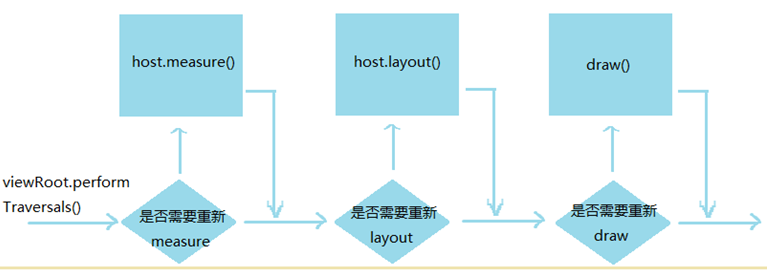
1. **概述**

**视图的绘制过程**：每一个视图的绘制过程都必须经历三个最主要的阶段，即onMeasure()、onLayout()和onDraw()。



1. **onMeasure()：调用该方法其实是在ViewRoot内部调用View的host.measure()方法。**

measure()方法接受两个参数widthMeasureSpec和heightMeasureSpec，这两个值分别用于确定视图的高度和宽度的规格和大小。

其中的MeasureSpec由specSize和specMode共同组成，其中specSize记录的是大小，specMode记录的是规格。

* 1. **specMode包括3种类型**

1. **EXACTLY**：表示父视图希望子视图的大小应该是由specSize的值来决定的，系统默认会按照这个规则来设置子视图的大小；开发人员当然也可以按照自己的意愿设置成任意的大小。一般设置了明确的值或者是MATCH\_PARENT。
2. **AT\_MOST**：表示子视图最多只能是specSize中指定的大小，开发人员应该尽可能小得去设置这个视图，并且保证不会超过specSize。即表示子布局限制在一个最大值内，一般为WRAP\_CONTENT。
3. **UNSPECIFIED**：表示开发人员可以将视图按照自己的意愿设置成任意的大小，没有任何限制。（不太会用到）
   1. **widthMeasureSpec和heightMeasureSpec的值的由来**

通常情况下，这两个值都是由父视图经过计算后传递给子视图的，说明父视图会在一定程度上决定子视图的大小。

最外层的根视图ViewRoot的这两个值是通过**getRootMeasureSpec()**方法去获取widthMeasureSpec和heightMeasureSpec的值，注意方法中传入的参数，其中lp.width和lp.height在创建ViewGroup实例的时候就被赋值了，它们都等于MATCH\_PARENT。

在getRootMeaSureSpec()中用了**MeasureSpec.makeMeasureSpec()**方法来组装一个**MeasureSpec**，设置它的**specMode**和**specSize**。

* 1. **measure()方法**

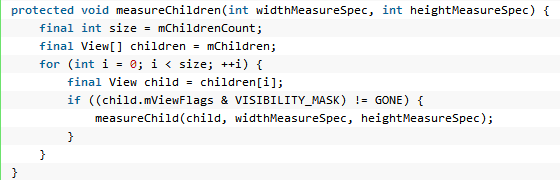
注意：measure()这个方法是final的，因此我们无法在子类中去重写这个方法，说明Android是不允许我们改变View的measure框架的。

在该方法中会调用onMeasure(widthMeasureSpec, heightMeasureSpec)这才是***真正***的去测量并设置View大小的地方。该方法默认会调用**getDefaultSize**(int size, int **measureSpec**)方法来获取视图的大小。

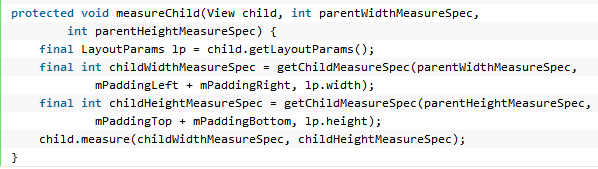
**measureSpec**是从measure()方法中传递过来的。然后调用MeasureSpec.getMode()方法可以解析出specMode，调用MeasureSpec.getSize()方法可以解析出specSize。接下来进行判断，如果specMode等于AT\_MOST或EXACTLY就返回specSize，这也是系统默认的行为。之后会在onMeasure()方法中调用setMeasuredDimension()方法来设定测量出的大小，这样一次measure过程就结束了。

* 1. **对子视图进行测量**

在一个布局中会有很多子视图，每个视图都需要进行一次measure过程，ViewGroup定义了一个measureChildren()方法来去测量子视图的大小。

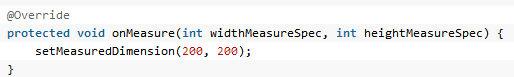


在代码中可以看出，首先会去遍历当前布局下所有的子视图，然后逐个调用measureChild()方法去测量相应的子视图大小。



其中调用了getChildMeasureSpec()方法来去计算子视图的MeasureSpec，计算的依据就是布局文件中定义的MATCH\_PARENT、WRAP\_CONTENT等值，这个方法的内部细节就不再贴出。然后调用子视图的**measure(**)方法，并把计算出的**MeasureSpec**传递进去，之后的流程就和前面所介绍的一样。

* 1. **重写onMeasure()方法**



如果你不想使用系统默认的测量方式，可以按照自己的意愿进行定制。把View默认的测量流程覆盖掉了，不管在布局文件中定义MyView这个视图的大小是多少，最终在界面上显示的大小都将会是200\*200。

**注意：**在setMeasuredDimension()方法调用之后，我们才能使用getMeasuredWidth()和getMeasuredHeight()来获取视图测量出的宽高，以此之前调用这两个方法得到的值都会是0。

1. **onLayout()方法**

measure过程结束后，视图的大小就已经测量好了，接下来就是layout的过程，即给视图进行布局，也就是确定视图的位置。ViewRoot的performTraversals()方法会在measure结束后继续执行，并调用View的**host.layout()**方法来执行此过程。

host.layout(0, 0, host.mMeasuredWidth, host.mMeasuredHeight);

layout()方法接收四个参数，分别代表着左、上、右、下的坐标，当然这个坐标是相对于当前视图的父视图而言的。将测量好的宽度和高度传入。

在layout方法中会通过调用**setFrame**()方法来判断视图的大小是否发生过变化，以确定有没有必要对当前的视图进行重绘，同时还会在这里把传递过来的四个参数分别赋值给mLeft、mTop、mRight和mBottom这几个变量。接着就会调用**onLayout**()方法，但是onLayout()方法是一个空方法，这是因为onLayout方法是为了确定视图在布局中所在的位置，而这个操作应该是由布局来完成的，即父视图决定子视图的显示位置。

所以需要看一下ViewGroup中的onLayout方法：

**protected** **abstract** **void** onLayout(**boolean** changed, **int** l, **int** t, **int** r, **int** b);

是一个抽象方法，所以需要所有的ViewGroup的子类都必须重写这个方法。

在**onLayout()**过程结束后，我们就可以调用getWidth()方法和getHeight()方法来获取视图的宽高了。

注意：getWidth()方法和getMeasureWidth()方法到底有什么区别呢？

首先getMeasureWidth()方法在measure()过程结束后就可以获取到了，而getWidth()方法要在layout()过程结束后才能获取到。另外，getMeasureWidth()方法中的值是通过setMeasuredDimension()方法来进行设置的，而getWidth()方法中的值则是通过视图右边的坐标减去左边的坐标计算出来的。

1. **onDraw()方法：开始对视图进行绘制**

ViewRoot中的代码会继续执行并创建出一个Canvas对象，然后调用View的draw()方法来执行具体的绘制工作。draw()方法内部的绘制过程总共可以分为六步（第二步和第五步都不常用）。

3.1 第一步：这一步的作用是对视图的背景进行绘制。

3.2 第二步：

3.3 第三步：这一步的作用是对视图的内容进行绘制。可以看到，这里去调用了一下onDraw()方法，那么onDraw()方法里又写了什么代码呢？进去一看你会发现，原来又是个空方法啊。其实也可以理解，因为每个视图的内容部分肯定都是各不相同的，这部分的功能交给子类来去实现也是理所当然的。

3.4 第四步：这一步的作用是对当前视图的所有子视图进行绘制。但如果当前的视图没有子视图，那么也就不需要进行绘制了。因此你会发现View中的dispatchDraw()方法又是一个空方法，而ViewGroup的dispatchDraw()方法中就会有具体的绘制代码。

3.5 第五步：

3.6 第六步：这一步的作用是对视图的滚动条进行绘制。那么你可能会奇怪，当前的视图又不一定是ListView或者ScrollView，为什么要绘制滚动条呢？其实不管是Button也好，TextView也好，任何一个视图都是有滚动条的，只是一般情况下我们都没有让它显示出来而已。

总结：View是不会帮我们绘制内容部分的，因此需要每个视图根据想要展示的内容来自行绘制。

1. （文本View）自定义View，即继承View，并重新按照自己的设计来定义View控件

主要步骤：

1、自定义View的属性

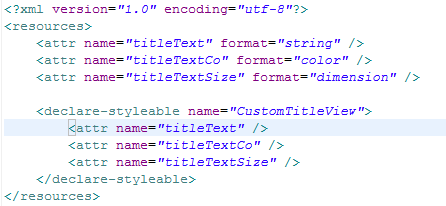
2、在View的构造方法中获得我们自定义的属性

[ 3、重写onMesure ]

4、重写onDraw

3用[]标出了，所以说3不一定是必须的，当然了大部分情况下还是需要重写的。

（1）首先在res/value目录下创建一个attrs.xml文件，里面用于存放自定义控件的一些属性。



（2）创建自定义View类，继承View，实现三个构造函数（注意三个构造函数之间的关系），在三个参数的构造函数中获取自定义属性值，并初始化。

a.获取自定义attrs.xml文件中的所有属性

TypedArray a = context.getTheme().obtainStyledAttributes(attrs,

R.styleable.*CustomTitleView*, defStyleAttr, 0);

int count = a.getIndexCount();//获取属性的个数

注意：属性的个数不一定等于自定义的attrs.xml中的属性个数，而是等于主布局文件中所用到的属性个数

b.根据这个属性集，分别获取不同的属性，再对其做不同的处理。

（3）重写onMeasure()（可直接super父类）

（4）重写onDraw(Canvas)给View绘制背景和添加文本等等。

（5）在主布局文件中添加自定义的View控件，

1.在布局标签属性（如LinearLayout）中添加：xmlns:custom="http://schemas.android.com/apk/res/*com.lhqj.myview*"

a.第一部分是自己定义的（名字任何都可以）

b.第二部分是本应用的包名（在AndroidManifest文件中）

2. <com.lhqj.myview.DefineView

android:layout\_width=*"wrap\_content"*

android:layout\_height=*"wrap\_content"*

custom:titleText=*"3712"*

android:padding=*"10dp"*

custom:titleTextCo=*"#ff0000"*

custom:titleTextSize=*"40sp"*

android:layout\_centerInParent=*"true"*

></com.lhqj.myview.DefineView>

要设置自定义属性的值，则需要用custom:属性 = 属性值。

若不重写onMeasure()方法，则会出现一个问题：

系统帮我们测量的高度和宽度都是MATCH\_PARNET，当我们设置明确的宽度和高度时，系统帮我们测量的结果就是我们设置的结果，当我们设置为WRAP\_CONTENT,或者MATCH\_PARENT系统帮我们测量的结果就是MATCH\_PARENT的长度。所以，当设置了WRAP\_CONTENT时，我们需要自己进行测量，即重写onMesure方法。

先了解MeasureSpec的specMode,一共三种类型：

**EXACTLY**：一般是设置了明确的值或者是MATCH\_PARENT

**AT\_MOST**：表示子布局限制在一个最大值内，一般为WARP\_CONTENT

**UNSPECIFIED**：表示子布局想要多大就多大，很少使用

@Override

**protected** **void** onMeasure(**int** widthMeasureSpec, **int** heightMeasureSpec)

我们根据specMode来自定义设计View的长宽。

最后setMeasuredDimension(width, height); //在布局中重新测量空间的尺寸

1. **一些小问题**

在自定义控件时，在计算控件的大小时，getWidth,getLeft,getRight,getTop,getBottom等这些值的大小是一个相对值，是相等于View的父布局（上一个布局（有点像FrameLayout）），具体分析如下：

1. 坐标

坐标系在二维视图中通过X轴和Y轴两个数字为组合表示某个点的绝对坐标。 例如(30, 100) 通常表示X轴30， Y轴100交叉的一个点

在Android中可以把left相当于X轴值， top相当于Y轴值， 通过这两个值Android系统可以知道视图的绘制起点，在通过Wdith 和 Height 可以得到视图上下左右具体值，就可以在屏幕上绝对位置绘制视图。right 与 bottom计算如下：

**right = left + width;**

**bottom = top + height;**

1. 常用的函数API

视图左侧位置 view.getLeft()

视图右侧位置 view.getRight()

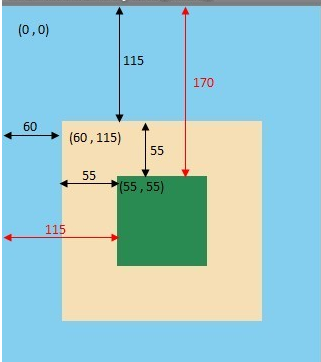
视图顶部位置 view.getTop();

视图底部位置 view.getBottom();

视图宽度 view.getWidth();

视图高度 view.getHeight()

1. 分析



蓝色区域位置 left = 0，   top = 0      坐标(0, 0 )

黄色区域位置 left = 60， top = 115    坐标(60, 115)

绿色区域位置 left = 115， top = 170  坐标(115, 170)

  绿色区域，这里理解错误，我认为绿色区域的位置是针对于蓝色区域的(0, 0)坐标的值，从上图的右下角打印出的坐标值就可以看出与下方我列出的值不一致，看看下面的图就明白了

实际上🡺视图的left ， top ， right ， bottom 的值是针对其父视图的相对位置， 绿色区域是针对其父视图(即黄色区域为(0, 0)点)的坐标，不应该是(115, 170 ) 而是 (55, 55)