探究 Behavior 的真实面目

推酷 (http://www.tuicool.com/)
时间 2016-06-05 20:23:09 zjutkz's blog (/sites/rmimaqf)

原文
http://zjutkz.net/2016/06/05/探究Behavior的真实面目/ (http://zjutkz.net/2016/06/05/探究Behavior的真实面目/?utm_source=tuicool&utm_medium=referral)
主题 安卓开发 (/topics/11080010)
v7包下的组件类似CoordinatorLayout推出也有一段时间了，大家使用的时候应该会体会到其中很多的便利，今天这篇文章带大家来了解一个比较重要的ui组件——Behavior。从字面意思上就可以看出它的作用，就是用来规定某些组件的行为的，那它到底是什么，又该怎么用呢？看完这篇文章希望大家会有自己的收获～

前言

写这篇文章的起因是因为我无意中在GitHub上发现了 Jake Wharton (https://github.com/JakeWharton) 大神新建了一个Repo，内容是 JakeWharton/DrawerBehavior (https://github.com/JakeWharton/DrawerBehavior) 。有兴趣的同学可以去看看，其实就是通过Behavior去构造一个类似于DrawerLayout的布局。想了想已经挺长时间没有搞ui方面的代码了，所以趁着这个机会复习了一下，顺便写一篇文章巩固，也给想要了解这方面内容的同学一个平台吧。

Behavior是什么

在文章的开始，我们先要了解什么是Behavior。

```
/**
 * Interaction behavior plugin for child views of {@link CoordinatorLayout}.
 *
 * <p>A Behavior implements one or more interactions that a user can take on a child view.
 * These interactions may include drags, swipes, flings, or any other gestures.</p>
 *
 * @param <V> The View type that this Behavior operates on
 */
public static abstract class Behavior<V extends View> {
    ...........

    public boolean onTouchEvent(CoordinatorLayout parent, V child, MotionEvent ev)         {
        return false;
    }

    public boolean onLayoutChild(CoordinatorLayout parent, V child, int layoutDirection) {
        return false;
    }

    ...........
}
```

它是CoordinatorLayout的内部类，从它的注释和其中的方法可以看出来，它其实就是给CoordinatorLayout的子View提供了一些交互的方法，用来规范它们的交互行为，比如上面出现的onTouchEvent可以用来规范子View的触摸事件，onLayoutChild可以用来规范子View的布局。

说到这里，大家可能会有一个问题，CoordinatorLayout又是个什么东西？

```
public class CoordinatorLayout extends ViewGroup implements NestedScrollingParent {
}
```

可以看出，它其实就是一个ViewGroup，实现了NestedScrollingParent用来执行嵌套滑动。至于嵌套滑动的机制大家可以看我博客的第一篇文章，这不是我们这篇文章的重点。

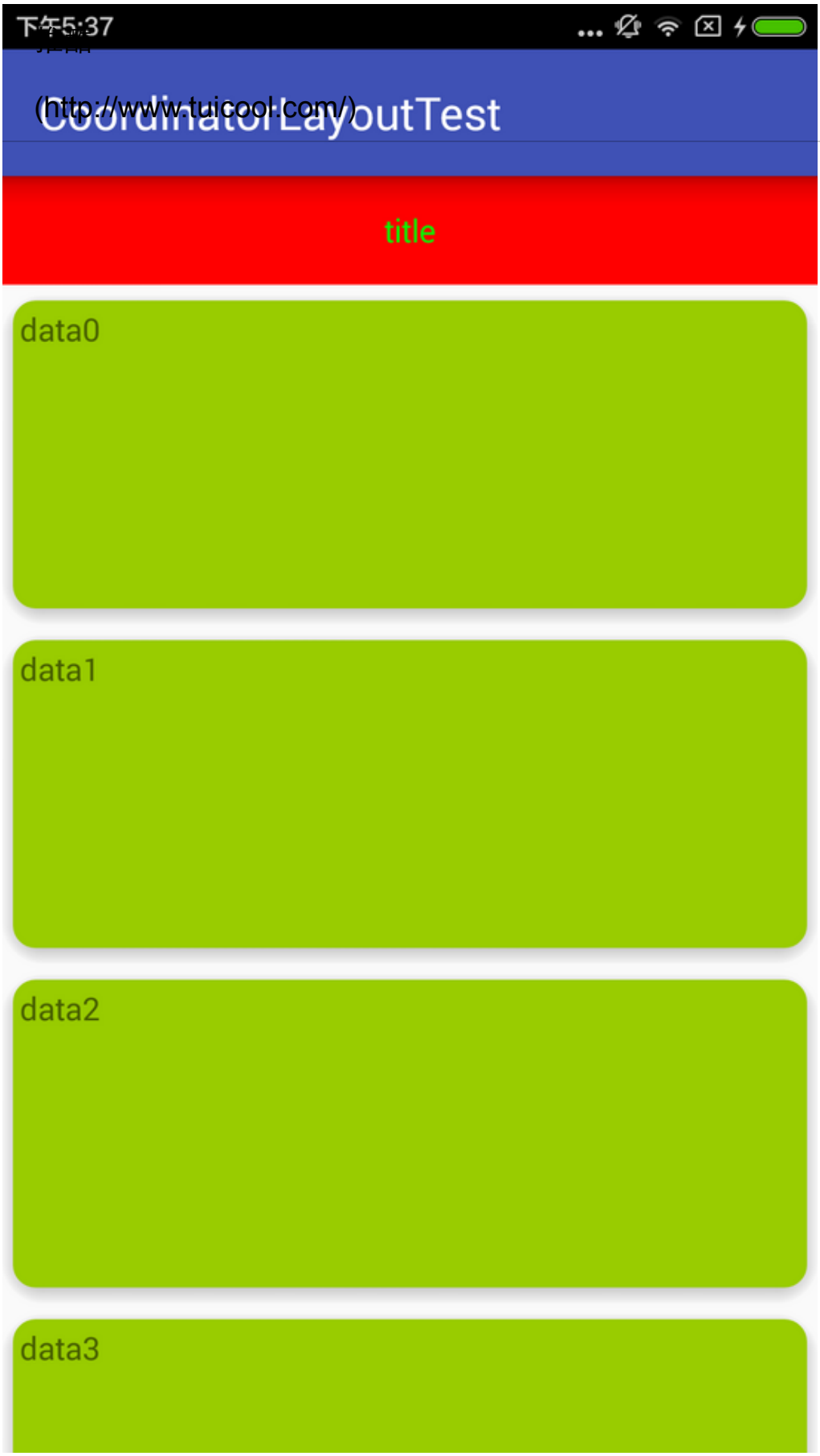
既然CoordinatorLayout仅仅只是一个ViewGroup，它又为什么能展示出它在xml布局中展示的威力呢？其中的秘密就是在Behavior中。我们可以这么说，CoordinatorLayout利用了Behavior作为一个代理，去控制管理其下的子View做到各种布局和动画效果。那为什么要使用Behavior呢？我想原因大概就是解耦吧，如果把所有的逻辑都写死在CoordinatorLayout中，一来不利于维护，二来我们就没有做一些自定义的事情，会显得非常的笨重。

为什么要用Behavior

这里我们举一个非常简单的例子。首先来看看我们的布局文件。

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/main_fragment_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.design.widget.AppBarLayout
        android:id="@+id/app_bar"
        android:layout_width="match_parent"
        android:layout_height="48dp">

        <TextView
            android:background="#ff0000"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:text="title"
            android:textColor="#00ff00"
            android:gravity="center"/>
    </android.support.design.widget.AppBarLayout>

    <android.support.v7.widget.RecyclerView
        android:id="@+id/list"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_behavior="@string/appbar_scrolling_view_behavior">

    </android.support.v7.widget.RecyclerView>

</android.support.design.widget.CoordinatorLayout>
```

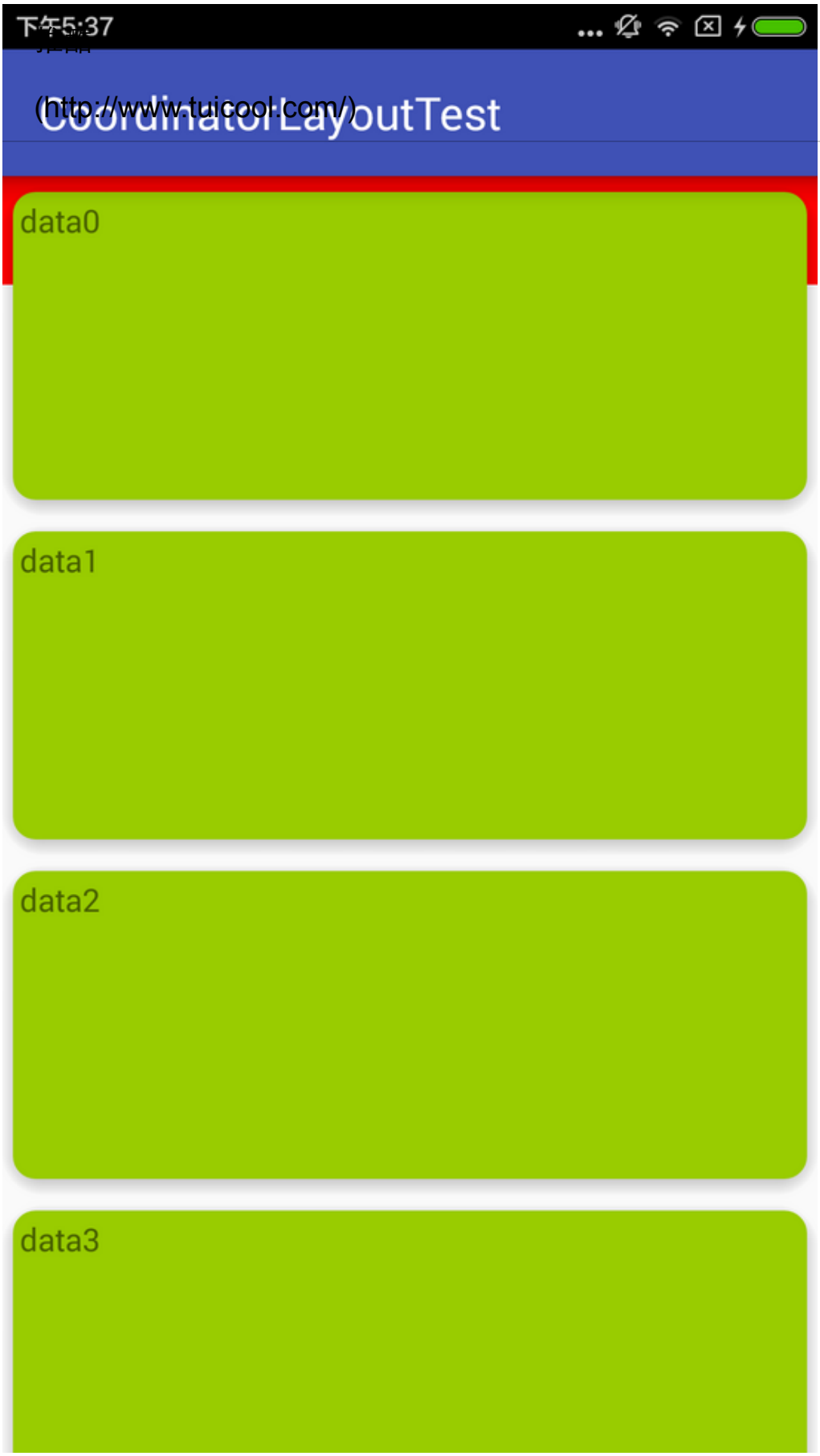
非常简单有木有，CoordinatorLayout作为根布局，里面一个AppBarLayout一个RecyclerView。让我们看看界面是怎么样的。



可以看到显示是正确的。但是如果我把xml里RecyclerView的那行layout_behavior删掉呢？就像这样。

推酷
(http://www.tuicool.com/)

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/main_fragment_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.design.widget.AppBarLayout
        android:id="@+id/app_bar"
        android:layout_width="match_parent"
        android:layout_height="48dp">

        <TextView
            android:background="#ff0000"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:text="title"
            android:textColor="#00ff00"
            android:gravity="center"/>
    </android.support.design.widget.AppBarLayout>

    <android.support.v7.widget.RecyclerView
        android:id="@+id/list"
        android:layout_width="match_parent"
        android:layout_height="match_parent">

    </android.support.v7.widget.RecyclerView>

</android.support.design.widget.CoordinatorLayout>
```



最终界面的展示就像这样，RecyclerView把AppBarLayout给覆盖了。这里其实很好理解，如刚才的代码所示，CoordinatorLayout其实只是一个ViewGroup，它不像LinearLayout那样具有特定的布局特点，设置可以说它内部的逻辑和Framework是没什么差别的，所以如果你不设置对应的Behavior的话，布局就会有问题。从这里也可以反映出Behavior的作用，就是规范子View的显示和交互。

原理&系统是怎么用Behavior的

说完了Behavior的作用，那该怎么用它呢？这一小节让我们来讲讲Behavior的原理以及系统是如何使用它的。

首先先看原理。我们知道Behavior是用来帮助CoordinatorLayout的，所以我们要从CoordinatorLayout中寻找答案。首先，我们可以看到CoordinatorLayout中有一个LayoutParams，它的子类的LayoutParams都是这个，其中它的构造函数如下。


```
LayoutParams(Context context, AttributeSet attrs) {
    super(context, attrs);

    .........
    if (mBehaviorResolved) {
        mBehavior = parseBehavior(context, attrs, a.getString(
                R.styleable.CoordinatorLayout_LayoutParams_layout_behavior));
    }

    a.recycle();
}
```

可以看到它通过parseBehavior去得到了对应子View的Behavior。大家可以试试用RecyclerView的getLayoutParams方法去获取LayoutParams并且调用getBehavior方法，可以得到的就是我们在xml文件中设置的那个Behavior。

知道了如何将Behavior设置进去，那它是如何发挥作用的呢？让我们来看看onLayout函数。

```
@Override
protected void onLayout(boolean changed, int l, int t, int r, int b) {
    final int layoutDirection = ViewCompat.getLayoutDirection(this);
    final int childCount = mDependencySortedChildren.size();
    for (int i = 0; i < childCount; i++) {
        final View child = mDependencySortedChildren.get(i);
        final LayoutParams lp = (LayoutParams) child.getLayoutParams();
        final Behavior behavior = lp.getBehavior();

        if (behavior == null || !behavior.onLayoutChild(this, child, layoutDirection)) {
            onLayoutChild(child, layoutDirection);
        }
    }
}
```

可以看到的是其中会先调用behavior.onLayoutChild(this, child, layoutDirection)。也就是说，Behavior的逻辑要优先于CoordinatorLayout自己的逻辑。其实不止是onLayout，我们还可以看看onTouchEvent这个函数。

```
public boolean onTouchEvent(MotionEvent ev) {
    boolean handled = false;
    boolean cancelSuper = false;
    MotionEvent cancelEvent = null;

    final int action = MotionEventCompat.getActionMasked(ev);

    if (mBehaviorTouchView != null || (cancelSuper = performIntercept(ev, TYPE_ON_TOUCH))) {
        // Safe since performIntercept guarantees that
        // mBehaviorTouchView != null if it returns true
        final LayoutParams lp = (LayoutParams) mBehaviorTouchView.getLayoutParams();
        final Behavior b = lp.getBehavior();
        if (b != null) {
            handled = b.onTouchEvent(this, mBehaviorTouchView, ev);
        }
    }

    .........

    return handled;
}
```

可以看到也是调用了Behavior的onTouchEven，我们可以下判断说Behavior中的那些方法在CoordinatorLayout中都会在合适的时机去调用。这也证明了我们刚才的那句话：［Behavior就是CoordinatorLayout的代理，帮助它去管理子View］。

我们做一个总结，Behavior可以代理哪些行为呢?

1.Measure和Layout的布局行为。

2.onTouchEvent和onInterceptTouchEvent的触摸行为。比如design包中的SwipeDismissBehavior就是通过这样的方式完成的。

3.嵌套滑动行为(NestedScrollingParent和NestedScrollingChild中的逻辑)。

4.子View间的依赖行为。

对于第四点我们这里可以细说一下，什么叫子View的依赖行为呢？这里我们举个例子，我们都知道如果在CoordinatorLayout中使用了FAB并且点击展示SnackbarLayout的话，FAB会在Snackbar显示的时候对应的上移，这是因为FAB依赖了SnackbarLayout。

```
public static class Behavior extends CoordinatorLayout.Behavior<FloatingActionButton> {
    @Override
    public boolean layoutDependsOn(CoordinatorLayout parent,
            FloatingActionButton child, View dependency) {
        // We're dependent on all SnackbarLayouts (if enabled)
        return SNACKBAR_BEHAVIOR_ENABLED && dependency instanceof Snackbar.SnackbarLayout;
    }

   @Override
   public boolean onDependentViewChanged(CoordinatorLayout parent, FloatingActionButton child,View depen
dency) {
       if (dependency instanceof Snackbar.SnackbarLayout) {
                updateFabTranslationForSnackbar(parent, child, dependency);
       } else if (dependency instanceof AppBarLayout) {
                // If we're depending on an AppBarLayout we will show/hide it automatically
                // if the FAB is anchored to the AppBarLayout
                updateFabVisibility(parent, (AppBarLayout) dependency, child);
       }
       return false;
    }

    ........
}
```

这是FAB中的Behavior，可以看到它重写了layoutDependsOn和onDependentViewChanged，里面的逻辑很简单的就可以看明白。这里我们［将代码翻译成语言］就是说FAB要依赖的组件是SnackbarLayout，所以在之后的操作里当DependentView(SnackbarLayout)发生了改变，自己(FAB)也会相应的做出改变。

值得一提的是，onDependentViewChanged这个函数的调用时机并不是在onLayout之前，而是在onPreDraw中，具体代码如下：

```
class OnPreDrawListener implements ViewTreeObserver.OnPreDrawListener {
    @Override
    public boolean onPreDraw() {
        dispatchOnDependentViewChanged(false);
        return true;
    }
}
```

如此简单的处理View间的依赖，可见Behavior配合CoordinatorLayout是有多强大。下面我们可以再举一个例子来讲讲Behavior的作用。还记得我们上面说的吗？RecyclerView设置了一个Behavior它就可以和AppBarLayout很好的展示出来。这个Behavior的名字是：

```
app:layout_behavior="@string/appbar_scrolling_view_behavior"

<string name="appbar_scrolling_view_behavior" translatable="false">android.support.design.widget.AppBarL
ayout$ScrollingViewBehavior</string>
```

可以看到它是AppBarLayout里的一个内部类，让我们看看它做了什么。

```
@Override
public boolean layoutDependsOn(CoordinatorLayout parent, View child, View dependency) {
    // We depend on any AppBarLayouts
    return dependency instanceof AppBarLayout;
}

@Override
public boolean onDependentViewChanged(CoordinatorLayout parent, View child,
        View dependency) {
    offsetChildAsNeeded(parent, child, dependency);
    return false;
}

private void offsetChildAsNeeded(CoordinatorLayout parent, View child, View dependency) {
    final CoordinatorLayout.Behavior behavior =
            ((CoordinatorLayout.LayoutParams) dependency.getLayoutParams()).getBehavior();
    if (behavior instanceof Behavior) {
        // Offset the child, pinning it to the bottom the header-dependency, maintaining
        // any vertical gap, and overlap
        final Behavior ablBehavior = (Behavior) behavior;
        final int offset = ablBehavior.getTopBottomOffsetForScrollingSibling();
        child.offsetTopAndBottom((dependency.getBottom() - child.getTop())
                + ablBehavior.mOffsetDelta
                + getVerticalLayoutGap()
                - getOverlapPixelsForOffset(dependency));
    }
}
```

我们知道，如果不设置这个Behavior的话，RecyclerView会覆盖AppBarLayout。而上面这段代码里的逻辑就可以很好的解释这个原因了。值得一提的是，在offsetChildAsNeeded方法中有这么一段：

```
final CoordinatorLayout.Behavior behavior = ((CoordinatorLayout.LayoutParams) dependency.getLayoutParams
()).getBehavior();
if (behavior instanceof Behavior) {
    // Offset the child, pinning it to the bottom the header-dependency, maintaining
    // any vertical gap, and overlap
    final Behavior ablBehavior = (Behavior) behavior;
```

这里dependency就是AppBarLayout，所以我们可以知道，AppBarLayout中有两个Behavior，一个是我们前面提到的ScrollingViewBehavior，用来处理它和其他滑动View的关系，另外一个就是Behavior，用来处理自己的逻辑，比如Layout。通过这种巧妙的方式，我们就可以做到非常简便的控制View本身和View之间的逻辑。

如何自定义Behavior

本来想写个demo给大家看一看的，不过感觉还是不要重复造轮子了，还是没用的轮子。推荐大家看 SwipeDismissBehavior用法及实现原理 (http://www.jcodecraeer.com/a/anzhuokaifa/androidkaifa/2015/1103/3650.html) 这篇文章和一开始提到的Jake大神的新作DrawerBehavior (https://github.com/JakeWharton/DrawerBehavior) 。如果你把这两个东西搞懂，那么Behavior你可以说已经完全没问题了～

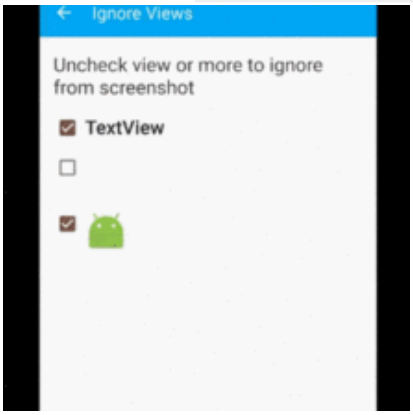
后记

最近一段时间都在搞hotPatch和插件化相关的东西，看了很多Framework层的源码，要做的东西也做的七七八八，希望快点解决最后的几个bug并且之后能开源和大家见面吧～

另外马上就要毕业了，论文写完就差答辩，大学四年的时间过的真的好快。出来实习半年，让我个人的长进非常多，也认识了很多大神，每天跟着他们学习和自我学习都是我最享受的事。不过越来越觉得除了工作上的事，其他事情都无所谓了，想想高中的自己是一个非常较真的人，为了能扣篮搞负重深蹲，甚至有段时间天天蛙跳回家，最后还是只能扣排球并且搞得膝盖积水，现在打球都不会较真了，碰瓷打打出出外就好。有时候会打球看着球场上那些投几个球就踹得不行的大伯也会想他们当年是不是和我一样飞天遁地，或者我以后是不是也会和他们一样。不过最近为了证明自己不是那样的人最近开始每天跑步＋运球练习，希望可以找回当时的拼劲吧，不管工作还是生活，都要做一个有追求的人啊，拜托了。

推酷 (http://www.tuicool.com/)

- by 等終等於等相遇 (/kans/1489390680) 《Android》 (/kans/1489390680) *3*

- 

《匿名收藏》 185

我来评几句

请输入评论内容...

登录后评论

已发表评论数(1)

Meta (/user/3588684099)06-08 14:35
好e

相关站点

zjutkz's blog (/sites/rmimaqf)

+ 订阅

热门文章

- 1. 一个完整的示例：Android Things 和 TensorFlow 能擦出怎样的火花？ (/articles/FVvyayE)
- 2. Android 中使用 RecyclerView + SnapHelper 实现类似 ViewPager 效果 (/articles/uyIFfyn)
- 3. 通用PopupWindow，几行代码搞定PopupWindow弹窗（续） (/articles/f63IrmE)
- 4. 分享一个 Android XML 的命名规范 (/articles/nimmmui)
- 5. StateBackgroundUtil - 仅使用一张资源图片为 View 设置具有按下效果的背景 (/articles/3mM3Ir6)

(https://sspaas.com/)
推酷
(https://mos.meituan.com/firework/newcustomer?site=tuicool&campaign=20170401sales)

(http://www.tuicool.com/)

(https://www.mysubmail.com/sms?s=tuicool)

关于我们 (http://www.tuicool.com/about) 移动应用 (http://www.tuicool.com/mobile) 意见反馈 (http://www.tuicool.com/bbs/go/issues) 官方微博 (http://e.weibo.com/tuicool2012)