Android 下 dvm 进程的创建

版本号：0.01　　版本更新、技术讨论请前往 http://www.seaforestmountain.com/bbs
作者：暗夜之眸　　

整个 android 系统下，所有 dvm 进程的创建都是通过 zygote 进程 fork 来完成。Zygote 进程是由 app_main 演进而来，在完成了相关的初始工作后，Zygote 进程就演化为 android 系统 dvm 进程母体，她驻留在系统中，等待系统的请求，生出一个个 dvm 进程。

还是位于 Zygoteinit.java 文件中，我们略去其他部分，集中分析 Zygote 作为系统母体的工作。

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

            …
//创建一个 socket，Zygote 就是通过这个 socket 和系统其他组件交互
            registerZygoteSocket();

            if (ZYGOTE_FORK_MODE) {
                runForkMode();
            } else {
                runSelectLoopMode();
            }
            …
    }

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

//Zygote 母体就等在 sServerSocket 上，取出 sServerSocket 的文件描述符，放到 fds，以供 select 用
        fds.add(sServerSocket.getFileDescriptor());
//这时没有 peer，所以 peers 置空
        peers.add(null);

//Zygote 进程一直驻留内存，每隔一段时间做一次 gc，GC_LOOP_COUNT 控制 gc 间隔
        int loopCount = GC_LOOP_COUNT;
        while (true) {
```

```
                int index;
                if (loopCount <= 0) {
//ok，gc 时间到了
                    gc();
                    loopCount = GC_LOOP_COUNT;
                } else {
                    loopCount--;
                }


                try {
                    fdArray = fds.toArray(fdArray);
//select这些文件fd，static native int selectReadable(…)本身的jni实现就是取出找到fdArray 数组里对应文件描述符，再将他们应用到 linux 的 select 编程模型上。
                    index = selectReadable(fdArray);
                } catch (IOException ex) {
                    throw new RuntimeException("Error in select()", ex);
                }

                if (index < 0) {
                    throw new RuntimeException("Error in select()");
                } else if (index == 0) {
//发现 sServerSocket 上有 peer 来连接，accept 该 peer，通常这个 peer 就是位于 systemserver 进程里的 activitymanagerservice。
                    ZygoteConnection newPeer = acceptCommandPeer();
                    peers.add(newPeer);
//将该 accept 该 peer 的 socket 放入 fds 数组，以供 select
                    fds.add(newPeer.getFileDesciptor());
                } else {
                    boolean done;
//peer 发来请求了，对应这个连接，运行 class ZygoteConnection 的 runOnce 函数
                    done = peers.get(index).runOnce();

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

        boolean runOnce() throws ZygoteInit.MethodAndArgsCaller {

            String args[];
```

```
            Arguments parsedArgs = null;
            FileDescriptor[] descriptors;

            try {
                args = readArgumentList();
                descriptors = mSocket.getAncillaryFileDescriptors();
            } catch (IOException ex) {
                Log.w(TAG, "IOException on command socket " + ex.getMessage());
                closeSocket();
                return true;
            }

            if (args == null) {
                // EOF reached.
                closeSocket();
                return true;
            }

            /** the stderr of the most recent request, if avail */
            PrintStream newStderr = null;

            if (descriptors != null && descriptors.length >= 3) {
                newStderr = new PrintStream(
                        new FileOutputStream(descriptors[2]));
            }

            int pid;

            try {
//这里很重要，这是 android 安全体系的最重要的组成部分，说来话长，参见 android 安全模型章节
                parsedArgs = new Arguments(args);

                applyUidSecurityPolicy(parsedArgs, peer);
                applyDebuggerSecurityPolicy(parsedArgs);
                applyRlimitSecurityPolicy(parsedArgs, peer);
                applyCapabilitiesSecurityPolicy(parsedArgs, peer);

                int[][] rlimits = null;

                if (parsedArgs.rlimits != null) {
                    rlimits = parsedArgs.rlimits.toArray(intArray2d);
                }
//fork()一个进程出来以容纳，新的 dvm 进程。
```

```
                pid = Zygote.forkAndSpecialize(parsedArgs.uid, parsedArgs.gid,
                        parsedArgs.gids, parsedArgs.debugFlags, rlimits);
            } catch (IllegalArgumentException ex) {
                logAndPrintError (newStderr, "Invalid zygote arguments", ex);
                pid = -1;
            } catch (ZygoteSecurityException ex) {
                logAndPrintError(newStderr,
                        "Zygote security policy prevents request: ", ex);
                pid = -1;
            }

            if (pid == 0) {
//如果是子进程，返回到这里
                handleChildProc(parsedArgs, descriptors, newStderr);
                // should never happen
                return true;
            } else { /* pid != 0 */
        //父进程即 zygote 进程接着从这里走
                // in parent...pid of < 0 means failure
                return handleParentProc(pid, descriptors, parsedArgs);
            }
        }
```

在 activitymanagerservice 里用启动一个新的 dvm 进程

```
        private final void startProcessLocked(ProcessRecord app,
                String hostingType, String hostingNameStr) {
                int pid = Process.start("android.app.ActivityThread",
                        mSimpleProcessManagement ? app.processName : null, uid, uid,
                        gids, debugFlags, null);

}


        public static final int start(final String processClass,
                                      final String niceName,
                                      int uid, int gid, int[] gids,
                                      int debugFlags,
                                      String[] zygoteArgs)
        {
            if (supportsProcesses()) {
                try {
//通过 Zygote 进程启动新的 dvm 进程
```

```
                return startViaZygote(processClass, niceName, uid, gid, gids,
                        debugFlags, zygoteArgs);
            } catch (ZygoteStartFailedEx ex) {
                Log.e(LOG_TAG,
                        "Starting VM process through Zygote failed");
                throw new RuntimeException(
                        "Starting VM process through Zygote failed", ex);
            }
        } else {
            // Running in single-process mode

            Runnable runnable = new Runnable() {
                        public void run() {
                            Process.invokeStaticMain(processClass);
                        }
            };

            // Thread constructors must not be called with null names (see spec).
            if (niceName != null) {
                new Thread(runnable, niceName).start();
            } else {
                new Thread(runnable).start();
            }

            return 0;
        }
}
```