**[Gradle构建之Groovy语法详解](https://dp2px.com/2019/04/12/gradle5/)**

**前言**

前面几篇基本上对Gradle构建在Android中的使用场景和流程都有了大致了解，包括如何使用以及如何自定义插件等，这些基本都是基于官方的教程引导来了解的。接下来我们通过几篇文章来深入了解一下具体的几个核心知识点，今天就以Groovy语法为切入点，接下来几篇你可能还会看到关于标准的Gradle插件、Maven插件、自定义构建等具体的知识点相关文章。

**Groovy初识**

在Apache的groovy-lang.org官网可以看到一句很醒目的话：

A multi-faceted language for the Java platform / 一个Java平台的多窗口语言

从这一句话中我们可以得知它的重要性，它可以帮助我们方便的使Java平台和其他语言交互，它也是一个面向对象的语言。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。由于其运行在JVM上的特性，Groovy可以使用其他Java语言编写的库。Groovy的语法与Java非常相似，大多数Java代码也符合Groovy的语法规则，尽管可能语义不同。

**和Java的差异**

Groovy语言在设计之初就让其可以很快速的让Java开发者使用，很多地方只会比Java更方便，而且让熟悉Java的你不感到别扭。

**省略**

Groovy语法允许省略分号和修改符。而且Groovy允许定义简单脚本，同时无需定义正规的class对象。

**默认导包**

默认情况下导入所有这些包和类，即您不必使用显式import语句来使用它们，所以在使用Groovy的时候不要导包。

java.io.\*

java.lang.\*

java.math.BigDecimal

java.math.BigInteger

java.net.\*

java.util.\*

groovy.lang.\*

groovy.util.\*

**没有类型的Java代码**

在Java中，如果要声明一个String变量，则必须输入：

String value = "Hello World";

但是，如果仔细想想，就会看出，等号右侧的字符已经表明value的类型是String。所以，Groovy允许省略value前面的String类型变量，并用def代替。

def value = "Hello World"

**调用重载方法**

在Groovy中在运行期间根据实际的参数类型来调用重载方法，而Java中在编译期间根据声明类型来调用重载方法。

int method(String arg) { //声明为String类型参数

return 1;

}

int method(Object arg) { //声明为Object类型参数

return 2;

}

Object o = "Object"; //声明类型为Object，而实际类型是String

int result = method(o);

所以上面代码在Java中执行result的结果是2，而在Groovy中执行result的结果是1

**默认public**

我们知道在Java中省略类、方法、属性的可见修饰符默认为default，只有在同一个包内可见，而且不可继承。

| **作用域** | **当前类** | **同一个包** | **子类** | **其他包** |
| --- | --- | --- | --- | --- |
| public |  |  |  |  |
| protected |  |  |  |  |
| default |  |  |  |  |
| private |  |  |  |  |

但是在Groovy中默认都是public修饰符，如果你想让字段只在同包内访问，可以使用@PackageScope修饰符。

class Person {

@PackageScope String name

}

**内部类**

静态内部类和匿名内部类与Java完全一致：

class A {

static class B {}

}

new A.B()

非静态内部类的唯一不同就是创建实例的不同，Groovy不支持y.new X()语法。相反，你必须写new X(y)：

public class Y {

public class X {}

public X foo() {

return new X()

}

public static X createX(Y y) {

//return y.new X(); Java中可以这样创建内部类

return new X(y)

}

}

**Lambda表达式**

Groovy不支持Lambda表达式语法，但有闭包替代。

**不一样的String**

在Groovy中用单引号的字符串是和Java相同的String对象，而用双引号的字符串不一定是String对象，因为Groovy支持GString（也就是字符串中携带变量）

def arg = "123"

def iamstring = 'i am String'

def ianstr = "i am String"

def iamgstr = "i am GString $arg" //GString

**不一样的双等于**

在Java中使用==来比较两个基本类型或者对象的equals()，而在Groovy中==会被翻译成a.compareTo(b)==0，需要实现Comparable和equals方法。如果要检查是否是相同类型则可以使用a.is(b)

**酷比了的循环**

在Groovy中可以使用循环范围，是不是觉得很酷呢？

def repeat(val){

for(i in 1..5){ //从1开始循环到5(包含5),循环5次

println val

}

}

如果我们要上面的循环不包括5可以使用小于号，例如：for(i in 1..<5){ }

**函数默认参数**

Groovy的函数中可以给函数参数定义默认值，如下下面的repeat参数：

def repeat(val, repeat=5){

for(i in 0..<repeat){

println val

}

}

另外值得注意的是在Groovy中什么函数参数不需要类型或者def声明。

**数组初始化**

在Groovy中，{ …​ }块被保留用于闭包。这意味着您无法使用以下语法创建数组：

int[] array = { 1, 2, 3}

替代方式，也就是Groovy中的方法如下：

def array = [1,2,3]

def hash = [name:"Andy", age:45]

**额外的关键字**

Groovy中的关键字比Java中的更多。不要将它们用于变量名称。

as

def

in

trait

**闭包**

和Java中的Lambda表达式（可以参考[《Lambda编程》](http://dp2px.com/2019/01/31/kotlin2/" \t "_blank)）不同的是，在Groovy中使用的是闭包（Closure），它是Groovy中非常重要的一个数据类型或者说一种概念了。

闭包，是一种数据类型，它代表了一段可执行的代码。语法格式def xxx = {paramters -> code}举例如下：

def aClosure = {//闭包是一段代码，所以需要用花括号括起来..

param1, param2 -> //这个箭头很关键。箭头前面是参数定义，箭头后面是代码

println"this is code" //这是代码，最后一句是返回值，

//也可以使用return，和Groovy中普通函数一样

}

从C/C++语言的角度看，闭包和函数指针很像。闭包在Groovy中大量使用，比如很多类都定义了一些函数，这些函数最后一个参数都是一个闭包。

def acoll = ["Groovy", "Java", "Ruby"]

acoll.each{

println it

}

这段代码是不是有些奇怪呢？事实上这个each()是个函数，只不过省略了括号而已（Groovy中，当函数的最后一个参数是闭包的话，可以省略圆括号）。

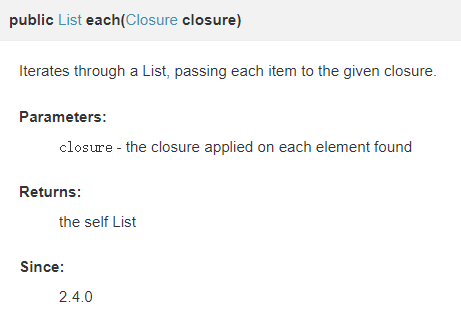
这个时候机智的你是不是想到了android构建中的android{ },这个时候理解了它原来是一个android( { } )函数。

**Groovy API**

我们都知道Java语言有一套JDK来帮助我们快速开发，而Groovy语言也一样，[API文档地址](http://www.groovy-lang.org/api.html" \t "_blank)。

为了方便Groovy语言更好的使用Java API，另外Groovy对JDK进行了包装，具体[API文档请参考这里](http://groovy-lang.org/gdk.html" \t "_blank)，我们可以利用Groovy的特性方便的使用Java API,这样极大的减少了我们的学习成本。

例如上面的each()方法就在java.util.List接口中定义了。

[](https://dp2px.com/post/gradle5/image1.png)Groovy的each方法

可以看到这个Closure就是一个闭包对象基类，所以这里接收了一个闭包作为参数。

作为一门语言，Groovy是复杂的，是需要深入学习和钻研的。一本厚书甚至都无法描述Groovy的方方面面，庆幸的是在Gradle中我们使用的只是Groovy中的一些简单知识，上面这些知识基本够用了。