Groovy

[Gradle]

・この本を参考にまとめる

概要

・JVM 上で動作する
・Java と直接的な連携が特徴
・動的言語であり直接スクリプトを実行できる
・バイトコードがメモリ上に直接生成される (中間的に Java ソースは生成されない)
・groovyc でクラスファイルも作成可能

構成

import

■デフォルトでインポート

・java.lang.*
・java.io.*
・java.net.*
・java.util.*
・groovy.lang.*
・groovy.util.*
・java.math.BigDecimal
・java.math.BigInteger

■別名

・import as で別名を付けられる

```
import java.util.ArrayList as AL
def l = new AL()
l.add(1)
print l
```

スクリプト

・見た目上クラスに所属しないコード (コンパイルされると所属する)
・groovy コマンドで実行できる

■実行する

```
>groovy scriptname.groovy
```

■バインディング変数

・型も def も使用しない
・常にオブジェクト型

クラス定義

・public クラス定義を含むファイル名はクラスメイト一致する必要はない
・デフォルトで public アクセス
・以下のようなクラスを定義した Groovy ファイルは実行される
    ・main メソッドを持つクラス
    ・JUnit などのテストケース
    ・Runnable 実装クラス

■メソッドとコンストラクタ

・Map で引数と受け取ることで名前付き引数の利用が可能
・コンストラクタで名前付き引数を呼び出すとメンバを初期化 (デフォルトコンストラクタ以外にコンストラクタを定義する場合は明示的に実装する)
・引数のデフォルト値

```
class Person {
    String name
    int age

    def profile(){
        "i am ${name}, ${age} years old."
    }
    def changeProfile(name='foo',age=1){
        this.name=name
        this.age = age
    }
    String toString(){
        profile()
    }
    static main(args) {
        def p = new Person(name:'yagi',age:45)
        println p.profile()
        p.changeProfile()
        println p
    }
}
```

```
> groovy .\Person.groovy
i am yagi, 45 years old.
i am foo, 1 years old.
```

GroovyBeans

■ゲッター、セッターの自動生成

・フィールドのアクセッサが自動生成される
・アクセス修飾子を指定することで抑制できる

```
class SampleBean {
    def name
    def address
    private def age
    private def profAge;

    def getAddress(){
        address.toUpperCase();
    }
    def getProfAge(){
        age - 5;
    }
}
def bean = new SampleBean(name:'yagi',address:'aich',age:45)
println "${bean.name},${bean.address},${bean.age},${bean.profAge}"
```

```
> groovy .\GroovyBeans.groovy
```

```
yagi,AICH,45,40
```

プロパティ、フィールド、メソッド参照

■フィールド、プロパティ参照方法

```
class Sample {
    def name = 'Sample'
}

def s = new Sample()

println '1.' + s.name
println '2.' + s.'name'
def propName = 'name'
println '3.' + s."${propName}"
println '4.' + s['name']
println '5.' + s[propName]
println '6.' + s.getProperty(propName)


> groovy .\PropertyMethod.groovy
1.Sample
2.Sample
3.Sample
4.Sample
5.Sample
6.Sample
```

■メソッド参照

```
 class Sample {
    def method() {
        "method"
    }
}
def bean = new Sample()

println '1.' + bean.method()
def methodName = 'method'
println '2.' + bean."${methodName}"()


> groovy .\PropertyMethod.groovy
1.method
2.method
```

■プロパティ、メソッドの一覧

- obj.properties
- obj.metaClass.methods

```
groovy:000> f = new File(/c:\work/)
===> c:\work
groovy:000> f.properties
 ===> [directory:true, canonicalFile:C:\work, file:false, freeSpace:80082784256, invalid:false,
canonicalPath:C:\work, usableSpace
  :80082784256, hidden:false, totalSpace:217750581248, path:c:\work, name:work, prefixLength:3,
absolute:true, class:class java.io.
 File, parentFile:c:\, absolutePath:c:\work, parent:c:\, absoluteFile:c:\work]
groovy:000> f.metaClass.methods
===> [public boolean java.lang.Object.equals(java.lang.Object), public final native java.lang.Class
java.lang.Object.getClass(),
 public native int java.lang.Object.hashCode(), public final native void java.lang.Object.notify(),
public final native void java.
 lang.Object.notifyAll(), public java.lang.String java.lang.Object.toString(), public final void
java.lang.Object.wait() throws ja
  va.lang.InterruptedException, public final native void java.lang.Object.wait(long) throws
java.lang.InterruptedException, public
 final void java.lang.Object.wait(long,int) throws java.lang.InterruptedException, public boolean
```

```
java.io.File.canExecute(), publi
    :
```

・obj.metaClass.methods.name.sort().unique()

```
groovy:000> f.metaClass.methods.name.sort().unique()
===> [canExecute, canRead, canWrite, compareTo, createNewFile, createTempFile, delete, deleteOnExit,
equals, exists, getAbsoluteF
 ile, getAbsolutePath, getCanonicalFile, getCanonicalPath, getClass, getFreeSpace, getName,
getParent, getParentFile, getPath, get
 TotalSpace, getUsableSpace, hashCode, isAbsolute, isDirectory, isFile, isHidden, lastModified,
length, list, listFiles, listRoots
 , mkdir, mkdirs, notify, notifyAll, renameTo, setExecutable, setLastModified, setReadOnly,
setReadable, setWritable, toPath, toSt
 ring, toURI, toURL, wait]
```

データ型

■型指定を省略できる

・省略する場合、def を利用する
・def を指定もしくは、def も省略した場合 Object 型を指定したのと同様
・型指定してもコンパイル時にチェックされるわけではない
・既存メソッドをオーバーライドする場合は、型指定省略不可

■プリミティブ型の扱い

・ほとんどの場合、ラッパー型に変換される

```
int i = 100
println i.class.name
```

```
> groovy .\DataType.groovy
java.lang.Integer
```

■真偽値の扱い

・ラッパー型として扱われる

型	真	偽
整数	!0	0
浮動小数	!0.0	0.0
文字列	長さ >0	長さ 0
リスト、マップ	空ではない	空
参照型	null 以外	null

```
def map = [:]

if (!map) {
    map[1] = 'one'
}
if (map[1]) {
    map[2] = 'two'
}
println map
```

```
> groovy .\DataType.groovy
[1:one, 2:two]
```

■数値型の扱い

・接尾語を指定しない浮動小数リテラルは BigDecimal となる

接尾語	型
I/i	Integer
L/l	Long
F/f	Float
D/d	Double
G/g	BigInteger/BigDecimal

```
def a = 1
def b = 1.0
def c = 1D

println a.class.name
println b.class.name
println c.class.name


> groovy .\DataType.groovy
java.lang.Integer
java.math.BigDecimal
java.lang.Double
```

■文字列型

・GString
  ・ダブルクオート、スラッシュで囲った定数中の「$ 変数名」[${ 式 }」を実行時に展開する
・シングルクオートは、Java の文字列と同義
・/ 〜 /\ をエスケープ文字として扱わない、GString を展開する
・ダブル、シングルクオートをそれぞれ 3 つで囲む、$/ 〜 /$ と複数行

```
def num = 123
println "1.$num,${num*2}"
println '2.$num,${num*2}'

println """
** 3. **
$num
********
"""
println '''
** 4. **
$num
********
'''

def s = /5.c:\work/
println s

def s2 = $/
6.c:\work
  c:\work\test
/$
println s2


> groovy .\Strings.groovy
1.123,246
```

```
2.$num,${num*2}

** 3. **
123
********


** 4. **
$num
********

5.c:\work

6.c:\work
  c:\work\test
```

クロージャ

・生成時のコンテキストを含んだコードブロック
　・クロージャが生成された場所で可視である変数を参照、変更可能
・{ 〜 } で囲んだコードとして表記

```
def msg = 'hello'
c = {
    def d = new Date()
    println "$msg,$d"
}
c();


> groovy .\ClosureSample.groovy
hello,Sat Mar 04 18:21:42 JST 2017
```

■ -> を使って引数を与えることができる

・仮引数を指定しないと、引数を取らないクロージャとなる

```
 c = {
     msg ->
     def d = new Date()
     println "$msg,$d"
 }
  c('hello');


groovy .\ClosureSample.groovy
hello,Sat Mar 04 22:30:42 JST 2017
```

■引数リストを書かないと暗黙の引数 it を一つとるとみなされる

```
 c = {
     def d = new Date()
     println "$it,$d"
 }
  c('hello');


> groovy .\ClosureSample.groovy
hello,Sat Mar 04 22:32:58 JST 2017
```

■制御構造として利用

```
(1..10).each {
    print "$it,"
}
```

```
> groovy .\ClosureSample.groovy
1,2,3,4,5,6,7,8,9,10,
```

■リソース開放やクローズ

・http://docs.groovy-lang.org/latest/html/documentation/working-with-io.html

```
new File(/C:\Users\piroto\workspace\vscode\groovy_lesson\ClosureSample.groovy/).withReader {
    reader ->
    reader.each{
        println it
    }
}
```

```
> groovy .\ClosureSample.groovy
new File(/C:\Users\piroto\workspace\vscode\groovy_lesson\ClosureSample.groovy/).withReader {
    reader ->
    reader.each{
        println it
    }
}
```

■コレクション処理

grep

```
even = (1..10).grep{it % 2 == 0}
println even
```

```
> groovy .\ClosureSample.groovy
[2, 4, 6, 8, 10]
```

collect

```
square = (1..10).collect{it ** 2}
println square
```

```
> groovy .\ClosureSample.groovy
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

コレクション型

・http://groovy-lang.org/gdk.html

■範囲

```
println (1..5)
println (1 ..< 5)
```

> groovy .\CollectionSample.groovy

```
[1, 2, 3, 4, 5]
[1, 2, 3, 4]
```

■リスト

```
def list = [1,2,3,4,5]
println list[0]
println list[-1]
println list + [6,7,8]
println list
println list - [1,2]
println list
println list * 2
println list
println list << [9,10]
println list
```

```
> groovy .\CollectionSample.groovy
1
5
[1, 2, 3, 4, 5, 6, 7, 8]
[1, 2, 3, 4, 5]
[3, 4, 5]
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5, 1, 2, 3, 4, 5]
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5, [9, 10]]
[1, 2, 3, 4, 5, [9, 10]]
```

■マップ

```
def map = [1:'a',2:'b',3:'c']
println map
println map[1]
map[1] = 'd'
println map
println (1 in map)
map = ['a':1,'b':2]
println map."a"
def key = 'b'
println map."$key"
```

```
> groovy .\CollectionSample.groovy
[1:a, 2:b, 3:c]
a
[1:d, 2:b, 3:c]
true
1
2
```

演算子

・演算子の多くは特定メソッド呼び出しのシンタックスシュガー
・すべてではないが、オーバーロード可能

演算子	説明
?.	null セーフ
?:	エルビス演算子
.&	メソッドをクロージャとして参照
.@	フィールド / 属性値を推定
<=>	大小比較
..	範囲の生成 (閉区間)
..<	範囲の生成 (半開区間)

*.	展開ドット
*	リスト展開
*:	マップ展開
as	強制型変換
in	包含の判定
==~	正規表現マッチ
=~	Matcher オブジェクト正史江
~	パターンオブジェクト生成

```
println "=== ?. ==="
def s = null
println s?.toUpperCase()

println "=== ?: ==="
def c = { it?:"empty"}
println c("value")
println c(null)

println "=== *. ==="
println  ( ['a','b','c'] *.toUpperCase() )

println "=== .& ==="
println Math.abs(-2.5)
def abs = Math.&abs
println abs(-2.5)

println "=== .@ ==="
class Test {
    def name = "field"
    def getName() {
        "method"
    }
}
def t = new Test()
println t.name
println t.@name

println "=== * ==="
println ([1,2,*[3,4]])

println "=== *: ==="
println ([1:'a',2:'b',*:[3:'c',4:'d']])

println "=== as ==="
def ss = "s"
println ss.class.name
def cc = ss as char
println cc.class.name
```

```
> groovy .\OperatorSample.groovy
=== ?. ===
null
=== ?: ===
value
empty
=== *. ===
[A, B, C]
=== .& ===
2.5
2.5
=== .@ ===
method
field
=== * ===
[1, 2, 3, 4]
=== *: ===
[1:a, 2:b, 3:c, 4:d]
=== as ===
```

```
java.lang.String
java.lang.Character
```

■その他の演算子

・== は、equals() 同じ意味
・java での == は、Object#is() を利用

■マルチ代入

```
groovy:000> l = [1,2,3]
===> [1, 2, 3]
groovy:000>
groovy:000> (a,b,c) = l
===> [1, 2, 3]
groovy:000> a
===> 1
groovy:000> b
===> 2
groovy:000> c
===> 3
```

制御構造

■ switch-case 文

・機能的に if-else と同等
・isCase() を定義することで case 式で利用できるようになる

■ for 文

・通常の for および拡張 for が利用できる

```
for (i=0; i<10; i++) {}
for (int i: 0 ..<10) {}
```

・ in の利用が可能

```
for (i in 0..10){}
```

■ return 文

・return 直後に関数やクロージャ末尾に到達することが確実である場合、省略可能

例外処理

・catch する例外の型を省略した場合、java.lang.Exception を catch しているのと同じ
・チェック例外を明示的に処理しなくともよい

正規表現

・==~ 文字列全体にマッチするか
・=~ パターンにマッチする部分が含まれるか (Matcher オブジェクトを生成)
　・真偽判定で利用される場合には、find() が暗黙で呼ばれる

```
println ( 'pppiroto@gmail.com' ==~ /[a-z]+@[a-z]+\.[a-z]+/ )
println ( 'pppiroto@gmail.com' ==~ /[a-z]+/ )
println ( 'pppiroto@gmail.com' =~ /[a-z]+/ )
println (( 'pppiroto@gmail.com' =~ /[a-z]+/ )?"found":"not found")


> groovy .\RegexSample.groovy
true
false
java.util.regex.Matcher[pattern=[a-z]+ region=0,18 lastmatch=]
found
```

■後方参照、補足グループ

('pppiroto@gmail.com' =~ /([a-z])+@([a-z]+\.[a-z])+/).each{

```
    g0, g1, g2 ->
    println "id=$g1,host=$g2,addr=$g0"
}

('pppiroto@gmail.com' =~ /[(a-z)]+/ ).each{
    println it
}


>  groovy .\RegexSample.groovy
id=o,host=gmail.c,addr=pppiroto@gmail.c
pppiroto
gmail
com
```

Groovy API

- http://docs.groovy-lang.org/latest/html/gapi/

ビルダー

XML

- http://www.ibm.com/developerworks/jp/java/library/j-pg05199/

```
import groovy.xml.MarkupBuilder

def sw = new StringWriter()
def xml = new MarkupBuilder(sw)

xml.Book(null,{
    Book(type:'book',{
        Author('Hoge')
        Price(1000)
    })
})
println sw


> groovy .\XmlBuilderSample.groovy
<Book>
  <Book type='book'>
    <Author>Hoge</Author>
    <Price>1000</Price>
  </Book>
</Book>
```

Groovy JDK (GDK)