|  |
| --- |
| COOEE |
| 文件夹文档 |
| Android文件夹概述 |
|  |
| **leexingwang** |
| **2014/11/27** |

|  |
| --- |
| 简要介绍文件夹的功能、数据库中的存储结构、布局以及事件执行流程 |

目录

[Android文件夹概述 2](#_Toc404870388)

[Android文件夹的主要功能 2](#_Toc404870389)

[较Turbar launcher中的文件夹功能相比功能上的缺陷 2](#_Toc404870390)

[Android文件夹的数据库存储结构 3](#_Toc404870391)

[表结构 3](#_Toc404870392)

[文件夹的保存数据 3](#_Toc404870393)

[Android文件夹布局 8](#_Toc404870394)

[FolderIcon的布局 8](#_Toc404870395)

[Folder的布局 10](#_Toc404870396)

[Android文件夹功能实现流程 14](#_Toc404870397)

[文件夹如何形成（FolderIcon） 14](#_Toc404870398)

[文件夹的启动 20](#_Toc404870399)

[文件夹中的事件执行流程 20](#_Toc404870400)

## Android文件夹概述

Android文件夹是Android系统在1.6以后加入的，其目的就是方便用户更好的管理自己的运用，对应用进行分类，快速进入应用。Android文件夹类主要有两个重要的类一个是folder.java一个是foldericon.java.

### Android文件夹的主要功能

1. 装载快捷方式；
2. 可以拖动到workspace，或者hotseat上面；
3. 可以创建多个文件夹装载不同的类型的快捷方式；
4. 文件中的快捷方式可以拖动到workspace或者hotseat上；
5. Workspace或者hotseat上的快捷方式可以拖动到文件夹里面；
6. 文件夹里的快捷方式可以任意拖动，并伴随着相应位置的更替。

### 较Turbar launcher中的文件夹功能相比功能上的缺陷

1. 此文件不能放在desktop里面；
2. 此文件夹不能一下放入多个快捷方式；
3. 此文件内部的快捷方式不能进行排序；
4. 在文件夹内部没有添加按钮进行快捷方式的添加；
5. 此文件夹最多只能放置16个快捷方式；
6. 此文件夹中的快捷方式不能进行左右滑动；
7. 此文件夹显示在workspace上面时是取了文件夹内部快捷方式的前三个垂直放置在文件夹背景上面没有原来文件夹简单明朗。

## Android文件夹的数据库存储结构

### 表结构

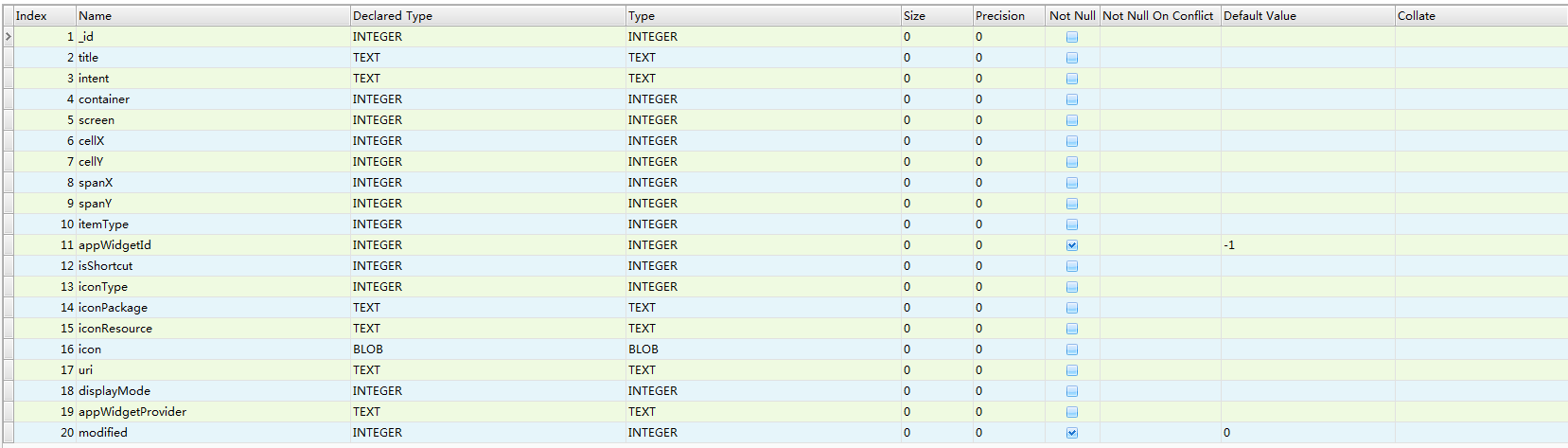
整个launcher数据库文件包括三个表

一个android\_metadat表，一个favorites表，一个workspaceScreens表

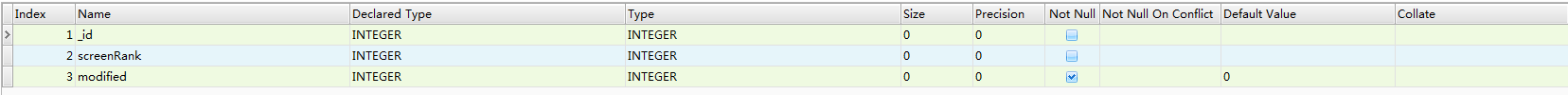
android\_metadat结构



favorites表



workspaceScreens表

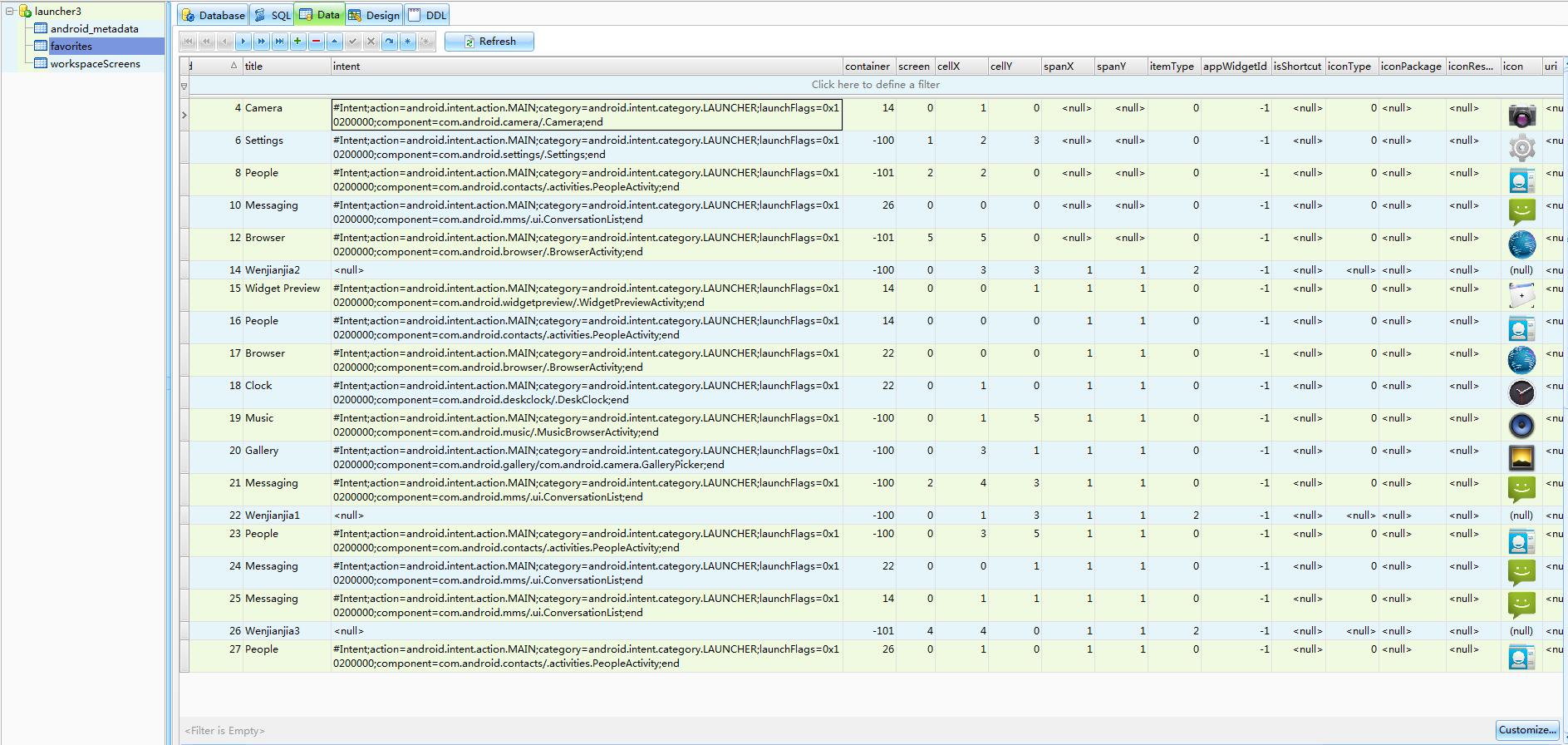


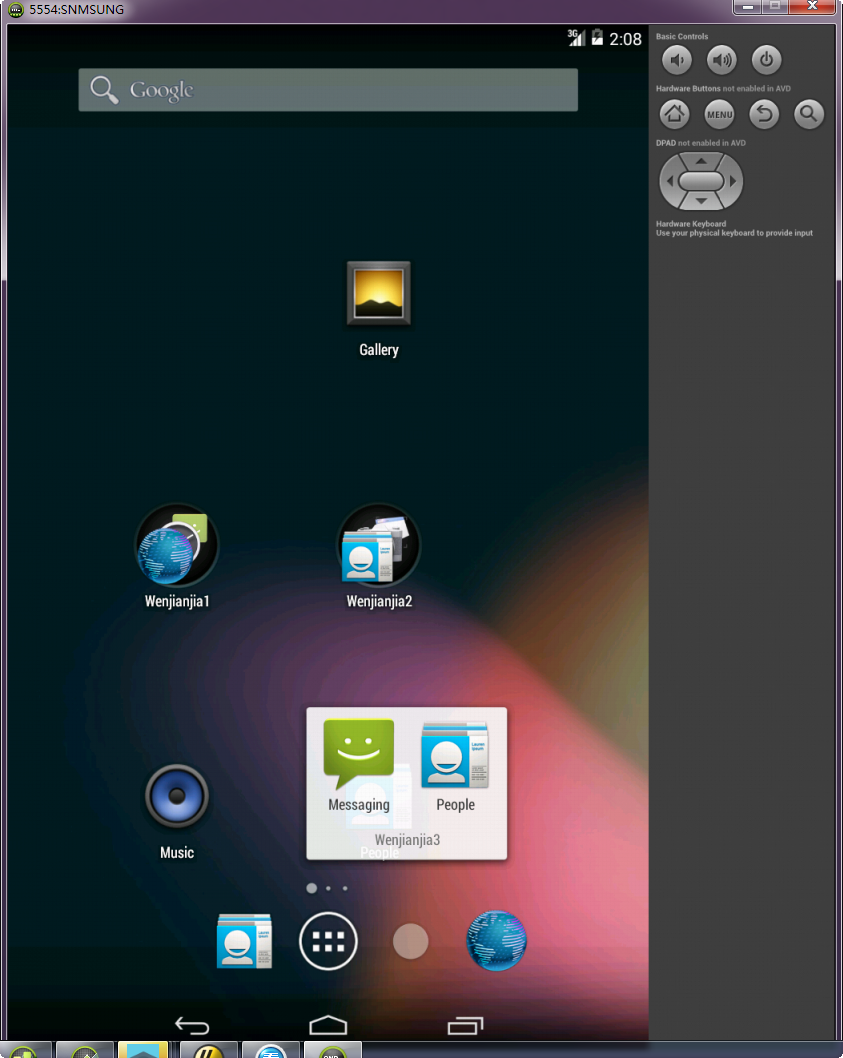
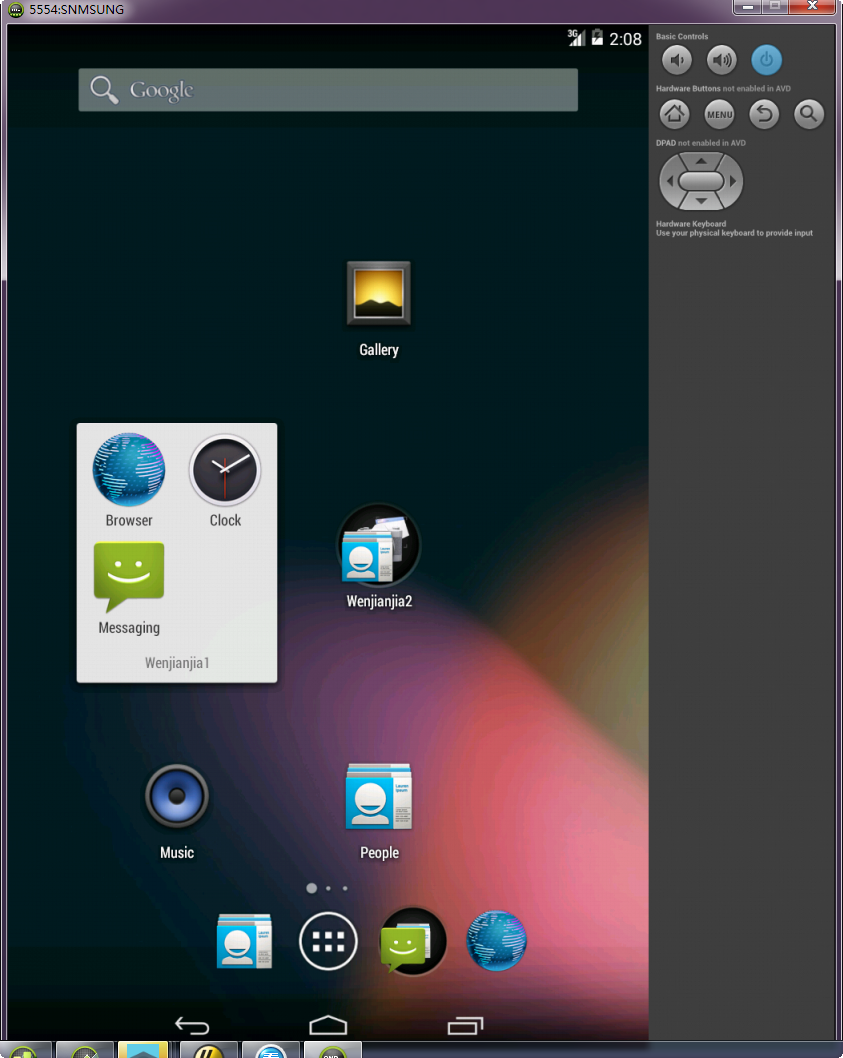
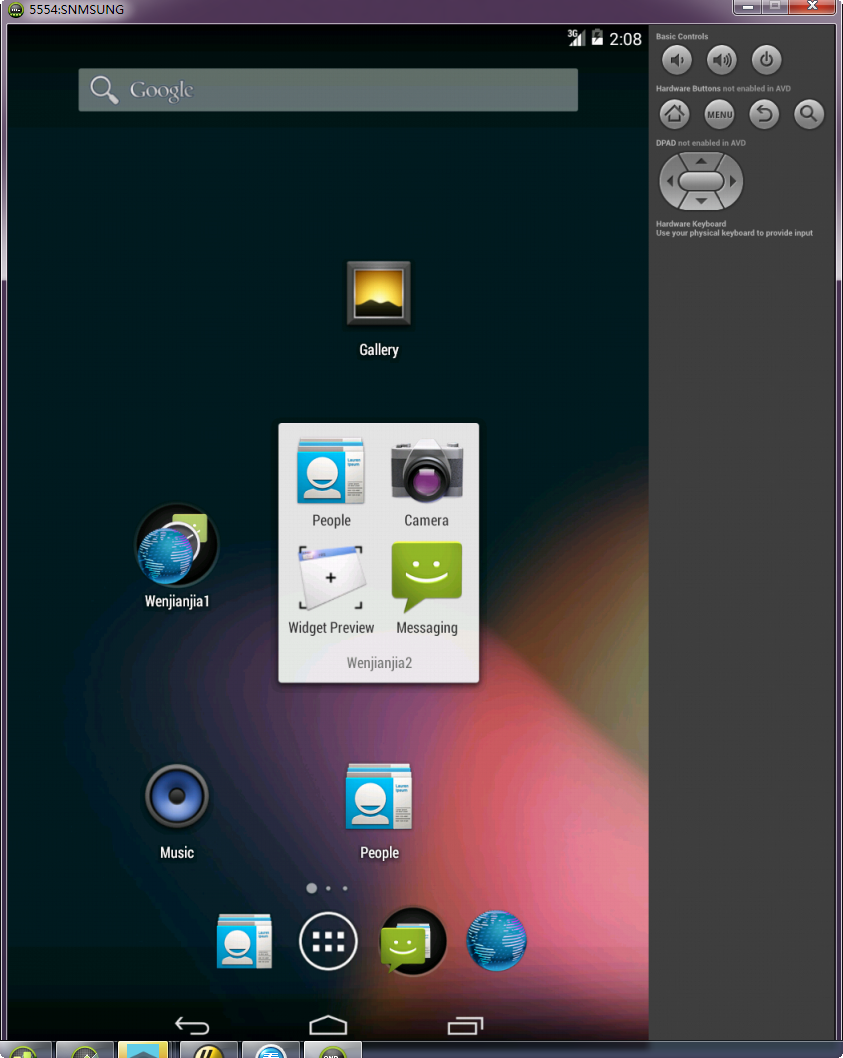
### 文件夹的保存数据

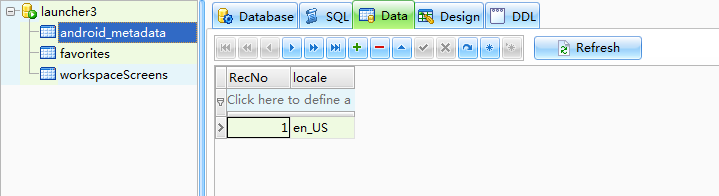
不管是文件夹还是快捷方式都是存在favorites表中的

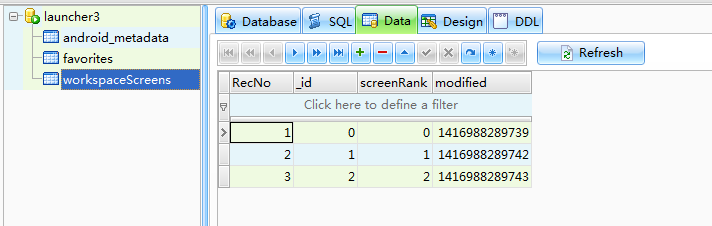
其中文件夹和文件夹中的快捷方式的关联

是通过文件夹的ID以及快捷方式的container关联的









## Android文件夹布局

### FolderIcon的布局

布局文件XML

<com.android.launcher3.FolderIcon

xmlns:android=*"http://schemas.android.com/apk/res/android"*

android:layout\_width=*"match\_parent"*

android:layout\_height=*"match\_parent"*

android:orientation=*"vertical"*

android:focusable=*"true"*

android:background=*"@drawable/focusable\_view\_bg"*>

<ImageView

android:id=*"@+id/preview\_background"*

android:layout\_gravity=*"center\_horizontal"*

android:layout\_width=*"wrap\_content"*

android:layout\_height=*"wrap\_content"*

android:antialias=*"true"*

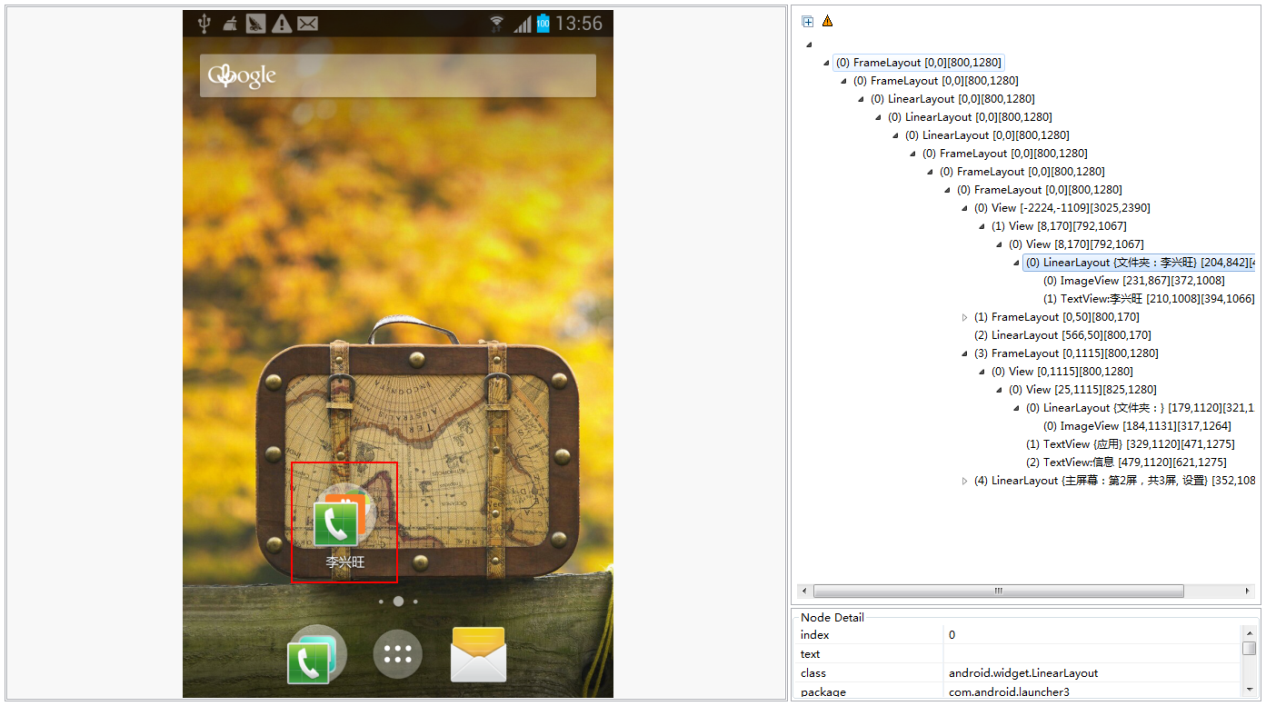
android:src=*"@drawable/portal\_ring\_inner\_holo"*/>

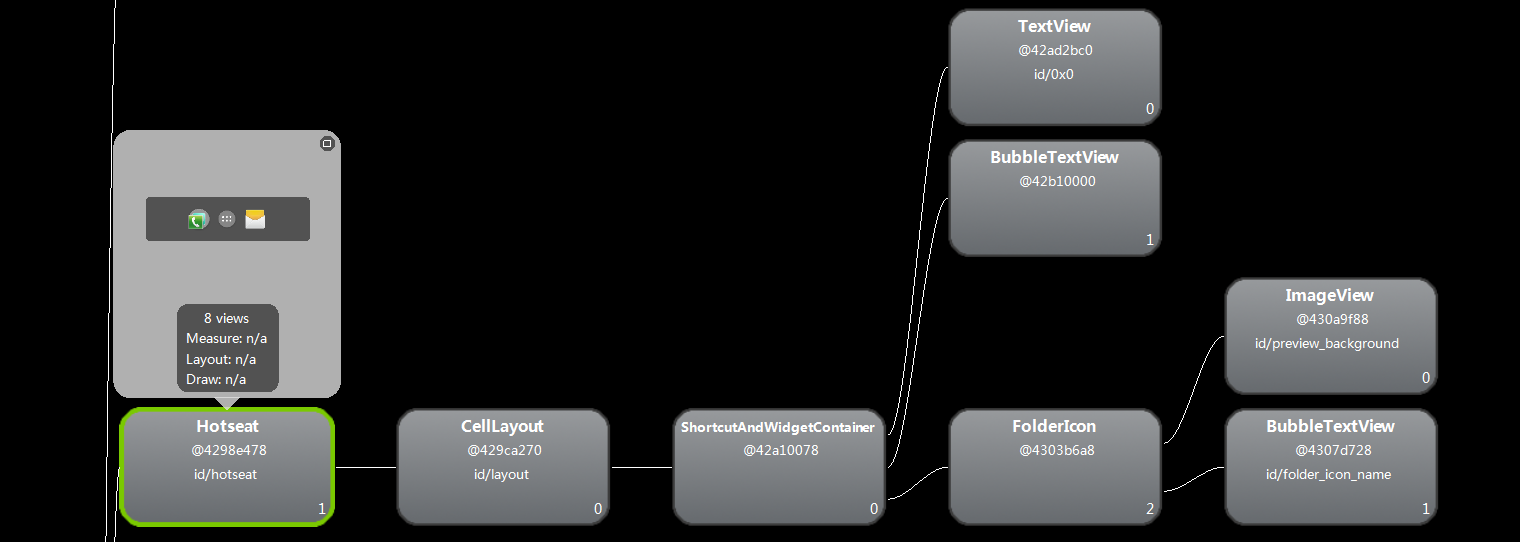
<com.android.launcher3.BubbleTextView

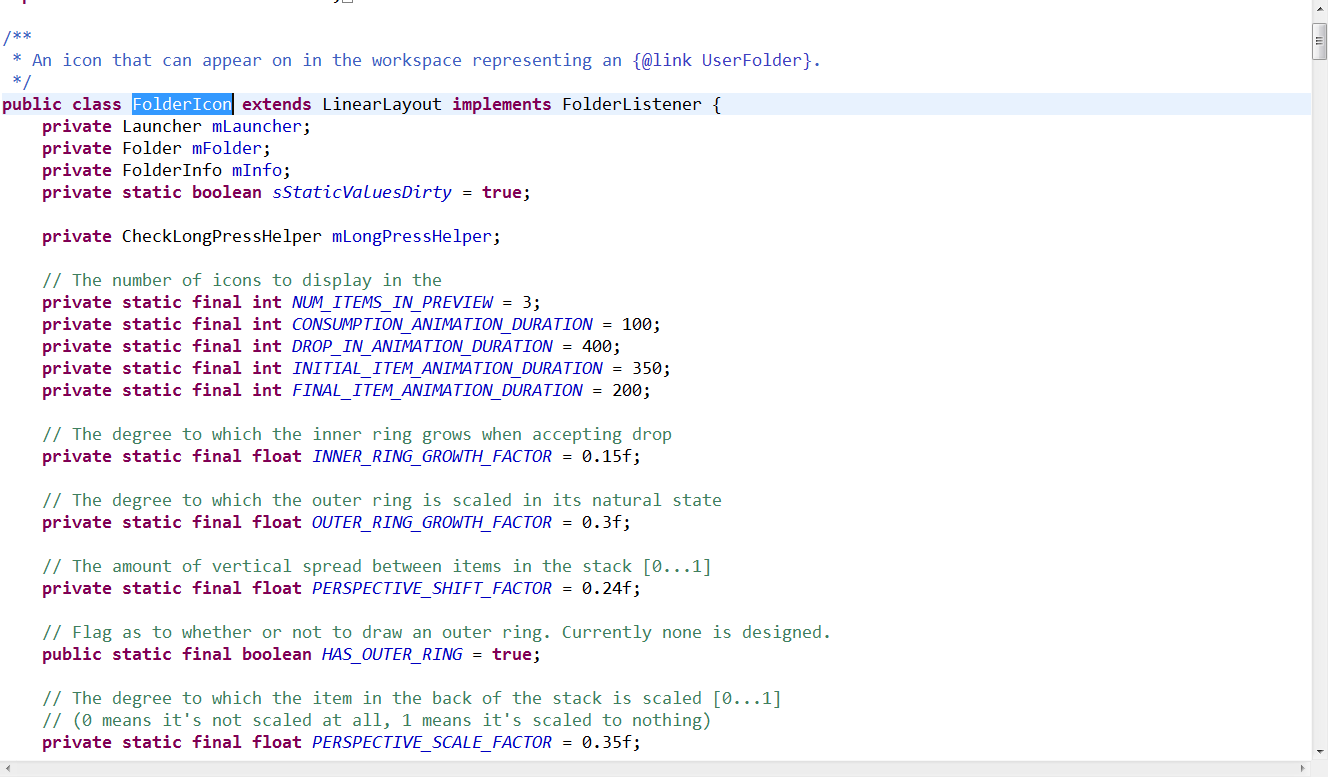
style=*"@style/WorkspaceIcon"*

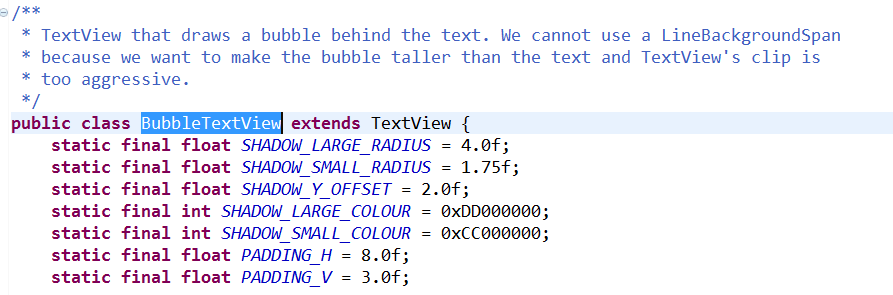
android:id=*"@+id/folder\_icon\_name"* />

</com.android.launcher3.FolderIcon>









### Folder的布局

<com.android.launcher3.Folder

xmlns:android=*"http://schemas.android.com/apk/res/android"*

xmlns:launcher=*"http://schemas.android.com/apk/res-auto/com.android.launcher3"*

android:layout\_width=*"wrap\_content"*

android:layout\_height=*"wrap\_content"*

android:orientation=*"vertical"*

android:background=*"@drawable/portal\_container\_holo"*>

<ScrollView

android:id=*"@+id/scroll\_view"*

android:layout\_width=*"match\_parent"*

android:layout\_height=*"match\_parent"*>

<com.android.launcher3.CellLayout

android:id=*"@+id/folder\_content"*

android:layout\_width=*"wrap\_content"*

android:layout\_height=*"wrap\_content"*

android:cacheColorHint=*"#ff333333"*

android:hapticFeedbackEnabled=*"false"* />

</ScrollView>

<com.android.launcher3.FolderEditText

android:id=*"@+id/folder\_name"*

android:layout\_width=*"match\_parent"*

android:layout\_height=*"wrap\_content"*

android:layout\_centerHorizontal=*"true"*

android:paddingTop=*"@dimen/folder\_name\_padding"*

android:paddingBottom=*"@dimen/folder\_name\_padding"*

android:background=*"#00000000"*

android:hint=*"@string/folder\_hint\_text"*

android:textSize=*"14sp"*

android:textColor=*"#ff777777"*

android:textColorHighlight=*"#ffCCCCCC"*

android:textCursorDrawable=*"@null"*

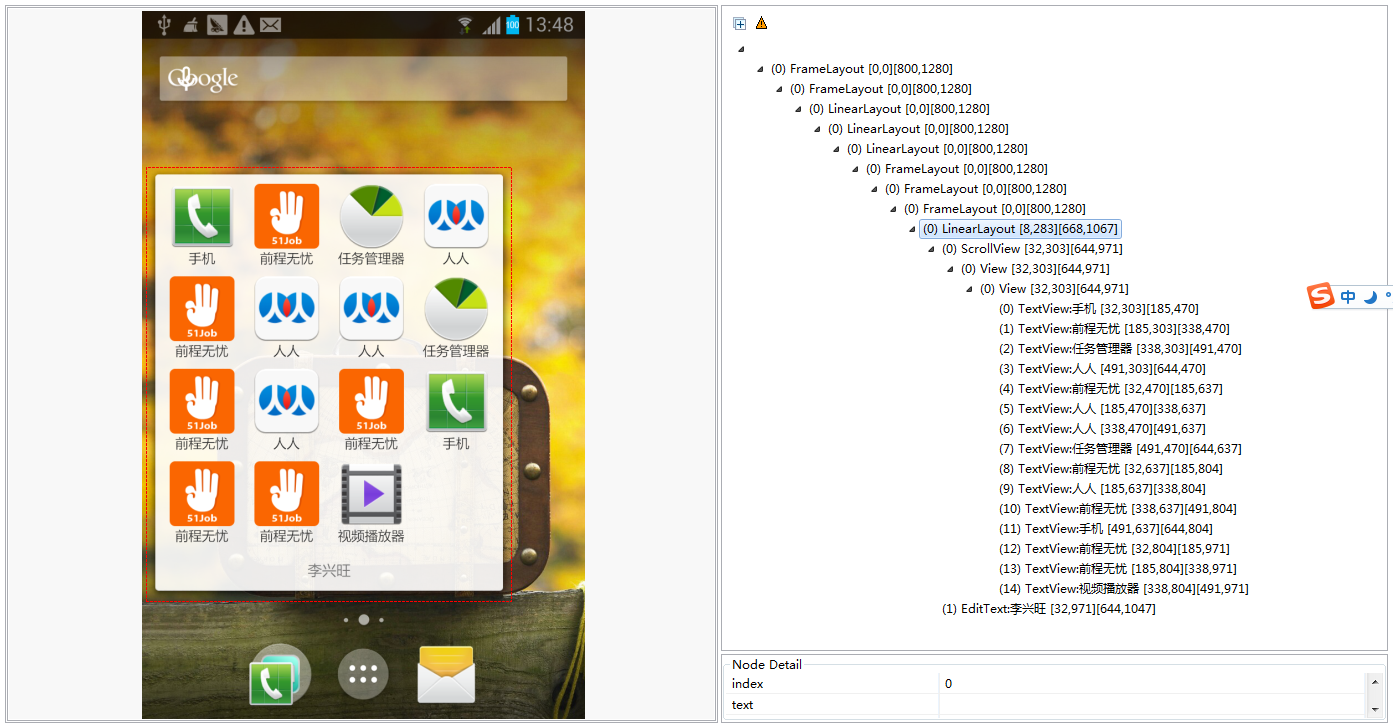
android:gravity=*"center\_horizontal"*

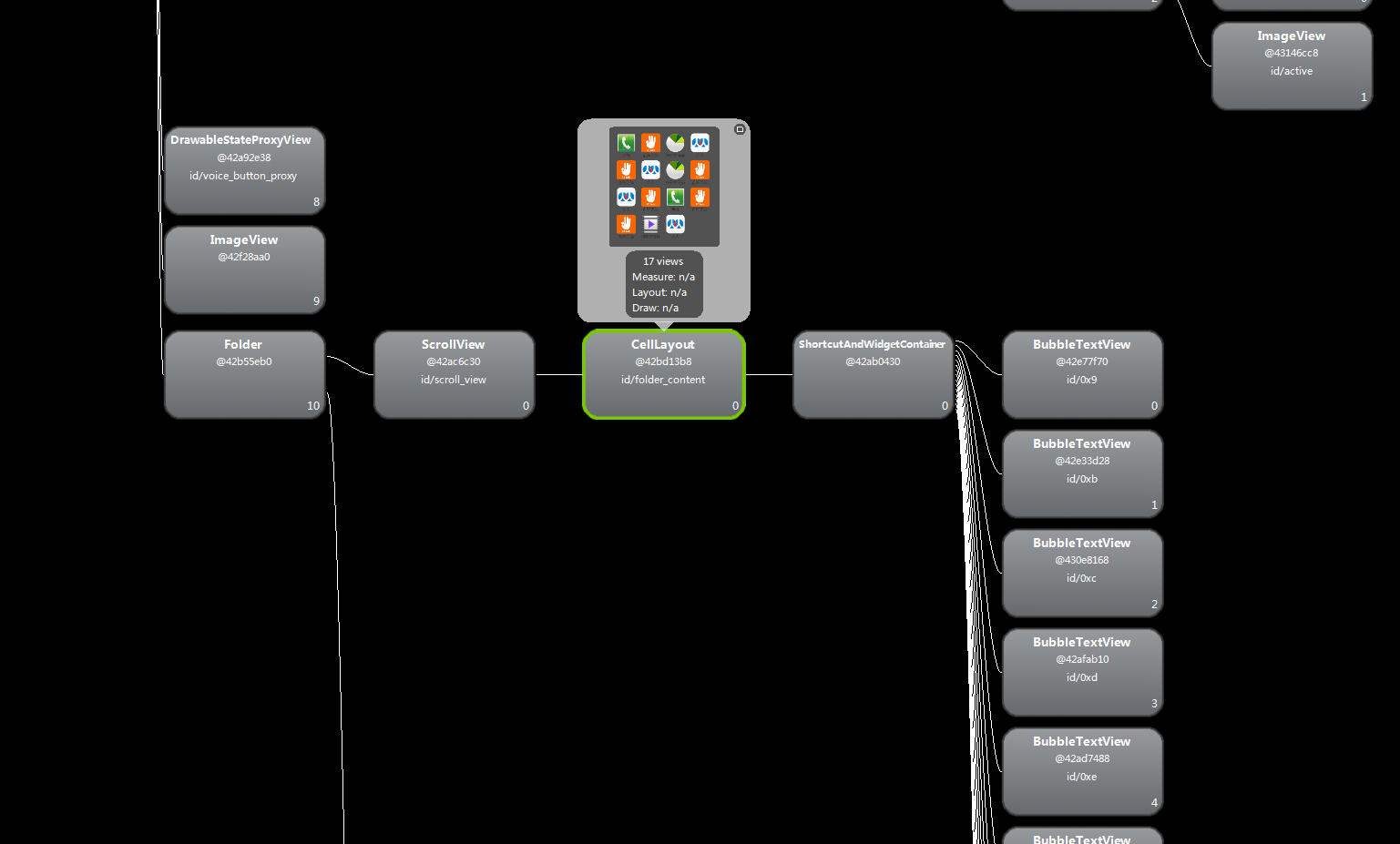
android:singleLine=*"true"*

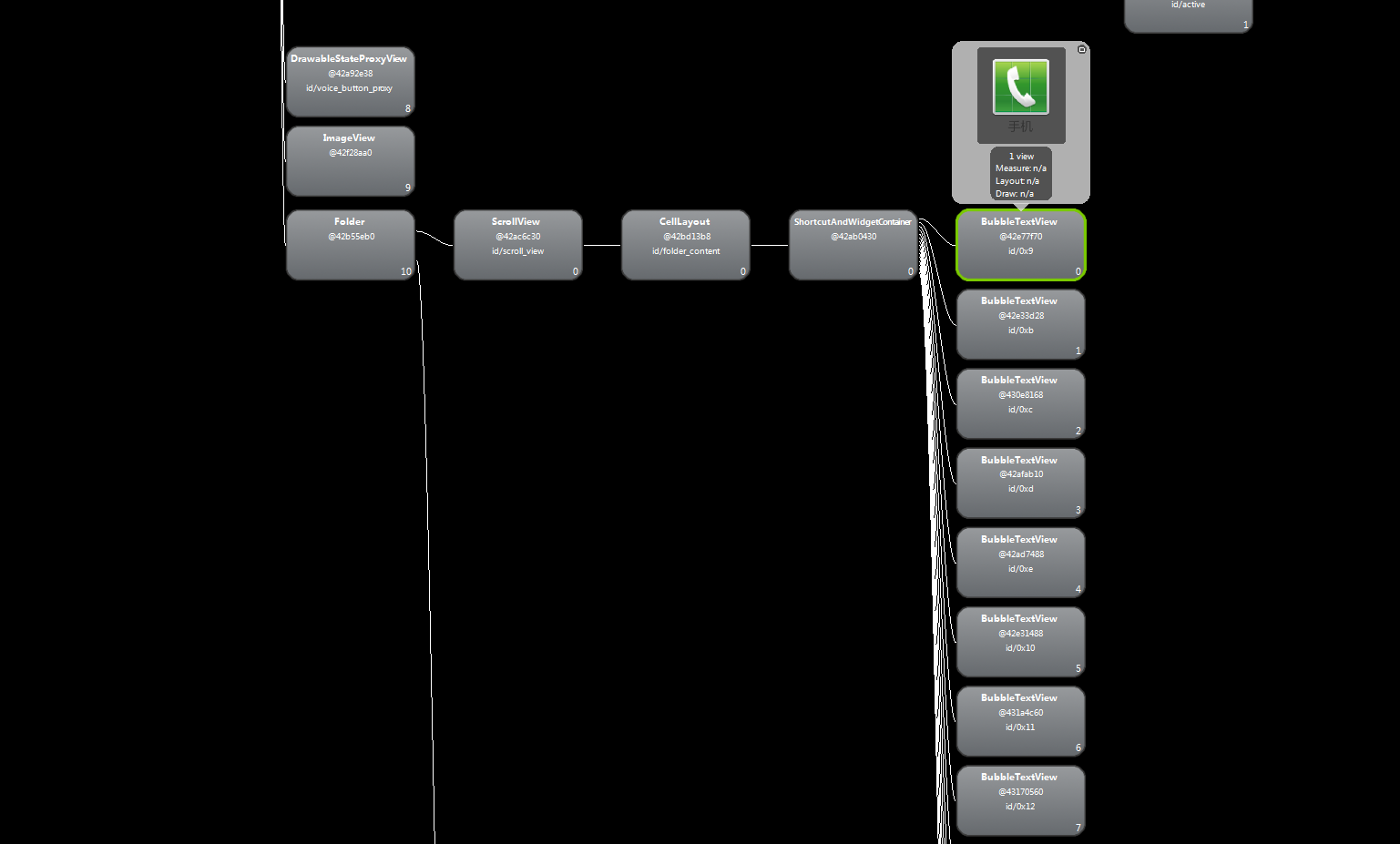
android:imeOptions=*"flagNoExtractUi"*

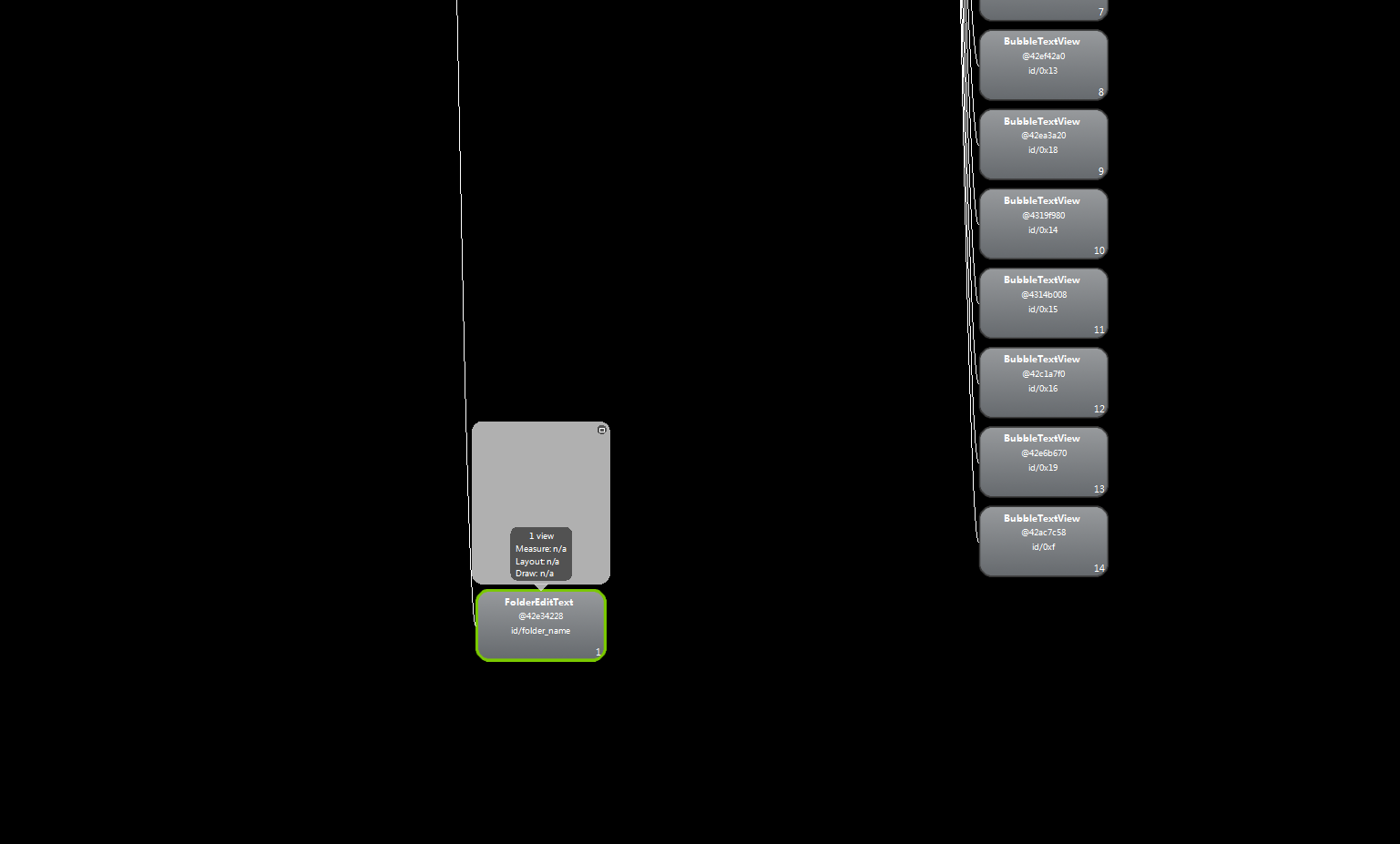
android:fontFamily=*"sans-serif-condensed"*/>

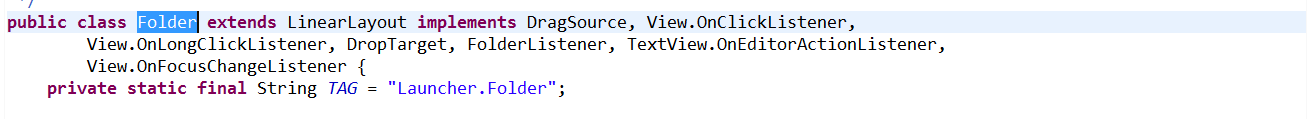
</com.android.launcher3.Folder>

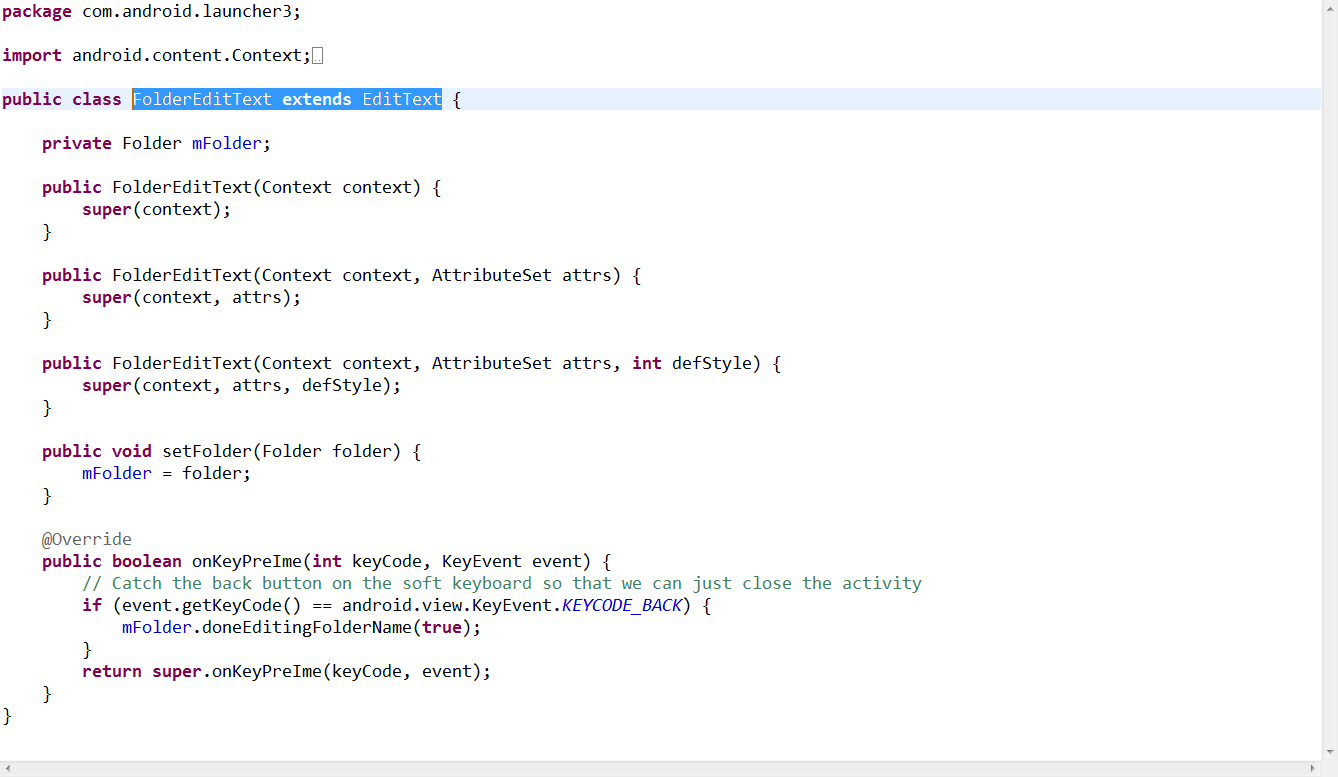












## Android文件夹功能实现流程

### 文件夹如何形成（FolderIcon）

1、在拖动桌面图标后，首先在workspace中的onDrop（）方法中会判断是否会形成一个fold。代码如下：

1. **public** **void** onDrop(DragObject d) {
3. ...  ...
5. // If the item being dropped is a shortcut and the nearest drop
6. //cell also contains a shortcut, then create a folder with the two shortcut
7. **if** (!mInScrollArea && createUserFolderIfNecessary(cell, container,
8. dropTargetLayout, mTargetCell, **false**, d.dragView, **null**)) {
9. **return**;
10. }
12. ... ...
13. }

2、在Workspace.java类的createUserFolderIfNecessary（）方法中来增加fold，具体代码如下：

1. **boolean** createUserFolderIfNecessary(View newView, **long** container, CellLayouttarget,  **int**[] targetCell, **boolean** external, DragView dragView, Runnable postAnimationRunnable) {
2. FolderIcon fi =
3. mLauncher.addFolder(target, container, screen, targetCell[0], targetCell[1]);
4. destInfo.cellX = -1;
5. destInfo.cellY = -1;
6. sourceInfo.cellX = -1;
7. sourceInfo.cellY = -1;
8. }

通过mLauncher.addFolder来传递folder的信息，包含一些位置信息绑定哪个屏幕的。

3、在Launcher.java类的addFolder（）这个方法是真正形成folder的，以及在launcher的数据库中插入一条信息，代码如下：

1. FolderIcon addFolder(CellLayout layout, **long** container, **final** **int** screen, **int** cellX,   **int** cellY) {
2. **final** FolderInfo folderInfo = **new** FolderInfo();
3. folderInfo.title = getText(R.string.folder\_name);
4. // Update the model
5. LauncherModel.addItemToDatabase(Launcher.**this**, folderInfo, container, screen, cellX, cellY,   **false**);
6. sFolders.put(folderInfo.id, folderInfo);
7. // Create the view
8. FolderIcon newFolder =   FolderIcon.fromXml(R.layout.folder\_icon, **this**, layout, folderInfo, mIconCache);
9. mWorkspace.addInScreen(newFolder, container, screen, cellX, cellY, 1, 1,   isWorkspaceLocked());
10. **return** newFolder;
11. }

FolderIcon.fromXml（）这个方法是从xml中形成folder，addInScreen（），把相应的信息插入数据库。

4、在FolderIcon.java中fromXml（）方法中的代码如下：

**static** FolderIcon fromXml(**int** resId, Launcher launcher, ViewGroup group,

1. FolderInfo folderInfo, IconCache iconCache) {
3. **if** (INITIAL\_ITEM\_ANIMATION\_DURATION >= DROP\_IN\_ANIMATION\_DURATION) {
4. **throw** **new** IllegalStateException("DROP\_IN\_ANIMATION\_DURATION must be greater than " +
5. "INITIAL\_ITEM\_ANIMATION\_DURATION, as sequencing of adding first two items " +
6. "is dependent on this");
7. }
9. FolderIcon icon = (FolderIcon) LayoutInflater.from(launcher).inflate(resId, group, **false**);
11. icon.mFolderName = (BubbleTextView) icon.findViewById(R.id.folder\_icon\_name);
12. icon.mFolderName.setText(folderInfo.title);
13. icon.mPreviewBackground = (ImageView) icon.findViewById(R.id.preview\_background);
15. icon.setTag(folderInfo);
16. icon.setOnClickListener(launcher);
17. icon.mInfo = folderInfo;
18. **icon.mLauncher = launcher;**
19. icon.setContentDescription(String.format(launcher.getString(R.string.folder\_name\_format),
20. folderInfo.title));
21. Folder folder = Folder.fromXml(launcher);
22. folder.setDragController(launcher.getDragController());
23. folder.setFolderIcon(icon);
24. folder.bind(folderInfo);
25. icon.mFolder = folder;
27. icon.mFolderRingAnimator = **new** FolderRingAnimator(launcher, icon);
28. folderInfo.addListener(icon);
30. **return** icon;
31. }

     Folder folder = Folder.fromXml(launcher);是真正产生了一个folder对象。代码如下：

1. /\*\*
2. \* Creates a new UserFolder, inflated from R.layout.user\_folder.
3. \*
4. \* @param context The application's context.
5. \*
6. \* @return A new UserFolder.
7. \*/
8. **static** Folder fromXml(Context context) {
9. **return** (Folder) LayoutInflater.from(context).inflate(R.layout.user\_folder, **null**);
10. }

并且给folder设置拖拽的控制器，绑定folderInfo设置folderInfo.addListener(icon)图标改变的监听。  
这个接口 interface  FolderListener定义了一个方法---->如下：

1. **interface** FolderListener {
2. **public** **void** onAdd(ShortcutInfo item);
3. **public** **void** onRemove(ShortcutInfo item);
4. **public** **void** onTitleChanged(CharSequence title);
5. **public** **void** onItemsChanged();
6. }

在folder.bind(folderInfo);方法中的操作如下：

1. **void** bind(FolderInfo info) {
2. mInfo = info;
3. ArrayList<ShortcutInfo> children = info.contents;
4. ArrayList<ShortcutInfo> overflow = **new** ArrayList<ShortcutInfo>();
5. setupContentForNumItems(children.size());
6. **int** count = 0;
7. **for** (**int** i = 0; i < children.size(); i++) {
8. ShortcutInfo child = (ShortcutInfo) children.get(i);
9. **if** (!createAndAddShortcut(child)) {
10. overflow.add(child);
11. } **else** {
12. count++;
13. }
14. }
16. // We rearrange the items in case there are any empty gaps
17. setupContentForNumItems(count);
19. // If our folder has too many items we prune them from the list. This is an issue
20. // when upgrading from the old Folders implementation which could contain an unlimited
21. // number of items.
22. **for** (ShortcutInfo item: overflow) {
23. mInfo.remove(item);
24. LauncherModel.deleteItemFromDatabase(mLauncher, item);
25. }
27. mItemsInvalidated = **true**;
28. updateTextViewFocus();
29. mInfo.addListener(**this**);
31. **if** (!sDefaultFolderName.contentEquals(mInfo.title)) {
32. mFolderName.setText(mInfo.title);
33. } **else** {
34. mFolderName.setText("");
35. }
36. }

主要的操作是：给拖拽进来的快捷方式安排位置，判断Folder是否已经放满，设置监听，设置folder的Name；

5、folder桌面的缩略图怎么形成的，是在第2步Workspace.java的createUserFolderIfNecessary（）方法中

1. // If the dragView is null, we can't animate
2. **boolean** animate = dragView != **null**;
3. **if** (animate) {
4. fi.performCreateAnimation(destInfo, v, sourceInfo, dragView, folderLocation, scale,    postAnimationRunnable);
5. } **else** {
6. fi.addItem(destInfo);
7. fi.addItem(sourceInfo);
8. }

fi.performCreateAnimation（）这个方法是给folder添加个动画。

6、在FolderIcon.java中的performCreateAnimation（）方法中：

1. **public** **void** performCreateAnimation(**final** ShortcutInfo destInfo, **final** View destView,
2. **final** ShortcutInfo srcInfo, **final** View srcView, Rect dstRect,
3. **float** scaleRelativeToDragLayer, Runnable postAnimationRunnable) {
5. Drawable animateDrawable = ((TextView) destView).getCompoundDrawables()[1];
6. computePreviewDrawingParams(animateDrawable.getIntrinsicWidth(), destView.getMeasuredWidth());
8. // This will animate the dragView (srcView) into the new folder
9. onDrop(srcInfo, srcView, dstRect, scaleRelativeToDragLayer, 1, postAnimationRunnable);
11. // This will animate the first item from it's position as an icon into its
12. // position as the first item in the preview
13. animateFirstItem(animateDrawable, INITIAL\_ITEM\_ANIMATION\_DURATION);
15. postDelayed(**new** Runnable() {
16. **public** **void** run() {
17. addItem(destInfo);
18. }
19. }, INITIAL\_ITEM\_ANIMATION\_DURATION);
20. }

computePreviewDrawingParams（）这个方法是计算绘制folder图标的方法；

7、在FolderIcon.java类中的computePreviewItemDrawingParams（）方法中:

1. **private** PreviewItemDrawingParams computePreviewItemDrawingParams(**int** index,
2. PreviewItemDrawingParams params) {
3. index = NUM\_ITEMS\_IN\_PREVIEW - index - 1;
4. **float** r = (index \* 1.0f) / (NUM\_ITEMS\_IN\_PREVIEW - 1);
5. **float** scale = (1 - PERSPECTIVE\_SCALE\_FACTOR \* (1 - r));
6. **float** offset = (1 - r) \* mMaxPerspectiveShift;
7. **float** scaledSize = scale \* mBaselineIconSize;
8. **float** scaleOffsetCorrection = (1 - scale) \* mBaselineIconSize;
9. // We want to imagine our coordinates from the bottom left, growing up and to the
10. // right. This is natural for the x-axis, but for the y-axis, we have to invert things.
11. **float** transY = mAvailableSpaceInPreview - (offset + scaledSize + scaleOffsetCorrection);
12. **float** transX = offset + scaleOffsetCorrection;
13. **float** totalScale = mBaselineIconScale \* scale;
14. **final** **int** overlayAlpha = (**int**) (80 \* (1 - r));
16. **if** (params == **null**) {
17. params = **new** PreviewItemDrawingParams(transX, transY, totalScale, overlayAlpha);
18. } **else** {
19. params.transX = transX;
20. params.transY = transY;
21. params.scale = totalScale;
22. params.overlayAlpha = overlayAlpha;
23. }
24. **return** params;
25. }

主要工作是：计算图标的排列，每一个相对上一个有点偏移的距离；效果图如下：  
                                                    
以上基本是文件夹产生流程。

### 文件夹的启动

如果系统一开始有fold，一启动launcher的时候，在Launcher.java类中bindFolders回调方法中：

1. /\*\*
2. \* Implementation of the method from LauncherModel.Callbacks.
3. \*/
4. **public** **void** bindFolders(HashMap<Long, FolderInfo> folders) {
5. setLoadOnResume();
6. sFolders.clear();
7. sFolders.putAll(folders);
8. }

绑定所有fold的对象交给sFolders，去处理。  
private static HashMap<Long, FolderInfo> sFolders = new HashMap<Long, FolderInfo>();

### 文件夹中的事件执行流程

#### Folder类中点击事件执行流程

因为Folder.java类是继承了View.OnClickListener点击事件由自己的onCLick（）处理

**public** **void** onClick(View v) {

Object tag = v.getTag();

**if** (tag **instanceof** ShortcutInfo) {

mLauncher.onClick(v);

}

}

其中调用了mLauncher.onClick(v)即Launcher中的onCLick（）方法处理。

**public** **void** onClick(View v) {

// Make sure that rogue clicks don't get through while allapps is launching, or after the

// view has detached (it's possible for this to happen if the view is removed mid touch).

。。。。。。。。。

**if** (tag **instanceof** ShortcutInfo) {

// Open shortcut

**final** ShortcutInfo shortcut = (ShortcutInfo) tag;

**final** Intent intent = shortcut.intent;

// Check for special shortcuts

**if** (intent.getComponent() != **null**) {

**final** String shortcutClass = intent.getComponent().getClassName();

**if** (shortcutClass.equals(WidgetAdder.**class**.getName())) {

showAllApps(**true**, AppsCustomizePagedView.ContentType.*Widgets*, **true**);

**return**;

} **else** **if** (shortcutClass.equals(MemoryDumpActivity.**class**.getName())) {

MemoryDumpActivity.*startDump*(**this**);

**return**;

} **else** **if** (shortcutClass.equals(ToggleWeightWatcher.**class**.getName())) {

toggleShowWeightWatcher();

**return**;

}

}

// Start activities

**int**[] pos = **new** **int**[2];

v.getLocationOnScreen(pos);

intent.setSourceBounds(**new** Rect(pos[0], pos[1],

pos[0] + v.getWidth(), pos[1] + v.getHeight()));

**boolean success = startActivitySafely(v, intent, tag);**

。。。。。。。。

}

}

#### Folder类中长按事件执行流程

1、因为Folder.java类是继承了View. OnLongClickListener点击事件由自己的OnLongClickListener（）处理

**public** **boolean** onLongClick(View v) {

// Return if global dragging is not enabled

**if** (!mLauncher.isDraggingEnabled()) **return** **true**;

Object tag = v.getTag();

**if** (tag **instanceof** ShortcutInfo) {

ShortcutInfo item = (ShortcutInfo) tag;

**if** (!v.isInTouchMode()) {

**return** **false**;

}

mLauncher.dismissFolderCling(**null**);

mLauncher.getWorkspace().onDragStartedWithItem(v);

mLauncher.getWorkspace().beginDragShared(v, **this**);

mIconDrawable = ((TextView) v).getCompoundDrawables()[1];

mCurrentDragInfo = item;

mEmptyCell[0] = item.cellX;

mEmptyCell[1] = item.cellY;

mCurrentDragView = v;

mContent.removeView(mCurrentDragView);

mInfo.remove(mCurrentDragInfo);

mDragInProgress = **true**;

mItemAddedBackToSelfViaIcon = **false**;

}

**return** **true**;

}

OnLongClickListener（）中主要是保存拖拽前的一些状态位置并且从mInfo中移除了这个快捷方式的信息mInfo.remove(mCurrentDragInfo);同样拖拽事件是交给Workspace来处理，最后也是统一交给DragController.java类处理和分发相应的事件。

2、后执行onDragEnter(DragObject d)方法

**public** **void** onDragEnter(DragObject d) {

mPreviousTargetCell[0] = -1;

mPreviousTargetCell[1] = -1;

mOnExitAlarm.cancelAlarm();

}

3、后执行Folder.java类中的onDragOver（）这个方法，当在文件夹中拖拽到另一个快捷方式的上面的时候，发生交换。

来看代码如下：

1. **public** **void** onDragOver(DragObject d) {
2. **float**[] r = getDragViewVisualCenter(d.x, d.y, d.xOffset, d.yOffset, d.dragView, **null**);
3. mTargetCell = mContent.findNearestArea((**int**) r[0], (**int**) r[1], 1, 1, mTargetCell);
4. **if** (mTargetCell[0] != mPreviousTargetCell[0] || mTargetCell[1] != mPreviousTargetCell[1]) {
5. mReorderAlarm.cancelAlarm();
6. mReorderAlarm.setOnAlarmListener(mReorderAlarmListener);
7. mReorderAlarm.setAlarm(150);
8. mPreviousTargetCell[0] = mTargetCell[0];
9. mPreviousTargetCell[1] = mTargetCell[1];
10. }
11. }

这个方法主要做的操作是：判断拖拽的是哪个对象mContent.findNearestArea((int) r[0], (int) r[1], 1, 1, mTargetCell);判断在哪个目标的附近，然后判断和是否是正在拖拽的对象的坐标，设置mReorderAlarmListener来进行交换，设置150毫秒用来处理动画的。

4、在ReorderAlarmListener内部类的代码如下：

1. OnAlarmListener mReorderAlarmListener = **new** OnAlarmListener() {
2. **public** **void** onAlarm(Alarm alarm) {
3. realTimeReorder(mEmptyCell, mTargetCell);
4. }
5. };

5、在realTimeReorder（）方法中传递这个快捷方式在屏幕x轴，y轴上的位置，进行交换,代码如下：

1. **private** **void** realTimeReorder(**int**[] empty, **int**[] target) {
2. **boolean** wrap;
3. **int** startX;
4. **int** endX;
5. **int** startY;
6. **int** delay = 0;
7. **float** delayAmount = 30;
8. **if** (readingOrderGreaterThan(target, empty)) {
9. wrap = empty[0] >= mContent.getCountX() - 1;
10. startY = wrap ? empty[1] + 1 : empty[1];
11. **for** (**int** y = startY; y <= target[1]; y++) {
12. startX = y == empty[1] ? empty[0] + 1 : 0;
13. endX = y < target[1] ? mContent.getCountX() - 1 : target[0]
14. **for** (**int** x = startX; x <= endX; x++) {
15. View v = mContent.getChildAt(x,y);
16. **if** (mContent.animateChildToPosition(v, empty[0], empty[1],   REORDER\_ANIMATION\_DURATION, delay)) {
17. empty[0] = x;
18. empty[1] = y;
19. delay += delayAmount;
20. delayAmount \*= 0.9;
21. }
22. }
23. }
24. } **else** {
25. wrap = empty[0] == 0;
26. startY = wrap ? empty[1] - 1 : empty[1];
27. **for** (**int** y = startY; y >= target[1]; y--) {
28. startX = y == empty[1] ? empty[0] - 1 : mContent.getCountX() - 1;
29. endX = y > target[1] ? 0 : target[0];
30. **for** (**int** x = startX; x >= endX; x--) {
31. View v = mContent.getChildAt(x,y);
32. **if** (mContent.animateChildToPosition(v, empty[0], empty[1],   REORDER\_ANIMATION\_DURATION, delay)) {
33. empty[0] = x;
34. empty[1] = y;
35. delay += delayAmount;
36. delayAmount \*= 0.9;
37. }
38. }
39. }
40. }
41. }

readingOrderGreaterThan（）这个方法的作用是判断是从上往下拖动，还是从下往上拖动，这两种情况的交换方式不一样。循环也就不一样。交换的过程中通过animateChildToPosition（）；这个方法设置了一个动画。

 6、然后执行onDragExit (DragObject d)

**public** **void** onDragExit(DragObject d) {

// Exiting folder; stop the auto scroller.

mAutoScrollHelper.setEnabled(**false**);

// We only close the folder if this is a true drag exit, ie. not because

// a drop has occurred above the folder.

**if** (!d.dragComplete) {

mOnExitAlarm.setOnAlarmListener(mOnExitAlarmListener);

mOnExitAlarm.setAlarm(*ON\_EXIT\_CLOSE\_DELAY*);

}

mReorderAlarm.cancelAlarm();

mDragMode = DRAG\_MODE\_NONE;

}

这个方法的主要作用既是停止循环滚动的动画

7、在放下的时候会触发Folder.java的onDrop（）方法，

**public** **void** onDrop(DragObject d) {

// Dragged from self onto self, currently this is the only path possible, however

// we keep this as a distinct code path.

**if** (item == mCurrentDragInfo) {

ShortcutInfo si = (ShortcutInfo) mCurrentDragView.getTag();

CellLayout.LayoutParams lp = (CellLayout.LayoutParams) mCurrentDragView.getLayoutParams();

si.cellX = lp.cellX = mEmptyCell[0];

si.cellX = lp.cellY = mEmptyCell[1];

mContent.addViewToCellLayout(mCurrentDragView, -1, (**int**)item.id, lp, **true**);

**if** (d.dragView.hasDrawn()) {

mLauncher.getDragLayer().animateViewIntoPosition(d.dragView, mCurrentDragView);

} **else** {

d.deferDragViewCleanupPostAnimation = **false**;

mCurrentDragView.setVisibility(*VISIBLE*);

}

mItemsInvalidated = **true**;

setupContentDimensions(getItemCount());

mSuppressOnAdd = **true**;

}

**mInfo.add(item);**

}

作用是设置放下的view可见，把当前的这个快捷方式添加到mInfo中。

8、当执行完onDrop（）方法后会走onDropCompleted（）方法:

1. **public** **void** onDropCompleted(View target, DragObject d, **boolean** success) {
2. ... ...
3. // Reordering may have occured, and we need to save the new item locations. We do this once
4. // at the end to prevent unnecessary database operations.
5. updateItemLocationsInDatabase();
6. ...  ...
7. }

这个方法的作用是更新item的位置信息在数据库中。

1. **private** **void** updateItemLocationsInDatabase() {
2. ArrayList<View> list = getItemsInReadingOrder();
3. **for** (**int** i = 0; i < list.size(); i++) {
4. View v = list.get(i);
5. ItemInfo info = (ItemInfo) v.getTag();
6. LauncherModel.moveItemInDatabase(mLauncher, info, mInfo.id, 0,
7. info.cellX, info.cellY);
8. }
9. }