# **Gradle从入门到了解**

#### Gradle基本组件

每一个build.gradle文件代表着一个Project。Tasks在build.gradle中定义。当初始化构建进程时，gradle会基于build文件，集合所有的Project和Tasks,一个Tasks包含了一系列动作，然后它们将会按照顺序执行，一个动作就是一段被执行的代码，很像Java中的方法。

Project  
每一个待编译的工程（可以是一个jar包，一个web应用，或者一个android app等）都称为一个Project。

Task  
每一个Project在构建的时候都包含一系列的Task。一个Task其实就是构建过程中一个原子性的操作。比如一个Android APK的编译可能包含：Java源码编译Task、资源编译Task、JNI编译Task、lint检查Task、打包生成APK的Task、签名Task等。

Plugin  
Gradle是一个框架，作为框架，它负责定义流程和规则。而具体的编译工作则是通过插件的方式来完成的。比如编译Java有Java插件，编译Groovy有Groovy插件，编译Android APP有Android APP插件，编译Android Library有Android Library插件。  
简单来说，插件就是一系列任务的集合，主要作用是把一些重复利用的逻辑打包，这样就可以在不同的项目中可以重复的使用。  
要使用插件，可以通过引入依赖的方式添加。

##### As如何依赖Gradle让Gradle作为自身的构建工具呢？

* 答：Google开发了一个Gradle插件，让As项目依赖这个插件，就相当于让Gradle作为自身的的构建工具。

现在比如我们新建一个As项目，打开项目的根目录的gradle.build文件。有如下代码：

buildscript {

repositories {

jcenter() //表示编译过程中依赖的仓库

}

dependencies {

//依赖android开发使用的gradle插件

classpath 'com.android.tools.build:gradle:2.2.0'

}

}

而要引入Android APP插件，就需要在build.gradle引用Android APP插件：

//申明使用插件，表明要编译的内容和产物

apply plugin: 'com.android.application'

//配置插件属性

android {

compileSdkVersion 24

buildToolsVersion "24.0.1"

defaultConfig {

applicationId "zhj.gradledemo"

minSdkVersion 15

targetSdkVersion 24

versionCode 1

versionName "1.0"

testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"

}

buildTypes {

release {

minifyEnabled false

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

}

}

}

Android其实就是写了两个插件：

* com.android.application和com.android.library。

应用这两个插件就可以实现Android APP和Android Library的构建。

#### As项目中的Gradle

#### .gradle文件夹

.gradle文件夹 是gradle 运行以后生成的缓存文件夹。

#### Project中的build.gradle文件

project下的build.gradle是基于整个project的配置，主要配置gradle 版本及 全局依赖仓库、库或者其他全部参数。

// Top-level build file where you can add configuration options common to all sub-projects/modules.

buildscript {

repositories {

//这里依赖的jcenter仓库是gradle脚本自身需要的资源

jcenter()

}

dependencies {

classpath 'com.android.tools.build:gradle:2.2.0'

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

// in the individual module build.gradle files

}

}

allprojects {

repositories {

//这里依赖的jcenter仓库是项目所有模块需要的资源

jcenter()

}

}

task clean(type: Delete) {

delete rootProject.buildDir

}

#### module中build.gradle文件

//申明使用插件，表明要编译的内容和产物

apply plugin: 'com.android.application'

android {

compileSdkVersion 24

buildToolsVersion "24.0.1"

//默认配置，会同时应用到debug和release版本上

defaultConfig {

applicationId "zhj.gradledemo"

minSdkVersion 15

targetSdkVersion 24

versionCode 1

versionName "1.0"

testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"

}

buildTypes {

release {

minifyEnabled true //是否混淆

proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro' //混淆文件的位置

}

debug {

minifyEnabled false

}

}

// 多渠道

productFlavors {

//可以设置不同渠道渠道号，应用名称

pro {

}

fre {

}

}

}

//依赖第三方库

dependencies {

//编译libs目录下所以jar包

compile fileTree(include: ['\*.jar'], dir: 'libs') //导入所有的jar包

androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {

exclude group: 'com.android.support', module: 'support-annotations'

})

compile 'com.android.support:appcompat-v7:24.2.0'

compile 'com.android.support:design:24.2.0'

testCompile 'junit:junit:4.12'

proCompile 'com.android.support:recyclerview-v7:24.2.0'

}

#### Project中setting.gradle

这个文件是全局的项目配置文件，里面主要声明Project中所包括的所有module

//一个Project中所包括的所有moduleinclude ':Gotobus', ':android-support-v7-appcompat'include ':google-play-services\_lib'include ':TakeTours'include ':Common'include ':CompanyCommon'

#### Project中gradle.properties

gradle.properties为gradle的配置文件，里面可以定义一些常量供build.gradle使用，比如可以配置签名相关信息如keystore位置，密码，keyalias等,build.gradle就可以直接引用  
gradle 中的一些配置参数建议写到gradle.properties

//编译版本信息

APPLICATION\_ID = com.jin.myAPP

COMPILE\_SDK\_VERSION = 23

BUILD\_TOOLS\_VERSION = 23.0.1

MIN\_SDK\_VERSION = 15

TARGET\_SDK\_VERSION = 1

VERSION\_CODE = 1

VERSION\_NAME = 1.0.0.0

//keystore信息

STORE\_FILE = ../app/mykey.keystore

STORE\_PASSWORD = your password

KEY\_ALIAS = your alias

KEY\_PASSWORD = your password

#### 配置应用的签名信息

在android.signingConfigs{}下定义一个或者多个签名信息，然后在buildTypes{}配置使用即可。比如这里

android {

signingConfigs {

release {

storeFile file("release.keystore")

keyAlias "release"

keyPassword "123456"

storePassword "123456"

}

debug {

...

}

}

buildTypes {

release {

signingConfig signingConfigs.release

}

debug {

signingConfig signingConfigs.debug

}

}

}

* storeFile是签名证书文件，keyAlias是别名，keyPassword是key的密码，storePassword是证书的密码。配好好相关信息即可在buildTypes配置使用。

一般重要的信息，例如签名信息，可以直接将信息写到gradle.properties，然后在然后在build.gradle中引用即可。

* buildTypes是指建构的类型，一般只用两种默认类型 debug 和 release ，顾名思义 debug 用来配置开发过程中的一些内容；release 用来配置正式发布版本的内容。有时我们需要发布介于debug与release之间的preview 版本。

#### Build Variant 差异管理

比如app生成不同版本（免费，收费），适配特殊机型，多渠道等需要发多个包，最终能编译出的apk的数量是由Product Flavor（产品种类）与Build Type（构建类型）决定的，  
公式：Build Variant = Build Type x Product Flavor

BuildType（构建类型）  
默认有debug和release两种，标示编译的类型，通常在混淆代码、可调式、资源压缩上做一些区分。

Product Flavor（产品种类）  
为了满足“同一个project，根据一个很小的区分，来打不同的包”这个需求。实现多渠道打包。注意:这里的Flavor名如果是数字开头，必须用引号引起来。

#### 调整module的目录结构sourceSets

默认情况下，java文件和resource文件分别在src/main/java和src/main/res目录下，在build.gradle文件的andorid{}里面添加下面的代码，便可以将java文件和resource文件放到src/java和src/resources目录下。

sourceSets {

main {

manifest.srcFile 'AndroidManifest.xml'

//设置java文件的位置

java.srcDirs = ['src']

resources.srcDirs = ['src']

aidl.srcDirs = ['src']

renderscript.srcDirs = ['src']

res.srcDirs = ['res']

assets.srcDirs = ['assets']

}

}

#### 全局变量定义及引用

可以在顶层build.gradle脚本中定义一些全局变量，提供给子脚本引用

ext {

// global variables definition

compileSdkVersion = 'Google Inc.:Google APIs:23'

buildToolsVersion = "23.0.3"

minSdkVersion = 14

targetSdkVersion = 23

}

子脚本引用

android {

compileSdkVersion rootProject.ext.compileSdkVersion

buildToolsVersion rootProject.ext.buildToolsVersion

defaultConfig {

minSdkVersion rootProject.ext.minSdkVersion

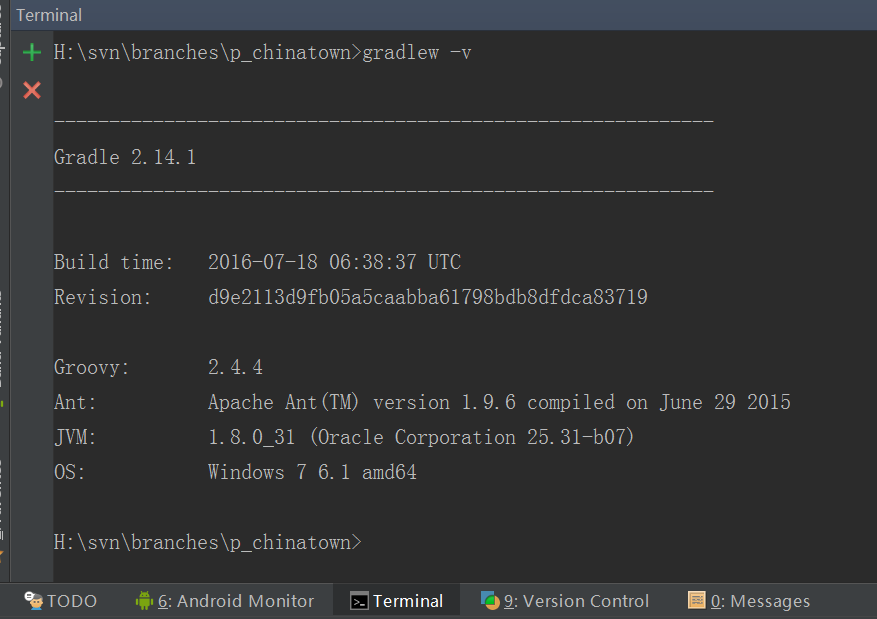
targetSdkVersion rootProject.ext.targetSdkVersion

}

}

#### Gradle常用命令介绍

打开Android Studio内置的Terminal终端，输入如下命令



执行gradlew -v

| **指令代码** | **指令功能** |
| --- | --- |
| gradlew | 下载更新gradle |
| gradlew -v | 查询版本号 |
| gradlew clean | 清楚项目的output文件 |
| gradlew check | 运行检测和测试任务 |
| gradlew build | 运行check和assemble,检查依赖并编译打包（debug、release环境的包） |
| gradlew clean build | 运行 clean 和 build 两个 gradle task |
| gradlew assemble | 编译并打Debug和Release包 |
| gradlew assembleDebug | 编译并把本项目下所有模块所有渠道的Debug版本打包 |
| gradlew assembleRelease | 编译并把本项目下所有模块所有渠道的Release版本打包 |
| gradlew assembleWandoujia | 生成wandoujia渠道的Release和Debug版本 |
| gradlew assembleWandoujiaRelease | 打包wandoujia渠道的Release版本 |
| gradlew assembleWandoujiaRelease -p app | 打包app工程下wandoujia渠道的Release版本（使用-p选项，决定执行哪个工程） |
| gradlew installRelease | Release模式打包并安装 |
| gradlew uninstallRelease | 卸载Release模式包 |

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。

assemble 命令创建task有如下语法：

* 允许直接构建一个Variant版本，例如assembleFlavor1Debug。
* 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。
* 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。

#### 批量修改生成的apk文件名

在我们打包发版的时候，一次性打几十个包，这时候我们就想让生成的apk文件名有区分，比如一眼就能看出这个apk是哪个版本的，哪个渠道的，是哪天打的包等等，这就需要我们在生成apk文件的时候动态修改生成的apk文件名达到这一目的。代码如下：

def buildTime() {

def date = new Date()

def formattedDate = date.format('yyyyMMdd')

return formattedDate

}

android {

buildTypes {

release {

applicationVariants.all { variant ->

variant.outputs.each { output ->

if (output.outputFile != null && output.outputFile.name.endsWith('.apk')

&&'release'.equals(variant.buildType.name)) {

def apkFile = new File(

output.outputFile.getParent(),

"Gtobus\_${variant.flavorName}\_v${variant.versionName}\_${buildTime()}.apk")

output.outputFile = apkFile

}

}

}

}

}

}

以baidu渠道为例，以上的代码会生成一个名字为Gtobus\_\_gotobus\_v5.1.2\_20161115.apk安装包。

这里是循环处理每个applicationVariant，当他们的输出文件名以apk结尾并且buildType是release时，重新设置新的输出文件名，这样就达到了我们批量修改生成的文件名的目的。

#### 项目中的问题

android studio的编译时屏蔽掉lint检查，可以避免由于编译条件太过严格而编译不过的问题：

lintOptions {

abortOnError false

}

如果遇到多个jar包中的某个文件冲突，可以在对应module下的build.gradle文件的android标签下加上如下属性：

packagingOptions {

exclude 'META-INF/NOTICE.txt'// 这里是具体的冲突文件全路径

exclude 'META-INF/LICENSE.txt'

}

##### 依赖版本冲突

依赖冲突是所以依赖管理中最头痛的问题，这常常出现在传递依赖中。Gradle对解决传递依赖提供了两种策略，使用最新版本或者直接导致构建失败。默认的策略是使用最新版本。虽然这样的策略能够解决一些问题，但是还是不够。常见的一种情况是，NoSuchMethond或者ClassNotFound。这时候，你可能需要一些特殊手段，比如排除不想要的传递依赖。

##### 排除传递依赖

排除传递依赖有多种原因，远程仓库中不存在，运行时不需要，或者版本冲突。排除传递依赖的方式有两种：1.直接在configuration中排除 2.在具体的某个dependency中排除

dependencies {

compile 'com.android.support:support-v4:21.0.2'

compile ('com.thoughtworks.xstream:xstream:1.4.7'){

exclude group: 'xmlpull'

exclude module: 'xpp3\_min'

}

}

错误:Execution failed for task ':app:transformClassesWithDexForDebug'.

com.android.build.api.transform.TransformException: java.lang.RuntimeException: com.android.ide.common.process.ProcessException: java.util.concurrent.ExecutionException: com.android.ide.common.process.ProcessException: org.gradle.process.internal.ExecException: Process 'command 'D:\Java\jdk1.8.0\_31\bin\java.exe'' finished with non-zero exit value 1

## 上面都是扯淡，下面才是重点！！！

#### 替换清单文件中的数据

manifestPlaceholders = [APP\_LOGO:"@drawable/coachrun\_icon"]

// 在AndroidManifest.xml里调用

<application

android:name="com.gotobus.main.GotoBusApplication"

android:icon="${APP\_LOGO}"

android:label="@string/app\_name">

#### 运行时修改静态常量

在build.gradle中配置buildConfigField参数，编译后会在..\app\build\generated\source\buildConfig文件夹下会自动生成对应版本对应module的BuildConfig.java。  
BuildConfig就会包含对应版本的配置信息。程序中可以直接引用这些数据。  
需要注意的是BuildConfig定义的变量实在当前module下，如果还有一个基础module需要直接在基础module的gradle下配置。

buildConfigField "String", "APPNAME\_GOTOBUS", '"coachrun"'

//使用举例static{

APPNAME= BuildConfig.APPNAME\_GOTOBUS;

}

#### 运行时修改strings.xml数据

resValue "string", "app\_name", "GotoBus"

// 在Activity里调用

getString(R.string.app\_name) // 输出GotoBus

#### 运行时修改颜色

//使用和字符串相似

resValue "color", "flavor\_color", "#0000ff"

#### 资源文件和manifest的合并

在打包app之前，Android插件会合并main中的代码和构建的代码。当然，依赖项目也可以提供额外的资源，它们也会被合并。你可能需要额外的Android权限针对debug变体。举个例子，你不想在main中申明这个权限，因为这可能导致一些问题，所以你可以添加一个额外的mainfest文件在debug的文件夹中，申明额外的权限。

资源和mainfests的优先级是这样的：



优先级

如果一个资源在main中和在flavor中定义了，那么那个在flavor中的资源有更高的优先级。这样那个在flavor文件夹中的资源将会被打包到apk。而在依赖项目申明的资源总是拥有最低优先级。

#### Gradle编译优化

* 在添加依赖的时候尽量明确版本号，省去gradle查找最新版的时间

不要使用

compile ‘com.facebook.fresco:fresco:latest’

compile ‘com.facebook.fresco:fresco:1.+’，

使用daemon  
构建初始化的很多工作是关于java虚拟机的启动，加载虚拟机环境，加载class文件等，如果这些动作交给一个单独的后台进程去做，那么，第一次初始化之后的修改代码再构建是不是可以节省很多时间呢？答案是肯定的，通过在gradle.properties加入这样一句来开启，如果想让修改全局所有项目都生效，那么修改这个文件~/.gradle/gradle.properties

org.gradle.daemon=true

并行构建模块化项目  
将你的项目拆分成多个子项目并开启并行构建也是一个不错的主意，比如将相对独立的模块拆分成独立的库工程(Library projects)，主工程(Application project)依赖这些库工程，这样的话，开启并行构建才会发挥作用。并行构建开启方式是修改文件gradle.properties，加入如下行：

org.gradle.parallel=true