**CoordinatorLayout的了解**

CoordinatorLayout 是一个容器，实现了 NestedScrollingParent 接口。

这就涉及到了两个接口：NestedScrollingParent, NestedScrollingChild。Android 就是通过这两个接口, 来实现 子 View 与父View 之间的嵌套滑动。NestedScrolling 机制能够让父 View 和子 View 在滚动式进行配合，而要实现这样的交互机制，首先父 view 要实现 NestedScrollingParent 接口，而子 View 需要实现 NestedScrollingChild 接口，在这套机制中子 View是发起者，父 view 是接受回调并做出响应的。

**一．NestedScrolling机制：**

主要接口

NestedScrollingChild

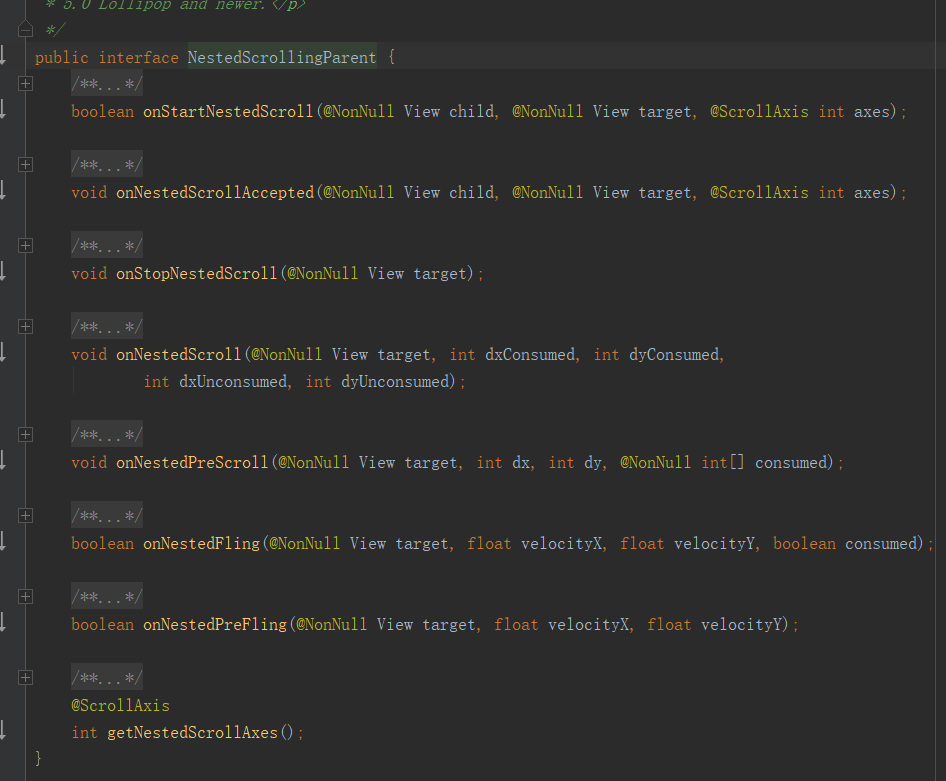
NestedScrollingParent

帮助类

NestedScrollingChildHelper

NestedScrollingParentHelper

NestedScrollingParent接口，实现它需要实现如下方法：

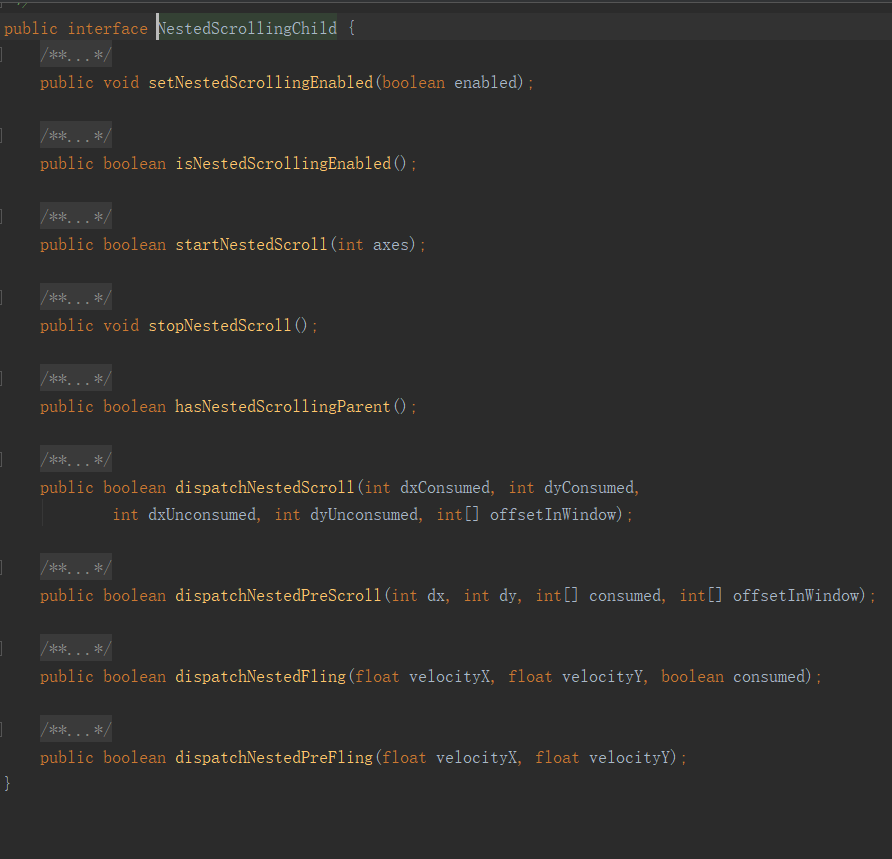


onStartNestedScroll该方法，一定要按照自己的需求返回true，该方法决定了当前控件是否能接收到其内部View(非并非是直接子View)滑动时的参数；假设你只涉及到纵向滑动，这里可以根据nestedScrollAxes这个参数，进行纵向判断。

onNestedPreScroll该方法的会传入内部View移动的dx,dy，如果你需要消耗一定的dx,dy，就通过最后一个参数consumed进行指定，例如我要消耗一半的dy，就可以写consumed[1]=dy/2

onNestedFling你可以捕获对内部View的fling事件，如果return true则表示拦截掉内部View的事件。

NestedScrollingChild接口，实现它需要实现如下方法：



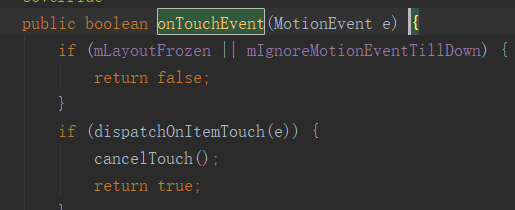
NestedScrolling机制原理：

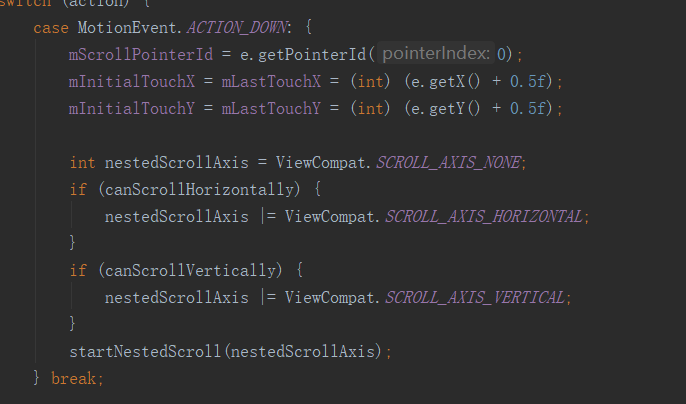
外部View实现NestedScrollingParent，比如CoordinatorLayout，然后会配合一个可滑动的内部view一起使用，该子View会实现NestedScrollingChild接口，比如RecyclerView。

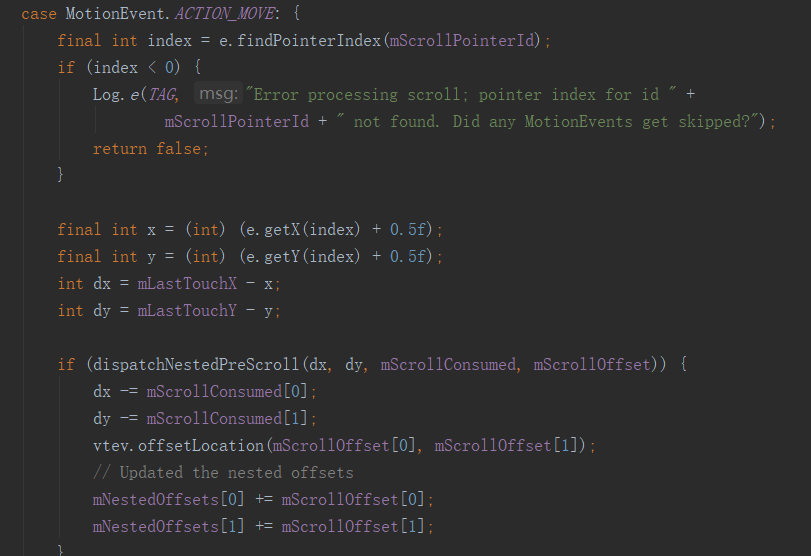
NestedScrollingParent内部的View，在滑动到时候，会首先将dx、dy传入给NestedScrollingParent，NestedScrollingParent可以决定是否对其进行消耗，一般会根据需求消耗部分或者全部(不过这里并没有实际的约束，你可以随便写消耗多少，可能会对内部View造成一定的影响）。事件分发是这样的：子View首先得到事件处理权，处理过程中，父View可以对其拦截，但是拦截了以后就无法再还给子View（本次手势内）。

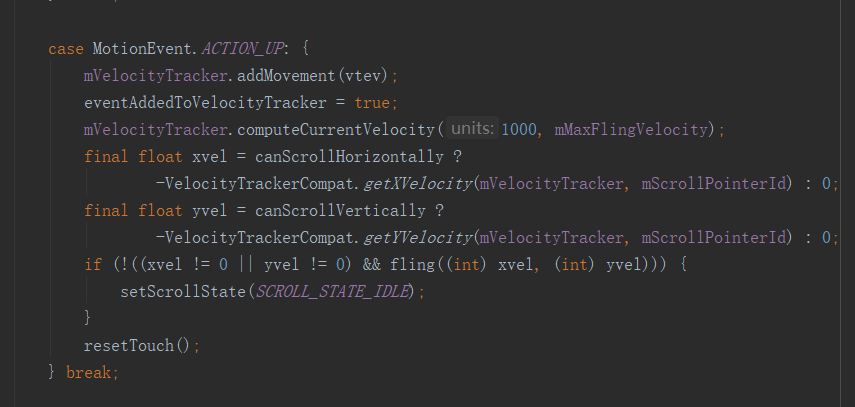
NestedScrolling机制是这样的：内部View在滚动的时候，首先将dx,dy交给NestedScrollingParent，NestedScrollingParent可对其进行部分消耗，剩余的部分还给内部View。

原理其实就是看内部View什么时候回调NestedScrollingParent各种方法的，直接定位到内部View的onTouchEvent：









1. 在 Action\_Down 的时候，Scrolling child 会调用 startNestedScroll 方法，通过 childHelper 回调 Scrolling Parent 的 startNestedScroll 方法
2. 在 Action\_move 的时候，Scrolling Child 要开始滑动的时候，会调用dispatchNestedPreScroll 方法，通过 ChildHelper 询问 Scrolling Parent 是否要先于 Child 进行 滑动，若需要的话，会调用 Parent 的 onNestedPreScroll 方法，协同 Child 一起进行滑动
3. 当 ScrollingChild 滑动完成的时候，会调用 dispatchNestedScroll 方法，通过 ChildHelper 询问 Scrolling Parent 是否需要进行滑动，需要的话，会 调用 Parent 的 onNestedScroll 方法
4. 在 Action\_down,Action\_move 的时候，会调用 Scrolling Child 的stopNestedScroll ，通过 ChildHelper 询问 Scrolling parent 的 stopNestedScroll 方法。
5. 如果需要处理 Fling 动作，我们可以通过 VelocityTrackerCompat 获得相应的速度，并在 Action\_up 的时候，调用 dispatchNestedPreFling 方法，通过 ChildHelper 询问 Parent 是否需要先于 child 进行 Fling 动作
6. 在 Child 处理完 Fling 动作时候，如果 Scrolling Parent 还需要处理 Fling 动作，我们可以调用 dispatchNestedFling 方法，通过 ChildHelper ，调用 Parent 的 onNestedFling 方法

## ****NestedScrollingChild****

目前已知的实现子类有 HorizontalGridView, NestedScrollView, RecyclerView, SwipeRefreshLayout, VerticalGridView

* boolean startNestedScroll(int axes)

在开始滑动的时候会调用这个方法，axes 代表滑动的方向，ViewCompat.SCROLL\_AXIS\_HORIZONTAL 代表水平滑动，ViewCompat.SCROLL\_AXIS\_VERTICAL 代表垂直滑动，

返回值是布尔类型的，根据返回值，我们可以判断是否找到支持嵌套滑动的父View ，返回 true，表示在scrolling parent （需要注意的是这里不一定是直接scrolling parent ，间接scrolling parent 也可会返回 TRUE） 中找到支持嵌套滑动的。反之，则找不到。

* boolean dispatchNestedPreScroll(int dx, int dy, int[] consumed, int[] offsetInWindow)

在scrolling child 滑动之前，提供机会让scrolling parent 先于scrolling child滑动。

dx，dy 是输入参数，表示scrolling child 传递给 scrolling parent 水平方向，垂直方向上的偏移量，consumed 是输出参数，consumed[0] 表示父 View 在水平方向上消费的值，，consumed[1 表示父 View 在垂直方向上消费的值。

返回值也是布尔类型的，根据这个值 ，我们可以判断scrolling parent 是都消费了相应距离 。

* boolean dispatchNestedScroll(int dxConsumed, int dyConsumed, int dxUnconsumed, int dyUnconsumed, int[] offsetInWindow)

在scrolling child 滑动之后，调用这个方法，提供机会给scrolling parent 滑动，dxConsumed，dyConsumed 是输入参数，表示scrolling child 在水平方向，垂直方向消耗的值，dxUnconsumed，dyUnconsumed 也是输入参数，表示scrolling child 在水平方向，垂直方向未消耗的值。

* boolean dispatchNestedPreFling(float velocityX, float velocityY, boolean consumed)

调用这个方法，在scrolling child 处理 fling 动作之前，提供机会scrolling parent 先于scrolling child 处理 fling 动作。

三个参数都是输入参数，velocityX 表示水平方向的速度，velocityY 表示垂直方向感的速度，consumed 表示scrolling child 是否消费 fling 动作 。

返回值也是布尔类型的，表示scrolling parent 是否有消费了fling 动作或者对 fling 动作做出相应的 处理。true 表示有，false 表示没有。

* boolean dispatchNestedFling(float velocityX, float velocityY, boolean consumed)

在 Scrolling child 处理 fling 动作之后，提供机会给 Scrolling Parent 处理 fling 动作。各个参数的意义这里就不再意义阐述了，跟 dispatchNestedFling 参数的意义是一样的。

* void stopNestedScroll

当嵌套滑动的时候，会调用这个方法。

在 RecyclerView 中，当 Action\_UP 或者 Actioon\_cancel 或者 item 消费了 Touch 事件的时候，会调用这个方法。

## ****NestedScrollingParent****

Android 中已知的实现子类有 CoordinatorLayout, NestedScrollView, SwipeRefreshLayout。它通常是配合 NestedScrollingChild 进行嵌套滑动的。

* boolean onStartNestedScroll(View child, View target, int nestedScrollAxes)

在 Scrolling Child 开始滑动的时候会调用这个方法

当 Scrolling Child 调用 onStartNestedScroll 方法的时候，通过 NestedScrollingChildHelper 会回调 Scrolling parent 的 onStartNestedScroll 方法，如果返回 true， Scrolling parent 的 onNestedScrollAccepted(View child, View target, int nestedScrollAxes) 方法会被回调。

target 表示发起滑动事件的 View，Child 是 ViewParent 的直接子View，包含 target，nestedScrollAxes 表示滑动方向。

* void onNestedScrollAccepted(View child, View target, int nestedScrollAxes)

如果 Scrolling Parent 的onStartNestedScroll 返回 true， Scrolling parent 的 onNestedScrollAccepted(View child, View target, int nestedScrollAxes) 方法会被回调。

* boolean onNestedPreScroll(View target, int dx, int dy, int[] consumed)

在 Scrolling Child 进行滑动之前，Scrolling Parent 可以先于Scrolling Child 进行相应的处理

如果 Scrolling Child 调用 dispatchNestedPreFling(float velocityX, float velocityY) ，通过 NestedScrollingChildHelper 会回调 Scrolling parent 的 onNestedPreScroll 方法



startNestedScroll内部实际上如上图。通过Helper来实现该接口NestedScrollingChildHelper，去寻找NestedScrollingParent，然后回调onStartNestedScroll和onNestedScrollAccepted。

dispatchNestedPreScroll中会回调onNestedPreScroll方法，内部的scrollByInternal中还会回调onNestedScroll方法。

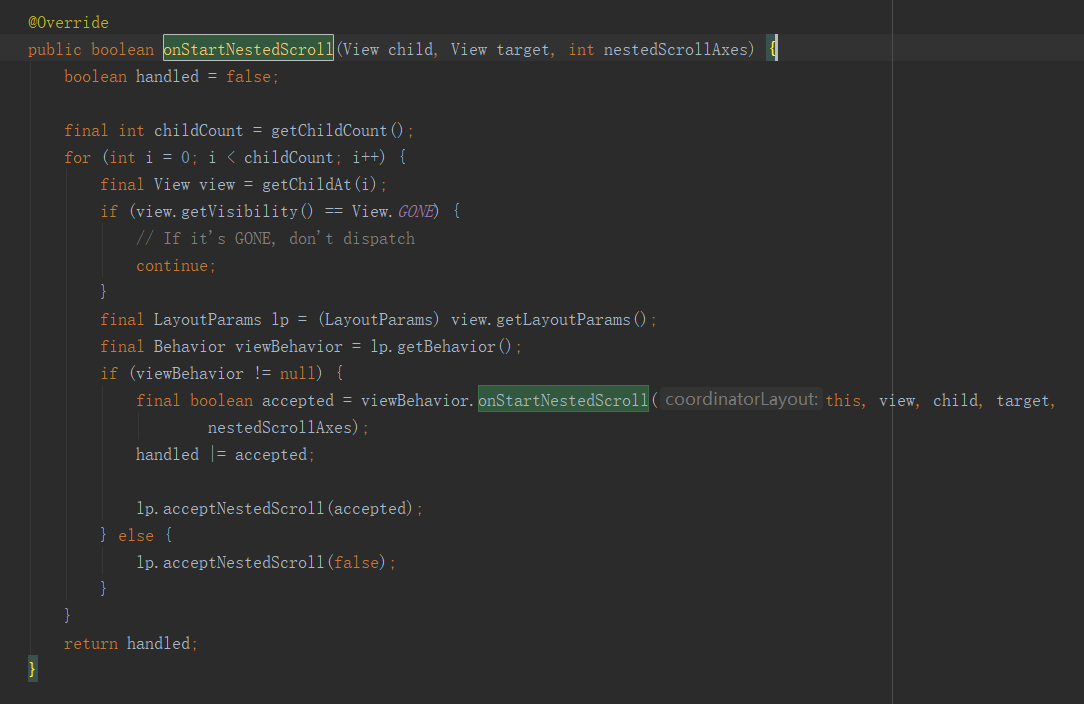
fling中会回调onNestedPreFling和onNestedFling方法。

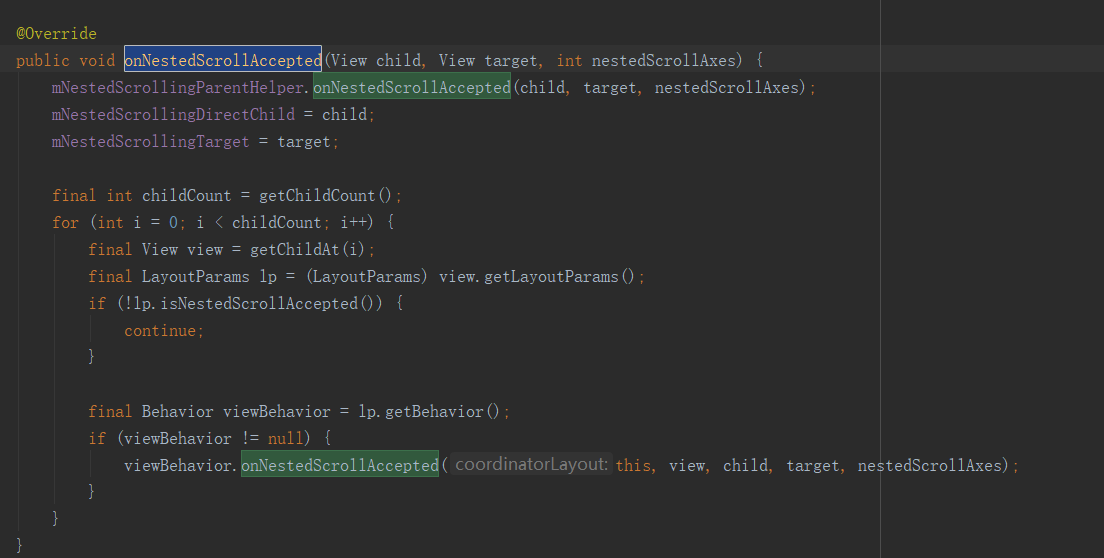
resetTouch中则会回调onStopNestedScroll。

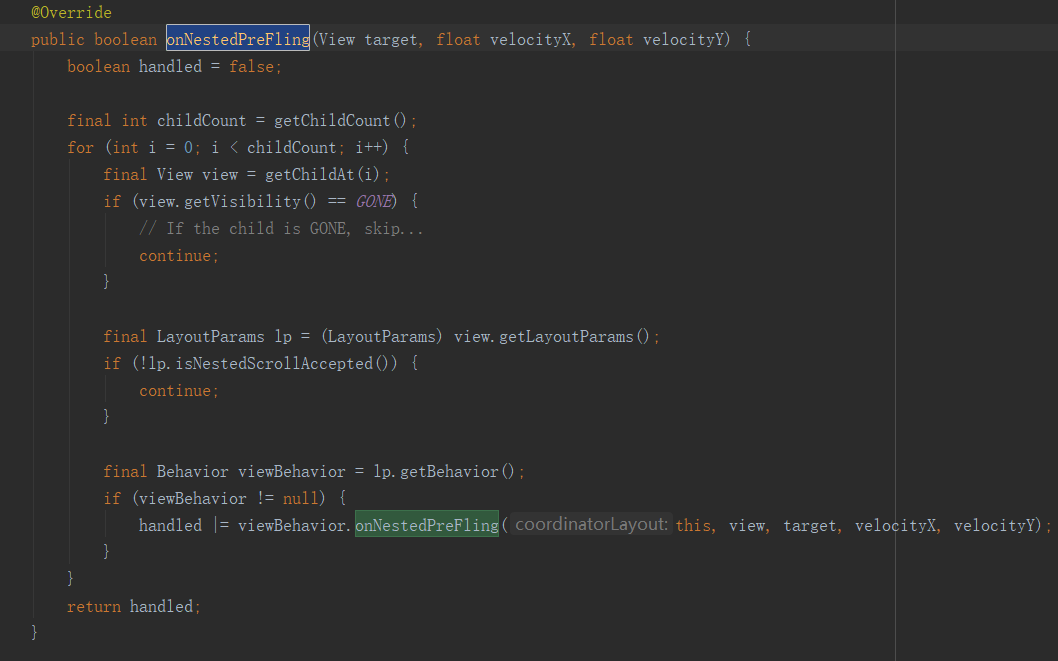
**Behavior ：**

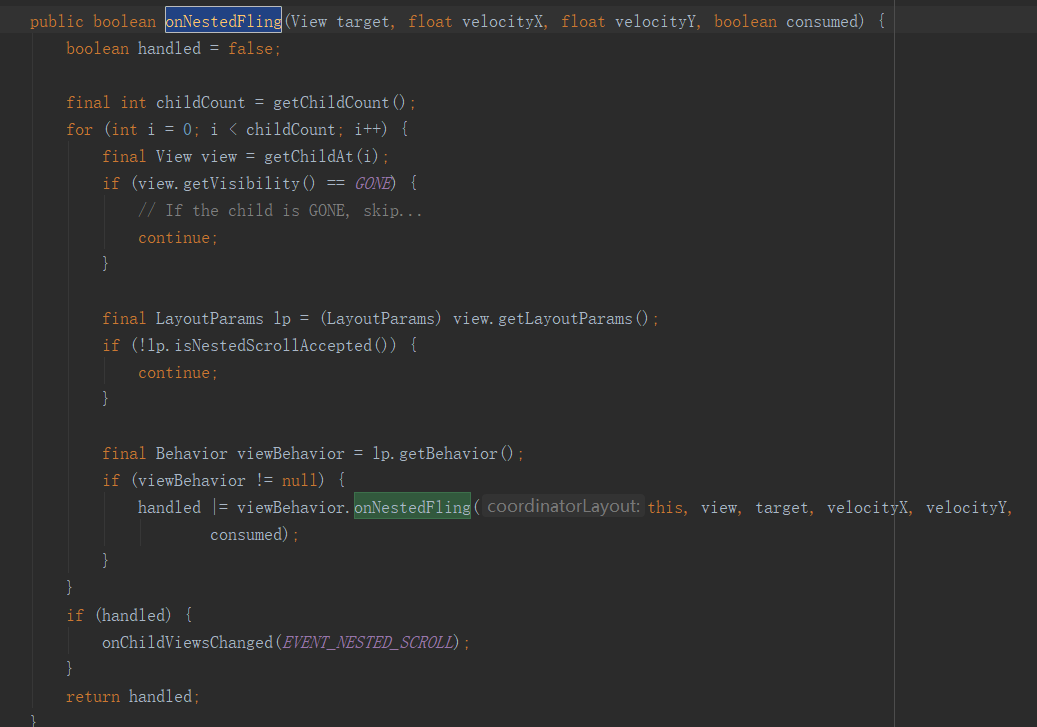
## Behavior 方法与 NestedScrollingParent 方法之间的关系：

NestedScrollingParent 中有onStartNestedScroll、onNestedScrollAccepted， onNestedPreScroll， onNestedScroll， onNestedPreFling等方法，在 Behavior 方法里面也有这些方法，几乎也是一一对应的。在 CoordinatorLayout 里面。NestedScrollingParent 接口的方法的具体 实现逻辑 都会交给 Behavior 对应的方法去处理。



****



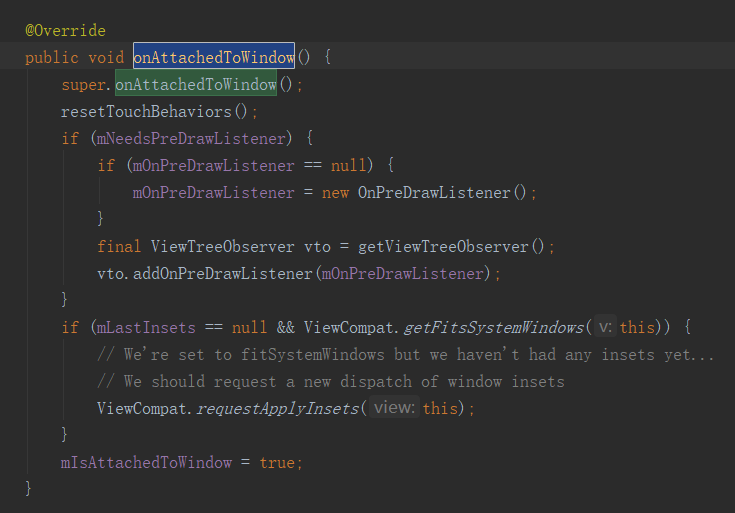


### **CoordinatorLayout 是监听 View 的状态**

public boolean layoutDependsOn(CoordinatorLayout parent, V child, View dependency)；

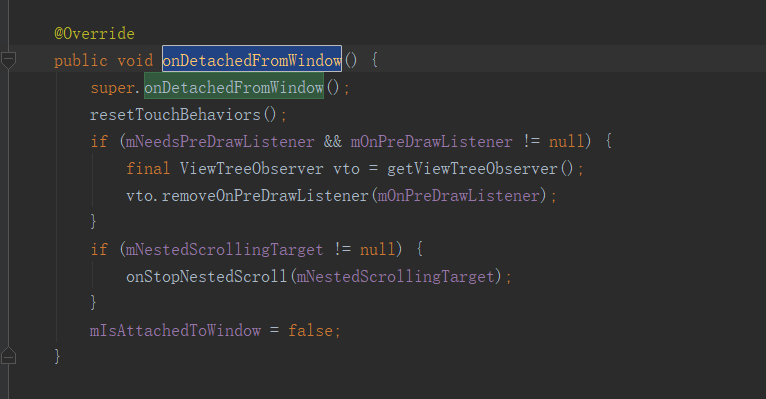
返回 true，当 dependency 改变的 时候，将会回调 onDependentViewChanged 方法，onDependentViewRemove ，当 dependency 移除的时候，会回调这个方法。

onDependentViewChanged 和 onDependentViewRemove 这两个方法是如何监听得到 View 变化和移除的？其实是在 onAttachedToWindow 方法里面，他会为 ViewTreeObserver 视图树添加 OnPreDrawListener 监听。



在 OnPreDrawListener 监听里面会调用 onChildViewsChanged 方法，在该方法里面会根据 View的状态回调 onDependentViewRemoved 或者 onDependentViewChanged 方法。

我们知道当 View 被销毁的时候，会回调 onDetachedFromWindow 这个方法，因此适合在这个方法里面移除 View 视图树的 PreDrawListener 监听。



**二．CoordinatorLayout + AppBarLayout + Toolbar**

其中AppBarLayout可以实现随着recyclerView滑出屏幕的效果。AppBarLayout必须作为CoordinatorLayout的直接子View，否则它的大部分功能将不会生效，如layout\_scrollFlags等。

scrollFlags，通过设置它的值可以实现不同的滚动模式，有四种值

scroll ，滚动。所有的Flag都要设置这个值，设置了之后可以向上滚动出屏幕。

enterAlways ，设置了这个值的话，该View会在向下滑动的时候立刻显示出来。

exitUntilCollapsed ，向上滑动时，所有组件都会滚出屏幕，但Toolbar除外。

enterAlwaysCollapsed ，如果你的View设置了最小高度（minHeight）,该View只会以这个最小高度滚出屏幕。

1. **CoordinatorLayout+CollapsingToolbarLayout实现视差滚动效果**

layout\_collapseMode，设置折叠模式，设置 parallax 为折叠，Pin 是不折叠

app:layout\_collapseMode="parallax"

视差效果中：

app:contentScrim="?attr/colorPrimary"

app:expandedTitleMarginEnd="64dp"

app:expandedTitleMarginStart="48dp"

contentScrim：作用是当整个视图收缩时，整个视图的颜色。  
expandedTitleMarginStart：设置Tittle文本的边距，当视图收缩后，Tittle离左边的距离  
expandedTitleMarginStart：设置Tittle文本的边距，当视图扩展后，Tittle离左边的距离

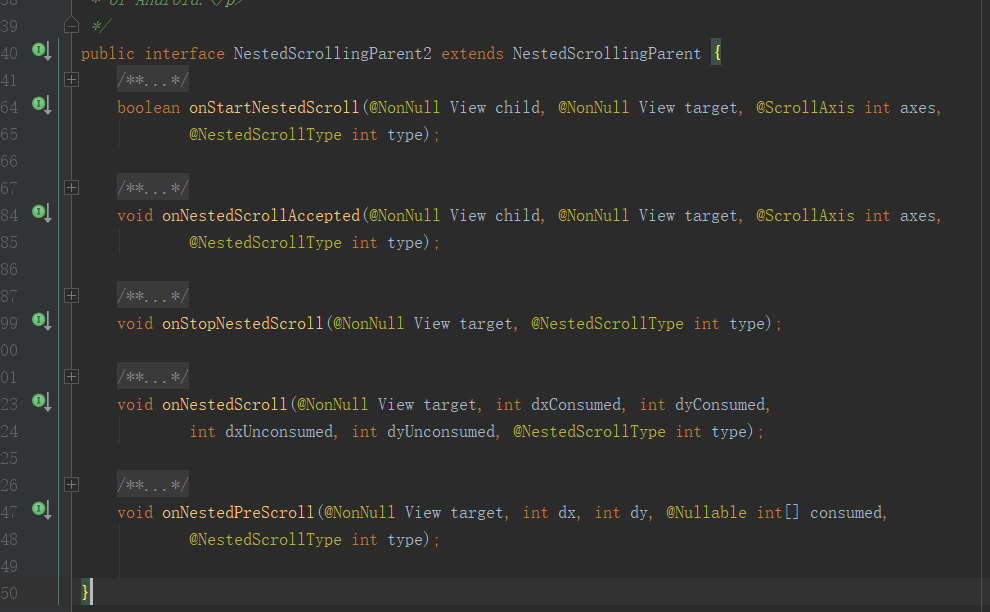
图片视差中：

app:layout\_collapseParallaxMultiplier=”0.7”

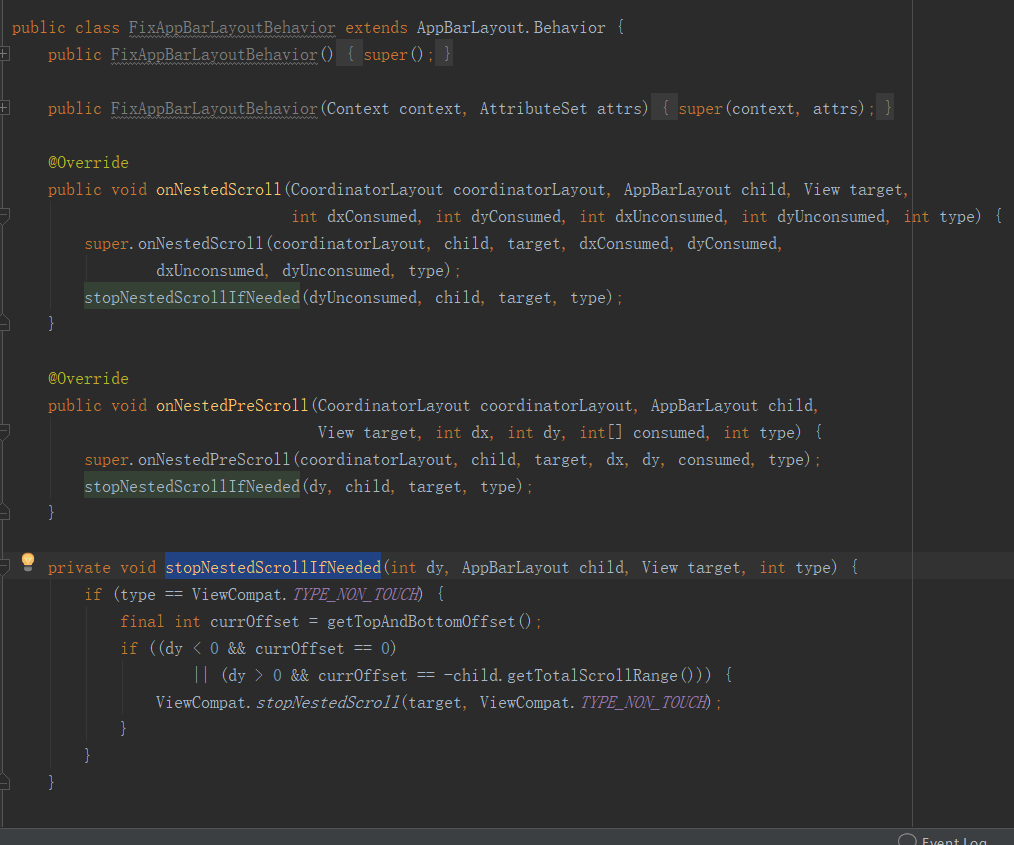
layout\_collapseParallaxMultiplier：视差滚动因子，自动收缩的比例值。当手指操作收缩到宽展的70%时，放开会自动收缩。

**CoordinatorLayout使用的坑：**

1. 使用过程中，发现CoordinatorLayout和recyclerview存在明显的滑动冲突，但是当升级com.android.support:design:25.2.0以上时，就不存在滑动冲突了。分别使用两个版本，发现26.1.0版本以后，CoordinatorLayout实现的接口不再是NestedScrollingParent，而是NestedScrollingParent2，这个其实是NestedScrollingParent的一个子类，就是用来解决滑动冲突的。区别就是多了一个type参数，这个参数就是区分滑动的。所以，我们现在最好是直接用25.2.0以上的版本。



1. 当使用26.1.0版本以上时，滑动冲突时解决了，但是又存在另外一个bug，当滑动到顶部后，点击屏幕任意一个view，不生效。点击第二次生效。这是修复之前版本滑动冲突时候引入的bug，原因是点击时，其实CoordinatorLayout还在判断滑动。之前说过NestedScrollingParent 会把对于的逻辑交给Behavior处理，我们只需要自定义一个Behavior，在页面滑到最顶部的时候，手动去stopNestedScroll即可。



1. 当使用26.1.0版本以上时，还有一个bug。当recyclerView 的Item比较少时，当AppBarLayout上滑出界面后，下滑时，recyclerView的下拉和AppBarLayout下滑存在冲突，当item很多时则不会出现该bug。

修复办法：给AppBarLayout添加一个OnOffsetChangedListener的监听，只有当AppBarLayout的滑动距离变为0时，下拉刷新才可用即可。

https://blog.csdn.net/gdutxiaoxu/article/details/71616547