冷启动的定义

冷启动是指应用程序从新开始启动：系统进程在之前还没有创建应用程序进程。

冷启动发生的两种情况：

1. 应用程序自设备启动后第一次启动；
2. 系统终止应用程序后再次启动。

在冷启动开始时，系统有 3 个任务：

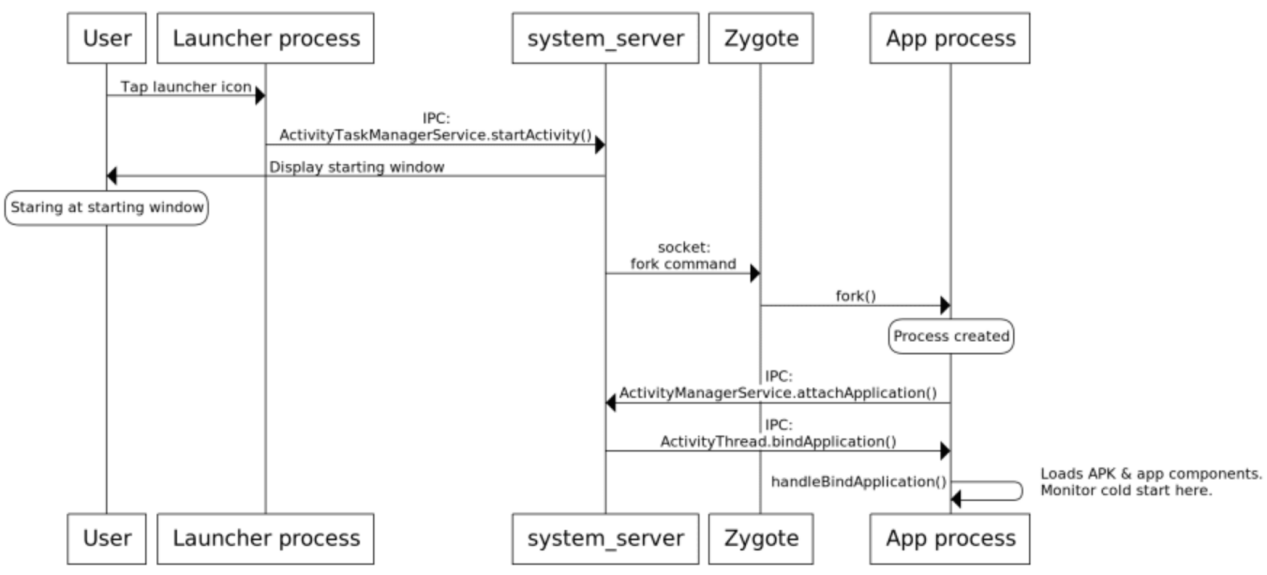
1、加载和启动应用程序；

2、显示启动窗口；

3、创建应用程序进程；

点击桌面应用图标启动APP

1. 桌面进程通过调用Activity.startActivity()进而调用Instrumentation.execStartActivity()；
2. 桌面进程通过IPC跨进程调用system\_server进程的ActivityTaskManagerService.startActivity()；
3. system\_server进程在创建新的app进程之前，首先通过PhoneWindowManager.addSplashScreen()创建一个启动window；
4. 这个启动window就是app进程启动后直到创建第一个Activity并绘制第一帧之前用户所能看见的app样子；
5. 这个启动window的内容就是从将要启动的Activity设置的windowSplashscreenContent和windowBackground加载的；
6. 在启动window展示完毕后，system\_server进程通过ZygoteProcess.startViaZygote()进而调用ZygoteProcess.zygoteSendArgsAndGetResult()，通过socket发送参数给Zygote进程，进而启动新的app进程；
7. 为了启动app进程，系统会派生 Zygote 进程，然后在新进程中加载并运行应用程序的代码。这种方法允许为framework代码和资源分配的大部分 RAM 页面在所有应用进程之间共享；
8. 当系统启动的时候，Zygote进程通过调用ZygoteInit.main()预加载framework代码、资源、共享库、图形驱动等。预加载的动作不仅节约内存还节省启动时间。随后调用 ZygoteServer.runSelectLoop()打开了socket并进行轮询；
9. 当socket接收到fork指令，将调用ZygoteConnection.processOneCommand()方法通过ZygoteArguments.parseArgs()解析指令，最后开始创建新进程Zygote.forkAndSpecialize()；（Android10 通过USAP创建一个已经被forked的Zygote进程池等待被解析；Android11 通过IORap进一步优化了这个过程）
10. app进程创建后将执行 RuntimeInit.commonInit()，它创建了一个默认的UncaughtExceptionHandler。然后执行 ActivityThread.main()；
11. 在ActivityThread.main()方法中做了两件重要的事：Looper.loop()不断循环等待新消息发送到它的 MessageQueue；ActivityThread.attach()通过IPC调用system\_server进程ActivityManagerService.attachApplication()，它让app的主线程准备开始执行；
12. system\_server进程通过调用ActivityManagerService.attachApplication()进而调用 ActivityManagerService.attachApplicationLocked()通过IPC调用app进程的ActivityThread.bindApplication()进而在app进程的主线程调用 ActivityThread.handleBindApplication()。在application相关的创建结束后，才启动相关的activity、service、broadcast。
13. app进程的主线程通过ActivityThread.handleBindApplication()加载所有app的组建：创建AppComponentFactory的子类实例；创建AppComponentFactory.instantiateClassLoader()；调用AppComponentFactory.instantiateApplication()加载Application的子类实例；对于所有声明的 ContentProvider，按照优先级调用AppComponentFactory.instantiateProvider() 创建他们的实例，并调用ContentProvider.onCreate()；最后调用Application.onCreate()；（综上可知，开发者对于ActivityThread.handleBindApplication()之前的流程没有介入的可能，所以app冷启动应该从这里开始）
14. 初始化时机：最早的hook时机在AppComponentFactory类被加载的时候，可以通过给AndroidManifest.xml的application标签增加appComponentFactory属性；在androidx中使用tools:replace="android:appComponentFactory"；使用静态初始化代码款初始化相关动作；只能在Android P+才能使用；app相关的初始化动作最好在Application.onCreate()实现；library相关的初始化最好放在ContentProvider.onCreate()；androidx提供的 App Startup library也是基于ContentProvider进行调度的，通过减少ContentProvider的数量来加快app的启动速度；



冷启动耗时检测

1、范围查找：Systrace获取关键系统函数的耗时情况；细化查找：函数插桩获取业务函数的耗时情况；

冷启动优化

1. 在冷启动过程中将不必要的初始化延迟；
2. 使用ExecutorService将SDK初始化从App.OnCreate()转移到后台线程，因为ExecutorService在Zygote的时候就被预加载，会比kotlin 协程快得多；
3. 根据不同的任务区分不同的线程池，将不同的任务放在不同的线程池执行；
4. 检测项目中使用到的ContentProvider的数量，他会占用启动的初始化时间；
5. 将启动的Service任务延迟或者放到线程池中启动；
6. 不要从主线程执行有关IO文件、数据库、网络的操作；
7. 禁止使用DateTime API，内部会使用流加载当前时间；
8. MultiDex优化：main dex中植入一个LoadDexActivity用于判断当前是否需要另外开个进程进行MultiDex加载；
9. 类的预加载：在线程池中进行类的预加载；
10. 将首页布局优化；