# Groovy语法

许宏川 关注

0.37 2015.08.13 02:12:53 字数 1,617 阅读 12,600

这篇文章看完只是对Groovy有所了解，只是作为学Gradle的前置知识。
安装问题请看配置Groovy开发环境(Windows)。

## 简史

Groovy的1.0版本发布于2007年1月2日。
2012年的年中,Groovy的2.0版本发布了。
目前最新版本是2.4.4。

## Java的东西Groovy都能用，包括语法和类库

例如，新建一个SuperTest.groovy
输入一下Java代码：

```
public class SuperTest {

    public static void main(String[] args) {
        for (int i = 0; i < 3; i++) {
            System.out.println("Java的东西Groovy都能用");
        }
        System.out.println("随便输入点啥试试");
        Scanner sc = new Scanner(System.in); // 不需要导入java.util包
        String str = sc.next();
        System.out.println("你刚才输入了：" + str);
    }

}
```

运行结果：

```
Java的东西Groovy都能用
Java的东西Groovy都能用
Java的东西Groovy都能用
随便输入点啥试试
放开那条PM，让如花来
你刚才输入了：放开那条PM，让如花来
```

## 自动导入常用包

在上面的例子里使用了Scanner却不需要导入java.util包，是因为Groovy自动导入下列包:
java.lang
java.util
java.io
java.net
java.math.BigDecimal

java.math.BigInteger
groovy.lang
groovy.util

## Groovy特点

从以上Groovy可以执行Java的例子可以感受到到以下两个特点：

- Groovy继承了Java的所有东西，就是你突然忘了Groovy的语法可以写成Java代码，也就是Groovy和Java混在一起也能执行。
- Groovy和Java一样运行在JVM，源码都是先编译为class字节码。

除此之外，Groovy又对Java的语法进行了简化，功能进行了扩充。这个得学了具体语法才能感受到，不过可以先来感受下和Groovy相比Java有多啰嗦，就拿上面的循环三遍的例子来说，以下的代码实现了相同的效果：

```
for (i in 0..2) {
    println 'Java的东西Groovy都能用'
}
```

或者

```
3.times {
    println 'Java的东西Groovy都能用'
}
```

----------开始语法内容----------

## 关键字

<code>
as、assert
break
case、catch、class、const、continue
def、default、do
else、enum、extends
false、finally、for
goto
if、implements、import、in、instanceof、interface
new、null
package
return
super、switch
this、throw、throws、trait、true、try
while
</code>

## 语句不需要分号结尾

写了也没事

## 注释

```
// 单行注释
println "hello groovy," /* 我是块注释 */ + "my name is xuhongchuan."
```

```
hello groovy,my name is xuhongchuan.
```

## 标示符

和Java一样：
建议只用字母、数字、美元$和下划线组成。
以字母，美元符号$或者下划线开头，不能以数字开头。

## 定义变量

用def定义变量，不写def也行。
定义变量不用指定数据类型且可以随意更换。

```
def var1 = 1024
var2 = "def不写也行"

var1 = "Integer 改 String"
```

## 定义方法

定义方法也是用def当然也可以不写。

```
// 使用def
def String getName() {
    return "许宏川"
}

// 不写def
String getName() {
    return "许宏川"
}
```

Groovy所有的方法都有返回类型，如果不写则返回null，没有void。返回类型可以省略不写，不写的话自动取于最后一行代码的类型。
不写返回类型的方法就必须加上def。
另外return也可以省略不写，都是取最后一行。

```
def getName() {
    "许宏川"
    1234 // 这行是最后一行，返回Integer
}
```

方法参数类型可写可不写

```
// param1写了数据类型， 调用时必须传进来String参数
// param2没写数据类型，调用时可以传进来任意类型的参数
```

```
def getString(String param1, param2) {

}
```

## public**是默认的**

Groovy的类和方法的默认修饰符都是public，且可以省略不写。由于修饰符可以省略、方法返回类型可以省略、方法参数类型可以省略。所以Java的类和main方法的结构可以简化为：

```
class SuperTest {

    static main(args) {

    }

}
```

甚至也可以不写类和main结构，直接写main方法里的代码。编译成class文件时会自动给添加上。

## 字符串

分三种，单引号，双引号和三引号三种。

- 单引号是输入什么就是什么。
  例如：

```
println('my name is $ xuhongchuan')
```

打印结果为：

```
my name is $ xuhongchuan
```

$也正常打印出来了。

- 而双引号可以用$引用变量的值。
  例如

```
name = "xuhongchuan"
println("my name is $name")
```

打印结果为：

```
my name is xuhongchuan
```

- 三引号是输出一段文本，可以直接的加空格和换行。
  例如：

```
println('''这是一段文本。
换行啦！！！
    前面有四个空。。。
有换行啦！！！！''')
```

打印结果为：

```
这是一段文本。
换行啦！！！
    前面有四个空。。。
有换行啦！！！！
```

这个好啊，想想写sql语句时可自由换行有多清爽。

## 数据类型

分基本数据类型、容器和闭包三种。

## 基本数据类型

Groovy是纯面向对象的语言，没有Java里的byte、int、double、boolean等八个值类型。但是有对应的包装类如Integer、Double和Boolean。其中整数默认是Integer，浮点数默认是。。。你想说是Double？不，是BigDecimal。
如果想显式指定Long类型，在后面加L，Double则加D。

```
var = 5
println var.class

var = 5.5
println var.class

var = 5L
println var.class

var = 5D
println var.class

var = 'hehe'
println var.class

var = false
println var.class
```

输出结果为：

```
class java.lang.Integer
class java.math.BigDecimal
class java.lang.Long
class java.lang.Double
class java.lang.String
class java.lang.Boolean
```

## 容器类

分List、Map和Range。

- List

```
def demoList = [121, 3.14, 'hello', false, null] // 使用[]定义，元素之间用,隔开
println demoList.size // 获取集合大小
prinltn demoList[2] // 获取index为2的元素
// 在结尾添加元素的两种写法
demoList.add(100)
demoList << 100
//在指定位置添加元素，原本index大于等于3的元素往后退一位
demoList.add(3, 100)
demoList.remove(0) // 删除指定index的元素
demoList -= [3.14, false] // 删除某集合的元素
demoList.clear() // 清空集合
// 使用集合直接调用.each可以对集合进行遍历
demoList.each {
    println it // it是迭代过程中的每一个元素
}
```

- Map

  使用[key : value]定义，元素之间用,隔开。

  key必须是String，也可以不加引号自动转为String。

```
def demoMap = ['name' : '许宏川', 'age' : 18, 'isGay' : false]
println demoMap.size() // 获取map大小
println demoMap.name // 通过key获取值
demoMap << ['hehe' : '777'] // 添加元素
// 遍历map
demoMap.each {
    println it.key
    println it.value
}
```

- Range

```
// 范围从1到10
def demoRange = 1..10
// 范围从1到9
def demoRange2 = 1..<10
println(demoRange2.from) // 获取起始值
println(demoRange2.to) // 获取最大值
```

# 闭包

闭包是一段代码块，注意闭包也是数据类型，所以可以把闭包作为方法的参数或者返回类型。
如果我们要筛选指定数n范围内的奇数，普通写法如下：

```
def getOdd(n) {
    for (i in 1..n) {
        if (i % 2 != 0)
            println i
    }
}

getOdd(10)
```

如果要获取偶数，又要再写一个方法：

```
def getEven(n) {
    for (i in 1..n) {
        if (i % 2 == 0)
            println i
    }
}

getEven(10)
```

这两个方法其实for循环部分的内容是重合的。
而如果用闭包就不会这样了，例如下面的pick接受两个参数，一个参数n，另外一个是闭包（closure是变量名随便取）。再重复一遍闭包是一个代码块，这里传进来你想在遍历过程做什么。至于怎么把便利过程的i传递给闭包，闭包有一个隐式变量叫it，可以接收一个参数。
看代码：

```
def pick(n, closure) {
    for (i in 1..n) {
        closure(i)
    }
}

// 打印奇数
pick(10, {
    if (it % 2 != 0) // it代表传进来的参数，也就是上面closure(i)的i
            println it
})

// 打印偶数
pick(10, {
    if (it % 2 == 0)
        println it
})
```

总之循环结构不需要自己写了，你只需要写你想在遍历过程中做什么，例如如果要打印全部数的平方可以这样：

```
// 平方
pick(10, {
    println it **= 2
})
```

这个时候善于思考的同学就要问了，我还要自己写这些行为？
亲，这不就是动态的魅力么，谁知道你要在遍历过程做什么呢？但是如果有一些行为是经常用的，你也给闭包取个名字固定下来啊就像定义变量一样。
例如如果把刚才的的打印奇数、打印偶数和打印平方定义成变量可以改成这样：

```
def pick(n, closure) {
    for (i in 1..n) {
        closure(i)
    }
}

// 打印奇数
def getOdd = {
    if (it % 2 != 0)
        println it
}

// 打印偶数
def getEven = {
    if (it % 2 == 0)
```

```
        println it
}

// 打印平方
def getSquare = {
    println it **= 2
}

pick(10, getOdd)
pick(10, getEven)
pick(10, getSquare)
```

这个时候，善于思考的同学又要问了，隐式变量it只能代表一个参数吧？闭包怎么接收多个参数？
是这样的，用 -> 把参数列表和行为隔开即可。假设我们定义一个闭包接受两个参数求他们的和：

```
def getSum = {
    x, y -> println x + y
}

getSum(3, 4) // 闭包可以直接调用
```

关于闭包还有个说的，就是假设你的闭包不需要接收参数，但是还是会自动生成隐式it，只不过它的值为null。也就是说，闭包至少包含一个参数。