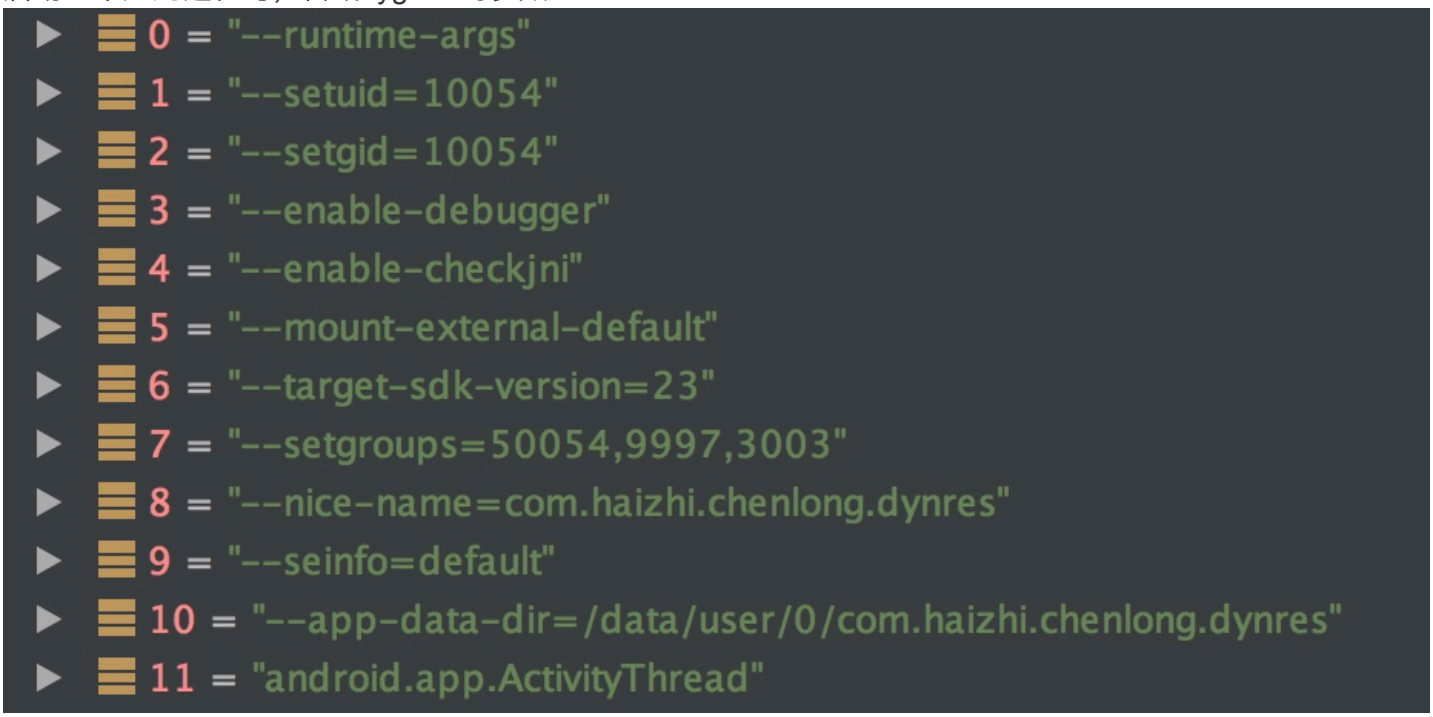
Android进程启动流程

问题
1. 插件app运行在哪个进程?
2. zygote的启动过程，system_server的启动过程
   其他进程的启动过程
3. 由zygote fork 出新的进程是在哪里发生的?
4. sytem_server 包含哪些功能模块?
file:///Users/chenlong/git/XposedFramework/XposedBridge/app/build/docs/javadoc/reference/de/robv/android/xposed/services/package-summary.html

一个异常栈：

```
at android.os.Handler.handleCallback(Handler.java:743)
at android.os.Handler.dispatchMessage(Handler.java:95)
at android.os.Looper.loop(Looper.java:150)
at android.app.ActivityThread.main(ActivityThread.java:5643)
at java.lang.reflect.Method.invoke(Native Method)
at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:799)
at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:689)
```

启动一个应用进程时，传给zygote 的参数




```
->init.rc

  service zygote /system/bin/app_process -Xzygote /system/bin --zygote \
    --start-system-server

->base/cmd/app_process/app_mian.cpp
    /**
     * 从init.rc中传入了两个参数,
     * --zygote
     * --start-system-server
     **/
```

```
    void main(int argc,char* const argv[]){
      ...
      //解析参数
      zygote = true
      nicename = "zygote64"//64位，32位下为"zyogte"
      增加了一个参数"--abi-list="
      ...
      if(zygote){
        runtime->start("com.android.internal.os.ZygoteInit");
      }else{
        runtime->start("com.android.internal.os.RuntimeInit")
      }
    }


    /**
     * JavaVM 初始化之后，开启线程池
     **/
    virtual void onZygoteInit(){
      sp<ProcessState> proc = ProcessState::self();
      proc->startThreadPool();
    }
```

```
->base/core/jni/android/AndroidRuntime.cc

    int start(const char* className,const Vector<String8>& options,bool zygote){

      /*
        打开libart.so 库，
        初始化以下三个方法：
        1. JNI_GetDefaultJavaVMInitArgs
        2. JNI_CreateJavaVM
        3. JNI_GetCreatedJavaVMs
       */
      JniInvocation jniInvocation;
      jniInvocation.Init(NULL);



      ...
      //启动虚拟机
      //JavaVM是每一个进程一个，JniEnv每个线程一个
      startVm(&mJavaVm,env,zygote);
      ...
      //执行entryClass的main函数
      env->CallStaticVoidMethod(startClass,startMeth,strArray);
    }



    int startVm(JavaVM** pJavaVM,JNIEnv** pEnv,bool zygote){
      JNI_CreateJavaVm(pJavaVM,pEnv,&initArgs);
    }

    /**
     * 从zygote fork 进程并且JavaVM 初始化之后的回调
     **/
    static void com_..._nativeZygoteInit(JNIEnv* env, jobject clazz)
    {
```

```
    gCurRuntime->onZygoteInit();
   }
```

```
->art/runtime/java_vm_ext.cc
  int JNI_CreateJavaVm(JavaVM** p_vm,JNIEnv* p_env,void* vm_args){
      Runtime::Create(options,ignore_unrecongized);//创建Runtime的单例
      Runtime* runtime = Runtime::Current();//
      runtime->Start()
      *p_env = Thread::Current()->GetJniEnv();
      *p_vm  = runtime->GetJavaVm();
  }
```

```
->art/runtime/runtime.cc

  bool Runtime::Create(const RuntimeOptions& options,bool ignore_unrecongized){
      //创建实例，注意此处runtime实例是一个单例
      instance_ = new Runtime;
      //执行初始化方法
      instance_->Init(options,ignore_unrecognized)
  }
  /**
   * 初始化runtime
   **/
  bool Runtime:Init(const RuntimeOptions& options,bool ignore_unrecongized){
      ...
      is_zygote_ = ....
      java_vm_ = new JavaVMExt(this,runtime_options);
      ...
  }
  /**
   * 启动虚拟机
   **/
  bool Runtime::Start(){
      ...
      system_class_loader_ = CreateSystemClassLoader(this)
      ...
      if(is_zygote_){//初始化zygote进程
        InitZygote();
      }else{//从zygote进程fork子进程
        DidForkFromZygote(...)
      }
  }
```

```

```

```
->com/android/internal/os/ZygoteInit.java
  /**
   * 启动Zygote
   * 注意，此处依然运行在app_process的进程中
   **/
  public static void main(String[] args){

    try{
        //注册socket服务端
        registerZygoteSocket(socketName);
        //预加载一些类和资源
        preload();
        //对zygote进程做GC
```

```
        gcAndFinalize();
        if(startSystemServer){
          //启动SystemServer
          startSystemServer(abiList,sockerName);
        }
        //死循环
        runSelectLoop(abiList);
        //关闭socket服务端
        closeServerSocket();
      }catch(MethodAndArgsCaller caller){
        caller.run();
      }catch(RuntimeException e){
        closeServerSocket();
      }

    }
    /**
     * 启动socket服务端
     **/
    private static void registerZygoteSocket(String socketName){
      if(sServerSocker == null){
        //获取"ANDROID_SOCKET_zygote"对应的文件描述符
        int fileDesc = System.getenv("ANDROID_SOCKET_zygote");
        FileDescriptor fd = new FileDescriptor();
        fd.setInt$(fileDesc);
        sServerSocker = new LocalServerScoket(fd);
      }
    }

    /**
     * 预加载系统类和资源
     **/
    static void preload(){
      //读取/system/etc/preloaded-classes文件,
      //文件每一行都是类名，使用当前的虚拟机实例加载它们
      preloadClasses();
      //加载R.array.preloaded_drawables中定义的drawable
      //加载R.array.preloaded_color_state_lists中定义的colorStateList
      preloadResource();
      //加载OpenGL库
      preloadOpenGL();
      //加载"android", "compiler_rt","jnigraphics"
      preloadSharedLibraries();
      //加载国际化处理的一些东西
      preloadTextResources();
      //初始化webview，应该是加载浏览器内核
      WebViewFactory.prepareWebViewInZygote();

    }

    /**
     * 启动SystemServer
     **/
    private static boolean startSystemServer(String abiList,String socketName){
        String args[] = {
          ...
          "--nice-name=system_server"
          "com.android.server.SystemServer"
        }
        ZygoteConnection.Arguments parsedArgs = new Arguments(args);
        int pid = Zygote.forkSystemServer(...parsedArgs);
```

```
75
76      //fork()函数调用一次，返回两次，在父进程返回子进程pid
77      //在子进程返回0
78      if(pid == 0){
79        handleSystemServerProcess(...)
80      }
81
82
83    }
84    /**
85     * 处理system_server 进程
86     **/
87    private static void handleSystemServerProcess(Zygoteconnection.Arguments parsedArgs){
88      //由于是从zygote中fork出来的，携带了zygote的socket服务端，
89      //先关掉
90      closeServerSocket();
91      //获取SystemServer的classpath
92      String systemServerClassPath
93            = OS.getenv("SYSTEMSERVERCLASSPATH");
94      //执行dexopt
95      performSystemServerDexOpt(systemServerClassPath);
96
97      //设置当前线程的上下文classloader，增加"SYSTEMSERVERCLASSPATH"
98      //对应的jar包
99      ClassLoader cl= new PathClassLoader(
100                       systemServerClassPath,
101                       ClassLoader.getSystemClassLoader());
102     Thread.currentThread().setContextClassLoader(cl);
103
104     //初始化com.android.server.SystemServer类
105     RuntimeInit.zygoteInit(parsedArgs.targerSdkVersion,
106                         parsedArgs.remainingArgs,cl);
107
108   }
109
110   /**
111    * 接收socket消息
112    **/
113   private static void runSelectLoop(String abiList){
114     List<ZygoteConnection> peers = new ArrayList<>();
115     while(true){
116       Os.poll(pollFds,-1);
117       ZygoteConnection newPeer = acceptCommandPeer(abiList)
118       peers.add(newPeer);
119       newPeer.runOnce();
120     }
121   }
```

```
1  ->framworks/base/core/java/com/android/internal/os/ZygoteConnection.java
2    private void runOnce(){
3      pid = Zygote.forkAndSpecialize(...)
4      if(pid == 0){
5        closeSocket();
6        ZygoteInit.closeServerSocket();
7        RuntimeInit.zygoteInit(...);
8      }else{
9        handleParentProc(...)
10     }
11   }
```

```
1  ->framworks/base/core/java/com/android/internal/os/Zygote.java
```

```

  public static int forkSystemServer(....){
    int pid = nativeForkSystemServer(....)
  }

->framworks/base/core/jni/com_android_intenral_os_Zygote.cpp

  /**
   * 从zygote fork SystemServer的进程
   **/
  static jint com_..._Zygote_nativeForkSystemServer(...){
    ...
    pid_t pid = ForkAndSpecicalzeCommon(...)
    ...
    waitpid(pid,&status,WNOHANG);

  }

  static pid_t ForkAndSpecicalzeCommon(...){
    ...
    pid_t pid = fork();
    ...
  }

```

```
->framworks/base/core/java/com/android/internal/os/RuntimeInit.java

  public static final void zygoteInit
                              int targetSdkVersion,
                              String[] args,
                              ClassLoader classloader){
      //初始化一些东西
      commonInit();1
      //触发回调
      nativeZygoteInit();
      //初始化应用
      applicationInit(targetSdkVersion,args,classloader);

  }

  private static final void commonInit(){
    //设置异常捕获
    Thread.setDefaultUncaughtExcptionHandler(new UncaughtHandler());
    ...
    //设置时区
    ...
    //配置log
    ...
    //设置user-agent
    ...
    //设置socketTagger
    ...
  }

  private static void applicationInit(
                                int targerSdkVersion,
                                String[] argv,
                                ClassLoader cl){
      ...
      //执行startClass的main方法
      /*
```

```
        流程如下:
        1. 使用前一步创建的classloader加载目标类
        2. 获取main方法
        3. throw new ZygoteInit.MethondAndArgsCaller(m,argv);

        第三步中抛出的异常会在ZygoteInit中catch，然后再执行
          run()方法，run方法中只是invoke目标方法
      */
      invokeStaticMain(args.startClass,args.startArgs,classloader)
      ...
  }

```

整体的流程分析:

1. kernel启动->init进程启动->读取init.rc文件，准备启动app_process
2. 启动app_process，初始化AppRuntime对象
3. AppRuntime创建一个虚拟机
4. 虚拟机初始化并执行目标类‘ZygoteInit’的main函数
5. ZygoteInit的main函数执行，注册sokcet服务端，加载framework的类和资源
6. fork一个新的进程，启动SystemServer
7. 开启socket 循环，接收指令

```
->/frameworks/base/services/java/com/android/server/SystemServer.java

```

```
AndroidRuntime
/frameworks/base/include/android_runtime/AndroidRuntime.h

/**
 * VM 创建后，执行代码之前调用。
 **/
virtual void onVmCreated(JNIEnv* env);

/**
 * JavaVM 初始化之后调用
 **/
virtual void onStarted() = 0;

/**
 * 从zygote fork进程并且JavaVM初始化后调用。
 **/
virtual void onZygoteInit() { }

/**
 * 当Java应用退出后进程结束前调用
 **/
virtual void onExit(int code) { }



/frameworks/base/core/jni/android/AndroidRuntime.cc

/**
 * 构造方法
 * 每个进程一个实例
 */
AndroidRuntime(char* argBlockStart, size_t argBlockSize){
```

```
  gCurRuntime = this;
}
```

```
->/frameworks/base/services/java/com/android/server/SystemServer.java
public static main(String[] argvs){
  new SystemServer().run();
}

private void run(){
  ...
  System.loadLibrary("android_servers");
  ...
  createSystemContext();
  mSystemServiceManager = new SystemServiceManager(mSystemContext);
  LocalServices.addService(SystemServiceManager.class,mSystemServiceManager);

  /**
   * 启动系统服务
   * 1. Installer
   * 2. ActivityManagerService
   * 3. PowerManagerService
   * 4. PackageManagerService
   * 5. WindowManagerService
   * 6. LauncherAppService
   * ...
   **/
  startBootstrapServices();
  startCoreServices();
  startOtherServices();
}

```