登录
下午好! 2017年3月28日 星期二

阳和移动开发

关注移动互联网和移动APP开发工具、开发框架、测试工具、微信开发、Android源码、Android开源类库以及各种开源组件的IT科技网站

Android开发——CoordinatorLayout用法详解

2015年08月03日 / Android开发经验 / 共 8369字 / 字号 小 中 大 / 评论关闭

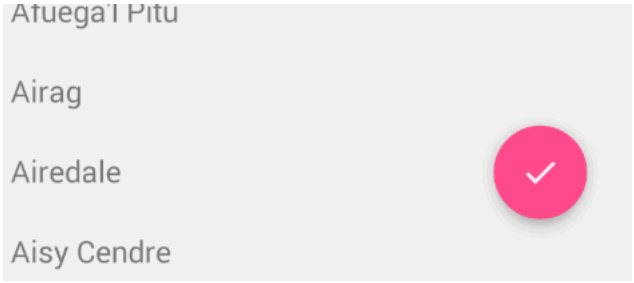
本文重点讲解如何在Android5.0+上使用CoordinatorLayout，涉及到CoordinatorLayout的概念和使用技巧，现在总结出来分享给广大的Android程序员兄弟们。

这篇文章专门讲解和CoordinatorLayout相关的知识点，这也是Design Support Library中最重要与最难的部分。

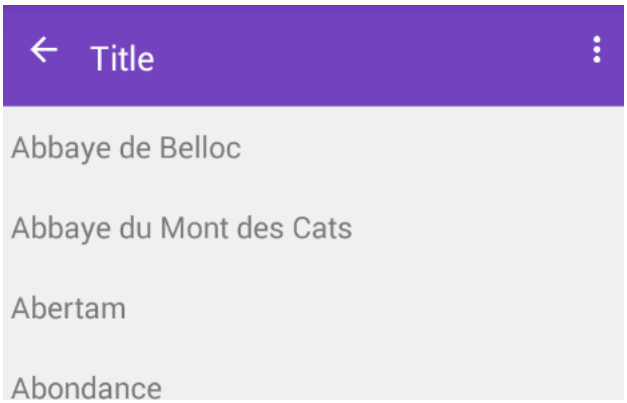
概览

CoordinatorLayout 实现了多种Material Design中提到的滚动效果。目前这个框架提供了几种不用写动画代码就能工作的方法，这些效果包括：

- 让浮动操作按钮上下滑动，为Snackbar留出空间。



- 扩展或者缩小Toolbar或者头部，让主内容区域有更多的空间。



- 控制哪个view应该扩展还是收缩，以及其显示大小比例，包括视差滚动效果动画。

设置

首先确保遵循了Design Support Library的使用说明。

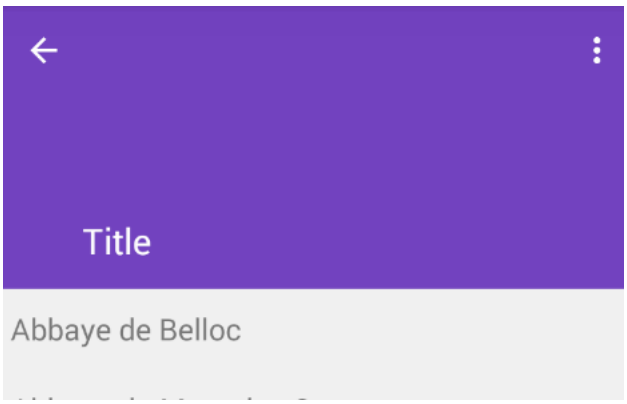
浮动操作按钮与Snackbar

CoordinatorLayout可以用来配合浮动操作按钮的 layout_anchor 和 layout_gravity属性创造出浮动效果，详情请参见浮动操作按钮指南。

当Snackbar在显示的时候，往往出现在屏幕的底部。为了给Snackbar留出空间，浮动操作按钮需要向上移动。

只要使用CoordinatorLayout作为基本布局，将自动产生向上移动的动画。浮动操作按钮有一个 默认的 behavior来检测Snackbar的添加并让按钮在Snackbar之上呈现上移与Snackbar等高的动画。

```
<android.support.design.widget.CoordinatorLayout
                android:id="@+id/main_content"
                xmlns:android="http://schemas.android.com/apk/res/android"
                xmlns:app="http://schemas.android.com/apk/res-auto"
                android:layout_width="match_parent"
                android:layout_height="match_parent">

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

    <android.support.design.widget.FloatingActionButton
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_gravity="bottom|right"
                android:layout_margin="16dp"
                android:src="@mipmap/ic_launcher"
                app:layout_anchor="@id/rvToDoList"
                app:layout_anchorGravity="bottom|right|end"/>
</android.support.design.widget.CoordinatorLayout>
```



Toolbar的扩展与收缩

首先需要确保你不是使用已经过时的ActionBar。务必遵循 使用ToolBar作为actionbar这篇文章的指南。同样，这里也需要CoordinatorLayout作为主布局容器。

```
<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_content"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:fitsSystemWindows="true">

            <android.support.v7.widget.Toolbar
                            android:id="@+id/toolbar"
                            android:layout_width="match_parent"
                            android:layout_height="?attr/actionBarSize"
                            app:popupTheme="@style/ThemeOverlay.AppCompat.Light"  />

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

接下来，我们必须使用一个容器布局： AppBarLayout 来让Toolbar响应滚动事件。响应滚动事件

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

    <android.support.v7.widget.Toolbar
                            android:id="@+id/toolbar"
                            android:layout_width="match_parent"
                            android:layout_height="?attr/actionBarSize"
```

```
                app:popupTheme="@style/ThemeOverlay.AppCompat.Light"  />

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

然后，我们需要定义AppBarLayout与滚动视图之间的联系。在RecyclerView或者任意支持嵌套滚动的view比如NestedScrollView上添加app:layout_behavior。support library包含了一个特殊的字符串资源@string/appbar_scrolling_view_behavior，它和AppBarLayout.ScrollingViewBehavior相匹配，用来通知AppBarLayout 这个特殊的view何时发生了滚动事件，这个behavior需要设置在触发事件（滚动）的view之上。注意：根据官方的谷歌文档，AppBarLayout目前必须是第一个嵌套在CoordinatorLayout里面的子view。

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

AppBarLayout里面定义的view只要设置了app:layout_scrollFlags属性，就可以在RecyclerView滚动事件发生的时候被触发：当CoordinatorLayout发现RecyclerView中定义了这个属性，它会搜索自己所包含的其他view，看看是否有view与这个behavior相关联。AppBarLayout.ScrollingViewBehavior描述了RecyclerView与AppBarLayout之间的依赖关系。RecyclerView的任意滚动事件都将触发AppBarLayout或者AppBarLayout里面view的改变。

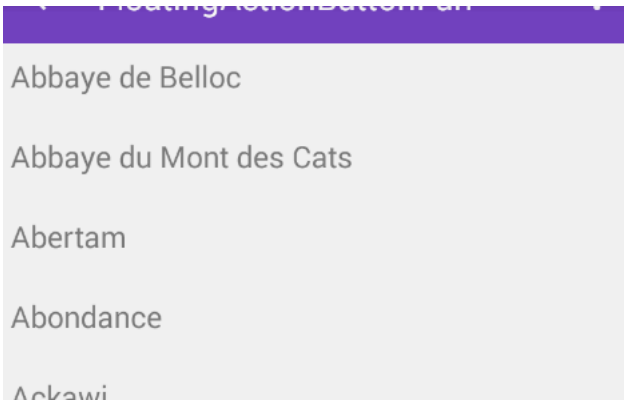
```
<android.support.design.widget.AppBarLayout
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:fitsSystemWindows="true"
                android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar">

                        <android.support.v7.widget.Toolbar
                                android:id="@+id/toolbar"
                                android:layout_width="match_parent"
                                android:layout_height="?attr/actionBarSize"
                                app:layout_scrollFlags="scroll|enterAlways"/>

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

enterAlways: 一旦向上滚动这个view就可见。app:layout_scrollFlags属性里面必须至少启用scroll这个flag，这样这个view才会滚动出屏幕，否则它将一直固定在顶部。可以使用的其他flag有：

- enterAlwaysCollapsed: 顾名思义，这个flag定义的是何时进入（已经消失之后何时再次显示）。假设你定义了一个最小高度（minHeight）同时enterAlways也定义了，那么view将在到达这个最小高度的时候开始显示，并且从这个时候开始慢慢展开，当滚动到顶部的时候展开完。
- exitUntilCollapsed: 同样顾名思义，这个flag时定义何时退出，当你定义了一个minHeight，这个view将在滚动到达这个最小高度的时候消失。

记住，要把带有scroll flag的view放在前面，这样收回的view才能让正常退出，而固定的view继续留在顶部。

此时，你应该注意到我们的Toolbar能够响应滚动事件了。

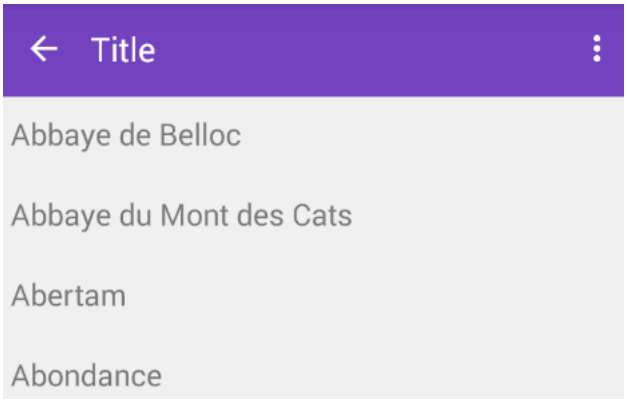


回到顶部

制造折叠效果

如果想制造toolbar的折叠效果，我们必须把Toolbar放在CollapsingToolbarLayout中：

```
<android.support.design.widget.CollapsingToolbarLayout
                        android:id="@+id/collapsing_toolbar"
                        android:layout_width="match_parent"
                        android:layout_height="match_parent"
                        android:fitsSystemWindows="true"
                        app:contentScrim="?attr/colorPrimary"
                        app:expandedTitleMarginEnd="64dp"
                        app:expandedTitleMarginStart="48dp"
                        app:layout_scrollFlags="scroll|exitUntilCollapsed">

                        <android.support.v7.widget.Toolbar
                                android:id="@+id/toolbar"
                                android:layout_width="match_parent"
                                android:layout_height="?attr/actionBarSize"
                                app:layout_scrollFlags="scroll|enterAlways"></android.support.v7.widget.Toolbar>
                </android.support.design.widget.CollapsingToolbarLayout>
```



现在效果就成了：

通常，我们我们都是设置Toolbar的title，而现在，我们需要把title设置在CollapsingToolBarLayout上，而不是Toolbar。

```
CollapsingToolbarLayout  collapsingToolbar  =(CollapsingToolbarLayout)  findViewById(R.id.collapsing_toolbar);
collapsingToolbar.setTitle("Title");
```

制造视差效果

回到顶部

CollapsingToolbarLayout还能让我们做出更高级的动画，比如在里面放一个ImageView，然后在它折叠的时候渐渐淡出。同时在用户滚动的时候title的高度也会随着改变。

为了制造出这种效果，我们添加一个定义了app:layout_collapseMode="parallax" 属性的ImageView。

```
<android.support.design.widget.CollapsingToolbarLayout
                android:id="@+id/collapsing_toolbar"
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:fitsSystemWindows="true"
                app:contentScrim="?attr/colorPrimary"
                app:expandedTitleMarginEnd="64dp"
                app:expandedTitleMarginStart="48dp"
                app:layout_scrollFlags="scroll|exitUntilCollapsed">

                <android.support.v7.widget.Toolbar
                        android:id="@+id/toolbar"
                        android:layout_width="match_parent"
                        android:layout_height="?attr/actionBarSize"
                        app:layout_scrollFlags="scroll|enterAlways"></android.support.v7.widget.Toolbar>
                <ImageView
                        android:src="@drawable/cheese_1"
                        app:layout_scrollFlags="scroll|enterAlways|enterAlwaysCollapsed"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:scaleType="centerCrop"
                        app:layout_collapseMode="parallax"
                        android:minHeight="100dp"/>

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

在CoordinatorLayout 与浮动操作按钮中我们讨论了一个自定义behavior的例子。注： 译文
http://www.jcodecraeer.com/a/anzhuokaifa/androidkaifa/2015/0718/3197.html 。

自定义Behavior

CoordinatorLayout的工作原理是搜索定义了CoordinatorLayout Behavior 的子view，不管是通过在xml中使用app:layout_behavior标签还是通过在代码中对view类使用@DefaultBehavior修饰符来添加注解。当滚动发生的时候，CoordinatorLayout会尝试触发那些声明了依赖的子view。

要自己定义CoordinatorLayout Behavior，你需要实现layoutDependsOn() 和onDependentViewChanged()两个方法。比如AppBarLayout.Behavior 就定义了这两个关键方法。这个behavior用于当滚动发生的时候让AppBarLayout发生改变。

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

public boolean onDependentViewChanged(CoordinatorLayout parent, View child, View dependency) {
        // check the behavior triggered
        android.support.design.widget.CoordinatorLayout.Behavior behavior = ((android.support.design.widget.CoordinatorLayout.LayoutParams) dependency.getLayoutI
        if (behavior instanceof AppBarLayout.Behavior) {
                // do stuff here
        }
}
```

理解如何实现这些自定义behavior的最好途径是研究AppBarLayout.Behavior 和 FloatingActionButtion.Behavior。虽然这些源代码还没有放出来，但是你可以使用Android Studio 1.2集成的反编译器来查看。

参考：Android的材料设计兼容库（Design Support Library）

本文到此结束，需要的朋友可以参考下。

返回

【上篇】Auto Installer——Android Apk 自动安装器
【下篇】Android开发——数据绑定时采用ViewModel代替Presenter

作者: summer

- 该日志由 summer 于2年前发表在Android开发经验分类下，最后更新于 2015年08月03日.
- 转载请注明: Android开发——CoordinatorLayout用法详解 | 阳和移动开发 +复制链接
-

抱歉!评论已关闭.

栏目导航

-

资讯

- 手机资讯
- 移动开发资讯

-

开发工具相关

- 开发工具和框架
- UI设计与开发
- 集成IDE
- SDK
- 游戏开发
- 代码转换
- 在线开发工具

-

数据库相关

- 数据库
- NoSQL

-

组件类库

- ORM
- REST
- 图片处理
- 广告
- JSON
- 类库管理

- 网络通信
- 通用框架
- 任务管理
- 评级(Ratings)
- PDF
- 工具(Toolkits)
- 实用工具类(Utils)
- XMPP
- 日志(Logging)
- 消息推送(Push)
- 发布平台
- Bug报告(Crash Reports)
- 缓存(Caching)
- 切面(Aspects)
- 音频(Audio)
- 条形码(Barcodes)
- Blocks
- 特效(Specially Effect)
- 浏览器组件
- 并发(Concurrency)
- 云(Cloud)
- 文件系统(File system)
- 地图(Maps)
- 图表(Chart)
- 内容提供(ContentProvider)
- 应用锁(AppLock)
- Parcelables
- RSS
- 动画(Animations)
- 框架架构(Architecture)
- SVG
- Gradle插件(Gradle Plugins)
- 安全(Security)
- 依赖注入(Dependency Injections)
- 字体(Fonts)
- 视频(Video)
- 本地化(Localization)
- 后台处理(Background Processing)
- 事件总线(Event Buses)
- Preferences
- 选择器（Picker）
- 日期选择器(Date Pickers)
- 图片选择器(Image Pickers)
- 颜色选择器(Color Pickers)
- 视图效果（View Effects）
- APT
- 蓝牙(Bluetooth)
- 变更日志(ChangeLog)
- 视图适配器(View Adapters)
- 翻页效果(Curl/Flip Effects)
- 键盘
- 视图切换(View Transition)
- 图标(Icons)
- 相机(Camera)
- XML
- ANR(应用无响应)
- 波纹效果(Ripple Effects)
- 通知(Notifications)
- 分析（Analytics）
- 任务调度(Job Schedulers)
- OpenGL
- FRP
- USB
- Intent（意图）
- Drawables
- 定位(Location)
- SOAP
- Purchases
- 天气(Weather)

UI组件

- 滑动面板(Sliding Panels)
- 表格视图(Grid Views)
- 对话框(Dialogs)
- 进度条
- UI布局(Layouts)
- 按钮(Buttons)
- 日历(Calendars)
- 滚动视图(ScrollView)
- 文本输入(Textfield)
- 操作栏(Action Bars)
- 列表视图(List Views)
- 下拉刷新(Pull Refresh)
- 文字视图(Text Views)
- Toasts
- 标签栏(Tab Bars)
- 菜单（Menu）
- 图标提醒(Badges)
- Table Views
- 状态栏(Status Bars)
- Range Bars
- View Pagers
- 模板引擎(Template Engines)
- 引导页（Intro&Guide View）
- 指示器(ActivityIndicator)
- 弹出视图（Popup View）
- 开关（Switch）
- 标签(Label)
- 手势交互(Gesture)
- 滑杆(Slider)
- Cards
- 导航栏(Navigation Bar)
- 透明指示层(HUD)
- 视图(Views)
- Recycler Views
- Carousels
- 校验(Validation)

测试工具

调试工具

社交分享组件

插件

开发语言

其它

- 关于本站

返回首页