Android应用程序进程管理

罗升阳

http://weibo.com/shengyangluo

http://blog.csdn.net/luoshengyang

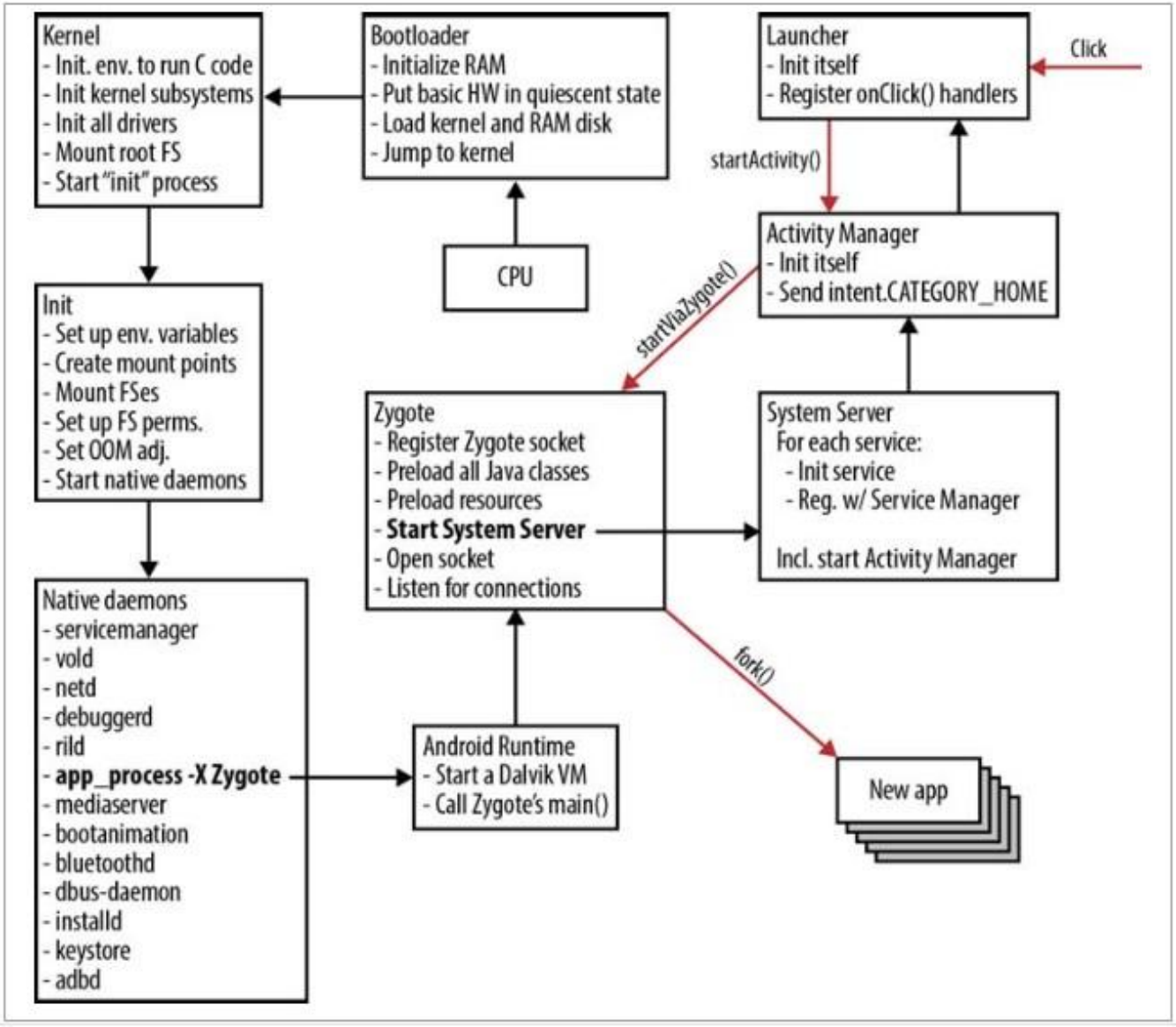
About Me

- 《老罗的Android之旅》博客作者
- 《Android系统源代码情景分析》书籍作者
- 博客：http://blog.csdn.net/Luoshengyang
- 微博：http://weibo.com/shengyangluo

Agenda

- Android系统启动概述
- Zygote进程启动过程分析
- System Server进程启动过程分析
- Android应用程序进程启动过程分析
- Android应用程序进程回收机制

Android系统启动概述



Zygote进程启动过程分析

- Zygote进程由Init进程启动

```
service zygote /system/bin/app_process -Xzygote /system/bin --zygote --start-sys
tem-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
```

  - 加载文件：/system/app_process
  - --start-system-server：启动System Server进程
  - 创建名称为zygote的socket：用来和ActivityManagerService通信

Zygote进程启动过程分析(续)

- app_process

```
class AppRuntime : public AndroidRuntime
{
    ......
};


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

    AppRuntime runtime;

    if (zygote) {
        runtime.start("com.android.internal.os.ZygoteInit",
                startSystemServer ? "start-system-server" : "");
    }

    ......
}
```

Zygote进程启动过程分析(续)

- AndroidRuntime::start

```
void AndroidRuntime::start(const char* className, const char* options)
{
    ......
    /* start the virtual machine */
    JNIEnv* env;
    if (startVm(&mJavaVM, &env) != 0) {
        return;
    }
    ......
    /*
     * Register android functions.
     */
    if (startReg(env) < 0) {
        ALOGE("Unable to register all android natives\n");
        return;
    }
    ......
    /*
     * Start VM.  This thread becomes the main thread of the VM, and will
     * not return until the VM exits.
     */
    char* slashClassName = toSlashClassName(className);
    jclass startClass = env->FindClass(slashClassName);
    if (startClass == NULL) {
        ......
    } else {
        jmethodID startMeth = env->GetStaticMethodID(startClass, "main",
            "([Ljava/lang/String;)V");
        ......
    }
    ......
}
```

Zygote进程启动过程分析(续)

- 启动Dalvik虚拟机
  - 创建一个Dalvik虚拟机实例
  - 加载Java核心类及其JNI方法
  - 初始化主线程的JNI环境
- 加载部分Android核心类及其JNI方法
  - android.os.*
  - android.graphics.*
  - android.opengl.*
  - android.hardware.*
  - android.media.*
  - ……

Zygote进程启动过程分析(续)

- ZygoteInit.main

```
public class ZygoteInit {
    ......
    public static void main(String argv[]) {
      try {
          ......
          registerZygoteSocket();
          ......
          preload();
          ......
          if (argv[1].equals("start-system-server")) {
              startSystemServer();
          }
          ......
          if (ZYGOTE_FORK_MODE) {
              runForkMode();
          } else {
              runSelectLoopMode();
          }
          ......
      } catch (MethodAndArgsCaller caller) {
          ......
      } catch (RuntimeException ex) {
          ......
      }
    }
    ......
}
```

Zygote进程启动过程分析(续)

- Preload Classes
  - 参考frameworks/base/preloaded-classes文件
    - android.accounts.*
    - android.app.*
    - android.view.*
    - ……

```
# Classes which are preloaded by com.android.internal.os.ZygoteInit.
# Automatically generated by frameworks/base/tools/preload/WritePreloadedClassFi
le.java.
# MIN_LOAD_TIME_MICROS=1250
# MIN_PROCESSES=10
android.R$styleable
android.accounts.Account
android.accounts.Account$1
android.accounts.AccountManager
android.accounts.AccountManager$13
android.accounts.AccountManager$6
android.accounts.AccountManager$AmsTask
android.accounts.AccountManager$AmsTask$1
android.accounts.AccountManager$AmsTask$Response
android.accounts.AccountManagerFuture
android.accounts.IAccountManager
android.accounts.IAccountManager$Stub
android.accounts.IAccountManager$Stub$Proxy
android.accounts.IAccountManagerResponse
android.accounts.IAccountManagerResponse$Stub
android.accounts.OnAccountsUpdateListener
```

Zygote进程启动过程分析(续)

- Preload Drawables
  - 参考frameworks/base/core/res/res/values$/arrays.xml文件
    - @drawable/toast_frame_holo
    - @drawable/btn_check_on_pressed_holo_light
    - @drawable/btn_check_on_pressed_holo_dark
    - ……

```
<resources xmlns:xliff="urn:oasis:names:tc:xliff:document:1.2">

    <!-- Do not translate. These are all of the drawable resources that should b
e preloaded by
         the zygote process before it starts forking application processes. -->
    <array name="preloaded_drawables">
        <item>@drawable/toast_frame_holo</item>
        <item>@drawable/btn_check_on_pressed_holo_light</item>
        <item>@drawable/btn_check_on_pressed_holo_dark</item>
        <item>@drawable/btn_check_on_holo_light</item>
        <item>@drawable/btn_check_on_holo_dark</item>
        <item>@drawable/btn_check_on_focused_holo_light</item>
        <item>@drawable/btn_check_on_focused_holo_dark</item>
        <item>@drawable/btn_check_on_disabled_holo_light</item>
        <item>@drawable/btn_check_on_disabled_holo_dark</item>
        <item>@drawable/btn_check_on_disabled_focused_holo_light</item>
        <item>@drawable/btn_check_on_disabled_focused_holo_dark</item>
        <item>@drawable/btn_check_off_pressed_holo_light</item>
        <item>@drawable/btn_check_off_pressed_holo_dark</item>
        <item>@drawable/btn_check_off_holo_light</item>
        <item>@drawable/btn_check_off_holo_dark</item>
```

Zygote进程启动过程分析(续)

- Preload Color State List
  - 参考frameworks/base/core/res/res/values$/arrays.xml文件
    - @color/primary_text_dark
    - @color/primary_text_dark_disable_only
    - @color/primary_text_dark_nodisable
    - ……

```
    <!-- Do not translate. These are all of the color state list resources that
should be
         preloaded by the zygote process before it starts forking application pr
ocesses. -->
    <array name="preloaded_color_state_lists">
        <item>@color/primary_text_dark</item>
        <item>@color/primary_text_dark_disable_only</item>
        <item>@color/primary_text_dark_nodisable</item>
        <item>@color/primary_text_disable_only_holo_dark</item>
        <item>@color/primary_text_disable_only_holo_light</item>
        <item>@color/primary_text_holo_dark</item>
        <item>@color/primary_text_holo_light</item>
        <item>@color/primary_text_light</item>
        <item>@color/primary_text_light_disable_only</item>
        <item>@color/primary_text_light_nodisable</item>
        <item>@color/primary_text_nodisable_holo_dark</item>
        <item>@color/primary_text_nodisable_holo_light</item>
        <item>@color/secondary_text_dark</item>
        <item>@color/secondary_text_dark_nodisable</item>
        <item>@color/secondary_text_holo_dark</item>
```

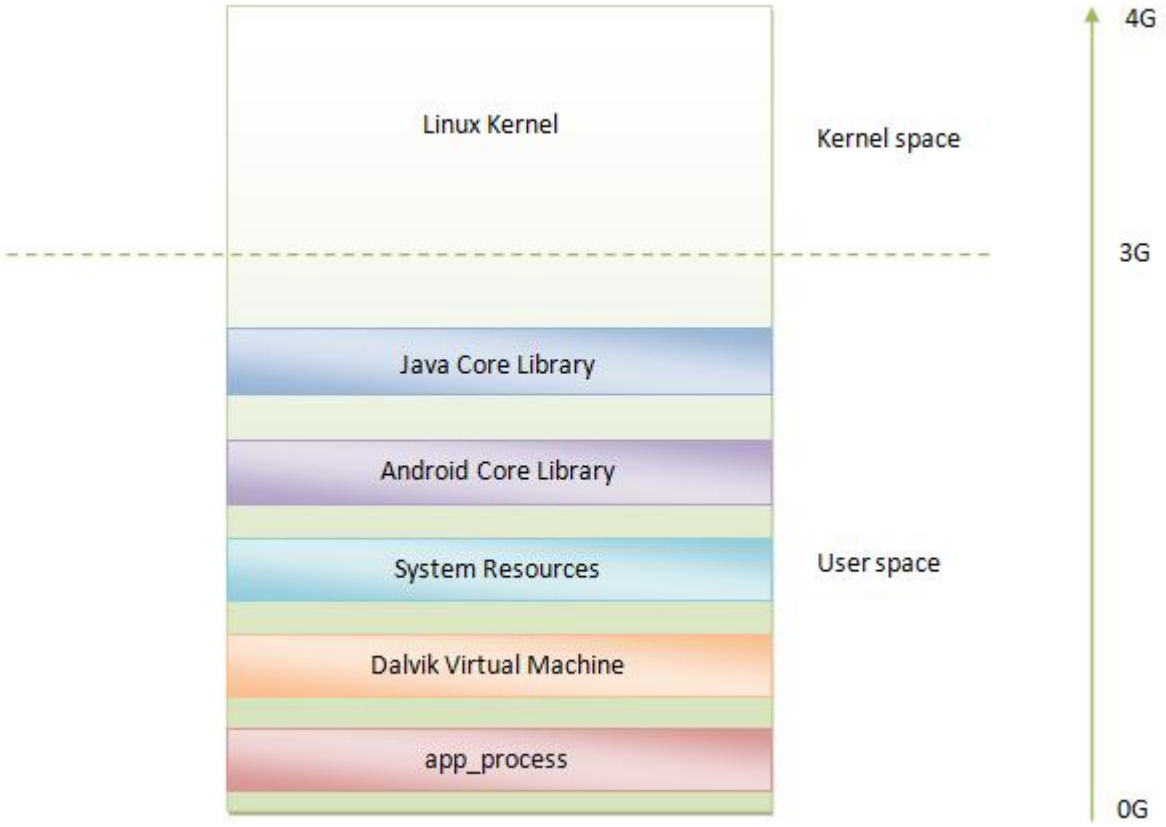
Zygote进程启动过程分析(续)

- runSelectLoopMode

```
public class ZygoteInit {
    ......
    private static void runSelectLoopMode() throws MethodAndArgsCaller {
        ArrayList<FileDescriptor> fds = new ArrayList();
        ArrayList<ZygoteConnection> peers = new ArrayList();
        FileDescriptor[] fdArray = new FileDescriptor[4];

        fds.add(sServerSocket.getFileDescriptor());
        peers.add(null);

        while (true) {
            int index;
            ......
            try {
                fdArray = fds.toArray(fdArray);
                index = selectReadable(fdArray);
            } catch (IOException ex) {
            }

            if (index < 0) {
            } else if (index == 0) {
                ZygoteConnection newPeer = acceptCommandPeer();
                peers.add(newPeer);
                fds.add(newPeer.getFileDesciptor());
            } else {
                boolean done;
                done = peers.get(index).runOnce();

                if (done) {
                    peers.remove(index);
                    fds.remove(index);
                }
            }
        }
    }
    ......
}
```

Zygote进程启动过程分析(续)

- Zygote进程启动完成后的地址空间



System Server进程启动过程分析

- Zygote在启动的过程中创建System Server进程

```
public class ZygoteInit {
    ......
    public static void main(String argv[]) {
      try {
          ......
          registerZygoteSocket();
          ......
          preload();
          ......
          if (argv[1].equals("start-system-server")) {
              startSystemServer();
          }
          ......
          if (ZYGOTE_FORK_MODE) {
              runForkMode();
          } else {
              runSelectLoopMode();
          }
          ......
      } catch (MethodAndArgsCaller caller) {
          ......
      } catch (RuntimeException ex) {
          ......
      }
    }
    ......
}
```

System Server进程启动过程分析(续)

- startSystemServer

```
public class ZygoteInit {
    ------
    private static boolean startSystemServer()
        /* Hardcoded command line to start the system server */
        String args[] = {
            "--setuid=1000",
            "--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",
        };
        ZygoteConnection.Arguments parsedArgs = null;

        int pid;

        try {
            parsedArgs = new ZygoteConnection.Arguments(args);
            ------
            /* Request to fork the system server process */
            pid = Zygote.forkSystemServer(
                    parsedArgs.uid, parsedArgs.gid,
                    parsedArgs.gids,
                    parsedArgs.debugFlags,
                    null,
                    parsedArgs.permittedCapabilities,
                    parsedArgs.effectiveCapabilities);
        } catch (IllegalArgumentException ex) {
        }

        /* For child process */
        if (pid == 0) {
            handleSystemServerProcess(parsedArgs);
        }

        return true;
    }
    ------
}
```

System Server进程启动过程分析(续)

- handleSystemServerProcess

```
public class ZygoteInit {
    ......
    private static void handleSystemServerProcess(
            ZygoteConnection.Arguments parsedArgs)
            throws ZygoteInit.MethodAndArgsCaller {
        closeServerSocket();

        // set umask to 0077 so new files and directories will default to owner-only permissions.
        Libcore.os.umask(S_IRWXG | S_IRWXO);

        if (parsedArgs.niceName != null) {
            Process.setArgV0(parsedArgs.niceName);
        }
        if (parsedArgs.invokeWith != null) {
        } else {
            /*
             * Pass the remaining arguments to SystemServer.
             */
            RuntimeInit.zygoteInit(parsedArgs.targetSdkVersion, parsedArgs.remainingArgs);
        }

        /* should never reach here */
    }
    ......
}
```

System Server进程启动过程分析(续)

- RuntimeInit.zygoteInit

```
public class RuntimeInit {
    ......
    public static final void zygoteInit(int targetSdkVersion, String[] argv)
            throws ZygoteInit.MethodAndArgsCaller {
        ......
        nativeZygoteInit();

        applicationInit(targetSdkVersion, argv);
    }
    ......
}
```

System Server进程启动过程分析(续)

- nativeZygoteInit--启动Binder线程池

```
static void com_android_internal_os_RuntimeInit_nativeZygoteInit(JNIEnv* env, jobject clazz)
{
    gCurRuntime->onZygoteInit();
}
```

```
class AppRuntime : public AndroidRuntime
{
public:
    ......

    virtual void onZygoteInit()
    {
        sp<ProcessState> proc = ProcessState::self();
        ALOGV("App process: starting thread pool.\n");
        proc->startThreadPool();
    }

    ......
};
```

System Server进程启动过程分析(续)

- applicationInit—调用SystemServer.main

```
public class RuntimeInit {
    ......
    private static void applicationInit(int targetSdkVersion, String[] argv)
            throws ZygoteInit.MethodAndArgsCaller {
        ......

        final Arguments args;
        try {
            args = new Arguments(argv);
        } catch (IllegalArgumentException ex) {
            Slog.e(TAG, ex.getMessage());
            // let the process exit
            return;
        }

        // Remaining arguments are passed to the start class's static main
        invokeStaticMain(args.startClass, args.startArgs);
    }

    ......
}
```

System Server进程启动过程分析(续)

- SystemServer.main

```
public class SystemServer {
    ......

    public static void main(String[] args) {
        ......
        System.loadLibrary("android_servers");
        init1(args);
    }

    ......
}
```

System Server进程启动过程分析(续)

- Init1—启动C/C++ Rutime Framework Service

```
static void android_server_SystemServer_init1(JNIEnv* env, jobject clazz)
{
    system_init();
}

extern "C" status_t system_init()
{
    sp<ProcessState> proc(ProcessState::self());
    ......
    char propBuf[PROPERTY_VALUE_MAX];
    property_get("system_init.startsurfaceflinger", propBuf, "1");
    if (strcmp(propBuf, "1") == 0) {
        // Start the SurfaceFlinger
        SurfaceFlinger::instantiate();
    }
    property_get("system_init.startsensorservice", propBuf, "1");
    if (strcmp(propBuf, "1") == 0) {
        // Start the sensor service
        SensorService::instantiate();
    }
    ......
    AndroidRuntime* runtime = AndroidRuntime::getRuntime();

    JNIEnv* env = runtime->getJNIEnv();
    ......
    jclass clazz = env->FindClass("com/android/server/SystemServer");
    ......
    jmethodID methodId = env->GetStaticMethodID(clazz, "init2", "()V");
    ......

    env->CallStaticVoidMethod(clazz, methodId);

    ProcessState::self()->startThreadPool();
    IPCThreadState::self()->joinThreadPool();

    return NO_ERROR;
}
```

System Server进程启动过程分析(续)

- Init2—启动Java Runtime Framework Service

```
public class SystemServer {
    ......

    public static final void init2() {
        Slog.i(TAG, "Entered the Android system server!");
        Thread thr = new ServerThread();
        thr.setName("android.server.ServerThread");
        thr.start();
    }

    ......
}
```

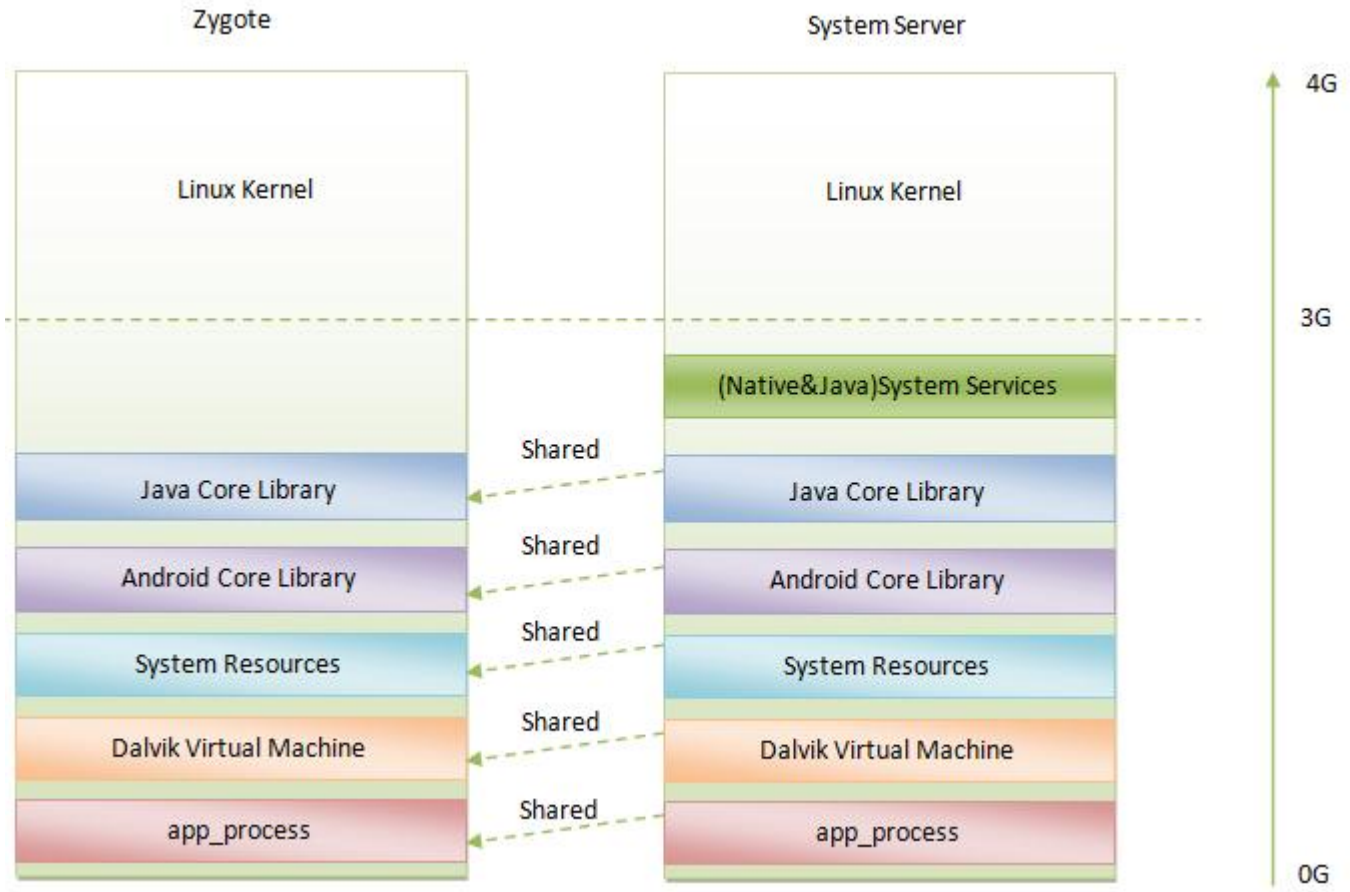
System Server进程启动过程分析(续)

- ServerThread.run

```
class ServerThread extends Thread {
    ......
    public void run() {
        ......
        Looper.prepareMainLooper();
        ......
        LightsService lights = null;
        PowerManagerService power = null;
        ......
        try {
            ......
            power = new PowerManagerService();
            ServiceManager.addService(Context.POWER_SERVICE, power);
            ......
        } catch (RuntimeException e) {
        }
        ......

        ActivityManagerService.self().systemReady(new Runnable() {
            public void run() {
                if (!headless) startSystemUi(contextF);
                try {
                    if (mountServiceF != null) mountServiceF.systemReady();
                } catch (Throwable e) {
                }
                ......
            }
        });

        Looper.loop();
    }
    ......
}
```

System Server进程启动过程分析(续)

- System Server进程启动完成后的地址空间



Android应用程序进程启动过程分析

- ActivityManagerService.startProcessLocked

```
public final class ActivityManagerService extends ActivityManagerNative
        implements Watchdog.Monitor, BatteryStatsImpl.BatteryCallback {
    ......
    private final void startProcessLocked(ProcessRecord app,
            String hostingType, String hostingNameStr) {
        ......
        try {
            int uid = app.uid;
            int[] gids = null;
            ......
            if (!app.isolated) {
                int[] permGids = null;
                try {
                    final PackageManager pm = mContext.getPackageManager();
                    permGids = pm.getPackageGids(app.info.packageName);
                    ......
                } catch (PackageManager.NameNotFoundException e) {
                }
                ......
                if (permGids == null) {
                    gids = new int[1];
                } else {
                    gids = new int[permGids.length + 1];
                    System.arraycopy(permGids, 0, gids, 1, permGids.length);
                }
                gids[0] = UserHandle.getSharedAppGid(UserHandle.getAppId(uid));
            }
            ......
            Process.ProcessStartResult startResult = Process.start("android.app.ActivityThread",
                    app.processName, uid, uid, gids, debugFlags, mountExternal,
                    app.info.targetSdkVersion, null, null);
            ......
        } catch (RuntimeException e) {
        }
    }
    ......
}
```

Android应用程序进程启动过程分析(续)

- Process.start

```
public class Process {
    ......
    public static final ProcessStartResult start(final String processClass,
                                  final String niceName,
                                  int uid, int gid, int[] gids,
                                  int debugFlags, int mountExternal,
                                  int targetSdkVersion,
                                  String seInfo,
                                  String[] zygoteArgs) {
        try {
            return startViaZygote(processClass, niceName, uid, gid, gids,
                    debugFlags, mountExternal, targetSdkVersion, seInfo, zygoteArgs);
        } catch (ZygoteStartFailedEx ex) {
            Log.e(LOG_TAG,
                    "Starting VM process through Zygote failed");
            throw new RuntimeException(
                    "Starting VM process through Zygote failed", ex);
        }
    }
    ......
}
```

Android应用程序进程启动过程分析(续)

- Process.startViaZygote

```
public class Process {
    ......
    private static ProcessStartResult startViaZygote(final String processClass,
                                  final String niceName,
                                  final int uid, final int gid,
                                  final int[] gids,
                                  int debugFlags, int mountExternal,
                                  int targetSdkVersion, String seInfo, String[] extraArgs)
                                  throws ZygoteStartFailedEx {
        synchronized(Process.class) {
            ArrayList<String> argsForZygote = new ArrayList<String>();
            ......
            // --runtime-init, --setuid=, --setgid=,
            // and --setgroups= must go first
            argsForZygote.add("--runtime-init");
            argsForZygote.add("--setuid=" + uid);
            argsForZygote.add("--setgid=" + gid);
            ......
            // --setgroups is a comma-separated list
            if (gids != null && gids.length > 0) {
                StringBuilder sb = new StringBuilder();
                sb.append("--setgroups=");
                int sz = gids.length;
                for (int i = 0; i < sz; i++) {
                    if (i != 0) {
                        sb.append(',');
                    }
                    sb.append(gids[i]);
                }
                argsForZygote.add(sb.toString());
            }
            ......
            return zygoteSendArgsAndGetResult(argsForZygote);
        }
    }
    ......
}
```

Android应用程序进程启动过程分析(续)

- Process.zygoteSendArgsAndGetResult

```
public class Process {
    ......
    private static ProcessStartResult zygoteSendArgsAndGetResult(ArrayList<String> args)
            throws ZygoteStartFailedEx {
        openZygoteSocketIfNeeded();

        try {
            ......
            sZygoteWriter.write(Integer.toString(args.size()));
            sZygoteWriter.newLine();

            int sz = args.size();
            for (int i = 0; i < sz; i++) {
                String arg = args.get(i);
                if (arg.indexOf('\n') >= 0) {
                }
                sZygoteWriter.write(arg);
                sZygoteWriter.newLine();
            }

            sZygoteWriter.flush();

            // Should there be a timeout on this?
            ProcessStartResult result = new ProcessStartResult();
            result.pid = sZygoteInputStream.readInt();
            if (result.pid < 0) {
                throw new ZygoteStartFailedEx("fork() failed");
            }
            result.usingWrapper = sZygoteInputStream.readBoolean();
            return result;
        } catch (IOException ex) {
        }
    }
    ......
}
```

Android应用程序进程启动过程分析(续)

- ZygoteConnection.runOnce

```
class ZygoteConnection {
    ......
    boolean runOnce() throws ZygoteInit.MethodAndArgsCaller {
        String args[];
        Arguments parsedArgs = null;
        ......
        try {
            args = readArgumentList();
            ......
        } catch (IOException ex) {
        }
        ......
        try {
            parsedArgs = new Arguments(args);
            ......
            pid = Zygote.forkAndSpecialize(parsedArgs.uid, parsedArgs.gid, parsedArgs.gids,
                    parsedArgs.debugFlags, rlimits, parsedArgs.mountExternal, parsedArgs.seInfo,
                    parsedArgs.niceName);
        } catch (IOException ex) {
        } catch (ErrnoException ex) {
        } catch (IllegalArgumentException ex) {
        } catch (ZygoteSecurityException ex) {
        }
        ......
        try {
            if (pid == 0) {
                // in child
                ......
                handleChildProc(parsedArgs, descriptors, childPipeFd, newStderr);
                ......
                return true;
            }
        } finally {
        }
    }
    ......
}
```

Android应用程序进程启动过程分析(续)

- ZygoteConnection.handleChildProc

```
class ZygoteConnection {
    ......
    private void handleChildProc(Arguments parsedArgs,
            FileDescriptor[] descriptors, FileDescriptor pipeFd, PrintStream newStderr)
            throws ZygoteInit.MethodAndArgsCaller {
        ......
        if (parsedArgs.runtimeInit) {
            if (parsedArgs.invokeWith != null) {
            } else {
                RuntimeInit.zygoteInit(parsedArgs.targetSdkVersion,
                        parsedArgs.remainingArgs);
            }
        } else {
        }
        ......
    }
    ......
}
```

Android应用程序进程启动过程分析(续)

- RuntimeInit.zygoteInit
  - nativeZygoteInit
  - applicationInit
    - Invoke main of ActivityThread

Android应用程序进程启动过程分析(续)

- ActivityThread.main

```
public final class ActivityThread {
    ......
    public static void main(String[] args) {
        ......

        Looper.prepareMainLooper();

        ActivityThread thread = new ActivityThread();
        thread.attach(false);
        ......

        Looper.loop();

        throw new RuntimeException("Main thread loop unexpectedly exited");
    }
}
```

Android应用程序进程回收机制

- Linux的内存回收机制--Out of Memory Killer
  - 每一个进程都有一个oom_adj值，取值范围[-17,15]，可以通过/proc/<pid>/oom_adj访问
  - 每一个进程的oom_adj初始值都等于其父进程的oom_adj值
  - oom_adj值越小，越不容易被杀死，其中，-17表示不会被杀死
  - 内存紧张时，OOM Killer综合进程的内存消耗量、CPU时间、存活时间和oom_adj值来决定是否要杀死一个进程来回收内存

Android应用程序进程回收机制(续)

- Android的内存回收机制—Low Memory Killer
  - 进程的oom_adj值由ActivityManagerService根据运行在进程里面的组件的状态来计算
  - 进程的oom_adj值取值范围为[-16,15]，oom_adj值越小，就不容易被杀死
  - 内存紧张时， LMK基于oom_adj值来决定是否要回收一个进程
  - ActivityManagerService和WindowManagerService在特定情况下也会进行进程回收

Android应用程序进程回收机制(续)

- LMK的进程回收策略
  - 当系统内存小于i时，在oom_adj值大于等于j的进程中，选择一个oom_adj值最大并且消耗内存最多的进程来回收

```
class ProcessList {
    ......
    // These are the various interesting memory levels that we will give to
    // the OOM killer.  Note that the OOM killer only supports 6 slots, so we
    // can't give it a different value for every possible kind of process.
    private final int[] mOomAdj = new int[] {
            FOREGROUND_APP_ADJ/*0*/, VISIBLE_APP_ADJ/*1*/, PERCEPTIBLE_APP_ADJ/*2*/,
            BACKUP_APP_ADJ/*4*/, HIDDEN_APP_MIN_ADJ/*9*/, HIDDEN_APP_MAX_ADJ/*15*/
    };
    // These are the low-end OOM level limits.  This is appropriate for an
    // HVGA or smaller phone with less than 512MB.  Values are in KB.
    private final long[] mOomMinFreeLow = new long[] {
            8192, 12288, 16384,
            24576, 28672, 32768
    };
    // These are the high-end OOM level limits.  This is appropriate for a
    // 1280x800 or larger screen with around 1GB RAM.  Values are in KB.
    private final long[] mOomMinFreeHigh = new long[] {
            32768, 40960, 49152,
            57344, 65536, 81920
    };
    ......
}
```

Android应用程序进程回收机制(续)

- 应用程序进程的oom_adj值
  - SYSTEM_ADJ(-16)：System Server进程
  - PERSISTENT_PROC_ADJ(-12)：android:persistent属性为true的系统App进程，如PhoneApp
  - FOREGROUND_APP_ADJ(0)：包含前台Activity的进程
  - VISIBLE_APP_ADJ(1)：包含可见Activity的进程
  - PERCEPTIBLE_APP_ADJ(2)：包含状态为Pausing、Paused、Stopping的Activity的进程，以及运行有Foreground Service的进程
  - HEAVY_WEIGHT_APP_ADJ(3)：重量级进程， android: cantSaveState属性为true的进程，目前还不开放
  - BACKUP_APP_ADJ(4)：正在执行备份操作的进程
  - SERVICE_ADJ(5)：最近有活动的Service进程
  - HOME_APP_ADJ(6)：HomeApp进程
  - PREVIOUS_APP_ADJ(7)：前一个App运行在的进程
  - SERVICE_B_ADJ(8)：SERVICE_ADJ进程数量达到一定值时，最近最不活动的Service进程
  - HIDDEN_APP_MIN_ADJ(9)和HIDDEN_APP_MAX_ADJ(15)：含有不可见Activity的进程，根据LRU原则赋予[9,15]中的一个值
- Init进程的oom_adj值被设置为-16，由Init进程所启动的daemon和service进程的oom_adj值也等于-16
- 如果运行在进程A中的Content Provider或者Service被绑定到进程B，并且进程B的oom_adj值比进程A的oom_adj小，那么进程A的oom_adj值就会被设置为进程B的oom_adj值，但是不能小于FOREGROUND_APP_ADJ

Android应用程序进程回收机制(续)

- ActivityManagerService在以下四种情况下会更新应用程序进程的oom_adj值，以及杀掉那些已经被卸载了的App所运行在的应用程序进程
  - activityStopped：停止Activity
  - setProcessLimit：设置进程数量限制
  - unregisterReceiver：注销Broadcast Receiver
  - finishReceiver：结束Broadcast Receiver
- WindowManagerService在处理窗口的过程中发生Out Of Memroy时，也会通知ActivityManagerService杀掉那些包含有窗口的应用程序进程

Q&A

Thank You