启动优化的意义

对于App来说，也是同样如此，如果用户点击App后，App半天都打不开，用户就可能失去耐心卸载应用。

启动速度是用户对我们App的第一体验，打开应用后才能去使用其中提供的强大功能，就算我们应用的内部界面设计的再精美，功能再强大，如果启动速度过慢，用户第一印象就会很差。

原则：

- 启动任务化管理；要控制入队和监控耗时任务
- 主流程（主进程，主线程，是高优的）要少做要排满（不能有空闲或等子线程的问题）trace可以发现；
- 能子线程的就不要主线程（异步）；
- 能延迟的，就延迟；

什么是任务化

启动时，每一个方法，任务（runnable，callable）sdk init, addview load都是一个任务

设计任务注意的点：

- 一个任务的执行需要知道自己是在哪个进程里需要执行，
- 一个任务的执行需要知道自己是在哪个线程（主，子，HandlerThread），除非必要，尽量在子线程里运行
- 大子线程也要区分子线程的优先级（nice值，cgroup）
- 在子线程时，要注意对其它任务的依赖关系（也就是执行顺序），任务依赖关系（执行顺序）
- 各线程，各任务的相互锁依赖，特别是与主线程的依赖
- 大任务拆分成一组小任务，这样就会让部分小任务延迟，挪移到子线程进行）
- 主业务流程优化（这是一个典型的大任务）
- 三方sdk 出事化，这个一组方法，包括sdk 初始化，设置一些配置，等一些回调，这类的一搬，要在主进程，子线程，并能延迟执行
- UI解析（这是一个典型的要在主线程进行的大任务，可以拆分成N个view容器，先运行框架UI，子内容可以拆分到子线程去初始化UI（等数据你准备好了后），再加入到框架UI中去，在展示给用户）

- 隐示开始执行的操作，class load
- 系统IPC(跨进程)调用（Binder），ActivityThread&H.handleMessage
- 对于一个大的，一定会用到的对像（WebView）要在子线程预先初始化
- 设计独立的线程池管理运行这些任务；

任务中线程使用准则

1、任务中严禁使用new Thread方式，要用任务管理器提供基础线程池供各个业务线任务使用，避免各个业务线各自维护一套线程池，导致线程数过多。
2、根据任务类型选择合适的异步方式：优先级低，长时间执行，HandlerThread；定时执行耗时任务，线程池。
3、创建线程任务管理器提供基础线程池会自动命名，以方便定位线程归属，在运行期Thread.currentThread().setName修改名字。
4、关键异步任务监控，注意异步不等于不耗时，建议使用AOP的方式来做监控。
5、重视优先级设置（根据任务具体情况），Process.setThreadPriority();可以设置多次。

启动任务的切入点：

启动主流程的类的出事方法，主要组件的初始生命周期都切入点，app 启动时要执行的任务都从这个些个切入点开始执行

- Application.attachBaseContext Application.onCreate contentProvider
- logo activity onCreate（路由处理，广告sdk初始化，logo渲染）.onResume.Onstart
- main activity onCreate（主UI解析，主数据加载）.onResume.Onstart
- class static{} {} //class加载时就是执行的
- class{} //class object 构造时

应用启动的类型

冷启动

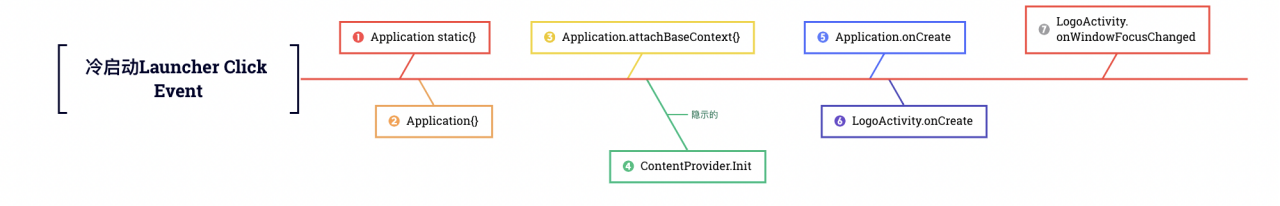
从点击应用图标到UI界面完全显示且用户可操作的全部过程。

特点

耗时最多，衡量标准

启动流程

Click Event -> IPC -> Process.start -> ActivityThread -> bindApplication -> LifeCycle -> ViewRootImpl



统计计时：1--7的时间总和；

热启动

因为会从已有的应用进程启动，所以不会再创建和初始化Application，只会重新创建并初始化Activity。

特点

耗时较少

启动流程

LifeCycle -> ViewRootImpl

统计计时：5--7的时间总和；

ViewRootImpl

ViewRoot是GUI管理系统与GUI呈现系统之间的桥梁。每一个ViewRootImpl关联一个Window，ViewRootImpl最终会通过它的setView方法绑定Window所对应的View，并通过其performTraversals方法对View进行布局、测量和绘制。

方法：

主题切换

使用Activity的windowBackground主题属性预先设置一个启动图片（layer-list），在启动后，在Activity的onCreate()方法中的super.onCreate()前再setTheme(R.style.AppTheme)。

优点

- 使用简单。
- 避免了启动白屏和点击启动图标不响应的情况。

缺点

- 治标不治本，表面上产生一种快的感觉。
- 对于中低端机，总的闪屏时间会更长，建议只在Android6.0/7.0以上才启用“预览闪屏”方案，让手机性能好的用

户可以有更好的体验。

注意主线程的空闲时间监控

注意主线程的掉帧（大message）

SharedPreferences优化（IO，线程优化）

用mmkv无感替换，注意mmkv的初始化

类预加载优化

在Application中提前异步加载初始化耗时较长的类。

如何找到耗时较长的类?

替换系统的ClassLoader，打印类加载的时间，按需选取需要异步加载的类。

注意：

- Class.forName()只加载类本身及其静态变量的引用类。
- new 类实例 可以额外加载类成员变量的引用类。

WebView启动优化

- 1、WebView首次创建比较耗时，需要预先创建WebView提前将其内核初始化。
- 2、使用WebView缓存池，用到WebView的时候都从缓存池中拿，注意内存泄漏问题。
- 3、本地离线包，即预置静态页面资源。

启动阶段不启动子进程

子进程会共享CPU资源，导致主进程CPU紧张。此外，在多进程情况下一定要可以在onCreate中去区分进程做一些初始化工作。

闪屏页与主页的绘制优化

- 1、布局优化。
- 2、过渡绘制优化。

关于绘制优化可以参考Android性能优化之绘制优化。

典型问题：

启动白屏（黑屏）长

启动无反应

启动时卡顿（反应慢，不跟手）

特殊问题：

长效机制；

线上监控

报警管理

线下监控