2019Android View 总结

1. View 的滑动方式

a.layout(left,top,right,bottom):通过修改 View 四个方向的属性值来修改 View 的坐标，从而滑动 View

b.offsetLeftAndRight() offsetTopAndBottom():指定偏移量滑动 view

c.LayoutParams,改变布局参数：layoutParams 中保存了 view 的布局参数，可以通过修改布局参数的方式滑动 view

d.通过动画来移动 view：注意安卓的平移动画不能改变 view 的位置参数，属性动画可以

e.scrollTo/scrollBy:注意移动的是 view 的内容，scrollBy(50,50)你会看到屏幕上的内容向屏幕的左上角移动了，这是参考对象不同导致的，你可以看作是它移动的是手机屏幕，手机屏幕向右下角移动，那么屏幕上的内容就像左上角移动了

f.scroller:scroller 需要配置 computeScroll 方法实现 view 的滑动，scroller 本身并不会滑动 view，它的作用可以看作一个插值器，它会计算当前时间点 view 应该滑动到的距离，然后 view 不断的重绘，不断的调用 computeScroll 方法，这个方法是个空方法，所以我们重写这个方法，在这个方法中不断的从 scroller 中获取当前 view 的位置，调用 scrollTo 方法实现滑动的效果

2. View 的事件分发机制

点击事件产生后，首先传递给 Activity 的 dispatchTouchEvent 方法，通过 PhoneWindow 传递给 DecorView,然后再传递给根 ViewGroup,进入 ViewGroup 的 dispatchTouchEvent 方法，执行 onInterceptTouchEvent 方法判断是否拦截，再不拦截的情况下，此时会遍历 ViewGroup 的子元素，进入子 View 的 dispatchToucnEvent 方法，如果子 view 设置了 onTouchListener,就执行 onTouch 方法，并根据 onTouch 的返回值为 true 还是 false 来决定是否执行 onTouchEvent 方法，如果是 false 则继续执行 onTouchEvent，在 onTouchEvent 的 Action Up 事件中判断，如果设置了 onClickListener ,就执行 onClick 方法。

3. View 的加载流程

View 随着 Activity 的创建而加载，startActivity 启动一个 Activity 时，在 ActivityThread 的 handleLaunchActivity 方法中会执行 Activity 的 onCreate 方法，这个时候会调用 setContentView 加载布局创建出 DecorView 并将我们的 layout 加载到 DecorView 中，当执行到 handleResumeActivity 时，Activity 的 onResume 方法被调用，然后 WindowManager 会将 DecorView 设置给 ViewRootImpl,这样，DecorView 就被加载到 Window 中了，此时界面还没有显示出来，还需要经过 View 的 measure，layout 和 draw 方法，才能完成 View 的工作流程。我们需要知道 View 的绘制是由 ViewRoot 来负责的，每一个 DecorView 都有一个与之关联的 ViewRoot,这种关联关系是由 WindowManager 维护的，将 DecorView 和 ViewRoot 关联之后，ViewRootImpl 的 requestLayout 会被调用以完成初步布局，通过 scheduleTraversals 方法向主线程发送消息请求遍历，最终调用 ViewRootImpl 的 performTraversals 方法，这个方法会执行 View 的 measure layout 和 draw 流程

4. View 的 measure layout 和 draw 流程

在上边的分析中我们知道，View 绘制流程的入口在 ViewRootImpl 的 performTraversals 方法，在方法中首先调用 performMeasure 方法，传入一个 childWidthMeasureSpec 和 childHeightMeasureSpec 参数，这两个参数代表的是 DecorView 的 MeasureSpec 值，这个 MeasureSpec 值由窗口的尺寸和 DecorView 的 LayoutParams 决定，最终调用 View 的 measure 方法进入测量流程

measure：

View 的 measure 过程由 ViewGroup 传递而来，在调用 View.measure 方法之前，会首先根据 View 自身的 LayoutParams 和父布局的 MeasureSpec 确定子 view 的 MeasureSpec，然后将 view 宽高对应的 measureSpec 传递到 measure 方法中，那么子 view 的 MeasureSpec 获取规则是怎样的？分几种情况进行说明

1.父布局是 EXACTLY 模式：

a.子 view 宽或高是个确定值，那么子 view 的 size 就是这个确定值，mode 是 EXACTLY（是不是说子 view 宽高可以超过父 view？见下一个）

b.子 view 宽或高设置为 match_parent,那么子 view 的 size 就是占满父容器剩余空间，模式就是 EXACTLY

c.子 view 宽或高设置为 wrap_content,那么子 view 的 size 就是占满父容器剩余空间，不能超过父容器大小，模式就是 AT_MOST

2.父布局是 AT_MOST 模式：

a.子 view 宽或高是个确定值，那么子 view 的 size 就是这个确定值，mode 是 EXACTLY

b.子 view 宽或高设置为 match_parent,那么子 view 的 size 就是占满父容器剩余空间,不能超过父容器大小，模式就是 AT_MOST

c.子 view 宽或高设置为 wrap_content,那么子 view 的 size 就是占满父容器剩余空间，不能超过父容器大小，模式就是 AT_MOST

3.父布局是 UNSPECIFIED 模式：

a.子 view 宽或高是个确定值，那么子 view 的 size 就是这个确定值，mode 是 EXACTLY

b.子 view 宽或高设置为 match_parent,那么子 view 的 size 就是 0，模式就是 UNSPECIFIED

c.子 view 宽或高设置为 wrap_content,那么子 view 的 size 就是 0，模式就是 UNSPECIFIED

获取到宽高的 MeasureSpec 后，传入 view 的 measure 方法中来确定 view 的宽高，这个时候还要分情况

1.当 MeasureSpec 的 mode 是 UNSPECIFIED,此时 view 的宽或者高要看 view 有没有设置背景，如果没有设置背景，就返回设置的 minWidth 或 minHeight,这两个值如果没有设置默认就是 0，如果 view 设置了背景，就取 minWidth 或 minHeight 和背景这个 drawable 固有宽或者高中的最大值返回

2.当 MeasureSpec 的 mode 是 AT_MOST 和 EXACTLY，此时 view 的宽高都返回从

MeasureSpec 中获取到的 size 值，这个值的确定见上边的分析。因此如果要通过继承 view 实现自定义 view，一定要重写 onMeasure 方法对 wrap_conten 属性做处理，否则，他的 match_parent 和 wrap_content 属性效果就是一样的

layout:

layout 方法的作用是用来确定 view 本身的位置，onLayout 方法用来确定所有子元素的位置，当 ViewGroup 的位置确定之后，它在 onLayout 中会遍历所有的子元素并调用其 layout 方法，在子元素的 layout 方法中 onLayout 方法又会被调用。layout 方法的流程是，首先通过 setFrame 方法确定 view 四个顶点的位置，然后 view 在父容器中的位置也就确定了，接着会调用 onLayout 方法，确定子元素的位置，onLayout 是个空方法，需要继承者去实现。

getMeasuredHeight 和 getHeight 方法有什么区别？getMeasuredHeight（测量高度）形成于 view 的 measure 过程，getHeight（最终高度）形成于 layout 过程，在有些情况下，view 需要 measure 多次才能确定测量宽高，在前几次的测量过程中，得出的测量宽高有可能和最终宽高不一致，但是最终来说，还是会相同，有一种情况会导致两者值不一样，如下，此代码会导致 view 的最终宽高比测量宽高大 100px

```
public void layout(int l,int t,int r, int b){

    super.layout(l,t,r+100,b+100);}
```

draw:

View 的绘制过程遵循如下几步：

a.绘制背景 background.draw(canvas)

b.绘制自己（onDraw）

c.绘制 children（dispatchDraw）

d.绘制装饰（onDrawScrollBars）

View 绘制过程的传递是通过 dispatchDraw 来实现的，它会遍历所有的子元素的 draw 方法，如此 draw 事件就一层一层的传递下去了

ps：view 有一个特殊的方法 setWillNotDraw，如果一个 view 不需要绘制内容，即不需要重写 onDraw 方法绘制，可以开启这个标记，系统会进行相应的优化。默认情况下，View 没有开启这个标记，默认认为需要实现 onDraw 方法绘制，当我们继承 ViewGroup 实现自定义控件，并且明确知道不需要具备绘制功能时，可以开启这个标记，如果我们重写了 onDraw,那么要显示的关闭这个标记

子 view 宽高可以超过父 view？能

1.android:clipChildren = "false" 这个属性要设置在父 view 上。代表其中的子 View 可以超出屏幕。
2.子 view 要有具体的大小，一定要比父 view 大 才能超出。比如 父 view 高度 100px 子 view 设置高度 150px。子 view 比父 view 大，这样超出的属性才有意义。（高度可以在代码中动态赋值，但不能用 wrap_content / match_partent）。
3.对父布局还有要求，要求使用 linearLayout(反正我用 RelativeLayout 是不行)。你如果必须用其他布局可以在需要超出的 view 上面套一个 linearLayout 外面再套其他的布局。
4.最外面的布局如果设置的 padding 不能超出

5. 自定义 view 需要注意的几点

1.让 view 支持 wrap_content 属性，在 onMeasure 方法中针对 AT_MOST 模式做专门处理，否则 wrap_content 会和 match_parent 效果一样（继承 ViewGroup 也同样要在 onMeasure 中做这个判断处理）

```
if(widthMeasureSpec == MeasureSpec.AT_MOST && heightMeasureSpec == MeasureSpec.AT_MOST){
```

```
    setMeasuredDimension(200,200); // wrap_content 情况下要设置一个默认值，200 只是举个例子，最终的值需要计算得到刚好包裹内容的宽高值
}else if(widthMeasureSpec == MeasureSpec.AT_MOST){
    setMeasuredDimension(200,heightMeasureSpec );
}else if(heightMeasureSpec == MeasureSpec.AT_MOST){
    setMeasuredDimension(heightMeasureSpec ,200);
}
```

2.让 view 支持 padding（onDraw 的时候，宽高减去 padding 值，margin 由父布局控制，不需要 view 考虑），自定义 ViewGroup 需要考虑自身的 padding 和子 view 的 margin 造成的影响

3.在 view 中尽量不要使用 handler，使用 view 本身的 post 方法

4.在 onDetachedFromWindow 中及时停止线程或动画

5.view 带有滑动嵌套情形时，处理好滑动冲突

ACTION_DOWN 没有拦截，ACTION_MOVE ACTION_UP 还会拦截吗