[SCA] Application生命周期分发组件与Startup结合使用文档

1.Application生命周期分发

目前供应链内几个APP均采用多Module形式开发，不同的业务模块对应不同的子Module。这样的开发模式可以将不同业务模块的代码彼此隔离，互不影响，但是实际开发时存在这样的场景：

一些子module需要监听Application的生命周期并进行一些(反)初始化的操作。

现状：目前是将这部分(反)初始化代码下沉统一写在BaseModule的BaseApplicaiton.java中，这就导致BaseApplication.java内的代码较多且杂，需要同时负责BaseModule和所有业务子Module的(反)初始化，导致BaseApplication.java类的职责不清晰且代码耦合。

解决方式：针对上面的问题，可以接入Application生命周期分发组件。该组件可以将Application的生命周期分发至各个业务子Module，分发后BaseApplication只需要负责BaseModule的(反)初始化等操作，而业务子Module的(反)初始化由Module自己处理。

Application生命周期分发组件的接入方式可参考：[SCA] Application生命周期分发SDK使用介绍

2.提升APP的启动时间

app冷启动时，先创建application，然后创建action为launcher的activity。这个启动流程由系统负责执行，像创建application、创建activity等这些操作由系统执行，它的执行时间决定我们无法干预。但是在这个启动流程中会执行Application的onCreate()方法，一般我们开发时都会自定义一个BaseApplication类继承自系统的Application类，并在onCreate()中执行一些初始化操作，如果onCreate()方法中执行的操作耗时较多，必然会拖慢APP的启动时间。

现状/可优化点：为了提升APP的启动时间，有必要对Application的OnCreate()进行优化，减少该方法的耗时。

解决方式：先将Application.onCreate()中执行的操作做一下分类，哪些操作是需要阻塞主线程优先执行；哪些操作及时性要求不高，是可以放在子线程中执行的。对任务进行分类后，就可以接入启动框架，利用启动框架将这些任务分别放在主线程和子线程执行。

启动框架的接入方式可参考：[SCA] Startup接入文档

3.实现细节

3.1 先分发生命周期

先利用Application生命周期分发组件，优化BaseApplicaiton的代码。具体的，将BaseApplication的生命周期分发至各个业务子module中，各个Module只需要关心负责自己模块的(反)初始化操作即可；

3.2 优化BaseApplication的onCreate()

在3.1的基础上，BaseApplication的OnCreate()方法中就只包含了BaseModule需要初始化的方法。然后将这些方法分类，利用启动框架将这些方法分别放在主/子线程执行；

3.3. 优化业务子Module的'onCreate()'

同理，在3.1的基础上BaseApplication的onCreate()已经分发至各个业务子Module了，查看这些业务子module的的onCreate()是否有耗时操作，若有，也是利用启动框架将这些方法放在主/子线程执行；

考虑到业务子module一般不会存在较多耗时操作，同时也考虑到代码的复杂性，我们目前只需要做3.1和3.2即可，3.3可暂时不处理。

4.优化应用启动时间的效果

利用Startup框架将Application.onCreate()中的同步任务和异步任务分开执行，同时结合一些其他的优化方法(可参考第5小节)对代码进行修改后观察效果如下：

可以利用adb命令查看启动时间：adb shell am start -W packagename/launcherActivity

字段含义：

ThisTime 该activity启动耗时

TotalTime 应用自身启动耗时=ThisTime+应用application等资源启动时间

WaitTime 系统启动应用耗时=TotalTime+系统资源启动时间

下面表格中的时间均以adb命令执行后打印的**TotalTime**为参考来做对比。

OPS PDA:

adb命令：adb shell am start -W com.shopee.spx.pda/com.shopee.spx.pda.login.view.LoginActivity

启动方式	优化前(ms)	优化后(ms)
冷启动	2707	1809
	2721	1705
热启动	123	115
	147	141

Driver APP:

adb命令：adb shell am start -W com.shopee.fms/com.shopee.spx.login.ui.activity.SplashActivity

启动方式	优化前(ms)	优化后(ms)
冷启动	3410	2672
	3703	2602
热启动	121	142
	125	135

WMS PDA:

adb命令：adb shell am start -W com.shopee.wms.pda/com.shopee.wms.pda.SplashActivity

启动方式	优化前(ms)	优化后(ms)
冷启动	6086	6047
	6048	6108
热启动	142	155
	141	158

WMS PDA的启动瓶颈不在于是否有将Application.onCreate()中的部分任务改为异步任务，而在于NetMonitor的初始化开启了一个子进程(考虑将开启子进程的时机延后来提高启动速度，大概可以提高2-3s左右，待优化)。

TWS:

adb命令：adb shell am start -W com.shopee.twms/com.shopee.twms.ui.activity.SplashActivity

启动方式	优化前(ms)	优化后(ms)
冷启动	2744	1596
	2650	1636
热启动	123	131
	118	114

Service Point:

adb命令：adb shell am start -W com.shopee.servicepoint/com.shopee.servicepoint.SplashActivity

启动方式	优化前(ms)	优化后(ms)
冷启动	5505	4944
	5389	4716
热启动	160	170
	168	163

Service Point的启动优化后，还存在一个启动瓶颈：早期为了提升RN性能，在应用启动时就开始创建了RN进程，这里可以考虑在应用启动完成后再立刻去创建RN进程，这样可以加快应用启动速度，且因为用户人工操作不会这么快，应用启动后再创建RN进程对RN性能影响应该不大的。同时由于代码中存在跨进程通信，由于跨进程通信时如果不存在目标进程，会自动创建目标进程，所以如果在应用启动过程中存在跨进城通信，最好在通信前先判断目标进程存不存在，如果不存在看是否有必要通信，如果无必要就无需跨进程通信了。(如果将开启RN进程的时机延后，应用启动速度大概可以提高2-3s左右，待优化)。

总结：优化对冷启动耗时比较明显，对热启动基本无影响(因为热启动不会走Application的onCreate()，所以这个结果是符合预期的)。

5 关于提升应用启动时间的一些建议

1.尽量简化Application.onCreate()中方法的调用，一些对及时性要求不高的方法可以考虑移到子线程中去做，或者延后执行等应用启动完成后再执行；

这样可以减少应用启动期间主线程需要做的事情，更快地启动我们laucher Activity，呈现页面；

2.如果项目集成了Arouter，记得在项目中应用Arouter插件：classpath "com.alibaba:arouter-register:x.x.x" (因为不应用该插件项目也能正常运行，所以集成时容易被遗忘)；

应用该插件，可以极大的减少Arouter的初始化时间，即ARouter.init(Application application)方法的运行时间。

以Ops PDA为例，未应用该插件Arouter初始化时间未856ms，应用该插件后Arouter初始化时间为16ms；

以WMS为例，未应用该插件Arouter初始化时间未1015ms，应用该插件后Arouter初始化时间为36ms；

3.尽量不要在Application的onCreate()方法中创建子进程，如果要创建子进程，尽量延后至应用启动完成后再创建；

因为启动新的子进程会重新走Application的创建等一系列流程，会极大的拖慢应用的启动时间；