Groovy Concepts

Alexis Plantin

Summary

- Qu’est ce que Groovy?
- Groovy Features, Partie 1
- Groovy Classes & POGOs
- Groovy & Java
- Groovy Features, Partie 2

Qu’est ce que Groovy?

Qu’est ce que Groovy?

- Un langage dynamique pour la JVM (2nd langage sur la JVM)
- Un langage orienté objet
- Inspiré de Smalltalk, Ruby…
- Né en 2003
- Release 1.0 en 2007

Qu’est ce que Groovy?

- Groovy ressemble à Java
  - Facile à apprendre pour un développeur Java
  - Plus simple que java pour les débutants
- Groovy est intégré avec Java
  - On peut mixer Groovy et Java ensemble

Qu’est ce que Groovy?

- Le code Groovy est d’abord compilé en bytecode
- Le bytecode est ensuite interprété

Groovy Features, Partie 1

Basics

- Basics

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

- Closures

```
def square = { return it * it }
square 2
```

```
def mult = { n1, n2 -> return n1 * n2}
mult 3, 15
```

Basics

- Exercice: La closure pow

GStrings

```
def name = 'Alexis'
println "Hello $name"
println "Hello ${name}"
```

```
def name = 'Alexis'
println """Hello ${name},

Today we are the ${new Date()}

                Good Bye"""
```

GStrings

- Exercice: afficher FirstName + LASTNAME

Lists, Maps, Ranges

```
def simpleList = [1, 2, 3]
simpleList.each { num ->
    println num
}

simpleList << 'one' << 'two' << 'three'
println simpleList
println "Size : ${simpleList.size()}"
simpleList.each { element->
    println element.class.name
}
println simpleList.findAll { element ->
    element.class.name == 'java.lang.String'
}
```

Lists, Maps, Ranges

- Exercice: afficher tous les éléments qui sont supérieurs ou égal à 2 dans la liste suivante [1, 2, 3]

Lists, Maps, Ranges

```
def annuary = [ Paolo: '1111111', Juergen: '2222222']
annuary.each { element ->
    println "${element.key}'s identifier is ${element.value}"
}
annuary.Alexis = '2187989'
annuary.each { key, value ->
    println "${key}'s identifier is ${value}"
}
```

Lists, Maps, Ranges

```
def range = 1..5
println range

range = 1..<5
println range

for(i in range) {
    println "Hello " + i
}
```

Groovy Classes & POGOs

Groovy Classes & POGOs

```
class AlbumDemo {

    // statically typed property
    String title

    // dynamically typed property
    def price

    def printAlbum() {
        println title + ': ' + price
    }

    static void main(String[] args) {
        def album = new AlbumDemo(title: 'Best Of 2010', price:
20.5)
        album.printAlbum()
    }
}
```

Groovy Classes & POGOs

- POJO (Plain Old Java Object)
  - A besoin de constructeurs
  - A besoin de getters et de setters

- Les développeurs déclarent des attributs
  - Les IDEs peuvent générer tous les constructeurs, getters et setters

- Idée Groovy
  - Si l'IDE peut générer tout ce code, pourquoi le compilateur ou le runtime ne pourrait pas?

Groovy Classes & POGOs

```
class Person {
     String firstName
     String lastName
 }

 def person1 = new Person(firstName: 'Paolo', lastName:
'Caroglia')
 println person1.firstName
 println person1.lastName


 def person2 = new Person()
 person2.firstName = 'Juergen'
 person2.lastName = 'Voigt'
 println person2.firstName
 println person2.lastName
```

Groovy Classes & POGOs

```
class Person {
     String firstName
     String lastName
     String getFirstName() {
          return firstName.toUpperCase()
      }
 }

 def person1 = new Person(firstName: 'Paolo', lastName:
'Coraglia')
 println person1.firstName
 println person1.lastName
```

Groovy & Java

Java Code

```
// PrintChristmasDay.java
import java.util.Calendar;
import java.util.Date;

public class PrintChristmasDay {

    public static void main(String[] args) {
        Calendar calendar = Calendar.getInstance();
        calendar.clear();
        calendar.set(Calendar.MONTH, Calendar.DECEMBER);
        calendar.set(Calendar.DATE, 25);
        calendar.set(Calendar.YEAR, 2010);
        Date time = calendar.getTime();
        System.out.println(time);
    }
}
```

Groovy Code

```
// PrintChristmasDay.groovy
import java.util.Calendar;
import java.util.Date;

public class PrintChristmasDay {

    public static void main(String[] args) {
        Calendar calendar = Calendar.getInstance();
        calendar.clear();
        calendar.set(Calendar.MONTH, Calendar.DECEMBER);
        calendar.set(Calendar.DATE, 25);
        calendar.set(Calendar.YEAR, 2010);
        Date time = calendar.getTime();
        System.out.println(time);
    }
}
```

Pas d'imports de util

```
// PrintChristmasDay.groovy

public class PrintChristmasDay {

    public static void main(String[] args) {
        Calendar calendar = Calendar.getInstance();
        calendar.clear();
        calendar.set(Calendar.MONTH, Calendar.DECEMBER);
        calendar.set(Calendar.DATE, 25);
        calendar.set(Calendar.YEAR, 2010);
        Date time = calendar.getTime();
        System.out.println(time);
    }
}
```

Pas de point virgule

```
// PrintChristmasDay.groovy

public class PrintChristmasDay {

    public static void main(String[] args) {
        Calendar calendar = Calendar.getInstance()
        calendar.clear()
        calendar.set(Calendar.MONTH, Calendar.DECEMBER)
        calendar.set(Calendar.DATE, 25)
        calendar.set(Calendar.YEAR, 2010)
        Date time = calendar.getTime()
        System.out.println(time)
    }
}
```

Pas de Getters

```
// PrintChristmasDay.groovy

public class PrintChristmasDay {

    public static void main(String[] args) {
        Calendar calendar = Calendar.instance
        calendar.clear()
        calendar.set(Calendar.MONTH, Calendar.DECEMBER)
        calendar.set(Calendar.DATE, 25)
        calendar.set(Calendar.YEAR, 2010)
        Date time = calendar.time
        System.out.println(time)
    }
}
```

Pas de Static typing

```
// PrintChristmasDay.groovy

public class PrintChristmasDay {

    public static void main(String[] args) {
        def calendar = Calendar.instance
        calendar.clear()
        calendar.set(Calendar.MONTH, Calendar.DECEMBER)
        calendar.set(Calendar.DATE, 25)
        calendar.set(Calendar.YEAR, 2010)
        def time = calendar.time
        System.out.println(time)
    }
}
```

Pas de System.out

```
// PrintChristmasDay.groovy

public class PrintChristmasDay {

    public static void main(String[] args) {
        def calendar = Calendar.instance
        calendar.clear()
        calendar.set(Calendar.MONTH, Calendar.DECEMBER)
        calendar.set(Calendar.DATE, 25)
        calendar.set(Calendar.YEAR, 2010)
        def time = calendar.time
        println(time)
    }
}
```

Pas de Class

```
// PrintChristmasDay.groovy

def calendar = Calendar.instance
calendar.clear()
calendar.set(Calendar.MONTH, Calendar.DECEMBER)
calendar.set(Calendar.DATE, 25)
calendar.set(Calendar.YEAR, 2010)
def time = calendar.time
println(time)
```

Pas de Parenthèses

```
// PrintChristmasDay.groovy

def calendar = Calendar.instance
calendar.clear()
calendar.set Calendar.MONTH, Calendar.DECEMBER
calendar.set Calendar.DATE, 25
calendar.set Calendar.YEAR, 2010
def time = calendar.time
println time
```

Metaprogramming

```
// PrintChristmasDay.groovy

def calendar = Calendar.instance
 calendar.with {
     clear()
     set MONTH, DECEMBER
     set DATE, 25
     set YEAR, 2010
     println time
}
```

De Java à Groovy

```
// PrintChristmasDay.java
import java.util.Calendar;
import java.util.Date;

public class PrintChristmasDay {

    public static void main(String[] args) {
        Calendar calendar = Calendar.getInstance();
        calendar.clear();
        calendar.set(Calendar.MONTH,
Calendar.DECEMBER);
        calendar.set(Calendar.DATE, 25);
        calendar.set(Calendar.YEAR, 2010);
        Date time = calendar.getTime();
        System.out.println(time);
    }
}
```

```
// PrintChristmasDay.groovy

def calendar =
Calendar.instance
 calendar.with {
     clear()
     set MONTH, DECEMBER
     set DATE, 25
     set YEAR, 2010
     println time
}
```

Qu’est ce qui est identique?

- Keywords et statements
- Gestion des exceptions (try/catch/finally)
- Class, interface, attributs et définitions de méthodes
- Instanciation des objets avec l’opérateur new
- Packaging et imports
- Operateurs, expressions et assignement
- Structures de contrôles
- Commentaires
- Annotations, Generics, static imports, enum types

Groovy Features, Partie 2

Tout est objet

- Méthodes sur les primitives

```
println 'Loop 1'
3.times {
    println it
}
println 'Loop 2'
3.upto(9) {
    println it
}
println 'Loop 3'
3.step(9, 3) {
    println it
}
```

Tout est objet

- Surcharge d'opérateurs

```
class Complex {
    Double re
    Double im

    String toString() {
        re + ' + ' + im + 'i'
    }

    Complex plus(Complex c) {
        new Complex(re: this.re + c.re, im: this.im + c.im)
    }
}

def c1 = new Complex(re:2, im:3.5)
def c2 = new Complex(re:4.2, im:7.1)
println 'c1 : ' + c1
println 'c2 : ' + c2
def result = c1 + c2
println 'result : ' + result
```

Tout est objet

- Groovy et la vérité

```
def nullVariable = null
def emptyString = ''
def zero = 0

if(nullVariable) {
    println 'A null variable is not evaluate to null'
}
else {
    println 'A null variable is evaluate to null'
}

if(emptyString) {
    println 'An empty string is not evaluate to null'
}
else {
    println 'An empty string is evaluate to null'
}

if(zero) {
    println 'zero is not evaluate to null'
}
else {
    println 'zero is evaluate to null'
}
```

Metaprogramming

```
def text = 'hello world!'
text.metaClass.methods.each { method ->
    //println method.name
}

String.metaClass.capitalize = {
    def firstLetter = delegate[0]
    def rest = delegate.substring(1)
    firstLetter.toUpperCase() + rest
}
println text.capitalize()
```

GPath

- Path expression language integré à Groovy

```
def xml = '''
<persons>
    <person id="1">
        <firstname>Paolo</firstname>
        <lastname>Coraglia</lastname>
    </person>
    <person id="2">
        <firstname>Juergen</firstname>
        <lastname>Voigt</lastname>
    </person>
    <person id="3">
        <firstname>Jean</firstname>
        <lastname>Damay</lastname>
    </person>
    <person id="4">
         <firstname>Alexis</firstname>
         <lastname>Plantin</lastname>
     </person>
</persons>
'''
```

GPath

- Navigation is easy

```
def persons = new XmlSlurper().parseText(xml)
def firstnames = persons.person.firstname
println firstnames.collect {it.text()}
def lastnames = persons.person.lastname
println lastnames.collect {it.text()}
```

Nouveux opérateurs

- Opérateur Elvis
  - Une version raccourcie de l'opérateur ternaire

```
def variable = null
variable ?: 'The variable is null'
```

- Opérateur Safe Navigation
  - Utilisé pour éliminer les NullPointerException

```
// this might be null if 'admin' does not exist
def user = User.find( "admin" )
// streetName will be null if user or user.address is null - no
NPE thrown
def streetName = user?.address?.street
```

Questions?