|  |
| --- |
| 1.好处  减少应用程序安装包的大小，不仅仅减少了用户的网络数据流量，还减少了下载等待的时间。  2.思路  通常来说，减少程序安装包的大小有两条规律：要么减少程序资源的大小，要么就是减少程序的代码量。  3.压缩代码  开启混淆 minifyEnabled true。  android {  buildTypes {  release {  minifyEnabled true//false //true//开启混淆  shrinkResources true//false//true//开启资源压缩  zipAlignEnabled true//开启Zipalign优化  proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'  }  }  }  4.精简类库，jar包  5.精简 so 资源：微信、微博、QQ 只保留了 armeabi，Facebook、Twitter、Instagram 只保留了 armeabi\_v7a。  6.安装包拆分：由于国内应用分发市场的现状，这一条几乎没有办法执行。  7.支持插件化  8.有损压缩图片  9.使用更小的图片  10. 尽量不要在项目中使用帧动画 |

# 好处

减少应用程序安装包的大小，不仅仅减少了用户的网络数据流量还减少了下载等待的时间。

# 思路

通常来说，减少程序安装包的大小有两条规律：要么减少程序资源的大小，要么就是减少程序的代码量。

# APK 构成

分析APK 由以下主要部分组成：

lib/ 存放 so 文件，可能会有 armeabi、armeabi-v7a、arm64-v8a、x86、x86\_64、mips，大多数情况下只需要支持 armabi 与 x86 的架构即可，如果非必需，可以考虑拿掉 x86 的部分。

res/ 存放编译后的资源文件，例如：drawable、layout 等等。

assets/ 应用程序的资源，应用程序可以使用 AssetManager 来检索该资源。

META-INF/ 该文件夹一般存放于已经签名的 APK 中，它包含了 APK 中所有文件的签名摘要等信息。

classes(n).dex classes 文件是 Java Class，被 DEX 编译后可供 Dalvik/ART 虚拟机所理解的文件格式。

resources.arsc 编译后的二进制资源文件。

AndroidManifest.xml Android 的清单文件，格式为 AXML，用于描述应用程序的名称、版本、所需权限、注册的四大组件。

当然还会有一些其它的文件，例如上图中的 org/、src/、push\_version 等文件或文件夹。

# 压缩代码

可以通过开启 ProGuard 来实现代码压缩，可以在 build.gradle 文件相应的构建类型中添加 minifyEnabled true。

打开这些编译属性之后，程序在打包的时候就不会把没有引用到的代码编译进来，以此达到减少安装包大小的目的。

android {

buildTypes {

release {

minifyEnabled true//false //true//开启混淆

shrinkResources true//false//true//开启资源压缩

zipAlignEnabled true//开启Zipalign优化

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

}

}

}

除了 minifyEnabled 属性外，还有用于定义 ProGuard 规则的 proguardFiles 属性：

getDefaultProguardFile(‘proguard-android.txt') 是从Android SDK tools/proguard/ 文件夹获取默认 ProGuard 设置。

proguard-rules.pro 文件用于添加自定义 ProGuard 规则。默认情况下，该文件位于模块根目录（build.gradle 文件旁）。

# 精简类库，jar包

# 精简 so 资源

一般的应用完全可以根据自己业务需求选择使用 armeabi 或者 armeabi\_v7a 一种支持就行。

比如：微信、微博、QQ 只保留了 armeabi，Facebook、Twitter、Instagram 只保留了 armeabi\_v7a。

# 安装包拆分

设想一下，一个 low dpi，API<14 的用户手机下载安装的 APK 里面却包含了大量 xxhdpi 的资源文件，对于这个用户来说，这个 APK 是存在很大的资源浪费的。

幸好 Android 平台为我们提供了拆分 APK 的方法，它能够根据 API Level，屏幕大小以及 GPU 版本的不同进行拆分，使得对应平台的用户下载到最合适自己手机的安装包。

由于国内应用分发市场的现状，这一条几乎没有办法执行。

# 8.支持插件化

插件化技术雨后春笋一样的都冒了出来，这些技术支持动态的加载代码和动态的加载资源，把 APP 的一部分分离出来了，对于业务庞大的项目来说非常有用，极大的分解了 APK 大小。

因为插件化技术需要一定的技术保障和服务端系统支持，有一定的风险，如无必要（比如一些小型项目，也没什么扩展业务）就不需要了，建议酌情选择。

# 有损压缩图片

Android 打包本身会对 PNG 进行无损压缩，所以使用像 Tinypng 这样的有损压缩是有必要的。

重点是 Tinypng 使用智能有损压缩技术，以尽量少的失真换来图片大小的锐减，效果非常好，强烈推荐。

# 使用更小的图片

如果对于非透明的大图，JPG 将会比 PNG 的大小有显著的优势，虽然不是绝对的，但是通常会减小到一半都不止。在启动页，活动页等之类的大图展示区采用 JPG 将是非常明智的选择。

# 10. 尽量不要在项目中使用帧动画

一个帧动画几十张图片，再怎么压缩都还是占很大内存比重的。所以建议是让UI去搞，这里可以参考使用lottie-android，如果项目中动画效果多的话效果更加明显。当然这就要辛苦我们UI设计师大大了。