Apresentaçao da linguagem Groovy

Eduardo Motta Ramos
Caio Saracuza

23/03/2018

Universidade Federal Fluminense

Linguagem escolhida

História

Início do desenvolvimento em 2003, mas a versão 1.0 foi lançada oficialmente em 2007.
Similar a Java, porém mais compacta por não precisar de todos os elementos que Java precisa.

Groovy traz suporte nativo para várias linguagens de marcação, como XML e HTML.

Compilação de Groovy em algum ponto é representada em uma AST(Abstract Syntax Tree), com propósito de deixar os desenvolvedores modificarem a AST antes de se transformar no bytecode que a JVM utilizará.

Linguagem escolhida

Groovy:

- Paradigmas: Orientada a objeto, imperativa, funcional, script
- Interpretada em tempo de execução
- Também é possivel compilar o código para conseguir melhorar ainda mais a performance. (comparável com Java)
- Possui inferência de tipos (def)

Exemplos de uso da linguagem

Groovy traz uma forma simples e consistente de lidar com Listas e HashMap.

```
// Looks like an array, but is a list
def movieList = ['Dersu Uzala', 'Ran', 'Seven Samurai']
assert movieList[2] == 'Seven Samurai'
movieList[3] = 'Casablanca' // Adds an element to the list
assert movieList.size() == 4

// Declares a map
def monthMap = [ 'January' : 31, 'February' : 28, 'March' : 31 ]
assert monthMap['March'] == 31 // Accesses an entry
monthMap['April'] = 30 // Adds an entry to the map
assert monthMap.size() == 4
```

Exemplos de uso da linguagem

Definindo Classes:

```
class AGroovyBean {
        String color
}

def myGroovyBean = new AGroovyBean()

myGroovyBean.setColor('baby blue')
assert myGroovyBean.getColor() == 'baby blue'

myGroovyBean.color = 'pewter'
assert myGroovyBean.color == 'pewter'
```

Exemplos de uso da linguagem

Usando GStrings:

```
BigDecimal account = 10.0
def text = "The account shows currently a balance of $account"
assert text == "The account shows currently a balance of 10.0"

BigDecimal minus = 4.0
text = "The account shows currently a balance of ${account - minus}"
assert text == "The account shows currently a balance of 6.0"
```

Closures

```
// This block of code contains expressions without reference to an
implementation
def operations = {
        declare 5
        sum 4
        divide 3
        print
}
```

Suporte à construção de compiladores

```
def closure = { ' HELLO WORLD! '}
assert closure() == 'HELLO WORLD!'

def sum = { a, b -> a + b }
assert sum(2,3) == 5
```

Recursive

factorial(6)
6 * factorial(5)
6 * (5 * factorial(4))
6 * (5 * (4 * factorial(3)))
6 * (5 * (4 * (3 * factorial)))
6 * (5 * (4 * (3 * (2 * factorial(1)))))
6 * (5 * (4 * (3 * (2 * 1))))
6 * (5 * (4 * (3 * 2)))
6 * (5 * (4 * 6))
6 * (5 * 24)
6 * 120
720

Prós e contras

Disdvantages

During the first years of its life Groovy's performances were poor. Very poor. This was mainly due to a feature of the JVM that was blocking dynamic languages. From Java7 a new feature called invokeDynamic greatly improves dynamic code execution; the compiled groovy classes usually share the same execution time of Java (kind of). However, Groovy is still seen as the "slow" JVM language. Let's change opinion! People coming from iterative Java will not understand functional paradigms and will say that Groovy is too difficoult or crazy. Many are asking, now that Java8 is here and supports lambda expressions (aka functional paradigm), do we still need Groovy? Well, if you also need a dynamic scripting language the answer is yes.