

[主页](#) [更新日志](#) [产品说明](#) [使用技巧](#) [技术分享](#) [用户之声](#) [Bugtags 官网](#)[🔍 搜索](#)

CODE

关注我们：



# 拥抱 Android Studio 之三：溯源，Groovy 与 Gradle 基础

2016-01-04

#ANDROID #EMBRACEANDROIDSTUDIO #GRADLE #GROOVY

## 关于学习方式续

回忆起大学那个白衣飘飘的年代，开始金工实习却发现基础学的不够牢靠，越来越胆小，越来越糊涂。所幸得到一位高年级学姐指导，赶紧找当时的书或者笔记，快速把基础知识温习一遍，再结合实践中思考，终于豁然开朗。

### 目录

- 1. 关于学习方式续
- 2. 问题与解决方案
- 3. 学习 Groovy
  - 3.1. Groovy 概述
  - 3.2. 环境安装
  - 3.3. 初探
  - 3.4. 语言基础
    - 3.4.1. 文件与类，变量与函数
    - 3.4.2. 字符串
    - 3.4.3. List，Array 和 Map
    - 3.4.4. import
  - 3.5. 语言特性及其本质

下一篇

[Bugtags Web 2016-01-15 更新内容](#)

上一篇

[Bugtags Web 12-26 更新内容](#)

### 微信公众号

关注「bugtags」公众号，第一时间获取产品更新：

相信看过前一篇 [《Android Studio 与 Gradle 深入》](#) 的同学，有一部分就会遇到我初识 Gradle 时的困惑：代码我也依稀看得懂，但就是不知道还能这样写，为什么这样写。

## 问题与解决方案

回想我在 Gradle 的学习过程中遇到的问题与及其解决方案，总结出下面三点：

- 原理不懂：学习 Groovy 与 Gradle 的基础原理
- Gradle 实践不懂：学会找示例，学习开源例子
- 方法和属性不懂：学会查文档

下面的我将以解决三个问题为线索，介绍 Groovy 和 Gradle。

## 学习 Groovy

### Groovy 概述

[3.5.1. Closure \(闭包\)](#)

[3.5.2. 面向对象特性](#)

[3.6. Groovy 基础小结](#)

[4. 学习 Gradle](#)

[4.1. Gradle 安装](#)

[4.2. Gradle Build 的生命周期](#)

[4.2.1. Gradle 构建三个阶段：](#)

[4.2.2. Settings.gradle](#)

[4.2.3. Task](#)

[4.2.4. Plugin](#)

[5. Gradle 实践参考](#)

[6. 参考文档](#)

[7. 总结](#)

[8. 系列导读](#)

[9. 番外](#)



BUGTAGS官方交流群

126207501

#### 最新文章

[UPDATE](#)

BUGTAGS WEB 2017-02-10 更新内容  
2017-02-10

[UPDATE](#)

BUGTAGS 2016-11-16 更新内容  
2016-11-16

[UPDATE](#)

BUGTAGS 2016-11-02 更新内容  
2016-11-02

[INTRO](#)

BUGTAGS 在线修复功能介绍  
2016-10-21

Gradle 采用了 Groovy 语言作为主要的脚本语言。一个 build.gradle 文件，其实是一个 Groovy 类。

Groovy 是一个基于 JVM 的语言，代码最终编译成字节码（bytecode）在 JVM 上运行。它具有类似于 Java 的语法风格，但是语法又比 Java 要灵活和方便，同时具有动态语言（如 ruby 和 Python）的一些特性。

Groovy 的诸多特定，很适合用来定义 DSL（Domain Specific Language）。

“简单的来讲 DSL 是一个面向特定小领域的语言，如常见的 HTML、CSS 都是 DSL，它通常是以配置的方式进行编程，与之相对的是通用语言（General Purpose Language），如 Java 等。

既然是一门语言，就肯定有自己的特性。我们要从下面几个步骤来介绍 Groovy：

- 环境安装
- 语言基础
- 语言特性及其本质

## 环境安装

Groovy 官方[安装文档](#)提供多种方式进行安装，确保你不会在跪在环境配置的路上 ^-^：

### INTRO

BUGTAGS 远程配置功能介绍

2016-10-20

### 分类

▸ case (5)

▸ code (9)

▸ intro (10)

▸ news (3)

▸ skill (6)

▸ update (20)

### 归档

▸ February 2017 (1)

▸ November 2016 (2)

▸ October 2016 (3)

▸ September 2016 (1)

▸ August 2016 (2)

- Windows 下推荐 binary 包配置环境变量
- Mac 下推荐使用 sdkman 或者 Brew 进行安装
- Linux 下推荐 sdkman
- 嵌入在程序中，则推荐使用 Maven 远程依赖

初学者也没有必要使用IDE，平添障碍，后期用 IntelliJ IDEA Community 版本足矣。

下面只介绍 Mac 下使用 sdkman 的安装方式。

- 下载安装 sdkman，执行下面命令，按照提示安装即可

```
1 $ curl -s http://get.sdkman.io | bash
```

- 使环境变量生效

```
1 $ source "$HOME/.sdkman/bin/sdkman-init.sh"
```

- 安装 Groovy

```
1 $ sdk install groovy
```

▸ July 2016 (4)

▸ June 2016 (3)

▸ May 2016 (2)

▸ April 2016 (3)

▸ March 2016 (2)

▸ February 2016 (4)

▸ January 2016 (12)

▸ December 2015 (10)

▸ November 2015 (5)

标签

▸ Android (5)

▸ BTGLog (1)

▸ Beta (1)

▸ Bugtags (45)

▸ CSDN 大奖 (1)

▸ EmbraceAndroidStudio (5)

- 查看当前版本，如果能否运行且输出对应版本，就是成功了

```
1 $ groovy -version
2 Groovy Version: 2.4.4 JVM: 1.8.0_25 Vendor: Oracle Corporation OS:
```



## 初探

安装好环境之后，先来一个 hello, world !

- 新建文件

```
1 $ vim test.groovy
```

- 在其中写上内容

```
1 println "hello, world!"
```

- 保存退出，执行

▸ Gradle (5)

▸ Groovy (2)

▸ Live (1)

▸ Maven (1)

▸ PHP (1)

▸ android (2)

▸ canary (1)

▸ docs (1)

▸ gitbook (1)

▸ ndk (1)

▸ plugin (1)

▸ sendFeedback (1)

▸ setUserData (1)

▸ 产品测试 (6)

▸ 产品经理 (1)

▸ 企业版 (2)

▸ 使用手册 (2)

```
1 $ groovy test.groovy
2 hello, world!
```

Wow, So easy!

## 语言基础

下面将会用一些实际的例子，介绍一些最重要的点，

“例子都已经传到 [github](#) 的 [demo](#) 项目中。  
第一次使用 *demo* 项目的时候，需要等待自动下载几个远程包。  
笔者一个 *Java* 程序员，可以你能够看到很多 *Java* 的习性还是留在代码中。

## 文件与类，变量与函数

Groovy 代码文件，支持不显式声明类：

“ *ScriptClass.groovy*

```
1 println 'hello,world'
```

▸ 使用技巧 (5)

▸ 关键词 (1)

▸ 发展历程 (1)

▸ 后端 (1)

▸ 在线修复 (2)

▸ 在线标注 (1)

▸ 实时跟踪 (1)

▸ 小技巧 (1)

▸ 插件 (1)

▸ 日志工具 (1)

▸ 春节 (1)

▸ 更新 (20)

▸ 最具成长潜力奖 (1)

▸ 活动 (1)

▸ 测试任务 (1)

▸ 测试用例 (1)

▸ 热修复 (1)

这样一个 Groovy 脚本，经过编译之后，会产生一个继承自 `groovy.lang.Script` 类的子类：

是不是能看出点什么？

“ `groovy/build/classes/main/io/kvh/as/groovy/ScriptClass.class`

```
1  public class ScriptClass extends Script {
2      public ScriptClass() {
3          CallSite[] var1 = $getCallSiteArray();
4      }
5
6      public ScriptClass(Binding context) {
7          CallSite[] var2 = $getCallSiteArray();
8          super(context);
9      }
10
11     public static void main(String... args) {
12         CallSite[] var1 = $getCallSiteArray();
13         var1[0].call(InvokerHelper.class, ScriptClass.class, args);
14     }
15
16     public Object run() { // 关键方法
17         CallSite[] var1 = $getCallSiteArray();
18         return var1[1].callCurrent(this, "hello,world");// got it
19     }
20 }
```

▸ 瑞士军刀 (1)

▸ 用例导出 (1)

▸ 用户反馈 (15)

▸ 知乎 (1)

▸ 移动测试 (43)

▸ 第三方转发 (1)

▸ 红包 (1)

▸ 自定义数据 (1)

▸ 视频介绍 (2)

▸ 远程配置 (2)

▸ 重复标签 (1)

## 标签云

Android BTGLog Beta Bugtags CSDN 大奖  
EmbraceAndroidStudio Gradle Groovy Live  
Maven PHP android canary docs gitbook ndk  
plugin sendFeedback setData 产品测试 产品  
经理 企业版 使用手册 使用技巧 关键词 发展历  
程 后端 在线修复 在线标注 实时跟踪 小技巧 插  
件 日志工具 春节 更新 最具成长潜力奖 活动  
测试任务 测试用例 热修复 瑞士军刀 用例导出

Groovy 支持如下的方式来定义变量和函数：

[用户反馈](#) [知乎](#) [移动测试](#) [第三方转发](#) [红包](#) [自定义数据](#) [视频介绍](#) [远程配置](#) [重复标签](#)

## “ *VarAndMethod.groovy*

```
1  def varAndMethod() {  
2      def a = 1//不显式声明变量类型  
3      a = "abc"//运行时改变类型  
4  
5      println a//无需；结束一行代码  
6      a = 4//最后一行作为返回值  
7  }  
8  def ret = varAndMethod()//文件内运行方法  
9  
10 println ret//输出4
```

链接

► Bugtags

## 字符串


Groovy 支持单引号，双引号，三单引号声明一个字符串；

## “ *Quoted.groovy*

```
1  def quoted() {  
2      def singleQ = 'hello, single quot'// 声明为java.lang.String  
3      def doubleQ = "hello, double quot ${singleQ}"// 如果有${},!  
4      def tripleQ = '''hello,
```



```
5  triple quot'''// 允许多行，而不需要+号
6
7      println singleQ
8      println doubleQ
9      println tripleQ
10 }
```



Groovy 还支持以：

```
1  """..."""
2  /.../
3  $/.../$
```

来声明字符串，详情参见参考文档。

## List , Array 和 Map

Groovy 默认使用 `java.util.ArrayList` 来提供 List 服务，但提供了更加灵活易用的操作方式：

“ *Collections.groovy*

```
1  def playList() {
```

```
2      def lst = ["a",2,true]//支持不同类型元素
3
4      println(lst)
5  }
6
7  playList()
```

要使用 Array，需要显式声明：

```
1  def playArray() {
2      def intArr = [1, 2, 3] as int[]//显示声明
3      String[] strArr = ["a", "b"]//另外一种方式
4
5      println(intArr)
6      println(strArr)
7  }
8
9  playArray()
```

使用 key:value 的方式定义 Map，注意 key 的正确使用方式：

```
1  def playMap() {
2      def map = [a: "a", b: "b"]
3
4      println(map)
5  }
```

```
6      def key = "name"
7      def map2 = [key: 'a']//未使用
8      def map3 = [(key): 'a']//使用
9
10     println(map2)
11     println(map3)
12 }
13
14 playMap()
```

## import

Groovy 提供了更强大的 import

- 默认 import，这些类都是被默认 import 到代码中的，可以直接使用

```
1 import java.lang.*
2 import java.util.*
3 import java.io.*
4 import java.net.*
5 import groovy.lang.*
6 import groovy.util.*
7 import java.math.BigInteger
8 import java.math.BigDecimal
```

- import alias

引入一个类，通过 `as` 关键字赋予一个别名，有点 JavaScript 的意思么？

“ *Import.groovy*

```
1 import java.lang.String as KString
2
3 println(new KString("aaa"))
```

## 语言特性及其本质

### Closure（闭包）

闭包的概念不再赘述，大概就是可以将函数作为参数传递和使用，详情参见 [wikipedia](http://en.wikipedia.org/wiki/Closure)。

```
1 { [closureParameters -> ] statements }
```

可以省略方括号内的内容，也就是说，可以没有参数列表。

“ *Closure.groovy*

当闭包不声明参数列表，默认参数是 `it`；闭包被定义之后，是一个 Closure 对象，可以对其调用 `call` 方法使其执行。


```
1  def defaultIt() {
2      3.times {
3          println it //默认参数 it
4      }
5  }
6
7  defaultIt()
8  def closureObj() {
9      def obj = { a ->
10         ++a
11     }
12     println obj.call(1)
13 }
14
15 closureObj()
```

## 面向对象特性

- 定义和实例化一个类

“ *GroovyClass.groovy*

```
1  class People{
2      String name
3      int age
4  }
5
6  People p1 = new People();
7  People p2 = new People(name:"Luis",age: 29)//通过类似 map 的方式赋值参
```



- 方法的默认参数

```
1  def foo(String p1, int p2 = 1) {
2      println(p1)
3      println(p2)
4  }
5
6  foo("hello")
```

- Field 和 Property

Field 是以各种修饰符修饰的变量。Property是私有变量和自带的 getters/setters ,

下面的类具有私有变量 name、age , 并自带这两个变量的 getter 和 setter :

```
1 class People{
2     String name
3     int age
4 }
```

当变量声明为 final 的时候，默认就没有 setter

- Trait

Groovy 提供了一个叫做 Trait 特性实现了多继承，还有很多强大的功能，读者可以自己探索。

```
1 trait Fly {
2     void fly() {
3         println("fly")
4     }
5 }
6
7 trait Walk {
8     void walk() {
9         println("walk")
10    }
11 }
12
13 class Duck implements Fly, Walk {
14
15 }
```

```
16
17 Duck duck = new Duck()
18 duck.fly()
19 duck.walk()
```

## Groovy 基础小结

至此，我们已经熟悉了 Groovy 的基本语法和特性，相信你也能够使用 Groovy 写一些基础程序了。Groovy 还有很多深入的内容，请用到的时候，参考这个 pdf：[《Programming Groovy 2》](#)。

下面开始介绍使用 Groovy 写 Gradle 程序，主要的内容来自 [《Gradle Sser Guide》](#)。

## 学习 Gradle

### Gradle 安装

三种方式安装 Gradle：

-



[下载 zip 安装包](#)

- 

Mac 下使用 home brew

```
1 brew install gradle
```

- **推荐：**使用 IntelliJ IDEA ( Android Studio ) 自带的 wrapper 结构来下载 Gradle

“ *Wrapper 是为了让不同版本的插件能够使用其对应版本的 Gradle 的一个机制*

*Gradle Wrapper 会把不同的版本 Gradle 安装在：*

```
1 $USER_HOME/.gradle/wrapper/dists
```

## Gradle Build 的生命周期

回忆一下《Android Studio 与 Gradle 深入》中的 Android Studio 项目文件结构：

```
1  .
2  |— app                                //app module
3  |   |— build.gradle                  //app module 的 build.gradle
4  |— build.gradle                      //项目 build.gradle, 通常配置
5  |— gradle.properties                //项目属性文件, 通常可以放置一些常量
6  |— lib                                //lib module
7  |   |— build.gradle                  //lib module 的 build.gradle
8  |— settings.gradle                  //项目总体设置, 通常是配置项目中
```

## Gradle 构建三个阶段：

- 

初始化：Gradle 支持单 module 构建和多 module 构建（Android Studio 创建的项目默认是多 module）。初始化阶段，Gradle 会为每一个 module 中的 build.gradle 文件创建一个 Project 实例。

- 

配置：项目根目录的 build.gradle 会首先被执行

- 执行：执行所选取的 task

## Settings.gradle

多 module 构建要求在项目根目录下有一个 settings.gradle，用来指定哪些 module 参与构建，如：

“ *settings.gradle*

```
1 include ':app', ':groovy'
2
3 println 'print in settings.gradle'
```

在 settings.gradle 文件中，添加一行打印语句，在控制台中，切换到当前项目根目录下执行：

```
1 ./gradlew -p groovy build
```

可以看出 settings.gradle 的代码每次都会率先执行。

## Task

接下来，我们开始学习 Gradle 的核心 Task。

“ *groovy/build.gradle*

定义一个 Task：

```
1 task hello {  
2     doLast {  
3         println 'Hello,'  
4     }  
5 }
```

执行命令，查看输出：

```
1 $ ./gradlew hello  
2 Hello,
```

Task 也可以这样定义：

```
1 task World << {
```

```
2     println 'World!'
3 }
```

注意，如果定义成这样：

```
1 task hi {
2     println 'description hi'
3 }
```

在进行初始化和配置的时候，下面语句就会运行。

```
1 println 'hi'
```

这种语法通常是用来定义 task 的描述信息。

Task 可设置 `dependsOn` 和 `finalizedBy`：

```
1 task hello {
2     doLast {
3         println 'Hello,'
4     }
5 }
```

```
6
7  task intro(dependsOn: hello) << {
8      println 'intro'
9  }
10
11  World.finalizedBy hello
```

执行 intro 之前，会先执行 hello；执行 World 之后，会自动执行 hello。

## Plugin

Gradle 的核心代码，只提供了一个框架，具体的功能（如构建 Android 工程）是通过插件机制来实现的。

Gradle 提供了大量官方的插件，如 Maven、Groovy、Java、Publishing、Signing 等，也有大量第三方的插件（Android），甚至每个人都可以自己实现一个插件(如 笔者开发的 Bugtags 插件，这个将在最后一篇讲述)。

这些 plugin 定义了一系列的 task、DSL 和约定，在 build.gradle 文件使用这些 plugin：

```
1  apply plugin: java
```

当你写了一个独立的 file\_uri.gradle 文件，你可以通过：

```
1  apply from: 'file_uri.gradle'
```

来引入你的 gradle 文件，这个文件甚至可以在某个服务器上。

## Gradle 实践参考

学习了基础理论之后，如果你还是不知道如何开始写，那就先来实现一个自定义 apk 名称的功能吧！

```
1  android.applicationVariants.all { variant -> //获取 variant 参数，就
2      variant.outputs.each { output -> //获取输出文件
3          def file = output.outputFile //修改实例
4          output.outputFile = new File(
5              (String) file.parent,
6              (String) file.name.replace(
7                  file.name,
8                  // alter this string to change output file
9                  "Your_Apk_Name_" + variant.name + "_" + va
10             )
11         )
12     }
13 }
```

你问我怎么知道 android 下有个 `applicationVariants`？其实我也不知道的，也得找文档。

因为使用的是 Android 的插件，那就得在谷歌搜 “android gradle plugin dsl”，果然有个 [Android Plugin DSL Reference](#)。

点进去找找，里面有关于 build variant 的文档：[applicationVariants](#)，既然是一个 Set，那就可以调用 `all` 方法。

写代码调试，再配合文档，你就晓得该怎么写了。

如果你还是不知道如何入手，那我提供几个开源参考：

- [gradle-bintray-plugin](#)：bintray 提供的开源插件
- [gradle-node-plugin](#)：一个运行 NodeJS 脚本的插件
- [linkedin-gradle-plugins](#)：linkedin 的 Gradle 插件集合

## 参考文档

相信参照开源项目动手写了几个小程序之后，你已经小有感觉了，那就记得把文档地址备齐了，用到的时候，查一下：

- [Groovy Documentation](#)：Groovy 的详细介绍文档



- [Groovy API Reference](#)：Groovy 的 API 文档，必要的时候查阅
- [Gradle User Guid](#)：Gradle 的详细介绍文档，很有必要过一遍
- [Gradle Build Language Reference](#)：Gradle DSL 参考，重点的几个 DSL 过一下，其他的用到再查
- [Android Plugin DSL Reference](#)：使用 Android 插件必备

另外，也有大量很好的中文文档，比如这几篇：

- [邓凡平老师的 Gradle 介绍](#)
- [Gradle User Guide 中文版](#)

## 总结

笔者从 Gradle 入门到现在略懂，经历了大量懵懂的时光。最后狠下心去系统学习了 Groovy 和 Gradle 的基础之后，最终茅塞顿开。希望读者遇到类似的情况，一定要沉下心，多学多练。

在接下来的两篇，我将分别介绍将发布远程库和编写 Gradle 插件。

## 系列导读

本文是笔者《拥抱 Android Studio》系列第三篇，其他篇请点击：

[拥抱 Android Studio 之一：从 ADT 到 Android Studio](#)

[拥抱 Android Studio 之二：Android Studio 与 Gradle 深入](#)

[拥抱 Android Studio 之三：溯源，Groovy 与 Gradle 基础](#)

[拥抱 Android Studio 之四：Maven 公共仓库使用与私有仓库搭建](#)

[拥抱 Android Studio 之五：Gradle 插件使用与开发](#)

“ 有问题？在文章下留言或者加 qq 群：453503476，希望能帮到你。

## 番外

笔者 kvh 在开发和运营 [bugtags.com](http://bugtags.com)，这是一款能够极大的提升 app 开发者测试效率的 SDK 产品，欢迎使用、转发推荐。

笔者目前关注点在于移动 SDK 研发，后端服务设计和实现。

我们团队长期求 PHP 后端研发，有兴趣请加下面公众号勾搭：

bugtags-logo

bugtags-logo

## 相关文章推荐

- [拥抱 Android Studio 之五：Gradle 插件开发](#)
- [拥抱 Android Studio 之四：Maven 仓库使用与私有仓库搭建](#)
- [拥抱 Android Studio 之二：Android Studio 与 Gradle 深入](#)

---

分享到：    [QQ空间](#)    [新浪微博](#)    [腾讯微博](#)    [人人网](#)    [微信](#)



© 2017 Bugtags, Ltd.

Powered by Hexo. Theme by PPOffice