[关闭]

@jy00747195 2017-03-31 15:00 字数 6257 阅读 13

Gradle轻入门——半小时懂套路，一小时可开车

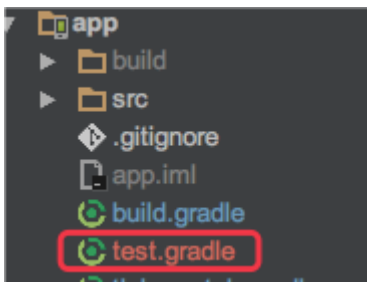
前言

作为新一代的构建工具，gradle的技术文章很多，要么过度讲细节，学习成本较高，要么一上来直接贴代码说结果，不懂的你还是不懂，本次分享已快速上手为目的，仅对最基础的知识点进行讲解，让你能最低成本的弄明白gradle，为后期的深入学习提供便利。

目录

1.快速实践groovy（四步骤）

1.1 在build.gradle同一目录下创建，test.gradle



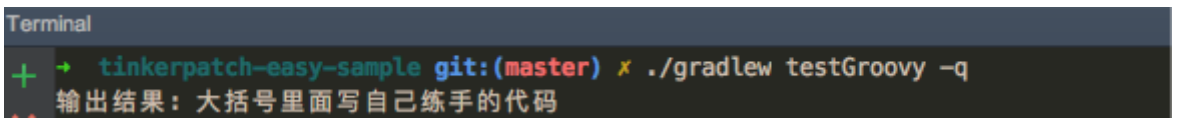
1.2 在build.gradle中加入如下代码

```
apply from: 'test.gradle'
```

1.3 在test.gradle里创建任意命名的task,注意命名后带上”<<“(后面解释原因)

```
task testGroovy<<{
    //groovy代码练手处
    String test = '大括号里面写自己练手的代码'
    println "输出结果: $test"
}
```

1.4 键入gradle命令运行对应task即可,下图已mac的命令行为例



2.Groovy与Java的一些区别

2.1 不需要分号

在 Groovy 中，分号是可选择采用的，你可以忽略不用它们，而且往往这种方法才是地道的用法。

2.2 随处可见的def类型

我们不用像Java一样，给每个赋值的对象都指出确切的类型。使用def即可。

```
String name = "Guillaume"
def name = "Guillaume"
```

在 Groovy 中使用def时，实际的类型持有者是Object，所以可以将任何对象赋予利用 def 定义的变量，如果一个方法声明为返回 def 类型值，则它会返回任何类型的对象

2.3 ==的行为差异

Java 的 == 实际相当于 Groovy 的 is() 方法，而 Groovy 的 == 则是一个更巧妙的 equals()。要想比较对象的引用，不能用==，而应该用 a.is(b)。

```
status != null &&status.equals(ControlConstants.STATUS_COMPLETED)
//等同于
status == ControlConstants.STATUS_COMPLETED
```

2.4 String与GString

```
println "Java风格的字符串拼接:"+name
println "Groovy风格的字符串拼接:${name}"
println "Groovy风格的字符串拼接:$name"
```

2.5 Getter 与 Setter

没有任何修饰符的独立“字段”导致 Groovy 编译器为你生成了一个私有字段和 getter 及 setter，你不必自己创建再字段的getter/setter，也不必写private。

```
//java写法
class Person {
    private String name
    String getName() { return name }
    void setName(String name) { this.name = name }
}
//groovy写法
class Person {
    String name
}
```

```
Person p = new Person();

//设置名字
p.setName('myName');
p.name = 'myName';//groovy写法

//取名字
def name = p.getName();
def name = p.name;//groovy写法
```

2.6 闭包（Closure）

闭包是一段代码，所以需要用花括号括起来，他在groovy中作为一等公民可以直接在方法中传递

```
//规范："->"左面是参数，右面是逻辑
{ Type1 param1, Type2 param2... ->
//闭包代码...
}

def c1 = { println "无参闭包" }
def c2 = { def x -> println "单参闭包，参数值：$x" }
def c3 = { def x, def y -> println "多参闭包，参数值：$x ,$y" }
def c4 = { x, y -> println "多参闭包(省略类型),参数值：$x ,$y" }


//以下调用闭包的形式均是合法的
c1.call()
c2.call 'a'
c3('a', 'b')
c4 'a', 'b'
```

3.groovy的一些简化的写法

3.1 省略方法后的括号

```
println("Hello")
println "Hello"

method(a, b)
method a, b
```

注意有些情况下，Groovy是不允许去除括号的。遇到顶级的表达式，自然可以忽略括号，但对于内嵌的方法调用则不允许忽略括号的。如：

```
def foo(x,y) {  return x+y }
println foo 1,2 // 不允许
```

3.2 当闭包成为方法调用的最后一个参数时，闭包可以写在括号外：

```
def testClosure1(Closure closure){
    println "this is testClosure1"
    closure.call()
}

def testClosure2(def parame,Closure closure){
    println "parame: $parame"
    closure.call()
}


testClosure1({})
testClosure1(){}
testClosure1{}
testClosure2(123,{})
testClosure2(123){}
```

3.3 retrun省略

在 Groovy 中，方法主体内部的最后一个求值表达式不必非得带上 return 关键字就能返回。所以对于短方法和闭包而言，忽略这个关键字会显得更简洁。

```
String toString() { return "a server" }
String toString() { "a server" }
def toString(){ "a server" }

def foo(n) {
    if(n == 1) {
        "Roshan"
    } else {
        "Dawrani"
    }
}
assert foo(1) == "Roshan"
assert foo(2) == "Dawrani"
```

注:如果不想让method有返回值 请用void修饰方法

3.4 闭包里的it

如果闭包携带一个参数，则可以不用给参数命名，直接使用it

```
def list = [1,2,3]

list.each { child ->
    println("list-child: $child")
}

list.each {
    println("list-it: $it")
}
```

4.Groovy的学习资料

- Groovy API文档
- Groovy入门，里面有详细的语法、GDK等介绍

我们来猜测下下面这段代码会返回什么？（查询API文档）

```
def map = [a: 1, b: 2]
map.each {
    println(it)
}
```

5. 什么是Gradle

> 从功能角度来说gradle就是一款自动化构建工具。从编程角度来说gradle和android一个样，就是一个编程框架,它有自己的API，自己的工作流程（生命周期）,我们编写所谓的编译脚本其实就是在这套框架里玩 Gradle 的 API。

既然都是编程框架，我们花一分钟回顾下android的基础都学了什么:AndroidManifest，四大组件，生命周期等。。
那么我们gralde也只需要学习一些基础对象，配置文件和工作流程就可以算是入门成功了。

6. 一些gradle常用到的对象（project、task、action）

6.1 Project对象

整个gralde最重要的对象，这是一个我们天天都在用的东西，引用官方文档的介绍：

> This interface is the main API you use to interact with Gradle from your build file. From a Project, you have programmatic access to all of Gradle's features.

此接口是您用于与构建文件中的Gradle进行交互的主要API。通过Project，您可以访问Gradle的所有功能进行编程。

project对象在哪里？我们该如何使用project?

通过命令我们先来找一下taqu项目下的project

```
./gradlew projects
```

```
Root project 'taqu'
+--- Project ':app'
+--- Project ':library-diagnose-network'
+--- Project ':library_base'
+--- Project ':library_http'
+--- Project ':library_taquwidget'
\--- Project ':library_utils'
```

我们看到在每一个module（app&library）都是一个独立的project对象，gradle在配置阶段（后面会介绍）为每个module自动生成project对象以及build.gradle，并将两者进行关联，形成一对一的关系，build.gradle就是project的配置脚本，其实我们在build.gradle的操作都是在配置project对象。

简单点记忆，我们通过build.gradle就可以访问Gradle的所有功能进行编程。

我们来看一个project方法的API

```
void apply(Map<String, ?> options)
```

Applies a plugin or script, using the given options provided as a map. Does nothing if the plugin has already been applied.

The given map is applied as a series of method calls to a newly created `ObjectConfigurationAction`. That is, each key in the map is expected to be the name of a method `ObjectConfigurationAction` and the value to be compatible arguments to that method.

The following options are available:

- `from`: A script to apply. Accepts any path supported by `Project.uri(java.lang.Object)`.
- `plugin`: The id or implementation class of the plugin to apply.
- `to`: The target delegate object or objects. The default is this plugin aware object. Use this to configure objects other than this object.

是不是很熟悉？这就是我们build.gradle最常用的apply方法。
所以想要深入学习build.gradle里我能做什么，请去看project的API文档。

6.2 Task

Task 是 Gradle 中的最基本一种数据类型，它代表了一些要执行或者要干的工作，作为开发者你的意图都是在task中实现。每一个 Task 都需要和一个Project关联。一个project可以有无数个task。gradle执行构建时也就是在有序的跑N个设定好的task。

```
[build.gradle]
//Task 是和 Project 关联的，所以，我们要利用 Project 的 task 函数来创建一个 Task
task myTask  <==myTask 是新建 Task 的名字

task myTask { configure closure }
task myType << { task action }  <==注意，<<符号 是 doLast 的缩写

task myTask(type: SomeType)
task myTask(type: SomeType) { configure closure }
```

6.3 Action

一个Task由若干个Action组成，Action就是一个闭包，一个Task包含若干Action,Task通过doFirst和doLast两个方法添加Action，所以，你可以理解为比task还小一层的代码块。

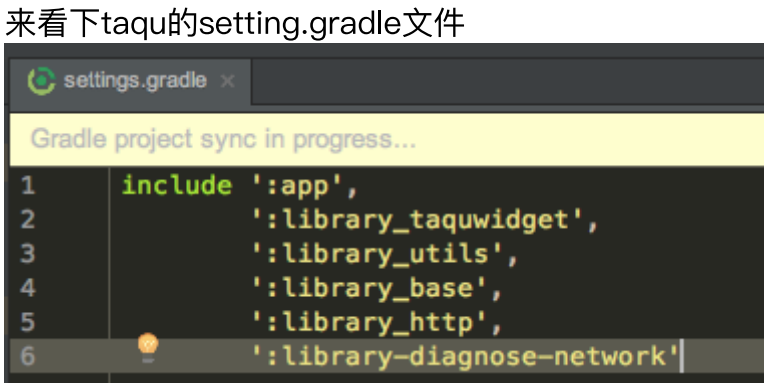
```
task testTask{
    doFirst {println 'doFirst'}
    doLast { println 'doLast'}
}

testTask.doFirst {println 'doFirst'}
testTask.doLast { println 'doLast'}
testTask<<{ println 'doLast'}
```

7. Gradle下的各个配置文件说明

7.1 setting.gradle

我们知道每个module都是一个独立的Project，如果我们想让几个project共同工作怎么办？gradle提供了一个Multi-Projects build的方法，让project产生关联，也就是我们的setting.gradle。

来看下taqu的setting.gradle文件



这个文件很重要，名字必须是 settings.gradle。它里边用来告诉 Gradle，这个 multiprojects 包含多少个子 Project。

settings.gradle 除了可以 include 外，还可以设置一些函数。这些函数会在 gradle 构建整个工程任务的时候执行，所以，可以在 settings 做一些初始化的工作

7.2 build.gradle

前面有讲过，build.gralde就是project的配置文件，每个project对应一个build.gralde，在使用Multi-Projects build方式开发时，rootProject也有一个build.gradle。

7.3 gradle.properties

这个文件实际上是用来配置gradle的一些基础属性的。

8. Plugin & Build Script Block

8.1 Plugin

虽说task是gradle的基础对象，但是在我们日常使用gradle进行配置工作的时候很少看到编写task逻辑的部分，是因为这些task都在plugin里。

plugin其实可以理解为java的jar包。对于gradle来说，作用是一样的一样的。作者在Plugin里面写好了一套task的逻辑和执行顺序，并且已DSL方法形式开放设置项，让其他程序员能定制自己要的配置方式。

实际上plugin是工作中最常遇到的，构建java用java plugin,构建android用android plugin。我们在定制android各项配置工作的时候，实际应该去了解的是plugin里的方法（下面会有android dsl文档）

8.2 Build Script Block

Build script structure

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.

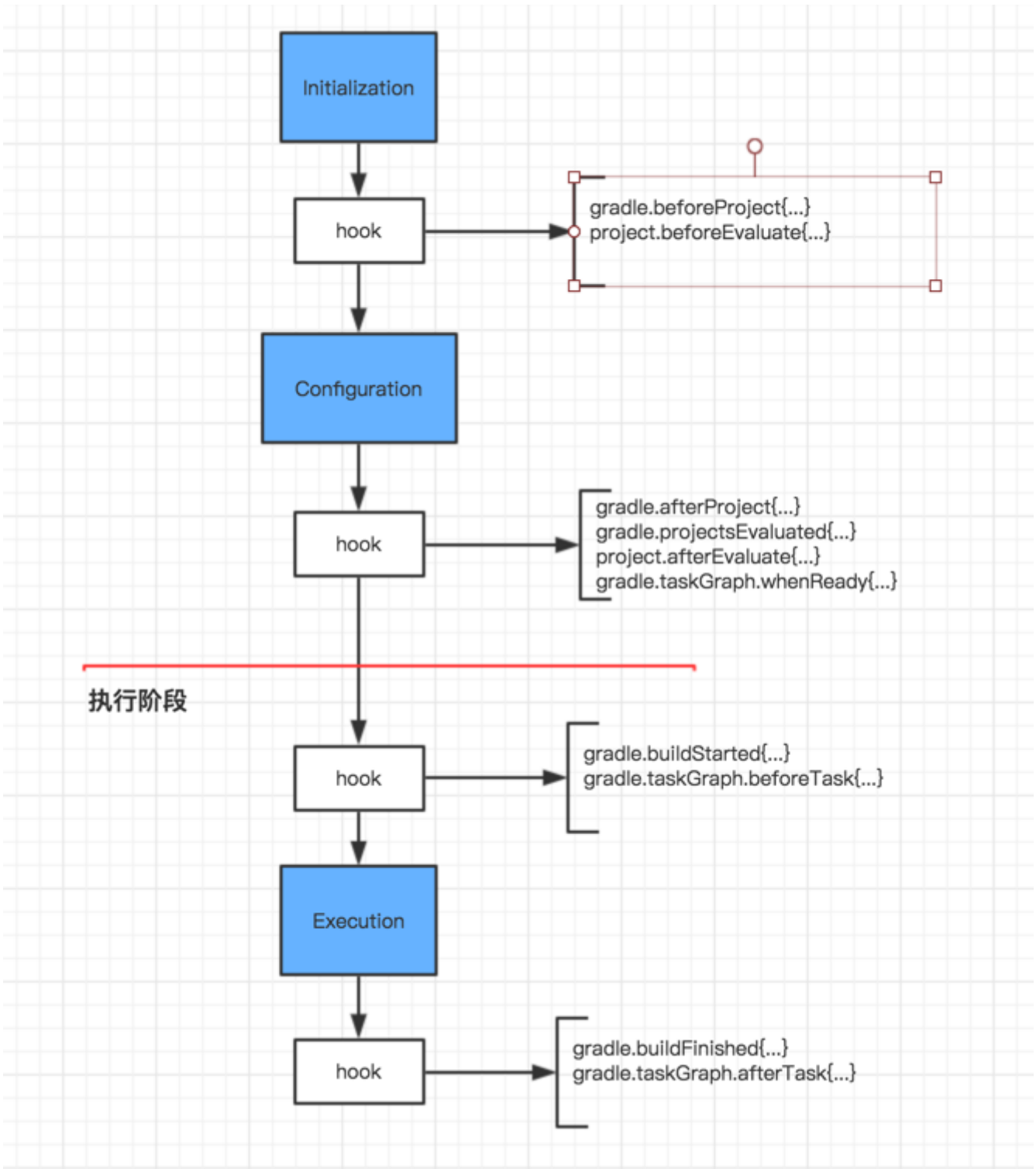
Block	Description
`allprojects { }`	Configures this project and each of its sub-projects.
`artifacts { }`	Configures the published artifacts for this project.
`buildscript { }`	Configures the build script classpath for this project.
`configurations { }`	Configures the dependency configurations for this project.
`dependencies { }`	Configures the dependencies for this project.
`repositories { }`	Configures the repositories for this project.
`sourceSets { }`	Configures the source sets of this project.
`subprojects { }`	Configures the sub-projects of this project.
`publishing { }`	Configures the `PublishingExtension` added by the publishing plugin.

有那么一类专门做配置的DSL方法，可以称为Script Block,你会经常在.gradle里看到，通过他们的API说明就知道他们的工作都是用来配置一些功能或者属性的。

到此，我们gradle日常工作中常会遇到的东西都已经叙述完了，了解这些东西可以在你阅读别人的代码能分清什么是什么。我们现在来阅读下taqu项目下的配置文件，看看我们是否阅读起来顺利很多。

9. Gradle的工作流程

就像android四大组件的生命周期一样，掌握了gradle的工作流程，会让你清楚选择什么时机去做什么事。一张图了解gradle的工作流程:



gradle的工作包含3阶段
1.initilalization初始化阶段，解释setting.gradle,决定哪些projects参与构建，并且为每一个项目创建一个Project类的实例对象。

2.Configration阶段,在这个阶段配置每个Project的实例对象,解析每个 project 中的build.gradle象，做一些配置插件属性、添加Task、修改Task的行为，为之后的Execution阶段做准备。我们平常在app下的build.gralde给android插件做的各个配置都是这个阶段用到。

3.Execution阶段（build阶段），Gradle在这个阶段确保有顺序的去执行事先编写好的各个task。android,freeline，tinker等插件实际上都是在这个阶段发挥作用的。

切记，切记，gradle的Execution阶段本质上只是在有序的跑Task！！！！！

这里说一个延展知识点：

```
//在Configration阶段输出
task testTask {
    println 'in Configration'
}
```

```
//在Execution阶段输出
task testTask <<{
    println 'in Execution'
}
```

图一没有<<的task，在Gradle里称之为任务配置块(task configuration)，会在Configration阶段执行。

10. Gradle学习资料

- Gradle API文档
- Android Plugin DSL文档
- Gradle In Action —— 经典的gradle教程
- gradle学习资料 —— 极客学院

后记

本次分享讲的东西都很基础，有些知识点到为止，如果你有兴趣更进一步的学习gradle，你可以去了解这些东西：

- task types
- task的依赖（决定task的执行顺序）
- extra property(ext属性)
- 自定义plugin

- **内容目录**

在作者公开此批注前，只有你和作者可见。

保存 取消

修改 保存 取消 删除

- 私有
- 公开
- 删除

查看更早的 5 条回复

回复批注

×

通知

取消 确认

-
-