|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 分类号 |  | |  | | | 密 级 | |  |
| U D C |  | |  | | | 编 号 | | 0001 |
|  | | | | | | | | |
| **读 书 笔 记** | | | | | | | | |
|  | | | | | | | | |
| Gradle | | | | | | | | |
|  | |  | |  |  | |  | |
|  | |  |  | |
|  | |  |  | |
|  | |  |  | |
|  | |  |  | |
|  | |  |  | |
|  | | | | | | | | |
| By key.guan  May, 2017 | | | | | | | | |

# 前 言

Gradle Recipes for Android （英文版） 介绍了Gradle在Android开发过程中的常见运用。本文即是对该书的读后笔记,重点以[3]为参考撰写此文。

关于本书本书包含 Gradle 作为 Android 项目的构建系统时相关的实用技巧。Gradle 是 Java 世界中最著名的构建工具之一，并正在逐渐扩展到其他语言，如 C++。Google 的 Android 团队在2013年春天正式将Gradle作为首选的构建系统，之后其使用人群也在逐渐稳固增长。  
 因为 Gradle 来自 Groovy 生态系统，很多 Android 开发者可能不熟悉。然而对于 Java 开发人员来说学习 Groovy 是非常容易的。本书的目的是通过示例帮你使用 Gradle 完成Android 应用程序中最常见的一些构建任务。  
 所有的示例使用的 Android SDK 的版本都是 23，模拟器使用的是 Marsh-mallow (Android 6) 或 Lollipop (Android 5.\*)。使用 Android Studio 2.0 或者 2.1（beta）作为主要的 IDE，其中集成了 Gradle 2.10 或者更高版本作为构建工具。  
 预备知识  
 Android 的 Gradle 插件至少需要一些 Java、Groovy、Gradle 和 Android 的知识。学习这些中的每一个主题，都能找到一整本书，所以本书中不会详细讲解每一个主题。  
 本书主要针对那些熟悉 Android 开发的人。本书没有提供很多 Android 背景，但是所有示例代码都能够在本书的 Github Repo 中找到。理解 Android 意味着理解 Java，所以本书假设读者也熟悉 Java。  
 然而本书假设读者对 Groovy 和 Gradle 的了解有限。附录 A 包含对 Groovy 语法和技巧的快速总结。在介绍具体事例的时候也会复习一些遇到的 Groovy 概念。附录 B 有一些基本的 Gradle 信息，但是本书本身就会详细地讨论 Gradle。除了这些局限以外，本书尽可能地被设计得很独立，在合适的地方提供了外部参考（特别是文档）的链接。  
 本书广泛应用了 Android Studio, 因为它是目前 Android 开发时官方唯 一支持的 IDE。Android Studio 提供了关于 Gradle 的视图和工具，我们在描述很多技巧的时候会展示。虽然本书并不是一本 Android Studio 的教程，但是其相关功能还是在可能的时候被提及到，而且如果因此而帮助读者学习了 IDE, 那就更好了

**关键词**：Gradle；Android Stdio；构建

# Android开发Gradle基础

提到了gradle 在android开发中使用，常见属性配置，从命令行和AS分别怎样构建APK，依赖的添加和仓库配置。可参见：[Gradle系列《一》](http://blog.csdn.net/cai_iac/article/details/51860419" \t "_blank)

## 什么是Gradle

Gradle是一个基于Apache Ant和Apache Maven概念的项目自动化建构工具。它使用一种基于Groovy的特定领域语言来声明项目设置，而不是传统的XML。当前其支持的语言限于**[Java](http://lib.csdn.net/base/javase" \t "_blank" \o "Java SE知识库)**、Groovy和[Scala](http://lib.csdn.net/base/scala" \t "_blank" \o "Scala知识库)，计划未来将支持更多的语言。——维基百科

以下简要介绍几个相关概念：

1. 建构工具做的工作包括：编译源代码、运行测试、拷贝Class文件到目标目录、签名、打包、依赖管理等。目的是让开发人员更加专注于代码的编写。

2. **Groovy**：是Java平台上设计的面向对象编程语言。这门动态语言拥有类似Python、Ruby和Smalltalk中的一些特性，可以作为Java平台的脚本语言使用。Groovy的语法与Java非常相似。

相对于Ant和Maven来说，Gradle可以算是新一代的自动化构建工具，而且Gradle在基于它们的基础上优化的同时还可以兼容这两者。

## Gradle与Android Studio

Android Studio(以下采用缩写AS)采用的构建工具即Gradle，Gradle可以辅助我们简化很多复杂的操作流程如：依赖管理、多渠道打包、签名信息配置、批量修改生成的apk文件名等，因此学习Gradle显得更为重要。当然，以下的介绍的Gradle相关特性和使用都是以AS为例的。

## **AS工程与Gradle相关的目录**

使用Android Studio新建一个工程之后，其目录结构是这样的（需要强调一下，以下的目录结构以及文件内容都只是新建项目时默认的样子，更高级的用法以及特点将会在后续的章节提及）：   
├── app········· #Android App目录   
│ ├── app.iml···············AS识别项目的配置文件   
│ ├── **build**··················**#模块构建输出目录（代码编译后生成的文件存放的位置、生成的Apk文件等）**   
│ ├── **build.gradle**····**#module构建脚本**   
│ ├── libs····················· 相关库文件目录   
│ ├── proguard-rules.pro··proguard混淆配置   
│ └── src·······················源代码，资源等   
│   
├── **build**··············**#构建输出目录**   
│ ├── generated   
│ └── intermediates   
│   
├── **build.gradle**·····**#工程构建文件**   
│   
├── **gradle**   
│ └── wrapper   
│   
├── **gradle.properties**······**#gradle相关的全局属性设置**   
├── gradlew ·························#[**Linux**](http://lib.csdn.net/base/linux)下的gradle wrapper可执行文件   
├── gradlew.bat ·················**#windows下的gradle wrapper可执行文件**  
├── local.properties···········配置Androod SDK位置文件   
└── **settings.gradle**··········**#工程配置**   
 接下来详细介绍几个与Gradle相关的文件以及目录。

### **build.**gradle

顶级构建文件，可以为所有子项目/模块添加常用的配置选项。见Code 1‑2

Code ‑ build.gradle

|  |
| --- |
| //buildscript中的声明是gradle脚本自身需要使用的资源  //可以声明的资源包括依赖项、第三方插件、maven仓库地址等。  buildscript {  repositories {  jcenter() //使用jcenter库  //mavenCentral()  }  dependencies {  // 配置使用gradle所需要依赖的版本库  classpath 'com.android.tools.build:gradle:2.1.0'  //注意：不要在这里配置项目所需要的依赖  }  }  //项目自身所需要的一些配置  allprojects {  repositories {  jcenter()  }  } |

### **app/build.gradle**

模块（module）中的构建文件，对当前模块生效。开发过程中最需要注重的一个文件，应用的相关特性都在这里进行配置。

Code ‑

|  |
| --- |
| //apply是gradle的一个函数  //若该module是应用（application）模块，则com.android.application  //若library则为 'com.android.library'）  apply plugin: 'com.android.application'  android {  //编译项目所用的SDK版本(即编译时的API版本)，com.android.support与//这个一致,建议采用最新版本  compileSdkVersion 23  //构建工具版本  buildToolsVersion "23.0.3"  defaultConfig {  //安装时，依据该ID区分是否为同一个应用  applicationId "com.wiky.supporttest"  //最低支持的系统版本（必须>=所有依赖库所支持的最低版本）  minSdkVersion 9  targetSdkVersion 23  versionCode 1  versionName "1.0"  }  // buildTypes是构建类型，常用的有release和debug两种，可以在这里面启用混淆，启用zipAlign以及配置签名信息等。后面再具体介绍  buildTypes {  release {  minifyEnabled false  proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'  }  }  }  //该module所需的依赖库配置  dependencies {  compile fileTree(dir: 'libs', include: ['\*.jar'])  testCompile 'junit:junit:4.12'  compile 'com.android.support:appcompat-v7:23.3.0'  } |

关于compileSdkVersion 和 buildToolsVersion这两个版本，在我们导入一些开源的项目时，也是需要关注的。最好在导入项目之前，找到对应的build.gradle文件，然后将其版本号改为当前可正常使用的版本号（即你本地能够顺利运行的项目中对应的版本号）。这样可以很大程度上避免导入项目时的“卡死”（构建中…）现象。

### **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，即告诉gradle项目中有哪些模块（module）需要处理，文件内容：

* include ':app'，表示settings.gradle与app在同一级目录，有多个module的话，格式如：include ':app', ':otherModule',.....
* includeFlat ‘K-Core’，表示settings.gradle的父目录（settings.gradle/..）与K-Core在同一级目录

### gradle.properties

该文件配置了一些builde.gradle中需要用到的常（变）量、编译优化。比如：

org.gradle.daemon=true

org.gradle.parallel=true

org.gradle.jvmargs=-Xmx8000M -XX:MaxPermSize=1024m

module\_KCore = k.kcore:K-Core:1.1.1

MAVEN\_URL=http://IP/nexus/

### gradle-wrapper

它允许你的机器在不需要安装运行的情况下就能运行Gradle脚本（我们在下载AS后可以直接开始使用gradle、开发项目，而无需另外下载及安装gradle，主要归功于这一特性），而且更重要的是它确保了build脚本运行在指定版本的Gradle。它会从中央仓库中自动下载Gradle，解压到你的文件系统，然后用来build。

## 升级gradle版本

由于Android Studio中已经默认支持了Gradle的使用，所以会发现我们在从未手动进行Gradle的下载、安装的情况下也能正常工作。这是因为AS（IDE）在我们创建一个新的Android App的时候，会自动生成gradlew(Gradle Wrapper)的相关文件以及脚本（在Window平台下，为gradlew.bat）。若当前不存在Gradle,则这些文件便会自动下载及安装Gradle。

既然都不需要我们下载安装的话，那么这个Gradle是如何存在的以及何时存在的呢？   
我们可以看到gradle/wrapper下有一个gradle-wrapper.properties文件，注意里面一句：

|  |
| --- |
| distributionUrl=https\://services.gradle.org/distributions/gradle-3.1-all.zip |

即表示了当前Gradle（Wrapper）的下载路径以及版本（3.1），（若不存在对应版本）它将会在我们第一次使用gradlew（如输入指令：gradlew -v）的时候开始下载安装。（注意，Gradle Wrapper对应的指令是gradlew而不是gradle）。

因此，当我们有需要更新Gradle(Wrapper)版本时，即通过修改distributionUrl的路径即可，版本信息可见：[Gradle版本仓库](https://services.gradle.org/distributions" \t "_blank),，以Window系统为例，下载完的文件通常存在于/Users/”your pc\_name”/.gradle/wrapper/dists,还有源代码可以看

# Gradle高级用法

## app/build.gradle

这里面的配置是与我们应用开发最为息息相关的。首先，先看看我们都可以在这个文件中配置哪些内容（功能）。另外，这里略去了部分上文提到的一些配置：

Code ‑ app/build.gradle

|  |
| --- |
| android {  defaultConfig {  //默认配置项  }  buildTypes {    }  compileOptions {  // Java 的版本配置  //配置使用JDK1.8（8），比如需要使用Lambda特性时，那么就可以在这里进行JDK版本的配置  sourceCompatibility JavaVersion.VERSION\_1\_8  targetCompatibility JavaVersion.VERSION\_1\_8  }  sourceSets {  //源码设置(项目目录结构的设置)  }  packagingOptions {  //打包时的相关配置  }  lintOptions {  //编译的 lint 开关，程序在buid的时候，会执行lint检查，有任何的错误或者警告提示，都会终止构建，我们可以将其关掉。  //abortOnError false  }  productFlavors {  //产品发布的一些东西，比如渠道、包名等  flavor1 {  }  flavor2 {  }  }  signingConfigs {  //签名的配置  release {  }  }  } |

### buildTypes

|  |
| --- |
| 1. release { 2. //为发布版本启用混淆 3. minifyEnabled true 4. //混淆文件 5. proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro' 6. } 7. debug { 8. //同时在一台设备安装debug和release版apk。 9. applicationIdSuffix ".debug" 10. } |

### compileOptions–配置JDK版本

Groovy中花括号包含的部分成为一个闭包（Closure），compileOptions 是一个 Method， 它的参数是一个闭包，这个闭包内依次执行了两个方法 -- sourceCompatibility 和targetCompatibility， 参数都是JavaVersion.VERSION17。

|  |
| --- |
| 1. compileOptions { 2. //配置使用JDK1.8（8），比如需要使用Lambda特性时，那么就可以在这里进行JDK版本的配置 3. sourceCompatibility JavaVersion.VERSION\_1\_8 4. targetCompatibility JavaVersion.VERSION\_1\_8 5. } |

### **sourceSets**

配置项目的目录结构，其中较为常见的一个应用场景是，将Eclipse 中的项目迁移至AS，由于两者目录结构相差较大，所以需要手动指定，部分代码如：

|  |
| --- |
| sourceSets {  main {   * + - 1. manifest.srcFile 'AndroidManifest.xml'       2. java.srcDirs = ['src']       3. resources.srcDirs = ['src']       4. aidl.srcDirs = ['src']       5. renderscript.srcDirs = ['src']       6. res.srcDirs = ['res']       7. assets.srcDirs = ['assets']       8. jniLibs.srcDirs = ['libs']   }  } |

### **packagingOptions –打包时的相关配置**

当项目中依赖的第三方库越来越多时，有可能会出现两个依赖库中存在同一个（名称）文件。如果这样，Gradle在打包时就会提示错误（警告）。那么就可以根据提示，然后使用以下方法将重复的文件剔除。

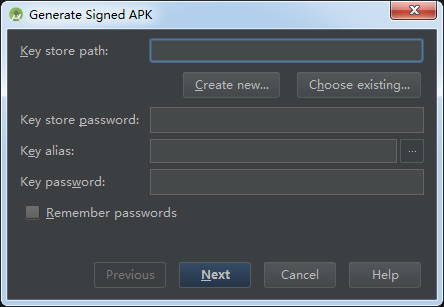
|  |
| --- |
| packagingOptions {  //这个是在同时使用[butterknife、dagger2](http://blog.csdn.net/m00123456789/article/details/54693454)做的一个处理（待验证）。同理，遇到类似的问题，只要根据gradle的提示，做类似处理即可。  exclude 'META-INF/services/javax.annotation.processing.Processor'  } |

### productFlavors

### signingConfigs–应用签名配置

|  |
| --- |
| 1. android { 2. signingConfigs { 3. release { 4. //签名证书文件 5. storeFile file("release.keystore") 6. //别名 7. keyAlias "release" 8. //key的密码 9. keyPassword "123456" 10. //证书的密码 11. storePassword "123456" 12. } 13. debug { 14. } 15. } 16. //需要注意的是：配置好签名信息后，需要在buildTypes配置使用 17. buildTypes { 18. release { 19. signingConfig signingConfigs.release 20. } 21. debug { 22. } 23. }   } |

当然，应用签名可以借用AS(build – > generate signed apk）提供的图形化界面来完成，如:



### 批量修改生成的apk文件名

|  |
| --- |
| 1. //获取并格式化时间信息，注意，应该是与android{}同一层级 2. def buildTime() { 3. return new Date().format("yyyy\_MM\_dd", TimeZone.getTimeZone("UTC")) 4. } 5. android { 6. buildTypes { 7. release { 8. //只针对release版本进行配置 9. applicationVariants.all { 10. variant -> 11. variant.outputs.each { 12. output -> 13. def outputFile = output.outputFile 14. if (outputFile != null && outputFile.name.endsWith('.apk') && 'release'.equals(variant.buildType.name)) { 15. // 输出apk格式命名如：Gradle\_v1.0\_2016\_07\_01\_wandoujia.apk 16. def fileName = "Gradl\_v${variant.versionName}\_${buildTime()}\_${variant.flavorName}.apk" 17. output.outputFile = new File(outputFile.parent, fileName) 18. } 19. } 20. } 21. } 22. } 23. } |

### AndroidManifest里的占位符

我们知道，项目中的很多配置都需要在这里定义。而当在需要生成多个版本并要为每个版本配置相应的属性值时，以我们最基本的做法（每生成一个版本前修改对应的属性值）来看，这将是一件麻烦的事情。现在Gradle为我们提供了一个解决方案，就是利用manifestPlaceholders，进行动态赋值。它的使用有点类似HashMap。现在还是以地图key的配置举例：我们需要在AndroidManifest里配置一个name为ampapkey的meta-data，这样这个meta-data的值就表示这个apk。

|  |
| --- |
| serverApp {  buildConfigField "boolean", "BUILD\_KPAD", "false"  applicationId = appId1  manifestPlaceholders = [packagePlace: applicationId1, applicationLabel: "@string/app\_name", amapKey: "amapKey"]  }  clientApp {  buildConfigField "boolean", "BUILD\_KPAD", "false"  applicationId = appId2  manifestPlaceholders = **[packagePlace: applicationId2, applicationLabel: "@string/app\_name", amapKey: "amapKey2"]**  } |

可以发现，AndroidManifest中的一些属性值也是可以动态改变的，这样在对特定包进行特定处理也将变得十分方便，更多的用法依次类推即可

### 自定义你的BuildConfig

BuildConfig.[Java](http://lib.csdn.net/base/javase)是Android Gradle自动生成的一个java类文件，无法手动编译，但是可以通过Gradle控制，也就是说他是动态可配置的。比如在build.gradle中配置**buildConfigField “boolean”, “LOG\_DEBUG”, “false”**后，我们点击“Sync now”后，即可在代码中使用BuildConfig.java新加的LOG\_DEBUG属性，如：

|  |
| --- |
| public final class BuildConfig {  //...  // Fields from build type: debug  public static final boolean LOG\_DEBUG = true;  } |

buildConfigField 一共有3个参数，第一个是数据类型，即新增成员变量的类型（与Java的类型是对等）；第二个参数是成员变量名，如：LOG\_DEBUG；第三个参数是常量值。

以下我们介绍一种常见的用法，场景：我们需要在debug版本包中输出log，而不希望在release版本中输出log，那么就可以通过以下方法进行配置（当然，我们代码中的log开关是使用BuildConfig中相关属性进行设置的）：

另外，还用一种常见的情况是：当我们的服务器有debug和release环境时，我们可能也需要在App对应的版本中配置对应的信息。如在debug版本中配置服务端debug环境对应的url。这样我们在开发的过程中将减少很多麻烦的工作。

此外，在对特定渠道包处理时进行特定的配置。

总之，在定义常量的时候，就可以考虑使用这个方法.

# 构建类型和定制

## 这一章主要解决了以下几种问题：  1. 当debug/release版本有部分代码或者资源需要不同的时候如何处理？  2. 当debug与release两个版本不够满足具体情况时，比如我需要：免费版（debug/release）、收费版（debug/release）时如何处理？

可参考：[Build Variant在项目中的应用](http://blog.csdn.net/cai_iac/article/details/52330535) ，文章整理的比较简陋，如果有遇到以上提到的两种情况，建议直接学习原文。

# 3.1处理构建类型51 3.2产品定制和变种54 3.3合并资源58 3.4定制维度65 3.5合并不同定制间的Java源代码69 第4章自定义任务75 4.1编写自定义任务75 4.2添加自定义任务到构建过程78 4.3排除任务81 4.4自定义代码集83 4.5使用Android库86 第5章测试95 5.1单元测试95 5.2使用Android测试支持库测试101 5.3Robotium用于功能测试106 5.4使用Espresso进行Activity测试110 第6章性能和文档117 6.1性能推荐117 6.2DSL文档123 附录A刚好够用的Groovy知识127 附录BGradle基础137

# 经典算法

正面人验复现。

## 算法原理

本节详。

### 基于线性

线性物体类（Linear Object Class, LOC）理论是基于样本学习的正面人脸合成的重要理论基石，同时也是流形假设的前身。

#### 算法思想

人脸图像可

#### 算法描述

下面我们将详细阐述算法过程：

。

## 本章小结

本章主要介绍了

# 参考

1. 张勇迪. 2014.基于移动终端的人脸识别身份认证技术研究[D]. 湖南大学..
2. 董小慧，高戈，陈亮，等. 2014. 数据驱动局部特征转换的噪声人脸幻构[J]. 计算机应用. 34(12):3576-3579.
3. 2016.11.[Gradle Recipes for Android学习.](http://blog.csdn.net/cai_iac/article/details/53305374)
4. 2014.10. [Gradle学习系列之](http://www.cnblogs.com/CloudTeng/p/3417762.html).
5. 2015.10 key网复制-[深入理解 Android 之 Gradle](http://wiki.jikexueyuan.com/project/deep-android-gradle/five.html)
6. [Gradle In Action详细笔记](https://lippiouyang.gitbooks.io/gradle-in-action-cn/content/index.html)

## QA

Allprojects含义

Repositories的定义，工作原理

task clean(**type**: Delete) {  
 delete rootProject.buildDir  
}这个含义呢？

编译优化：org.gradle.daemon=true

org.gradle.parallel=true

org.gradle.jvmargs=

找本编译原理的书籍

Gradle怎么编译的

Gitbook是否可以搞起？

app/build.gradle 如何添加变量啊

packagingOptions {  
 pickFirst **'lib/armeabi-v7a/libstlport\_shared.so'**}

？

productFlavors未学习完成

# 附录 部分推导与数学证明

## 附录A [Android SDK版本逻辑](http://www.cnblogs.com/ashitaka/p/6069904.html)

Manifest配置/gradle配置：compileSdkVersion, minSdkVersion 和 targetSdkVersion

OS接口：@TargetApi(19)/ android.os.Build.VERSION.SDK\_INT(ro.build.version.sdk)，[运行时检查系统版本](http://developer.android.com/training/basics/supporting-devices/platforms.html?utm_campaign=adp_series_sdkversion_010616&utm_source=medium&utm_medium=blog" \l "version-codes" \t "_blank)

|  |  |  |  |
| --- | --- | --- | --- |
|  | compileSdkVersion | minSdkVersion | targetSdkVersion |
| 定义 | Gradle**编译**App的SDK版本，**不会被打包**的APK文件中 | 应用可以运行的最低系统版本，**会被打包**的APK文件中 | 应用在运行时行为的系统版本。**会被打包**的APK文件中 |
| 作用 | 要和 Support Library 保持一致，它纯粹只是在编译的时候使用。（你真的应该修复这些警告，他们的出现一定是有原因的） | 开发时：[lint](http://developer.android.com/tools/debugging/improving-w-lint.html?utm_campaign=adp_series_sdkversion_010616&utm_source=medium&utm_medium=blog) 默认会在项目中运行，它在你使用了高于 minSdkVersion  的 API 时会警告你，帮你避免调用不存在的 API 的运行时问题。  安装时：minSdkVersion 是 Google Play 商店用来判断用户设备是否可以安装某个应用的标志之一 | （必须>=所有依赖库所支持的最低版本）；举例说明其作用：假设我们compileSdkVersion采用了23（6.0），但是项目暂时还未对6.0的特性（如运行时权限）做相应的处理，  那么可以设置targetSdkVersion的版本低于23。这样，应用仍可以正常运行在6.0的机子上（当然也就不具有运行时权限的特性？）。 |
| 注意 | 1. 修改 compileSdkVersion 不会改变运行时的行为。当你修改了 compileSdkVersion 的时候，可能会出现新的编译警告、编译错误，但新的 compileSdkVersion 不会被包含到 APK 中： 2. 强烈推荐总是使用最新的 SDK 进行编译。在现有代码上使用新的编译检查可以获得很多好处，避免新弃用的 API ，并且为使用新的 API 做好准备。 | 如果只在较高版本的系统上才使用某些 API，通常使用[运行时检查系统版本](http://developer.android.com/training/basics/supporting-devices/platforms.html?utm_campaign=adp_series_sdkversion_010616&utm_source=medium&utm_medium=blog" \l "version-codes" \t "_blank)的方式解决。 | Android系统平台的行为变更，只有targetSdkVersion的属性值被设置为大于或等于该系统平台的API版本时，才会生效 |
|  | 1. 按照上面示例那样配置，你会发现这三个值的关系是：minSdkVersion <= targetSdkVersion <= compileSdkVersion 2. 理想上，在稳定状态：minSdkVersion (lowest possible) <= targetSdkVersion == compileSdkVersion (latest SDK)用较低的 minSdkVersion 来覆盖最大的人群，用最新的 SDK 设置 target 和 compile 来获得最好的外观和行为。 | | |

如果你