移动端“开发、测试、优化、发布”标准

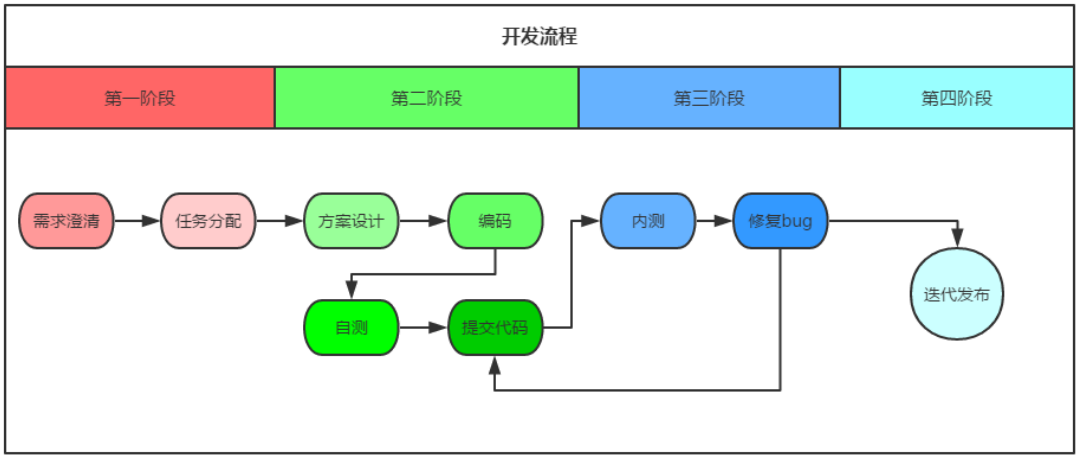
修订历史记录

版本	日期	编写人	审核人
1.0	2017.06.14	宋院林，匡凌波	匡凌波
1.1	2017.06.23	宋院林，匡凌波	匡凌波
1.2	2017.07.27	宋院林，匡凌波	匡凌波

目 录

一、开发流程

APP 的开发流程标准如下：



1.需求澄清

这一步是决定迭代成败的关键，是每个迭代的方向。参与迭代开发的同事（PM，RD，QA，运维）一定要把这个迭代要达到的目标、各需求，及相关关键资源理解清楚，否则，任何一个环节都会导致迭代失败。在需求澄清阶段各角色人员对需求不清楚，一定要及时提出自己的疑问，把需求完全搞清楚，千万不能模棱两可、任意发挥。

需求标准：

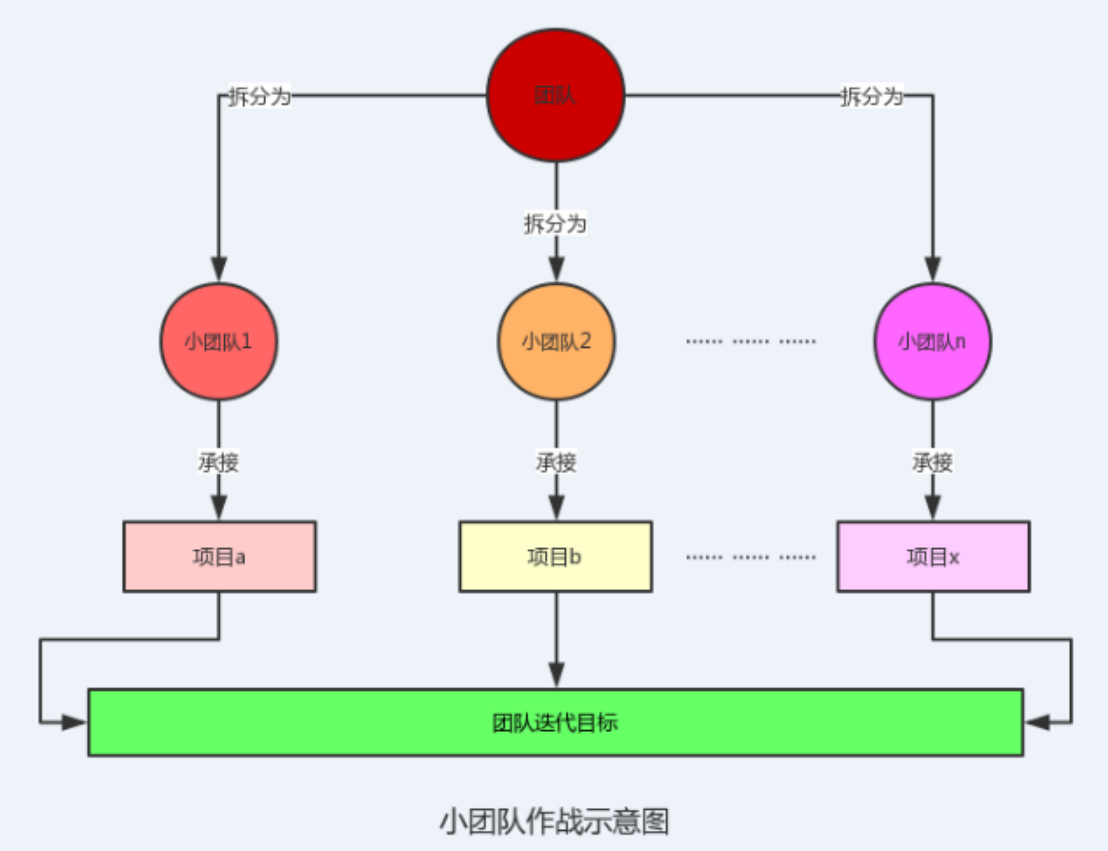
（1）交互稿要完备；
（2）需求对异常边界有清楚明确的界定；
（3）需求有明确的验收标准；
（4）ued 设计稿。

2.任务分配

针对团队成员特点，合理分配迭代任务，有助于提升迭代效率。可以从以下几个方面去做：

（1）小团队作战

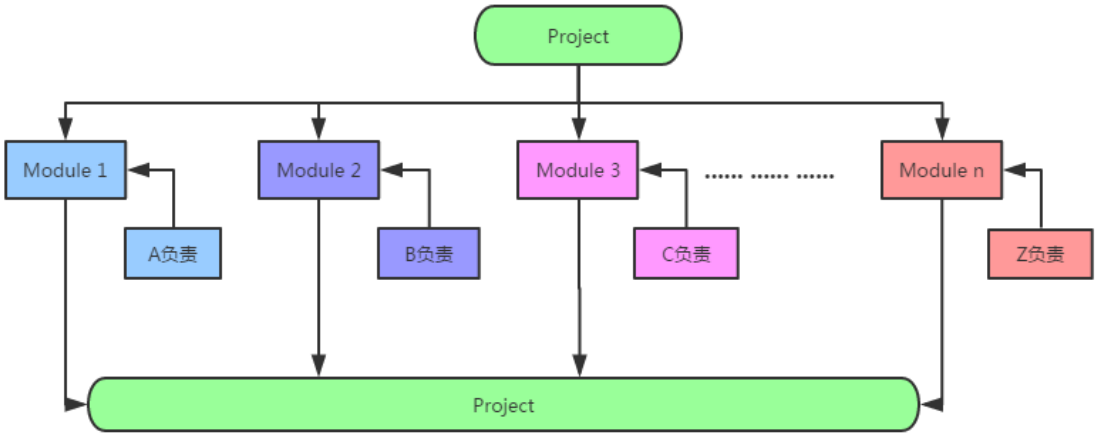
如果一个团队人员较多(如十几人),可拆分为几个小团队承接不同的项目,同时给每个小团队在每个迭代指定一个经验稍丰富的同事来主导这个迭代的开发,这样能更好地调动团队各成员的主观能动性,真正实现团队成员的自组织化,如下图。



小团队作战示意图

（2）按功能模块分配

现在的开发模式基本都是每个人独立开发某些功能模块,同事甲和同事乙不会同时开发同一个功能模块，这样可防止代码冲突、提高开发效率。所以在分配任务时，我们可以按照功能模块进行划分。当然，在不同的迭代周期内，不同成员之间可以交换模块来开发，但要保证在同一个周期内，同一个模块只有一个人开发。



3.方案设计

需求澄清、任务分配后，每个人需要从以下几个方面去做：

（1）功能细化

把自己所负责的模块细化。如，登录注册模块，就需要细分为登录、注册两个小模块，然后针对小模块进行功能开发。

（2）功能内联

一个模块细化为多个小模块进行开发后，需要考虑小模块之间是否有关联、如何关联。如，登录、注册两个功能之间是有关联，注册成功后，如何调用登录功能进行操作。

（3）功能外联

理清自己负责的模块与其他模块之间的关系，然后要设计好其他模块调用本模块功能的 API，遵循简洁、扩展性好的原则。

（4）异常情况

每次在 coding 之前，尽可能多想一下自己开发的模块存在哪些边界情况、异常情况，怎么处理这些边界、异常情况。如，OOM、数组越界、NullPointer、内存泄露及其他异常问题，需要我们思考这些异常情况产生的可能原因，解决方案等。

4.编码

（1）Coding 之前

开发之前，想清楚要实现的是什么功能，怎么来实现，有哪些方案来实现这个功能,在多个方案拿捏不准的时候，需发起站立会讨论并得出结论。想清楚后再开始 coding，这样能起到事半功倍的效果，否则，很可能导致开发的功能不是需求真正要实现的功能，严重时要返工，同时会影响迭代进度。

（2）Coding 过程中

编码时，要遵循代码规范，充分考虑代码的可读性（包括类、方法、变量的命名，逻辑清晰等）、健壮性（即代码的稳定性、易扩展性）。

开发过程中，我们经常会用到第三方开源框架或 sdk（下文统称“sdk”）。这个时候并不是随便找一个相关 sdk 就直接拿来用。我们要从以下几个方面去考虑：

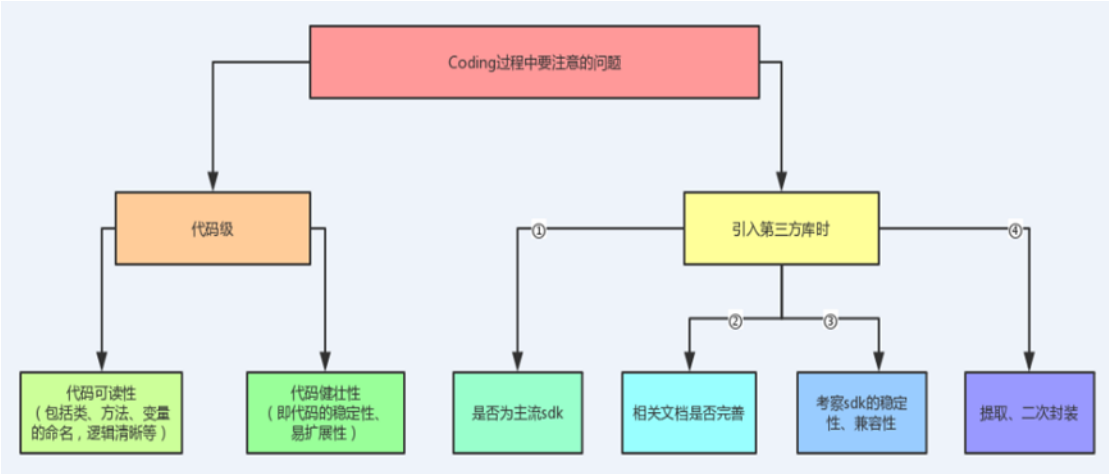
第一，所用 sdk 是不是现在主流的（就是：这个领域用的人比较多）；

第二，sdk 相关文档是否完善，这对于后期找问题原因有帮助；

第三，跑一下 sdk 的 demo 以及相关的评论，考察下 sdk 的稳定性、兼容性；

第四，可以考虑抽取原开源 sdk 中我们真正用到的部分，然后再次封装，这样才能完全掌控这个 sdk，后续有任何问题也便于解决。

用了第三方 sdk 后，要知会本项目团队的其他成员，避免其他成员又一次添加相关的 sdk，引起不必要的资源浪费和冲突隐患。



（3）Coding 完成

开发完成后，要进行代码、性能上的优化，一定进行充分完整地自测，保证开发的功能没有问题且不能影响其他同事开发的功能，然后提交代码到服务器，每次提交需写明完成了什么功能或解决了什么问题，便于代码回滚、查找问题原因。

（4）进度把控

把控好项目进度，关系到迭代任务能否按时完成。那么，如何把控项目进度？
第一，项目成员要明确自己在本次迭代的故事有哪些。
第二，项目成员要给自己的每个故事预估开发完成点数。
第三，每次晨会，项目成员要详细说明自己所负责的故事开发进度，对比预估点数，是否有延迟。
第四、项目成员每天都要跟进自己的开发进度，尽量提前；项目负责人每天要跟进项目开发进度。

5.自测

切记，功能开发完成后，一定要在尽量多真机（覆盖主流分辨率，主流 rom）上测试（只有真机才能测出真正的问题）。
原则：自己开发的功能模块要重点测试，其他功能模块也要测试，主要看自己开发的功能对其他模块是否有影响，尽量减少与其他模块之间的耦合，增强自己功能模块的内聚。
步骤：
（1）Self Review，自我审核代码。主要是实现的基本流程、方案、内存是否存在问题。
（2）设计完整的、常规性自测用例。一定要覆盖自己开发的全部功能。针对具体的需求和验收标准来设计。
（3）设计异常情况的自测用例。如：快递小哥正在巴枪上用 4G 网络上传快件信息，此时有来电，快件信息没有上传成功，应当如何处理？
（4）设计边界情况的自测用例。如：网络请求相关，有网络，但网络很弱的情况；有 wifi，但 wifi 需要登录验证的情况等。

6.提交代码

（1）前提：提交代码前，一定要编译、自测通过才能提交。
（2）原则：现在全公司都在使用 git 管理代码，一定要遵循 pull→add→commit→push 的顺序来提交代码，不要漏传、多传文件，避免他人更新代码编译出错。
（3）规范：一定要按照规范提交代码，详见[提交规范]。

7.自动化测试

这个步骤需要做自动化测试的人员通过自动化测试工具，如 Appium、

UiAutomator 等，编写相关的 TestCase 代码来执行自动化测试，具体可见[指引与实践---自动化测试]。

8.内测

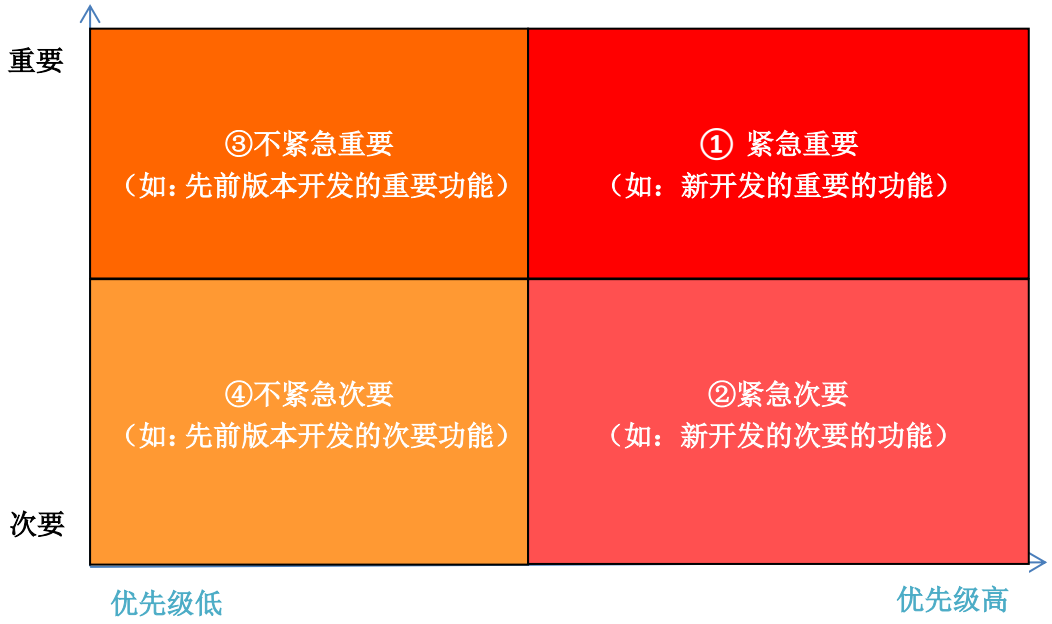
这一步主要由公司专门负责测试的同事来完成，要遵循的原则：

（1）测试用例要完备

测试用例不仅要包含完备常规测试用例，还要尽可能多地包含异常情况测试用例。

a.常规测试用例

常规测试用例就是针对 App 常规操作（即 App 主流程、基本功能等）而设计的用例，这是最基本的用例，一定要囊括 App 的所有功能，不要遗漏，同时，测试用例针对 App 的功能要有优先级和重要、次要之分。



如上图所示，测试人员设计的常规测试用例需要按照上图进行优先级排序，按照此图来安排测试顺序。

b.异常测试用例

有了常规测试用例作为基础，App 在常规操作下一般不会有什么问题出现，出现的问题大部分发生在异常情况，所以测试人员尽可能多地设计异常情况测试用例也尤为重要。

（2）bug 描述要准确

bug 四要素：名称、描述（包含复现条件、路径）、等级、复现率。

a.bug 名称

一般由：“页面名称+触发事件+出现的 bug”组成，如：在“我”页面点击“妥投历史”闪退。

b.bug 描述

测试人员在提 bug 时，一定要把 bug 描述清楚，其中最重要的就是操作步骤（即复现条件和路径）要详细，让开发人员看到 bug 就明白如何操作来复现这个 bug，这样就能更快地定位问题、解决 bug。

c.bug 等级

测试人员提出 bug，一定要进行严格地等级划分，严重、一般、轻微。如，“闪退、卡顿、功能未完成等”属于“严重”等级，“UI 效果未完全按照效果图实现，按钮大小、颜色有偏差等”属于“一般”等级，其他不影响主流程、主功能、不影响 UI 效果的可归属于“轻微”等级。

d.bug 复现率

bug 有必现的，也有偶现。

必现的 bug，复现率即为 100%，这类 bug 很容易重现，也利于开发者定位问题。

偶现的 bug，可能操作 10 次，出现 1 次，也可能操作几十次、上百次也难得出现一次，这类 bug（很可能与内存泄露有关）需要开发人员通过监控日志进行跟踪监控、同时需要测试人员持续关注和跟进。

所以，测试人员在提 bug 时，一定要注明 bug 的复现率。

（3）测试报告要详细

测试人员测试完成，最终要输出详细的测试报告。包括：

a.安装包版本、生成时间。
b.测试结论。
c.测试用例数、bug 数。
d.bug 比例。
e.不同等级 bug 各自的占比。
f.所有 bug 所属的功能模块分布。
g.闪退的 bug、功能上 bug、UI 上的 bug 要分别注明。

如下截图，是详细的测试报告模板

“丰小弟”测试报告-20170727												
安装包版本	V1.0.2					安装包时间	2017/7/27 17:30					
测试结论	不通过											
测试用例数					bug数				bug率(测试用例数/bug数)			
50					35				70%			
bug总数		严重bug			一般bug			轻微bug				
35		3(8.6%)			20(57.1%)			12(34.3%)				
bug所属模块分布		模块名									bug数	
		注册模块									1	
		登录模块									2	
		收件模块									5	
		派件模块									8	
		消息推送									3	
		个人中心									6	

测试报告总览

测试报告 | 闪退bug | 功能bug | UI bug

闪退bug			
序号	bug描述	bug等级	bug复现率
1	打开“扫一扫”功能，app闪退	严重	100%
2	点击派件，此时来了推送消息，app闪退	严重	100%

9.兼容性测试

这是 App 开发要重点关注的一块内容。IOS 开发在兼容性这一块还比较好做，毕竟 iPhone 手机的屏幕尺寸以及 IOS 系统版本都是很容易做的。Android 就完全不一样，除了 google 原生，国内各大厂商都有自己深度定制的 ROM，这就给 Android 的兼容性适配增加了难度，所以兼容性测试这一块很重要。

具体操作请见[兼容性测试]。

10.性能测试

性能测试，主要是为后续的优化提供方向。主要包括：启动时间、内存消耗、CPU 占用率、GPU、耗电量、流量等方面。拿到这些数据与行业的平均水平、最优水平进行对比，找出差距，有针对性地的进行性能优化。

具体操作请见[性能测试]。

11.修复 bug

（1）顺序

开发在修复 bug 时，可按照“严重 bug→一般 bug→轻微 bug”的优先级来 Fix。

（2）原则

在 Fix bug 时，绝不能改了旧 bug，引起新 bug。

（3）步骤

Fix→Self Test→Commit。bug fix 后，一定要“充分自测”，“充分自测”，“充分自测”，重要的事情要说三遍！

12.优化

具体见第三部分[指引与实践]。

13.迭代发布

（1）版本发布

测试通过后，要发布正式版本的 App 或 SDK，同时附带测试报告，邮件告知相关项目组的相关成员。

（2）迭代回顾

形式不限，可以是项目组相关成员组织讨论会，也可以是每个人自己对这个迭代进行总结。
主要内容：

a.出现了哪些不好的情况，是怎么处理的，下个迭代如何避免此类问题再次发生。

b.有哪些好的方面，有哪些技术我们可以做技术沉淀，可以继续保持、发扬光大。

二、规范

1.编码规范

（1）命名规范

a.命名不要以下划线或美元符号开始，也不能以下划线或美元符号结束。如，_name、$Object、name_、object$都是不合法的命名。

b.类的命名使用 UpperCamelCase 风格，遵循驼峰的形式。如，MainActivity、DownloadFileManager 都是正确的命名形式。

c.方法名、参数名、成员变量、局部变量的命名使用 lowerCamelCase 风格，同样要遵循驼峰的形式。如，getRouteLabel()、downloadFileByUrl(String fileUrl)、loginUserId、localValue 都是正确的命名形式。

d.常量命名全部使用大写，单词间用下划线隔开，常量的语义要力求表达清楚，不要嫌名字太长。如，MAX_THREAD_COUNT 是正确的，而 MAX_COUNT 则语义不清，要杜绝这类命名。

e.数组的中括号是数组的一部分，所以数组的定义如下 int[] args，而不是 int args[]。

f.包名统一使用小写字母，点号分隔符之间有且仅有一个英文单词。如，com.sf.push.sdk 就是正确的包名命名方式。

g.在类中使用了设计模式，在类名定义时最好加上具体的设计模式，这样有利于其他人理解此类的架构设计思想。如，public class ImageLoaderFactory。

h.枚举类名最好带上 Enum 后缀，枚举成员命名要全大写且各单词间用下划线隔开。如，枚举类 NetworkStatusEnum，枚举成员 ENABLE_STATUS。

（2）coding 规范

a.代码缩进采用 4 个空格，不要使用 tab 字符，如果使用 tab 缩进，必须设置 1 个 tab 为 4 个空格。

b.if/for/while/switch/do 等保留字与括号之间要加 1 个空格。

c.左小括号与字符间不要加空格，同样，右小括号与字符间也不要加空格。如，if (a >= b)就是规范的写法。

d.大括号的使用，如果大括号内为空，则可写成{}，如果是非空的代码块，则：
1）左大括号前加 1 个空格且不换行。
2）左大括号后换行。
3）右大括号前换行。
4）右大括号后有 else 等代码块则不换行，表示终止的右大括号后必须有换行。
如：

```
private void init() {
    // 左大括号前必须加1个空格且不换行，左大括号后换行
    if (a > b) {
        System.out.println("hello");
    // 右大括号后有else等代码块则不换行
    } else {
        System.out.println("world");
    // 在右大括号后直接结束，则必须换行
    }
}
```

e.单行字符数限制不超过 120 个，超过就必须换行，换行时要遵循如下规则：
1）第二行相对第一行缩进 4 个空格，从第三行开始，不再继续缩进。
2）运算符与下文一起换行。
3）方法调用的点符号与下文一起换行。
4）多个参数超长，在逗号后换行。
5）小括号前不要换行。

f.在给方法定义和传入参数时，参数之间的逗号后必须加 1 个空格。如，
init(“a”, “b”, “c”);

g.方法体内的变量定义、执行语句、不同的业务逻辑之间需要插入 1 个空行，相同业务逻辑之间不需要插入空行。

h.源文件编码格式为 UTF-8。

（3）注释规范

a.注释的作用是辅助说明，力求准确、精简。

b.类、类的属性、类方法的注释必须使用/**内容*/格式，而不是//xxx 的格式。

c.所有类都必须添加创建者和创建日期。

d.方法内部单行注释，在被注释语句上方另起一行，使用// 注释，并且注释内容与注释符//之间要有 1 个空格；方法内部多行注释，使用/* */。

e.所有的抽象方法，参数、返回值、异常情况、该方法的用途，以及子类的实现或调用方法，也需要说明。

f.所有枚举类型字段要有注释说明每一个数据项的用途或意义。

g.代码修改了，注释也要随之修改。

（4）异常日志规范

a.日志的输出，必须使用条件输出形式。如，

```
if (LoggerUtils.isEnableDebug()) {
    Log.d(TAG, “log content….”);
}
```

b.针对运行时异常 RuntimeException，可以通过预先检查来规避，而不是通过 catch 来处理，如，IndexOutOfBoundsException、NullPointerException 等异常。

eg：

```
if (object != null) {
   …… …… ……
}
```

c.生产环境禁止 debug 日志输出。

（5）提交规范

a.代码及时提交，不要和服务器上的代码有太大的差别。

b.提交代码前必须先更新，一般遵循 pull→add→commit→push 的顺序。

c.提交代码前，一定要在本地编译、充分自测通过后，方可提交，本地没有验证的代码不要提交。

d.提交代码时发生冲突，一定要解决冲突后才能提交。

e.提交代码时一定要保证服务器上的版本是正常的，当服务器上的版本有错误时，先不要提交代码，待服务器上的版本正常后再提交。

f.不要提交自己不明白的代码。

g.提交代码，一定要遵循如下 commit 日志规范，这样方便查看每次提交修改的内容，也便于出现问题时版本回溯。
1）New：新增文件、新增功能。如，New：新增会员登陆功能。
2）FixBug：修复 bug、完善功能。如，FixBug：修复“首页”图片加载时 OOM 的问题。
3）Other：优化代码、去掉无用文件。如，Other：优化“快件查询模块”的代码。

（6）模块责任人制度

我们所讲的模块，一般就是每个人独立负责的一个功能块，可以是大模块，也可以是小模块。如，扫描模块、订单模块，相对“登录”模块来说，就是大模块。

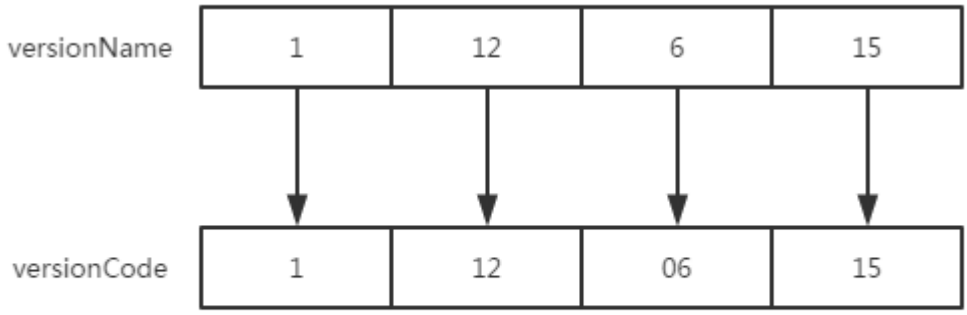
模块责任人制度的含义：一个模块 M 从无到最终完成上线，都是 A 同学在负责（A 即为模块 M 的责任人），但在后期的迭代中，A 去做其他模块开发，B 同学接手模块 M 的开发和完善，此时，在 B 提交代码后，需要告知 A，由 A 及其他相关人员进行 code review，确保模块 M 的持续稳定。

模块责任人制度的意义：提升每个人对自己所负责的模块的责任感，让每个人能感受到自己的劳动成果，同时打造一个“干净”的代码环境。

2.版本规范

（1）App 版本规范

versionName 采用“四段式”表示，versionCode 由 versionName 去掉数字之间的点符号演变而来。如，versionName = “1.12.6.15”，versionCode = 1120615，versionCode 的生成规则如下：



备注：versionCode每一段均采用两位数表示，若最高位小于10，用一位数表示，其他位小于10，高位补0.

这种规范的好处：versionCode 的值依赖 versionName，不需要每次都去查询上一次发版的 versionCode 值。

（2）SDK 版本规范

我们的 SDK 有快照版本和 release 版本，在发布版本时快照版本尽可能与 release 版本一致，如，release 版本是 1.0.6.8，快照版本就应该是 1.0.6.8-SNAPSHOT。

3.架构设计规范

（1）架构的概念

架构，就是平时开发 App、系统所采用的框架，能够清晰描述 App、系统的层次以及各层次间的关系。

（2）架构的原则

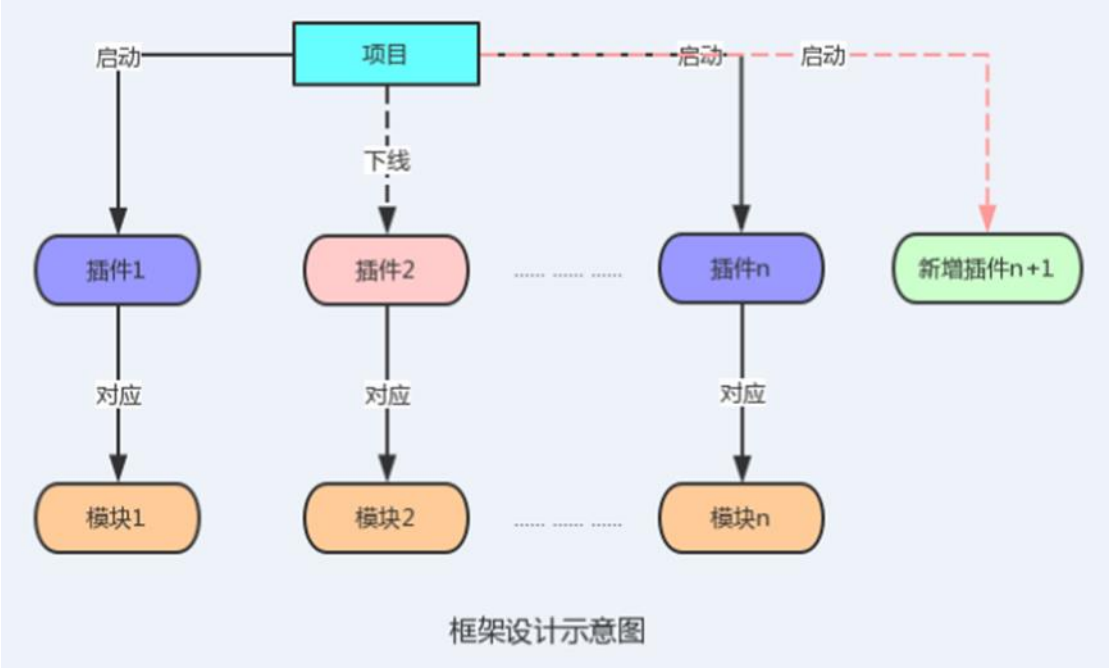
我们在设计或选用架构时，要遵循以下原则：
a.可维护性
即架构是在持续维护、更新的，是每一阶段是稳定、可用的。
b.可扩展性
我们对原有架构有较深地理解，可以在原有架构的基础上，进行功能的扩展、延伸和完善。
c.有相关的文档说明
文档是对架构的原理、使用的说明，有助于使用者更快地熟悉此架构，也是我们设计或选择架构的原则之一。

（3）架构的规范

a.插件化/组件化的思想

在当前开发领域，基本都会考虑插件化或组件化开发，因为插件化组件化开发便于项目管理和维护，对于后期要下线或新增某个模块，都比较方便。

一个项目包含很多不同的模块，我们可以考虑把这些模块作为工程的单个组件或插件来开发，这样，便于项目的管理和维护。



框架设计示意图

b.基类的封装

activity 和 fragment 都会有很多重复的操作，我们可以提供一个 BaseActivity 和 BaseFragment 的基类，让所有的 activity 和 fragment 都继承这个基类，这样能大大减少一些重复的操作。

c.必要的注释

在编码过程中，必要的注释不要少，这有助于自己和其他人阅读代码，提高代码的易读性，快速理解实现原理。

d.提供统一的数据入口

无论是 MVP、MVC，还是 MVVM，提供一个统一的数据入口，这有利于代

码更加易于维护。例如，我们在开发时，提供一个 DataManager 类，无论是 http、sharedPreference，还是 dataBase，我们都在 DataManager 里操作，这样有利于数据相关代码的维护。

e.少用继承、多用组合

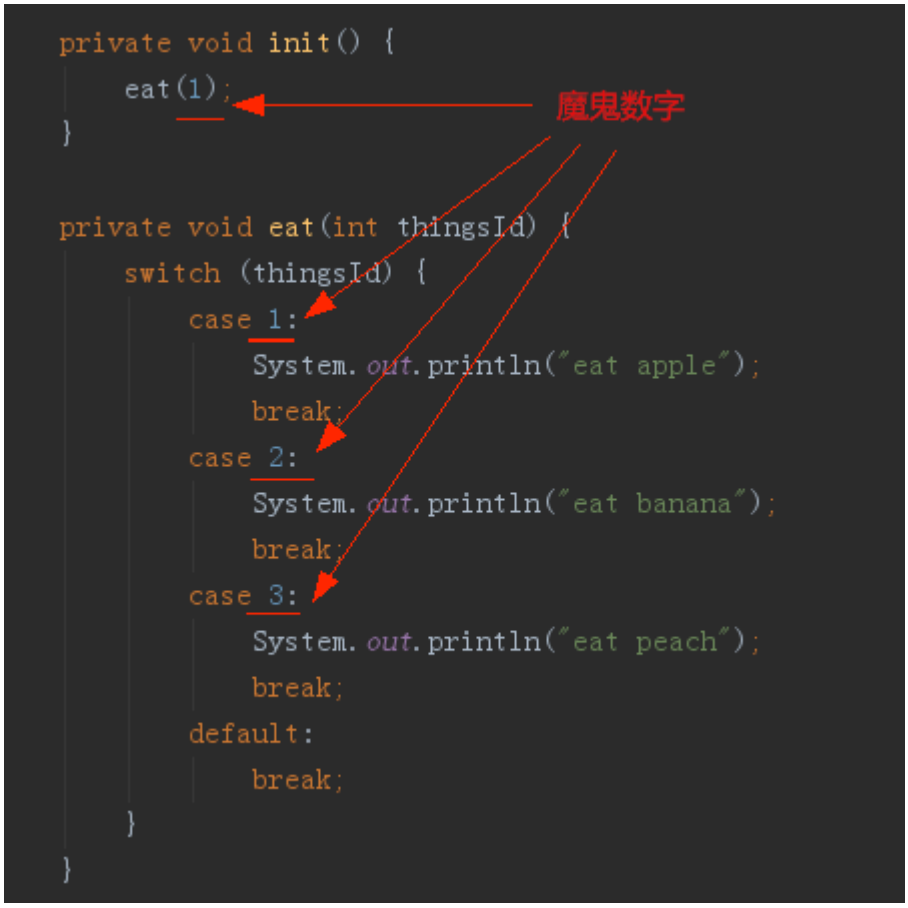
因为 java 最多只能继承一个父类，这就有很大的限制，而且不利于解耦。对于一些共有的属性、方法，可以抽取出来作为一个公共的 Common 类，其他的类可以通过 Common 类的实例来调用其中的方法。

f.提取方法，去除重复代码

对于一些方法，在多个地方都会用到，需要考虑将这些方法提取出来，作为一些工具类供其他代码调用，如，httputils 用于网络请求、pullRefresh 用于下拉刷新、imageloader 用于图片加载、download 用于文件下载、ToastUtil 用于在不同的 rom 统一 toast 样式，这样可以大大减少重复的代码，是代码更简洁、更易于维护和扩展。

g.正确的命名

不要使用魔鬼数字/颜色值/尺寸值/字符串来命名，正确的命名方式有利于提高代码可读性、可维护性。如，




```
private void init() {
    eat(1);
}

private void eat(int thingsId) {
    switch (thingsId) {
        case 1:
            System.out.println("eat apple");
            break;
        case 2:
            System.out.println("eat banana");
            break;
        case 3:
            System.out.println("eat peach");
            break;
        default:
            break;
    }
}
```

规范的做法应该是：

```java
private static final int APPLE_ID = 1;
private static final int BANANA_ID = 2;
private static final int PEACH_ID = 3;

private void init() {
    eat(APPLE_ID);
}

private void eat(int thingsId) {
    switch (thingsId) {
        case APPLE_ID:
            System.out.println("eat apple");
            break;
        case BANANA_ID:
            System.out.println("eat banana");
            break;
        case PEACH_ID:
            System.out.println("eat peach");
            break;
        default:
            break;
    }
}
```

规范命名示例

h.减少模块之间的耦合

可引入 Dagger2 框架，使用代码自动生成创建依赖关系所需的代码，减少模块化的代码，降低模块之间的耦合度。

i.引入响应式编程

使用 RxJava 和 RxAndroid 这些响应式编程，可以大大减少逻辑代码。

j.使用事件总线

如 EventBus，可以在“数据层”发送事件，在“视图层”订阅事件，当数据

发生变化时，视图层更新 UI，这样既可以减少回调，又使得架构层次更清晰。

k.增加日志打印

我们可以使用开源的 logger 或者我们自己的 log sdk 打印相关的日志，用于检查、定位错误。但是，我们在 release 发版时，要把 log 日志打印关闭，所以我们的日志打印需要有一个 flag 来控制日志的打开、关闭。

三、指引与实践

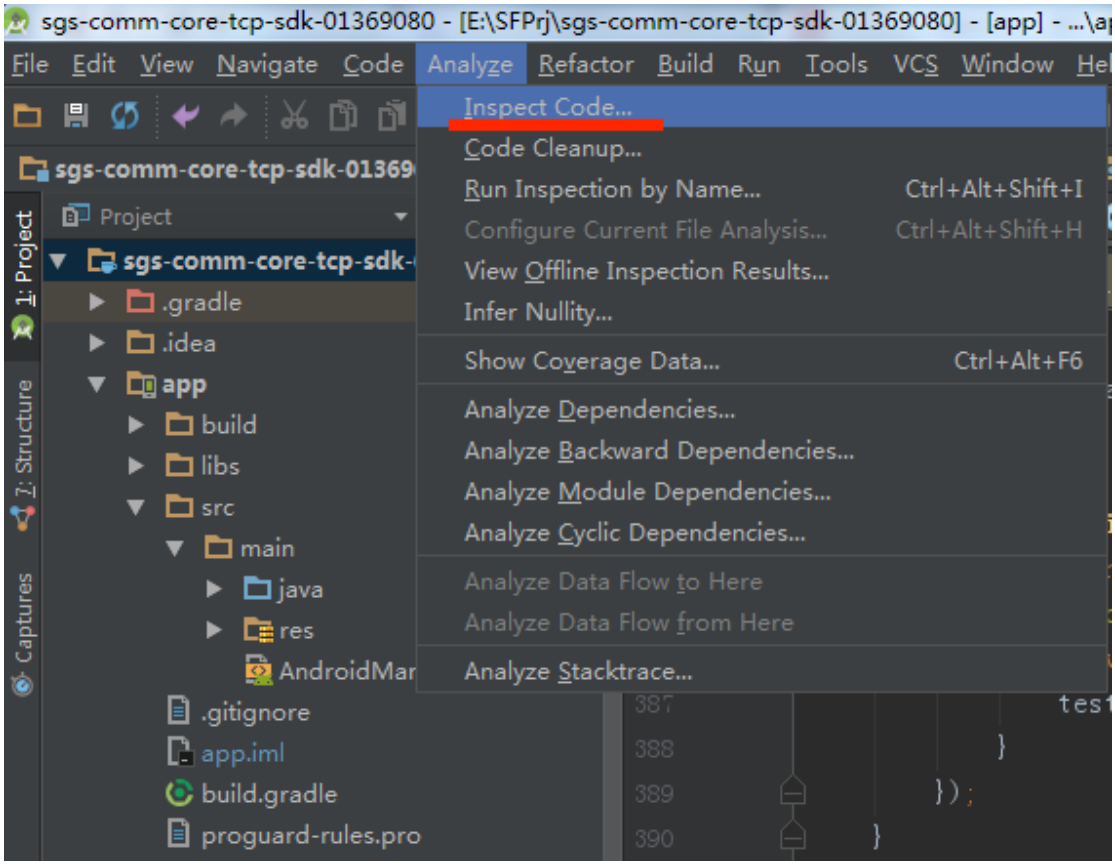
1.代码优化

（1）安装 checkStyle 或 FindBugs 插件

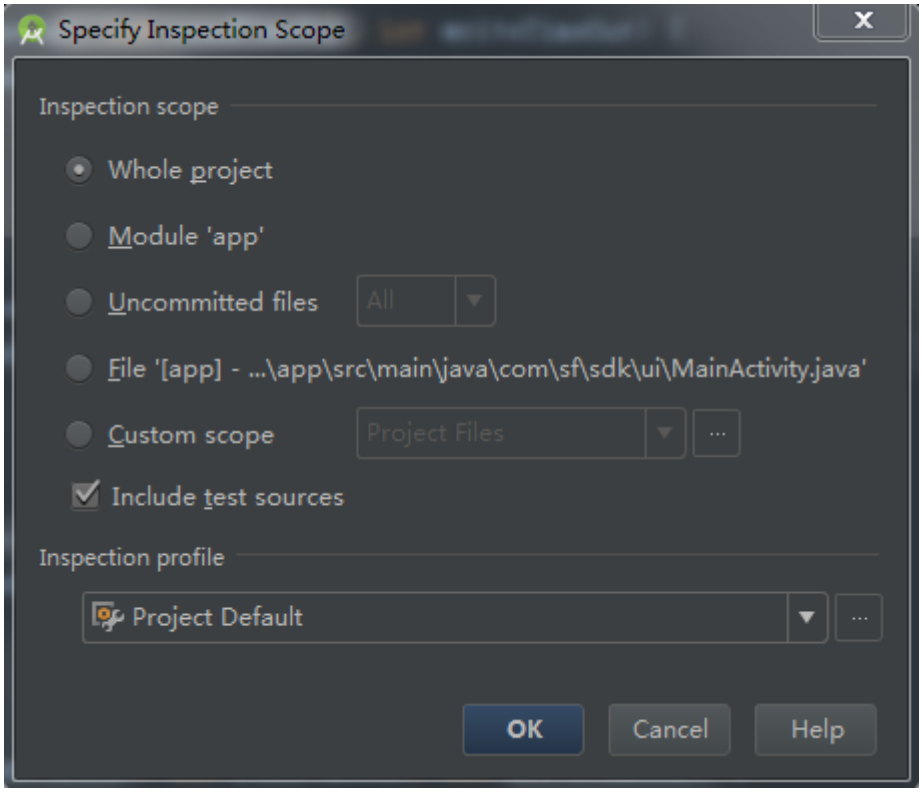
通过这些插件来检测代码上存在 bug。

（2）利用 Android Studio（以下简称 AS）自带的代码检测功能

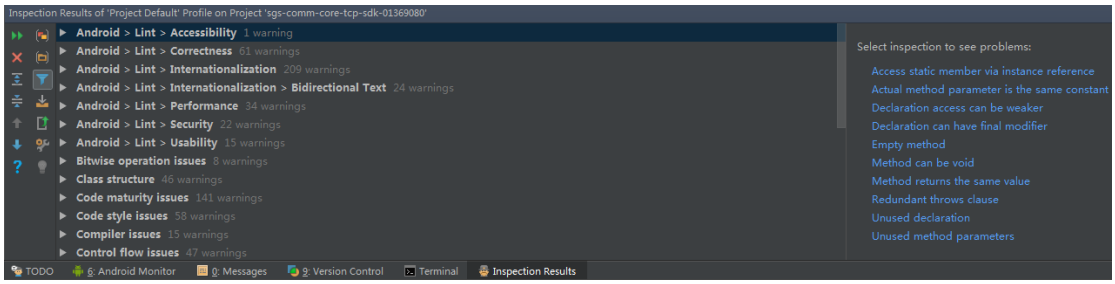
第一步、选择“Analyze->Inspect Code…”菜单。



第二步、选择“Inspect Code…”菜单，弹出对话框，可以选择整个项目，也可选择其中某个 module。



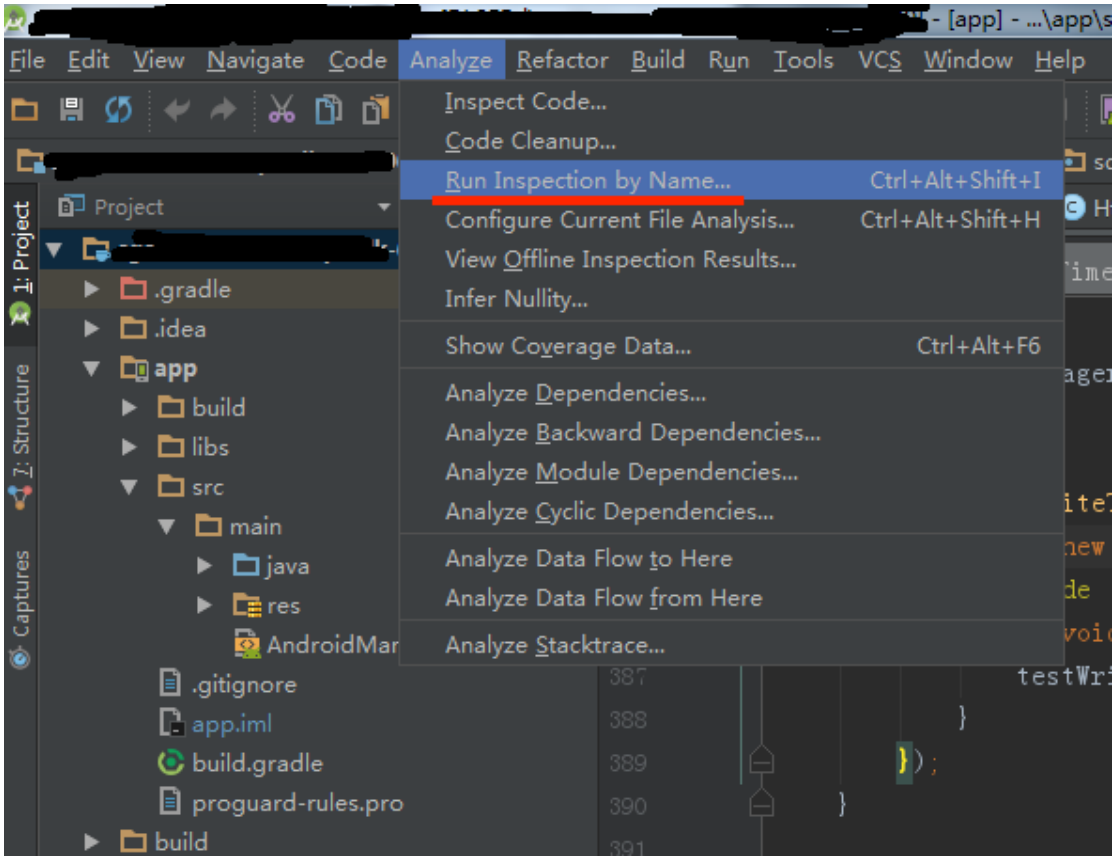
第三步、根据 AS 分析得出的代码相关的“warning”进行判断、修改，也并不是所有的 warning 要全部修改。如，一些不符合语法、不推荐的写法或者错误的写法一般要改掉，hardCode（硬编码）的写法一般也要改掉。



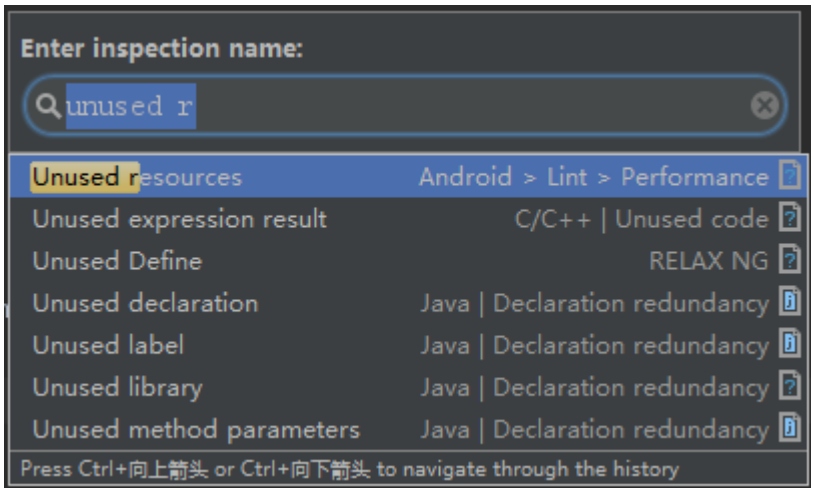
2.资源优化

我们在使用 AS 进行开发时，每个迭代都会引入资源文件（如：图片、xml 等），同时必定会留下一些无用的资源。这个时候，如果我们手动一个一个去删除，效率太低，我们可以利用 AS 自带的功能来删除无用资源。

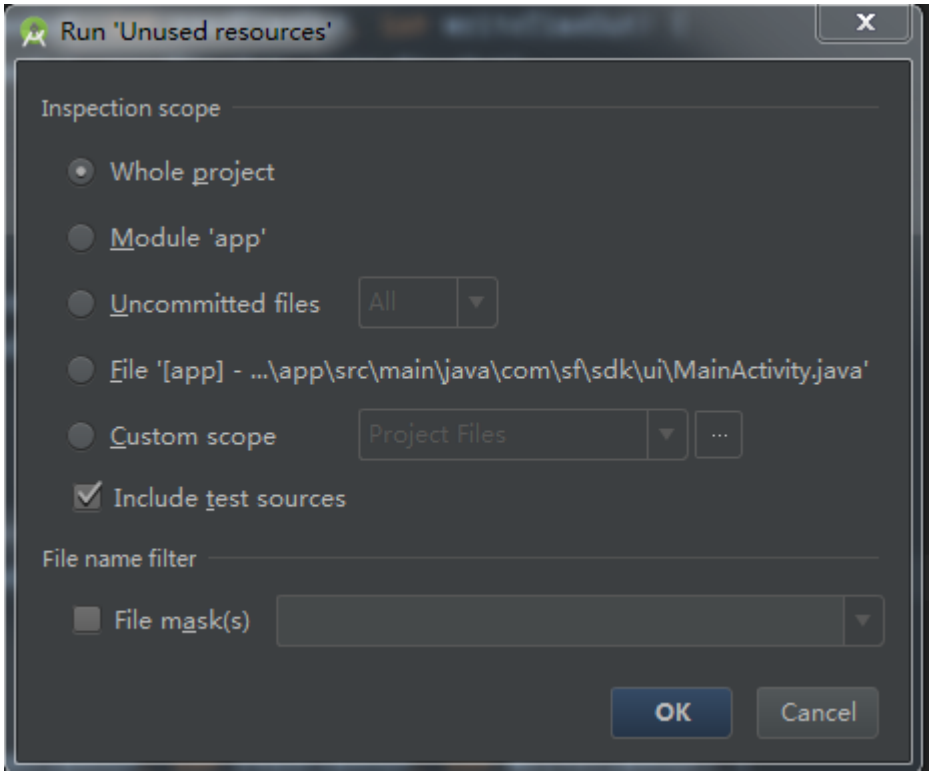
第一步、打开 AS，选择主菜单“Analyze->Run Inspection by Name…”。



第二步、点击“Run Inspection by Name…”会弹出一个对话框，在对话框输入“unused resource”。

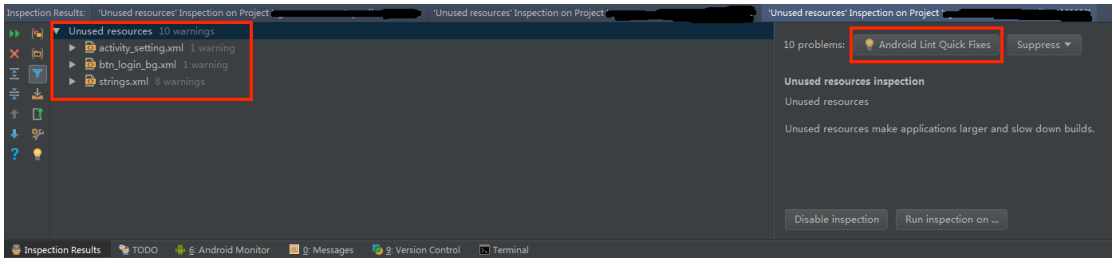


第三步、点击下拉列表中的“unused resource”，会弹出一个对话框，如下图。



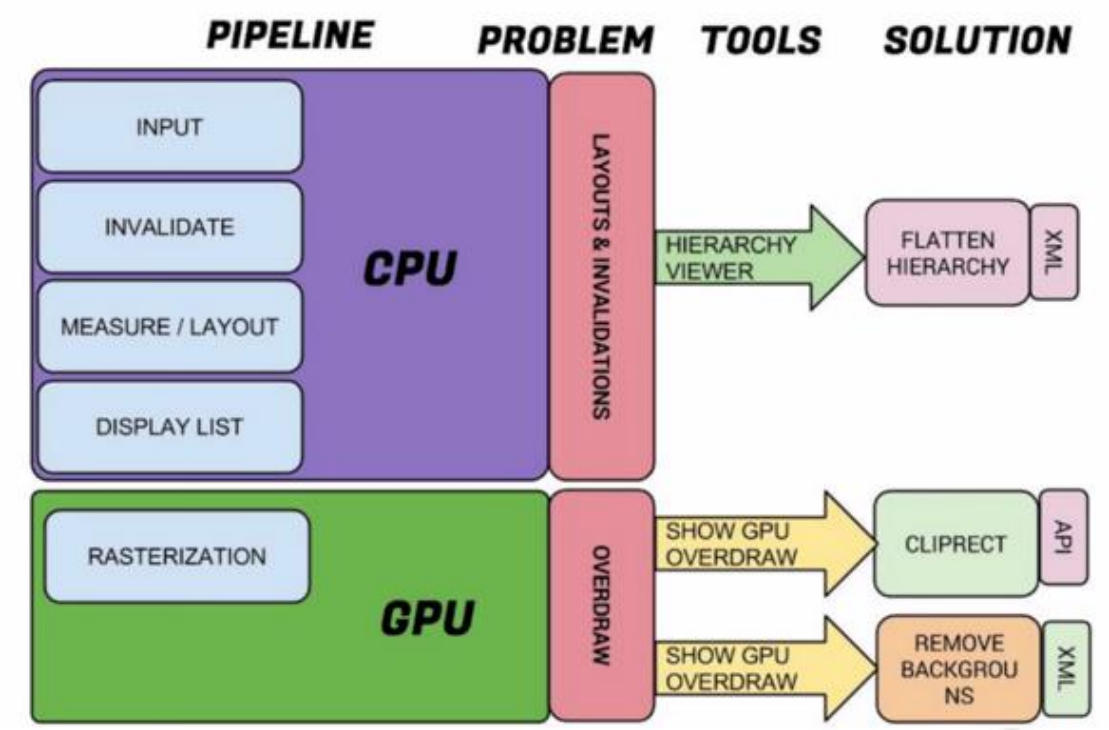
第四步、扫描资源。

在对话框中，我们可以选择 lint 扫描这个项目，也可以指定 lint 扫描某个目录。AS 就会自动扫描分析，然后根据分析结果清除无用资源文件，如下图。



3.UI 优化

我们在实现 App 时，有时候会感觉界面卡顿，大多数情况是因为布局嵌套过多，或者自定义布局的 onDraw 等方法做了过多耗时操作。Android 系统的渲染机制是每隔 16ms 渲染一次，如果 16ms 内无法完成渲染，就会引起丢帧，造成卡顿。那么有什么工具检测这些问题来帮助我们优化 UI 呢？见下图：



从上图可以看出，有两个工具，三种方法来优化 UI。

（1）Hierarchy View 工具

通过可以查看 UI 布局哪些地方嵌套过多，会显示红色（此处 view 较其他 view 运行速度慢，也不一定是真的慢，我们需要自己判断），从而我们可以根据提示来优化布局，减少布局的嵌套。

（2）Show GPU OverDraw

打开步骤“设置 -> 开发者选项 -> 调试 GPU 过度绘制 -> 显示 GPU 过度绘制”。常用优化方法：

第一、移除不必要的 background。

第二、利用 ClipRect，主要是在自定义 View 使用。

4.耗电量优化

（1）发起网络请求之前，先判断网络是否连接正常

因为网络请求也是一个耗电的过程，所以在发起网络请求之前，应该先判断网络是否连接正常，如果网络连接不正常，就不发起网络请求。

（2）使用效率高的数据格式和解析方法

主流的数据格式，Json 和 Protobuf 明显比 xml 的效率高，所以对于移动端开发，我们需选择轻量级的 Json 为佳，当然，如果服务端支持 Protobuf 就更好。

目前数据格式的解析，主要使用树形解析（如 DOM）和流式解析（SAX）。因为 DOM 是对整个文档读取后，再根据各节点层次进行组织，而流失解析是边读取数据边解析，数据读取完，也就解析完了，所以流式解析比树形解析的效率高。

（3）对于大数据量的下载，尽量使用 Gzip 格式下载

目前，我们发起的 http 请求通常都做了 Gzip 压缩，这样能有效减少网络传输时间，减少耗电。

（4）大文件的下载，支持断点下载

对于大文件的下载，要做到断点下载，当然这需要服务端支持。当正在下载某一个文件时，网络突然中断，当网络再次正常时，需要能够在先前已下载的位置继续下载。

（5）其他

如果对定位精度要求不高，尽量使用 wifi 或移动网络 cell 定位，不要使用 GPS，因为 GPS 定位耗电远远高于 wif 和移动网络 cell 定位。

当需要每隔一定时间就去执行某操作时，尽量使用 AlarmManager 来定时启动服务，而不是让 service 不停地在后台去执行和 sleep。

5.常规测试

（1）自测

自测是开发人员对自己开发的功能负责，需要考虑常规操作的和异常情况的测试，这样能在开发阶段，早发现、早解决。这个阶段主要需要开发人员自己对自己开发的功能设计相关的 TestCase。

（2）自动化测试

这个步骤需要通过自动化测试工具，如 Appium、UiAutomator 等，编写相关的 TestCase 代码来执行自动化测试。

这里以 Appium 为例来讲解自动化测试。Appium 是一款开源的移动自动化测试框架，既支持 Android 也支持 iOS。具体使用方法如下：

a.配置 Appium 测试环境

可查看[Appium 环境配置]。

b.识别元素 id

这里用到 Android SDK Tools 目录下的 uiautomatorViewer，执行 uiautomatorViewer.bat 即可。

c.编写测试用例

具体写法大家可搜索相关的资料查看。

（3）测试人员测试

这一步是保证软件质量的关键，极其重要。作为专业的测试人员，要做到以下几个方面：

a.设计 TestCase

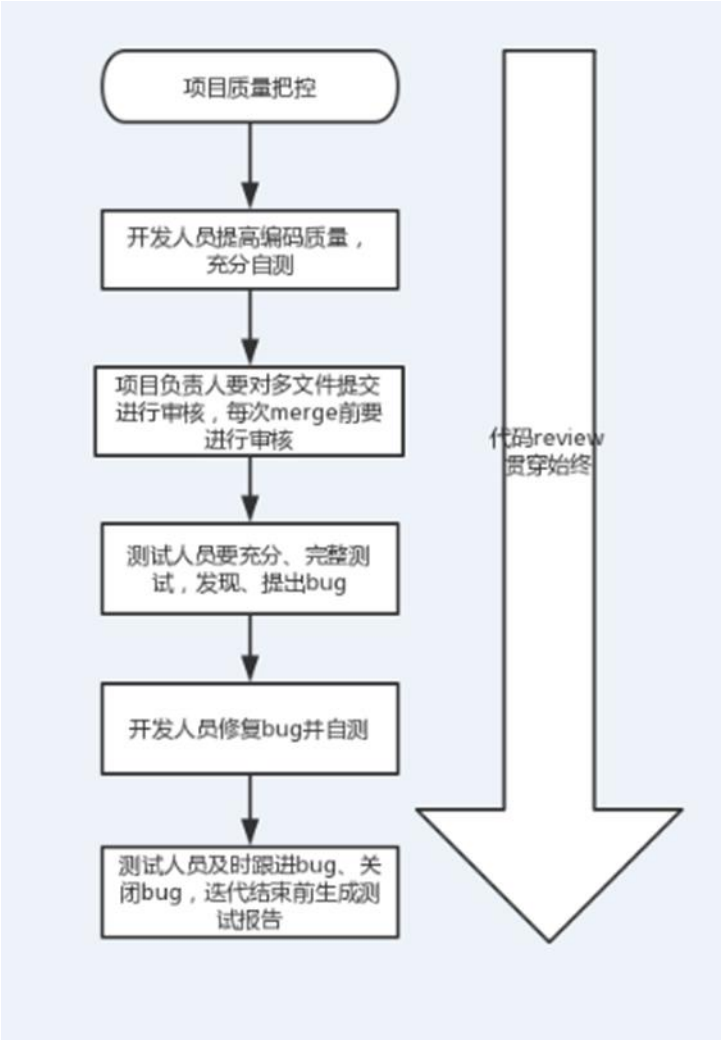
如利用 XMind 工具来编写。针对不同的功能需求，设计不同的、完善的测试用例，特别异常情况的测试用例。

b.测试用例的评审

一般以会议的模式进行，邀请开发人员参与讨论，由测试人员讲解 TestCase，大家一起进行评审，看有没有遗漏的地方。

c.bug 管理

测试过程中，要非常细致、反复测试，保证常规操作不出问题，确保能考虑到的异常情况不出差错。测试人员在测试过程中，发现问题，要及时提 bug 给开发人员。提 bug 时，要对 bug 进行等级分类（如严重、一般、轻微等），写出 bug 的复现率（是 100%复现，还是概率性偶现），最重要的是要详细写出 bug 复现的路径，这有利于开发人员快速定位问题、解决问题。



6.兼容性测试

这是 App 开发完成后，很重要的一块。因为国内 Android 手机品牌众多，所以兼容性测试，一般需要做到机型覆盖市面上大部分手机，如 Top100-300 的手机。大公司，资金充足，技术过硬，可以自己开发测试平台，覆盖市面上大部分手机。一般的公司，可以通过市面上常用的测试平台进行兼容性测试，测试完成会自动生成测试报告。具体步骤：

（1）选择测试平台

目前常用的测试平台有 TestBird、TestIn、百度的 MTC、腾讯的 WeTest 以及阿里的 MQC 等。

（2）上传需要测试 APK

（3）测试完成自动生成测试报告

不过，对于我们顺丰来说，如果都是内部使用的 App，而且是在我们自己定制的 Android 手机上使用，这种情况下，我们只需要在我们自己定制的手机上进行兼容、适配上的测试即可。

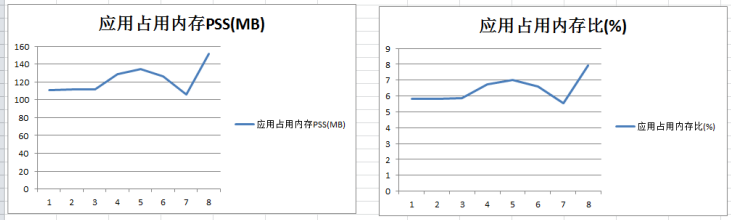
7.性能测试

性能测试，主要包括启动时间（分冷启动、热启动，以及页面切换的所需的时间）、内存、CPU、GPU、耗电量、流量等。两种方案：

（1）利用工具来完成

如 Emmagee 工具（由网易出品），这是一个安装在手机上的 app，可以选择要测试的 app，点击“开始测试”即可开始，当完成测试时，点击“停止测试”就会在 sd 卡根目录生成一个 csv 的测试报告，打开报告，我们利用 excel 图表功能可以生成统计图表。在开始测试前，我们可以在 Emmagee 设置邮箱，测试完成会发测试报告到对应的邮箱。如下图：

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	应用包名	com.sgs.parttiner																		
2	应用名称	丰小弟																		
3	应用PID	6609																		
4	机器内存大	1913.17MB																		
5	机器CPU型	AArch64 Processor rev 4 (aarch64)																		
6	Android系	5.1																		
7	手机型号	HHT6C																		
8	UID	10132																		
9	时间	栈顶Activ	应用占用内	应用占用内	机器剩余内	应用占用C	CPU总使用率(%)	cpu0总使用	cpu1总使用	cpu2总使用	cpu3总使用	cpu4总使用	cpu5总使用	cpu6总使用	cpu7总使用	流量(KB)	电量(%)	电流(mA)	温度(C)	电压(V)
10	########	Component	111.17	5.81	765.56	0	0	0	0	0	0	0	0	0	0	1	0	0	31	3.292
11	########	Component	111.5	5.83	764.29	5.32	20.24	20.48	0	0	0	0	0	0	0	1	0	0	31	3.292
12	########	Component	111.93	5.85	763.3	5.52	20.34	19.44	0	0	0	0	0	0	0	1	1	0	31	3.293
13	########	Component	128.67	6.73	733.53	0.42	0.7	39.4	0	0	0	0	0	0	0	32	0	0	32	3.296
14	########	Component	134.24	7.02	711.38	15.14	39.69	58.85	56.38	32.69	12.26	0	0	0	0	4553	1	0	32	3.298
15	########	Component	125.93	6.58	728.63	20.54	44.33	67.62	64.08	32.34	13.82	0	0	0	0	22130	0	0	32	3.303
16	########	Component	105.94	5.54	711.38	2.03	31.65	38.32	37.25	30.37	20.9	0	0	0	0	22253	0	0	32	3.303
17	########	Component	151.5	7.92	693.23	5.85	21.35	36.37	31.48	14.2	3.77	0	0	0	0	22254	1	0	32	3.307
18	注释:已知部分不支持的机型可在此查阅:https://github.com/MetLase/Emmagee/wiki/Some-devices-are-not-supported																			
19	电流：小于0是放电大于0是充电																			
20	启动时间：为空是应用已启动或者未搜集到启动时间																			
21	N/A：不支持或者数据异常																			



（2）利用测试平台来完成

测试平台及操作步骤与“兼容性测试”一致。

8.内存问题

（1）内存泄露

a.检测工具

LeakCanary（Square 出品）以及 AS 自带的 Memory Monitor、Heap Viewer、Allocation Tracker，相关工具的具体使用，这里就不展开，大家可以自行查询。内存泄露是我们平时开发中经常遇到的问题，大部分的内存问题也是由内存泄露引起的。

b.常见内存泄露

a.单例（主要因为单例一般是全局的，有时候会引用到一些生命周期比较短的变量，导致这些变量无法被回收）。
b.静态变量（主要也是因为其生命周期长，有时候会引导生命周期短的变量，导致内存泄露）。
Handler（主要因为 Handler 与 Activity 生命周期不一致，handler 又持有 Activity 的引用，导致内存泄露）。
c.匿名内部类（匿名内部类会引用外部类，导致无法释放）。
d.资源使用完未关闭(如 BroadcastReceiver、ContentProvider、File、Cursor、Stream、

Bitmap 等）。

（2）图片分辨率的大小

DensityDpi	分辨率	res	Density
160dpi	320×533	mdpi	1
240dpi	480×800	hdpi	1.5
320dpi	720×1280	xhdpi	2
480dpi	1080×1920	xxhdpi	3
560dpi	1440×2560	xxxhdpi	3.5

同一张图片在放在 res 的不同目录下，对 APP 所占用的内存是有影响的，如果对图片质量要求不高，我们一般放在 hdpi 或者 xhdpi 即可，如果放在 xxhdpi 目录，相对于 xhdpi 目录，图片就会被缩放 1.5 倍，导致占用内存升高为原来的 2.25 倍。

（3）图片压缩

我们在利用 BitmapFactory 解码图片，可以设置一个 option，如：
inBitmap：官方推荐使用，意思是可以重复利用图片内存，减少内存分配。在 Android4.4 以前，只有相同大小的图片内存区域可以复用，在 Android4.4 以后只要原图比要解码的图片大就可以复用。
inJustDecodeBunds：只解析图片边界，如果我们只要获取图片大小，就可以使用，而没必要加载图片。
inPreferredConfig：默认使用 ARGB_8888，这种模式，一个像素点占 4 个字节，当我们图片的透明度或图片质量要求不高时，使用 RGB_565，一个像素点占 2 个字节，所占用内存立马减少 50%。
inSampleSize：当设置为小于 1 时，按照 1 处理，相当于没有压缩；当设置为大于 1 时，会被按照 2 的倍数处理。例如，设置为 2，图片宽高同时缩小为原来的 1/2，所占内存变为原来的 1/4。
inTargetDensity：表示要被绘画的目标像素。

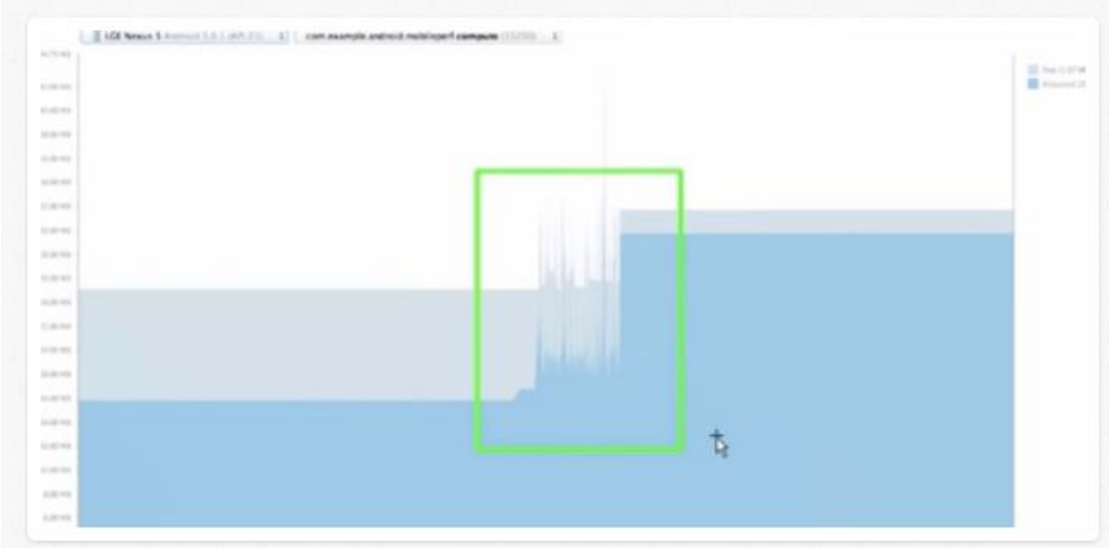
平时我们在开发 App 时，拿到 UI 提供的图片资源，我们可以利用 TinyPNG（https://tinypng.com/）来压缩图片，减少图片占用的内存。

（4）缓存池的大小

很多图片加载组件不仅仅使用了软引用和弱引用，还会用到 LruCache，如 Glide 默认使用的就是 LruCache。使用 LruCache 比较好控制，但是如果缓存池设置过大会造成资源浪费，过小会导致图片经常被回收。所以我们可以根据 App 具体情况以及设备分辨率，计算一个比较合理的值。

（5）内存抖动

在 Android 里面是指内存频繁的分配和回收。而频繁的分配和回收，会造成很多内存碎片，从而可能会引起 OOM。所以，我们在开发过程中，要尽量避免频繁的分配、回收内存，尽可能可以内存复用。



（6）其他类型

a.ListView 复用

我们在用到 ListView 的适配器时，在 getView 里尽量复用 convertView，因为如果不复用的话，getView 频繁调用，会频繁生成对象，造成内存消耗，引起卡顿。

b.枚举

目前 Android 官方建议，使用枚举要谨慎，因为枚举类型比 int 占用内存多大 2 倍以上。

c.资源问题

如果可以，尽可能使用系统资源、图片以及控件的 id。

d.数据相关

序列化数据时，尽量使用 json 格式，如果服务端支持，可以考虑使用 protobuf，

因为 json 和 protobuf 比 xm 效率高，而且 protobuf 比 xml 少占用内存 30%。

e.代码优化、dex 优化

我们平时在开发过程中，总会有一些冗余的代码片段，如，一些 unused 的方法、类或 java 文件，需要我们进行代码优化；另外还有一些类似可以用 int 的地方，就不要用 long 来定义变量。

f.引用第三方库

在开发 App 时，我们经常会引入第三方库，这个时候，如果我们一股脑地把第三方的整个库加入我们的工程，我们的 App 占用的内存也会加大， 此时需要从第三方库中抽取出我们要用到的代码，然后再引入到我们的工程，利用 proguard 优化代码。

9.APP 的安全性

这里的安全性，主要是指对 APK 加固以及 APP 数据传输的安全性。

（1）APK 加固

a.为什么要加固

APK 加固，主要是为了减少 APK 的二次打包，没有加固，APK 很容易被破解，被重新打包，甚至会被一些不法分子增加恶意代码重新打包，严重影响厂家利益，威胁用户信息安全。

b.加固的基本原理

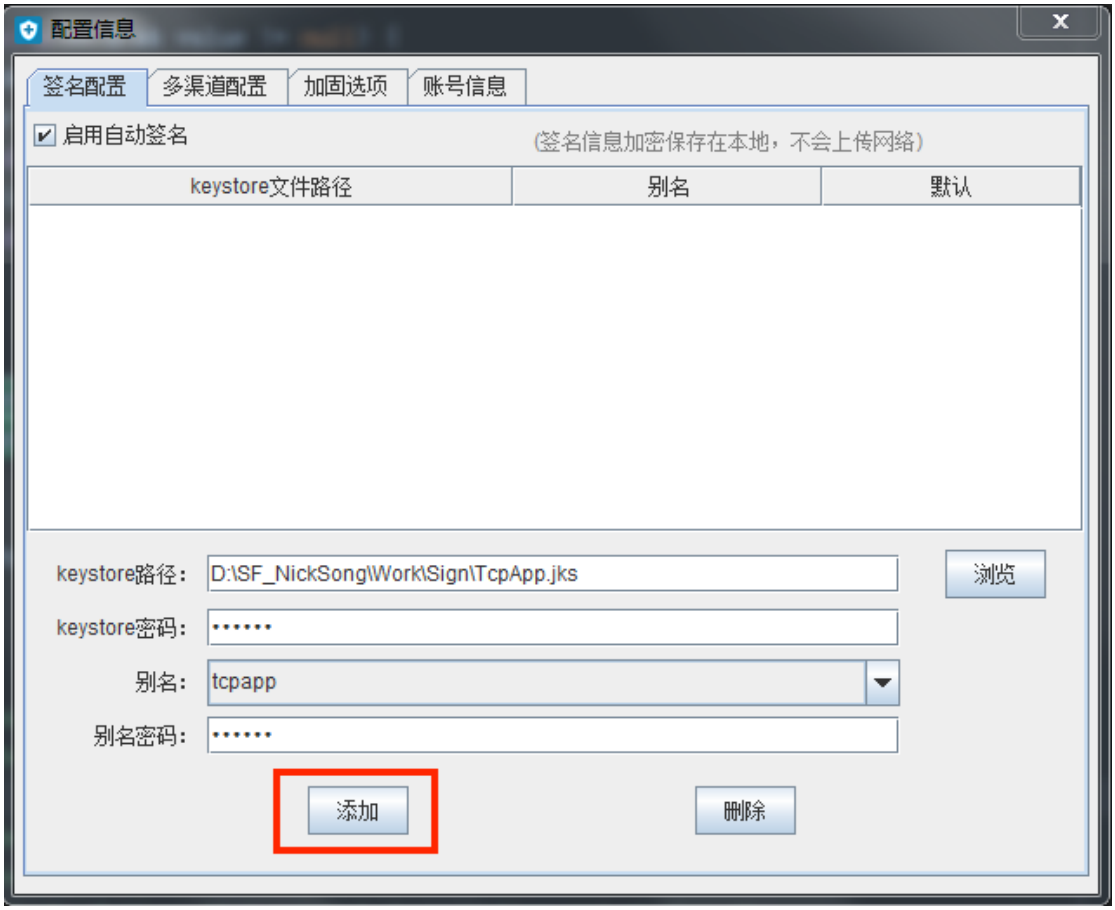
目前基本原理，一般是把 dex、so 加密存于 APK 中，在运行时会先运行壳的代码，壳的代码会对 dex、so 进行解密后加载。不同的厂商可能会有些区别，但基本思路是这样的。

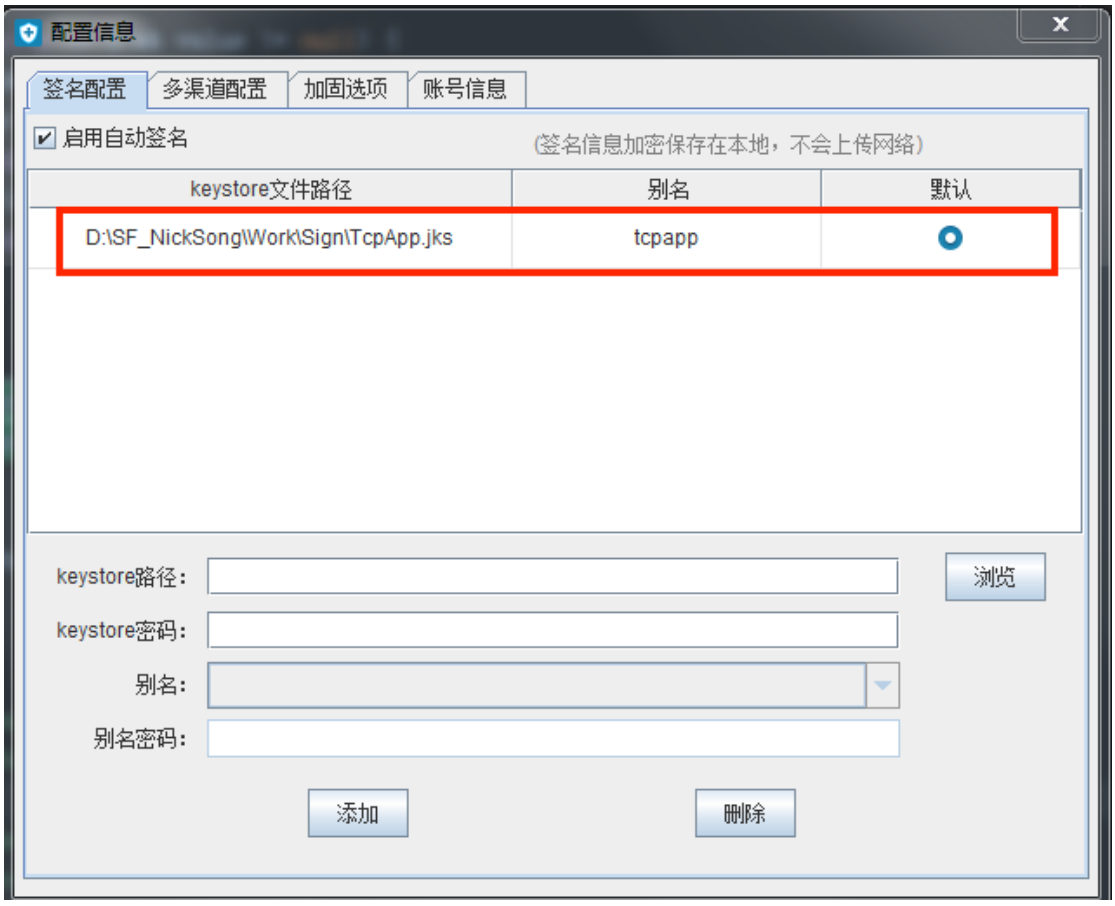
c.加固工具

如梆梆加固、360 加固、娜迦科技、阿里聚等，操作比较简单：选择要加固的 APK，设置签名文件及密码，然后进行加固操作。下面以 360 加固为例，具体操作如下：

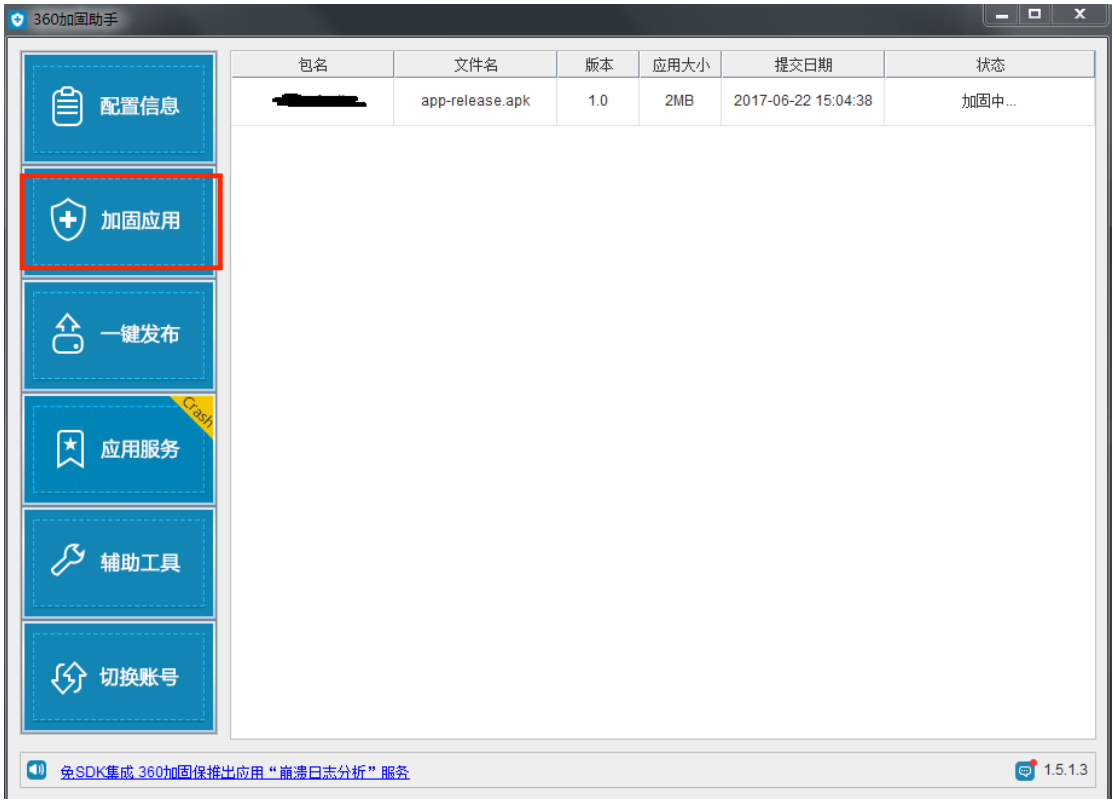
a.启动加固软件。

b.配置加固信息（包括签名证书、密码等信息）。

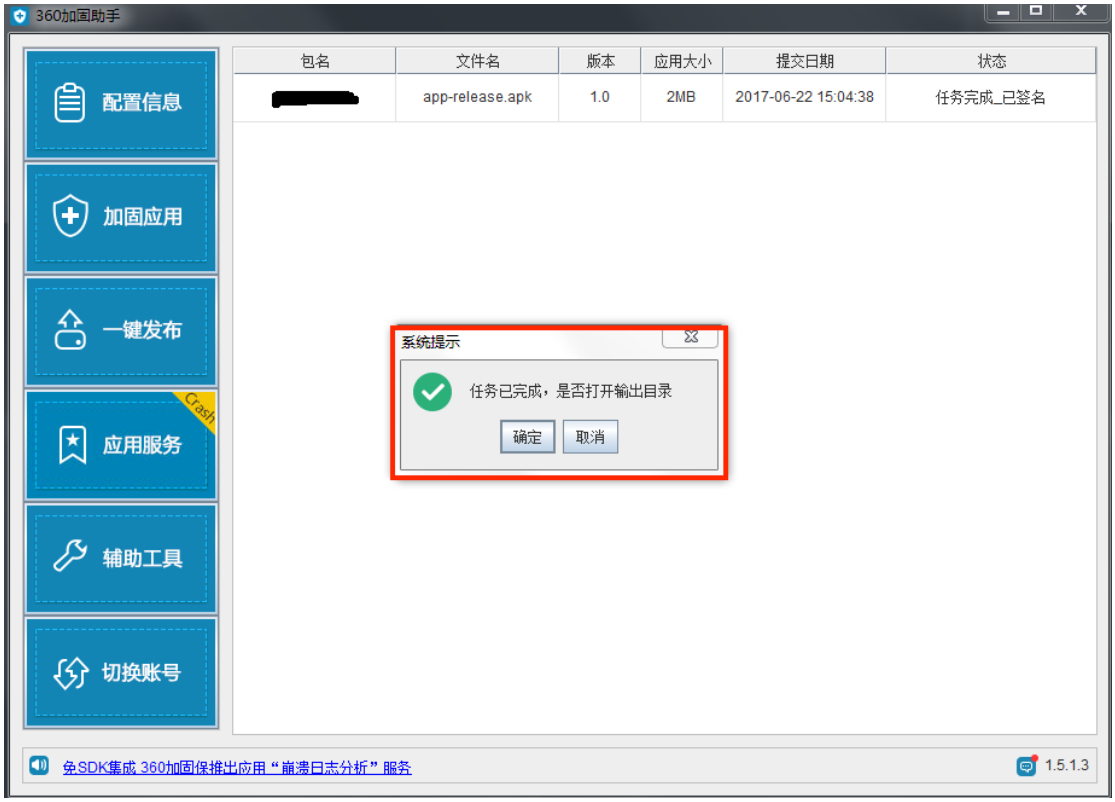




c.选择要加固的 APK。



d.开始加固。



（ps：如果我们公司选择的是爱加密的加固服务，功能步骤类似）

（2）数据的安全性

防止被抓包，造成用户信息泄露，威胁厂商利益。

a.禁止明文传输

重要信息要做加密处理，如非对称加密与对称加密混合使用。例如，比较普遍的做法：利用对称加密的密钥对重要字段进行加密处理，使用非对称加密的公钥对对称加密的密钥进行加密，然后再发起请求传给后台服务器。

b.使用 https 请求

当客户端与服务端证书不一致时，服务端就会拒绝客户端的请求，保证服务端不被随意访问到，提高数据的安全性。

c.敏感数据要脱敏处理

app 在显示用户的手机号、身份证号、银行卡号、金额等敏感信息时，要

先脱敏再显示，如，手机号 13800138000，中间四位脱敏后 138****8000，身份证号 101110198808081234，中间八位脱敏后 101110********1234。

10.APK 瘦身

（1）资源瘦身

a.参照 “资源优化”找出、去除无用资源文件，减少资源占用体积大小。
b.使用 TinyPNG（https://tinypng.com/）压缩图片。

（2）代码瘦身

参照“代码优化”，删除无用代码片段、进一步优化代码。

（3）第三方库瘦身

针对第三方库，抽取我们需要的代码，重新封装。

（4）利用 Proguard 优化瘦身

我们一般都知道利用 Proguard 可以混淆代码，其实 Proguard 还有压缩、优化、预检的功能，所以在 APK 打包时，尽量使用 Proguard 优化代码。

11.SDK 优化

（1）第三方 SDK

a.开源的 SDK

我们应该参照前面讲到的[架构的原则]进行，尽量抽取、简化代码，二次封装，这样有利于我们完全掌控这个 SDK、有利于解决 SDK 出现的问题。

b.未开源的 SDK

这类 SDK，我们一定要选“大厂”的，如 Google、Facebook、BAT 等，这样有利于后续出现问题的时候，能找到相关的技术支持。

（2）自研的 SDK

a.模块化打包

一个主 SDK 如果包含多个子 SDK 工程，我们可以根据业务部门需求，通过 build_local.sh 脚本进行本地模块化打包，或者通过 build_jenkins.sh 脚本进行 jenkins 模块化打包，避免每次都是打出全量版本的 SDK。

b.详细的文档和 demo

我们自研的 SDK，主要就是服务于各业务部门，为了避免各业务部门嵌入我们的 SDK 出现各种问题，我们需要提供详细的集成说明文档。这个文档需要包含以下部分：
① SDK 的功能描述。

② 版本号及更新日志。

③ AS 集成方式（可能还需说明 Eclipse 的集成方式）。

④ 权限说明。

⑤ 如何配置 AndroidManifest 文件。

⑥ 混淆配置。

⑦ SDK 接口使用说明

⑧ 接入范例

⑨ 接口说明

⑩ Demo 下载

⑪ QA 常见问题解答

12.热修复

目前市面热修复用的比较多的有阿里系的 AndFix 以及最近推出的 Sophix(不过现在补丁生成工具有 bug，无法生成补丁)、腾讯系的 Tinker 以及 QZone 的方案，还有饿了么推出的 Amigo 方案。三者多维度的对比如下：

方案对比	Sophix	Tinker	Amigo
DEX修复	同时支持即时生效和冷启动修复	冷启动修复	冷启动修复
资源更新	差量包，不用合成	差量包，需要合成	全量包，不用合成
SO库更新	插桩实现，开发透明	替换接口，开发不透明	插桩实现，开发透明
性能损耗	低，仅冷启动情况下有些损耗	高，有合成操作	低，全量替换
四大组件	不能修复	不能修复	能修复
生成补丁	直接选择已经编好的新旧包在本地生成	编译新包时设置基线包	上传完整新包到服务端
补丁大小	小	小	大
接入成本	傻瓜式接入	复杂	一般
Android版本	全部支持	全部支持	全部支持
安全机制	加密传输及签名校验	加密传输及签名校验	加密传输及签名校验
服务端支持	支持服务端控制	支持服务端控制	支持服务端控制

通过上面的对比，发现阿里系的 Sophix 方案在多方面更胜一筹，不过目前 Sophix 还处于公测阶段，还没有完全开放，以及它的补丁生成工具有 bug，技术社区有多人提出无法生成补丁的问题，至今没有阿里的开发人员回复。鉴于此，我们可以选择腾讯系的热修复方案 Tinker 或者 Bugly（集成的也是 Tinker 方案）。

13.APP 监控日志

（1）集成第三方 SDK

目前市面上用的比较多的有腾讯的 Bugly、Umeng 的 SDK，这些 SDK 都有对应的后台管理系统，我们 App 只需要在这些第三方后台注册 AppId，集成相应的 SDK，就可以通过后台来监控 App 的相关数据。

（2）自研 App 监控 SDK

如果这个 SDK 我们自己研发，需要仔细考虑 SDK 需要采集哪些信息
a.设备信息：手机品牌、型号、经纬度、网络状态、ip、mac 地址。
b.App 版本。
c.启动时间点。
d.异常抛出点：异常信息、所属类、方法、代码行数。
e.其他信息。

有了自研的 SDK，就要开发对应的后台管理系统，统计分析 App 上传的监控日志，生成相应的统计图表、数据统计、异常信息分类、Crash 率等。

后台管理系统的开发，这是一个不小的工作量，而且要做好，持续开发的时间会很长。鉴于目前市面很多“大厂”都有相应的 SDK 和后台管理系统，我们可以直接集成，节省时间，把时间花在我们需要更加关注的技术点上，在这一点上，我们可以选择腾讯的 Bugly 作为我们的 App 监控 SDK。

14.总结、创新和分享

（1）迭代总结

上个迭代的回顾总结对于下个迭代的开展有一定的好处。迭代结束后，每个成员小团队、大团队也可以自行组织回顾总结：上个迭代出现过哪些问题，为什么会出现那些问题，那些问题怎么解决的，在下个迭代如何避免再次发生，上个迭代做的好的地方要继续保持。这样我们的迭代开发才能越做越好，每个团队成员的自组织能力才会越来越强。

（2）创新和分享

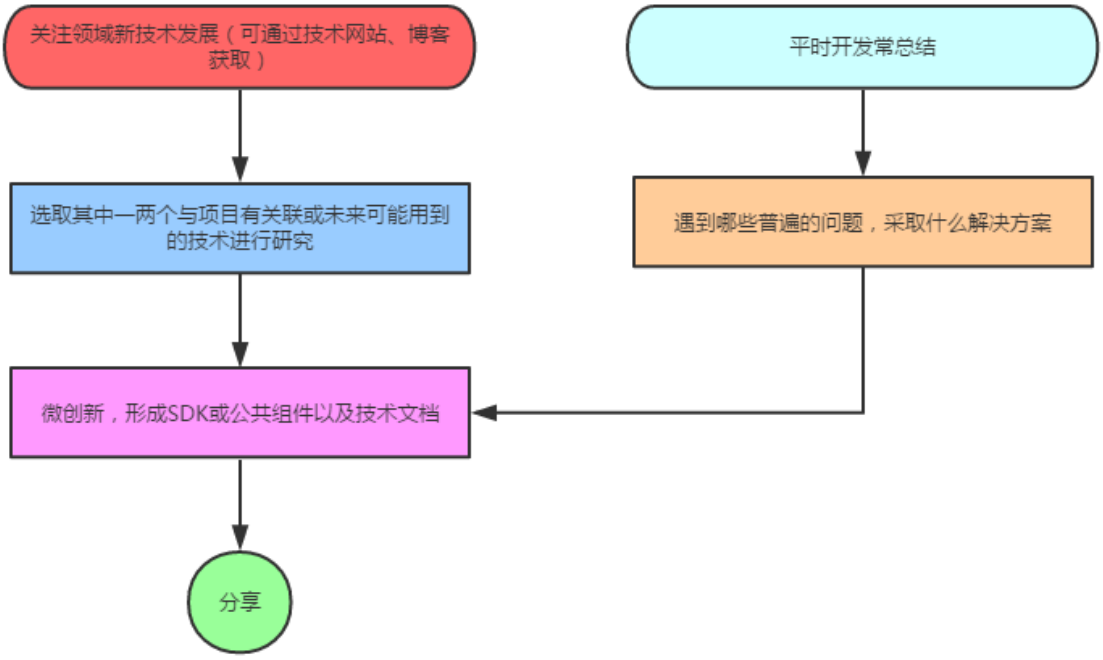
在开发领域，一定要养成常总结的习惯，保持一个好奇、常思考的心态。

a.常总结、微创新

我们在平时开发中，碰到一些技术问题，就要想办法去解决问题，这个过程，我们要尝试多种方式去解决，从中选择最优方案。如果这类问题比较普遍，我们可进行“**微创新**”，开发形成 sdk 或公共组件以及技术文档，供大家使用，提高大家的工作效率。

b.关注新技术的发展

特别是移动端，技术日新月异。新技术的信息获取，可以通过一些常见的技术网站、博客去获取，例如，开源中国、CSDN、github、infoQ、移动开发前线、36 氪等，当有一些新技术开始兴起时，我们可以选取其中一两个与我们项目有关联或未来可能会用到的技术进行研究，研究后，也可进行一定的“**微创新**”，形成文档，并在团队内部进行技术分享，提升团队的技术氛围。


关注领域新技术发展（可通过技术网站、博客获取）
平时开发常总结
选取其中一两个与项目有关联或未来可能用到的技术进行研究
遇到哪些普遍的问题，采取什么解决方案
微创新，形成SDK或公共组件以及技术文档
分享