Groovy

Cleisson Santos Guterres
Lucas Augusto Santos
Renato Menezes Machado

Introdução

•Groovy é uma linguagem de programação orientada a objeto para a plataforma Java.

•É dinamicamente compilada na JVM, utiliza dos códigos e bibliotecas de Java e a maioria do código é sintaticamente válido em Java puro.

•Sua primeira versão foi lançada em 2 de janeiro de 2007. A versão 2.0 foi lançada em Julho de 2012 e se espera(va) uma versão nova em 2014.

Introdução

Características

•Grande proximidade sintática com Java facilita a sua redigibilidade, legibilidade e aprendizado.

•Funcionalidades não presentes em Java inspiradas em Python, Ruby e Smalltalk.

•Também é interpretada pela JVM em tempo de execução.

•Compila diretamente para bytecode Java (bytecodes gerados pelas duas linguagens são literalmente idênticos).

Introdução

Características

•A grande maioria de arquivos escritos em Java também são válidos em Groovy.

•É uma linguagem de tipagem dinâmica.

•Possível utilização de meta-programação.

•Vários pacotes e classes já importadas como java.lang, java.io e java.util.

Palavras Reservadas

as	assert	break	case
catch	class	const	continue
def	default	do	else
enum	extends	false	finally
for	goto	if	implements
import	in	instanceof	interface
new	null	package	return
super	switch	this	throw
throws	true	try	while

Identificadores

- O nome das variáveis pode conter letras, números, dolar sign($) e underscore(_). Porém não podem começar com um número.
- A linguagem é case sensitive.

Exemplos:

```
2 def var, Var, $var, _var, _var$30F
3 //0x00AC não é possível pois inicia com um número
```

Identificadores

- Podemos usar identificadores com aspas, eles são úteis pois podem conter caracteres ilegais em nomes de váriaveis como por exemplo exclamação(!), hífen(-), espaço.
- Estes identificadores são usados sempre depois de pontos.

Exemplo:

```
5  def mapa = [:]
6  mapa.'identificador com espaço!' = true
7  mapa."outro-com-hífen" = true
```

Escopo

- Escopo estático com blocos aninhados.

Sobrecarga de identificadores

- Em uma mesma classe pode existir uma variável e um método com mesmo nome.

Sobrecarga de métodos

- Subclasses podem sobrescrever métodos de suas superclasses.

Expressões e Comandos

Operações aritméticas simples:

- Soma
- Subtração
- Multiplicação
- Divisão
- Módulo
- Potência

```
15  assert  1  + 2 == 3
16  assert  4  - 3 == 1
17  assert  3  * 5 == 15
18  assert  3  / 2 == 1.5
19  assert 10  % 3 == 1
20  assert  2 ** 3 == 8
```

Expressões e Comandos

Operadores Unários

```
22  assert +3 == 3
23  assert -4 == 0 - 4
24  assert -(-1) == 1
25
26  def a = 2
27  def b = a++ * 3
28  assert a == 3 && b == 6
29
30  def c = 3
31  def d = c-- * 2
32  assert c == 2 && d == 6
```

```
34  def e = 1
35  def f = ++e + 3
36  assert e == 2 && f == 5
37
38  def g = 4
39  def h = --g + 1
40  assert g == 3 && h == 4
```

Expressões e Comandos

Operações aritméticas com atribuição

```
42  def a = 4
43  a += 3
44  assert a == 7
45
46  def b = 5
47  b -= 3
48  assert b == 2
49
50  def c = 5
51  c *= 3
52  assert c == 15
```

```
54  def d = 10
55  d /= 2
56  assert d == 5
57
58  def e = 10
59  e %= 3
60  assert e == 1
```

Expressões e Comandos

Operadores relacionais

```
62  assert 1 + 2 == 3
63  assert 3 != 4
64
65  assert -2 < 3
66  assert 2 <= 2
67  assert 3 <= 4
68
69  assert 5 > 1
70  assert 5 >= -2
```

Expressões e Comandos

Operadores lógicos

```
72  assert !false
73  assert true && true
74  assert true || false
```

Precedência nestes operadores

```
76  assert !false && true
77  assert false || true && true
```

Existe curto-circuito.

Expressões e Comandos

Operadores bit-a-bit

```
73  assert (1 << 2) == 4
74  assert (4 >> 1) == 2
75  assert (15 >>> 1) == 7
76  assert (3 | 6) == 7
77  assert (3 & 6) == 2
78  assert (3 ^ 6) == 5
79  int mostlyOnes = 0xFFFFFFFE
80  assert ~mostlyOnes == 1
```

Expressões e Comandos

Operador Ternário e Elvis Operator

```
90  def a=21
91  println a==21?"Vinte e um":"Outro"  //Ternário
92  println a?:"Vinte e um"             //Elvis
93
94  String str
95  println str?:"A variável str é nula"  //Elvis
96  str="Músico"
97  println str?:"A variável str é nula"  //Elvis
```

Expressões e Comandos

Operadores Spaceship(<=>) e Safe Navigation(?.)

```
 99  def x=10, y=9, z=11, w=10
100  println x<=>y
101  println x<=>w
102  println x<=>z
103
104  objeto?.metodo
```

Expressões e Comandos

Estruturas condicionais de controle

```
104 def x=15
105 if(x>5)
106     println x
107 else println 0
108
109 switch(x){
110     case 5:
111         assert false
112         break
113     case 10:
114         assert false
115         break
116     case 15:
117         assert true
118         break
119 }
```

Expressões e Comandos

Estruturas iterativas de controle e Escapes

```
121 def x=15
122 while(x>10){
123     println x
124     x--
125 }
126
127 for(String z in 'a'..'f')
128     print z
```

```
130 for(y in 4..15){
131     if(y%2==0){
132         println y
133         continue
134     }
135     else break
136     y--
137 }
```

Ausência de Tipos Primitivos

Em Groovy, tudo é objeto (diferente de Java). “Tipos primitivos” são na verdade pertencentes à respectiva classe wrapper.

int Pareco_Int_Mas_Sou_Integer = 1

- Semelhante ao autoboxing de Java 5

Tipo Boolean


```
groovy> String x = false
groovy> if(x){
groovy>     println("(Houve uma conversão implicita de booleano false para string \"false\")")
groovy>     println("\nEm Groovy, qualquer valor não nulo é equivalente ao valor booleano true")
groovy> }
groovy> boolean y = false
groovy> if (y){
groovy>     println("Óbvio que não vai entrar aqui!") ;
groovy> }else{
groovy>     println("Mas Groovy também possui o tipo boolean") ;
groovy> }

(Houve uma conversão implicita de booleano false para string "false")

Em Groovy, qualquer valor não nulo é equivalente ao valor booleano true
Mas Groovy também possui o tipo boolean
```

Tipo Enumerado


```
groovy> enum Dia{
groovy>     Sabado, Domingo, Segunda, Terca, Quarta, Quinta, Sexta
groovy> }
groovy> def ComoSaoAsCoisas(Dia dia) {
groovy>     switch (dia) {
groovy>         case Dia.Sabado:
groovy>         case Dia.Domingo:
groovy>             println "Finais de Semanas são divertidos!"
groovy>             break
groovy>         default:
groovy>             println "Dias normais não são tão divertidos"
groovy>             break
groovy>     }
groovy> }
groovy> ComoSaoAsCoisas(Dia.Sabado)

Finais de Semanas são divertidos!
```

Tipo Decimal

Groovy possui a classe BigDecimal (java.math.BigDecimal)

```
groovy> BigDecimal x = new BigDecimal(9950322536262374,5)

Result: 99503225362.62374
```

Tipo Ponto Flutuante

Groovy possui os tipos float e double, igual Java

Tipo Char

- Funciona como em C

```
groovy> int x = 'c'
groovy> println(x)
groovy> char c = 105
groovy> println(c)

99
i
```

Tipo String

- Parecido com Java;
- Pode ser definido por aspas simples ou duplas

```
groovy> String filme = "Avengers Age of Ultron"
groovy> String produtora = "Marvel"
groovy> filme = produtora +" - "+ filme
groovy> println(filme)

Marvel - Avengers Age of Ultron
```

Tipo Intervalo de Inteiros

Na Biblioteca groovy.lang existe a classe IntRange que implementa a interface Range

```
groovy> IntRange x = new IntRange(5,10)
groovy> IntRange y = new IntRange(6,9)
groovy> println(x.equals(y))
groovy> println(x.containsWithinBounds(7))
groovy> println(x.get(5))

false
true
10
```

Tipos Compostos

Produto Cartesiano

A biblioteca padrão groovy.lang possui a classe Tupla

```
groovy> Tuple t = new Tuple("maça",1,true)
groovy> Tuple m = new Tuple("banana",2,false)
groovy> println ( t.size() )
groovy> println ( t.equals(m) )
groovy> println ( t.get(1) )

3
false
1
```

Mapeamentos Finitos

Vetores

Semelhante a Java, checagem dinâmica dos índices

Conjunto Potência

Possui a classe Set, de java.util, que se comporta de forma semelhante a um conjunto potência

```
groovy> Set s1 = ["a", 2, true]
groovy> println(s1)
groovy> Set s2 = s1 + [false]
groovy> println(s2)

[2, a, true]
[false, true, 2, a]
```

Funções

```
def par(int n){
    return (n%2 == 0)
}

par(2)
```

Tipos Recursivos

Igual a Java.

Ponteiros e Referências

O programador não pode manipular endereços de memória. As variáveis são do tipo referência

Tipo Mapa


```
groovy> Map map= ['matricula':'???', 'nome':'Cleisson', 'n':7, 123:'laranja']
groovy> println(map)
groovy> map['nome'] = "Renato"
groovy> map[123] = 15
groovy> map = map + ['profissao':'programador']
groovy> println(map)

[matricula:???, nome:Cleisson, n:7, 123:laranja]
[matricula:???, nome:Renato, n:7, 123:15, profissao:programador]
```

Correspondência entre parâmetros

- Posicional

- Pode ser por valores default:

```
def function(x = 10, y = 0, w = 2){
    println (x + y + w)
}

function()              // imprime 12
function("oi",198)      // imprime oi1982
```

- O número de parâmetros pode variar:

```groovy
def metodo1(int... args){
    for (int i = 0; i<args.length; i++){
        println(args[i])
    }
}

metodo1(1,3,5,7,9,11)

def metodo2(Object[] args){
    for (int i = 0; i<args.length; i++){
        println(args[i])
    }
}

metodo2 ("oi","meu nome é Cleisson",true,1234,"tchau")
```

Passagem de Parâmetros


```groovy
def inicializa(int x, int[] vetor){
    x = 0
    for (int i = 0; i<vetor.length; i++){
        vetor[i] = 0
    }
    vetor = [1,2,3,5]
    println(vetor)
}

int[] vetor = new int[6]
vetor[5] = 1
vetor[2] = 3
vetor[4] = 10
int x = 100

inicializa(x, vetor)

println(vetor)
println(x)
```

Resultado do código:

```
[1, 2, 3, 5]
[0, 0, 0, 0, 0, 0]
100
```

Podemos concluir que nesse aspecto Groovy se comporta igual a Java.
Além disso, o momento da passagem é normal / eager

TADs como Classes em Groovy


```groovy
class Pessoa{
    String nome
    int idade
    Pessoa(){}
    Pessoa(String nome, int idade){
        this.nome = nome
        this.idade = idade
    }
    public int getIdade(){
        return this.idade
    }
    public String getNome(){
        return nome
    }
}

Pessoa p
String nome = p?.getNome()
Integer idade = p?.getIdade()

// sem o '?' daria erro na execução
```

Classes Internas Anônimas

Disponível a partir de Groovy 1.7

Pacotes

- Classes agrupadas em um mesmo local

```
166 package meuPacote
167
168 class primeiraClasse{} // public por default
169
170 private class segundaClasse{}
```

- Utilizando classes de um pacote

```
172 meuPacote.primeiraClasse c = new meuPacote.primeiraClasse()
173 import meuPacote.* // importa tudo que existe no pacote
174 import meuPacote.segundaClasse // importa somente a classe segundaClasse
175 segundaClasse s = new segundaClasse()
176 // depois de importada a classe não é preciso informar o pacote
```

Verificação de Tipos

• A tipagem é dinâmica, ou seja, fica a critério do programador declarar ou não o tipo da variável.

• Erros de tipos só são checados em tempo de execução

• Maior flexibilidade, menor confiabilidade

```
178  int x = "oi" // Compila, mas dá erro na execução
179  def y = "oi" // Compila e executa
```

• A partir do Groovy 2.0, se tornou possível ativar um recurso adicional para fazer tipagem estática. Antes da classe adiciona-se a seguinte anotação: `@groovy.transform.TypeChecked` (Também é necessária uma biblioteca)

Equivalência de Tipos

• Como visto anteriormente, há muitos exemplos de conversão implícita, como atribuições de char para int, de int para char, e atribuições de quase todos os tipos para String.

• Porém não são consideradas equivalentes classes diferentes implementadas pelo programador com os mesmos atributos. Ex: Uma instância da classe “pessoa” tem o atributo nome e idade, e uma instância da classe “aluno” tem o atributo nome e idade. Não é possível passar um aluno como parâmetro para um método que pede uma pessoa como parâmetro.

Polimorfismo - Coerção


```groovy
def imprimeFloat(float n){
    println(n)
}

def imprimeStrig(String n){
    println(n)
}

int x = 1

imprimeFloat(x)    // faz a conversão implícita

String s = x        // faz a conversão implícita

imprimeString(x)    // dá erro
```

Polimorfismo - Sobrecarga

- De métodos: igual Java
- De operadores: basta sobre-escrever métodos equivalentes

Operator	Method
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)

Operator	Method
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()

Operator	Method
a == b	a.equals(b) or a.compareTo(b) == 0 **
a != b	! a.equals(b)
a <=> b	a.compareTo(b)
a > b	a.compareTo(b) > 0
a >= b	a.compareTo(b) >= 0
a < b	a.compareTo(b) < 0
a <= b	a.compareTo(b) <= 0

Polimorfismo - Sobrecarga


```
class Ponto {
    public int x
    public int y
    Ponto(){
    }
    Ponto(int x, int y){
        this.x = x
        this.y = y
    }
    public Ponto plus(Ponto b){
        this.x = this.x + b.x
        this.y = this.y + b.y
        return this
    }
    public String toString(){
        return "X: "+x+", Y: "+y ;

    }
    static void main (String[] args){
        Ponto a = new Ponto(1,2)
        Ponto b = new Ponto(2,1)
        a = a + b
        println(a)
    }
}
```

Polimorfismo - Paramétrico

Tipos genéricos a partir de Groovy 1.5

Exemplo de uso na definição:

```
import java.lang.reflect.Method
Iterable<Method> methods = String.methods.grep{ it.name.startsWith('get') }
assert methods.name == [ "getBytes", "getBytes", "getBytes", "getBytes", "getChars",
"getClass" ]
```

Exemplo de uso na classe:

```
class A extends ArrayList<Long> {}

class B<T> extends HashMap<T,List<T>> {}

class C<Y,T extends Map<String,Map<Y,Integer>>> {}

class D {
    static < T > T foo(T t) {return null}
}
```

Polimorfismo - Inclusão (Herança)


```groovy
class Pessoa{
    String nome
    int idade
    Pessoa(){}
    Pessoa(String nome, int idade){
        this.nome = nome
        this.idade = idade
    }
    public void trabalhar(){
        println("Trabalhando")
    }
}

class Estudante extends Pessoa{
    String escola
    Estudante(){}
    Estudante(String nome, int idade, String escola){
        super(nome,idade)
        this.escola = escola
    }
    public void trabalhar(){
        println("Estudando")
    }
}


Pessoa p = new Estudante("Cleisson",20,"UFES")        //Ampliação
p.trabalhar()                                        // Vai imprimir "Estudando"

Estudante e = p                                     //Estreitamento
```

- Igual Java;
- Amarração tardia
- Sem herança múltipla
- Permite Ampliação e Estreitamento
- Possui Classes Abstratas e Interfaces
- Aceita Metaclasses

Variáveis e Constantes

- Para definir constantes é usada a palavra chave final.

```
final int x = 1
x = 2                    // Compila e executa!
```

- Tudo é objeto, por isso todas variáveis são armazenadas dinamicamente no monte.

Variáveis e Constantes

- Utiliza-se do coletor de lixo do Java para finalizar as variáveis.

- Possui serialização, que pode ser executada igual em Java, implementando a classe Serializable.

Variáveis e Constantes

- Não permite acesso à endereços de memória.

- Acesso à memória secundária é através de objetos de classes, como por exemplo InputStream/OutputStream e Reader/Writer.

Closures

- Um bloco de código reutilizável delimitado por chaves. Semelhante a uma classe interna.
- Pode ser definido fora de uma classe.
- Executado somente quando chamado.
- Agem como métodos, mas são objetos da classe groovy.lang.Closure .
- Pode conter parâmetros.

Closures

- Modifica variáveis fora da própria closure.
- Invocados pelos métodos *call()* ou *doCall()*, além de poder ser invocado pelo próprio nome.

- Quando possui apenas um parâmetro, não é necessário defini-lo, basta usar a palavra reservada **it** para referenciá-lo.

Closures


```
139 Closure cloj1 = {println 'Hello, World!'}
140 // Sem parâmetro
141
142 def cloj2 = { obj -> println "Hello, $obj!"}
143 // Parâmetro sem tipo definido
144
145 def cloj3 = {println "Hello, $it!"}
146 // Parâmetro acessado pela palavra-chave 'it'
147
148 cloj1() // Hello, World!
149 cloj2.call('Groovy') // Hello, Groovy
150 cloj3.doCall('Groovy') // Hello, Groovy
151 cloj3('Groovy') // Hello, Groovy
```

Closures


```
153 def soma = {a,b->print "A soma é: ${a+b}"}
154
155 map = ['a':1,'b':2]
156 map.each{key,value->map[key]=value*2}
157 print map //[a:2, b:4]
158
159 letras='a'..'z'
160 letrasUpper=[]
161 letras.collect(letrasUpper,{it*2})
162
163 println letras //[a, b, c, d, ...]
164 println letrasUpper //[aa, bb, cc, ...]
```

Groovy Beans

- Sintaxe mais simples

```groovy
class Customer {
    // properties
    Integer id
    String name
    Date dob

    // sample code
    static void main(args) {
        def customer = new Customer(id:1, name:"Gromit", dob:new Date())
        println("Hello ${customer.name}")
    }
}
```

- Prioriza redigibilidade

Exceções

•Exceções e erros são conhecidos por eventos “Throwable”.

•Erros geralmente causam a falha de um programa e o seu término, já exceções podem e devem ser tratadas no programa.

•O “throws” presente na assinatura de um método não é checado pelo compilador, já que não existe diferença entre exceções checadas e não-checadas em Groovy.

•São tratadas exatamente como em um programa Java:

```
try{
   'moo'.toLong()   // gerará uma exceção
    assert false       //assegurando que não se deve chegar nesse ponto
}catch(e){ assert e in NumberFormatException }
```

Concorrência

•Groovy usa as mesmas facilidades de Java para lidar com a concorrência e combina-as com threads e closures quando necessário.

```
import java.util.concurrent.atomic.AtomicInteger
 def counter = new AtomicInteger()
synchronized out(message) {
  println(message)
}
 def th = Thread.start {
       for( i in 1..8 ) {
  sleep 30
   out "thread loop $i"
  counter.incrementAndGet()
       }
}
```

Concorrência


```
for( j in 1..4 ) {
  sleep 50
  out "main loop $j“
   counter.incrementAndGet()
                        }
th.join()
assert counter.get() == 12
```

```
thread loop 1
main loop 1
thread loop 2
thread loop 3
main loop 2
thread loop 4
thread loop 5
main loop 3
thread loop 6
main loop 4
thread loop 7
thread loop 8
```

Comparativo entre linguagens OO

Critérios Gerais	Groovy	C++	JAVA
Aplicabilidade	Parcial	Sim	Parcial
Confiabilidade	Sim	Não	Sim
Aprendizado	Não	Não	Não
Eficiência	Parcial	Sim	Parcial
Portabilidade	Sim	Não	Sim
Método de Projeto	OO	Estruturado e OO	OO
Evolutibilidade	Sim	Parcial	Sim
Reusabilidade	Sim	Sim	Sim
Integração	Parcial	Sim	Parcial
Custo	Dependente da ferramenta	Dependente da Ferramenta	Dependente da Ferramenta

Critérios Específicos	Groovy	C++	JAVA
Escopo	Sim	Sim	Sim
Expressões e Comandos	Sim	Sim	Sim
Tipos Primitivos	Não	Sim	Sim
Gerenciamento de Memória	Sistema	Programador	Sistema
Persistência de Dados	Biblioteca de classes atualizada e modificada	Biblioteca de Classes e Funções	JDBC, Biblioteca de Classes e Serialização

Passagem de Parâmetros	Por valor e por cópia de referência	Lista variável, default, por valor e por referência	Por valor e por cópia de referência
Encapsulamento e Proteção	Sim	Sim	Sim
Sist. de Tipos	Não	Parcial	Sim
Verificação de Tipos	Estática e Dinâmica	Estática Dinâmica	Estática Dinâmica
Polimorfismo	Todos	Todos	Coerção, Sobrecarga e Inclusão
Exceções	Sim	Parcial	Sim
Concorrência	Sim	Não	Sim

Referências

- http://groovy.codehaus.org/
- Curso de Groovy e Grails ministrado no ENUCOMP 2010
- http://en.wikipedia.org/wiki/Groovy_(programming_language)