AmS与插件化的运行时Hook

问题：
1. theme究竟意味着什么?
2. Activity生命周期的所有阶段是不是都需要通知AmS?
3. 应用的进程是什么时候创建的?
4. app->AmS->app的大致流程是怎样的?
5. TaskRecord,ActivityRecord,LoadedApk,ActivityThread,Application,Context,ContextImpl,ThemedContextWrapper,ActivityInfo,Resource,ResourceManager,AssetManager,

```
if (entryPoint == null) entryPoint = "android.app.ActivityThread";
Trace.traceBegin(Trace.TRACE_TAG_ACTIVITY_MANAGER, "Start proc: " +
        app.processName);
checkTime(startTime, "startProcess: asking zygote to start proc");
Process.ProcessStartResult startResult = Process.start(entryPoint,
        app.processName, uid, uid, gids, debugFlags, mountExternal,
        app.info.targetSdkVersion, app.info.seinfo, requiredAbi, instructionSet,
        app.info.dataDir, entryPointArgs);
```

```
root@generic_x86_64:/ # cat init.zygote32.rc
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
```

6.0.1

```
enum {
    kEMDefault,
    kEMIntPortable,
    kEMIntFast,
    kEMJitCompiler,
} executionMode = kEMDefault;
```

使用aapt编译资源的时候，如果指定编译选项为library，生成的资源id中packageID为0x00，在初始化应用时，在此处会触发R.java的onResourceLoaded方法，重写了packageID

```
// Rewrite the R 'constants' for all library apks.
SparseArray<String> packageIdentifiers = getAssets(mActivityThread)
        .getAssignedPackageIdentifiers();
final int N = packageIdentifiers.size();
for (int i = 0; i < N; i++) {
    final int id = packageIdentifiers.keyAt(i);
    if (id == 0x01 || id == 0x7f) {
        continue;
    }

    rewriteRValues(getClassLoader(), packageIdentifiers.valueAt(i), id);
}
```

```
try {
    callback = rClazz.getMethod("onResourcesLoaded", int.class);
} catch (NoSuchMethodException e) {
    // No rewriting to be done.
    return;
}

Throwable cause;
try {
    callback.invoke(null, id);
    return;
} catch (IllegalAccessException e) {
    cause = e;
} catch (InvocationTargetException e) {
    cause = e.getCause();
}
```

应用进程生命周期的开始。

```
thread.bindApplication(processName, appInfo, providers, app.instrumentationClass
        profilerInfo, app.instrumentationArguments, app.instrumentationWatcher,
        app.instrumentationUiAutomationConnection, testMode, enableOpenGlTrace,
        isRestrictedBackupMode || !normalMode, app.persistent,
        new Configuration(mConfiguration), app.compat,
        getCommonServicesLocked(app.isolated),
        mCoreSettingsObserver.getCoreSettingsLocked());
```

Activity内部触发启动Activity

```
从Launcher 启动应用程序第一个Activity的过程

->Activity
    public void startActivityForResult(Intent intent, int requestCode, @Nullable Bundle
options){
        Instrumentation.ActivityResult ar =
            mInstrumentation.execStartActivity(
                this, mMainThread.getApplicationThread(), mToken, this,
                intent, requestCode, options);

    }

->Instrumentation

    /**
     *
     * @param who The Context from which the activity is being started.
     *             启动Activity的上下文
     *
     * @param contextThread The main thread of the Context from which the activity
     *                      is being started.
     *            当前进程的ApplicationThread实例，用于IPC
     *
     * @param token Internal token identifying to the system who is starting
     *              the activity; may be null.
     *            当前Activity的标示，是一个IBinder类型
     *
     * @param target Which activity is performing the start (and thus receiving
     *               any result); may be null if this call is not being made
     *               from an activity.
     *            当前的Activity，如果不是从Activity启动，则为空
     * @param intent The actual Intent to start.
     *            实际的Intent
     * @param requestCode Identifier for this request's result; less than zero
     *                    if the caller is not expecting a result.
     *
     * @param options Addition options.
     *
     * @return To force the return of a particular result, return an
     *         ActivityResult object containing the desired data; otherwise
     *         return null.  The default implementation always returns null.
     *
     * @throws android.content.ActivityNotFoundException
     *
     * @see Activity#startActivity(Intent)
     * @see Activity#startActivityForResult(Intent, int)
     * @see Activity#startActivityFromChild
     *
     * {@hide}
     */
    public ActivityResult execStartActivity(
            Context who, IBinder contextThread, IBinder token, Activity target,
            Intent intent, int requestCode, Bundle options) {
        //获取AmS的远程接口，启动一个Activity
        int result = ActivityManagerNative.getDefault()
            .startActivity(
                whoThread, //由contextThread转型来的
                who.getBasePackageName(), //包名
                intent,//实际的intent
                intent.resolveTypeIfNeeded(who.getContentResolver()),
                token,//当前Activity的token
                target != null ? target.mEmbeddedID : null,
                requestCode,
```

```
                0,
                null,
                options);
    }

->获取AmS的远程接口，执行startActivity方法
  /**
   * 通过 ActivityManagerNative.getDefault()方法会返回AmS的远程接口
   * ActivityManagerProxy。
   *
   *
   * @param caller         当前进程的IBinder
   * @param callingPackage 当前进程的包名
   * @param intent         实际的intent
   * @param resolvedType
   * @param resultTo       接受结果的Activity的IBinder
   * @param resultWho      如果是从Activity中启动其他Activity，那么这个值是当前Activity的
   *                       mEmbeddedID，值是在Activity 的attach方法中赋值的。
   * @param requestCode    请求码
   * @param startFlags     启动模式
   * @param profilerInfo
   * @param options        额外的参数
   **/
  public int startActivity(IApplicationThread caller, String callingPackage, Intent intent,
          String resolvedType, IBinder resultTo, String resultWho, int requestCode,
          int startFlags, ProfilerInfo profilerInfo, Bundle options) {
      //通过Binder驱动将方法调用发送到AmS的同名方法
  }
```

ActivityManagerService 内部处理启动Activity的流程

```


  进入AmS内部

  ->ActivityManagerService

  public final intstartActivity(...){
      return startActivityAsUser(caller, callingPackage, intent, resolvedType, resultTo,
          resultWho, requestCode, startFlags, profilerInfo, options,
          UserHandle.getCallingUserId());
  }

  public final int startActivityAsUser(...) {
      //看上去是做一些安全校验....暂时不管
      enforceNotIsolatedCaller("startActivity");
      userId = handleIncomingUser(Binder.getCallingPid(), Binder.getCallingUid(), userId,
          false, ALLOW_FULL_ONLY, "startActivity", null);
      // TODO: Switch to user app stacks here.
      return mStackSupervisor.startActivityMayWait(caller, -1, callingPackage, intent,
          resolvedType, null, null, resultTo, resultWho, requestCode, startFlags,
          profilerInfo, null, null, options, false, userId, null, null);
    }

  ->ActivityStackSupervisor

  final int startActivityMayWait(
          IApplicationThread caller,//启动Activity的进程
          int callingUid, // -1
          String callingPackage, // 当前进程的包名
          Intent intent, //实际的intent
```

```
            String resolvedType,//
            IVoiceInteractionSession voiceSession, //null
            IVoiceInteractor voiceInteractor,//null
            IBinder resultTo, //接受结果的Activity
            String resultWho, //mEmbeddedID
            int requestCode, //
            int startFlags,  //启动模式
            ProfilerInfo profilerInfo,
            WaitResult outResult, //null
            Configuration config,//null
            Bundle options, //额外的参数
            boolean ignoreTargetSecurity, //false
            int userId, //userId
            IActivityContainer iContainer, //null
            TaskRecord inTask){//null

        ...

        // 收集Activity的信息
        ActivityInfo aInfo =
                resolveActivity(intent, resolvedType, startFlags, profilerInfo, userId);

        ...

        int res = startActivityLocked(caller, intent, resolvedType, aInfo,
                    voiceSession, voiceInteractor, resultTo, resultWho,
                    requestCode, callingPid, callingUid, callingPackage,
                    realCallingPid, realCallingUid, startFlags, options,
ignoreTargetSecurity,
                    componentSpecified, null, container, inTask);
    }

    /**
     * 从PackageManager获取将要启动的Activity信息
     *
     **/
    ActivityInfo resolveActivity(Intent intent, String resolvedType, int startFlags,
            ProfilerInfo profilerInfo, int userId) {
        ....
        try {
            ResolveInfo rInfo =
                AppGlobals.getPackageManager().resolveIntent(
                        intent, resolvedType,
                        PackageManager.MATCH_DEFAULT_ONLY
                                | ActivityManagerService.STOCK_PM_FLAGS, userId);
            aInfo = rInfo != null ? rInfo.activityInfo : null;
        } catch (RemoteException e) {
            aInfo = null;
        }
        ....
    }

    final int startActivityLocked(
            IApplicationThread caller,
            Intent intent,
            String resolvedType,
            ActivityInfo aInfo,//上一步中从获取的Activity的相关信息
            IVoiceInteractionSession voiceSession,
            IVoiceInteractor voiceInteractor,
            IBinder resultTo,
            String resultWho,
            int requestCode,
            int callingPid,
```

```
93              int callingUid,
94              String callingPackage,
95              int realCallingPid,
96              int realCallingUid,
97              int startFlags,
98              Bundle options,
99              boolean ignoreTargetSecurity,
100             boolean componentSpecified,
101             ActivityRecord[] outActivity,//null
102             ActivityContainer container, //null
103             TaskRecord inTask) {
104
105         //获取调用者的进程信息
106        ProcessRecord callerApp = mService.getRecordForAppLocked(caller);
107
108        ......
109
110         ActivityRecord sourceRecord = null;
111         ActivityRecord resultRecord = null;
112         if (resultTo != null) {
113             sourceRecord = isInAnyStackLocked(resultTo);
114             if (sourceRecord != null) {
115                 if (requestCode >= 0 && !sourceRecord.finishing) {
116                     resultRecord = sourceRecord;
117                 }
118             }
119         }
120
121         ......
122         //创建目的Activity的ActivityRecord
123         ActivityRecord r = new ActivityRecord(
124                 mService,//AmS
125                 callerApp,//调用方的进程信息
126                 callingUid, //调用方的Uid
127                 callingPackage,//调用方的包名
128                 intent, //实际的intent
129                 resolvedType,
130                 aInfo, //Activity信息
131                 mService.mConfiguration,
132                 resultRecord,//接受结果的Activity
133                 resultWho,
134                 requestCode,
135                 componentSpecified, //是否指定了component
136                 voiceSession != null,
137                 this, //ActivityStackSupervisor
138                 container,
139                 options);
140
141         ......
142
143         err = startActivityUncheckedLocked(r, sourceRecord, voiceSession, voiceInteractor,
144                 startFlags, true, options, inTask);
145
146
147     }
148
149     final int startActivityUncheckedLocked(
150             final ActivityRecord r,           //目标Activity的record
151             ActivityRecord sourceRecord,      //源Activity的record
152             IVoiceInteractionSession voiceSession,
153             IVoiceInteractor voiceInteractor,
154             int startFlags, //启动模式
155             boolean doResume, //是否resume
```

```
156         Bundle options,
157         TaskRecord inTask ) {//
158 
159     ...
160     //一堆代码计算Activity的launchFlags
161     ...
162 
163     //对于从launcher启动的Activity，走到这，获取需要进入的ActivityStack
164     targetStack = computeStackFocus(r, newTask);
165 
166 
167     r.setTask(targetStack.createTaskRecord(getNextTaskId(),
168                       newTaskInfo != null ? newTaskInfo : r.info,
169                       newTaskIntent != null ? newTaskIntent : intent,
170                       voiceSession, voiceInteractor, !launchTaskBehind /* toTop */),
171                       taskToAffiliate);
172 
173     /*
174       //创建一个新的task，并添加到ActivityStack
175       TaskRecord createTaskRecord(int taskId, ActivityInfo info, Intent intent,
176                                   IVoiceInteractionSession voiceSession,
177                                   IVoiceInteractor voiceInteractor,
178                                   boolean toTop) {
179           TaskRecord task = new TaskRecord(
180                                     mService, //AmS的引用
181                                     taskId, info, intent, voiceSession,
182                                     voiceInteractor);
183           addTask(task, toTop, false);
184           return task;
185       }
186 
187     */
188     //在ActivityStack中启动Activity
189     targetStack.startActivityLocked(r, newTask, doResume, keepCurTransition, options);
190 }
191 
192 ->ActivityStack
193 final void startActivityLocked(ActivityRecord r, boolean newTask,
194         boolean doResume, boolean keepCurTransition, Bundle options) {
195     //继续往下
196     mStackSupervisor.resumeTopActivitiesLocked(this, r, options);
197 }
198 
199 ->ActivityStackSupervisor
200 boolean resumeTopActivitiesLocked(ActivityStack targetStack, ActivityRecord target,
201         Bundle targetOptions) {
202       ...
203       if (isFrontStack(targetStack)) {
204           result = targetStack.resumeTopActivityLocked(target, targetOptions);
205       }
206 }
207 
208 ->ActivityStack
209 private boolean resumeTopActivityInnerLocked(ActivityRecord prev, Bundle options) {
210 
211 
212 
213 
214     ...
215     mStackSupervisor.startSpecificActivityLocked(next, true, true);
216 
217 }
218 
```

```
219   ->ActivityStackSupervisor
220   void startSpecificActivityLocked(ActivityRecord r,
221           boolean andResume, boolean checkConfig) {
222       // Is this activity's application already running?
223       ProcessRecord app = mService.getProcessRecordLocked(r.processName,
224               r.info.applicationInfo.uid, true);
225       ...
226       if (app != null && app.thread != null) {
227           realStartActivityLocked(r, app, andResume, checkConfig);
228           return;
229       }
230       if (app == null){
231          mService.startProcessLocked(r.processName, r.info.applicationInfo,true,
    0,"activity", r.intent.getComponent(), false, false, true);
232       }
233   }
234
235   /**
236    * 通知ActivityThread启动Activity
237    **/
238   final boolean realStartActivityLocked(ActivityRecord r,
239           ProcessRecord app, boolean andResume, boolean checkConfig)
240           throws RemoteException {
241     ...
242     app.thread.scheduleLaunchActivity(
243                   new Intent(r.intent),
244                   r.appToken,
245                   System.identityHashCode(r),
246                   r.info,
247                   new Configuration(mService.mConfiguration),
248                   new Configuration(stack.mOverrideConfig),
249                   r.compat,
250                   r.launchedFromPackage,
251                   task.voiceInteractor,
252                   app.repProcState,
253                   r.icicle,
254                   r.persistentState,
255                   results,
256                   newIntents,
257                   !andResume,
258                   mService.isNextTransitionForward(),
259                   profilerInfo);
260     ...
261
262   }
263
264
265   ->ActivityManagerServices
266
267    final ProcessRecord startProcessLocked(String processName,
268           ApplicationInfo info, boolean knownToBeDead, int intentFlags,
269           String hostingType, ComponentName hostingName, boolean allowWhileBooting,
270           boolean isolated, boolean keepIfLarge) {
271       return startProcessLocked(
272               processName, info, knownToBeDead, intentFlags, hostingType,
273               hostingName, allowWhileBooting, isolated,
274               0 /* isolatedUid */,
275               keepIfLarge,
276               null /* ABI override */,
277               null /* entryPoint */,
278               null /* entryPointArgs */,
279               null /* crashHandler */);
280     }
```

```
    //经过一系列的跳转走到下面这个函数
   private final void startProcessLocked(...){
     ...
     if (entryPoint == null) entryPoint = "android.app.ActivityThread";
     ...

     //内部的实现逻辑是建立和zygote的socket连接，拼接了一堆参数丢过去
     //zygote接收到参数后，从自身fork一个进程出来，然后返回pid,新的进程
     //会执行entryPoint的main函数，此处entryPoint是ActivityThread
     Process.ProcessStartResult startResult = Process.start(
                    entryPoint,
                    app.processName,
                    uid, uid, gids, debugFlags, mountExternal,
                    app.info.targetSdkVersion,
                    app.info.seinfo,
                    requiredAbi,
                    instructionSet,
                    app.info.dataDir,
                    entryPointArgs);
   }

    @Override
    public final void attachApplication(IApplicationThread thread) {
        ...
        //获取调用者的进程id
        final long origId = Binder.clearCallingIdentity();
        attachApplicationLocked(thread, callingPid);
        ...
    }

    private final boolean attachApplicationLocked(IApplicationThread thread,
            int pid) {
        //根据进程id取回ProcessRecord
        ProcessRecord app = mPidsSelfLocked.get(pid);
        //获取应用信息
        ApplicationInfo appInfo = app.instrumentationInfo != null
                    ? app.instrumentationInfo : app.info;

        //回到应用进程。执行bindApplicatoin
        thread.bindApplication(
              processName,
              appInfo,
              providers,
              app.instrumentationClass,
              profilerInfo,
              app.instrumentationArguments,
              app.instrumentationWatcher,
              app.instrumentationUiAutomationConnection,
              testMode,
              enableOpenGlTrace,
              isRestrictedBackupMode || !normalMode,
              app.persistent,
              new Configuration(mConfiguration),
              app.compat,
              getCommonServicesLocked(app.isolated),
              mCoreSettingsObserver.getCoreSettingsLocked());

    }
```

```
->ActivityThread
```

```
public static void main(String[] args) {
    ...
    //设置当前进程的nicename，在此时，ddm还无法attach到这个进程中，
    //所以如何debug这个main函数，需要好好想想
    Process.setArgV0("<pre-initialized>");
    //初始化事件循环
    Looper.prepareMainLooper();
    //创建ActivityThread实例
    ActivityThread thread = new ActivityThread();

    //初始化生命周期
    thread.attach(false);

    //设置当前线程的回调函数，此处获取的就是H类
    if (sMainThreadHandler == null) {
        sMainThreadHandler = thread.getHandler();
    }
    //开启事件循环，接收事件
    Looper.loop();
}

private void attach(boolean system) {
    ...
    //获取AmS的远程接口，通知AmS赋予当前应用进程生命周期
    final IActivityManager mgr = ActivityManagerNative.getDefault();
    mgr.attachApplication(mAppThread);
    ...
}

private void handleBindApplication(AppBindData data) {
      ....
        try {
                java.lang.ClassLoader cl = instrContext.getClassLoader();
                mInstrumentation = (Instrumentation)
                    cl.loadClass(data.instrumentationName.getClassName()).newInstance();
            } catch (Exception e) {
                throw new RuntimeException(
                    "Unable to instantiate instrumentation "
                    + data.instrumentationName + ": " + e.toString(), e);
            }
      ....

      Application app = data.info.makeApplication(data.restrictedBackupMode, null);
      .....
      mInstrumentation.callApplicationOnCreate(app);
      .....

}

```