## 综述

毫无疑问，混淆是打包过程中最重要的流程之一，在没有特殊原因的情况下，所有 app 都应该开启混淆。

首先，这里说的的混淆其实是包括了代码压缩、代码混淆以及资源压缩等的优化过程。依靠 ProGuard，混淆流程将主项目以及依赖库中未被使用的类、类成员、方法、属性移除，这有助于规避64K方法数的瓶颈；同时，将类、类成员、方法重命名为无意义的简短名称，增加了逆向工程的难度。而依靠 Gradle 的 Android 插件，我们将移除未被使用的资源，可以有效减小 apk 安装包大小。

本文由两部分构成，第一部分给出混淆的最佳实践，力求让零基础的新手都可以直接使用混淆；第二部分会介绍一下混淆的整体、自定义混淆规则的语法与实践、自定义资源保持的规则等。

## 一、Android混淆最佳实践

### 1. 混淆配置

一般情况下，app module 的 build.gradle 文件默认会有如下结构：

android {

buildTypes {

release {

minifyEnabled false

proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'

}

}

}

因为开启混淆会使编译时间变长，所以debug模式下不应该开启。我们需要做的是：   
1. 将release下minifyEnabled的值改为true，打开混淆；   
2. 加上shrinkResources true，打开资源压缩。

修改后文件内容如下：

android {

buildTypes {

release {

minifyEnabled true

shrinkResources true

proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'

}

}

}

### 2. 自定义混淆规则

在 app module 下默认生成了项目的自定义混淆规则文件 proguard-rules.pro，多方调研后，一份适用于大部分项目的混淆规则最佳实践如下：

#指定压缩级别

-optimizationpasses 5

#不跳过非公共的库的类成员

-dontskipnonpubliclibraryclassmembers

#混淆时采用的算法

-optimizations !code/simplification/arithmetic,!field/\*,!class/merging/\*

#把混淆类中的方法名也混淆了

-useuniqueclassmembernames

#优化时允许访问并修改有修饰符的类和类的成员

-allowaccessmodification

#将文件来源重命名为“SourceFile”字符串

-renamesourcefileattribute SourceFile#保留行号

-keepattributes SourceFile,LineNumberTable

#保持所有实现 Serializable 接口的类成员

-keepclassmembers class \* implements java.io.Serializable { static final long serialVersionUID;

private static final java.io.ObjectStreamField[] serialPersistentFields;

private void writeObject(java.io.ObjectOutputStream);

private void readObject(java.io.ObjectInputStream);

java.lang.Object writeReplace();

java.lang.Object readResolve();

}

#Fragment不需要在AndroidManifest.xml中注册，需要额外保护下

-keep public class \* extends android.support.v4.app.Fragment-keep public class \* extends android.app.Fragment

# 保持测试相关的代码

-dontnote junit.framework.\*\*-dontnote junit.runner.\*\*-dontwarn android.test.\*\*-dontwarn android.support.test.\*\*-dontwarn org.junit.\*\*

真正通用的、需要添加的就是上面这些，除此之外，需要每个项目根据自身的需求添加一些混淆规则：   
- 第三方库所需的混淆规则。正规的第三方库一般都会在接入文档中写好所需混淆规则，使用时注意添加。   
- 在运行时动态改变的代码，例如反射。比较典型的例子就是会与 json 相互转换的实体类。假如项目命名规范要求实体类都要放在model包下的话，可以添加类似这样的代码把所有实体类都保持住：-keep public class \*\*.\*Model\*.\*\* {\*;}   
- JNI中调用的类。   
- WebView中JavaScript调用的方法   
- Layout布局使用的View构造函数、android:onClick等。

### 3. 检查混淆结果

混淆过的包必须进行检查，避免因混淆引入的bug。

一方面，需要从代码层面检查。使用上文的配置进行混淆打包后在<module-name>/build/outputs/mapping/release/ 目录下会输出以下文件：   
- dump.txt   
描述APK文件中所有类的内部结构   
- mapping.txt   
提供混淆前后类、方法、类成员等的对照表   
- seeds.txt   
列出没有被混淆的类和成员   
- usage.txt   
列出被移除的代码

我们可以根据 seeds.txt 文件检查未被混淆的类和成员中是否已包含所有期望保留的，再根据 usage.txt 文件查看是否有被误移除的代码。

另一方面，需要从测试方面检查。将混淆过的包进行全方面测试，检查是否有 bug 产生。

### 4. 解出混淆栈

混淆后的类、方法名等等难以阅读，这固然会增加逆向工程的难度，但对追踪线上 crash 也造成了阻碍。我们拿到 crash 的堆栈信息后会发现很难定位，这时需要将混淆反解。

在 <sdk-root>/tools/proguard/ 路径下有附带的的反解工具（Window 系统为 proguardgui.bat，Mac 或 Linux 系统为 proguardgui.sh）。

这里以 Window 平台为例。双击运行 proguardgui.bat 后，可以看到左侧的一行菜单。点击 ReTrace，选择该混淆包对应的 mapping 文件（混淆后在 <module-name>/build/outputs/mapping/release/ 路径下会生成mapping.txt 文件，它的作用是提供混淆前后类、方法、类成员等的对照表），再将 crash 的 stack trace 黏贴进输入框中，点击右下角的 ReTrace ，混淆后的堆栈信息就显示出来了。

以上使用 GUI 程序进行操作，另一种方式是利用该路径下的 retrace 工具通过命令行进行反解，命令是

retrace.bat|retrace.sh [-verbose] mapping.txt [<stacktrace\_file>]

例如：

retrace.bat -verbose mapping.txt obfuscated\_trace.txt

### 注意事项：

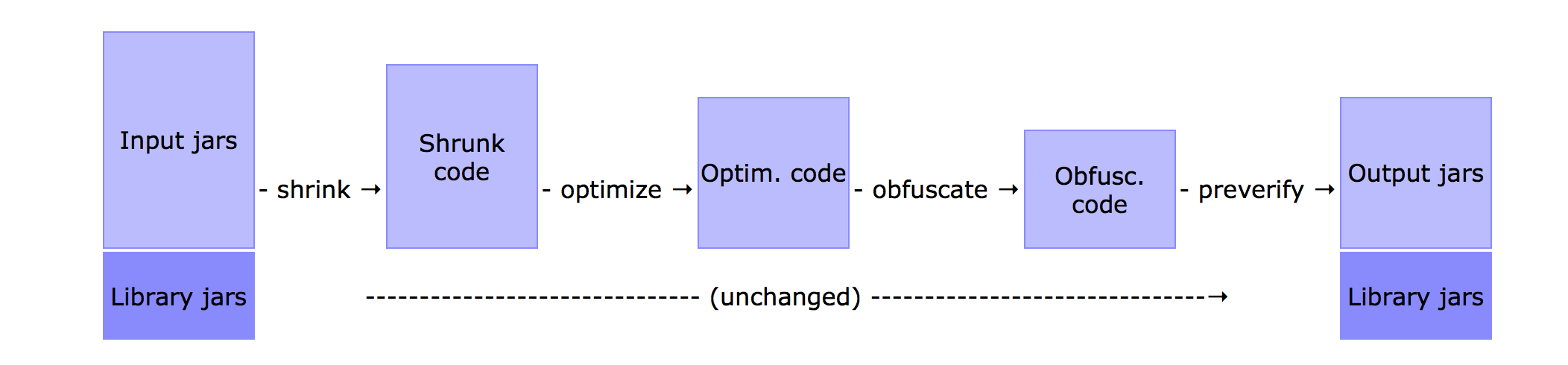
1) 所有在 AndroidManifest.xml 涉及到的类已经自动被保持，因此不用特意去添加这块混淆规则。（很多老的混淆文件里会加，现在已经没必要）

2) proguard-android.txt 已经存在一些默认混淆规则，没必要在 proguard-rules.pro 重复添加，该文件具体规则见附录1：

## 二、混淆简介

Android中的“混淆”可以分为两部分，一部分是 Java 代码的优化与混淆，依靠 proguard 混淆器来实现；另一部分是资源压缩，将移除项目及依赖的库中未被使用的资源(资源压缩严格意义上跟混淆没啥关系，但一般我们都会放一起讲)。

### 1. 代码压缩



代码混淆是包含了代码压缩、优化、混淆等一系列行为的过程。如上图所示，混淆过程会有如下几个功能：   
1. 压缩。移除无效的类、类成员、方法、属性等；   
2. 优化。分析和优化方法的二进制代码；根据proguard-android-optimize.txt中的描述，优化可能会造成一些潜在风险，不能保证在所有版本的Dalvik上都正常运行。   
3. 混淆。把类名、属性名、方法名替换为简短且无意义的名称；   
4. 预校验。添加预校验信息。这个预校验是作用在Java平台上的，Android平台上不需要这项功能，去掉之后还可以加快混淆速度。

这四个流程默认开启。

在 Android 项目中我们可以选择将“优化”和“预校验”关闭，对应命令是-dontoptimize、-dontpreverify（当然，默认的 proguard-android.txt 文件已包含这两条混淆命令，不需要开发者额外配置）。

### 2. 资源压缩

资源压缩将移除项目及依赖的库中未被使用的资源，这在减少 apk 包体积上会有不错的效果，一般建议开启。具体做法是在 build.grade 文件中，将 shrinkResources 属性设置为 true。需要注意的是，**只有在用**minifyEnabled true**开启了代码压缩后，资源压缩才会生效**。

资源压缩包含了“合并资源”和“移除资源”两个流程。

“合并资源”流程中，名称相同的资源被视为重复资源会被合并。需要注意的是，**这一流程不受**shrinkResources**属性控制，也无法被禁止，** gradle 必然会做这项工作，因为假如不同项目中存在相同名称的资源将导致错误。gradle 在四处地方寻找重复资源：   
- src/main/res/ 路径   
- 不同的构建类型（debug、release等等）   
- 不同的构建渠道   
- 项目依赖的第三方库

合并资源时按照如下优先级顺序：

依赖 -> main -> 渠道 -> 构建类型

举个例子，假如重复资源同时存在于main文件夹和不同渠道中，gradle 会选择保留渠道中的资源。

同时，如果重复资源在同一层次出现，比如src/main/res/ 和 src/main/res2/，则 gradle 无法完成资源合并，这时会报资源合并错误。

“移除资源”流程则见名知意，需要注意的是，类似代码，混淆资源移除也可以定义哪些资源需要被保留，这点在下文给出。

## 三、自定义混淆规则

在上文“混淆配置”中有这样一行代码

proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'

这行代码定义了混淆规则由两部分构成：位于 SDK 的 tools/proguard/ 文件夹中的 proguard-android.txt 的内容以及默认放置于模块根目录的 proguard-rules.pro 的内容。前者是 SDK 提供的默认混淆文件（内容见附录1），后者是开发者自定义混淆规则的地方。

### 1. 常见混淆命令：

* optimizationpasses
* dontoptimize
* dontusemixedcaseclassnames
* dontskipnonpubliclibraryclasses
* dontpreverify
* dontwarn
* verbose
* optimizations
* keep
* keepnames
* keepclassmembers
* keepclassmembernames
* keepclasseswithmembers
* keepclasseswithmembernames

在第一部分 Android 混淆最佳实践中已介绍部分需要使用到的混淆命令，这里不再赘述，详情请查阅[官网](http://proguard.sourceforge.net/)。需要特别介绍的是与保持相关元素不参与混淆的规则相关的几种命令：

| **命令** | **作用** |
| --- | --- |
| -keep | 防止类和成员被移除或者被重命名 |
| -keepnames | 防止类和成员被重命名 |
| -keepclassmembers | 防止成员被移除或者被重命名 |
| -keepnames | 防止成员被重命名 |
| -keepclasseswithmembers | 防止**拥有该成员的**类和成员被移除或者被重命名 |
| -keepclasseswithmembernames | 防止**拥有该成员的**类和成员被重命名 |

### 2. 保持元素不参与混淆的规则

形如：

[保持命令] [类] {

[成员]

}

“类”代表类相关的限定条件，它将最终定位到某些符合该限定条件的类。它的内容可以使用：   
- 具体的类   
- 访问修饰符（public、protected、private）   
- 通配符\*，匹配任意长度字符，但不含包名分隔符(.)   
- 通配符\*\*，匹配任意长度字符，并且包含包名分隔符(.)   
- extends，即可以指定类的基类   
- implement，匹配实现了某接口的类   
- $，内部类

“成员”代表类成员相关的限定条件，它将最终定位到某些符合该限定条件的类成员。它的内容可以使用：   
- 匹配所有构造器   
- 匹配所有域   
- 匹配所有方法   
- 通配符\*，匹配任意长度字符，但不含包名分隔符(.)   
- 通配符\*\*，匹配任意长度字符，并且包含包名分隔符(.)   
- 通配符\*\*\*，匹配任意参数类型   
- …，匹配任意长度的任意类型参数。比如void test(…)就能匹配任意 void test(String a) 或者是void test(int a, String b) 这些方法。   
- 访问修饰符（public、protected、private）

举个例子，假如需要将name.huihui.test包下所有继承Activity的public类及其构造函数都保持住，可以这样写：

-keep public class name.huihui.test.\*\* extends Android.app.Activity {

<init>

}

### 3. 常用的自定义混淆规则

* 不混淆某个类

-keep public class name.huihui.example.Test { \*; }

* 不混淆某个包所有的类

-keep class name.huihui.test.\*\* { \*; }

* 不混淆某个类的子类

-keep public class \* extends name.huihui.example.Test { \*; }

* 不混淆所有类名中包含了“model”的类及其成员

-keep public class \*\*.\*model\*.\*\* {\*;}

* 不混淆某个接口的实现

-keep class \* implements name.huihui.example.TestInterface { \*; }

* 不混淆某个类的构造方法

-keepclassmembers class name.huihui.example.Test { public <init>();

}

* 不混淆某个类的特定的方法

-keepclassmembers class name.huihui.example.Test {

public void test(java.lang.String);

}

* 不混淆某个类的内部类

-keep class name.huihui.example.Test$\* { \*;

}

## 四、自定义资源保持规则

### 1. keep.xml

用shrinkResources true开启资源压缩后，所有未被使用的资源默认被移除。假如你需要定义哪些资源必须被保留，在 res/raw/ 路径下创建一个 xml 文件，例如 keep.xml。

通过一些属性的设置可以实现定义资源保持的需求，可配置的属性有：   
- tools:keep 定义哪些资源需要被保留（资源之间用“,”隔开）   
- tools:discard 定义哪些资源需要被移除（资源之间用“,”隔开）   
- tools:shrinkMode 开启严格模式

当代码中通过 Resources.getIdentifier() 用动态的字符串来获取并使用资源时，普通的资源引用检查就可能会有问题。例如，如下代码会导致所有以“img\_”开头的资源都被标记为已使用。

String name = String.format("img\_%1d", angle + 1);

res = getResources().getIdentifier(name, "drawable", getPackageName());

我们可以设置 tools:shrinkMode 为 strict 来开启严格模式，使只有确实被使用的资源被保留。

以上就是自定义资源保持规则相关的配置，举个例子：

<?xml version="1.0" encoding="utf-8"?><resources xmlns:tools="http://schemas.android.com/tools"

tools:keep="@layout/l\_used\*\_c,@layout/l\_used\_a,@layout/l\_used\_b\*"

tools:discard="@layout/unused2"

tools:shrinkMode="strict"/>

### 2. 移除替代资源

一些替代资源，例如多语言支持的 strings.xml，多分辨率支持的 layout.xml 等，在我们不需要使用又不想删除掉时，可以使用资源压缩将它们移除。

我们使用 resConfig 属性来指定需要支持的属性，例如

android {

defaultConfig {

...

resConfigs "en", "fr"

}

}

其他未显式声明的语言资源将被移除。

## 参考资料

* [Shrink Your Code and Resources](https://developer.android.com/studio/build/shrink-code.html)
* [proguard](http://proguard.sourceforge.net/)
* [Android安全攻防战，反编译与混淆技术完全解析（下）](http://blog.csdn.net/sinyu890807/article/details/50451259)
* [Android混淆从入门到精通](http://www.jianshu.com/p/7436a1a32891)
* [Android代码混淆之ProGuard](http://rensanning.iteye.com/blog/2224635)

## 附录

1. proguard-android.txt文件内容

#包名不混合大小写

-dontusemixedcaseclassnames

#不跳过非公共的库的类

-dontskipnonpubliclibraryclasses

#混淆时记录日志

-verbose

#关闭预校验

-dontpreverify

#不优化输入的类文件

-dontoptimize

#保护注解

-keepattributes \*Annotation\*

#保持所有拥有本地方法的类名及本地方法名

-keepclasseswithmembernames class \* {

native <methods>;

}

#保持自定义View的get和set相关方法

-keepclassmembers public class \* extends android.view.View {

void set\*(\*\*\*);

\*\*\* get\*();

}

#保持Activity中View及其子类入参的方法

-keepclassmembers class \* extends android.app.Activity {

public void \*(android.view.View);

}

#枚举

-keepclassmembers enum \* {

\*\*[] $VALUES;

public \*;

}

#Parcelable

-keepclassmembers class \* implements android.os.Parcelable {

public static final android.os.Parcelable$Creator CREATOR;

}

#R文件的静态成员

-keepclassmembers class \*\*.R$\* {

public static <fields>;

}

-dontwarn android.support.\*\*

#keep相关注解

-keep class android.support.annotation.Keep

-keep @android.support.annotation.Keep class \* {\*;}

-keepclasseswithmembers class \* {

@android.support.annotation.Keep <methods>;

}

-keepclasseswithmembers class \* {

@android.support.annotation.Keep <fields>;

}

-keepclasseswithmembers class \* {

@android.support.annotation.Keep <init>(...);

}