**Android从开机到调起home桌面启动流程**

以Android4.2源码了解Android手机从按下电源到桌面启动起来的流程。

**一、启动Linux内核（按下电源键后）**

**加载BootLoader**

**作用：**

* BootLoader 是嵌入式设备在加电后执行的第一段代码
* 这里会完成CPU和相关硬件的初始化，将操作系统的镜像或者要加载的应用程序加载到内存中
* 通过BootLoader 启动Linux内核 (加载各种设备的驱动,初始化数据结构,并且启动第一个用户级的进程 init进程 init进程启动的代码 init.c )

**二、启动Android**

**1、init进程的启动**

init进程是c进程init.c，位于（system/core/init/init.c）中

init.c的main方法是程序入口，在main方法中，先创建一系列的目录并将其挂载起来，之后就会解析配置文件init.rc

**int main(int argc, char \*\*argv){**

**...**

**//创建目录并挂载**

**mkdir("/dev", 0755);**

**mkdir("/proc", 0755);**

**mkdir("/sys", 0755);**

**mount("tmpfs", "/dev", "tmpfs", MS\_NOSUID, "mode=0755");**

**mkdir("/dev/pts", 0755);**

**mkdir("/dev/socket", 0755);**

**mount("devpts", "/dev/pts", "devpts", 0, NULL);**

**mount("proc", "/proc", "proc", 0, NULL);**

**mount("sysfs", "/sys", "sysfs", 0, NULL);**

**...**

**//关键代码**

**//解析 init.rc文件 init.rc 实际上是一个配置文件 里面有大量的linux命令**

**//解析之后会执行这些命令**

**init\_parse\_config\_file("/init.rc");**

**...**

**}**

**总结：**

**启动的时候 第一个进程 就是init进程 对应的代码 init.c**

**init.c 创建并且加载了一些关键目录 解析init.rc文件**

**init.rc文件中就包含了跟zygote相关的启动命令**

**init.c的main方法 解析了init.rc 启动了zygote进程**

**2、Zygote进程启动**

**a、Zygote进程介绍：**

在android系统中，Zygote进程所有的应用程序进程的**母进程**，用来运行关键系统服务的System进程 都是由 Zygote进程负责创建的，所以我们管它叫孵化器进程(直接翻译是受精卵的意思)。Zygote进程是通过**复制（fork）自身**的方式来创建系统进程和应用程序进程的。由于zygote进程在创建时会在内部创建一个虚拟机的实例，所以复制自身的时候就会复制出一个新的虚拟机实例来。那么通过这种方式 就可以很快的为新的进程创建一个虚拟机。

**b、启动过程**

* 在init.rc配置文件中会有很多配置，最主要的是准备启动一个Zygote进程。

**// service zygote 说明是以服务的形式启动zygote进程**

**// /system/bin/app\_process 指定了 zygote进程对应的文件的位置**

**// -Xzygote /system/bin --zygote --start-system-server 执行命令所带的参数**

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

**class main**

**socket zygote stream 660 root system**

**onrestart write /sys/android\_power/request\_state wake**

**onrestart write /sys/power/state on**

**onrestart restart media**

**onrestart restart netd**

* 此时就会执行到app\_process目录下找到 /system/bin/app\_process/App\_main.cppc++代码中的main方法。

**int main(int argc, const char\* const argv[]{**

**...**

**//创建了一个AppRuntime zygote进程就是由这个runtime 创建的**

**AppRuntime runtime;**

**...**

**//遍历执行main方法时传入的字符串数组**

**while (i < argc) {**

**const char\* arg = argv[i++];**

**if (!parentDir) {**

**parentDir = arg;**

**//如果参数中包含了 --zygote**

**} else if (strcmp(arg, "--zygote") == 0) {**

**//将zygote 赋值为true**

**zygote = true;**

**//给进程的别名赋值**

**niceName = "zygote";**

**//如果参数中包含了 "--start-system-server"**

**} else if (strcmp(arg, "--start-system-server") == 0) {**

**//将startSystemServer赋值为true**

**startSystemServer = true;**

**} else if (strcmp(arg, "--application") == 0) {**

**application = true;**

**} else if (strncmp(arg, "--nice-name=", 12) == 0) {**

**niceName = arg + 12;**

**} else {**

**className = arg;**

**break;**

**}**

**}**

**//遍历完成后执行此处**

**if (niceName && \*niceName) {**

**setArgv0(argv0, niceName);**

**//设置当前进程名字为zygote**

**set\_process\_name(niceName);**

**}**

**runtime.mParentDir = parentDir;**

**//在启动zygote进程是 这个zygote 为true**

**//此时将调用runtime.start方法启动zygote进程，**

**if (zygote) {**

**//在调用 start函数的时候 传递的两个参数就是**

**//"com.android.internal.os.ZygoteInit"和"start-system-server"**

**runtime.start("com.android.internal.os.ZygoteInit",**

**startSystemServer ? "start-system-server" : "");**

**} else if (className) {**

**start-system-server**

**}**

**...**

**...**

**}**

* **AndroidRuntime.cpp**中start方法中，首先现将c++的字符串参数转成java的String类型，保存在java的String数组中，然后以反射的方式把**ZygoteInit.java**的main方法启动起来。

**void AndroidRuntime::start(const char\* className, const char\* options){**

**.....**

**//region 此处创建了一个java虚拟机**

**/\* start the virtual machine\*/**

**JNIEnv env;**

**if (startVm(&mJavaVM, &env) != 0) {**

**return;**

**}**

**onVmCreated(env);**

**//endregion**

**//注册jni函数表**

**/\* Register android functions.\*/**

**if (startReg(env) < 0) {**

**ALOGE("Unable to register all android natives\n");**

**return;**

**}**

**//region 将c++中的字符串转成java中的String数组**

**/\*把调用start函数的两个参数 转化成jstring类型**

**\*这两个参数就是 const char\* className "com.android.internal.os.ZygoteInit"**

**\*const char\* options start-system-server**

**\*/**

**jclass stringClass;**

**jobjectArray strArray;**

**jstring classNameStr;**

**jstring optionsStr;**

**//找到java的String字节码**

**stringClass = env->FindClass("java/lang/String");**

**assert(stringClass != NULL);**

**//创建了java的String数组**

**strArray = env->NewObjectArray(2, stringClass, NULL);**

**assert(strArray != NULL);**

**//把className（com.android.internal.os.ZygoteInit）字符串转成java的String类型**

**classNameStr = env->NewStringUTF(className);**

**assert(classNameStr != NULL);**

**//把转后的java字符串classNameStr放到java的String数组的0位置**

**env->SetObjectArrayElement(strArray, 0, classNameStr);**

**//把options（start-system-server）字符串转成java的String类型**

**optionsStr = env->NewStringUTF(options);**

**//把转后的java字符串optionsStr放到java的String数组的1位置**

**env->SetObjectArrayElement(strArray, 1, optionsStr);**

**//endregion**

**char\* slashClassName = toSlashClassName(className);**

**jclass startClass = env->FindClass(slashClassName);**

**if (startClass == NULL) {**

**ALOGE("JavaVM unable to locate class '%s'\n", slashClassName);**

**/\* keep going \*/**

**} else {**

**//通过反射的方式找到了 com.android.internal.os.ZygoteInit 字节码 找到Main方法**

**jmethodID startMeth = env->GetStaticMethodID(startClass, "main",**

**"([Ljava/lang/String;)V");**

**if (startMeth == NULL) {**

**ALOGE("JavaVM unable to find main() in '%s'\n", className);**

**/\* keep going \*/**

**} else {**

**//调用com.android.internal.os.ZygoteInit的main方法，并将java类型的String数组传入**

**env->CallStaticVoidMethod(startClass, startMeth, strArray);**

**....**

**}**

**....**

**}**

**}**

* 执行ZygoteInit.main方法，此时就从C++的世界到了java的世界。

在main方法中，创建了一个ServerSocket服务端，用来等待ActivityMangerService请求zygote进程创建新的应用程序进程；同时预加载资源（.class文件，图片，样式，系统资源等）；然后开启一个系统服务systemserver进程。

**public static void main(String argv[]) {**

**try {**

**// Start profiling the zygote initialization.**

**SamplingProfilerIntegration.start();**

**//创建一个socket服务端(ServerSocket实例) 用来等待 ActivityManagerService请求 //zygote进程创建新的应用程序进程**

**registerZygoteSocket();**

**EventLog.writeEvent(LOG\_BOOT\_PROGRESS\_PRELOAD\_START,**

**SystemClock.uptimeMillis());**

**//预加载资源 .class文件 图片 样式 ..系统资源**

**preload();**

**EventLog.writeEvent(LOG\_BOOT\_PROGRESS\_PRELOAD\_END,**

**SystemClock.uptimeMillis());**

**// Finish profiling the zygote initialization.**

**SamplingProfilerIntegration.writeZygoteSnapshot();**

**// Do an initial gc to clean up after startup**

**gc();**

**// If requested, start system server directly from Zygote**

**//ZYGOTE\_FORK\_MODE = false 所以执行的是 runSelectLoopMode();**

**if (argv.length != 2) {**

**throw new RuntimeException(argv[0] + USAGE\_STRING);**

**}**

**if (argv[1].equals("start-system-server")) {**

**//启动systemserver进程**

**startSystemServer();**

**} else if (!argv[1].equals("")) {**

**throw new RuntimeException(argv[0] + USAGE\_STRING);**

**}**

**Log.i(TAG, "Accepting command socket connections");**

**if (ZYGOTE\_FORK\_MODE) {**

**runForkMode();**

**} else {**

**//这个方法中创建了一个死循环,应用就停在这里 等待ActivityManagerService**

**//的socket客户端连接到 serverSocket, 请求创建新的进程**

**runSelectLoopMode();**

**}**

**//当执行到这里时说明上面的死循环执行完了，不是应用挂了 就是关机了**

**closeServerSocket();**

**} catch (MethodAndArgsCaller caller) {**

**caller.run();**

**} catch (RuntimeException ex) {**

**Log.e(TAG, "Zygote died with exception", ex);**

**closeServerSocket();**

**throw ex;**

**}**

**}**

runSelectLoopMode()方法中开启一个死循环，等待AMS这个socket客户端发送消息，请求创建新进程

**private static void runSelectLoopMode() throw MethodAndArgsCaller{**

**....**

**//这里开启一个死循环，等待AMS给它发消息**

**while(true){**

**...**

**//如果接收到消息，则拿到一个ZygoteConnection对象**

**if(index <0){**

**throw new RuntimeException("Error in select");**

**}else if (index == 0){**

**ZygoteConnection newPeer = acceptCommandPeer();**

**peers.add(newPeer);**

**fds.add(newPeer.getFileDesciptor());**

**}else{**

**...**

**}**

**...**

**}**

**...**

**}**

至此Zygote进程启动完成，之后每开启一个新应用时，Zygote进程都是以复制自身的形式创建新应用的进程。

**c、Zygote进程总结**

① 启动了一个java虚拟机  
②注册了socket服务端 如果启动起一个新的应用activity manager service 就是通过这个socket服务  
端通知zygote进程复制新的进程  
③ preload()预加载资源 包括用到的字节码对象和 系统的各种资源(xml 图片 声音 …)  
④ 创建了 systemserver进程  
⑤跑起一个死循环 等待ams连接

**3、systemserver进程的开启**

**a、SystemServer进程开启**

systemServer进程是Zygote进程分叉的第一个进程

zygote进程通过调用startSystemServer()方法启动systemServer进程，此方法内，给启动system server进程配置参数，然后分叉出子进程启动SystemServer进程，并对系统服务进程进行处理

**private static boolean startSystemServer()**

**throws MethodAndArgsCaller, RuntimeException {**

**//启动 system server进程时要传的参数**

**String args[] = {**

**"--setuid=1000",//系统服务的id**

**"--setgid=1000",**

**"--setgroups=1001,1002,1003,1004,1005,1006,1007,1008,1009,1010,1018,3001,3002,3003,3006,3007",**

**"--capabilities=130104352,130104352",**

**"--runtime-init",**

**"--nice-name=system\_server",**

**"com.android.server.SystemServer",//路径**

**};**

**ZygoteConnection.Arguments parsedArgs = null;**

**int pid;**

**try {**

**//解析上述参数**

**parsedArgs = new ZygoteConnection.Arguments(args);**

**ZygoteConnection.applyDebuggerSystemProperty(parsedArgs);**

**ZygoteConnection.applyInvokeWithSystemProperty(parsedArgs);**

**// 在Zygote进程中调用fork()函数 分叉出子进程在子进程中启动SystemServer进程**

**//pid进程编码**

**pid = Zygote.forkSystemServer(**

**parsedArgs.uid, parsedArgs.gid,**

**parsedArgs.gids,**

**parsedArgs.debugFlags,**

**null,**

**parsedArgs.permittedCapabilities,**

**parsedArgs.effectiveCapabilities);**

**} catch (IllegalArgumentException ex) {**

**throw new RuntimeException(ex);**

**}**

**/\* For child process \*/**

**//如果pid==0 说明当前的代码是执行在新的进程中**

**//这个进程就是system\_server进程**

**if (pid == 0) {**

**//通过这个方法 处理系统服务进程**

**handleSystemServerProcess(parsedArgs);**

**}**

**return true;**

**}**

因为新fock出的系统服务进程是zygote进程的复制品，进程内包含zygote进程的所有资源和配置，所以在handleSystemProcess（）函数中对其进程处理。

**private static void handleSystemServerProcess(**

**ZygoteConnection.Arguments parsedArgs)**

**throws ZygoteInit.MethodAndArgsCaller {**

**//关闭serversocket (socket服务端)**

**//关闭socket服务端 由于当前进程是由zygote进程分叉出来的 所以包含跟zygote相同的资源**

**//也包含了zygote的socket服务端 当前进程不需要这个而服务端 所以把socket服务端关闭**

**closeServerSocket();**

**// set umask to 0077 so new files and directories will default to owner-on**

**ly permissions.Libcore.os.umask(S\_IRWXG | S\_IRWXO);**

**if (parsedArgs.niceName != null) {**

**//给当前进程起名字 system\_server**

**Process.setArgV0(parsedArgs.niceName);**

**}**

**if (parsedArgs.invokeWith != null) {**

**WrapperInit.execApplication(parsedArgs.invokeWith,parsedArgs.niceName, parsedArgs.targetSdkVersion, null, parsedArgs.remainingArgs);**

**} else {**

**/\***

**\* Pass the remaining arguments to SystemServer.**

**\*/**

**//启动SystemServer**

**//通过zygoteInite方法 处理systemserver相关逻辑**

**RuntimeInit.zygoteInit(parsedArgs.targetSdkVersion, parsedArgs.remainingArgs);**

**}**

**/\* should never reach here \*/**

**}**

**c、RunTimeInit类处理SystemServer相关逻辑**

通过zygoteInit()方法 调用到了->applicationInit()方法调用->invokeStaticMain(args.startClass, args.startArgs)方法，此方法调用**SystemServer.java**的静态main()方法；

**private static void invokeStaticMain(String className, String[] argv)**

**throws ZygoteInit.MethodAndArgsCaller {**

**Class<?> c1;**

**try{**

**//通过传进来的className参数获取到对应的字节码对象,className就是开始系统服务进程是设置的路径**

**c1 = Class.forName(className);**

**}catch(ClassNotFoundException ex){**

**throw new RuntimeException("Missing class when invoking static main " + className, ex);**

**}**

**Method m ;**

**try{**

**//通过反射的方式调用了 SystemServer.java的main方法**

**m = c1.getMethod("main" , new Class[]{String[].class});**

**}catch(NoSuchMethodException ex){**

**throw new RuntimeException("Missing class when invoking static main " + className, ex);**

**}catch(SecurityException ex){**

**throw new RuntimeException("Problem getting static main on " + className, ex);**

**}**

**....**

**}**

**d、SystemServer的main方法**

通过jni调用到了**System\_server.cpp**的 init1()方法

**public static void main(String[] args){**

**....**

**//首先设置了一下系统时间**

**SystemClock.setCurrentTimeMillis(EARLIEST\_SUPPORTED\_TIME);**

**....**

**//main方法关键代码**

**System.loadLibrary("android\_servers");**

**//init1是本地函数 它的实现在android\_server\_SystemServer.cpp**

**init1(args);**

**....**

**}**

本地方法android\_server\_SystemServer\_init1()中又通过jni 反射的方式调回到**SystemServer.java** 的init2()方法

**android\_server\_SystemServer.cpp**的init1()方法

**static void android\_server\_SystemServer\_init1(JNIEnv\* env, jobject clazz){**

**//这个方法在c中开启了surfaceFlinger(跟显示驱动相关)**

**//开启了一个SensorService（传感器相关 ）**

**//之后回调SystemServer.java中的init2方法**

**system\_init();**

**}**

system\_init()方法中首先开始了一些系统服务，包括传感器和显示驱动有关的服务等等，然后回调**SystemServer.java**中的init2()方法

**SystemServer.java**的init2()方法，这里主要就是开启一个线程

**public static final void init2() {**

**Slog.i(TAG, "Entered the Android system server!");**

**Thread thr = new ServerThread();4. thr.setName("android.server.ServerThread");**

**thr.start();**

**}**

**ServerThread.java**的run()方法中

创建了大量的系统服务， 并且以k-v的形式保存到了ServiceManager中，在所有服务都加载之后，就调用**ActivityMangerService.java**的systemReady()方法，判断当前有没有activity在运行，没有的话就启动Launcher

**@Override**

**public void run() {**

**..............**

**Installer installer = null;**

**AccountManagerService accountManager = null;**

**ContentService contentService = null;**

**LightsService lights = null;**

**PowerManagerService power = null;**

**DisplayManagerService display = null;**

**BatteryService battery = null;**

**VibratorService vibrator = null;**

**AlarmManagerService alarm = null;**

**MountService mountService = null;**

**NetworkManagementService networkManagement = null;**

**NetworkStatsService networkStats = null;**

**NetworkPolicyManagerService networkPolicy = null;**

**ConnectivityService connectivity = null;**

**WifiP2pService wifiP2p = null;**

**WifiService wifi = null;**

**NsdService serviceDiscovery= null;**

**IPackageManager pm = null;**

**Context context = null;**

**WindowManagerService wm = null;**

**BluetoothManagerService bluetooth = null;**

**DockObserver dock = null;**

**UsbService usb = null;**

**SerialService serial = null;**

**TwilightService twilight = null;**

**UiModeManagerService uiMode = null;**

**RecognitionManagerService recognition = null;**

**ThrottleService throttle = null;**

**NetworkTimeUpdateService networkTimeUpdater = nul**

**Slog.i(TAG, "Power Manager");**

**power = new PowerManagerService();**

**ServiceManager.addService(Context.POWER\_SERVICE, power);**

**Slog.i(TAG, "Activity Manager");**

**context = ActivityManagerService.main(factoryTest);**

**Slog.i(TAG, "Display Manager");**

**display = new DisplayManagerService(context, wmHandler, uiHandler);**

**ServiceManager.addService(Context.DISPLAY\_SERVICE, display, true);**

**Slog.i(TAG, "Telephony Registry");**

**telephonyRegistry = new TelephonyRegistry(context);**

**ServiceManager.addService("telephony.registry", telephonyRegistry);**

**Slog.i(TAG, "Scheduling Policy");**

**ServiceManager.addService(Context.SCHEDULING\_POLICY\_SERVICE,**

**new SchedulingPolicyService());**

**.......**

**//当所有的系统服务都加载并且添加到 ServiceManager中，ServiceManager.addService方法**

**//ActivityManagerService.java将调用systemReady()方法**

**ActivityManagerService.self().systemReady(new Runnable() {**

**public void run() {**

**....**

**}**

**});**

**}**

systemReady()中调用了 mMainStack.resumeTopActivityLocked(null);

**public void systemReady(final Runnable goingCallBack){**

**....**

**// 检查任务栈中是否有activity如果没有就启动luncher**

**mMainStack.resumeTopActivityLocked(null);**

**.....**

**}**

这方法resumeTopActivityLocked(null)检查了当前有没有activity在运行 如果没有就把launcher启动起来

**final boolean resumeTopActivityLocked(ActivityRecord prev, Bundle options) {**

**// Find the first activity that is not finishing.**

**//查找是否有没有没关闭的activity**

**ActivityRecord next = topRunningActivityLocked(null);**

**// Remember how we'll process this pause/resume situation, and ensure**

**// that the state is reset however we wind up proceeding.**

**final boolean userLeaving = mUserLeaving;**

**mUserLeaving = false;**

**//没有查找到，也就是没开启的activity，即第一次开启应用**

**if (next == null) {**

**// There are no more activities! Let's just start up the**

**// Launcher...**

**if (mMainStack) {**

**ActivityOptions.abort(options);**

**//Launcher就被启动起来**

**return mService.startHomeActivityLocked(mCurrentUser);**

**}**

**}**

**}**

至此Android系统开启到桌面。

**e、SystemServer进程启动总结**

①zygote分叉出来的第一个进程 systemserver进程  
②关闭了socket服务端 给当前线程起名字 system\_server  
③通过jni 打开了 surfaceFlinger 打开了传感器的服务  
④创建了 ServerThread子线程 并且执行了run方法，在run方法中注册了所有的系统服务  
⑤创建服务对象 通过ServiceManger.addService方法 保存到ServiceManger中  
⑥当所有的服务都加载之后，ActivityMangerService.systemready方法，在这个方法中，判断当前有没有activity在运行 如果没有启动launcher

**三、总结**

**一、init.c**

启动的时候 第一个进程 就是init进程 对应的代码 init.c  
init.c 创建并且加载了一些关键目录 解析init.rc文件  
init.rc文件中就包含了跟zygote相关的启动命令  
init.c的main方法 解析了init.rc 启动了zygote进程

**二、zygote的启动**

zygote启动过程做的事儿  
① 启动了一个java虚拟机  
②注册了socket服务端 如果启动起一个新的应用activity manager service 就是通过这个socket服务  
端通知zygote进程复制新的进程  
③ preload()预加载资源 包括用到的字节码对象和 系统的各种资源(xml 图片 声音 …)  
④ 创建了 systemserver进程  
⑤跑起一个死循环 等待ams连接

**三、systemserver进程**

①zygote分叉出来的第一个进程 systemserver进程  
②关闭了socket服务端 给当前线程起名字 system\_server  
③通过jni 打开了 surfaceFlinger 打开了传感器的服务  
④创建了 ServerThread子线程 并且执行了run方法，在run方法中注册了所有的系统服务  
⑤创建服务对象 通过ServiceManger.addService方法 保存到ServiceManger中  
⑥当所有的服务都加载之后，ActivityMangerService.systemready方法，在这个方法中，判断当前有没有activity在运行 如果没有启动launcher

**目录**

1. [一、启动Linux内核（按下电源键后）](https://blog.csdn.net/wwp9527/article/details/83065492#t0)
   1. [加载BootLoader](https://blog.csdn.net/wwp9527/article/details/83065492#t1)
2. [二、启动Android](https://blog.csdn.net/wwp9527/article/details/83065492#t2)
   1. [1、init进程的启动](https://blog.csdn.net/wwp9527/article/details/83065492#t3)
   2. [2、Zygote进程启动](https://blog.csdn.net/wwp9527/article/details/83065492#t4)
      1. [a、Zygote进程介绍：](https://blog.csdn.net/wwp9527/article/details/83065492#t5)
      2. [b、启动过程](https://blog.csdn.net/wwp9527/article/details/83065492#t6)
      3. [c、Zygote进程总结](https://blog.csdn.net/wwp9527/article/details/83065492#t7)
   3. [3、systemserver进程的开启](https://blog.csdn.net/wwp9527/article/details/83065492#t8)
      1. [a、SystemServer进程开启](https://blog.csdn.net/wwp9527/article/details/83065492#t9)
      2. [b、handleSystemProcess（）处理系统服务进程配置](https://blog.csdn.net/wwp9527/article/details/83065492#t10)
      3. [c、RunTimeInit类处理SystemServer相关逻辑](https://blog.csdn.net/wwp9527/article/details/83065492#t11)
      4. [d、SystemServer的main方法](https://blog.csdn.net/wwp9527/article/details/83065492#t12)
      5. [e、SystemServer进程启动总结](https://blog.csdn.net/wwp9527/article/details/83065492#t13)
3. [三、总结](https://blog.csdn.net/wwp9527/article/details/83065492#t14)
   1. [一、init.c](https://blog.csdn.net/wwp9527/article/details/83065492#t15)
   2. [二、zygote的启动](https://blog.csdn.net/wwp9527/article/details/83065492#t16)
   3. [三、systemserver进程](https://blog.csdn.net/wwp9527/article/details/83065492#t17)