groovys Documentation

1.0.0

zj

2016 01 14

Contents

Contents:

CHAPTER 1

1.1 Groovy

1.1.1

Groovy windows `Mac OSX`,`Linux` `Cygwin` GVM Groovy Groovy

- zipBinary Release | Source Release
- JavaDoc and zipped online documentation
- & & : Distribution bundle

`invokedynamic`

Groovy , (snapshot) CI `Artifactory's OSS`

Groovy 2.4 Java 6 Java 8 Java 9 snapshot `groovy-nio` Java 7 `invokeDynamic` Java 7+ Java 8
Groovy CI Java Groovy 10000 Java Groovy

1.1.2 Maven Repository

Groovy , `maven repositories` JCenter maven repository

Gradle	Maven	Explanation
org.codehaus.groovy:groovy:2.4.5	**<groupId>org.codehaus.groovy</groupId>** <artifactId>groovy</artifactId> <version>2.4.5</version>	Just the core of groovy without the modules (see below)
org.codehaus.groovy:groovy- $module:2.4.5‘	<groupId>org.codehaus.groovy</groupId> <artifactId>groovy- $module</artifactId> <version>2.4.5</version>	**Example:** <artifactId>groovy-sql </artifactId>
org.codehaus.groovy:groovy-all:2.4.5	<groupId>org.codehaus.groovy</groupId> <artifactId>groovy-all</artifactId> <version>2.4.5</version>	

`InvokeDynamic` jars Gradle `:indy` Maven `<classifier>indy</classifier>`

1.1.3 GVM (the Groovy enVironment Manager)

GVM (Mac OSX, Linux, Cygwin, Solaris or FreeBSD) Bash Groovy

`terminal`

```
$ curl -s get.gvmtool.net | bash
```

`terminal`

```
$ source "$HOME/.gvm/bin/gvm-init.sh"
```

Groovy

```
$ gvm install groovy
```

```
$ groovy -version
```

1.1.4 Groovy

Mac OS X

`MacOS` MacPorts :

```
sudo port install groovy
```

`MacOS` Homebrew :

```
       brew install groovy

Windows
```

Windows NSIS Windows installer.

Other Distributions

You may download other distributions of Groovy from this site.

Source Code

GitHub

1.1.5 IDE plugin

IDE , IDE

1.1.6

Groovy.

-
- `GROOVY_HOME`
- `GROOVY_HOME/bin` PATH
- `JAVA_HOME` JDK OS X `/Library/Java/Home` unix

`/usr/java` Ant Maven

Groovy :

```
groovysh
```

Groovy shell Groovy Swing interactive console ,

```
groovyConsole
```

`Groovy script`

```
groovy SomeScript
```

1.2 Groovy

1.2.1 Working with IO

Groovy IO JAVA Groovy Groovy reader

- the `java.io.File` class: http://docs.groovy-lang.org/latest/html/groovy-jdk/java/io/File.html
- the `java.io.InputStream` class: http://docs.groovy-lang.org/latest/html/groovy-jdk/java/io/InputStream.html
- the `java.io.OutputStream` class: http://docs.groovy-lang.org/latest/html/groovy-jdk/java/io/OutputStream.html
- the `java.io.Reader` class: http://docs.groovy-lang.org/latest/html/groovy-jdk/java/io/Reader.html
- the `java.io.Writer` class: http://docs.groovy-lang.org/latest/html/groovy-jdk/java/io/Writer.html
- the `java.nio.file.Path` class: http://docs.groovy-lang.org/latest/html/groovy-jdk/java/nio/file/Path.html

GDK API .

```
new File(baseDir, 'haiku.txt').eachLine { line ->
    println line
}
```

eachLine Groovy File

```
new File(baseDir, 'haiku.txt').eachLine { line, nb ->
    println "Line $nb: $line"
}
```

eachLine Groovy IO

Reader Groovy reader

```
def count = 0, MAXSIZE = 3
new File(baseDir,"haiku.txt").withReader { reader ->
    while (reader.readLine()) {
        if (++count > MAXSIZE) {
            throw new RuntimeException('Haiku should only have 3 verses')
        }
    }
}
```

list

```
def list = new File(baseDir, 'haiku.txt').collect {it}
```

as

```
def array = new File(baseDir, 'haiku.txt') as String[]
```

byte[]

Groovy

```
byte[] contents = file.bytes
```

IO Groovy

InputStream :

```
def is = new File(baseDir,'haiku.txt').newInputStream()
// do something ...
is.close()
```

inputstream Groovy withInputStream

```
new File(baseDir,'haiku.txt').withInputStream { stream ->
    // do something ...
}
```

Writer

```
new File(baseDir,'haiku.txt').withWriter('utf-8') { writer ->
    writer.writeLine 'Into the ancient pond'
    writer.writeLine 'A frog jumps'
    writer.writeLine 'Water's sound!'
}
```

<<

```
new File(baseDir,'haiku.txt') << '''Into the ancient pond
A frog jumps
Water's sound!'''
```

Writer

```
file.bytes = [66,22,11]
```

```
def os = new File(baseDir,'data.bin').newOutputStream()
// do something ...
os.close()
```

withOutputStream

```
new File(baseDir,'data.bin').withOutputStream { stream ->
    // do something ...
}
```

Groovy

```
dir.eachFile { file ->    //<1>
    println file.name
}
dir.eachFileMatch(~/.*\.txt/) { file ->   //<2>
    println file.name
}
```

<1>

<2>

eachFileRecurse:

```
dir.eachFileRecurse { file ->    //<1>
    println file.name
}

dir.eachFileRecurse(FileType.FILES) { file ->  //<2>
    println file.name
}
```

<1>

<2>

traverse

```
dir.traverse { file ->
    if (file.directory && file.name=='bin') {
        FileVisitResult.TERMINATE               //<1>
```

```
    } else {
        println file.name
        FileVisitResult.CONTINUE                //<2>
    }

}
```

<1> bin

<2>

Data & objects

JAVA java.io.DataOutputStream & java.io.DataInputStream Groovy

```
boolean b = true
String message = 'Hello from Groovy'
// Serialize data into a file
file.withDataOutputStream { out ->
    out.writeBoolean(b)
    out.writeUTF(message)
}
// ...
// Then read it back
file.withDataInputStream { input ->
    assert input.readBoolean() == b
    assert input.readUTF() == message
}
```

Serializable

```
Person p = new Person(name:'Bob', age:76)
// Serialize data into a file
file.withObjectOutputStream { out ->
    out.writeObject(p)
}
// ...
// Then read it back
file.withObjectInputStream { input ->
    def p2 = input.readObject()
    assert p2.name == p.name
    assert p2.age == p.age
}
```

Groovy reader Groovy *exxcute()*

```
def process = "ls -l".execute()         //<1>
println "Found text ${process.text}"     //<2>
```

<1> ls

<2>

`execute() java.lang.Process in/out/err streams`

```
def process = "ls -l".execute()          //<1>
process.in.eachLine { line ->            //<2>
    println line                                    //<3>
}
```

<1>

<2>

<3>

out .

shell windows

```
def process = "dir".execute()
println "${process.text}"
```

IOException Cannot run program “dir”: CreateProcess error=2, The system cannot find the file specified dir CreateProcess error=2,

`dir cmd.exe`

```
def process = "cmd /c dir".execute()
println "${process.text}"
```

`java.lang.Process` javadoc

: Because some native platforms only provide limited buffer size for standard input and output streams, failure to promptly write the input stream or read the output stream of the subprocess may cause the subprocess to block, and even deadlock

---

Groovy

```
def p = "rm -f foo.tmp".execute([], tmpDir)
p.consumeProcessOutput()
p.waitFor()
```

`consumeProcessOutput` `StringBuffer`, `InputStream`, `OutputStream` GDK API for java.lang.Process .

`pipeTo`

Pipes in action

```
proc1 = 'ls'.execute()
proc2 = 'tr -d o'.execute()
proc3 = 'tr -d e'.execute()
proc4 = 'tr -d i'.execute()
proc1 | proc2 | proc3 | proc4
proc4.waitFor()
if (proc4.exitValue()) {
    println proc4.err.text
} else {
    println proc4.text
}
```

Consuming errors

```
def sout = new StringBuilder()
def serr = new StringBuilder()
proc2 = 'tr -d o'.execute()
proc3 = 'tr -d e'.execute()
proc4 = 'tr -d i'.execute()
proc4.consumeProcessOutput(sout, serr)
proc2 | proc3 | proc4
[proc2, proc3].each { it.consumeProcessErrorStream(serr) }
proc2.withWriter { writer ->
    writer << 'testfile.groovy'
}
proc4.waitForOrKill(1000)
println "Standard output: $sout"
println "Standard error: $serr"
```

1.2.2 Working with collections

Groovy lists maps ranges Java Groovy

Lists

lists []

```
def list = [5, 6, 7, 8]
assert list.get(2) == 7
assert list[2] == 7
assert list instanceof java.util.List

def emptyList = []
assert emptyList.size() == 0
emptyList.add(5)
assert emptyList.size() == 1
```

list java.util.List lists lists

```
def list1 = ['a', 'b', 'c']
//construct a new list, seeded with the same items as in list1
def list2 = new ArrayList<String>(list1)

assert list2 == list1 // == checks that each corresponding element is the same

// clone() can also be called
def list3 = list1.clone()
assert list3 == list1
```

list

```
def list = [5, 6, 7, 8]
assert list.size() == 4
assert list.getClass() == ArrayList     // the specific kind of list being used

assert list[2] == 7                     // indexing starts at 0
assert list.getAt(2) == 7               // equivalent method to subscript operator []
assert list.get(2) == 7                 // alternative method
```

```
list[2] = 9
assert list == [5, 6, 9, 8,]           // trailing comma OK

list.putAt(2, 10)                       // equivalent method to [] when value being changed
assert list == [5, 6, 10, 8]
assert list.set(2, 11) == 10            // alternative method that returns old value
assert list == [5, 6, 11, 8]

assert ['a', 1, 'a', 'a', 2.5, 2.5f, 2.5d, 'hello', 7g, null, 9 as byte]
//objects can be of different types; duplicates allowed

assert [1, 2, 3, 4, 5][-1] == 5             // use negative indices to count from the end
assert [1, 2, 3, 4, 5][-2] == 4
assert [1, 2, 3, 4, 5].getAt(-2) == 4       // getAt() available with negative index...
try {
    [1, 2, 3, 4, 5].get(-2)                 // but negative index not allowed with get()
    assert false
} catch (e) {
    assert e instanceof ArrayIndexOutOfBoundsException
}
```

.

```
assert ![]              // an empty list evaluates as false

//all other lists, irrespective of contents, evaluate as true
assert [1] && ['a'] && [0] && [0.0] && [false] && [null]
```

each eachWithIndex .. code-block:: groovy

[1, 2, 3].each { println “Item: $it” // *it* is an implicit parameter corresponding to the current element

} [’a’, ‘b’, ‘c’].eachWithIndex { it, i -> // *it* is the current element, while *i* is the index

println “$i: $it”

}

mapping Groovy *collect* :

```
assert [1, 2, 3].collect { it * 2 } == [2, 4, 6]

// shortcut syntax instead of collect
assert [1, 2, 3]*.multiply(2) == [1, 2, 3].collect { it.multiply(2) }

def list = [0]
// it is possible to give `collect` the list which collects the elements
assert [1, 2, 3].collect(list) { it * 2 } == [0, 2, 4, 6]
assert list == [0, 2, 4, 6]
```

Groovy :

```
assert [1, 2, 3].find { it > 1 } == 2           // find 1st element matching criteria
assert [1, 2, 3].findAll { it > 1 } == [2, 3]   // find all elements matching critieria
assert ['a', 'b', 'c', 'd', 'e'].findIndexOf {      // find index of 1st element matching criteria
    it in ['c', 'e', 'g']
} == 2

assert ['a', 'b', 'c', 'd', 'c'].indexOf('c') == 2  // index returned
assert ['a', 'b', 'c', 'd', 'c'].indexOf('z') == -1 // index -1 means value not in list
assert ['a', 'b', 'c', 'd', 'c'].lastIndexOf('c') == 4

assert [1, 2, 3].every { it < 5 }               // returns true if all elements match the predicate
assert ![1, 2, 3].every { it < 3 }
assert [1, 2, 3].any { it > 2 }                 // returns true if any element matches the predicate
assert ![1, 2, 3].any { it > 3 }

assert [1, 2, 3, 4, 5, 6].sum() == 21                // sum anything with a plus() method
assert ['a', 'b', 'c', 'd', 'e'].sum {
    it == 'a' ? 1 : it == 'b' ? 2 : it == 'c' ? 3 : it == 'd' ? 4 : it == 'e' ? 5 : 0
    // custom value to use in sum
} == 15
assert ['a', 'b', 'c', 'd', 'e'].sum { ((char) it) - ((char) 'a') } == 10
assert ['a', 'b', 'c', 'd', 'e'].sum() == 'abcde'
assert [['a', 'b'], ['c', 'd']].sum() == ['a', 'b', 'c', 'd']

// an initial value can be provided
assert [].sum(1000) == 1000
assert [1, 2, 3].sum(1000) == 1006

assert [1, 2, 3].join('-') == '1-2-3'           // String joining
assert [1, 2, 3].inject('counting: ') {
    str, item -> str + item                     // reduce operation
} == 'counting: 123'
assert [1, 2, 3].inject(0) { count, item ->
    count + item
} == 6
```

Groovy :

```
def list = [9, 4, 2, 10, 5]
assert list.max() == 10
assert list.min() == 2

// we can also compare single characters, as anything comparable
assert ['x', 'y', 'a', 'z'].min() == 'a'

// we can use a closure to specify the sorting behaviour
def list2 = ['abc', 'z', 'xyzuvw', 'Hello', '321']
assert list2.max { it.size() } == 'xyzuvw'
assert list2.min { it.size() } == 'z'
```

Comparator

```
Comparator mc = { a, b -> a == b ? 0 : (a < b ? -1 : 1) }

def list = [7, 4, 9, -6, -1, 11, 2, 3, -9, 5, -13]
assert list.max(mc) == 11
assert list.min(mc) == -13

Comparator mc2 = { a, b -> a == b ? 0 : (Math.abs(a) < Math.abs(b)) ? -1 : 1 }
```

```
assert list.max(mc2) == -13
assert list.min(mc2) == -1

assert list.max { a, b -> a.equals(b) ? 0 : Math.abs(a) < Math.abs(b) ? -1 : 1 } == -13
assert list.min { a, b -> a.equals(b) ? 0 : Math.abs(a) < Math.abs(b) ? -1 : 1 } == -1
```

.

[] <<

```
def list = []
assert list.empty

list << 5
assert list.size() == 1

list << 7 << 'i' << 11
assert list == [5, 7, 'i', 11]

list << ['m', 'o']
assert list == [5, 7, 'i', 11, ['m', 'o']]

//first item in chain of << is target list
assert ([1, 2] << 3 << [4, 5] << 6) == [1, 2, 3, [4, 5], 6]

//using leftShift is equivalent to using <<
assert ([1, 2, 3] << 4) == ([1, 2, 3].leftShift(4))
```

:

```
assert [1, 2] + 3 + [4, 5] + 6 == [1, 2, 3, 4, 5, 6]
// equivalent to calling the `plus` method
assert [1, 2].plus(3).plus([4, 5]).plus(6) == [1, 2, 3, 4, 5, 6]

def a = [1, 2, 3]
a += 4      // creates a new list and assigns it to `a`
a += [5, 6]
assert a == [1, 2, 3, 4, 5, 6]

assert [1, *[222, 333], 456] == [1, 222, 333, 456]
assert [*[1, 2, 3]] == [1, 2, 3]
assert [1, [2, 3, [4, 5], 6], 7, [8, 9]].flatten() == [1, 2, 3, 4, 5, 6, 7, 8, 9]

def list = [1, 2]
list.add(3)
list.addAll([5, 4])
assert list == [1, 2, 3, 5, 4]

list = [1, 2]
list.add(1, 3) // add 3 just before index 1
assert list == [1, 3, 2]

list.addAll(2, [5, 4]) //add [5,4] just before index 2
assert list == [1, 3, 5, 4, 2]

list = ['a', 'b', 'z', 'e', 'u', 'v', 'g']
```

```
list[8] = 'x' // the [] operator is growing the list as needed
// nulls inserted if required
assert list == ['a', 'b', 'z', 'e', 'u', 'v', 'g', null, 'x']
```

<< , + list

Groovy

```
assert ['a','b','c','b','b'] - 'c' == ['a','b','b','b']
assert ['a','b','c','b','b'] - 'b' == ['a','c']
assert ['a','b','c','b','b'] - ['b','c'] == ['a']

def list = [1,2,3,4,3,2,1]
list -= 3           // creates a new list by removing `3` from the original one
assert list == [1,2,4,2,1]
assert ( list -= [2,4] ) == [1,1]
```

:

```
def list = [1,2,3,4,5,6,2,2,1]
assert list.remove(2) == 3          // remove the third element, and return it
assert list == [1,2,4,5,6,2,2,1]
```

remove

```
def list= ['a','b','c','b','b']
assert list.remove('c')             // remove 'c', and return true because element removed
assert list.remove('b')             // remove first 'b', and return true because element removed

assert ! list.remove('z')           // return false because no elements removed
assert list == ['a','b','b']
```

clear

```
def list= ['a',2,'c',4]
list.clear()
assert list == []
```

.Set operations

Groovy

```
assert 'a' in ['a','b','c']             // returns true if an element belongs to the list
assert ['a','b','c'].contains('a')      // equivalent to the `contains` method in Java
assert [1,3,4].containsAll([1,4])       // `containsAll` will check that all elements are found

assert [1,2,3,3,3,3,4,5].count(3) == 4  // count the number of elements which have some value
assert [1,2,3,3,3,3,4,5].count {
    it%2==0                             // count the number of elements which match the predicate
} == 2

assert [1,2,4,6,8,10,12].intersect([1,3,6,9,12]) == [1,6,12]  //

assert [1,2,3].disjoint( [4,6,9] )  //
assert ![1,2,3].disjoint( [2,4,6] )
```

.

Groovy :

```
assert [6, 3, 9, 2, 7, 1, 5].sort() == [1, 2, 3, 5, 6, 7, 9]

def list = ['abc', 'z', 'xyzuvw', 'Hello', '321']
assert list.sort {
    it.size()
} == ['z', 'abc', '321', 'Hello', 'xyzuvw']

def list2 = [7, 4, -6, -1, 11, 2, 3, -9, 5, -13]
assert list2.sort { a, b -> a == b ? 0 : Math.abs(a) < Math.abs(b) ? -1 : 1 } ==
        [-1, 2, 3, 4, 5, -6, 7, -9, 11, -13]

Comparator mc = { a, b -> a == b ? 0 : Math.abs(a) < Math.abs(b) ? -1 : 1 }

// JDK 8+ only
// list2.sort(mc)
// assert list2 == [-1, 2, 3, 4, 5, -6, 7, -9, 11, -13]

def list3 = [6, -3, 9, 2, -7, 1, 5]

Collections.sort(list3)
assert list3 == [-7, -3, 1, 2, 5, 6, 9]

Collections.sort(list3, mc)
assert list3 == [1, 2, -3, 5, 6, -7, 9]
```

.

Groovy

```
assert [1, 2, 3] * 3 == [1, 2, 3, 1, 2, 3, 1, 2, 3]
assert [1, 2, 3].multiply(2) == [1, 2, 3, 1, 2, 3]
assert Collections.nCopies(3, 'b') == ['b', 'b', 'b']

// nCopies from the JDK has different semantics than multiply for lists
assert Collections.nCopies(2, [1, 2]) == [[1, 2], [1, 2]] //not [1,2,1,2]
```

1.2.3 Maps

Map literals

Groovy `[:]` map:

```
def map = [name: 'Gromit', likes: 'cheese', id: 1234]
assert map.get('name') == 'Gromit'
assert map.get('id') == 1234
assert map['name'] == 'Gromit'
assert map['id'] == 1234
assert map instanceof java.util.Map

def emptyMap = [:]
assert emptyMap.size() == 0
```

```
emptyMap.put("foo", 5)
assert emptyMap.size() == 1
assert emptyMap.get("foo") == 5
```

Map key String : [a:1] ['a':1] Map keys are strings by default: [a:1] is equivalent to ['a':1] a a map key

```
def a = 'Bob'
def ages = [a: 43]
assert ages['Bob'] == null // `Bob` is not found
assert ages['a'] == 43     // because `a` is a literal!

ages = [(a): 43]            // now we escape `a` by using parenthesis
assert ages['Bob'] == 43   // and the value is found!
```

Map clone :

```
def map = [
        simple : 123,
        complex: [a: 1, b: 2]
]
def map2 = map.clone()
assert map2.get('simple') == map.get('simple')
assert map2.get('complex') == map.get('complex')
map2.get('complex').put('c', 3)
assert map.get('complex').get('c') == 3
```

map

Map property notation

Maps beans key map

```
def map = [name: 'Gromit', likes: 'cheese', id: 1234]
assert map.name == 'Gromit'     // can be used instead of map.get('Gromit')
assert map.id == 1234

def emptyMap = [:]
assert emptyMap.size() == 0
emptyMap.foo = 5
assert emptyMap.size() == 1
assert emptyMap.foo == 5
```

map.foo map key foo foo.class *null* map *class* key key *class* getClass()

```
def map = [name: 'Gromit', likes: 'cheese', id: 1234]
assert map.class == null
assert map.get('class') == null
assert map.getClass() == LinkedHashMap // this is probably what you want

map = [1       : 'a',
       (true)  : 'p',
       (false): 'q',
       (null)  : 'x',
       'null' : 'z']
assert map.containsKey(1) // 1 is not an identifier so used as is
assert map.true == null
assert map.false == null
assert map.get(true) == 'p'
```

```
assert map.get(false) == 'q'
assert map.null == 'z'
assert map.get(null) == 'x'
```

maps

Groovy each eachWithIndex map map

```
def map = [
        Bob   : 42,
        Alice: 54,
        Max   : 33
]

// `entry` is a map entry
map.each { entry ->
    println "Name: $entry.key Age: $entry.value"
}

// `entry` is a map entry, `i` the index in the map
map.eachWithIndex { entry, i ->
    println "$i - Name: $entry.key Age: $entry.value"
}

// Alternatively you can use key and value directly
map.each { key, value ->
    println "Name: $key Age: $value"
}

// Key, value and i as the index in the map
map.eachWithIndex { key, value, i ->
    println "$i - Name: $key Age: $value"
}
```

Map

map put putAll :

```
def defaults = [1: 'a', 2: 'b', 3: 'c', 4: 'd']
def overrides = [2: 'z', 5: 'x', 13: 'x']

def result = new LinkedHashMap(defaults)
result.put(15, 't')
result[17] = 'u'
result.putAll(overrides)
assert result == [1: 'a', 2: 'z', 3: 'c', 4: 'd', 5: 'x', 13: 'x', 15: 't', 17: 'u']
```

clear map

```
def m = [1:'a', 2:'b']
assert m.get(1) == 'a'
m.clear()
assert m == [:]
```

Maps object equals hashcode hash code key GString map key GString hash code String hash code

```
def key = 'some key'
def map = [:]
def gstringKey = "${key.toUpperCase()}"
map.put(gstringKey,'value')
assert map.get('SOME KEY') == null
```

```
def map = [1:'a', 2:'b', 3:'c']

def entries = map.entrySet()
entries.each { entry ->
  assert entry.key in [1,2,3]
  assert entry.value in ['a','b','c']
}

def keys = map.keySet()
assert keys == [1,2,3] as Set
```

Mutating values returned by the view (be it a map entry, a key or a value) is highly discouraged because success of the operation directly depends on the type of the map being manipulated. In particular, Groovy relies on collections from the JDK that in general make no guarantee that a collection can safely be manipulated through keySet, entrySet, or values.

Groovy http://www.groovy-lang.org/groovy-dev-kit.html#List-Filtering[list] The Groovy development kit contains filtering, searching and collecting methods similar to those found for lists:

```
def people = [
    1: [name:'Bob', age: 32, gender: 'M'],
    2: [name:'Johnny', age: 36, gender: 'M'],
    3: [name:'Claire', age: 21, gender: 'F'],
    4: [name:'Amy', age: 54, gender:'F']
]

def bob = people.find { it.value.name == 'Bob' } // find a single entry
def females = people.findAll { it.value.gender == 'F' }

// both return entries, but you can use collect to retrieve the ages for example
def ageOfBob = bob.value.age
def agesOfFemales = females.collect {
    it.value.age
}

assert ageOfBob == 32
assert agesOfFemales == [21,54]

// but you could also use a key/pair value as the parameters of the closures
def agesOfMales = people.findAll { id, person ->
    person.gender == 'M'
}.collect { id, person ->
    person.age
}
assert agesOfMales == [32, 36]

// `every` returns true if all entries match the predicate
assert people.every { id, person ->
    person.age > 18
}

// `any` returns true if any entry matches the predicate
```

```
assert people.any { id, person ->
    person.age == 54
}
```

```
assert ['a', 7, 'b', [2, 3]].groupBy {
    it.class
} == [(String)   : ['a', 'b'],
      (Integer)  : [7],
      (ArrayList): [[2, 3]]
]

assert [
        [name: 'Clark', city: 'London'], [name: 'Sharma', city: 'London'],
        [name: 'Maradona', city: 'LA'], [name: 'Zhang', city: 'HK'],
        [name: 'Ali', city: 'HK'], [name: 'Liu', city: 'HK'],
].groupBy { it.city } == [
        London: [[name: 'Clark', city: 'London'],
                 [name: 'Sharma', city: 'London']],
        LA    : [[name: 'Maradona', city: 'LA']],
        HK    : [[name: 'Zhang', city: 'HK'],
                 [name: 'Ali', city: 'HK'],
                 [name: 'Liu', city: 'HK']],
]
```

Ranges

Ranges Ranges List Range `java.util.List` Ranges `..` Ranges `..<`

```
// an inclusive range
def range = 5..8
assert range.size() == 4
assert range.get(2) == 7
assert range[2] == 7
assert range instanceof java.util.List
assert range.contains(5)
assert range.contains(8)

// lets use a half-open range
range = 5..<8
assert range.size() == 3
assert range.get(2) == 7
assert range[2] == 7
assert range instanceof java.util.List
assert range.contains(5)
assert !range.contains(8)

//get the end points of the range without using indexes
range = 1..10
assert range.from == 1
assert range.to == 10
```

int ranges java

Ranges `java.lang.Comparable` java next() previous() String :

```
// an inclusive range
def range = 'a'..'d'
```

```
assert range.size() == 4
assert range.get(2) == 'c'
assert range[2] == 'c'
assert range instanceof java.util.List
assert range.contains('a')
assert range.contains('d')
assert !range.contains('e')
```

for

```
for (i in 1..10) {
    println "Hello ${i}"
}
```

Groovy each

```
(1..10).each { i ->
    println "Hello ${i}"
}
```

Ranges switch

```
switch (years) {
    case 1..10: interestRate = 0.076; break;
    case 11..25: interestRate = 0.052; break;
    default: interestRate = 0.037;
}
```

lists maps Groovy

```
def listOfMaps = [['a': 11, 'b': 12], ['a': 21, 'b': 22]]
assert listOfMaps.a == [11, 21] //GPath notation
assert listOfMaps*.a == [11, 21] //spread dot notation

listOfMaps = [['a': 11, 'b': 12], ['a': 21, 'b': 22], null]
assert listOfMaps*.a == [11, 21, null] // caters for null values
assert listOfMaps*.a == listOfMaps.collect { it?.a } //equivalent notation
// But this will only collect non-null values
assert listOfMaps.a == [11,21]
```

Spread operator *putAll*

```
assert [ 'z': 900,
         *: ['a': 100, 'b': 200], 'a': 300] == ['a': 300, 'b': 200, 'z': 900]
//spread map notation in map definition
assert [*: [3: 3, *: [5: 5]], 7: 7] == [3: 3, 5: 5, 7: 7]

def f = { [1: 'u', 2: 'v', 3: 'w'] }
assert [*: f(), 10: 'zz'] == [1: 'u', 10: 'zz', 2: 'v', 3: 'w']
//spread map notation in function arguments
f = { map -> map.c }
assert f(*: ['a': 10, 'b': 20, 'c': 30], 'e': 50) == 30

f = { m, i, j, k -> [m, i, j, k] }
//using spread map notation with mixed unnamed and named arguments
assert f('e': 100, *[4, 5], *: ['a': 10, 'b': 20, 'c': 30], 6) ==
        [["e": 100, "b": 20, "c": 30, "a": 10], 4, 5, 6]
```

```
```

*.

```
assert [1, 3, 5] == ['a', 'few', 'words']*.size()

class Person {
    String name
    int age
}
def persons = [new Person(name:'Hugo', age:17), new Person(name:'Sandra',age:19)]
assert [17, 19] == persons*.age
```

listsmapsarrays

```
def text = 'nice cheese gromit!'
def x = text[2]

assert x == 'c'
assert x.class == String

def sub = text[5..10]
assert sub == 'cheese'

def list = [10, 11, 12, 13]
def answer = list[2,3]
assert answer == [12,13]
```

ranges

```
list = 100..200
sub = list[1, 3, 20..25, 33]
assert sub == [101, 103, 120, 121, 122, 123, 124, 125, 133]
```

:

```
list = ['a','x','x','d']
list[1..2] = ['b','c']
assert list == ['a','b','c','d']
```

List array String

```
text = "nice cheese gromit!"
x = text[-1]
assert x == "!"

def name = text[-7..-2]
assert name == "gromit"
```

```
text = "nice cheese gromit!"
name = text[3..1]
assert name == "eci"
```

lists , maps ranges Groovy

Groovy API

- methods added to Iterable can be found here

- methods added to Iterator can be found here
- methods added to Collection can be found here
- methods added to List can be found here
- methods added to Map can be found here

1.2.4 Handy utilities ()

ConfigSlurper

ConfigSlurper Groovy script Java `*.properties` `ConfigSlurper`

```
def config = new ConfigSlurper().parse('''
    app.date = new Date()                       //<1>
    app.age  = 42
    app {                                                   //<2>
        name = "Test${42}"
    }
''')

assert config.app.date instanceof Date
assert config.app.age == 42
assert config.app.name == 'Test42'
```

<1>

<2>

`parse` `groovy.util.ConfigObject` `ConfigObject` `java.util.Map` `ConfigObject` `null`.

```
def config = new ConfigSlurper().parse('''
    app.date = new Date()
    app.age  = 42
    app.name = "Test${42}"
''')

assert config.test != null      //<1>
```

<1> *config.test* *ConfigObject*

```
def config = new ConfigSlurper().parse('''
    app."person.age"  = 42
''')

assert config.app."person.age" == 42
```

`ConfigSlurper` `environments` *ConfigSlurper(String)*

```
def config = new ConfigSlurper('development').parse('''
  environments {
        development {
            app.port = 8080
        }

        test {
            app.port = 8082
        }
```

```
        production {
            app.port = 80
        }
    }
''')

assert config.app.port == 8080
```

ConfigSlurper environments environments registerConditionalBlock environments

```
def slurper = new ConfigSlurper()
slurper.registerConditionalBlock('myProject', 'developers')             // <1>

def config = slurper.parse('''
  sendMail = true

  myProject {
        developers {
            sendMail = false
        }
    }
''')

assert !config.sendMail
```

<1> *ConfigSlurper*

Java , toProperties ConfigObject java.util.Properties *.properties Properties

```
def config = new ConfigSlurper().parse('''
    app.date = new Date()
    app.age  = 42
    app {
        name = "Test${42}"
    }
''')

def properties = config.toProperties()

assert properties."app.date" instanceof String
assert properties."app.age" == '42'
assert properties."app.name" == 'Test42'
```

Expando

Expando

```
def expando = new Expando()
expando.name = 'John'

assert expando.name == 'John'
```

```
def expando = new Expando()
expando.toString = { -> 'John' }
expando.say = { String s -> "John says: ${s}" }

assert expando as String == 'John'
assert expando.say('Hi') == 'John says: Hi'
```

list, map and set

Groovy lists, map, sets `java.beans.PropertyChangeEvent`

`PropertyChangeEvent` *ObservableList.ElementAddedEvent*

```
def event                                           // <1>
def listener = {
    if (it instanceof ObservableList.ElementEvent)  {           //<2>
        event = it
    }
} as PropertyChangeListener


def observable = [1, 2, 3] as ObservableList                                //<3>
observable.addPropertyChangeListener(listener)                              //<4>

observable.add 42

assert event instanceof ObservableList.ElementAddedEvent

def elementAddedEvent = event as ObservableList.ElementAddedEvent
assert elementAddedEvent.changeType == ObservableList.ChangeType.ADDED
assert elementAddedEvent.index == 3
assert elementAddedEvent.oldValue == null
assert elementAddedEvent.newValue == 42
```

<1> `PropertyChangeEventListener`

<2> `ObservableList.ElementEvent`

<3>

<4>

<5>

`ObservableList.ElementAddedEvent` `PropertyChangeEvent` `clear()` `ObservableList.ElementClearedEvent`

```
def event
def listener = {
    if (it instanceof ObservableList.ElementEvent)  {
        event = it
    }
} as PropertyChangeListener


def observable = [1, 2, 3] as ObservableList
observable.addPropertyChangeListener(listener)

observable.clear()

assert event instanceof ObservableList.ElementClearedEvent

def elementClearedEvent = event as ObservableList.ElementClearedEvent
assert elementClearedEvent.values == [1, 2, 3]
assert observable.size() == 0
```

`ObservableMap` and `ObservableSet` `ObservableList`

Indices and tables

- genindex
- modindex
- search