这一次，彻底了解 Gradle 吧！

前行的乌龟 鸿洋 今天

本文作者

作者：**前行的乌龟**
链接：
https://juejin.im/post/6882178101191639053
本文由作者授权发布。

Gradle 内容真是超乎寻常的多，在写本文之前我以为有个万把字就差不多了，但随着越看越多，我发现想写的话一本书都是可以写出来的 〒▽〒

因为内容多，我只能拆成多篇文章了，希望能写全吧。

我写文章都是喜欢以小白为出发点的，希望对那些一点都不了解 Gralde 的朋友能所有帮助，也希望能大大缩短大家学习 Gralde 的时间成本。Gradle 这东西对于一般人真的是难，非常难理解。

相关的技术文章都是18年后才开始涌现出来的，之前的文章(尤其是15年那会AS出现时的文章)真的是非常非常少，可见难度之大。我想也只有之前精通后端，熟悉 Ant，Maven 构建工具，转到 Android 的那些高手们才能一上来就玩转 Gradle 吧 (ว´ω`)ว*ฉ

1 Gradle 学习指南

1. Gradle 基本学习路线

1. 先了解什么是 Gradle、Groovy
2. 熟悉 Groovy 语法
3. 熟悉什么是 plugin 插件、task 任务
4. 熟悉 Gradle 核心对象：Gradle、Setting、Project，Gradle 构建流程、生命周期、及其 hook 勾子函数
5. 熟悉 Android 项目构建，application 这个插件
6. 了解自定义 task，自定义 plugin 并上传 maven 以及使用
7. 各种 自定义 plugin 花样

这是基本的学习思路，结合我提供的学习资料，我想至少可以给大家减少很多寻找资料、理清脉络、反复折腾的时间，下面贴一下具体学习指南

2. Gradle 学习资料食用指南

Gradle 很复杂、学习难度很大的，一遍基本是不够的，请大家耐心反复看几遍 <(￣ˇ￣)/

1. 首先还是希望大家能先阅读下本文，先对 Gradle 有些基本理解再看后面，尤其是一点 Gradle 基础都没有的同学，我是真的建议大家先把我这篇文章看完，我写文章从来都是从小白出发

2. 来自Gradle开发团队的Gradle入门教程
https://www.bilibili.com/video/BV1DE411Z7nt

优先推荐大家看看这个，概念解释清晰，言简意赅、逻辑条理 Nice，会帮你理解 Gradle 的全貌，虽然具体内容不是很多。但是带着官方的理解再去看其他资料能减少很多概念理解上的歧义，能帮助提升大家的学习效率

3. Gradle系统详解
https://www.bilibili.com/video/BV1J4411C7Q5

这个4个来小时，尝试把 Gradle 都讲一遍，但是效果不怎么好，推荐大家过一遍，增加了解.

4. gradle快速入门 | 自定义编写Gradle插件

这2篇都是讲 plugin 插件、task 的，大家看完之后对这2块会有比较深的了解
https://www.bilibili.com/video/BV1dK411K7Pg?p=8
https://www.bilibili.com/video/BV15K411T7xb

5. 用Artifactory和Retryofit助力Gradle提高开发效率
https://www.bilibili.com/video/BV15K411T7xb

这个是讲解 Gradle 自动化打包、上传、发布的，适合有需求的朋友看

6. 掘金小册-Mastering Gradle --> 这本掘金小册真的是新人杀手、劝退指南。文章虽然内容混乱，质量还是不错的，推荐大家对 Gradle 有系统了解之后再来看.

7. Gradle--官方文档
http://www.groovy-lang.org/api.html

管方文档最后还是推荐大家看看的，结合 Google 浏览器的自动翻译插件，还是能看的，完全没问题，剩下的就是 Gradle 插件的应用了，这块大家自行查阅 掘金--android--gradle 板块内的文章即可。

这一套下来，基本上 Gradle 对于大家来说就没什么大问题了，剩下的就是在项目中实际应用了。再有掘金上有黑科技文章出来大家也不会看不懂，不会用了。总体耗时会长一些，但是真的可以一次学习，终身受益，不用反复折腾了

2 理解 Gradle、Groovy

对于拦路虎、大 Boss，理论先于实际。手握理论的浮尘，方能遇坑过坑、遇水搭桥 (๑•̀ㅂ•́)و✧

1. 什么是构建工具

简单的说就是自动化的编译、打包程序。

我们来回忆一下，入门 java 那会，大家都写过 Hello Wrold！吧。然后老师让我们干啥，javac 编译，java 运行。在这里编译需要我们手动执行一次 javac，大家想过没有，要是有100个文件呢？那我们就得手动 100次 javac 编译指令。

到这里大家都会想到自动化吧，是的，自动化编译工具就是最早的构建工具了。然后我们拓展其功能，比如说：

- 100个文件，编译后我要分10个文件夹保存
- 哎呀，文件夹不好使了，别人要我提供 .jar 文件
- 我去，产品加功能了，要加入 C++ 文件进来，C、java 文件要一起编译
- 产品要有展示图片，还要有声音，多媒体资源也要加进来
- 业务拓展了好几个渠道，每一个渠道都要提供一个定制化的 .jar 出来
- 业务拓展了，要全平台了，win、android、ios 都要支持

上面都是我臆想的，不过我觉得发展的历程大同小异。随着需求叠加、平台扩展，对代码最终产品也是有越来越多的要求。

jar/aar/exe 这些打包时有太多的不一样，我们是人不是机器，不可能记得住的这些差异不同。那就必须依靠自动化技术、工具，要能支持平台、需求等方面的差异、能添加自定义任务的、专门的用来打包生成最终产品的一个程序、工具，这个就是构建工具。

构建工具本质上还是一段代码程序。

2. Gradle 也是一种构建工具

Android 项目这么多东西，既有我们自己写的 java、kotlin、C++、Dart 代码，也有系统自己的 java、C++ 代码，还有引入的第三方代码，还有图片、音乐、视频文件，这么多代码、资源打包成 APK 文件肯定要有一个规范，干这个活的就是我们熟悉的 gradle 了。

APK 文件我们解压可以看到好多文件和文件夹，具体不展开了

不用把 Gradle 想的太难了，Gradle 就是帮我们打包生成 apk 的一个程序。

难点的在于很灵活，我们可以在其中配置、声明参数、执行自己写的脚本、甚至导入自己的写的插件，来完成我们自定义的额外的任务。但是不要本末倒置，Gradle 就是帮我们打包 APK 的一个工具罢了。

下面3段话大家理解下，我觉得说的都挺到位的，看过后面还可以翻回来看这3句话，算是对 Gradle 的总结性文字了，很好~

> Gradle 是通用构建、打包程序，可以支持 java、web、android 等项目，具体到你的平台怎么打包，还得看你引入的什么插件，插件会具体按照我们平台的要求去编译、打包。比如我引入的：apply plugin: 'com.android.application'，我导入的是 android 编译打包插件，那么最终会生成 APK 文件，就是这样。我引入的：apply plugin: 'com.android.library' android lib 库文件插件，那么最终会生成 aar 文件

> Gradle中，每一个待编译的工程都叫一个Project。每一个Project在构建的时候都包含一系列的Task。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。一个Project到底包含多少个Task，其实是由编译脚本指定的插件决定。插件是什么呢？插件就是用来定义Task，并具体执行这些Task的东西

> 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。Gradle中每一个待编译的工程都是一个Project，一个具体的编译过程是由一个一个的Task来定义和执行的。一个Project到底包含多少个Task，其实是由编译脚本指定的插件决定。插件是什么呢？插件就是用来定义Task，并具体执行这些Task的东西

3. Gradle 是个程序、Groovy 是特定领域 DSL 语言

- Gradle 是运行在 JVM 实例上的一个程序，内部使用 Groovy 语言
- Groovy 是一种 JVM 上的脚本语言，基于 java 扩展的动态语言

Gradle 简单来说就是在运行在 JVM 上的一个程序罢了，虽然其使用的是 Groovy 这种脚本语言，但是 Gradle 会把 .gradle Groovy 脚本编译成 .class java字节码文件在 JVM 上运行，最终还是 java 这套东西。

Android 项目里 settings.gradle、诸多build.gradle 脚本都会编译成对应的 java 类：

Setting、Project 再去运行，引入的插件也是会编译成对应的 java 对象再执行构建任务。

Gradle 内部是一个个编译、打包、处理资源的函数或者插件(函数库)，可以说 Gradle 其实就是 API 集合，和我们日常使用的 Okhttp 框架没什么区别，里面都是一个个 API，区别是干的活不同罢了。

打开 Gradle 文件目录看看，核心的 bin 文件就一个 gradle 脚本，这个脚本就是 Gradle 核心执行逻辑了，他会启动一个 JVM 实例去加载 lib 中的各种函数去构建项目，这么看 gradle 其实很简单、不难理

解。

红框里的是 Gradle 自带的内置插件，apply plugin: 'com.android.library'、apply plugin: 'com.android.application' 这些都是 gradle 自带的内置插件。

3 Gradle JVM 进程

Gradle 构建工具在不同场景下会分别使用3个 JVM 进程：

- client
- Daemon
- wrapper

1. client 进程

client 进程是个轻量级进程，每次构建开始都会创建这个进程，构建结束会销毁这个进程。

client 进程的任务是查找并和 Daemon 进程通信：

- Daemon 进程没启动，client 进程会启动一个新的 Daemon 进程
- Daemon 进程已经存在了，client 进程就给 Daemon 进程传递本次构建相关的参数和任务，然后接收 Daemon 进程发送过来的日志

像 gradle.properties 里面设置的参数，全局 init.gradle 初始化脚本的任务这些都需要 client 进程传递给 Daemon 进程。

2. Daemon 进程

Daemon 进程负责具体的构建任务。我们使用 AS 打包 APK 这依靠的不是 AS 这个 IDEA 开发工具，而是 Gradle 构建工具自己启动的、专门的一个负责构建任务的进程：Daemon。

Daemon 进程不依赖 AS 而是独立存在，是一个守护进程，构建结束 Daemon 进程也不会销毁，而是会休眠，等待下一次构建，这样做是为了节省系统资源，加快构建速度，Daemon 进程会缓存插件、依赖等资源。

必须注意：每一个 Gradle 版本都会对应一个 Daemon 进程，机器内若是运行过多个版本的 Gradle，那么机器内就会存在多个 Daemon 进程，AS 开发 android 项目，我推荐使用 Gradle 本地文件，不依靠每个 android 项目中 wrapper 管理 gradle 版本，具体后面会说明。

从性能上讲：

- Gradle 在 JVM 上运行，会使用一些支持库，这些库都需要初始化时间，一个长期存在的后台进程有利于节省编译时间
- daemon 进程会跨构建缓存一些插件、库等缓存数据，这样对加快构建速度的确非常有意义

gradle --status 命令可以查看已启动的 daemon 进程情况：

```
➜ ~ jps
39554 KotlinCompileDaemon
39509 GradleDaemon
39608
39675 Jps
➜ ~ gradle --status
  PID STATUS  INFO
 39509 IDLE   6.6.1

// INFO 是 gradle 版本号
// Kotlin 语言编写的 Gradle 脚本需要一个新的 daemon 进程出来
```

若是机器内已经启动了多个 Daemon 进程也不要紧，自己手动杀进程就是了。

Daemon 进程在以下情况时会失效，需要启动新的 Daemon 进程，判断 Daemon 进程是否符合要求是上面说的 client 进程的任务：

- 修改 JVM 配置会造成启动新的构建进程
- Gradle 将杀死任何闲置了3小时或更长时间的守护程序
- 一些环境变量的变化，如语言、keystore、keyStorePassword、keyStoreType 这些变化都会造成旧有的守护进程失效

即便是同一个版本的 Gradle，也会因为 VM 配置不同而存在多个相同 Gradle 版本的 Daemon 进程。比如同时启动好几个项目，项目之间使用的 Gradle 版本相同，但是 VM 使用的不同配置

3. wrapper 进程

wrapper 进程啥也不干，不参与项目构建，唯一任务就是负责下载管理 Gradle 版本。我们导入 Gradle 项目进来，client 进程发现所需版本的 Gradle 本机没有，那么就会启动 wrapper 进程，根据 gradle.properties 里面的参数自行去 gradle-wrapper.jar 里面的下载程序去下载 Gradle 文件，完事 wrapper 进程会关闭。

4 Gradle 安装

上文说Gradle 运行在 JVM 之上, 因此需要 JDK 环境。

1. 下载 Gradle 版本

从 Gradle 官方上下载，地址：Gradle 官网下载地址，选择 .all 的包下载，我下的是 6.6.1，尽量选择较新的版本
https://services.gradle.org/distributions/

2. 配置项目根目录 build.gradle 脚本文件 Gradle 工具版本号

```
buildscript {

    repositories {
        google()
        jcenter()
    }
    dependencies {
        ...
        classpath 'com.android.tools.build:gradle:4.0.1'
        ...
    }
}
```

这里 Gradle 工具的版本号要跟着 AS 的版本号走，AS 是哪个版本，这里就写哪个版本。
Gradle 工具中的 API 是给 AS 用的，自然要跟着 AS 的版本变迁。

当然这也会对 Gradle 构建工具版本有要求：

- 第一，大家进来使用比较新的版本号
- 第二，若是 Gradle 版本太低，编译时会有提示的，告诉你最低 Gradle 构建工具版本是多少

3. 使用本地 Gradle 文件编译项目

Gradle 拥有良好的兼容性，为了在没有 Gradle 环境的机器上也能顺利使用 Gradle 构建项目，AS 新创建的项目默认会在根目录下添加 wrapper 配置

其中 gradle-wrapper.properties 文件中提供了该项目使用的 Gradle 构建工具远程下载地址，这里会对应一个具体的版本号，IDE 开发工具默认会根据这个路径去下载 Gradle 给该项目使用

```
distributionUrl=https\://services.gradle.org/distributions/gradle-6.1.1-all.zip
```

这样就会产生一个问题：

每个项目单独管理自己的 gradle，很可能会造成机器上同时存在多个版本的 Gradle，进而存在多个版本的 Daemon 进程，这会造成机器资源吃紧，即便关闭 AS 开发工具也没用，只能重启机器才会好转。

> 所以这里我推荐，尤其是给使用 AS 的朋友推荐：在本地创建 Gradle 环境，统一管理 Gradle 构建工具，避免出现多版本同时运行的问题。AS 本身就很吃内存了，每一个 Daemon 构建进程起码都是 512M 内存起步的，多来几个 Daemon 进程，我这 8G 的 MAC 真的搂不住

1. 打开 AS 中 Gradle 配置：

gradle-wrapper.properties -- 使用 wrapper 也就是 AS 来管理 Gradle
Specifiled location -- 使用本地文件，也就是我们自己管理 Gradle

2. 在本地解压 Gradle 压缩包，记住路径，下面配 path 需要

这样配置后，AS 会忽略 gradle-wrapper.properties 文件

4. 配置 path

```
export GRADLE_HOME=/Users/zbzbgo/gradle/gradle-6.6.1
export PATH=${PATH}:/Users/zbzbgo/gradle/gradle-6.6.1/bin

----------------官方写法如下--------------------------------

export GRADLE_HOME=/Users/zbzbgo/gradle/gradle-6.6.1
export PATH=$PATH:$GRADLE_HOME/bin
```

5. 测试 Gradle 安装是否成功

运行 gradle --version，出现版本号则 Gradle 配置成功

6. 执行一次 Gradle 命令

学习新语言我们都喜欢来一次 hello world，这里我们也来一次。

随便创建一个文件夹，在其中创建一个文件，以.gradle结尾，使用 text 编辑器打开，输入：

```
println("hello world!")
```

然后 gradle xxx.gradle 执行该文件

OK，成功，大家体验一下，groovy 是种语言，gradle 是种构建工具，可以编译 .gradle 文件。

5 Gradle Task

先剧透一下

Task 是 Gradle 执行的基本单元，为什么这么说？实际上根据 Gradle 构建项目的流程，是先把所有的 .gradle 脚本执行一遍，编译生成对应的 Gradle、Setting、Project 对象，然后根据我们在构建脚本中设置的构建配置，生成一张 Task 组成的：有向无环图，先来看看这张图

最终，Gradle 会按照图上一个个 Task 的关联顺序挨个执行，每一个 Task 都完成一个特定功能，按照顺序把 Task 图执行一遍，整个项目的构建任务就完成了 ┗| ` O′|┛ 嗷~~

什么是 Task

> Gradle中，每一个待编译的工程都叫一个Project。每一个Project在构建的时候都包含一系列的Task。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。一个Project到底包含多少个Task，其实是由编译脚本指定的插件决定。插件是什么呢？插件就是用来定义Task，并具体执行这些Task的东西

Task 我们可以看成一个个任务，这个任务可以是编译 java 代码、编译 C/C++ 代码、编译资源文件生成对应的 R 文件、打包生成 jar、aar 库文件、签名、混淆、图片压缩、打包生成 APK 文件、包/资源发布等。不同类型的项目 Task 任务种类不同，Gradle 现在可以构建 java、android、web、lib 项目。

下图中这些都是一个个 Task 任务，每一个 Task 任务都是为了实现某个目的，可以理解为一个步奏，最终一个个步奏按序执行就完成了整个项目构建。

Task 是完成一类任务，实际上对应的也是一个对象。而 Task 是由无数个 Action 组成的，Action 代表的是一个个函数、方法，每个 Task 都是一堆 Action 按序组成的执行图，就好像我们在 Class 的 main 函数中按照逻辑调用一系列方法一样。

创建 Task

```
task hello{
  println "hello world"
}

task(hello2){
  println "hello world2"
}
```

```
task ('hello3'){
  println "hello world3"
}
```

运行 Task

命令行执行 gradle hello

Task 的 action

上面说过 Task 内部 action 也不是唯一的，而是一个集合，我们可以往里面添加各种 action，要注意 action 之间有前后执行顺序的，这个是规定好了的。这些 action 接收都是闭包，看下面 Task 中的声明

```
//在 Action 队列头部添加 Action
Task doFirst(Action<? super Task> action);
Task doFirst(Closure action);

//在 Action 队列尾部添加 Action
Task doLast(Action<? super Task> action);
Task doLast(Closure action);

//删除所有的 Action
Task deleteAllActions();
```

doFirst{...}、doLast{...} 接受的都是闭包

- doFirst 添加的 action 最先执行
- task 自身的 action 在中间执行，这个无法在 task 外部添加，可以在自定义 task 时写
- doLast 添加的 action 最后执行

另外 task 也可以向对象那样操作，task 里面也可以写代码，比如打印 AA，但是这些代码只能在配置阶段执行，而 task 的 action 都是在运行阶段执行的。配置阶段和执行阶段不了解的看下面内容

```
task speak{
  println("This is AA!")
  doFirst {
    println("This is doFirst!")
  }
  doLast {
    println("This is doLast!")
  }
}

speak.doFirst {
  println("This is doFirst!")
 }
```

action 可以添加多个的，按照添加顺序执行，比如 doLast 就可以添加多个进来

```
task speak{
  println("This is AA!")
  doFirst {
    println("This is doFirst!")
  }
  doLast {
    println("This is doLast1...!")
  }
}

speak.configure {
  doLast {
    println 'his is doLast2...'
  }
}
speak.doLast{
    println 'his is doLast3...'
}
```


动态创建 Task

```
4.times { counter ->
  task "task$counter" {
    doLast {
      println "I'm task number $counter"
    }
  }
}
```

Task 之间共享数据

2个 Task 之间使用、操作同一个数据。早先在 .gradle 脚本中直接声明变量，Task 直接使用该变量的方式是不行的、会报错，需要借助 ext 全局变量。但在 6.6.1 我发现可以直接用变量了，应该是官方优化了

```
ext {
   name = "AAA"
}

def age = 18

task s1 {
   doLast {
      age = 12
      rootProject.ext.name = "BBB"
      println("This is s1...")
   }
}

task s2 {
   doLast {
      println("age --> " + age)
      println("name --> " + rootProject.ext.name)
      println("This is s2...")
   }
}
```

Task 依赖

Task 依赖是指我们可以指定 Task 之间的执行顺序，重点理解 dependsOn 就行了

- A.dependsOn B --> 执行A的时候先执行B
- A.mustRunAfter B --> 同时执行 A/B，先执行B再执行A，若执行关系不成立报错
- A.shouldRunAfter B --> 同 mustRunAfter，但是执行关系不成立不会报错

1. dependsOn -->

```
task s1{
  doLast {
    println("This is s1...")
  }
}

task s2{
  doLast {
    println("This is s2...")
  }
}

s1.dependsOn s2

-----------或者----------------

task s2{
  dependsOn s1
  doLast {
    println("This is s2...")
  }
}

-----------dependsOn 还没声明出来的 task 要加 ""----------------

task s1{
  dependsOn "s2"
  doLast {
    println("This is s2...")
  }
}
```


2. mustRunAfter -->

```
task s1{
  doLast {
    println("This is s1...")
  }
}

task s2{
  doLast {
    println("This is s2...")
  }
}

s1.mustRunAfter s2
```

自定义 Task

Gradle 中 Task 都是继承自 DefaultTask，我们自定义 Task 也需要继承这个类，重点是写自己需要的方法，然后加上@TaskAction注解表示这个方法是 Task 中的 action，可以加多个，按照倒序执行

```
class MyTask extends DefaultTask {

   String message = "mytask..."

   @TaskAction
   def ss1() {
      println("This is MyTask --> action1!")
   }

   @TaskAction
   def ss2() {
      println("This is MyTask --> action2!")
   }

}

task speak(type: MyTask) {
   println("This is AA!")
   doFirst {
      println("This is doFirst!")
   }
   doLast {
      println("This is doLast!")
   }
}
```

系统默认 Task

gradle 默认提供了很多 task 给我们使用，比如 copy、delete

1. copy

copy 复制文件

```
task speak (type: Copy) {
   ...
}
```

```
//数据源目录，多个目录
public AbstractCopyTask from(Object... sourcePaths)

//目标目录，单一
public AbstractCopyTask into(Object destDir)

//过滤文件 包含
public AbstractCopyTask include(String... includes)

//过滤文件 排除
public AbstractCopyTask exclude(String... excludes)

//重新命名，老名字 新名字
public AbstractCopyTask rename(String sourceRegEx, String replaceWith)

//删除文件 Project 接口
boolean delete(Object... paths);
```

复制图片：多个数据源 -->

```
复制图片：多个数据源 -->

task copyImage(type: Copy) {
  from 'C:\\Users\\yiba_zyj\\Desktop\\gradle\\copy' ,
     'C:\\Users\\yiba_zyj\\Desktop\\gradle\\copy'
  into 'C:\\Users\\yiba_zyj\\Desktop'
}
```

复制文件：过滤文件，重命名 -->

```
复制文件：过滤文件，重命名 -->

task copyImage(type: Copy) {
  from 'C:\\Users\\yiba_zyj\\Desktop\\gradle\\copy'
  into 'C:\\Users\\yiba_zyj\\Desktop'
  include "*.jpg"
  exclude "image1.jpg"
  rename("image2.jpg","123.jpg")
}
```

2. Delete

删除文件

```
删除桌面上的文件 -->

task deleteFile(type: Delete) {
  //删除系统桌面 delete
```

```
    delete "C:\\Users\\yiba_zyj\\Desktop\\gradle\\delete"
}
```

设置默认 Task 任务

设置这个的意思的是指，脚本中我们不调用该 task，设置的 task 也会执行

```
defaultTasks 'clean', 'run'

task clean {
    doLast {
        println 'Default Cleaning!'
    }
}

task run {
    doLast {
        println 'Default Running!'
    }
}

task other {
    doLast {
        println "I'm not a default task!"
    }
}
```

```
> gradle -q

Default Cleaning!
Default Running!
```

Task 中使用外部依赖

代码来自官网，buildscript{...} 引入远程仓库和依赖 path，import 导入进来就可以用了

```
import org.apache.commons.codec.binary.Base64

buildscript {
    // 导入仓库
    repositories {
        mavenCentral()
    }
    // 添加具体依赖
    dependencies {
        classpath group: 'commons-codec', name: 'commons-codec', version: '1.2'
    }
}

task encode {
    doLast {
        def byte[] encodedString = new Base64().encode('hello world\n'.getBytes())
        println new String(encodedString)
    }
}
```

6 Gradle Plugin

理解什么是插件

我们写项目要使用大量的第三方代码，Gradle 作为构建工具，自然要拥有管理第三方代码的能力，要不怎么打包生成最终输出物。因为这些额外的代码要打入最终的生成物中，所以管理第三方代码的功能 Gradle 是必须要有的.

这些第三方代码有些是参与业务代码的，有些则是参与项目构建的：

- 参与项目构建的第三方代码叫 --> 插件
- 参与代码逻辑的第三方代码叫 --> 依赖

不管是插件，还是依赖，本质都是一堆类、函数，就是 API，区别是使用的地方不同。

> 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。Gradle中每一个待编译的工程都是一个Project，一个具体的编译过程是由一个一个的Task来定义和执行的。一个Project到底包含多少个Task，其实是由编译脚本指定的插件决定。插件是什么呢？插件就是用来定义Task，并具体执行这些Task的东西

插件的作用

Gradle 本身是一个通用的构建系统, 它并不知道你要编译的项目或代码是 Java 还是 C。Java 代码需要 javac 把 .java 编译为 .class，而 C 代码需要 gcc 把 .c 编译为 .o

在介绍 Task 的部分我们说过整个构建过程就是由一个个 Task 任务构成的，一个项目的构建包括很多步奏：代码编译、资源编译、依赖库打包等操作。

基于提取公共、抽象特异的模板思路，Gradle 作为通用项目构建工具，Gradle 封装的是项目构建的公共部分，而针对每种项目的特点和不同，就由每种项目对应的构建插件来接过差异部分的构建。

这些差异要是让 coder 我们自己来做，谁也记不住这么多不同不是，谁没事去背这个，因此插件就诞生了。

apply plugin: 'com.android.application' 是 Android 项目的构建插件。

apply plugin: 'com.android.library' 是 Android 依赖项目的构建插件。

这些插件封装了对应项目的整个构建流程，具体就是说这些插件内部已经定义好了Task 有向无环图。

我们只需要在 .gradle 构建脚本中引入插件, 根据项目情况配置几个属性, 即可实现项目的构建。

当然为了完成某个功能，我们也可以把这些代码写成插件存入远程仓库中供大家使用~

Gradle 内置插件

Gradle 内置了很多插件, 比如 java 插件, build-scan 插件, 引入这种插件我们直接通过 Gradle 内置函数就可以, 不用指定 id 和 version

比如这样的就是内置插件：

```
repositories {
  google()
  jcenter()
}
```

google()、jcenter() 明显就是一个函数嘛 （○｀3′○） 我们有看到插件的 group:name:version 了吗，明显没有呀，那为什么我们还能引入这个插件呢？

答案就是 Gradle 已经帮我们写好这块啦 (>▿<) 我们点击 google()、jcenter() 会链接到一个类：RepositoryHandler。内置的意思就是 Gradle 已经写好啦，我们拿来直接用就好了。

gradle 内置插件可以看到都是插件仓库，当然仓库本身也是一种插件就是了，理解起来有点绕

导入插件

插入插件，我们先要导入仓库，也就声明从哪些仓库查找插件和远程依赖，然后再导入插件。

只要我们在根项目的 build.gradle 构建脚本中声明导入的插件，那么所有子项目就都可以使用该插件了

根项目 build.gradle 导入插件

```
buildscript {

    repositories {
        google()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:3.5.0'
    }
}
```

```
allprojects {
  repositories {
    google()
    jcenter()
  }
}
```

子项目使用插件

```
app build.gradle -->

apply plugin: 'com.android.application'
apply plugin: 'kotlin-android'
apply plugin: 'kotlin-android-extensions'

android {
  ....

  defaultConfig {
    ....
  }
  buildTypes {
    ....
  }
}

dependencies {
  ....
}
```

其中几个 {...} 闭包解释下：

- buildscript {...} --> 声明仓库、添加插件专用闭包
- repositories {...} --> 设置远程仓库
- dependencies {...} --> 添加插件、远程依赖
- apply --> 使用插件，插件虽然导入进来了，但是子项目用不用就是个事了，要是不用的话，是不需要打包进来的，所以要主动声明下

配置插件属性

> 这个是重点，务必要理解 （○｀3′○）

前文说过插件是参与项目构建的第三方代码集合，这些代码不光会参与整个项目构建，同样也需要用户输入一些配置参数，参数配置也是用闭包的方式传入。

插件内部会有对应的方法，接收这些在 build.gradle 脚本中配置的参数。给插件传递配置参数的闭包叫 DSL 代码段，最经典的场景就是 android {...} 了

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

android {
  compileSdkVersion(28)
```

```
    defaultConfig {
        ....
    }
}
```

android {...} 这个 DSL 可不是 Gradle 自身的配置，而是来源于 com.android.application 这个插件。大家只要明白这个后面再看 build.gradle 就明白多了，大家把 DSL 当做插件的 init 方法好了，本质上就是这么回事。

7 Gradle 构建过程

> 下面我说：Gradle 脚本的执行 --> 是指 gradle 编译那些 .gradle 脚本文件，生成对应的对象和 Task 任务

Gradle 的构建过程是通用的，任何由 Gradle 构建的项目都遵循这个过程。

Gradle 构建分为三个阶段，每个阶段都有自己的职责，每个阶段都完成一部分任务，前一阶段的成果是下一阶段继续执行的前提：

- Initialization --> 初始化阶段。按顺序执行 init.gradle -> settings.gradle 脚本，生成 Gradle、Setting、Project 对象
- Configuration --> 编译阶段，也叫配置阶段。按顺序执行 root build.gradle -> 子项目 build.gradle 脚本，生成 Task 执行流程图
- Execution --> 执行阶段。按照 Task 执行图顺序运行每一个 Task，完成一个个步骤，生成最终 APK 文件

看图：

整个构建过程，官方在一些节点都设置有 hook 钩子函数，以便我们在构建阶段添加自己的逻辑进来，影响构建过程。hook 钩子函数可以理解为监听函数，另外也可以正儿八经的设置 Linsener 监听函数。

Initialization 阶段

Initialization 是初始化阶段，一上来会马上把全局的 Gradle 内置对象 new 出来，Gradle 对象中的参数都是本次 gradle 构建的全局属性，通过 Gradle 对象可以干很多事。

- 比如可以获取 Gradle Home、Gradle User Home 目录
- 添加全局监听
- 给所有项目添加设置、依赖等

Initialization 阶段会按照先后顺序运行2个 Groovy 脚本：

- Init Script：创建内置对象 Gradle
- Setting Script：创建内置对象 Setting、每个 module 对应的 Project 对象

Configuration 阶段

Initialization 阶段后就是 Configuration 阶段了，Configuration 阶段会执行所有的 build.gradle 脚本，先执行根目录即根项目的构建脚本，再根据子项目之间的依赖关系，挨个执行子项目的构建脚本。

Gradle 中有根项目和子项目之分，不管是什么项目在 Gradle 中都对应一个 Project 对象。根项目只有一个，而子项目可以有多个，android 中即便是 app module 我们经常说的壳工程，其实也是一个子项

目。

Gradle 默认的根项目是空的，没有内容的，跟项目只有 .gradle 有实际意义，其他都没用，不要在意。

Gradle 中每一个项目都必须有一个 .gradle 构建脚本，Configuration 阶段会执行所有项目的构建脚本，从跟项目开始一直到把所有参与本次构建的子项目构建脚本都执行完，在这个过程中，会根据 .gradle 构建脚本内容，创建对应的 Project 对象，在后面的环节，1个 Project 对象就对应着1个项目啦，rootProject 表示根项目，Project 表示子项目。

> .gradle 脚本里的 DSL 配置块都是 Project 对象的方法而已

然后根据脚本中的配置，生成 Configuration 阶段的最终产物：Task 有向无环图，给下一阶段执行。Configuration 阶段的目的就是根据脚本配置计算出整个构建过程需要的逻辑和流程，可以理解为动态生成代码的过程。有了动态生成的代码，后面才好有东西执行不是 （^ - ^）

Execution 阶段

Execution 阶段没啥说的了，就是拿到 Configuration 阶段计算出来的 task 执行图，按照顺序一个个跑这些 task 就能完成构建了。

这是 Android 构建流程，其中每一个环节都可以认为是一个 Task(实际每个环节都有多个Task)

大家仔细体会上面这张图，不难的，挨个环节看，整个构建流程就是由这一个个细分的步骤组成的，至于有哪些步骤，哪些先开始，哪些最后执行，这就要看你在项目中导入的是什么插件了。

前文说过，Gradle 是通用构建工具，用于制定规则，详细的每种项目该怎么构建。过程是什么样的，插件负责的就是这些具体的内容、流程。

最后推荐一下我做的网站，玩Android: wanandroid.com ，包含详尽的知识体系、好用的工具，还有本公众号文章合集，欢迎体验和收藏！

推荐阅读：

看 Google 是如何设计的 View 机制？

对标 RxJava，你好 Flow，协程实战篇

官方也无力回天？“SharedPreferences 存在什么问题？”

扫一扫 关注我的公众号
如果你想要跟大家分享你的文章，欢迎投稿~

┏(^ 0 ^)┛明天见！