APK的内容

解压最新版微信APP之后可以得到以下内容：

META-INF/：包含 CERT.SF 和 CERT.RSA 签名文件，以及 MANIFEST.MF 清单文件。 apk 签名后，将生成这些文件。随着文件的增加对应的文件大小将不断增加。

.MF: 每一个资源都有一个SHA1签名，存放在这里。

.SF: 文件存放.MF经过base64编码后的签名。

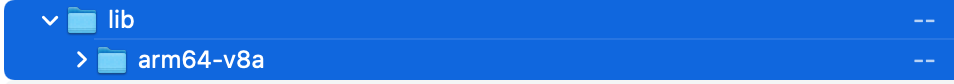
.RSA: 对.SF文件使用SHA1算法生成数字摘要（注意:.MF中是对每一个资源进行SHA1，这里是对文件），然后进行RSA加密，再用开发者私钥进行签名，安装时使用公钥解密。

这样子，一个app安装在手机时，解密这一数字摘要，然后与内部的.MF文件比对，如果相符，证明资源内容没有被修改。

assets/：包含应用程序的资产，这是希望稍后在应用程序中使用的一些文件。

res/：包含未编译到 resources.arsc 中的资源。图像和布局将保留在这里。

lib/：包含特定于处理器软件层 (ABI) 的编译代码，具有每种平台类型（如 armeabi、armeabi-v7a、arm64-v8a、x86、x86\_64 和 mips）。



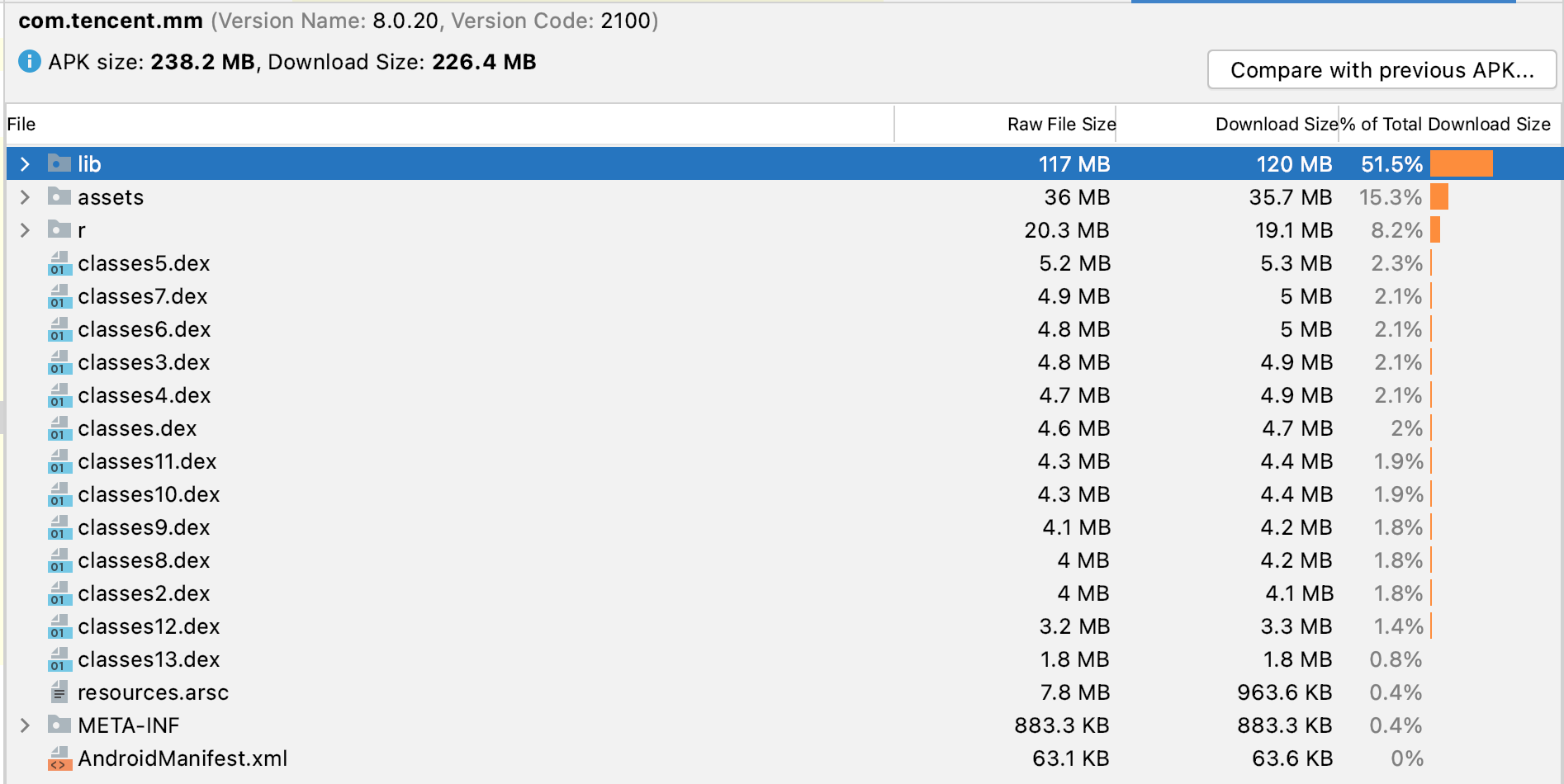
resources.arsc：包含编译资源。此文件包含来自 res/values/ 文件夹的所有配置的 XML 内容。打包工具提取此 XML 内容，将其编译为二进制形式，并将内容存档。此内容包括语言字符串和样式，以及未直接包含在 resources.arsc 文件中的内容的路径，例如布局文件和图片。

classes.dex：包含以 Dalvik/ART 虚拟机可以理解的 DEX 文件格式编译的类。



AndroidManifest.xml：包含核心 Android 清单文件。该文件列出了应用程序的名称、版本、访问权限和引用的库文件。该文件使用 Android 的二进制 XML 格式。

一般来说，使用 android studio 中的 Build > Analyze APK。打开后你会看到这样的东西：



一、减小 ABI 的大小

1、lib 文件夹中有什么？

lib 文件夹内包含不同的 Android 手机使用不同的 CPU，进而支持不同的指令集。 CPU 和指令集的每种组合都有其自己的应用程序二进制接口或 ABI。 ABI 非常精确地定义了应用程序的机器代码在运行时应该如何与系统交互。

2、如何减小尺寸？

我们仍然可以通过删除不必要的文件夹来减小大小，即构建多个 apk，每个 apk 只包含正确的 .so 文件。幸运的是，gradle 确实支持在不同的 cpu 上拆分构建。

要构建拆分 apk，请将代码片段添加到 build.gradle 文件中的 productFlavors 部分：

splits {

abi {

// Enables building multiple APKs per ABI only when build with property

enable project.hasProperty('splitApks')

reset()

// Resets the list of ABIs that Gradle should create APKs for to none.

// Specifies a list of ABIs that Gradle should create APKs for.

      include "armeabi-v7a", "arm64-v8a", "x86"

// You can set this param to false if you donot need a universal apk.

// However, having a universal apk could help when you need to send apk by mail.

universalApk true

}

}

我们仅在存在名为 splitApk 的属性时才启用拆分 apk 构建。构建脚本应该这样修改：

./gradlew -PsplitApks clean assembleRcRelease

通过微信最新版本的APK中我们知道现在只保存了arm64-v8a类型的ABI。

1. android:extractNativeLibs

android:extractNativeLibs = true时，gradle打包时会对工程中的so库进行压缩，最终生成apk包的体积会减小。

因为so是压缩存储的，因此用户安装时，系统会将so解压出来，重新存储一份。因此安装时间会变长，占用的用户磁盘存储空间反而会增大。

1. 减少资源使用
2. 使用矢量图

从 android M (21) 开始，原生支持矢量 drawable，这意味着您需要将 min sdk 设置为 21或使用支持库。如果使用支持库，必须指定使用矢量图，否则将被转换为png：

android {

defaultConfig {

...

vectorDrawables.useSupportLibrary = true

}

}

图标字体是替换图像的另一个技巧。它将图像另存为字体，这使得它非常小且可缩放。但是，它仅支持单色（就像字体一样），并且仍然需要额外的工作。最重要的是，图标通常不会太大，这使其对 apk 大小的影响较小。要将字体用作图标，请将矢量图添加到 https://icomoon.io/，并在 textview 中显示图标，如下所示：

// the textview to show the image

TextView tv1=(TextView)findViewById(R.id.textview);

Typeface typeface=Typeface.createFromAsset(getAssets(),"fonts/iconfont.ttf");

tv1.setTypeface(typeface);

// the unicode you add in tff

// instead of hard code it here, you should put it to your string file or make it to constant, so that it can be managed better.

tv1.setText("\ue600");

如果愿意，还可以将其作为扩展 textview 的类来管理图标。

除了手动创建自己的字体外，还可以尝试现成的解决方案，例如 Android-Iconics（https://github.com/mikepenz/Android-Iconics）。

1. 重用图像

看看你的图标。其中有多少是同质的？你有多少复选图标或箭头图标？随着产品发展和设计师发布不同版本的设计指南，通常在 res 文件夹中存在多个图像，每个图像具有完全相同的含义，只有当它们并排放置时才可见的差异很小。花一些时间与设计师一起审查这些图像肯定会发现一些可以重复使用的图像，尤其是当没有人审查这些图标时。

可以做的另一种重用类型是旋转图像以删除重复的图像（如指向不同方向的箭头）。在drawable文件夹中创建一个 xml，如下所示：

<?xml version="1.0" encoding="utf-8"?>

<rotate

xmlns:android="http://schemas.android.com/apk/res/android"

android:drawable="@drawable/original\_image"

android:fromDegrees="90"

android:pivotX="50%"

android:pivotY="50%"/>

1. 资源解析

我建议只使用 xxhdpi 和 mipmap-xxxhdpi 作为启动器图标。超超超高密度 (xxxhdpi) 使用的资源 (~640dpi)。仅将其用于启动器图标。 mipmap-xxxhdpi 限定符仅用于提供在 xxhdpi 设备上看起来比平常更大的启动器图标。不需要为所有应用程序的图像提供 xxxhdpi 资产。

1. 删除未使用的资源

我们很可能保留了一些我们在源代码中不使用的资源。就像永远占用内存的内存泄漏一样，资源将永远占用 apk 空间，直到将它们踢出。幸运的是，我们的服务有 lint，它会扫描项目并显示有关可能问题的警告，包括未使用的资源。

./gradlew clean lint

在这里，我使用 android-resource-remover（https://github.com/KeepSafe/android-resource-remover） 根据 lint 报告删除资源。基本用法如下：

# install android-resource-remover

pip install android-resource-remover

# there's some issue on remover currently, need to move manifest manually

cp ./<pathToManifest>/AndroidManifest.xml AndroidManifest.xml

android-resource-remover --xml <yourLintReport>

在 android-resource-remover 中还有一个 bug 需要修复。它确实从文件引用中删除了资源 ID，但没有从文件夹中删除文件。在这里，我们需要一个脚本来从文件夹中删除文件：

#!/usr/bin/python

# -\*- coding: utf-8 -\*-

# Due to bugs in android resouce remover, the unused layout files is not removed

import xml.etree.ElementTree as ET

import os

import fnmatch

def find(pattern, path):

result = []

for root, dirs, files in os.walk(path):

for name in files:

if fnmatch.fnmatch(name, pattern):

result.append(os.path.join(root, name))

return result

print 'find lint xml'

lintfile = find('lint-results\*.xml', './')

print '\nstart to remove what android res remover did not clean out...'

tree = ET.ElementTree(file=lintfile[0])

for elem in tree.iterfind('issue[@id="UnusedResources"]'):

for lac in elem.findall('location'):

path = lac.get('file')

if '/res/layout/' in path or '/res/drawable' in path or '/res/menu' in path:

try:

os.remove(path)

print 'remove file: ', path

except OSError:

pass

所以整个删除资源代码应该是这样的：

./gradlew clean lint<taskName>

LINT\_PATH=$(find ./ -iname lint-results-\*\*.xml)

# there's some issue on remover currently, need to move manifest manually

cp ./<ManifestPath>/AndroidManifest.xml AndroidManifest.xml

android-resource-remover --xml $LINT\_PATH

# remove unused layout files

python rm\_unused\_res.py

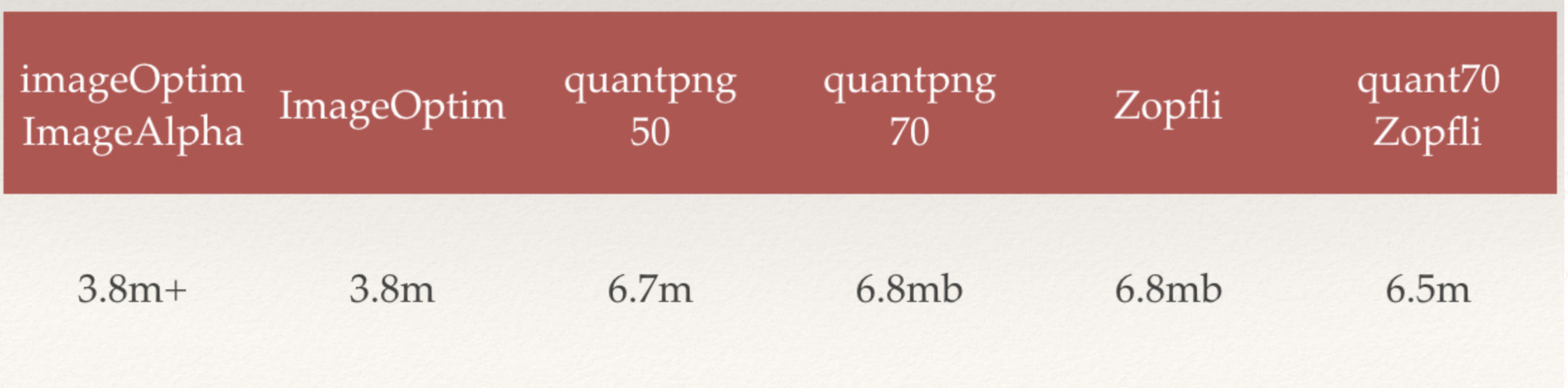
1. 减少资源大小
2. 压缩 Resource.arsc（不推荐）

由于大量资源在 Resource.arsc 中，如果压缩它可以为您节省很多空间，对吧？好吧，虽然它确实减少了 apk 的大小，但解压会占用额外的内存，并且会影响应用程序启动时间。

1. 图像压缩

通常，图像将占用 apk 中的大部分空间。特别是当应用程序设计师没有研究压缩部分时。让我们从手头的一些工具开始。这里我只考虑我们可以不受限制地使用的开源库。可以选择一些像 tinyPng 这样的在线服务，但它可能需要一些钱，并且没有可以测试的配置。最重要的是，需要将文件上传给他们，不知道他们将如何处理文件。

我在一些大约 6.9mb 的样本 res 文件夹上测试了 Quantpng、ImageOptim 和 Zopfli。最后，ImageOptim 显示出最佳的压缩率。结果如下：



与 Quantpng 和 Zopfli 相比，ImageOptim 执行更好的压缩，因为它实际上结合了 OptiPNG、PNGCrush、AdvPNG、JpegOptim、JpegTran、PNGOUT 和 Gifsicle。请注意，ImageOptim 仅适用于 Mac。

我确实在 ImageOptim 之后尝试了 Zopfli，它可以节省大约 300kb。运行 JpegTran 节省了大约 400kb。

除了常规的 png 压缩之外，如果您的 min sdk 大于 4.3(18)，您可以将 png 转换为 webp。在之前的案例中，ImageOptim 压缩后节省了大约 200~300kb。

简而言之，我建议使用质量为 70 且 Zopfli 已停用的 ImageOptim。这不太可能区分，但你应该创建一些样本并与设计师讨论，以确保你们都对此感到满意。

1. 开始使用 ImageOptim-CLI

ImageOptim-CLI（https://github.com/JamieMason/ImageOptim-CLI） 在 ImageOptim 上添加了 shell 支持，使我们能够使用脚本运行它（当然你不会想手动执行）。要安装，只需运行：

curl --output imageoptim-cli.zip https://codeload.github.com/JamieMason/ImageOptim-CLI/zip/1.14.8

unzip imageoptim-cli.zip

并将 imageOptim、imageOptimAppleScriptLib、imageOptimBashLi、bpngquant 移动到空闲 mac 上可以访问 github 的文件夹中，并添加以下脚本进行每周修剪：

while [[ true ]]; do

echo "pull latest images"

git clone <git path> --branch <branch name> --single-branch

# step1, run imageOtim

./imageoptim -s -d ./<res path>

# step2, convert webp

# webp is not supported until 4.3 (api 18)

# echo "locate all image files"

# find . -path "./<res path>/\*" -type f \( -iname "\*.png" -o -iname "\*.PNG" -o -iname "\*.jpg" -o -iname "\*.JPG" \) -print0 | while IFS= read -r -d $'\0' file;

# do

# base="${file:0:${#file}-4}"

# webpfile="${file:0:${#file}-4}.webp"

#

# if ! [[ $file == \*".9"\* ]] && [[ $file == \*".png"\* ]]

# then

# echo

# cwebp -q 70 "$file" -o "$webpfile" -m 6 -quiet

#

# I=$(stat -f "%z" $webpfile)

# J=$(stat -f "%z" $file)

#

# echo "convert $file to webp (webp size:" $I " original size:" $J ")"

#

# # delete the WebP file if it is equal size or larger than the original PNG

# if [ $I -gt $J ]

# then

# echo "delete WebP file that is no smaller than PNG"

# rm -f "$base".webp

# else

# echo "replace png with smaller webp"

# rm -f "$file"

# fi

# fi

# done

#

echo "push image changes..."

git add -A

git commit -am "weekly image triming"

git push origin <branch name>

cd ..

echo "job done, see you next week"

sleep 604800

done

注意，您仍然需要自己安装 ImageOptim、ImageAlpha 或 Jpegmini（不是免费的），并且您必须手动更改 ImageOptim 中的设置，因为 ImageOptim-CLI 不通过参数配置它。而且 ImageOptim-CLI 只能在 mac 上运行，所以你需要有一个备用的 mac 来运行脚本。

1. 质量可以接受吗？

谷歌意识到了这一点，并制作了 Butteraugli，这是一个可以分辨人眼感知差异的库。您可以使用它来找到可接受的质量设置。

如果我们可以通过参数配置 ImageOptim 的设置，我们可以想出一个脚本来优化每个图像并运行 Butteraugli。在每个回合中，我们都会稍微降低质量。如果质量不可接受，请使用透水的。这样，我们使用自己的配置优化了每个图像，该配置具有低磁盘大小和可接受的质量。遗憾的是，ImageOptim-Cli 和 ImageOptim 还不能修改设置。如果是这样，我们可以制作一个优化图像的终极工具。

1. 从 Resource.arsc 节省更多

我们之前讨论过 Resource.arsc 压缩，它会占用额外的内存并减慢应用启动速度。但是，我们仍然可以对 Resources.arsc 做一些事情。

在 Resources.arsc 中，它包含所有资源的引用和一些简单的资源，如样式。可以通过重命名 id 和协调 res 路径来保护 id 和文件夹的映射。 apk 大小将减少大约几百 kb。

Andresguard（https://github.com/shwenzhang/AndResGuard）是一个保护和减少 Resources.arsc 的工具。您只需将 apk 与打包工具和密钥库一起传递即可发挥作用。它将解压 apk，混淆 res 名称/路径，并 7zip/签名 apk。 7zip 比原来的压缩性能更好，节省了大约 1mb。目前不支持 Signature schemeV2。简而言之，它混淆了 res 名称和路径，并使用 7zip 进行更好的压缩。

for apk in $(find ./<project path>/build/outputs/ -name '\*.apk' );

do

printf "\nshrinking ${apk}\n"

java -jar andredguard/AndResGuard-cli-1.1.11.jar $apk -config ./andredguard/config.xml -zipalign ./andredguard/zipalign\_unix

done

该脚本在 unix 上的 Jenkins 上运行。 zipalign\_unix 是 unix 的 zipalign（不是您在 mac 上使用的那个），或者您可以指定 zipalign 文件夹的路径。

在配置文件中，您应该指定白名单，包括一些用于 3rd 方库的字符串、启动器图标以及与 getIdentifier() 一起使用的资源。注意如果你没有安装 7zip，你应该安装。这是 andresguard 的示例配置文件：

<?xml version="1.0" encoding="UTF-8"?>

<resproguard>

<!--defaut property to set -->

<issue id="property">

<!--whether use 7zip to repackage the signed apk, you must install the 7z command line version in window -->

<!--sudo apt-get install p7zip-full in linux -->

<!--and you must write the sign data fist, and i found that if we use linux, we can get a better result -->

<seventzip value="true"/>

<!--the sign data file name in your apk, default must be META-INF-->

<!--generally, you do not need to change it if you dont change the meta file name in your apk-->

<metaname value="META-INF"/>

<!--if keep root, res/drawable will be kept, it won't be changed to such as r/s-->

<keeproot value="false"/>

</issue>

<!--whitelist, some resource id you can not proguard, such as getIdentifier-->

<!--isactive, whether to use whitelist, you can set false to close it simply-->

<issue id="whitelist" isactive="true">

<!--you must write the full package name, such as com.tencent.mm.R -->

<!--for some reason, we should keep our icon better-->

<!--and it support \*, ?, such as com.tencent.mm.R.drawable.emoji\_\*, com.tencent.mm.R.drawable.emoji\_?-->

<path value="<packageName>.R.drawable.icon"/>

<path value="<packageName>.R.string.com.crashlytics.\*"/>

<path value="<packageName>.R.string.google\_app\_id"/>

<path value="<packageName>.R.string.gcm\_defaultSenderId"/>

<path value="<packageName>.R.string.default\_web\_client\_id"/>

<path value="<packageName>.R.string.ga\_trackingId"/>

<path value="<packageName>.R.string.firebase\_database\_url"/>

<path value="<packageName>.R.string.google\_api\_key"/>

<path value="<packageName>.R.string.google\_crash\_reporting\_api\_key"/>

<!--images which will be loaded with getResources().getIdentifier() should be put in here-->

<path value="<packageName>.R.drawable.drawable\_loaded\_with\_getidentifier"/>

<path value="<packageName>.R.drawable.if\_you\_have\_other\_launcher"/>

</issue>

<!--keepmapping, sometimes if we need to support incremental upgrade, we should keep the old mapping-->

<!--isactive, whether to use keepmapping, you can set false to close it simply-->

<!--if you use -mapping to set keepmapping property in cammand line, these setting will be overlayed-->

<issue id="keepmapping" isactive="false">

<!--the old mapping path, in window use \, in linux use /, and the default path is the running location-->

<path value="{your\_mapping\_path}"/>

</issue>

<!--compress, if you want to compress the file, the name is relative path, such as resources.arsc, res/drawable-hdpi/welcome.png-->

<!--what can you compress? generally, if your resources.arsc less than 1m, you can compress it. and i think compress .png, .jpg is ok-->

<!--isactive, whether to use compress, you can set false to close it simply-->

<issue id="compress" isactive="true">

<!--you must use / separation, and it support \*, ?, such as \*.png, \*.jpg, res/drawable-hdpi/welcome\_?.png-->

<path value="\*.png"/>

<path value="\*.jpg"/>

<path value="\*.jpeg"/>

<path value="\*.gif"/>

<!-- <path value="resources.arsc"/> -->

</issue>

<!--sign, if you want to sign the apk, and if you want to use 7zip, you must fill in the following data-->

<!--isactive, whether to use sign, you can set false to close it simply-->

<!--if you use -signature to set sign property in cammand line, these setting will be overlayed-->

<issue id="sign" isactive="true">

<!--the signature file path, in window use \, in linux use /, and the default path is the running location-->

<path value="./<your\_keystore>.keystore"/>

<!--storepass-->

<storepass value="<storepass>"/>

<!--keypass-->

<keypass value="<keypass>"/>

<!--alias-->

<alias value="<alias>"/>

</issue>

</resproguard>

可以将脚本集成到您的构建脚本中，以便它会自动重新打包。它需要不到一分钟的时间，但确实会对 apk 大小产生一些影响。

1. 减少代码大小
2. Proguard

Proguard 是一个从代码中混淆和删除未使用的类的工具。它将做的是遍历您的代码结构并删除孤立的代码并对代码进行一些优化/混淆。通过设置 minifyEnable 可以删除未使用的类，如下所示：

android {

buildTypes {

release {

minifyEnabled true

proguardFiles '../proguard.cfg'

}

}

...

}

Proguard.cfg 包含删除/混淆代码时要省略的规则和内容。请注意，您可能需要根据您使用的 3rd 方库修改 proguard 配置。您可以从以下位置获取默认配置：/tools/proguard/proguard-android.txt。本文未介绍如何设置 proguard.cfg 文件。

从官方网站来看，单个枚举可以为您的应用程序的 classes.dex 文件增加大约 1.0 到 1.4 KB 的大小。但压缩后的差异应该小于 20kb。 Proguard 默认删除它们并使其成为常量整数，只是您可能需要修改配置以防止崩溃。

如果在 minifyEnable 之后添加 shrinkResources true，Proguard 确实提供了一种删除未引用资源的方法。只是它不会删除 value 文件夹中的文件。

1. R8
2. Redex

Redex（https://github.com/facebook/redex） 是 facebook 的一个开源库，可以优化你的 dex 代码。它用作 apk-in apk-out 工具。安装：

brew install autoconf automake libtool python3

brew install boost jsoncpp

并应用 dex 优化：

redex <input apk> -o <output apk>

请注意，您不应直接使用 redex 中的工具进行签名，因为它使用常规压缩。您应该使用上一节中的 Andresguard。如果你同时应用这两种优化，Andresguard 应该是最后一个可以使用 7zip 进行压缩的。

根据我的测试，它会在 8.4mb dex 上压缩前减少 200kb，在 apk 大小上总共节省 80kb。好吧，总比没有好。我没有将它应用于已发布的 apk，因此我不确定是否会有一些并发症。

从官方博客来看，它的作用是删除永远无法到达的死代码，内联函数/参数，并删除一些只能用于调试的字符串。它应该会减少内存占用，但我没有运行基准测试它的改进程度。

1. 关于第 3 方库

dex 膨胀的主要原因之一是您使用了不是为移动平台设计的库。有些甚至带有从未使用过的资源。 Android 平台及其库在不断变化，有时您的库已被废弃，但仍占用您的 apk。您应该每隔几个月进行一次第 3 方库审查。

可以将以下脚本添加到您的 gradle：

// ./gradlew libsize to list each lib size

task(libsize) << {

def size = 0;

configurations.<config>.collect { it.length() / (1024 \* 1024) }.each { size += it }

println "Total dependencies size: ${Math.round(size \* 100) / 100} Mb"

configurations

.<config>

.sort { -it.length() }

.each { println "${it.name} : ${Math.round(it.length() / (1024) \* 100) / 100} kb" }

}

根据 buildTypes 和 productFlavors 的名称，配置可能会有所不同。您可以通过打印 gradle 中的所有配置来获取配置：

task(configs) << {

configurations.each { println it.name }

}

通过运行 libsize，它将打印出每个 aar 的大小，您可以尝试一劳永逸地删除该库，或者修剪未使用的类。