Groovy Programming language specifications

Manish Tanwar

February 22, 2017

Abstract

We describe a core language for the Object Oriented language and also a functional programming language named "Groovy", presenting its Features and informal static and advantages of this language. We also discuss about language specification at abstract level and other open issues, and sketch a syntax for XML and JSON also we discuss about abstract syntax tree transformation.

1 Introduction

Groovy Language is an intense, alternatively written and dynamic dialect, with static-writing and static gathering capacities, for the Java stage went for enhancing designer profitability on account of a compact, commonplace and simple to learn grammar. It coordinates easily with any Java program, and quickly conveys to your application intense elements, including scripting capacities, Domain-Specific Language creating, runtime and arrange time meta-programming and useful programming.

2 History

James Strachan first talked about the development of Groovy on his blog in August 2003. Several versions were released between 2004 and 2006. After the Java Community Process (JCP) standardization effort began, the version numbering changed and a version called "1.0" was released on January 2, 2007. After various betas and release candidates numbered 1.1, on December 7, 2007, Groovy 1.1 Final was released and immediately renumbered as Groovy 1.5 to reflect the many changes made.

In 2007, Groovy won the first prize at JAX 2007 innovation award. In 2008, Grails, a Groovy web framework, won the second prize at JAX 2008 innovation award. In November 2008, Spring Source acquired the Groovy and Grails company (G2One). In August 2009 VMWare acquired Spring Source. Strachan had left the project silently a year before the Groovy 1.0 release in 2007.

In March 2004, Groovy had been submitted to the JCP as JSR 241 and accepted by ballot. After 8 years of inactivity, the Spec Lead changed its status to dormant in April 2012. On July 2, 2012, Groovy 2.0 was released, which, among other new features, added static compiling and static type checking.

3 Features

Most substantial Java records are likewise legitimate Groovy documents. In spite of the fact that the two dialects are comparative, Groovy code can be more reduced, in light of the fact that it needn't bother with every one of the components that Java needs. This makes it feasible for Java software engineers to learn Groovy bit by bit by beginning with well known Java sentence structure before securing more Groovy programming sayings.

Since In version 2 Groovy likewise underpins particularity (having the capacity to dispatch just the required containers as per the venture needs, accordingly lessening the span of Groovy's library), sort checking, static assembling, Project Coin language structure upgrades, multicatch squares and progressing execution improvements utilizing JDK7's conjure dynamic direction.

4 Functional programming

Although Groovy is mostly an object-oriented language, it also offers functional programming features.

4.1 Closures

As indicated by Groovy’s documentation: "closures in Groovy work like a ’technique pointer’, empowering code to be composed and keep running in a later point in time". Awesome’s terminations bolster free factors, i.e. factors which have not been expressly passed as a parameter to it, yet exist in its assertion setting, incomplete application (which it terms ’currying’), appointment, certain, wrote and untyped parameters.

4.2 Curry

Typically called fractional application, this Groovy component permits closures’ parameters to be set to a default parameter in any of their contentions, making another conclusion with the bound esteem. Providing one contention to the curry() strategy will settle contention one.

```
def joinTwoWordsWithSymbol =  symbol, first, second -> first + symbol + second
assert joinTwoWordsWithSymbol(#  ’Hello’, ’World’) == ’Hello’#Ẃorld’

    def concatWords = joinTwoWordsWithSymbol.curry(’ ’)
assert concatWords(’Hello’, ’World’) == ’Hello World’

    def prependHello = concatWords.curry(’Hello’)
// def prependHello = joinTwoWordsWithSymbol.curry(’ ’, ’Hello’)
assert prependHello(’World’) == ’Hello World’
```

Providing N contentions will settle contentions 1..N. def power = BigDecimal value, BigDecimal power -> value ** power

```
    def square = power.rcurry(2)
def cube = power.rcurry(3)

    assert power(2, 2) == 4
assert square(4) == 16
assert cube(3) == 27
```

5 XML and JSON processing

On XML and JavaScript Object Notation (JSON) processing Groovy employs the Builder pattern, making the production of the data structure less verbose. For example, the following XML:

```
    <languages>
<language year="1995">
<name>java</name>
<paradigm>Object oriented</paradigm>
<typing>Static</typing>
</language>
<language year="1995">
<name>ruby</name>
<paradigm>Object oriented, Functional</paradigm>
<typing>Dynamic, duck typing</typing>
</language>
<language year="2003">
<name>groovy</name>
```

```
<paradigm>Object oriented, Functional</paradigm>
<typing>Dynamic, Static, Duck typing</typing>
</language>
</languages>
```

Can be generated via the following Groovy code:

```
def writer = new StringWriter()
def builder = new groovy.xml.MarkupBuilder(writer)
builder.languages
language(year: 1995)
name "java"
paradigm "Object oriented"
typing "Static"

language (year: 1995)
name "ruby"
paradigm "Object oriented, Functional"
typing "Dynamic, Duck typing"

language (year: 2003)
name "groovy"
paradigm "Object oriented, Functional"
typing "Dynamic, Static, Duck typing"
```

6 Abstract syntax tree transformation

As indicated by Groovy's own documentation, "When the Groovy compiler incorporates Groovy scripts and classes, sooner or later all the while, the source code will wind up being spoken to in memory as a Concrete Syntax Tree, then changed into an Abstract Syntax Tree. The motivation behind AST Transformations is to give designers a chance to guide into the gathering procedure to have the capacity to change the AST before it is transformed into bytecode that will be controlled by the JVM. AST Transformations gives Groovy enhanced accumulate time metaprogramming abilities permitting effective adaptability at the dialect level, without a runtime execution punishment. Examples of ASTs in Groovy are:

6.1 Singleton transformation

6.2 Category and Mixin transformation

6.3 Immutable AST Macro

6.4 Newify transformation

7 Traits

According to Groovy's documentation, "Traits are a structural construct of the language which allow: composition of behaviors, runtime implementation of interfaces, behavior overriding, and compatibility with static type checking/compilation."

8 Summary

At last we got 40 percent less code, a strong and stable design and we totally expelled the Visitor from the Visitable. I caught wind of guest executions in light of Reflection to get a more non specific form. All things considered, with this you see there is truly no compelling reason to do such thing. On the off chance that we include new sorts we don't have to change anything. It is said that the guest design doesn't fit outrageous programming methods extremely well since you have to roll out improvements to such a large number of classes constantly. I think I demonstrated this is a direct result of Java not on account of the example is awful or something.

There are variations of the Visitor design, similar to the non-cyclic guest design, that tries to take care of the issue of including new hub sorts with exceptional guests. I don't care for that in particular, it works with throws, gets the ClassCastException and other dreadful things. At last it tries to tackle something we don't get with the Groovy rendition.

One all the more thing. NodeType1Counter could be actualized in Java also. Awesome will perceive the visit techniques and call them as required in light of the fact that DefaultVisitor is still Groovy and does all the enchantment