SwipeDismissBehavior用法及实现原理

泡在网上的日子 / 文 发表于2015-11-03 15:48 第7042次阅读 滑动删除 (/tags.php?/滑动删除/)
(http://www.jiathis.com/share)

编辑推荐：稀土掘金 (https://juejin.im/)，这是一个针对技术开发者的一个应用，你可以在掘金上获取最新最优质的技术干货，不仅仅是Android知识、前端、后端以至于产品和设计都有涉猎，想成为全栈工程师的朋友不要错过！

引文

无意间发现design兼容库中有一个叫做SwipeDismissBehavior的类，顾名思义它就是用来实现滑动删除的了。莫非现在滑动删除又有更简单的解决办法了？鉴于之前RecyclerView中已经有ItemTouchHelper，而且也非常简单，所以很好奇到底有何不同，于是决定研究研究，看看它的实现原理以及应用场景：真的能替代其他的（不管是第三方还是RecyclerView自带的ItemTouchHelper）滑动删除吗？。

很不幸SwipeDismissBehavior现在的文档还很少，只有stackoverfolw上有点价值的讨论。

先来直接从API的角度使用SwipeDismissBehavior，然后再讲解SwipeDismissBehavior的原理。从而说明为什么SwipeDismissBehavior只能和CoordinatorLayout一起使用？为什么SwipeDismissBehavior对CoordinatorLayout中RecyclerView的item不起作用。

SwipeDismissBehavior的用法

SwipeDismissBehavior的用法非常简单。

第一步：引入design库：

```
1.  compile 'com.android.support:appcompat-v7:23.1.0'
2.  compile 'com.android.support:design:23+'
```

第二步：把要滑动删除的View放在CoordinatorLayout中：

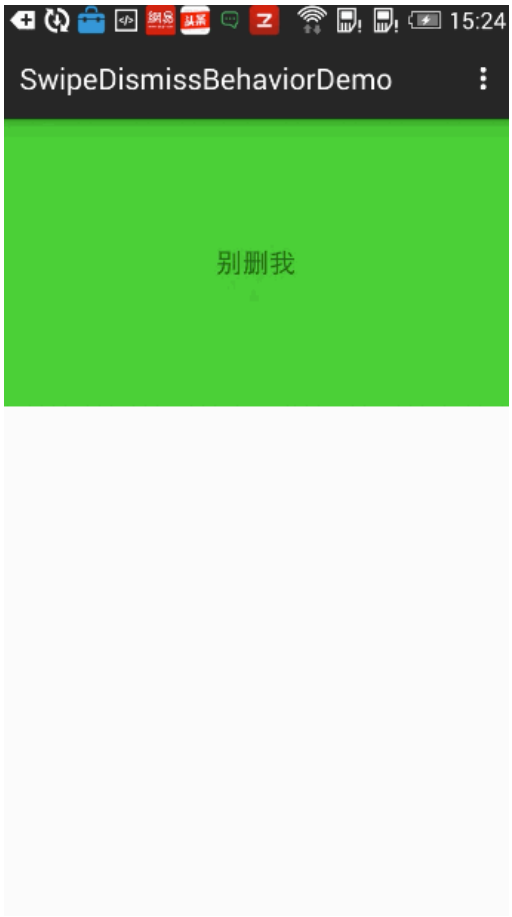
xml代码：

```
<?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"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/coordinatorLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    >

    <TextView
        android:id="@+id/swip"
        android:layout_width="match_parent"
        android:layout_height="200dip"
        android:background="#32CD32"
        android:text="别删我"
        android:textSize="20dip"
        android:gravity="center"
        />

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

第三步：在MainActivity中为View设置一个SwipeDismissBehavior对象：

```
package com.jcodecraeer.swipedismissbehaviordemo;

import android.os.Bundle;
import android.support.design.widget.CoordinatorLayout;
import android.support.design.widget.SwipeDismissBehavior;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        TextView swipeView = (TextView)findViewById(R.id.swip);
        final SwipeDismissBehavior<View> swipe
                = new SwipeDismissBehavior();

        swipe.setSwipeDirection(
                SwipeDismissBehavior.SWIPE_DIRECTION_ANY);

        swipe.setListener(
                new SwipeDismissBehavior.OnDismissListener() {
                    @Override public void onDismiss(View view) {

                    }

                    @Override
                    public void onDragStateChanged(int state) {}
                });

        CoordinatorLayout.LayoutParams coordinatorParams =
                (CoordinatorLayout.LayoutParams) swipeView.getLayoutParams();

        coordinatorParams.setBehavior(swipe);
    }

}
```

然后运行就能得到如下效果：



就是这么简单。

下面来讲讲SwipeDismissBehavior的原理。

要讲SwipeDismissBehavior，得先讲讲CoordinatorLayout.Behavior。因为

SwipeDismissBehavior类是如下定义的。

```
1.  public class SwipeDismissBehavior<V extends View> extends CoordinatorLayout.Behavior<V>
```

从这个定义可以看出SwipeDismissBehavior继承自CoordinatorLayout.Behavior。

CoordinatorLayout与Behavior

Behavior是CoordinatorLayout的一个内部类

```
1.  public static abstract class Behavior<V extends View>
```

它只定义了一些抽象方法，其中最主要的当属下面两个（与本文相关）：

```
1.  public boolean onInterceptTouchEvent(CoordinatorLayout parent, V child, MotionEvent ev) {
2.      return false;
3.  }
4.
5.  public boolean onTouchEvent(CoordinatorLayout parent, V child, MotionEvent ev) {
6.      return false;
7.  }
```

CoordinatorLayout会在自己的onInterceptTouchEvent()方法中调用Behavior的

onInterceptTouchEvent：

```
1.  b.onInterceptTouchEvent(this, child, ev);
```

把自己(this)、与此Behavior对象相关的子view（child）以及MotionEvent ev传递过去。这三个参数对于实现一个Behavior都至关重要。

CoordinatorLayout遍历子view，判断子view的mLayoutParam变量中是否有Behavior成员，如果有则调用Behavior的onInterceptTouchEvent和onTouchEvent方法。上面的代码中，swipeView通过

```
1.  CoordinatorLayout.LayoutParams coordinatorParams =
2.          (CoordinatorLayout.LayoutParams) swipeView.getLayoutParams();
3.
4.  coordinatorParams.setBehavior(swipe);
```

给自己设置了一个类型为SwipeDismissBehavior的Behavior，而且它又是CoordinatorLayout的子view，因此当CoordinatorLayout遍历到了这个cardview的时候，会尝试从这个swipeView获得Behavior：

```
1.  final LayoutParams lp = (LayoutParams) child.getLayoutParams();
2.  final Behavior b = lp.getBehavior();
```

注：这里的LayoutParams是CoordinatorLayout.LayoutParams类，跟ViewGroup的还是有所区别，他是CoordinatorLayout的内部类。其实任何一个布局比如LinearLayout都有自己的LayoutParams类型，也都是定义在布局类的内部。

如果检测到这个Behavior不为空，就调用它的onInterceptTouchEvent和onTouchEvent方法。

```
1.  if (!intercepted && b != null) {
2.      switch (type) {
3.          case TYPE_ON_INTERCEPT:
4.              intercepted = b.onInterceptTouchEvent(this, child, ev);
5.              break;
6.          case TYPE_ON_TOUCH:
7.              intercepted = b.onTouchEvent(this, child, ev);
8.              break;
9.      }
10.     if (intercepted) {
11.         mBehaviorTouchView = child;
12.     }
13. }
```

自此CoordinatorLayout的任务完成，因为我们设置的这个Behavior是SwipeDismissBehavior对象，所以接下来该怎么处理就交给SwipeDismissBehavior了。

以上过程基本都在CoordinatorLayout的performIntercept(MotionEvent ev, final int type)方法里。

SwipeDismissBehavior

那么我们看看SwipeDismissBehavior的onInterceptTouchEvent和onTouchEvent方法到底做了什么呢?

```
1.  public boolean onInterceptTouchEvent(CoordinatorLayout parent, V child, MotionEvent event) {
2.      switch(MotionEventCompat.getActionMasked(event)) {
3.      case 1:
4.      case 3:
5.          if(this.mIgnoreEvents) {
6.              this.mIgnoreEvents = false;
7.              return false;
8.          }
9.          break;
10.     default:
11.         this.mIgnoreEvents = !parent.isPointInChildBounds(child, (int)event.getX(), (int)event.getY());
12.     }
13.
14.     if(this.mIgnoreEvents) {
15.         return false;
16.     } else {
17.         this.ensureViewDragHelper(parent);
18.         return this.mViewDragHelper.shouldInterceptTouchEvent(event);
19.     }
20. }
21.
22. public boolean onTouchEvent(CoordinatorLayout parent, V child, MotionEvent event) {
23.     if(this.mViewDragHelper != null) {
24.         this.mViewDragHelper.processTouchEvent(event);
25.         return true;
26.     } else {
27.         return false;
28.     }
29. }
```

在onInterceptTouchEvent(CoordinatorLayout parent, V child, MotionEvent event) 中，调用了ensureViewDragHelper(parent):

```
1.  private void ensureViewDragHelper(ViewGroup parent) {
2.      if(this.mViewDragHelper == null) {
3.          this.mViewDragHelper = this.mSensitivitySet?ViewDragHelper.create(parent, this.mSensitivity, this.mDragCall
4.      }
5.
6.  }
```

可以看到这里其实是用parent参数创建了一个ViewDragHelper，根据前面的分析，这里的parent其实就是CoordinatorLayout对象。如果你熟悉ViewDragHelper，那么基本上都能猜到SwipeDismissBehavior要做些什么了。

SwipeDismissBehavior就是根据 MotionEvent和parent创建了一个实现了滑动删除的ViewDragHelper。

具体实现的代码请看SwipeDismissBehavior的mDragCallback变量。

```
1.  private final Callback mDragCallback = new Callback() {
2.      private int mOriginalCapturedViewLeft;
3.
4.      public boolean tryCaptureView(View child, int pointerId) {
5.          this.mOriginalCapturedViewLeft = child.getLeft();
6.          return true;
7.      }
8.
9.      public void onViewDragStateChanged(int state) {
10.         if(SwipeDismissBehavior.this.mListener != null) {
11.             SwipeDismissBehavior.this.mListener.onDragStateChanged(state);
12.         }
13.
14.     }
15.
16.     public void onViewReleased(View child, float xvel, float yvel) {
17.         int childWidth = child.getWidth();
18.         boolean dismiss = false;
19.         int targetLeft;
20.         if(this.shouldDismiss(child, xvel)) {
21.             targetLeft = child.getLeft() < this.mOriginalCapturedViewLeft?this.mOriginalCapturedViewLeft - childWi
22.             dismiss = true;
23.         } else {
24.             targetLeft = this.mOriginalCapturedViewLeft;
25.         }
26.
27.         if(SwipeDismissBehavior.this.mViewDragHelper.settleCapturedViewAt(targetLeft, child.getTop())) {
28.             ViewCompat.postOnAnimation(child, SwipeDismissBehavior.this.new SettleRunnable(child, dismiss));
29.         } else if(dismiss && SwipeDismissBehavior.this.mListener != null) {
30.             SwipeDismissBehavior.this.mListener.onDismiss(child);
31.         }
32.
33.     }
34.
35.     private boolean shouldDismiss(View child, float xvel) {
36.         if(xvel != 0.0F) {
37.             boolean distance1 = ViewCompat.getLayoutDirection(child) == 1;
38.             return SwipeDismissBehavior.this.mSwipeDirection == 2?true:(SwipeDismissBehavior.this.mSwipeDirection =
39.         } else {
40.             int distance = child.getLeft() - this.mOriginalCapturedViewLeft;
41.             int thresholdDistance = Math.round((float)child.getWidth() * SwipeDismissBehavior.this.mDragDismissThre
42.             return Math.abs(distance) >= thresholdDistance;
43.         }
44.     }
45.
46.     public int getViewHorizontalDragRange(View child) {
47.         return child.getWidth();
48.     }
49.
50.     public int clampViewPositionHorizontal(View child, int left, int dx) {
51.         boolean isRtl = ViewCompat.getLayoutDirection(child) == 1;
52.         int min;
53.         int max;
54.         if(SwipeDismissBehavior.this.mSwipeDirection == 0) {
55.             if(isRtl) {
56.                 min = this.mOriginalCapturedViewLeft - child.getWidth();
57.                 max = this.mOriginalCapturedViewLeft;
58.             } else {
59.                 min = this.mOriginalCapturedViewLeft;
60.                 max = this.mOriginalCapturedViewLeft + child.getWidth();
61.             }
62.         } else if(SwipeDismissBehavior.this.mSwipeDirection == 1) {
63.             if(isRtl) {
64.                 min = this.mOriginalCapturedViewLeft;
65.                 max = this.mOriginalCapturedViewLeft + child.getWidth();
66.             } else {
67.                 min = this.mOriginalCapturedViewLeft - child.getWidth();
68.                 max = this.mOriginalCapturedViewLeft;
69.             }
70.         } else {
71.             min = this.mOriginalCapturedViewLeft - child.getWidth();
72.             max = this.mOriginalCapturedViewLeft + child.getWidth();
73.         }
```

```
74.
75.           return SwipeDismissBehavior.clamp(min, left, max);
76.       }
77.
78.       public int clampViewPositionVertical(View child, int top, int dy) {
79.           return child.getTop();
80.       }
81.
82.       public void onViewPositionChanged(View child, int left, int top, int dx, int dy) {
83.           float startAlphaDistance = (float)this.mOriginalCapturedViewLeft + (float)child.getWidth() * SwipeDismissBe
84.           float endAlphaDistance = (float)this.mOriginalCapturedViewLeft + (float)child.getWidth() * SwipeDismissBeha
85.           if((float)left <= startAlphaDistance) {
86.               ViewCompat.setAlpha(child, 1.0F);
87.           } else if((float)left >= endAlphaDistance) {
88.               ViewCompat.setAlpha(child, 0.0F);
89.           } else {
90.               float distance = SwipeDismissBehavior.fraction(startAlphaDistance, endAlphaDistance, (float)left);
91.               ViewCompat.setAlpha(child, SwipeDismissBehavior.clamp(0.0F, 1.0F - distance, 1.0F));
92.           }
93.
94.       }
95.   };
```

问题

那么我们的问题来了？SwipeDismissBehavior可以替代RecyclerView的ItemTouchHelper或者其他列表滑动删除库吗?

答案是不能。

因为CoordinatorLayout遍历子View的时候，只遍历了第一层view，而列表的滑动删除对象是在RecyclerView的里面，不是CoordinatorLayout的直接子view。

再者，既然是RecyclerView的item，那么它的LayoutParams就是RecyclerView.LayoutParams 它无法强制转换成CoordinatorLayout.LayoutParams，所以运行的时候会报错：

java.lang.ClassCastException: android.support.v7.widget.RecyclerView$LayoutParams cannot be cast to android.support.design.widget.CoordinatorLayout$LayoutParams

因此SwipeDismissBehavior只适合本文开始的那种用法。

 (/member/index.php?uid=ITHTT)  (/member/index.php?uid=jinrainfall2015)  (/member/index.php?uid=benio)

 (/member/index.php?uid=aiven)  (/member/index.php?uid=凡心)  (/member/index.php?uid=Sinyuk)

 (/member/index.php?uid=huazai30000)  (/member/index.php?uid=kince)  (/member/index.php?uid=xesam)

 (/member/index.php?uid=layman)  (/member/index.php?uid=隔壁班胖子)

相关文章

Android实现类似QQ的滑动删除效果 (/a/anzhuokaifa/androidkaifa/2015/0422/2771.html) 2015-04-22

上一篇：**比葫芦娃还可怕的百度全系APP SDK漏洞 - WormHole虫洞漏洞分析报告 (/a/anzhuokaifa/androidkaifa/2015/1102/3649.html)**

”You can’t have a back door in the software because you can’t have a back door that’s only for the good guys.“ - Apple CEO Tim Cook ”你不应该给软件装后门因为你不能保证这个后门只有好人能够使用。” – 苹果CEO 库克 0x00 序 最早接触网络

下一篇：**android app开发计划 (/a/anzhuokaifa/androidkaifa/2015/1104/3655.html)**

android app完整开发计划 本人做android两年多，虽然算不上技术大牛，但是自认为也小有所成。平时没什么爱好，喜欢折腾IDE，总觉得工欲善其事，必先利其器。然后喜欢追求新鲜的事物，比如新的技术和流行的框架，还有是一个完美主义者，外观控，所以对UI设计

发表评论

网友115.172.109.237 . 2015-11-06

还是有局限性啊，有没有大神在什么场景用过的?

0 0 回复

网友115.193.187.8 . 2015-11-04

一个字好

0 0 回复

网友210.22.156.162 . 2015-11-04

膜害。。。

0 0 回复

网友221.222.171.18 . 2015-11-03

菠萝太牛了

0 0 回复

推荐文章

App开发架构指南（谷歌官方文档译文） (/a/anzhuokaifa/androidkaifa/2017/0523/7963.html)

SQLBrite:一个响应式的数据查询框架 (/a/anzhuokaifa/androidkaifa/2015/0306/2552.html)

Android插件化原理解析——概要 (/a/anzhuokaifa/androidkaifa/2016/0227/4005.html)

终于等到你Depth-LIB-Android (/a/anzhuokaifa/androidkaifa/2016/0429/4200.html)

将Eclipse代码导入到Android Studio的两种方式 (/a/anzhuokaifa/androidkaifa/2015/0104/2259.html)

NumberProgressBar：一个简约性感的数字ProgressBar (/a/anzhuokaifa/androidkaifa/2014/0813/1645.html)

赞助商