# Todo

Libcore

Libnativehelper

Developers

Toolchain

Tools

Development

bionic

dji.json 网络配置

C:\Users\key.guan\Desktop\20172300.log ANR剖析

C:\Users\key.guan\Desktop\anr\_log.log

Chaechrevorey:MISUC 为何不会擦除，LAST\_LOG

Ps等Android命令研究

if (mSystemPropertyUpdater == null) {

mSystemPropertyUpdater = new Runnable() {

@Override public void run() {

synchronized (mLock) {

for (int i = mRoots.size() - 1; i >= 0; --i) {

mRoots.get(i).loadSystemProperties();

}

}

}

};

SystemProperties.addChangeCallback(mSystemPropertyUpdater);

}

* [https://blog.csdn.net/s278777851/article/details/6956226](javascript:%20void%200)



* 17:05

[https://github.com/KoVszone/GamePad](javascript:%20void%200)

* 17:24

[http://www.technorange.com/2017/01/how-to-map-gamepad-or-joystick-on-android-to-play-games-using-usb-bt-joycenter/](javascript:%20void%200)

[https://blog.csdn.net/AWNUXCVBN/article/details/12285761](javascript:%20void%200)

* 20:2

[https://blog.csdn.net/luoshengyang/article/details/6882903](javascript:%20void%200)

JNI层，JNI的架构和实现方式，虚拟机原理？

# 栈管理

我们知道,a ctivity 在 AMS 中的形式是 ActivityRecord,task 在 AMS 中的形式为TaskRecord,进程在 AMS 中的管理形式为 ProcessRecord。

**Android系统界面用户体验组织（TASKS），**，用户体验到的直观界面都是由很多Tasks组成的。比如，我们从最近任务栏，就可以看到很多TASKS。用户可以随意的切换到其中的一个TASK。也可以按HOME键退出一个TASK。TASK的组织形式是ActivityStack，一个ActivityStack是由好多Activity组成的堆栈。从Android系统设计者的角度来看，一个Task定义了一组行为，而这组行为是由跨多个应用程序的多个Activity组织而构成的。这个打破了以往由应用程序（进程）来定义的资源边界。一个Task就是一个场景的实现。从此构建系统用户体验和行为的边界在于TASK，而非应用程序或者进程。

## 概念

### Task

Task 是activities的集合，通过activity stack来管理，依靠先进后出队列来实现;每个task中都至少有一个activity，新实例出来的activity置于栈顶，Task可以被切换到后台

### Activity Stack

如上所诉，Activity承担了大量的显示和交互工作，从某种角度上将，我们看见的应用程序就是许多个Activity的组合。为了让这许多 Activity协同工作而不至于产生混乱，Android平台设计了一种堆栈机制用于管理Activity，其遵循先进后出的原则，系统总是显示位于栈顶的Activity，从逻辑上将，位于栈顶的Activity也就是最后打开的Activity，这也是符合逻辑的。

在操作应用程序时，每次启动新的Activity，都会将此压入Activity Stack，当用户执行返回操作时，移除Activity Stack顶上的Activity，这样就实现了返回上一个Activty的功能。直到用户一直返回到Home Screen，这时候可以理解为移除了Activity Stack所有的Activity，这个Activity Stack不再存在，应用程序也结束了运行。

可以通过 adb shell dumpsys 查看 TASKS的ActivityStacks

### task的taskAffinity

 任务归属，taskAffinity 这个属性主要是决定持有每个activity属于哪个task。 默认情况下，同一个包中的activity共享同一个affinity（任务共用性）。

### **task的launchMode**

 standard(default)：standard，标准的Activity是可以随意插入到TASK中去的一个组织结构，可以去TaskA，也可以去TaskB，也可以去TaskC，直接并无任何的联系

 singleTop，如果在任务的栈顶正好存在该Activity的实例，就重用该实例，否则就创建新的实例并放入栈顶。

 singleTask，如果在栈中已经有该Activity的实例，就重用该实例(会调用实例的onNewIntent())。重用时，会让该实例回到栈顶，因此在它上面的实例将会被移除栈。如果栈中不存在该实例，将会创建新的实例放入栈中。

 singleInstance

## 源码分析

### 主要涉及4个类

**1) ActivityRecord：activity记录的基本数据结构，**存在历史栈的一个实例，代表一个Activity。

**2) TaskRecord：**Activity栈，内部维护一个ArrayList<ActivityRecord>

**3) ActivityStack：**并不是一个Activity栈，真正意义上的Activity栈是TaskRecord，这个类是负责管理各个Activity栈，内部维护一个ArrayList<TaskRecord>

**4) ActivityStackSupervisor：**内部持有一个ActivityStack，而ActivityStack内部也持有ActivityStackSupervisor，相当于ActivityStack的辅助管理类

#### **ActivityRecord**

/frameworks/base/services/core/java/com/android/server/am/ActivityRecord.java

**void** **[dump](http://androidxref.com/5.1.0_r1/s?refs=dump&project=frameworks)**([PrintWriter](http://androidxref.com/5.1.0_r1/s?defs=PrintWriter&project=frameworks) **[pw](http://androidxref.com/5.1.0_r1/s?refs=pw&project=frameworks)**, [String](http://androidxref.com/5.1.0_r1/s?defs=String&project=frameworks) **[prefix](http://androidxref.com/5.1.0_r1/s?refs=prefix&project=frameworks)**) 定义的方法用于dumpsys activity对于2.2.1.1. ActivityRecord的打印，

final class ActivityRecord {

TaskRecord **task**; // the task this is in.

final IApplicationToken.Stub appToken;

final int userId;

int theme;

int launchMode;

...

}

成员变量task表示自己所在的TaskRecord，这样要找到自己所在的TaskRecord就不必遍历查找了。

#### TaskRecord

final class TaskRecord {

/\*\* List of all activities in the task arranged in history order \*/

final ArrayList<ActivityRecord> mActivities;

/\*\* Current stack \*/

ActivityStack stack;

}

同样的道理，成员变量stack表示自己所在的ActivityStack

#### ActivityStack

final class ActivityStack {

private ArrayList<TaskRecord> **mTaskHistory** = new ArrayList<>();

/\*\* Run all ActivityStacks through this \*/

final ActivityStackSupervisor mStackSupervisor;

ActivityStack(ActivityStackSupervisor.ActivityContainer activityContainer, RecentTasks recentTasks) {

mStackSupervisor = activityContainer.getOuter();

...

}

}

#### ActivityStackSupervisor

public final class ActivityStackSupervisor {

private ActivityStack mFocusedStack;

}

### 场景分析

#### 二、场景解析

##### 1、从桌面第一次启动App

startActivityLocked里构造一个ActivityRecord  
新建一个TaskRecord，并存入mTaskHistory  
ActivityRecord存入mActivities



final int startActivityUncheckedLocked(...) {

final int startActivityUncheckedLocked(...) {

if (reuseTask == null) {

r.setTask(targetStack.createTaskRecord(...);

...

targetStack.startActivityLocked(r, newTask, doResume, keepCurTransition, options);

...

}

}

1) TaskRecord存入mTaskHistory

TaskRecord createTaskRecord(int taskId, ActivityInfo info, Intent intent,

IVoiceInteractionSession voiceSession, IVoiceInteractor voiceInteractor,

boolean toTop) {

TaskRecord task = new TaskRecord(mService, taskId, info, intent, voiceSession,

voiceInteractor);

addTask(task, toTop, false);

return task;

}

void addTask(final TaskRecord task, final boolean toTop, boolean moving) {

task.stack = this;

if (toTop) {

insertTaskAtTop(task, null);

} else {

mTaskHistory.add(0, task);

updateTaskMovement(task, false);

}

...

}

private void insertTaskAtTop(TaskRecord task, ActivityRecord newActivity) {

...

mTaskHistory.add(taskNdx, task);

updateTaskMovement(task, true);

}

2) ActivityRecord存入mActivities



final void startActivityLocked(ActivityRecord r, boolean newTask, ...) {

...

task = mTaskHistory.get(taskNdx);

...

task.addActivityToTop(r);

}

void addActivityToTop(ActivityRecord r) {

addActivityAtIndex(mActivities.size(), r);

}

void addActivityAtIndex(int index, ActivityRecord r) {

...

mActivities.add(index, r);

...

}

##### 2、App启动一个Activity

会不会新建一个TaskRecord取决于launchMode，默认的standard模式不会创建新的TaskRecord  
构造一个ActivityRecord存入mActivities，与上面第二步一样

##### 3、回退



/\*\* @return true if this was the last activity in the task \*/

boolean removeActivity(ActivityRecord r) {

mActivities.remove(r);

...

if (mActivities.isEmpty()) {

return !mReuseTask;

}

...

return false;

}

#### 4、再次回退，回到桌面

从mActivities移除当前ActivityRecord与上面一样，只是当mActivities为空时，会触发mTaskHistory移除当前TaskRecord，如果mTaskHistory为空，则切换到桌面，给mStackSupervisor.mFocusedStack重新赋值

private void removeActivityFromHistoryLocked(ActivityRecord r, String reason) {

...

final TaskRecord task = r.task;

if (task != null && task.removeActivity(r)) {

if (mStackSupervisor.isFrontStack(this) && task == topTask() &&

task.isOverHomeStack()) {

mStackSupervisor.moveHomeStackTaskToTop(task.getTaskToReturnTo(), reason);

}

removeTask(task, reason);

}

}

void removeTask(TaskRecord task, String reason, boolean notMoving) {

...

mTaskHistory.remove(task);

...

if (mTaskHistory.isEmpty()) {

final boolean notHomeStack = !isHomeStack();

if (isOnHomeDisplay()) {

String myReason = reason + " leftTaskHistoryEmpty";

if (mFullscreen || !adjustFocusToNextVisibleStackLocked(null, myReason)) {

mStackSupervisor.moveHomeStack(notHomeStack, myReason);

}

}

...

}

...

task.stack = null;

}

如果不是从Activity调用startActivity，那么目标Activity就不知道自己该属于哪个TaskRecord，所以得指定FLAG\_ACTIVITY\_NEW\_TASK，就会新建一个TaskRecord

作者：风风风筝  
链接：http://www.jianshu.com/p/dbea4645cbc7  
來源：简书  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

## REF

[Activity（三）栈管理](http://www.jianshu.com/p/dbea4645cbc7)

# startService的原理分析

http://blog.csdn.net/luoshengyang/article/details/6677029

# android 通信

Application-四大组件通信

## Activity通信

隐式调用，有利于降低发送者和接收者之间的耦合，它一般用在没有明确指出目标组件名称的前提下

### Intent Filter match过程

#### 使用规范

隐式 Intent

Intent intent = new Intent();

intent.setAction("com.wooyun.test"); startActivity(intent);

* a. action匹配规则：要求intent中的action 存在 且 必须和过滤规则中的其中一个相同 区分大小写；
* b. category匹配规则：系统会默认加上一个android.intent.category.DEAFAULT，所以intent中可以不存在category，但如果存在就必须匹配其中一个；
* c. data匹配规则：data由两部分组成，mimeType和URI，要求和action相似。如果没有指定URI，URI但默认值为content和file（schema）

### 源码分析

https://www.yuanmas.com/info/9ezZpgB0y6.html

主线流程：先match action, 再match data, 最后match category



# QA

1. adb install –r A.apk 执行原理？tmp目录作用，INSTALL\_FAILED\_ILLEGITIMATE\_APK各种安装报错原因，cnt的作用，为啥可以。
2. 系统目录结构是啥
3. App和系统签名的原理是什么？
4. 在sws下正常运行as，执行一次系统签名呢

framework如何更新呢？

pms包管理服务分析-PackageManagerService构造函数和包扫描过程

恢复出厂究竟干了啥

rwxrwxrwx root root 2017-09-15 10:10 bugreports -> /data/data/com.android.shell/files/bugreports

限制第三方cnt原理

updated-package 是否是判断已经升级的标志，在settings界面的时候

SystemServer工作原理，Zygote如何启动的？

alreadyDexOpted.add(frameworkDir.getPath() + "/framework-res.apk");总是失败的原因

adb install执行原理

权限方式来控制第三方安装

最好是在拷贝文件之前就处理下

机子都能root了，其实就没有必要处理版本问题了

# 参考

[Android系统应用的开发和测试](http://www.epubit.com.cn/book/onlinechapter/30644)

[Android 源代码目录结构1 - bionic](http://blog.csdn.net/kickxxx/article/details/6927272)

[Android init.rc文件解析过程详解(一)](http://blog.csdn.net/mk1111/article/details/16357327)

[Android的权限机制之—— “沙箱”机制sharedUserId和签名](http://dengzhangtao.iteye.com/blog/1989065)

# Task

Settings.Global

Wifi热点的ip的查询

Root和非root区别是什么？是否存在root也无法修改的build.prop

书籍: 2017年06月深入理解Android内核设计思想 第2版（上下册）

2017年04月 深入浅出Android源代码：基于Android 6.0和实际开发案例剖析

~~2015年11月 Android系统优化从入门到精通~~

2016年10月 深入解析Android虚拟机

2015年07月深入理解Android系统

2015年06月 构建嵌入式Android系统

深入理解android 卷1 2 3

[RK3399][Android7.1] 调试笔记: <http://blog.csdn.net/kris_fei/article/category/7318364>

Android 安全框架 -- 总概http://blog.csdn.net/blue\_rush/article/details/55045546

https://security.tencent.com/index.php/blog/msg/38