The Complete Apache Groovy Developer Course

Jacopo De Angelis

1 giugno 2021

Indice

Capitolo 1

Introduzione

1.1 groovysh

Groovysh è un'applicazione da linea di comando fornita tramite l'SDK. Per accedervi basta scrivere nel prompt "groovysh" e si attiverà la shell.

Essendo un linguaggio di scripting è possibile scrivere semplici comandi da fare valutare (e.g. 1+1, println "Hello, World!").

```
groovy:000> 1+1
===> 2

groovy:000> println "Hello"
Hello
===> null
```

Groovy riconosce anche quando una linea di comando non è terminata, attendendo la valutazione e segnalandolo tramite il numero a inizio linea.

```
groovy:000> class Person {
groovy:001> def sayHello(){
groovy:002> println "Hello"
groovy:003> }
groovy:004> }
```

```
===> true

groovy:000> person = new Person()
===> Person@2dbfcf7

groovy:000> person.sayHello()
Hello
===> null
```

1.2 groovyc

É il corrispettivo di javac, lo script viene tradotto in Java byte-code.

```
groovyc -help
Usage: groovyc [options] <source-files>
       [<source-files>...] The groovy source files to compile, or
           ↪ @-files
                                    containing a list of source files
                                        ↪ to compile
       -cp, -classpath, --classpath=<path>
                                   Specify where to find the class
                                       ↪ files - must be
                                     first argument
       -sourcepath, --sourcepath=<path>
                                   Specify where to find the source
                                       ↪ files
       --temp=<temp> Specify temporary directory
       --encoding=<encoding> Specify the encoding of the user
           ↪ class files
  -d=<dir> Specify where to place generated class files
  -e, --exception Print stack trace on error
       -pa, --parameters Generate metadata for reflection on
           ↪ method
                                    parameter names (jdk8+ only)
       -pr, --enable-preview Enable preview Java features (JEP
           ↪ 12) (jdk12+
                                    only) - must be after classpath
                                        ↪ but before other
```

```
                                    arguments
-j, --jointCompilation Attach javac compiler to compile .java
    ↪ files
-b, --basescript=<class> Base class name for scripts (must
    ↪ derive from
                                    Script)
-J=<property=value> Name-value pairs to pass to javac
-F=<flag> Passed to javac for joint compilation
    --indy Enables compilation using invokedynamic
    --configscript=<script>
                                A script for tweaking the
                                    ↪ configuration options
-h, --help Show this help message and exit
-v, --version Print version information and exit
    --compile-static Use CompileStatic
    --type-checked Use TypeChecked
```

I file possono essere successivamente eseguiti tramite "groovy [file.class]".

1.3 groovyConsole

Digitare nel prompt "groovyConsole" apre la console, ovvero un ambiente di sviluppo base dove scrivere e testare i nostri script.

1.4 Alcune note per passare da Java a Groovy

- return non serve in quanto verrà automaticamente restituito l'ultimo output del metodo
- public non serve per gli oggetti
- ; non servono a meno di seperare due comandi sulla stessa linea
- le proprietà di un oggetto sono private di default
- getter e setter non servono

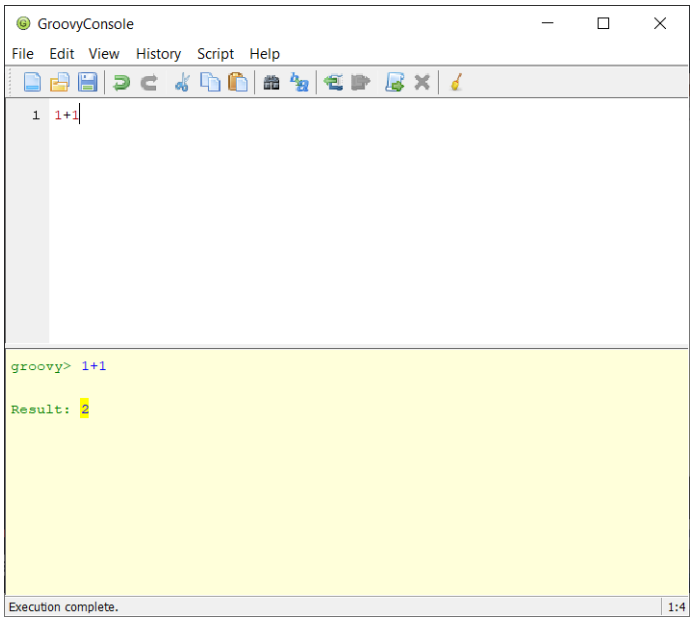

Figura 1.1: Groovy console

- println non necessita dei System.out. prima, è un metodo di default, le parentesi sono superflue
- @groovy.transform.toString() usato come annotazione sulla classe crea automaticamente la redifinizione del metodo
- i costruttori sono superflui in quanto posso costruire manualmente l'istanza (e.g. "new User(firstName:"nome", lastname:"cognome")"

Capitolo 2

Le basi

2.1 Import di default

```
import java.lang.*
import java.util.*
import java.io.*
import java.net.*
import groovy.lang.*
import groovy.util.*
import java.math.BigInteger
import java.math.BigDecimal
```

Questi import automatici riducono il boilerplate. Nel caso siano richiesti altri import è uguale a Java.

2.2 Assertions

assert funziona per valutare se l'espressione successiva sia vera.

2.3 Numeri

Groovy fa un boxing automatico dei numeri nelle loro versioni oggetto (e.g. int -> Integer)

2.4 Control structure

swtich, if e while funzionano allo stesso modo di Java eccetto che per le stringhe true vale solo se la stringa contiene un valore, la stringa vuota è false.

2.4.1 for

I for possono avere la forma iterativa

```
for(x in collection){
  ...
}
```

oppure la funzione .each.

2.5 Annotazioni e AST transformation

Prima di tutto bisogna importare il package groovy.transform.*.

Le annotazioni funzionano tramite "@codiceDell'implementazione", ad esempio:

```
import groovy.transform.Immutable

@Immutable
class Customer {
  String first, last
  int age
  Date since
  Collection favItems
```

```
}
```

In questo caso la classe sarà in sola lettura.

2.6 Operatori

```
assert 1 + 2 == 3
assert 4 - 3 == 1
assert 3 * 5 == 15
assert 3 / 2 == 1.5
assert 10 % 3 == 1
assert 2 ** 3 == 8
assert 2++ == 3
assert 3-- == 2
```

Abbiamo anche gli operatori di assegnamento

```
def a = 4
a += 3

assert a == 7

def b = 5
b -= 3

assert b == 2

def c = 5
c *= 3

assert c == 15

def d = 10
d /= 2

assert d == 5

def e = 10
e %= 3
```

```
assert e == 1

def f = 3
f **= 2

assert f == 9
```

In groovy sono stati aggiunti === e !== che implicano identità, quindi non uguali in contenuto ma uguali in identità.

```
@EqualsAndHashCode
class Creature { String type }

def cat = new Creature(type: 'cat')
def copyCat = cat
def lion = new Creature(type: 'cat')

assert cat.equals(lion) // Java logical equality
assert cat == lion // Groovy shorthand operator

assert cat.is(copyCat) // Groovy identity
assert cat === copyCat // operator shorthand
assert cat !== lion // negated operator shorthand
```

2.6.1 Elvis operator

Novità è l'elvis operator, ovvero un'abbreviazione dell'operatore ternario

```
import groovy.transform.ToString

displayName = user.name ? user.name : 'Anonymous'
displayName = user.name ?: 'Anonymous'
```

Un esempio di uso

```
@ToString
class Element {
```

```
    String name
    int atomicNumber
}

def he = new Element(name: 'Helium')
he.with {
    name = name ?: 'Hydrogen' // existing Elvis operator
    atomicNumber ?= 2 // new Elvis assignment shorthand
}
assert he.toString() == 'Element(Helium, 2)'
```

2.6.2 Safe navigation

Altro operatore utile è quello di safe navigation, il quale accede al campo se e solo se questo non è null.

```
def person = Person.find { it.id == 123 }
def name = person?.name
assert name == null
```

2.7 Grave

É un tool di dependecy management, può essere usato in forma estesa o in forma contratta:

```
@Grab(group='org.springframework', module='spring-orm',
    ↪ version='3.2.5.RELEASE')
import org.springframework.jdbc.core.JdbcTemplate

@Grab('org.springframework:spring-orm:3.2.5.RELEASE')
import org.springframework.jdbc.core.JdbcTemplate
```

In questo modo non ci serve avere il jar per passare un file, verrà automaticamente risolto.

Capitolo 3

Tipi

var : Java = def : groovy

Solo che def è totalmente dinamico, non dobbiamo tenere lo stesos tipo di variabile, ”def” implica che può contenere una qualsiasi cosa, quindi potremmo dire

```
def x = 10
x = ”stringa”
```

3.1 times, upto, downto, step

times è un metodo di Integer che permette di eseguire una funzione n volte

```
20.times {
  ...
}
```

upto funziona da x a y

```
1.upto(10) {
  ...
}
```

downto è l'esatto contrario

```
10.downto(1) {
  ...
}
```

step è come upto ma permette di stabilire l'incremento, ad esempio qua farà 0, 0.1, 0.2,...1

```
0.step(1, 0.1) {
  ...
}
```

3.2 Redifinizione degli operatori

Una cosa utile è ridefinire le operazioni base in modo da potere prescrivere comportamenti specifici per nuove classi. Ad esempio, se volessimo ridefinire il + potremmo scrivere

```
[oggetto] plus([altro oggetto]){
  ...
}
```

Qua possiamo vedere la lista delle operazioni.

3.3 Stringhe

Cosa da ricordare, se vogliamo creare una stringa multilinea bisogna rispettare un passo ulteriore, ovvero l'utilizzo del triplo apice/doppio apice:

```
def aLongMessage = '''
Questo
è
un
messaggio
multilinea
'''
```

3.3.1 $

Le stringhe possono contenere caratteri speciali, ad esempio in quel caso si possono ”escapare” singolarmente o in blocco, ad esempio:

```
”c:\\cartella\\interna”

// oppure

$c:\cartella\interna$
```

3.4 Regex

Funzionano come in java ma vengono dichiarate diversamente, ovvero tramite l'operatore .

```
def pattern = ~[regex]

def finder = text ~= pattern
def matcher = text ~== pattern
```

La differenza tra find (=) e match (==) è che find restituisce una lista di ritrovamenti, match restituisce un boolean che rappresenta se la stringa è generata dalla regex o no.

Un uso particolare è quello

String.replaceFirst(regex, newString) che permette di sostituire la prima istanza che viene trovata dalla regex con newString.

Capitolo 4

Materiali utili

Documentazione

- Groovy in action
- Making Java Groovy
- Programming Groovy 2