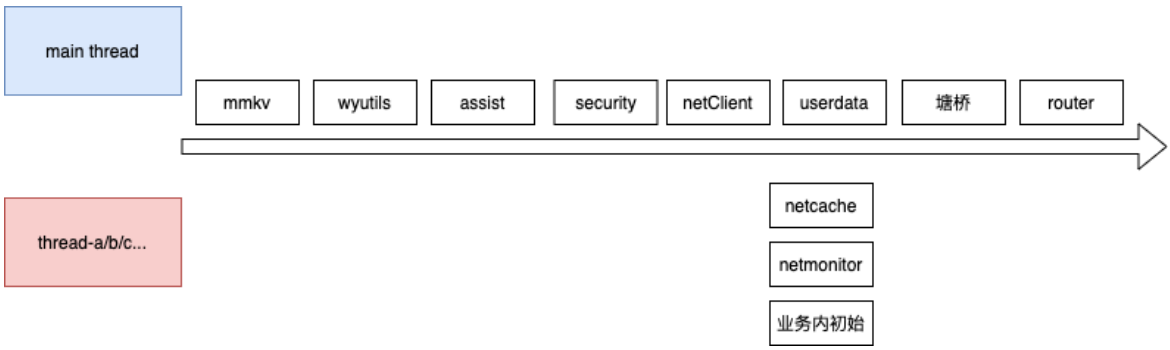
项目背景

当前工程在启动流程做了以下处理, 针对每个初始项包装为独立的Runnable, 以进程为维度进行任务执行的区分, 根据是否可延时加载, 区分同步和异步加载, 当前的缺陷在于, 当初始化项存在复杂依赖的情况下, 为了保证依赖执行顺序, 将被依赖项均放于主线程串行执行, 再保证被依赖项执行完成后, 再异步执行无依赖关系的任务项, 导致没有利用好每个线程. 同时, 由于不存在显示依赖配置, 导致后期维护畏手畏脚, 在增加初始任务的时候, 只敢在上面堆叠, 而不会对启动顺序进行一个梳理和调整. 下图为当前微医生主进程上的初始流程



优化方向

基于此, 异步启动器的设计核心就是, 将初始项包装并暴露配置, 开发同学仅需要关注对应初始项任务的优先级及其依赖关系, 底层根据配置的任务行成有向无环图, 做拓扑排序, 行成pipeline任务有序执行, 实现并行效率最大化.

这个设计方向在微信模块化演进以及手淘的启动优化分享中均有讨论到

设计目标:

- 任务抽象, 显式声明任务依赖, 便于管理
- 底层针对任务排序与包装, 可统一调度并进行任务执行观测
- 合理利用线程, 减少整体项目的启动时间

api设计

任务抽象

对于上层来说, 仅需要维护针对任务的描述即可, 当前初期设计api如下, 由于我们工程支持多进程, 针对进程任务的设计会做组概念的包装

```
interface ITask{
    /**
     * 执行工作
     */
    fun run()

    /**
     * 是否运行在主线程上
     * @return Boolean
     */
    fun runOnMainThread() : Boolean

    /**
     * 当前task依赖的tasks
     * @return List<Class<out Task>>
     */
    fun dependsOn(): List<Class<out Task>>?

    /**
     * 异步执行任务的情况下, 主线程是否需要等待当前任务完成
     * @return Boolean
     */
    fun needWait() : Boolean

    /**
```

```
     * 是否必须运行在主进程上
     * @return Boolean
     */
    fun runOnlyInMainProcess() : Boolean

    /**
     * 执行线程约束
     * @return ExecutorService
     */
    fun runOn() : ExecutorService

    /**
     * 线程优先级
     * @return Int
     */
    @IntRange(from = Process.THREAD_PRIORITY_FOREGROUND.toLong() ,to = Process.THREAD_PRIORITY_LOWEST.toLo
    fun priority() : Int
}
```

执行调用

```
        TaskDispatcher.init(this)
        val taskDispatcher = TaskDispatcher.createInstance()
            .addTask(ATask())
            .addTask(BTask())
            .addTask(CTask())
            .addTask(DTask())
            .addTask(ETask())
            .addTask(FTask())

        taskDispatcher.startTasks()
        taskDispatcher.await()
```

参考

1. 张绍文 - Android高手课 - 启动优化
2. 微信 - 模块化实践
3. 阿里巴巴 - Alpha