<http://blog.csdn.net/u011913612/article/details/51732632>

[理解与配置Android studio中的gradle](http://blog.csdn.net/u011913612/article/details/51732632)

标签： [android](http://www.csdn.net/tag/android" \t "_blank)[studio](http://www.csdn.net/tag/studio)[gradle](http://www.csdn.net/tag/gradle)[DSL](http://www.csdn.net/tag/DSL)[配置](http://www.csdn.net/tag/%e9%85%8d%e7%bd%ae)

2016-06-22 18:02 10230人阅读 [评论](http://blog.csdn.net/u011913612/article/details/51732632#comments)(2) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/u011913612/article/details/51732632#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

Android gradle（1） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

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/u011913612/article/details/51732632)

使用gradle构建**[Android](http://lib.csdn.net/base/android" \o "Android知识库" \t "_blank)**应用时，你总是需要这样一个文件：build.gradle。你应该已经看过这个文件了，如果没有看过的话，你现在就可以看一下，它没有多少内容。它的简洁性得益于它提供了很多对设置和属性的默认值。gradle是基于groovy语言的，不过如果只是用它构建普通的工程的话，是可以不去学groovy的，如果想深入的做一下自定义的构建插件，可以考虑学一下groovy，因为它是基于**[Java](http://lib.csdn.net/base/javase" \o "Java SE知识库" \t "_blank)**的，所以你有java基础的话，学习不会很难。

这篇博客旨让任何一个人能看懂android studio的gradle scripts，主要会从gradle的简单语法，gradle scripts的脚本结构，每一个脚本（build.gradle,settings.gradle）的作用，脚本中每一项的意义等方面说明gradle scripts.如果想了解如何详细配置gradle,比如实现一个工程中，使用同一部分java代码，不同的资源res,一次生成多个不同渠道商的apk，可以看下我的这篇博客,它对如何配置gradle有较细致的介绍：[详细配置android studio中的gradle](http://blog.csdn.net/u011913612/article/details/51744302" \t "_blank)

**1.projects , tasks and action**

    是的，工程，任务和行为。一个项目至少要有一个工程，一个工程至少要有一个任务，一个任务由一些action组成。如果project比较抽象的话，可以这么理解，一个build.gradle对应一个project，而action就好像java中的方法，他就是一段代码的集合。在工程构建的过程中，gradle会根据build.gradle中的配置信息生成相应的project和task。

    Project实质上是一系列task的集合，每一个task执行一些工作，比如编译类文件，解压缩文件，删除文件等等。

**1.1构建过程**

    1.1.1初始化阶段。首先会创建一个Project对象，然后执行build.gradle配置这个对象。如果一个工程中有多个module,那么意味着会有多个Project,也就需要多个build.gradle.

    1.1.2配置阶段。这个阶段，配置脚本会被执行，执行的过程中，新的task会被创建并且配置给Project对象。

    1.1.3执行阶段。这个阶段，配置阶段创建的task会被执行，执行的顺序取决于启动脚本时传入的参数和当前目录。

**1.2 task**

    task标示一个逻辑上的执行单元，你可能已经用过它很多次了，不知道你有没有意识到。当你当你重新编译工程的时候，会用到一个叫做build 的task，当你清理工程的时候，会用到一个叫做clean 的task(后面会讲到)，gradle 已经为你准备了很多的task,可以使用 gradle tasks 来查看，比如，这里列出来一些：

**[html]** [view plain](http://blog.csdn.net/u011913612/article/details/51732632) [copy](http://blog.csdn.net/u011913612/article/details/51732632)

[print?](http://blog.csdn.net/u011913612/article/details/51732632)[在CODE上查看代码片](https://code.csdn.net/snippets/1725292)

1. assemble - Assembles all variants of all applications and secondary packages.
2. build - Assembles and tests this project.
3. buildDependents - Assembles and tests this project and all projects that depend on it.
4. buildNeeded - Assembles and tests this project and all projects it depends on.
5. clean - Deletes the build directory.

此外，你还可以自己声明一个task,比如像这样:

**[java]** [view plain](http://blog.csdn.net/u011913612/article/details/51732632) [copy](http://blog.csdn.net/u011913612/article/details/51732632)

[print?](http://blog.csdn.net/u011913612/article/details/51732632)[在CODE上查看代码片](https://code.csdn.net/snippets/1725292)

1. task haha {
2. println "haha"
3. }

然后使用gradle haha命令，就会打印出haha。这里，haha这个任务被执行了，所以说task就是个执行单元。你还可以使用如下方法来定义task:

**[html]** [view plain](http://blog.csdn.net/u011913612/article/details/51732632) [copy](http://blog.csdn.net/u011913612/article/details/51732632)

[print?](http://blog.csdn.net/u011913612/article/details/51732632)[在CODE上查看代码片](https://code.csdn.net/snippets/1725292)

1. task hello **<<** {
2. println "hello world"
3. }

这和前者是有区别的,“<<”意思是给hello这个task添加一些action,其实就是调用了task的doLast方法，所以，它和以下代码时等价的：

**[html]** [view plain](http://blog.csdn.net/u011913612/article/details/51732632) [copy](http://blog.csdn.net/u011913612/article/details/51732632)

[print?](http://blog.csdn.net/u011913612/article/details/51732632)[在CODE上查看代码片](https://code.csdn.net/snippets/1725292)

1. task hello {
2. doLast{
3. println "hello world"
4. }
5. }

关于haha 和 hello的区别，你还可以这样加深影响：

首先，进入到你的工程目录，执行gradle   （后面没有任何参数，另外，这个时候，build.gradle中同时有hello 和 haha 两个tasks），结果如下：

**[html]** [view plain](http://blog.csdn.net/u011913612/article/details/51732632) [copy](http://blog.csdn.net/u011913612/article/details/51732632)

[print?](http://blog.csdn.net/u011913612/article/details/51732632)[在CODE上查看代码片](https://code.csdn.net/snippets/1725292)

1. E:\android\androidwork2.0\GradleTest**>**gradle
2. haha
3. Incremental java compilation is an incubating feature.
4. :help
6. Welcome to Gradle 2.13.
8. To run a build, run gradle **<task>** ...
10. To see a list of available tasks, run gradle tasks
12. To see a list of command-line options, run gradle --help
14. To see more detail about a task, run gradle help --task **<task>**
16. BUILD SUCCESSFUL
18. Total time: 21.877 secs
19. E:\android\androidwork2.0\GradleTest**>**gradle tasks

可以按到haha,被打印了，而hello没有被打印，注意，这个时候默认执行的task 是help,也就是说并没有执行haha 这个task,可它还是被打印了，就说明使用定义haha 这种方式定义的task在初始化阶段就会被执行，而使用定义hello这种方法定义的task在执行阶段才会被执行。

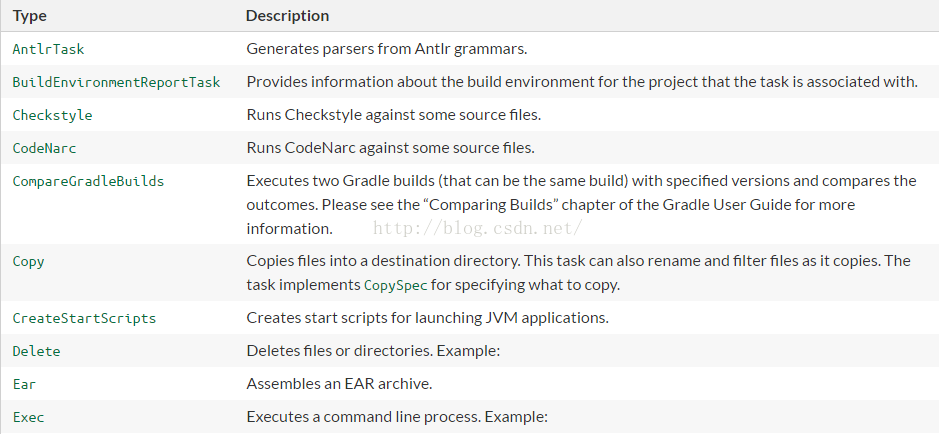
在android studio的顶层build.gradle中有这样一个task:

**[java]** [view plain](http://blog.csdn.net/u011913612/article/details/51732632) [copy](http://blog.csdn.net/u011913612/article/details/51732632)

[print?](http://blog.csdn.net/u011913612/article/details/51732632)[在CODE上查看代码片](https://code.csdn.net/snippets/1725292)

1. task clean(type: Delete) {
2. delete rootProject.buildDir
3. }

可以看到这个task有一个类型type，task有很多种类型的，以下列出来一些：



这里用到了delete类型，task的类型可以这样理解吧：task中文就是任务，任务有很多种类，Delete就是说这是个删除文件的任务。

这里就不更深入的探讨task了，这些类容已经可以使我们可以理解android studio中遇到的内容了。

**2.Closures**

**2.1 定义闭包**

理解gradle需要首先理解闭包的概念，Closure就是一段代码块，代码块一般要用{}包起来，所以闭包的定义可以向以下的样子：

**[java]** [view plain](http://blog.csdn.net/u011913612/article/details/51732632) [copy](http://blog.csdn.net/u011913612/article/details/51732632)

[print?](http://blog.csdn.net/u011913612/article/details/51732632)[在CODE上查看代码片](https://code.csdn.net/snippets/1725292)

1. def haha = { println 'haha!' }
2. haha()
3. #output:haha!

可以看到闭包虽然可以认为是一段代码块，但它可以向函数一样调用，而且它还可以接受参数，比如像下面这样：

**[java]** [view plain](http://blog.csdn.net/u011913612/article/details/51732632) [copy](http://blog.csdn.net/u011913612/article/details/51732632)

[print?](http://blog.csdn.net/u011913612/article/details/51732632)[在CODE上查看代码片](https://code.csdn.net/snippets/1725292)

1. def myClosure = {String str -> println str }
2. myClosure('haha!')
3. #output: haha!

这样这个闭包就有参数了，多个参数只需要在->前面添加就好了。

**2.2 委托**

另外一个很酷的点是closure的上下文是可以改变的，通过Closure#setDelegate()。这个特性非常有用：

**[java]** [view plain](http://blog.csdn.net/u011913612/article/details/51732632) [copy](http://blog.csdn.net/u011913612/article/details/51732632)

[print?](http://blog.csdn.net/u011913612/article/details/51732632)[在CODE上查看代码片](https://code.csdn.net/snippets/1725292)

1. def myClosure = {println myVar} //I'm referencing myVar from MyClass class
2. MyClass hello = **new** MyClass()
3. myClosure.setDelegate(hello)
4. myClosure()
6. **class** MyClass {
7. def myVar = 'Hello from MyClass!'
8. }
10. #output: Hello from MyClass!

如上所示，创建closure的时候，myVar并不存在。但是没关系，因为当执行closure的时候，在closure的上下文中，myVar是存在的。这个例子中。因为在执行closure之前改变了它的上下文为hello，因此myVar是存在的。

**2.3闭包作为参数**

闭包是可以作为参数的传递的，以下是闭包作为参数的一些情况：

**[java]** [view plain](http://blog.csdn.net/u011913612/article/details/51732632) [copy](http://blog.csdn.net/u011913612/article/details/51732632)

[print?](http://blog.csdn.net/u011913612/article/details/51732632)[在CODE上查看代码片](https://code.csdn.net/snippets/1725292)

1. 1.只接收一个参数，且参数是closure的方法： myMethod(myClosure)
2. 2.如果方法只接收一个参数，括号可以省略： myMethod myClosure
3. 3.可以使用内联的closure： myMethod {println ‘Hello World’}
4. 4.接收两个参数的方法： myMethod(arg1, myClosure)
5. 5.和4类似，单数closure是内联的： myMethod(arg1, { println ‘Hello World’ })
6. 6.如果最后一个参数是closure，它可以从小括号从拿出来： myMethod(arg1) { println ‘Hello World’ }

**3.gradle DSL**

DSL(Domain Specific Language)，中文意思是特定领域的语言。gradle DSL就是gradle领域的语言。为了更好理解gradle，学习gradle DSL是有必要的。gradle的脚本虽然非常简短，但它有它的语法，如果不搞懂DSL，即便你知道了怎么修改脚本得到你想要的结果，你也不会理解为什么要这样修改。

**3.1 你必须知道的基本概念**

第一. gradle script是配置脚本，当脚本被执行的时候，它配置一个特定的对象。比如说，在android studio工程中，build.gradle被执行的时候，它会配置一个Project对象，settings.gradle被执行时，它配置一个Settings对象。Project,Settings这种对象就叫做委托对象，下图展示了不同脚本对应的不同的委托对象：



第二.每一个Gradle script实现了一个Script接口，这意味着Script接口中定义的方法和属性都可以在脚本中使用。

**3.2构建脚本的结构**

一个构建脚本由零个或多个statements和 script blocks组成。以下是对他们的说明，为了避免翻译错误，这里把原文贴出来。

A build script is made up of zero or more statements and script blocks. Statements can include method calls, property assignments, and local variable definitions. A script block is a method call which takes a closure as a parameter. The closure is treated as a configuration closure which configures some delegate object as it executes. The top level script blocks are listed below.

大概意思statments可以包括方法调用，属性分配，本地变量定义；script bolck则是一个方法，它的参数可以是一个闭包。这个闭包是一个配置闭包，因为当它被执行的时候，它用来配置委托对象。以android studio的build.gradle为例：

**[java]** [view plain](http://blog.csdn.net/u011913612/article/details/51732632) [copy](http://blog.csdn.net/u011913612/article/details/51732632)

[print?](http://blog.csdn.net/u011913612/article/details/51732632)[在CODE上查看代码片](https://code.csdn.net/snippets/1725292)

1. apply plugin: 'com.android.application'
3. android {
4. compileSdkVersion 23
5. buildToolsVersion "23.0.3"
7. defaultConfig {
8. applicationId "com.konka.gradletest"
9. minSdkVersion 16
10. targetSdkVersion 23
11. versionCode 1
12. versionName "1.0"
13. }
14. buildTypes {
15. release {
16. minifyEnabled **false**
17. proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
18. }
19. }
20. }
22. dependencies {
23. compile fileTree(dir: 'libs', include: ['\*.jar'])
24. }

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

**以上就是一条statements,其中apply 是一个方法，后面是它的参数。这行语句之所以比较难理解是因为它使用了缩写，写全应该是这样的：**

**[java]** [view plain](http://blog.csdn.net/u011913612/article/details/51732632) [copy](http://blog.csdn.net/u011913612/article/details/51732632)

[print?](http://blog.csdn.net/u011913612/article/details/51732632)[在CODE上查看代码片](https://code.csdn.net/snippets/1725292)

1. project.apply([plugin: 'com.android.application'])

这样是不是就很清楚了？project调用了apply方法，传入了一个Map作为参数，这个Map的key是plugin,值是com.android.application.

**[java]** [view plain](http://blog.csdn.net/u011913612/article/details/51732632) [copy](http://blog.csdn.net/u011913612/article/details/51732632)

[print?](http://blog.csdn.net/u011913612/article/details/51732632)[在CODE上查看代码片](https://code.csdn.net/snippets/1725292)

1. dependencies {
2. compile fileTree(dir: 'libs', include: ['\*.jar'])
3. }

它以上就是一条script block,但它却很难被理解，之所以这么难理解，是因为gradle语法中用了大量的简写，dependencies写完整应该是这样的：

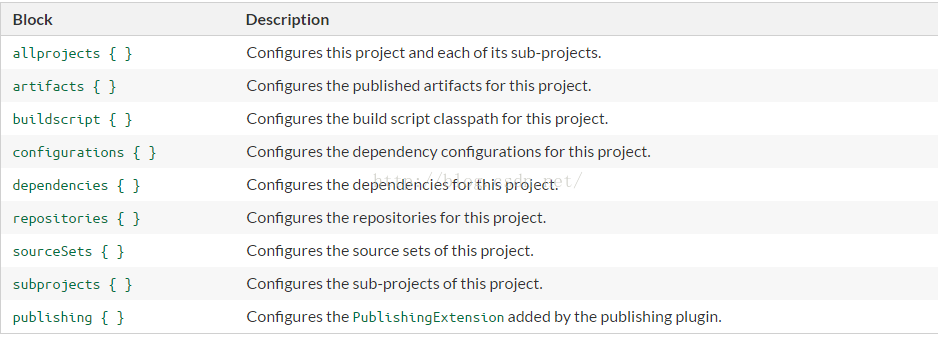
**[java]** [view plain](http://blog.csdn.net/u011913612/article/details/51732632) [copy](http://blog.csdn.net/u011913612/article/details/51732632)

[print?](http://blog.csdn.net/u011913612/article/details/51732632)[在CODE上查看代码片](https://code.csdn.net/snippets/1725292)

1. project.dependencies({
2. add('compile', 'com.android.tools.build:gradle:2.0.', {
3. // Configuration statements
4. })
5. })

我们知道block是一个闭包，这里首先调用project下的dependencies方法，这个方法的参数是一个闭包，这个闭包被传递给DependencyHandler，DependencyHandler有一个方法：add,这个add有三个参数，分别是'compile','...'和一个闭包。

gradle中有以下顶层build script block:



这里再以allprojects{ }为例，说一下script block是怎么工作的：

**[java]** [view plain](http://blog.csdn.net/u011913612/article/details/51732632) [copy](http://blog.csdn.net/u011913612/article/details/51732632)

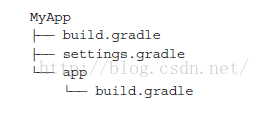
[print?](http://blog.csdn.net/u011913612/article/details/51732632)[在CODE上查看代码片](https://code.csdn.net/snippets/1725292)

1. allprojects {
2. repositories {
3. jcenter()
4. }
5. }

allprojects{ }一般是顶层build.gradle中的一个script block，它就是一个方法，这个方法接受一个闭包作为参数。gradle工具会先创建一个Project对象，它是一个委托对象（delegate object），它创建以后，build.gradle被执行，执行的过程中，allproject{ }方法被调用，这个方法的参数是一个闭包，然后闭包会被执行，用来配置Project对象。

**4.Understanding the Gradle files**

     理解了Project,task和action的概念以后，就可以就理解gradle的配置文件了。在android studio的工程中一般会有三个配置文件，它们各有各的功能。这三个文件的位置应该是这样的：



    构建一个工程的时候，会有以下顺序：

   1.创建一个Settings对象。

   2.检查settings.gradle是否存在，不存在就什么都不做，存在就用它来配置Settings对象。

   3.使用Settings对象创建Project对象，多Module工程中，会创建一系列的Project.

   4.检查build.gradle是不是存在，存在的话就用它来配置Project对象。

**4.1 settings.gradle**

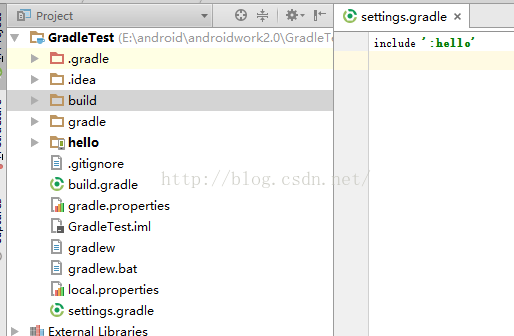
如果一个新的工程只包含一个android app,那么settings.gradle应该是这样的：

**[java]** [view plain](http://blog.csdn.net/u011913612/article/details/51732632) [copy](http://blog.csdn.net/u011913612/article/details/51732632)

[print?](http://blog.csdn.net/u011913612/article/details/51732632)[在CODE上查看代码片](https://code.csdn.net/snippets/1725292)

1. include ':app'

如果你的工程里只有一个 app,那么settings.gradle文件可以不要。include ':app'中的app指明你要构建的模块名，android studio默认的模块名师app，你可以把app目录的名字改掉，比如改成hello,那么这个时候你就必须把settings.gradle中的app也改成hello。这会是你非常有意义的一次尝试，因为有了这次尝试，以后你就可以按你所愿修改这个文件了。比如就像这样修改：



那么这个时候你肯定已经想试试一次性构建多个app了吧？你以前如果做过，那么你很厉害，你就不用看了，如果你没有试过，那么就和我一起试试吧：

第一步：在你的工程上右键，选择新建mudole。

第二步：你成功了！

是的就这么简单，现在看看工程的样子：



是的，这个时候，settings.gradle中多了一项，他就是我们新加的module的名字，它其实就是工程顶层目录下的一个目录的名字。这个名字你可以随便改，module你也可以随便加。

注意：settings.gradle实在初始化阶段被读入的，读入以后会生成一个Settings对象，然后会调用这个对象的一些方法。你没有必要了解这个对象，你知道它的存在对你理解项目构建的过程有所帮助。

**4.2 The top-level build file**

就是顶层的build.gradle脚本。这个文件中配置内容将会应用到所有modules中（上一步我们已经创建了两个module了，一个hello,一个gradletest2）。所以，每个module中都有的共同的属性，都会在顶层的build.gradle中配置，它默认有以下内容：

**[java]** [view plain](http://blog.csdn.net/u011913612/article/details/51732632) [copy](http://blog.csdn.net/u011913612/article/details/51732632)

[print?](http://blog.csdn.net/u011913612/article/details/51732632)[在CODE上查看代码片](https://code.csdn.net/snippets/1725292)

1. <pre style="font-family: 宋体; font-size: 9pt; background-color: rgb(255, 255, 255);"><pre name="code" **class**="java">buildscript {
2. repositories {
3. jcenter()
4. }
5. dependencies {
6. classpath 'com.android.tools.build:gradle:2.0.0'
8. // NOTE: Do not place your application dependencies here; they belong
9. // in the individual module build.gradle files
10. }
11. }
13. allprojects {
14. repositories {
15. jcenter()
16. }
17. }
19. task clean(type: Delete) {
20. delete rootProject.buildDir
21. }

这个脚本是由buildscript {},allprojects{} 两个script block组成，buildsctipt是一个顶层的build script block，正如2.2中所说的那样，是一个方法，参数是一个闭包，这个闭包里面又有一些script block，这些script bolck也是方法，参数也是一个闭包。最终这些闭包会被执行，用来配置对应的委托对象。比如，repositories这个方法的闭包调用了jcenter方法，这个方法会配置gradle的远程仓库，配置好了以后，在工程构建过程中，如果缺少依赖，就会在远程仓库中查找。顶层build.gradle中的配置会应用到所有的工程中，顶层build.gradle的委托对象是root Project，子工程目录下的build.gradle对应它自己的Project，总之，一个build.gradle对应一个Project。

至于每个script block的意义，但从字面意思上就能猜出一些来，比如allprojects {}就是为所有的project配置闭包中的内容，这里就是配置远程仓库，仓库有很多种，想使用其他仓库就可以在这里修改。buildsctipt{}为所有project配置构建用的仓库的工具，它里面的dependecbies{}就是配置构建工具的信息，从中可以看到构建工具是gradle,版本是2.0.0;所以，修改gradle的版本就可以在这里改。不过单从名字得到的信息是远远不够的，为了获取更多的信息，你可以看看《gradle for android》这本书。

**4.3子工程下的build.gradle**

**[java]** [view plain](http://blog.csdn.net/u011913612/article/details/51732632) [copy](http://blog.csdn.net/u011913612/article/details/51732632)

[print?](http://blog.csdn.net/u011913612/article/details/51732632)[在CODE上查看代码片](https://code.csdn.net/snippets/1725292)

1. apply plugin: 'com.android.application'
3. android {
4. compileSdkVersion 23
5. buildToolsVersion "23.0.3"
7. defaultConfig {
8. applicationId "com.konka.gradletest"
9. minSdkVersion 16
10. targetSdkVersion 23
11. versionCode 1
12. versionName "1.0"
13. }
14. buildTypes {
15. release {
16. minifyEnabled **false**
17. proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
18. }
19. }
20. }
22. dependencies {
23. compile fileTree(dir: 'libs', include: ['\*.jar'])
24. testCompile 'junit:junit:4.12'
25. compile 'com.android.support:appcompat-v7:23.3.0'
26. }

4.3.1第一行是一个statement,调用了apply方法，这个方法的定义如下：

**[java]** [view plain](http://blog.csdn.net/u011913612/article/details/51732632) [copy](http://blog.csdn.net/u011913612/article/details/51732632)

[print?](http://blog.csdn.net/u011913612/article/details/51732632)[在CODE上查看代码片](https://code.csdn.net/snippets/1725292)

1. **void** apply(Map<String, ?> options);

它的作用是检查gradle有没有所声明的这个插件，有就什么都不做，没有的话就会使插件可用。

具体的每一个script block，它们大部分都是都是方法，都可以在android studio 中按住ctrl+鼠标左键，点进去看它的声明，每个方法都有注释来解释它的作用。

4.3.1 android block

android 是这个脚本中最大的块，它包含了andoird特有的插件，这些插件可以使用是因为之前调用了

apply **plugin**: **'com.android.application',**

此外，这里设置了编译android用的参数，构建类型等。

4.3.2 dependencies block

dependecies也是一个接受闭包作为参数的方法，这里设置了编译当前的app的依赖。如果当前app依赖外部的包，可以把这个包放到libs目录下面，然后右键，选择add as library,然后就会在这里生成一条compile ' ... '的记录。

4.3.3还有其他的一些配置，比如：

**[java]** [view plain](http://blog.csdn.net/u011913612/article/details/51732632) [copy](http://blog.csdn.net/u011913612/article/details/51732632)

[print?](http://blog.csdn.net/u011913612/article/details/51732632)[在CODE上查看代码片](https://code.csdn.net/snippets/1725292)

1. //这个是解决lint报错的代码
3. lintOptions {
5. abortOnError **false**
7. }
9. //<span style="font-family: Arial, Helvetica, sans-serif; font-size: 9pt;">签名设置</span>
11. signingConfigs {
13. myConfigs {
15. storeFile file("签名文件地址")
17. keyAlias "..."
19. keyPassword "..."
21. storePassword "..."
23. }
25. }
27. //<span style="font-family: Arial, Helvetica, sans-serif; font-size: 9pt;">混淆设置</span>
29. buildTypes {
31. release {
33. signingConfig signingConfigs.myConfigs
35. runProguard **true**
37. proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
39. }
41. }
43. //<span style="font-family: Arial, Helvetica, sans-serif; font-size: 9pt;">渠道打包（不同包名）</span>
45. productFlavors {
47. aaa{
49. applicationId = '包名'
51. }
53. bbb{
55. applicationId='包名'
57. }
59. }
61. }
63. //<span style="font-family: Arial, Helvetica, sans-serif; font-size: 9pt;">so文件的导入</span>
65. task copyNativeLibs(type: Copy) {
67. from fileTree(dir: 'libs', include: 'armeabi/\*.so') into 'build/lib'
69. }

总结：以上的所有内容展示了一个gradle工作的大致过程，gradle脚本的组成方式，概括性的介绍了android studio中每个gradle配置脚本的功能，大概的阐述了一些script block的作用。由于这篇博客旨在理解android studio的gradle的工作方式和脚本的做成结构，所以，如果想更详细的理解每一个script block的作用，可以看下《gradle for android》这本书。此外，后续的文章也会有详细的对常用script block的探讨。