eBook Gratuit

APPRENEZ groovy

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#groovy

Table des matières

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: groovy

It is an unofficial and free groovy ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official groovy.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Commencer avec groovy

Remarques

Groovy est

- est un langage dynamique éventuellement typé pour la machine virtuelle Java
- s'appuie sur les atouts de Java mais possède des fonctionnalités supplémentaires inspirées de langages tels que Python, Ruby et Smalltalk
- met à la disposition des développeurs Java des fonctionnalités de programmation modernes avec une courbe d'apprentissage presque nulle
- offre la possibilité de taper de manière statique et de compiler statiquement votre code pour la robustesse et la performance
- prend en charge les langages spécifiques à un domaine et toute autre syntaxe compacte pour que votre code soit facile à lire et à gérer
- Facilite l'écriture de scripts de shell et de build avec ses puissantes primitives de traitement, ses capacités OO et son DSL Ant
- Augmente la productivité des développeurs en réduisant le code d'échafaudage lors du développement d'applications Web, d'interface graphique, de base de données ou de console
- simplifie les tests en prenant en charge les tests unitaires et les moqueries
- s'intègre parfaitement à toutes les classes et bibliothèques Java existantes
- compile directement en Java bytecode pour que vous puissiez l'utiliser partout où vous utilisez Java

Versions

Version	Notes de version	Date de sortie
2.4	http://groovy-lang.org/releasenotes/groovy-2.4.html	2015-01-21
2.3	http://groovy-lang.org/releasenotes/groovy-2.3.html	2014-05-05
2.2	http://groovy-lang.org/releasenotes/groovy-2.2.html	2013-11-18
2.1	http://groovy-lang.org/releasenotes/groovy-2.1.html	2013-01-24
2.0	http://groovy-lang.org/releasenotes/groovy-2.0.html	2012-06-28

Version	Notes de version	Date de sortie
1.8	http://groovy-lang.org/releasenotes/groovy-1.8.html	2011-04-27
1,7	http://groovy-lang.org/releasenotes/groovy-1.7.html	2009-12-22
1.6	http://groovy-lang.org/releasenotes/groovy-1.6.html	2009-02-18
1,5	http://groovy-lang.org/releasenotes/groovy-1.5.html	2007-12-07
1.0		2007-01-02

Examples

Installation ou configuration

Il existe deux méthodes courantes pour installer Groovy.

Télécharger

Le binaire Groovy peut être téléchargé sur la page de téléchargement du site Groovy. Vous pouvez décompresser l'archive et ajouter le chemin d'accès à `%GROOVY_HOME%/bin/groovy.bat` à la variable d'environnement système PATH, où% GROOVY_HOME% est le répertoire dans lequel Groovy est décompacté.

SDKMAN

L'autre option consiste à utiliser SDKMAN . Cette option a rapidement gagné en popularité et rend la gestion de plusieurs versions de Groovy très simple. Il supporte également d'autres applications dans l'écosphère "GR8". Cette option fonctionne très bien sur Linux et Mac, mais nécessite Cygwin sous Windows.

En suivant les instructions de la page de téléchargement Groovy , vous pouvez procéder comme suit pour installer SDKMAN.

```
$ curl -s get.sdkman.io | bash
```

Une fois SDKMAN installé, vous avez maintenant accès à la commande `sdk` . Avec cette commande, vous pouvez faire beaucoup de choses utiles.

Installer Groovy

```
$ sdk install groovy
```

Cela va installer la dernière version de Groovy.

Liste des versions de Groovy

```
$ sdk ls groovy
```

Cela vous permet d'exécuter une commande Linux style `ls` sur le logiciel Groovy, répertoriant

toutes les options disponibles. Il y a un `*` côté de chaque version installée et un `>` pour indiquer vos versions actuelles.

Changer de version de Groovy

```
$ sdk use groovy 2.4.7
```

Cela changera la version actuelle de Groovy à 2.4.7. Si vous avez d'autres versions installées, vous pouvez passer à l'une d'entre elles.

Vous pouvez lister la version actuelle de groovy avec la commande `groovy -version` .

posh-gvm

Le nom initial de SDKMAN était GVM et posh-gvm est un port de GVM pour Windows Powershell. Donc, si vous développez sur une machine Windows et ne voulez pas utiliser SDKMAN sur Cygwin, posh-gvm est fait pour vous. Il fonctionne de la même façon que SDKMAN, mais au lieu de `sdk` , la commande est `gmv` . Alors

```
PS C:\Users\You> gmv install groovy
```

va installer groovy à travers posh-gvm sur votre machine Windows.

Bonjour le monde

La version Groovy de Hello World.

```
println 'Hello World!'
```

Bonjour tout le monde dans groovy

L'exemple suivant illustre le plus simple `Hello World` dans groovy using script, placez l'extrait de code suivant dans un fichier, par exemple `helloWorld.groovy`

```
println 'Hello World!'
```

Comment exécuter: Dans la ligne de commande, `groovy helloWorld.groovy`

Sortie: `Hello World!`

Utiliser Groovy sur un projet Java

Groovy a accès à toutes les classes java, en fait les classes Groovy sont des classes Java et peuvent être exécutées directement par JVM. Si vous travaillez sur un projet Java, utiliser Groovy comme langage de script simple pour interagir avec votre code Java est une évidence.

Pour rendre les choses encore meilleures, presque toutes les classes Java peuvent être renommées en .groovy et compilées et exécutées et fonctionneront exactement comme avant.

Groovy a un REPL. `groovysh` est livré avec Groovy et peut être utilisé pour instancier et tester rapidement une classe Java si votre classpath est configuré correctement. Par exemple, si votre `classpath` de classe pointe vers votre répertoire eclipse "classes / bin", vous pouvez alors enregistrer votre fichier dans eclipse, sauter à `groovysh` et instancier la classe pour le tester.

Les raisons d'utiliser Groovy pour faire cela au lieu de simplement Java sont les suivantes: Le chargeur de classes est excellent pour ramasser les nouvelles classes à mesure qu'elles sont compilées. Vous n'avez généralement pas besoin de quitter / redémarrer `groovysh` fur et à mesure de votre développement.

La syntaxe est TERSE. Ce n'est pas idéal pour le code maintenable, mais pour les scripts et les tests, il peut réduire considérablement votre code. L'une des grandes choses qu'il fait est d'éliminer les exceptions vérifiées (ou, plus précisément, de transformer toutes les exceptions vérifiées en exceptions non vérifiées). Cela transforme le code comme ceci (Imprimer bonjour après une seconde):

```
class JavaClass {
    public static void main(String[] args) {
        try {
            Thread.sleep(1000);
        } catch(InterruptedException e) {
            // You shouldn't leave an empty catch block, but who cares if this was
interrupted???
        }
        System.out.println("Hello!");
    }
}
```

dans Groovy's:

```
Thread.sleep(1000)
print "Hello!"
```

Groovy a également une syntaxe d'initialisation très serrée. Cela vous permet de spécifier des données comme vous l’aimez sans y penser:

En Java pour initialiser une carte, vous devriez probablement faire quelque chose comme ceci:

```
String[] init = { "1:Bill", "2:Doug", "3:Bev" };
// Note the rest of this can be put in a function and reused or maybe found in a library, but
I always seem to have to write this function!
Map m = new HashMap<Integer, String>();
for(String pair : int) {
    String[] split = pair.split(":");
    m.put(new Integer(split[0]), split[1])
}
```

Ce n'est pas mal, mais c'est autre chose à maintenir. En groovy vous utiliseriez simplement:

```
Map map = { 1 : "Bill", 2 : "Doug", 3 : "Bev" }
```

Et vous avez terminé. La syntaxe des listes est tout aussi simple.

L'autre grand avantage est la syntaxe de fermeture de groovy. C'est incroyablement laconique et amusant, un peu plus difficile à maintenir, mais pour les scripts qui ne sont pas une priorité. Par exemple, voici un code groovy pour trouver tous les fichiers `.txt` contenant le mot `Hello` dans le répertoire courant:

```
println new File('.').files.findAll{ it.name.endsWith('.txt') && it.text.contains('Hello')
}.collect{ it.name }
```

Cet exemple utilise quelques astuces "Groovy":

- `.files` fait référence à la méthode `getFiles()` - groovy peut basculer entre getter / setter et la syntaxe des propriétés à volonté
- `it.` fait référence à l'élément actuel d'une itération. `{ it }` est un raccourci pour `{ it -> it }` , par exemple:

  [1, 2, 3] .collect {it ^ 2} == [1, 4, 9]

- `it.text` (où `it` s'agit d'un fichier) utilise une méthode groovy ajoute à `File` pour récupérer l'intégralité du texte du fichier. Ceci est incroyablement utile dans les scripts.

Bonjour le monde Shebang (linux)

Étant donné un fichier hello.groovy avec un contenu:

```
#!/usr/bin/env groovy
println "Hello world"
```

Peut être exécuté à partir de la ligne de commande si l'autorisation d'exécution est donnée

```
$ ./hello.groovy
```

Utilisation de inject () On List pour créer une chaîne CSV

Dans Groovy, la méthode inject () est l'une des méthodes cumulatives qui nous permet d'ajouter (ou d'injecter) de nouvelles fonctionnalités dans tout objet qui implémente la méthode inject (). Dans le cas d'une collection, nous pouvons appliquer une fermeture à une collection d'objets de manière uniforme, puis assembler les résultats en une seule valeur. Le premier paramètre de la méthode inject () est la valeur initiale du cumul et le second paramètre est la fermeture.

Dans cet exemple, nous allons prendre une liste de chaînes en tant que paramètre et afficher les valeurs de ces chaînes délimitées par des virgules. J'ai utilisé cette fonctionnalité pour ajouter une liste de valeurs à une chaîne de requête REST et, si vous la modifiez un peu, je l'ai utilisée pour inclure des valeurs dans une instruction SQL dans le cadre d'une clause IN. Voici le code pour faire ceci:

```
public String convertToCSV( List<String> list ) {
    if (list == null) {
        return ""
    }
    return list.inject( '' ) { result, item ->
        result + ( result && item ? ',' : '' ) + ( item ? "${item.trim()}" : '' )
    }
}

assert convertToCSV( null ) == ""
assert convertToCSV( ["aaa", "bbb  ", null, "  ccc  "] ) == "aaa,bbb,ccc"
```

Dans cet exemple, le premier paramètre de la méthode inject () est une chaîne de longueur zéro, ce qui signifie que lors du traitement du premier élément de la liste, le résultat est également une chaîne de longueur nulle. Cela se résume en faux dans la première évaluation ternaire, ce qui explique pourquoi nous n'obtenons pas de virgule au début de la chaîne. Avec chaque itération consécutive des éléments de la liste, le résultat devient la concaténation de lui-même, une virgule, puis l'élément suivant jusqu'à ce que nous atteignions le dernier élément de la liste.

L'avantage de cette approche est que vous n'avez pas besoin d'une variable en dehors d'une construction en boucle pour contenir le résultat de chaîne concaténé. L'implication étant que cela peut entraîner des effets secondaires dans votre code. Avec l'approche inject (), ce comportement est injecté et la collection associe le résultat des appels à la fermeture pour vous. L'inconvénient de cette approche peut être la lisibilité. Mais avec de l'expérience, cela devient plus facile à lire et à comprendre, et j'espère que cet exemple vous aidera à atteindre cet objectif.

Lire Commencer avec groovy en ligne: https://riptutorial.com/fr/groovy/topic/966/commencer-avec-groovy

Chapitre 2: Chaînes et littéraux GString

Syntaxe

- 'Chaîne unique citée'
- "Double guillemets"
- " 'Chaîne multiligne' "
- "" "Chaîne triple citée" ""
- / Chaîne Slashy /
- $ / Dollar barre oblique / $

Remarques

Groovy a deux types de chaînes, `java.lang.String` et `groovy.lang.GString` , ainsi que plusieurs formes de littéraux de chaîne (voir la syntaxe et des exemples).

La principale différence entre les deux types de chaînes est que GString prend en charge l'interpolation de chaînes.

Examples

Chaîne unique citée

```
def str = 'Single quoted string'
assert str instanceof String
```

Chaîne entre guillemets (sans espace réservé pour l'interpolation)

```
def str = "Double quoted string"
assert str instanceof String
```

Chaîne entre guillemets (interpolation)

```
def param = 'string'
def str = "Double quoted ${param}"
assert str instanceof GString
assert str == 'Double quoted string'
```

Le paramètre est résolu par défaut avec impatience, cela signifie que ceci s'applique:

```
def param = 'string'
def str = "Double quoted ${param}"
param = 'another string'
assert str == 'Double quoted string'
```

Pour charger le paramètre paresseusement chaque fois que la chaîne est utilisée, cela peut être fait:

```
def param = 'string'
def str = "Double quoted ${ -> param}"
assert str == 'Double quoted string'
param = 'lazy load'
assert str == 'Double quoted lazy load'
```

Chaîne multiligne

```
def str = '''multiline
string'''
assert str instanceof String
```

Chaîne multiligne (nouvelle ligne supplémentaire)

```
def str = '''
multiline
string'''
assert str.readLines().size() == 3
```

Chaîne multiligne (sans nouvelle ligne de fin)

```
def str = '''\
multiline
string'''
assert str.readLines().size() == 2
```

Triple guillemet double

```
def param = 'string'
def str = """
multiline
$param
"""
assert str instanceof GString
assert str.readLines().size() == 3
assert str == '''
multiline
string
'''
```

Slashy string (pas d'espace réservé pour l'interpolation)

```
def str = /
multiline string
no need to escape slash
\n
/
assert str instanceof String
```

```
assert str.readLines().size() == 4
assert str.contains('\\n')
```

Slashy string (interpolation)

```
def param = 'string'
def str = /
multiline $param
no need to escape slash
\n
/
assert str instanceof GString
assert str.readLines().size() == 4
assert str.contains('\\n')
assert str.contains('string')
```

Chaîne de coupe de dollar

```
def param = 'string'
def str = $/
multiline $param
no need to escape slash
\n
$
$$
/$
assert str instanceof GString
assert str.readLines().size() == 6
assert str.contains('\\n')
assert str.contains('$')
```

Lire Chaînes et littéraux GString en ligne: https://riptutorial.com/fr/groovy/topic/3409/chaines-et-litteraux-gstring

Chapitre 3: Currying

Syntaxe

- closing.curry (paramètre)
- closure.rcurry (paramètre)
- closure.ncurry (index, paramètres ...)

Remarques

- Currying une fermeture produit une nouvelle fermeture avec un ou plusieurs de ses paramètres ayant une valeur fixe
- Currying gauche ou droit d'une fermeture qui n'a pas de paramètres ou basée sur l'index curing d'une fermeture qui a moins de deux paramètres lève une `IllegalArgumentException`

Examples

Curry gauche

```
def pow = { base, exponent ->
    base ** exponent
}
assert pow(3, 2) == 9

def pow2 = pow.curry(2) //base == 2
assert pow2(3) == 8
```

Droit currying

```
def dividable = { a, b ->
    a % b == 0
}
assert dividable(2, 3) == false
assert dividable(4, 2) == true

def even = dividable.rcurry(2) // b == 2
assert even(2) == true
assert even(3) == false
```

Curry basé sur index

```
def quatNorm = { a, b, c, d ->
    Math.sqrt(a*a + b*b + c*c + d*d)
}
assert quatNorm(1, 4, 4, -4) == 7.0

def complexNorm = quatNorm.ncurry(1, 0, 0) // b, c == 0
```

```
assert complexNorm(3, 4) == 5.0
```

Fermeture de curry sans paramètre explicite

```
  def noParam = {
      "I have $it"
  }

  def noParamCurry = noParam.curry(2)
  assert noParamCurry() == 'I have 2'
```

Fermeture de curry sans paramètres

```
def honestlyNoParam = { ->
    "I Don't have it"
}

// The following all throw IllegalArgumentException
honestlyNoParam.curry('whatever')
honestlyNoParam.rcurry('whatever')
honestlyNoParam.ncurry(0, 'whatever')
```

Lire Currying en ligne: https://riptutorial.com/fr/groovy/topic/4400/currying

Chapitre 4: Fermetures

Examples

Fermeture avec paramètres explicites

```
def addNumbers = { a, b -> a + b }
addNumbers(-7, 15) // returns 8
```

Fermeture avec des paramètres implicites

```
['cat', 'dog', 'fish'].collect { it.length() }
```

`it` le nom par défaut du paramètre si vous avez un seul paramètre et ne nommez pas explicitement le paramètre. Vous pouvez également déclarer le paramètre en option.

```
['cat', 'dog', 'fish'].collect { animal -> animal.length() }
```

Conversion de méthodes en fermetures

Une méthode peut être convertie en une fermeture à l'aide de l'opérateur **&** .

```
def add(def a, def b) { a + b }

Closure addClosure = this.&add
assert this.add(4, 5) == addClosure(4, 5)
```

Fermeture avec une cible personnalisée pour les appels de méthode avec récepteur implicite

```
class MyHello {
  def sayHello() {
    "Hello, world"
  }
}

def cl = { sayHello() }
cl() // groovy.lang.MissingMethodException
cl.delegate = new MyHello()
cl(); // "Hello, world"
```

Très utilisé par les DSL Groovy.

Comportement d'emballage autour d'une fermeture avec un procédé

Il existe des comportements fréquents qui peuvent entraîner beaucoup de code passe-partout. En

déclarant une méthode qui prend une `Closure` en paramètre, vous pouvez simplifier votre programme. Par exemple, il est courant de récupérer une connexion à une base de données, de lancer une transaction, de travailler, puis de valider la transaction ou de restaurer la connexion (en cas d'erreur), puis de fermer la connexion:

```
def withConnection( String url, String user, String pass, Closure closure) {
    Connection conn = null
    try {
        conn = DriverManager.getConnection( url, user, pass )
        closure.call( conn )
        conn.commit()
    } catch (Exception e) {
        log.error( "DB Action failed", e)
        conn.rollback()
    } finally {
        conn?.close()
    }
}


withConnection( DB_PATH, DB_USER, DB_PASS ) { Connection conn ->
    def statement = conn.createStatement()
    def results = statement.executeQuery( 'SELECT * FROM users' )
    // ... more processing ...
}
```

Créer des fermetures, attribuer des propriétés et appeler

Créons une carte et une fermeture pour imprimer Bonjour

```
def exMap = [:]

def exClosure = {
    println "Hello"
}
```

Attribuer une fermeture à une propriété sur la carte

```
exMap.closureProp = exClosure
```

Fermeture d'appel

```
exMap.closureProp.call()
```

Sortie

```
Hello
```

Un autre exemple - Permet de créer une classe avec une propriété de base et d'affecter la même fermeture à l'objet de celle-ci

```
class Employee {
```

```
    def prop
}

def employee = new Employee()

employee.prop = exClosure
```

Fermeture d'appel via cette propriété

```
employee.prop.call()
```

Sortie

```
Hello
```

Lire Fermetures en ligne: https://riptutorial.com/fr/groovy/topic/2684/fermetures

Chapitre 5: Fonctions mémo

Examples

Fonctions mémo

La mémorisation est essentiellement un moyen de mettre en cache les résultats des méthodes. Cela peut être utile lorsqu'une méthode est souvent appelée avec les mêmes arguments et que le calcul du résultat prend du temps, ce qui augmente les performances.

À partir de Groovy 2.2, les méthodes peuvent être annotées avec l'annotation `@Memoized` .

Imaginez la classe suivante:

```
class MemoDemo {
  def timesCalculated = 0

  @Memoized
  def power2(a) {
    timesCalculated++
    a * a
  }
}
```

Maintenant, lors du premier appel de cette méthode avec un numéro avec lequel il n'a pas été appelé auparavant, la méthode sera exécutée:

```
assert power2(2) == 4
assert timesCalculated == 1
```

Cependant, si nous l'appelons à nouveau avec le même argument:

```
assert power2(2) == 4
assert timesCalculated == 1
```

`timesCalculated` est resté inchangé, mais la méthode a renvoyé le même résultat. Cependant, l'appelant avec un argument différent:

```
assert power2(3) == 9
assert timesCalculated == 2
```

entraîne le rappel du corps de la méthode.

Lire Fonctions mémo en ligne: https://riptutorial.com/fr/groovy/topic/6176/fonctions-memo

Chapitre 6: Fonctions mémo

Examples

Memoize sur les fermetures

Depuis *Groovy* 1.8, une méthode `memoize()` est ajoutée aux fermetures:

```
// normal closure
def sum = { int x, int y  ->
    println "sum ${x} + ${y}"
    return x + y
}
sum(3, 4)
sum(3, 4)
// prints
// sum 3 + 4
// sum 3 + 4

// memoized closure
def sumMemoize = sum.memoize()
sumMemoize(3, 4)
// the second time the method is not called
// and the result it's take from the previous
// invocation cache
sumMemoize(3, 4)
// prints
// sum 3 + 4
```

Mémo sur les méthodes

Depuis *Groovy 2.2, les* annotations `groovy.transform.Memoized` sont ajoutées aux méthodes de mémorisation pratiques en ajoutant simplement l'annotation `@Memoized` :

```
import groovy.transform.Memoized

class Calculator {
    int sum(int x, int y){
        println "sum ${x} + ${y}"
        return x+y
    }

    @Memoized
    int sumMemoized(int x, int y){
        println "sumMemoized ${x} + ${y}"
        return x+y
    }
}

def calc = new Calculator()

// without @Memoized, sum() method is called twice
calc.sum(3,4)
calc.sum(3,4)
```

```
// prints
// sum 3 + 4
// sum 3 + 4

// with @Memoized annotation
calc.sumMemoized(3,4)
calc.sumMemoized(3,4)
// prints
// sumMemoized 3 + 4
```

Lire Fonctions mémo en ligne: https://riptutorial.com/fr/groovy/topic/6471/fonctions-memo

Chapitre 7: Groovy Code Golfing

Introduction

Conseils pour jouer au golf à Groovy

Examples

Opérateur de point d'épandage (*.)

L'opérateur de point d'épandage peut être utilisé à la place de la méthode de collecte

```
(1..10)*.multiply(2) // equivalent to (1..10).collect{ it *2 }
d = ["hello", "world"]
d*.size() // d.collect{ it.size() }
```

Traitement parallèle à l'aide de Gpars

Gpars offre des moyens intuitifs pour gérer les tâches simultanément

```
import groovyx.gpars.*
GParsPool.withPool { def result = dataList.collectParallel { processItem(it) } }
```

Lire Groovy Code Golfing en ligne: https://riptutorial.com/fr/groovy/topic/10651/groovy-code-golfing

Chapitre 8: Groovy Truth (la vérité)

Remarques

Groovy évalue les conditions dans les instructions **if** , **while** et **for de la même manière que Java pour les conditions Java standard** : en Java, vous devez fournir une expression booléenne (une expression qui donne une valeur booléenne) et le résultat est le résultat de l'évaluation.

Dans Groovy, le résultat est le même qu'en Java pour ces conditions (aucun exemple fourni, ceci est le standard Java).

L'autre **mécanisme d'évaluation de la véracité** présenté par les exemples peut être résumé comme suit:

- nombres: une valeur nulle est fausse, non nulle à vraie.
- objects: une référence d'objet null est évaluée à false, une référence non null à true.
- Caractère: un caractère avec une valeur zéro est évalué à faux, vrai sinon.
- Chaîne: une chaîne est évaluée à true si elle n'est pas nulle et non vide, false si elle est nulle ou vide (s'applique également à GStrings et CharSequences).
- Collections et cartes (y compris les sous-classes **List** , **Map** , **Set** , **HashSet** ...): prend également en compte la taille, évaluée à true si la collection n'est pas nulle et non vide, false si nulle ou vide.
- Les énumérations et les itérateurs sont évalués à true s'ils ne sont pas nuls et ce sont d'autres éléments (groovy évalue **hasMoreElements** ou **hasNext** sur l'objet), false si null ou aucun autre élément.
- Matcher: un matcher est évalué à true s'il y a au moins une correspondance, si false s'il n'y a pas de correspondance est trouvé.
- Fermeture: une fermeture évalue à l'évaluation du résultat renvoyé par la fermeture.

La méthode asBoolean peut être remplacée dans une classe définie par l'utilisateur pour fournir une évaluation booléenne personnalisée.

Examples

Evaluation booléenne des nombres

pour les nombres, une valeur de zéro vaut false, non zéro à true

```
    int i = 0
...
    if (i)
        print "some ${i}"
    else
        print "nothing"
```

imprimera "rien"

Évaluation booléenne de chaînes

une chaîne (y compris GStrings) est évaluée à true si elle n'est pas nulle et non vide, false si nulle ou vide

```
def s = ''
...
if (s)
    println 's is not empty'
else
    println 's is empty'
```

va imprimer: 's is empty'

Évaluation booléenne des collections et des cartes

Les collections et les cartes sont évaluées à true si elles ne sont pas nulles et ne sont pas vides et false si elles sont nulles ou vides

```
/* an empty map example*/
def userInfo = [:]
if (!userInfo)
    userInfo << ['user': 'Groot', 'species' : 'unknown' ]
```

ajoutera l' `user: 'Groot' , species : 'unknown'` par défaut userInfo puisque la map userInfo est vide (notez que la map n'est pas nulle ici)

Avec un objet nul, le code est légèrement différent, on ne peut pas invoquer << sur userInfo car il est nul, il faut faire une affectation (voir aussi l'évaluation booléenne Object):

```
/* an example with a null object (def does not implies Map type) */
def userInfo = null
if (!userInfo)
    userInfo = ['user': 'Groot', 'species' : 'unknown' ]
```

Et avec une carte nulle:

```
/* The same example with a null Map */
Map<String,String> userInfo = null
if (!userInfo)
    userInfo = ['user': 'Groot', 'species' : 'unknown' ]
```

Evaluation booléenne d'objet

une référence d'objet NULL donne la valeur false, une référence non null à true, mais pour les chaînes, les collections, les itérateurs et les énumérations, elle prend également en compte la taille.

```
def m = null
```

```
if (!m)
    println "empty"
else
    println "${m}"
```

imprimera "vide"

```
def m = [:]

if (!m)
    println "empty"
else
    println "${m}"
```

La carte n'est pas nulle mais vide, ce code affichera "vide"

Après avoir fait

```
m << ['user' : 'Groot' ]
```

il imprimera la carte:

```
[user:Groot]
```

Remplacement de l'évaluation booléenne dans une classe définie par l'utilisateur

Parfois, il peut être utile d'avoir une définition asBoolean spécifique dans votre propre programme pour certains types d'objets.

```
/** an oversimplified robot controller */
class RunController {

    def complexCondition
    int position = 0

    def asBoolean() {
        return complexCondition(this);
    }
    def advanceTo(step) {
        position += step
    }
}
def runController = new RunController(complexCondition : { c -> c.position < 10 } )

assert runController
runController.advanceTo(5)
assert runController
runController.advanceTo(5)
// The limit has been reached : the controller evaluates to false
assert !runController
```

Ce code montre un contrôleur de robot trop simplifié qui vérifie que la position du robot ne

dépasse pas 10 (avec une fermeture pour évaluation de la condition)

Évaluation de personnage

un caractère est évalué à true si sa valeur n'est pas zéro, faux si zéro

```
assert ! new Character((char)0)
assert ! new Character('\u0000Hello Zero Char'.charAt(0))
assert  new Character('Hello'.charAt(0))
```

Évaluation matcher

un Matcher est évalué à true s'il peut trouver au moins une correspondance, false si aucune correspondance n'est trouvée

```
// a match is found => true
assert 'foo' =~ /[a-z]/
// the regexp does not match fully => no match => false
assert !( 'foo' ==~ /[a-z]/ )
// a match is found => true
assert 'foo' =~ /o/
// no match => false
assert !( 'foo' =~ /[A-Z]/ )
```

Évaluation de la fermeture

L'évaluation d'une fermeture est l'évaluation du résultat de la fermeture.

Toutes les règles s'appliquent: si la fermeture renvoie un nombre nul ou nul ou une chaîne, une collection, un mappage ou un tableau vide, la valeur false est définie sur true.

```
// Closure return non zero number => true
assert { 42 }()
// closure returns 0 => false
assert ! ( { 0 }())
// closure returns null => false
assert !( { }())
```

Lire Groovy Truth (la vérité) en ligne: https://riptutorial.com/fr/groovy/topic/5117/groovy-truth--la-verite-

Chapitre 9: Interpolation de chaîne

Syntaxe

- $
- $ {}
- $ {->}

Examples

De base

```
def str = 'nice'
assert "Groovy is $str" == 'Groovy is nice'
```

Expression en pointillé

```
def arg = [phrase: 'interpolated']
assert "This is $arg.phrase" == 'This is interpolated'
```

Expression désagréable

```
def str = 'old'
def interpolated = "I am the ${str} value"
assert interpolated == 'I am the old value'
str = 'new'
assert interpolated == 'I am the old value'
```

Expression paresseuse

Nous pouvons avoir une interpolation paresseuse dans les chaînes. Ceci est différent de l'interpolation normale car le GString peut potentiellement avoir des valeurs différentes, selon la fermeture, chaque fois qu'il est converti en chaîne.

```
def str = 'old'
def interpolated = "I am the ${ -> str} value"
assert interpolated == 'I am the old value'
str = 'new'
assert interpolated == 'I am the new value'
```

Expression

```
def str = 'dsl'
def interpolated = "Groovy ${str.length() + 1} easy ${str.toUpperCase()}"
assert interpolated == 'Groovy 4 easy DSL'
str = 'Domain specific language'
```

```
assert interpolated == 'Groovy 4 easy DSL'
```

Lire Interpolation de chaîne en ligne: https://riptutorial.com/fr/groovy/topic/3125/interpolation-de-chaine

Chapitre 10: JSON

Examples

Analyser une chaîne json

```
import groovy.json.JsonSlurper;

def jsonSlurper = new JsonSlurper()
def obj = jsonSlurper.parseText('{ "foo": "bar", "baz": [1] }')

assert obj.foo == 'bar'
assert obj.baz == [1]
```

Analyser un fichier json

```
import groovy.json.JsonSlurper;

def jsonSlurper = new JsonSlurper()

File fl = new File('/path/to/fils.json')

// parse(File file) method is available since 2.2.0
def obj = jsonSlurper.parse(fl)

// for versions < 2.2.0 it's possible to use
def old = jsonSlurper.parse(fl.text)
```

Ecrivez un json à la chaîne

```
import groovy.json.JsonOutput;

def json = JsonOutput.toJson([foo: 'bar', baz: [1]])

assert json == '{"foo":"bar","baz":[1]}'
```

En plus des cartes, des listes et des primitives, `groovy.json.JsonOutput` prend également en charge une sérialisation de *POJO* :

```
import groovy.json.JsonOutput;

class Tree {
    def name
    def type
}

Tree willow = new Tree(name:'Willow',type:'Deciduous')
Tree olive = new Tree(name:'Olive',type:'Evergreen')

assert JsonOutput.toJson(willow) == '{"type":"Deciduous","name":"Willow"}'
assert JsonOutput.toJson([willow,olive]) ==
```

```
'[{"type":"Deciduous","name":"Willow"},{"type":"Evergreen","name":"Olive"}]'
```

Jolie-imprimer une chaîne json

```
import groovy.json.JsonOutput;

def json = JsonOutput.toJson([foo: 'bar', baz: [1]])

assert json == '{"foo":"bar","baz":[1]}'

def pretty = JsonOutput.prettyPrint(json)

assert pretty == '''{
    "foo": "bar",
    "baz": [
        1
    ]
}'''
```

Ecrit un json dans un fichier

```
import groovy.json.JsonOutput;

def json = JsonOutput.toJson([foo: 'bar', baz: [1]])

new File("/tmp/output.json").write(json)
```

Lire JSON en ligne: https://riptutorial.com/fr/groovy/topic/5352/json

Chapitre 11: Langues spécifiques au domaine

Examples

Capacités linguistiques

Le Pipeline DSL Jenkins est utilisé comme exemple pour un tel langage:

```
node {
  git 'https://github.com/joe_user/simple-maven-project-with-tests.git'
  def mvnHome = tool 'M3'
  sh "${mvnHome}/bin/mvn -B -Dmaven.test.failure.ignore verify"
  archiveArtifacts artifacts: '**/target/*.jar', fingerprint: true
  junit '**/target/surefire-reports/TEST-*.xml'
 }
```

Le but de ce DSL est de définir et d'exécuter des tâches de génération Jenkins (ou de meilleurs pipelines) dans un langage plus naturel.

L'écriture d'un langage spécifique à un domaine dans Groovy bénéficie des fonctionnalités clés de Groovy telles que:

- Caractère facultatif (par exemple, omettre les parenthèses)
- Surcharge de l'opérateur
- Méta-programmation (par exemple, résolution des propriétés ou des méthodes manquantes)
- Fermetures et stratégies de délégation
- Personnalisation du compilateur
- Prise en charge des scripts et capacités d'intégration

Lire Langues spécifiques au domaine en ligne: https://riptutorial.com/fr/groovy/topic/5948/langues-specifiques-au-domaine

Chapitre 12: Manières d'itération à Groovy

Introduction

Groovy a plus de façons de boucler que de supporter les itérations Java.

Groovy étend la classe `Integer` avec les méthodes `step()` , `upto()` et `times()` . Ces méthodes prennent une fermeture comme paramètre. Dans la clôture, nous définissons le morceau de code que nous voulons exécuter plusieurs fois.

Il ajoute également les méthodes `each()` et `eachWithIndex()` pour parcourir les collections.

Examples

Comment puis-je faire quelque chose n fois?

Comment puis-je imprimer le *monde* 5 fois *bonjour* ?

```
5.times{
    println "hello world"
}
```

Chaque et chaque index

`each` et `eachWithIndex` sont des méthodes pour parcourir les collections.

ont chacun `it` (par défaut iterator) et `eachWithIndex` ont `it` , `index` (iterator par défaut, index par défaut).

Nous pouvons également modifier l'itérateur / index par défaut. S'il vous plaît voir ci-dessous des exemples.

```
def list = [1,2,5,7]
list.each{
    println it
}

list.each{val->
    println val
}

list.eachWithIndex{it,index->
    println "value " + it + " at index " +index
}
```

Lire Manières d'itération à Groovy en ligne: https://riptutorial.com/fr/groovy/topic/9844/manieres-d-iteration-a-groovy

Chapitre 13: Méthodes de mémorisation de fermeture

Syntaxe

- closure.memoize ()
- closure.memoizeAtMost (n)
- closure.memoizeAtLeast (n)
- closing.memoizeBetween (n, m)

Remarques

La mémorisation est une méthode de mise en cache du résultat d'une invocation de fermeture. La fonction memoize appliquée à une fermeture renvoie une nouvelle fermeture dont la valeur de retour est mise en cache en fonction de ses paramètres d'entrée. Les caches utilisés pour les trois variantes modifiées des méthodes de mémo sont les caches LRU, c'est-à-dire que l'élément le moins récemment utilisé est d'abord supprimé du cache.

Examples

Mémo simple

```
def count = 0

nonmemoized = { long n -> println "nonmemoized: $n"; count++ }

nonmemoized(1)
nonmemoized(2)
nonmemoized(2)
nonmemoized(1)
assert count == 4


def mcount = 0

memoized = { long n -> println "memoized: $n"; mcount++ }.memoize()

memoized(1)
memoized(2)
memoized(2)
memoized(1)
assert mcount == 2
```

Lire Méthodes de mémorisation de fermeture en ligne:
https://riptutorial.com/fr/groovy/topic/6308/methodes-de-memorisation-de-fermeture

Chapitre 14: Opérateur d'épandage

Remarques

Dans la plupart des cas, l'opérateur de propagation `*.` est identique à l'appel à `.collect { it._____ }`.

```
def animals = ['cat', 'dog', 'fish']
assert animals*.length() == animals.collect { it.length() }
```

Mais si le sujet est nul, ils se comportent différemment:

```
def animals = null
assert animals*.length() == null
assert animals.collect { it.length() } == []
```

Examples

Appeler une méthode

```
assert ['cat', 'dog', 'fish']*.length() == [3, 3, 4]
```

Notez que lors du mélange de types dans la collection si la méthode n'existe pas sur certains éléments, une `groovy.lang.MissingMethodException` peut être `groovy.lang.MissingMethodException` :

```
['cat', 'dog', 'fish',3]*.length()
// it throws groovy.lang.MissingMethodException: No signature of method:
java.lang.Integer.length()
```

Accéder à une propriété

```
class Vector {
    double x
    double y
}
def points = [
    new Vector(x: 10, y: -5),
    new Vector(x: -17.5, y: 3),
    new Vector(x: -3.3, y: -1)
]

assert points*.x == [10, -17.5, -3.3]
```

Note: Le `*` est optionnel. Nous pourrions également écrire l'instruction ci-dessus comme dans la ligne ci-dessous et le compilateur Groovy serait toujours content.

```
assert points.x == [10, -17.5, -3.3]
```

Son null-safe

S'il existe un objet `null` sur la collection, il ne génère pas de `NPE` , mais renvoie une valeur `null` :

```
assert ['cat', 'dog', 'fish', null]*.length() == [3, 3, 4, null]
```

En l'utilisant directement dans un objet `null` , il est également null-safe:

```
def nullCollection = null
assert nullCollection*.length() == null
```

Lire Opérateur d'épandage en ligne: https://riptutorial.com/fr/groovy/topic/2725/operateur-d-epandage

Chapitre 15: Opérateur de navigation sécurisé

Examples

Utilisation de base

L' *opérateur de navigation sûr* de Groovy permet d'éviter les `NullPointerException` s lors de l'accès à des méthodes ou à des attributs sur des variables pouvant prendre `null` valeurs `null` . C'est équivalent à `nullable_var == null ? null : nullable_var.myMethod()`

```
def lst = ['foo', 'bar', 'baz']

def f_value = lst.find { it.startsWith('f') }    // 'foo' found
f_value?.length()    // returns 3

def null_value = lst.find { it.startsWith('z') }    // no element found. Null returned

// equivalent to  null_value==null ? null : null_value.length()
null_value?.length()    // no NullPointerException thrown

// no safe operator used
null_value.length()    // NullPointerException thrown
```

Concaténation d'opérateurs de navigation sécurisés

```
class User {
  String name
  int age
}

def users = [
  new User(name: "Bob", age: 20),
  new User(name: "Tom", age: 50),
  new User(name: "Bill", age: 45)
]

def null_value = users.find { it.age > 100 }    // no over-100 found. Null

null_value?.name?.length()    // no NPE thrown
//  null ?. name  ?. length()
// (null ?. name) ?. length()
// (    null    ) ?. length()
// null

null_value?.name.length()    // NPE thrown
//  null ?. name  . length()
// (null ?. name) . length()
// (    null    ) . length()  ===> NullPointerException
```

la navigation sécurisée sur `null_value?.name` renvoie une valeur `null` . Ainsi, `length()` devra

effectuer une vérification de la valeur `null` pour éviter une `NullPointerException` .

Lire Opérateur de navigation sécurisé en ligne:
https://riptutorial.com/fr/groovy/topic/5116/operateur-de-navigation-securise

Chapitre 16: Opérateur de vaisseau spatial

Examples

Utilisation de base

l'opérateur de vaisseau spatial renvoie `-1` lorsque l'opérateur de gauche est plus petit, `0` lorsque les opérateurs sont égaux et `1` sinon:

```
assert 10 <=> 20 == -1
assert 10 <=> 10 == 0
assert 30 <=> 10 == 1

assert 'a' <=> 'b' == -1
assert 'a' <=> 'a'== 0
assert 'b' <=> 'a' == 1
```

C'est équivalent à la méthode Comparable.compareTo:

```
assert 10.compareTo(20) == (10 <=> 20)
assert 'a'.compareTo('b') == ('a' <=> 'b')
```

Opérateur de vaisseau spatial pour les tri sur mesure

```
class User {
  String name
  int age
}

def users = [
  new User(name: "Bob", age: 20),
  new User(name: "Tom", age: 50),
  new User(name: "Bill", age: 45)
]

// sort by age
users.sort { a, b -> a.age <=> b.age }
```

Utilisation avec Comparator et SortedSet

```
Comparator cmp = [ compare:{ a, b -> a <=> b } ] as Comparator
def col = [ 'aa', 'aa', 'nn', '00' ]
SortedSet sorted = new TreeSet( cmp )
sorted.addAll col
assert '[00, aa, nn]' == sorted.toString()
```

Lire Opérateur de vaisseau spatial en ligne: https://riptutorial.com/fr/groovy/topic/4394/operateur-de-vaisseau-spatial

Chapitre 17: Opérateurs de collecte

Examples

Itérer sur une collection

Des listes

```
def lst = ['foo', 'bar', 'baz']
// using implicit argument
lst.each { println it }

// using explicit argument
lst.each { val -> println val }

// both print:
// foo
// bar
// baz
```

Itérer avec index

```
def lst = ['foo', 'bar', 'baz']
// explicit arguments are required
lst.eachWithIndex { val, idx -> println "$val in position $idx" }

// prints:
// foo in position 0
// bar in position 1
// baz in position 2
```

Plans

```
def map = [foo: 'FOO', bar: 'BAR', baz: 'BAZ']

// using implicit argument
map.each { println "key: ${it.key}, value: ${it.value}"}

// using explicit arguments
map.each { k, v -> println "key: $k, value: $v"}

// both print:
// key: foo, value: FOO
// key: bar, value: BAR
// key: baz, value: BAZ
```

Créer une nouvelle liste en utilisant collect

```
def lst = ['foo', 'bar', 'baz']
lst.collect { it } // ['foo', 'bar', 'baz']

lst.collect { it.toUpperCase() } // ['FOO', 'BAR', 'BAZ']
```

Pour collecter des clés ou des valeurs d'une carte

```
def map = [foo: 'FOO', bar: 'BAR', baz: 'BAZ']
def keys = map.collect { it.key } // ['foo', 'bar', 'baz']
def vals = map.collect { it.value } // ['FOO', 'BAR', 'BAZ']
```

L'exemple ci-dessus équivaut à appeler `map.keySet()` et `map.values()`

Filtrer une liste avec findAll

```
def lst = [10, 20, 30, 40]

lst.findAll { it > 25 } // [30, 40]
```

Trouvez le premier élément correspondant à une condition

```
def lst = [10, 20, 30, 40]

lst.find { it > 25 } // 30. Note: it returns a single value
```

Créer des cartes avec collectEntries

Des listes

```
def lst = ['foo', 'bar', 'baz']

// for each entry return a list containing [key, value]
lst.collectEntries { [it, it.toUpperCase()] } // [foo: FOO, bar: BAR, baz: BAZ]

// another option, return a map containing the single entry
lst.collectEntries { [(it): it.toUpperCase()] } // [foo: FOO, bar: BAR, baz: BAZ]
```

Des cartes

```
def map = [foo: 'FOO', bar: 'BAR', baz: 'BAZ']

map.collectEntries { [it.key*2, it.value*2] } // [foofoo: FOOFOO, barbar: BARBAR, bazbaz:
BAZBAZ]

// using explicit arguments k and v
map.collectEntries { k, v -> [k*2, v*2] } // [foofoo: FOOFOO, barbar: BARBAR, bazbaz: BAZBAZ]
```

Appliquer la transformation aux collections imbriquées

Appliquez la transformation aux entrées autres que celles de collection, en explorant également

les collections imbriquées et en préservant la structure entière.

```
def lst = ['foo', 'bar', ['inner_foo', 'inner_bar']]

lst.collectNested { it.toUpperCase() } // [FOO, BAR, [INNER_FOO, INNER_BAR]]
```

Aplatir une liste imbriquée

```
def lst = ['foo', 'bar', ['inner_foo', 'inner_bar']]

lst.flatten() // ['foo', 'bar', 'inner_foo', 'inner_bar']
```

Supprimer les doublons

```
def lst = ['foo', 'foo', 'bar', 'baz']

// *modifies* the list removing duplicate items
lst.unique() // [foo, bar, baz]

// setting to false the "mutate" argument returns a new list, leaving the original intact
lst.unique(false) // [foo, bar, baz]

// convert the list to a Set, thus removing duplicates
lst.toSet() // [baz, bar, foo]

// defining a custom equality criteria. For example: to elements are equal if have the same
first letter
println lst.unique() { it[0] } // [foo, bar]. 'bar' and 'baz' considered equal
```

Construire une carte à partir de deux listes

```
nrs = [1, 2, 3, 4, 5, 6, 7, 8, 9]
lets = ['a', 'b', 'c', 'd', 'e', 'f']

println GroovyCollections.transpose([nrs, lets])
        .collect {le -> [(le[0]):le[1]]}.collectEntries { it }

or

println [nrs,lets].transpose().collectEntries{[it[0],it[1]]}

// [1:a, 2:b, 3:c, 4:d, 5:e, 6:f]
```

Lire Opérateurs de collecte en ligne: https://riptutorial.com/fr/groovy/topic/5103/operateurs-de-collecte

Chapitre 18: Opérateurs Ternaires et Elvis

Remarques

L'opérateur Elvis évalue en fonction de *Groovy-Truth* de la partie condition.

Examples

Forme standard vs forme Elvis

```
// long form
String sayHello(String name){
    "Hello, ${name ? name : 'stranger'}."
}

// elvis
String sayHello(String name){
    "Hello, ${name ?: 'stranger'}."
}
```

Notez que le format "elvis" omet le terme "true" car la valeur de comparaison d'origine doit être utilisée dans le cas "true". Si `name` est Groovy `true` , alors il sera renvoyé comme valeur de l'expression.

Utilisation (avec condition) en affectation

```
def results = []
(1..4).each{
    def what = (it%2) ? 'odd' :  'even'
    results << what
}
assert results == ['odd', 'even', 'odd', 'even']
```

Ici, la condition if `(parentheses)` est légèrement plus complexe que le simple test de l'existence / Groovy-Truth.

Lire Opérateurs Ternaires et Elvis en ligne: https://riptutorial.com/fr/groovy/topic/3912/operateurs-ternaires-et-elvis

Chapitre 19: RESTClient

Introduction

Utilisation du client HTTP Groovy, exemples et pièges.

Examples

Demande GET

```
@Grab(group='org.codehaus.groovy.modules.http-builder', module='http-builder', version='0.7' )

import groovyx.net.http.RESTClient

try {
    def restClient = new RESTClient("http://weathers.co")
    def response = restClient.get(path: '/api.php', query: ['city': 'Prague'])
    println "Status     : ${response.status}"
    println "Body       : ${response.data.text}"
} catch (Exception e) {
    println "Error      : ${e.statusCode}"
    println "Message    : ${e.response.data}"
}
```

Lire RESTClient en ligne: https://riptutorial.com/fr/groovy/topic/8919/restclient

Chapitre 20: Traits

Introduction

Les traits sont des objets de construction structurels dans le langage Groovy. Les traits permettent l'implémentation des interfaces. Ils sont compatibles avec la vérification de type statique et la compilation. Les traits se comportent comme des interfaces avec des implémentations et des états par défaut. La déclaration d'un trait est en utilisant le mot-clé **trait** . ---------- La portée des méthodes de caractères ne prend en charge que **les** méthodes **publiques** et **privées** .

Examples

Utilisation de base

Un `trait` est un ensemble réutilisable de méthodes et de champs pouvant être ajoutés à une ou plusieurs classes.

```
trait BarkingAbility {
    String bark(){ "I'm barking!!" }
}
```

Ils peuvent être utilisés comme des interfaces normales, en utilisant le mot-clé `implements` :

```
class Dog implements BarkingAbility {}
def d = new Dog()
assert d.bark() = "I'm barking!!"
```

Ils peuvent également être utilisés pour mettre en œuvre l'héritage multiple (en évitant la question du diamant).

Les chiens peuvent se gratter la tête, alors:

```
trait ScratchingAbility {
    String scratch() { "I'm scratching my head!!" }
}

class Dog implements BarkingAbility, ScratchingAbility {}
def d = new Dog()
assert d.bark() = "I'm barking!!"
assert d.scratch() = "I'm scratching my head!!"
```

Problème d'héritage multiple

La classe peut implémenter plusieurs traits. Si un trait définit une méthode avec la même signature comme un autre trait, il y a un problème d'héritage multiple. Dans ce cas, la méthode du **dernier trait déclaré** est utilisée:

```
trait Foo {
  def hello() {'Foo'}
}
trait Bar {
  def hello() {'Bar'}
}

class FooBar implements Foo, Bar {}

assert new FooBar().hello() == 'Bar'
```

Lire Traits en ligne: https://riptutorial.com/fr/groovy/topic/6687/traits

Chapitre 21: Transformations AST

Examples

@ CompileStatic

Permet de compiler statiquement un code. Son bytecode sera plus proche de celui de Java, ce qui améliorera ses performances, même si certaines fonctionnalités dynamiques ne seront pas disponibles.

```
@groovy.transform.CompileStatic
class ListMath {
    def countSize(List<String> strings) {
        strings.collect { it.size() }.sum()
    }
}


assert new ListMath().countSize(["a", "bb", "ccc"]) == 6
```

Lire Transformations AST en ligne: https://riptutorial.com/fr/groovy/topic/4635/transformations-ast

Chapitre 22: Utilisez ConfigSluper (au lieu des fichiers de propriétés)

Introduction

ConfigSlurper vous permet d'utiliser un autre script groovy en tant que fichier de configuration pour votre script au lieu d'utiliser, par exemple, un fichier .properties. Vous pouvez faire des configurations intéressantes avec des propriétés typées et vous n'avez pas besoin de convertir à partir d'une chaîne. Vous pouvez utiliser des listes, des cartes ou une valeur basée sur un calcul ou une fermeture.

Examples

ConfigSlurper en utilisant string, number, boolean ou list

Dans le fichier myConfig.groovy est le contenu suivant.

```
message = 'Hello World!'
aNumber=42
aBoolean=false
aList=["apples", "grapes", "oranges"]
```

Ensuite, dans votre script principal, vous créez un ConfigSlurper pour votre fichier myConfig.groovy , qui est vraiment un autre script groovy.

```
config = new ConfigSlurper().parse(new File('/path/to/myConfig.groovy').toURL())
```

Ensuite, pour utiliser les éléments de la configuration, vous pouvez simplement vous y référer.

```
assert 'Hello World!' == config.message
assert 42 == config.aNumber
assert false == config.aBoolean
assert ["apples", "grapes", "oranges"] == config.aList
```

Lire Utilisez ConfigSluper (au lieu des fichiers de propriétés) en ligne:
https://riptutorial.com/fr/groovy/topic/8291/utilisez-configsluper--au-lieu-des-fichiers-de-proprietes-

Chapitre 23: Visibilité

Examples

Les champs et méthodes privés ne sont pas privés dans groovy

```
class MyClass {
    private String privateField
}

def prvtClss = new MyClass(privateField: 'qwerty')
println prvtClss.privateField
```

nous imprimera 'qwerty'

Ce problème est connu depuis la version 1.1 et il existe un rapport de bogue à ce sujet: http://jira.codehaus.org/browse/GROOVY-1875 . Il n'est pas résolu même avec groovy 2 release.

Lire Visibilité en ligne: https://riptutorial.com/fr/groovy/topic/6522/visibilite

Crédits

S. No	Chapitres	Contributeurs
1	Commencer avec groovy	Andrii Abramov, Ashish Patel, Bill K, cjstehno, Community, Eric Siebeneich, Gergely Toth, IronHorse, lospejos, Michael Schaefer, mnd, Piotr Chowaniec, Rao, rdmueller, SerCe
2	Chaînes et littéraux GString	Gergely Toth, OsaSoft, Rao
3	Currying	Gergely Toth
4	Fermetures	Andrii Abramov, Anshul, August Lilleaas, Ben, Craig Trader, Eric Siebeneich
5	Fonctions mémo	mnoronha, OsaSoft
6	Groovy Code Golfing	Charanjith A C
7	Groovy Truth (la vérité)	ARA, Piotr Chowaniec
8	Interpolation de chaîne	Aseem Bansal, Gergely Toth, jdv, mnoronha
9	JSON	albciff, Batsu, nachoorme, Stefan van den Akker
10	Langues spécifiques au domaine	gclaussn
11	Manières d'itération à Groovy	Anshul, dsharew
12	Méthodes de mémorisation de fermeture	Gergely Toth, hippocrene, John Mercier, mnoronha
13	Opérateur d'épandage	adarshr, albciff, Batsu, Eric Siebeneich, Martin Neal
14	Opérateur de navigation sécurisé	Batsu, mnoronha
15	Opérateur de vaisseau spatial	Batsu, injecteer, jwepurchase, mnoronha

16	Opérateurs de collecte	Batsu, Bill K, mnoronha, traneHead
17	Opérateurs Ternaires et Elvis	cjstehno, fheub, Piotr Chowaniec
18	RESTClient	sm4
19	Traits	NachoB, Piotr Chowaniec, Rotem
20	Transformations AST	mnoronha, Will
21	Utilisez ConfigSluper (au lieu des fichiers de propriétés)	dallyingllama
22	Visibilité	Anton Hlinisty, Gergely Toth