Android--自定义View

资料

https://hencoder.com/tag/hui-zhi/

扔物线自定义view绘制

推荐书籍

为什么要学自定义view？

在我看来，现有android提供的控件满足不了某些用户提的需求，所以我们才要根据需求来绘制view控件

画图

首先我们画一个图至少需要两个工具：纸和笔，而在android中绘制自定义view则需要canvas和paint。

paint

Paint类用于定义绘图时的参数，主要包含颜色、文本、图形样式、位图模式、滤镜等几个方面。通过控制这些参数，我们就可以控制Paint的样式，绘制不同风格的文本、图片等

https://blog.csdn.net/itrenj/article/details/53596378

```
paint类拥有风格和颜色信息如何绘制几何学,文本和位图。
Paint 代表了Canvas上的画笔、画刷、颜料等等;
Paint类常用方法:
setARGB(int a, int r, int g, int b) // 设置 Paint对象颜色，参数一为alpha透明值
setAlpha(int a) // 设置alpha不透明度，范围为0~255
setAntiAlias(boolean aa) // 是否抗锯齿
setColor(int color)  // 设置颜色，这里Android内部定义的有Color类包含了一些常见颜色定义
setTextScaleX(float scaleX)  // 设置文本缩放倍数，1.0f为原始
setTextSize(float textSize)  // 设置字体大小
setUnderlineText(booleanunderlineText)  // 设置下划线
```

canvas

这个类相当于一个画布，你可以在里面画很多东西;

- drawRect(RectF rect, Paint paint) //绘制区域，参数一为RectF一个区域
- drawPath(Path path, Paint paint) //绘制一个路径，参数一为Path路径对象
- drawLine(float startX, float startY, float stopX, float stopY, Paintpaint) //画线，参数一起始点的x轴位置，参数二起始点的y轴位置，参数三终点的x轴水平位置，参数四y轴垂直位置，最后一个参数为Paint 画刷对象。
- drawPoint(float x, float y, Paint paint) //画点，参数一水平x轴，参数二垂直y轴，第三个参数为Paint对象。
- drawText(String text, float x, floaty, Paint paint) //渲染文本，Canvas类除了上面的还可以描绘文字，参数一是String类型的文本，参数二x轴，参数三y轴，参数四是Paint对象。
- drawOval(RectF oval, Paint paint)//画椭圆，参数一是扫描区域，参数二为paint对象;
- drawCircle(float cx, float cy, float radius,Paint paint)// 绘制圆，参数一是中心点的x轴，参数二是中心点的y轴，参数三是半径，参数四是paint对象;
- drawArc(RectF oval, float startAngle, float sweepAngle, boolean useCenter, Paint paint)//画弧，参数一是RectF对象，一个矩形区域椭圆形的界限用于定义在形状、大小、电弧，参数二是起始角(度)在电弧的开始，
  参数三扫描角(度)开始顺时针测量的，参数四是如果这是真的话,包括椭圆中心的电弧,并关闭它,如果它是假这将是一个弧线,参数五是Paint对象;

画图的步骤

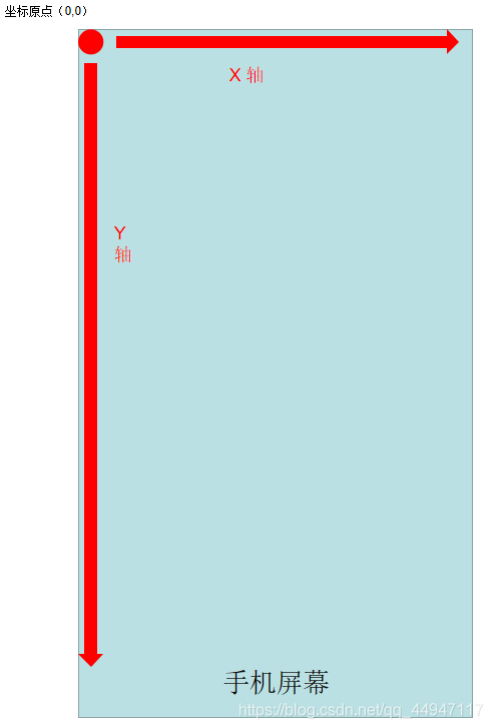
在手机上画一个矩形

- 1.先建立一个坐标系
- 2.测量你画的矩形的长和宽
- 3.确定矩形的摆放的位置
- 4.绘制

Android的坐标体系

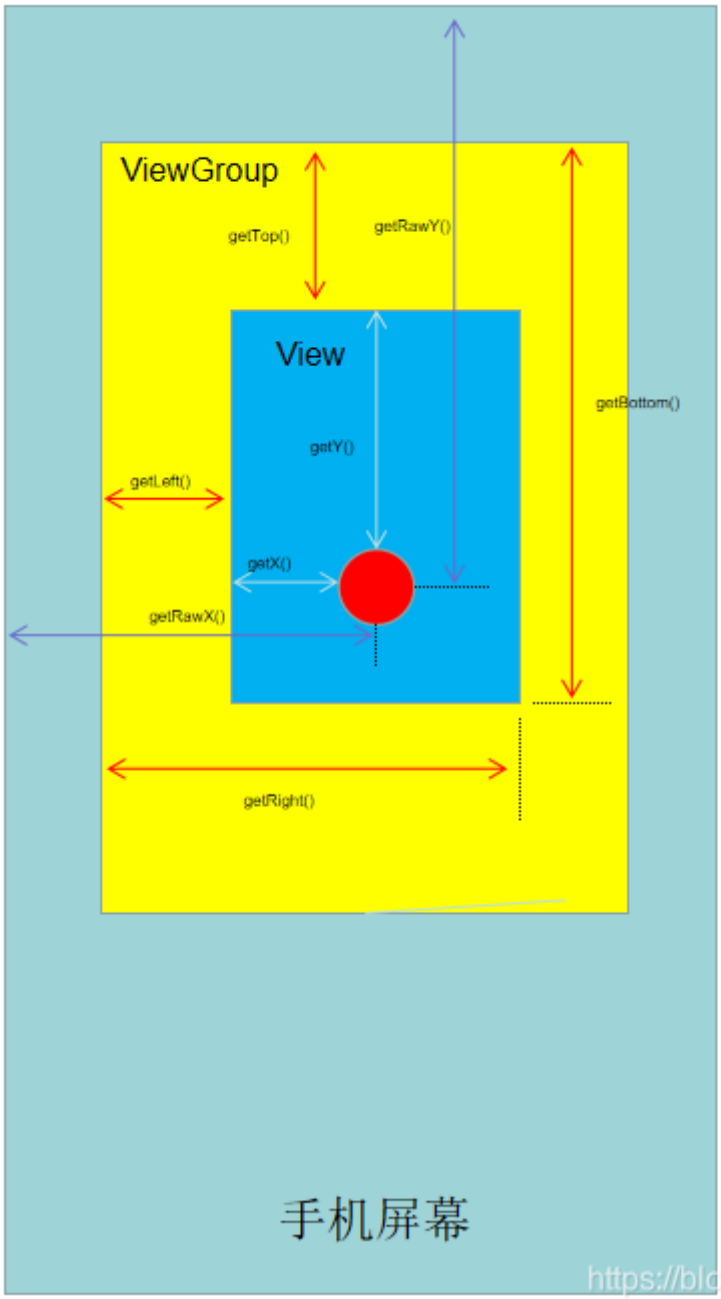
android坐标系，

以屏幕上的左上角为原点，向右为X轴，向下为y轴，如下图：



视图坐标系

视图坐标系也叫view坐标系，如下图所示：



红点为触摸点

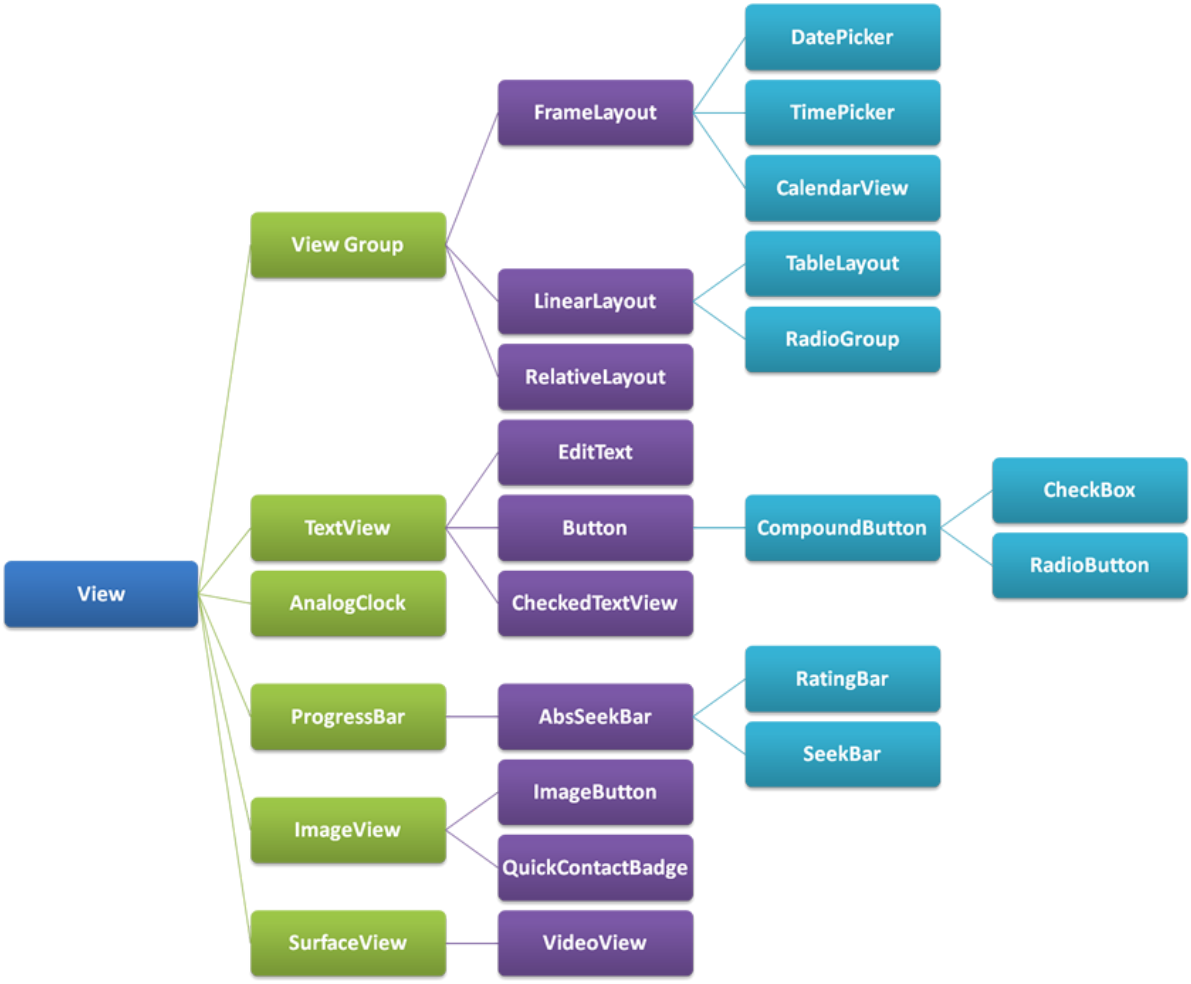
一个view控件怎么去获取自身的高度和宽度呢：

width=getRight()-getleft()

height=getBottom()-getTop()

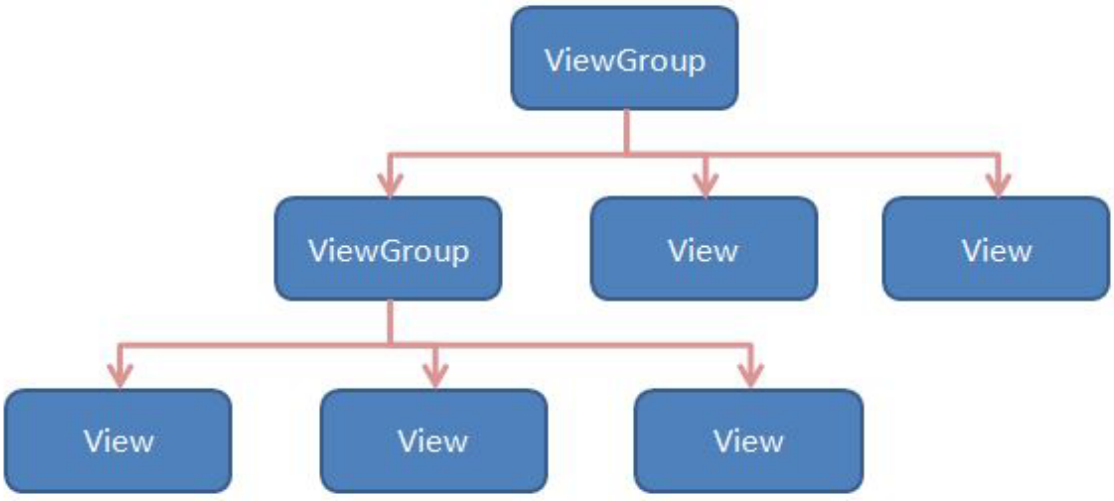
这里view源码中提供的getwidth()和getHeight()获取view宽高是相同的；

Android的View体系



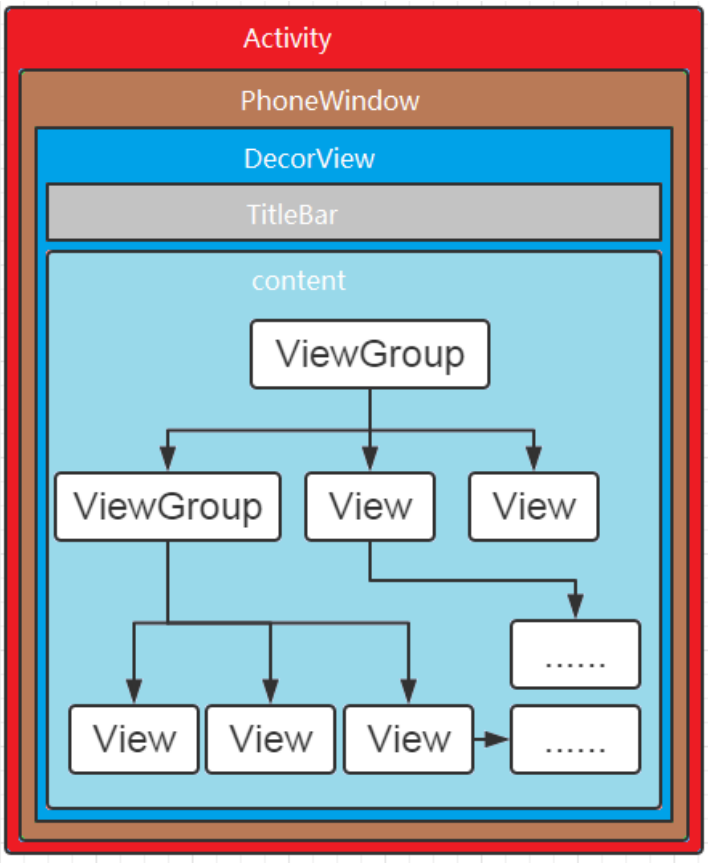
Android的ViewTree

View 是 Android 中所有控件的基类，ViewGroup 也是继承自 View。ViewGroup 是 View 的组合，可以包含 ViewGroup 也可以包含 View，以此类推，组成一个 View 树



在每个view中，都会有 onMeasure,onLayout,onDraw等方法，在view中实现这些方法并通过层层的调用，就能够进行绘制。

绘制流程（重点）



1. 从Activity的启动流程可以得到Activity创建Window（抽象类）的过程，PhoneWindow则是window的实现类
2. 创建PhoneWindow -> 创建WindowManager （onCreate()）-->setContentView()
3. 然后PhoneWindow创建decorView -> 在利用**windowManager中的addview()**把DecorView显示到屏幕上.

   windowManger中的addview()这一步是在ActivityThread的handleResumeActivity方法中被执行方法
4. 回调onResume方法的时候，DecorView还没有被添加到屏幕，所以当onResume被回调，指的是屏幕即将到显示，而不是已经显示，经过一系列操作，接着decorview添加到viewrootimpl中,进行decorview与viewrootimpl的绑定。在此之后可以通过viewRootImpl进行view的测量、布局、绘制，然后才显示。

windowmanger：WindowManager是一个接口，是用来管理窗口的。被WindowManagerImpl实现，而WindowManagerImpl实现WindowManager接口主要是通过WindowManagerGlobal，WindowManagerGlobal是在WindowManagerImpl中被初始化的：

viewrootImpl：负责管理Activity的view，主要处理view的测量，布局，绘制等事务，即measure，layout和traversal。

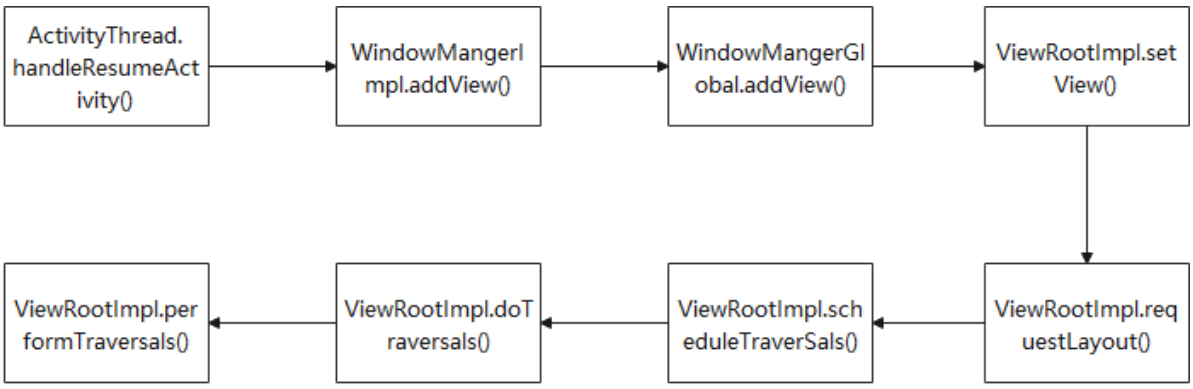
ActivityThread：Android应用的主线程（UI线程）

DecorView

DecorView是一个应用窗口的根容器，它本质上是一个FrameLayout。DecorView有唯一一个子View，它是一个垂直LinearLayout，包含两个子元素，一个是TitleView（ActionBar的容器），另一个是ContentView（窗口内容的容器）。

```
public void handleResumeActivity(IBinder token, boolean finalStateRequest,
boolean isForward,String reason) {
    // 调用Activity的onResume方法
    final ActivityClientRecord r = performResumeActivity(token,
finalStateRequest, reason);
    ...
    // 让decorView显示到屏幕上
    if (r.activity.mVisibleFromClient) {
        r.activity.makeVisible();
    }

    void makeVisible() {
    if (!mWindowAdded) {
        ViewManager wm = getWindowManager();
        wm.addView(mDecor, getWindow().getAttributes());
        mWindowAdded = true;
    }
    mDecor.setVisibility(View.VISIBLE);
}
```



requestlayout：请求布局，重新绘制

scheduleTraverSals：发送绘制消息

doTraversals：执行绘制

```
void scheduleTraversals() {
        if (!mTraversalScheduled) {
            mTraversalScheduled = true;
            //异步消息
            mTraversalBarrier = mHandler.getLooper().postSyncBarrier();
            mChoreographer.postCallback(
                    Choreographer.CALLBACK_TRAVERSAL, mTraversalRunnable, null);
            if (!mUnbufferedInputDispatch) {
```

```
                scheduleConsumeBatchedInput();
            }
            notifyRendererOfFramePending();
        }
    }

final class TraversalRunnable implements Runnable {
        @Override
        public void run() {
            doTraversal();
        }
}


void doTraversal() {
        if (mTraversalScheduled) {
            mTraversalScheduled = false;
            mHandler.getLooper().removeSyncBarrier(mTraversalBarrier);

            if (mProfile) {
                Debug.startMethodTracing("ViewAncestor");
            }

            Trace.traceBegin(Trace.TRACE_TAG_VIEW, "performTraversals");
            try {
                performTraversals();
            } finally {
                Trace.traceEnd(Trace.TRACE_TAG_VIEW);
            }

            if (mProfile) {
                Debug.stopMethodTracing();
                mProfile = false;
            }
        }
```

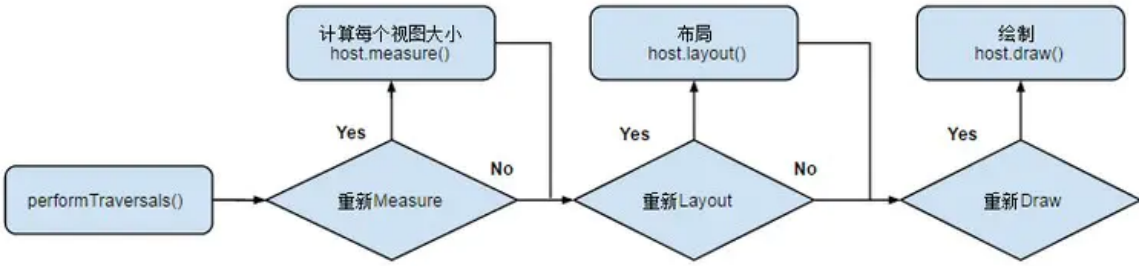
可以看到，最终**performTraversals()**方法触发了View的绘制。该方法内部，依次调用了performMeasure(),performLayout(),performDraw(),将View的measure，layout，draw过程，从顶层View分发了下去。

额外补充LayoutParams——代码动态布局

LayoutParams继承于Android.View.ViewGroup.LayoutParams相当于一个Layout的信息包，它封装了Layout的位置、高、宽等信息。假设在屏幕上一块区域是由一个Layout占领的，如果将一个View添加到一个Layout中，最好告诉Layout用户期望的布局方式，也就是将一个认可的layoutParams传递进去。

子view给父view传递一个意图。

正式进入view绘制的三个阶段



view的主要绘制流程为：

measure(): 根据父view传递的measurespec进行计算大小

layout()：根据measure子view所得到的布局大小和布局参数，将子view进行合适的位置摆放。

draw()：绘制view

Measure()

父容器调用子View的measure方法把上一步获得的MeasureSpec信息传递过去，子View的measure方法调用View#onMeasure()，onMeasure调用setMeasuredDimension()设置自身大小。

measure()-->onMeasure()-->setMeasuredDimension()

```
 protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
     //setMeasureDimentsion作用就是将测量好的宽跟高进行存储
     //getDefaultSize()的作用
     //Android会将View想要的尺寸以及其父控件对其尺寸限制信息measureSpec传递给
getDefaultSize方法，该方法要根据这些综合信息计算最终的量算的尺寸。
         setMeasuredDimension(getDefaultSize(getSuggestedMinimumWidth(),
widthMeasureSpec),
                 getDefaultSize(getSuggestedMinimumHeight(), heightMeasureSpec));
 }

 //返回建议view设置的最小宽度
// mMinWidth 对应与android:minWidth 这个属性所指定的值，这个属性如果不指定，
//  那么mMinWidth 则默认为0
//可以看到mBackground  == null 为没有设置背景，那么返回mMinWidth ，也就是
android:minWidth 这个属性所指定的值，这个值可以是0 ;如果View 设置了背景，则返回mMinWidth
与背景的最小宽度这两者的最大值。
//getSuggestedMinimumWidth() 的返回值就是View 在UNSPECIFIED 情况下的测量宽
 protected int getSuggestedMinimumWidth() {
         return (mBackground == null) ? mMinWidth : max(mMinWidth,
mBackground.getMinimumWidth());
     }
```

view

如果是view则通过xml获得宽高measureSpec，然后setMeasuredDimension(int, int)，设置一个布局大小的建议值，还不是最终值。

viewgroup

如果是viewgroup则先getChildCount()获得子view的数量，然后用getChildAt(int)，调用子view的measure()（这里会调用子view的onMeasure方法），然后累加（或者取最大）子view的宽高，算出最终的自己的大小，然后setMeasuredDimension(int, int)，为后续onLayout提供意见

```
  // 遍历测量 ViewGroup 中所有的 View
  protected void measureChildren(int widthMeasureSpec, int heightMeasureSpec) {
    final int size = mChildrenCount;
    final View[] children = mChildren;
    for (int i = 0; i < size; ++i) {
        final View child = children[i];
        if ((child.mViewFlags & VISIBILITY_MASK) != GONE) {
            measureChild(child, widthMeasureSpec, heightMeasureSpec);
        }
    }
  }

  // 测量某个指定的 View
  protected void measureChild(View child, int parentWidthMeasureSpec,
            int parentHeightMeasureSpec) {
    final LayoutParams lp = child.getLayoutParams();

    final int childWidthMeasureSpec =
getChildMeasureSpec(parentWidthMeasureSpec,
                mPaddingLeft + mPaddingRight, lp.width);
    final int childHeightMeasureSpec =
getChildMeasureSpec(parentHeightMeasureSpec,
                mPaddingTop + mPaddingBottom, lp.height);

    child.measure(childWidthMeasureSpec, childHeightMeasureSpec);
  }
```

measureSpeac

来自博客的解释是：

MeasureSpec 封装了从父View 传递给到子View的布局需求。每个MeasureSpec代表宽度或高度的要求。每个MeasureSpec都包含了size（父view需要告诉子View要量算的尺寸的上限）和mode（父view 的模式）

MeasureSpec是一个32位二进制的整数型，前两位是mode后30位是size

为什么要用measureSpeac?

这里设计到android手机适配的问题，安装android系统的手机尺寸版本有很多，通过父view提供子view的尺寸限制大小和父view的模式设置子view的measuresepc就避免了将控件尺寸写死，达到适配更多手机。

measurespeac的三种模式：

- EXACTLY：父容器已经测量出子View的大小。对应是 View 的LayoutParams的match_parent或者精确数值。
- AT_MOST：父容器已经限制子view的大小，View 最终大小不可超过这个值。对应是 View 的LayoutParams的wrap_content

- UNSPECIFIED：父容器不对View有任何限制，要多大给多大，这种情况一般用于系统内部，表示一种测量的状态。

```
protected void measureChildWithMargins(View child,
        int parentWidthMeasureSpec, int widthUsed,
        int parentHeightMeasureSpec, int heightUsed) {

    //获取子View的LayoutParam
    final MarginLayoutParams lp = (MarginLayoutParams) child.getLayoutParams();
    //通过父View的MeasureSpec和子View的margin，父View的padding计算，算出子View的
MeasureSpec
    final int childWidthMeasureSpec =
getChildMeasureSpec(parentWidthMeasureSpec,
            mPaddingLeft + mPaddingRight + lp.leftMargin + lp.rightMargin
                    + widthUsed, lp.width);
    final int childHeightMeasureSpec =
getChildMeasureSpec(parentHeightMeasureSpec,
            mPaddingTop + mPaddingBottom + lp.topMargin + lp.bottomMargin
                    + heightUsed, lp.height);
    //通过计算出来的MeasureSpec，让子View自己测量。
    child.measure(childWidthMeasureSpec, childHeightMeasureSpec);
}


public static int getChildMeasureSpec(int spec, int padding, int childDimension)
{
    int specMode = MeasureSpec.getMode(spec);
    int specSize = MeasureSpec.getSize(spec);
    //计算子View的大小
    int size = Math.max(0, specSize - padding);

    int resultSize = 0;
    int resultMode = 0;

    switch (specMode) {
    // 父View是EXACTLY的
    case MeasureSpec.EXACTLY:
        //子View的width或height是个精确值,则size为精确值，mode为 EXACTLY
        if (childDimension >= 0) {
            resultSize = childDimension;
            resultMode = MeasureSpec.EXACTLY;
        //子View的width或height是MATCH_PARENT,则size为父视图大小，mode为 EXACTLY
        } else if (childDimension == LayoutParams.MATCH_PARENT) {

            resultSize = size;
            resultMode = MeasureSpec.EXACTLY;
        //子View的width或height是WRAP_CONTENT,则size为父视图大小，mode为 AT_MOST
        } else if (childDimension == LayoutParams.WRAP_CONTENT) {

            resultSize = size;
            resultMode = MeasureSpec.AT_MOST;
        }
        break;

    // 2、父View是AT_MOST的
```

```
    case MeasureSpec.AT_MOST:
        //子View的width或height是个精确值,则size为精确值，mode为 EXACTLY
        if (childDimension >= 0) {
            // Child wants a specific size... so be it
            resultSize = childDimension;
            resultMode = MeasureSpec.EXACTLY;
        //子View的width或height是MATCH_PARENT,则size为父视图大小，mode为 AT_MOST
        } else if (childDimension == LayoutParams.MATCH_PARENT) {

            resultSize = size;
            resultMode = MeasureSpec.AT_MOST;
        //子View的width或height是MATCH_PARENT,则size为父视图大小，mode为 AT_MOST
        } else if (childDimension == LayoutParams.WRAP_CONTENT) {
            resultSize = size;
            resultMode = MeasureSpec.AT_MOST;
        }
        break;

    // 父View是UNSPECIFIED的
    case MeasureSpec.UNSPECIFIED:
             //子View的width或height是个精确值,则size为精确值，mode为 EXACTLY
        if (childDimension >= 0) {
            resultSize = childDimension;
            resultMode = MeasureSpec.EXACTLY;
        } else if (childDimension == LayoutParams.MATCH_PARENT) {
            //因父布局没有对子View做出限制，当子View为MATCH_PARENT时则大小为0
            resultSize = View.sUseZeroUnspecifiedMeasureSpec ? 0 : size;
            resultMode = MeasureSpec.UNSPECIFIED;
        } else if (childDimension == LayoutParams.WRAP_CONTENT) {
            //因父布局没有对子View做出限制，当子View为WRAP_CONTENT时则大小为0
            resultSize = View.sUseZeroUnspecifiedMeasureSpec ? 0 : size;
            resultMode = MeasureSpec.UNSPECIFIED;
        }
        break;
    }
    return MeasureSpec.makeMeasureSpec(resultSize, resultMode);
}
```

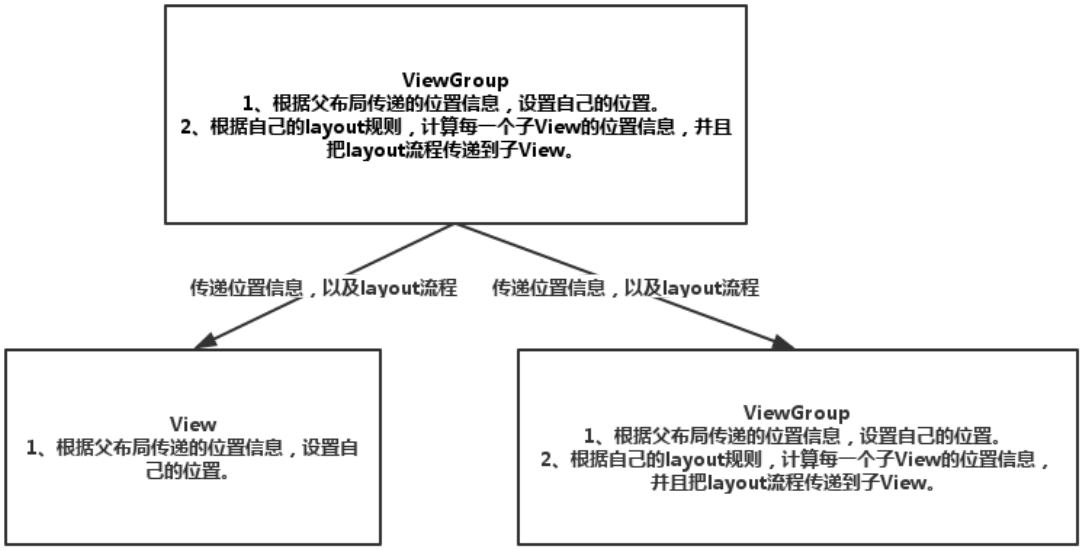
layout()

由于measure()我们已经测量出view 的大小了，然后就要进行view位置的排版布局。

layout()之前，会有一个判断动画的过程，毕竟如果view在执行动画的操作，是不能够进行定位具体位置的。

在view中是layout()空方法的，不需要实现

在viewgroup中是必须实现的，对子view进行排版布局，当然如果子view很多，还是和上面的viewgroup的measure()一样先得到子view的数量，在进行排版布局。



draw()

1.绘制背景

2.绘制自身内容（onDraw）

3.遍历子View，调用其draw方法把绘制过程分发下去（dispatchDraw）

4.绘制装饰（onDrawForeground)

Android屏幕的刷新机制

由于人眼与大脑之间的协作一般情况无法感知超过60FPS的画面更新。如果所看到画面的帧率高于12帧的时候，就会认为是连贯的，达到24帧便是流畅的体验，这也就是胶片电影的播放速度（24FPS）

Android 正常情况下屏幕刷新频率也是60FPS，也就是每秒更新60次，16.67ms

所以在Android中，当我们谈到 **布局优化**、**卡顿优化** 时，通常都知道 需要减少布局层级、减少主线程耗时操作，这样可以减少**丢帧**。如果丢帧比较严重，那么界面可能会有明显的卡顿感。

> 如果界面一直保持没有改变，那么屏幕的刷新又是怎么样的呢?
>
> 我们常说的Android每隔16.6 ms刷新一次屏幕其实是指底层会以这个固定频率来切换每一帧的画面，而这个每一帧的画面数据就是我们app在接收到屏幕刷新信号之后去执行遍历绘制View树工作所计算出来的屏幕数据。而 app 并不是每隔16.6ms 的屏幕刷新信号都可以接收到，只有当app向底层注册订阅一个屏幕刷新信号之后，才能接收到下一个屏幕刷新信号到来的通知。
>
> 而只有当某个View发起了刷新请求时，app才会去向底层注册订阅一个屏幕刷新信号。
>
> 也就是说，只有当界面有刷新的需要时，我们app才会在订阅一个屏幕刷新信号来时，遍历绘制View树来重新计算屏幕数据，在屏幕刷新信号来时，才会更新屏幕信息。如果界面没有刷新的需要，一直保持不变时，我们app就不会去接收每隔16.6ms 的屏幕刷新信号事件了，但底层仍然会以这个固定频率来切换每一帧的画面，只是后面这些帧的画面都是相同的而已。

Android动画（简单的提一下）

视图动画：补间动画（可以写在xml文件中，也可以代码里），逐帧动画

属性动画：valueAnimator,objectAnimator

视图动画与属性动画的区别：

> 视图动画只是对view进行了简单的平移，拉伸，旋转，调透明度等操作。视图动画利用动画改变了view的位置和大小，但view真正的属性没有改变，这就会导致许多问题，视图动画过后我们再去操作view则得不到响应。而属性动画是对view 的属性进行操作，从根本上改变了view的属性，并且属性动画还支持监听动画过程，在动画过程中自己操作控件进行改变。

补间动画分为四种形式

- alpha（淡入淡出）
- translate（位移）
- scale（缩放大小）
- rotate（旋转）

补间动画的属性值

xml属性	java方法	解释
android:detachWallpaper	setDetachWallpaper(boolean)	是否在壁纸上运行
android:duration	setDuration(long)	动画持续时间，毫秒为单位
android:fillAfter	setFillAfter(boolean)	控件动画结束时是否保持动画最后的状态
android:fillBefore	setFillBefore(boolean)	控件动画结束时是否还原到开始动画前的状态
android:fillEnabled	setFillEnabled(boolean)	与android:fillBefore效果相同
android:interpolator	setInterpolator(Interpolator)	设定插值器（指定的动画效果，譬如回弹等）
android:repeatCount	setRepeatCount(int)	重复次数
android:repeatMode	setRepeatMode(int)	重复类型有两个值，reverse表示倒序回放，restart表示从头播放
android:startOffset	setStartOffset(long)	调用start函数之后等待开始运行的时间，单位为毫秒
android:zAdjustment	setZAdjustment(int)	表示被设置动画的内容运行时在Z轴上的位置（top/bottom/normal），默认为normal

属性动画常见的属性值

- Duration：动画的持续时间；
- TimeInterpolation：定义动画变化速率的接口，所有插值器都必须实现此接口，如线性、非线性插值器；
- TypeEvaluator：用于定义属性值计算方式的接口，有int、float、color类型，根据属性的起始、结束值和插值一起计算出当前时间的属性值；
- Animation sets：动画集合，即可以同时对一个对象应用多个动画，这些动画可以同时播放也可以对不同动画设置不同的延迟；
- Frame refreash delay：多少时间刷新一次，即每隔多少时间计算一次属性值，默认为10ms，最终刷新时间还受系统进程调度与硬件的影响；
- Repeat Country and behavoir：重复次数与方式，如播放3次、5次、无限循环，可以让此动画一直重复，或播放完时向反向播放；