概述

启动时间，从刚开始的十几秒，到现在接近秒开的水平，前后经历了三个优化阶段，断断续续花费了几个月的时间。

- 优化前
  点击图标，没有任何反应，过一会儿，出现了启动界面，又过了一会儿进入主界面。整个过程大概需要10s。点击图标后，没有及时响应，这个体验不好。
- 优化后
  点击图标，立刻出现启动界面，大概一秒左右进入主界面，已经接近秒开的水平，从项目的实际角度讲，这已经是优化的极限了。

注：测试手机： 三星s8

测试工具准备

logcat 日志输出，过滤 Displayed

```
021-06-10 16:19:12.927 1202-1424/? I/ActivityManager: Displayed com.qizhidao.clientapp/.home.HomeActivity: +1s82
```

上面输出的日志表明，HomeActivity 绘制完成，使用了 1s823ms。total 表示该进程从创建到 HomeActivity 绘制完成使用了 2s906ms

当然你还需要

- 一个可以统计某几行代码，某个方法，或几个类，执行时间的工具
- 初始化的代码分散在各个模块，需要去正确调用。如 AutoInject
- 性能优化相关的工具，查询Android官方说明
- StrictMode使用，可以定位到具体的代码行。日志过滤 StrictMode

当然，启动过程时间越短越好，哪怕cpu 100% 也不关心，关心的是时间，StrictMode 就非常好用了。

App 的启动流程知识储备

关于App的启动流程，网上的资料多如牛毛，这里就不多说了。下面的讨论都是基于冷启动的。

知识准备

官方说明

https://developer.android.com/topic/performance/vitals/launch-time

我们直觉印象

- 我们看到的桌面，是一个Launcher,也是一个Activity，点击图标，执行startActivity...
- 系统需要fork一个新的进程。
- 关于 ActivityThread
- ....

在冷启动开始时，系统做三件事

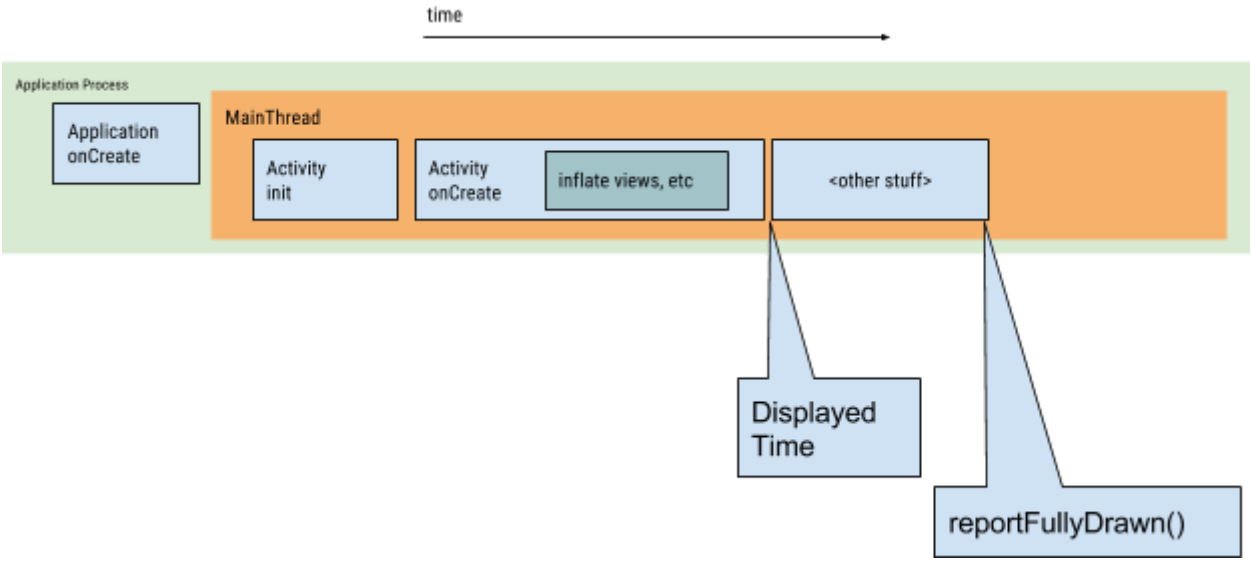
- 加载并启动应用
- 在启动后立即显示应用的空白启动窗口（注意这里，优化点包含这里的优化）
- 创建应用进程

上面的三个步骤，除了第二点，其它的是没法干预的。

当应用进程创建之后，应用进程开始接手工作，也是我们可以优化的点

- 创建应用对象
- 启动主线程
- 创建主Activity
- 扩充视图
- 布局屏幕
- 执行初始绘制

当应用进程完成初始绘制之后，就会替换掉当前显示的后台窗口为主Activity，此时用户就可以使用应用了



优化过程

启动优化的本质是，缩短冷启动的时间，这个时间越短越好。

优化前，点击图标后，没有及时响应，可以通过设置主Activity 的 windowBackground来解决。启动时替换掉空白窗口，让用户知道，系统已经开始响应点击事件。

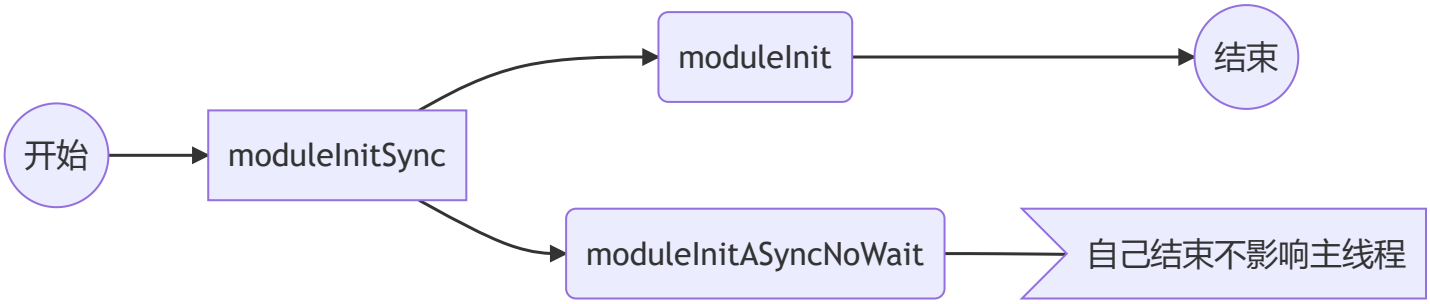
通过上面的启动流程图可知道，在Application创建完成之后，才开始Activity 的初始化，因此，Application 中的事情不能做的太多，或者不能耗费太多的时间在上面。
另一个方面，一个Activitty 的创建，大概需要 500ms 的样子，在这个时间段，可以异步做其它事情，这段时间不能闲着。

整个优化零零碎的进行了四个月，前后发布了三个版本，最终达到要求。

优化前的结构

application 初始化任务，分为三个任务模块，每个模块说明如下：

- moduleInitSync - 最先执行，其它的任务依赖该任务完成，必须在主线程执行
- moduleInit - 该任务完成后，appplication 中的任务结束，在子线程中。
- moduleInitASyncNoWait -- 一个单独的线程中，不影响启动过程



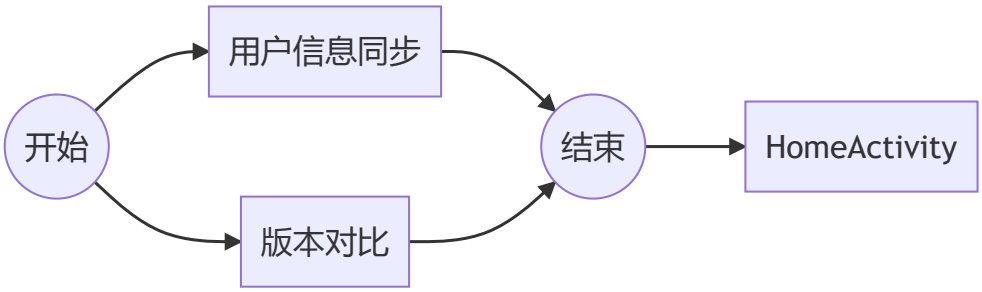
moduleInitSync+ moduleInit 二个任务比较耗时，执行完要好几秒，这也是点击了图标，半天没有反馈的原因所在。

Application 任务执行完，就完了？嘿，这还没完

SplashActivity

到这里，用户终于看到了一个启动界面，但是SplashActivity中还有其它的业务逻辑

- 用户信息同步
- 版本对比



而且这两个任务需要和服务端交互，时间不定，虽然设置了一个最大结束时间10s。

SplashActivity业务做完后，终于进入HomeActivity界面了。

整个启动过程简化如下：

第一次优化

在业务没有彻底弄清楚前，贸然改动，会造成不可预知的后果。第一轮优化不动业务逻辑。而是找出耗时点

抓大放小

sqlcipher

通过工具，我们发现了 sqlcipher 初始化很慢,初始化的时间竟然要2s。而且应用是多库结构，应用中至少有三个库，这三个库竟然是一起初始化的！也就是说这三个数据库初始化完都要个5~6s 的时间

> 在定位到数据库初始化代码的时候，我一度以为是 GreenDao 初始化太慢，然后查询相关资料也没有发现有类似的问题，还准备换成 Room ， 最后经过一段时间的折腾，终于看到了sqlcipher。

优化的两个方向：

- 将多库的一次性初始化，改成按需初始化。
- 仔细阅读 sqlcipher 相关文档，找出官方提供的优化方案，进行配置优化

> 经过反复测试，关闭了内存的安全验证,将初始时间控缩短到1s以内。当然也尝试了其它的方式，如分页大小的设置，密码复杂度的验证...反复折腾！！！

```
"PRAGMA cipher_memory_security = OFF"
```

UserAgent 获取引发的问题

下面的这行代码有没有问题：

```
var userAgent = WebSettings.getDefaultUserAgent(context)
```

初看好像是没什么问题，但上面的代码会引起 WebView 的初始化，而WebView的初始化是非常耗时的，以至于，你可以单独的在一个线程去处理。在加上使用了 x5 内核，双内核引发的灾难，更不用说了。

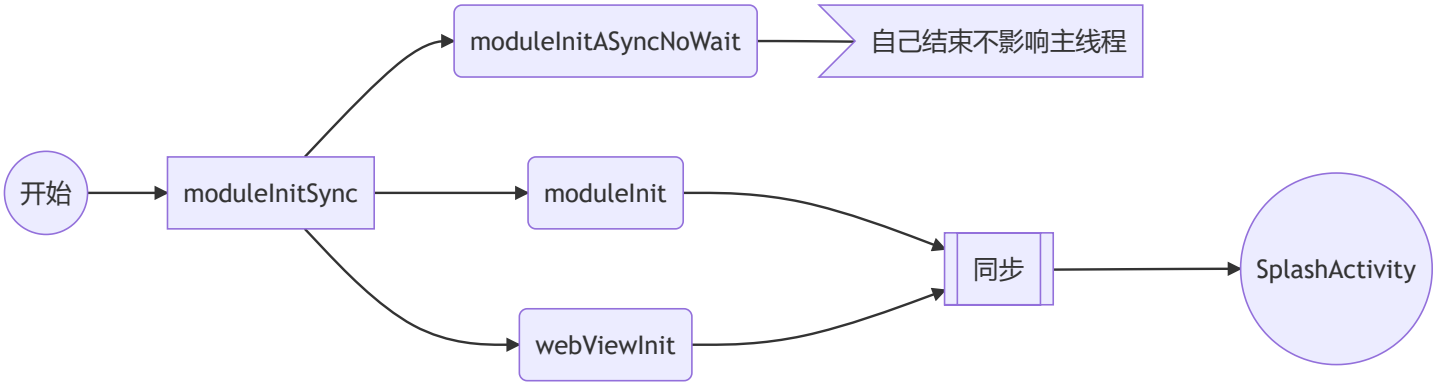
优化方案：
使用一个新的任务模块，专门处理WebView 的初始化

第一次优化的结果

- 整理了moduleInitSync 任务，将一些不必要的初始化代码，放到其它模块，尽可能精简该任务。
- 将webView 初始化分离出一个新任务。
- 数据库按需初始化，并对参数进行了优化配置
- 主Acticity的windowBackground 设置

经过第一轮优化，初始化时间缩短了一半，大概需要5~8秒启动时间

第一轮优化工作结束
Application 中任务情况为：



第二次优化

第二次优化，依然是不改变业务逻辑。方向是能省一点时间就省一点时间吧，能优化一行代码就优化一行吧

如这些细节

> HashMap -> ArrayMap 或者 SparseArray

我们知道 一个Activity 由系统去创建，而这个创建时间大概需要 500ms ~ 1S ,其实，这个时间我们可以用起来。

砍掉 SplashActivity

将SplashActivity 的业务（用户信息初始化和版本对比升级）拿出来，创建一个新的任务——SpalshTask。并放到 Application 中初始化，当 SplashTask 任务结束后，发消息给 HomeActivity

- 需要改造HomeActicity ,splashTask 任务完成后，才加载具体的业务逻辑代码
- SplashActivity 中版本升级涉及到存储权，下载存储位置由外部位置放到内部。当然存储权限后面进行了大改，这是后话了。

对 Application 中的任务进行进一步的精细化调度，简称为配平任务

在 moduleInitSync 任务中只留下了必要的初始化代码，其它的统统移到其它任务中。任务执行的时间尽可能一致。

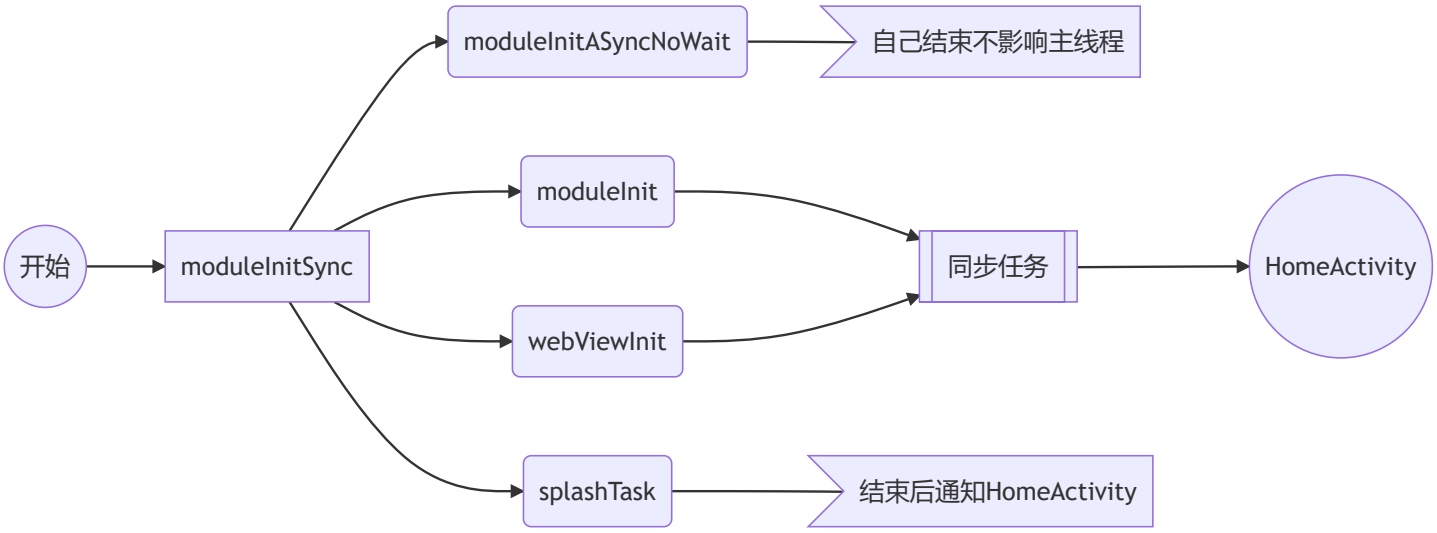
最终的整理

第二次对任务调度进行了更精细化的操作，去掉SplashActivity，创建splashTask，将版本对比和同步用户信息业提前，省去了SplashActivity创建时间。

该版本启动的时间 控制到 3~5s。如果网络不好，那就呵呵了，该版本作为优化的阶段性胜利，上架了。

问题是，启动的时间不可控，快的时候3s，慢的时候 3+10s。取决了splashTask任务时间，甚至登陆和不登陆的时间不一样，第一次使用和第二次使用冷启动时间不一样。

splashTask - 用户信息同步和版本对比



具体的代码优化细节，这里不在详细说了。

第三次优化

结论，本次优化结束后，达到预定的目标，这个也是能优化的天花板，毕竟sqlcipher 初始化都要 1s。在三星 s8上，稳定的冷启动时间在 2.3s 左右，其它手机接近秒开。

统一WebView的内核问题，解决userAgent问题

WebView 的内核统一为x5。

考虑到除非是 手机系统升级，userAgent 值是不变的，我们可以缓存该值，下次直接拿缓存数据，所以，userAgent的初始化时机可以滞后了，在webview初始化完成后在获取该值。

> 用户第一次安装时，使用 System.getProperty("http.agent") 先给一个默认值，然后替换成更具体的。

业务逻辑变更

用户信息的同步以及版本的对比，不再影响主进程。

> app 在启动时读取数据库中的用户信息数据，同步用户信息任务为独立的任务，同步成功后，在通知相关的信息更新。

版本对比，从splashTask 中拿取出来，成为一个单独的任务，不影响主进程

于是 splashTask 中，只剩下用户信息的初始化。这个任务是可控的，所以整个启动过程是可控的。

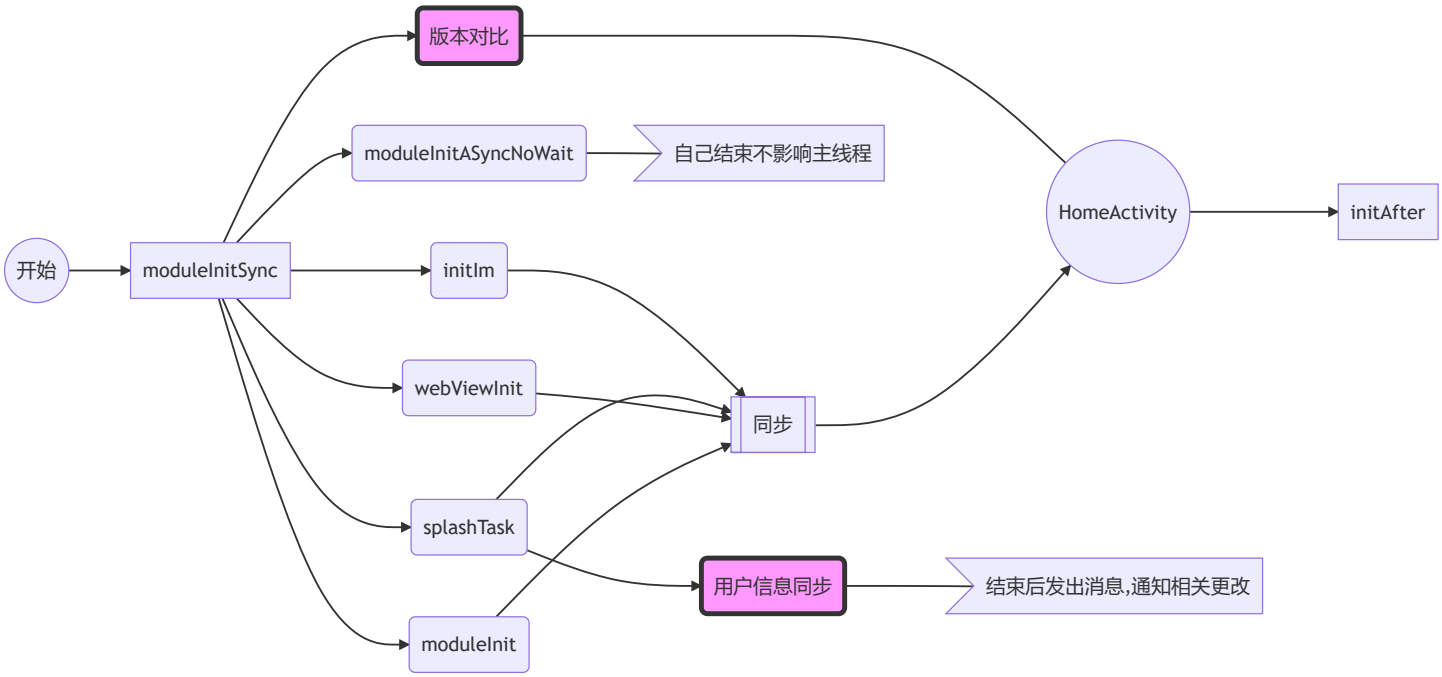
关于百度地图的定位初始化

百度地图，初始化时间大概是500ms ,由于sdk的限制这个初始化过程不能放到子线程中进行，所以，将新建一个 initAfter 模块，作为最后初始化的任务。

关于 IM 初始化

Im 初始化涉及到数据库的初始化，整个过程大概要 700ms，新建一个任务模块 initIm.

于是初始化任务过程为：



尾声

启动优化零零碎碎的进行了 4个月的时间，中途由于其它开发任务的介入。暂停了一段时间，在完成 5月开发任务后，终于腾出时间，完成了第三个阶段的优化开发。第三个阶段的优化非常顺利，得益于业务逐渐熟悉，优化的方向非常明确。

从最初启动在10s以上到现在稳定在2s 左右，启动速度提升了 80%，在主流手机上，去壳的情况下接近秒开，这是在优化前没有想到的。

总的来说，第一阶段抓大放小，不改业务，第二阶精细任务调度，优化相关的每一行代码，去掉没必要的时间损耗，多查阅相关资料，反复测试，也没有动业务流程。第三阶段，才开始优化业务流程，任务可以做到更加精细化的调度。

新发现的问题：**HomeActivity渲染的好慢**