Apache Groovy

Apache Groovy is a Java-syntax-compatible object-oriented programming language for the Java platform. It is both a static and dynamic language with features similar to those of Python, Ruby, and Smalltalk. It can be used as both a programming language and a scripting language for the Java Platform, is compiled to Java virtual machine (JVM) bytecode, and interoperates seamlessly with other Java code and libraries. Groovy uses a curly-bracket syntax similar to Java's. Groovy supports closures, multiline strings, and expressions embedded in strings. Much of Groovy's power lies in its AST transformations, triggered through annotations.

Groovy 1.0 was released on January 2, 2007, and Groovy 2.0 in July, 2012. Since version 2, Groovy can be compiled statically, offering type inference and performance near that of Java.[3][4] Groovy 2.4 was the last major release under Pivotal Software's sponsorship which ended in March 2015.[5] Groovy has since changed its governance structure to a Project Management Committee in the Apache Software Foundation.[6]

Contents

Groovy

Groovy Logo

Paradigm	Object-oriented, imperative, scripting
Designed by	James Strachan
Developer	Guillaume Laforge (PMC Chair) Jochen Theodorou (Tech Lead) Paul King Cedric Champeau
First appeared	2003
Stable release	3.0.9 (September 2, 2021[1]) [±] (https://en.wikipedia.org/w/index.php?title=Template:Latest_stable_software_release/Groovy&action=edit)
Preview release	4.0.0-beta-1 / September 6, 2021[1]
Typing discipline	Dynamic, static, strong, duck
Platform	Java SE
License	Apache License 2.0

Filename extensions	`.groovy`, `.gvy`, `.gy`, `.gsh`[2]
Website	groovy-lang.org (https://groovy-lang.org)
Major implementations	
Gradle, Grails	
Influenced by	
Java, Python, Ruby, Smalltalk	
Influenced	
Kotlin	

History

James Strachan first talked about the development of Groovy on his blog in August 2003.[7] In March 2004, Groovy was submitted to the JCP as JSR 241[8] and accepted by ballot. 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.[9] In 2008, Grails, a Groovy web framework, won the second prize at JAX 2008 innovation award.[10]

In November 2008, SpringSource acquired the Groovy and Grails company (G2One).[11] In August 2009 VMware acquired SpringSource.[12]

In April 2012, after eight years of inactivity, the Spec Lead changed the status of JSR 241 to dormant.[8]

Strachan had left the project silently a year before the Groovy 1.0 release in 2007. In Oct 2016, Strachan stated "I still love groovy (jenkins pipelines are so groovy!), java, go, typescript and kotlin".[13]

On July 2, 2012, Groovy 2.0 was released, which, among other new features, added static compiling and static type checking.

When the Pivotal Software joint venture was spun-off by EMC Corporation (EMC) and VMware in April 2013, Groovy and Grails formed part of its product portfolio. Pivotal ceased sponsoring Groovy and Grails from April 2015.[5] That same month, Groovy changed its governance structure from a Codehaus repository to a Project Management Committee (PMC) in the Apache Software Foundation via its incubator.[6] Groovy graduated from Apache's incubator and became a top-level project in November 2015.[14]

Features

Most valid Java files are also valid Groovy files. Although the two languages are similar, Groovy code can be more compact, because it does not need all the elements that Java needs.[15] This makes it possible for Java programmers to learn Groovy gradually by starting with familiar Java syntax before acquiring more Groovy programming idioms.[16]

Groovy features not available in Java include both static and dynamic typing (with the keyword `def`), operator overloading, native syntax for lists and associative arrays (maps), native support for regular expressions, polymorphic iteration, string interpolation, added helper methods, and the safe navigation operator `?.` to check automatically for null pointers (for example, `variable?.method()`, or `variable?.field`).[17]

Since version 2 Groovy also supports modularity (being able to ship only the needed jars according to the project needs, thus reducing the size of Groovy's library), type checking, static compiling, Project Coin syntax enhancements, multicatch blocks and ongoing performance enhancements using the

`invokedynamic` instruction introduced in Java 7.[18]

Groovy provides native support for various markup languages such as XML and HTML, accomplished via an inline Document Object Model (DOM) syntax. This feature enables the definition and manipulation of many types of heterogeneous data assets with a uniform and concise syntax and programming methodology.

Unlike Java, a Groovy source code file can be executed as an (uncompiled) script, if it contains code outside any class definition, if it is a class with a *main* method, or if it is a *Runnable* or *GroovyTestCase*. A Groovy script is fully parsed, compiled, and generated before executing (similar to Python and Ruby). This occurs under the hood, and the compiled version is not saved as an artifact of the process.[19]

GroovyBeans, properties

GroovyBeans are Groovy's version of JavaBeans. Groovy implicitly generates getters and setters. In the following code, `setColor(String color)` and `getColor()` are implicitly generated. The last two lines, which appear to access color directly, are actually calling the implicitly generated methods.[20]

```
class AGroovyBean {
  String color
}

def myGroovyBean = new AGroovyBean()

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

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

Groovy offers simple, consistent syntax for handling *lists* and *maps*, reminiscent of Java's *array* syntax.[21]

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

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

Prototype extension

Groovy offers support for prototype extension through `ExpandoMetaClass`, Extension Modules (only in Groovy 2), Objective-C-like Categories and `DelegatingMetaClass`.[22]

`ExpandoMetaClass` offers a domain-specific language (DSL) to express the changes in the class easily, similar to Ruby's open class concept:

```
Number.metaClass {
  sqrt = { Math.sqrt(delegate) }
}
```

```
assert 9.sqrt() == 3
assert 4.sqrt() == 2
```

Groovy's changes in code through prototyping are not visible in Java, since each attribute/method invocation in Groovy goes through the metaclass registry. The changed code can only be accessed from Java by going to the metaclass registry.

Groovy also allows overriding methods as `getProperty()`, `propertyMissing()` among others, enabling the developer to intercept calls to an object and specify an action for them, in a simplified aspect-oriented way. The following code enables the class `java.lang.String` to respond to the `hex` property:

```
enum Color {
  BLACK('#000000'), WHITE('#FFFFFF'), RED('#FF0000'), BLUE('#0000FF')
  String hex
  Color(String hex) {
    this.hex = hex
  }
}

String.metaClass.getProperty = { String property ->
  def stringColor = delegate
  if (property == 'hex') {
    Color.values().find { it.name().equalsIgnoreCase stringColor }?.hex
  }
}

assert "WHITE".hex == "#FFFFFF"
assert "BLUE".hex == "#0000FF"
assert "BLACK".hex == "#000000"
assert "GREEN".hex == null
```

The Grails framework uses metaprogramming extensively to enable GORM dynamic finders, like `User.findByName('Josh')` and others.[23]

Dot and parentheses

Groovy's syntax permits omitting parentheses and dots in some situations. The following groovy code

```
take(coffee).with(sugar, milk).and(liquor)
```

can be written as

```
take coffee with sugar, milk and liquor
```

enabling the development of domain-specific languages (DSLs) that look like plain English.

Functional programming

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

Closures

According to Groovy's documentation: "Closures in Groovy work similar to a 'method pointer', enabling code to be written and run in a later point in time".[24] Groovy's closures support free variables, i.e. variables that have not been explicitly passed as a parameter to it, but exist in its declaration context, partial application (that it terms 'currying'[25]), delegation, implicit, typed and untyped parameters.

When working on Collections of a determined type, the closure passed to an operation on the collection can be inferred:

```
list = [1, 2, 3, 4, 5, 6, 7, 8, 9]

/*
 * Non-zero numbers are coerced to true, so when it % 2 == 0 (even), it is false.
 * The type of the implicit "it" parameter can be inferred as an Integer by the IDE.
 * It could also be written as:
 * list.findAll { Integer i -> i % 2 }
 * list.findAll { i -> i % 2 }
 */
def odds = list.findAll { it % 2 }

assert odds == [1, 3, 5, 7, 9]
```

A group of expressions can be written in a closure block without reference to an implementation and the responding object can be assigned at a later point using delegation:

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

```
/*
 * This class will handle the operations that can be used in the closure above. Another class
 * could be declared having the same methods, but using, for example, webservice operations
 * in the calculations.
 */
class Expression {
  BigDecimal value

  /*
   * Though an Integer is passed as a parameter, it is coerced into a BigDecimal, as was
   * defined. If the class had a 'declare(Integer value)' method, it would be used instead.
   */
  def declare(BigDecimal value) {
    this.value = value
  }

  def sum(BigDecimal valueToAdd) {
    this.value += valueToAdd
  }

  def divide(BigDecimal divisor) {
    this.value /= divisor
  }

  def propertyMissing(String property) {
    if (property == "print") println value
  }
}
```

```
// Here is defined who is going to respond the expressions in the block of code above.
operations.delegate = new Expression()
operations()
```

Curry

Usually called *partial application*,[25] this Groovy feature allows closures' parameters to be set to a default parameter in any of their arguments, creating a new closure with the bound value. Supplying one argument to the `curry()` method will fix argument one. Supplying N arguments will fix arguments 1 .. N.

```
def joinTwoWordsWithSymbol = { symbol, first, second -> first + symbol + second }
assert joinTwoWordsWithSymbol('#', 'Hello', 'World') == 'Hello#World'

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'
```

Curry can also be used in the reverse direction (fixing the last N arguments) using `rcurry()`.

```
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
```

Groovy also supports lazy evaluation,[26][27] reduce/fold,[28] infinite structures and immutability,[29] among others.[30]

JSON and XML processing

On JavaScript Object Notation (JSON) and XML 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>functional, object oriented</paradigm>
    <typing>duck typing, dynamic</typing>
  </language>
  <language year="2003">
    <name>Groovy</name>
    <paradigm>functional, object oriented</paradigm>
    <typing>duck typing, dynamic, static</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 "functional, object oriented"
    typing "duck typing, dynamic"
  }
  language (year: 2003) {
    name "Groovy"
    paradigm "functional, object oriented"
    typing "duck typing, dynamic, static"
  }
}
```

and also can be processed in a streaming way through `StreamingMarkupBuilder`. To change the implementation to JSON, the `MarkupBuilder` can be swapped to `JsonBuilder`.[31]

To parse it and search for a functional language, Groovy's `findAll` method can serve:

```
def languages = new XmlSlurper().parseText writer.toString()

// Here is employed Groovy's regex syntax for a matcher (=~) that will be coerced to a
// boolean value: either true, if the value contains our string, or false otherwise.
def functional = languages.language.findAll { it.paradigm =~ "functional" }
assert functional.collect { it.name } == ["Groovy", "Ruby"]
```

String interpolation

In Groovy, strings can be interpolated with variables and expressions by using GStrings:[32]

```
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"
```

GStrings containing variables and expressions must be declared using double quotes.

A complex expression must be enclosed in curly brackets. This prevents parts of it from being interpreted as belonging to the surrounding string instead of to the expression:

```
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"

// Without the brackets to isolate the expression, this would result:
text = "The account shows currently a balance of $account - minus"
assert text == "The account shows currently a balance of 10.0 - minus"
```

Expression evaluation can be deferred by employing arrow syntax:

```
BigDecimal tax = 0.15
text = "The account shows currently a balance of ${->account - account*tax}"
tax = 0.10

// The tax value was changed AFTER declaration of the GString. The expression
// variables are bound only when the expression must actually be evaluated:
assert text == "The account shows currently a balance of 9.000"
```

Abstract syntax tree transformation

According to Groovy's own documentation, "When the Groovy compiler compiles Groovy scripts and classes, at some point in the process, the source code will end up being represented in memory in the form of a Concrete Syntax Tree, then transformed into an Abstract Syntax Tree. The purpose of AST Transformations is to let developers hook into the compilation process to be able to modify the AST before it is turned into bytecode that will be run by the JVM. AST Transformations provides Groovy with improved compile-time metaprogramming capabilities allowing powerful flexibility at the language level, without a runtime performance penalty."[33]

Examples of ASTs in Groovy are:

- Category and Mixin transformation
- Immutable AST Macro
- Newify transformation
- Singleton transformation

among others.

Traits

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

Traits can be seen as interfaces carrying both default implementations and state. A trait is defined using the trait keyword:

```
trait FlyingAbility { /* declaration of a trait */
  String fly() { "I'm flying!" } /* declaration of a method inside a trait */
}
```

Then, it can be used like a normal interface using the keyword `implements`:

```
class Bird implements FlyingAbility {} /* Adds the trait FlyingAbility to the Bird class
capabilities */
def bird = new Bird() /* instantiate a new Bird */
assert bird.fly() == "I'm flying!" /* the Bird class automatically gets the behavior of the
FlyingAbility trait */
```

Traits allow a wide range of abilities, from simple composition to testing.

Adoption

Notable examples of Groovy adoption include:

- Adaptavist ScriptRunner, embeds a Groovy implementation to automate and extend Atlassian tools, in use by more than 20000 organisations around the world.[34][35]
- Apache OFBiz, the open-source enterprise resource planning (ERP) system, uses Groovy.[36][37]
- Eucalyptus, a cloud management system, uses a significant amount of Groovy.

- Gradle is a popular build automation tool using Groovy.
- LinkedIn uses Groovy and Grails for some of their subsystems.[38]
- LogicMonitor, a cloud-based monitoring platform, uses Groovy in script-based data sources.[39]
- Jenkins, a platform for continuous integration. With version 2, Jenkins includes a *Pipeline* plugin that allows for build instructions to be written in Groovy.[40]
- Liferay, uses groovy in their kaleo workflow
- Sky.com uses Groovy and Grails to serve massive online media content.[41]
- SmartThings, an open platform for smart homes and the consumer Internet of Things, uses a security-oriented subset of Groovy[42]
- SoapUI provides Groovy as a language for webservice tests development.[43]
- Survata, a market research startup, uses Groovy and Grails.
- The European Patent Office (EPO) developed a dataflow programming language in Groovy "to leverage similarities in the processes for communicating with each individual country's patent office, and transform them into a single, universal process."
- Though Groovy can be integrated into any JVM environment, the JBoss Seam framework provides Groovy, besides Java, as a development language, out of the box.[44]
- vCalc.com uses Groovy for all of the user defined mathematics in its math crowd-sourcing engine.[45]
- Wired.com uses Groovy and Grails for the Product Reviews standalone section of the website.[46]
- XWiki SAS uses Groovy as scripting language in their collaborative open-source product.[47]

IDE support

Many integrated development environments (IDEs) and text editors support Groovy:

- Android Studio, IDE used for making Android apps
- Atom editor
- Eclipse, through Groovy-Eclipse
- Emacs, using the groovy-emacs-mode project's groovy-mode.
- IntelliJ IDEA, Community Edition, Grails/Griffon in the Ultimate Edition only
- JDeveloper, for use with Oracle ADF
- jEdit, an advanced text editor for the Java platform
- Kate, an advanced text editor for KDE supports Groovy and over 200 other file formats
- NetBeans, since version 6.5
- Notepad++, an advanced text editor for Microsoft Windows
- Sublime Text, a cross platform text editor
- TextMate
- Visual Studio Code
- UltraEdit, general purpose program editor

Dialects

There is one alternative implementation of Groovy:

- Grooscript converts Groovy code to JavaScript code.[48] Although Grooscript has some limitations compared to Apache Groovy, it can use domain classes in both the server and the client.[49] Plugin support for Grails version 3.0 is provided, as well as online code conversions.[50]

See also

- Comparison of programming languages
- Griffon (framework) – a desktop framework
- Project Zero
- Spock (testing framework)

References

Citations

1. "Releases - apache/groovy" (https://github.com/apache/groovy/releases). Retrieved 2021-10-27 – via GitHub.
2. https://mrhaki.blogspot.com/2011/10/groovy-goodness-default-groovy-script.html
3. "Groovy 2.0 Performance compared to Java" (http://objectscape.blogspot.com.br/2012/08/groovy-20-performance-compared-to-java.html). 25 Aug 2012.
4. "Java vs Groovy2.0 vs Scala Simple Performance Test" (https://web.archive.org/web/20121210210126/http://blog.evan-wong.com/2012/07/java-vs-groovy20-vs-scala-simple.html). 10 Jul 2012. Archived from the original (http://blog.evan-wong.com/2012/07/java-vs-groovy20-vs-scala-simple.html) on 10 December 2012. Retrieved 7 October 2012.
5. "Groovy 2.4 And Grails 3.0 To Be Last Major Releases Under Pivotal Sponsorship" (http://blog.pivotal.io/pivotal/news-2/groovy-2-4-and-grails-3-0-to-be-last-major-releases-under-pivotal-sponsorship). 19 Jan 2015.
6. "Groovy joins Apache Incubator" (https://blogs.apache.org/foundation/entry/groovy_submitted_to_become_a). 11 Mar 2015.
7. James Strachan (29 Aug 2003). "Groovy - the birth of a new dynamic language for the Java platform" (https://web.archive.org/web/20030901064404/http://radio.weblogs.com/0112098/2003/08/29.html). Archived from the original (http://radio.weblogs.com/0112098/2003/08/29.html) on 1 September 2003.
8. "Java Community Process JSR 241" (http://www.jcp.org/en/jsr/detail?id=241).
9. "Groovy wins first prize at JAX 2007 innovation award" (https://web.archive.org/web/20150513184206/http://docs.codehaus.org/display/GROOVY/2007/04/26/Groovy+wins+first+prize+at+JAX+2007+innovation+award). 2007-04-26. Archived from the original (http://docs.codehaus.org/display/GROOVY/2007/04/26/Groovy+wins+first+prize+at+JAX+2007+innovation+award) on 2015-05-13. Retrieved 2012-10-07.
10. "They say a lot can happen over a cup of coffee" (https://web.archive.org/web/20110419130810/http://jax-award.de/jax_award08/proposal_view_de.php?id=240). Archived from the original (http://jax-award.de/jax_award08/proposal_view_de.php?id=240&show=more) on 2011-04-19. Retrieved 2012-10-07.
11. "SpringSource Acquires Groovy and Grails company (G2One)" (http://www.indicthreads.com/2138/springsource-acquires-groovy-and-grails-company-g2one/). 11 Nov 2008.
12. "VMWare Acquires SpringSource" (https://techcrunch.com/2009/08/10/vmware-acquires-springsource/). 10 Aug 2009.

13. "Tweet from James Strachan" (https://twitter.com/jstrachan/status/784333918078169088). November 24, 2016. Retrieved 2016-11-24.
14. "Announcement on dev mailing list" (http://mail-archives.apache.org/mod_mbox/groovy-dev/201511.mbox/%3CCAEWfVJ%3DBz-tiTTYHiPc8vY26CopLm3pPy_LADvvFh4vjVs%3Dosw%40mail.gmail.com%3E).
15. König 2007, pg. 32
16. "Groovy style and language feature guidelines for Java developers" (https://web.archive.org/web/20150117214709/http://groovy.codehaus.org/Groovy+style+and+language+feature+guidelines+for+Java+developers). Groovy.codehaus.org. Archived from the original (http://groovy.codehaus.org/Groovy+style+and+language+feature+guidelines+for+Java+developers) on 2015-01-17. Retrieved 2015-01-22.
17. "Groovy – Differences from Java" (https://web.archive.org/web/20090317025737/http://groovy.codehaus.org/Differences+from+Java). Groovy.codehaus.org. Archived from the original (http://groovy.codehaus.org/Differences+from+Java) on 2009-03-17. Retrieved 2013-08-12.
18. "What's new in Groovy 2.0?" (http://www.infoq.com/articles/new-groovy-20). 28 Jun 2012.
19. König 2007, pp. 37-8
20. König 2007, pp. 38-9
21. König 2007, pp. 41-3
22. "JN3525-MetaClasses" (https://web.archive.org/web/20121001122409/http://groovy.codehaus.org/JN3525-MetaClasses). Archived from the original (http://groovy.codehaus.org/JN3525-MetaClasses) on 2012-10-01. Retrieved 2012-10-07.
23. "Metaprogramming Techniques in Groovy and Grails" (http://www.slideshare.net/zenMonkey/metaprogramming-techniques-in-groovy-and-grails). 11 Jun 2009.
24. "Groovy - Closures" (https://web.archive.org/web/20120522213410/http://groovy.codehaus.org/Closures). Archived from the original (http://groovy.codehaus.org/Closures) on 2012-05-22.
25. "Does groovy call partial application 'currying' (http://programmers.stackexchange.com/questions/152868/does-groovy-call-partial-application-currying)", 10 Aug 2013
26. "Groovy - Lazy Transformation" (https://web.archive.org/web/20121008091312/http://groovy.codehaus.org/Lazy+transformation). Archived from the original (http://groovy.codehaus.org/Lazy+transformation) on 2012-10-08. Retrieved 2012-10-07.
27. "Side Notes: Lazy lists in Groovy" (http://ndpar.blogspot.com.br/2011/02/lazy-lists-in-groovy.html). 3 Feb 2011.
28. "Groovy's Fold" (https://web.archive.org/web/20150213033355/http://bendoerr.me/posts/2011-06-20-groovy-inject.html). 20 Jun 2011. Archived from the original (http://bendoerr.me/posts/2011-06-20-groovy-inject.html) on 13 February 2015. Retrieved 12 February 2015.
29. "Functional Programming with Groovy" (http://www.slideshare.net/arturoherrero/functional-programming-with-groovy). 5 Nov 2011.
30. "Function programming in Groovy" (https://web.archive.org/web/20121008095622/http://groovy.codehaus.org/Functional+Programming+with+Groovy). Archived from the original (http://groovy.codehaus.org/Functional+Programming+with+Groovy) on 2012-10-08. Retrieved 2012-10-07.
31. "JsonBuilder" (https://web.archive.org/web/20121002221510/http://groovy.codehaus.org/gapi/groovy/json/JsonBuilder.html). Archived from the original (http://groovy.codehaus.org/gapi/groovy/json/JsonBuilder.html) on 2012-10-02. Retrieved 2012-10-07.
32. "Groovy Strings - Different ways of creating them" (http://rajakannappan.blogspot.com.br/2009/12/groovy-strings-different-ways-of.html). 26 Dec 2009.
33. "Compile-time Metaprogramming - AST Transformations" (https://web.archive.org/web/20121014094900/http://groovy.codehaus.org/Compile-time+Metaprogramming+-+AST+Transformations). Archived from the original (http://groovy.codehaus.org/Compile-time+Metaprogramming+-+AST+Transformations) on 2012-10-14. Retrieved 2012-10-07.

34. "ScriptRunner Documentation" (https://scriptrunner.adaptavist.com/latest/index.html).
35. "ScriptRunner Press Release with adoption stats" (http://www.adaptavist.com/blog/adaptavist-completes-another-acquisition-against-a-backdrop-of-explosive-global-growth/).
36. "Groovy DSL for OFBiz business logic" (https://cwiki.apache.org/confluence/display/OFBIZ/Groovy+DSL+for+OFBiz+business+logic). *Apache OFBiz Project Open Wiki*.
37. "Simple-methods examples using Groovy" (https://cwiki.apache.org/confluence/display/OFBIZ/Simple-methods+examples+using+Groovy). *Apache OFBiz Project Open Wiki*.
38. "Grails at LinkedIn" (http://blog.linkedin.com/2008/06/11/grails-at-linkedin/). Retrieved 2015-06-02.
39. "Embedded Groovy Scripting" (https://www.logicmonitor.com/support/terminology-syntax/scripting-support/embedded-groovy-scripting). *www.logicmonitor.com*. Retrieved 2020-11-20.
40. "Jenkins Pipeline" (https://jenkins.io/doc/book/pipeline/overview/).
41. Rocher, Graeme (October 2, 2008). "Graeme Rocher's Blog: Sky.com relaunches written in Grails" (http://graemerocher.blogspot.com.br/2008/10/skycom-relaunches-written-in-grails.html). *Graeme Rocher's Blog*. Retrieved 2015-06-02.
42. Security Analysis of Emerging Smart Home Applications (https://iotsecurity.eecs.umich.edu/img/Paper27_CameraReady_SmartThings_Revised_IEEEGen.pdf)
43. "Scripting and the Script Library | Scripting & Properties" (http://www.soapui.org/Scripting-Properties/scripting-and-the-script-library.html). *www.soapui.org*. Retrieved 2015-06-02.
44. "Chapter 11. Groovy integration" (http://docs.jboss.org/seam/2.0.2.GA/reference/en-US/html/groovy.html). *docs.jboss.org*. Retrieved 2015-06-02.
45. "vCalc, the First ever Social Platform for the world of Math" (http://www.anu.ac.ke/1403/vcalc-the-first-ever-social-platform-for-the-world-of-math/). 4 November 2014. Retrieved 2016-05-05.
46. "Wired.Com" (http://www.springsource.org/files/uploads/all/pdf_files/customer/S2_CaseStudy_Wired_USLET_EN.pdf) (PDF). *www.springsource.org*. Retrieved 2015-06-02.
47. "XWiki SAS" (http://www.springsource.org/files/uploads/all/pdf_files/customer/S2_CaseStudy_XWiki.pdf) (PDF). *www.springsource.org*. Retrieved 2015-06-02.
48. "Grooscript Documentation" (https://web.archive.org/web/20170628212516/http://grooscript.org/doc.html). 12 Sep 2016. Archived from the original (http://grooscript.org/doc.html) on 28 June 2017. Retrieved 4 July 2017.
49. "Presentation at SpringOne/2GX on Grooscript" (https://www.infoq.com/presentations/grooscript). 13 Dec 2015.
50. "Grooscript online conversions" (https://web.archive.org/web/20170709170154/http://grooscript.org/conversions.html). 15 May 2017. Archived from the original (http://grooscript.org/conversions.html) on 9 July 2017. Retrieved 4 July 2017.

Sources

- König, Dierk; Paul King; Guillaume Laforge; Hamlet D'Arcy; Cédric Champeau; Erik Pragt; Jon Skeet (2015). *Groovy in Action, Second Edition*. Manning. ISBN 978-1-935182-44-3.
- Barclay, Kenneth; John Savage (2007). *Groovy Programming: An Introduction for Java Developers* (https://web.archive.org/web/20100112040339/http://www.elsevier.com/wps/find/bookdescription.cws_home/709814/description#description). ISBN 978-0-12-372507-3. Archived from the original (http://www.elsevier.com/wps/find/bookdescription.cws_home/709814/description#description) on 2010-01-12. Retrieved 2007-07-19.
- Davis, Scott (2008). *Groovy Recipes: Greasing the Wheels of Java* (https://archive.org/details/isbn_9780978739294). ISBN 978-0-9787392-9-4.
- Subramaniam, Venkat (2008). *Programming Groovy: Dynamic Productivity for the Java Developer* (https://archive.org/details/programminggroov0000subr). ISBN 978-1-934356-09-

8.

External links

- Official website (https://groovy-lang.org)
- Groovy (https://curlie.org/Computers/Programming/Languages/Java/Extensions/Groovy) at Curlie