这篇文章主要给大家介绍了Android View移动的三种方式，在介绍这三种方式之前先介绍了Android坐标系的定义规则以及View的一些位置参数。有需要的朋友们可以参考借鉴。

影响view布局显示的因素

mFollowerView.setTranslationY(0);

mFollowerView.setY(0);

FrameLayout.LayoutParams LayoutParams\_videoRoot=(FrameLayout.LayoutParams)videoRoot.getLayoutParams();

LayoutParams\_videoRoot.height= LayoutParams.MATCH\_PARENT;

LayoutParams\_videoRoot.topMargin=0;

videoRoot.setLayoutParams(LayoutParams\_videoRoot);

videoRoot.offsetTopAndBottom(0);

this.setPadding(0,0,0,0);

videoRoot.getScrollY()

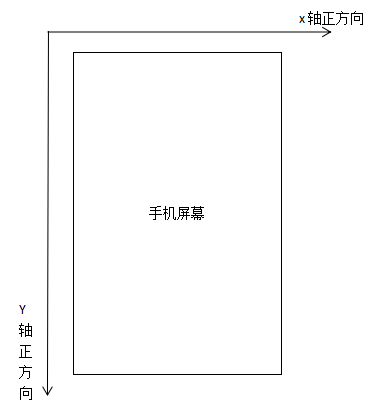
videoRoot.getScaleY()

View的角度也会有印象，View锚点，相结合，有不同的效果

**前言**

在Android开发中，View一直是Android开发人员的一块心病，一方面想要进阶，一方面又害怕进阶，可以说Android的View是进阶路上的最大绊脚石，因为它涉及的东西太多了，比如本次我们此次要写的View移动，另外还包括View的触摸事件的传递，创建自定义View，这些都是极其重要且不得不面对的难题。但是无论如何，现在不克服的困难将来就会被困难克服。

在此之前，我们还是先了解Android坐标系的定义规则以及View的一些位置参数。



**Android坐标系**

View的位置及大小是由四个参数决定，即left、top、right、bottom，并且这四个参数都是相对于其父View的。

[?](http://www.jb51.net/article/92788.htm)

|  |  |
| --- | --- |
| 1  2 | int width = right-left;   int height = bottom-top; |

在Activity中布局完成后，我们可以通过View一些方法获取这些参数信息：

[?](http://www.jb51.net/article/92788.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5 | //left,top,right,bottom值的获取   int left = getLeft();   int top = getTop();   int right = getRight();   int bottom = getBottom(); |

另外Android 3.0以后加入x，y，translationX，translationY等参数。（x,y）表示为View在ViewGroup中左上角的x,y的值，translationX，translationY在用于平移一个View。默认是都为0，在调用了View的setTranslationX()/setTranslationY()之后发生改变。

[?](http://www.jb51.net/article/92788.htm)

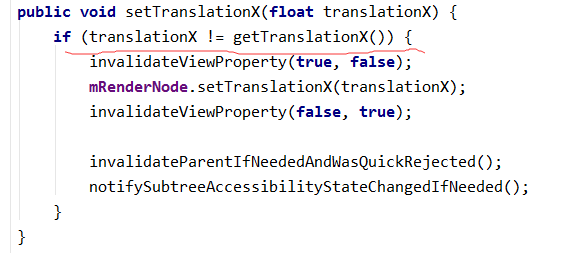
|  |  |
| --- | --- |
| 1  2  3  4  5 | //x,y,translationX,translationY参数的获取   int x = getX();   int y = getY();   int translationX = getTranslationX();   int translationY = getTranslationY(); |

PS：调用View的setTranslationX()和setTranslationY()方法虽然可以使得View平移指定距离，但是这一过程是瞬间完成的。为了使View的移动使得更为平滑，因此可以使用View的属性动画来指定translationX和translationY。

[?](http://www.jb51.net/article/92788.htm)

|  |  |
| --- | --- |
| 1  2  3 | ObjectAnimator valueAnimator = ObjectAnimator.ofFloat(textView, "translationX", 200);   valueAnimator.setDuration(2000);   valueAnimator.start(); |

另外，如果给View设置setTranslationX()和setTranslationY()后，如果设置的值没有发生变化，那么其只会移动一次，即首次指定的移动距离。查看源码后我们发现原因：原来在设置值之后其会将设置进去的值和当前的translationX，translationY进行对比，不一致时才进行移动。



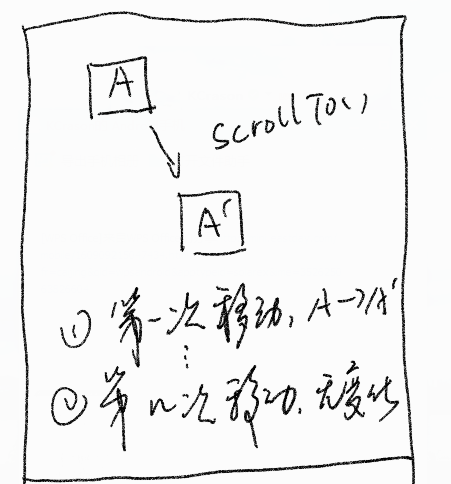
了解了View的一些基本参数之后，我们看关于View的三种移动方式。

**一、使用Android系统提供的scrollTo()/scrollBy()方法实现View的移动。**

不管是scrollTo()还是scrollBy()其移动的本质都是View/ViewGroup中的内容。并且其移动的过程是瞬间完成的，因此，为了实现更好的移动效果，他需要与Scroller类结合使用。另外，它不同于上面的Translation，移动的是View本身，这一点需要好好理解一下。

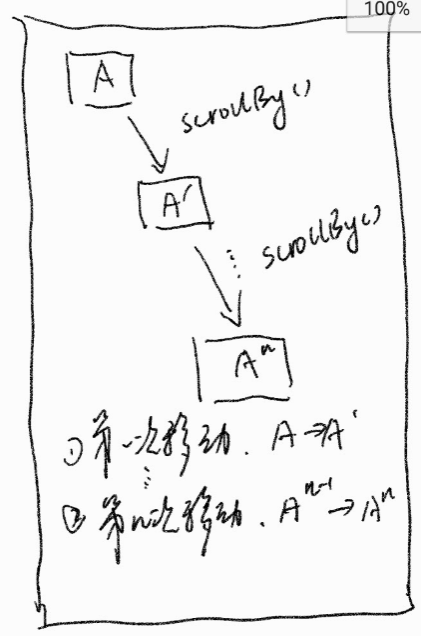
scrollTo()和scrollBy()都是View中的方法， 不是Scroller中的方法 ，但是控制View的平滑移动与Scroller类密不可分。

scrollTo() :指是的移动的绝对位置，如果位置没有变化，多次调用则不会起作用。



**scrollTo移动过程示意图**

scrollBy() :其本质依然是调用的scrollTo() ，指的的移动当前位置的相对距离（每次都是先将当前的位置和设置的距离相加之和调用scrollTo()，这样如果你多次调用，你就会发现其每次都会移动一段距离，这是和scrollTo()的本质区别）



**scrollBy移动过程示意图**

**PS:**关于上面两张图，其实一直以来，我自己都没完全搞明白什么相对绝对，所以两张手图可能会让人更容易理解。还有就是scrollTo()和scrollBy()移动方向问题，上面我们已经画过Android的坐标系，x轴左→右为正，y轴从上→下为正。但是这并不适用于scrollTo和scrollBy，scrollTo和scrollBy刚好相反，即x轴左→右为负，y轴从上→下为负，简直是有点坑爹啊。

Scroller类分析：而为什么使用Scroller类中的方法可以对View/ViewGroup的内容进行移动呢？下面我们试着分析一下。

**首先**

我们创建一个Scroller类的对象mScroller。

**然后**

要使View在规定的时间中移动到指定的位置，我们会调用startScroll()方法，startScroll()是Scroller类中的方法，另外Scroller类中还有一个filing()方法也是很常用的，它主要是处理平滑的移动，一般营造滑动之后的惯性效果，使得View的移动更逼真。下面我们看startScroll()的源码：

[?](http://www.jb51.net/article/92788.htm)

|  |  |
| --- | --- |
| 1  2  3  4 | //其接收四个/五个参数。如果duration不设置，则为默认。这四个参数都不难理解，这里不再做解释。   public void startScroll(int startX, int startY, int dx, int dy, int duration) {   ...   } |

而一般我们调用这个方法后都要去调View的 invalidate() ，这个方法可以触发View的draw()方法。而draw()中调用了 computeScroll() ，源码中我们发现computeScroll()是个空方法，这也是为什么我们需要重写 computeScroll()方法的原因。因为正在的移动操作就是在computeScroll()中进行的。

[?](http://www.jb51.net/article/92788.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | @Override   public void computeScroll() {   if (mScroller.computeScrollOffset()) {    scrollTo(mScroller.getCurrX(), mScroller.getCurrY());    //必须调用View的postInvalidate()/invalidate()，如果不加会导致View的移动只会第一帧。    postInvalidate();   }   super.computeScroll();   } |

上面我们看到Scroller类中还有一个computeScrollOffset()方法，它又是干啥的呢？它的主要作用就是判断mCurrX，和mCurrY是否有改变，有则返回true，无则返回false。通过这个方法的判断可以指点是否需要持续的调用scrollTo()去移动View。这里再给出一个示例，使用scrollTo()让View跟着手指移动：

[?](http://www.jb51.net/article/92788.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  85 | public class CuView extends LinearLayout {     private float mStartX;   private float mStartY;   private Scroller mScroller;   /\*\*   \* 第一次滑动是否完成   \*/   private boolean isFirstFinish;     public CuView(Context context) {   super(context);   init(context);   }     public CuView(Context context, AttributeSet attrs) {   super(context, attrs);   init(context);   }     private void init(Context context) {   mScroller = new Scroller(context);   }     public CuView(Context context, AttributeSet attrs, int defStyleAttr) {   super(context, attrs, defStyleAttr);   init(context);   }     @TargetApi(Build.VERSION\_CODES.LOLLIPOP)   public CuView(Context context, AttributeSet attrs, int defStyleAttr, int defStyleRes) {   super(context, attrs, defStyleAttr, defStyleRes);   init(context);   }       /\*\*   \* 让View跟着你的手指走吧   \* @param event   \* @return   \*/   @Override   public boolean onTouchEvent(MotionEvent event) {   int action = event.getAction();   switch (action) {    case MotionEvent.ACTION\_DOWN:    /\*\*     \* 第一次移动完成后，我们不需要再去拿开始的位置了，否则造成View重新移动的最起始的位置。     \*/    if (!isFirstFinish) {     mStartX = event.getRawX();     mStartY = event.getRawY();    }    break;    case MotionEvent.ACTION\_MOVE:    scrollTo((int) (mStartX - event.getRawX()), (int) (mStartY - event.getRawY()));    break;    case MotionEvent.ACTION\_UP:    //第一次移动完成    isFirstFinish = true;    break;   }   return true;   }     /\*\*   \* 测试startScroll   \*/   public void startScroll() {   /\*\*    \* 注意Scroller移动方向，    \*/   mScroller.startScroll(20, 20, -500, -500, 5000);   invalidate();   }     @Override   public void computeScroll() {   if (mScroller.computeScrollOffset()) {    scrollTo(mScroller.getCurrX(), mScroller.getCurrY());    invalidate();   }   super.computeScroll();   }  } |

**二、使用动画实现View的移动。**

这里包括View的Tween Animation/Frame Animation，以及3.0之后加入的Property Animation。其移动的是View的一个映像，View本身的位置及大小并没有发生任何改变。

**三、设置View的LayoutParams来移动View**

[?](http://www.jb51.net/article/92788.htm)

|  |  |
| --- | --- |
| 1  2  3 | LinearLayout.LayoutParams layoutParams = (LinearLayout.LayoutParams) textView.getLayoutParams();   layoutParams.leftMargin = 50;   textView.requestLayout(); |

**总结**

以上就是总结Android View移动的3种方式的全部内容了，希望本文的内容对大家开发Android的时候能有所帮助，如果有疑问大家可以留言交流。