vol.2

Magazine

Japan Grails/Groovy User Group

個人サポータ制度のお知らせ

JGGUGでは, 2011年度より個人サポータ制度を始めました.

今までJGGUGの経済的な基盤はすべて法人会員の年会費に依存していました。 しかし、個人会員も金銭面からサポートしたいという声があがり、JGGUG運営基盤の裾野を拡げるためにも、個人サポータという形で個人会員からの寄付を受け入れることにした次第です。

個人サポータとなっていただいた方には、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ば掲載しません)。

個人サポータとなるには、まず supporters@jggug.org にメイルで

- 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

Contents

緊急企画

Grails 1.4.0 M1 リリース

検証！Grails 1.4.0 の世界

山本　剛（株式会社ニューキャスト）
出版・印刷関連のシステム設計開発等に従事するテクニカル DTP アーキテクト。
日本 Grails/Groovy ユーザーグループ名古屋支部長。2006 年より Grails のドキュメント翻訳、
その後、Grails 公式の Acegi プラグインを開発。書籍『Grails 徹底入門』(翔泳社発行) の 9、10、11 章を執筆。

今回は、「Grails 1.4.0 M1 リリース」緊急企画として、楽しみにしてた方には申し訳ないですが、前回の続き「Grailsをコントロールせよ！ Part 2」をお休みして、先日リリースされた、Grails 1.4.0 M1の追加機能・変更点など一部を紹介します。

Grails 1.4.x系最初のリリース！

去年に発表されたロードマップの情報から、様々な先行情報が見えてきた、Grails次期バージョン1.4系の最初のマイルストーンがリリースされました。Grails 1.4.x系では変更点が多く、内部的にはかなり深いところまで改善・向上されています。今回の更新内容を公式サイトのロードマップから内容を拝借しますと、

- Groovy 1.8, Spring 3.1 M1, Hibernate 3.6,Servlet 3.0, Tomcat 7への更新。
- エージェントベースリロード
- プラグイン使用統計トラッキング
- AST変換を使用したGORM APIとJava統合
- HTML5で強化したスカッフォルドUI
- 静的リソースハンドリングの向上
- Mixinベースの Unit テストサポート
- バイナリープラグイン
- PrototypeからjQueryへの変更
- 基本データベースのH2への変更
- GORMの機能強化

これだけあると、今回の記事だけでは全ての情報を掲載できません。今回は、このマイルストーンで著者が気になった部分の一部を紹介します。今後も、残りの情報・最新情報があればできる限り記事にしていきたいと思っています。では、早速見ていきましょう。

またしてもUIデザインの変更！

これで３回目となる、ベースデザインの変更です。最初の青っぽいデザインから、次にリニューアルされたSpringカラーのUI、そして時代に合わせてか今回はSpringカラーを採用しながらのHTML5化です。個人的な意見では、CSSの修正だけでもよかったのでは？と思っていますが・・・。スクリーンショットを掲載しておきます。

今回の変更ではHTML5を採用したデザイン内容になっています。中身はModernizr（http://www.modernizr.com/）が使われています。大幅に変わりすぎて、いや、違和感がありすぎて別物に見えたりします。iPhoneのSafari用の調整もしてあるので、iOSでも上手くはまっています。apple-touch-iconも定義してあるので、iOSでのアイコンもバッチリです。

個人的な意見は多々あると思いますが、別テンプレートを作ってしまおうかと。ってのはウソで、がんばって慣れようかと思います。

[新デザイン]

[iOS画面]

[iOSアイコン]

H2になっただけでは無くて！データベースコンソールもつかえる！

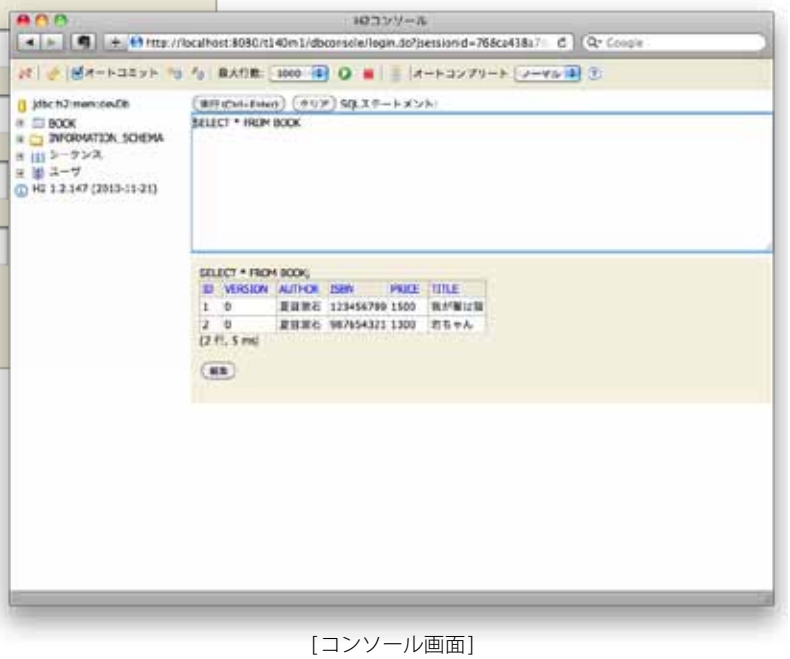
1.4.0からデータベースが、HSQLDBに代わりH2になりました。そして、ただH2になっただけでは無く、H2付属のデータベース管理コンソールも使用できるようになりました。使用方法は単純に、アプリケーションのURLに、/dbconsoleと追加するだけです。例えば、http://localhost:8080/t140m1/ であれば、http://localhost:8080/t140m1/dbconsole/ となります。管理コンソールを参照すると、DBの指定、ドライバー、JDBC URL等を設定するログイン画面が現れます。ここまで来たら、後は接続ボタンを押すだけ！と行きたいところですが、状況によっては、JDBC URLがデフォルトの状態になっている場合があるので設定します。

JDBC URL・ユーザ名・パスワード等の設定を行って接続すると、SQLの実行を行ったりとデータベースの操作ができる管理コンソールが現れます。これは便利！

データベース管理コンソールは、基本的には開発(development)モードでしか動作しません。場合によっては、本運用環境でも管理コンソールを使用したいケースもあります。そんな時はConfig.groovyの環境別設定(environmentsブロック)のproduction以下に、"grails.dbconsole.enabled = true"と、URLの指定"grails.dbconsole.urlRoot = '/admin/dbconsole'"を指定します。本運用で使用する際はセキュリティを考慮して、必ず、spring-security-coreプラグインなどでアクセス制御しましょう。

```
environments {
    production {
        grails.serverURL = "http://www.changeme.com"
        grails.dbconsole.enabled = true
        grails.dbconsole.urlRoot = '/admin/dbconsole'
    }
    development {
        grails.serverURL =
                "http://localhost:8080/${appName}"
    }
    test {
        grails.serverURL =
                "http://localhost:8080/${appName}"
    }
}
```

[DB管理コンソールログイン画面]



[コンソール画面]

Grails 1.4からのUnitテスト

今回のリリースで、Unitテストが大幅改良されて、今まで可能では無かった部分まで､Unitテストでテストを書くことができます。

例えば、インテグレーションテストが必要だったコントローラ関連の、フィルター・URLマッピング・ファイルアップロード・コンテントネゴシエーション・テンプレートビューレンダリングや、ドメインクラスでは、クライテリアクエリーなどもUnitテストでサポートされています。

新Unitテストの一部を試してみたので紹介します。詳しくは1.4.x系のドキュメントを参考にしてください。

■コントローラのUnitテスト

Grails 1.4.x系のUnitテストでは、従来のGrailsUnitTestCaseを使用しません。下位互換用にクラスは存在していますが非推奨です。代わりに、アノテーションgrails.test.mixin.TestForにテストするコントローラクラスを指定します。このアノテーションを指定する事によって、grails.test.mixin.web.ControllerUnitTestMixinと関連するAPIが自動で設定されます。

```
import grails.test.mixin.*
import org.junit.*

@TestFor(HomeController)
class HomeControllerTests {

    void testSomething() {
        //ここにテストを書くよ。
    }
}
```

さらには、今まで、在って当然と思っていた静的スカッフォルドで生成されたコントローラのUnitテストが、grails generate-allコマンドで同時に生成されるようになりました。

生成されたコントローラのUnitテストは、完全な形ではなく、必要な(後で変更があるであろう)ドメインクラスやパラメータの部分はTODO:と記載されており、その部分は実装に応じて自分で記述する必要があります。

例えばmyapp.Bookといったドメインクラスがあるとします。

```
package myapp

class Book {
    String title
    String author
    String isbn
    Long price
    static constraints = {
        title()
        author()
        isbn minSize:9
        price()
    }
}
```

静的スカッフォルドを生成します。

```
% grails generate-all myapp.Book
```

生成されたUnitテストのコード BookControllerTests.groovy は次のようになります。(一部抜粋)

生成されたコードの中にTODO:と書かれた部分があるので、その辺りにドメインクラスの内容に沿ったコードを追記します。(①の部分)

```
import org.junit.*
import grails.test.mixin.*

@TestFor(BookController)
@Mock(Book)
class BookControllerTests {
    @Test
    void testIndex() {
        controller.index()
        assert "/book/list" == response.redirectedUrl
    }
    ..... 中略 .....
    @Test
    void testSave() {
        controller.save()

        assert model.bookInstance != null
        assert view == '/book/create'

        // TODO: Populate valid properties
        // ①ここの部分に追記
        controller.save()

        assert response.redirectedUrl == '/book/show/1'
        assert controller.flash.message != null
        assert Book.count() == 1
    }
}
```

サンプルコード内①の部分が含まれているテストは、testSave()なので、フォームの内容を投げて保存するといった動作になります。ここの部分に必要なのは、フォームに入るべきパラメータ、アクションでのparamsになります。なので次のコードように、サンプルコード内①以下に、paramsマップにフォームに実際に入力する値をセットします。サンプルコード内②のcontroller.save() が実行されることによって、saveアクションのテストが行えます。

```
@Test
void testSave() {
    controller.save()

    assert model.bookInstance != null
    assert view == '/book/create'

    // TODO: Populate valid properties
    // ①ここの部分に追記
    params.title='吾輩は猫'
    params.author='夏目漱石'
    params.isbn='0-00000000-000000'
    params.price=1000

    controller.save()//②

    assert response.redirectedUrl == '/book/show/1'
    assert controller.flash.message != null
    assert Book.count() == 1
}
```

他のアクション、例えば show、edit等の予めドメインクラスが必要になるコードの場合も似たような追記をします。

アクションshowの例を紹介します。今度は、(a)の部分に通常行うのと同じように、Bookのインスタンスを生成して、それぞれのプロパティを設定します。そして、book.save()を実行することで、Unitテストで内部的に保存されBookのidが取得できます。params.idに、Bookのidを設定して、controller.show()を実行することで、showアクションのテストが完成です。

```
@Test
void testShow() {
    controller.show()

    assert flash.message != null
    assert response.redirectedUrl == '/book/list'

    // TODO: populate domain properties (a)
    def book = new Book()
    book.title='吾輩は猫'
    book.author='夏目漱石'
    book.isbn='0-00000000-000000'
    book.price=1000

    assert book.save() != null

    params.id = book.id

    def model = controller.show()

    assert model.bookInstance == book
}
```

ここで気になるのは、Bookクラスの存在です。先ほど、生成されたコードのアノテーション部分をみると @Mock(Book)と指定してあります。この指定でBookクラスのモックが定義されています。それによって、この同一Unitテスト内では、Bookを扱うことができるのです。

```
import org.junit.*
import grails.test.mixin.*

@TestFor(BookController)
@Mock(Book)
class BookControllerTests {
..... 省略 .....
```

@Mockの指定は、複数のドメインクラスを指定する事も可能です。その場合は @Mock([Book, Author]) と、配列で定義するだけです。

@Mockを指定する事で、ドメインクラスをデータベースに繋ぐことなくGORMの機能をテストできる、DomainClassUnitTestMixinが設定されます。DomainClassUnitTestMixinを使用するとメモリ上のConcurrentHashMap内でGORMのデータが構築され動作します。

@TestMixinで、直接DomainClassUnitTestMixinを定義することも可能です。この場合は、ドメインクラスのモックは、mockDomain(ドメインクラス名)で直接生成します。

```
@TestFor(BookController)
@TestMixin(DomainClassUnitTestMixin)
class BookControllerTests {
    void testSave() {
        mockDomain(Author)
        mockDomain(Book)
    }
}
```

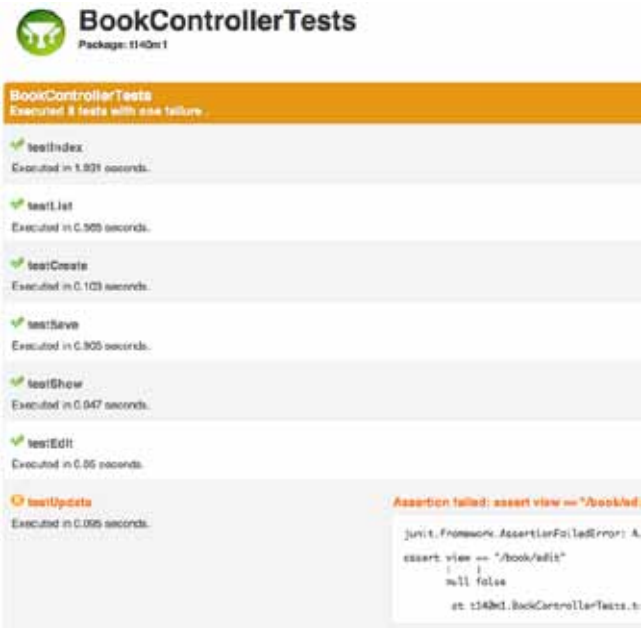
■ドメインクラスのUnitテスト

ここまででわかるように、コントローラUnitテストでドメインクラスのUnitテストも簡単に行えるのがわかります。ただ、やはり制約などは、ドメインクラス用のテストを書きたいと思います。従来通りドメインクラスをcreate-domain-classコマンドで生成すると、ドメインクラスのUnitテストコードも生成されます。但し内容は以前と同じ何も記述されていない状態です。

先ほどのドメインクラスBookのUnitテストの例を紹介します。※誌面の関係上今回は多く説明しません。今後の参考になればと思います。

```
import grails.test.mixin.*
import org.junit.*

@TestFor(Book)
class BookTests {

    void testSomething() {
        def book = new Book()
        book.title='吾輩は猫'
        book.author='夏目漱石'
        book.isbn='123456789'
        book.price=1000
        assert book.save() != null

        assert book.id != null
        assert book.title=='吾輩は猫'
    }

    void testConstraints(){
        mockForConstraintsTests Book

        def book = new Book()
        assert !book.validate()
        assert "nullable" == book.errors["title"]
        assert "nullable" == book.errors["author"]

        book.title='吾輩は猫'
        book.author='夏目漱石'

        book.isbn='12345678'
        assert !book.validate()
        assert "minSize" == book.errors["isbn"]

        assert !book.validate()

        book.isbn='123456789'
        book.price=1000
        assert book.validate()
    }
}
```

最後にテストレポートもデザインが変更になったので、スクリーンショット載せておきます。



[テストレポート]

今回は、UI・DB・Unitテストを紹介しました。まだ正式版のリリースは時期が2011 Q3としかわかっていません。これからM2,M3?と1.4系のマイルストーンリリースラッシュになるのは確かです。速度・機能と共に向上したGrails 1.4系、とても楽しみですね。

次号からも、時間がある限り、複数の著者で順番に「検証！Grails 1.4.0 の世界。」シリーズを掲載していきたいと思います。

series 02

Griffon 不定期便 ～第 3 回 スレッド編 その 1～

奥 清隆 (おく きよたか)
仕事でもときどき Groovy と戯れるプログラマ。
日本 Grails/Groovy ユーザーグループ関西支部長。
著書：『Seasar2 による Web アプリケーションスーパーサンプル』

G*なみなさま、こんにちは。今回からスレッド編として2回に分けてGriffonアプリケーションでのスレッドの扱い方を紹介していきます。今回はSwingアプリケーションでスレッドを扱うための基本となるSwingのスレッドポリシーとGroovyのSwingBuilderでスレッドを扱う方法を紹介し、次回はGriffonでのスレッドの使い方をサンプルアプリケーションを作りながら紹介したいと思います。

Webアプリケーションの開発がメインでSwingアプリケーションにあまり馴染みのない方は、これまでにスレッドを扱う機会が少なかったかと思います。Webアプリケーションでは独自のスレッドを生成して処理するということはあまり必要とされません。しかし、Swingアプリケーションではスレッドの扱いを理解しておくことが大変重要になります。

Swingのスレッドポリシー

Swingアプリケーションで発生するイベントはすべて、イベントディスパッチスレッドと呼ばれる1つのスレッドで処理されます。

また、基本的にSwingコンポーネントはスレッドセーフではないため、Swingコンポーネントに対する操作はすべてイベントディスパッチスレッド上で実行する必要があります。「基本的に」と言いましたが、例外もあります。例えば、テキストエリアにテキストを追加するjavax.swing.JTextAreaクラスのappend(String)メソッドはスレッドセーフになっています。どのメソッドがスレッドセーフかどうかはJavadocに記述されています。例えばjavax.swing.JTextAreaのappend(String)メソッドのJavadocには以下の注意書きが記述されています。

> *このメソッドはスレッドに対して安全ですが、ほとんどの Swing メソッドは違います。*

スレッドセーフなメソッドをイベントディスパッチスレッド内で呼び出しても問題ありませんが、スレッドセーフではないメソッドをイベントディスパッチスレッド外で呼び出した場合は、予期しない動作をすることがあるでしょう。スレッドセーフに関する問題は、再現性がないため慎重に実装する必要があります。

イベントディスパッチスレッド上で処理を行うにはいくつか方法があります。まず、各Swingコンポーネントに登録したイベントリスナの各アクションイベントを受け取るメソッド(java.awt.event.ActionListenerクラスのactionPerformed()メソッドなど)はイベントディスパッチスレッド上で実行されます。イベントディスパッチスレッド外からSwingコンポーネントに対する操作を行う場合はjavax.swing.SwingUtilitiesに用意されているstaticメソッドを利用してイベントディスパッチスレッド上で処理を行うようにスケジュールします。

SwingUtilities

SwingUtilitiesに用意されているイベントディスパッチスレッド上で処理をするためのメソッドにはinvokeLater(Runnable)、invokeAndWait(Runnable)があります。それぞれイベントディスパッチスレッド上で実行する処理をjava.lang.Runnableインターフェースに実装したクラスのインスタンスを引数に取ります。invokeLaterメソッドはイベントディスパッチスレッドにスケジュールします。スケジュールされた処理は、スケジュールされた順番に実行されます。invokeLaterメソッドはスケジュールしたあとで呼び出し元に処理が戻りますが、invokeAndWaitメソッドはスケジュールした処理が実行されるのを待ってから呼び出し元に戻ります。次のコードでそれぞれのメソッドの動作を確認できます。

▼invokeLater_sample.groovy

```
import javax.swing.SwingUtilities
import groovy.swing.SwingBuilder

println "1 - ${Thread.currentThread().name}"

new SwingBuilder().edt() {
    frame(show:true, pack:true) {
        button 'Click', actionPerformed: {
            doOutside {
                SwingUtilities.invokeLater {
                    sleep 1000
                    println "2 - ${Thread.
                            currentThread().name}"
                }
                println "3 - ${Thread.
                            currentThread().name}"
            }
        }
    }
}
```

▼invokeAndWait_sample.groovy

```
import javax.swing.SwingUtilities
import groovy.swing.SwingBuilder

println "1 - ${Thread.currentThread().name}"

new SwingBuilder().edt() {
    frame(show:true, pack:true) {
        button 'Click', actionPerformed: {
            doOutside {
                SwingUtilities.invokeAndWait {
                    sleep 1000
                    println "2 - ${Thread.
                            currentThread().name}"
                }
                println "3 - ${Thread.
                            currentThread().name}"
            }
        }
    }
}
```

両方とも、同じウィンドウが表示されますが、ボタンをクリックしたときの挙動が異なります。それぞれdoOutside()を使用してイベントディスパッチスレッド外からSwingUtilitiesクラスのメソッドを呼び出すようにしています。（doOutside()メソッドについては後で説明します。）

最初のサンプルコードを実行し、ボタンをクリックすると標準出力に次のように表示されます。

```
1 - main
3 - Thread-4
2 - AWT-EventQueue-0
```

invokeLater()メソッドに渡した処理を待たずに3番目のprintlnが実行されているのがわかります。invokeAndWaitを使用するサンプルコードの実行結果は次のようになります。

```
1 - main
2 - AWT-EventQueue-0
3 - Thread-4
```

こちらは、invokeAndWait()メソッドに渡した処理が実行されて終わるのを待ってから3番目のprintlnが実行されているのがわかります。イベントディスパッチスレッド外からSwingコンポーネントを操作する場合は状況に合わせてこの2つのメソッドを使用することでスレッドセーフなアプリケーションになります。

すべての処理をイベントディスパッチスレッド上で実行すればスレッドセーフに関する問題はなくなりますが、時間のかかる処理をイベントディスパッチスレッド上で実行するとアプリケーションがフリーズしたような状態になります。

次のようなコードを実行してみましょう。

```
import groovy.swing.SwingBuilder

new SwingBuilder().edt {
    frame(show:true, pack:true) {
        gridLayout(cols:1, rows:2)
        label id:'label'
        button '更新', id:'button',
                                actionPerformed: {
            sleep 3000
            label.text = new Date().
                            format('HH時mm分ss秒')
            println 'Clicked!'
        }
    }
}
```

更新ボタンをクリックすると、3秒後に時刻が表示され、標準出力に「Clicked!」と出力されます。javax.swing.JLabel#setText(String)メソッドは、スレッドセーフではありませんが、ボタンのactionPerformedに設定したクロージャはイベントディスパッチスレッド内で実行されるため、このアプリケーションはスレッドセーフです。しかし、更新ボタンをクリックしたときにボタンが押下状態のままになり、アプリケーションがフリーズしているように見えます。

押下状態のままボタンをクリックしても動いていないように見えますが、実はクリックしたときに発生するイベントがイベントディスパッチ上にスケジュールされています。しばらく押下状態が続いたのち、クリックした回数だけ標準出力に「Clicked!」と表示されます。もしこれが何かデータを登録する処理だった場合、ユーザがクリックした回数だけデータが登録されてしまいます。このようにアプリケーションが止まっているような状態になるのを回避するためには、イベントディスパッチスレッドから新しくスレッドを生成して処理する必要があります。この方法については後ほど紹介いたします。

Swingのスレッドポリシーについては Javadocに記述されていますので、一度読んでおくといいでしょう。

http://java.sun.com/javase/ja/6/docs/ja/api/javax/swing/package-summary.html#threading

SwingBuilderでスレッドを扱う

Swingでのスレッドの扱い方を紹介しましたが、GroovyのSwingBuilderではスレッドの扱いが簡単でさらに賢く使えるようになっています。SwingBuilderではスレッドを扱うメソッドとして、edt()、doOutside()、doLater()の3つが用意されています。それぞれのメソッドはSwingUtilitiesと以下のように対応します。

- edt()
  SwingUtilities#invokeAndWait()に相当。イベントディスパッチスレッド上で呼び出された場合は引数のクロージャをそのまま実行。
- doLater()
  SwingUtilities#invokeLater()に相当。
- doOutside()
  イベントディスパッチスレッド上で呼び出された場合は、新しいスレッドを生成して、引数のクロージャをイベントディスパッチスレッド外で実行。イベントディスパッチスレッド外で呼び出された場合はクロージャをそのまま実行。

これらのメソッドを使わずに、SwingUtilitiesを使用することもできますが、SwingBuilderのメソッドを使用することで処理がどのスレッドで実行しているかが分かりやすくなります。またSwingUtilities#invokeAndWait()メソッドはイベントディスパッチスレッド内から呼び出すとエラーが発生しますが、SwingBuilderのedt()メソッドは呼び出されたスレッドがイベントディスパッチスレッドかどうかを判断してくれるので使い勝手もよくなります。

SwingBuilderを使用して先ほどのサンプルコードを正しく実装し直してみましょう。

```
import groovy.swing.SwingBuilder

new SwingBuilder().edt {
    frame(show:true, pack:true) {
        gridLayout(cols:1, rows:2)
        label id:'label'
        button '更新', id:'button',
                                actionPerformed: {
            // ①
            button.enabled = false
            label.text = ''
            // ②
            doOutside {
                3.times {
                    // ③
                    edt { label.text += '.' }
                    sleep 1000
                }
                // ④
                doLater {
                    label.text = new Date().
                          format('HH時mm分ss秒')
                    button.enabled = true
                    println 'Clicked!'
                }
            }
        }
    }
}
```

ボタンをクリックしたときのイベントはイベントディスパッチスレッド上で実行されます。処理が終わるまでボタンをクリック出来ない様にjavax.swing.JButton#setEnabled(boolean)メソッドを使用してボタンを無効化しています（①）。

次に時間のかかる処理をバックグラウンドスレッドで実行するようにdoOutsideを使用しています（②）。doOutsideに渡すクロージャはイベントディスパッチスレッドとは別の新しいスレッドで実行されます。3秒待つために、1秒間スレッドを止める処理を3回ループしています。このループの中でラベルのテキストに「.」を追加しています（※）。これはユーザに処理中であることを知らせる簡易的なプログレスバーと考えてください。このとき、処理はバックグラウンドスレッド内で実行されているので、イベントディスパッチスレッド上でラベルのテキストを変更する必要があります。ラベルのテキストを変更したあとでスレッドを止めるにはedt()メソッドを使用します（③）。doLater()を使用してもイベントディスパッチスレッド上で実行されますが、処理をスケジュールするだけですので後続のスレッド停止処理と並行して実行されることになります。

3秒スレッドを停止したら、ラベルに時刻を表示して、ボタンを有効化させます（④）。ここではdoLaterを使用して処理をスケジュールさせます。バックグラウンドスレッドはこれで終了しますが、後でイベントディスパッチスレッド上でラベルに時刻が表示され、ボタンがクリック出来る状態になります。

※ボタンを無効化させるだけでもアプリケーションとしては問題ありませんが、それだけでは待たされる方としてはアプリケーションが動いているのかどうかわかりません。今回は説明をシンプルにするためにラベルのテキストを変更していますが、実際にアプリケーションを作成する場合はjavax.swing.JProgressBarなどを使用して処理の進捗状況を知らせるのが良いでしょう。SwingBuilderではprogressBar()メソッドを使用してjavax.swing.JProgressBarコンポーネントを利用できます。

まとめ

イメージとしては、仕事中の自分がイベントディスパッチスレッドと考えてください。何か作業が発生（イベントが発生）し、その作業を他の誰かに振るのがdoOutside()です。作業を他の人に振ったので別の作業に取り掛かれます。作業を振られた人が作業の途中であなたに確認を取らなければいけなくなり、「確認お願いします」と言い、あなたは「そこに置いといて下さい、手が開いたときにやっときます。」というのがedt()です。確認をお願いした方は確認が終わるまで作業を終えることができませんし、他のこともできません（ひどい話ですが例えばの話です）。

また別の作業があって、その作業は誰にも振らず自分でやるとします。自分でやりますが別の作業があって今やれないときは「あとでやる」ことにします。これがdoLater()です。

あまり関係ないですが、私は「あとでやる」というのが好きです。あとでやるというか「明日やる」ことにするのが好きです。そしてギリギリになってやる羽目になり泣きながらやります。この記事を書いている今も少し泣いています。

少しややこしくなりましたが、Swingのスレッドポリシーを理解することでアプリケーションの出来が変わってきます。SwingBuilderのedt()、doLater()、doOutside()の使い方をマスターすれば、より簡単にスレッドを扱うことが可能になります。今回はSwingのスレッドポリシーとGroovyで使う方法の紹介で終わってしまいましたが、次回はいよいよGriffonでスレッドを扱う方法を紹介いたします。それではまた次回。

series 03

CodeNarcを利用してGROOVYのコード品質を上げる ～第２回 開発ツールと連携～

荒井健太郎
某ITベンダーのセキュリティコンサルタント、ソフトウエアをよりセキュアにするために日々精進しています。
テスト系スクリプトは、G* を利用して書いています。

はじめに

前回はCodeNarcの基本操作を解説した。

今回は開発の現場でより実践的に利用できるようにするために方法として次の項目を解説する。

- IntelliJ IDEAとの統合
- Grailsプラグインの利用

最終回の第3回は解析ルールのカスタマイズ方法を解説する。

本稿は次の環境を前提で話を進める。

- Groovy1.7がインストールされていること
- JDK1.6がインストールされていること

IntelliJ IDEAとの統合

■IntelliJ IDEAのインストール

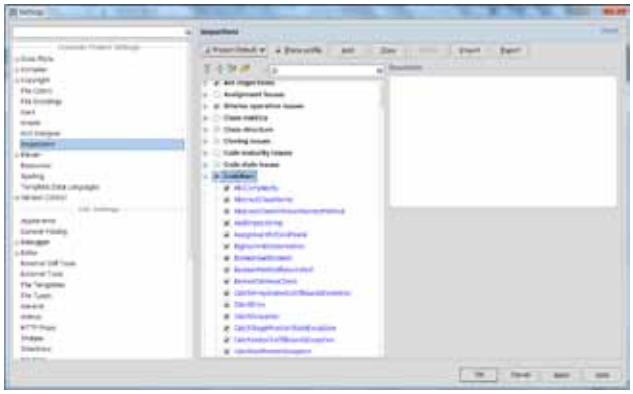
IntelliJ IDEAは、チェコのJetBrains社により開発されている統合開発環境である。Javaの開発は有償版でないと実施できないがGroovyの開発はCommunity版で可能である。機能の詳細等はWeb等の記事を参考にして頂きたい。

Community版 は、http://www.jetbrains.com/idea/download/index.html からダウンロード可能である。インストールはインストーラーの指示に従って簡単にインストールすることができるので各自でインストールして頂きたい。

■CodeNarcプラグインの組込み

IntelliJ IDEAを起動し、_File_→Settingsの順番にクリックし左側ペインで**Plugins**を選択する。右側ペインで**Available**タブをクリックし**CodeNarc**を選択し右クリックし**Download and Install**をクリックする。ダウンロード及びインストールが実行された後に**OK**をクリックすると再起動をうながすポップアップがあがるので再起動する。

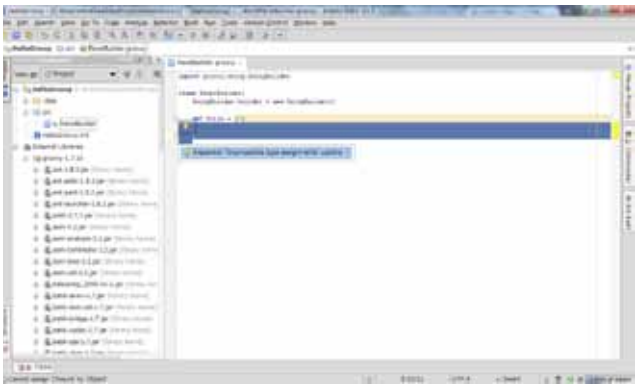
再起動後、**File**→**Settings**をクリックし右側ペインで**Inspections**をクリックすると右側ペインにCodeNarcが表示されているのでチェックをいれる(図1)。



[図1]

■解析の実行

Inspectionsは有効にすると即座にコードに反映される。問題を検知した行はエディタの右側に黄色で表示され、該当行はうすいオレンジ色で反転される。反転されている場所にカーソルを合わせるとライトのマークが左に表示されクリックすると適用されているルールや変更適用などの項目などが表示される(図2)。



[図2]

Grailsプラグインの利用

GrailsのCodeNarcプラグインを利用すると、Grailsで作成されたWebアプリケーションに対して容易にCodeNarcの解析を実行することができる。

本稿はGrailsを利用して作成したアプリケーションが既にある前提で話を進める。Grailsの詳細については、G*Magazine準備号の「Grails - JavaEE 開発をライトにするフルスタックフレームワーク」等の記事を参照して頂きたい。

■Grailsプラグインの導入

Grailsプロジェクトディレクトリに移動し次のコマンドを実行する。

```
grails install-plugin codenarc
```

Grailsのスクリプトが実行されプラグインが導入される。

■Grailsプラグインの実行

Grailsプロジェクトディレクトリに移動し次のコマンドを実行する。

```
grails install-plugin codenarc
```

CodeNarcの解析がGrailsプロジェクトに対して実行され、**CodeNarcReport.html**がプロジェクトディレクトリに作成される。

■Grailsプラグインの設定

●標準の設定

Grailsプラグインは標準の設定でGrailsプロジェクトの次のディレクトリに対して解析を実行し、HTML形式の**CodeNarcReport.html**レポートを生成する。

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

また、解析に利用されるルールは、rulesets/basic.xml,rulesets/exceptions.xml, rulesets/imports.xml,rulesets/grails.xml, rulesets/unused.xmlである。

設定は、_grails-app/conf/config.groovy_ に _codenarc.XXX_ を追加し変更できる。代表的なものを次節以降で紹介する。

●レポートのカスタマイズ

出力レポートのカスタマイズは次の書式で変更する。

```
codenarc.reports = {
      それぞれのレポートに対して以下の設定を実施する:
        REPORT-NAME(レポート形式) {
            プロパティ名 = プロパティ値
            プロパティ名 = プロパティ値
        }
}
```

レポート形式は、_html_, _xml_, _text_, CodeNarcのReportWriterを継承したクラス名を指定できる。

次は、XML形式でファイル名がCodeNarc-Report.xml、タイトルがSample Reportのレポートと HTML形式でファイル名がCodeNarc-Report.html、タイトルがSample Reportのレポートを出力する例である。プロパティを編集しGrailsプラグインを実行すると、設定した値でレポートが作成されることが確認できる。

```
codenarc.reports = {
    MyXmlReport('xml') {
// The report name "MyXmlReport" is user-defined;
Report type is 'xml'
        outputFile = 'CodeNarc-Report.xml'
// Set the 'outputFile' property of the (XML)
Report
        title = 'Sample Report'
// Set the 'title' property of the (XML) Report
    }
    MyHtmlReport('html') {
// Report type is 'html'
        outputFile = 'CodeNarc-Report.html'
        title = 'Sample Report'
    }
}
```

●ルールのカスタマイズ

利用するルールのカスタマイズは次の書式で変更する。

```
codenarc.ruleSetFiles="file:利用するルールファイル"
```

次は、カスタムルール(次回に解説予定)MyRuleSet.groovyを利用する例である。

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

●解析対象ファイルのカスタマイズ

- codenarc.processSrcGroovy - src/groovyを解析対象とするか。(標準はtrue)
- codenarc.processControllers - grails-app/controllersを解析対象とするかを指定。(標準はtrue)
- codenarc.processDomain - grails-app/domainを解析対象とするかを指定。(標準はtrue)

- codenarc.processServices - grails-app/servicesを解析対象とするかを指定。(標準はtrue)
- codenarc.processTaglib - grails-app/taglibを解析対象とするかを指定。(標準はtrue)
- codenarc.processTestUnit - test/unitを解析対象とするかを指定(標準はtrue)
- codenarc.processTestIntegration - test/integrationを解析対象とするかを指定(標準はtrue)
- codenarc.processViews - grails-app/views以下のGSPを解析対象とするかを指定(標準はfalse)
- codenarc.extraIncludeDirs - 上記以外のソースを解析対象とする場合に指定

次は、'grails-app/jobs/*.groovy'を解析対象とする場合の例である。

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

■まとめ

今回は、CodeNarcをIntelliJ IDEAと統合する方法及びGrailsプラグインを利用して静的解析する方法を解説した。これによって、CodeNarcがより現場で利用されるようになればと考えている。

今回紹介した以外にもさまざまなフレームワーク、ツールと連携することが可能である。詳細は、http://codenarc.sourceforge.net/codenarc-other-tools-frameworks.htmlを参照して頂きたい。次回はプロジェクト固有のルールを作成する方法を解説し最終回とする予定である。

series 06

もし新人女子 Java プログラマが『Groovy イン・アクション』を読んだら ～第１章～

吉田　健太郎

仕事では Java、趣味でライフワークの如く Groovy と戯れるプログラマ。
日本 Grails/Groovy ユーザーグループ運営委員参画予定。
ブログ：『No Programming, No Life』http://d.hatena.ne.jp/fumokmm/

第１章　もかは『Groovy イン・アクション』と出会った

■新人女子 Java プログラマ

4月、それは新社会人にとって新たな幕開けの季節だ。もかにとってもそれはそうだった。七海 萌香(ななみ もか)、はソフトハウスとしては中堅どころのジェイガ株式会社へとプログラマ志望で入社した新入社員の一人だった。

新人教育でビジネスマナーやら簡単な手続きを済ませた後は、早速プログラミング講習が待っていた。言語はJavaだった。もかはこれまで簡単なホームページをHTMLやCSS、それからWeb1.0時代のJavaScriptで書いたことがあるくらいだったので、これが本格的なプログラミング言語との出会いだった。

今年の新人は男性が８名、女性が２名であった。もう一人の新人女子社員は一ノ瀬 小夏(いちのせ こなつ)という名前の女の子だった。こなつは情報系専門学校を卒業しており、当初はゲーム開発に憧れて情報系に進んだらしいのだが、現実の厳しさなどを目の当たりにし、それから色々あってジェイガに入ることに決めたらしい。どうあれ、新人女子社員は二人だけということもあり、もかとこなつはすぐに打ち解けることができた。

■ある日の定時後

もか「あー今日も終わった終わった～！お疲れちゃ～ん！」
こなつ「はい…お、お疲れ様です」
もか「へへーん、頑張ったわ私！　さてっと…ねぇねぇ、こなつぅ！ちょっといい？」

私と違ってこなつはちっちゃくて目もクリっとしててかわいいんだよね、ついつい誘いたくなっちゃう。

こなつ「え、あ、はい…何ですか？」
もか「今日ってさ、このあと時間ある？ どこかにパーッと遊びに行きたいなぁって思ってるんだけど」
こなつ「え、あ、はい…でも私ちょっと寄りたいところが…」

こなつは困った表情をしている。

もか「あ～そうなんだ…。ふーん、じゃあ仕方ないかぁ」
こなつ「私、Javaの講義について行けてなくて…会社からもらった参考資料だけじゃだめだなーって思うんです。だからちょっと本屋さんに寄って、Javaの本を買いたいなって思ってるんです！」

もか「なるほどね、こなつは勉強熱心だよね、感心しちゃうよ」
こなつ「いえいえ、そんなことないです…私、出来が悪いし…みんなについていくのに必死なんです」

それを聞いていたもかは目を閉じてしばし無言になった。しばらく何か考えている様子だったが、ゆっくりと目を開いてからこう言った。

もか「私も一緒について行っていい？」
こなつ「え、あ、はい…いいですよ」

きょとんとしているこなつを尻目に、もかは嬉しそうににんまりと笑顔を見せた。

こなつは勉強熱心なのだが、どうやらプログラミングには自信がないらしく苦労しているようである。会社でもJavaの講義に追いついていくのがやっとといったところのようである。だから少しでも追いつこうとして、参考書を購入しようというわけだ。

もか「本屋さんって近くにあるの？私、どうも本って苦手でさぁ…一人では本屋さんなんてあまり行かないから」
こなつ「え、そ…そうなんですか？ごめんなさい、私誘ちゃったみたいで…」
もか「え？ううん、今日は私がついて行きたいって言い出したんだし、こなつは全然悪くないよ！むしろ謝らなきゃいけないのは私の方だよね」
こなつ「ううん、一緒に来てくれるの嬉しいですよ、一緒に選んでもらえるとすごく助かります」

こなつはえへっと笑顔を見せた。

■書店にて

もか「さーて、着いたわね。こなつはもうどれを買うか決めてるの？」

こなつ「え、えーと…か、簡単そうなのがいいなぁと思って…講義の内容がちょっと難しくて。ほら、このあいだ、classの話になったあたりから急に難しくなったでしょう？だから分かりやすく丁寧に書かれている参考書が欲しいなって思ってるんです」

それを聞いたもかは、ある本を指差しながらこう言った。

もか「ふーん、なるほどね。じゃあさ、これなんてどう？『たのしいJava』っていうタイトルからしていかにもって感じじゃない？」

もかはその本を取って渡してあげた。

こなつ「わぁ～いいかもです。ちょっと見て見ますね」

こなつがその本を見ている間、もかはJavaの本が並ぶ棚の他の本を眺めて回っていた。

もかはJavaこそ今回の研修で初めてやったとはいえ、プログラミングというものにはずっと興味を抱いていたし、JavaScriptで簡単なアルゴリズムだったり、クラスだったりの概念は理解しているつもりである。今日まで行われた講義も分からなくなるところは全然なかった。

いっぱい本あるなぁ…と、そんなことを思いながら何気なく本を眺めていると、もかの視界にとある本が飛び込んできた。その本のタイトルは、

『Groovyイン・アクション』

んっ？ぐるーびー？何だろうこれ。なんでJavaのコーナーに？

他の本はJavaという名前がつく本ばかりであるのに。気になったのでちょっと手に取ってみた。ずっしりと重みを感じる。それもそのはず、600ページ近くもあってかなり分厚い。表紙にはなにやら日本人形のような踊り子さんが描かれている。帯にはこうあった。

●Groovyとは

Javaプラットフォームのためのアジャイルで動的な言語です。PythonやRuby、Smalltalkなどの言語から多くの機能を取り入れ、Java開発者がJava風の構文を使ってこれらを利用できるようになっています。スクリプト言語として言及されることも多いのですが、その枠にとどまるものではありません。Java実行環境上で動作するため、Javaバイトコードにコンパイルすることも可能であるなど、非常に強い「Javaとの親和性」を備えています。

ふーん…？なんだかすごそうね。PythonとかRubyとかは聞いた事はあるけど、Groovyってどんな言語なんだろう？すもーるとーくっていうのは知らないけど…。ちょっと興味がわいたのでパラパラとページをめくってみた。

もかはふとP63のコードに目が止まった。そのページのタイトルは「3.4.3 JavaからGroovyへ」となっていた。

```
System.out.println("Hello Groovy!");
```

うんうん、これはJavaのコードよね、一番初歩的はハローワールドだわ。そしてすぐ下はこう続いていた。

```
System.out.println('Hello Groovy!');
```

ダブルクォーテーションがシングルクォーテーションになったのね、JavaScriptみたいなものかしら。

もかはそこまで気に留めず、例をさらに読み進めた。

```
print('Hello Groovy!');
```

すごいすっきりしてきたわ。Javaだと System.out.println ってドットでつないで長く書いていかないといダメけど、Groovyだと System.out が要らないんだ。

```
print 'Hello Groovy!'
```

わぁ、括弧までなくなっちゃった。すごくスッキリした感じ。もかはさらに読み進めた。

```
greeting = 'Hello Groovy!'
assert greeting.startsWith('Hello')

assert greeting.getAt(0) == 'H'
assert greeting[0]       == 'H'

assert greeting.indexOf('Groovy') >= 0
assert greeting.contains('Groovy')

assert greeting[6..11] == 'Groovy'

assert 'Hi' + greeting - 'Hello' == 'Hi Groovy!'

assert greeting.count('o') == 3

assert 'x'.padLeft(3) == '  x'
assert 'x'.padRight(3,'_') == 'x__'
assert 'x'.center(3) == ' x '
assert 'x' * 3 == 'xxx'
```

え？ほんと？こんな記述でいいの？assertってあれ？こないだの講義でやったJUnitってやつにも出て来たやつだよね、assertEqualsとかってやつ。ってことはこれはつまり、テストコードなんだわ！（後にもかはP28, P29ページのassertの解説を読んでassertEqualsよりもGroovyのassertがもっと軽量に使えて便利だということを知ることになる）

え？うそうそ！？文字列に対してマイナスが使えるの？？あと[6..11]で切り出したりできるの？すごい！それから、最後の

padLeft, padRight, centerっていうので文字を空白埋めしたり、アンダーバーで埋めたりできちゃうんだ！それからそれから、文字のかけ算も？

もかはすごく興奮しだしている自分を感じていたが、頭の中で、これはJavaじゃないんだから今の私とは関係ないないんだ！と言い聞かせるようにしていた。

そうこうしているうちに、こなつがニコニコしながらもかの側に寄って来た。手には先ほどの『たのしいJava』を持っている。

こなつ「もかちゃん！私この本にするね」
もか「そうだね、その本ならJavaに限らずプログラミングの基本的なところも勉強できるしいい本だと思うわ」
こなつ「私、この本読んで頑張る！みんなについて行けるように」
もか「うんうん、頑張って勉強して一緒にかっこいいプログラマになろうねっ！」
こなつ「ありがとう、もかちゃん。あれ？ところでもかちゃんも何か本を買うの？」

こなつはもかが持っている『Groovyイン・アクション』に目がいった。

もか「あー、えっと、なんかこれね、Javaとは違うプログラミング言語の本なんだけど、ちょっと面白そうなのよ。まだちょっと立ち読みしてただけだから、あまりわからないんだけどJavaとの親和性が高いんだって」
こなつ「えーー？すごいね、やっぱりもかちゃんは…私なんてJavaだけで精一杯なのにもかちゃんは違うプログラミング言語も勉強してるの？私どんどん置いて行かれちゃう…」
もか「ちがうちがう、私もJavaは今回の講義で始めたばっかりだし、良くわかってないところが多いし…まずはJavaをマスターしなきゃ！」

そう言いながら、もかは『Groovyイン・アクション』を棚に戻した。

■自室にて

その日の夜。自室のベッドに入って寝る前にもかは今日本屋さんで見た『Groovyイン・アクション』のことを思い出していた。なんとなく興奮が止まらない。プログラミングは好きだけど、こんなにドキドキしたことって今までなかったな。もしかしたらこれって運命の出会い？とか言っちゃって。なにそれ…素敵な男の子じゃなくてプログラミング言語との出会いなんて。もかはそんなことを考えながら苦笑する。もう…こんなこと考えてるから私ってモテないのかなぁ、女子力っていうの？全くない！こなつがうらやましいなぁ…ちっちゃくて、かわいくて。ああいう子は絶対にモテるんだよなぁ…。

とりとめのないことを考えながら、いつのまにかもかは眠りについていた。

■Javaの演習

次の日、出社したもかたちはJavaの講義を受けているところだ。今日も恒例の課題が出題されるようだ。

「さーて、みんなそろそろJavaに慣れてきたところだろう、次はこんな問題だ。周りと相談しないで自分だけで解くように」

正の整数nが与えられたときに、高さnのピラミッドを出力するプログラムを作ってください。
n=4の時の出力は下のようになります。

```
   *
  ***
 *****
*******
```

もかはピンときた。なるほどね、これは単純な文字列操作の例ね。よーし、作るぞぉ！もかは意気込んでこの課題に取り組んだ。数十分後、Javaのコードが出来上がった。

```
public class Pyramid {
    /**
     * ピラミッドを出力するプログラム♪
     * @author 七海 萌花
     * @param args 何段にするか
     */
    public static void main(String[] args) {
        int dan = Integer.parseInt(args[0]);
        for (int i = 0; i < dan; i++) {
            String hidari = "";
            String migi = "";
            for (int j = 0; j < dan - (i + 1); j++) {
                hidari += " ";
                migi += " ";
            }
            String moji = "";
            int mojisuu = 1 + (2 * i);
            for (int j = 0; j < mojisuu; j++) {
                moji += "*";
            }
            System.out.println(hidari + moji + migi);
        }
    }
}
```

ふぅ～、なんとかできた。こんなもんだよね？他の人はできた

のかなぁ〜、ちょっと遊びに行ってみようかな。もかはこなつの席に近寄ってみた。すると苦悶の表情を浮かべているこなつがそこにいた。

もか「こ…こなつ？大丈夫？苦しそうだけど…」
こなつ「え、あ、はい…ちょっと考え過ぎちゃって頭がぐるぐるなんです…」
もか「そっかそっか…まだ時間はゆっくりあるし、頑張って考えてみるといいよ」
こなつ「はい…もかちゃんはもう出来たの？」
もか「うん、まぁね。あれであってるかどうかは分からないけど」
こなつ「やっぱりもかちゃんすごいね！他の誰よりも早いもの…すごいです！」

こなつが突然大きな声を出すので、みんなの視線が二人の方に集まった。

もか「（小声で）こ、コラー（笑）恥ずかしいじゃん」
こなつ「（小声で）ごめんなさい（笑）」

途中でこなつのコードをちらっと見せてもらったのだが、こなつは各段を全部打ち込んでしまっているようで…

```
String hoshi4_1 = "   *   ";
String hoshi4_2 = "  ***  ";
String hoshi4_3 = " ***** ";
String hoshi4_4 = "*******";
String hoshi5_1 = "    *    ";
String hoshi5_2 = "   ***   ";
String hoshi5_3 = "  *****  ";
String hoshi5_4 = " ******* ";
String hoshi5_5 = "*********";
if (args[0].equals("4")) {
    System.out.println(hoshi4_1);
    System.out.println(hoshi4_2);
    System.out.println(hoshi4_3);
    System.out.println(hoshi4_4);
} else if (args[0].equals("5")) {
            .
            .
            .
}
```

のような感じになってしまっていた。これだともっといろんな段数のパターン全部作るわけにはいかないよね？と教えてあげたら数十秒固まったあとになんとか納得してくれたようで、こなつはまた苦渋の表情を浮かべながらソースコードとにらめっこを始めたのだった。

■もしGroovyだったら

自分の席に戻ったもかは課題をちょっとだけ早く終わったことで空いた時間、昨日のことを思い出していた。そういえばこの問題、もしかしたらGroovyで書いたらもっと簡単に書けるんじゃないのかな？ええと確か、そうそう、centerっていうのがあったわよね。たとえば、1段目なんて、"*".center(7) とかすればできるんじゃないかしら？わくわくしてきて、もかはそのあとずっとGroovyのことを考えていた。

その日の研修が終わり定時になると、もかは誰とも話をせずに書店に走っていた。目的はもちろん一つ、『Groovyイン・アクション』を手に入れることだった。書店に到着したもかはさっそく、昨日『Groovyイン・アクション』を見つけた棚へと走った。

「はぁ…はぁ…、あ、あったわ！」

そんなに焦らなくてもよかったのかもしれないけれど、もかは本がそこにあったことで安堵した。あらためて本を手に取るもか、踊り子さんも微笑んでくれている。よーし買うぞぉ！もかは値段を確認してみた。本の裏表紙には¥5,800＋税と書かれていた。

「うわっ…ちょっと高い…けど！これはきっと運命なんだわ！」

興奮していたので、ちょっとだけ声が漏れてしまったようだ。周りの人に白い目で見られたが、もかはそんなことを気にはしていなかった。

■自室にて

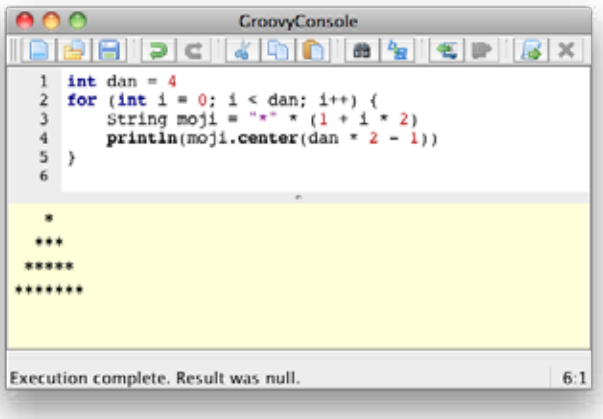
早速買って来た『Groovyイン・アクション』を片手にまずはGroovyのインストール作業を行った。これはP536を見ながらもかにも簡単にできた。その後、P15に書かれていたgroovyConsoleというもので、簡単にコード片を動かして試せることが分かったので、もかは今日の講義中に考えていたGroovyのコードを早速書いて実行させてみた。

```
int dan = 4
for (int i = 0; i < dan; i++) {
    String moji = "*" * (1 + i * 2)
    println(moji.center(dan * 2 - 1))
}
```

出来上がったコードはこれ。動かしてみたもかは感動した。

「わぁ～！ちゃんとピラミッドが出て来た！やった～！」

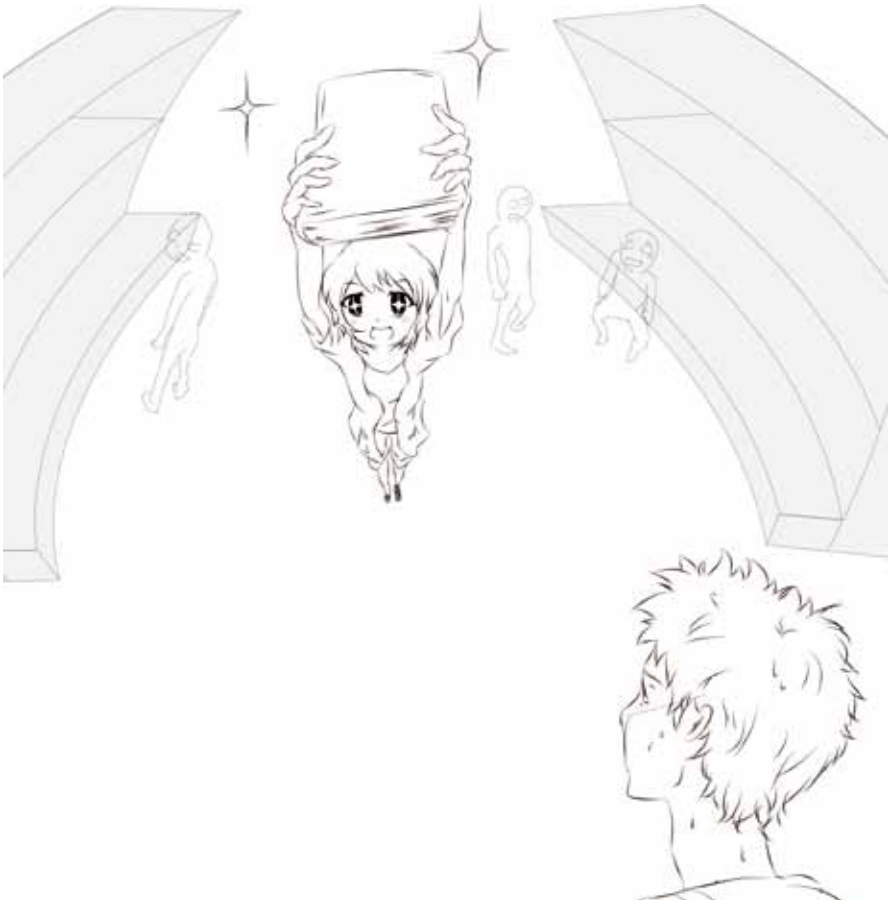
それにしてもすごいわ…たった5行で出来ちゃった。それに、すごく見やすいし、分かりやすい！



こうして、『Groovyイン・アクション』ともかは出会ったのであった。

その日、遅くまでもかの部屋の明かりは消えることが無かった。

（続く）



イラスト・けんや

Grails Plugin 探訪

第 3 回

~ MongoDB GORM プラグイン ~

URL: http://grails.org/plugin/mongodb
プラグインのバージョン: 1.0-M5
対応する Grails のバージョン: 1.3.5 以上

杉浦孝博

最近は 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。
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

今回紹介する Grails プラグインは、MongoDB GORM プラグインです。

このプラグインをインストールすることで、GORM API を使って Grails から MongoDB にアクセスできるようになります。

今回は、以下のバージョンの Grails、MongoDB で試しました。

- Grails : 1.3.7
- MongoDB : 1.8.0
- OS : MacOS X 10.6

MongoDB とは

MongoDB は、C++ で開発されたドキュメント指向のデータベースです。BSON とよばれる JSON ライクなバイナリドキュメントを管理します。

- 詳細は、MongoDB の公式サイト http://www.mongodb.org/ を参照してください。

MongoDB GORM プラグインのインストール

最初に、Grails アプリケーションのプロジェクトに MongoDB GORM プラグインをインストールします。

```
$grails install-plugin mongodb
```

設定

何の設定もしない場合、MongoDB が 27017 ポートで動作しているとみなし、Grails アプリケーションは動作します。

MongoDB の設定をしたい場合、grails-app/config/Config.groovy を編集します。

```
grails {
    mongo {
        host = "localhost"
        port = 27107
        username = "jggug"
        password = "jggugpass"
        databaseName = "jggugdb"
    }
}
```

DB 名は、指定しない場合、Grails アプリケーション名となります。

RDBMS と MongoDB を両方使いたい場合

Hibernate プラグインをアンインストールせず、MongoDB GORM プラグインと両方のプラグインがインストールされている場合、grails-app/domain ディレクトリ配下のドメインクラスは、Hibernate 経由で RDBMS に永続化されます。

特定のドメインクラスを MongoDB に永続化したい場合、ドメインクラスに mapWith プロパティで指定する必要があります。

```
static mapWith = "mongo"
```

あるいは、Hibernate のエンティティに追加された "mongo" というスコープを使用します。

```
// Hibernate 経由で RDBMS からインスタンスを取得
def hibernateBook = Book.get(1)
// RDBMS からインスタンスを MongoDB に保存
hibernateBook.mongo.save()
// MongoDB からインスタンスを取得
def mongoBook = Book.mongo.get(1)
```

MongoDBだけ使う場合やメインのDBとして使う場合

MongoDBだけ使う場合やメインのDBとして使う場合、デフォルトでインストールされているHibernateプラグインをアンインストールします。

```
$ grails uninstall-plugin hibernate
```

こうすることで、grails-app/domainディレクトリ配下のドメインクラスは、MongoDBに永続化されるようになります。

ただし、Hibernateプラグインをアンインストールし GORM APIを使う場合、NullPointerExceptionが発生することがあるかもしれません。

そのような場合は、ドメインクラスにmapWithプロパティを設定し明示的にMongoDBを使うことを宣言してください。

高度な設定

設定はConfig.groovyで行いますが、先程紹介した以外にも設定できる項目があります。

optionsブロックで、MongoDBとの接続設定を変更できます。

```
grails {
    mongo {
        options {
            autoConnectRetry = true
            connectTimeout = 300
        }
    }
}
```

また、本番環境ではマスター/スレーブ、レプリケーションといった形式、複数のMongoDBサーバーを動作させることがあると思います。

そのような設定も、Config.groovyで行います。

```
grails {
    mongo {
        replicaSet = [ "localhost:27017",
                       "localhost:27018"]
    }
}
```

簡単なサンプル

ここでは、MongoDBを使った簡単なサンプルを作ってみます。

■Grailsアプリケーションの作成

Grailsアプリケーションを作成します。ここでは、pluginmongodbという名前とします。

```
$ grails create-app pluginmongodb
$ cd pluginmongodb
```

■ドメインクラスの作成

ドメインクラスを作成します。本をあらわすBookドメインクラスと、出版社をあらわすPublisherドメインクラスを作成します。

```
$ grails create-domain-class Book
$ grails create-domain-class Publisher
```

Bookドメインクラスの内容は次のとおりです。

```
package pluginmongodb

class Book {
    String title
    Integer price
    Publisher publisher

    static constraints = {
    }

    static mapWith = "mongo"

    String toString() {
        title
    }
}
```

Publisherドメインクラスの内容は次のとおりです。

```
package pluginmongodb

class Publisher {
    String name

    static constraints = {
    }

    static mapWith = "mongo"

    String toString() {
        name
    }
}
```

■コントローラとビューのスカフォールド

generate-allコマンドで、コントローラとビューをスカフォールドします。

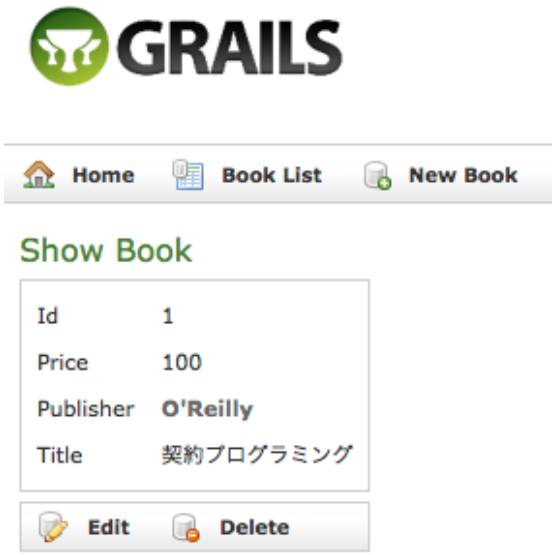
```
$ grails generate-all pluginmongodb.Book
$ grails generate-all pluginmongodb.Publisher
```

スカフォールドされるコントローラおよびビューは、GORM API(list, count, saveなど)が使われた通常のファイルとなります。

■アプリケーションの実行

アプリケーションを実行します。アプリケーションは、MongoDBに対しドメインクラスのインスタンスのCRUDを行います。

［一覧画面］



［詳細画面］

低水準なAPI

MongoDB GORMプラグインは、GORM APIだけでなく、低水準なAPIも提供しています。

Gmongoと呼ばれるMongo Java DriverのGroovyラッパーがあり、コレクションに簡単にアクセスできるように拡張されています。

「mongo」という名前でBeanが定義されていますので、コントローラやサービスにインジェクションできます。

```
class BookService {
    def mongo
    ....
}
```

getDB()メソッドでDBの参照を取得し、その後コレクションに対して操作を行います。

```
class BookService {
    def mongo
    def addBook(params) {
        def db = mongo.getDB("jggugdb")
        db.books.insert([title:params.title,
price:params.price, publisher:params.publisher])
    }
}
```

Config.groovyのgrails.mongo.databaseNameで定義したDB名に"DB"を追加した名前の、DBを参照するBeanも定義されます。

また、insert()メソッドの代わりに、左シフト演算子が使用できます。

上記の例は、以下のようになります。

```
class BookService {
    def jggugdbDB
    def addBook(params) {
        jggugdbDB.books << [title:params.title,
  price:params.price, publisher:params.publisher]
    }
}
```

トランザクションについて

MongoDBは直接トランザクションをサポートしていませんが、MongoDB GORMプラグインを使用することで、トランザクション処理を行うことができます。

トランザクションの指定は、次のとおりです。

1. サービスクラスにトランザクション指定をする
2. withTransaction()メソッドを使う

■サービスクラスにトランザクション指定をする

サービスクラスのtransactionalプロパティに'mongo'という値を設定することで、トランザクション処理が行われます。

```
class BookService {
    static transactional = 'mongo'
    ...
}
```

■withTransaction()メソッドを使う

ドメインクラスのwithTransaction()メソッドを使うことでも、トランザクション処理が行われます。部分的なトランザクション処理やコントローラでのトランザクション処理などに使用できます。

```
Book.withTransaction { status ->
    new Book(title:'Groovy in Action',
                                price:5000).save()
    throw new RuntimeException("bad")
    new Book(title:'Grails in Action',
price:5500).save()
}
```

おわりに

いろいろなNoSQL DBがあり、その一つとしてMongoDBが注目されています。GrailsアプリケーションからMongoDBにアクセスする場合、MongoDB GORMプラグインを使ってみてはいかがでしょうか。

リリース情報 2011.05.20

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: 1.2.5, 1.3.7

■更新情報

- 1.3.7では、Groovyが1.7.8に更新されたり、スタックトレースログに出力されるリクエストパラメータの値をマスキングしたり、といった改良が行われています。
- 1.3.7リリースノート: http://www.grails.org/1.3.7+Release+Notes

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: 1.7.10, 1.8.0

■更新情報

- 1.7.10では、Groovydocの子クラスでpackage-templates、doc-templates、class-templatesがオーバーライド可能になったり、いくつかのバグフィックスが行われています。
- 1.7.10リリースノート: http://jira.codehaus.org/secure/ReleaseNote.jspa?projectId=10242&version=17229
- 1.8.0では、Groovyの動的な表現力の向上や、JSONをネイティブサポート、GParsライブラリのバンドルなどが行われています。
- 1.8.0リリースノート: http://docs.codehaus.org/display/GROOVY/Groovy+1.8+release+notes

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: http://griffon.codehaus.org/
バージョン: 0.9.3-beata-1

■更新情報

- 0.9.3-beata-1では、新しいarchetypeが追加されたり、依存するGroovyのバージョンが1.8.0に、Gantのバージョンが1.9.5にバージョンアップされたり、いくつか改良されています。
- 0.9.3-beata-1リリースノート: http://griffon.codehaus.org/Griffon+0.9.3-beta-1

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: 1.9.5

■更新情報

- 1.9.5の変更箇所は不明です。
- 1.9.5リリースコメント: http://groovy.329449.n5.nabble.com/Gant-1-9-5-released-2011-05-03T09-30-01-00-td4366728.html

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: 0.9.2, 1.0-milestone-3

■更新情報

- 1.0-milestone-3では、Ivyリポジトリの定義が簡単になったり、新しいAPIが追加されたり、いくつかのバグフィックスが行われています。
- 1.0-milestone-3リリースノート: http://wiki.gradle.org/display/GRADLE/Gradle+1.0-milestone-3+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: 0.7

■更新情報

- 0.7では、Groovyが1.8.0に、GAE SDKが1.5にアップグレードされたり、Fileサービスがサポートされるようになりました。
- 0.7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: http://code.google.com/intl/ja/appengine/
バージョン: 1.5.0

■更新情報

- 1.5.0では、Backendがサポートされたり、Taskキューのpullモードがサポートされたり、いくつかのバグフィックスが行われています。
- 1.5.0リリースノート: http://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: 0.11GA、0.12-beta-1

■更新情報

- 0.12-beta-1では、構成可能な非同期機能や、Active Objectが追加されました。

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: 0.4.243

■更新情報

- 0.4.243での変更点は不明です。
- 0.4.243リリースメッセージ: http://groups.google.com/group/groovyplusplus/browse_thread/thread/52a7fb1dca43783d/632913b0785b02f8?show_docid=632913b0785b02f8

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: 0.5

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: 0.7

■更新情報

- 0.7では、groovyserverの起動時にCLASSPATH情報を表示したり(MacOS X, Linuxのみ)、-serverオプションを使用するようにしたり、バグフィックスがいくつか行われています。
- 0.7チェンジログ: http://kobo.github.com/groovyserv/changelog.html#0.7

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://geb.codehaus.org/
バージョン: 0.5.1

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: 0.9.8

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: 0.8.1

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: 0.5.1

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: 0.13

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: 0.3

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: https://github.com/andresteingress/gcontracts
バージョン: 1.2.3

▼ぐるーびーたん　第1話のあらすじ

2011年春、熊本からプログラマを目指して上京してきたぐるーびーたん。就職先のソフトウェア開発会社（㈱ジーアスターソフト）の扉を勢いよく開けて初出社！しかし、そこには寝袋にくるまった怪しい人影が。。。

※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。

G* Magazine vol.2 2011.05
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：㈱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

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