Android包大小优化的多一种方式

前言

很多时候，需要对android apk包大小进行优化，目前几种常见的方式如下：

- 混淆优化
- android lint检查无用资源
- 压缩工具压缩资源图片
- 资源图片去重
- 使用webp、矢量图等
- 资源混淆

本次要讨论的不是以上资源优化等方式，而是对于apk中常用到本地类库(so)进行压缩，达成优化包大小的目的。不过这里也有一个前提，能够优化的so是能够延迟加载的，即不是必须app启动时就要即时加载的。

实现思路

- 干预gradle apk打包流程，在gradle merge本地库之后，package apk之前将原文件进行压缩，生成压缩文件并保存到assets目录之下。
- task的执行顺序(Develop为productFlavor名称)：

- 在app启动时，解压assets目录下的压缩文件，反射classloader，加入解压后的本地库路径

使用方式

- 在build.gradle的dependencies中加入

```
classpath 'com.hangman.plugin:nativelibcompressionplugin:1.1.5'
```

- 主module的gradle.gradle中应用插件

```
apply plugin: 'nativelibcompressionplugin'
```

- 定义extension

```
soCompressConfig {
    // tarFileNameArray定义了需要打包压缩的本地库文件列表
    tarFileNameArray = ['test1.so', 'test2.so', 'test3.so']
    // compressFileNameArray 需要压缩本地库文件文件名
    compressFileNameArray = ['test4.so', 'test5.so']
    // optinal属性 是否打印整个过程的日志，默认false
    printLog = true
    // optional属性 本地库filter，默认armeabi-v7a
    abiFilters = ['armeabi-v7a']
    // optional属性 压缩算法，apache commons compress支持的算法，默认为lzma
    algorithm = 'lzma'
    // optional属性 debug包时是否执行本工具，默认为false
    debugModeEnable = false
    // optional属性，压缩过程中是否对文件进行校验，默认为true
    verify = true
}
```

- 运行时解压与反射库 在主module的build.gradle中加入

```
implementation 'com.hangman.library:NativeLibDecompression:1.1.7'
```

在Application的onCreate方法中解压

```
val nativeLibDecompression = NativeLibDecompression(context!!, DEFAULT_ALGORITHM, true)
nativeLibDecompression.decompression(false, object : SpInterface {
    override fun saveString(key: String, value: String) {
        // 解压完成后保存文件名与MD5
        globalSp.putString(key, value)
    }

    override fun getString(key: String): String {
        return globalSp.getString(key)
    }
}, object : LogInterface {
    override fun logE(tag: String, message: String) {
        // 打印日志
        Log.e(TAG, message)
    }

    override fun logV(tag: String, message: String) {
        Log.e(TAG, message)
    }
}, object : DecompressionCallback {
    // result 是否成功  hadDecompressed 是否进行过解压操作
    override fun decompression(result: Boolean, hadDecompressed: Boolean) {
        Log.e(TAG, "decompression result = $result  hadDecompressed = $hadDecompressed")
    }
})
```

实现代码

- **SoCompressPlugin**自定义gradle plugin 创建task，task主要工作是对配置的本地库文件进行压缩，生成压缩文件保存到assets目录下。

```
@Override
void apply(Project project) {
    noteApply()
    def extension = project.extensions.create('soCompressConfig', SoCompressConfig)
    project.afterEvaluate {
        project.android.applicationVariants.all { variant ->
        addTaskDependencies(project, variant.name, extension)
    }
    project.gradle.taskGraph.addTaskExecutionListener(new TaskExecutionListener() {
        def time = 0

        @Override
        void beforeExecute(Task task) {
            time = System.currentTimeMillis()
        }

        @Override
        void afterExecute(Task task, TaskState taskState){
            if (task instanceof SoCompressTask) {
                def map = task.infoMap
                def compressTotalTime = 0
                def uncompressTotalTime = 0
                    if (!map.isEmpty()) {
                        map.each {
                            compressTotalTime +=it.value.compressTime
                            uncompressTotalTime +=it.value.uncompressTime
                        }
                }
                println "task ${task.name} cost ${System.currentTimeMillis() - time}  [compress cost
${compressTotalTime} , uncompress cost ${uncompressTotalTime}]"}
            }
        })
    }
}
```

在apply方法中，主要是添加自定义task，并记录其执行时间

```
def addTaskDependencies(Project project, String variantName, SoCompressConfig extension) {
    def uppercaseFirstLetterName = uppercaseFirstLetter(variantName)
    def preTask =
project.tasks.getByName("transformNativeLibsWithMergeJniLibsFor${uppercaseFirstLetterName}")
    def followTask = project.tasks.getByName("package${uppercaseFirstLetterName}")
    def printLog = extension.printLog
    def debugModeEnable = extension.debugModeEnable
    def abiFilters = extension.abiFilters
    if (preTask == null || followTask == null) {
        return
    }
    if (debugModeEnable || (!variantName.endsWith('Debug') && !variantName.endsWith('debug'))) {
        if (printLog) {
            println "add task for variant $variantName"
        }
        //def abiFilters = project.android.defaultConfig.ndk.abiFilters
        //if (printLog) {
        //    println "abiFilters =  $abiFilters"
        //}
        SoCompressTask task = project.tasks.create("soCompressFor$uppercaseFirstLetterName",
SoCompressTask) {
            abiFilterSet = abiFilters
            taskVariantName = variantName
            config = extension
            inputFileDir = preTask.outputs.files.files
            outputFileDir = followTask.inputs.files.files
        }
        task.dependsOn preTask
        if (printLog) {
            println '=========================================='
            println "${task.name} dependsOn ${preTask.name}"
        }
        followTask.dependsOn task
        if (printLog) {
            println "${followTask.name} dependsOn ${task.name}"
            println '=========================================='
        }
    }
}
```

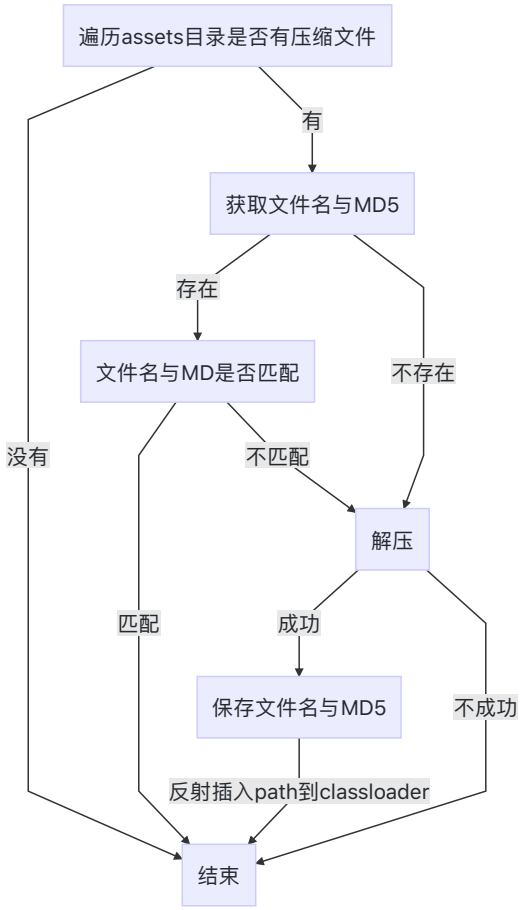
初始化自定义task，传入自定义extension配置项。在自定义配置项时，由于能够直接通过代码读取，本来没有打算把abiFilter当做一个可配置项，同时由于gradle的灵活性，可以在较多地方定义abiFilter，会导致代码的过多的冗余，所以直接将abiFilter用配置项来处理，简化过程，默认值是armeabi-v7a

- **soCompressTask** 自定义的gradle task，主要操作task的输入输出目录，对配置项中的本地库文件进行检查，去重，并压缩生成新文件

```
@TaskAction
void taskAction() {
    def printLog = config.printLog
    if (printLog) {
        println "current variant name is ${taskVariantName}"
    }
    if (inputFileDir == null || outputFileDir == null) {
        if (printLog) {
            print """|inputFileDir $inputFileDir
                     |outputFileDir $outputFileDir""".stripMargin()
        }
        return
    }
    if (printLog) {
        println "taskName ${this.name}"
        println "$config"
    }
    if (!SUPPORT_ALGORITHM.contains(config.algorithm)) {
```

```
        throw new IllegalArgumentException("only support one of
${Arrays.asList(SUPPORT_ALGORITHM).toString()}")
    }

    def gradleVersion = 0

project.rootProject.buildscript.configurations.classpath.resolvedConfiguration.resolvedArtifacts.each
{
        if (it.name == 'gradle') {
            gradleVersion = it.moduleVersion.id.version.replace('.', '').toInteger()
        }
    }
    // 找到输入输出目录
    def libInputFileDir = null
    def libOutputFileDir = null

    inputFileDir.each { file ->
        if (printLog) {
            println "inputFileDir ${file.getAbsolutePath()}"
        }
        if (file.getAbsolutePath().contains('transforms/mergeJniLibs')) {
            libInputFileDir = file
        }
    }
    outputFileDir.forEach { file ->
        if (printLog) {
            println "outputFileDir ${file.getAbsolutePath()}"
        }
        if (gradleVersion >= 320 && file.getAbsolutePath().contains('intermediates/merged_assets')) {
            libOutputFileDir = file
        } else if (gradleVersion < 320 && file.getAbsolutePath().contains('intermediates/assets')) {
            libOutputFileDir = file
        }
    }
    if (libInputFileDir == null) {
        throw new IllegalStateException('libInputFileDir is null')
    }
    if (libOutputFileDir == null) {
        throw new IllegalStateException('libOutputFileDir is null')
    }
    if (printLog) {
        println "libInputFileDir ${libInputFileDir}"
        println "libOutputFileDir ${libOutputFileDir}"
    }
    String[] tarFileArray = config.tarFileNameArray
    String[] compressFileArray = config.compressFileNameArray

    tarFileArray.each { fileName ->
        if (compressFileArray.contains(fileName)) {
            throw new IllegalArgumentException("${fileName} both in tarFileNameArray &
compressFileNameArray")
        }
    }

    def soCompressDir = new File(libOutputFileDir, CompressConstant.SO_COMPRESSED)
    soCompressDir.deleteDir()

    if (tarFileArray.length != 0) {
        tarFileArray.sort()
        compressTar(tarFileArray, libInputFileDir, libOutputFileDir, printLog)
    }

    if (compressFileArray.length != 0) {
        compressFileArray.sort()
        compressSoFileArray(compressFileArray, libInputFileDir, libOutputFileDir, printLog)
    }
}
```

- 压缩与解压 主要用到了Apache Commons Compress™，相关逻辑可以看代码。解压主要发生在app启动时，



效果

lzma压缩方式比zip压缩方式压缩率更高，可以获得较好的文件大小优化， 压缩概况

后记

- 整个工具的思路比较清晰，存在进一步优化的空间
- 对于必须要即时加载的本地库文件不能进行优化
- app启动后，并不会每次安装都会解压，如果本地已经解压过，不会重新解压