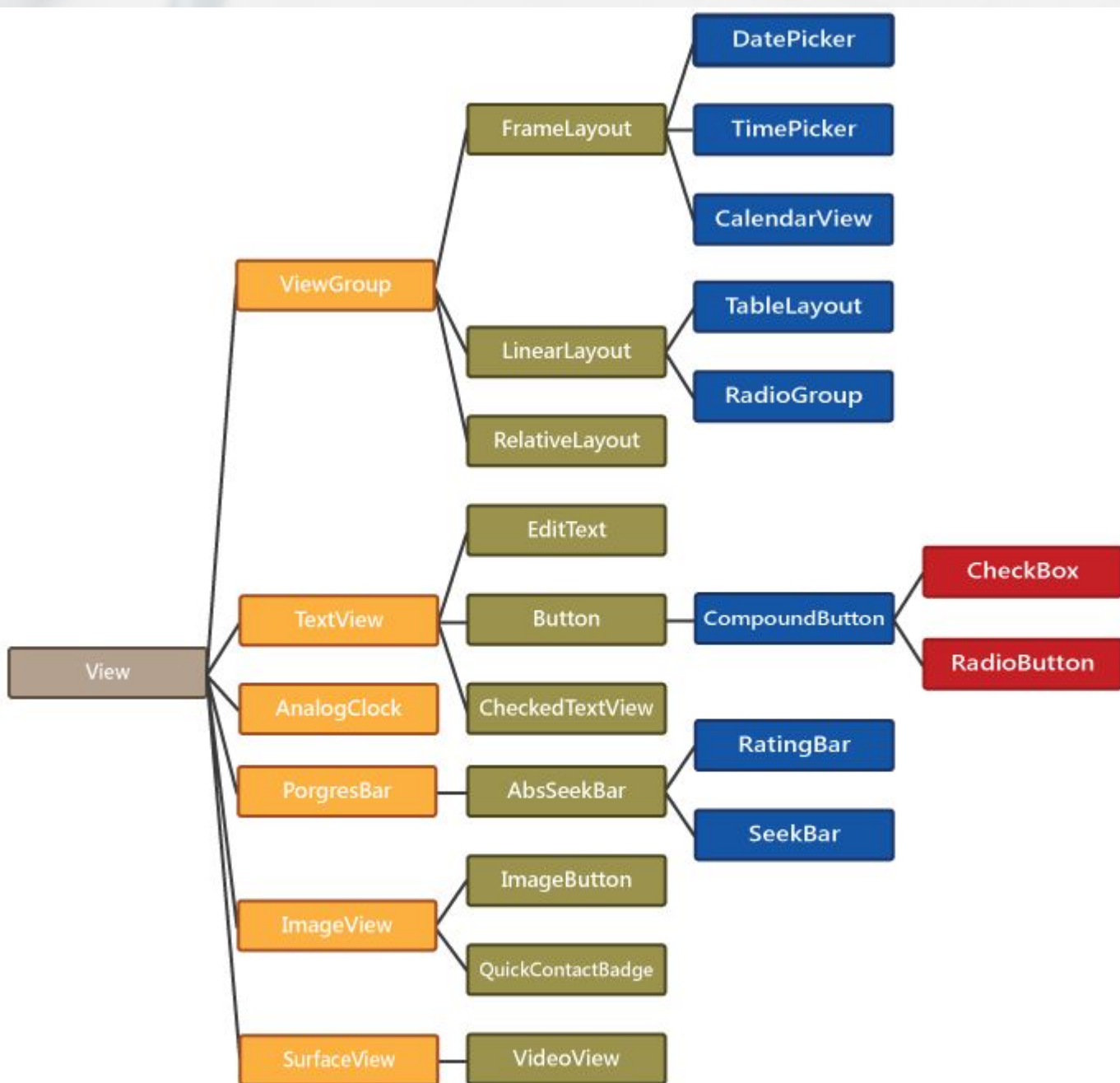




## View自定义及滑动事件



# 自定义View

继承View或者ViewGroup

A

三个构造方法

Context context  
AttributeSet attrs  
int defStyleAttr

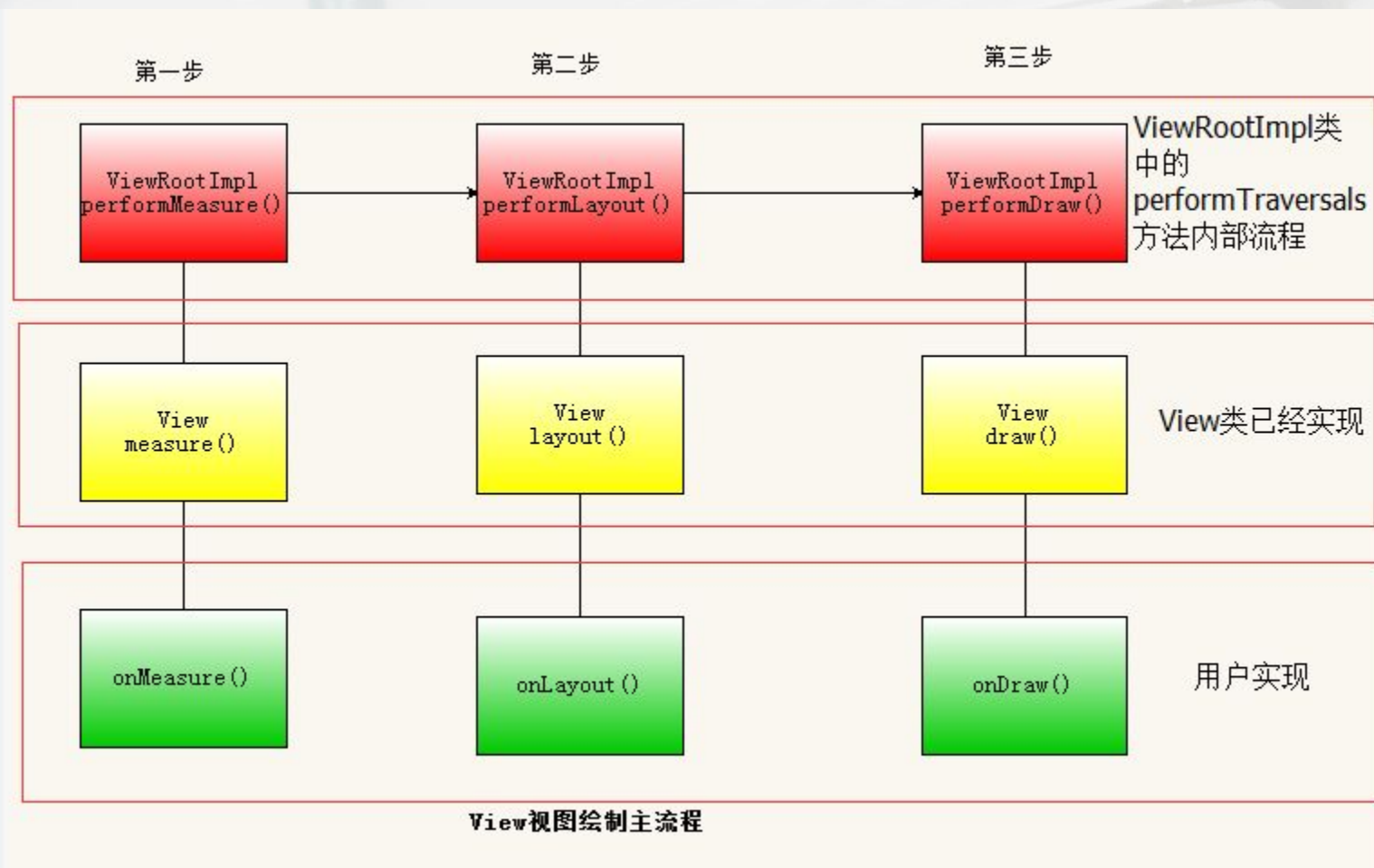
自定义属性（注意引用）

C

B

D

绘制View（三个方法）



# 自定义ViewGroup



**Shock Spock**

2 hr • San Francisco, CA



## Possible UI:

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
    <ProfilePhoto
        android:layout_width="40dp"
        android:layout_height="40dp"/>
    <LinearLayout
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:orientation="vertical">
        <Title
            android:layout_width="match_parent"
            android:layout_height="wrap_content"/>
        <Subtitle
            android:layout_width="match_parent"
            android:layout_height="wrap_content"/>
        </LinearLayout>
    <Menu
        android:layout_width="20dp"
        android:layout_height="20dp"/>
</LinearLayout>
```



**Shock Spock**

2 hr • San Francisco, CA

# Possible UI: measurement

```
> LinearLayout [horizontal]      [w: 1080  exactly,      h: 1557  exactly ]
  > ProfilePhoto                 [w: 120   exactly,      h: 120   exactly ]
  > LinearLayout [vertical]      [w: 0     unspecified,  h: 0     unspecified]
    > Title                      [w: 0     unspecified,  h: 0     unspecified]
    > Subtitle                   [w: 0     unspecified,  h: 0     unspecified]
    > Title                      [w: 222   exactly,      h: 57    exactly ]
    > Subtitle                   [w: 222   exactly,      h: 57    exactly ]
  > Menu                         [w: 60    exactly,      h: 60    exactly ]
  > LinearLayout [vertical]      [w: 900   exactly,      h: 1557  at_most ]
    > Title                      [w: 900   exactly,      h: 1557  at_most ]
    > Subtitle                   [w: 900   exactly,      h: 1500  at_most ]
```



```
protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
```

```
    // 1. Setup initial constraints.
```

```
    int widthConstraints = getPaddingLeft() + getPaddingRight();
    int heightConstraints = getPaddingTop() + getPaddingBottom();
    int width = 0;
    int height = 0;
```

```
    // 2. Measure the ProfilePhoto
```

```
    measureChildWithMargins(
        mPhoto,
        widthMeasureSpec,
        widthConstraints,
        heightMeasureSpec,
        heightConstraints);
```

```
    // 3. Update the constraints.
```

```
    widthConstraints += mPhoto.getMeasuredWidth();
    width += mPhoto.getMeasuredWidth();
    height = Math.max(mPhoto.getMeasuredHeight(), height);
```

```
    // 4. Measure the Menu.
```

```
    measureChildWithMargins(
        mMenu,
        widthMeasureSpec,
        widthConstraints,
        heightMeasureSpec,
        heightConstraints);
```

```
    // 5. Update the constraints.
```

```
    widthConstraints += mMenu.getMeasuredWidth();
    width += mMenu.getMeasuredWidth();
    height = Math.max(mMenu.getMeasuredHeight(), height);
```

```
    // 6. Prepare the vertical MeasureSpec.
```

```
    int verticalWidthMeasureSpec = MeasureSpec.makeMeasureSpec(
        MeasureSpec.getSize(widthMeasureSpec) - widthConstraints,
        MeasureSpec.getMode(widthMeasureSpec));
```

```
    int verticalHeightMeasureSpec = MeasureSpec.makeMeasureSpec(
        MeasureSpec.getSize(heightMeasureSpec) - heightConstraints,
        MeasureSpec.getMode(heightMeasureSpec));
```

```
    // 7. Measure the Title.
```

```
    measureChildWithMargins(
        mTitle,
        verticalWidthMeasureSpec,
        0,
        verticalHeightMeasureSpec,
        0);
```

```
    // 8. Measure the Subtitle.
```

```
    measureChildWithMargins(
        mSubTitle,
        verticalWidthMeasureSpec,
        0,
        verticalHeightMeasureSpec,
        mTitle.getMeasuredHeight());
```

```
    // 9. Update the sizes.
```

```
    width += Math.max(mTitle.getMeasuredWidth(),
        mSubTitle.getMeasuredWidth());
    height = Math.max(mTitle.getMeasuredHeight() +
        mSubTitle.getMeasuredHeight(), height);
```

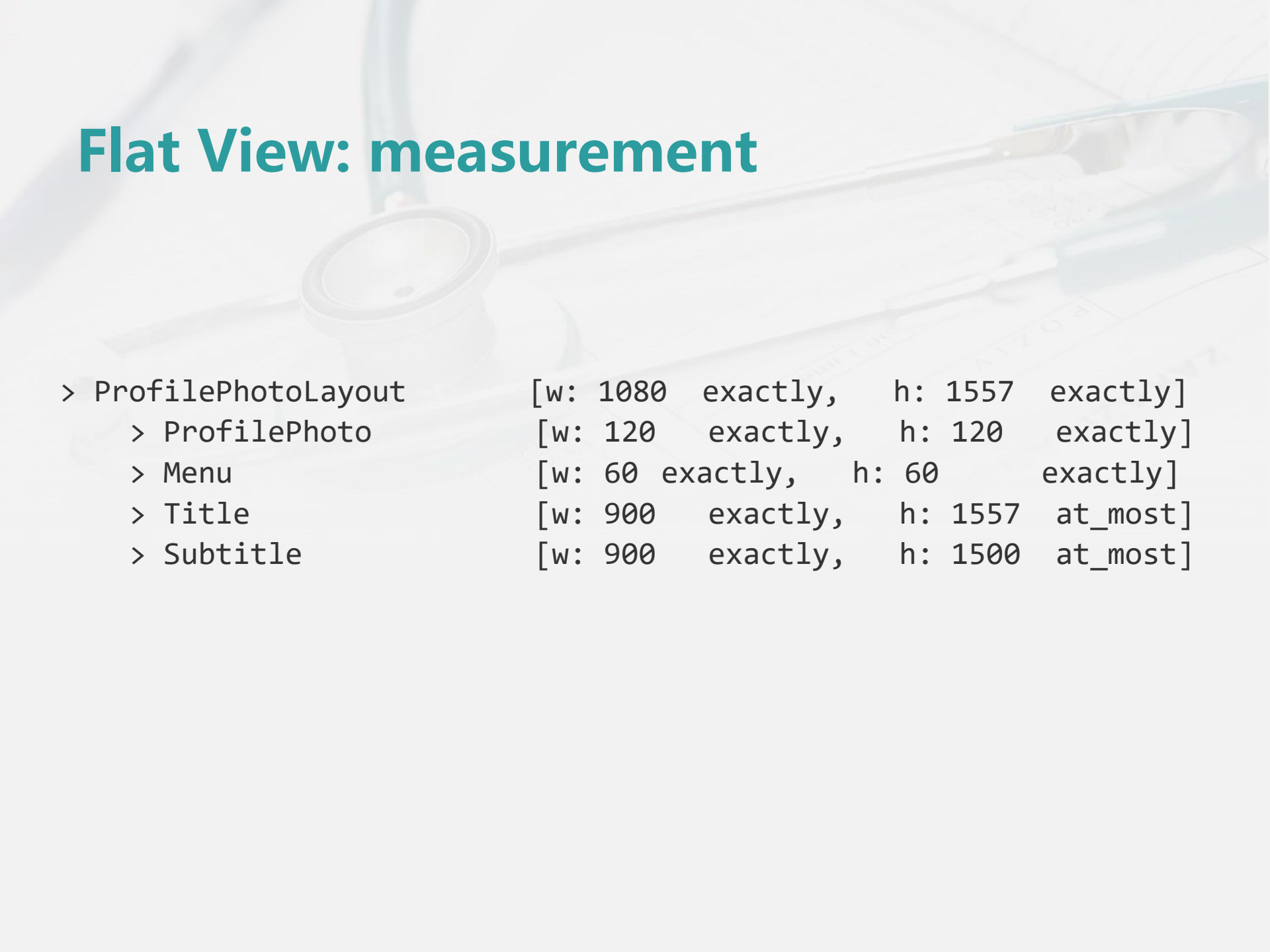
```
    // 10. Set the dimension for this ViewGroup.
```

```
    setMeasuredDimension(
        resolveSize(width, widthMeasureSpec),
        resolveSize(height, heightMeasureSpec));
```

```
}
```



# Flat View: measurement



> ProfilePhotoLayout	[w: 1080 exactly, h: 1557 exactly]
> ProfilePhoto	[w: 120 exactly, h: 120 exactly]
> Menu	[w: 60 exactly, h: 60 exactly]
> Title	[w: 900 exactly, h: 1557 at_most]
> Subtitle	[w: 900 exactly, h: 1500 at_most]

# 自定义View优化

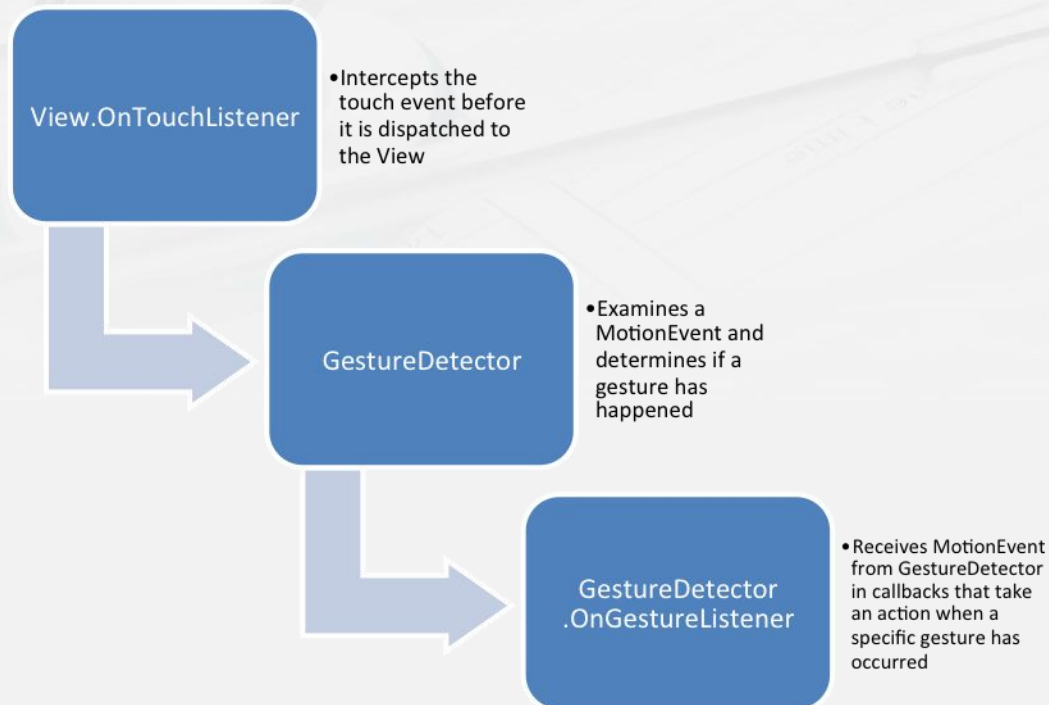
## 降低刷新频率

大部分时候调用 `onDraw()` 方法就是调用 `invalidate()` 的结果，所以减少不必要的调用 `invalidate()` 方法。有可能的，调用四种参数不同类型的 `invalidate()`，而不是调用无参的版本。无参变量需要刷新整个 view，而四种参数类型的变量只需刷新指定部分的 view。这种高效的调用更加接近需求，也能减少落在矩形屏幕外的不必要刷新的页面。

## 使用硬件加速

要注意使用的版本在 11 之上；  
注意使用手机 GPU 擅长的一些任务（测量，翻转，位移等），避免不擅长的操作（画直线或者曲线等）

# 手势操作的原理



# 常见类和接口、方法



## MotionEvent

这个类用于封装手势、触摸笔、轨迹球等等的动作事件。



## GestureDetector

识别各种手势。



## OnGestureListener

这是一个手势交互的监听接口，其中提供了多个抽象方法。



## 方法举例

onDown、onFling、onLongPress、onScroll、onShowPress

# 滑动冲突与解决方案

## ✚ 滑动的冲突

- 1) 滑动控件的嵌套
- 2) 纵向滑动嵌套横向滑动

## ✚ 冲突的解决方案

- 1) `public boolean dispatchTouchEvent(MotionEvent ev)` 这个方法用来分发 `TouchEvent`
- 2) `public boolean onInterceptTouchEvent(MotionEvent ev)` 这个方法用来拦截 `TouchEvent`
- 3) `public boolean onTouchEvent(MotionEvent ev)` 这个方法用来处理 `TouchEvent`

# 滑动事件的三种处理方式

1 |

ViewPager ( 官方支持库 )

2 |

ViewFlipper : 1 ) 静态加载 ; 2 ) 动态加载

3 |

ViewFlow ( 开源项目 )

# 嵌套滑动机制

谷歌在发布安卓 Lollipop 版本之后，为了更好的用户体验，Google 为 Android 的滑动机制提供了 NestedScrolling 特性。

NestedScrolling 提供了一套父 View 和子 View 滑动交互机制。要完成这样的交互，父 View 需要实现 NestedScrollingParent 接口，而子 View 需要实现 NestedScrollingChild 接口。

NestedScrollingChild  
NestedScrollingParent  
NestedScrollingChildHelper  
NestedScrollingParentHelper

NestedScrollingChild 当中的一系列方法，主要是做判断、分发、开始和停止等操作，大致的方法有以下几个：  
setNestedScrollingEnabled, isNestedScrollingEnabled, startNestedScroll, stopNestedScroll, dispatchNestedFling。



### 1) startNestedScroll

首先子 view 需要开启整个流程（内部主要是找到合适的能接受 nestedScroll 的 parent），通知父 View，我要和你配合处理 TouchEvent

### 2) dispatchNestedPreScroll

在子 View 的 onInterceptTouchEvent 或者 onTouch 中(一般在 MotionEvent.ACTION\_MOVE事件里)，调用该方法通知父 View 滑动的距离。该方法的第三第四个参数返回父 view 消费掉的 scroll 长度和子 View 的窗体偏移量。如果这个 scroll 没有被消费完，则子 view 进行处理剩下的一些距离，由于窗体进行了移动，如果你记录了手指最后的位置，需要根据第四个参数 offsetInWindow 计算偏移量，才能保证下一次的 touch 事件的计算是正确的。如果父 view 接受了它的滚动参数，进行了部分消费，则这个函数返回 true，否则为 false。

这个函数一般在子 view 处理 scroll 前调用。

### 3) dispatchNestedScroll

向父 view 汇报滚动情况，包括子 view 消费的部分和子 view 没有消费的部分。如果父 view 接受了它的滚动参数，进行了部分消费，则这个函数返回 true，否则为 false。

这个函数一般在子 view 处理 scroll 后调用。

### 4) stopNestedScroll

结束整个流程。

子view	父view
startNestedScroll	onStartNestedScroll、onNestedScrollAccepted
dispatchNestedPreScroll	onNestedPreScroll
dispatchNestedScroll	onNestedScroll
stopNestedScroll	onStopNestedScroll



# THANKS

谢 谢 聆 听