Android 一步一步分析Behavior

时间 2016-09-26 15:36:19 亦枫的博客 (/sites/RZZFBr7)

原文
http://yifeng.studio/2016/09/23/behavior-analyzation/ (http://yifeng.studio/2016/09/23/behavior-analyzation/?utm_source=tuicool&utm_medium=referral)
主题 安卓开发 (/topics/11080010)
在MD系列的前几篇文章中，通过基础知识和实战案例配合讲解的形式介绍了 `CoordinatorLayout` 与 `AppBarLayout` 、`Toolbar` 、 `CollapsingToolbarLayout` 的使用，并实现了几种MD风格下比较炫酷的交互效果。学会怎么用之后，我们再想想，为什么它们之间能够产生这样的交互行为呢？其实就是因为 `CoordinatorLayout.Behavior` 的存在，这也是本文所要讲述的内容。
至此，Android Material Design系列的学习已进行到第八篇，大家可以点击以下链接查看之前的文章：

- Android TabLayout 分分钟打造一个滑动标签页 (http://www.jianshu.com/p/39a66373498c)
- Android 一文告诉你到底是用Dialog，Snackbar，还是Toast (http://www.jianshu.com/p/9eb3b17b0e77)
- Android FloatingActionButton 重要的操作不要太多，一个就好 (http://www.jianshu.com/p/5328b2eee827)
- Android 初识AppBarLayout 和 CoordinatorLayout (http://www.jianshu.com/p/ab04627cce58)
- Android CoordinatorLayout实战案例学习《一》 (http://www.jianshu.com/p/4b0f3c80ebc9)
- Android CoordinatorLayout 实战案例学习《二》 (http://www.jianshu.com/p/360fd368936d)
- Android 详细分析AppBarLayout的五种ScrollFlags (http://www.jianshu.com/p/7caa5f4f49bd)

关于Behavior

官网对于 `CoordinatorLayout.Behavior` 的介绍已经将它的作用说明得很清楚了，就是用来协调 `CoordinatorLayout` 的Child Views之间的交互行为：

Interaction behavior plugin for child views of CoordinatorLayout.

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.

之前学习 `CoordinatorLayout` 的使用案例时，用的都是系统的特定控件，比如design包中的 `FloatingActionButton` 、`AppBarLayout` 等，而不是普通的控件，如 `ImageButton` 之类的，就是因为design包中的这些特定控件已经被系统默认定义了继承自 `CoordinatorLayout.Behavior` 的各种 `Behavior` ，比如 `FloatingActionButton.Behavior` 和

`AppBarLayout.Behavior` 。而像系统的 `ToolBar` 控件就没有自己的 `Behavior` ，所以只能将其搁置到 `AppBarLayout` 容器里才能产生相应的交互效果。

看到这里就能清楚一点了，如果我们想实现控件之间任意的交互效果，完全可以通过自定义 `Behavior` 的方式达到。看到这里大家可能会有一个疑惑，就是 `CoordinatorLayout` 如何获取Child Views的 `Behavior` 的呢，为什么在布局中，有些滑动型控件定义了 `app:layout_behavior` 属性而系统类似 `FloatingActionButton` 的控件则不需要明确定义该属性呢？看完 `CoordinatorLayout.Behavior` 的构造函数就明白了。

```
/**
 * Default constructor for instantiating Behaviors.
 */
public Behavior() {
}

/**
 * Default constructor for inflating Behaviors from layout. The Behavior will have
 * the opportunity to parse specially defined layout parameters. These parameters will
 * appear on the child view tag.
 *
 * @param context
 * @param attrs
 */
public Behavior(Context context, AttributeSet attrs) {
}
```

`CoordinatorLayout.Behavior` 有两个构造函数，注意看第二个带参数的构造函数的注释，里面提到，在这个构造函数中，`Behavior` 会解析控件的特殊布局属性，也就是通过 `parseBehavior` 方法获取对应的 `Behavior` ，从而协调Child Views之间的交互行为，可以在 `CoordinatorLayout` 类中查看，具体源码如下：

```
static Behavior parseBehavior(Context context, AttributeSet attrs, String name) {
    if (TextUtils.isEmpty(name)) {
        return null;
    }

    final String fullName;
    if (name.startsWith(".")) {
        // Relative to the app package. Prepend the app package name.
        fullName = context.getPackageName() + name;
    } else if (name.indexOf('.') >= 0) {
        // Fully qualified package name.
        fullName = name;
    } else {
        // Assume stock behavior in this package (if we have one)
        fullName = !TextUtils.isEmpty(WIDGET_PACKAGE_NAME)
                ? (WIDGET_PACKAGE_NAME + '.' + name)
                : name;
    }

    try {
        Map<String, Constructor<Behavior>> constructors = sConstructors.get();
        if (constructors == null) {
            constructors = new HashMap<>();
            sConstructors.set(constructors);
        }
        Constructor<Behavior> c = constructors.get(fullName);
        if (c == null) {
            final Class<Behavior> clazz = (Class<Behavior>) Class.forName(fullName, true,
                    context.getClassLoader());
            c = clazz.getConstructor(CONSTRUCTOR_PARAMS);
            c.setAccessible(true);
            constructors.put(fullName, c);
        }
        return c.newInstance(context, attrs);
    } catch (Exception e) {
        throw new RuntimeException("Could not inflate Behavior subclass " + fullName, e);
    }
}
```

`parseBehavior` 方法告诉我们，给Child Views设置 `Behavior` 有两种方式：

1. `app:layout_behavior` 布局属性

   在布局中设置，值为自定义 `Behavior` 类的名字字符串（包含路径），类似在 `AndroidManifest.xml` 中定义四大组件的名字一样，有两种写法，包含包名的全路径和以”.”开头的省略项目包名的路径。

2. `@CoordinatorLayout.DefaultBehavior` 类注解

   在需要使用 `Behavior` 的控件源码定义中添加该注解，然后通过反射机制获取。这个方式就解决了我们前面产生的疑惑，系统的 `AppBarLayout` 、 `FloatingActionButton` 都采用了这种方式，所以无需在布局中重复设置。

看到这里，也告诉我们一点，在自定义 `Behavior` 时，一定要重写第二个带参数的构造函数，否则这个 `Behavior` 是不会起作用的。

根据 `CoordinatorLayout.Behavior` 提供的方法，这里将自定义 `Behavior` 分为两类来讲解，一种是 `dependent` 机制，一种是 `nested` 机制，对应着不同的使用场景。

dependent 机制

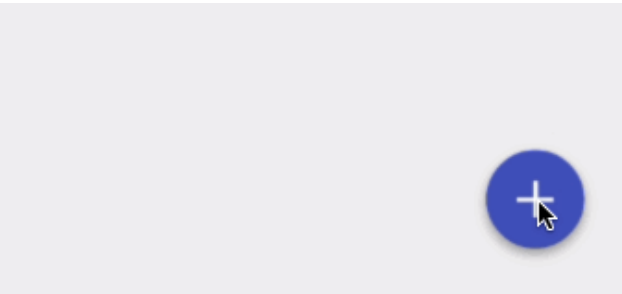
这种机制描述的是两个Child Views之间的绑定依赖关系，设置 `Behavior` 属性的Child View跟随依赖对象Dependency View的大小位置改变而发生变化，对应需要实现的方法常见有两个：

```
/**
 * Determine whether the supplied child view has another specific sibling view as a
 * layout dependency.
 *
 * <p>This method will be called at least once in response to a layout request. If it
 * returns true for a given child and dependency view pair, the parent CoordinatorLayout
 * will:</p>
 * <ol>
 *     <li>Always lay out this child after the dependent child is laid out, regardless
 *     of child order.</li>
 *     <li>Call {@link #onDependentViewChanged} when the dependency view's layout or
 *     position changes.</li>
 * </ol>
 *
 * @param parent the parent view of the given child
 * @param child the child view to test
 * @param dependency the proposed dependency of child
 * @return true if child's layout depends on the proposed dependency's layout,
 *         false otherwise
 *
 * @see #onDependentViewChanged(CoordinatorLayout, android.view.View, android.view.View)
 */
public boolean layoutDependsOn(CoordinatorLayout parent, V child, View dependency) {
    return false;
}
```

和

```
/**
 * Respond to a change in a child's dependent view
 *
 * <p>This method is called whenever a dependent view changes in size or position outside
 * of the standard layout flow. A Behavior may use this method to appropriately update
 * the child view in response.</p>
 *
 * <p>A view's dependency is determined by
 * {@link #layoutDependsOn(CoordinatorLayout, android.view.View, android.view.View)} or
 * if {@code child} has set another view as it's anchor.</p>
 *
 * <p>Note that if a Behavior changes the layout of a child via this method, it should
 * also be able to reconstruct the correct position in
 * {@link #onLayoutChild(CoordinatorLayout, android.view.View, int) onLayoutChild}.
 * <code>onDependentViewChanged</code> will not be called during normal layout since
 * the layout of each child view will always happen in dependency order.</p>
 *
 * <p>If the Behavior changes the child view's size or position, it should return true.
 * The default implementation returns false.</p>
 *
 * @param parent the parent view of the given child
 * @param child the child view to manipulate
 * @param dependency the dependent view that changed
 * @return true if the Behavior changed the child view's size or position, false otherwise
 */
public boolean onDependentViewChanged(CoordinatorLayout parent, V child, View dependency) {
    return false;
}
```

具体含义在注释中已经很清楚了，`layoutDependsOn()` 方法用于决定是否产生依赖行为，`onDependentViewChanged()` 方法在依赖的控件发生大小或者位置变化时产生回调。`dependent` 机制最常见的案例就是 `FloatingActionButton` 和 `SnackBar` 的交互行为，效果如下：



系统的 `FloatingActionButton` 已经默认定义了一个 `Behavior` 来协调交互，如果不用系统的FAB控件，比如改用GitHub上的一个库 `futuresimple/android-floating-action-button` (https://github.com/futuresimple/android-floating-action-button)，再通过自定义一个 `Behavior`，也能很简单的实现与 `SnackBar` 的协调效果：

```
package com.yifeng.mdstudysamples;

import android.content.Context;
import android.support.design.widget.CoordinatorLayout;
import android.support.design.widget.Snackbar;
import android.util.AttributeSet;
import android.view.View;

/**
 * Created by yifeng on 16/9/20.
 *
 */
public class DependentFABBehavior extends CoordinatorLayout.Behavior {

    public DependentFABBehavior(Context context, AttributeSet attrs) {
        super(context, attrs);
    }

    /**
     * 判断依赖对象
     * @param parent
     * @param child
     * @param dependency
     * @return
     */
    @Override
    public boolean layoutDependsOn(CoordinatorLayout parent, View child, View dependency) {
        return dependency instanceof Snackbar.SnackbarLayout;
    }

    /**
     * 当依赖对象发生变化时,产生回调,自定义改变child view
     * @param parent
     * @param child
     * @param dependency
     * @return
     */
    @Override
    public boolean onDependentViewChanged(CoordinatorLayout parent, View child, View dependency) {
        float translationY = Math.min(0, dependency.getTranslationY() - dependency.getHeight());
        child.setTranslationY(translationY);
        return true;
    }
}
```

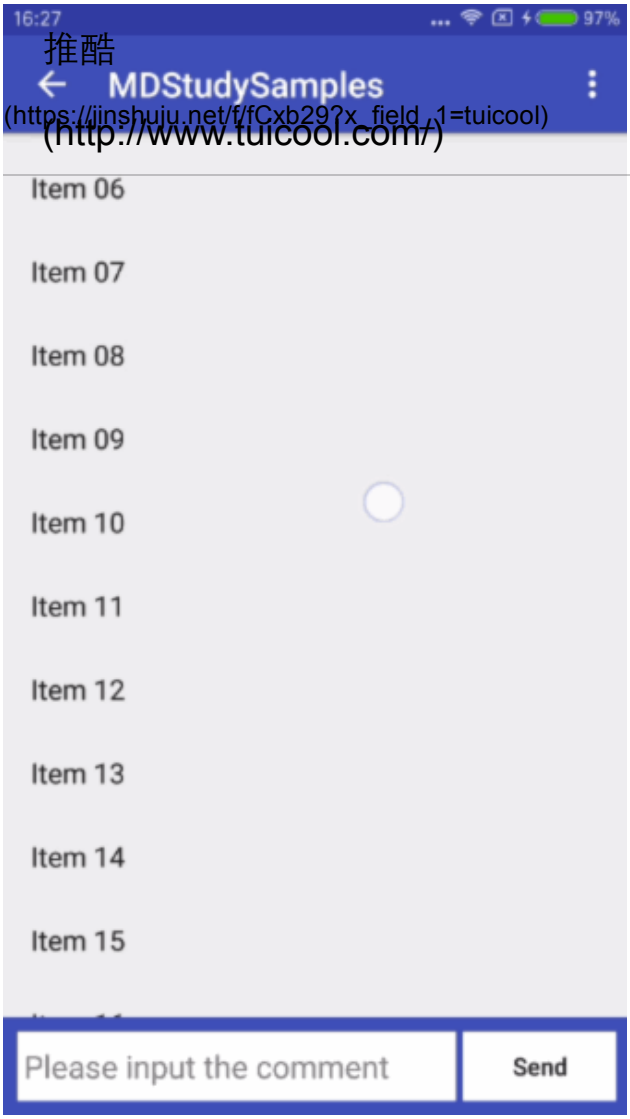
很简单的一个自定义 `Behavior` 处理，然后再为对应的Child View设置该属性即可。由于这里我们用的是第三方库，采用远程依赖的形式引入的，无法修改源码，所以不方便使用注解的方式为其设置 `Behavior` ，所以在布局中为其设置，并且使用了省略包名的方式:

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

    <android.support.design.widget.AppBarLayout
        android:id="@+id/appbar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar">

        <include
            layout="@layout/include_toolbar"/>

    </android.support.design.widget.AppBarLayout>

    <com.getbase.floatingactionbutton.FloatingActionButton
        xmlns:fab="http://schemas.android.com/apk/res-auto"
        android:id="@+id/fab_add"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="@dimen/dp_16"
        android:layout_gravity="bottom|right"
        android:onClick="onClickFab"
        fab:fab_icon="@mipmap/ic_toolbar_add"
        fab:fab_colorNormal="?attr/colorPrimary"
        fab:fab_colorPressed="?attr/colorPrimaryDark"
        app:layout_behavior=".DependentFABBehavior"/>

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

这样，采用 `dependent` 机制自定义 `Behavior` ，与使用系统FAB按钮一样，即可与 `SnackBar` 控件产生如上图所示的协调交互效果。

比如我们再看一下这样一个效果：

推酷
(https://jinshuju.net/f/fCxb29?x_field_1=tuicool)
(http://www.tuicool.com/)



列表上下滑动式，底部评论区域随着顶部 `Toolbar` 的移动而移动，这里我们就可以自定义一个 `Dependent` 机制的 `Behavior` ，设置给底部视图，让其依赖于包裹 `Toolbar` 的 `AppBarLayout` 控件：

```
package com.yifeng.mdstudysamples;

import android.content.Context;
import android.support.design.widget.AppBarLayout;
import android.support.design.widget.CoordinatorLayout;
import android.util.AttributeSet;
import android.view.View;

/**
 * Created by yifeng on 16/9/23.
 *
 */

public class CustomExpandBehavior extends CoordinatorLayout.Behavior {

    public CustomExpandBehavior(Context context, AttributeSet attrs) {
        super(context, attrs);
    }

    @Override
    public boolean layoutDependsOn(CoordinatorLayout parent, View child, View dependency) {
        return dependency instanceof AppBarLayout;
    }

    @Override
    public boolean onDependentViewChanged(CoordinatorLayout parent, View child, View dependency) {
        int delta = dependency.getTop();
        child.setTranslationY(-delta);
        return true;
    }
}
```

布局内容如下：

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

    <android.support.design.widget.AppBarLayout
        android:id="@+id/appbar"
        android:layout_width="match_parent"
        android:layout_height="@dimen/dp_56"
        android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar">

        <include
            layout="@layout/include_toolbar"/>

    </android.support.design.widget.AppBarLayout>

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

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="@dimen/dp_56"
        android:layout_gravity="bottom"
        app:layout_behavior=".CustomExpandBehavior"
        android:padding="8dp"
        android:background="@color/blue">

        <Button
            android:id="@+id/btn_send"
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:text="Send"
            android:layout_alignParentRight="true"
            android:background="@color/white"/>

        <EditText
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_toLeftOf="@id/btn_send"
            android:layout_marginRight="4dp"
            android:padding="4dp"
            android:hint="Please input the comment"
            android:background="@color/white"/>

    </RelativeLayout>

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

注意，这里将自定义的 `Behavior` 设置给了底部内容的外层容器 `RelativeLayout` ，即可实现上述效果。

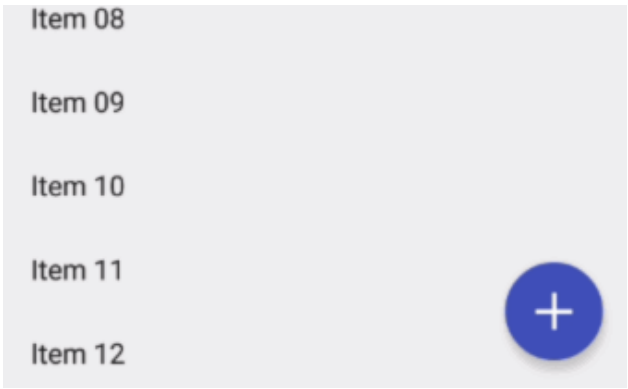
Nested 机制

`Nested` 机制要求 `CoordinatorLayout` 包含了一个实现了 `NestedScrollingChild` 接口的滚动视图控件，比如v7包中的 `RecyclerView` ，设置 `Behavior` 属性的Child View会随着这个控件的滚动而发生变化，涉及到的方法有：

```
onStartNestedScroll(View child, View target, int nestedScrollAxes)

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

onNestedPreFling(View target, float velocityX, float velocityY)

onNestedScroll(View target, int dxConsumed, int dyConsumed, int dxUnconsumed, int dyUnconsumed)

onNestedFling(View target, float velocityX, float velocityY, boolean consumed)

onStopNestedScroll(View target)
```

其中， `onStartNestedScroll` 方法返回一个boolean类型的值，只有返回true时才能让自定义的 `Behavior` 接受滑动事件。同样的，举例说明一下。

通过查看系统FAB控件的源码可以知道，系统FAB定义的 `Behavior` 能够处理两个交互，一个是与 `SnackBar` 的位置交互，效果如上面的图示一样，另一个就是与 `AppBarLayout` 的展示交互，都是使用的 `Dependent` 机制，效果在之前的文章 – Android CoordinatorLayout 实战案例学习《二》 (http://www.jianshu.com/p/360fd368936d) 中可以查看，也就是 `AppBarLayout` 滚动到一定程度时，FAB控件的动画隐藏与展示。下面我们使用 `Nested` 机制自定义一个 `Behavior` ，实现如下与列表协调交互的效果：



为了能够使用系统FAB控件提供的隐藏与显示的动画效果，这里直接继承了系统FAB控件的 `Behavior` ：

```
package com.yifeng.mdstudysamples;

import android.content.Context;
import android.support.design.widget.CoordinatorLayout;
import android.support.design.widget.FloatingActionButton;
import android.support.v4.view.ViewCompat;
import android.util.AttributeSet;
import android.view.View;

/**
 * Created by yifeng on 16/8/23.
 *
 */
public class NestedFABBehavior extends FloatingActionButton.Behavior {

    public NestedFABBehavior(Context context, AttributeSet attrs) {
        super();
    }

    @Override
    public boolean onStartNestedScroll(CoordinatorLayout coordinatorLayout, FloatingActionButton child,
 View directTargetChild, View target, int nestedScrollAxes) {
        return nestedScrollAxes == ViewCompat.SCROLL_AXIS_VERTICAL ||
                super.onStartNestedScroll(coordinatorLayout, child, directTargetChild, target,
                        nestedScrollAxes);
    }

    @Override
    public void onNestedScroll(CoordinatorLayout coordinatorLayout, FloatingActionButton child, View tar
get, int dxConsumed, int dyConsumed, int dxUnconsumed, int dyUnconsumed) {
        super.onNestedScroll(coordinatorLayout, child, target, dxConsumed, dyConsumed, dxUnconsumed, dyU
nconsumed);
        if (dyConsumed > 0 && child.getVisibility() == View.VISIBLE) {
            //系统FAB控件提供的隐藏动画
            child.hide();
        } else if (dyConsumed < 0 && child.getVisibility() != View.VISIBLE) {
            //系统FAB控件提供的显示动画
            child.show();
        }
    }
}
```

然后在布局中添加 `RecyclerView` ，并为系统FAB控件设置自定义的 `Behavior` ，内容如下：

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

    <android.support.design.widget.AppBarLayout
        android:id="@+id/appbar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar">

        <include
            layout="@layout/include_toolbar"/>

    </android.support.design.widget.AppBarLayout>

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

    <android.support.design.widget.FloatingActionButton
        android:id="@+id/fab_add"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="@dimen/dp_16"
        android:src="@mipmap/ic_toolbar_add"
        app:layout_anchor="@id/rv_content"
        app:layout_anchorGravity="bottom|right"
        app:backgroundTint="@color/fab_ripple"
        app:layout_behavior="com.yifeng.mdstudysamples.NestedFABBehavior"/>

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

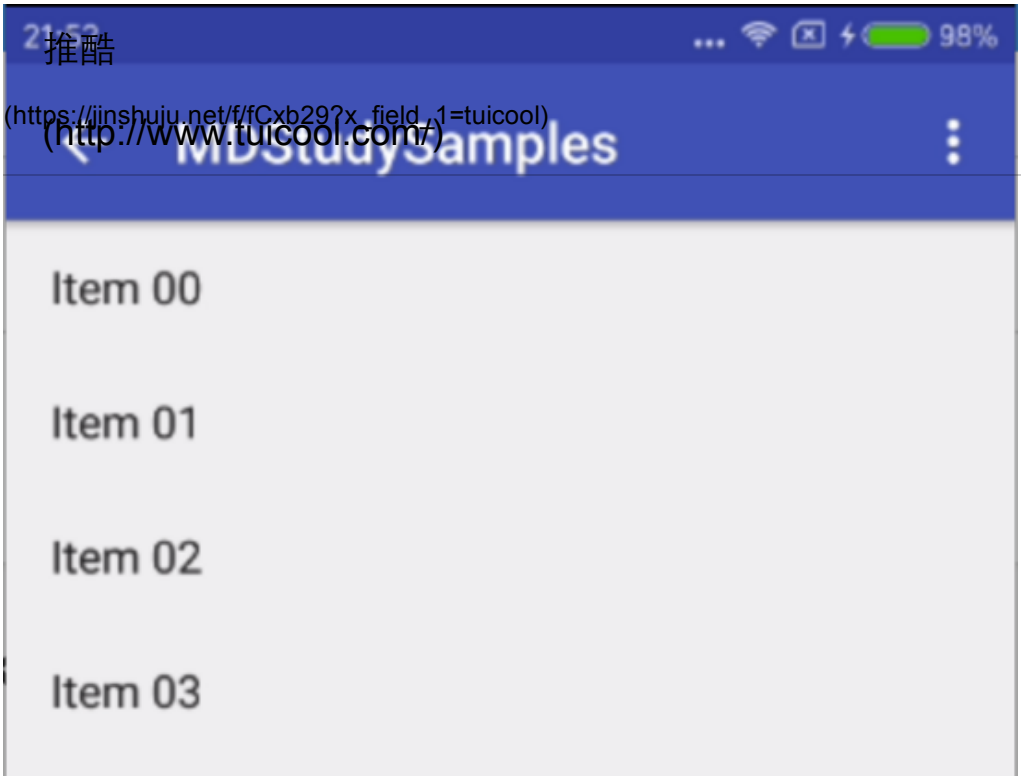
这样，即可实现系统FAB控件与列表滑动控件的交互效果。

@string/appbar_scrolling_view_behavior

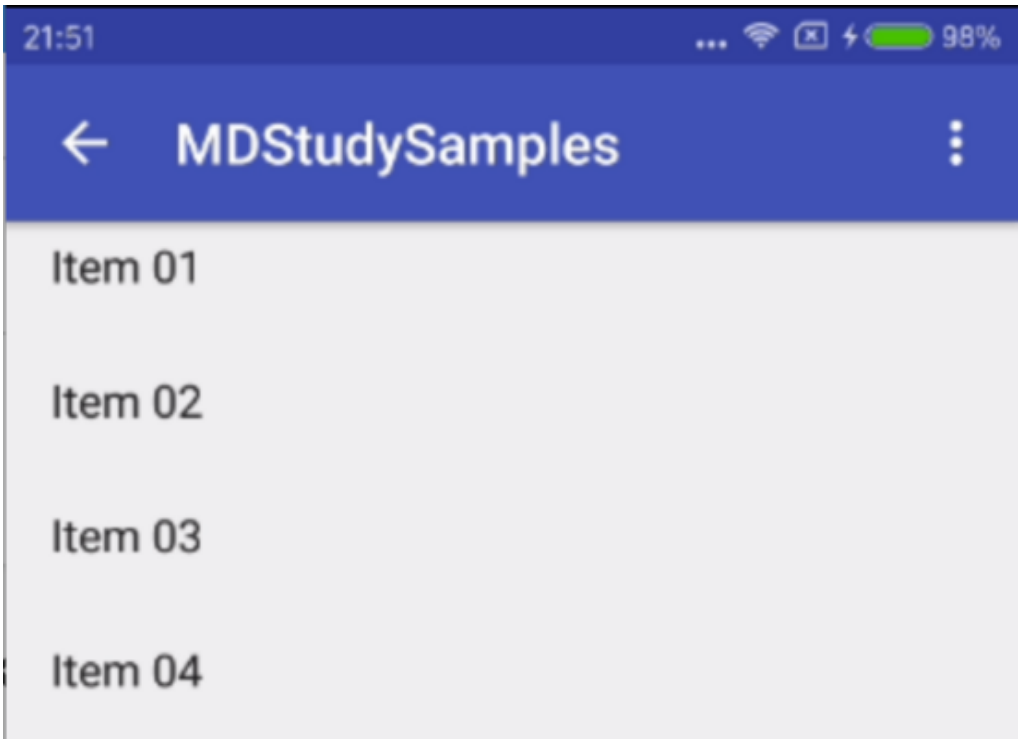
这是一个系统字符串，值为：

```
android.support.design.widget.AppBarLayout$ScrollingViewBehavior
```

在 `CoordinatorLayout` 容器中，通常用在 `AppBarLayout` 视图下面（不是里面）的内容控件中，比如上面的 `RecyclerView` ，如果我们不给它添加这个 `Behavior` ， `Toolbar` 将覆盖在列表上面，出现重叠部分，如图



添加之后， `RecyclerView` 将位于 `Toolbar` 下面，类似在 `RelativeLayout` 中设置了 `below` 属性，如图：



推荐文章

- 1. 关于Android strings.xml – 你应该了解的几个原则 (/articles/YfAzqm7)

- 2. Android 中的 Calendar，听说你有这样的需求 (/articles/qYjEji3)
- 3. Android开源项目规范总结 (/articles/2UruU3a)
- 4. 第204期：为什么我们要阅读源码？ (/articles/NNfeuiV)
- 5. 「Android」 APK瘦身探索 (/articles/ry6zUfB)
- 6. 一个完整的示例：Android Things 和 TensorFlow 能擦出怎样的火花？ (/articles/FVvyayE)

相关推刊

- by 不冬眠的熊  (/kans/1160746691) 《默认推刊》 (/kans/1160746691) 7
- 

《匿名收藏》 326

我来评几句

请输入评论内容...

登录后评论

已发表评论数(0)

相关站点

亦枫的博客 (/sites/RZZFBr7)

+ 订阅

热门文章

- 1. Android 中的 Calendar，听说你有这样的需求 (/articles/qYjEji3)
- 2. 「Android」 APK瘦身探索 (/articles/ry6zUfB)
- 3. 关于RxJava背压 (/articles/IFzauam)
- 4. 一个完整的示例：Android Things 和 TensorFlow 能擦出怎样的火花？ (/articles/FVvyayE)
- 5. Android 中使用 RecyclerView + SnapHelper 实现类似 ViewPager 效果 (/articles/uyIFfyn)