Estudo sobre a Linguagem de Programação Groovy

Aluno Bruno Rodrigues Caputo e Charles Hugo Macêdo Alencar
[1]Departamento de Ciência da Computação – Universidade Federal de Roraima (UFRR)
Boa Vista – RR – Brasil
bruno_c@mail.com, discenteufrr2016@gmail.com

Resumo. *Este projeto tem por objetivo discorrer sobre Groovy.Aprender esta linguagem, pode ser um investimento inteligente.Para, primordialmente, Java,Groovy traz benefícios e vantagens de linguagens avançadas como:*

- *"Closures",termo que se refere ao modo como funções definidas dentro de um contexto léxico, por exemplo,o corpo de uma função, um bloco, um arquivo fonte, acessam variáveis definidas nesse contexto.*
- *Métodos dinâmicos,o que pode ser visto como a declaração de um método , e modificar o comportamento a tempo de execução de código,dando mais controle ao programador.*
- *Protocolo de meta objeto, quando "Groovy chama um método" não o faz diretamente, mas sim através de camadas intermediárias para ligar e executar o comportamento esperado, que por sua vez permitem que sejam modificados internamente.O protocolo é uma coleção dessas regras e formatos que podem ser utilizados.*

1. História do surgimento da linguagem

Durante o *GroovyOne* 2004(um encontro de desenvolvedores em Londres),James Strachan(Co-fundador da linguagem) enunciou a história de como fora criada.

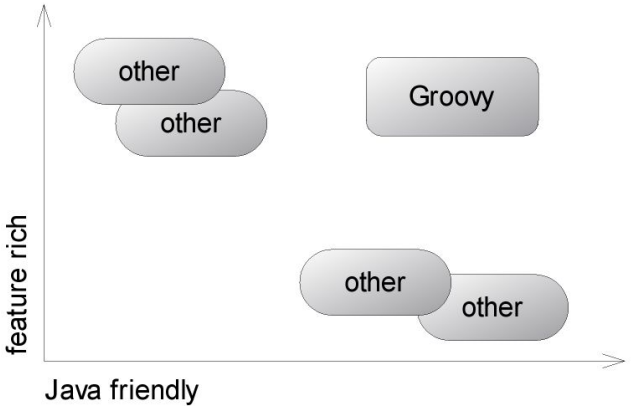
Ele e a sua esposa estavam esperando por um avião.Enquanto ela foi fazer algumas compras, ele visitou um internet café e espontaneamente decidiu entrar no site de Python e estudar a linguagem.Durante o processo de aprendizagem ele ficou cada vez mais intrigado,sendo um programador de Java experiente, ele reconheceu que muitas características úteis e interessantes de Python,como o suporte nativo para tipos de datas comum em uma sintaxe expressiva e , mais importante, o comportamento dinâmico,faltavam.Assim a ideia de integrar isto a Java nasceu.

Isto levou os princípios chaves que guiaram o desenvolvimento da linguagem,ser rica em ferramentas disponíveis e integrada a Java,trazendo os benefícios de linguagens dinâmicas para uma plataforma com bom suporte e robusta.

Uma das imagens que mostram a posição da linguagem em relação a outras do java na visão dos autores do livro "Groovy em ação,segunda edição".

Figura 1.1 : Imagem retirada do livro mencionado acima.

Figura 1.1



Várias versões foram lançadas entre 2004 e 2006.Depois da padronização que ocorreu com o Java Community Process(mecanismo formal que permite grupos para padronizar especificações técnicas para tecnologias de Java),a numeração das versões mudou e uma versão chamda “1.0” foi anunciada no segundo dia de janeiro em 2007.Depois de vários testes betas fora anunciado a versão 1.1,no dia 7/12/2007,Groovy 1,1 foi lançado. e imediatamente renomeado como Groovy 1.5 como um reflexo de muitas mudanças que foram feitas.

Em 2007,Groovy ganhou o primeiro colocado de inovação do JAX 2007.Em 2008, Grails, um framework web,ganhou o segundo colocado de inovação no JAX 2008.

Em novembro de 2008,”SpringSource” adquiriu “Groovy and Grails “.Em agosto de 2009 “VMWare” adquiriu “SpringSource”.

Em Júlio de 2009, Strachan escreveu no seu blog “Eu posso sinceramente dizer que se alguém tivesse me mostrado o livro Programando em Scala por Martin Odersky, Lex Spoon & Bill Venners em 2003.Eu provavelmente nunca teria criado o Groovy”.Strachan teria saído do projeto em silêncio um ano antes do Groovy 1.0 ter sido lançado em 2007.

De 2004 a 2012,groovy ficou praticamente inativo,somente em dia 2 de Júlio de 2012 Groovy 2.0 foi lançado, o qual, entre outras incrementações foram adicionados compilação estática e checagem de tipo estáticos.Em abril de 2015,Pivotal parou de patrocinar Groovy and Grails.Groovy então mudou sua estrutura de governância de um repositório do “Codehaus” para “Project Management Committee (PMC)” do Apache Software Foundation via o seu incubador.

2. Domínios de aplicação

Exemplificação de semelhanças entre as linguagens Java e Groovy

Código em Java

```
class Example{

        static void main(String[] args){
          static void main(String[] args){
            println(“Hello World”);
                                        }
                                      }
}
```

Código em Groovy

```
class Example{

        static void main(String[] args){
          static void main(String[] args){
            println(“Hello World”);
                                        }
                                      }
}
```

Fatorial em Groovy

```
def fatorial(x)
{
        if (x < 2) {
                return 1
        } else {
                return x * fatorial(x - 1)
        }
}


fatorial(20)
```

Programa que soma 10 números e o divide por 10,utilizando um vetor

```
class Average
{
  public static void main(String[] args){
     double aux=0;
     int i;
     def vetor = new double[10];
     for(i=0;i<10;i++){
       vetor[i]=(double)i;
                  }

     for(i=0;i<10;i++){
       aux = aux + vetor[i];
                  }
       aux = aux/10;



System.out.println(aux);


                              }
```

}

Apresentação de Listas

Adicionar um elemento a lista

```
def myList = ["A", "B", "C"]
myList << "D"
println myList
```

Saída: ["A", "B", "C", “D”]

```
def myList = ["A", "B", "C"]
myList.add("D")
myList.add("E")
println myList
```

Saída: ["A", "B", "C",”D”,”E”]

Remover um elemento da lista

```
def myList = ["A", "B", "C"]
myList.remove(1)
println myList
```

Saída: ["A","C"]

```
def myList = ["A", "B", "C"]
myList.remove("C")
println myList
```

Saída: ["A","B"]

3. Paradigmas suportados pela linguagem

Paradigma Orientado a Objeto:

```
class Estudante {
   int numEstudante;
   String nomeEstudante;

   static void main(String[] args) {
      Estudante aluno = new Estudante();
      aluno.StudentID = 1;
      aluno.StudentName = "Lucas"
   }
}
```

```
class Estudante {

   private int numEstudante;

   private String nomeEstudante;


   void setEstudantenum(int pnum) {

      Estudantenum = pnum;

   }


   void setEstudanteName(String pnome) {

      nomeEstudante = pnome;

   }


   int getnumEstudante() {

      return this.Estudantenum;

   }


   String getnomeEstudante() {

      return this.nomeEstudante;
```

```
    }

    static void main(String[] args) {
        Estudante aluno = new Estudante();
        aluno.setnumEstudante(1);
        aluno.setnomeEstudante("Lucas");

        println(aluno.getnumEstudante());
        println(aluno.getnomeEstudante());
    }
}
```

Paradigma Funcional:

```
def fac(n) { n == 0 ? 1 : n * fac(n - 1) }
assert 24 == fac(4)

def fac2 = { n -> n == 0 ? 1 : n * call(n - 1) }
assert 24 == fac2(4)
```

Programação Meta-Objeto:

```
Def myStudent = new Student()
myStudent.Name = ”Joe”;
myStudent.Display()
```

Apesar da classe não conter a variável Name ou o metodo Display(), o código acima ainda funciona,contudo para funcionar é preciso implementar a interface GroovyInterceptable para ligar o processo em execução ao Groovy.

Os principais diferenciais entre as linguagens consiste em:

ponto e vírgula opcional: se você for digitar apenas um comando em uma linha, não precisa digitar o ponto e vírgula. No caso de mais de um comando, no entanto, já é necessário.

Exemplo:

```
println "Sou um comando sem ponto e virgula. Não é legal?"

println "Eu já tenho ponto e virgula no final. Sou mais tradicional";

println "Mais de um"; println "comando"; println "na mesma"; println "linha.";
```

conceito de verdade: Groovy extende o conceito de verdadeiro com o qual estamos habituados a trabalhar em Java. Além do conceito tradicional, Groovy também considera verdadeiro qualquer valor que seja diferente de null, tal como nos exemplos abaixo:

```
String str = "Sou uma string não nula"
String strNula = null
if (str) {
  println "Eu com certeza serei impresso"

}

if (! strNula) {
   println "Eu vou ser impresso, pois meu teste consistiu em uma string nula negada!"

```

```
11      }
```

“main opcional”: se estiver trabalhando com Groovy na forma de um script, pode considerar o seu arquivo de código fonte como um main. Tudo o que estiver fora das chaves será considerado código a ser executado, tal como no exemplo abaixo:

```
1   println "Veja, sou um script feito em Groovy"
2
3   boolean maiorQue(int a, int b) {
4
5     return a < b
6
7   }
8
9   println "Reparou como eu simplesmente 'ignorei' a função maiorQue e continuei
    imprimindo?"
1
0
1   &#91;/code&#93;
1
1
    Gerará a saida
2
1
3   Veja, sou um script feito em Groovy
1   Reparou como eu simplesmente 'ignorei' a função maiorQue e continuei imprimindo?
4
1
5
```

Tudo é considerado objeto: ao contrário do Java em que possuímos tipos primitivos e não primitivos, em Groovy tudo é considerado objeto. Sendo assim, os tipos primitivos do Java são convertidos para suas respectivas classes encapsuladoras de forma transparente para o programador, tal como no exemplo abaixo:

```
int inteiro = 4;
print inteiro.toString()
```

Uma maneira fácil de se habituar a este detalhe da linguagem consiste em compará-la com o autoboxing incluído na linguagem Java a partir da versão 5.

return opcional: a instrução return com a qual já estamos habituados a trabalhar em Java também existe em Groovy. Porém, é opcional. Dada uma função, o valor de retorno do último comando corresponde ao valor de retorno da mesma, tal como no exemplo abaixo:

```
        boolean maiorQuetradicional(int a, int b) {

            // a maneira "tradicional" de se trabalhar com Java
            return a > b

        }

        boolean maiorQueGroovy(int a, int b) {
```

```
9
10                  a > b // bem mais simples!
11
12          }
```

4. Variáveis e tipos de dados

- **byte** – Usado para representar um valor de byte,EX: 2.
- **short** – Usado para representar um número pequeno,EX: 10.
- **int** – Usado para representar números inteiros,EX: 1234.
- **long** – Usado para representar inteiros longos,EX: 10000090.
- **float** – Usado para representar tipos de um ponto flutuante(racional) de 32-bits EX:12.34.
- **double** – Usado para representar tipos de um ponto flutuante maiores de 64-bits, EX: 12.3456565.
- **char** – Usado para definir um caracter literal,EX: 'a'.
- **Boolean** – Usado para representar valoroes falsos ou verdadeiros,EX: True.
- **String** – Usado para representar textos literais que são dispostos como uma. cadeia de caracteres,EX:"Linguagem Groovy".

Tabela mostra os valores máximos permitidos

byte	-128 to 127
short	-32,768 to 32,767

int	-2,147,483,648 to 2,147,483,647
long	-9,223,372,036,854,775,808 to +9,223,372,036,854,775,807
float	1.40129846432481707e-45 to 3.40282346638528860e+38
double	4.94065645841246544e-324d to 1.79769313486231570e+308d

Além dos tipos primitivos temos também os tipos de objetos que podem ser:

- java.lang.Byte
- java.lang.Short
- java.lang.Integer
- java.lang.Long
- java.lang.Float
- java.lang.Double

E por fim,variáveis também podem ser definidas com a palavra-chave “def”,que age como se fosse um coringa para declaração de variáveis.Groovy é caso sensitivo,variáveis podem ser compostas por letras,dígitos e underscore,e devem começar com uma letra ou underscore.

4.1 Operadores

Um operador é um símbolo que diz ao compilador para executar manipulações matemáticas ou lógicas.

Groovy contém os seguintes tipos de operadores :

- Operadores aritméticos
- Operadores relacionais
- Operadores lógicos
- Operadores Bitwise

- Operadores de atribuição

Operadores aritméticos:

Operator	Description	Example
+	Adição de dois operadores.	1 + 2 = 3
−	Subtrai o segundo operando do primeiro.	2 − 1 = 1
*	Multiplica ambos os operadores.	2 * 2 = 4
/	Divisão do numerador pelo denominador.	3 / 2 = 1.5
%	Resto da divisão de inteiros ou floats.	3 % 2 = 1
++	Incrementa o valor do operando por 1.	int x = 5; x++; x = 6

--	Diminui o valor do operando por 1.	int x = 5; x--; x = 4

Operadores relacionais:

Operator	Description	Example
==	Testa se dois valores são iguais	2 == 2 irá dar true
!=	Testa se dois valores são diferentes	3 != 2 Irá dar true
<	Checa se o operando da esquerda é menor que o operando da direita	2 < 3 Irá dar true
<=	Checa se o operando da esquerda é menor ou igual que o da direita	2 >= 3 Irá dar false
>	Checa se o operando da esquerda é maior que o operando da direita	3 < 2 Irá dar true

>=	Checa se o operando da esquerda é maior ou igual que o operando da direita	3 <= 2 irá dar false

Operadores lógicos:

Operator	Description	Example
&&	Este é o operador lógico “e”	true && true dará true
||	Este é o operador lógico “ou”	true || false dará true
!	Este é o operador lógica “não”	!true dará falso

Operadores bitwise:

Operator	Description
&	Este é o operador bitwse para “e”
|	Este é o operador bitwise para “ou”
^	Este é o operador bitwise para “xor”
~	Este é o operador bitwise de negação

Tabela verdade sobre os operadores

p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Operadores de atribuição:

Operator	Description	Example
+=	Adiciona o operando da direita para o da esquerda e atribui o valor para o operando a esquerda.	def A = 5 A+=3 Saída A=8

-= -=	Diminui o operando da direita para o da esquerda e atribui o valor para o operando a esquerda.	def A = 5 A-=3 Saída A=2
*=	Multiplica o operando da direita para o da esquerda e atribui o valor para o operando a esquerda.	def A = 5 A*=3 Saída A=15
/=	Divide o operando da direita para o da esquerda e atribui o valor para o operando a esquerda.	def A = 6 A/=3 Saída A= 2
%=	Tira o resto do operando da direita para o da esquerda e atribui o valor para o operando a esquerda.	def A = 5

		A%=3 Saída A=2

5. Comandos de controle

Comandos de controle requerem que o programador especifique uma ou mais condições para serem validadas ou testadas por um programa,junto com ações que serão executadas se a condição for verdadeira e opcionalmente,outras ações se a condição for falsa.

Num.	Comandos e descrição
1	**if** Uma condição é avaliada se a condição for verdadeira então se executa as expressões desejadas if(a>b) println(“Sim”);
2	**if/else** Uma condição é avaliada se a condição for verdadeira então se executa as expressões desejadas, e para antes da condição do else.Se a condição for

	falsa então executa as expressões que estão no bloco do else e sai da condição. if(a<b) println(“Não”); else println(“Sim”);
3	**Nested If** Pode ser preciso ter If dentro de If. if(media>7) if(adicional == 3) media = 10
4	**Switch** Semelhante ao if/else porém pode ser utilizado para apresentar soluções de maneiras mais elegantes

	`` def x = 1.23 def result = "" switch (x) { case "foo": result = "found foo" // lets fall through case "bar": result += "bar" case [4, 5, 6, 'inList']: result = "list" break case 12..30: result = "range" break case Integer: result = "integer" break case Number: result = "number" break case { it > 3 }: result = "number > 3" break default: result = "default" } assert result == "number" ```
5	**Nested Switch** Também é possível ter Switch dentro de Switch.

6. Escopo (regras de visibilidade)

Apresentar um exemplo para cada escopo suportado pela linguagem e programação estudada.

7. Exemplo prático de uso da linguagem de programação

Para se ter um controle do número de vagas disponíveis em um estacionamento é preciso de um sistema que seja capaz de mostrar quantos carros já estão estacionados e também quantos carros o estacionamento pode comportar.Deste modo o cliente não perderá muito tempo caso o estacionamento esteja cheio e sem vagas.

Pesquise um problema e proponha um programa (o código fonte deve ser apresentado) escrito na linguagem de programação escolhida para resolver o problema. Deve ser descrito qual paradigma foi escolhido e justificado sua escolha. Adicionalmente, deve ser descrito o seu ambiente de desenvolvimento. Exemplo: Um sistema de cadastro de alunos e notas, onde o sistema calcule a nota final e o percentual de faltas, bem como apresente um gráfico de cada nota do aluno.

8. Conclusões

Fica evidente as vantagens apresentadas para o aperfeiçoamento de Java,Groovy disponibiliza ao programador mais ferramentas e muitas vezes de maneira mais simplificada,por ser tratar de uma linguagem de múltiplo paradigma é possível encarar problemas de diversas linhas de raciocínio, e assim possivelmente,apresentar uma maior adequação a questão e ser por vezes mais eficiente, por ter uma integração forte com Java, o conhecimento que já fora adquirido por um programador é utilizado,facilitando a curva de aprendizado da linguagem.Uma linguagem que apesar dos anos de dormência pode ser uma das apostas para a evolução da programação com o modelo dinâmico que está cada vez mais sendo utilizado.

9. Referencias

Retirado do sítio “http://www.itexto.net/devkico/?p=231”

Groovy in Action, Second Edition Dierk König, Paul King, Guillaume Laforge, Hamlet D'Arcy, Cédric Champeau, Erik Pragt, and Jon Skeet *Foreword by James Gosling*

Retirado do sítio “http://www.groovy-lang.org/”