CoordinatorLayout源码解析

官方介绍：

CoordinatorLayout is a super-powered FrameLayout.

CoordinatorLayout is intended for two primary use cases:

1. As a top-level application decor or chrome layout

2. As a container for a specific interaction with one or more child views

CoordinatorLayout是一个具有超级强大功能的FrameLayout

主要适用于2个情况

1. 作为顶层布局

2. 作为与一个或者多个子视图特定交互的容器

通过指定子视图behaviors来实现子视图之间交互

场景：

1. 一个容器中的某个viewA状态改变了，比如大小/位置[肯定会触发重绘]，但是同时，其他view需要监控该viewA的变化来做出其他反应，如果实现该功能？

2. 容器中有一个列表，在用户手指进行滑动的时候，需要优先滑动其他View位置，比如已经处于扩展状态的ImageView，需要先收起该ImageView再滑动list，中间不可以中断，如何流程的处理滑动事件？

引入CoordinatorLayout（协调布局）

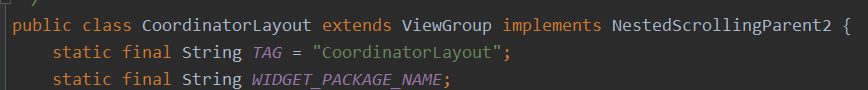
源码：

androidx.coordinatorlayout.widget.CoordinatorLayout

版本： 1.0.0

主要就是指定View的behaviors来绑定View之间的关系，并且交给CoordinatorLayout来分发view改变状态并通知关心的Views们

CoordinatorLayout结构：



继承自ViewGroup，实现NestedScrollingParent2接口

可以看出：

1. 自定义实现的ViewGroup类，实现3个重要方法，onMeasure，onLayout，onDraw

2. 实现NestedScrollingParent2接口，是一个嵌套滑动父类，可以接收嵌套滑动子视图的滑动事件并处理，用于分发给其他关心的View，实现滑动交互2问题（目测）

构造函数：

1. 调用setWindowInsets，状态栏适配

2. 添加层级监听

public CoordinatorLayout(@NonNull Context context, @Nullable AttributeSet attrs, @AttrRes int defStyleAttr) {

。。。

setupForInsets();

super.setOnHierarchyChangeListener(new HierarchyChangeListener());  
}

HierarchyChangeListener：通知View的添加移除监听[可以通知依赖的View移除]

添加视图监听：

@Override  
public void onAttachedToWindow() {  
 super.onAttachedToWindow();  
 resetTouchBehaviors(false);  
 if (mNeedsPreDrawListener) {  
 if (mOnPreDrawListener == null) {  
 mOnPreDrawListener = new OnPreDrawListener();  
 }  
 final ViewTreeObserver vto = getViewTreeObserver();  
 vto.addOnPreDrawListener(mOnPreDrawListener);  
 }  
 。。。  
 mIsAttachedToWindow = true;  
}  
  
@Override  
public void onDetachedFromWindow() {  
 super.onDetachedFromWindow();  
 resetTouchBehaviors(false);  
 if (mNeedsPreDrawListener && mOnPreDrawListener != null) {  
 final ViewTreeObserver vto = getViewTreeObserver();  
 vto.removeOnPreDrawListener(mOnPreDrawListener);  
 }  
 if (mNestedScrollingTarget != null) {  
 onStopNestedScroll(mNestedScrollingTarget);  
 }  
 mIsAttachedToWindow = false;  
}

这是一对方法：

onAttachedToWindow

1. resetTouchBehaviors取消所有触摸事件，因为view刚刚被attach/detach需要通知所有添加behaviors的View touch事件：

private void resetTouchBehaviors(boolean notifyOnInterceptTouchEvent) {  
 final int childCount = getChildCount();  
 for (int i = 0; i < childCount; i++) {  
 final View child = getChildAt(i);  
 final LayoutParams lp = (LayoutParams) child.getLayoutParams();  
 final Behavior b = lp.getBehavior();  
 if (b != null) {  
 final long now = SystemClock.*uptimeMillis*();  
 final MotionEvent cancelEvent = MotionEvent.*obtain*(now, now,  
 MotionEvent.*ACTION\_CANCEL*, 0.0f, 0.0f, 0);  
 if (notifyOnInterceptTouchEvent) {  
 b.onInterceptTouchEvent(this, child, cancelEvent);  
 } else {  
 b.onTouchEvent(this, child, cancelEvent);  
 }  
 cancelEvent.recycle();  
 }  
 }  
  
 for (int i = 0; i < childCount; i++) {  
 final View child = getChildAt(i);  
 final LayoutParams lp = (LayoutParams) child.getLayoutParams();  
 lp.resetTouchBehaviorTracking();  
 }  
 mBehaviorTouchView = null;  
 mDisallowInterceptReset = false;  
}

清空点击事件

2. 当前ViewGroup添加OnPreDrawListener实现类，作用：

class OnPreDrawListener implements ViewTreeObserver.OnPreDrawListener {  
 @Override  
 public boolean onPreDraw() {  
 onChildViewsChanged(*EVENT\_PRE\_DRAW*);  
 return true;  
 }  
}

只实现了onChildViewsChanged方法

核心方法onChildViewsChanged：

final void onChildViewsChanged(@DispatchChangeEvent final int type) {  
 final int layoutDirection = ViewCompat.*getLayoutDirection*(this);  
 final int childCount = mDependencySortedChildren.size();  
 final Rect inset = *acquireTempRect*();  
 final Rect drawRect = *acquireTempRect*();  
 final Rect lastDrawRect = *acquireTempRect*();  
  
 for (int i = 0; i < childCount; i++) {  
 final View child = mDependencySortedChildren.get(i);  
 final LayoutParams lp = (LayoutParams) child.getLayoutParams();  
 if (type == *EVENT\_PRE\_DRAW* && child.getVisibility() == View.*GONE*) {  
 // Do not try to update GONE child views in pre draw updates.  
 continue;  
 }  
  
 // Check child views before for anchor  
 for (int j = 0; j < i; j++) {  
 final View checkChild = mDependencySortedChildren.get(j);  
  
 if (lp.mAnchorDirectChild == checkChild) {  
 offsetChildToAnchor(child, layoutDirection);  
 }  
 }  
  
 // Get the current draw rect of the view  
 getChildRect(child, true, drawRect);  
  
 // Accumulate inset sizes  
 if (lp.insetEdge != Gravity.*NO\_GRAVITY* && !drawRect.isEmpty()) {  
 final int absInsetEdge = GravityCompat.*getAbsoluteGravity*(  
 lp.insetEdge, layoutDirection);  
 switch (absInsetEdge & Gravity.*VERTICAL\_GRAVITY\_MASK*) {  
 case Gravity.*TOP*:  
 inset.top = Math.*max*(inset.top, drawRect.bottom);  
 break;  
 case Gravity.*BOTTOM*:  
 inset.bottom = Math.*max*(inset.bottom, getHeight() - drawRect.top);  
 break;  
 }  
 switch (absInsetEdge & Gravity.*HORIZONTAL\_GRAVITY\_MASK*) {  
 case Gravity.*LEFT*:  
 inset.left = Math.*max*(inset.left, drawRect.right);  
 break;  
 case Gravity.*RIGHT*:  
 inset.right = Math.*max*(inset.right, getWidth() - drawRect.left);  
 break;  
 }  
 }  
  
 // Dodge inset edges if necessary  
 if (lp.dodgeInsetEdges != Gravity.*NO\_GRAVITY* && child.getVisibility() == View.*VISIBLE*) {  
 offsetChildByInset(child, inset, layoutDirection);  
 }  
  
 if (type != *EVENT\_VIEW\_REMOVED*) {  
 // Did it change? if not continue  
 getLastChildRect(child, lastDrawRect);  
 if (lastDrawRect.equals(drawRect)) {  
 continue;  
 }  
 recordLastChildRect(child, drawRect);  
 }  
  
 // Update any behavior-dependent views for the change  
 for (int j = i + 1; j < childCount; j++) {  
 final View checkChild = mDependencySortedChildren.get(j);  
 final LayoutParams checkLp = (LayoutParams) checkChild.getLayoutParams();  
 final Behavior b = checkLp.getBehavior();  
  
 if (b != null && b.layoutDependsOn(this, checkChild, child)) {  
 if (type == *EVENT\_PRE\_DRAW* && checkLp.getChangedAfterNestedScroll()) {  
 // If this is from a pre-draw and we have already been changed  
 // from a nested scroll, skip the dispatch and reset the flag  
 checkLp.resetChangedAfterNestedScroll();  
 continue;  
 }  
  
 final boolean handled;  
 switch (type) {  
 case *EVENT\_VIEW\_REMOVED*:  
 // EVENT\_VIEW\_REMOVED means that we need to dispatch  
 // onDependentViewRemoved() instead  
 b.onDependentViewRemoved(this, checkChild, child);  
 handled = true;  
 break;  
 default:  
 // Otherwise we dispatch onDependentViewChanged()  
 handled = b.onDependentViewChanged(this, checkChild, child);  
 break;  
 }  
  
 if (type == *EVENT\_NESTED\_SCROLL*) {  
 // If this is from a nested scroll, set the flag so that we may skip  
 // any resulting onPreDraw dispatch (if needed)  
 checkLp.setChangedAfterNestedScroll(handled);  
 }  
 }  
 }  
 }  
  
 *releaseTempRect*(inset);  
 *releaseTempRect*(drawRect);  
 *releaseTempRect*(lastDrawRect);  
}

这边我们将拆解方法单独介绍：

1. for之外

final int layoutDirection = ViewCompat.*getLayoutDirection*(this);  
final int childCount = mDependencySortedChildren.size();  
final Rect inset = *acquireTempRect*();  
final Rect drawRect = *acquireTempRect*();  
final Rect lastDrawRect = *acquireTempRect*();

1.1 获取布局方向，View个数，初始化3个rect矩形

获取ChildView（这里使用的mDependencySortedChildren初始化如下：）

void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {  
 prepareChildren();  
 ensurePreDrawListener();

prepareChildren保存子列表View

ensurePreDrawListener确保能够回调behavior状态改变

1.2

for (int i = 0; i < childCount; i++) {  
 final View child = mDependencySortedChildren.get(i);  
 final LayoutParams lp = (LayoutParams) child.getLayoutParams();  
 // Get the current draw rect of the view  
 getChildRect(child, true, drawRect);

if (type != *EVENT\_VIEW\_REMOVED*) {  
 // Did it change? if not continue  
 getLastChildRect(child, lastDrawRect);  
 if (lastDrawRect.equals(drawRect)) {  
 continue;  
 }  
 recordLastChildRect(child, drawRect);  
}

}

getChildRect方法获取drawRect当前View的边界与位置

getLastChildRect(child, lastDrawRect) 获取上一次View的位置

如果上一次位置与当前位置一致，说明View没有发生变化，那么continue，查询下一个childView

if (lastDrawRect.equals(drawRect)) {  
 continue;  
}

否则刷新当前位置

1.3 查询相关监听的behavior，分发结果

// Update any behavior-dependent views for the change  
for (int j = i + 1; j < childCount; j++) {  
 final View checkChild = mDependencySortedChildren.get(j);  
 final LayoutParams checkLp = (LayoutParams) checkChild.getLayoutParams();  
 final Behavior b = checkLp.getBehavior();  
  
 if (b != null && b.layoutDependsOn(this, checkChild, child)) {  
 if (type == *EVENT\_PRE\_DRAW* && checkLp.getChangedAfterNestedScroll()) {  
 // If this is from a pre-draw and we have already been changed  
 // from a nested scroll, skip the dispatch and reset the flag  
 checkLp.resetChangedAfterNestedScroll();  
 continue;  
 }  
  
 final boolean handled;  
 switch (type) {  
 case *EVENT\_VIEW\_REMOVED*:  
 // EVENT\_VIEW\_REMOVED means that we need to dispatch  
 // onDependentViewRemoved() instead  
 b.onDependentViewRemoved(this, checkChild, child);  
 handled = true;  
 break;  
 default:  
 // Otherwise we dispatch onDependentViewChanged()  
 handled = b.onDependentViewChanged(this, checkChild, child);  
 break;  
 }  
  
 if (type == *EVENT\_NESTED\_SCROLL*) {  
 // If this is from a nested scroll, set the flag so that we may skip  
 // any resulting onPreDraw dispatch (if needed)  
 checkLp.setChangedAfterNestedScroll(handled);  
 }  
 }  
}

2. for之外收尾

*releaseTempRect*(inset);  
*releaseTempRect*(drawRect);  
*releaseTempRect*(lastDrawRect);

使用Pool类提供缓存，拿取方便，但是容量有限

private static final Pools.Pool<Rect> *sRectPool* = new Pools.SynchronizedPool<>(12);

此时就完成了协调view变化通知变化的能力

其他：

嵌套滑动NestedScrollingParent2父类通知behavior变化如下：

// for逐一查询对应behavior并分发

举例：

@Override  
public boolean onStartNestedScroll(View child, View target, int axes, int type) {  
 boolean handled = false;  
  
 final int childCount = getChildCount();  
 for (int i = 0; i < childCount; i++) {  
 final View view = getChildAt(i);  
 if (view.getVisibility() == View.*GONE*) {  
 // If it's GONE, don't dispatch  
 continue;  
 }  
 final LayoutParams lp = (LayoutParams) view.getLayoutParams();  
 final Behavior viewBehavior = lp.getBehavior();  
 if (viewBehavior != null) {  
 final boolean accepted = viewBehavior.onStartNestedScroll(this, view, child,  
 target, axes, type);  
 handled |= accepted;  
 lp.setNestedScrollAccepted(type, accepted);  
 } else {  
 lp.setNestedScrollAccepted(type, false);  
 }  
 }  
 return handled;  
}

1. 遍历childView

2. 取childView的behavior

3. 执行NestedScrollingParent2方法

此时，协调布局源码分析完毕

扩展：

setupForInsets()方法解析

1. 什么是windowInset？

解释：

代表window窗口内嵌View框架边距，例如：虚拟导航键/状态栏

如果我们需要将应用全屏展示，并且占据虚拟导航键/状态栏位置，那么需要处理window的边距

此时View# onApplyWindowInsets(WindowInsets insets)获取windowInsets

private void setupForInsets() {  
 if (Build.VERSION.*SDK\_INT* < 21) {  
 return;  
 }  
  
 if (ViewCompat.*getFitsSystemWindows*(this)) {  
 if (mApplyWindowInsetsListener == null) {  
 mApplyWindowInsetsListener =  
 new androidx.core.view.OnApplyWindowInsetsListener() {  
 @Override  
 public WindowInsetsCompat onApplyWindowInsets(View v,  
 WindowInsetsCompat insets) {  
 return setWindowInsets(insets);  
 }  
 };  
 }  
 // First apply the insets listener  
 ViewCompat.*setOnApplyWindowInsetsListener*(this, mApplyWindowInsetsListener);  
  
 // Now set the sys ui flags to enable us to lay out in the window insets  
 setSystemUiVisibility(View.*SYSTEM\_UI\_FLAG\_LAYOUT\_STABLE* | View.*SYSTEM\_UI\_FLAG\_LAYOUT\_FULLSCREEN*);  
 } else {  
 ViewCompat.*setOnApplyWindowInsetsListener*(this, null);  
 }  
}

//通过ViewCompat.setOnApplyWindowInsetsListener方法返回windowInset

如果全屏模式下

setSystemUiVisibility(View.*SYSTEM\_UI\_FLAG\_LAYOUT\_STABLE* | View.*SYSTEM\_UI\_FLAG\_LAYOUT\_FULLSCREEN*)

并且layout中：

android:fitsSystemWindows="true"

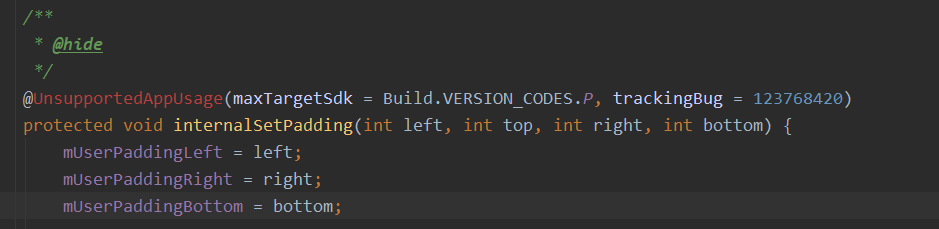
返回windowInset不为null，那么View将产生内边距

源码如下：

View# fitSystemWindowsInt

private boolean fitSystemWindowsInt(Rect insets) {  
 if ((mViewFlags & *FITS\_SYSTEM\_WINDOWS*) == *FITS\_SYSTEM\_WINDOWS*) {  
 mUserPaddingStart = *UNDEFINED\_PADDING*;  
 mUserPaddingEnd = *UNDEFINED\_PADDING*;  
 Rect localInsets = *sThreadLocal*.get();  
 if (localInsets == null) {  
 localInsets = new Rect();  
 *sThreadLocal*.set(localInsets);  
 }  
 boolean res = computeFitSystemWindows(insets, localInsets);  
 mUserPaddingLeftInitial = localInsets.left;  
 mUserPaddingRightInitial = localInsets.right;  
 internalSetPadding(localInsets.left, localInsets.top,  
 localInsets.right, localInsets.bottom);  
 return res;  
 }  
 return false;  
}

修改padding值



此时如果View设置了padding值会被新的padding覆盖更新