**[Android应用坐标系统全面详解](http://blog.csdn.net/yanbober/article/details/50419117)**

# ****1 背景****

去年有很多人私信告诉我让说说自定义控件，其实通观网络上的很多博客都在讲各种自定义控件，但是大多数都是授之以鱼，却很少有较为系统性授之于渔的文章，同时由于自己也迟迟没有时间规划这一系列文章，最近想将这一系列文章重新提起来，所以就来先总结一下自定义控件的一个核心知识点——坐标系。

很多人可能不屑一顾**[Android](http://lib.csdn.net/base/android" \o "Android知识库" \t "http://blog.csdn.net/yanbober/article/details/_blank)**的坐标系，但是如果你想彻底学会自定义控件，我想说了解Android各种坐标系及一些API的坐标含义绝对算一个小而不可忽视的技能；所谓Android自定义View那几大主要onXXX()方法的重写实质其实大多数都是在处理坐标逻辑运算，所以我们就先来就题重谈一下Android坐标系。

# ****2 Android坐标系****

说到Android坐标系其实就是一个三维坐标，Z轴向上，X轴向右，Y轴向下。这三维坐标的点处理就能构成Android丰富的界面或者动画等效果，所以Android坐标系在整个Android界面中算是盖楼房的尺寸草图，下面我们就来看看这些相关的概念。

## ****2-1 Android屏幕区域划分****

我们先看一副图来了解一下Android屏幕的区域划分（关于这个东西的深入探讨你可以看下[《Android应用setContentView与LayoutInflater加载解析机制源码分析 》](http://blog.csdn.net/yanbober/article/details/45970721)一文，那儿给出了部分原理的解释），如下：



通过上图我们可以很直观的看到Android对于屏幕的划分定义。下面我们就给出这些区域里常用区域的一些坐标或者度量方式。如下：

//获取屏幕区域的宽高等尺寸获取

DisplayMetrics metrics = new DisplayMetrics();

getWindowManager().getDefaultDisplay().getMetrics(metrics);

int widthPixels = metrics.widthPixels;

int heightPixels = metrics.heightPixels;

//应用程序App区域宽高等尺寸获取

Rect rect = new Rect();

getWindow().getDecorView().getWindowVisibleDisplayFrame(rect);

//获取状态栏高度

Rect rect= new Rect();

getWindow().getDecorView().getWindowVisibleDisplayFrame(rect);

int statusBarHeight = rectangle.top;

//View布局区域宽高等尺寸获取

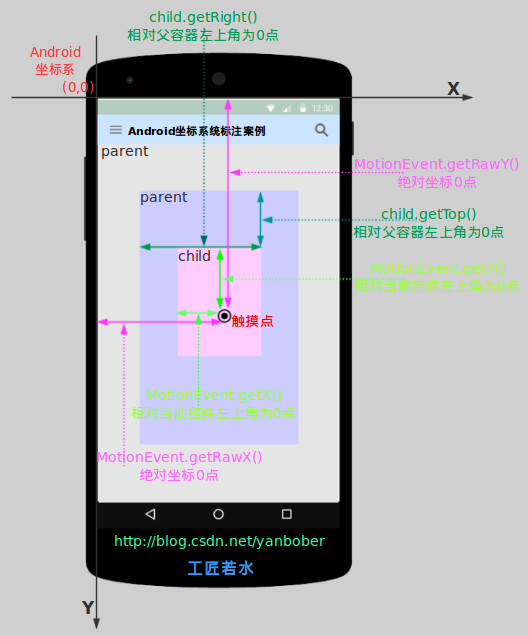
Rect rect = new Rect();

getWindow().findViewById(Window.ID\_ANDROID\_CONTENT).getDrawingRect(rect);

**特别注意：**上面这些方法最好在Activity的onWindowFocusChanged ()方法或者之后调运，因为只有这时候才是真正的显示OK，不懂的可以看我之前关于setContentView相关的博客。

## ****2-2 Android View绝对相对坐标系****

上面我们分析了Android屏幕的划分，可以发现我们平时开发的重点其实都在关注View布局区域，那么下面我们就来细说一下View区域相关的各种坐标系。先看下面这幅图：



通过上图我们可以很直观的给出View一些坐标相关的方法解释，不过必须要明确的是上面这些方法必须要在layout之后才有效，如下：

| **View的静态坐标方法** | **解释** |
| --- | --- |
| getLeft() | 返回View自身左边到父布局左边的距离 |
| getTop() | 返回View自身顶边到父布局顶边的距离 |
| getRight() | 返回View自身右边到父布局左边的距离 |
| getBottom() | 返回View自身底边到父布局顶边的距离 |
| getX() | 返回值为getLeft()+getTranslationX()，当setTranslationX()时getLeft()不变，getX()变。 |
| getY() | 返回值为getTop()+getTranslationY()，当setTranslationY()时getTop()不变，getY()变。 |

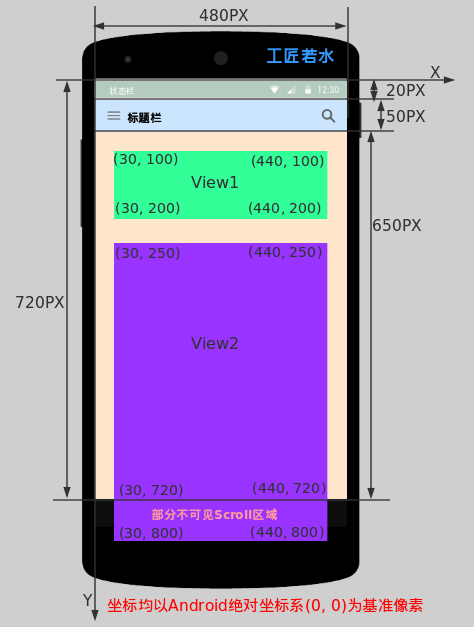
同时也可以看见上图中给出了手指触摸屏幕时MotionEvent提供的一些方法解释，如下：

| **MotionEvent坐标方法** | **解释** |
| --- | --- |
| getX() | 当前触摸事件距离当前View左边的距离 |
| getY() | 当前触摸事件距离当前View顶边的距离 |
| getRawX() | 当前触摸事件距离整个屏幕左边的距离 |
| getRawY() | 当前触摸事件距离整个屏幕顶边的距离 |

上面就解释了你在很多代码中看见各种getXXX方法进行数学逻辑运算判断的含义。不过上面只是说了一些相对静止的Android坐标点关系，下面我们来看看几个和上面方法紧密相关的View方法。如下：

| **View宽高方法** | **解释** |
| --- | --- |
| getWidth() | layout后有效，返回值是mRight-mLeft，一般会参考measure的宽度（measure可能没用），但不是必须的。 |
| getHeight() | layout后有效，返回值是mBottom-mTop，一般会参考measure的高度（measure可能没用），但不是必须的。 |
| getMeasuredWidth() | 返回measure过程得到的mMeasuredWidth值，供layout参考，或许没用。 |
| getMeasuredHeight() | 返回measure过程得到的mMeasuredHeight值，供layout参考，或许没用。 |

上面解释了自定义View时各种获取宽高的一些含义，下面我们再来看看关于View获取屏幕中位置的一些方法，不过这些方法需要在Activity的onWindowFocusChanged ()方法之后才能使用。如下图：



下面我们就给出上面这幅图涉及的View的一些坐标方法的结果（结果采用使用方法返回的实际坐标，不依赖上面实际绝对坐标转换，上面绝对坐标只是为了说明例子中的位置而已），如下：

| **View的方法** | **上图View1结果** | **上图View2结果** | **结论描述** |
| --- | --- | --- | --- |
| getLocalVisibleRect() | (0, 0 - 410, 100) | (0, 0 - 410, 470) | 获取View自身可见的坐标区域，坐标以自己的左上角为原点(0,0)，另一点为可见区域右下角相对自己(0,0)点的坐标，其实View2当前height为550，可见height为470。 |
| getGlobalVisibleRect() | (30, 100 - 440, 200) | (30, 250 - 440, 720) | 获取View在屏幕绝对坐标系中的可视区域，坐标以屏幕左上角为原点(0,0)，另一个点为可见区域右下角相对屏幕原点(0,0)点的坐标。 |
| getLocationOnScreen() | (30, 100) | (30, 250) | 坐标是相对整个屏幕而言，Y坐标为View左上角到屏幕顶部的距离。 |
| getLocationInWindow() | (30, 100) | (30, 250) | 如果为普通Activity则Y坐标为View左上角到屏幕顶部（此时Window与屏幕一样大）；如果为对话框式的Activity则Y坐标为当前Dialog模式Activity的标题栏顶部到View左上角的距离。 |

到此常用的相关View的静态坐标获取处理的方法和含义都已经叙述完了，下面我们看看动态的一些解释（所谓动静只是我个人称呼而已）。

## ****2-3 Android View动画相关坐标系****

其实在我们使用动画时，尤其是补间动画时，你会发现其中涉及很多坐标参数，一会儿为相对的，一会儿为绝对的，你可能会各种蒙圈。那么不妨看下[《Android应用开发之所有动画使用详解 》](http://blog.csdn.net/yanbober/article/details/46481171)这篇博客，这里面详细介绍了关于Android动画相关的坐标系统，这里不再累赘叙述。

## ****2-4 Android View滑动相关坐标系****

关于View提供的与坐标息息相关的另一组常用的重要方法就是滚动或者滑动相关的，下面我们给出相关的解释（**特别注意：View的scrollTo()和scrollBy()是用于滑动View中的内容，而不是改变View的位置；改变View在屏幕中的位置可以使用offsetLeftAndRight()和offsetTopAndBottom()方法，他会导致getLeft()等值改变。**），如下：

| **View的滑动方法** | **效果及描述** |
| --- | --- |
| offsetLeftAndRight(int offset) | 水平方向挪动View，offset为正则x轴正向移动，移动的是整个View，getLeft()会变的，**自定义View很有用**。 |
| offsetTopAndBottom(int offset) | 垂直方向挪动View，offset为正则y轴正向移动，移动的是整个View，getTop()会变的，**自定义View很有用**。 |
| scrollTo(int x, int y) | 将**View中内容（不是整个View）**滑动到相应的位置，参考坐标原点为ParentView左上角，x，y为正则向xy轴反方向移动，反之同理。 |
| scrollBy(int x, int y) | 在scrollTo()的基础上继续滑动xy。 |
| setScrollX(int value) | 实质为scrollTo()，只是只改变Y轴滑动。 |
| setScrollY(int value) | 实质为scrollTo()，只是只改变X轴滑动。 |
| getScrollX()/getScrollY() | 获取当前滑动位置偏移量。 |

关于Android View的scrollBy()和scrollTo()参数传递正数却向坐标系负方向移动的特性可能很多人都有疑惑，甚至是死记结论，这里我们简单给出产生这种特性的真实原因—-源码分析，如下：

public void scrollTo(int x, int y) {

if (mScrollX != x || mScrollY != y) {

int oldX = mScrollX;

int oldY = mScrollY;

mScrollX = x;

mScrollY = y;

invalidateParentCaches();

onScrollChanged(mScrollX, mScrollY, oldX, oldY);

if (!awakenScrollBars()) {

postInvalidateOnAnimation();

}

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13

View的该方法注释里明确说明了调运他会触发onScrollChanged()和invalidated()方法，那我们就将矛头转向invalidated()方法触发的draw()过程，draw()过程中最终其实会触发下面的invalidate()方法，如下：

public void invalidate(int l, int t, int r, int b) {

final int scrollX = mScrollX;

final int scrollY = mScrollY;

//scroller时为何参数和坐标反向的真实原因

invalidateInternal(l - scrollX, t - scrollY, r - scrollX, b - scrollY, true, false);

}

* 1
* 2
* 3
* 4
* 5
* 6
* 1
* 2
* 3
* 4
* 5
* 6

核心就在这里，相信不用我解释大家也知道咋回事了，自行脑补。

**scrollTo()和scrollBy()方法特别注意：**如果你给一个ViewGroup调用scrollTo()方法滚动的是ViewGroup里面的内容，如果想滚动一个ViewGroup则再给他嵌套一个外层，滚动外层即可。

**【工匠若水** <http://blog.csdn.net/yanbober> **未经允许严禁转载，请尊重作者劳动成果。**[私信联系我](http://weibo.com/yanbober)**】**

# ****3 总结****

可以发现，上面只是说明了一些View里常用的与坐标相关的概念，关于自定义控件了解学习这些坐标概念只是一个基础，也是一个后续内容的铺垫，所以有必要先完全吃透此部分内容才能继续拓展学习新的东东。

View中还有一些其他与坐标获取相关的方法，但是一般都比较不常用，所以用到时可以现查API或者Debug看现象进行学习即可，这里篇幅和时间有限就不一一道来了。