**[CoordinatorLayout的使用如此简单](http://blog.csdn.net/huachao1001/article/details/51554608)**

# CoordinatorLayout能做什么

在学习CoordinatorLayout之前，很有必要了解CoordinatorLayout能帮我们做什么，从名字上可以看出，就是帮我们协调子View的。怎么个协调法呢?就是它根据我们的定制，帮助我们协调各个子View的布局。

稍微解释一下这个动画，蓝色的矩形是我们一个普通View，黄色的Hello是一个Button。我们水平拖动蓝色矩形时，黄色Button查着与蓝色矩形相反方向移动；竖直移动蓝色矩形时，黄色也跟着竖直。简而言之：它们在竖直方向同步移动，在水平方向相反。

这个效果如果让你不用CoordinatorLayout去实现，应该没有任何问题，但是代码的耦合度应该非常大，你的代码必须要持有2个View的引用，然后在onTouchEvent里面做各种判断。如果我们想要实现的功能是，有更多的View要根据蓝色的View的移动相应作出响应，那么那就得在蓝色View的onTounchEvent里面针对其他的View处理各种逻辑。这耦合度未免太伤感了~

而CoordinatorLayout既然号称能帮我们协调子View的布局，我们接下来看看CoordinatorLayout如何实现~

# CoordinatorLayout使用

CoordinatorLayout的使用核心是Behavior，Behavior就是执行你定制的动作。在讲Behavior之前必须先理解两个概念：Child和Dependency，什么意思呢？Child当然是子View的意思了，是谁的子View呢，当然是CoordinatorLayout的子View；其实Child是指要执行动作的CoordinatorLayout的子View。而Dependency是指Child依赖的View。比如上面的gif图中，蓝色的View就是Dependency，黄色的View就是Child，因为黄色的View的动作是依赖于蓝色的View。简而言之，就是如过Dependency这个View发生了变化，那么Child这个View就要相应发生变化。发生变化是具体发生什么变化呢？这里就要引入Behavior，Child发生变化的具体执行的代码都是放在Behavior这个类里面。

怎么使用Behavior呢，首先，我们定义一个类，继承CoordinatorLayout.Behavior<T>,其中，泛型参数T是我们要执行动作的View类，也就是Child。然后就是去实现Behavior的两个方法：

/\*\*

\* 判断child的布局是否依赖dependency

\*/

@Override

public boolean layoutDependsOn(CoordinatorLayout parent, T child, View dependency) {

boolean rs;

//根据逻辑判断rs的取值

//返回false表示child不依赖dependency，ture表示依赖

return rs;

}

/\*\*

\* 当dependency发生改变时（位置、宽高等），执行这个函数

\* 返回true表示child的位置或者是宽高要发生改变，否则就返回false

\*/@Overridepublic boolean onDependentViewChanged(CoordinatorLayout parent, T child, View dependency) {

//child要执行的具体动作

return true;

}

有了上面的概念后，我们看看具体怎么去实现吧~

为了响应跟随手指移动的操作，我们定义一个非常简单的View，这个View只响应跟随手指移动，将这个View作为Dependency。由于过于简单，这个View源码不粘贴，我们只需知道这个View的类名叫：TempView。

我们看看Behavior的使用：

package com.hc.studyCoordinatorLayout;

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

/\*\*

\* Package com.hc.studyCoordinatorLayout

\* Created by HuaChao on 2016/6/1.

\*/public class MyBehavior extends CoordinatorLayout.Behavior<Button> {

private int width;

public MyBehavior(Context context, AttributeSet attrs) {

super(context, attrs);

DisplayMetrics display = context.getResources().getDisplayMetrics();

width = display.widthPixels;

}

@Override

public boolean layoutDependsOn(CoordinatorLayout parent, Button child, View dependency) {

//如果dependency是TempView的实例，说明它就是我们所需要的Dependency

return dependency instanceof TempView;

}

//每次dependency位置发生变化，都会执行onDependentViewChanged方法

@Override

public boolean onDependentViewChanged(CoordinatorLayout parent, Button btn, View dependency) {

//根据dependency的位置，设置Button的位置

int top = dependency.getTop();

int left = dependency.getLeft();

int x = width - left - btn.getWidth();

int y = top;

setPosition(btn, x, y);

return true;

}

private void setPosition(View v, int x, int y) {

CoordinatorLayout.MarginLayoutParams layoutParams = (CoordinatorLayout.MarginLayoutParams) v.getLayoutParams();

layoutParams.leftMargin = x;

layoutParams.topMargin = y;

v.setLayoutParams(layoutParams);

}

}

OK，现在我们为Button类指定了Dependency，并且定义好了跟随Dependency一直变化的动作（Behavior），接下来我们就要指定好为哪个具体的Button实例来绑定这些。方法很简单，直接在布局文件指定就好：

<?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:layout\_width="match\_parent"

android:layout\_height="match\_parent"

tools:context="com.hc.studyCoordinatorLayout.MainActivity">

<Button android:id="@+id/btn"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_marginLeft="300dp"

android:layout\_marginTop="300dp"

android:background="#FFCC00"

android:text="Hello"

app:layout\_behavior="com.hc.studyCoordinatorLayout.MyBehavior" />

<com.hc.studyCoordinatorLayout.TempView

android:layout\_width="100dp"

android:layout\_height="100dp"

android:layout\_marginLeft="300dp"

android:layout\_marginTop="300dp"

android:background="#3366CC" /></android.support.design.widget.CoordinatorLayout>

是不是很简单呢？我们只需关注Behavior的编写就好了，把Child和Dependency之间的关系完全解耦了~