参考：<http://blog.csdn.net/heqiangflytosky/article/details/50853268>

[Gradle命令和配置](http://blog.csdn.net/heqiangflytosky/article/details/50853268)

标签： [Android](http://www.csdn.net/tag/Android)[Gradle](http://www.csdn.net/tag/Gradle)[命令](http://www.csdn.net/tag/%e5%91%bd%e4%bb%a4)[配置](http://www.csdn.net/tag/%e9%85%8d%e7%bd%ae)

2016-03-11 09:32 17174人阅读 [评论](http://blog.csdn.net/heqiangflytosky/article/details/50853268#comments)(5) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/heqiangflytosky/article/details/50853268#report)

 分类：

开发工具（9）

版权声明：本文为博主原创文章，未经博主允许不得转载。

目录[(?)[+]](http://blog.csdn.net/heqiangflytosky/article/details/50853268)

[我的博客原文地址](http://www.heqiangfly.com/2016/03/11/development-tool-gradle-command-config/)

Gradle是一种构建工具，它抛弃了基于XML的构建脚本，取而代之的是采用一种基于Groovy的内部领域特定语言，建议可以先熟悉一下Groovy脚本。   
[在线文档](https://docs.gradle.org/current/dsl/)

Gradle命令：

常用命令：

gradle明明一般是./gradlew +参数， gradlew代表 gradle wrapper，意思是gradle的一层包装，大家可以理解为在这个项目本地就封装了gradle，即gradle wrapper， 在gradle/wrapper/gralde-wrapper.properties文件中声明了它指向的目录和版本。只要下载成功即可用grdlew wrapper的命令代替全局的gradle命令。

* ./gradlew -v 版本号
* ./gradlew clean 清除app目录下的build文件夹
* ./gradlew build 检查依赖并编译打包
* ./gradlew tasks 列出所有task

这里注意的是 ./gradlew build 命令把debug、release环境的包都打出来，如果正式发布只需要打Release的包，该怎么办呢，下面介绍一个很有用的命令 assemble， 如：

* ./gradlew assembleDebug 编译并打Debug包
* ./gradlew assembleRelease 编译并打Release的包

除此之外，assemble还可以和productFlavors结合使用：

* ./gradlew installRelease Release模式打包并安装
* ./gradlew uninstallRelease 卸载Release模式包

加入自定义参数：

比如我们想根据不同的参数来进行不用的编译配置，可以在./gradlew中加入自定义参数。

* ./gradlew assembleDebug -Pcustom=true

就可以在build.gradle中使用下面代码来判断：

if (project.hasProperty('custom')){

}

* 1
* 2
* 3
* 1
* 2
* 3

assemble结合Build Variants来创建task

assemble 还能和 Product Flavor 结合创建新的任务，其实 assemble 是和 Build Variants 一起结合使用的，而 Build Variants = Build Type + Product Flavor，举个例子大家就明白了：   
如果我们想打包 wandoujia 渠道的release版本，执行如下命令就好了：

* ./gradlew assembleWandoujiaRelease

如果我们只打wandoujia渠道版本，则：

* ./gradlew assembleWandoujia

此命令会生成wandoujia渠道的Release和Debug版本   
同理我想打全部Release版本：

* ./gradlew assembleRelease

这条命令会把Product Flavor下的所有渠道的Release版本都打出来。   
总之，assemble 命令创建task有如下用法：

1. assemble<Variant Name>： 允许直接构建一个Variant版本，例如assembleFlavor1Debug。
2. assemble<Build Type Name>： 允许构建指定Build Type的所有APK，例如assembleDebug将会构建Flavor1Debug和Flavor2Debug两个Variant版本。
3. assemble<Product Flavor Name>： 允许构建指定flavor的所有APK，例如assembleFlavor1将会构建Flavor1Debug和Flavor1Release两个Variant版本。

Gradle配置：

Gradle构建脚本 build.gradle:   
Gradle属性文件 gradle.properties   
Gradle设置文件 settings.gradle

build.gradle

先看整个项目的gradle配置文件：

buildscript {

repositories {

jcenter()

}

dependencies {

classpath 'com.android.tools.build:gradle:1.3.0'

// NOTE: Do not place your application dependencies here; they belong

// in the individual module build.gradle files

}

}

allprojects {

repositories {

jcenter()

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17

内容主要包含了两个方面：一个是声明仓库的源，这里可以看到是指明的jcenter(), 之前版本则是mavenCentral(), jcenter可以理解成是一个新的中央远程仓库，兼容maven中心仓库，而且性能更优。   
另一个是声明了[**Android**](http://lib.csdn.net/base/android) gradle plugin的版本，[**android**](http://lib.csdn.net/base/android) studio 1.0正式版必须要求支持gradle plugin 1.0的版本

某个Moudle的gradle配置文件：

buildscript

buildscript {

repositories {

maven { url 'http://\*\*\*\*\*\*\*\*\*' }

}

dependencies {

classpath 'com.android.tools.build:gradle:1.3.1'

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* buildscript{}设置脚本的运行环境。
* repositories{}支持java依赖库管理，用于项目依赖。
* dependencies{}依赖包的定义。支持maven/ivy，远程，本地库，也支持单文件。如果前面定义了repositories{}maven 库，则使用maven的依赖库，使用时只需要按照用类似于com.android.tools.build:gradle:0.4，gradle 就会自动的往远程库下载相应的依赖。

apply

//声明引用 com.android.application 插件，那么这个模块就是一个Android应用程序

apply plugin: 'com.android.application'

* 1
* 2
* 1
* 2
* apply plugin:声明引用插件的类型。如果是库的话就加

apply plugin: 'com.android.library'

* 1
* 1
* apply from：表示引用其他的配置文件，比如 apply from:"config.gradle"

android

android {

// 编译SDK的版本

compileSdkVersion 22

// build tools的版本

buildToolsVersion "23.0.1"

//aapt配置

aaptOptions {

//不用压缩的文件

noCompress 'pak', 'dat', 'bin', 'notice'

//打包时候要忽略的文件

ignoreAssetsPattern "!.svn:!.git"

//分包

multiDexEnabled true

//--extra-packages是为资源文件设置别名：意思是通过该应用包名+R，com.android.test1.R和com.android.test2.R都可以访问到资源

additionalParameters '--extra-packages', 'com.android.test1','--extra-packages','com.android.test2'

}

//默认配置

defaultConfig {

//应用的包名

applicationId "com.example.heqiang.androiddemo"

minSdkVersion 21

targetSdkVersion 22

versionCode 1

versionName "1.0"

}

//编译配置

compileOptions {

// java版本

sourceCompatibility JavaVersion.VERSION\_1\_7

targetCompatibility JavaVersion.VERSION\_1\_7

}

//源文件目录设置

sourceSets {

main {

//jni lib的位置

jniLibs.srcDirs = jniLibs.srcDirs << 'src/jniLibs'

//定义多个资源文件夹,这种情况下，两个资源文件夹具有相同优先级，即如果一个资源在两个文件夹都声明了，合并会报错。

res.srcDirs = ['src/main/res', 'src/main/res2']

//指定多个源文件目录

java.srcDirs = ['src/main/java', 'src/main/aidl']

}

}

//签名配置

signingConfigs {

debug {

keyAlias 'androiddebugkey'

keyPassword 'android'

storeFile file('keystore/debug.keystore')

storePassword 'android'

}

}

buildTypes {

//release版本配置

release {

debuggable false

// 是否进行混淆

minifyEnabled true

//去除没有用到的资源文件，要求minifyEnabled为true才生效

shrinkResources true

// 混淆文件的位置

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

signingConfig signingConfigs.debug

//ndk的一些相关配置，也可以放到defaultConfig里面。

//指定要ndk需要兼容的架构(这样其他依赖包里mips,x86,arm-v8之类的so会被过滤掉)

ndk {

abiFilter "armeabi"

}

}

//debug版本配置

debug {

debuggable true

// 是否进行混淆

minifyEnabled false

//去除没有用到的资源文件，要求minifyEnabled为true才生效

shrinkResources true

// 混淆文件的位置

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

signingConfig signingConfigs.debug

//ndk的一些相关配置，也可以放到defaultConfig里面。

//指定要ndk需要兼容的架构(这样其他依赖包里mips,x86,arm-v8之类的so会被过滤掉)

ndk {

abiFilter "armeabi"

}

}

}

// lint配置

lintOptions {

//移除lint检查的error

abortOnError false

//禁止掉某些lint检查

disable 'NewApi'

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75
* 76
* 77
* 78
* 79
* 80
* 81
* 82
* 83
* 84
* 85
* 86
* 87
* 88
* 89
* 90
* 91
* 92
* 93
* 94
* 95
* 96
* 97
* 98
* 99
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75
* 76
* 77
* 78
* 79
* 80
* 81
* 82
* 83
* 84
* 85
* 86
* 87
* 88
* 89
* 90
* 91
* 92
* 93
* 94
* 95
* 96
* 97
* 98
* 99

android{}设置编译android项目的参数，构建android项目的所有配置都写在这里。   
除了上面写的，在android{}块中可以包含以下直接配置项：

* productFlavors{ } 产品风格配置，ProductFlavor类型
* testOptions{ } 测试配置，TestOptions类型
* dexOptions{ } dex配置，DexOptions类型
* packagingOptions{ } PackagingOptions类型
* jacoco{ } JacocoExtension类型。 用于设定 jacoco版本
* splits{ } Splits类型。

几点说明：

* 文件开头apply plugin是最新gradle版本的写法，以前的写法是apply plugin: 'android', 如果还是以前的写法，请改正过来。
* minifyEnabled也是最新的语法，很早之前是runProguard,这个也需要更新下。
* proguardFiles这部分有两段，前一部分代表系统默认的android程序的混淆文件，该文件已经包含了基本的混淆声明，免去了我们很多事，这个文件的目录在 sdk目录/tools/proguard/proguard-android.txt , 后一部分是我们项目里的自定义的混淆文件，目录就在 app/proguard-rules.txt , 如果你用Studio 1.0创建的新项目默认生成的文件名是 proguard-rules.pro , 这个名字没关系，在这个文件里你可以声明一些第三方依赖的一些混淆规则。最终混淆的结果是这两部分文件共同作用的。
* aaptOptions更多介绍 <http://blog.csdn.net/heqiangflytosky/article/details/51009123>

repositories

repositories {

flatDir {

//本地jar依赖包路径

dirs '../../../../main/libs'

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 1
* 2
* 3
* 4
* 5
* 6

dependencies

dependencies {

compile files('libs/android-support-v4.jar')

//在flatDir.dirs下面找依赖的aar

compile (name:'ui', ext:'aar')

// 编译extras目录下的ShimmerAndroid模块

// 使用transitive属性设置为false来排除所有的传递依赖，默认为true

compile project(':extras:ShimmerAndroid'){

transitive = false

}

// 编译CommonSDK模块，但是去掉此模块中对com.android.support的依赖，防止重复依赖报错

compile (project(':CommonSDK')) { exclude group: "com.android.support" }

provided fileTree(dir: 'src/android5/libs', include: ['\*.jar'])

provided 'com.android.support:support-v4:21.0.3'

provided project(':main-host')

//通用使用exclude排除support-compat模块的依赖

compile ('com.jakewharton:butterknife:8.5.1'){

exclude module: 'support-compat'

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* compile和provided   
  compile表示编译时提供并打包进apk。   
  provided表示只在编译时提供，不打包进apk。
* exclude 防止重复依赖，后面会重点介绍
* transitive 排除所有的传递依赖，后面会重点介绍
* include

CommonSDK模块的定义可以参考settings.gradle   
其他的介绍可以参考 依赖库管理。

几点说明

* 看到上面的两个一模一样的repositories和dependencies了吗？他们的作用是不一样的，在buildscript里面的那个是插件初始化环境用的，用于设定插件的下载仓库，而外面的这个是设定工程依赖的一些模块和远程library的下载仓库的。

settings.gradle

这个文件是全局的项目配置文件，里面主要声明一些需要加入gradle的module。   
一般在setting.gradle中主要是调用include方法，导入工程下的各个子模块。   
那我们在setting.gradle里面还能写什么呢？因为setting.gradle对应的是gradle中的Settings对象，那查下Settings的文档（<https://docs.gradle.org/current/dsl/org.gradle.api.initialization.Settings.html>），看下它都有哪些方法，哪些属性，就知道在setting.gradle能写什么了；

include ':AndroidDemo'

include ':CommonSDK'

project(':CommonSDK').projectDir = new File(settingsDir, '../../CommonSDK/')

* 1
* 2
* 3
* 4
* 1
* 2
* 3
* 4

include调用后，生成了一个名为:CommonSDK的Project对象，project(':CommonSDK')取出这个对象，设置Project的 projectDir属性。projectDir哪里来的？请看Project类的文档。

gradle.properties

可以在 gradle.properties 文件中配置一些变量，这些变量在这个工程下的所有module的build.gradle文件里都可以使用。这样就可以把一些共用的变量放到这里，这样后面修改的时候就可以只修改这个变量，不用各个模块都要修改了。   
比如我们在 gradle.properties SDK 版本以及应用的版本号：

MIN\_SDK\_VERSION=21

TARGET\_SDK\_VERSION=22

VERSION\_CODE=200100

VERSION\_NAME=2.1.0

* 1
* 2
* 3
* 4
* 1
* 2
* 3
* 4

那么在 build.gradle 中可以通过project进行引用：

defaultConfig {

applicationId "com.example.heqiang.testsomething"

minSdkVersion project.MIN\_SDK\_VERSION as int

targetSdkVersion project.TARGET\_SDK\_VERSION as int

versionCode project.VERSION\_CODE as int

versionName project.VERSION\_NAME

/\*

\* as int 关键字是用来进行类型转换的，因为从配置里面读取出来的默认是字符串

\*还可以使用 versionCode Integer.parseInt(project.VERSION\_CODE)

\* \*/

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11

调试

我们在进行一些配置的时候可能需要知道一些变量的值，这时候可以在 build.gradle 中添加打印进行调试，比如：

defaultConfig {

applicationId "com.example.hq.testsomething"

minSdkVersion project.MIN\_SDK\_VERSION as int

targetSdkVersion project.TARGET\_SDK\_VERSION as int

versionCode project.VERSION\_CODE as int

versionName project.VERSION\_NAME

println('\*\* build versionName=' + versionName)

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9

在 Gradle Console 中就可以看到打印：

\*\* build versionName=2.1.0

* 1
* 1

依赖库管理

本地依赖

dependencies {

//单文件依赖

compile files('libs/android-support-v4.jar')

//某个文件夹下面全部依赖

compile fileTree(dir: 'src/android6/libs', include: ['\*.jar'])

compile (name:'ui', ext:'aar')

compile (project(':CommonSDK')) { exclude group: "com.android.support" }

provided fileTree(dir: 'src/android5/libs', include: ['\*.jar'])

provided 'com.android.support:support-v4:21.0.3'

provided project(':main-host')

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11

远程依赖

gradle同时支持maven，ivy，以maven作为例子：

repositories {

//从中央库里面获取依赖

mavenCentral()

//或者使用指定的本地maven 库

maven{

url "file://F:/githubrepo/releases"

}

//或者使用指定的远程maven库

maven{

url "https://github.com/youxiachai/youxiachai-mvn-repo/raw/master/releases"

}

}

dependencies {

//应用格式: packageName:artifactId:version

compile 'com.google.android:support-v4:r13'

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17

项目依赖

对于项目依赖android library的话，在这里需要使用gradle mulit project机制。   
Mulit project设置是gradle约定的一种格式，如果需要编译某个项目之前，要先编译另外一个项目的时候，就需要用到。结构如下（来自于官方文档）：

MyProject/

| settings.gradle

+ app/

| build.gradle

+ libraries/

+ lib1/

| build.gradle

+ lib2/

| build.gradle

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9

需要在workplace目录下面创建settings.gradle 的文件，然后在里面写上：

include ':app', ':libraries:lib1', ':libraries:lib2'

* 1
* 1

例如：

include ':AndroidDemo'

include ':CommonSDK'

project(':CommonSDK').projectDir = new File(settingsDir, '../../CommonSDK/')

* 1
* 2
* 3
* 4
* 1
* 2
* 3
* 4

如此，gradle mutil project 就设置完毕。   
对于app project如果需要应用libraries目录下的lib1，只需要在app project的build.gradle文件里的依赖中这么写：

compile project(':libraries:lib1')

* 1
* 1

类似前面的

provided project(':main-host')

* 1
* 1

即可完成，写完以后可以用gradle dependencies可以检查依赖状况

Gradle依赖的统一管理

我们可以在项目的根目录创建一个gradle配置文件config.gradle，内容如下：

ext{

android=[

compileSdkVersion: 22,

buildToolsVersion: "23.0.1",

minSdkVersion: 21,

targetSdkVersion: 22,

versionCode: 1,

versionName: "1.0"

]

dependencies=[

compile:'com.android.support:support-v4:21.0.3',

compile: (project(':CommonSDK')) { exclude group: "com.android.support" },

provided: fileTree(dir: 'src/android5/libs', include: ['\*.jar']),

provided: project(':main-host')

]

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16

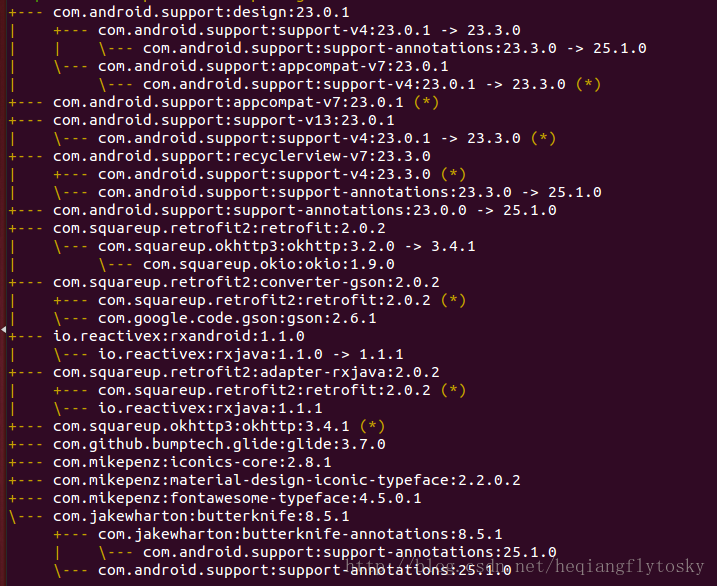
targetSdkVersion的版本还有依赖库的版本升级都在这里进行统一管理，所有的module以及主项目都从这里同意读取就可以了。   
在build.gradle文件中加入：

apply from:"config.gradle"

* 1
* 1

意思是所有的子项目或者所有的modules都可以从这个配置文件中读取内容。   
android节点读取ext中android对应项，dependencies读取dependencies对应项，如果配置有变化就可以只在config.gradle中修改，是不是很方便进行配置的管理呢？

检查依赖报告

运行命令./gradlew <projectname>:dependencies --configuration compile （projectname为settings.gradle里面配置的各个project，如果没有配置，直接运行./gradlew dependencies --configuration compile），会把依赖树会打印出来，依赖树显示了你 build 脚本声明的顶级依赖和它们的传递依赖：   
  
仔细观察你会发现有些传递依赖标注了\*号，表示这个依赖被忽略了，这是因为其他顶级依赖中也依赖了这个传递的依赖，Gradle会自动分析下载最合适的依赖。

排除传递依赖

Gradle允许你完全控制传递依赖，你可以选择排除全部的传递依赖也可以排除指定的依赖。

* exclude：前面已经介绍过，可以设置不编译指定的模块，排除指定模块的依赖。后的参数有group和module，可以分别单独使用，会排除所有匹配项。

// 编译CommonSDK模块，但是去掉此模块中对com.android.support的依赖，防止重复依赖报错

compile (project(':CommonSDK')) { exclude group: "com.android.support" }

compile ('com.jakewharton:butterknife:8.5.1'){

exclude module: 'support-compat'

exclude group: 'com.android.\*\*.\*\*\*', module: '\*\*\*-\*\*\*'

}

* 1
* 2
* 3
* 4
* 5
* 6
* 1
* 2
* 3
* 4
* 5
* 6
* transitive：前面已经介绍过，用于自动处理子依赖项，默认为true，gradle自动添加子依赖项。设置为false排除所有的传递依赖，可以用来解决一些依赖冲突的问题，比如一些 Error:java.io.IOException: Duplicate zip entry 报错。

// 使用transitive属性设置为false来排除所有的传递依赖

compile project(':extras:ShimmerAndroid'){

transitive = false

}

* 1
* 2
* 3
* 4
* 1
* 2
* 3
* 4
* force：强制设置某个模块的版本。

configurations.all{

resolutionStrategy{

force'org.hamcrest:hamcrest-core:1.3'

}

}

* 1
* 2
* 3
* 4
* 5
* 1
* 2
* 3
* 4
* 5

这样，应用中对org.hamcrest:hamcrest-core 依赖就会变成1.3版本。

动态版本声明

如果你想使用一个依赖的最新版本，你可以使用latest.integration，比如声明 Cargo Ant tasks的最新版本，你可以这样写org.codehaus .cargo:cargo-ant:latest-integration，你也可以用一个+号来动态的声明：

dependencies {

//依赖最新的1.x版本

compile "org.codehaus.cargo:cargo-ant:1.+"

}

* 1
* 2
* 3
* 4
* 1
* 2
* 3
* 4

然后在依赖树里面可以清晰的看到选择了哪个版本：

\--- org.codehaus.cargo:cargo-ant:1.+ -> 1.3.1

* 1
* 1

<http://www.open-open.com/lib/view/open1431391503529.html>   
<http://www.jianshu.com/p/429733dbbc34>

多渠道打包:

主要借助

android {

productFlavors{

……

}

}

* 1
* 2
* 3
* 4
* 5
* 1
* 2
* 3
* 4
* 5

来实现。   
网上多是类似友盟的配置，copy过来：   
<http://blog.csdn.net/maosidiaoxian/article/details/42000913>   
<https://segmentfault.com/a/1190000004050697>   
在AndroidManifest.xml里面写上：

<meta-data

android:name="UMENG\_CHANNEL"

android:value="Channel\_ID" />

* 1
* 2
* 3
* 1
* 2
* 3

里面的Channel\_ID就是渠道标示。我们的目标就是在编译的时候这个值能够自动变化。

android {

productFlavors {

xiaomi {

manifestPlaceholders = [UMENG\_CHANNEL\_VALUE: "xiaomi"]

}

\_360 {

manifestPlaceholders = [UMENG\_CHANNEL\_VALUE: "\_360"]

}

baidu {

manifestPlaceholders = [UMENG\_CHANNEL\_VALUE: "baidu"]

}

wandoujia {

manifestPlaceholders = [UMENG\_CHANNEL\_VALUE: "wandoujia"]

}

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16

或者批量修改

android {

productFlavors {

xiaomi {}

\_360 {}

baidu {}

wandoujia {}

}

productFlavors.all {

flavor -> flavor.manifestPlaceholders = [UMENG\_CHANNEL\_VALUE: name]

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12

然后用 ./gradlew assembleRelease 这条命令会把Product Flavor下的所有渠道的Release版本都打出来。   
assemble<Product Flavor Name>： 允许构建指定flavor的所有APK，例如assembleFlavor1将会构建Flavor1Debug和Flavor1Release两个Variant版本。   
在上面当中，我们也可以指定一个默认的渠道名，如果需要的话。指定默认的值是在defaultConfig节点当中添加如下内容：

manifestPlaceholders = [ CHANNEL\_NAME:"Unspecified"]

* 1
* 1

这里的Unspecified换成你实际上的默认的渠道名。   
使用manifestPlaceholders的这种配置，同样适用于manifest的其他配置。比如你需要在不同渠道发布的apk里面，指定不同的启动Activity。比如在豌豆荚里面发布的，启动的Activity显示的是豌豆荚首发的界面，应用宝里面启动的是应用宝首发的界面（哈哈，有点坏），你就可以对你的activity的值使用 {activity\_name}的方式，然后在productFlavors里面配置这个{activity\_name}的值。