### Launcher3基础

**第一部分 桌面的结构**

整体布局：

Launcher.xml:

<FrameLayout

<DragLaye>

<!-桌面布局->

<Workspace>

<!—属性配置-- >

</Workspace>

<includeandroid:id=*"@+id/qsb\_bar"*layout=*"@layout/qsb\_bar"* />

<include layout=*"@layout/hotseat"/>*

*……*

*<!-抽屉布局->*

<include layout=*"@layout/apps\_customize\_pane"/>*

</DragLayer>

</FrameLayout>

桌面的布局文件还是很清晰的，整个桌面只有一个Activity， Workspace和AllApp之间的切换是通过设置彼此是否可见（visibility）实现的。

（1）快捷方式层（Workspace）

搜索/垃圾箱/应用信息

BubbleText

DeleteDropTarget/InfoDropTarget/SearchDropTargetBar

Workspace

Indicator

Hotseat

Workspace是一个PageView容器，里面包含了多页的Celllayout。Celllayout跟Turbo有个比较大的区别是：Celllayout的直接子节点并不是页面上的小元素而是----ShortcutAndWidgetContainer的一个实例：mShortcutsAndWidgets,这个实例的子节点才是真正的桌面可视元素：BubbleTextView。

比较Turbo，BubbleTextView对应Icon3D，有个细微的差别是BubbleTextView中的ItemInfo是以一个Tag的形式保存的，并没有直接在类中声明一个成员变量。

（2）抽屉层

<com.android.launcher3.AppsCustomizeTabHost>

<com.android.launcher3.FocusOnlyTabWidget/>

<com.android.launcher3.AppsCustomizePagedView

<!---应用图片和指示器->

/>

</ com.android.launcher3.AppsCustomizeTabHost>

抽屉层的架构是非常接近于Turbo的，分了两部分：TabHost和AppsPage。

AppsCustomizePagedView和Workspace一样也是继承于PageView。

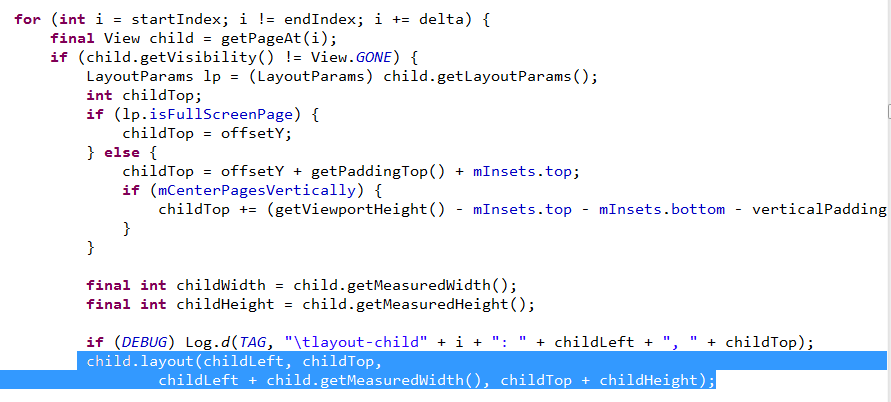
**第二部分 桌面的触摸事件**

1. Workspace滑动，翻页

Workspace继承于PagedView。Workspace的翻页，滑动效果是有PagedView类实现的。

在理解launcher3如何移动屏幕之前，我们需要先了解Workspace是如何布局Celllayout的。

Workspace.java的onLayout函数：



从这个函数可以看出桌面的Cellayout是线性布局的，如果下图。

Cellayout0

Workspace可视区域

Cellayout1

Cellayout2

Cellayout3

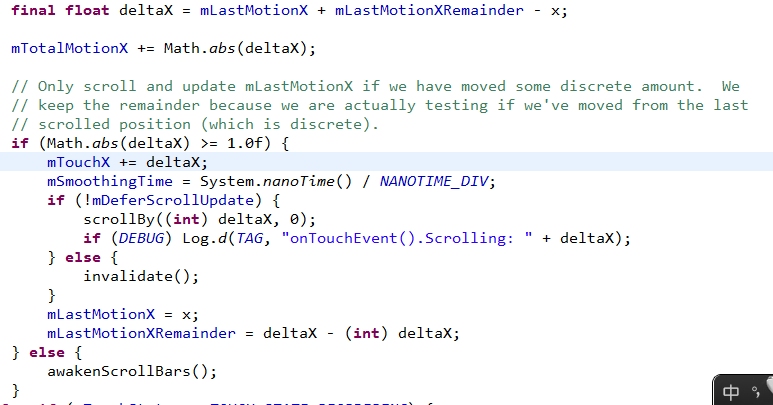
翻页滑动的实现原理是通过Scroller这个类根据滑动的距离来移动Page的位置。Scroller类是一个非常简单的帮助类他本身并不复制绘画，类似于Tween补间动画，你只要设置一个终点值，Scroller类会在每一帧返回一个过程值，我们拿到这个过程值就可以知道屏幕需要移动的位置了。有几个比较重要的函数，如果搞清楚了这几个函数的用途，就不难理解Launcher3是如何处理屏幕的移动和翻页了。

Android每个View都有两个函数，scrollBy,scrollTo。前者是表示移动相对位置，后者表示绝对位置，其中scrollBy是通过ScrollT函数实现的。所以只要知道了当前的偏移量就可以通过调用这两个函数实现它子视图的滑动。

1.1桌面如何滑动

PagedView.java

public boolean onTouchEvent(MotionEvent ev){

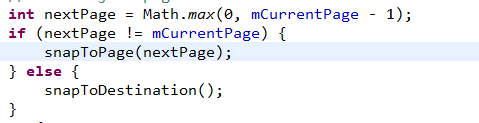


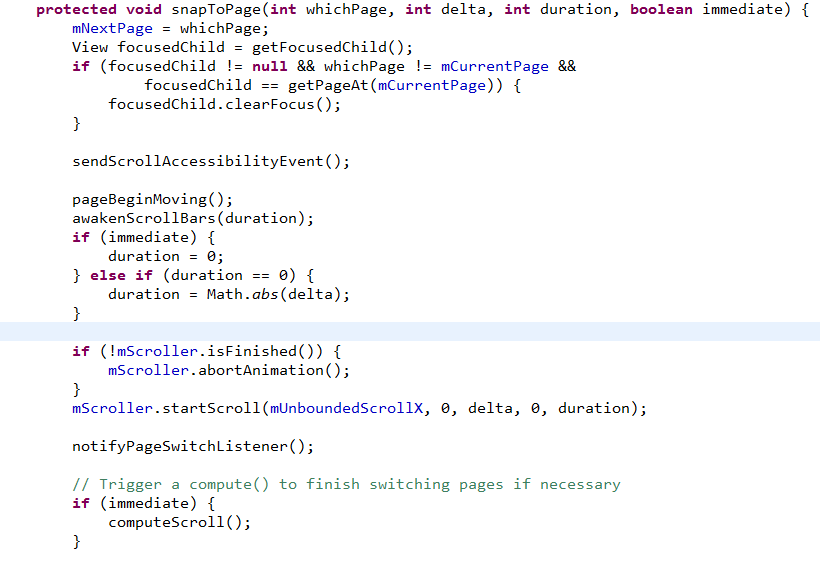
}

可以看到直接使用scrollBy函数实现滑动。

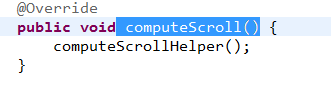
1.2桌面如何翻页

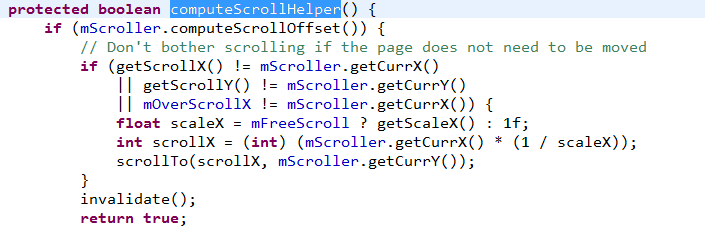
MotionEvent.ACTION\_UP：





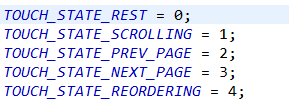
实际上通过Scroller函数实现自动翻页效果。mScroller.startScroll函数会回调View的





整个滑动，翻页效果介绍完毕。

Workspace有4个常量来标示当前屏幕的状态：



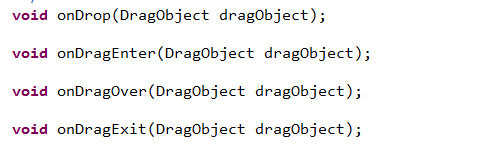
分别标示：静止，滑动，前一页，后一页

(2)拖拽

2.1 主要类的介绍

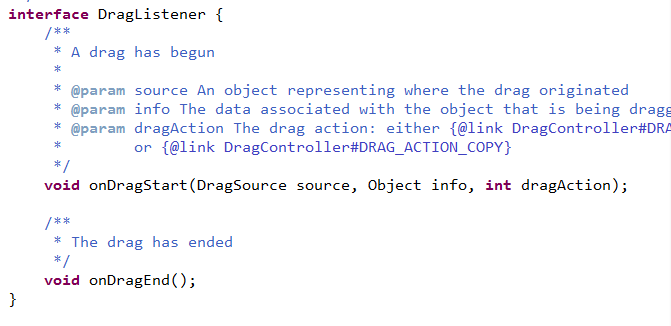
对于Turbo而言，我们已经很熟悉了Draglayer3D的实现原理。后面部分会对照Draglayer3D来分析launcher3是如何实现在桌面拖拽的。Launcher3主要有4个重要的文件来实现这两个功能：DragController.java，DragLayer.java，DropTarget.java,DragView.java,DragSource.java。

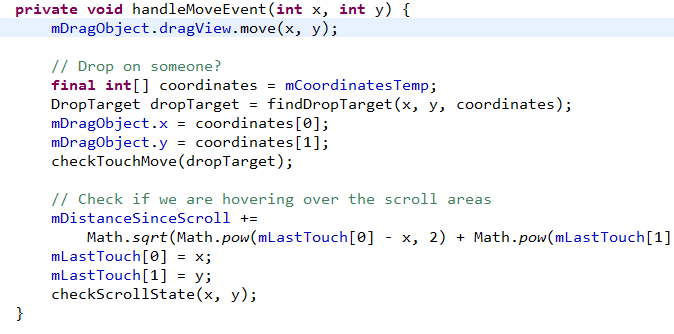
其中DragController可以认为是逻辑层，DragLayer可以认为是一个辅助类，为了帮助逻辑层截获Touch事件或者获得坐标位置等信息。DragSource表示这个容器中可以生成一个拖拽的View.和DragSource相反，所有可以接收被拖拽的View类（比如Workspace，Hotseat）都必须实现接口DropTarget.java，后面我们称实现了这个接口的类叫做“目标容器”，需要重构以下几个接口：



看到这几个接口的名字就可以跟Turbo对应起来了。他们的调用时机依次是：当DragView（这个类后面介绍）在目标容器中掉落的时候，当DragView进入目标容器的时候，当DragView在目标容器移动的时候，当DragView离开目标容器的时候。拖拽的时候会在桌面上生成一个残影，表示如果放手了这个DragView会被drop到哪里。

**DragController.java**：Class for initiating a drag within a view or across multiple views.





这个函数对应Draglayer3D.java的scroll函数。

DragLayer.java: A ViewGroup that coordinates dragging across its descendants

这个类主要是计算拖拽的View的坐标和做移动动画等，（android中的动画有点类似Tween这个引擎，方式都是设置目标值，过程自动完成）后续如果我们要在拖拽上做文章的话，可能这个类是最好实现的地方。

对Turbo而言，我们的Draglayout3D，整合了这两个类的内容，实现了里面的基本事件。

2.2 拖拽的响应时机

触发拖拽事件的条件—长按桌面上的可视View

Launcher.java

Launcher 注册了长按事件，当某个view被长按的时候会调用onLongClick(View v)函数。

public boolean onLongClick(View v){

//经过一系列的判断最终走入workspace的startDrag函数。

mWorkspace.startDrag(longClickCellInfo);

}

Workspace.java

void startDrag(CellLayout.CellInfo cellInfo){

//先把被拖拽的这个view设置不可见。

View child = cellInfo.cell;

mDragInfo = cellInfo;

child.setVisibility(INVISIBLE);

//生成悬浮在桌面的轮廓图片

mDragOutline = createDragOutline(child, canvas,

DRAG\_BITMAP\_PADDING);

beginDragShared(child, this);

}

Workspace：

public void beginDragShared(View child, DragSource source){

// The drag bitmap follows the touch point around on the screen

//出去图标标题什么的…

**final** Bitmap b = createDragBitmap(child, **new** Canvas(), *DRAG\_BITMAP\_PADDING*);

mDragController.startDrag(b,…);

}

DragController.java

**public** **void** startDrag(Bitmap b, **int** dragLayerX, **int** dragLayerY,

DragSource source, Object dragInfo, **int** dragAction, Point dragOffset, Rect dragRegion,

**float** initialDragViewScale){

//通知所有注册了拖拽事件的容器

for (DragListener listener : mListeners) {

listener.onDragStart(source, dragInfo, dragAction);

}

//设置拖拽状态，这个很重要，后面的touch事件都是根据这个来判断是否处于拖拽状态的

mDragging = true;

//创建一个DragView

**final** DragView dragView = mDragObject.dragView = **new** DragView(mLauncher, b,…);

//这个show函数主要的作用就是把自己加到draglayer中

// mDragLayer.addView(this);

dragView.show(mMotionDownX, mMotionDownY);

//进入move流程，这个函数会在后面介绍

handleMoveEvent(mMotionDownX, mMotionDownY);

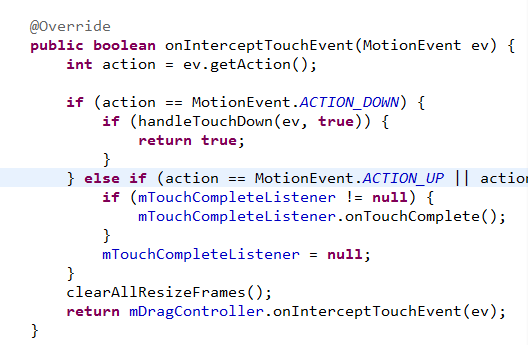
}

总结下以上过程：首先是launcher得到了长按事件，launcher获得被拖拽的view，并且把这个view传递给了DragController。DragController得到这个view后生成了一个DragView并且把它添加到了DragLayer中。

2.3 拖拽过程

手指在移动的过程中，touch事件会从root节点开始寻找需要consume这个事件的view。

从launcher.xml文件中可以知道Draglayer是最顶级节点，所以touch事件首先会被传到Draglayer类中。



这个函数最主要的部分在最后的return mDragController.onInterceptTouchEvent(ev);

DragController.java

public boolean onInterceptTouchEvent(MotionEvent ev) {

…….

//返回这个标志

return mDragging;

}

当返回true后，桌面所有的touch事件都将被Draglayer消费掉。我们这里只关心launcher是如何处理Move事件的。

Draglayer的onTouchEvent**函数会调用DragController的onTouchEvent函数，所以真正处理所有的touch事件的代码是在DragController.java文件中。**

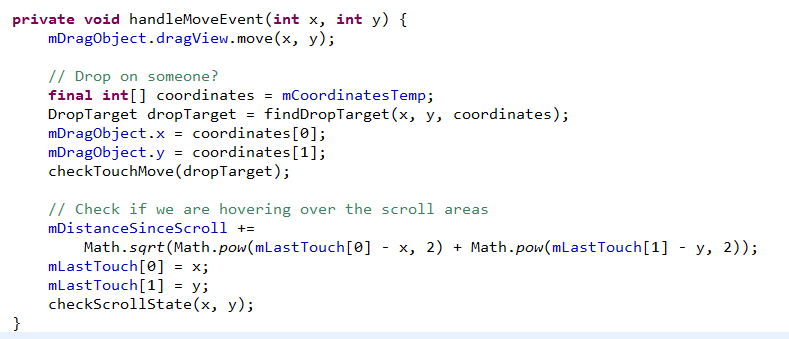
DragController.java

**public boolean onTouchEvent(MotionEvent ev){**

****

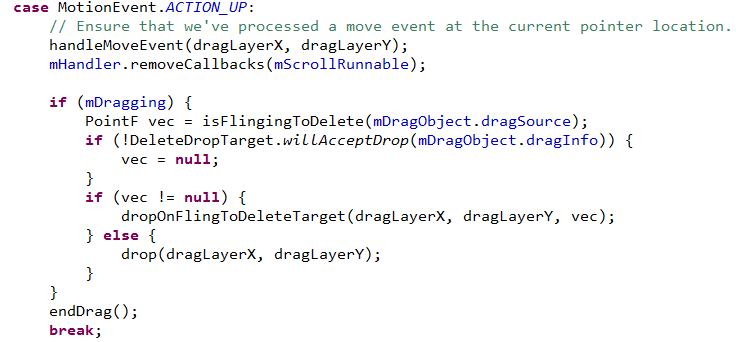
**}**

**这个函数在之前呼出DragView的时候有提到过，下面将这个函数的主要部分抄出来。**

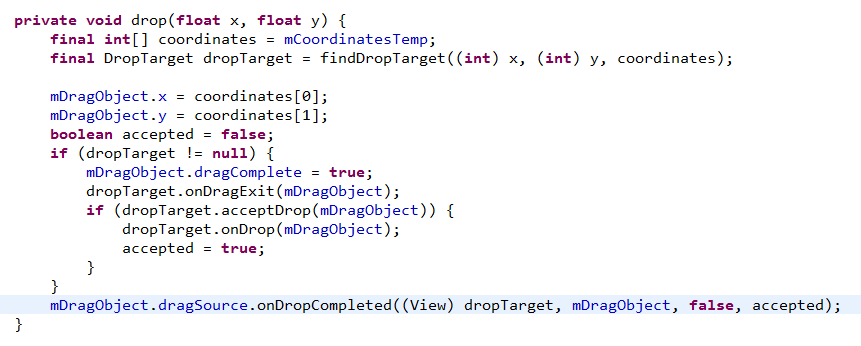
****

**这个函数首先将dragView的坐标移动到当前手指的位置，然后找到手指位于哪一个目标容器，最后调用函数checkScrollState(x, y)判断是否需要翻页。整体的逻辑上还是比较清晰的。**

**2.4 Drop（放手）**

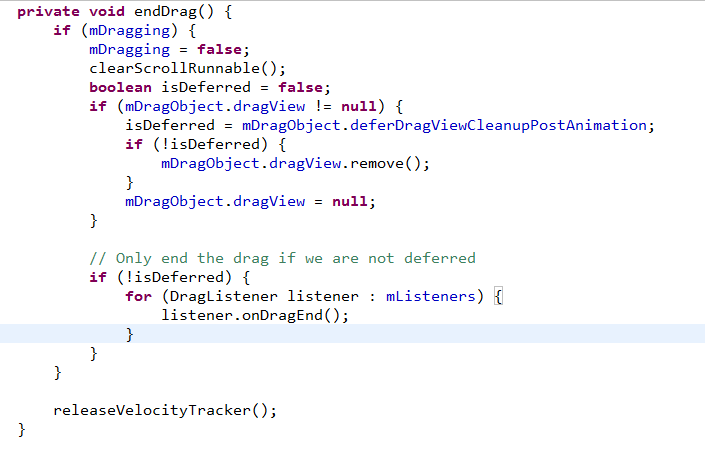
****

**这段代码最主要的是两个函数：drop和 endDrag，这两个函数的区别是前者是调用目标容器的drop事件，比如把DragView加入到目标容器中，目标容器处理退出拖拽事件。**

****

**endDrag函数主要是针对所有已经注册了拖拽事件的容器进行一次“广播”通知，比如垃圾箱要隐藏起来，设置拖拽状态为false 。**

**主要代码如下：**

****