一、什么是 Launcher

1.1 概念

Launcher是开机完成后第一个启动的应用，俗称“HomeScreen”，也是我们开机后看到的第一个App，用来展示应用列表和快捷方式、小部件等。

Launcher本质上与其他Android应用一样，都是apk应用程序，可以独立安装运行，我们平常使用的系统Launcher都是手机厂商定制后预制到系统里面的。

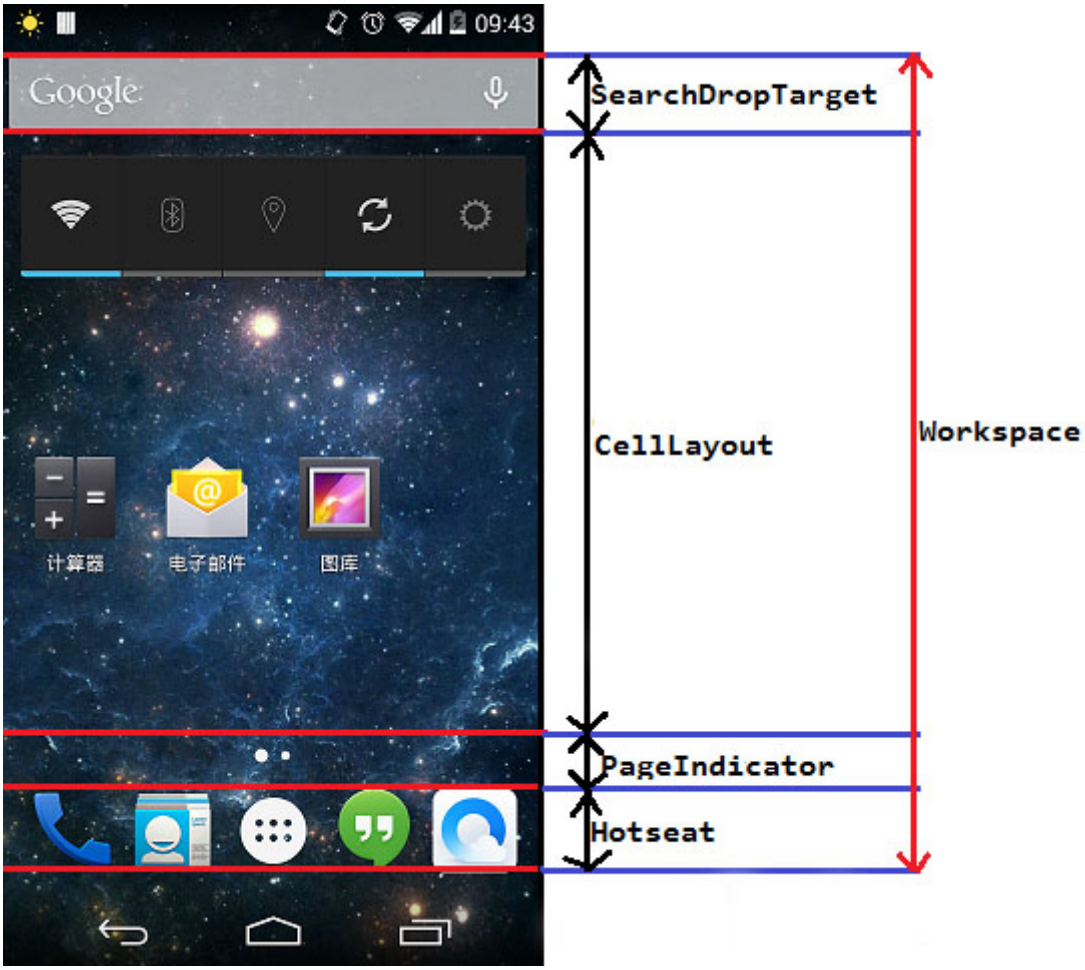
1. 为什么按Home健就会启动Launcher，Launcher有什么不同?
2. 为什么桌面应用列表不显示Launcher自身图标?
   看一下Launcher的主Activity配置

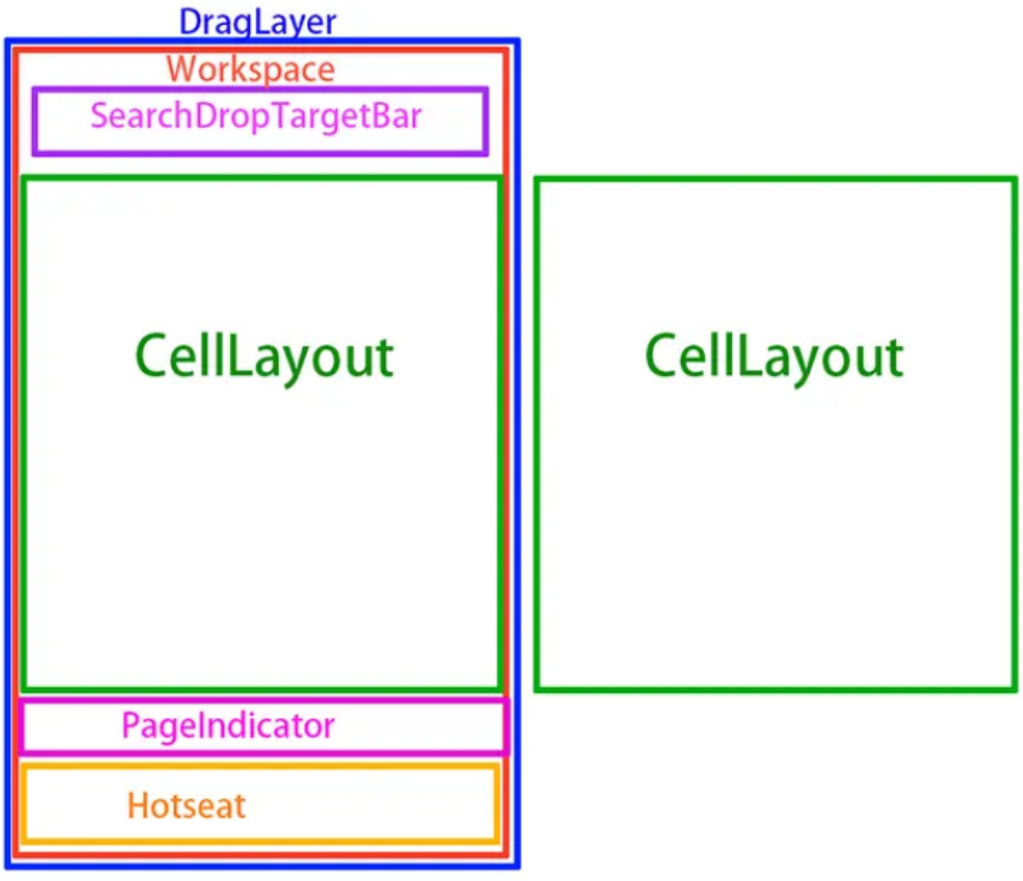
```
<activity
            android:name="com.android.launcher3.Launcher"
            android:launchMode="singleTask"
            android:clearTaskOnLaunch="true"
            android:stateNotNeeded="true"
            android:windowSoftInputMode="adjustPan"
            android:screenOrientation="unspecified"

android:configChanges="keyboard|keyboardHidden|mcc|mnc|navigation|orientation|sc
reenSize
            |screenLayout|smallestScreenSize"
            android:resizeableActivity="true"
            android:resumeWhilePausing="true"
            android:taskAffinity=""
            android:enabled="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.HOME" />
                <category android:name="android.intent.category.DEFAULT" />
                <category android:name="android.intent.category.MONKEY"/>
                <category android:name="android.intent.category.LAUNCHER_APP" />
            </intent-filter>
        </activity>
```

这里通过设置和 来告诉系统这是一个Launcher应用，只要你的应用主Activity配置里加上这两行，就可以成为Launcher应用了。

如果你希望你的应用图标显示在桌面应用列表中，只要保证intent-filter中同时存在和就可以了

1.2 Launcher 页面结构



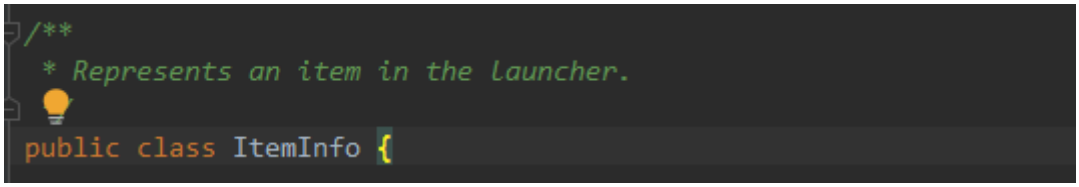


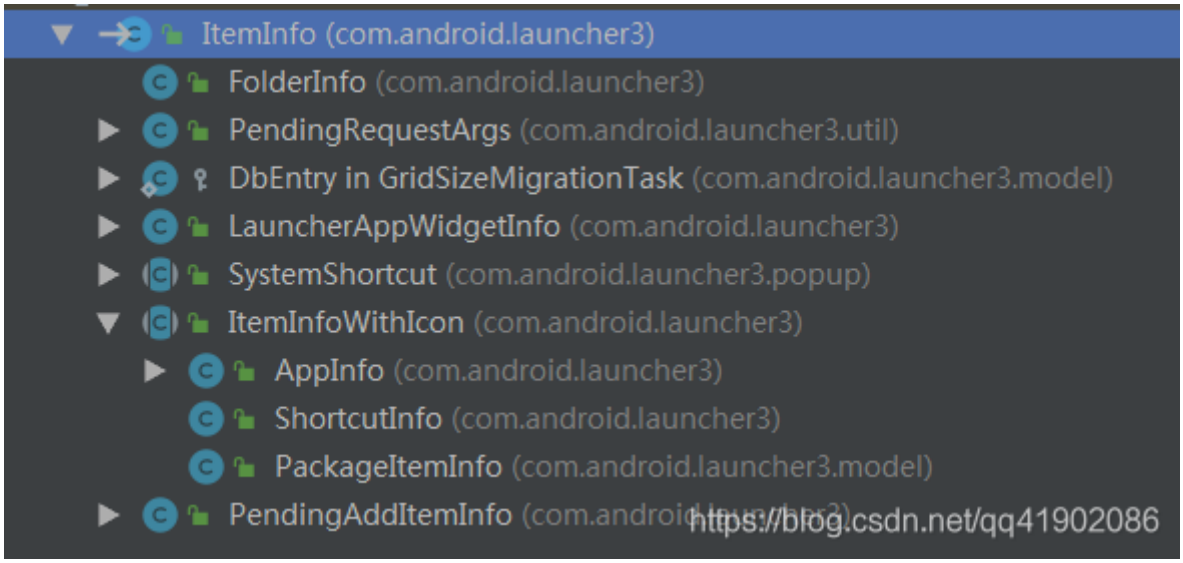
Launcher 界面主要由一下5个部分构成：

- drag_layer 监听拖拽层，所有其他控件都包裹在drag_layer内部
- workspace 也就是我们通俗所说的桌面，显示应用的容器，但超过一屏时，可以左右滑动（类似于ViewPager）
- CellLayout workspace 的每一屏就是一个 CellLayout（类似于ViewPage的每个Item）
- page_indicator 滑动页面指示器，用来提示当前在哪一页，国内厂商基本显示的圆点，原生显示的横线
- HotSeat 底部固定的一排应用，不跟随屏幕左右滑动
- drop_target_bar 长按拖动图标时，屏幕顶端的提示“移除”或“卸载”
- BubbleTextView 每一个应用图标加上应用名称，整体是一个BubbleTextView

1.3 桌面图标类型

ItemInfo 是所有所有类型的父类，看一下源码对ItemInfo的解释：





继承自ItemInfo，衍生出以下类型：

```
     * {@link LauncherSettings.Favorites#ITEM_TYPE_APPLICATION},
     * {@link LauncherSettings.Favorites#ITEM_TYPE_SHORTCUT},
     * {@link LauncherSettings.Favorites#ITEM_TYPE_DEEP_SHORTCUT}
     * {@link LauncherSettings.Favorites#ITEM_TYPE_FOLDER},
     * {@link LauncherSettings.Favorites#ITEM_TYPE_APPWIDGET} or
     * {@link LauncherSettings.Favorites#ITEM_TYPE_CUSTOM_APPWIDGET}.
```

- ppInfo: 对应于ITEM_TYPE_APPLICATION，表示一个应用
- ShortcutInfo: 对应于ITEM_TYPE_SHORTCUT，表示一个快捷方式，Android O以后新增一种shortcut，就是桌面长按图标弹出的每一个Item，还可以长按拖动到桌面，形成一个快捷方式，也就对应着ITEM_TYPE_DEEP_SHORTCUT
- FolderInfo: 对应于ITEM_TYPE_FOLDER，表示一个由多个应用图标组成的文件夹

- LauncherAppWidgetInfo: 对应于ITEM_TYPE_APPWIDGET，表示一些桌面小部件，类似于时钟，天气等

二、Launcher 数据加载流程

2.1 需要用的相关知识

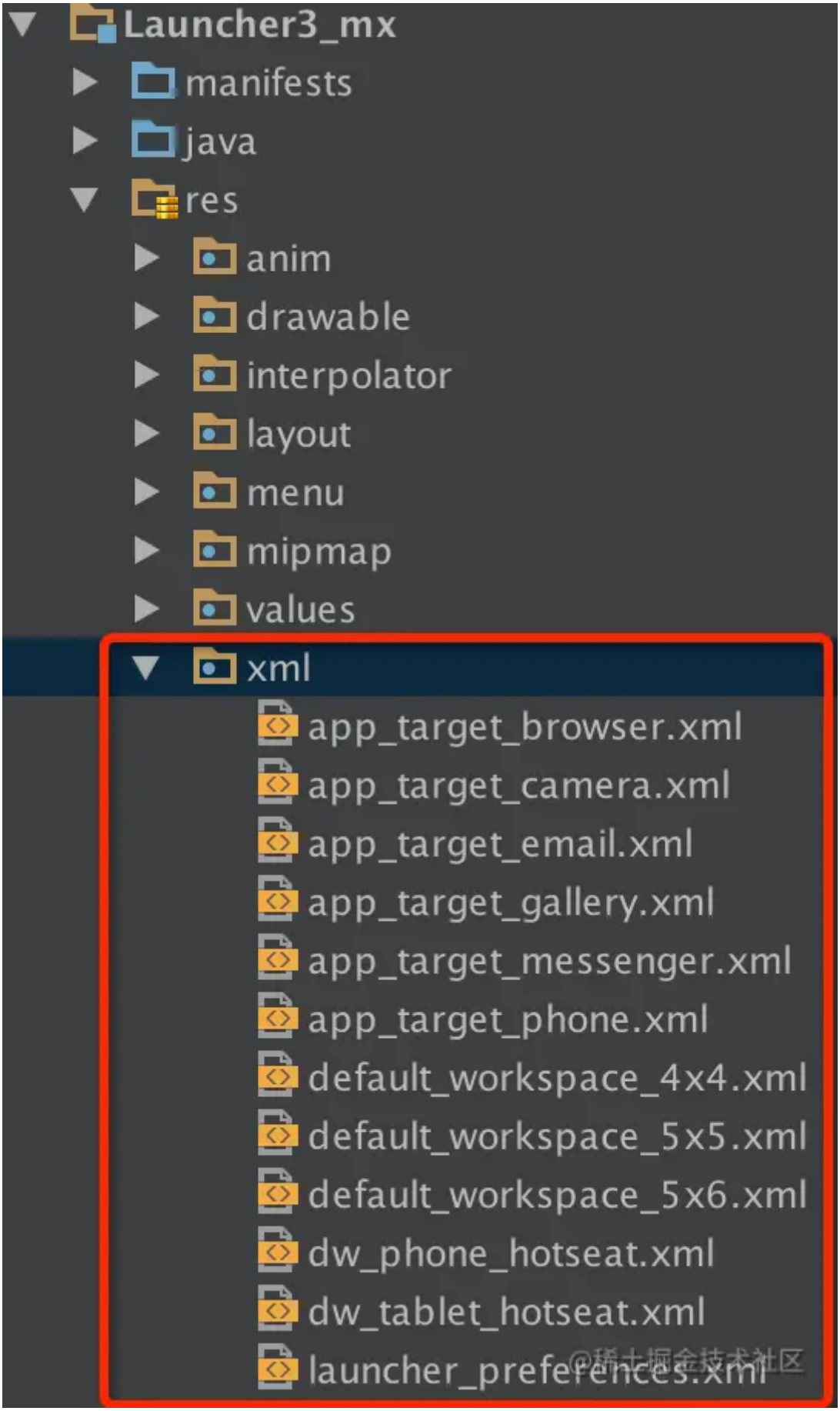
- Launcher：继承Activity,是桌面的主界面,因此可知,桌面其实就是一个activity,只是和平常的应用不同,他用来显示图标、Widget和文件夹等;
- LauncherModel：继承BroadcastReceiver,由此可知他是一个广播接收器,用来接收广播,另外,LauncherModel还主要加载数据;
- LauncherProvider：继承ContentProvider,主要是处理数据库操作;
- LauncherAppState：单例模式的全局管理类,主要是初始化一些对象,注册广播等.

2.2 默认图标配置

们在买回新的手机或者第一次安装新的Launcher后,会发现手机的第一页已经有了一些应用的图标和时钟或者天气插件,那么这个是怎么实现的呢?

其实,**手机在出厂的时候或者Launcher发到市场的时候已经默认排布了一些应用,在第一启动时就会加载并且判断手机中是否有这些图标,如果有则显示到固定位置,这个位置其实是已经写好的**.

下面是Launcher的资源文件,这个比我们平时的多一个xml文件夹,里面有很多xml文件。有三个文件,分别为default_workspace_4x4.xml,default_workspace_5x5.xml和default_workspace_5x6.xml,这三个文件就是我们默认的布局文件,后面的跟着的4x4、5x5和5x6表示桌面图标的列数和行数,也就是4行4列,5行5列,5行6列,



先看一下default_workspace_4x4.xml这个文件中的代码：

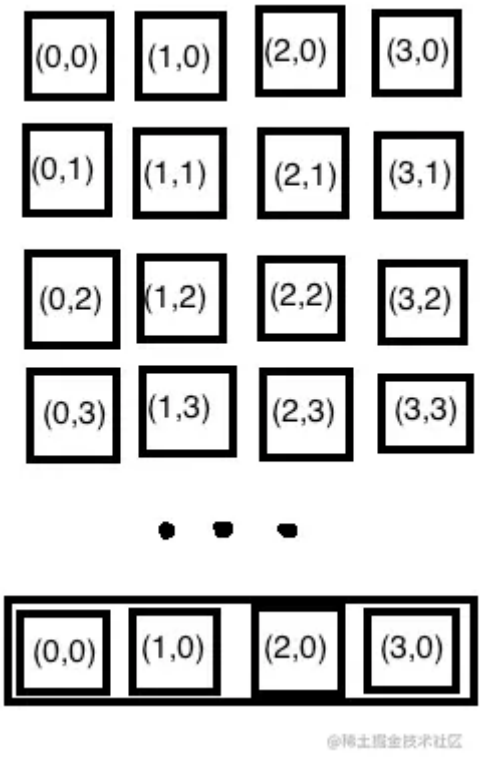
```xml
<favorites xmlns:launcher="http://schemas.android.com/apk/res-auto/com.android.launcher3">

    <!-- Hotseat -->
    <include launcher:workspace="@xml/dw_phone_hotseat" />

    <!-- Bottom row -->
    <resolve
        launcher:screen="0"
        launcher:x="0"
        launcher:y="3" >
        <favorite launcher:uri="#Intent;action=android.intent.action.MAIN;category=android.intent.category.APP_EMAIL;end" />
        <favorite launcher:uri="mailto:" />
    </resolve>

    <resolve
        launcher:screen="0"
        launcher:x="1"
        launcher:y="3" >
        <favorite launcher:uri="#Intent;action=android.intent.action.MAIN;category=android.intent.category.APP_GALLERY;end" />
        <favorite launcher:uri="#Intent;type=images/*;end" />
    </resolve>

    <resolve
        launcher:screen="0"
        launcher:x="3"
        launcher:y="3" >
        <favorite launcher:uri="#Intent;action=android.intent.action.MAIN;category=android.intent.category.APP_MARKET;end" />
        <favorite launcher:uri="market://details?id=com.android.launcher" />
    </resolve>

</favorites>
```

@稀土掘金技术社区

第20行是一个include的文件,在xml文件夹中的名字dw_phone_hotseat文件,我们后面在看,接着看上图的下面的代码,下面是三个resolve文件,里面包含一些信息,

- screen 表示第几屏
- x 表示横向的位置
- y 表示纵向的位置

那么这个位置怎定的呢,看下图



@稀土掘金技术社区

上半部分,就是我们说的4x4部分,每一格表示一个图标,在我们绘制图标的时候已经分好了格,每格的大小,只要知道知道他的位置即可绘制图标到相应的位置,那么代码中的x,y就是这个图标的位置.

上面resolve中还有两个favorite,在第一个中最后面有个"APP_",这个我们一看就知道是应用的属性,其实这就表示我们配置了那个app在这个位置。

再看一下上面介绍的hotseat那个xml文件:

```
<favorites xmlns:launcher="http://schemas.android.com/apk/res-auto/com.android.launcher3">
    <!-- Hotseat (We use the screen as the position of the item in the hotseat) -->
    <!-- Dialer, Messaging, [All Apps], Browser, Camera -->
    <resolve
        launcher:container="-101"
        launcher:screen="0"
        launcher:x="0"
        launcher:y="0" >
        <favorite launcher:uri="#Intent;action=android.intent.action.DIAL;end" />
        <favorite launcher:uri="tel:123" />
        <favorite launcher:uri="#Intent;action=android.intent.action.CALL_BUTTON;end" />
    </resolve>

    <resolve
        launcher:container="-101"
        launcher:screen="1"
        launcher:x="1"
        launcher:y="0" >
        <favorite launcher:uri="#Intent;action=android.intent.action.MAIN;category=android.intent.category.APP_MESSAGING;end" />
        <favorite launcher:uri="sms:" />
        <favorite launcher:uri="smsto:" />
        <favorite launcher:uri="mms:" />
        <favorite launcher:uri="mmsto:" />
    </resolve>

    <!-- All Apps -->

    <resolve
        launcher:container="-101"
        launcher:screen="3"
        launcher:x="3"
        launcher:y="0" >
        <favorite
            launcher:uri="#Intent;action=android.intent.action.MAIN;category=android.intent.category.APP_BROWSER;end" />
        <favorite launcher:uri="http://www.example.com/" />
    </resolve>
```

@稀土掘金技术社区

2.3 Launcher启动过程

todo

2.4 Launcher 初始化

todo

2.5 Launcher 数据加载

todo

三、 数据绑定

3.1 默认配置图标、Widget、文件夹的绑定（bind）

3.2 所有应用绑定（bind）

3.3 所有Widget的绑定（bind）

四、应用安装、更新、卸载时的数据加载

应用的安装和更新都是通过应用市场来启动，而应用的卸载是通过桌面或者系统的app管理来启动的，因此我们将应用的安装和更新一起来讲，而应用的卸载单独来讲。首先我们先看一下应用的安装和更新时桌面的数据加载。

4.1 应用安装和更新

当我们通过应用市场安装或者更新应用时，会调用系统的安装界面，并执行安装程序，在应用安装或者更新完成后系统会发出对应的广播，通过对应广播Launcher会执行相应的加载程序。

首先我们看一个App管理的兼容库：LauncherAppsCompat，这里面有一个接口和一些抽象方法，我们用的到底主要是这个接口还有两个抽象方法

接口类：

```
public interface OnAppsChangedCallbackCompat {
        void onPackageRemoved(String packageName, UserHandleCompat user);
        void onPackageAdded(String packageName, UserHandleCompat user);
        void onPackageChanged(String packageName, UserHandleCompat user);
        void onPackagesAvailable(String[] packageNames, UserHandleCompat user,
boolean replacing);
        void onPackagesUnavailable(String[] packageNames, UserHandleCompat user,
boolean replacing);
    }
```

从这个接口中的方法我们可以了解到这个是对App移除、添加、改变、可用和不可用的各种情况的处理。

抽象方法：

```
 public abstract void addOnAppsChangedCallback(OnAppsChangedCallbackCompat
listener);
 public abstract void removeOnAppsChangedCallback(OnAppsChangedCallbackCompat
listener);
```

这两个方法主要是添加、删除App管理的监听。

我们看到LauncherAppsCompat是一个抽象类，

todo

通过上面代码我们可知最后都是调用PackageUpdatedTask这个任务执行的，只是传入的参数不同，这个任务中代码很多也不贴了，我简单介绍下任务执行过程，首先区分收到广播是安装、更新、移除还是不可用，然后对不同的操作执行不同的处理，对于添加的调用addAppsToAllApps这个方法进行处理，最后调用callbacks.bindAppsAdded方法进行绑定，更新的调用callbacks.bindAppsUpdated这个方法进行更新操作，代码很简单，自己看一下就好了，对于卸载的，更新图标缓存，将其移除掉，在这些操作的同时，要还行数据的数据更新操作，如果有小部件的，也要对小部件做相应的操作处理。

4.2 应用的卸载

原生桌面的卸载应用是将图标拖拽到卸载框进行卸载的，关于拖拽的操作流程我们后续会详细讲解，这里我们直接看拖拽到相应位置的处理，在完成拖拽时会调用completeDrop这个方法，我们看看哪里实现了这个方法:

在这里有三处实现，其实从名字我们可以看出最后一个是卸载应用的实现，

```
void completeDrop(final DragObject d) {
        ...
        if (startUninstallActivity(mLauncher, d.dragInfo)) {

            final Runnable checkIfUninstallWasSuccess = new Runnable() {
                @Override
                public void run() {
                    String packageName = componentInfo.first.getPackageName();
                    boolean uninstallSuccessful =
!AllAppsList.packageHasActivities(
                            getContext(), packageName, user);
                    sendUninstallResult(d.dragSource, uninstallSuccessful);
                }
            };
            mLauncher.addOnResumeCallback(checkIfUninstallWasSuccess);
        } else {
            sendUninstallResult(d.dragSource, false);
        }
    }
```

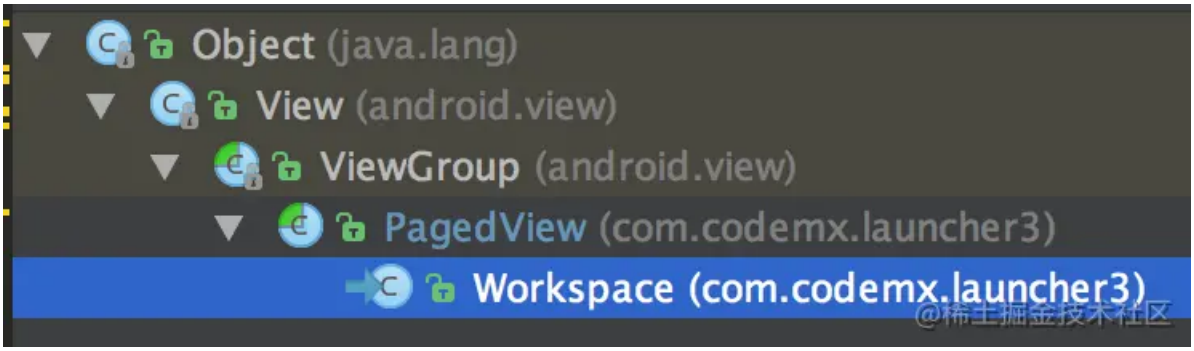
首先判断startUninstallActivity，这个方法中调用startApplicationUninstallActivity方法，如果是系统应用返回false，如果不是启动卸载界面并且返回true，启动卸载界面是通过Intent.ACTION_DELETE这个action启动的，如果能够卸载，执行完卸载返回到桌面时，或者取消返回到桌面时，检测是否卸载成功，然后调用sendUninstallResult方法，在这个方法中调用onUninstallActivityReturned回调函数，这个回调函数是在Folder或者workspace中实现的，其实代码这两个地方都一样，最后都执行onDropCompleted方法，如果移除成功，则调用removeWorkspaceItem方法，这个方法主要是从CellLayout中删除对应的图标，如果没有成功则刷新UI，也就是对应的图标放回原处。

五、Workspace滑动

Workspace包含多个CellLayout，每个CellLayout是一个页面，多个CellLayout可以通过滑动切换，这样就可以找到不同的图标，

5.1 Workspace布局：

首先我们先看一下Workspace的继承逻辑：



Workspace继承PagedView，而PagedView又继承ViewGroup，由名字我们可以猜出，PagedView是分页的自定义View。我们直接看Workspace是如何布局的，其实，workspace的布局是在PagedView里面处理的，首先是onMeasure方法，我们看下源码：

```
protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
        // 如果没有子View则按照父类的尺寸进行测量
        if (getChildCount() == 0) {
            super.onMeasure(widthMeasureSpec, heightMeasureSpec);
            return;
        }

        // We measure the dimensions of the PagedView to be larger than the pages
so that when we
        // zoom out (and scale down), the view is still contained in the parent
        //上面这句话是说我们在测量尺寸时要比我们正常状态下的尺寸要大，为什么要
        //大，我们在第一章概述中讲过，当你长按桌面时，桌面的workspace会缩小，
        //此时弹出菜单，CellLayout缩小，然后你可以拖动CellLayout改变顺序，
        //如果你没有放大PagedView的尺寸，你在缩小时，在整个屏幕上的
        //workspace就不会沾满整个屏幕，导致你拖动困难。

        ...

        //这里将最大尺寸放大了两倍
        int parentWidthSize = (int) (2f * maxSize);
        int parentHeightSize = (int) (2f * maxSize);
        int scaledWidthSize, scaledHeightSize;

        ...

        mViewport.set(0, 0, widthSize, heightSize);

        ...

        setMeasuredDimension(scaledWidthSize, scaledHeightSize);
    }
```

需要注意的地方已经在上面代码注释了，省略的代码是找到测量尺寸和测量模式，最后将相应的尺寸和模式放置到父View和子View中。

测量完成后就开始布局，也就是回调onLayout函数：

```
protected void onLayout(boolean changed, int left, int top, int right, int
bottom) {
        if (getChildCount() == 0) {
            return;
        }

        ...

        // 此处用到一个mIsRtl,这个是判断手机布局是从左到右还是从右到左，我们正常的习惯
        // 是从左到右，一些国家，比如阿拉伯语情况下是从右到左，因此此处要进行处理。
        final int startIndex = mIsRtl ? childCount - 1 : 0;
        final int endIndex = mIsRtl ? -1 : childCount;
        final int delta = mIsRtl ? -1 : 1;

        ...

        for (int i = startIndex; i != endIndex; i += delta) {
            final View child = getPageAt(i);
            if (child.getVisibility() != View.GONE) {
                lp = (LayoutParams) child.getLayoutParams();
                int childTop;
                if (lp.isFullScreenPage) {
                    childTop = offsetY;
                } else {
                    childTop = offsetY + getPaddingTop() + mInsets.top;
                    if (mCenterPagesVertically) {
                        childTop += (getViewportHeight() - mInsets.top -
mInsets.bottom - verticalPadding - child.getMeasuredHeight()) / 2;
                    }
                }

                final int childWidth = child.getMeasuredWidth();
                final int childHeight = child.getMeasuredHeight();

                child.layout(childLeft, childTop,
                        childLeft + child.getMeasuredWidth(), childTop +
childHeight);

                ...

                childLeft += childWidth + pageGap + getChildGap();
            }
        }

        ...

    }
```

上面代码是个for循环，就是从第一个CellLayout到最后一个进行设置位置参数，然后进行布局，Workspace是横向滑动的，因此布局时，所有的CellLayout的顶部和底部距离是一样的，只是要考虑顶部状态栏的高度，横向上，从第一个开始由左向右或者由右向左进行排布即可，（由左向右举例：）也就是固定第一个CellLayout后调整左边距的位置即可，每增加一个CellLayout，后一个的左侧到Workspace左侧边距就增加一个CellLayout的作站用的宽度，依次类推，就可以将所有CellLayout布局完成。这段代码并不难，主要是自定义View的知识。

5.2 Workspace滑动

workspace滑动就是onTouchEvent事件，关键代码也在这个方法里面，workspace继承PagedView，因此他的onTouchEvent事件是在PagedView中实现的，我们看一下代码：

```
public boolean onTouchEvent(MotionEvent ev) {
        super.onTouchEvent(ev);

        // Skip touch handling if there are no pages to swipe
        if (getChildCount() <= 0) return super.onTouchEvent(ev);

        acquireVelocityTrackerAndAddMovement(ev);

        final int action = ev.getAction();

        switch (action & MotionEvent.ACTION_MASK) {
            case MotionEvent.ACTION_DOWN:
            ...
                if (mTouchState == TOUCH_STATE_SCROLLING) {
                    ...
                }
                break;

            case MotionEvent.ACTION_MOVE:
                if (mTouchState == TOUCH_STATE_SCROLLING) {//滚动
                    ...
                } else if (mTouchState == TOUCH_STATE_REORDERING) {//拖动重新排序
                    ...
                } else {
                    determineScrollingStart(ev);
                }
                break;

            case MotionEvent.ACTION_UP:
                if (mTouchState == TOUCH_STATE_SCROLLING) {
                    ...
                } else if (mTouchState == TOUCH_STATE_PREV_PAGE) {
                    ...
                } else if (mTouchState == TOUCH_STATE_NEXT_PAGE) {
                    ...
                } else if (mTouchState == TOUCH_STATE_REORDERING) {
                    ...
                } else {
                    ...
                }
                ...
                break;

            case MotionEvent.ACTION_CANCEL:
                ...
                break;

            case MotionEvent.ACTION_POINTER_UP:
                ...
                break;
```

```
        }

        return true;
    }
```

六、拖拽

todo

七、小部件的加载、添加以及大小调节

todo

八、加载Icon、设置壁纸

对于Icon的操作其实主要是加载、更新以及删除，加载主要是启动Launcher、安装应用，更新是在更新应用时更新Icon、删除是卸载应用时会删除Icon，因此我们可以从这几方面分析Icon的处理。

8.1 Launcher启动时Icon加载

todo

定制自己的Launcher

todo