简介

zygote是受精卵的意思，它是Android中的一个非常重要的守护进程服务(Daem Service),所有的其他Dalvik虚拟机进程都是通过zygote孵化（fork）出来的。Android应用程序是由Java语言编写的，运行在各自独立的Dalvik虚拟机中。如果每个应用程序在启动之时都需要单独运行和初始化一个虚拟机，会大大降低系统性能，因此Android首先创建一个zygote虚拟机，然后通过它孵化出其他的虚拟机进程，进而共享虚拟机内存和框架层资源，这样大幅度提高应用程序的启动和运行速度。

Zygote是Android中最重要的一个进程，和Init进程，SystemServer进程是支撑Android世界的三极。Zygote进程在Init进程中以service的方式启动的。

启动流程

ZygoteInit类负责Zygote进程Java的初始化工作，首先来看下ZygoteInit类的入口main()方法：

```
public static void main(String argv[]) {
    // Mark zygote start. This ensures that thread creation will throw
    // an error.
    ZygoteHooks.startZygoteNoThreadCreation();

    try {
        Trace.traceBegin(Trace.TRACE_TAG_DALVIK, "ZygoteInit");
        //开启DDMS
        RuntimeInit.enableDdms();
        // Start profiling the zygote initialization.
        SamplingProfilerIntegration.start();

        ....

        //1）注册zygote的socket监听端口
        registerZygoteSocket(socketName);
        Trace.traceBegin(Trace.TRACE_TAG_DALVIK, "ZygotePreload");
        EventLog.writeEvent(LOG_BOOT_PROGRESS_PRELOAD_START,
            SystemClock.uptimeMillis());
        //2)预加载系统资源
        preload();
        EventLog.writeEvent(LOG_BOOT_PROGRESS_PRELOAD_END,
            SystemClock.uptimeMillis());
        Trace.traceEnd(Trace.TRACE_TAG_DALVIK);

        ....

        //3）启动startServier进程
        if (startSystemServer) {
            startSystemServer(abiList, socketName);
        }

        Log.i(TAG, "Accepting command socket connections");
        //4）进入监听和接受消息的循环
        runSelectLoop(abiList);

        closeServerSocket();
    } catch (MethodAndArgsCaller caller) {
        caller.run();
```

```
    } catch (RuntimeException ex) {
    }
}
```

主要做了以下几件事：1）注册zygote的socket端口监听；2）预加载系统资源；3）启动SystemServer进程；4）进入监听和接受消息的循环；依次看下这四个方法：

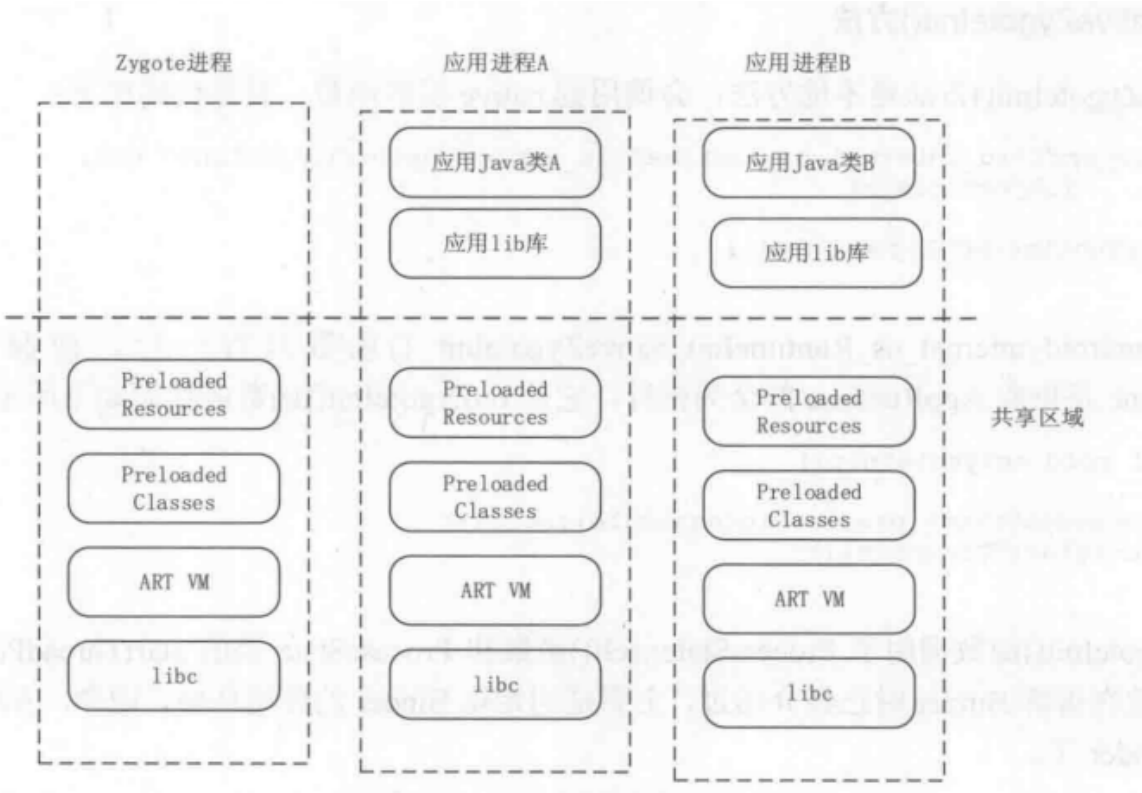
1）registerZygoteSocket()方法

```
private static void registerZygoteSocket(String socketName) {
    if (sServerSocket == null) {
        int fileDesc;
        final String fullSocketName = ANDROID_SOCKET_PREFIX + socketName;
        try {
            String env = System.getenv(fullSocketName);
            fileDesc = Integer.parseInt(env);
        }

        try {
            FileDescriptor fd = new FileDescriptor();
            fd.setInt$(fileDesc);
            sServerSocket = new LocalServerSocket(fd);
        }
    }
}
12345678910111213141516
```

创建一个本地的socket,然后等待调用runSelectLoop()来进入等待socket等待连接的循环中；

2）预加载系统资源，preload()方法；
先来看下Android应用进程共享内存图



▲图 8.4　Android 应用进程共享内存图

通过上图可以很容易理解在Zygote进程预加载系统资源后，然后通过它孵化出其他的虚拟机进程，进而

共享虚拟机内存和框架层资源，这样大幅度提高应用程序的启动和运行速度。 现在来看下preload()方法;

```
 static void preload() {
    Log.d(TAG, "begin preload");
    Trace.traceBegin(Trace.TRACE_TAG_DALVIK, "BeginIcuCachePinning");
    beginIcuCachePinning();
    Trace.traceEnd(Trace.TRACE_TAG_DALVIK);
    Trace.traceBegin(Trace.TRACE_TAG_DALVIK, "PreloadClasses");
    //2.1)预加载系统类
    preloadClasses();
    Trace.traceEnd(Trace.TRACE_TAG_DALVIK);
    Trace.traceBegin(Trace.TRACE_TAG_DALVIK, "PreloadResources");
    //2.2）预加载系统资源
    preloadResources();
    Trace.traceEnd(Trace.TRACE_TAG_DALVIK);
    Trace.traceBegin(Trace.TRACE_TAG_DALVIK, "PreloadOpenGL");
    //预加载OpenGL资源
    preloadOpenGL();
    Trace.traceEnd(Trace.TRACE_TAG_DALVIK);
    //2.3）预加载共享的so库
    preloadSharedLibraries();
    preloadTextResources();
    // Ask the WebViewFactory to do any initialization that must run in the
zygote process,
    // for memory sharing purposes.
    //预加载WebView资源库
    WebViewFactory.prepareWebViewInZygote();
    endIcuCachePinning();
    warmUpJcaProviders();
    Log.d(TAG, "end preload");
}
```

主要是预加载各种系统资源，主要看下2.1）预加载系统类；2.2）预加载系统资源；2.3）预加载so库

2.1 preloadClasses();

```
private static final String PRELOADED_CLASSES = "/system/etc/preloaded-classes";
private static void preloadClasses() {
    final VMRuntime runtime = VMRuntime.getRuntime();

    InputStream is;
    try {
        is = new FileInputStream(PRELOADED_CLASSES);
    } catch (FileNotFoundException e) {
        Log.e(TAG, "Couldn't find " + PRELOADED_CLASSES + ".");
        return;
    }
   ....
    float defaultUtilization = runtime.getTargetHeapUtilization();
    runtime.setTargetHeapUtilization(0.8f);

    try {
        BufferedReader br= new BufferedReader(new InputStreamReader(is), 256);
        int count = 0;
        String line;
```

```
        while ((line = br.readLine()) != null) {
            // Skip comments and blank lines.
            line = line.trim();
            if (line.startsWith("#") || line.equals("")) {
                continue;
            }

             ......
            try {
                //装载Java类信息
                Class.forName(line, true, null);
                count++;
            } catch (ClassNotFoundException e) {}
      .....

    } finally {
        Trace.traceBegin(Trace.TRACE_TAG_DALVIK, "PreloadDexCaches");
        runtime.preloadDexCaches();
        Trace.traceEnd(Trace.TRACE_TAG_DALVIK);
    }
}
```

去读PRELOADED_CLASSES文件下的文件，得到InputStream对象，在转换为BufferedReader，逐行读取文件的内容，每行通过trim(),过滤掉空行，然后调用 Class.forName()方法，加载Java类信息，而不是创建一个对象；

2.2 preloadResources();预加载系统资源

```
private static void preloadResources() {
    final VMRuntime runtime = VMRuntime.getRuntime();

    try {
        mResources = Resources.getSystem();
        mResources.startPreloading();
        if (PRELOAD_RESOURCES) {
            //加载系统Drawable资源
            TypedArray ar =
mResources.obtainTypedArray(com.android.internal.R.array.preloaded_drawables);
            int N = preloadDrawables(ar);
            ar.recycle();
            //加载系统颜色资源
            ar =
mResources.obtainTypedArray(com.android.internal.R.array.preloaded_color_state_l
ists);
            N = preloadColorStateLists(ar);
            ar.recycle();
        }
        mResources.finishPreloading();
    } catch (RuntimeException e) {
    }
}
```

2.3 preloadSharedLibraries();加载系统共享so库

```
private static void preloadSharedLibraries() {
    Log.i(TAG, "Preloading shared libraries...");
    System.loadLibrary("android");
    System.loadLibrary("compiler_rt");
    System.loadLibrary("jnigraphics");
}
```

3 startSystemServer() 启动SystemServer进程

```
private static boolean startSystemServer(String abiList, String socketName)
        throws MethodAndArgsCaller, RuntimeException {

    ....
    //3.1 为启动SystemServer进程准备参数
    String args[] = {
        "--setuid=1000",
        "--setgid=1000",
        "--
setgroups=1001,1002,1003,1004,1005,1006,1007,1008,1009,1010,1018,1021,1032,3001,
3002,3003,3006,3007,3009,3010",
        "--capabilities=" + capabilities + "," + capabilities,
        "--nice-name=system_server",
        "--runtime-args",
        "com.android.server.SystemServer",
    };
    ZygoteConnection.Arguments parsedArgs = null;

    int pid;

    try {
        parsedArgs = new ZygoteConnection.Arguments(args);
        ZygoteConnection.applyDebuggerSystemProperty(parsedArgs);
        ZygoteConnection.applyInvokeWithSystemProperty(parsedArgs);

        //3.2 fork出SystemServer进程
        /* 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) {
        throw new RuntimeException(ex);
    }

    /* For child process */
    if (pid == 0) {
        if (hasSecondZygote(abiList)) {
            waitForSecondaryZygote(socketName);
        }
        //3.3 fork出SystemServer进程之后，初始化SystemServer进程
        handleSystemServerProcess(parsedArgs);
    }
```

```
    return true;
}
```

主要做了三件事，3.1）为启动SystemServer进程准备参数，可以看到SystemServer的进程Id和组Id均为1000，SystemServer的执行类是com.android.server.SystemServer；3.2）fork出SystemServer进程；3.3）fork出SystemServer进程之后，初始化SystemServer进程；

看下handleSystemServerProcess()方法

```
 private static void handleSystemServerProcess( ZygoteConnection.Arguments
parsedArgs)
        throws ZygoteInit.MethodAndArgsCaller {
    //关闭zygote的socket
    closeServerSocket();
    //设置umask为0077；只有SystemServer进程可以访问；
    // set umask to 0077 so new files and directories will default to owner-only
permissions.
    Os.umask(S_IRWXG | S_IRWXO);
    //由3.1可以看出nice-name=system_server，设置进程的名称为system_server；
    if (parsedArgs.niceName != null) {
        Process.setArgV0(parsedArgs.niceName);
    }

    final String systemServerClasspath = Os.getenv("SYSTEMSERVERCLASSPATH");
    if (systemServerClasspath != null) {
        performSystemServerDexOpt(systemServerClasspath);
    }

    //由3.1可以看出invokeWith为null;
    if (parsedArgs.invokeWith != null) {

    } else {
        ClassLoader cl = null;
        if (systemServerClasspath != null) {
            cl = createSystemServerClassLoader(systemServerClasspath,
                                               parsedArgs.targetSdkVersion);

            Thread.currentThread().setContextClassLoader(cl);
        }

        RuntimeInit.zygoteInit(parsedArgs.targetSdkVersion,
parsedArgs.remainingArgs, cl);
    }
}
```

4）runSelectLoop()方法

```
private static void runSelectLoop(String abiList) throws MethodAndArgsCaller {
    ArrayList<FileDescriptor> fds = new ArrayList<FileDescriptor>();
    ArrayList<ZygoteConnection> peers = new ArrayList<ZygoteConnection>();

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

```
    while (true) {
        StructPollfd[] pollFds = new StructPollfd[fds.size()];
        for (int i = 0; i < pollFds.length; ++i) {
            pollFds[i] = new StructPollfd();
            pollFds[i].fd = fds.get(i);
            pollFds[i].events = (short) POLLIN;
        }
        try {
            Os.poll(pollFds, -1);
        } catch (ErrnoException ex) {
            throw new RuntimeException("poll failed", ex);
        }
        for (int i = pollFds.length - 1; i >= 0; --i) {
            if ((pollFds[i].revents & POLLIN) == 0) {
                continue;
            }
            //4.1）接受连接请求
            if (i == 0) {
                ZygoteConnection newPeer = acceptCommandPeer(abiList);
                peers.add(newPeer);
                fds.add(newPeer.getFileDesciptor());
            //4.2) 接受消息
            } else {
                boolean done = peers.get(i).runOnce();
                if (done) {
                    peers.remove(i);
                    fds.remove(i);
                }
            }
        }
    }
}
```

主要做了两件事4.1）接受连接请求；i=0,说明请求连接的事件过来了，调用acceptCommandPeer()和客户端建立socket连接，然后加入监听数组，等待这个socket上命令的到来；4.2）接受消息；i>0 说明已经连接上的socket已经有数据到了，调用ZygoteConnection类的runOnce()方法处理完成后，会断开和客户端的连接，并且从监听数组中移除；

以上就是Zygote进程的启动流程和在main()方法中主要做的四件事的解析，如有问题，请多指教，谢谢！

Android FrameWork的文章现在有很多，相关的书籍也有不少，都写的很通俗易懂，我写相关的文章主要是为了记录在学习FrameWork过程中的点滴。