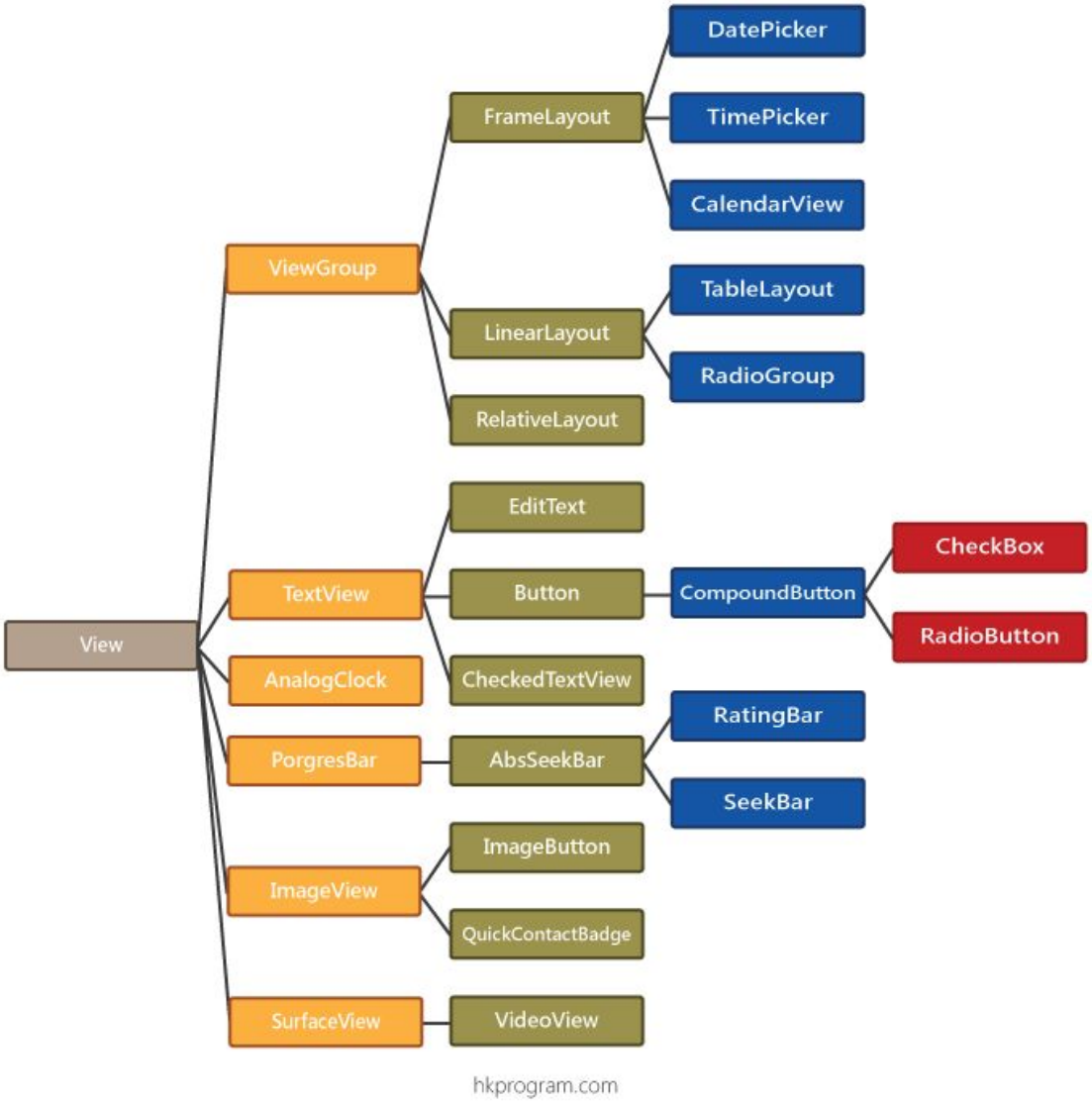
View 自定义及滑动

一、View 的自定义

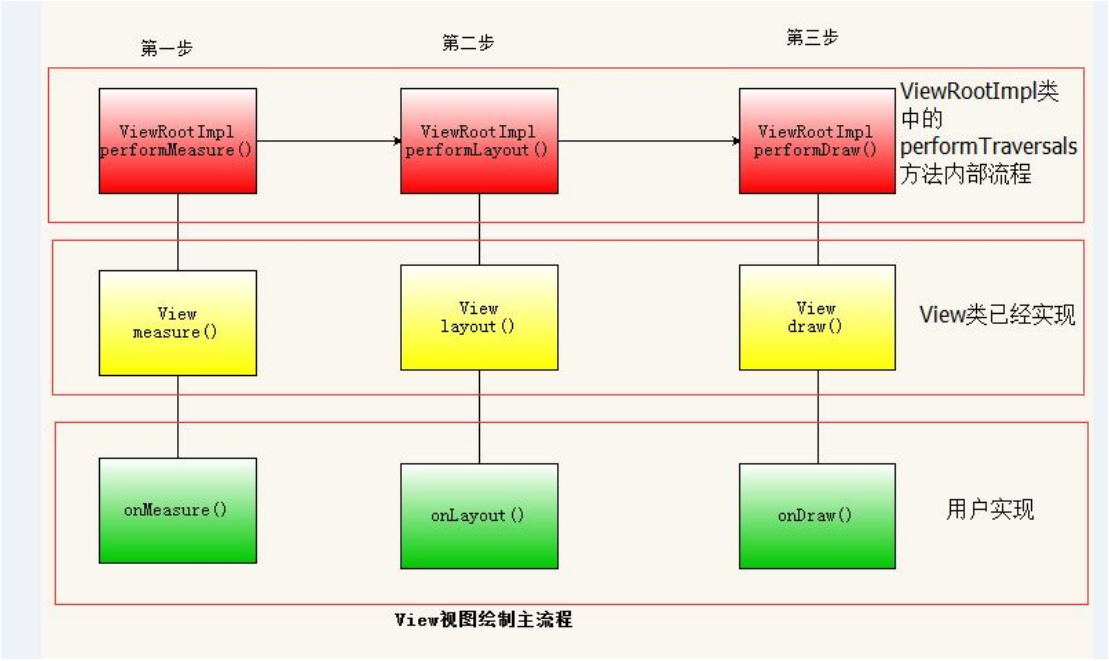
1、自定义 View 的核心

1.1 继承 View 完全自定义或继承 View 的派生子类
1.2 必须提供一个能够获取 Context 和作为属性的 AttributeSet 对象的构造函数。
1.3 重写三个重要的方法：onLayout() onMeasure() onDraw()

View 的家族体系，如下：



View 的绘制流程图示：



2、自定义 View 的属性

2.1 自定义 View 的步骤

1）继承一个 View（自定义一个 View 类）
2）编写 values/attrs.xml，定义自定义属性
3）在布局文件中的嵌入自定义 view，使用自定义的属性（注意 namespace）
4）应用自定义的属性，在 CustomView 的构造方法中通过 TypedArray 获取
5）添加属性和行为，暴露出一些合适的 getter 与 setter 方法
6）（设计可访问性：比如为残障人士等的一些特殊设计）

2.2 View 属性的关键点

1）要清楚 format 格式类型（自定义的属性）
2）三个构造函数的区别和使用范围
3）使用 LayoutParams 的添加控件到 ViewGroup
4）引用自定义 View 的属性，注意控件属性的引用空间
①在 Eclipse 中：需要跟上完整的包名
xmlns:custom="http://schema.android.com/apk/res/com.zanelove.topbardemo //完整的包名
②在 Studio 中：xmlns:custom="http://schema.android.com/apk/res-auto"//只需这样写即可

https://www.zybuluo.com/linux1s1s/note/104916

2.3 自定义属性的类型

format 表示的属性类型可以为 boolean, string, integer, dimension, float, reference, color, fraction, enum, flag 及其混合。
(1) boolean 表示布尔值，调用如 xx:attr1="false"
(2) integer 表示整型，调用如 xx:attr1="1"
(3) dimension 表示尺寸值，调用如 xx:attr1="42dp"
(4) float 表示浮点型，调用如 xx:attr1="0.7"
(5) color 表示颜色值，调用如 xx:attr1="#00FF00"
(6) string 表示字符串，调用如 xx:attr1="#adbddd"
(7) reference 表示参考某一资源 id，调用如 xx:attr1 = "@drawable/图片 ID"
(8) fraction 表示百分数，调用如 xx:attr1="30%"
以上类型定义都为<attr name="attr1" format="xxxtype"/>
属性的详细解析，可以查看如下文档：
http://stormzhang.com/android/2013/07/30/android-custome-attribute-format/

2.4 自定义 View 案例

源码介绍：

3、自定义 ViewGroup

Android 提供了几个 ViewGroup 如 LinearLayout, RelativeLayout, FrameLayout 来固定子 view 的位置。在这些普通的 ViewGroup 中有多种使用选择。

这样的一幅图示。使用 LinerLayout 布局方式

```
<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>
```

这里的测量情况如下：

```
> 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       ]
```

使用 RelativeLayout 方式

```
<RelativeLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <ProfilePhoto
        android:layout_width="40dp"
        android:layout_height="40dp"
```

```
        android:layout_alignParentTop="true"
        android:layout_alignParentLeft="true"/>

    <Menu
        android:layout_width="20dp"
        android:layout_height="20dp"
        android:layout_alignParentTop="true"
        android:layout_alignParentRight="true"/>

    <Title
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toRightOf="@id/profile_photo"
        android:layout_toLeftOf="@id/menu"/>

    <Subtitle
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/title"
        android:layout_toRightOf="@id/profile_photo"
        android:layout_toLeftOf="@id/menu"/>

</RelativeLayout>
```

测量情况如下：

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

我们从已知的约束条件开始 — 所有边的内边距，另外还需要考虑的约束是使用固定值的控件的高和宽。Android 提供了一个帮助方法 -measureChildWithMargins()用于测量 ViewGroup 内的子 view.然而它总是添加 padding 作为约束条件的一部分。 因此我们复写这个方法自己来管理这些约束条件。从测量 ProfilePhoto 开始，测量完成后更新一下 constraints。对 menu 按钮的测量亦 是如此。

现在还剩下 Title 和 Subtitle 的宽度没有测量。Android 还提供了另外一个帮助方法 -makeMeasureSpec()，用于构造 MeasureSpec,传入相应的 size 和 mode 返回一个 MeasureSpec。接下来我们传入 Title 和 Subtitle 可用的 width 和 height 及相应的 MeasureSpec 来测量 Title 和 Subtitle。最后更新一下 ViewGroup 的尺寸。在这一步可以明确每个 view 都只被测量一次。

https://docs.google.com/presentation/d/18Rfv4S8Eu_1a_lW9caEAi2ApncWf7Q-vj7EGKMnBTcs

/edit#slide=id.p

http://www.jcodecraeer.com/a/anzhuokaifa/androidkaifa/2015/0524/2920.html

深入理解自定义布局：
http://ju.outofmemory.cn/entry/103672

4、自定义 View 的优化

4.1 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。

4.2 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）

http://www.cnblogs.com/lhyz/p/4430409.html
http://blog.csdn.net/guolin_blog/article/details/17357967

http://hukai.me/android-training-course-in-chinese/ui/custom-view/index.html

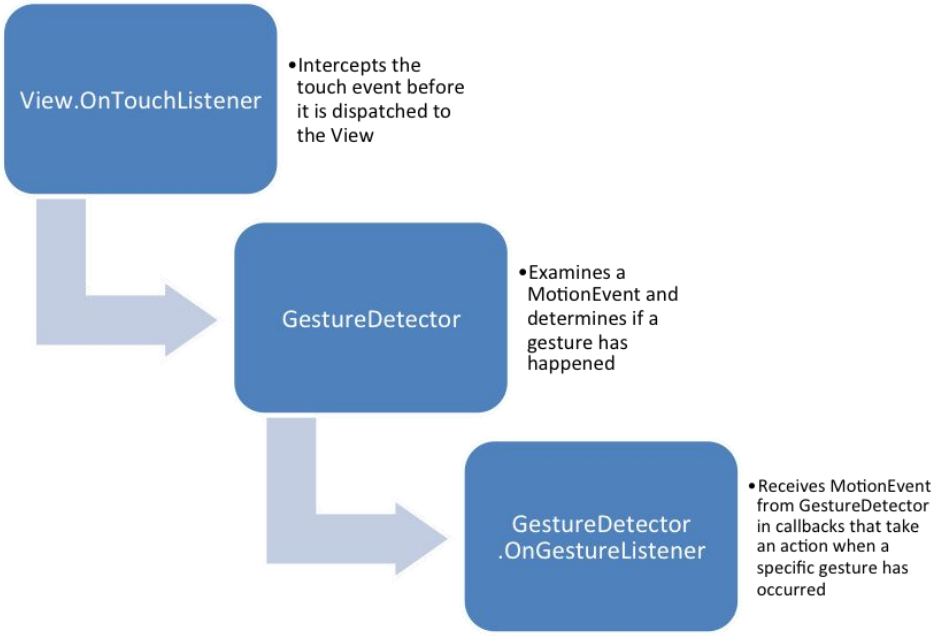
三、手势操作的原理

http://www.jcodecraeer.com/a/anzhuokaifa/androidkaifa/2014/1212/2145.html

1、手势操作原理

首先，在 Android 系统中，每一次手势交互都会依照以下顺序执行。
1. 接触接触屏一刹那，触发一个 MotionEvent 事件。

2. 该事件被 OnTouchListener 监听，在其 onTouch()方法里获得该 MotionEvent 对象。
3. 通过 GestureDetector（手势识别器）转发次 MotionEvent 对象至 OnGestureListener。
4. OnGestureListener 获得该对象，听根据该对象封装的的信息，做出合适的反馈。
这个顺序可以说就是手势操作的原理。



2、手势操作类、接口和方法

2.1 类和接口

下面一同来了解一下 MotionEvent、GestureDetector 和 OnGestureListener。

MotionEvent: 这个类用于封装手势、触摸笔、轨迹球等等的动作事件。其内部封装了两个重要的属性 X 和 Y，这两个属性分别用于记录横轴和纵轴的坐标。

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，并根据 GestureDetector 的手势识别结果调用相对应的方法。

接口：OnGestureListener 手势识别接口，OnDoubleTapListener 双击与单击识别接口
内部类:SimpleOnGestureListener 这个内部类是静态内部类，实现了 OnGestureListener, OnDoubleTapListener 接口，相当于是这两个接口功能的集合。

2.2 处理手势的方法

按下（onDown）： 刚刚手指接触到触摸屏的那一刹那，就是触的那一下。

抛掷（onFling）： 手指在触摸屏上迅速移动，并松开的动作。

长按（onLongPress）： 手指按在持续一段时间，并且没有松开。

滚动（onScroll）： 手指在触摸屏上滑动。

按住（onShowPress）： 手指按在触摸屏上，它的时间范围在按下起效，在长按之前。

抬起（onSingleTapUp）：手指离开触摸屏的那一刹那。

除了这些定义之外，鄙人也总结了一点算是经验的经验吧，在这里和大家分享一下。

任何手势动作都会先执行一次按下（onDown）动作。

长按（onLongPress）动作前一定会执行一次按住（onShowPress）动作。

按住（onShowPress）动作和按下（onDown）动作之后都会执行一次抬起（onSingleTapUp）动作。

长按（onLongPress）、滚动（onScroll）和抛掷（onFling）动作之后都不会执行抬起（onSingleTapUp）动作。

触摸事件的一个检测：
http://m.blog.csdn.net/blog/u010687392/43953455

三、滑动事件的冲突

1、冲突的场景

有一个横向滑动的控件，外层有一个纵向的滑动控件，这样在滑动的时候会出现一些冲突。
http://liucanwen.iteye.com/blog/2020004
http://www.cnblogs.com/tianzhijiexian/p/4397537.html

2、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

重写父控件的 onInterceptTouchEvent 事件并返回 false，这样就会传递事件给子控件

```
public boolean onInterceptTouchEvent(MotionEvent ev) {
    return false;
}
```

自定义可以滑动的子控件（继承 ListView、GridView、ScrollView 等），重写 dispatchTouchEvent 或者 onInterceptTouchEvent 事件，声明 getParent().requestDisallowInterceptTouchEvent(true); 让子控件处理自己的事件，父控件不响应，这样就不会有事件冲突了。

```
public boolean onTouch(View v, MotionEvent event) {
    switch (event.getAction()) {
    case MotionEvent.ACTION_MOVE:
        pager.requestDisallowInterceptTouchEvent(true);
        break;
    case MotionEvent.ACTION_UP:
    case MotionEvent.ACTION_CANCEL:
        pager.requestDisallowInterceptTouchEvent(false);
        break;
    }
}
```

http://www.jcodecraeer.com/a/anzhuokaifa/androidkaifa/2013/0803/1500.html

http://m.blog.csdn.net/blog/u014466785/39450771

四、滑动的三种处理

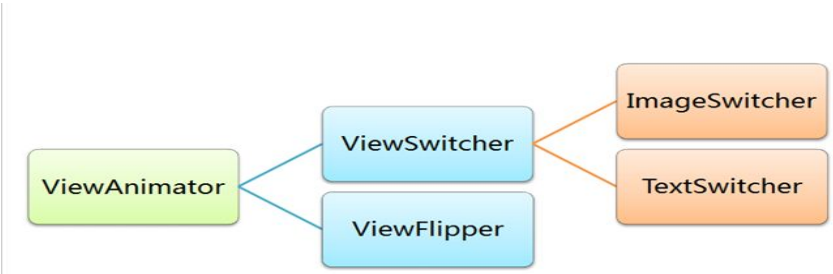
1、ViewPager

该类是一个布局管理器,它允许用户通过滑动左、右页的数据。你必须要一个实现了 PagerAdapter 接口从而生成的页面视图。

核心：1）ViewPager+Fragment 配合使用更好；2）每个页面的响应事件我们可以在 OnPageChangeListener 监听器类中进行捕获和处理对应事件。

2、ViewFlipper

ViewFilpper 控件是系统自带控件之一，主要用于在同一个屏幕间的切换及设置动画效果、间隔时间，且可以自动播放。



View 的动画图示，如上！

2.1 静态加载

如下是一个布局文件，包括四个不同的界面：

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <ViewFlipper
        android:id="@+id/body_flipper"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:background="#f0f0f0" >

         <include
            android:id="@+id/layout01"
            layout="@layout/page1" />
        <include
            android:id="@+id/layout02"
            layout="@layout/page2" />
        <include
            android:id="@+id/layout02"
            layout="@layout/page3" />
        <include
            android:id="@+id/layout02"
            layout="@layout/page4" />

    </ViewFlipper>
</RelativeLayout>
```

Java 代码中，处理方式，实现 OnTouchListener 和实例化一个 ViewFlipper，根据滑动手势的判断，调用 viewFlipper.showPrevious();或者 viewFlipper.showNext()分别显示上一个界面或者下一个界面。

2.2 动态加载

对于上面的静态加载的情况，我们写了四个界面存放在 xml 文件中，实际上可能目前的页面是不确定个数的，那么就可以采用动态加载的方式。
MyGestureListener 类：自定义滑动事件监听器，主要用来做监听事件的判断，左右滑动，上下滑动，然后定义滑动的回调接口，供外部访问。继承了 SimpleGestureListener 手势监听类，复写了该类 onFling()方法，用于监听用户按下滑动事件的处理；还自定义了滑动的回调接口 OnFlingListener（包含了两个抽象方法 flingToNext()，flingToPrevious）。
MyViewFlipper 类：自定义 View 滑动类，它会监听滑动事件，并做切换视图的处理。是一个自定义 ViewFlipper，该类首先实现和绑定了上一个类中的滑动的回调接口 OnFlingListener，完成了接口中两个重要的方法。同时定义了一个 View 变化监听回调接口 OnViewFlipperListener（包含了两个抽象方法 getNextView(),getPreviousView()）。

3、ViewFlow

它是 gethub 上的一个开源项目，利用 ViewFlow 可以产生视图切换的效果。ViewFlow 相当于 Android UI 部件提供水平滚动的 ViewGroup，使用 Adapter 进行条目绑定，例如 ViewPager 或是 ViewFlipper。
一般情况下，当你需要做一个滑动然而不确定 view 的数目时，可以考虑使用 ViewFlow。如果你的 view 数目确定，使用 Fragments 或兼容库里的 ViewPager 比较好。
开发 App 的同学对这个应该有所耳闻解或者了解，不做过多介绍，对于不太清楚的同学，提供如下的访问地址：https://github.com/pakerfeldt/android-viewflow

五、嵌套滑动机制

1、背景介绍

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。

在这之前，我们知道 Android 对 Touch 事件的分发是有自己一套机制的。主要是有是三个函数：
dispatchTouchEvent、onInterceptTouchEvent 和 onTouchEvent。
但是这里有一个问题，如果子 view 获得处理 touch 事件机会的时候，父 view 就再也没有机

会去处理这个 touch 事件了，直到下一次手指再按下。换句话说，我们在滑动子 View 的时候，如果子 View 对这个滑动事件不想要处理的时候，只能抛弃这个 touch 事件，而不会把这些传给父 view 去处理。
在新提供的中，我们需要关注这样的四个类（接口）：
NestedScrollingChild
NestedScrollingParent
NestedScrollingChildHelper
NestedScrollingParentHelper

2、嵌套机制原理

2.1 关键方法分析

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。
NestedScrollingChildHelper 是帮你实现一些跟 NestedScrollingParent 交互的一些方法。

```
/**
     * Start a new nested scroll for this view.
     *
     * <p>This is a delegate method. Call it from your {@link android.view.View View} subclass
     * method/{@link NestedScrollingChild} interface method with the same signature to implement
     * the standard policy.</p>
     *
     * @param axes Supported nested scroll axes.
     *                 See {@link NestedScrollingChild#startNestedScroll(int)}.
     * @return true if a cooperating parent view was found and nested scrolling started successfully
     */
    public boolean startNestedScroll(int axes) {
        if (hasNestedScrollingParent()) {
            // Already in progress
            return true;
        }
        if (isNestedScrollingEnabled()) {
            ViewParent p = mView.getParent();
            View child = mView;
            while (p != null) {
                if (ViewParentCompat.onStartNestedScroll(p, child, mView, axes)) {
                    mNestedScrollingParent = p;
```

```
                    ViewParentCompat.onNestedScrollAccepted(p, child, mView, axes);
                    return true;
                }
                if (p instanceof View) {
                    child = (View) p;
                }
                p = p.getParent();
            }
        }
        return false;
    }
```

ViewParentCompat 是一个和父 view 交互的兼容类，它会判断 api version，如果在 Lollipop 以上，就是用 view 自带的方法，否则判断是否实现了 NestedScrollingParent 接口，去调用接口的方法。

2.2 整个流程分析

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 MontionEvent.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。

一般是子 view 发起调用，父 view 接受回调。

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

3、核心实现

实现 NestedScrollingChild

首先来说 NestedScrollingChild。如果你有一个可以滑动的 View，需要被用来作为嵌入滑动的子 View，就必须实现本接口。在此 View 中，包含一个 NestedScrollingChildHelper 辅助类。NestedScrollingChild 接口的实现，基本上就是调用本 Helper 类的对应的函数即可，因为 Helper 类中已经实现好了 Child 和 Parent 交互的逻辑。原来的 View 的处理 Touch 事件，并实现滑动的逻辑大体上不需要改变。

需要做的就是，如果要准备开始滑动了，需要告诉 Parent，你要准备进入滑动状态了，调用 startNestedScroll()。你在滑动之前，先问一下你的 Parent 是否需要滑动，也就是调用 dispatchNestedPreScroll()。如果父类滑动了一定距离，你需要重新计算一下父类滑动后剩下给你的滑动距离余量。然后，你自己进行余下的滑动。最后，如果滑动距离还有剩余，你就再问一下，Parent 是否需要在继续滑动你剩下的距离，也就是调用 dispatchNestedScroll()。

实现 NestedScrollingParent

作为一个可以嵌入 NestedScrollingChild 的父 View，需要实现 NestedScrollingParent，这个接口方法和 NestedScrollingChild 大致有一一对应的关系。同样，也有一个 NestedScrollingParentHelper 辅助类来默默的帮助你实现和 Child 交互的逻辑。滑动动作是 Child 主动发起，Parent 就收滑动回调并作出响应。

从上面的 Child 分析可知，滑动开始的调用 startNestedScroll()，Parent 收到 onStartNestedScroll() 回调，决定是否需要配合 Child 一起进行处理滑动，如果需要配合，还会回调 onNestedScrollAccepted()。

每次滑动前，Child 先询问 Parent 是否需要滑动，即 dispatchNestedPreScroll()，这就回调到 Parent 的 onNestedPreScroll()，Parent 可以在这个回调中“劫持”掉 Child 的滑动，也就是先于 Child 滑动。

Child 滑动以后，会调用 onNestedScroll()，回调到 Parent 的 onNestedScroll()，这里就是 Child 滑动后，剩下的给 Parent 处理，也就是 后于 Child 滑动。
最后，滑动结束，调用 onStopNestedScroll() 表示本次处理结束。

4、源码实战

案例参考：https://github.com/race604/FlyRefresh

http://segmentfault.com/a/1190000002873657
http://www.race604.com/android-nested-scrolling/