Android的SystemUI应用,为用户提供查看最近使用应用的列表，当用户点击Switch按键时，PhoneWindowManager会拦截此次key事件，拦截后调用StatusBarManagerService 接口，StatusBarManagerService通过调用mBar（SystemUI 注册到StatusBarManagerService的Binder对象）通知SystemUI启动RecentsActivity显示最近使用应用的列表。   
启动的大概流程图如下：   


# RecentsActivity的启动流程

## step1拦截key\_Switch事件：

在key事件分发之前，会调用PhoneWindowManager的interceptKeyBeforeDispatching函数优先处理key事件.

**[java]** [view plain](http://blog.csdn.net/qq1028850792/article/details/79346153) [copy](http://blog.csdn.net/qq1028850792/article/details/79346153)

1. **public** **long** interceptKeyBeforeDispatching(WindowState win, KeyEvent event, **int** policyFlags) {
2. } **else** **if** (keyCode == KeyEvent.KEYCODE\_APP\_SWITCH) {
3. **if** (!keyguardOn) {
4. **if** (down && repeatCount == 0) {
5. preloadRecentApps();//预加载流程自行分析
6. } **else** **if** (!down) {
7. toggleRecentApps();
8. }
9. }
10. **return** -1;
11. }

preloadRecentApps();主要是预加载流程，和启动流程关键点基本一致，感兴趣自行分析。 当收到swith事件，不是down事件时调用toggleRecentApps函数：

## step 2 toggleRecentApps函数

f

**[java]** [view plain](http://blog.csdn.net/qq1028850792/article/details/79346153) [copy](http://blog.csdn.net/qq1028850792/article/details/79346153)

1. **private** **void** toggleRecentApps() {
2. mPreloadedRecentApps = **false**; // preloading no longer needs to be canceled
3. **try** {
4. IStatusBarService statusbar = getStatusBarService();
5. **if** (statusbar != **null**) {
6. statusbar.toggleRecentApps();
7. }
8. } **catch** (RemoteException e) {
9. Slog.e(TAG, "RemoteException when toggling recent apps", e);
10. // re-acquire status bar service next time it is needed.
11. mStatusBarService = **null**;
12. }
13. }

直接调用StatusBarManagerService的toggleRecentApps函数

## step 3 StatusBarManagerService .toggleRecentApps()

**[java]** [view plain](http://blog.csdn.net/qq1028850792/article/details/79346153) [copy](http://blog.csdn.net/qq1028850792/article/details/79346153)

1. **public** **void** toggleRecentApps() {
2. **if** (mBar != **null**) {
3. **try** {
4. mBar.toggleRecentApps();
5. } **catch** (RemoteException ex) {}
6. }
7. }

mBar 是SystemUI启动的时候，在BaseStatusBar Start函数中通过调用StatusBarManagerService 的registerStatusBar函数注册过来的如下：

mCommandQueue = new CommandQueue(this, iconList);

mBarService.registerStatusBar(mCommandQueue, iconList, switches, binders);

mBar 是一个（IStatusBar）binder代理对象，他的本地对象就是运行在SystemUI的mCommandQueue对象。

这样StatusBarManagerService 就通过mBar进程间调用调用到mCommandQueue的toggleRecentApps函数。

## step 4 mCommandQueue. toggleRecentApps

**[java]** [view plain](http://blog.csdn.net/qq1028850792/article/details/79346153) [copy](http://blog.csdn.net/qq1028850792/article/details/79346153)

1. @Override
2. **public** **void** toggleRecentApps() {
3. **int** msg = MSG\_TOGGLE\_RECENTS\_APPS;
4. mHandler.removeMessages(msg);
5. mHandler.sendEmptyMessage(msg);
6. }

这个函数只是发送了一个消息到主线程消息队列。最后处理该消息的代码如下：

mCallbacks.toggleRecentApps();

mCallbacks 是BaseStatusBar 在step3中初始化CommandQueue时传入this对象。

BaseStatusBar 又发送消息到消息队列。最后处理该消息的是BaseStatusBar 的toggleRecents函数。

## Step5 BaseStatusBar. toggleRecents

**[java]** [view plain](http://blog.csdn.net/qq1028850792/article/details/79346153) [copy](http://blog.csdn.net/qq1028850792/article/details/79346153)

1. **protected** **void** toggleRecents() {
2. **if** (mRecents != **null**) {
3. sendCloseSystemWindows(mContext, SYSTEM\_DIALOG\_REASON\_RECENT\_APPS);
4. mRecents.toggleRecents(mDisplay, mLayoutDirection, getStatusBarView());
5. }
6. }

mRecents 是Recents类对象他继承自SystemUI，在系统启动的时候，SystemServer会调用startSystemUi函数启动SystemUI应用的SystemUIService服务，

在SystemUIService 的的onCreate函数中会调用

SystemUIApplication 的startServicesIfNeeded，

在这个函数中会启动会实例化多个继承自SystemUI的类，并调用他们的Start函数，其中就包括Recents的start函数。

在Start函数中，调用putComponent(RecentsComponent.class, this);把Recents对象保存到了SystemUI的mComponents中。

BaseStatusBar中的mRecents成员变量也就是从里面取出来的。

## Step6 Recents. toggleRecents

**[java]** [view plain](http://blog.csdn.net/qq1028850792/article/details/79346153) [copy](http://blog.csdn.net/qq1028850792/article/details/79346153)

1. **public** **void** toggleRecents(Display display, **int** layoutDirection, View statusBarView) {
2. **if** (mUseAlternateRecents) {
3. mAlternateRecents.onToggleRecents(statusBarView);
4. **return**;
5. }
6. ......
7. }

mUseAlternateRecents 默认为真，mAlternateRecents.onToggleRecents(statusBarView);该函数直接调用了toggleRecentsActivity

## step7 AlternateRecentsComponent. toggleRecentsActivity

**[java]** [view plain](http://blog.csdn.net/qq1028850792/article/details/79346153) [copy](http://blog.csdn.net/qq1028850792/article/details/79346153)

1. **void** toggleRecentsActivity() {
2. ActivityManager.RunningTaskInfo topTask = getTopMostTask();
3. AtomicBoolean isTopTaskHome = **new** AtomicBoolean();
4. **if** (isRecentsTopMost(topTask, isTopTaskHome)) {
5. Intent intent = **new** Intent(ACTION\_TOGGLE\_RECENTS\_ACTIVITY);
6. intent.setPackage(mContext.getPackageName());
7. intent.addFlags(Intent.FLAG\_RECEIVER\_REGISTERED\_ONLY\_BEFORE\_BOOT |
8. Intent.FLAG\_RECEIVER\_FOREGROUND);
9. mContext.sendBroadcastAsUser(intent, UserHandle.CURRENT);
10. mLastToggleTime = System.currentTimeMillis();
11. **return**;
12. } **else** {
13. startRecentsActivity(topTask, isTopTaskHome.get());
14. }
15. }

这个函数主要是首先判断当前界面是不是RecentsActivity

如果是就发送一个广播，关闭RecentsActivity.广播肯定是RecentsActivity接收了，接收后调用退出动画，退出完成后调用RecentsActivity的finish函数。

如果不是则调用startRecentsActivity启动RecentsActivity。

RecentsActivity启动：界面的展示离不开数据，从界面上看，RecentsActivity至少需要应用名称，图片，以及截图，接下来看RecentsActivity数据的获取流程

## Step8 在RecentsActivity的onStart函数中有调用updateRecentsTasks来更新获取界面显示的数据

D

**[java]** [view plain](http://blog.csdn.net/qq1028850792/article/details/79346153) [copy](http://blog.csdn.net/qq1028850792/article/details/79346153)

1. **void** updateRecentsTasks(Intent launchIntent) {
2. ……
3. RecentsTaskLoader loader = RecentsTaskLoader.getInstance();
4. SpaceNode root = loader.reload(**this**,
5. Constants.Values.RecentsTaskLoader.PreloadFirstTasksCount,
6. mConfig.launchedFromHome);
7. ArrayList<TaskStack> stacks = root.getStacks();
8. **if** (!stacks.isEmpty()) {
9. mRecentsView.setTaskStacks(root.getStacks());
10. }
11. ……
12. **if** (mConfig.launchedWithNoRecentTasks) {
13. **if** (mEmptyView == **null**) {
14. mEmptyView = mEmptyViewStub.inflate();
15. }
16. mEmptyView.setVisibility(View.VISIBLE);
17. mRecentsView.setSearchBarVisibility(View.GONE);
18. } **else** {
19. **if** (mEmptyView != **null**) {
20. mEmptyView.setVisibility(View.GONE);
21. }
22. **if** (mRecentsView.hasSearchBar()) {
23. mRecentsView.setSearchBarVisibility(View.VISIBLE);
24. } **else** {
25. addSearchBarAppWidgetView();
26. }
27. }
28. mScrimViews.prepareEnterRecentsAnimation();
29. }

这个函数主要就是

1 调用RecentsTaskLoader的reload函数获取相关最近使用应用的相关信息。

2 调用mRecentsView.setTaskStacks，把获得的TaskStack设置到RecentsView中创建UI结构图中的TaskStackView

3 如果没有最近使用的信息则显示空的提示信息。这一步比较简单只是设置View的显示隐藏。

## Step9 RecentsTaskLoader.reload函数

先看一下RecentsTaskLoader构造函数中初始化的几个重要的类和变量

1 mSystemServicesProxy = new SystemServicesProxy(context);

SystemServicesProxy 这个类主要承接了RecentsActivity和系统服务的交互，包含获取最近任务列表，启动制定任务所在的应用等等。

2 mLoadQueue = new TaskResourceLoadQueue();一个队列主要用于缓存任务

3 mApplicationIconCache = new DrawableLruCache(iconCacheSize);

mThumbnailCache = new BitmapLruCache(thumbnailCacheSize);

mActivityLabelCache = new StringLruCache(100);

三个缓存，主要用于缓存应用图标，名称，截屏。

4 mLoader = new TaskResourceLoader(mLoadQueue, mApplicationIconCache, mThumbnailCache, mDefaultThumbnail, mDefaultApplicationIcon);

TaskResourceLoader 继承自Runnable 他的构造函数中启动了一个HandlerThread，在他的Run函数中一直循环从mLoadQueue中读取任务并获取任务所对应的截图，并缓存到mThumbnailCache中，然后通知TaskView加载截图并显示。

## Step10 mRecentsView.setTaskStacks

**[java]** [view plain](http://blog.csdn.net/qq1028850792/article/details/79346153) [copy](http://blog.csdn.net/qq1028850792/article/details/79346153)

1. **public** **void** setTaskStacks(ArrayList<TaskStack> stacks) {
2. **int** childCount = getChildCount();
3. **for** (**int** i = childCount - 1; i >= 0; i--) {
4. View v = getChildAt(i);
5. **if** (v != mSearchBar) {
6. removeViewAt(i);
7. }
8. }
9. mStacks = stacks;
10. **int** numStacks = mStacks.size();
11. **for** (**int** i = 0; i < numStacks; i++) {
12. TaskStack stack = mStacks.get(i);
13. TaskStackView stackView = **new** TaskStackView(getContext(), stack);
14. stackView.setCallbacks(**this**);
15. **if** (mConfig.debugModeEnabled) {
16. stackView.setDebugOverlay(mDebugOverlay);
17. }
18. addView(stackView);
19. }
20. mAlreadyLaunchingTask = **false**;
21. }

这个函数先移除RecentsView中的所有子View然后把获取的Taskstack保存到mStacks中，然后创建TaskStackView 视图，并把视图中要显示的TaskStack保存到TaskStackView中。

最后把创建的TaskStackView视图作为子View添加到RecentsView中。

到此装载过程完成。

当RecentsActivity中ViewRootImpl对所有的View进程一次Measure ,过程中。会调用TaskStackView的onMeasure函数，在onMeasure函数中会调用到TaskView的synchronizeStackViewsWithModel函数。

## Step11 TaskStackView. synchronizeStackViewsWithModel

这个函数比较长就不贴代码了，主要完成两个工作，

1 创建UI结构图中的TaskView并作为子View添加到TaskStackView中。

2 为TaskView计算在TaskStackView中的坐标并做一个TaskViewTransform平移动画。（这部分代码就不详细分析了）

在这个函数中就是通过调用tv = mViewPool.pickUpViewFromPool(task, task);来创建TaskView

mViewPool = new ViewPool

**[java]** [view plain](http://blog.csdn.net/qq1028850792/article/details/79346153) [copy](http://blog.csdn.net/qq1028850792/article/details/79346153)

1. V pickUpViewFromPool(T preferredData, T prepareData) {
2. V v = **null**;
3. **boolean** isNewView = **false**;
4. **if** (mPool.isEmpty()) {
5. v = mViewCreator.createView(mContext);
6. isNewView = **true**;
7. }
8. ……
9. mViewCreator.prepareViewToLeavePool(v, prepareData, isNewView);
10. **return** v;
11. }

从ViewPool的构造函数可以知道mViewCreator 就是TaskStackView

TaskStackView的createView就是创建了一个TaskView

在调用TaskStackView的prepareViewToLeavePool函数

## Step12 TaskStackView.prepareViewToLeavePool函数

**[java]** [view plain](http://blog.csdn.net/qq1028850792/article/details/79346153) [copy](http://blog.csdn.net/qq1028850792/article/details/79346153)

1. **public** **void** prepareViewToLeavePool(TaskView tv, Task task, **boolean** isNewView) {
2. tv.onTaskBound(task);
3. ……
4. RecentsTaskLoader.getInstance().loadTaskData(task);
5. tv.setClipViewInStack(**true**);
6. **if** (isNewView) {
7. addView(tv, insertIndex);
8. tv.setTouchEnabled(**true**);
9. tv.setCallbacks(**this**);
10. } **else** {
11. attachViewToParent(tv, insertIndex, tv.getLayoutParams());
12. }
13. }

首先调用TaskView的onTaskBound函数，这个函数主要就是设置TaskView所关联的Task，也就是说所要显示的应用，然后设置TaskView为Task的成员变量mCb，而mCb主要用来通知刷新的。然后把创建的TaskView添加到TaskStackView中。

然后调用RecentsTaskLoader.getInstance().loadTaskData(task)函数

## Step13 RecentsTaskLoader. loadTaskData

**[java]** [view plain](http://blog.csdn.net/qq1028850792/article/details/79346153) [copy](http://blog.csdn.net/qq1028850792/article/details/79346153)

1. **public** **void** loadTaskData(Task t) {
2. Drawable applicationIcon = mApplicationIconCache.getAndInvalidateIfModified(t.key);
3. Bitmap thumbnail = mThumbnailCache.getAndInvalidateIfModified(t.key);
4. **boolean** requiresLoad = (applicationIcon == **null**) || (thumbnail == **null**);
5. applicationIcon = applicationIcon != **null** ? applicationIcon : mDefaultApplicationIcon;
6. **if** (requiresLoad) {
7. mLoadQueue.addTask(t);
8. }
9. t.notifyTaskDataLoaded(thumbnail == mDefaultThumbnail ? **null** : thumbnail, applicationIcon);
10. }

这个函数主要是判断当前Task的应用启动图标和截图是否已经在mApplicationIconCache中缓存。

如果没有缓存，则把Task加入到等待加载截图的队列mLoadQueue中，前面也介绍过这个队列TaskResourceLoader的run函数会一直从mLoadQueue 读取Task去获取其对应的应用截图和启动图标。

## Step14 TaskResourceLoader.run

**[java]** [view plain](http://blog.csdn.net/qq1028850792/article/details/79346153) [copy](http://blog.csdn.net/qq1028850792/article/details/79346153)

1. **public** **void** run() {
2. **while** (**true**) {
3. **if** (mCancelled) {
4. ……
5. } **else** {
6. SystemServicesProxy ssp = mSystemServicesProxy;
7. **final** Task t = mLoadQueue.nextTask();
8. **if** (t != **null**) {
9. Drawable cachedIcon = mApplicationIconCache.get(t.key);
10. Bitmap cachedThumbnail = mThumbnailCache.get(t.key);
11. **if** (cachedIcon == **null**) {
12. cachedIcon = getTaskDescriptionIcon(t.key, t.icon, t.iconFilename, ssp,
13. mContext.getResources());
14. **if** (cachedIcon == **null**) {
15. ActivityInfo info = ssp.getActivityInfo(t.key.baseIntent.getComponent(),
16. t.key.userId);
17. **if** (info != **null**) {
18. cachedIcon = ssp.getActivityIcon(info, t.key.userId);
19. }
20. }
22. **if** (cachedIcon == **null**) {
23. cachedIcon = mDefaultApplicationIcon;
24. }
25. mApplicationIconCache.put(t.key, cachedIcon);
26. }
27. **if** (cachedThumbnail == **null**) {
28. cachedThumbnail = ssp.getTaskThumbnail(t.key.id);
29. **if** (cachedThumbnail != **null**) {
30. cachedThumbnail.setHasAlpha(**false**);
31. } **else** {
32. cachedThumbnail = mDefaultThumbnail;
33. }
34. mThumbnailCache.put(t.key, cachedThumbnail);
35. }
36. **if** (!mCancelled) {
37. **final** Drawable newIcon = cachedIcon;
38. **final** Bitmap newThumbnail = cachedThumbnail == mDefaultThumbnail
39. ? **null** : cachedThumbnail;
40. mMainThreadHandler.post(**new** Runnable() {
41. @Override
42. **public** **void** run() {
43. t.notifyTaskDataLoaded(newThumbnail, newIcon);
44. }
45. });
46. }
47. }
48. ……
49. }
50. }
51. }
52. }

这个函数比较好分析，首先从缓存中查看对应的Task的应用启动图标和截图是否已经存在，如果不存在就去获取，获取过程都是调用的系统接口，自行查看。

获取完成后缓存，然后往主线程发送一个消息，去执行t.notifyTaskDataLoaded(newThumbnail, newIcon);函数。

## Task. notifyTaskDataLoaded

**[java]** [view plain](http://blog.csdn.net/qq1028850792/article/details/79346153) [copy](http://blog.csdn.net/qq1028850792/article/details/79346153)

1. /\*\* Notifies the callback listeners that this task has been loaded \*/
2. **public** **void** notifyTaskDataLoaded(Bitmap thumbnail, Drawable applicationIcon) {
3. **this**.applicationIcon = applicationIcon;
4. **this**.thumbnail = thumbnail;
5. **if** (mCb != **null**) {
6. mCb.onTaskDataLoaded();
7. }
8. }

这个函数就是保存获取的应用启动图标和应用截图，然后调用mCb. onTaskDataLoaded函数。

在step12中为TaskView关联要显示的Task后，还把TaskView设置为Task的mCb

TaskView的onTaskDataLoaded函数大家就知道了，把刚刚装载的应用启动图标和截图设置到TaskView对应的View中显示出来。

# REF

[RecentsTask分析Android5.0](http://blog.csdn.net/zhangyawen1i/article/details/53350006)