链滴

# App 混淆

作者：Yangzzz

原文链接：https://ld246.com/article/1502486683404

来源网站：链滴

一、前言

为了保护自己的劳动成果，几乎多数的app都会使用代码混淆，尽可能的增加app被反编译的难度

在这里记录下常用到的一些混淆规则

二、ProGuard的作用

Android提供了ProGuard机制对app进行混淆、优化、压缩

1、混淆代码

将代码中的包名、类名、方法名、属性改名成一系列没有可读性字母，从而增加阅读难度

2、字节码优化

对代码进行迭代优化，增加运行速度

3、压缩 （在代码优化前后各执行一次）

减少app体积，移除未使用的类、变量

三、 ProGuard的主要命令

1、类和类成员
-keep 防止被移除或重命名
-keepnames 防止重命名

2、仅类成员
-keepclassmembers 防止被移除或重命名
-keepclassmembersnames 防止重命名

3、保留拥有某成员的类和类成员
-keepclasseswithmembers 防止被移除或重命名
-keepclasseswithmembernames 防止重命名

四、主要配置项

混淆和Java反射存在冲突，所以反射用到的类需要避免混淆

1、保留类名
-keep class com.whipser.test.* 当前包下的类名会被保留 但不包含子包
-keep class com.whisper.test.** 当前包以及子包的类名都会保留

2、保留类名以及类中的内容
-keep class com.whisper.test.* {*;} 当前包下的类和类中的内容都会保留

3、保留特定类
-keep class public class * extends android.app.Activity 保留所有继承Activity的类

4、保留内部类
-keepclassmembers class com.whisper.test.ui.fragment.TestFragment$InnerClass { public *; }
保留TestFragment中内部类InnerClass的所有public内容

原文链接：App 混淆

5、保留内容部分内容
<init>  构造方法
<fields>  所有属性
<methods> 所有方法

且可以为其指定private 、public、native、protected等修饰符

如：保留A类中所有public方法 当然类名也会保留 否则无意义
```
-keep class com.whisper.test.A {
  public <methods>;
}
```

如保留A参数为String的构造方法
```
-keep class com.whisper.test.A {
  public <init>(java.lang.String)
}
```

五、注意

1、反射用到的元素不能混淆

2、jni 方法不能混淆

```
-keepclasswithmembernames class * {
native <methods>;
}
```

3、AndroidManifest中的类不能混淆

4、自定义view不能混淆

5、Gson 、fastjson 等orm映射对象不能混淆

6、webview js调用接口方法不能混淆

7、Parcelable的子类和Creator静态成员不混淆，避免BadParcelableException

8、使用enum是，会存在反射调用，需要加入如下混淆：

```
-keepclassmembers enum * {
public static **[] values();
public static ** valueOf(java.lang.String);
}
```

六、 开启混淆

androidstuidio环境：

build.gradle 中配置 minifyEnable为ture 并指定好混淆文件

另外zipAlignEnabled可以设置为true, app包中资源按4字节对齐，减少运行时内存消耗

七、拓展运用

原文链接：App 混淆

需求： 有的时候包下的部分类不需要混淆，而需混淆的类也不少，若在文件中配置必然要加入诸多混代码，同时当代码中
的类名、方法名、属性名改变时，配置文件亦需要改变，较为繁琐，那么有没有更加灵活易用的防混方式呢?
下面介绍使用编译时注解来解决这个问题

第一步： 定义注解

```
@Retention(RetentionPolicy.CLASS)
@Target({ElementType.TYPE,ElementType.METHOD,ElementType.CONSTRUCTOR,ElementTyp
.FIELD})
public @interface NoProguard {
}
```

第二步： 配置混淆代码 对使用该注解的元素保留

```
# 自定义注解NoProguard的混淆支持
-keep @com.whisper.life.lib.NoProguard class * {*;}
-keep class * { @com.whisper.life.lib.NoProguard <fields>;}
-keepclassmembers class * { @com.whisper.life.lib.NoProguard <methods>;}
```

提示： 这三行混淆代码分别支持类、属性、方法的免混淆

第三步： 使用NoProguard注解

结论： NoProguard注解 提供了另一种防混淆的方式，而且使用灵活性好