<http://www.jianshu.com/p/60e556a968de>

用过android studio的对gradle应该都不陌生了，gradle文件的基本配置大同小异，略做了解使用应该是没什么问题了。但是深入细致的了解一下对于理解项目还是很有帮助的，尤其是遇到一些配置复杂的github项目，不了解gradle可能会遇到跑不起来又束手无策的情形。下面对gradle相关知识、用法做一下总结。

**DSL （domain specific language）**

即所谓领域专用语言，其基本思想是“求专不求全”，不像通用目的语言那样目标范围涵盖一切软件问题，而是专门针对某一特定问题的计算机语言。  
-DSL之于程序员正如伽南地之于以色列人，是最初也是最终的梦想。几乎自计算机发明伊始，人们就开始谈论DSL使用DSL了。  
-前几年迅速走红的Ruby on Rails就被誉为“Web开发领域专用语言”

**DSL 约等于 整洁的代码**

从概念上说，程序的编写过程就是把业务领域中的问题通过代码或者程序模型表达出来：  
计算机的程序模型较为单一（归根结底都是运算和存储）  
在面向对象技术成为主流的今天，通常情况下，计算机程序不太可能做到与业务领域中的概念一致，或者具有某些直觉的对应。因此，软件的修改和可维护性并没有想象中的容易。我们必须不断地将业务领域中的概念转换成相应的代码模型，然后再进行修改。这种间接性直接造成了软件的复杂度。  
而DSL的主要目的就是要消除这样的复杂度（或者说，以构造DSL的复杂度代替这种复杂度），DSL就要是要以贴近业务领域的方式来构造软件。因此，DSL的简洁性往往是一种思维上的简洁性，使我们不用费太多的气力就能看懂代码所对应的业务含义。

**DSL多以文本代码的形式出现**

多年来软件工程实践表明文本代码是最有效率的编辑形式。但是一些特殊领域，文本代码并不是最佳的表现形式，为了更好的贴近业务领域中的概念，我们可能会选择使用一些图形化的DSL。如：[DSM（Domain Specific Modeling）工具GEMS（Generic Eclipse Modeling System）中就大量地使用了不同的图形化的DSL来表述系统的各个不同侧面。

Gradle向我们提供了一整套DSL，所以在很多时候我们写的代码似乎已经脱离了groovy，但是在底层依然是执行的groovy  
为了从命令行运行gradle测试样例，首先

**配置环境变量**

1. 创建变量名：GRADLE\_HOME ，变量值：  
   C:\Users\jjx.gradle\wrapper\dists\gradle-2.5-all\d3xh0kipe7wr2bvnx5sk0hao8\gradle-2.5
2. 加入path  
   ;%GRADLE\_HOME%\bin;
3. 检查,如下就ok。
4. C:\Users\jjx>gradle -v
5. Gradle 2.5
6. Build time: 2015-07-08 07:38:37 UTC
7. Build number: none
8. Revision: 093765bccd3ee722ed5310583e5ed140688a8c2b
9. Groovy: 2.3.10
10. Ant: Apache Ant(TM) version 1.9.3 compiled on December 23 2013
11. JVM: 1.7.0\_75 (Oracle Corporation 24.75-b04)

OS: Windows 7 6.1 amd64

**初尝禁果**

在d：盘建个文件夹Test，文件夹下建个文件build.gradle  
打开文件，写个简单的代码

task helloWorld << {

println "Hello World"

}

打开cmd， d： 执行gradle helloWorld

C:\Users\jjx>d:

D:\>cd Test

D:\Test>gradle helloWorld

:helloWorld

Hello World

BUILD SUCCESSFUL

Total time: 3.714 secs

D:\Test>

同时，这时候发现已经自动在Test目录下创建了.gradle文件。  
上面helloWorld后的“<<”表示**追加**的意思，即向helloWorld中加入执行过程  
使用doLast可以达到同样效果

task helloWorldTwo {

doLast {

println 'helloWorldTwo'}

}

如果需要向Task的最前面加入执行过程，我们可以使用doFirst：

task helloWorldThree {

doFirst {

println 'helloWorldThree'}

}

耶，懂了！

**关于 task**

Gradle将当前目录下的build.gradle文件作为项目的构建文件。在上面的例子中，我们创建了一个名为helloWorld的Task，在执行gradle命令时，我们指定执行这个helloWorld Task。这里的helloWorld是一个DefaultTask类型的对象，这也是定义一个Task时的默认类型，当然我们也可以显式地声明Task的类型，甚至可以自定义一个Task类型  
下面再看一个小case：  
我们在Test文件夹下建一个src目录，建一个dst目录，src目录下建立一个文件，命名为here.txt  
然后在build.gradle中append一个task：

task helloWorld << {

println "Hello World"

}

task copyFile(type: Copy){

from "src"

into "dst"

}

代码中（type：Copy）就是“显式地声明Task的类型”，helloworld没有就是默认得DefaultTask类型咯。

然后cmd中执行命令

D:\Test>gradle copyFile

:copyFile

BUILD SUCCESSFUL

Total time: 2.645 secs

D:\Test>gradle copyFile

:copyFile

BUILD SUCCESSFUL

Total time: 3.361 secs

D:\Test>

好了！ here.txt也跑到dst中去啦！简单吧！

加一把火，我们来看一下当前目录下（即Test目录文件，这里也可以将这个目录理解为一个project，不过还没写有生产力的代码，哈哈哈）定义的task

D:\Test>gradle tasks

:tasks

------------------------------------------------------------

All tasks runnable from root project

------------------------------------------------------------

Build Setup tasks

-----------------

init - Initializes a new Gradle build. [incubating]

wrapper - Generates Gradle wrapper files. [incubating]

Help tasks

----------

components - Displays the components produced by root project 'Test'. [incubating]

dependencies - Displays all dependencies declared in root project 'Test'.

dependencyInsight - Displays the insight into a specific dependency in root project 'Test'.

help - Displays a help message.

model - Displays the configuration model of root project 'Test'. [incubating]

projects - Displays the sub-projects of root project 'Test'.

properties - Displays the properties of root project 'Test'.

tasks - Displays the tasks runnable from root project 'Test'.

Other tasks

-----------

copyFile

helloWorld

To see all tasks and more detail, run gradle tasks --all

To see more detail about a task, run gradle help --task <task>

BUILD SUCCESSFUL

Total time: 2.895 secs

D:\Test>

是不是很清晰呢，展示了各种task类型，task的作用，以及另外两个task相关的命令，相信聪明的你也一看就懂。

Gradle本身的领域对象主要有Project和Task。  
Project为Task提供了执行上下文，所有的Plugin要么向Project中添加用于配置的Property，要么向Project中添加不同的Task。  
一个Task表示一个逻辑上较为独立的执行过程，比如编译Java源代码，拷贝文件，打包Jar文件，甚至可以是执行一个系统命令或者调用Ant。另外，一个Task可以读取和设置Project的Property以完成特定的操作。是不是很屌的样子。  
task关键字其实是一个groovy中方法的调用，该方法属于Project，而大括号之间的内容则表示传递给task()方法的一个闭包。

**task间的依赖**

task之间是可以存在依赖关系，比如TaskA依赖TaskB，那么在执行TaskA时，Gradle会先执行TaskB，再执行TaskA。我们可以在定义一个Task的同时声明它的依赖关系：

task helloWorldFour(dependsOn:helloWorldThree) << {

println 'hellohelloWorldFour'

}

或者

task helloWorldFour << {

println 'hellohelloWorldFour'

}

helloWorldFour.dependsOn helloWorldThree

**可配置的task**

一个Task除了执行操作之外，还可以包含多个Property，其中有Gradle为每个Task默认定义了一些Property，比如description，logger等。  
另外，每一个特定的Task类型还可以含有特定的Property，比如Copy的from和to等。  
当然，我们还可以动态地向Task中加入额外的Property。在执行一个Task之前，我们通常都需要先设定Property的值。

task helloWorld << {

description = "this is helloWorld"

println description

}

或者通过调用Task的configure()方法完成Property的设置：

task helloWorld << {

println description

}

helloWorld.configure {

description = "this is helloWorld"

}

**花式task**

task showDescription1 << {

description = 'this is task showDescription'

println description

}

task showDescription2 << {

println description

}

showDescription2.description = 'this is task showDescription'

task showDescription3 << {

println description

}

showDescription3 {

description = 'this is task showDescription'

}

对于每一个Task，Gradle都会在Project中创建一个同名的Property，所以我们可以将该Task当作Property来访问，showDescription2便是这种情况。另外，Gradle还会创建一个同名的方法，该方法接受一个闭包，我们可以使用该方法来配置Task，showDescription3便是这种情况。  
耶！简单吧

**关于 Groovy**

Gradle是一种声明式的构建工具。  
在执行时，Gradle并不会一开始便顺序执行build.gradle文件中的内容，而是分为两个阶段，第一个阶段是配置阶段，然后才是实际的执行阶段。  
配置阶段，Gradle将读取所有build.gradle文件的所有内容来配置Project和Task等，比如设置Project和Task的Property，处理Task之间的依赖关系等。  
Gradle的DSL只是Groovy语言的内部DSL，也必须遵循Groovy的语法规则。  
Groovy语言中的两个概念，一个是Groovy中的Bean概念，一个是Groovy闭包的delegate机制。

**bean**

Groovy中的Bean和Java中的Bean有一个很大的不同，即Groovy动态的为每一个字段都会自动生成getter和setter，并且我们可以通过像访问字段本身一样调用getter和setter

class GroovyBeanExample {

private String name

}

def bean = new GroovyBeanExample()

bean.name = 'this is name' //实际调用的是"bean.setName('this is name')"

println bean.name //实际调用的是

"println bean.getName()"

采用像直接访问的方式的目的是为了增加代码的可读性，使它更加自然，而在内部，Groovy依然  
是在调用setter和getter方法。

**闭包的delegate机制**

简单来说，delegate机制可以使我们将一个闭包中的执行代码的作用对象设置成任意其他对象。

class Child {

private String name

}

class Parent {

Child child = new Child();

void configChild(Closure c) {

c.delegate = child

c.setResolveStrategy Closure.DELEGATE\_FIRST

c()

}

}

def parent = new Parent()

parent.configChild {

name = "child name"

}

println parent.child.name

在上面的例子中，当调用configChild()方法时，并没有指出name属性是属于Child的，但是它的确是在设置Child的name属性。  
事实上光从该方法的调用中，我们根本不知道name是属于哪个对象的，你可能会认为它是属于Parent的。  
真实情况是，在默认情况下，name的确被认为是属于Parent的，但是我们在configChild()方法的定义中做了手脚，使其不再访问Parent中的name（Parent也没有name属性），而是Child的name。  
在configChild()方法中，我们将该方法接受的闭包的delegate设置成了child，然后将该闭包的ResolveStrategy设置成了DELEGATE\_FIRST。这样，在调用configChild()时，所跟闭包中代码被代理到了child上，即这些代码实际上是在child上执行的。  
此外，闭包的ResolveStrategy在默认情况下是OWNER\_FIRST，即它会先查找闭包的owner（这里即parent），如果owner存在，则在owner上执行闭包中的代码。这里我们将其设置成了DELEGATE\_FIRST，即该闭包会首先查找delegate（本例中即child），如果找到，该闭包便会在delegate上执行。

**联想gradle中声明的方法**

在使用Gradle时，我们并没有像上面的parent.configChild()一样指明方法调用的对象，而是在build.gradle文件中直接调用task()，apply()和configuration()等方法。这是  
因为在没有说明调用对象的情况下，Gradle会自动将调用对象设置成当前Project。  
比如调用apply()方法和调用project.apply()方法的效果是一样的。查查Gradle的Project文档，你会发现这些方法都是Project类的方法。  
对于configurations()方法，该方法实际上会将所跟闭包的delegate设置成ConfigurationContainer，然后在该ConfigurationContainer上执行闭包中的代码。再比如，dependencies()方法，该方法会将所跟闭包的delegate设置成DependencyHandler。

**终于到了gradle**

**自定义Property**

Gradle还为我们提供了多种方法来自定义Project的Property。

**在build.gradle文件中定义Property**

添加一个名为property1的Property：

ext.property1 = "this is property1"

或者采用闭包的形式

ext {

property2 = "this is property2"

}

定义了Property后，使用这些Property时我们则不需要ext，而是可以直接访问：

task showProperties << {

println property1

println property2

}

还可以在执行命令行的时候加属性

task showCommandLieProperties << {

println property3

}

//以下是cmd中执行命令

gradle -Property3="this is property3" showCommandLieProperties

//通过JVM系统参数定义Property,与java类似，但是前面要约定以“org.gradle.project”为前缀

gradle -D org.gradle.project.property3="this is another property3" showCommandLieProperties

此外还可以通过环境变量来为Gradle设置Property，但是每一个Property都需要以“ORG\_GRADLE\_PROJECT\_”为前缀：

ORG\_GRADLE\_PROJECT\_property3="this is yet another property3"

**Gradle 的 Plugin**

Gradle最常用的Plugin便是java Plugin了。和其他Plugin一样，java Plugin并没有什么特别的地方，只是向Project中引入了多个Task和Property。当然，java Plugin也有比较与众不同的地方，其中之一便是它在项目中引入了构建生命周期的概念，就像Maven一样。但是，和Maven不同的是，Gradle的项目构建生命周期并不是Gradle的内建机制，而是由Plugin自己引入的。

**依赖管理**

一个项目总会依赖于第三方，要么是一个第三方类库，要么是自己开发的另一个module  
配置Gradle的Repository，就是告诉Gradle在什么地方去获取这些依赖

repositories {

mavenCentral()

jCentral()

}

jCentral()是大于mavenCentral()的一个仓库，现在是studio默认的仓库

Gradle对依赖进行分组，允许编译时使用一组依赖，运行时使用另一组依赖。每一组依赖称为一个Configuration，在声明依赖时，我们实际上是在设置不同的Configuration。

要定义一个Configuration，我们可以通过以下方式完成：studio一般不需要设置，应该是有默认的，即为classpath

configurations {

myDependency

}

通过dependencies()方法向myDependency中加入实际的依赖项：

dependencies {

//下面的myDependency是关键

myDependency 'org.apache.commons:commons-lang3:3.0'

}

//类似studio中的classpath

dependencies {

classpath 'com.android.tools.build:gradle:1.3.0'

}

//还有 这里的compile，testCompile

dependencies {

compile project(':library')

compile 'com.android.support:recyclerview-v7:22.2.1'

compile 'com.android.support:design:22.2.1'

compile 'com.evernote:android-intent:1.0.1'

testCompile 'junit:junit:4.8.2'

}

myDependency，classpath，compile，testCompile都是Configuration（一组依赖）。  
除了myDependency都不使我们定义的，为啥呢，android Plugin会自动定义compile和testCompile分别用于编译Java源文件和编译Java测试源文件。classpath应该是用于所有，我类推的。  
Gradle还允许我们声明对其他Project或者文件系统的依赖。

dependencies {

//library是另一个module的名字

compile project(':library')

}

对于本地文件系统中的Jar文件，我们可以通过以下方式声明对其的依赖：

dependencies {

//java

compile files('spring-core.jar', 'spring-aap.jar')

compile fileTree(dir: 'deps', include: '\*.jar')

//studio中一般这么写

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

}

**构建多module的project**

Gradle为每个build.gradle都会创建一个相应的module领域对象，在编写Gradle脚本时，我们实际上是在操作诸如module这样的Gradle领域对象。在多module的项目中，我们会操作多个module领域对象。Gradle提供了强大的多module构建支持  
要创建多module的Gradle项目，我们首先需要在根（Root）Project中加入名为settings.gradle的配置文件，该文件应该包含各个子module（其实就是一个子project）的名称。如setting.gradle中：

include 'library', 'demo'

类似module（子project）的build.gradle，（Root）Project也有自己的build.gradle，在里面通常设置：

allprojects {

repositories {

jcenter()

}

//通常studio项目没有，咱自己加的

apply plugin: 'idea'

task allTask << {

println project.name

}

}

allprojects()方法将repositories配置一次性地应用于所有的module（子Project）和root-project本身，当然也包括定义的Task，这个task配置到所有module里面了和root-project。

subprojects()方法用于配置所有的子Project（不包含根Project）

**步入巅峰**

Gradle本身只是一个架子，真正起作用的是Task和Plugin。

**自定义Task**

Gradle中的Task要么是由不同的Plugin引入的，要么是我们自己在build.gradle文件中直接创建的。

* 在build.gradle文件中直接定义  
  需要定义的Task类型不多时  
  Gradle其实就是groovy代码，所以在build.gradle文件中，我们便可以定义Task类。

class HelloWorldTask extends DefaultTask {

//@Optional，表示在配置该Task时，message是可选的。

@Optional

String message = 'I am jjx'

//@TaskAction表示该Task要执行的动作,即在调用该Task时，hello()方法将被执行

@TaskAction

def hello(){

println "hello world $message"

}

}

//hello使用了默认的message值

task hello(type:HelloWorldTask)

//重新设置了message的值

task helloOne(type:HelloWorldTask){

message ="I am a android developer"

}

* 在当前工程中定义Task类型  
  只能应用在当前module中，没什么卵用，下面是全局可用的
* 在单独的项目中定义Task类型  
  项目中存在大量的自定义Task类型时,在另外的一个gradle文件中定义这些Task，然后再apply到build.gradle文件中。  
  可以参考印象笔记的demo：<https://github.com/evernote/evernote-sdk-android>  
  中的：
* //这是插件
* apply plugin: 'com.android.application'
* //这里gradle-quality.gradle就是另外单独定义了task的gradle

apply from: '../build-config/gradle-quality.gradle'

**自定义Plugin**

与自定义task极其类似，可以类推理解，也是有3中方式定义，只是代码不一样：

apply plugin: DateAndTimePlugin

dateAndTime {

timeFormat = 'HH:mm:ss.SSS'

dateFormat = 'MM/dd/yyyy'

}

class DateAndTimePlugin implements Plugin<Project> {

//该接口定义了一个apply()方法，在该方法中，我们可以操作Project，

//比如向其中加入Task，定义额外的Property等。

void apply(Project project) {

project.extensions.create("dateAndTime", DateAndTimePluginExtension)

project.task('showTime') << {

println "Current time is " + new Date().format(project.dateAndTime.timeFormat)

}

project.tasks.create('showDate') << {

println "Current date is " + new Date().format(project.dateAndTime.dateFormat)

}

}

}

//每个Gradle的Project都维护了一个ExtenionContainer，

//我们可以通过project.extentions进行访问

//比如读取额外的Property和定义额外的Property等。

//向Project中定义了一个名为dateAndTime的extension

//并向其中加入了2个Property，分别为timeFormat和dateFormat

class DateAndTimePluginExtension {

String timeFormat = "MM/dd/yyyyHH:mm:ss.SSS"

String dateFormat = "yyyy-MM-dd"

}

每一个自定义的Plugin都需要实现Plugin<T>接口，除了给Project编写Plugin之外，我们还可以为其他Gradle类编写Plugin。  
以上是在build.gradle文件中直接定义Plugin，还可以在当前工程中、单独的项目中创建Plugin，一般情况不需要了解。

怎么样，通过以上学习，再看android studio的gradle代码是不是小case了呢！

下篇有机会一句一句分析studio 项目里常用的gradle代码