自定义View基础一

> 前言

查看Android总结专题

自定义View总结：

- View基础
- measure方法
- layout方法
- draw方法
- Path类
- Canvas类

1.视图（View）定义

视图（`View`）表现为显示在屏幕上的各种视图，如`TextView`、`LinearLayout`等。

2.视图分类

- 视图`View`主要分两类：

类别	解释	
单一视图	即一个View，如TextView	不包
视图组	即多个View组成的ViewGroup,LinearLayout	包含

- Android 中的UI组件都由`View`、`ViewGroup`组成。

3.View类简介

- `View`类是`Android`中各种组件的基类，如`View`是`ViewGroup`基类。
- `View`的构造函数：共有4个，具体如下：

```
// 如果View是在Java代码里面new的，则调用第一个构造函数
 public CarsonView(Context context) {
        super(context);
    }

// 如果View是在.xml里声明的，则调用第二个构造函数
// 自定义属性是从AttributeSet参数传进来的
    public  CarsonView(Context context, AttributeSet attrs) {
        super(context, attrs);
    }

// 不会自动调用
// 一般是在第二个构造函数里主动调用
// 如View有style属性时
    public  CarsonView(Context context, AttributeSet attrs, int defStyleAttr) {
        super(context, attrs, defStyleAttr);
    }

    //API21之后才使用
    // 不会自动调用
    // 一般是在第二个构造函数里主动调用
    // 如View有style属性时
    public  CarsonView(Context context, AttributeSet attrs, int defStyleAttr, int defSt
        super(context, attrs, defStyleAttr, defStyleRes);
    }
```

自定义`View`必须重写至少一个构造函数。

> 构造方法参数

构造函数参数最多有四个：

- `Context`
- `AttributeSet`——XML属性（当从XML inflate的时候）
- `int defStyleAttr`——应用到`View`的默认风格（定义在主题中）
- `int defStyleResource`——如果没有使用`defstyleAttr`,应用到`View`的默认风格

除了`Context`，其它参数只是用来通过XML属性配置`View`的初始状态（从布局，`style`以及`theme`中）。

> 属性

让我们从如何定义XML属性开始讨论。这里是一个XML中最基本的`ImageView`：

```
<ImageView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:src="@drawable/icon"
  />
```

你有没有想过这里的`layout_width`，`layout_height`以及`src`是从哪里来的？是通过`<declare-styleable>`把这些属性明确的声明为系统需要处理的东西。比如，`src`就是在这里定义的：

```
<declare-styleable name="ImageView">
  <!-- Sets a drawable as the content of this ImageView. -->
  <attr name="src" format="reference|color" />
  <!-- ...snipped for brevity... -->
</declare-styleable>
```

每个`declare-styleable`产生一个`R.styleable.[name]`，外加每个属性的`R.styleable.[name]_[attribute]`。比如，上面的代码产生`R.styleable.ImageView`和`R.styleable.ImageView_src`。

这些资源是什么东西呢?
`R.styleable.[name]`是所有资源的数组，系统使用它来查找属性值。每个`R.styleable.[name]_[attribute]`只不过是这个数组的索引罢了，所以你可以一次性取出所有属性，然后分别查询每个的值。
如果你把它想象成一个`cursor`，`R.styleable.[name]`就可以看成是一个待查找的`column`列表，而`R.styleable.[name]_[attribute]`就是一个`column`索引。

> AttributeSet

上面写的XML是以一个`AttributeSet`的形式传递给`View`的。
通常不直接使用`AttributeSet`。而是使用`Theme.obtainStyledAttributes()`。这是因为原始的属性通常需要引用和应用样式。比如，如果你在XML中定义了`style="@style/MyStyle"`，这个方法先获取`MyStyle`，然后把它的属性混合进去。最终`obtainStyledAttributes()`返回`TypedArray`,可以用它来获取属性值。
这个过程简化之后就像这样：

```
public ImageView(Context context, AttributeSet attrs) {
  TypedArray ta = context.obtainStyledAttributes(attrs, R.styleable.ImageView, 0, 0);
  Drawable src = ta.getDrawable(R.styleable.ImageView_src);
  setImageDrawable(src);
  ta.recycle();
}
```

这里我们向`obtainStyledAttributes()`传递了两个参数。
第一个参数是`AttributeSet attrs`,即`XML`中的属性，表示从`layout`文件中直接为这个`View`添加的属性集合（一种直接使用`android:layout_width="wrap_content"`这种直接指定，还有一种通过`style="@style/somestyle"`指定）。
第二个参数是`R.styleable.ImageView`数组，它告诉这个方法我们想取哪个属性的值，这里表示要获取`ImageView`属性的值。
当获取了`TypedArray`之后，我们既可以获取单个属性了。我们需要使用`R.styleable.ImageView_src`来正确索引数组中的属性。

> Theme属性

`AttributeSet`并不是`obtainStyledAttribute()`获取属性值的唯一地方。属性也可以存在于主题中。但是它在`View inflate`中的过程中扮演着不重要的角色，以内主题一般不会设置`src`这样的属性，但是如果使用`obtainStyledAttributes()`获取主题属性的话就有作用了。

> Default style Attribute

你可能注意到了`obtainStyledAttributes()`的最后两个参数我是用的值为0。实际上它们是两个资源引用：`defStyleAttr`和`defStyleRes`。这里我们关注的是第一个。
`defStyleAttr`是`obtainStyledAttributes()`最令人困惑的参数。
它是一个为某类型的`View`定义一个基本样式的方法。比如，如果你想一次性修改app中所有`TextView`，你可以在主题中设置`textViewStyle`。如果不存在这个东西的话，你就需要手动为每个`TextView`定义样式了。
下面以`TextView`为例介绍他实际是如何工作的：
首先，它是一个属性（这里是`R.attr.textViewStyle`）。这里是安卓系统`textViewStyle`的地方：

```
<resources>
  <declare-styleable name="Theme">
    <!-- ...snip... -->
    <!-- Default TextView style. -->
    <attr name="textViewStyle" format="reference" />
    <!-- ...etc... -->
  </declare-styleable>
</resource>
```

又一次，我们适用了`declare-styleable`,但是这次是用来定义存在于`theme`中的属性。这里我们说`textViewstyle`是一个`reference`即，它的值只是一个资源的引用，这里，他应该是一个指向`style`的引用。
接下来，我们必须在当前主题设置`textViewstyle`.默认的Android主题如下：

```
<resources>
  <style name="Theme">
```

```
    <!-- ...snip... -->
    <item name="textViewStyle">@style/Widget.TextView</item>
    <!-- ...etc... -->
  </style>
</resource>
```

然后需要为`Appication`或者`Activity`设置这个主题。通常是通过`manifest`设置：

```
<activity
  android:name=".MyActivity"
  android:theme="@style/Theme"
  />
```

现在我们就可以在`obtainStyledAttributes()`里使用它了：

```
TypedArray ta = theme.obtainStyledAttributes(attrs, R.styleable.TextView, R.attr.textVi
```

最终的效果是，任何没有在`AttributeSet`中定义的属性都将用`textViewStyle`引用的样式填充。

> Default Style Resource

`defStyleRes`就要比`defStyleAttr`简单多了它只是一个`style`资源。不需要间接的通过`theme`。
只有在`defstyleAttris`没有定义的情况下，才会使用`style`中的`defStyleRes`（设置为0，或者没有在主题中设置）。

> 优先级

我们现在有了一系列 通过`obtainStyledAttributes()`获取属性值的方式。这里是它们的优先级，从高到低：

1. 定义在`AttributeSet`中的值；
2. 定义在`AttributeSet`中的`style`资源（比如：`style=@style/blah`）
3. `defStyleAttr`指定的默认样式
4. `defStyleResource`指定的默认样式资源（如果没有`defStyleAttr`）
5. 主题中的值

换句话说，任何直接在XML中设置的属性都将首先被使用。但是如果你没有设置，这些属性也可以从其它地方获取。

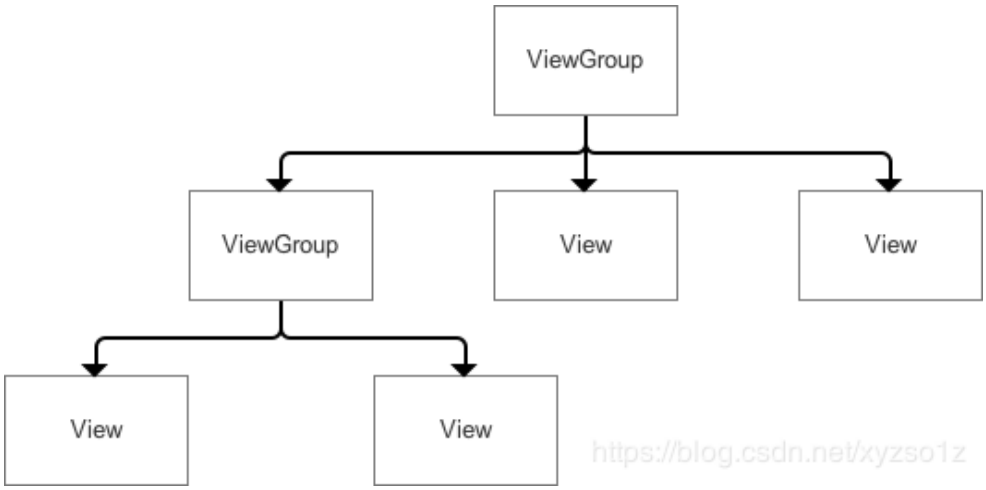
我们一般设置自定义`View`：

```
SomeView(Context context) {
  this(context, null);
}
SomeView(Context context, AttributeSet attrs) {
  // Call super() so that the View sets itself up properly
  super(context, attrs);
  // ...Setup View and handle all attributes here...
}
```

只需要这两个参数的构造方法你就能随意的使用`obtainStyledAttributess()`了。实现默认样式的一个简便方法是直接提供`defStyleRes`给它。那样你就不要忍受`defStyleAttr`的痛苦了。

4.View视图结构

- 对于多View的视图，结构时树形结构：最顶层是`ViewGroup`
- `ViewGroup`下可能有多个`ViewGroup`或`View`,如下图：



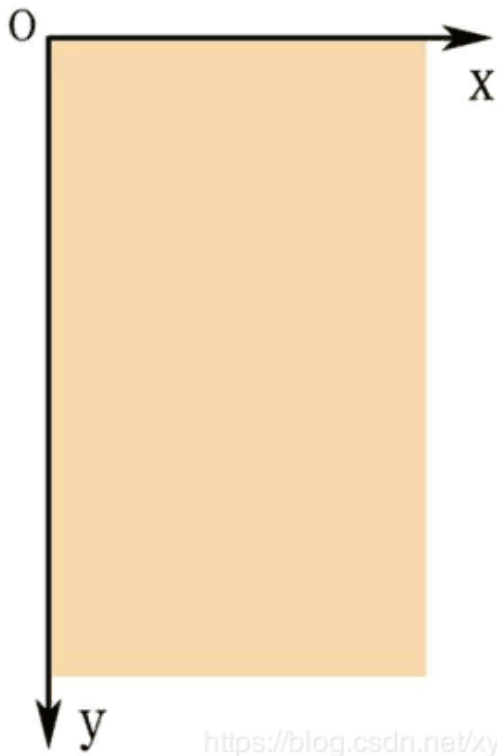
一定要记住：无论是`measure`过程、`layout`过程还是`draw`过程，永远都是从`View树`的根节点开始测量或计算（即从树的顶端开始）,一层一层、一个分支一个分支进行（即树形递归），最终计算整个View树中各个`View`,最终确定整个`View`树的相关属性。

5 Adroid 坐标系

Android的坐标系定义为：

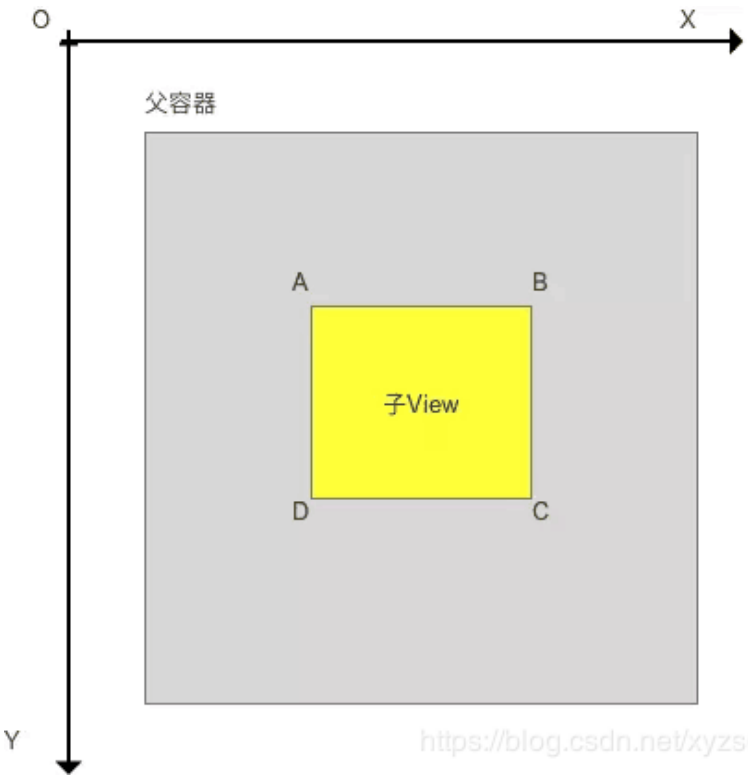
- 屏幕的左上角为坐标原点
- 向右为x轴方向

- 向下为y轴方向
  具体如下图：



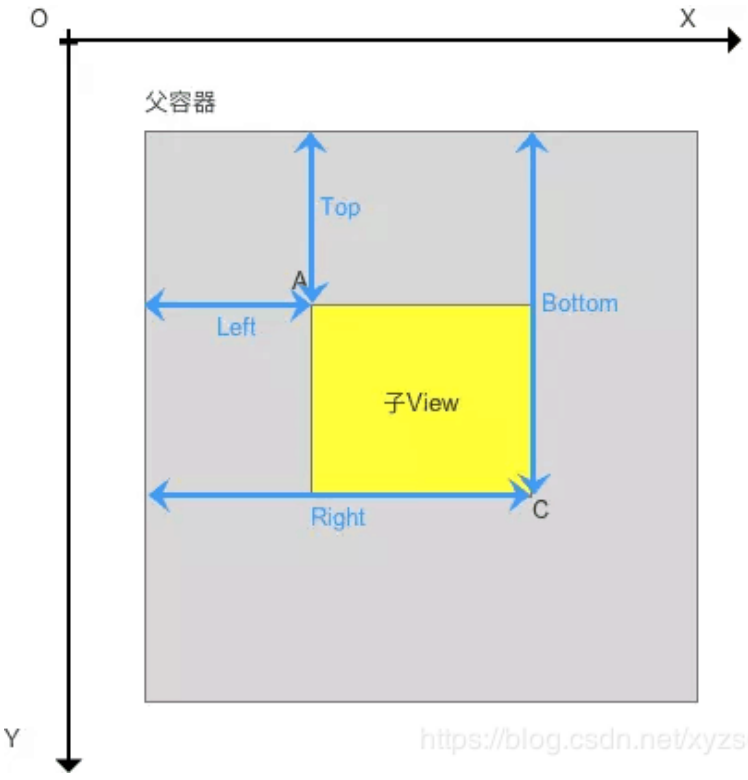
6.View位置（坐标）描述

- View 的位置由4个顶点决定的(如下A、B、C、D)



4个顶点的位置描述分别由4个值决定（ `View` 的位置是相对父控件而言的）

- `Top` :子 `View` 上边界到父 `View` 上边界的距离
- `Left` :子 `View` 左边界到父 `View` 左边界的距离
- `Bottom` :子 `View` 下边界到父 `View` 上边界的距离
- `Right` :子 `View` 右边界到父 `View` 左边界的距离



7.位置获取方式

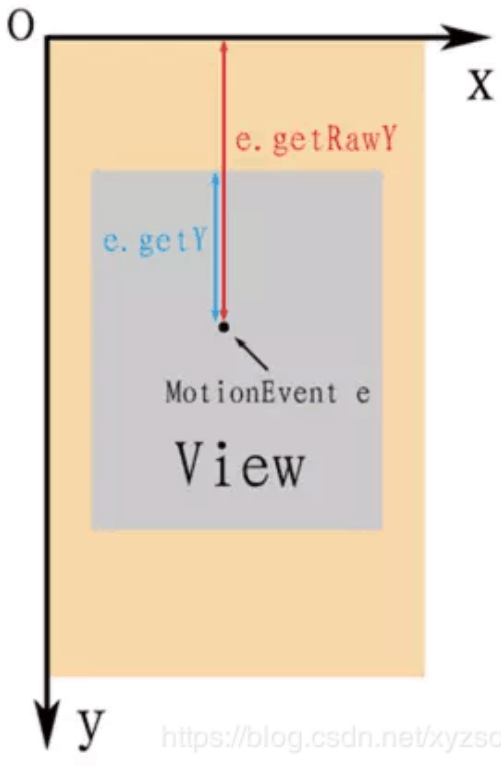
- `View`的位置是通过`view.getxxx`函数进行获取：

```
// 获取Top位置
public final int getTop() {
    return mTop;
}

// 其余如下：
  getLeft();      //获取子View左上角距父View左侧的距离
  getBottom();    //获取子View右下角距父View顶部的距离
  getRight();     //获取子View右下角距父View左侧的距离
```

- 与`MotionEvent`中`get()`和`getRaw()`的区别

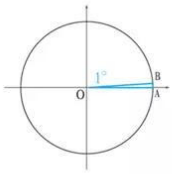
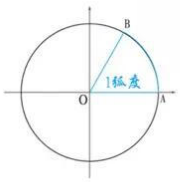
```
//get() ：触摸点相对于其所在组件坐标系的坐标
 event.getX();
 event.getY();

//getRaw() ：触摸点相对于屏幕默认坐标系的坐标
 event.getRawX();
 event.getRawY();
```

具体如下图：

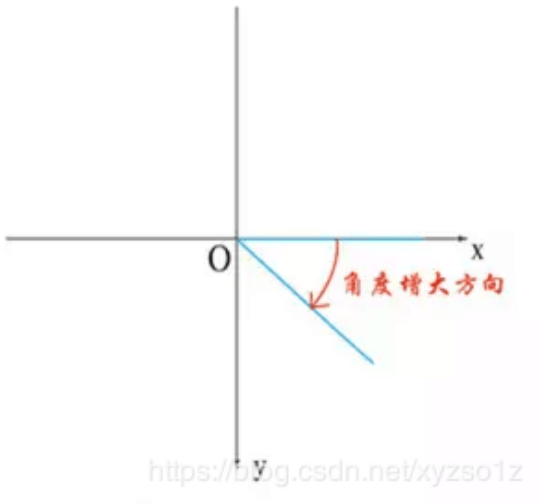


8. 角度（angle）&弧度（radian）

- 自定义 `View` 实际上是将一些简单的形状通过计算，从而组合到一起形成的效果。
  这会设计到画布的相关操作（旋转）、正余弦函数计算等，即会涉及到角度（ `angle` ）与弧（ `radian` ）的相关知识。
- 角度和弧度都是描述角的一种度量单位，区别如下图：

名称	定义	图解	备注	计算公式	
角度（angle）	两条从圆心向圆周射出的射线形成的夹角		当角度所对弧长正好等于圆周长的360分之一时，两条射线的夹角的大小为1度.	$\frac{弧长}{圆周长} \times 360° = 角度$.	相互转换： $ang = \frac{180}{\pi} \times rad$
弧度（radian）	两条从圆心向圆周射出的射线形成的夹角		当角度所对的弧长正好等于圆的半径时，两条射线的夹角为1弧度.	$\frac{弧长}{半径r} = 弧度$.	

在默认的屏幕坐标系中角度增大方向为顺时针。



注：在常见的数学坐标系中角度增大方向为逆时针。

9. 颜色相关

Android 中颜色相关内容包括颜色模式，创建颜色的方式，以及颜色的混合模式等。

> 颜色模式

颜色模式	解释
ARGB8888	四通道高精度（32位）
ARGB4444	四通道地精度（16位）
RGB565	android 屏幕默认模式（16位）
Alpha8	仅有透明通道（8位）
备注	1.字母表示通道类 2.数值表示该类型用多少位二进制来描述 3.例子：ARGB8888，表示有四个通道（ARGB）;每个对应的通道均用8位来描述。

以ARGB8888为例介绍颜色定义：

类型	解释	取值范围=0(0x00) – 255(0xff)	备注
A(Alpha)	透明度	透明 – 不透明	
R(Red)	红色	无色 – 红色	• 小 – 大 = 浅 – 深；； • 当RGB全取最小值(0或0x000000)时颜色为黑色，全取最大值(255或0xffffff)时颜色为白色
G(Green)	绿色	无色 – 绿色	
B(Blue)	蓝色	无色 – 蓝色	

9.2 定义颜色的方式

在java中定义颜色

```
//java中使用Color类定义颜色
int color = Color.GRAY;      //灰色

 //Color类是使用ARGB值进行表示
 int color = Color.argb(127, 255, 0, 0);    //半透明红色
 int color = 0xaaff0000;                    //带有透明度的红色
```

在xml文件中定义颜色

在/res/values/color.xml文件中如下定义：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    //定义了红色（没有alpha（透明）通道）
    <color name="red">#ff0000</color>
    //定义了蓝色（没有alpha（透明）通道）
    <color name="green">#00ff00</color>
</resources>
```

在`xml`文件中以“#”开头定义颜色，后面跟十六进制的值，有如下几种定义方式：

```
#f00             //低精度 - 不带透明通道红色
#af00            //低精度 - 带透明通道红色

#ff0000          //高精度 - 不带透明通道红色
#aaff0000        //高精度 - 带透明通道红色
```

9.3 引用颜色的方式

在java文件中引用xml中定义的颜色：

```
//方法1
int color = getResources().getColor(R.color.mycolor);

//方法2（API 23及以上）
int color = getColor(R.color.myColor);
```

在xml文件（layout或style)中引用或者创建颜色

```
<!--在style文件中引用-->
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <item name="colorPrimary">@color/red</item>
    </style>

 <!--在layout文件中引用在/res/values/color.xml中定义的颜色-->
  android:background="@color/red"

 <!--在layout文件中创建并使用颜色-->
  android:background="#ff0000"
```

xyzso1z
原创文章 80　获赞 40　访问量 2万+