Android Source Code Guide

|  |
| --- |
| **Xu Feng** |
| **2014/8/14** |

# Android系统启动过程

<https://www.ibm.com/developerworks/cn/linux/l-linuxboot/>

Linux系统启动

BIOS 固定地址引导程序

--> stage 1 512B加载程序

--> stage2 GRUB

--> 解压内核并运行 Head.s: startup\_32

--> start\_kernel

--> setup\_arch/rest\_init/trap\_init/init\_IRQ

--> kernel\_thread --> init

main.c: init

--> do\_basic\_setup

--> sock\_init/do\_initcalls

init.c: paging\_init

-->pagetable\_init

　 physmem.c: init\_maps

startup\_32

---> decompress\_kernel

---> startup\_32 (/arch/i386/kernel/head\_32.S) <进程 0>

---> i386\_start\_kernel

---> start\_kernel

---> rest\_init

---> kernel\_thread <进程 1>

---> do\_fork

---> kernel\_init

---> sys\_access

---> init\_post

---> run\_init\_process(ramdisk\_execute\_command); = '/init'

---> init.rc / init.xx.rc

---> service adbd /sbin/adbd

---> service servicemanager /system/bin/servicemanager

---> service ril-daemon /system/bin/rild

---> service surfaceflinger /system/bin/surfaceflinger

---> service zygote /system/bin/app\_process -Xzygote /system/bin --zygote --start-system-server

---> run\_init\_process(execute\_command);

---> run\_init\_process("/sbin/init");

---> run\_init\_process("/etc/init");

---> run\_init\_process("/bin/init");

---> run\_init\_process("/bin/sh");

---> cpu\_idle

ZygoteInit -> Main Zygote启动

---> registerZygoteSocket

---> new LocalServerSocket

---> preload

---> preloadClasses

---> preloadResources

---> preloadOpenGL

---> gc

---> startSystemServer 启动SystemServer

---> Zygote.forkSystemServer

---> nativeForkSystemServer 通过JNI调用native代码

---> Dalvik\_dalvik\_system\_Zygote\_forkSystemServer

---> forkAndSpecializeCommon

---> fork()

---> SystemServer.main

---> nativeInit

---> ServerThread.initAndLoop

---> new DisplayManagerService

---> new TelephonyRegistry

---> new PackageManagerService

---> new ServiceManager

---> new BatteryService

---> new VibratorService

---> new WindowManagerService

---> new WifiService 启动WifiService

---> new ConnectivityService 启动ConnectivityService

---> handleSystemServerProcess

---> RuntimeInit.zygoteInit

---> commonInit

---> nativeZygoteInit

---> applicationInit

---> runSelectLoop

---> closeServerSocket

# Android应用启动过程



# Binder机制

# AudioFlinger

# MultiMedia框架

# ActivityManagerService之实现

* 1. 启动

SystemServer-> Main systemServer启动

---> thr.initAndLoop

---> ActivityManagerService.main 启动Service

---> thr.start()

---> AThread.run

---> new ActivityManagerService

---> ActivityManagerService.setSystemProcess

---> ServiceManager.addService(Context.ACTIVITY\_SERVICE, m, true) 注册Binder

实现详解blog：

<http://blog.csdn.net/yueliangniao1/article/details/7227165>

ActivityManagerService.startActivity

---> ActivityManagerService.startActivityAsUser

---> ActivityStackSupervisor.startActivityMayWait

---> ActivityStackSupervisor.resolveActivity 从packageManager获取launchMode，permission，screenOrientation等信息

---> ActivityStackSupervisor.startActivityLocked

---> ActivityManagerService.checkPermission 验证是否允许启动activity

---> new ActivityRecord 创建ActivityRecord

---> ActivityStackSupervisor. startActivityUncheckedLocked 判断是否要创建新task或者重用已有的task

---> ActivityStackSupervisor .adjustStackFocus 重用或创建activityStack

---> r.setTask (targetStack.createTaskRecord(…)..) 创建新TaskRecord，并将其与activityRecord关联

# SEAndroid机制

进程安全上下文的设置

系统预设置的安全context （system.img）

build/core/Makefile

---> generate-userimage-prop-dictionary 生成system\_image\_info.txt

---> echo "selinux\_fc=$(SELINUX\_FC)" >> $(1) 参数SELINUX\_FC = build/external/sepolicy/file\_contexts

---> build\_image （build/tools/releasetools/build\_image.py）

--- > mkuserimg 根据file\_contexts的规则打包img里文件context

虚拟文件系统安装，以selinux安装为例

Sepolicy创建安全策略文件

external/sepolicy/Android.mk

--- > build\_policy 参数genfs\_contexts

Init

--- > selinux\_android\_load\_policy

--- > selinux\_android\_reload\_policy

--- > mmap

--- > security\_load\_policy

--- > 加载安全策略到内核LSM中

应用程序数据文件安全上下文设置

PackageManagerService

--- > SELinuxMMAC.readInstallPolicy 启动service是加载mac\_permissions.xml文件， 将解析的内容保存到sPackageSeinfo和sSigSeinfo两个HashMap中

PackageManagerService. installNewPackageLI

--- > PackageManagerService.scanPackageLI

--- > SELinuxMMAC.assignSeinfoValue 为安装的package分配seinfo

--- > 根据签名和报名查找sPackageSeinfo和sSigSeinfo表

--- > createDataDirsLI

--- > mInstaller.install 创建安装命令

--- > execute 将安装命令发送到installd后台进程完成安装

Installd

--- > do\_install

--- > install

--- > create\_pkg\_path 在/data/data下创建于包名同名的目录

--- > selinux\_android\_setfilecon2 为目录设置安全上下文

--- > seapp\_context\_init 解析seapp\_contexts文件

--- > seapp\_context\_lookup 根据seinfo查找seapp\_contexts文件中对应的item

--- > 对第三方应用将会匹配到 user=\_app domain=untrusted\_app type=app\_data\_file levelFrom=none

--- > setfilecon 设置新的安全上下文

# Davilk虚拟机进程/线程启动机制

Dalvik虚拟机进程的创建过程

ActivityManagerService. startProcessLocked

---> Process.start 将启动进程的指令通过socket发送给zygote进程

---> startViaZygote

ZygoteConnection.runOnce

---> Zygote.forkAndSpecialize

--- > nativeForkAndSpecialize = Dalvik\_dalvik\_system\_Zygote\_forkAndSpecialize

--- > forkAndSpecializeCommon

--- > fork

--- > dvmInitAfterZygot

--- > dvmGcStartupAfterZygote 进行一次GC

--- > dvmCompilerStartup 启动JIT

应用程序进程启动

ActivityManagerService. startProcessLocked

---> Process.start 将启动进程的指令通过socket发送给zygote进程

---> startViaZygote

ZygoteConnection.runOnce

---> Zygote.forkAndSpecialize

--- > nativeForkAndSpecialize = Dalvik\_dalvik\_system\_Zygote\_forkAndSpecialize

--- > forkAndSpecializeCommon

--- > fork

--- > setSELinuxContext 设置安全上下文

--- > selinux\_android\_setcontext

--- > setcon

--- > setprocattrcon

Dalvik虚拟机线程的创建过程

Thread.start

---> VMThread.create = Dalvik\_java\_lang\_VMThread\_create

---> dvmCreateInterpThread

--- > allocThread

--- > dvmInitInterpStack

--- > dvmInitInterpreterState

--- > pthread\_create

interpThreadStart 线程入口

---> dvmCreateJNIEnv

--- > dvmChangeThreadPriority

--- > dvmCallMethod 运行JAVA层线程入口函数

--- > dvmInterpret

--- > dvmInterpretPortable

--- > dvmDetachCurrentThread 线程结束，进行清理工作

能同时执行C/C++代码和Java代码的Native线程的创建过程

Thread::run

---> createThreadEtc

---> androidCreateThreadEtc

--- > gCreateThreadFn = javaCreateThreadEtc

--- > androidCreateRawThreadEtc

--- > pthread\_create

--- > AndroidRuntime::javaThreadShell

--- > javaAttachThread

--- > attachThread

--- > dvmAttachCurrentThread

--- > allocThread

--- > dvmCreateJNIEnv

--- > dvmCallMethod 执行JAVA代码

--- > start(userData) = Thread::\_threadLoop

只执行C/C++代码的Native线程的创建过程

Thread::run

---> androidCreateRawThreadEtc

---> pthread\_create

--- > Thread::\_threadLoop

Dalvik虚拟机线程的创建过程

Thread.start

---> VMThread.create = Dalvik\_java\_lang\_VMThread\_create

---> dvmCreateInterpThread

--- > allocThread

--- > dvmInitInterpStack

--- > dvmInitInterpreterState

--- > pthread\_create

interpThreadStart 线程入口

---> dvmCreateJNIEnv

--- > dvmChangeThreadPriority

--- > dvmCallMethod 运行JAVA层线程入口函数

--- > dvmInterpret

--- > dvmInterpretPortable

--- > dvmDetachCurrentThread 线程结束，进行清理工作