



## Gradle 基础教程

### 什么是 Gradle?

Gradle 是一种依赖管理工具，基于 Groovy 语言，面向 Java 应用为主，它抛弃了基于 XML 的各种繁琐配置，取而代之的是一种基于 Groovy 的内部领域特定 (DSL) 语言。

### 安装 Gradle

在 [Android Studio 系列教程一 - 下载与安装](#) 中新建项目成功后会下载 Gradle，貌似这个过程不翻墙也是可以下载，但是访问特别慢，建议翻墙下载。那么下载的 Gradle 到什么地方呢？

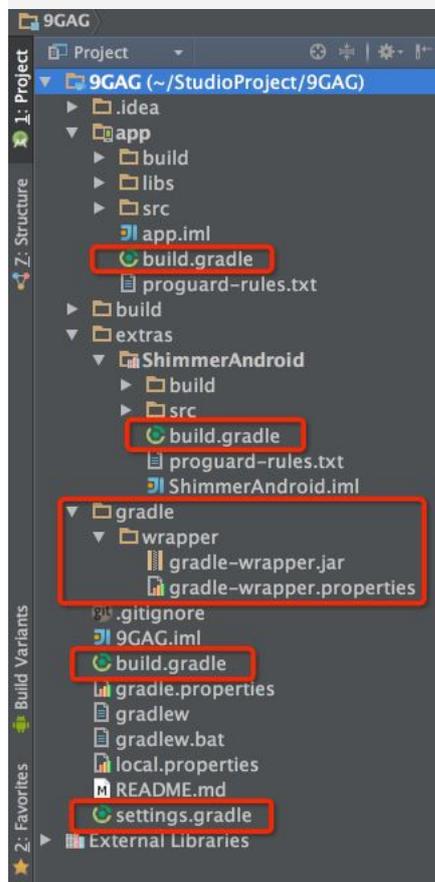
Mac 上会默认下载到 `**/Users/<用户名>/gradle/wrapper/dists**` 目录

Win 平台会默认下载到 `C:\Documents and Settings<用户名>\gradle\wrapper\dists` 目录

你会看到这个目录下有个 `gradle-x.xx-all` 的文件夹，如果下载实在太慢，但是又不想翻墙的话，可以自己手动到 [Gradle 官网](#) 下载对应的版本，然后将下载的 .zip 文件 (也可以解压) 复制到上述的 `gradle-x.xx-all` 文件夹下，不过还是建议让它直接下载的好。

### Gradle 基本概念

下面就以我的开源项目 [9GAG](#) 来详细讲解下和 Gradle 相关的知识，和 Gradle 相关的几个文件一般有如下几个：





红色标记部分从上到下咱们来一步步分析：

#### 1. 9GAG/app/build.gradle

这个文件是 app 文件夹下这个 Module 的 gradle 配置文件，也可以算是整个项目最主要的 gradle 配置文件，我们来看下这个文件的内容：

```
// 声明是 Android 程序

apply plugin: 'com.android.application'

android {

    // 编译 SDK 的版本

    compileSdkVersion 21

    // build tools 的版本

    buildToolsVersion "21.1.1"

    defaultConfig {

        // 应用的包名

        applicationId "me.storm.ninegag"

        minSdkVersion 14

        targetSdkVersion 21

        versionCode 1

        versionName "1.0.0"

        // java 版本

        compileOptions {

            sourceCompatibility JavaVersion.VERSION_1_7

            targetCompatibility JavaVersion.VERSION_1_7

        }

        buildTypes {

            debug {
```



```
        // debug 模式
    }

    release {

        // 是否进行混淆

        minifyEnabled false

        // 混淆文件的位置

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

// 移除 lint 检查的 error

lintOptions {

    abortOnError false
}
}

dependencies {

    // 编译 libs 目录下的所有 jar 包

    compile fileTree(dir: 'libs', include: ['*.jar'])

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

    compile 'com.etsy.android.grid:library:1.0.5'

    compile 'com.alexvasilkov:foldable-layout:1.0.1'

    // 编译 extras 目录下的 ShimmerAndroid 模块

    compile project(':extras:ShimmerAndroid')
```



这里需要说明几点：

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。

`buildToolsVersion` 这个需要你本地安装该版本才行，很多人导入新的第三方库，失败的原因之一是 `build version` 的版本不对，这个可以手动更改成你本地已有的版本或者打开 **SDK Manager** 去下载对应版本。

`applicationId` 代表应用的包名，也是最新的写法，这里就不在多说了。

android 5.0开始默认安装 jdk1.7才能编译，但是由于 mac 系统自带 jdk 的版本是1.6，所以需要手动下载 jdk1.7并配置下，具体可以见我这篇博客 [Mac 下安装和管理 Java](#)

`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 `**/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，由于是开源项目，9GAG 里并未进行混淆，具体混淆的语法也不是本篇博客讨论的范围。最终混淆的结果是这两部分文件共同作用的。

`compile project( ':extras:ShimmerAndroid' )`这一行是因为9GAG中存在其他 Module，不知道 Module 的概念可以看下这篇博客 [Android Studio 系列教程二 - 基本设置与运行](#)，总之你可以理解成 Android Library，由于 Gradle 的普及以及远程仓库的完善，这种依赖渐渐的会变得非常不常见，但是你需要知道有这种依赖的。

以上文件里的内容只是基本配置，其实还有很多自定义部分，如自动打包 debug, release, beta 等环境，签名，多渠道打包等，后续会单独拿出来讲解。

## 2. 9GAG/extras/ShimmerAndroid/build.gradle

每一个 Module 都需要有一个 gradle 配置文件，语法都是一样，唯一不同的是开头声明的是 `apply plugin: 'com.android.library'`

## 3. 9GAG/gradle

这个目录下有个 `wrapper` 文件夹，里面可以看到有两个文件，我们主要看下 `gradle-wrapper.properties` 这个文件的内容：

```
#Thu Dec 18 16:02:24 CST 2014
```

```
distributionBase=GRADLE_USER_HOME
```



```
distributionPath=wrapper/dists
```

```
zipStoreBase=GRADLE_USER_HOME
```

```
zipStorePath=wrapper/dists
```

```
distributionUrl=https://services.gradle.org/distributions/gradle-2.2.1-all.zip
```

可以看到里面声明了 gradle 的目录与下载路径以及当前项目使用的 gradle 版本, 这些默认的路径我们一般不会更改的, 这个文件里指明的 gradle 版本不对也是很多导包不成功的原因之一。

#### 4. 9GAG/build.gradle

这个文件是整个项目的 gradle 基础配置文件, 我们来看看这里面的内容

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

```
buildscript {  
  
    repositories {  
  
        jcenter()  
  
    }  
  
    dependencies {  
  
        classpath 'com.android.tools.build:gradle:1.0.0'  
  
    }  
  
}  
  
allprojects {  
  
    repositories {  
  
        jcenter()  
  
    }  
  
}
```



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 gradle plugin` 的版本，`android studio 1.0` 正式版必须要求支持 `gradle plugin 1.0` 的版本。

#### 5. 9GAG/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`，我们来看看 9GAG 该文件的内容：

```
include ':app', ':extras:ShimmerAndroid'
```

文件中的 `app`、`extras:ShimmerAndroid` 都是 `module`，如果还有其他 `module` 都需要按照如上格式加进去。

### 总结

关于 `gradle` 的基础知识就介绍到这里，接下来会介绍一种我常用的快速方便的编译查看第三方开源项目的方法，如何导入 `Android Studio`，`Gradle` 常用基本命令，多渠道打包配置等。有疑问或者发现错误欢迎大家直接博客留言。

### Gradle 常用命令

上面大家接触了一些命令如 `./gradlew -v`、`./gradlew clean`、`./gradlew build`，这里注意是 `./gradlew`，`./` 代表当前目录，`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 `9GAG/gradle/wrapper/gradle-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 `gradlew wrapper` 的命令代替全局的 `gradle` 命令。

理解了 `gradle wrapper` 的概念，下面一些常用命令也就容易理解了。

```
./gradlew -v 版本号
```

```
./gradlew clean 清除9GAG/app目录下的build文件夹
```

```
./gradlew build 检查依赖并编译打包
```

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，具体在下一篇多渠道打包进一步解释。

```
./gradlew installRelease Release模式打包并安装
```

```
./gradlew uninstallRelease 卸载Release模式包
```



废话不多说，以友盟统计为例，在 AndroidManifest.xml 里面会有这么一段：

```
<meta-data
    android:name="UMENG_CHANNEL"
    android:value="Channel_ID" />
```

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。

- 第一步 在 AndroidManifest.xml 里配置 Placeholder

```
<meta-data
    android:name="UMENG_CHANNEL"
    android:value="{UMENG_CHANNEL_VALUE}" />
```

- 第二步 在 build.gradle 设置 productFlavors

```
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"]
    }
}
```

### 或者批量修改

```
android {

    productFlavors {

        xiaomi {}

        _360 {}

        baidu {}

        wandoujia {}

    }

    productFlavors.all {

        flavor -> flavor.manifestPlaceholders = [UMENG_CHANNEL_VALUE: name]

    }

}
```

很简单清晰有没有？直接执行 `./gradlew assembleRelease` ， 然后就可以静静的喝杯咖啡等待打包完成吧

### assemble 结合 Build Variants 来创建 task

上一篇博客介绍了 `assemble` 这个命令，会结合 `Build Type` 创建自己的 task，如：

```
./gradlew assembleDebug
```

```
./gradlew assembleRelease
```

除此之外 `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：

**\*\*assemble\*\***: 允许直接构建一个 Variant 版本，例如 `assembleFlavor1Debug`。

**\*\*assemble\*\***: 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。

**\*\*assemble\*\***: 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。

## 完整的 gradle 脚本

最后福利大放送，来一份我在项目中使用的完整的 gradle 文件配置：

```
apply plugin: 'com.android.application'

def releaseTime() {
    return new Date().format("yyyy-MM-dd", TimeZone.getTimeZone("UTC"))
}

android {
    compileSdkVersion 21

    buildToolsVersion '21.1.2'

    defaultConfig {
        applicationId "com.boohee.*"
```



```
minSdkVersion 14

targetSdkVersion 21

versionCode 1

versionName "1.0"

// dex 突破65535的限制

multiDexEnabled true

// 默认是 umeng 的渠道

manifestPlaceholders = [UMENG_CHANNEL_VALUE: "umeng"]

}

lintOptions {

    abortOnError false

}

signingConfigs {

    debug {

        // No debug config

    }

    release {

        storeFile file("../yourapp.keystore")

        storePassword "your password"

        keyAlias "your alias"

        keyPassword "your password"

    }

}

buildTypes {
```



```
debug {

    // 显示 Log

    buildConfigField "boolean", "LOG_DEBUG", "true"

    versionNameSuffix "-debug"

    minifyEnabled false

    zipAlignEnabled false

    shrinkResources false

    signingConfig signingConfigs.debug
}

release {

    // 不显示 Log

    buildConfigField "boolean", "LOG_DEBUG", "false"

    minifyEnabled true

    zipAlignEnabled true

    // 移除无用的 resource 文件

    shrinkResources true

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

    signingConfig signingConfigs.release

    applicationVariants.all { variant ->

        variant.outputs.each { output ->

            def outputFile = output.outputFile

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

                // 输出 apk 名称为 boohee_v1.0_2015-01-15_wandoujia.apk
```



```
        def fileName =
"boohee_v${defaultConfig.versionName}_${releaseTime()}_${variant.productFlavors
[0].name}.apk"

        output.outputFile = new File(outputFile.parent, fileName)
    }
}
}
}

// 友盟多渠道打包
productFlavors {
    wandoujia {}
    _360 {}
    baidu {}
    xiaomi {}
    tencent {}
    taobao {}
    ...
}

productFlavors.all { flavor ->
    flavor.manifestPlaceholders = [UMENG_CHANNEL_VALUE: name]
}
}

dependencies {
```



```
compile fileTree(dir: 'libs', include: ['*.jar'])  
  
compile 'com.android.support:support-v4:21.0.3'  
  
compile 'com.jakewharton:butterknife:6.0.0'  
  
...
```

大家有问题或疑问、建议欢迎博客留言，Android Studio 的教程暂且到这里结束了，相信大家基本的都已会使用了，还有其他技巧与操作靠大家自己摸索了，之后有时间也会在博客上整理下一些 Tips 之类的，欢迎大家关注。

Action	Mac OSX	Win/Linux
注释代码(//)	Cmd + /	Ctrl + /
注释代码(/**)	Cmd + Option + /	Ctrl + Shift + /
格式化代码	Cmd + Option + L	Ctrl + Alt + L
清除无效包引用	Option + Control + O	Alt + Ctrl + O
查找	Cmd + F	Ctrl + F
查找+替换	Cmd + R	Ctrl + R
上下移动代码	Option + Shift + Up/Down	Alt + Shift + Up/Down
删除行	Cmd + Delete	Ctrl + Y
扩大缩小选中范围	Option + Up/Down	Ctrl + W/Ctrl + Shift + W
快捷生成结构体	Cmd + Option + T	Ctrl + Alt + T
快捷覆写方法	Cmd + O	Ctrl + O



快捷定位到行首/尾	Cmd + Left/Right	Ctrl + Left/Right
折叠展开代码块	Cmd + Plus,Minus	Ctrl + Plus/Minus
折叠展开全部代码块	Cmd + Shift + Plus,Minus	Ctrl + Shift + Plus,Minus
文件方法结构	Cmd + F12	Ctrl + F12
查找调用的位置	Ctrl + Option + H	Ctrl + Alt + H
大小写转换	Cmd + Shift + U	Ctrl + Shift + U