



Introduction au langage de script Groovy

par Eric Reboisson (<http://ericreboisson.developpez.com>) (Blog)

Date de publication : 22/10/2006








Dernière mise à jour : 22/10/2006



Cet article est une introduction à Groovy, un langage de script pour Java inspiré entre autres de  **Python**,  **Java**, Ruby et Smalltalk

- I - Introduction
- II - Installation
- III - Premiers pas, premiers scripts
 - II-A - Les variables
 - II-B - Les listes et les maps
 - II-C - Les tests conditionnels
 - II-D - Les closures
- IV - Les expressions régulières
- V - Le SQL avec Groovy
- VI - Intégrer Groovy dans du code Java
- VII - Quelques différences avec Java
- VIII - Conclusion
- IX - Liens
- X - Remerciements


I - Introduction

Initié en 2003 par James Strachan et Bob McWhirter, le projet Open Source Groovy est un langage de script dont les caractéristiques principales sont :

- C'est un langage orienté objet pour la  **JVM**  **Java**
- Groovy s'inspire entre autres de  **Python**,  **Java**, Ruby et Smalltalk
- Un langage dynamique et agile (ex: typage dynamique, facilités de codage : point virgule non obligatoire en fin de ligne)
- Une syntaxe proche de  **Java**
- Groovy génère directement du  **bytecode**
- Réutilisation des bibliothèques  **Java**

Groovy est aujourd'hui géré au sein du  **JCP** dans la  **JSR 241**, ce projet étant dirigé par le français Guillaume Laforge.

Alors Groovy, me direz vous, pour quelles utilisations ? Eh bien elles sont nombreuses :

- Le prototypage (ou développement rapide)
- Les fonctionnalités "clef en main" de Groovy pour manipuler du  **XML**, des collections, etc.
- Les tests
- Applications financières
- Etc.

Voilà, maintenant le décor est planté, c'est parti pour un voyage découverte de Groovy...

II - Installation

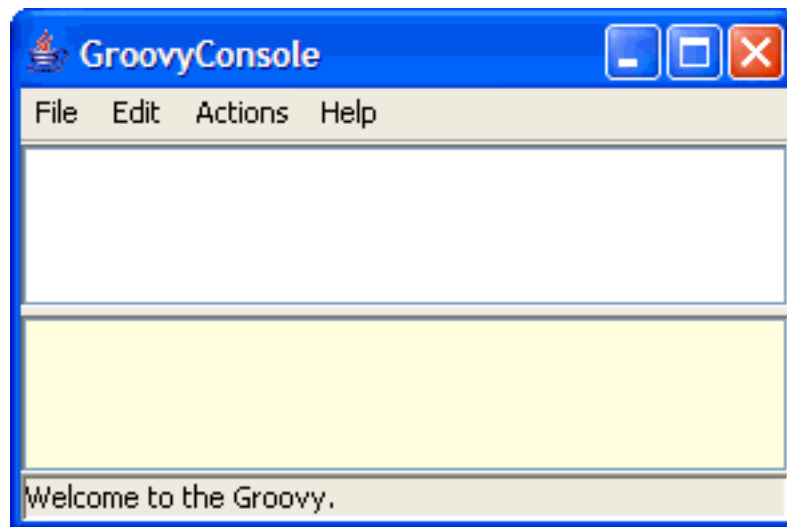
La page de téléchargement  <http://groovy.codehaus.org/Download> propose divers liens pour obtenir Groovy.

Les exemples de cet article utilisent la dernière version **groovy-x.x-jsr-xx.zip** de Groovy disponible à l'adresse suivante  <http://dist.codehaus.org/groovy/distributions>.

Extraire ensuite l'archive **groovy-x.x-jsr-xx.zip** dans le répertoire de votre choix et créer une variable d'environnement `GROOVY_HOME` avec pour valeur ce répertoire (ex: `GROOVY_HOME=c:/groovy-x.x-jsr-xx`).

Ajouter ensuite `%GROOVY_HOME%/bin` au `PATH`.

Ouvrir ensuite une console de commande et exécuter la commande `groovyConsole.bat`, la console de Groovy doit apparaître comme ceci :



III - Premiers pas, premiers scripts

Une fois la console ouverte, dans la partie haute taper l'instruction suivante puis la combinaison *CTRL + R* pour exécuter :

Le script tradition

```
println "Bonjour le monde!"
```

Vous observez alors la sortie suivante dans la partie basse de la console :

```
groovy> println "Bonjour le monde!"  
Bonjour le monde!
```

Voilà pour notre premier script, vous pouvez également demander l'évaluation d'un calcul, dans la partie haute taper :

Un calcul

```
12*12+6
```

Et vous obtiendrez (après avoir utilisé la combinaison *CTRL + R*, mais ça je ne le dirai plus):

```
groovy> 12*12+6  
Result: 150
```

II-A - Les variables

Exécutons maintenant ce script :

Concaténation de chaînes

```
age = 77  
phrase = "Le capitaine a " + age + " ans"  
println phrase
```

Le résultat est le suivant :

```
groovy> age = 77  
groovy> phrase = "Le capitaine a " + age + " ans"  
groovy> println phrase  
Le capitaine a 77 ans
```

Nous avons donc créé une variable *age*, puis une variable *phrase* et affiché une concaténation de ces deux variables.

Vous remarquerez qu'aucun type n'a été fixé pour ces variables, c'est une des fonctionnalités de Groovy : le typage dynamique.

On type en général les variables par *def* si on désire un typage dynamique (vérification à l'exécution comme en *smalltalk*), si on omet le mot clé *def* il est utilisé par défaut :



Typage dynamique

```
def film = "Mon nom est personne"
def nbvision = 7
println "J'ai vu $nbvision fois le film $film"
```

Donnera en sortie de console :

```
groovy> def film = "Mon nom est personne"
groovy> def nbvision = 7
groovy> println "J'ai vu $nbvision fois le film $film"

J'ai vu 7 fois le film Mon nom est personne
```

Mais on peut également typer les variables (comme en  **Java**) si on désire un  **typage fort** :

Typage fort

```
String film = "Mon nom est personne"
int nbvision = 7
println "J'ai vu $nbvision fois le film $film"
```

Donnera en sortie de console :

```
groovy> String film = "Mon nom est personne"
groovy> int nbvision = 7
groovy> println "J'ai vu $nbvision fois le film $film"

J'ai vu 7 fois le film Mon nom est personne
```

II-B - Les listes et les maps

Dans Groovy, les listes sont le pendant des interfaces *java.util.List* en Java.

Une liste se déclare comme ceci :

La banane par les deux bouts

```
liste = ['banane', 'poire', 'pomme', 'banane']
```

On peut par exemple utiliser la taille de la liste, accéder à un élément particulier, ce qui donne en sortie de console :

```
groovy> liste = ['banane', 'poire', 'pomme', 'banane']
groovy> println liste.size()
groovy> println liste[2]

4
pomme
```

Les maps sont des listes d'éléments associés à une clé. On peut alors accéder à un élément particulier, non pas par son index, mais par sa clé associée.

Une map se déclare comme ceci :

Utile lors des longs voyages

```
departement = [ "Ain":01, "Vosges":"88", "Rhône Alpes":69 ]
```

Et comme pour les listes, des méthodes pour travailler avec les maps :

Deux façons d'accéder aux valeurs d'une map

```
groovy> departement = [ "Ain":01, "Vosges":"88", "Rhône Alpes":69 ]
groovy> println departement.Vosges
groovy> println departement["Rhône Alpes"]

88
69
```

Vous remarquerez encore une fois que le type des valeurs de la map est hétérogène, et Groovy se débrouille très bien avec.

Pour modifier une valeur de la map :

Modifier une valeur de la map

```
groovy> departement = [ "Ain":01, "Vosges":"88", "Rhône Alpes":69 ]
groovy> departement["Vosges"] = 666
groovy> println departement["Vosges"]

666
```

Pour parcourir une liste c'est extrêmement simpliste, en reprenant l'exemple de nos fruits :

Affichage de tous les éléments d'une liste

```
liste = ['banane', 'poire', 'pomme', 'banane']
liste.each {
    println it
}
```

Donnera en sortie de console :

```
groovy> liste = ['banane', 'poire', 'pomme', 'banane']
groovy> liste.each {
groovy>     println it
groovy> }
```



```
banane
poire
pomme
banane
```

Enfin pour créer des listes vides et ensuite y ajouter des éléments :

Liste et map vide, ajouter un élément

```
mamap = [:]
maliste = []
mamap.put("TOTO", 44)
maliste.add("TITI")
println mamap["TOTO"]
println maliste[0]
```

Une autre variante pour peupler les listes et les maps :

Autres exemples d'ajout d'éléments

```
mamap = [:]
maliste = []
mamap["TOTO"] = 44
maliste << "TITI"
maliste + "TOTO"
```

II-C - Les tests conditionnels

Comme dans beaucoup de langages, Groovy permet de contrôler le flux d'un programme :

Un exemple avec le if

```
variable1 = "Homme"
variable2 = 12

if (variable1 == "Homme" && variable2 > 6 )
println "Hello, tu es rentré dans le test!"
```

Donnera en sortie de console :

```
groovy> variable1 = "Homme"
groovy> variable2 = 12
groovy> if (variable1 == "Homme" && variable2 > 6 )
groovy> println "Hello, tu es rentré dans le test!"

Hello, tu es rentré dans le test!
```

On retrouvera également les opérateurs habituels pour combiner des tests booléens :

- == : égalité
- != : différence
- > : supérieur strict
- >= : supérieur ou égal
- < : inférieur strict
- <= : inférieur ou égal

Finalement, c'est très proche de  **Java**, mais vous étiez prévenus en introduction.

II-D - Les closures

Groovy est capable d'interpréter du code contenu entre les caractères { et }, d'en faire le corps d'une méthode, et ceci sans avoir à créer une classe : c'est une fonctionnalité appelée *closure*.

La syntaxe d'une closure est la suivante :

```
{ [closureArguments->] statements }
```

Aussi, en utilisant le code suivant :

Créer une méthode avec les closures

```
helloWorld = {String nom, String profession -> println "Hello $nom, laisse moi regarder ma boule de cristal...tu es $profession"}  
helloWorld("eric", "informaticien")
```

On obtiendra en sortie de console :

```
groovy> helloWorld = {String nom, String profession -> println "Hello $nom, laisse moi regarde ma boule de cristal...tu es $profession"}  
groovy> helloWorld("eric", "informaticien")  
  
Hello eric, laisse moi regarder ma boule de cristal...tu es informaticien
```

Les possibilités des *closures* sont très étendues et mériteraient à elles seules un article, mais puisque vous le demandez si fort quelques exemples en plus à essayer :

Quelques closures en plus

```
affiche = {value -> println(value);}   
affiche(['a', 'b', 'c'])
```


Quelques closures en plus

```
def c = { arg1, arg2-> println "$arg1 $arg2" }  
def d = c.curry("Ici")  
d("Zebra 3")
```

Quelques closures en plus(je vous laisse imaginer le code suivant en Java)

```
trouvepetit = {liste-> liste.findAll { it.size() <= 4 }.each { println it } }  
trouvepetit(["Eric", "Jean-Claude", "Paul-Edouard", "Luc"])
```

IV - Les expressions régulières

Groovy supporte nativement les expressions régulières en utilisant l'expression `~"..."`, un objet  **Java** *Pattern* est alors compilé.

Quelques exemples d'utilisation :

Tester la présence d'une chaîne

```
assert "Introduction au langage Groovy" =~ "langage"
```

Remplacement de chaîne

```
def phrase = ("Introduction au langage Groovy" =~ /Introduction au/).replaceFirst("Tutoriel sur  
le")  
println phrase
```

Trouver un texte



```
def liste = "tokenUN TEXTE A TROUVERToken" =~ /token(.*)token/  
println liste[0][1]
```

Pour en savoir plus sur la syntaxe des expressions régulières,  [voir ici](#).

V - Le SQL avec Groovy

Pas de limites, Groovy est capable d'interagir avec votre  **SGBD** via un pilote  **JDBC** grâce à son module GSQL.

L'exemple suivant fonctionnera si une base *MySql* est installée localement, avec un compte dont les identifiants sont ROOT/ROOT (pour login/password).

Vous devrez également copier un pilote  **JDBC** dans le répertoire *lib* de l'installation de Groovy (On trouve ce genre de pilote dans toutes les bonnes crémèries, mais aussi sur la page suivante  **Download Connector/J 5.0**).

Un petit exemple ? Créons une table dans notre base de données :

Création d'une table

```
import groovy.sql.Sql

def sql = Sql.newInstance("jdbc:mysql://localhost:3306/test", "root",
                        "root", "com.mysql.jdbc.Driver")

sql.execute("create table DEPARTEMENT (CODE VARCHAR(3), LIBELLE VARCHAR(100))")
```

On va maintenant peupler cette table *DEPARTEMENT* :

Insertion de données

```
import groovy.sql.Sql

def sql = Sql.newInstance("jdbc:mysql://localhost:3306/test", "root",
                        "root", "com.mysql.jdbc.Driver")

sql.execute("insert into DEPARTEMENT values ('88','Vosges')")
sql.execute("insert into DEPARTEMENT values ('69','Rhone Alpes')")
```

Et enfin, parcourir les données qu'elle contient :

Sélection de données

```
import groovy.sql.Sql

def sql = Sql.newInstance("jdbc:mysql://localhost:3306/test", "root",
                        "root", "com.mysql.jdbc.Driver")

sql.eachRow("select * from DEPARTEMENT") {
    println "Département = ${it.LIBELLE}"
}
```

Le dernier script donnera alors en sortie de console :

```
groovy> import groovy.sql.Sql
groovy> def sql = Sql.newInstance("jdbc:mysql://localhost:3306/test", "root",
groovy>                                "root", "com.mysql.jdbc.Driver")
groovy> sql.eachRow("select * from DEPARTEMENT") {
```

```
groovy>      println "Département = ${it.LIBELLE}"
groovy> }
```



```
Département = Vosges
Département = Rhone Alpes
```

VI - Intégrer Groovy dans du code Java

Avec Groovy, un script peut être utilisé via un code Java qui pourra interagir avec ce script.

Un petit scénario : vous avez une application qui calcule une rentabilité, ce calcul est inconnu des personnes qui devraient le spécifier pour le moment, et ce que vous savez c'est qu'il sera amené à être modifié par la suite (ça vous est forcément arrivé un jour ce scénario ?).

Dans ce cas là Groovy va grandement vous aider à réaliser ce système de fonctionnement.

Un peu de pratique, nous avons ce script Groovy tout simple enregistré dans un fichier:

Contenu du fichier exemple1.gy (dans les sources en téléchargement)

```
println('Hello World : ' + argument)
return "Valeur de retour"
```

Et la classe suivante qui chargera le script, lui passera un paramètre et récupèrera sa valeur de retour :

Utiliser un script Groovy depuis Java

```
import groovy.lang.Binding;
import groovy.lang.GroovyShell;
import groovy.lang.Script;

import java.io.File;
import java.io.IOException;

import org.codehaus.groovy.control.CompilationFailedException;

public class MaClasseGroovy {

    public static void main(String[] args) {

        GroovyShell shell = new GroovyShell();
        Script script;
        try {
            // Chargement du script groovy
            script = shell.parse(new File("scripts/exemple1.gy"));
            Binding binding = new Binding();
            // Création d'un paramètre
            binding.setVariable("argument", "Saint Nicolas");
            script.setBinding(binding);

            // Exécution du script
            Object retour = script.run();
            // Affichage de la valeur de retour du script
            System.out.println(retour);

        } catch (CompilationFailedException e) {

            e.printStackTrace();
        } catch (IOException e) {

            e.printStackTrace();
        }
    }
}
```

Utiliser un script Groovy depuis Java

```
}  
}
```


En exécutant cette classe Java, la console affichera :



```
Hello World : Saint Nicolas  
Valeur de retour
```

Magnifique, non ?

Les sources de cet exemple sont disponibles [ici](#) sous la forme d'un projet pour  **Eclipse**.

VII - Quelques différences avec Java

Les aficionados de  **Java** remarqueront vite quelques différences entre la syntaxe utilisée par Groovy et leurs habitudes :

- Le point virgule est optionnel en fin de ligne
- Le typage des variables peut être dynamique ou statique (Nous l'avons vu dans certains exemples précédents), pour rappel là où en  **Java** nous aurions `String maChaine = "Mon texte"`; Groovy permet `maChaine = "Mon texte"`
- Le contenu d'un tableau en Groovy se déclare avec des crochets (ex: `int[] tab = [1,2,3]`) et en  **Java** avec des accolades (ex: `int[] tab = {1,2,3}`)
- Une boucle `for` en Groovy s'écrit `for (i in 0..len-1)`
- Les classes internes ne sont pas encore supportées en Groovy, les closures s'y substituent


De nombreuses différences existent encore, je vous invite à consulter la page suivante pour en savoir plus :



 **Differences from Java**

VIII - Conclusion

Voilà, une petite mise en bouche à Groovy qui peut-être trouvera son utilité dans un de vos projets.

Vous vous apercevrez vite que les possibilités de Groovy sont très étendues, pour exemple les fonctionnalités suivantes :

- Expressions régulières
- Gestion des langages de balises avec *GroovyMarkup*
- Le requêtage XML avec *GPath*
- Les servlets avec *Groovlets*
- **Grails Object Relational Mapping** (GORM) : un module de persistance
- GSP pour GroovyServer Pages : similaire aux pages  **JSP**
- Etc.Etc.(pour toutes celles que vous trouverez sur le site officiel de Groovy)


Vous comprendrez rapidement combien Groovy mérite que l'on s'y intéresse, ce n'est pas pour rien qu'il est discuté au  **JCP** dans une  **JSR 241: The Groovy Programming Language**

IX - Liens

Les sources de l'exemple  **Java** sont disponibles **ici** sous la forme d'un projet pour  **Eclipse**.

 <http://groovy.codehaus.org> : Le site officiel de Groovy

 **JSR 241: The Groovy Programming Language**

Vous avez une IDE Java de prédilection ? vous trouverez sûrement son plugin Groovy ici :  **IDE Support**

X - Remerciements

Un grand merci à Guillaume Laforge, **Denis Cabasson**, **Christophe Jollivet**, Xavier Mehaut et **Yann Marec** pour la relecture de cet article et leurs bons conseils.

