getMeasuredWidth();
        }
        heightUsed += maxLineHeight;
        setMeasuredDimension(getMeasuredWidth(), heightUsed);
    }

    @Override
    protected void onLayout(boolean changed, int l, int t, int r,
int b) {
        int childCount = getChildCount();
        for (int i = 0; i < childCount; i++) {
            View child = getChildAt(i);
            Rect rect = childrenBounds.get(i);
            child.layout(rect.left, rect.top, rect.right,
rect.bottom);
        }
    }

    @Override
    public LayoutParams generateLayoutParams(AttributeSet attrs) {
        return new MarginLayoutParams(getContext(), attrs);
    }
```

- 换行后需要重新measure一下,不然父View不知道他在哪