Groovy

Desenvolvimento ágil para plataforma Java

Lucas Mazzega Menegucci

Rodolfo Gobbi de Angeli

Ricardo Natale

Introdução ao Groovy

- Tudo começou em 29 de Agosto de 2003, com uma publicação no blog de james Strachan, o primeiro artigo sobre o que viria a ser groovy.

O Projeto Groovy

Segundo James:

“Minha idéia inicial é fazer uma linguagem Dinâmica, que seja compilada diretamente em classes Java e que tenha toda a produtividade e elegância encontrada em Ruby e Python”

O Projeto Groovy

- James uniu-se com Bob McWhirter, e juntos, em 2003 fundaram oficialmente o projeto Groovy.
- Em 2004, fundação do Groovyone;
- Mais desenvolvedores se uniram ao projeto

Entre eles Guillaume Laforge, hoje é o gerente oficial do projeto Groovy e líder do Expert Group que padroniza Groovy.

Groovy Hoje

- Linguagem com ascenção mais rápida no indice do Tiobe Index;
- Do limbo para o 38° lugar entre 200 linguagens em 2 anos e meio de vida;
- Hoje ocupa a 44° posição;
- http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html
- 1° lugar entre as linguagens da JVM.

Características do Groovy

- Linguagem ágil e dinâmica para java;
- Muitos recursos foram inspirados em linguagens como Ruby e Python;
- Sintaxe simples, enxuta e similar à linguagem Java;
- Interpretada pela JVM em tempo de execução;
- Linguagem de Scripts;
- Open Source;

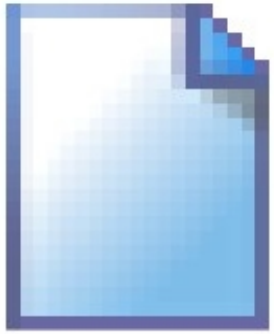
Características do Groovy

- Possui tipagem dinâmica;
- Possui tipagem forte;
- Totalmente OO;
- Não há tipos primitivos;
- TUDO é objeto;
- Possui alta legibilidade, redigibilidade e confiabilidade.
- Meta-Programação;
- Read–eval–print loop (REPL).

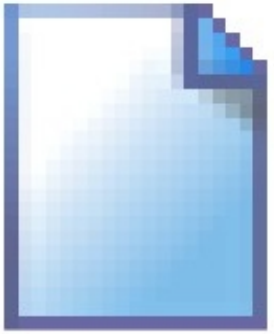
Compilação e Interpretação

- Groovy é semelhante ao java no quesito compilação e interpretação. Primeiramente é escrito um arquivo *file.groovy*, em seguida é gerado um *bytecode* que é interpretado pela máquina virtual, assim podemos estabelecer uma relação de igualdade com linguagem java onde o nível *bytecode* é equivalente entre as linguagens.

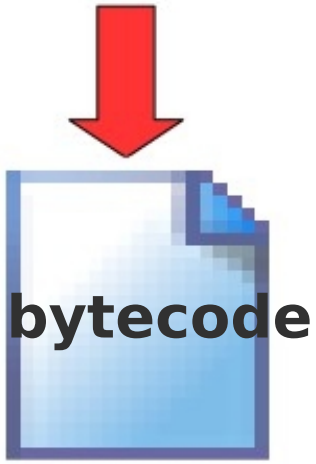
Tratamento com a JVM

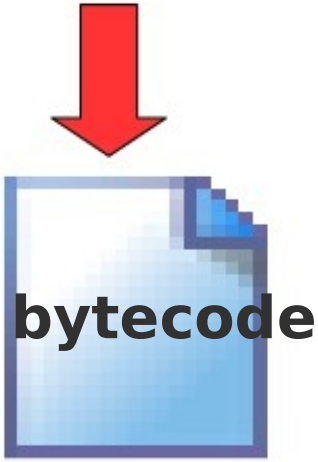


file.java



file.groovy





Máquina Virtual Java

Sintaxe

- Alguns Imports já implícitos:
  - java.io.*
  - java.lang.*
  - java.math.BigDecimal
  - java.math.BigInteger
  - java.net.*
  - java.util.*
  - groovy.lang.*
  - groovy.util.*

Operadores

- Todo operador
  - corresponde
  - a um método:

Operador	Método
a + b	a.plus(b)
a - b	a.minus(b)
a * b	a.multiply(b)
a ** b	a.power(b)
a / b	a.div(b)
a % b	a.mod(b)
a | b	a.or(b)
a & b	a.and(b)
a ^ b	a.xor(b)
a++ or ++a	a.next()
a-- or --a	a.previous()
a[b]	a.getAt(b)
a[b] = c	a.putAt(b, c)
a << b	a.leftShift(b)
a >> b	a.rightShift(b)
switch(a) { case(b) : }	b.isCase(a)
~a	a.bitwiseNegate()
-a	a.negative()
+a	a.positive()

Operador	Método
a == b	a.equals(b)
a != b	!a.equals(b)
a <=> b	a.compareTo(b)
a > b	a.compareTo(b)>0
a >= b	a.comparteTo(b)>=0
a < b	a.compareTo(b)<0
a <= b	a.compareTo(b)<=0

Operadores

- Aritméticos: +, -, *, /, %, **, ... etc
- Comparação: >, <, >=,==, !=,<=> ... etc
- Atribuição: =,+=, +=, <<, <<=, ... etc.
- Lógicos: &&, ||, ! ... etc
- Ternário: a == 2 ? “Dois” : “Outro”
- Elvis Operator: ?:
- Safe Navigation Operator: ?.

Operadores

- **(<=>)**
  - Operador de comparação que retorna -1, 0 ou 1 se for menor, igual ou maior que seu argumento.
- **(?:)** - Elvis Operator

```
String a;
a?:"A variável a é nula" //em Groovy

String a;  //em Java
a != null ? a : "A variável a é nula";
```

  - Se a variável não for nula, retorna ela mesma, senão, retorna a String.

Operadores

- **(?.)** - Safe Navigation Operator
  - Acessa um método ou propriedade se o objeto não for nulo.

```
obj?.facaAlgo() //em Groovy

//em Java
if(obj != null){
 obj.facaAlgo();
 }
```

Tipos

- Diferente do Java, para o groovy tudo é objeto.
  O groovy não possui tipos primitivos como os de Java.

  ```
  int numeroQueParecePrimitivo = 1
  ```

- Embora a declaração acima seja do tipo primitivo(int), o groovy considera a variável como sendo da classe integer.

Tipos

Tipo primitivo	Tipo Groovy
byte	java.lang.Byte
short	java.lang.Short
int	java.lang.Integer
long	java.lang.Long
float	java.lang.Float
double	java.lang.Double
char	java.lang.Character
boolean	java.lang.Boolean

Variáveis

- São objetos das classes das quais foram tipadas;
- Podem ser tipadas dinamicamente através da palavra **def**;

```
String str = 'String tipada'
def s = 'String dinâmica'
```

- Atribuição Múltipla:

```
def (a,b,c) = [2, 4, 6]
print a          //===> 2
```

Números

- Em Groovy, os números são tipados através do seu tamanho:

```
1.class                  //java.lang.Integer
1.class.superclass       //java.lang.Number

1000000000.class         //java.lang.Long
(100**100).class         //java.math.BigInteger
```

- Groovy suporta números de tamanhos imensos. Os números da classe BigInteger se restringem apenas à memória e ao processador da máquina.

Números

- Alguns métodos da classe Number:
  - times(Closure closure)

```
5.times {print 'k'}
//kkkkk
```

  - upto(Number to, Closure closure)

```
1.upto 5, {print it }
//12345
```

  - step(Number to, Number stepNumber, Closure c)

```
0.step( 10, 2 ) {print it}
//2,4,6,8,10
```

Strings

- String é um conjunto de caracteres.
- Strings Groovy são completamente MUTÁVEIS.
- Pode ser delimitado por aspas 'simples' ou "duplas"
- Pode ocupar uma ou várias linhas.
- Pode ser injetada a ela operações, variáveis...
- Métodos destrutivos e não destrutivos

Strings

- Strings interpoladas. Permitem adicionar pedaços de código dentro delas.

```
"1 + 1 = ${1+1}" // 1 + 1 = 2
```

- Strings como Arrays

```
s = 'Ola sou groovy'
s.getAt(1)    /* l */
s[1]          /* l */
s[0,13]       /* Oy */
s[8..13]      /* groovy */
```

Strings

- Brincando com os Operadores

```
s = 'Ola eu sou'

s + 'Groovy'          /* Ola eu sou Groovy */


s - 'sou' + 'Groovy'       /*Ola eu Groovy*/

s = 'k'
s * 3         /* kkk */
```

- Obs.: Os operadores são métodos não-destrutivos.

Strings

- Outros métodos interessantes:

```
s = 'hello, Groovy'
s.capitalize()          /*Hello, Groovy*/
s.center(15)            /* hello, Groovy */


s.padLeft(16, '-')      /*---hello, Groovy*/
s.padRight(16, '-')     /*hello, Groovy---*/
s.reverse()             /*yvoorG ,olleh*/
```

Strings

- Interpolação de valores em Strings;
- Possível apenas em Strings com aspas duplas.

```
def nome = 'João'
print "Olá, ${nome}, bem vindo ao curso!"
//===>Olá, João, bem vindo ao curso!'
```

Coleções

- **List -** Coleção de objetos;

```
def numeros = [1, 2, 3.5]
def string = ["1", '2', "3"]
def tudo = [10, "50", """Hello"""]
def list = [numeros, string, tudo]

list.each{println it}

//[1, 2, 3.5]
//[1, 2, 3]
//[10, 50, Hello]
```

Coleções

- **List** - Alguns métodos úteis:

```
def list = [1,2,3,9,5]

list.each {print it }          //123495
list.reverse()                 //[5,9,3,2,1]
list.putAt(3,4)                //[1,2,3,4,5]
list << 6                      //[1,2,3,9,5,6]
list.pop()                     //[1,2,3,9,5]

list + 10                      //1,2,3,4,9,5,10
list - 9                       //1,2,3,4,5
```

Coleções

- **Range**
  - Representa um intervalo de valores;

```
def range = (1..4)

range.each {print "$it "}
//1 2 3 4===> 1..4

('a'..'c').each {print "$it "}
//a b c ===> a..c

for (i in 1..10) {println i}
```

Coleções

- **Map**
  - O mesmo que array associativo, dicionários ou hashes;
  - Cada entrada é composta por uma chave e um valor;

```
def frameworks = [groovy: "Grails", ruby: "Rails"]
def map = [a:(1..4), 10:"Dez"]


    print framework.groovy //===>Grails
    print map.a            //===>1..4
    print map[10]          //===>Dez
```

Entrada de dados (Teclado)

- Para receber valores do usuário em Groovy, Pode utilizar-se do seguinte código:

```
System.in.withReader {
    print "Digite algo: "
    valor = it.readLine()
    println "Você me disse: ${valor.toUpperCase()}"
}
```

Arquivos

- A leitura e escrita de arquivos no groovy é feita utilizando as bibliotecas do Java : JavaReader, JavaWriter, InputStream e OutputStream.

- Utiliza-se ainda a classe File, para manipular um arquivo.

```
new File("file.txt")    //construtor que abre o arquivo
```

Estruturas de Controle

- O que é verdadeiro em Groovy?
  - Objetos não-nulos;
  - Números diferentes de 0;
  - Strings, GStrings, coleções não-vazias;

```
assert !'';
assert !0;
assert ![];
assert !false;
assert !null;
```

```
assert 'Olá'
assert 50
assert [2,'a']
assert true
assert new Object()
```

- Método *assert* é um método para testes unitários. Se retornar algo false, gera uma exceção.

Estruturas de Controle

- Estruturas de decisão:
  - If else

```
If(10 > 5){
    print 'Groovy'
    }
else{
    print 'Java'
}
```

  - Operador condicional - ?:

```
def retorno = (1==1) ? "Iguais" : "Diferentes"
```

Estruturas de Controle

- Switch

```
switch(ling) {
case 'Ruby'    : println 'Rails'   ;break
case 'Groovy'  : println 'Grails'  ;break
case 'PHP'     : println 'Cake'    ;break
default        : println 'Outra'   ;break
}
```

```
if(ling.isCase('Ruby'))  println 'Rails'
else if (ling.isCase('Groovy')) println 'Grails'
else if (ling.isCase('PHP')) println'Cake'
else 'Outra'
```

Estruturas de Controle

- Switch
  - Em Groovy, o switch suporta vários tipos, e não só chars.

```
switch (10) {
case 0              : assert false ;break
case 0..9           : assert false ;break
case [8,9,11]       : assert false ;break
case Float          : assert false ;break
case {it%3 == 0}    : assert false ;break
case ~/../          : assert true  ;break
default             : assert false ;break
}
```

Estruturas de Controle

- while

```
def list = [1,2,3]
while (list) {
   list.remove(0)
}
assert list == []


while (list.size() < 3)
   list << list.size()+1
assert list == [1,2,3]
```

Estruturas de Controle

- for

```
def store = ''
for (String i in 'a'..'c')  store += i
assert store == 'abc'


def j = 0
for (i in 0..10) {
    j += i }
print j     //55
```

Exceções

- O tratamento de exceções é igual ao do Java.

A única diferença em termos de exceção é que no Groovy é opcional colocar o throws na assinatura do método, tanto para exceções checadas quanto em não-checadas;

- Com isso, podemos escolher como e quando tratar essas exceções.

Exceções

```
def myMethod() {
   'string'.toLong()
   }

def log = []
try {
    myMethod()
    } catch (Exception e) {
       log << e.toString()
    } finally {
    log << 'finally'
  }
  log.each{println it}
```

Funções

- Funções em groovy são blocos de códigos identificados por um nome, e podem ou não receber parâmetros. Semelhante a c e Java, exige “{}” para indicar o início e o fim da função.

```
def func(){ /* Escopo do Código */}
```

- Se uma variável local for definida dentro do escopo da função, onde esta variável possui o mesmo nome de uma variável global, a variável local ofusca a global no escopo da função.

Métodos

- Em Groovy, todo método retorna alguma coisa. Quando nenhum valor é explicitamente mostrado, ele retorna null.
- O return opcional

```
def hello(nome){
   println 'Hello, meu nome é: ${nome}'
}

hello('Bruno')  //Hello, meu nome é: Bruno
     ou
hello 'Bruno'   //Hello, meu nome é: Bruno
```

Métodos

- Podemos declarar métodos com parâmetros com valores default

```
def soma(a,b = 2){ println a + b }
soma 2        //4
soma 2,3      //5
```

- Métodos com parâmetros opcionais

```
int sum(a, Object[] optionals){
   for(o in optionals) a+=o
   a
}
sum(1) //1
sum(20, 1, 10, 4) //35
```

Closures

- Um bloco de código reutilizável delimitado por chaves.
- Semelhante a uma classe interna;
- Podem ser declarados não só dentro das classes
- Não é executado quando definido, apenas quando for chamado;
- Apenas agem como métodos, mas são objetos Normais (groovy.lang.Closure).

Closures

- Pode conter parâmetros;
- Modifica variáveis declaradas fora da closure;
- Invocados pelo método
  - *call();*
  - *doCall();*
  - ou simplesmente pela seu nome;
- Se tiver apenas um parâmetro, não é necessário defini-lo, basta usar a palavra reservada *it* para referenciá-lo.

Closures

```
Closure cloj1 = {println 'Hello, World!'}
// Sem parâmetro

def cloj2 = { obj -> println "Hello, $obj!"}
// Parâmetro sem tipo definido

def cloj3 = {println "Hello, $it!"}
// Parâmetro acessado pela palavra-chave 'it'

cloj2.call('Groovy')    //===> Hello, Groovy
cloj3.doCall('Groovy')  //===> Hello, Groovy
cloj3('Groovy')         //===> Hello, Groovy
```

Closures

- Outros exemplos:

```
def soma = { a, b ->
    print "A soma é: ${a+b}"
    }


map = ['a':1, 'b':2]
map.each{ key, value -> map[key] = value * 2 }
print map    //[a:2, b:4]


letras = 'a'..'z'
letrasUpper=[]
letras.collect (letrasUpper){it * 2 }

println letras          //[a, b, c, d, ...]
println letrasUpper     //[aa, bb, cc, ...]
```

Orientação à Objetos

- Classes e Objetos
  - Classes é uma estrutura que abstrai coisas do mundo real para o mundo computacional;
  - Define o tipo dos objetos;
    - Se classes fossem formas de bolo, os objetos seriam os bolos;
    - Objetos são instancias das classes;
  - Objetos possuem dados em comportamentos;
    - Os dados são os atributos;
    - Comportamentos são os métodos;

Orientação à Objetos

- Classes e Objetos
  - Instanciando uma classe:

```
p = new Pessoa()
```

  - Para os tipos básicos basta fazer isso:

```
s = "String"
i = 50
array = [10,20,60]
```

  - Ou do modo brutal...

```
i = new Integer(25)
s = new String('Sou uma String')
```

Orientação à Objetos

- Herança:
  - Classes tem a capacidade de herdar comportamento de outras classes;
  - As classes que provêm os comportamentos herdados são chamadas de super-classes;

```
1.class                  //java.lang.Integer
1.class.superclass       //java.lang.Number
```

  - Para herdar de outras classes usa-se o *extends*

```
class Integer extends Number {
                          }
```

Orientação à Objetos

- Herança Múltipla:
  - Uma classe herda de mais de uma classe;
  - Em groovy não existe herança múltipla, assim como não existe em Java;
  - Porém é possível simular herança múltipla utilizando interfaces. Da mesma for a como é utilizada em Java.

Passagem de Parâmetros

- Tanto Java quanto Groovy possuem passagem unidirecional por cópia para métodos.

Polimorfismo

- Sobrecarga:
  - Similar a linguagem Java, com a exceção de que em groovy, operadores podem ser sobrecarregados pelo programador.
  - Permite a sobrecarga de operadores apenas redefinindo o método do operador.
  - Aumenta a legibilidade e a redigibilidade, porém aumenta a complexidade da LP.

Meta-Programação

- MOP - *Meta-Object Protocol*
- Mágica da programação;
- Adicionar comportamento às classes em tempo de execução;
  - Adicionar métodos às classes;
  - Adicionar propriedades;
- Todas as classes escritas em Groovy herdam de Object e implementam a interface GroovyObject;

Meta-Programação

- Meta Classes
  - As classes definem o comportamento dos objetos;
  - As meta classes definem o comportamento de certas classes ou de suas instancias;
    - De certa forma até as classes são objetos...

```
public interface GroovyObject {
  Object invokeMethod(String name, Object args);
  Object getProperty(String property);
  void setProperty(String property, Object newValue);
  MetaClass getMetaClass();
  void setMetaClass(MetaClass metaClass);
}
```

Meta-Programação

- Adicionando um método à classe String

```
String.metaClass.digaOla = { lang ->
   if(lang == 'English') println 'Hello'
   else
   if(lang == 'Swedish') println 'Hej'
}

'Chamando o diga Olá por uma String'.digaOla('Swedish')

class Pessoa{
   String nome, sobrenome
}

Pessoa.metaClass.getNomeCompleto = {
   nome + ", " + sobrebome }
   println new Pessoa(nome:'groovy',
   sobrenome : 'Grails').getNomeCompleto()
```

Java Bean

```
public class HelloWorld {
    private String nome;

    public void setNome(String nome) {
        this.nome = nome;
    }

    public String digaHello(){
        return "Hello " + nome + ".";
    }

    public static void main(String[] args) {
        HelloWorld hw = new HelloWorld();
        hw.setNome("Bruno");
        System.out.println(hw.digaHello());
    }
}
```

Em groovy

```
public class HelloWorld {
    private String nome;


    public void setNome(String nome) {
        this.nome = nome;
    }

    public String digaHello(){
        return "Hello " + nome + ".";
    }
    public static void main(String[] args) {
        HelloWorld hw = new HelloWorld();
        hw.setNome("Bruno");
        System.out.println(hw.digaHello());
    }
}
```

Groovy Bean

- Por defaulf as classes são public;
- Métodos modificadores dos atributos são gerados dinamicamente;
- Não é necessário criar um método construtor:

```
p = new Pessoa(nome:'Bruno')
```

Groovy Bean

```
class HelloWorld {
   String nome

   String digaHello(){
        "Hello, ${nome}."
   }
   static void main(args) {
       def hw = new HelloWorld(nome:"Groovy")
       print hw.digaHello()
        }
 }
```

Getters e Setters gerados automaticamente

Strings Groovy (GString)

Parâmetro do construtor

Groovy Bean

```
class HelloWorld {
   def nome

   def digaHello(){
      "Hello, $nome"
       }
}

print new HelloWorld(nome:"Groovy").digaHello()
```

”Duky Typing”
Tipagem dinâmica

”Script” Groovy solto/livre

Totalmente em Groovy

Closure com parâmetro

```
class HelloWorld {
    def digaHello = {nome-> "Hello ${nome}"}
}
print new HelloWorld().digaHello.call("Bruno")
```

Chamada da Closure

- Em todos esses exemplos têm o mesmo resultado:

>>Hello, Bruno

Java vs Groovy

Em java: 18 linhas

```java
public class HelloWorld {
    private String nome;

    public void setNome(String nome) {
        this.nome = nome;
    }

    public String digaHello(){
        return "Hello " + nome + ".";
    }

    public static void main(String[] args) {
        HelloWorld hw = new HelloWorld();
            hw.setNome("Bruno");
        System.out.println(hw.digaHello());
    }
}
```

Em Groovy: 4 linhas

```groovy
class HelloWorld {
    def digaHello = {nome-> "Hello ${nome}"}
}
print new HelloWorld().digaHello.call("Bruno")
```

Avaliação da LP

- Legibilidade:
  - Apesar de manter a prioridade na redigibilidade, groovy tenta manter uma alta legibilidade.

- Redigibilidade:
  - Groovy prioriza a redigibilidade de forma que não comprometa a legibilidade;
  - Tipagem dinâmica;
  - Closures.

Avaliação da LP

- Confiabilidade:
  - Tratamento de exceções;
  - Tipagem dinâmica;
  - Tipagem Forte.
- Reusabilidade.
- Portabilidade.

Referências

- Groovy in Action;
- Introduction to Groovy and Grails
  - Mohamed Seifeddine
- Getting Started Guide
  - http://groovy.codehaus.org/Getting+Started+Guide
- Em Busca do Grails
  - Fernando Anselmo
- Groovy Documentation
  - Oficial docs
- Básico do Groovy para quem for aprender Grails
  - http://www.itexto.net/devkico/?p=231

Referências

- Um pouco de Groovy
  - Artigo Dev-Media - Java Magazine - Marcelo Castellani
- Slide curso_ruby do ENUCOMP 2008
  - Regis Pires Magalhães