Groovy

Victor Herrero Cazurro

Table of Contents

1. ¿Que es Groovy?

Es un lenguaje basado en Java, que persigue la simplificación de las sentencias **Java** en lo estrictamente necesario.

Tambien es un lenguaje dinámico, ya que se pueden ejecutar sentencias directamente sobre una consola **groovycosole** o una shell **groovysh**.

Casi el 100% del código **Java** es compilable con **Groovy**, por lo que el salto de un lenguaje a otro se puede dar de forma paulatina.

1.1. Caracteristicas Básicas

El punto y coma (;) es opcional como delimitador de sentencias, el salto de linea es suficiente, solo será necesario cuando se introduzcan varias sentencias en una unica linea.

Los parentesis para invocar un método tambien son opcionales

```
void saludar(String nombre) {
    println "Hola " + nombre + "!!!!"
}

saludar "Victor"
```

No es necesaria la declaración de los tipos de las variables en los scripts, pudiendose emplear una misma variable para referenciar a objetos de distinto tipo, ya que no estan tipadas (tipado debil).

```
a = 1

a = "Hola"
```

En las clases si es necesario declarar las variables, aunque no su tipo, ya que se puede emplear la palabra reservada **def**, como tipo generico.

```
class saludador {
    def prefijo = "Hola "
    def sufijo = "!!!!"
    void saludar (Stringn nombre){
        prefijo + nombre + sufijo
    }
}
```

Al definir los métodos, tambien se puede emplear **def** para los tipos retornados.

```
class saludador {
    def saludar (Stringn nombre){
        prefijo + nombre + sufijo
    }
}
```

El modificador por defecto para atributos de clases es **private** y para métodos **public**.

La sentencia de **return** tambien es opcional, ya que de no indicarse, se retorna el resultado de la ejecución de la ultima sentencia.

```
int sumar(int a, int b) {
    a + b
}

int resultado = sumar 1, 2
```

Se pueden definir String multilinea

```
String multilinea = """Un string de
    varias líneas
"""
```

Se permite la definicion de rangos numericos de forma reducida, con el operador **..**

```
def x = 0
for (i in 0 .. 10) {
    x += i
}
```

Groovy importa por defecto las siguientes clases y paquetes

- groovy.lang
- groovy.util
- java.lang
- java.util
- java.net
- java.io
- java.math.BigInteger
- java.math.BigDecimal

1.2. GStrings

Tambien conocidos como string interpolados, permiten definir **Strings** empleando variables Groovy.

Se han de definir entre comillas dobles.

```
println "El precio de la acción $accion a día $fecha es $valor"
```

1.3. Expresiones regulares

Basta con declarar la expresión de búsqueda entre barras diagonales, como en Perl, y usar uno de los tres operadores disponibles: * =~ → búsqueda de ocurrencias (produce un java.util.regex.Matcher) * ==~ → coincidencias (produce un Boolean) * ~ → patrón

La siguiente asercion es cierta, ya que el texto contiene **ua**

```
def texto = "Groovy es un lenguaje dinámico"

assert texto =~ /ua/
```

1.4. Json

El empleo de Json es tambien muy sencillo, ya que se proporciona un API para procesar objetos Json.

Para generar un json como el siguiente

```
{
  "cliente": {
    "nombre": "Victor",
    "apellido": "Herrero",
    "direccion": {
      "ciudad": "Valladolid",
      "pais": "España"
    },
    "facturas": [
      1234,
      1235,
      1236
    ]
  }
```

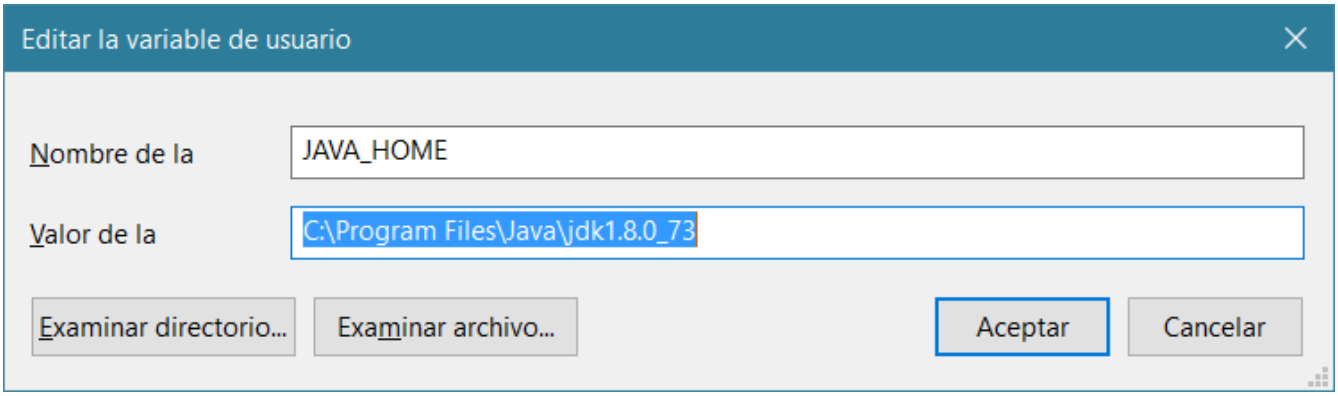
en Groovy unicamente debemos hacer

```groovy
import groovy.json.JsonBuilder

def builder = new JsonBuilder()
builder.
    cliente {
        nombre 'Victor'
        apellido 'Herrero'
        direccion(
            ciudad: 'Valladolid',
            pais: 'España'
        )
        facturas(
            1234,
            1235,
            1236
        )
    }

println builder.toPrettyString()
```

2. Instalación

Groovy esta basado en la maquina virtual de Java, por lo que hay que tener instalada la JDK para poder desarrollar con **Groovy**, se puede descargar desde aquí

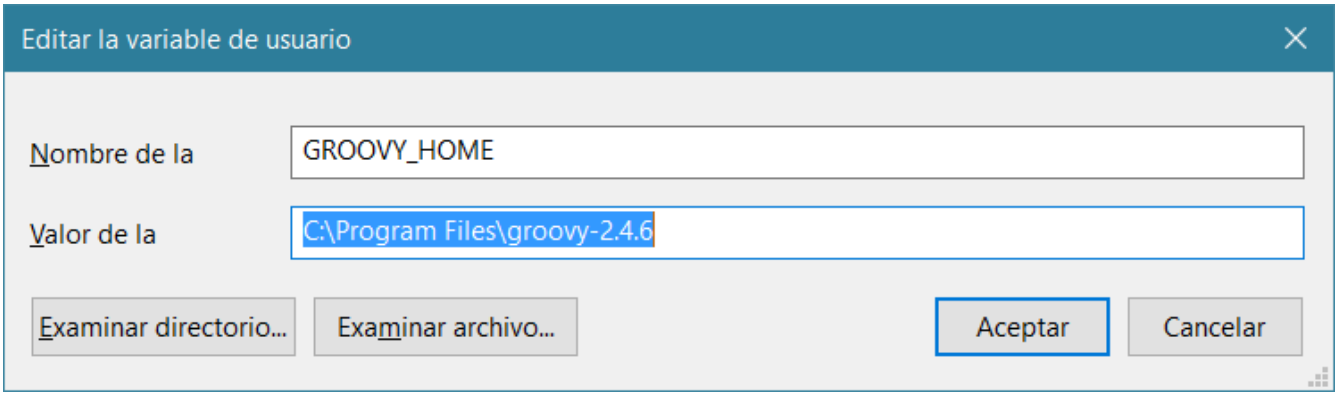
Además se ha de declarar la variable de entorno JAVA_HOME, que apuntará al directorio de la JDK.



Es recomendable hacer referencia en la variable de entorno PATH al direcotrio bin de la JDK.

De forma analoga, á que descargar **Groovy** e instalarlo, se puede descargar desde aquí

Además de definir la variable de entorno GROOVY_HOME, que apuntará al directorio instalación de Groovy.



De nuevo es recomendable hacer referencia en la variable de entorno PATH al direcotrio bin de **Groovy**.

Una de las herramientas que se obtienen, es la consola de Groovy, que se ejecuta con el comando **groovyconsole**

Como herramienta de desarrollo **IDE**, se pueden emplear varias,

- Eclipse + Plugin Groovy
- Intellij
- GTS Groovy Tool Suite
- Netbeans

2.1. Groovy Web Console

Existe una consola Web que se puede emplear para probar el codigo, sin necesidad de instalar nada, se accede aquí

3. POGO

Los equivalentes en groovy a los POJO de Java, siguiendo las siguientes reglas:

- Si la variable es declarada con un modificador de acceso (public, private o protected), entonces se genera unicamente el campo.
- Si es declarada sin modificador de acceso genera un campo privado con un getter y un setter públicos.
- Si es declarada como final, entonces el campo privado es creado como final y no se genera un setter.
- Se pueden redefinir los getter y setter.

Definicion de POGO

```
class Persona {
    String nombre
    int edad
}
```

Acceso a los Getter y Setter y del POGO

```
persona.nombre = 'Victor' // Equivalente a persona.setNombre('Victor')
```

Con la anotacion **@Canonical**, se obtienen implementaciones para equals, hascode y tostring, ademas de constructores con los atributos.

4. Operador safe navigator

Este operador **?**, permite evitar generar codigo para la validacion de variables distintas de null

En java para poder acceder de forma segura al nombre de un cliente asociado a un pedido, se debe hacer....

```
if (pedido != null) {
    if (pedido.getCliente() != null) {
        if (pedido.getCliente().getNombre() != null) {
            System.out.println(pedido.getCliente().getNombre());
        }
    }
}
```

Em Groovy unicamente

```
println pedido?.cliente?.nombre
```

El operador en caso de que las variables empleadas sean null, hace que la expresion retorne null

5. Operador ternario y operador Elvis

Se acepta el formato tradicional del operador ternario

```
println usuario.nombre ? usuario.nombre : 'Desconocido'
```

Y en el caso de querer el dato evaluado como resultado de la expresión, se acepta la expresion reducida conocida como **Elvis**

```
println usuario.nombre ?: 'Desconocido'
```

6. Listas

Se pueden definir listas de forma muy rapida, sin necesidad de hacer uso de la clase **Collections**

```
def lista = [1, 2, 3, 4] // ArrayList

List<Integer> lista = [1, 2, 3, 4]
```

Se pueden agregar elementos de forma tradicional o con el operador <<, este operador retorna el objeto lista.

```
lista.add("Grails")

lista << "Grails"

lista.addAll(["Grails","Griffon"])
```

Se pueden eliminar elementos de la lista de forma tradicional o con el operador -

```
lista.remove("Griffon")

lista = lista - 'Grails'
```

Se ofrecen métodos para iterar los elementos de las listas directamente

```
//Iterar por elemento
lista.each { println "Technology: $it"}

//Iterar por elemento e indice
lista.eachWithIndex { it, i -> println "$i: $it"}
```

Y métodos para buscar, de nuevo de forma tradicional y reducida

```
contained = lista.contains( 'Groovy' )

contained = 'Groovy' in lista

lista.containsAll(['Groovy','Grails'])
```

Ordenacion de elementos en las listas, modificando la lista original o proporcionando otro objeto

```
//Modifica la lista original
lista.sort()

//Obtiene otro objeto con la ordenación
sortedlista = lista.sort( false )
```

7. Mapas

Igual que con las listas, tambien se permite la construccion de mapas de forma sencilla.

```
def mapa = [nombre: 'Victor', edad: 22] // HashMap no tipado de 2 elementos, donde la clave es un string

println mapa.nombre // mapa.get("nombre");

mapa.edad = 37 // mapa.put("edad", 37);
```

Para iterar por elemento y tambien por elemento y su posicion en el mapa

```
// Iterar por elemento
mapa.each { println "$it.key: $it.value" }

// Iterar por eleemnto y posicion
mapa.eachWithIndex { it, i -> println "$i: $it"}
```

8. Closures

Permiten la definicion de un puntero a un método, que es invocable

Definicon de Closure

```
def clos = { println "Hola Mundo!" }
```

Invocacion de Closure

```
clos()
```

Se pueden definir con parametros

```
def clos = { b -> println "Hola $b" }
clos('Victor')
```

Las Closures pueden referenciar a variables fuera del método que definen

```
def prefijo = 'Hola '
def saludar = { nombre -> prefijo + nombre }
println saludar 'Victor'
```

9. Excepciones

En **Groovy** todas las excepciones son **Unchecked**.

Codigo que en java exigiria hacer algo con la excepcion que en groovy no.

```
File file = new File("E://file.txt")
FileReader fr = new FileReader(file)
```