



# Magazine

vol.8

# Contents

Series 10

---

**Groovy 臨機応変 (第三回)**  
～ Groovy 2.3 の新機能～ ..... 2

**Groovy 臨機応変 (第四回)**  
～ Groovy 2.4 の新機能～ ..... 7

Series 16

---

**Gradle Plugin 探訪**  
～第 1 回 Gradle SSH Plugin～ ..... 9

Series 05

---

**Grails Plugin 探訪**  
～第 9 回 CodeNarc plugin～ ..... 14

Information

---

リリース情報..... 19

# Groovy臨機応変 (第三回)

## ～Groovy 2.3の新機能～

series  
10

上原 潤二 (うえはら じゅんじ)

NTTソフトウェア株式会社Grails推進室所属。JGGUG (日本Grails/Groovyユーザ会) 運営委員。  
「Grails徹底入門」(翔泳社)、「プログラミングGROOVY」(技術評論社)執筆メンバーの1人。  
Groovy技術に関するブログ「Grな日々」を主宰している。

今回は、Groovy 2.3の新機能をピックアップして紹介します。  
詳しくはリリースノートを参照ください。

### トレイト (Traits)

Groovy 2.3における目玉機能としてトレイトが導入されました。Groovyのトレイトは、実装の継承が可能なインターフェースです。Java8では、インターフェースにおいて、サブクラスで定義しなかった場合の「デフォルトのメソッド」を定義することができるようになりましたが、Groovyのトレイトも同種の機構であると言えます。しかし、Groovyのトレイトは、メソッド以外にもメンバー変数を定義・使用したり、Proxyを経由した動的な実装をすることができます。またJDK7以前のJava上でも利用することもでき、より強力な機構であると言えます。GroovyのトレイトはScalaのトレイトと似ています。詳しくは以下を参照ください。

(参考リンク)

- <http://beta.groovy-lang.org/docs/groovy-2.3.0/html/documentation/core-traits.html>
- <http://www.slideshare.net/uehaj/groovy-trait>

### 新規AST変換の導入

@TailRecursive, @Builder, @Sortable, @SourceURIの4つのAST変換が導入されました。以下にそれぞれの説明を示します。

#### @TailRecursive

メソッドの自己再帰呼び出しをループに変換します。メソッドを跨った相互末尾再帰等には対応していないなどいくつかの制約があります。

以下の自己末尾再帰呼び出しを含むコードに@TailRecursiveを適用してみます。

```
@TailRecursive
def foo(int n, int lim) {
    if (n < lim) {
        foo(n+1, lim)
    }
    else {
        n
    }
}
```

上は以下のようにループに変換されます。

```
@groovy.transform.TailRecursive
public java.lang.Object foo(int n, int lim) {
    __lim__ = lim
    __n__ = n
    while (true) {
        try {
            if ( __n__ < __lim__ ) {
                java.lang.Integer __n__ = __n__
                java.lang.Integer __lim__ = __lim__
                __n__ = __n__ + 1
                __lim__ = __lim__
                continue
            } else {
                return __n__
            }
        }
        catch (org.codehaus.groovy.transform.tailrec.GotoRecurHereException ignore) {
            continue
        }
        finally {
        }
    }
    return null
}
```

繰り返し数に比例してスタックを消費することがなくなるので、スタックオーバーフローを気にすることなく、適切な場合には再帰呼び出しでわかりやすくロジックを書くことができます。

**(参考リンク)**

- <http://beta.groovy-lang.org/docs/latest/html/gapi/groovy/transform/TailRecursive.html>

**@Builder**

インスタンスを初期化するための、フルーエントなメソッド呼び出しによるビルダパターンを可能とするAST変換です。

使用例を以下に示します。

```
import groovy.transform.builder.Builder

@Builder
class Book {
    int price
    String title
}

def book = Book.builder().price(100).title("我輩は猫である").build()
```

上記以外に、いくつかのパターンでのサポートメソッドをAST変換の引数で指定することもできます。たとえば、setterをチェインさせていくようなパターンのビルダメソッドを生成することもできます。

**(参考リンク)**

- <http://beta.groovy-lang.org/docs/latest/html/gapi/groovy/transform/builder/Builder.html>

**@Sortable**

指定したクラスに対して以下を行います。

- compareToメソッドを生成し、インターフェースComparableを実装する。
- それぞれのフィールドに関して比較するComparatorを返すメソッド(comparatorByXXX())を生成する。
  - →返されたComparatorは、java.util.Collections.sort(List list, Comparator c)や、java.util.Arrays.sort(T[] a, Comparator<? super T> c)に渡してソート処理に適用できる。

コード例を以下に示します。

```
import groovy.transform.Sortable

@Sortable
class Book {
    int price
    String title
}

data = [new Book(price:10, title:"abc"),
        new Book(price:9, title:"def")]

Collections.sort(data)
assert data[0].price == 9 && data[0].title == "def"
assert data[1].price == 10 && data[1].title == "abc"
```

**(参考リンク)**

- <http://beta.groovy-lang.org/docs/latest/html/gapi/groovy/transform/Sortable.html>

**@SourceURI**

StringやURLクラス型の変数定義に指定すると、その変数の値にGroovyスクリプト自身のURIが初期設定されます。

```
import groovy.transform.SourceURI

@SourceURI String src

println src
```

上記のスクリプトを、ファイルに保存して実行するとfile:で始まるURIが表示されます。あるいは、スクリプトをWebサーバに置き、URLを指定してgroovyコマンドを実行すると、そのURLが表示されます。さらに、以下のように-eオプションでスクリプトを指定したり、GroovyShell.evaluate(String)でスクリプトを実行すると、@SourceURIはdata:で始まるデータURIを生成します。

```
$ groovy -e 'import groovy.transform.SourceURI; @SourceURI String src; println src'
data: ,import%20groovy.transform.SourceURI;%20@SourceURI%20String%20src;%20println%20src
```

**(参考リンク)**

- <http://beta.groovy-lang.org/docs/latest/html/gapi/groovy/transform/SourceURI.html>

## ライブラリの拡張・改善

### ■ マークアップテンプレートエンジン

新種のテンプレートエンジン、マークアップテンプレートエンジン (groovy.text.markup.MarkupTemplateEngine) が追加されました。マークアップテンプレートエンジンは、DOM的な構造を記述するためのDSLの一種であるとも言えます。参考リンクにある例を以下に示します。

```
yieldUnescaped '<!DOCTYPE html>'
html(lang:'en') {
  head {
    meta('http-equiv':
'"Content-Type" content="text/html; charset=utf-8"')
    title('My page')
  }
  body {
    p('This is an example of HTML contents')
  }
}
```

これは以下のようなXMLを生成するとのことです。

```
<!DOCTYPE html><html lang='en'><head><meta
http-equiv='\"Content-Type\" content='\"text/
html; charset=utf-8'\"/><title>My page</title></
head><body><p>This is an example of HTML
contents</p></body></html>
```

マークアップテンプレートエンジンは、MarkupBuilderと同様に、クロージャや疑似メソッドの呼び出しを使用してツリー構造を記述・構築し、文字列表現に落とすことができます。加えて、以下の特長があります。

- ・ 静的コンパイルで高速化
- ・ 静的型チェックが可能
- ・ 整形(インデント、エスケープ)機能付き
- ・ テンプレート記述を別ファイルにしてincludeなども可能

(参考リンク)

- ・ <http://beta.groovy-lang.org/docs/groovy-2.3.1/html/documentation/markup-template-engine.html>

### ■ JSON Slurper

Groovy 2.3 ライブラリではいくつかの性能改善がなされており、特に、JSON処理が従来より最大20倍高速化され、Javaベースのライブラリの中で最速の部類となっているとのことです。

JSON Slurperについては、以下のパラメータを指定できるようになりました。

パラメータ	説明
INDEX_OVERLAY	高速。REST呼び出し・WebSocketメッセージ・AJAXなどに適している。シリアライズされたオブジェクトの展開処理を特に高速化している。
LAX	コメントや、クオート無しのマップのキーを許す。
CHAR_BUFFER	int,date,longなどの貪欲(eager)なパース処理が特長。既存のSlurperの動作に近い。
CHARACTER_SOURCE	2MBを越えるJSONファイルの処理に適する。

これらのパラメータは以下のように指定します(参考リンクに示したAPIリファレンスより引用)。

```
parser = new JsonSlurper().setType(
JsonParserType.INDEX_OVERLAY );
```

(参考リンク)

- ・ <http://beta.groovy-lang.org/docs/groovy-2.3.0/html/gapi/groovy/json/JsonParserType.html>

### ■ ConfigSlurperの拡張

Grailsで定義されていて利用可能であるような、prod/dev/testといった「環境」を、新たに定義し指定できるようになりました。

(参考リンク)

- ・ <http://mrhaki.blogspot.jp/2014/05/groovy-goodness-extend-configslurper.html>

## Java8,7の対応

### ■ Java8

実行環境としてJDK 8が公式にサポートされました。

### ■ Java7

Java 7のNIO2に対応しました。例えば、いくつかのGDKメソッドで、java.io.Fileの代わりにjava.io.Pathクラスが使用できます。

以下はリリースノートからの例です。

```
path.withReader { Reader r -> ... }
path.eachLine { String line -> ... }
path.eachFileRecurse { Path p -> ... }
path << 'some content'
path << bytes
path.readLines()
```

## JUnit 4対応

groovy.test.GroovyAssertに以下の静的メソッドが追加されました(GROOVY-6588)。これらは、従来よりGroovyTestCaseのインスタンスメソッドとしては対応するものが定義されていたのですが、JUnit4でテストクラスをGroovyTestCaseから継承させない場合に使用するための、静的メソッドが定義されたクラスgroovy.test.GroovyAssertでは定義されていませんでした。

メソッド	説明
static void    assertScript(String script)	文字列のスクリプトを実行結果をassertする
static boolean   notYetImplemented(Object caller)	試験の成功/失敗を逆転させて扱う (FailならPassとし、PassならFailとする)。
static Throwable   shouldFail(Class clazz, String script)	文字列のスクリプトを実行し指定した例外が発生したらPassとする
static Throwable   shouldFail(String script)	文字列のスクリプトを実行し例外が発生したらPassとする

## Closure引数に対する静的型チェック

従来、クロージャを引数としてとるメソッドにおいて、クロージャ型の引数の引数に型を宣言する方法がありませんでした。

例えば、

```
void foo(Closure clos) {
    // クロージャ closの「引数の型」を宣言する方法がない。
    clos("ABC")
    // 引数 closのクロージャには、常に1つの文字列が引数
    // として与えられるのだが…。
}
```

というメソッドが定義されているとき、以下のコード：

```
@TypeChecked
def func() {
    foo { it -> it.toUpperCase()
        // 静的型エラー。itの型はObjectとみなされる。
    }
}
```

は、クロージャ引数itの型を知る方法が無く、itに対するtoUpperCase()の呼び出しが静的型チェックや静的コンパイルでエラーになるので、以下のように型を指定する必要がありました。

```
foo { String it ->
    it.toUpperCase()
}
```

しかし、fooの宣言時に@ClosureParamを以下のように使用することで、itの型は推論されるようになり、型を明示的に指定する必要がなくなります。

```
void foo(@ClosureParams(value=SimpleType.class,
options="java.lang.String") Closure clos) {
    clos("ABC")
}
```

話はジェネリクスが入ってくるともうちょっとややこしくなります<sup>[1]</sup>。次の例です。

```
["a","b","c"].collect { it.toUpperCase() }
```

この場合、「クロージャの引数の型」は、レシーバのListの「要素の型」と一致していなければなりません。

collectは、GDKメソッドとして、DefaultGroovyMethods.java(DGM)中で、以下のように定義されています。

```
public static <S,T> List<T> collect(Collection<S>
self,
@ClosureParams(FirstParam.FirstGenericType.class)
Closure<T> transform) {
    return (List<T>) collect(self,
        new ArrayList<T>(self.size()), transform);
}
```

「@ClosureParams(FirstParam.FirstGenericType.class) transform」では、クロージャの引数transformの引数の型は、第一引数(FirstParam)すなわちレシーバであるコレクションselfのジェネリクス型引数(S)である、と宣言しています。ちなみにTはクロージャの戻り値の型です。

@ClosureParamsを使用することで、ライブラリ側の定義は煩雑になりますが、それを呼び出すコードにおいて@TypeChecked、@CompileStaticはより賢く振舞うようになり、コードを簡潔にすること、及び動的Groovyコードに適用した際の修正を減らすことに貢献してくれます。

なお、Groovy 2.4 beta 4では、同様の指定が@DelegatesToアノテーションでも可能になりました。(GROOVY-6956)

(参考リンク)

- [http://melix.github.io/blog/2014/01/closure\\\_param\\\_inference.html](http://melix.github.io/blog/2014/01/closure\_param\_inference.html)
- [1] Java8ではGenericsの引数型に対してアノテーションが付与できるようになったので、このような間接的な指定方法ではなく直接指定できるはずである。あるいはGroovy自体の拡張で同種の指定も可能であっただろう。しかしながら、Groovyのコンパイル・動作環境はJava8には限定されておらず、またClosureParamsを使用する主な場所はJavaで定義されたDGM(DefaultGroovyMethods.java)であるため、Java 7以前でも使用できるこの方法が採用された。

## ツールの拡張

### ■ Groovyshの拡張と変更

Groovyshは着々と拡張されています。まず、補完が賢くなりました([GROOVY-6399](#),[GROOVY-6395](#))。たとえば、マップのキーを補完できるようになりました。

```
% groovysh
groovy:000> m = [abc:123, def:456]
groovy:001> m.a [タブを押す]
any( abc
groovy:001> m.ab [タブを押す]
groovy:001> m.abc // 補完される
```

この機能は、DOMツリーやASTなど、ツリーをたどりながら表示させる場合に絶大な効果を発揮します。

また、機能の変更として、groovysh中でのコマンド(load, edit, aliasなど)にはプリフィックス「:」が必要となりました。変数名や関数名と重なることがなくなります。([GROOVY-6397](#))

```
% groovysh
groovy:000> :load test.groovy // 従来はコロン無し
の「load」で実行していた
```

### ■ GroovyConsoleの拡張

GroovyConsoleには以下の拡張がなされています。

- プリファレンスAPIを通じてフォントが指定できるようになったとのことです。ただしフォント指定のためのGUIはまだ用意されていません。([GROOVY-6303](#))
- 選択範囲で実行(Run Selection)したときに、import文を意識してくれるようになりました。つまり、選択範囲にimport文が含まれていなくても、同じソース中の冒頭位置などに含まれているimportがあたかも選択範囲中にも指定されているかのように解釈して実行してくれます。
- コメント・コメント解除の機能とショートカットキーが追加されました。([GROOVY-6459](#))

## ドキュメントの改善

その他、ドキュメントが抜本的に改善されています。改善された内容は、現時点ではGroovyのβ版ドキュメントページで公開されていますが、将来的には公式サイトの内容に置換されます。

([参考リンク](#))

- <http://beta.groovy-lang.org/download.html>

# Groovy臨機応変 (第四回)

## ～Groovy 2.4の新機能～

series  
10

上原 潤二 (うえはら じゅんじ)

NTTソフトウェア株式会社Grails推進室所属。JGGUG (日本Grails/Groovyユーザ会) 運営委員。  
「Grails徹底入門」(翔泳社)、「プログラミングGROOVY」(技術評論社)執筆メンバーの1人。  
Groovy技術に関するブログ「Grな日々」を主宰している。

今回は、Groovy 2.4の新機能を、現時点で公開されているGroovy 2.4 beta4までのリリースノートから抜粋して紹介します。

正式リリースまでにはさらに機能が追加されるでしょうから、正確にはGroovy 2.4の機能の一部ということになります。

(参考リンク)

- <https://jira.codehaus.org/secure/ReleaseNote.jspa?projectId=10242&version=20369>
- <https://jira.codehaus.org/secure/ReleaseNote.jspa?projectId=10242&version=20433>
- <https://jira.codehaus.org/secure/ReleaseNote.jspa?projectId=10242&version=20544>
- <https://jira.codehaus.org/secure/ReleaseNote.jspa?projectId=10242&version=20612>

## Android対応

Groovy 2.4ではAndroidアプリケーションを開発できるようになりました。GroovyをAndroidで動作させるための試みは過去にもいくつかありましたが、今回はパッチなどではなく、公式のGroovyメインラインで対応されているのが特徴です。基本的には、Static Compile配下で動作する機能を使用します。このAndroid対応に合わせてSwissKnife、grooid-toolsなどの開発用ライブラリもできています。(GROOVY-6861)

(参考リンク)

- <https://github.com/Arasthel/SwissKnife>
- <https://github.com/karfunkel/grooid-tools>

## ライブラリの拡張

### toUnique(),toSorted()の追加。

Listなど、Iterableなコレクションに対するGDKメソッドの追加です。(GROOVY-6945)

従来からJDKに存在したList#sort()は破壊的操作でしたが、toSortedはイミュータブルな、「ソートしたものを返す操作」です。また、toUnique()も同様に、重複要素を削除したものを返し、もとのコレクションを変更しません。またこれらはIterableなコレクション一般もしくは配列に対する適用も可能です。

Java8では、コレクションに対するsortメソッドが新規に追加されましたが、GDKのsortと名前が被るので、混乱を避けるためという意味合いもあるようです。

例)

```
assert [4,1,2,2,3,3,4].toUnique() == [4,1,2,3]
assert [4,1,2,2,3,3,4].toSorted() == [1,2,2,3,3,4,4]
```

### init(), dropRight(), takeRight()の追加

Scalaの同名メソッドの導入です。(GROOVY-6867)  
メソッドの説明を以下に示します。

メソッド	意味	例(a = [1,2,3]のとき)
init()	末尾以外。	assert a.init() == [1,2]
dropRight()	末尾の指定個数を削除	assert a.dropRight(2) == [1]
takeRight()	末尾の指定個数を取得	assert a.takeRight(2) == [2,3]

### System.currentTimeMillis()の追加

エポックからの経過「秒数」を返すメソッドSystem.currentTimeMillis()が追加されました。JDKのSystem.currentTimeMillisの1000分の1を返します。(GROOVY-6294)

```
groovy:000> System.currentTimeSeconds()
====> 1416000159
groovy:000> System.currentTimeMillis()
====> 1416000167901
```

## Groovyshの拡張

groovyshが地道に拡張されています。

- groovyshではメソッド補完が効くようになっているのですが、補完候補の表示にANSIのエスケープシーケンスを使って、インスタンスメソッドはボールド表示、staticメソッドやsuperクラスのメソッドは通常表示となるようになりました。(GROOVY-6563)
- 以前のGroovyshでは、行の単位でGroovyの式を評価するため、シェル変数以外の型宣言した変数を行をまたがって使用することができませんでしたが、可能となります。(GROOVY-6623)



(例) 以下が可能となる。

```
String s = "abc"
println s
```

- groovyshの起動時に、-eオプションで任意のGroovyコードを実行できるようになりました。また、コマンドラインに指定したgroovyスクリプトを読み込むようになりました。

(例)

```
> cat test.groovy
println "hello"
> groovysh -e 'println 1+1' test.groovy
Groovy Shell (2.4.0-beta-3, JVM: 1.8.0_25)
Type ':help' or ':h' for help.
-----
-----
groovy:000> println 1+1
2
==> null
groovy:000> :load test.groovy
hello
==> null
groovy:000>
```

## SelfType アノテーション

トレイトの定義において、そのトレイトを実装するクラスが実装していなければならないクラスもしくはインターフェースを@SelfTypeというアノテーションで指定できるようになりました。SelfTypeアノテーションによって、以下の利点が得られるようになります。

- トレイトが、どんなクラス・インターフェース(およびそのサブクラス)に注入可能であるかを制約する。この制約を満していないクラスがそのトレイトを実装しようとするとコンパイルエラーとなる。
- その結果、トレイトで定義するメソッド中で、使用できるはずのメソッド群が静的に予期できるようになり、実装クラスのメソッドを自由に呼び出せる。
- トレイトのメソッド中でのthisの型が静的に確定するので、thisを他のメソッドに渡す際などに静的型チェック可能になる。

要は、トレイトと静的Groovy(@CompileSattic, @TypeChecked)との相性がより高まったということです。

コード例は以下のとおり。

```
import groovy.transform.*
import java.util.concurrent.CopyOnWriteArrayList

@CompileStatic
@SelfType(List)
trait LisLikeList {
    Object car() {
        first()
    }
    Collection cdr() {
        tail()
    }
}

class LisLikeCoWList<E> extends
CopyOnWriteArrayList<E> implements LisLikeList {
    LisLikeCoWList(list) {
        super(list)
    }
}

def list1 = new LisLikeCoWList([1,2,3])
assert list1.car() == 1
assert list1.cdr() == [2,3]
```

(参考リンク)

- [http://docs.groovy-lang.org/2.4.0-beta-4/html/documentation/core-traits.html#\\_self\\_types](http://docs.groovy-lang.org/2.4.0-beta-4/html/documentation/core-traits.html#_self_types)

## Macro Groovy

その他、AST変換におけるASTの生成を簡易に書けるようにする「Macro Groovy」機能の、Groovy公式機能としての導入が検討されています。この過程で、ASTの木のパターンマッチングサポートも検討されているようです。

(参考リンク)

- <https://github.com/bsideup/MacroGroovy>
- <http://groovy.329449.n5.nabble.com/State-of-the-macrostd5721429.html#a5721475>

Groovy 2.4正式版は本記事執筆後に公開されました。冒頭にお断りしているように、Groovy 2.4正式版では本記事で記載した以外にも多数の機能が追加されており、また、Macro Groovyの採用は2.4.0時点では見送られています。悪しからずご了承ください。

(参考リンク)

<http://docs.codehaus.org/display/GROOVY/Groovy+2.4+release+notes>

# Gradleプラグイン探訪 ～第1回 Gradle SSH Plugin～

series  
16

須江 信洋 (すえのぶひろ)

日本 Grails/Groovy ユーザーグループ サポートスタッフ。『Groovy イン・アクション』（毎日コミュニケーションズ）翻訳チーム、および『プログラミング GROOVY』（技術評論社）「Gradle 徹底入門（翔泳社）」執筆チームの一人。本稿は著者個人の考えおよび経験に基づいて記述したものであり、所属する会社や組織の意見を表すものではありません

Gradle SSH Pluginは、GradleのビルドスクリプトからSSHによるリモート操作や、SFTPによるファイル転送を行うためのプラグインです。ビルドプロセスにおけるファイル転送やサービス起動・停止など、さまざまな操作を自動化することができます。

操作対象のサーバーにはSSHでアクセス可能であればよく、特別なエージェントやデーモンなどを導入する必要がないため、導入の敷居が低いことも特徴です。加えて、パスワード認証や公開鍵認証、SSHエージェント、SOCKS プロキシ対応など、さまざまな利用パターンに対応しています。

また、Gradle SSH PluginのSSH接続機能だけを単体で切り出した Groovy SSH というライブラリも提供されており、Gradleとは無関係に単体のGroovyスクリプトから利用することも可能です。

## Gradle SSH Plugin 概要

開発者: @int128 (いわてい)氏

最新バージョン: 0.4.5 (2014/11/24時点)

プロジェクトWebサイト:

- Gradle SSH Plugin [<https://gradle-ssh-plugin.github.io/>]
- Groovy SSH [<https://groovy-ssh.github.io/>]

ライセンス: Apache License Version 2.0

本プラグインのバージョン0.4.xはGradle2.0以降が必要です。Gradle1.xから利用する場合はバージョン0.3.xを利用してください。

## Gradle SSH Plugin を使うメリット

Javaで実装されたSSHライブラリはいくつかありますが、標準入出力の処理など低レベルの処理を自分で実装しなければなりません。

Gradle SSH Pluginは独自のDSL(以降「SSH DSL」と表記します)を提供することでこのような煩雑さをなくし、かつ処理対象のグループ化などを見通しよく実装できるように工夫されています。(内部ではJSchを利用しています。)

SSH DSLの文法はGroovy SSHと共通なので、Gradleから利用するだけでなく、高機能なシェルスクリプトのように利用することも可能です。

## クイックスタート

開発者の@int128氏が[テンプレートプロジェクト](#)を提供してくれていますので、参考にするとよいでしょう。

本稿では、テンプレートプロジェクトに少し手を入れてVagrantでSSH可能な仮想マシンを準備するサンプルを用意しましたので、そちらを利用してGradle SSH Pluginの動作を確認してみましょう。以下のコマンドでサンプルを入手してください。

```
git clone -b gmag-8 https://github.com/nobusue/gradle-ssh-plugin-template.git
```

サンプルコード内のbuild.gradleは以下のようになっています。

```

plugins {
    id 'org.hidetake.ssh' version '0.4.5'
}

remotes {
    targethost {
        host = '192.168.33.10'
        user = 'vagrant'
        identity = file("${System.properties['user.home']}/.vagrant.d/insecure_private_key")
    }
}

ssh.settings {
    logging = 'stdout'
}

task showPlatformVersion << {
    ssh.run {
        session(remotes.targethost) {
            execute('uname -a')
            execute('cat /etc/*-release || true')
        }
    }
}

task wrapper(type: Wrapper) {
    gradleVersion = '2.1'
}

```

### remotes ブロック

ここではSSH接続先の情報を定義します。

サンプルではVagrantで起動したサーバー(192.168.33.10)に対してSSH鍵認証でログインする設定をしています。

複数のサーバーに対する定義を列挙することも可能です。

### ssh.settings ブロック

ここではSSH Plugin 全体の設定を行います。

サンプルではログ出力先を標準出力(および標準エラー出力)に設定しています。

### ssh.run ブロック

リモートサーバーに対する操作を定義します。

サンプルでは、remotes.targethostで定義したサーバーにSSH接続し、execute()でunameコマンドおよびcatコマンドを実行しています。

### サンプル実行

以下のコマンドでサンプルを実行します。(Vagrantを利用しますので、別途導入しておいてください。Vagrantを利用しない場

合は、remotesブロックの内容を適宜修正してください。)

```

vagrant up
./gradlew showPlatformVersion

```

実行結果は以下のようになります。

```

$ ./gradlew showPlatformVersion
:showPlatformVersion
Linux precise32 3.2.0-23-generic-pae #36-Ubuntu
SMP Tue Apr 10 22:19:09 UTC 2012 i686 i686 i386
GNU/Linux
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=12.04
DISTRIB_CODENAME=precise
DISTRIB_DESCRIPTION="Ubuntu 12.04 LTS"

BUILD SUCCESSFUL

Total time: 6.248 secs

```

このように、簡単にシェルコマンドを実行することができるので、複数のサーバーに対して同じシェルスクリプトを実行する場合などに威力を発揮します。

### シェルコマンドとGroovyの連携

Gradle SSH Pluginの便利なところは、単にシェルコマンドを実行できるだけでなく、実行結果をGroovyの文字列として受け取って処理することができる点にあります。

前述のサンプルのssh.runブロックを以下のように修正して実行してみてください。

```

ssh.run {
    session(remotes.targethost) {
        def uname = execute('uname -a')
        assert uname.contains('precise64')
        execute('cat /etc/*-release || true')
    }
}

```

実行結果は以下のようになります。

```

:showPlatformVersion FAILED

FAILURE: Build failed with an exception.

* Where:
Build file '/Users/nobusue/work/gmag/gradle-ssh-plugin-template/build.gradle' line: 21

* What went wrong:
Execution failed for task ':showPlatformVersion'.
> assert uname.contains('precise64')
   |   |
   |   false
   Linux precise32 3.2.0-23-generic-pae
#36-Ubuntu SMP Tue Apr 10 22:19:09 UTC 2012 i686
i686 i386 GNU/Linux

```

'uname -a'の結果に'precise64'が含まれていないため、assertが失敗して例外が発生しています。

これは非常に単純な例ですが、例えばシェルコマンドの実行結果をGroovyで解析して、その結果を次のシェルコマンドの入力として渡すことも可能です。うまく使いこなすと、非常に高度な処理を自動化することができます。

## Gradle SSH Pluginの機能

Gradle SSH Pluginは多くの機能を提供しているため、ここでは特に利用頻度が高いと思われるものについてまとめます。

詳細については[ユーザーガイド](#)を参照するとよいでしょう。

また、SSH DSLの利用方法については[acceptance-test](#)のコードも参考になります。

## リモートホストおよび接続定義

remotesブロックで定義します。remotes.<id>でRemote型のオブジェクトとして参照可能です。

以下のパラメータを指定できます。

キー	型	説明
host	String(必須)	ホスト名 or IPアドレス
port	Integer	ポート番号(デフォルト 22)
gateway	Remote	SSHポートフォワードを行う
proxy	Proxy	SOCKS/HTTPプロキシを指定
user	String(必須)	ユーザー名
password	String	パスワード
identity	File	秘密鍵ファイル
passphrase	String	秘密鍵のパスフレーズ
agent	Boolean	Putty or ssh-agentを利用
knownHosts	File	known hosts ファイル(デフォルト ~/.ssh/known_hosts) allowAnyHosts設定時はチェックを無効化
retryCount	Integer	接続リトライ回数(デフォルトはリトライなし)
retryWaitSec	Integer	リトライ間隔(秒数、デフォルト 0)

## プロキシ定義

proxiesブロックで定義します。proxies.<id>でProxy型のオブジェクトとして参照可能です。SOCKSおよびHTTPに対応しています。

以下のパラメータを指定できます。

キー	型	説明
host	String(必須)	ホスト名 or IPアドレス
port	Integer(必須)	ポート番号
type	ProxyType(必須)	プロキシのタイプ(SOCKS or HTTP)
user	String	ユーザー名
password	String	パスワード
socksVersion	Integer	SOCKSバージョン(4 or 5, デフォルト 5)

## ロール定義

role()でリモートサーバーを役割ごとにグループ化できます。各Remoteに対してロールは複数指定可能です。

```
ssh.remotes {
  server1 {
    role('web')
    role('all')
    host = 'x.x.x.x'
    user = 'root'
  }
  server2 {
    role('web')
    role('all')
    host = 'y.y.y.y'
    user = 'root'
  }
  server2 {
    role('db')
    role('all')
    host = 'z.z.z.z'
    user = 'root'
  }
}
```

ロールを定義しておく、remotes.role()でリモートサーバーをまとめて指定できます。

```
ssh.run {
  session(remotes.role('web')) {
    // operations for web
  }
  session(remotes.role('web','db')) {
    // operations for web and db
  }
  session(remotes.role('all')) {
    // operations for all
  }
}
```

## オペレーション実行

ssh.runブロックでSSH接続およびオペレーションの実行を定義します。

SSH接続ごとにsessionブロックを定義し、内部でオペレーションを記述します。

```
task sshTask << {
  ssh.run {
    session(remotes.server1) {
      // do operations for server1
    }
    session(remotes.server2) {
      // do operations for server2
    }
  }
}
```

複数のサーバーに同じオペレーションを行う場合は、sessionの引数にRemoteオブジェクトの配列を渡します。

```
task sshTask << {
  ssh.run {
    session( [remotes.server1, remotes.server2])
    {
      // do common operations
    }
  }
}
```

この場合、オペレーションはserver1とserver2に対して並列実行されるわけではなく、指定した順にシーケンシャルに行われることに注意してください。

オペレーションとして利用可能なものは以下の通りです。

オペレーション	動作
execute	コマンドを実行します
executeBackground	バックグラウンドでコマンドを実行します
executeSudo	'sudo -S -p'を指定してコマンドを実行します
shell	シェルによる対話処理を実行します
put	リモートサーバーにファイル/ディレクトリを送信します
get	リモートサーバーからファイル/ディレクトリを取得します

execute()によるコマンド実行は、コマンド終了までブロックされず。また、コマンドが0以外のリターンコードを返した場合には例外が発生します。

コマンドの実行結果は変数もしくはクロージャで受け取ることができます。具体的には以下のようにします。

```
def hostname = execute 'hostname -f'
println hostname

execute('hostname -f') { result -> println result
}
```

また、オペレーション実行時には以下のパラメータを指定できます。

キー	型	説明
dryRun	Boolean	ドライランを有効にします(デフォルト false)
pty	Boolean	PTY(擬似端末)を有効にします(デフォルト false)
logging	String	ログ出力先を指定します(デフォルト slf4j)
outputStream	OutputStream	コマンドの標準出力のリダイレクト先を指定します
errorStream	OutputStream	コマンドの標準エラー出力のリダイレクト先を指定します
encoding	String	入出力エンコーディングを指定します(デフォルト UTF-8)
interaction	Closure	対話処理のためのクロージャを指定します
extensions	List of classes	ミックスインする拡張モジュールのクラスを指定します

## 対話処理

オペレーションの実行結果に対してパターンマッチングを行うことで、対話的な処理を実現できます。expectコマンドと類似の処理が可能です。

詳細については[ユーザーガイド](#)を参照してください。

## Groovy SSH ライブラリ利用時の注意事項

Groovy SSH ライブラリは Gradle に依存しておらず、単体で利用することができます。

使い方は簡単で、Groovy SSH の JAR ファイルをダウンロードし、java -jar で実行するだけです。

```
$ curl -L -O https://github.com/int128/groovy-ssh/releases/download/v0.1.7/groovy-ssh-0.1.7-all.jar
$ java -jar groovy-ssh-0.1.7-all.jar sshTasks.groovy
```

Groovy SSH で SSH DSL を実行する場合にはコンテキストの指定が必要です。以下のように "ssh." を記述するようにしてください。

```
ssh.remotes {
  server {
    host = 'x.x.x.x'
    user = 'someone'
    identity = new File('~/.ssh/id_dsa')
  }
}

ssh.settings {
  logging = 'stdout'
}

ssh.run {
  session(ssh.remotes.server1) {
    execute 'hostname -f'
  }
}
```

## まとめ

Gradle SSH Plugin/Groovy SSH ライブラリは Gradle による作業の自動化だけでなく、シェルスクリプトの高機能な代替物として Groovy を利用可能にする、可能性を秘めたプロダクトです。

ドキュメントが英語のみということもあってか、なぜか日本よりも海外で人気があるようですが、この機会にぜひ試してみてください。

# Grails Plugin 探訪

## 第9回

### ～ CodeNarc プラグイン～

URL: <http://grails.org/plugin/codenarc>

プラグインのバージョン: 0.22

対応するGrailsのバージョン: 1.3以上



杉浦孝博

最近では Grails を使用したシステムの保守をしている自称プログラマー。

日本 Grails/Groovy ユーザーグループ事務局長。

共著『Grails 徹底入門』、共訳『Groovy イン・アクション』



## はじめに

今回で紹介するGrailsプラグインは、CodeNarcプラグインです。

本記事は、次の環境で動作確認をしております。

- OS: Windows 7 SP1, Mac OS X 10.9.5
- Java: 1.8.0\_25
- Grails: 2.4.4

なお、コマンドの実行結果については、紙面の都合上、出力結果を省略しており、実際の出力と異なる場合があります。ご了承ください。

## CodeNarc プラグインとは

CodeNarc プラグインは、Groovyのソースコードに対して静的コード分析を行うための機能を提供するプラグインです。

既存のツールであるCodeNarcを使用し静的コード分析を行います。

なお、CodeNarcについては、Vol.1からVol.3まで連載がありますので、そちらも参照してください。

## プラグインのインストール

CodeNarc プラグインのインストールは、Grailsのバージョンが1.xの場合、次のコマンドを入力してインストールします。

```
$ grails install-plugin codenarc
```

Grailsのバージョンが2.xの場合、BuildConfig.groovyのpluginsに「compile 'codenarc:0.22'」を追加します。

```
plugins {
    ....
    compile 'codenarc:0.22'
}
```

## 静的コード分析の実行

静的コード分析は、次のコマンドを実行します。

```
$ grails codenarc
```

デフォルトでは、分析結果はtarget/CodeNarcReport.htmlというレポートファイルに出力されます。

## サンプルコードで静的コード分析

実際に試してみましょう。

### サンプルプロジェクトの作成

サンプルのプロジェクトを作成します。

```
$ grails create-app testcodenarc
```

### サンプルのドメインクラスの作成

次にサンプルのドメインクラスを作成します。今回もいつもの様に、本(Book)のドメインクラスとします。

```
$ cd testcodenarc
$ grails create-domain-class Book
```

Bookドメインクラスのコードは次のとおりです。

```
package testcodenarc

class Book {
    String title
    int price
    String isbn13

    static constraints = {
        title blank: false
        price min: 0
        isbn13 blank: true
    }
}
```

### コントローラ、ビューの生成

ドメインクラスから、コントローラとビューを自動生成します。

```
$ grails generate-all testcodenarc.Book
```

### 静的コード分析の実行

自動生成したコードも含め、静的コード分析を実行します。

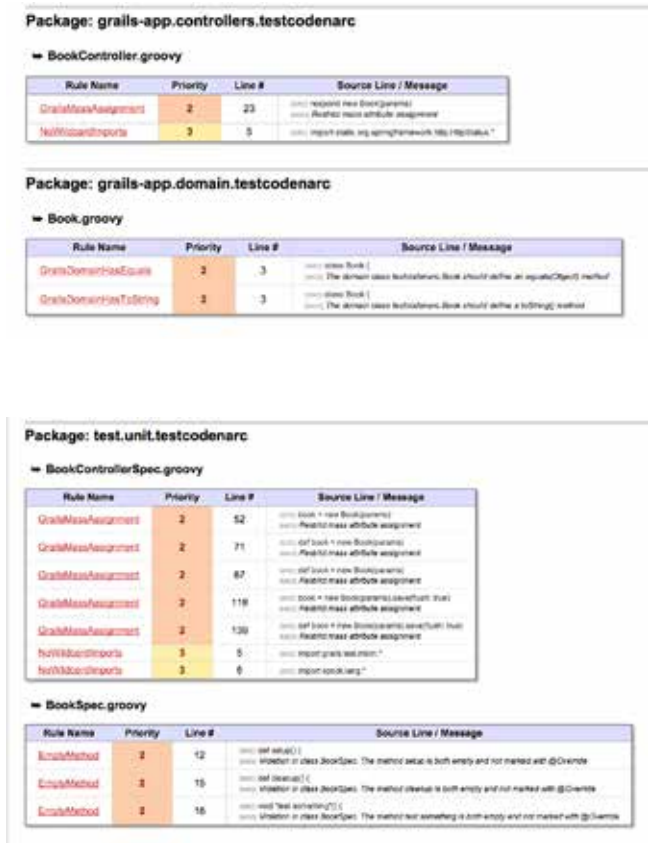
```
$ grails codenarc
...
..Running CodeNarc (rulesets/basic.xml,rulesets/
exceptions.xml,
rulesets/imports.xml,rulesets/grails.
xml,rulesets/unused.xml) ...
.CodeNarc finished; report(s) generated: [target/
CodeNarcReport.html]
```

### 静的コード分析結果の参照

出力されたファイルtarget/CodeNarcReport.htmlをWebブラウザで開きます。



Package	Total Files	Files with Violations	Priority 1	Priority 2	Priority 3
All Packages	4	4	-	11	3
grails-app.controllers.testcodenarc	1	1	-	1	1
grails-app.domain.testcodenarc	1	1	-	2	-
testunit.testcodenarc	2	2	-	8	2



Package: grails-app.controllers.testcodenarc

Rule Name	Priority	Line #	Source Line / Message
GrailsMassAssignment	2	23	...: Requested new Book() parameter ...: Requested mass attribute assignment
NoHttpCodeReports	3	5	...: Requested state, org.springframework.http.HttpStatus

Package: grails-app.domain.testcodenarc

Rule Name	Priority	Line #	Source Line / Message
GrailsDomainClassEquals	2	3	...: class Book { ...: The domain class testcodenarc.Book should define an equals() method
GrailsDomainClassToString	2	3	...: class Book { ...: The domain class testcodenarc.Book should define a toString() method

Package: test.unit.testcodenarc

Rule Name	Priority	Line #	Source Line / Message
GrailsMassAssignment	2	52	...: class Book { new Book() { ...: Requested mass attribute assignment
GrailsMassAssignment	2	71	...: def book = new Book() { ...: Requested mass attribute assignment
GrailsMassAssignment	2	87	...: def book = new Book() { ...: Requested mass attribute assignment
GrailsMassAssignment	2	118	...: class Book { new Book() { ...: Requested mass attribute assignment
GrailsMassAssignment	2	130	...: def book = new Book() { ...: Requested mass attribute assignment
NoHttpCodeReports	3	5	...: Requested state, HttpStatus
NoHttpCodeReports	3	6	...: Requested state, HttpStatus

Rule Name	Priority	Line #	Source Line / Message
EmptyMethod	2	12	...: def empty() { ...: Method in class BookSpec: The method body is both empty and not marked with @Override
EmptyMethod	2	18	...: def isEmpty() { ...: Method in class BookSpec: The method body is both empty and not marked with @Override
EmptyMethod	2	18	...: def test something() { ...: Method in class BookSpec: The method body something is both empty and not marked with @Override

ご覧のとおり、自作したドメインクラス、自動生成されたコントローラももちろんですが、テストコードもGroovyコードですので、分析の対象となっているのがわかります。



## CodeNarc プラグインの設定

CodeNarc プラグインは、デフォルトの設定でも十分使えますが、場合によっては設定を変更したい場合があると思います。そのような場合は、BuildConfig.groovyに「codenarc.」で始まるプロパティに設定を記述します。

変更できる内容は次のとおりです。

- レポートの形式やファイルパスなど
- CodeNarcルールセット
- 分析対象のソースコード
- CodeNarcのルールのプロパティ
- ルール違反数の上限

### レポートの設定

レポートは、デフォルトではHTML形式でtarget/CodeNarcReport.htmlファイルに出力されます。

分析結果のレポートについては、形式やファイルのパス、タイトルを変更することができます。

レポート形式については、HTML形式以外に、XML形式とテキスト形式もサポートしています。

レポートは、1つだけでなく、複数指定することができます。

設定の仕方は、

```
codenarc.reports = {
  レポート名(レポート形式) {
    outputFile = レポートのファイルパス
    title = レポートのタイトル
  }
  ...
}
```

と、クロージャで設定します。レポート名は任意の名前、レポート形式は次の表のとおり、ファイルパスは相対パスの場合、アプリケーションのルートディレクトリからのパスとなります。

形式	値
HTML	'html'
XML	'xml'
テキスト	'text'
任意の出力	CodeNarcのReportWriterインタフェースを実装したクラスの完全修飾クラス名

例を見てみましょう。次のとおり設定を記述すれば、targetディレクトリに3つのレポートファイルが作成されることとなります。

```
codenarc.reports = {
  MyXmlReport('xml') {
    outputFile = 'target/CodeNarc-Report.xml'
    title = 'Sample Report by XML'
  }
  MyHtmlReport('html') {
    outputFile = 'target/CodeNarc-Report.html'
    title = 'Sample Report by HTML'
  }
  MyTextReport('text') {
    outputFile = 'target/CodeNarc-Report.text'
    title = 'Sample Report by Text'
  }
}
```

### CodeNarcルールセットファイルの設定

CodeNarcのルールセットファイルは、デフォルトでは次のファイルが使用されます。

- rulesets/basic.xml
- rulesets/exceptions.xml
- rulesets/imports.xml
- rulesets/grails.xml
- rulesets/unused.xml

CodeNarcのルールセットファイルの設定を変更したい場合、BuildConfig.groovyに「codenarc.ruleSetFiles」プロパティを記述します。

codenarc.ruleSetFilesプロパティに指定するファイルは、クラスパスから参照できる必要があります。ファイルシステム上のファイルをし指定した場合、次のように「file:」を先頭に付加します。

```
codenarc.ruleSetFiles="file:grails-app/conf/MyRuleSet.groovy"
```

codenarc.ruleSetFilesプロパティに複数ファイルを指定したい場合、カンマ(,)で区切って1つの文字列で指定するか、リスト形式で複数の文字列で指定します。

```
codenarc.ruleSetFiles="rulesets/basic.xml,rulesets/exceptions.xml," +
    rulesets/imports.xml,rulesets/grails.xml,rulesets/unused.xml"
```

```
codenarc.ruleSetFiles=[
    "rulesets/basic.xml",
    "rulesets/exceptions.xml",
    "rulesets/imports.xml",
    "rulesets/grails.xml",
    "rulesets/unused.xml"
]
```

## 分析対象のソースコード

分析対象のソースコードは、デフォルトでは、分析対象となる Groovy コードは次のディレクトリ配下の Groovy ファイルが対象となります。

- src/groovy
- grails-app/controllers
- grails-app/domain
- grails-app/services
- grails-app/taglib
- grails-app/utils
- test/unit
- test/integration

分析対象のソースコードを変更したい場合、次の表に示すプロパティを BuildConfig に記述します。プロパティの値として true を指定した場合は指定したディレクトリ配下の Groovy ファイルが分析対象となり、false を指定した場合は分析対象とはなりません。

プロパティ名	対象ディレクトリ	デフォルト値
codenarc.processSrcGroovy	src/groovy	true
codenarc.processControllers	grails-app/controllers	true
codenarc.processDomain	grails-app/domain	true
codenarc.processServices	grails-app/services	true
codenarc.processTaglib	grails-app/taglib	true
codenarc.processTestUnit	test/unit	true
codenarc.processTestIntegration	test/integration	true
codenarc.processViews	grails-app/views	false
codenarc.extraIncludeDirs	任意のディレクトリ	—

codenarc.extraIncludeDirs プロパティで任意のディレクトリを対象にした場合、リスト形式で指定します。例えば、grails-app/jobs ディレクトリを指定したい場合、次のように指定します。

```
codenarc.extraIncludeDirs=['grails-app/jobs']
```

## CodeNarcのルールのプロパティ

CodeNarcの各ルールはいくつかのプロパティを持っており、BuildConfig.groovy中に codenarc.properties プロパティを記述して、ルールのプロパティ値を変更することができます。

形式は次のとおりです。

```
ルール名.プロパティ名 = プロパティ値
```

例えば、ドメインクラスが equals メソッド、toString メソッドを持つことを示すルール、GrailsDomainHasEquals ルールと GrailsDomainHasToString ルールを無効にしたい場合は、次のように指定します。

```
codenarc.properties = {
    GrailsDomainHasEquals.enabled = false
    GrailsDomainHasToString.enabled = false
}
```

また、CodeNarcのルールのプロパティを別ファイルですでに持っている場合、そのファイルパスを BuildConfig.groovy 中の codenarc.propertiesFile プロパティに記述することで同等のことを実現できます。

先程の例は、次と同等です。

```
// grails-app/conf/BuildConfig.groovy
codenarc.propertiesFile = 'grails-app/conf/codenarc.properties'

// grails-app/conf/codenarc.properties
GrailsDomainHasEquals.enabled = false
GrailsDomainHasToString.enabled = false
```

## ルール違反数の上限

CodeNarcのルールにはプライオリティが設定されており、プライオリティ毎に違反数の上限を設定することができます。デフォルトは、Integer.MAX\_VALUE です。上限を越えた場合、次のメッセージが標準出力に出力され、静的コード分析が失敗となります。ただし、静的コード分析は最後まで実行され、レポートファイルも出力されるようです。

```
FAILED -- Exceeded maximum number of priority 2
violations: (p1=0; p2=11; p3=3)
```

違反数の上限を設定する場合、BuildConfig.groovy に次のように設定します。

```
// プライオリティ1の違反数の上限
codenarc.maxPriority1Violations=10
// プライオリティ2の違反数の上限
codenarc.maxPriority2Violations=20
// プライオリティ3の違反数の上限
codenarc.maxPriority3Violations=30
```

### ルール違反数の上限超過時の動作

ルール違反数の上限を超過した場合、デフォルトでは、先程の失敗メッセージを標準出力に出力し、System.exit(1)を呼び出して終了します。この動作を変更し、例外をスローして終了することもできます。

BuildConfig.groovyでcodenarc.systemExitOnBuildExceptionプロパティを指定することで、動作を変更することができます。プロパティ値としてtrueを指定した場合、デフォルトの動作と同様、失敗メッセージを標準出力に出力し、System.exit(1)を呼び出して終了します。プロパティ値としてfalseを指定した場合、例外をスローして終了します。

```
codenarc.systemExitOnBuildException = false
```

## おわりに

今回は静的コード分析を行うプラグインをご紹介いたしました。Groovyコードの静的コード分析ツールとして定評のあるCodeNarcを使用していますので、一度お試しいしてはいかがでしょうか。

## リリース情報 2014.11.29

### Grails

Grailsは、GroovyやHibernateなどをベースとしたフルスタックのWebアプリケーションフレームワークです。

URL: <http://grails.org/>

バージョン: 1.3.9, 2.1.5, 2.2.5, 2.3.11, 2.4.4

#### ■更新情報

- 2.2.5では、ユーザガイドの"withFormat"セクションに"form"mime-typeについての記述が追加されたり、いくつかバグ対応が行われています。
- 2.2.5リリースノート: <https://grails.org/2.2.5+Release+Notes>
- 山本さんのブログ: <http://d.hatena.ne.jp/mottsnite/20140304/1393946239>
- 2.3.11では、いくつかバグ対応が行われています。
- 2.3.11リリースノート: <https://grails.org/2.3.11+Release+Notes>
- 山本さんのブログ: <http://d.hatena.ne.jp/mottsnite/20140626/1403777424>
- 2.4.4では、@DirtiesRuntimeアノテーションが追加されたり、ユニットテストでのforwardedUrlの値が変更になったり、いくつかバグ対応が行われています。
- 2.4.4リリースノート: <https://grails.org/2.4.4+Release+Notes>
- 山本さんのブログ: <http://d.hatena.ne.jp/mottsnite/20141028/1414514707>

### Groovy

Groovyは、JavaVM上で動作する動的言語です。

URL: <http://groovy.codehaus.org/>

バージョン: 1.8.9, 2.0.8, 2.1.9, 2.2.2, 2.3.8, 2.4.0-beta-4

#### ■更新情報

- 2.2.2では、@Delegateを使用した際スタックトレースに行番号が出力されるようになったり、いくつかバグ対応が行われています。
- 2.2.2リリースノート: <https://jira.codehaus.org/secure/ReleaseNote.jspa?projectId=10242&version=19832>
- 2.3.8では、BigInteger.power(BigInteger)メソッドが追加されたり、いくつかバグ対応が行われています。
- 2.3.8リリースノート: <http://jira.codehaus.org/secure/ReleaseNote.jspa?projectId=10242&version=20648>
- 2.4.0-beta-4では、JSONビルダーの実装が書きなおされたり、trait用に@SelfTypeアノテーションが追加されたり、いくつかバグ対応が行われています。
- 2.4.0-beta-4リリースノート: <http://jira.codehaus.org/secure/ReleaseNote.jspa?projectId=10242&version=20612>

### Griffon

Griffonは、デスクトップアプリケーションを開発するためのアプリケーションフレームワークです。

URL: 1.x <http://griffon.codehaus.org/>, 2.x [<http://new.griffon-framework.org/index.html>]

バージョン: 1.5.0, 2.0.0

#### ■更新情報

- 1.5.0では、Groovyのバージョンが2.2.1に、Springのライブラリのバージョンが3.2.7にそれぞれ変更されています。
- 1.5.0リリースノート: <http://docs.codehaus.org/display/GRIFTON/Griffon+1.5.0>
- 2.0.0では、依存するプラグインが最新のバージョンにアップグレードされています。
- 2.0.0リリースノート: [http://new.griffon-framework.org/news/griffon\\_2.0.0.html](http://new.griffon-framework.org/news/griffon_2.0.0.html)

### Gant

Gantは、XMLの代わりにGroovyでAntタスクを記述し実行するビルド管理ツールです。

URL: <http://gant.codehaus.org/>

バージョン: 1.9.11

#### ■更新情報

- 1.9.11での更新内容は不明です。

### GMaven

GMavenは、Maven用のGroovyプラグインです。

URL: 1.x <http://gmaven.codehaus.org/>, 2.x [<http://groovy.github.io/gmaven/>]

バージョン: 1.4, 2.0

### Gradle

Gradleは、Groovyでビルドスクリプトを記述し実行するビルド管理ツールです。

URL: <http://www.gradle.org/>

バージョン: 2.2.1

#### ■更新情報

- 2.2.1では、OS XとJava 7の組み合わせでオフライン時にGradleデーモンが起動しない問題に対応しました。
- 2.2.1リリースノート: <http://www.gradle.org/docs/2.2.1/release-notes>

### Gaelyk

Gaelykは、Groovyで記述するGoogle App Engine for Java用のライトウェイトなフレームワークです。

URL: <http://gaelyk.appspot.com/>

バージョン: 2.1.2

#### ■更新情報

- 2.1.2では、Groovy 2.3に対応し、いくつかバグ対応が行われています。
- 2.1.2リリースノート: <http://gaelyk.appspot.com/download>

### Google App Engine SDK for Java

Google App Engine SDK for Javaは、JavaでGoogle App Engine用のWebアプリケーションを開発するためのSDKです。

URL: <https://cloud.google.com/appengine/>

バージョン: 1.9.15

#### ■更新情報

- 1.9.15では、Datastoreの統計情報にエンティティのカウンタが表示されないバグ対応が行われています。
- 1.9.15リリースノート: <https://code.google.com/p/googleappengine/wiki/SdkForJavaReleaseNotes>

### GPars

GParsは、Groovyに直感的で安全な並行処理を提供するシステムです。

URL: <http://gpars.codehaus.org/>

バージョン: 1.2.1GA

#### ■更新情報

- 1.2.1GAでの更新内容は不明です。

### Groovy++

Groovy++は、Groovy言語に対して静的な機能を拡張します。

URL: <http://code.google.com/p/groovypptest/>

バージョン: 0.9.0

## Spock

Spockは、JavaやGroovy用のテストと仕様のためのフレームワークです。

URL: <http://code.google.com/p/spock/>

バージョン: 0.7

## GroovyServ

GroovyServは、Groovy処理系をサーバとして動作させることでgroovyコマンドの起動を見た目上高速化するものです。

URL: <http://kobo.github.com/groovyserv/>

バージョン: 1.0.0

### ■更新情報

- 1.0.0では、ログファイルやauthtokenファイルのパスをGROOVYSERV\_WORK\_DIRやGROOVYSERV\_LOG\_DIR環境変数で明示することができたり、パッケージ名が変更になったり、バグ対応が行われています。
- 1.0.0チェンジログ: <http://kobo.github.io/groovyserv/changelog.html>

## Geb

Gebは、Groovyを使用したWebブラウザを自動化する仕組みです。

URL: <http://www.gebish.org/>

バージョン: 0.10.0

### ■更新情報

- 0.10.0では、要素のCSSプロパティにアクセスするためにNavigatorにcss()が追加されたり、いくつかバグ対応が行われています。
- 0.10.0リリースノート: <http://www.gebish.org/manual/current/project.html#0100>

## Easyb

Easybは、ビヘイビア駆動開発(Behavior Driven Development:BDD)用のフレームワークです。

URL: <http://www.easyb.org/>

バージョン: 1.5

## Gmock

Gmockは、Groovy用のモック・フレームワークです。

URL: <http://code.google.com/p/gmock/>

バージョン: 0.8.3

## HTTPBuilder

HTTPBuilderは、HTTPベースのリソースに簡単にアクセスするための方法です。

URL: <http://groovy.codehaus.org/modules/http-builder/>

バージョン: 0.7.2

### ■更新情報

- 0.7.2での更新内容は不明です。

## CodeNarc

CodeNarcは、Groovy向けの静的コード解析ツールです。

URL: <http://codenarc.sourceforge.net/>

バージョン: 0.22

### ■更新情報

- 0.22では、新しく4つのルールやIDEのテキストレポートタ

イプが追加され、いくつかバグ対応が行われています。

- 0.22リリースアナウンス: <http://groovy.329449.n5.nabble.com/ANN-Announcing-CodeNarc-0-22-td5721472.html>

## GMetrics

GMetricsは、Groovyソースコードのサイズや複雑さを計算したり報告するためのツールです。

URL: <http://gmetrics.sourceforge.net/>

バージョン: 0.6

## GContracts

GContractsは、Groovyで契約プログラミングを行うためのフレームワークです。

URL: <http://gcontracts.org/>

バージョン: 1.2.12

## GroovyFX

GroovyFXは、JavaFXをGroovyで書きやすくするためのフレームワークです。

URL: <http://groovyfx.org/>

バージョン: 0.4.0

### ■更新情報

- 0.4.0では、ビルダーにcatch-allビーン・ノードが追加されたり、TableViewやComboBoxのバグ対応が行われています。
- 0.4.0リリースノート: <http://jira.codehaus.org/browse/GFX/fixforversion/19127>

## GBench

GBenchは、Groovyのためのベンチマーク・フレームワークです。

URL: <http://code.google.com/p/gbench/>

バージョン: 0.4.2

### ■更新情報

- 0.4.2では、新しいシステムプロパティをサポートしたり、いくつかバグ対応が行われています。
- 0.4.2リリースノート: <https://code.google.com/p/gbench/wiki/ReleaseNotes042>

## Betamax

Betamaxは、HTTP通信の内容を保存し再生するテストツールです。

URL: <http://freeside.co/betamax/>

バージョン: 1.1.2

## Caelyf

Caelyfは、Groovyで記述するCloud Foundry用のライトウェイトなツールキットです。

URL: <http://caelyf.net/>

バージョン: 1.3.1

### ■更新情報

- 1.3.1では、Groovy 2.3.7、Gradle 2.1に対応し、GPars 1.2.1を同梱しました。
- 1.3.1リリースメッセージ: <https://groups.google.com/forum/#!msg/caelyf/UW3XNI-jdjo/OwNxJCU9-sEJ>

## Vert.x

Vert.xは、非同期アプリケーション開発のためのフレームワーク

です。  
URL: <http://vertx.io/>  
バージョン: 2.1.5

#### ■更新情報

- 2.1.5では、Groovy言語モジュールが2.1.1にアップグレードされたり、SSLv2Helloプロトコルがサポートされたり、いくつかバグ対応が行われています。
- 2.1.5リリースメッセージ: <https://groups.google.com/forum/#!topic/vertx/x7Dlhkc8tuA>

## GVM (Groovy enVironment Manager)

GVMは、様々なGroovy関連のツールをインストールし、複数のバージョンを切り替えて使用するためのツールです。

URL: <http://gvmtool.net/>

バージョン:-

## Gaiden

Gaidenは、Markdownでドキュメントを作成するためのツールです。

URL: <https://github.com/kobo/gaiden/>

バージョン: 1.0



#### ▼ぐるーびーたん 第6話までのあらすじ

プログラマーを目指して熊本から上京したぐるーびーたん。その若さ故、色々飛びつくのは良いこと!?

※この作品は、たいがいフィクションです。実在の人物、団体とは関係ありません。



Delight Technologies

NEWCAST

meta bolics

TAS 株式会社  
つうけんアドバンス システムズ

## 個人サポーター制度のお知らせ

JGGUGでは、昨年に引き続き2015年度も個人サポーターを募集いたします。

個人サポーターとなっただいた方には、一年間にわたってJGGUGが発行するG\* Magazine（年数回刊。基本的に電子版として配布予定）に個人サポーターとしてお名前を掲載します（掲載を希望しない旨お申し出いただければ掲載しません）。

個人サポーターとなるには、まず [supporters@jggug.org](mailto:supporters@jggug.org) にメールで

- ・お名前
- ・予定金額

G\* Magazineへのご芳名掲載の可否

をお知らせください。追って、運営委員より振込先の情報などを返信します。

皆様のサポートをお待ちしております。

日本 Grails/Groovy ユーザーグループ  
代表 山田正樹

G\* Magazine vol.8 2015.2

<http://www.jggug.org>

発行人：日本 Grails/Groovy ユーザーグループ

編集長：川原正隆

編集：G\* Magazine 編集委員（杉浦孝博、奥清隆）

デザイン：(株)ニューキャスト

表紙：川原正隆

編集協力：JGGUG 運営委員会

Mail：[info@jggug.org](mailto:info@jggug.org)

Publisher：Japan Grails/Groovy User Group

Editor in Chief：Masataka Kawahara

Editors：G\* Magazine Editors Team

(Takahiro Sugiura, Kiyotaka Oku)

Design：NEWCAST inc.

CoverDesign：Masataka Kawahara

Cooperation：JGGUG Steering Committee

Mail：[info@jggug.org](mailto:info@jggug.org)

© 2015 JGGUG 掲載記事の再利用については  
[Creative Commons ライセンス](https://creativecommons.org/licenses/by-sa/4.0/)によります。

