Android 自定义 view 详解

2016-07-19 伯乐专栏/邵辉 安卓应用频道

(点击上方公众号，可快速关注)

> 来源：伯乐在线专栏作者 - 邵辉|CRR
> 链接：http://android.jobbole.com/83835/
> 点击 → 了解如何加入专栏作者

对于我这样一个Android初级开发者来说，自定义View一直是一个遥不可及的东西，每次看到别人做的特别漂亮的控件，自己心里那个痒痒啊，可是又生性懒惰，自己不肯努力去看书，只能望而兴叹，每次做需求用到自定义控件，就直接去Github上找，找到合适的就用，找不到合适的，凑合也用，反正从来没想过要自己来做这样的东西，可是毕业以后到了新公司，为了自己的荣誉，这次不得不硬着头皮自己来了，一个月的紧张开发过后，回头再看，自定义控件也无非那么回事，只要记得几个要领，几乎是手到擒来。

从继承开始

懂点面向对象语言知识的都知道：封装，继承和多态，这是面向对象的三个基本特征，所以在自定义View的时候，最简单的方法就是继承现有的View

```
public class SketchView extends View{

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

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

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

通过上面这段代码，我定义了一个SketchView，继承自View对象，并且复写了它的三个构造方法，我主要来分析一下这三个构造方法：

- 第一个构造方法就是我们普通在代码中新建一个view用到的方法，例如

```
SketchView sketchView = new SketchView(this);
```

就这样，一个自定义的view就被新建出来了，然后可以根据需求添加到布局里

- 第二个构造方法就是我们一般在xml文件里添加一个view

```
<me.shaohui.androidpractise.widget.SketchView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_marginRight="16dp"
    android:layout_marginTop="16dp" />
```

- 这样，我们就把一个SketchView添加到布局文件里，并且加了一些布局属性，宽高属性以及margin属性，这些属性会存放在第二个构造函数的AttributeSet参数里
- 第三个构造函数比第二个构造函数多了一个int型的值，名字叫defStyleAttr，从名称上判断，这是一个关于自定义属性的参数，实际上我们的猜测也是正确的，第三个构造函数不会被系统默认调用，而是需要我们自己去显式调用，比如在第二个构造函数里调用调用第三个函数，并将第三个参数设为0。

> 关于第三个参数defStyleAttr,其实也可以拿出来说一整篇文章，有想详细了解的读者可以去看下本篇文章最后的第三个参考链接，我在这里只是简单的说一下：defStyleAttr指定的是在Theme style定义的一个attr，它的类型是reference,主要生效在obtainStyledAttributes方法里，obtainStyledAttributes方法有四个参数，第三个参数是defStyleAttr，第四个参数是自己指定的一个style，当且仅当defStyleAttr为0或者在Theme中找不到defStyleAttr指定的属性时，第四个参数才会生效，这些指的都是默认属性，当在xml里面定义的，就以在xml文件里指定的为准，所以优先级大概是：xml>style>defStyleAttr>defStyleRes>Theme指定，当defStyleAttr为0时，就跳过defStyleAttr指定的reference，所以一般用0就能满足一些基本开发。

Measure->Layout->Draw

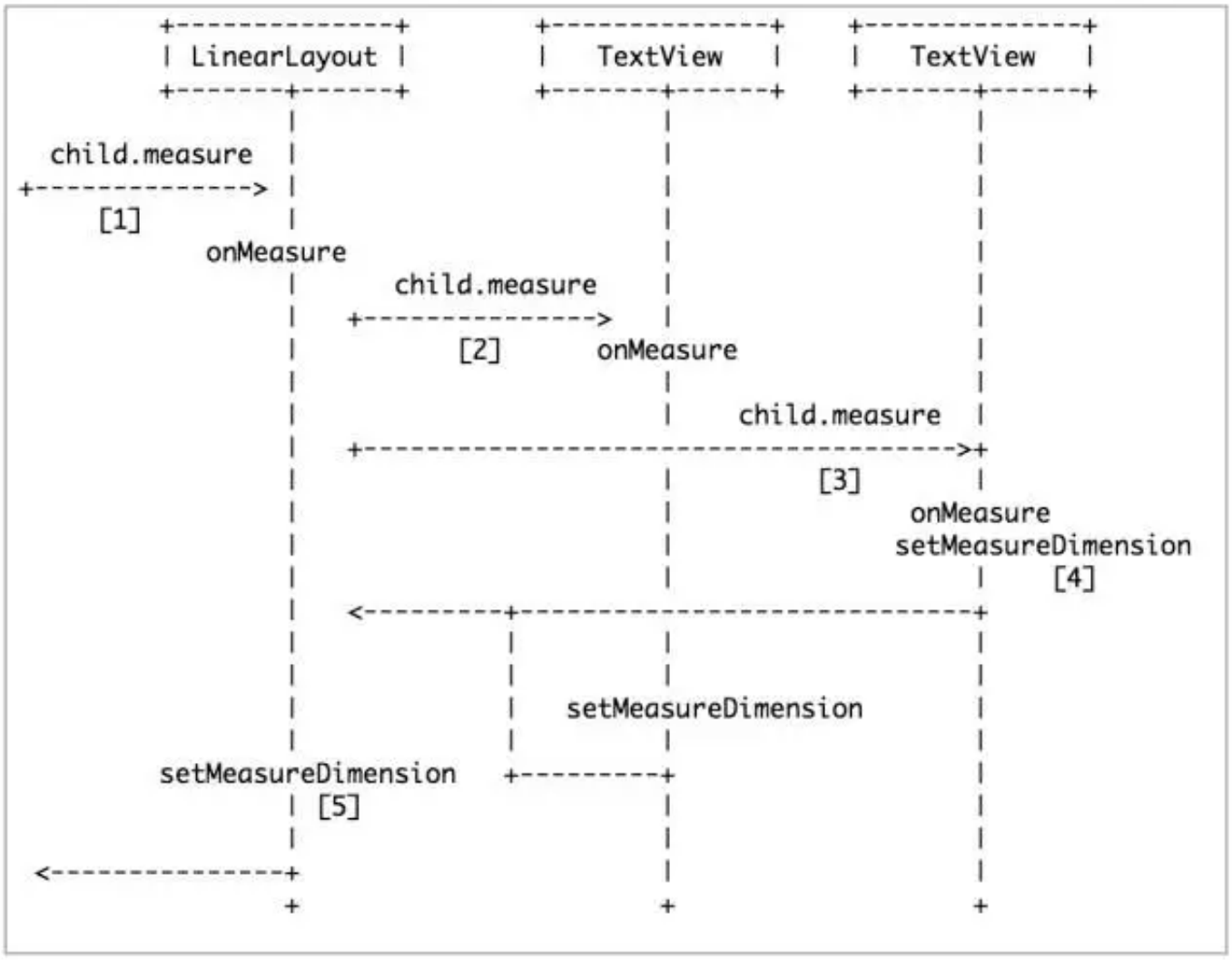
在学会如何写一个自定义控件之前，了解一个控件的绘制流程是必要的，在Android里，一个view的绘制流程包括：Measure，Layout和Draw，通过onMeasure知道一个view要占界面的大小，然后通过onLayout知道这个控件应该放在哪个位置，最后通过onDraw方法

将这个控件绘制出来，然后才能展现在用户面前，下面我将挨个分析一下这三个方法的作用。

- onMeasure 测量，通过测量知道一个一个view要占的大小，方法参数是两个int型的值，我们都知道，在java中，int型由4个字节（32bit）组成，在MeasureSpce中，用前两位表示mode，用后30位表示size

```
@Override
protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
    int widthMode = MeasureSpec.getMode(widthMeasureSpec);
    int widthSize = MeasureSpec.getSize(widthMeasureSpec);

    int heightMode = MeasureSpec.getMode(heightMeasureSpec);
    int heightSize = MeasureSpec.getSize(heightMeasureSpec);

    int measuredHeight, measuredWidth;

    if (widthMode == MeasureSpec.EXACTLY) {
        measuredWidth = widthSize;
    } else {
        measuredWidth = SIZE;
    }

    if (heightMode == MeasureSpec.EXACTLY) {
        measuredHeight = heightSize;
    } else {
        measuredHeight = SIZE;
    }

    setMeasuredDimension(measuredWidth, measuredHeight);
}
```

MeasureSpce的mode有三种：EXACTLY, AT_MOST，UNSPECIFIED，除却UNSPECIFIED不谈，其他两种mode：当父布局是EXACTLY时，子控件确定大小或者match_parent，mode都是EXACTLY，子控件是wrap_content时，mode为AT_MOST；当父布局是AT_MOST时，子控件确定大小，mode为EXACTLY，子控件wrap_content或者match_parent时，mode为AT_MOST。所以在确定控件大小时，需要判断MeasureSpec的mode，不能直接用MeasureSpec的size。在进行一些逻辑处理以后，调用setMeasureDimension()方法，将测量得到的宽高传进去供layout使用。



> 需要明白的一点是,测量所得的宽高不一定是最后展示的宽高，最后宽高确定是在onLayout方法里，layou（left，top，right，bottom），不过一般都是一样的。

- onLayout 实际上，我在自定义SketchView的时候是没有重写onLayout方法的，因为SketchView只是一个单纯的view，它不是一个view容器，没有子view，而onLayout方法里主要是具体摆放子view的位置，水平摆放或者垂直摆放，所以在单纯的自定义view是不需要重写onLayout方法，不过需要注意的一点是，子view的margin属性是否生效就要看parent是否在自身的onLayout方法进行处理，而view得padding属性是在onDraw方法中生效的。

其实在onLayout方法里有一个属性我一直关注并且没有弄得很明白，就是第一个参数boolean:changed，标示这个view的大小是否发生改变，后续了解到，会回来补坑。

- onDraw 终于说到了重头戏，一般自定义控件耗费心思最多的就是这个方法了，需要在这个方法里，用Paint在Canvas上画出你想要的图案，这样一个自定义view才算结束。下面会详细讲如何在画布上画出自己想要的图案。

> 关于onDraw方法，在补充一句，如果是直接继承的View，那么在重写onDraw的方法是时候完全可以把super.ondraw(canvas)删掉，因为它的默认实现是空。

得到一个正方形的View

上一部分主要说了一下，view的绘制流程，从这三个方法中，我们可以知道如何测量一个控件，如何摆放控件的子元素，如何绘制图案，下面我说一下自己通过onMeasure学到的一点小技巧。

在日常开发中，我们偶尔会需要一个正方形的imageView，一般都是通过指定宽高，但是当宽高不确定时，我们就只能寄希望于Android原声支持定义view的比例，但是现实是残酷的，系统好像是没有提供类似属性的，所以我们就只能自己去实现，其实自己写起来也特别的简单，只需要改一个参数就OK了，

```
@Override
  protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
     super.onMeasure(widthMeasureSpec, widthMeasureSpec);
  }
```

不仔细观察是看不出来其中的奥妙的，虽然这里复写了view的onMeasure，但是貌似没有做任何处理，直接调用了super方法，但是仔细观察的话就会发现，在调用super方法的时候，第二个参数变了，本来应该是heightMeasureSpec却换成了widthMeasureSpec，这样view的高度就是view的宽度，一个SquareView就实现了，甚至如果通过自定义属性实现一个自定义比例view。

自定义属性

自定义view没有自定义属性怎么得了，要给view支持自定义属性，需要在values/attrs.xml文件里定义一个name为自己定义view名字的declare-styleable

```
<resources>
   <declare-styleable name="SketchView">
      <attr name="background_color" format="color"/>
      <attr name="size" format="dimension"/>
   </declare-styleable>
</resources>
```

这样就可以在xml文件里使用自己定义的属性了

```
<me.shaohui.androidpractise.widget.SketchView
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_marginRight="16dp"
    android:layout_marginTop="16dp"
    app:background_color="@color/colorPrimary"
    app:size="24dp"/>
```

别忘了在前面加上自定义的命名空间，到这来看，增加自定义属性并不算什么，不过要自定义属性生效还是要耗费一些功夫的，这时候前面留下的伏笔：第三个构造方法的defStyleAttr参数就要登场了。

```
public SketchView(Context context, AttributeSet attrs, int defStyleAttr) {
    super(context, attrs, defStyleAttr);

    TypedArray a = context.obtainStyledAttributes(attrs, R.styleable.SketchView, defStyleAttr,
R.style.AppTheme);

    custom_size = a.getDimensionPixelSize(R.styleable.SketchView_size, SIZE);
    custon_background = a.getColor(R.styleable.SketchView_background_color, DEFAULT_COLOR);

    a.recycle();
  }
```

经过好一番操作，才能把xml定义的属性拿出来，具体取得的值为多少，我在前面已经解释过，就不在这里多说，接下来的操作就是拿着这些属性干你想干的事吧。

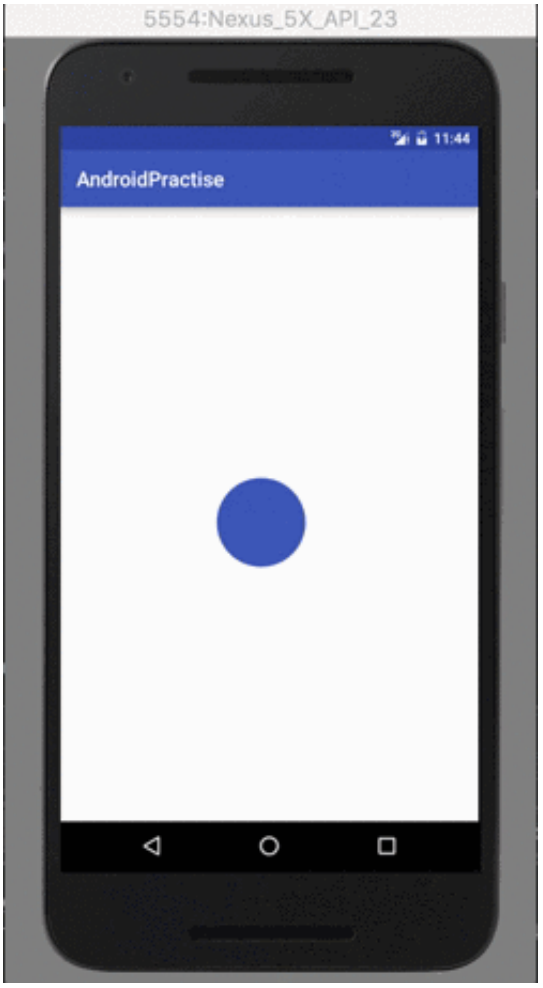
实战：一个动态view

下面将简单介绍一下如何在onDraw(Canvas canvas) 在画布的中心位置画一个自定义颜色的圆，并且通过一个ValueAnimator让这个圆动起来，废话不多说，直接上代码：

```
@Override
  protected void onDraw(Canvas canvas) {

    canvas.drawCircle(mWidth/2, mHeight/2, custom_size * scale, mPaint);

  }
```

```
private ValueAnimator mAnimator;

public void startAnimation() {
    mAnimator = ValueAnimator.ofFloat(1, 2);
    mAnimator.addUpdateListener(new ValueAnimator.AnimatorUpdateListener() {
        @Override
        public void onAnimationUpdate(ValueAnimator animation) {
            scale = (float) animation.getAnimatedValue();
            postInvalidate();
        }
    });

    // 重复次数 -1表示无限循环
    mAnimator.setRepeatCount(-1);

    // 重复模式, RESTART: 重新开始 REVERSE:恢复初始状态再开始
    mAnimator.setRepeatMode(ValueAnimator.REVERSE);

    mAnimator.start();
}
```

(只贴了核心代码，完整代码会在文章最后给链接)
可以看到在onDraw()方法里，我调用了canvas的drawCircle方法画了一个圆，圆心的位置是又画布的位置决定的，位于画布的中心，width和height参数是在onLayout()方法里拿到的，在此之前取到的height和width都是不准确的，这点要注意。圆的半径是xml文件中定义的size*scale，而这个scale是通过一个ValueAnimator确定的，变化范围是从1到2，ValueAnimator的值发生改变会赋给scale同时调用postInvalidate()方法，这个方法的作用就是重绘，然后圆的半径就会发生改变，这样刷新就会实现动画的效果。



requstLayout和invidious

在自定义view时，时常用到刷新view的方法，这时候就会有三个方法供我们选择：requestLayout()、invalidate()、postInvalidate()，其实invalidate和postInvalidate这两个方法作用是一样的，唯一不同的是invalidate用在主线程，而postInvalidate用在异步线程，下面对比一下requestLayout和invalidate的内部实现：

```
@Override
public void requestLayout() {
    if (!mHandlingLayoutInLayoutRequest) {
        checkThread();
        mLayoutRequested = true;
        scheduleTraversals();
    }
}
void invalidate() {
    mDirty.set(0, 0, mWidth, mHeight);
    if (!mWillDrawSoon) {
        scheduleTraversals();
```

```
    }
}
```

从代码可以看出，其实这两个方法内部都是调用的scheduleTraversals()方法，不同的是，requestLayout方法将mLayoutRequested标示置为true，scheduleTraversals这个方法以后找机会再细分析，现在只简单说下结论，

- requestLayout会调用measure和layout 等一系列操作，然后根据布局是否发生改变，surface是否被销毁，来决定是否调用draw，也就是说requestlayout肯定会调用measure和layout，但不一定调用draw，读者可以试着改下我上面写的那个小程序，将postInvalidate改成requestlayout，动画效果就消失了，因为布局没有发生改变。
- invalidate 只会调用draw，而且肯定会调，即使什么都没有发生改变，它也会重新绘制。

所以如果有布局需要发生改变，需要调用requestlayout方法，如果只是刷新动画，则只需要调用invalidate方法。

自定义view的状态保存

自定义view的状态保存和Activity的状态保存是类似的，都是在onSaveInstanceState()保存，然后在onRestoreInstanceState将数据安全取出，之所以在这是还是多说一嘴，主要是怕自己忘，给自己提个醒，也顺便给各位看客叨扰几句，还有一个就是，如果一个view没有id，这个view的状态是不会保存的。

事件处理onTouchEvent

Android的事件处理太过复杂，我会在以后另起一篇文章来好好聊一下Android里的事件处理。

In The End

自定义view其实是一个很重的知识点，里面包含了很多view的知识，不是一篇文章就能聊完的，但我又不习惯连载，有什么话总喜欢一口气说完，所以都写在这一篇文章里了，我这里只是简单的和大家开个头，具体深知里面的细节还是要看很多东西。文章里有什么不对的地方，望各位指出。

源码地址：Github地址
https://github.com/shaohui10086/AndroidPractise/blob/master/app/src/main/java/me/shaohui/androidpractise/widget/SketchView.java

参考链接

- http://ghui.me/post/2015/10/view-measure/
- http://blog.csdn.net/wzy_1988/article/details/49619773
- http://www.cnblogs.com/angeldevil/p/3479431.html

阅读原文