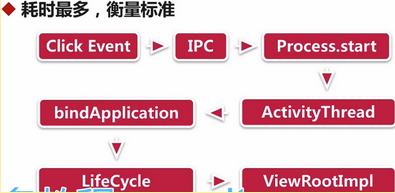
# **启动优化**

## **1.启动的三种方式**

### （1）冷启动



优化方向: Application和Activity生命周期

### （2）热启动

最快，后台切到前台

### （3）温启动

较快LifeCycle

## **2.启动时间测量方式**

### 1.adb命令

**adb shell am start -W packagename/packagename.MainActivity  
然后运行后出现三个时间  
ThisTime:最后一个Activity启动好使  
TotalTime:所有Activity启动耗时**

**说明：如果有启动页的话就是启动页和主页面启动的总耗时  
WaitTime:AMS启动Activity的总耗时**

**总结：**

* **时间耗时排序**

**ThisTime< TotalTime< WaitTime**

* **优缺点**
* **线下使用方便，不能带到线上**
* **非严谨、精确时间**

### 手动打点的方式-->（启动时埋点，启动结束埋点，二者差值） 1.开始时间为 application中执行attachBaseContext()方法为第一步 2.结束时间为 Feed的第一条展示的时间也就是Activity显示第一条数据的时间 （误区：认为 onWindowFocusChange这个方法是activity第一帧）

总结：

1. 精确可带到线上，推荐使用
2. 避开误区采用Feed第一条展示

## **3.工具类使用**

### 1.traceView

（1）traceView优缺点

优势

* 图形的形式展示执行时间，调用栈等
* 信息全面包含所有线程

缺点

* 运行时开销严重，整体会变慢
* 可能会带偏优化方向

（2）使用方式

Debug.startMethodTracing("")  
Debug.startMethodTracing()

生成文件在SD卡：Android /data/packagename/files



重点关注：call Chart 和 TopDown

### 2.systrace

（1）特点

* 结合Android内核的数据，生成html报告
* API 18以上使用，推荐TraceCompat

（2）优点

轻量级开销小，直观反映cpu利用率

（3）使用

方式一

1. **进入Android/Sdk/platform-tools/systrace目录下**
2. **可以配置一些参数，类似于通过Android Device Monitor抓取时步骤2配置的显示信息，若不选择则默认全部抓取**

**-o： 指定文件输出位置和文件名**

**-t： 抓取systrace的时间长度**

**-a： 指定特殊进程包名（自己加Label时必须加上）**

**eg: python systrace.py -t 10 -o /Users/Downloads/boot.html -a gaosi.com.learn**

1. **Chrome浏览器（必须）。在地址栏输入[chrome://tracing](chrome://tracing/)命令，然后将生成的trace.html文件拖进来，或者通过load按钮导入**

方式二、

TraceCompat.beginSection("beginSection")

TraceCompat.endSection()

（4）cputime和walltime区别

cputime：代码消耗cpu的时间（重要指标）举例：锁冲突

walltime：代码执行时间

## **AOP优雅获取方法执行耗时**

### (1) AOP特点

* 针对同一类问题进行处理
* 无侵入添加代码

### (2)Aspectj使用

1.添加依赖

classpath 'com.hujiang.aspectjx:gradle-android-plugin-aspectjx:2.0.10'

apply plugin: 'android-aspectjx'

implementation 'org.aspectj:aspectjrt:1.9.5'

2.JoinPoint🡪可以作为切面的地方

* 函数调用、执行
* 获取设置变量
* 类初始化

3.PointCut->带条件的JoinPoint

4.Advice->要插入代码的位置

* Before：PointCut之前执行
* After：PointCut之后执行
* Around：PointCut之前、之后都执行

语法：详解网址：https://www.jianshu.com/p/d32a2453786e

@Before("execution(\* android.app.Activity+.on\*\*(..))")

public void onActivityCalled (JoinPoint joinPoint) throws Throwable {

...

}

* Before：Advice具体插入位置
* execution:处理JoinPoint的类型

JoinPoint类型分为两种 1：call(插入在函数体里面) 2：excution(插入在函数体外面)

* (\* android.app.Activity+.on\*\*(..))" 匹配规则
* onActivityCalled:要插入的代码

## **异步优化**

核心思想：子线程分担主线程任务，并行减少时间

问题：

1. 不符合异步要求(比如：weex初始化必须在onCreate完成)
2. 需要在某阶段完成

Eg:放在子线程初始化，不知道什么时候初始化完成，主线程调用该sdk的方法可能子线程还没有初始化完成

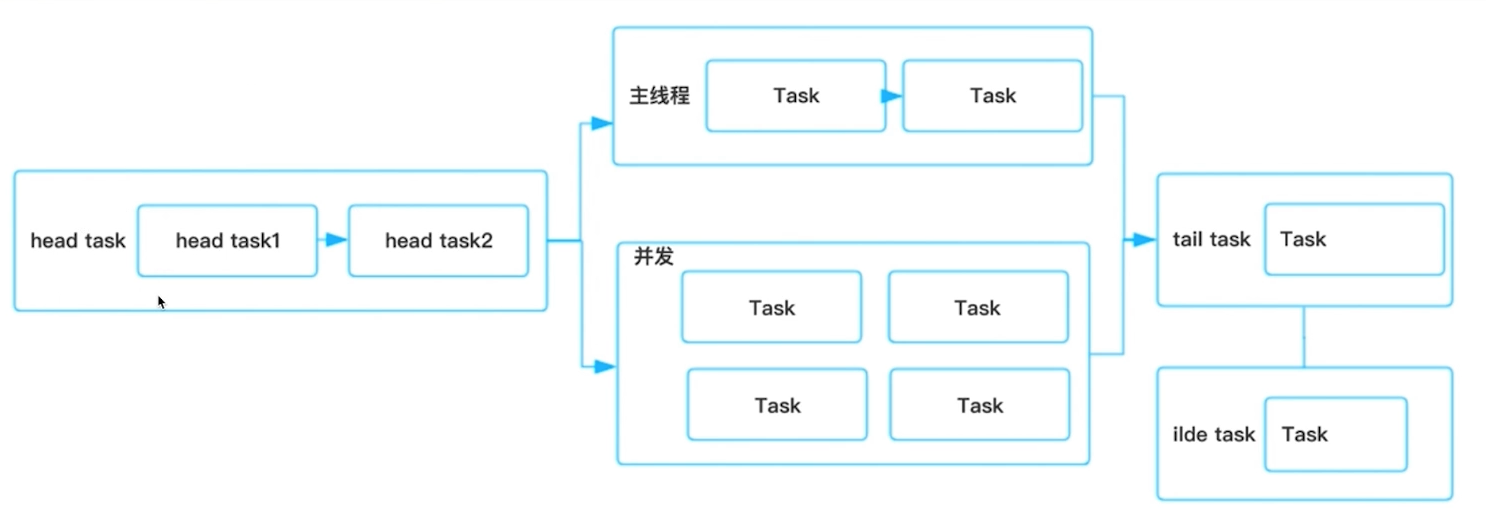
#### **异步优化的最优解—》启动器**

核心思想：充分利用cpu多核，自动梳理任务顺序

1、启动器流程：

1. 代码Task化，启动逻辑抽象为Task
2. 根据所有任务依赖关系排序生成一个有向无环图
3. 多线程按照排序后的优先级依次执行

2、流程图：



### 更优秀的延时初始化方案

#### **1.常规延迟初始化的痛点**

* 时机控制不准;比如：postDelay()
* 导致Feed卡顿 比如界面展示后延时初始化操作

#### **2.更优方案—》核心思想：对延时任务进行分批初始化**

* 利用IdleHandler特性，空闲执行