vol.3

Magazine

Japan Grails/Groovy User Group

今年もやります JGGUG合宿
今年は箱根に集合だっ！

by torazuka

by 合宿運営レンジャー

日本中からG*好きなエンジニアが集い、温泉宿に泊まり込みで、プログラミングに、LT大会に、そして懇親会に楽しい時を過ごします。開催概要は以下の通り：

日程：11/4(金)～11/6(日) ※一泊目はオプション
会場：箱根 美肌の宿 きのくにや http://hakone-kinokuniya.co.jp/
費用（概算）：一泊なら￥8,500+ α、二泊なら￥17,000+ α。
αは懇親会費（￥2,000前後？）
内容（予定）：
- LT大会
- 懇親会
- スペシャル企画
- 各自の自由テーマ など

JGGUGでは、一昨年、そして昨年秋と熱海で合宿を開催してきました。昨年はg100ponをいうスペシャル企画で、100個のプログラムの「お題」にリモート参加組を含めた16名で取り組み、合宿開催中だけで75%以上をクリア、たくさんの有用なサンプルコードを生み出しました。過去の合宿の様子は以下をご覧ください：

- JGGUG合宿2009レポート
  http://www.jggug.org/eventarchive/jggugcamp2009/report
- togetter: JGGUG合宿2010
  http://togetter.com/li/58838

今年は、原稿執筆時点ですでに15名の方（うち初参加の方5名を含む）が参加予定になっています。参加者は引き続き募集中ですので、興味のある方は下記のGoogleグループに参加の上、合宿参加の意思表示をお願いします：

- JGGUG合宿プロジェクト2011
  https://groups.google.com/group/jggugcamp2011

場所や日程があわず、残念ながら参加できないという方もご心配なく。今年もTwitterやUstreamなどをフル活用し、リモートからも楽しんでいただける体制を整える予定です。（リモートを含め）今年も多くのみなさまの参加をお待ちしています！

Contents

series 01

Grailsをコントロールせよ！ Part 2

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。

前号にて、筆者が別の記事で紹介したように、最新のマイルストーンリリースは、Grails 1.4.0系、改めGrails 2.0.0系がリリースされました。コントローラの部分にも若干の仕様変更・追加が行われました。この記事では、引き続きコントローラを解説しながら、なるべく部分的にでも将来のGrailsでの追加仕様についても紹介していきたいと思います。

Grails 2.0.0系に関しての詳しいまとめは、筆者のワークショップにて発表させていただいたスライドhttp://slidesha.re/q3tuH6、または、リリース情報をまとめたブログ記事 http://bit.ly/rms1In をご覧ください。（執筆時はGrails 2.0.0.M1 です）

注）この記事は、公式ドキュメントを意訳している筆者が、公式ドキュメントの内容を筆者目線で再編集（感想文!?）したような感じの構成になっています。一部、公式ドキュメント意訳と同じ内容を転載しているような部分もありますが、筆者が訳者なので致し方ないのです。ご理解とご了承お願いします。

ドメインとコントローラの更なる連携を！

前回はバインディングで単純なパラメータのバインドを行いました。おさらいとして、次のようなコードです。今回もこの記述が基本となります。

```
def user = new User(params)
//または
def user = new User()
user.properties = params
```

リレーショナルデータバインド

Grailsのデータバインディングでは、リレーショナルが存在するドメインクラスの場合でも、データバインディングを簡潔に実行できます。

データバインディングのコントローラでの記述は、基本的には先程のおさらいコードと全く同じになります。これもおさらいになりますが、データバインディングには、主に３つの方法があります。

1. ドメインクラスのコンストラクタにparamsを渡す。
2. ドメインクラスインスタンスのpropertiesプロパティにparamsを代入する。
3. コントローラのbindDataメソッドでドメインクラスインスタンスとparamsをバインドする。

同じと言うことは、今回は何を修得すれば良いのでしょう？

リレーショナルデータバインドでは、リレーショナル形式に対して決まったルールでリクエストパラメータを送信することでバインディングを行います。

この部分です。今回は主に、リクエストパラメータによって変わる、データバインディングの振る舞いを解説します。先ずは基礎ルールとなる部分の解説のために、単一終端関連（one-to-one、many-to-one）から見ていきましょう。

■ 多対１、１体１の連携

次のクラスを例とします。

```
class Book {
     Author author
}
class Author {
     ....
}
```

one-to-oneまたはmany-to-one等の単一終端関連のあるドメインでは、対象のリクエストパラメータに ".id" をサフィックスとして付加して指定します。次のように指定することで、データバインディング時に、値として与えられたidのAuthorクラスを検索します。また、リクエストパラメータに文字列で"null"と指定すると、対象の関連にはnullがセットされます。

```
//多対１、１対１の連携の指定。
/book/save?author.id=20
//  値に文字列でnullをあたえる。
/book/save?author.id=null

//データバインディング
def b = new Book(params)
```

ここで覚えておいて欲しいのが、Grailsでのリクエストパラメータのルールでは、Groovyでのクラスのプロパティ参照と同じように、リクエストパラメータのパラメータ名に".(ドット)"を使用してパラメータに値をあたえると、その対象を更新する事になります。

つまり、この例ではリクエストパラメータauthor.idを、ドメインクラスBookのプロパティauthorの該当クラスAuthorにあるプロパティidとして認識して、指定された値をセットしたという事になります。

このルールをもっと活用すると、さらに細かな事が可能になります。次のマルチドメインバインドで説明します。

■ マルチドメインバインド

先程のリクエストパラメータでの".（ドット）"ルールでは"id"だけを指定しましたが、同じ方法でパラメータを追加して、関連したドメインクラスの別のプロパティを更新することも可能です。

```
//リクエストパラメータ
/book/save?author.id=1&title=
                        TheBook&author.name=tyama
//コントローラアクションでの処理
def bookInstance = new Book(params)
```

この例では、新規Bookを作成する際に、author.id=1でAuthorを取得してBookに設定しつつ、author.name=tyamaで同時にAuthorのプロパティnameが更新されます。

更にこのルールを活用して、複数ドメインクラスをデータバインディングすることが可能です。

次のリクエストを例とします:

```
/book/save?book.title=TheStand&book.
                  pages=400&author.name=StephenKing
```

この場合は、それぞれリクエストパラメータが"book."、"author."で始まっています。もうわかると思いますが、接頭辞として付加した内容によって、データバインドを行うドメインクラスを区別できるようにしています。この内容で受け取ったパラメータは、多次元ハッシュマップであるparamsオブジェクトに、それぞれ区別されて登録されます。

リクエストパラメータの内容は、Grailsによって処理されて、paramsオブジェクト（ミュータブル多次元ハッシュマップ）へ登録されます。その中を見てみましょう。

```
//リクエストパラメータ
/book/save?book.title=Groovy&book.
pages=303&author.name=ksky
//paramsの内容
[
    book.title:Groovy,
    book:[title:Groovy, pages:303], //ここに注目
    author.name:ksky,
    author:[name:ksky], //ここに注目
    book.pages:303,
    action:save, controller:book
]
```

book:[title:Groovy, pages:303]とauthor:[name:ksky]のように、自動的にサブセットとして保持されているので、それらを対象のドメインクラスに渡せば良いと言うことになります。コントローラアクションでのコード記述は次のようになります:

```
def b = new Book(params.book)
def a = new Author(params.author)
```

コラム

リクエストパラメータのルールは、便利なのと同時に危険性も持ち合わせています。

この機能があることで、WebサイトがGrailsで開発されていて、且つフォームの内容を探りを入れて、パラメータが分かれば、関係の無いフォームからの関連ドメインの更新ができてしまいます。

ここでは説明のためGETメソッドの形で紹介をしていますが、先ず保存更新などをGETで実装することは無いと思うので（デフォルトでCRUDのsave等はPOSTです。）、URLから簡単にイタズラされることは無いでしょう。但しPOST等で攻撃されることもあると思うので設計全体での対策も必要です。

簡単な対策として、関連ドメインの更新の必要無いときは、単にparamsから必要なプロパティだけを選んで更新する方法を使用する:

```
book.properties['title','isbn'] = params
```

または、更新に関係の無い関連ドメインは、ドメインクラスのdiscard()メソッドで更新を制御する方法も考えられます。

```
//保存の前に
bookInstance.author.discard()
```

用途に応じて対策しましょう！

■ 複数終端関連 - 1体多、多対多の連携

Authorに複数のBookを持つ関連、次のドメインクラスを例とします。

```
class Author {
    static hasMany = [books: Book]
    String name
}
class Book {
    String title
    Publisher publisher
}
class Publisher {
    String name
}
```

Set型の関連ドメイン（hasManyでのデフォルト）の更新には、更新を行う対象のidを配列で送信します。次のタグでセレクトボックスを作成します。

```
<g:select name="books"
          from="${Book.list()}"
          size="5" multiple="multiple"
          optionKey="id"
          value="${author?.books}" />
```

このGSPタグで次のような複数セレクトボックスが生成されます。

リクエストパラメータから、paramsに収集された内容は次のようになります。books:[2,3]の部分がセレクトボックスからの内容になります。

```
//①複数選択した場合
[books:[2, 3], name:関谷和愛, create:Create,
               action:save, controller:author]
//②１つのみ選択した場合
[books:1, name:山本剛, create:Create,
               action:save, controller:author]
```

ここで気付くのが、選択された内容の件数の違いです。①の内容は配列にですが、②の内容は配列ではありません。Grailsでは、自動でデータバインディングする場合は、対象のプロパティがSet型などの場合は、自動的に配列として判別してくれます。

List型、Map型ベース関連の場合は、Setと違い順序とインデックスを指定することができるので、添字演算子を使用して関連ドメインのプロパティを更新・追加する事が可能です。

次のリクエスト例のように、リクエストパラメータに添字演算子を使用してフォームが送信されたとします。この例では、publisherとtitleを指定したBook用の値で新規にBookを作成しつつ新規にAuthorを作成しています。

```
//リクエスト例
/save?name=tyama&books[0].title=Grails%20
Ref&books[0].publisher.id=4
//paramsの内容
[
     books[0].publisher.id:4,
     //ここがBookに関連します。
     books[0]:[
          publisher.id:4,
          publisher:[id:4],
          title:Grails Ref
     ],
     name:tyama,
     books[0].title:Grails Ref
]
//コントローラアクション
def author = new Author(params)
author.save()
```

既存のデータに対して、更新を行うことも可能です。次の例では、関連ドメインBookのタイトルを変更します。

```
//リクエスト例
/?id=3&name=tyama&books[0].title=Grails Guide
//paramsの内容
[
     id:3,  // ←idを指定
     name:tyama,
     books[0].title:Grails Guide,
     books[0]:[
          title:Grails Guide // ←変更
     ]
]
//コントローラアクション
def author = Author.get(params.id)
author.properties = params
author.save()
```

関連ドメインの新規追加も可能です。この例では先の例のデータに対して、titleの内容を変更と新たにBookを追加します。対象のデータが持っている関連ドメイン（例ではbooks）の元の数よりも多いデータをバインドした場合は、自動的に新規追加されます。※但し関連ドメインの制約エラー等で追加・更新できない場合は無視されます（現状エラーメッセージも表示しません）。

```
//リクエスト例
/?id=3&name=tyama&books[0].title=
        Grails Guide 2&books[1].title=Grails 2.0
//paramsの内容（一部省略）
[
     id:3,
     //既存のtitleを更新
     books[0]:[title:Grails Guide 2],
     //追加するデータ
     books[1]:[title:Grails 2.0],
     name:tyama
]
//コントローラアクション
def author = Author.get(params.id)
author.properties = params
author.save()
```

あとはルールの応用になります。詳しくは公式ドキュメントを参照してください。

コマンドオブジェクト

コマンドオブジェクトはStrutsでのFormビーンと同じようなオブジェクトです。ドメインクラスのような永続化が必要無く、データバインディングとバリデーションが必要な場合に使用します。例えば、簡単なユーザ認証のフォームを想像してみてください。ユーザ認証では、アカウント、パスワード等のフィールドを必要として、それぞれフォームでのバリデーションとして、空文字列禁止（必須項目）や、必要で有れば文字数制限が必要な場合があります。

コマンドオブジェクトは、通常使用するコントローラと同じファイルに、クラス名がCommandで終わる名称で記述します。この例ではドメインクラスと同じように制約（constraints）も定義しています。（※ドメインクラスの詳細に関しては、今後執筆予定です。詳細は公式ドキュメントを参照してください。）

```
class UserController {
    ...
}
class LoginCommand {
    String username
    String password

    static constraints = {
        username(blank: false, minSize: 6)
        password(blank: false, minSize: 6)
    }
}
```

コントローラアクションの引数にコマンドオブジェクトを指定すると、コマンドオブジェクトにリクエストパラメータをバインドしたインスタンスを生成します。また、複数のコマンドオブジェクトを指定することも可能です。

```
class LoginController {
    def login = { LoginCommand cmd ->
        if (cmd.hasErrors()) {
            redirect(action: 'loginForm')
            return
        }

        // ここに処理。
    }
}
```

コマンドオブジェクトには依存注入(DI)も可能です。ドメインクラスやコントローラと同じように指定します。

```
class LoginCommand {
    def loginService //loginServiceビーンを依存注入

    String username
    String password

    static constraints = {
        username validator: { val, obj ->
            obj.loginService.
                        canLogin(obj.username,
                                 obj.password)
        }
    }
}
```

便利な共通処理と制御

コントローラには、データバインディングだけで無く、他にも、前後処理を行うインターセプタ、フォーム２重送信制御などコントローラで活用できる様々な制御の仕組みが存在します。これらの機能を活用して、セキュリティを高めたり、共通処理をスマートに記述する事が可能です。

■ デフォルトアクション

コントローラのルートURIにデフォルトURIを指定する方法です。スカッフォルド生成やcreate-controllerコマンドで生成されたコントローラは、コントローラのルートURIが、/indexになります。例としてBookControllerの場合、ルートURIは/bookでアクセスすると/book/indexの内容が返ります。この動作にはルールがあります。

- アクションが１つの場合は、そのアクションがデフォルトになります。
- indexという名称のアクションがある場合は、それがデフォルトになります。

・defaultActionを定義して指定できます。

```
static defaultAction = "list"
```

■ インターセプター

同一コントローラ内の、全て、あるいは指定した複数のアクションに、前処理と後処理を行うインターセプターを追加することができます。ユーザ認証やロギングなどに活用できます。インターセプターは、コントローラに、beforeInterceptorとafterInterceptorのプロパティで記述します。

● beforeInterceptor

beforeInterceptorでは、アクションが実行される前に処理を追加できます。beforeInterceptorに追加した処理で返値をfalseにすると、アクションは処理を行いません。beforeInterceptorには幾つかの記述方法があります。

コントローラ内全てのアクションに反映させるには、beforeInterceptorプロパティに、コードブロックを指定して処理を記述します。

```
def beforeInterceptor = {
    println "Tracing action ${actionUri}"
}
```

指定したアクション以外に処理を適用する場合は、実行するメソッドと、except条件を指定します。

コントローラ内のloginアクション以外の全ては、ユーザがログインしていないとloginアクションへ遷移する処理の例です。beforeInterceptorプロパティにマップの形式で、実行するメソッド名をaction:に指定、除外するアクション名をexcept:に指定しています。

```
def beforeInterceptor = [action: this.&auth,
        except: 'login']

private auth() {
    if (!session.user) {
        redirect(action: 'login')
        return false
    }
}

def login = {
    // ログインページ
}
```

この例と同じように、マップを指定する条件指定が他にもあります。

複数のアクションを指定:

```
def beforeInterceptor = [action: this.&auth,
        except: ['login', 'register']]
```

指定したアクションのみで実行、only:を使用します:

```
def beforeInterceptor = [action: this.&auth,
        only: ['secure']]
```

● afterInterceptor

afterInterceptorはアクションが実行された後に処理を追加します。afterInterceptorでは処理結果のモデルを引数として取得して、モデルやレスポンス（ModelAndView）を操作することができます。この例では返されたモデルの内容を元にビューを変更しています

```
def afterInterceptor = { model, modelAndView ->
    println "Current view is
                ${modelAndView.viewName}"
    if (model.someVar) modelAndView.viewName =
        "/mycontroller/someotherview"
    println "View is now
                ${modelAndView.viewName}"
}
```

■ メソッド制御

コントローラ内のアクションに対してHTTPメソッドの制御を行うことができます。allowedMethodsプロパティにマップ形式で、アクション名に対して許可をするHTTPメソッドを指定します。

```
static allowedMethods = [save: "POST",
        update: "POST", delete: "POST"]
```

■ フォーム２重送信制御

送信ボタンの２度押しなどで２重登録になってしまう等、何も制御しないと発生しますよね？実装方法は色々あると思いますが、Grailsでは、"同期トークンパターン"を使用した、フォーム二重送信ハンドリングに対応しています。

▼使用方法はGSPタグのフォームにuseToken属性を追加します。

```
<g:form useToken="true" ...>
```

コントローラアクションでフォーム２重送信制御用のwithFormメソッドを使用して、リクエストの制御処理を行います。

```
withForm {
   // 有効なリクエスト
}.invalidToken {
   // 無効なリクエスト
}
```

invalidTokenは省略することもできます。省略した場合は、無効のトークンをflash.invalidTokenに保持して元のページへリダイレクトします。元のページで、次の例のようにビューでトークンを使用します。

```
<g:if test="${flash.invalidToken}">
      ボタンが２度クリックされました。
</g:if>
```

（この例はあまり使わないと思いますが。。。）

まとめ

今回はここまでとします。ここまでのコントローラの解説では、コントローラでのリクエストの制御までを解説してきました。データバインディングや各種コントローラでの制御機能は、便利に活用できる反面、実装での使い方によっては、ルールに基づいた部分の危険性も出てきます。それぞれの機能を上手く使用して、便利な機能で効率よくセキュアなコントロールを心がけましょう。

さて次回は、コントローラ第３弾！「Grailsをコントロールせよ！ Part 3 - URLをコントロールせよ！」です。GrailsでのURLマッピング、フィルタ、RESTを解説します。それではまた次号で！

Griffon 不定期便 ～第 4 回 スレッド編 その 2～

series 02

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』

今回は簡単なTwitter検索アプリをGriffonで作成しながら、Griffonでのスレッドについて紹介していきます。

サンプルアプリケーションのソースコードは以下のサイトからダウンロードできます。

https://github.com/kiy0taka/twitterSearch

アプリケーションの作成

今回は最新版のバージョン0.9.3を使用します。インストールがまだの方はG*Magzine準備号の「Griffon不定期便 01」を参考にインストールして下さい。バージョンは異なりますが、同じ方法でインストールできます。

まずは、アプリケーションを作成しましょう。作成するアプリケーションは「TwitterSearch」という名前にします。

```
griffon create-app twitterSearch
```

今回は少しづつ作成しながら解説を進めていこうと思います。そのため何回もアプリケーションを起動しますが、毎回griffonコマンドを実行していると起動に時間がかかりますので、インタラクティブモードで実行することをお勧めします。インタラクティブモードの実行は次のコマンドで起動できます。

```
griffon interactive
```

作成したアプリケーションのディレクトリに移動してからインタラクティブモードで起動すると次のようにコマンドの入力を待つようになります。

```
$ griffon interactive
Welcome to Griffon 0.9.3 - http://griffon.codehaus.org/
Licensed under Apache Standard License 2.0
Griffon home is set to: /usr/local/griffon/current

Base Directory: /workspaces/griffon/twitterSearch
------------------------------------------------
Interactive mode ready. Enter a Griffon command
or type "exit" to quit interactive mode (hit
ENTER to run the last command):
```

試しにこの状態で「**run-app**」と入力してEnterキーを押してみましょう。最初のサンプルウインドウが立ち上がりましたが、普通に「**griffon run-app**」とするよりかは早く起動したと思いませんか？（思わなかった？）それでは今立ち上がっているウインドウを閉じてみましょう。するとまたコマンド入力を待機する状態になります。もう一度「**run-app**」を入力してみましょう。今度はさらに早く感じられたと思います。インタラクティブモードで「**run-app**」を2回実行しましたが、前回と同じコマンドを実行する場合はEnterキーの入力だけで前回のコマンドを実行できます。少し修正してからアプリケーションを起動して確認するにはインタラクティブモードで実行するのがお勧めです。

それではアプリケーションのView部分から実装していきましょう。検索用テキストフィールドと検索ボタン、そして検索結果をテーブルに表示するようにします。**griffon-app/views/twittersearch/TwitterSearchView.groovy** を次のように変更します。

```
package twittersearch

import static java.awt.BorderLayout.*

application(title: 'TwitterSearch',
  preferredSize: [350, 400],
  pack: true,
  locationByPlatform:true,
  iconImage: imageIcon('/griffon-icon-48x48.png').image,
  iconImages: [imageIcon('/griffon-icon-48x48.png').image,
               imageIcon('/griffon-icon-32x32.png').image,
               imageIcon('/griffon-icon-16x16.png').image])
{

    panel(constraints:NORTH) {
        textField columns:15
        button '検索'
    }
    scrollPane(constraints:CENTER) {
        table()
    }
}
```

図のようなウインドウになりました。レイアウトはこれで問題なさそうです。

少しだけアプリケーションに動きをつけてみたいと思います。まずは検索ボタンをクリックしたときに検索結果がテーブルに表示されるようにしてみましょう。ここではまだTwitterから検索せずにダミーデータを表示するようにします。

```
panel(constraints:NORTH) {
    textField columns:15
    button '検索', actionPerformed: {
        searchResult.rowsModel.value = [
            [fromUser:'kiy0taka',
              text:'Griffonなう。'],
            [fromUser:'kiy0taka',
              text:'@groovybook Gマガ読んだよ！']
        ]
        searchResult.fireTableDataChanged()
    }
}
scrollPane(constraints:CENTER) {
    table {
        tableModel(id:'searchResult') {
            propertyColumn header: 'User',
                           propertyName: 'fromUser'
            propertyColumn header: 'Tweet',
                           propertyName: 'text'
        }
    }
}
```

検索ボタンのイベント処理と、テーブルモデルを追加しました。これを実行して、検索ボタンをクリックするとテーブルにデータが表示されます。

なんとなくアプリケーションの動作がわかってきたので、検索ボタンのイベント処理をコントローラに移しましょう。**griffon-app/controllers/twitersearch/TwitterSearchController.groovy** を次のように修正します。

```
package twittersearch

class TwitterSearchController {

    def model
    def view

    def search = {
        view.searchResult.rowsModel.value = [
            [fromUser:'kiy0taka',
               text:'Griffonなう。'],
            [fromUser:'kiy0taka',
               text:'@groovybook Gマガ読んだよ！']
        ]
        view.searchResult.fireTableDataChanged()
    }
}
```

Viewにあったイベント処理をControllerの **search** クロージャに移動しました。Controller内ではViewのコンポーネントにアクセスするために「**searchResult**」を「**view.searchResult**」に変更しています。

Viewの方ではイベント処理をコントローラに委譲するように検索ボタンを次のように変更します。

```
button '検索', actionPerformed:
controller.&search
```

それでは次にModelを変更しましょう。テキストフィールドの値と検索結果に表示する一覧のデータをModelに定義します。**griffon-app/models/twittersearch/TwitterSearchModel.groovy**を次のように修正します。

```
package twittersearch

import groovy.beans.Bindable

class TwitterSearchModel {
    @Bindable String searchText
    @Bindable List tweets
}
```

Modelの値をViewにバインドするようにViewを変更します。バインディングについてはG*Magazine創刊号の「Griffon不定期便」を参照ください。テキストフィールドの値をModelにバインドし、テキストフィールドが入力されている場合にのみ検索ボタンを有効化します。テーブルに表示するデータもModelからバインドするようにします。

```
import groovy.model.ValueHolder
...
panel(constraints:NORTH) {
    textField columns:15,
        text: bind(target:model, 'searchText')
    button '検索', actionPerformed:
        controller.&search,
        enabled: bind{model.searchText}
}
def rowsModel = new ValueHolder()
scrollPane(constraints:CENTER) {
    table {
        tableModel(id:'searchResult',
                rowsModel:rowsModel) {
            propertyColumn header: 'User',
                propertyName: 'fromUser'
            propertyColumn header: 'Tweet',
                propertyName: 'text'
        }
    }
}
bean rowsModel, value:bind{model.tweets}
```

Controllerの方も少し変えましょう。検索結果を直接Viewに設定するのではなく、Modelに設定するようにします。

```
def search = {
    model.tweets = [
        [fromUser:'kiy0taka',
               text:'Griffonなう。'],
        [fromUser:'kiy0taka',
               text:'@groovybook Gマガ読んだよ！']
    ]
    view.searchResult.fireTableDataChanged()
}
```

それでは、実際にTwitterから検索するようにしてみましょう。Twitter4Jのライブラリの設定が必要ですが、GriffonにはTwitterプラグインがあるので今回はプラグインをインストールして使ってみましょう。Twitterプラグインのインストールは次のコマンドでインストールできます。

```
install-plugin twitter
```

Twitter4JのAPIを利用してTwitterから検索するようにControllerを変更します。

```
package twittersearch

import twitter4j.*

class TwitterSearchController {

    def model
    def view
    def twitter

    def search = {
        model.tweets = twitter.
                search(new Query(model.
                        searchText)).tweets
        view.searchResult.fireTableDataChanged()
    }
}
```

Controllerで **twitter** という名前の変数を定義しておくとTwitterプラグインによって **twitter4j.Twitter** 型のオブジェクトが設定されます。Twitterプラグインは他にも色んな機能がありますが、今回は特に使用しないので解説はしません。興味のある方は調べてみてください。

Griffonでのスレッドの扱い

さて、ここまででとりあえずのアプリケーションが実装できました。ここからがやっと本題です。ここまでで出来たアプリケーションは一見正しく実装しているように見えますが、まずい実装になっている箇所があります。

Griffonではバージョン0.9.2からコントローラに定義されているクロージャはすべてバックグラウンドスレッド上で実行されるようになりました。作成したアプリケーションのコントローラでは特に意識せず、ModelやViewを変更しました。しかし、これはまずい実装です。ViewやModelはイベントディスパッチスレッド上で変更する必要があります。Griffon 0.9.2以前のバージョンではコントローラのアクションはすべてイベントディスパッチスレッド上で呼び出されたのでこの実装でもあまり問題はありません。（後述しますが、少し問題は残っています。）この問題を解決するためにはいくつか方法があります。

■ コントローラのアクションをすべてEDT上で実行されるようにする

0.9.2からコントローラのアクションがすべてバックグラウンドスレッド上で実行されますが、0.9.2以前のバージョンと同じようにすべてのコントローラのアクションをEDT上で実行できるようにします。以下のいずれかの方法で「**griffon.disable.threading.injection=true**」を設定しておけばEDT上で実行されるようになります。

- コンパイル時にシステムプロパティとして指定する
  以下のようにシステムプロパティを指定してコンパイルします。

```
griffon -Dgriffon.disable.threading.
injection=true compile
```

  アプリケーション内のすべてのコントローラのアクションがEDT上で実行されます。
- **griffon-app/conf/BuildConfig.groovy** に指定する
  アプリケーション内のすべてのコントローラのアクションがEDT上で実行されます。
- **${USER_HOME}/.griffon/settings.groovy** に指定する
  settings.groovy がなければ新しく作成し、「**griffon.disable.threading.injection=true**」を追記します。この方法ではすべてのアプリケーションのコントローラのアクションがEDT上で実行されるようになります。

これらの方法は0.9.2以前のバージョンと互換性を保つために用意されている仕組みを利用します。0.9.2以前のバージョンを0.9.2以上のバージョンにアップグレードする場合はこの方法がいいかもしれません。しかし、0.9.2移行のバージョンではバックグラウンドスレッド上で実行するのが推奨されるためこれらの方法はあまりおすすめしません。

■ アノテーションで実行するスレッドを指定する

コントローラのクロージャに**@Threading**アノテーションを指定することで、実行するスレッドを指定することができます。次のように指定するとアクションがEDT上で実行されます。

```
import griffon.transform.Threading

class MyController {

    @Threading(Threading.Policy.INSIDE_UITHREAD_SYNC)
    def someAction = {
        ...
    }
}
```

アノテーションに **Threading.Policy** を指定することで実行するスレッドを指定できます。指定可能な **Threading.Policy** の値は次のとおりです。

- **OUTSIDE_UITHREAD**
  バックグラウンドスレッド上で実行されます。**Threading.Policy** を指定しない場合のデフォルト値となります。
- **INSIDE_UITHREAD_SYNC**
  アクションがEDT上で実行され、処理が終わるのを待ちます。**SwingUtilities#invokeAndWait()** に相当します。
- **INSIDE_UITHREAD_ASYNC**
  アクションがEDT上で実行されるようにスケジュールされます。**SwingUtilities#invokeLater()** に相当します。
- **SKIP**
  スレッドの管理を行いません。

■ Griffonが提供するメソッドを利用する

前回紹介した、**SwingBuilder** の **edt()**、**doLater()**、**doOutside()** と同じようなメソッドがGriffonでも定義されています。実装はGriffon独自になっていますが、**SwingBuilder** のメソッドとの対応は以下のとおりです。

Griffon	SwingBuilder
execSync()	edt()
execAsync()	doLater()
execOutside()	doOutside()

執筆時点の最新版のソースコード（Groovy 1.8.1、Griffon 0.9.3）を見ると、**doOutside()** は新しいスレッドを生成して実行するのに対し、**execOutside()** はスレッドプールを利用しているなどの違いはありますが、ほぼ同等の機能となっています。これらのメソッドを利用して今回のアプリケーションを正しく実装してみましょう。

何が問題？

正しく実装してみましょうといいましたが、今の実装は何が正しくないのでしょうか？一見問題なさそうにも見えますし、実行してみてもまずそうな動きはしていないように見えます。しかし、この実装には問題があります。Swingのスレッドポリシーを思い出してみましょう。

> *通常、Swing はスレッドに対して安全ではありません。すべての Swing コンポーネントと関連クラスには、特に説明がないかぎり、イベントディスパッチスレッド上でアクセスしてください。*

今のコントローラの **search** アクションはバックグランドスレッド上で実行されるようになっています。モデルのプロパティを変更し、Viewに対しても処理をしていますが、これれはすべてバックグラウンドスレッド上で実行されてしまうため、スレッドセーフではありません。もっと複雑な処理をした場合、表示が壊れてしまう可能性があります。

それでは、正しい実装に変えてみましょう。Twitter検索をする処理はネットワークの状態や、Twitterの負荷状況によって処理時間が予想以上にかかってしまう可能性があります。このような時間のかかる可能性のある処理はEDT上で実行するのは好ましくないため、バックグラウンドスレッド上で実行する方が良いでしょう。しかし、検索結果をモデルに設定してしまうとバインディング機能によってViewが変更されてしまうため戻り値を直接モデルに設定するのは良くありません。一度ローカル変数に保持するように変更しましょう。

```
def tweets = twitter.search(new Query(model.
searchText)).tweets
model.tweets = tweets
view.searchResult.fireTableDataChanged()
```

さらに、モデルとViewに対する処理をEDT上で行うようにします。

```
def tweets =
  twitter.search(new Query(model.searchText)).tweets
execAsync {
        model.tweets = tweets
        view.searchResult.fireTableDataChanged()
}
```

これで、めでたくスレッドセーフな実装になりました。

アプリケーションを更に改良

アプリケーションがスレッドセーフになり、問題はなくなりました。しかし、ユーザインターフェースとして不親切な点があります。検索ボタンをクリックしてから、検索結果が表示されるまでの間、何も変化がないように感じられます。ユーザは本当に検索をしているのか不安になり、何回も検索ボタンをクリックしてしまうかもしれません。検索中はプログレスバーを表示し、検索ボタンをクリックできないように変更してみましょう。

まず、検索中であることを示すプロパティをモデルに追加します。

```
class TwitterSearchModel {
    @Bindable String searchText
    @Bindable List tweets
    @Bindable boolean searching
}
```

次に、コントローラでsearchingプロパティを設定するようにします。

```
def search = {
    execSync {
        model.searching = true
    }
    def tweets =
        twitter.search(new Query(model.searchText)).tweets
    execAsync {
        model.tweets = tweets
        view.searchResult.fireTableDataChanged()
        model.searching = false
    }
}
```

searching プロパティの変更もEDT上で行います。また、プロパティの値を変更してから検索するためにここでは **execSync()** を利用しています。

検索が終わってViewに反映したら **searching** プロパティを **false** に戻しておきます。

最後にViewにプログレスバーを追加し、**searching** プロパティで表示を制御しましょう。

```
panel(constraints:NORTH) {
   textField columns:15,
   text:bind(target:model, 'searchText'),
   enabled:bind{!model.searching}
   button '検索',
   actionPerformed:controller.&search,
   enabled:bind{model.searchText &&
    !model.searching}
}
def rowsModel = new ValueHolder()
scrollPane(constraints:CENTER) {
   table {
       tableModel(id:'searchResult',
                rowsModel:rowsModel) {
           propertyColumn header: 'User',
                propertyName: 'fromUser'
           propertyColumn header: 'Tweet',
                propertyName: 'text'
       }
   }
}
bean rowsModel, value:bind{model.tweets}
progressBar constraints:SOUTH,
visible:bind{model.searching},
indeterminate:bind{model.searching}
```

プログレスバーの表示切替だけではなく、検索条件の編集付加と検索ボタンの無効化も制御しました。それではアプリケーションを実行してみましょう。

検索中はプログレスバーが表示されれるようになり、ユーザに優しいインタフェースになりましたね。実はこのアプリケーションはまだ不完全です。エラー処理を実装していません。検索時にエラーが発生しても、ユーザには通知されずプログレスバーが動いたままになってしまいます。今回はエラー処理の実装と解説はしません。興味のある方はぜひ実装してみてください。

まとめ

前回と今回でSwingアプリケーションのスレッドポリシーと、それに基づいた **SwingBuilder** とGriffonの使い方を紹介しました。普段Webアプリケーションを実装されている方には少しわかりにくい解説だったかもしれませんが、スレッドの使い方をきちんと理解しておけば、Swingアプリケーションも正しく実装することができます。それではまたお会いしましょう。

series 03

CodeNarcを利用してGROOVYのコード品質を上げる ～第3回 ルールのカスタマイズ～

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

静的コード解析はルールに基づいてソースコードを解析することでコードの問題を指摘することができる。

CodeNarcは、標準で264（2011年8月現在）のルールをもっている。標準ルールは、http://codenarc.sourceforge.net/codenarc-rule-index.htmlから参照可能である。

今回は、静的コード解析の心臓部ともいえるルールのカスタマイズ方法を解説する。

ルールのカスタマイズ概要

ルールカスタマイズは、以下の2つの方法がある(バージョン0.15ではカスタマイズ方法が追加されたようだが詳細はまだ調査中のため今回はバージョン0.13時点の情報を元に記述)。

- 標準ルールをカスタマイズする方法
- 新しいルールを記述する（標準ルールに存在しないルールを記述する）方法

■標準ルールのカスタマイズ

標準ルールのプロパティを変更することによってルールをカスタマイズする方法である。標準ルールのプロパティ値をXMLファイルで指定することによってカスタマイズする。

以下の例は、クラスの大きさをチェックするルールの閾値を500にカスタマイズしている例である。

各ルールが持っているプロパティは、http://codenarc.sourceforge.net/codenarc-rule-index.htmlで確認することが可能である。

```
<ruleset xmlns="http://codenarc.org/ruleset/1.0"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation=
         "http://codenarc.org/ruleset/1.0
          http://codenarc.org/ruleset-schema.xsd"
       xsi:noNamespaceSchemaLocation=
         "http://codenarc.org/ruleset-schema.xsd">

    <description>Class Size Check Rule</description>

     <ruleset-ref path='rulesets/size.xml'>
         <rule-config name='ClassSize'>
             <property name='maxLines' value='500'/>
         </rule-config>
     </ruleset-ref>

</ruleset>
```

■新しいルールの記述

新しいルールはCodeNarcのソースコードを変更することで実現する。コードスニペットを生成するスクリプトが用意されているので比較的容易に実現することが可能である。

今回は、tryブロックが空である場合に警告を発するルールを例として記述方法の流れを説明する。

●ソースコードのチェックアウト

ソースコードを変更することになるのでCodeNarcのSubversionからソースコードをチェックアウトする。以下はチェックアウトコマンドの例である。

```
svn co https://codenarc.svn.sourceforge.net/
                        svnroot/codenarc codenarc
```

●ソースコードのビルド

チェックアウトしたソースコードのルートへ移動しソースコードをビルドする。CodeNarcはMavenで管理されているためpom.xmlがあるディレクトリで以下のコマンドを実行するとビルドできる。

```
mvn install
```

●ルールのコードスニペットを作成

ルートディレクトリのcodenarc.groovyスクリプトを実行する。実行後、ルール名等の入力を促されるので入力する。

```
groovy codenarc.groovy create-rule
Enter your name:Kentaro Arai
Enter the rule name:EmptyTryBlock
Enter the rule category. Valid categories are:
  basic
  braces
  concurrency
  design
  dry
  exceptions
  formatting
  generic
  grails
  imports
  jdbc
  junit
  logging
  naming
  security
  serialization
  size
  unnecessary
  unused
exceptions
Enter the rule description:There is empty try block
    Created ./src/main/groovy/org/codenarc/rule/exceptions/
                            EmptyTryBlockRule.groovy
    Created ./src/test/groovy/org/codenarc/rule/exceptions/
                            EmptyTryBlockRuleTest.groovy
    Updated ./src/main/resources/codenarc-base-messages.properties
    Updated ./src/main/resources/rulesets/exceptions.xml
    Updated ./src/site/apt/codenarc-rules-exceptions.apt
    Updated ./CHANGELOG.txt
    adding to svn... A   src/main/groovy/org/codenarc/rule/
                            exceptions/EmptyTryBlockRule.groovy
    adding to svn... A   src/test/groovy/org/codenarc/rule/
                    exceptions/EmptyTryBlockRuleTest.groovy
     Finished
```

●プロパティファイルの編集

作成された./src/main/resources/codenarc-base-messages.propertiesを編集する。

このファイルは結果の表示順をコントロールするファイルである。ファイルの先頭に追加された以下の部分を任意の場所(希望する表示位置)に移動する。

```
# todo: manually sort your messages into the
correct location
EmptyTryBlock.description=There is empty try block
EmptyTryBlock.description.html=There is empty try block
```

●ルールファイルの編集

作成された./src/main/groovy/org/codenarc/rule/exceptions/AvoidPrintStackTraceRule.groovyを編集する。

ルールの実装は、AbstractRuleを継承する方法とAbstractAstVisitorRuleを継承する方法の2つがある。

今回は、Groovy ASTを利用して実装されているAbstractAstVisitorクラスを利用して実装した。詳細は、http://codenarc.sourceforge.net/apidocs/index.htmlを参照のこと

```
import org.codenarc.rule.AbstractAstVisitor
import org.codenarc.rule.AbstractAstVisitorRule
import org.codehaus.groovy.ast.stmt.
TryCatchStatement
import org.codenarc.util.AstUtil
/**
 * There is empty try block
 *
 * @author Kentaro Arai
 */
class EmptyTryBlockRule extends
AbstractAstVisitorRule {
    String name = 'EmptyTryBlock'
    int priority = 2
    Class astVisitorClass =
EmptyTryBlockAstVisitor
}

class EmptyTryBlockAstVisitor extends
AbstractAstVisitor {
   void visitTryCatchFinally(TryCatchStatement tryCatchStatement) {
      if (AstUtil.isEmptyBlock(tryCatchStatement.tryStatement)) {
        addViolation(tryCatchStatement)
      }
        super.visitTryCatchFinally(tryCatchStatement)
    }
}
```

●ビルドして実行

再度、ソースコードをビルドしてCodeNarcを実行するとtryブロック内が空の場合に警告があがってくる。

●その他

今回はスニペットとして作成されたテストコードの編集は行わなかったが、テストコードを作成するための便利なユーティリティがorg.codenarc.rule.AbstractRuleTestCaseに用意されている。詳細は、http://codenarc.sourceforge.net/apidocs/index.htmlを参照のこと。

■まとめ

今回はCodeNarcのルールを拡張する方法を2つ紹介した。標準ルールのカスタマイズはCodeNarcを実プロジェクトで利用する際に自社の基準と照らし合わせて利用して頂きたい。また、新しいルールを記述することは障害等が発生した場合のパターンをノウハウ(形式知化)とする場合に非常に威力を発揮する。慣れるまでは記述が煩雑であるがそれ以上の価値があるので是非実施して頂きたい。

今回でCodeNarcを利用した静的コード解析の連載は最終回である。私は、開発プロセスの初期から適用できるコード品質向上策として静的コード解析は非常に有効な手段だと考えている。この連載がコード品質向上の一助になることを願っている。

series 04

GebではじめるWebテスト ～第2回 Geb0.6 編～

綿引 琢磨（わたびき　たくま）
Groovy/Grails/Griffon/Java 界隈のやさぐれエンジニア。
日本 Grails/Groovy ユーザーグループ運営委員。

G*な皆さん、こんにちは。今回はvol.1で予告したGebのアーキテクチャを全編にわたって紹介する予定でしたが、予定を変更して6/28にリリースされたバージョン0.6での新機能と変更点を中心に、より強力になったGebの魅力についてお伝えしたいと思います。

Geb0.6の概要

Geb0.6では多くの機能が追加されただけでなく、既存機能に対しても再実装がなされるなど、大幅な変更が行われました。以下はマニュアルに記載されている新機能、変更点リストから一部を抜粋したものです。

- 設定の仕組み
- レポート機能の主要ドライバーへの対応
- 外部スクリプトのバインディング
- ダイレクトダウンロードAPI
- タイムアウト待ち時間、リトライ回数の設定機能
- ビルドシステムやフレームワークとの統合
- Navigator, Moduleクラスに大きさや位置を取得するための属性追加
- TestNGのサポート
- Browser#drive()インタフェースの変更
- Grails用のテストクラスの削除

これら以外にもクラスの振る舞いが変わったものが多数あります。詳細はGebのマニュアルを参照してください。

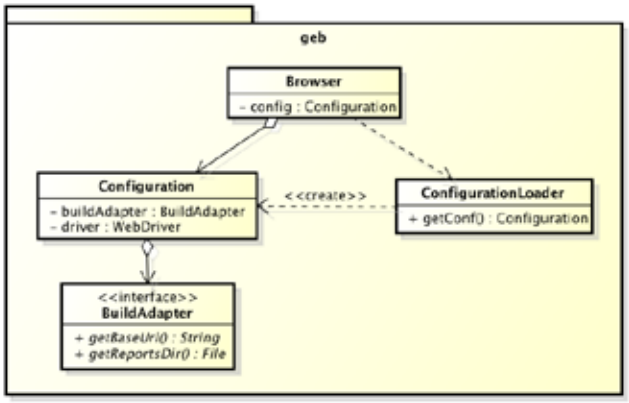
では、前述の中から特に注目したい新機能と重要な変更点について見ていきましょう。

注目の新機能

今回のバージョンアップで追加された機能には、新たな仕組みを導入して実現したものと既存機能を拡充して実現したものとがあります。本記事では新たに追加された設定の仕組みと、既存機能を拡充したレポート機能、外部スクリプトのバインディングについて解説します。

■ 設定の仕組み

Geb0.6では新たに設定の仕組みが導入されました。これまではベースとなるURLや使用するWebDriverなどは、Browserクラスの生成時または実行時に指定するようになっていましたが、今回追加された設定も含め、Gebの動作に関わる設定についてはConfigurationクラスおよびConfigurationLoaderクラスを通じて行うようになりました。これにより、Gebの中心のクラスであるBrowserクラスのコンストラクタやメソッドのインタフェースも変更されています。設定に関連するクラスは次のような構成になっています。



設定はテストコード内に直接記述するか、もしくは外部で指定することで行います。

●テストコード内での設定

テストコード内で設定する場合は、Configurationクラスを生成してプロパティの設定を行い、そのインスタンスをBrowserクラスのコンストラクタもしくはBrowser#drive()メソッドの引数として渡します。

▼NewGebTest.groovy

```
package hello

import org.junit.Test
import geb.Browser
import geb.Configuration;
import pages.*

class NewGebTest {

    @Test
    void helloJGGUG() {

        // 設定
        Configuration conf = new Configuration()
        // 使用するWebDriverの指定
        conf.setDriverConf('firefox')
        // レポート出力ディレクトリの指定
        conf.setReportsDir(new File('c:/temp'))

        Browser.drive(conf) {
            to HelloPage
            assert at(HelloPage)
            report 'hello_page'

            username = 'JGGUG'
            greetButton.click()
            assert at(GreetingPage)
            report 'greeting_page'

            assert h1Text == 'Hello, JGGUG!'
        }
    }
}
```

●外部からの設定

外部から設定を行う方法としては、設定スクリプト、システムプロパティ、BuildAdapterクラスの実装の３つの方法があります。BuildAdapterクラスの実装による設定は、現状ではベースURLとレポート出力ディレクトリしか指定することはできません。ここでは標準的な設定方法である設定スクリプトによる指定方法について説明します。

設定スクリプトは明示的にパスを指定してクラスローダーから読み込ませることもできますが、通常はデフォルトパッケージ上にGebConfig.groovyという名前のConfigSlurperスクリプトを用意して自動的に読み込ませます。Grailsで使用する場合にはtest/functionalの直下に、MavenやGradleなどのビルドツールから使用する場合にはtest/resourcesの直下に配置すれば良いでしょう。次の設定スクリプトはGrailsでの機能テスト用のスクリプトです。

▼GebConfig.groovy

```
import org.openqa.selenium.htmlunit.HtmlUnitDriver
import org.openqa.selenium.firefox.FirefoxDriver
import org.openqa.selenium.chrome.ChromeDriver

driver = {
    def driver = new HtmlUnitDriver()
    driver.javascriptEnabled = true
    driver
}

environments {

    // grails -Dgeb.env=chrome test-app
    chrome {
        driver = { new ChromeDriver() }
    }

    // grails -Dgeb.env=firefox test-app
    firefox {
        driver = { new FirefoxDriver() }
    }
}
```

この設定ではデフォルトではHtmlUnitDriverが使用されます。実行時にgeb.envプロパティを指定することで、ChromeDriverやFirefoxDriverを使用できますので、必要に応じて柔軟にWebDriverを変更することができます。

■レポート機能

前バージョンまでは、統合するテストフレークワークごとに用意されたレポート出力用のテストクラスを継承し、さらに使用するWebDriverがFirefoxWebDriverの場合でしかレポート機能を利用することができませんでした。今回のバージョンからは、専用のテストクラスを継承することなく、Browserクラスから直接レポート出力ができるようになり、出力できるWebDriverにも制約はなくなりました。

また、画面のスクリーンショットに関してはTakesScreenshotインタフェースを実装する全てのWebDriverで出力されるようになっています。Firefox, Chrome, IE などの標準的なWebDriverは上記のインタフェースを実装していますので、システム要件で使用ブラウザが制限されている場合でもレポート機能を利用することができます。

レポートを出力するためには、出力先のディレクトリを設定しておく必要があります。設定は前述のConfigurationクラス、システムプロパティ、またはBuildAdapterクラスの実装によって行います。

▼Configurationクラスによる設定

```
def conf = new Configuration()
conf.setReportsDir(new File('c:/temp'))
```

▼システムプロパティによる設定

```
System.setProperty('geb.build.reportsDir',
                                        'c:/temp')
```

▼CustomBuildAdapter.groovy

```
class CustomBuildAdapter implements BuildAdapter
{
    File getReportsDir() {
        new File('c:/temp')
    }
}
```

レポートはBrowser#report()メソッドで出力します。引数の文字列が出力ファイル名となり、デフォルトでは同名のhtmlファイルとpngファイルが出力されます。

```
// hello_page.htmlとhello_page.pngが出力される
report 'hello_page'
```

さらに、新機能としてレポートグループディレクトリの指定が可能になっています。レポートグループディレクトリとは、出力先ディレクトリのサブディレクトリとして作成されるディレクトリのことで、出力前にグループディレクトリを指定しておくと、そのディレクトリを作成し、レポートを出力します。

グループディレクトリをテストコード内で適時指定することによって、テストごとや画面単位で出力先を変更できますので、エビデンスの管理がしやすくなるのではないかと思います。

```
Configuration conf = new Configuration()
conf.setDriverConf('firefox')

// レポート出力ディレクトリの指定
conf.setReportsDir(new File('c:/temp'))

Browser.drive(conf) {
    to HelloPage
    assert at(HelloPage)

    // c:/temp/helloJGGUG-1ディレクトリに出力
    reportGroup 'helloJGGUG-1'
    report 'hello_page'

    username = 'JGGUG'
    greetButton.click()
    assert at(GreetingPage)

    // c:/temp/helloJGGUG-2ディレクトリに出力
    reportGroup 'helloJGGUG-2'
    report 'greeting_page'

    assert h1Text == 'Hello, JGGUG!'
}
```

■外部スクリプトのバインディング

もう一つ注目すべき新機能として、外部スクリプトのバインディングを取り上げます。これはEasyB Gebプラグインで実装されていた仕組みを汎用的に使用できるよう抽出した機能です。外部スクリプトとは、スクリプトとして外部に切り出されたテストコードをのことを指します。つまり、物理的に別で定義されているテストコードを、実行時にBrowserクラスとバインディングさせて、テストを実施することができるのです。バインディングを利用したテストは次のようになります。

▼GebConfig.groovy

```
driver = "firefox"
reportsDir = "c:/temp"
```

▼ScriptBindingTest.groovy

```
package hello

import org.junit.After
import org.junit.Before
import org.junit.Test
import geb.Browser
import geb.binding.BindingUpdater
import groovy.lang.Binding

class ScriptBindingTest {
    def binding = new Binding()
    def browser = new Browser()
    def updater = new BindingUpdater(binding, browser)

    @Before
    void setUp() {
        updater.initialize()
    }

    @Test
    void helloJGGUGFromScenario() {
        new GroovyShell(binding).evaluate(new File(
                "src/test/features/Scenario.groovy"))
    }

    @After
    void tearDown() {
        updater.remove()
    }
}
```

▼Scenario.groovy

```
import pages.*

to HelloPage
assert at(HelloPage)
reportGroup 'helloJGGUG-1'
report 'hello_page'

page.username = 'JGGUG'
page.greetButton.click()
assert at(GreetingPage)
reportGroup 'helloJGGUG-2'
report 'greeting_page'

assert page.h1Text == 'Hello, JGGUG!'
```

この仕組みを利用することによって、これまでは不可能だったCuke4Duke（JVM向けのCucumber）との統合も可能となりました。これは、単にGebをUATに利用する際のテストフレームワークの選択肢が増えただけでなく、さらに柔軟な形でテストコードを記述できる可能性を得たということになります。

重要な変更点

今回のバージョンアップでは機能が増えたことに伴い、インタフェースの見直しや振る舞いが変更されたメソッドが多くあります。また依存するライブラリのバージョンも変更になっていますので、重要なものに絞ってお伝えします。

■ 依存ライブラリのバージョン

GroovyとWebDriverの最小バージョンがそれぞれ引き上げられました。

- Groovy 1.7以上
- WebDriver 2.0rc1

Groovyは最新の1.8.1でも特に問題もなく安定して動作していますので、最新バージョンにしておいて問題ないと思いますが、WebDriverについてはブラウザのバージョンとの兼ね合いもあり、場合によっては使い分けが必要になるかもしれません。筆者のWindows環境ではInternetExplorerDriverを使用したテストで終了時にJVMの致命的エラーが発生する事象も起きていますので、問題が発生した場合にはブラウザを変更したり、使用するWebDriverのバージョンを変更するなどの対応をおすすめします。

■ インタフェースの見直し

前述の通り、Browserクラスではインタフェースが大きく変わりました。特にGebのエントリーポイントでもあるdrive()メソッドでは、Closureを引数とするメソッド以外は変更されています

ので、注意してください。追加されたAPIと削除されたAPIは次の通りです。

▼Geb0.6から追加されたBrowserクラスのAPI

```
// コンストラクタ
Browser() {}
Browser(Configuration config) {}
Browser(Map props, Configuration config) {}

// drive()メソッド
static Browser drive(Configuration conf,
                     Closure script) {}
static Browser drive(Map browserProperties,
                     Closure script) {}
static Browser drive(Browser browser,
                     Closure script) {}
```

▼Geb0.6から削除されたBrowserクラスのAPI

```
// コンストラクタ
Browser(Class pageClass, Map params = null) {}
Browser(String baseUrl, Class pageClass = null,
              Map params = null) {}
Browser(WebDriver driver = null, String baseUrl =
null, Class pageClass = null, Map params = null) {}

// drive()メソッド
static drive(Class pageClass, Closure script) {}
static drive(String baseUrl, Closure script) {}
static drive(String baseUrl, Class pageClass,
                     Closure script) {}
static drive(WebDriver driver, Closure script) {}
static drive(WebDriver driver, String baseUrl,
                     Closure script) {}
static drive(WebDriver driver, Class pageClass,
                     Closure script) {}
```

■ Grailsテストクラスの削除

Grailsで使用するためのテストクラスとして、以前まではテストフレームワークごとにサブクラスが用意されていましたが、今回のバージョンアップでは削除されました。今後はサブクラスの継承元になっていたクラスを直接使用します。

統合するテストフレームワーク	削除されたサブクラス	継承元のクラス（今後直接使用する）
JUnit 3	JUnit3GebTests	geb.junit3.GebReportingTest
JUnit 4	GebTests	geb.junit4.GebReportingTest
EasyB	GebEasybPlugin	geb.easyb.GebPlugin
SPock	GebSpec	geb.spock.GebReportingSpec

まとめ

如何でしたでしょうか？今回のバージョンアップでは、大きく変わりながらも、着実に使いやすくより実用的になったように思われます。レポート機能のディレクトリグループの設定は地味ではありますが、エビデンスを管理する上で非常に助かる機能ですし、外部スクリプトのバインディングは業務シナリオに特化したスクリプトを外部に置くことができ、UATでの利用がしやすくなったのではないでしょうか。

さて、次回はテストフレームワークと連携してGebをCIで実行する方法についてご紹介したいと思います。お楽しみに！

リンク

- Gebマニュアル　http://www.gebish.org/manual/current/index.html

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２章～

吉田　健太郎
日本 Grails/Groovy ユーザグループ運営委員。Groovyエバンジェリスト。
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。
ブログ：『No Programming, No Life』http://d.hatena.ne.jp/fumokmm/』

第２章 もかは『プログラミング Groovy』と出会った

●前回までのあらすじ

プログラマを志望して４月から新人プログラマとして駆け出した七海 萌香(ななみ もか)は同期である一ノ瀬 小夏(いちのせ こなつ)とプログラム研修の日々を過ごしていた。
ある日、こなつと共に書店に立ち寄ったもかはそこでGroovy イン・アクションと運命の出会いを遂げる。
Groovyイン・アクションを手に入れたもかはその内容に夢中になりGroovyの世界へと引き込まれてゆくのであった。

「よーし、こないだの結果を返すぞ～。」
「七海さん、君は名前を間違ってたぞ？仕事でこんなじゃ困るんだがな…」

もか「え？どれだろう？ああっ…コメント部分かぁ＞＜」

```
 * @author 七海 萌花
```

もか「もぅ、IMEのバカバカ！嫌い！もかって名前はいつも一発じゃ変換できないんだよねぇ…今度から気を付けなくっちゃ」

■お昼休み

ランチタイムの喫茶店は今日も混み合っていた。もかとこなつは会社近くのカフェにランチに来ていた。テーブルにはパスタとパニーニ、それからアイスコーヒー、エスプレッソなどが並んでいる。

もか「はぁ…」
こなつ「どうしたんですか、もかちゃん。ため息ですか？」
もか「あ、うん、今日のコメントのやつなんだけどさぁ…」
こなつ「そんなの気にすることないですよ、もかちゃんの書いたコードが凄すぎるからみんな嫉妬してるんですよ」
もか「えっ、そんな、すごいなんてことはないよ」
こなつ「ううん、そんなことあります。もかちゃんは私の目標なんです」
もか「ええっ？目標ってそんな…」

もかは照れ笑いを浮かべる。

こなつ「研修が終わっちゃうとみんなバラバラのプロジェクト配属になるんですよね。私、もかちゃんと一緒がいいです」
もか「そうだね、私もこなつと一緒になれるといいなぁって思うよ」
こなつ「佐々木君と、松上君の二人はどうもスラッツワン？っていうJavaのワークフレームを使ったプロジェクトに配属になるらしいんですよ」
もか「ええと、Strutsのバージョン１のことかな？あと多分ワークフレームじゃなくて、フレームワークだよ」
こなつ「えっ…＞＜ 私全然間違えちゃってますか？ はずかしいです…」

こなつは顔を真っ赤にしている。

もか「まぁ…分かりにくい名前だし、しょうがないよ」
こなつ「うわーん、もかちゃん優しいです…ぐすん」
もか「それにしてもあの二人、もう配属先決まってるんだ、やるなぁ」
こなつ「確か…出来がいいから、クライアントから早めのオファーがもらえたってことらしいですよ」
もか「ふーん」
こなつ「でも、あの二人、JDKのバージョンがいまだに1.4とかありえなくない？とかそんな話をしてたんですけど、私ついていけなくて…」
もか「あぁ、Strutsってたしか結構古いフレームワークなんだよね、この業界、最新ばかり追えるわけじゃないんだぜって口癖の人を一人知ってるわ」
こなつ「最新どころか基礎の基礎の基礎にもついて行けない私はどうしたらいいんですか＞＜」
もか「まぁまぁ…頑張ろうよこなつ！まだ始まったばっかりなんだし」
こなつ「ありがとうございます。やっぱりもかちゃんは優しいです」

感情のころころ変わる子だなぁ、まぁそこがかわいいんだけど。

こなつ「あっ！そういえば話変わりますけど、もかちゃん、あの日、本屋さんで見てた厚い本って結局買ったんですよね？すごいです」
もか「うん買ったよ。Groovyイン·アクションのことでしょ？ちょっとお財布は痛かったんだけどね（涙）」

こなつは目を輝かせてもかに話かけてくる。くりっとした目が小動物のようで、例えるならリス？

こなつ「あれってジーナちゃんですよね！私あの後、色々調べたんですよ。Groovyイン・アクションって海外とかではGinA って略されてて、それで、ジーナって呼ばれてるらしいんです」
もか「え、そっ、そうなの？じーいんえーって略すんだ？ で、ジ、ジーナ？そうなの？またどうして調べたりなんか…？」

話の展開が急すぎてもかは頭がついていっていない。しかしお構いなしにこなつは続ける。

こなつ「ジーナちゃんって名前はかわいいと思いませんか？だからそう呼ぶことにしたんです。色々調べ始めたのはもかちゃんが私の目標だからです」

目の輝きが先ほどよりも明らかに増している。この子ってもしかして、何かに夢中になるとすごい力を発揮するタイプなの？？

こなつ「それから聞いて下さい！実は、つい最近なんですけど、とある会場で『プログラミングGroovy』の著者の先生たちのイベントがあったみたいなんですよ。そのイベントの目玉企画として、

「@groovybook Gマガ読んだよ！ http://grails.jp/g_mag_jp/」

って「ついっといったー（注釈：ひとことつぶやきサービス）」でつぶやくと本を抽選でプレゼントするっていう企画があったみたいなんです。イベントに参加していなくても応募資格があるってことでしたので、私もちょっとつぶやいてみたんです。そうしたら当たっちゃいました！えへっ」

（注釈：こなつと同じように実際にみなさんもTwitterでつぶやいて下さい。抽選で1名様に『プログラミングGroovy』をプレゼント致します。当選者発表は10月開催予定のG*ワークショップで！）

こなつはプログラミングGroovyを自分の顔の横に持っきて、満面の笑みを浮かべている。

もか「待って待って＞＜ ええとまず『プログラミングGroovy』っていう本を私知らないんだけど…」

こなつは微笑んでこうつぶやく。

こなつ「ペンギン本です」
もか「ぺ、ペンギン…本？」

こなつ「はい、ジーナちゃん以来、国内ではGroovyの書籍が出てなかったんですけど、今回満を持して発売された本らしいんです。Groovyの最新版に対応しているらしいんですよ。それから真ん中に可愛いペンギンさんが付いてますよね？ほら、だから本を買ったみんながペンギン本って愛称を付けて呼んでるんですよ。かわいいですよね。青本って呼ぶ人もいるみたいですけど、私は断然、ペンギン本です！」
もか「へぇ…私Groovyイン…あっ、ジーナ…ちゃん？ばっかり読んでてGroovyが今どうなってるのかってあまり気にしてなかったなぁ…。そういえばバージョンもジーナちゃんが書かれたのって結構前らしくて、Groovy最

新版の機能とかまだ追いついてないところがあるみたいだし…そのまま書いても動かないところがあったりでちょっと困ってたところなんだよね」

こなつ「そういうことなら！ペンギン本は最新版のGroovyにも対応しているらしいんですよ。たしかバージョン1.8？だったと思います」

もか「ふぇー、すごいわ。ちょっと読ませてもらってもいい？」

こなつ「もちろんです」

もかはこなつからペンギン本を受け取る。目をぐっと引く深い青の表紙、真ん中のペンギンちゃん、右下にはJavaエンジニア必携！の文字もある。中身をパラパラっとめくってみる。

P80では範囲の説明。なんだか図解がキレイでわかりやすいかも…。P144にはSwingBuilderで作ったTODOリストアプリとかが載ってる。面白そうなアプリケーション。

…

…

…

（1分後）

もか「こなつ！ねぇ、また今日の夕方、本屋さんまで付き合ってくれる？ 私もこれ欲しい！」

こなつ「もちろんいいですよ」

■ファミレス

こうして、その日のうちにもかはプログラミングGroovyも購入することになった。興奮冷めやらぬ中、二人は今度は会社近くのファミレスで一緒に夕食を取ることになった。料理を注文してから届くまで、もちろん話題はプログラミングGroovyのことで持ち切りだった。

もか「うわー、この本、すごく読みやすいね！ジーナも丁寧な文章で読みにくくはないんだけど…ちょっと専門的すぎて>< 読み飛ばしちゃってるところもあったりするの」

こなつ「そうなんですね、私は逆にジーナちゃんを読んでないからわからないんですけど…でも購入したみなさんは、すごくいい本だって言ってます。私もせっかく当たったし、もかちゃんが気に入っているGroovyだし、頑張って読み始めたところなんですけど…Javaが分かってないからちょっとやっぱり難しいです、でも頑張ります！」

もか「ねぇ、こなつさぁ、そういえばちょっと気になってたことがあるんだけど、きいていい？」

こなつ「え？はい…どうぞ」

もか「えっと、こなつはどこでこのペンギン本を知ったの？私なんて全然知らなかったよ」

こなつ「えっと…はい、私のついったーのフォロワーさんで、ナナハヤさんって人がいるんですけど、私がJava分かんないよー><ってつぶやいてたら優しく教えてくれたので、それ以来、仲良くさせてもらってるんです」

もか「ナナ…ハヤ？んーなんかどっかで聞いた事あるような…」

こなつ「え？そうなんですか？そういえばもかちゃんもついったーやってるんですか？もしよかったらID教えて欲しいです」

もか「あぁ…昔登録したIDならあるよ。今は全然やってはいないんだけどね。そっかぁ、こなつがやってるんなら私も再開しようかな」

こなつ「うん！一緒にやりましょう！もかちゃんもやってくれたら楽しいと思います。ナナハヤさんも紹介しますよ！ナナハヤさん、Groovyのことも詳しいみたいなんですよ、だからもかちゃんにもぴったりかなぁって思うんですよね」

もか「なるほどね、そのナナハヤさんって人にペンギン本のこととか、イベントのこととか教えてもらったってことね？…でもネットの人ってなんか怖くない？」

こなつ「こわい…？そうですか…？」

この子は…人を疑うことを知らないのかしら。

もか「だって、どこに住んでるとか、何やってるとかぜんぜん分からないでしょ？こなつは怖くないの？」

こなつ「ナナハヤさんは、都内でSEをやってるらしいんですよ、歳は私たちよりも10くらい上みたいです。Slerさんらしいんです。お仕事はいつも大変そうですけど。なんだか同じ業界の先輩っ！て感じで、全然怖くないですよ」

もか「えっ…それって…もしかして、まってまって！」

こなつ「ん？…えっ？…ち、違いますよぉ…別にその、好きとかそんなんじゃぁ…」

もか「都内で、Slerで、ナナハヤ…七早…、七海 早瀬！」

こなつ「え？ あっ…（なんだぁ私の思い過ごしかぁ><）」

こなつ「ええと…ななみ はやせ、さんっていうのは、どちら様なんですか？」

もか「…うちの兄貴…」

こなつ「ええっ？？？ お、お兄さん？」

もか「うん、私、歳の離れた兄貴がいるんだけど、兄貴は東京で一人暮らししてる。最近ずっと会ってないけど会社はいわゆるSlerだね。ついったーやってるかどうかは知らないんだけど、ナナハヤってIDからすると兄貴なんじゃないかなぁ…」

もかは浮かない顔をしている…こなつは心配になって声をかける。

こなつ「それなら、直接聞いてみたらわかるんじゃないですか？」
もか「わかんないよ、電話番号とか、メールアドレスとか知らないし」
こなつ「ええっ…、兄妹なのになんでですか？」

こなつに質問されるともかはちょっとむっとした顔をした。こなつは困惑する。

こなつ「ああーっ、ええっと…ごめんなさい、なにかあるんですよね…よ、余計なことを聞いちゃいました私…ごめんなさい、ごめんなさい、ごめんなさい…」

こなつはずっと頭を下げたままずっと謝り続けている。

もか「ごめんね、こなつは何も悪くないよ。それにさ、本当にそのナナハヤさん？が兄貴かどうかもわからないし」
こなつ「ご、ごめんなさい＞＜」
もか「いいって、…」

そう言ってから沈黙がどのくらい続いただろう…その空気を破るようにもかは立ち上がった。

もか「ごめん、今日は帰るね、本ありがとう。また明日ね」

そう言うともかは足早に席を立ち、自分の分の精算を済ませ、ファミレスを後にした。

■帰り際

もか「あぁやっちゃったぁ私…明日こなつにちゃんと謝らなきゃなぁ…」

もかは二人兄妹で、早瀬という名の10歳上の兄がいる。兄は小さい頃からコンピュータに興味を持っていて、特にプログラミングが好きだった。そんな兄の影響で、もかは小さい頃から兄にプログラミングを教えてもらっていた。ただ、やはり幼いもかには理解できない部分が多かったため、なんとなく流れは理解できるものの、胸を張ってプログラミングできるレベルにはなれていなかった。ただ漠然とプログラミングって面白いなと思うようにはなっていた。

ある日、あのことがあるまではもかはプログラミングが好きな女の子だった。

■自室にて

お風呂から上がり、パジャマに着替えて自室の机にて。もかは自分のノートPCを開いた。傍らには今日買ってきたペンギン本。

もか「さぁ、色々考えたってしょうがない！せっかく買ったんだし、ペンギン本を読もうじゃないか！」

帯にHello Worldのコードがある。

```
String.metaClass.hello = { "Hello, $delegate!" }
println "Groovy".hello()
```

もか「Stringのメタクラスにhelloってメソッドを追加していて…えっと…helloメソッドは　"Hello" + 文字列自身の内容　を返却するのね。それで、println で、文字列 "Groovy" の helloメソッドを呼んでるんだから、えっと…わかったわ！ "Hello, Groovy!" って出力されるのね、なるほどぉ…いきなり捻った内容だわ」
もか「なんだかジーナを最初に買った日と同じわくわくどきどきだわ！」

どんどんページをめくって読み進めていくと、P10で目が止まる。

「1.3. Groovyの使いどころ：７つの導入パターン」

そこにはGroovyの特徴を伝える刺激的な７つの言葉が並べられていた。

万能接着剤 (Super Glue)
やわらかな心臓部 (Liquid Heart)
内視鏡手術 (Keyhole Surgery)
スマート設定 (Smart Configuration)
無制限の開放 (Unlimited Openness)
小人さんスクリプト (House-elf Scripts)
プロトタイプ (Prototype)

もか「Groovyってやっぱりすごいのね、GroovyはJavaを周りから助ける最高のパートナーみたいな言語なんだわ。ビルダーとかを使えば今研修でやっているJavaをコンパイルするために書いているAntのファイルとか、JUnitを実行させたり、本当に色々できるんだ…」

気がつくともう深夜0時をまわっている。『プログラミングGroovy』、面白くてもっと読んでいたいけど…

もか「明日もお仕事だからあまり夜更かしは良くないわ。こなつにも明日はちゃんと謝ろう。今日は色々あって疲れちゃった。もう寝よう。兄貴、今ごろ何してるのかな…おやすみなさい」

（続く…）

イラスト・けんや©

GBench in a Nutshell

永井 雅人（ながい まさと）

プログラマ。GBench, Gj の作者。
最近は Groovy 及びその周辺プロジェクトへの貢献が日課。
共著『パーフェクト Java』

GBench とは何か？

GBench は Groovy の為の強力なベンチマーキングフレームワークです。どのぐらい強力か、まずはその一片をお見せしましょう。例えばあなたは今ボトルネックになっているかもしれない次のメソッドのベンチマークを取りたいとします:

```
def doComplexTask() {
    // ...
}
```

そういった場合、通常は処理の前後にベンチマーク用のお決まりの処理を書き足すことになります。これはとても退屈な作業です。しかし、GBench を使えば既存のコードに変更を加える必要はありません。あなたがすべきことはたった1つ、アノテーションを付けるだけです:

```
@Benchmark
def doComplexTask() {
    // ...
}
```

これだけでメソッドの実時間と CPU 時間が出力されます:

```
java.lang.Object doComplexTask()  user:15600100
system:46800300 cpu:62400400 real:396709428
```

これは GBench の Benchmark AST Transformation と呼ばれる機能です。どうです、強力でしょう？しかし GBench の機能はこれだけではありません。Builder 構文を使って積極的なベンチマークを簡単に実現する BenchmarkBuilder もあります。GBench が Groovy の特性である AST Transformation と Builder 構文を生かしてることに注目してください。これは非常に重要な事実です。GBench を使えば Groovy のパフォーマンス向上の実証と一緒に Groovy が如何に強力な言語であるかを知らしめるられるからです。

GBench を手に入れる

GBench をインストールする最も手軽な手段は Grape を使うことです。

```
@Grab(group='com.googlecode.gbench',
                module='gbench',version='0.2.0')
```

しかし Groovy 使いの嗜みとしてここは最初からクラスローダに読み込ませておくべきでしょう。[GBench プロジェクト|http://code.google.com/p/gbench/] で jar を手に入れ、それを $GROOVY_HOME/lib に入れてください。

Benchmark AST Transformation

Benchmark AST Transformation は GBench の目玉機能です。既存のコードの変更を必要とせずベンチマークを取得できます。

■対象範囲の指定

範囲の指定方法は簡単です。対象としたい場所にアノテーションを付けるだけです。

メソッドを対象にする:

```
import gbench.*

class Klass {
    @Benchmark
    def foo() {
    }
}
```

```
Klass  java.lang.Object foo()  user:15600100
system:46800300 cpu:62400400 real:396709428
```

クラスが持つメソッド全てを対象にする:

```
import gbench.*

@Benchmark
class Klass {
    def foo() {
    }
    def bar() {
    }
    def static baz() {
    }
}
```

```
Klass  java.lang.Object foo()  user:15600100
  system:46800300 cpu:62400400 real:396709428
Klass  java.lang.Object bar()  user:842405400
  system:15600100 cpu:858005500 real:986409515
Klass  java.lang.Object baz()  user:31200200
  system:0 cpu:31200200 real:39076475
```

Groovy 1.8で追加されたASTTransformationCustomizerを使えば、プログラムの全てのメソッドのベンチマークも取得できます（但し、ベンチマーク対象のメソッドの呼び出しが入れ子になると、呼び出し側のベンチマーク結果にちょっとしたオーバーヘッドが含まれます）。

```
// BenchmarkGroovyc.groovy
import gbench.Benchmark

import org.codehaus.groovy.control.
CompilerConfiguration
import org.codehaus.groovy.control.customizers.
ASTTransformationCustomizer
import org.codehaus.groovy.tools.FileSystemCompiler

def cc = new CompilerConfiguration()
cc.addCompilationCustomizers(
      new ASTTransformationCustomizer(Benchmark))
new FileSystemCompiler(cc).commandLineCompile(args)
```

```
shell> groovy -cp gbench-0.2.0.jar
       BenchmarkGroovyc.groovy MyApp.groovy
shell> java -cp .;%GROOVY_HOME%\embeddable\
       groovy-all-1.8.0.jar;gbench-0.2.0.jar MyApp
```

●出力形式の指定

出力形式を変更したい場合、システムプロパティ"gbench.defaulthandle"に出力用の式を指定します。例えば次のようなスプレッドシート用にCSV形式で出力したいとします:

class	method	user	system	cpu	real
Klass	java.lang.Object foo()	15600100	46800300	62400400	396709428
Klass	java.lang.Object bar()	842405400	15600100	858005500	986409515

その場合次のように指定して実行します。ベンチマーク情報を取得する為に使えるバインド変数はklass(クラス名)、method(メソッドシグネチャ)、time(時間)の3つです。timeは各時間をプロパティとして持ちます:

```
shell> groovy -e "println('class,method,user,
  system,cpu,real')" > benchmarks.csv
shell> groovy -Dgbench.defaulthandle=
  "println([klass,method,time.user,time.system,
  time.cpu,time.real].join(','))" MyApp.groovy >>
  benchmarks.csv
```

特定範囲の出力形式を変更したい場合、出力用の式を含むクロージャを@Benchmarkに渡すのが最も簡単な方法です（※Groovy 1.7以前では使えません）:

```
@Benchmark({println("${method} of ${klass}:
${(time.real/1000000) as long} ms")})
def foo() {
}
```

もう少し複雑な処理、あるいはGroovy 1.7以前の場合はBenchmarkHandlerインターフェイスを使います。実装クラスにgetInstance()メソッドが必要な点に注意してください:

```
class MyHandler implements Benchmark.
BenchmarkHandler {
    static def instance = new MyHandler()
    static MyHandler getInstance() {
        instance
    }
    void handle(klass, method, time) {
        println("${method} of ${klass}: ${(time.
real/1000000) as long} ms")
    }
}
```

シングルトンなのであれば次のようにも書けますね:

```
@Singleton
class MyHandler implements Benchmark.
BenchmarkHandler {
    void handle(klass, method, time) {
        println("${method} of ${klass}:
              ${(time.real/1000000) as long} ms")
    }
}
```

後はクロージャと同じように@Benchmarkへ渡すだけです:

```
@Benchmark(MyHandler.class)
def foo() {
}
```

ここまでは便宜上printlnによる出力例のみをあげてきましたが、選択肢は様々です。いくつか例をあげておきます:

```
@Log
@Singleton
class MyHandler implements Benchmark.BenchmarkHandler {
    void handle(klass, method, time) {
        log.info(...)
    }
}
```

```
@Singleton
class MyHandler implements Benchmark.BenchmarkHandler {
    def file = new File('benchmark.log')
    void handle(klass, method, time) {
        file << ...
    }
}
```

BenchmarkBuilder

Benchmark AST Transformationがプログラムの既存コードのベンチマークを取るのに適しているのに対し、BenchmarkBuilderはベンチマークプログラムを書くのに適した機能です。例えば同じ用途のAPIが複数あり、それらのベンチマークを比較したいとします。通常は次のように書くことになります（好みによって多少の違いはあるでしょうが）:

```
def methods = [
    foo: {
        ...
    },
    bar: {
        ...
    },
    baz: {
        ...
    }
]
methods.each { name, method ->
    def time = 0
    (100 + 1).times { n ->
        def b = System.nanoTime()
        method()
        def a = System.nanoTime()
        if (n > 0) { // ignore dry run
            time += (a - b)
        }
    }
    println String.format('%-20s\t%10d', name, time)
}
```

同じものをBenchmarkBuilderを使えば次のように簡潔に書けます（BenchmarkBuilderは初期状態でウォームアップ用に各処理を1度空実行します）:

```
import gbench.*

def benchmarks = new BenchmarkBuilder().run repeat:100, {
    foo {
        ...
    }
    bar {
        ...
    }
    baz {
        ...
    }
}
benchmarks.prettyPrint()
```

Benchmark AST Transformation同様、結果にはCPU時間も含まれます。

```
        user       system        cpu         real
foo   65000000    20000000    80000000    110000000
bar   50000000    10000000    60000000     90000000
baz   55000000    15000000    70000000    100000000
```

ソートも可能です。例えばユーザ時間でソートしたい場合は次のようにします。run()の戻り値はプロパティとしてlabelとtimeを持つオブジェクトのリストです:

```
benchmarks.sort({it.time.user}).prettyPrint()
```

```
        user       system        cpu         real
bar   50000000    10000000    60000000     90000000
baz   55000000    15000000    70000000    100000000
foo   65000000    20000000    80000000    110000000
```

合計時間ではなく平均時間を取りたい場合は引数にaverage: trueを追加します:

```
new BenchmarkBuilder().run average: true, repeat: 100, {
```

あるいは、run()の代わりにsum()、average()を呼び出せばaverage引数を省略できます:

```
new BenchmarkBuilder().sum repeat: 100, {
```

```
new BenchmarkBuilder().average repeat: 100, {
```

BenchmarkBuilderのオプションはこれらだけではありません。全てのオプションを知るにはjavadocを参照して下さい。

最後に

GBenchへのフィードバックはプロジェクトのissue tracking system、作者のブログ、またはTwitter:@nagai_masato 経由で送ることができます。 あなたのフィードバックがGBenchの改善を継続する力になります。

Grails Plugin 探訪

第 4 回

~ Cloud Foundry プラグイン ~

URL: http://www.grails.org/plugin/cloud-foundry
プラグインのバージョン: 1.0.1
対応するGrailsのバージョン: 1.3.3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

今回ご紹介するGrailsプラグインは、Cloud Foundry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 : Mac OS X 10.6.8
- Java : 1.6.0_26
- Grails : 1.3.7

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、実際の出力異なる場合があります。ご了承願います。

Cloud Foundryとは

Cloud Foundryとは、VMware社が提供するPaaSプラットフォームです。

Springを使ったJavaアプリケーション、RailsやSinatraを使ったRubyアプリケーション、Nodeアプリケーション、GrailsやScala-Liftなど、JVM上で動作するアプリケーションをサポートします。

また、MySQLやRedis、MongoDBといったデータベースサービスも提供します。

詳しくは、Cloud Foundyを参照してください。

Cloud Foundryプラグインとは

Cloud Foundryプラグインとは、Grailsアプリケーションを Cloud Foundry上にデプロイしたり、更新したり、またCloud Foundry上のGrailsアプリケーションをモニタリングするのを、容易に実行するためのプラグインです。

"vmc"というRubyで作成されたコマンドラインのツールがありますが、Cloud Foundryプラグインをインストールすることで、"vmc"とほぼ同じ機能を実行することができます。

Cloud Foundryへのサインイン

Cloud Foundryを利用するには、サインアップする必要があります。サインアップは、ここから行えます。

サインアップで気を付けて欲しいのは、すぐにCloud Foundryが利用できるようになるのではなく、利用可能になったことを知らせる招待メールが来るまでしばらく待たないといけない、ということです。

筆者の場合、招待メールが来るまで、3週間くらいかかりました。

サンプルアプリケーションの作成

サンプルアプリケーションとして、今回は書店用のアプリケーションを作成します。

```
$ grails create-app store
```

ドメインクラスとして、書籍を表すBookドメインクラスを作成します。

```
$ cd store
$ 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。

```
ackage store

class Book {
    String title
    int price
    String isbn13

    static constraints = {
        title(blank: false)
        price()
        isbn13()
    }
}
```

コントローラとビューは、generate-allコマンドでスカッフォールドします。

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

Cloud Foundryプラグインのインストール

create-appコマンドでGrailsアプリケーションを作成した後、最初に行うことはCloud Foundryプラグインをインストールすることです。

```
$ grails install-plugin cloud-foundry
```

設定

Cloud Foundryを利用するための設定は、grails-app/conf/Config.groovyに記述します。

設定できるプロパティは、次のとおりです。

プロパティ名	説明	デフォルト値
grails.plugin.cloudfoundry.username	アカウント用のユーザ名/メールアドレス	なし(必須)
grails.plugin.cloudfoundry.password	アカウント用のパスワード	なし(必須)
grails.plugin.cloudfoundry.target	APIサーバ名	'api.cloudfoundry.com'
grails.plugin.cloudfoundry.appname	アプリケーション名	Grailsアプリケーションの名前
grails.plugin.cloudfoundry.showStackTrace	コンソールにスタックトレースを表示するかどうか	false
grails.plugin.cloudfoundry.testStartWithGet	GETリクエストで開始や再起動をテストするかどうか	true
grails.plugin.cloudfoundry.testStartGetUrl	上記プロパティがtrueの場合に使用するURL	アプリケーションのURL
grails.plugin.cloudfoundry.datasource.disableTimeoutAutoconfiguration	MySQLデータソースのコネクションタイムアウトチェックの自動設定を無効にするかどうか	true

最低限、grails.plugin.cloudfoundry.usernameとgrails.plugin.cloudfoundry.passwordを設定してください。

ただし、grails.plugin.cloudfoundry.usernameとgrails.plugin.cloudfoundry.passwordについては、$HOME/.grails/settings.groovyに記述することもできます。

初期状態

初期状態では、Cloud Foundry上で利用するサービスやアプリケーションの設定は何も行われていません。

cf-servicesコマンドを利用し、利用できるサービスを確認してみましょう。

```
$ grails cf-services

============== System Services ==============

+-----------+---------+-----------------------------+
| Service   | Version | Description                 |
+-----------+---------+-----------------------------+
| mongodb   | 1.8     | MongoDB NoSQL store         |
| redis     | 2.2     | Redis key-value store service |
| rabbitmq  | 2.4     | RabbitMQ messaging service  |
| mysql     | 5.1     | MySQL database service      |
| postgresql | 9.0     | PostgreSQL database service |
+-----------+---------+-----------------------------+


=========== Provisioned Services ============
None provisioned yet.
```

cf-servicesコマンドを実行すると、利用可能なサービスの一覧と、作成したサービスのインスタンスの一覧が表示されます。

初期状態なので、作成したサービスのインスタンスの一覧には何も表示されません。

また、cf-appsコマンドを利用し、デプロイ済みのアプリケーションの一覧を見てみましょう。

```
$ grails cf-apps

No Applications
```

初期状態なので、デプロイ済みのアプリケーションの一覧には何も表示されません。

最後に、cf-infoコマンドを利用し、リソースの使用状況を見て

みましょう。

```
$ grails cf-info

VMware's Cloud Application Platform
For support visit http://support.cloudfoundry.com

Target:   http://api.cloudfoundry.com (v0.999)

User:     your.email@yourserver.com
Usage:    Memory   (0B of 2.0G total)
          Services (0 of 16 total)
          Apps     (0 of 20 total)
```

サービスのインスタンスの作成

Cloud Foundry上のサービスを利用するためには、サービスのインスタンスを作成する必要があります。

サービスのインスタンスの作成は、cf-create-serviceコマンドで行います。

今回はMySQLを利用するため、MySQLのサービスのインスタンスを作成することにします。

```
$ grails cf-create-service mysql

Service 'mysql-a366716' provisioned.
```

再度cf-servicesコマンドを実行すると、MySQLサービスのインスタンスが作成されていることを確認できます。

```
$ grails cf-services
============== System Services ==============

+------------+---------+------------------------------+
| Service    | Version | Description                  |
+------------+---------+------------------------------+
| redis      | 2.2     | Redis key-value store service |
| mongodb    | 1.8     | MongoDB NoSQL store          |
| rabbitmq   | 2.4     | RabbitMQ messaging service   |
| postgresql | 9.0     | PostgreSQL database service  |
| mysql      | 5.1     | MySQL database service       |
+------------+---------+------------------------------+


=========== Provisioned Services ============

+---------------+---------+
| Name          | Service |
+---------------+---------+
| mysql-a366716 | mysql   |
+---------------+---------+
```

アプリケーションのデプロイ

サービスのインスタンスを作成したら、サンプルアプリケーションをデプロイしてみましょう。

アプリケーションのデプロイは、cf-pushコマンドを使用し、引数として先ほど作成したサービスのインスタンスを指定します。

```
$ grails prod cf-push --services=mysql-a366716

(省略)
      [jar] Building jar: /Users/rhapsody/
        Project/gmaga/plugin004/store/target/cf-
        temp-1314297372348.war
  [delete] Deleting directory
                /Users/rhapsody/.grails/1.3.7/projects/store/stage
                 Done creating WAR /Users/rhapsody/
                 Project/gmaga/plugin004/store/target/cf-
                 temp-1314297372348.war

Application Deployed URL: 'store.cloudfoundry.com'?
```

途中で、アプリケーションをデプロイするURLを指定します。

デフォルトでは、アプリケーション名.cloudfoundry.comとなりますが、別のURLすることもできます。

ここでは、そのまま(store.cloudfoundry.com)にしてみます。

```
Creating application store at store.cloudfoundry.
com with 512MB and services [mysql-a366716]:
Error: 400 Bad Request

com.vmware.appcloud.client.CloudFoundryException:
  400 Bad Request (The URI: "store.cloudfoundry.
        com" has already been taken or reserved)
(省略)
```

どうやら、store.cloudfoundry.comは既に存在するようで、別のURLにする必要があるみたいです。

```
$ grails prod cf-push --services=mysql-a366716

(省略)

Application Deployed URL: 'store.cloudfoundry.com'?
```

今度は、デプロイ先のURLとしてjggugstore.cloudfoundry.comと入力します。

```
jggugstore.cloudfoundry.com

Creating application store at jggugstore.
cloudfoundry.com with 512MB
and services [mysql-a366716]: OK

Uploading Application:
  Checking for available resources: OK
  Processing resources: OK
  Packing application: OK
  Uploading (257K): OK

Trying to start Application: 'store'.
......

Application 'store' started at http://jggugstore.cloudfoundry.com
```

ここまで表示されれば、アプリケーションのデプロイは成功です。

Webブラウザでhttp://jggugstore.cloudfoundry.com/にアクセスすると、スカッフォールドした画面が表示されます。

では、cf-appsコマンドを再度実行してみましょう。

```
$ grails cf-apps

+-------------+---+---------+--------------------------+--------------+
| Application | # | Health  | URLs                     | Services     |
+-------------+---+---------+--------------------------+--------------+
| store       | 1 | RUNNING | jggugstore.cloudfoundry.com | mysql-a366716 |
+-------------+---+---------+--------------------------+--------------+
```

デプロイしたアプリケーションの状態を確認することができます。

cfコマンド

アプリケーションのデプロイまで解説してきましたので、ここから残りは、Cloud Foundryプラグインが提供するコマンドを説明します。

■ヘルプ

●cf-help

使用できるコマンドの一覧を表示します。

▼使用方法

```
$ grails cf-help
```

▼実行例

```
$ grails cf-help

General syntax: grails [environment] <scriptName>
[<args>] [command_options]
Use 'grails help <scriptName> for more
information.

Currently available commands are:

  Getting Started
    cf-info        System and account information

  Applications
    cf-apps        List deployed applications

(途中省略)

  Help
    cf-help        Get general help
```

■インフォメーション

●cf-info

Cloud Foundryのリソース使用状況を表示します。

▼使用方法

```
$ grails cf-info
```

▼実行例

```
$ grails cf-info

VMware's Cloud Application Platform
For support visit http://support.cloudfoundry.com

Target:   http://api.cloudfoundry.com (v0.999)

User:     your.email@yourserver.com
Usage:    Memory   (512.0M of 2.0G total)
          Services (1 of 16 total)
          Apps     (1 of 20 total)
```

■アプリケーション

●cf-apps

アプリケーションの一覧を表示します。

▼使用方法

```
$ grails cf-apps
```

▼実行例

```
$ grails cf-apps

+------------+---+---------+--------------------------+--------------+
| Application | # | Health  | URLs                     | Services     |
+------------+---+---------+--------------------------+--------------+
| store      | 1 | RUNNING | jggugstore.cloudfoundry.com | mysql-a366716 |
+------------+---+---------+--------------------------+--------------+
```

■デプロイ

●cf-push

warを作成しアプリケーションをデプロイします。

▼使用方法

```
$ grails cf-push [--appname=アプリケーション名]
[--url=URL] [--memory=メモリサイズ]
                  [--warfile=warファイルパス]
[--services=サービスのインスタンス名] [--no-start]
```

▼実行例

```
$ grails cf-push --appname=store
--url=jggugstore.cloudfoundry.com
--services=mysql-99afdca
(省略)

Creating application store at jggugstore.
cloudfoundry.com with 512MB
and services [mysql-99afdca]: OK

Uploading Application:
  Checking for available resources: OK
  Processing resources: OK
  Packing application: OK
  Uploading (5K): OK

Trying to start Application: 'store'.
.....

Application 'store' started at http://jggugstore.cloudfoundry.com
```

■アプリケーションの操作

●cf-start

アプリケーションを起動します。

▼使用方法

```
$ grails cf-start [--appname=アプリケーション名]
```

▼実行例

```
$ grails cf-start

Trying to start Application: 'store'.
......

Application 'store' started at http://jggugstore.cloudfoundry.com
```

●cf-stop

アプリケーションを停止します。

▼使用方法

```
$ grails cf-stop [--appname=アプリケーション名]
```

▼実行例

```
$ grails cf-stop

Application 'store' stopped.
```

●cf-restart

アプリケーションを再起動します。

▼使用方法

```
$ grails cf-restart [--appname=アプリケーション名]
```

▼実行例

```
$ grails cf-restart

Application 'store' stopped.


Trying to start Application: 'store'.
.......

Application 'store' started at http://jggugstore.cloudfoundry.com
```

●cf-delete-app

アプリケーションを削除します。

▼使用方法

```
$ grails cf-delete-app [--appname=アプリケーション名] [--force]
```

▼実行例

```
$ grails cf-delete-app

Application 'store' uses 'mysql-e244f88' service,
would you like to delete it? ([y], n)
y

Application 'store' deleted.

Service 'mysql-e244f88' deleted.
```

●cf-delete-all-apps

すべてのアプリケーションを削除します。

▼使用方法

```
$ grails cf-delete-all-apps [--force]
```

▼実行例

```
$ grails cf-delete-all-apps --force

Application 'store' deleted.

Service 'mysql-eaae347' deleted.
```

●cf-rename-app

アプリケーション名を変更します。ただし、Config.groovy中のアプリケーション名は変更しません。

▼使用方法

```
$ grails cf-rename-app 新しい名前 [--appname=アプリケーション名]
```

▼実行例

```
$ grails cf-rename-app jggugstore

Renamed 'store' to 'jggugstore'.
If necessary, update @grails.plugin.cloudfoundry.appname@
in @Config.groovy@ with the new name.
```

■アプリケーションの更新

●cf-update

warを作成しアプリケーションを再デプロイします。

▼使用方法

```
$ grails cf-update [--appname=アプリケーション名]
[--warfile=warファイルパス]
```

▼実行例

```
$ grails cf-update

(省略)

Updating Application:
  Checking for available resources: OK
  Processing resources: OK
  Packing application: OK
  Uploading (5K): OK

Application 'store' stopped.


Trying to start Application: 'store'.
......

Application 'store' started at http://jggugstore.cloudfoundry.com

atropos:store rhapsody$
```

●cf-update-memory

メモリの割当量を変更します。

▼使用方法

```
$ grails cf-update-memory メモリサイズ [--appname=アプリケーション名]
```

▼実行例

```
$ grails cf-update-memory 1G

Updated memory reservation to '1.0G'.

Application 'store' stopped.

Trying to start Application: 'store'.
.........

Application 'store' started at http://jggugstore.cloudfoundry.com

atropos:store rhapsody$
```

●cf-map

アプリケーションにURLを登録します。1つのアプリケーションに複数のURLを登録することができます。

▼使用方法

```
$ grails cf-map URL [--appname=アプリケーション名]
```

▼実行例

```
$ grails cf-map http://jggugstore2.cloudfoundry.com

Succesfully mapped url

$ grails cf-app

+-------------+----+--------+-----------------------------------------------------------+---------------+
| Application | #  | Health | URLs                                                      | Services      |
+-------------+----+--------+-----------------------------------------------------------+---------------+
| store       | 1  | 0%     | jggugstore.cloudfoundry.com, jggugstore2.cloudfoundry.com | mysql-7926b66 |
+-------------+----+--------+-----------------------------------------------------------+---------------+
```

●cf-unmap

アプリケーションに登録してあるURLを解除します。

▼使用方法

```
$ grails cf-unmap URL [--appname=アプリケーション名]
```

▼実行例

```
$ grails cf-unmap jggugstore2.cloudfoundry.com

Succesfully unmapped url

$ grails cf-app

+-------------+----+--------+-----------------------------+---------------+
| Application | #  | Health | URLs                        | Services      |
+-------------+----+--------+-----------------------------+---------------+
| store       | 1  | 0%     | jggugstore.cloudfoundry.com | mysql-7926b66 |
+-------------+----+--------+-----------------------------+---------------+
```

●cf-update-instances

アプリケーションのインスタンス数を変更します。

▼使用方法

```
$ grails cf-update-instances インスタンス数
[--appname=アプリケーション名]
```

▼実行例

```
$ grails cf-update-instances 2

Scaled 'store' up to 2 instances.

$ grails cf-app

+-------------+---+--------+--------------------------+--------------+
| Application | # | Health | URLs                     | Services     |
+-------------+---+--------+--------------------------+--------------+
| store       | 2 | 0%     | jggugstore.cloudfoundry.com | mysql-7926b66 |
+-------------+---+--------+--------------------------+--------------+
```

■アプリケーション情報

●cf-crashes

アプリケーションのデプロイに失敗した際のクラッシュ情報を表示します。

▼使用方法

```
$ grails cf-crashes [--appname=アプリケーション名]
```

▼実行例

```
$ grails cf-crashes

+---------+----------------------------------+-------------------+
| Name    | Instance ID                      | Crashed Time      |
+---------+----------------------------------+-------------------+
| store-1 | 6afc1a56deaa18f29d90c1850af87b80 | 08/27/2011 08:42午前 |
| store-2 | fad8a76d6068a524c8a9a24ad9f5184f | 08/27/2011 08:42午前 |
| store-3 | 73b0f62474c9e9cc5bdb89e2d6b58d3b | 08/27/2011 08:43午前 |
| store-4 | 3cab1d09832c69b9a79211a055afa49d | 08/27/2011 08:43午前 |
+---------+----------------------------------+-------------------+
```

●cf-crashlogs

アプリケーションのデプロイに失敗した際のクラッシュ・ログを表示します。

対象は次のとおりです。

- logs/err.log
- logs/stderr.log
- logs/stdout.log
- logs/startup.log

▼使用方法

```
$ grails cf-crashlogs [--appname=アプリケーション
名] [--instance=インスタンス番号]
```

▼実行例

```
$ grails cf-crashlogs

==== logs/err.log ====

# Logfile created on 2011-08-26 23:43:08 +0000 by logger.rb/25413
F, [2011-08-26T23:43:08.609557 #17058] FATAL -- : Memory limit
of 256M exceeded.
F, [2011-08-26T23:43:08.609608 #17058] FATAL -- : Actual usage
was 290M, process terminated.



==== logs/stderr.log ====

Aug 26, 2011 11:43:05 PM org.apache.coyote.http11.Http11Protocol
init
INFO: Initializing Coyote HTTP/1.1 on http-53482
Aug 26, 2011 11:43:05 PM org.apache.catalina.startup.Catalina
load
INFO: Initialization processed in 366 ms
(省略)
```

●cf-logs

ログファイルの内容を表示します。

対象は次のとおりです。

- logs/stderr.log
- logs/stdout.log
- logs/startup.log

▼使用方法

```
$ grails cf-logs [ファイル名] [--appname=アプリケー
ション名] [--instance=インスタンス番号] [--stderr]
[--stdout] [--startup]
```

▼実行例

```
$ grails cf-logs
==== logs/stderr.log ====

Aug 27, 2011 4:07:07 AM org.apache.coyote.http11.Http11Protocol init
INFO: Initializing Coyote HTTP/1.1 on http-16924
Aug 27, 2011 4:07:07 AM org.apache.catalina.startup.Catalina load
INFO: Initialization processed in 368 ms
(省略)
Aug 27, 2011 4:07:13 AM org.apache.coyote.http11.Http11Protocol start
INFO: Starting Coyote HTTP/1.1 on http-16924
Aug 27, 2011 4:07:13 AM org.apache.catalina.startup.Catalina start
INFO: Server startup in 5440 ms

ERROR: There was an error retrieving logs/startup.log, please try
again
```

●cf-list-files

ディレクトリの内容を表示します。

▼使用方法

```
$ grails cf-list-files [ディレクトリパス]
[--appname=アプリケーション名] [--instance=インス
タンス番号]
```

▼実行例

```
$ grails cf-list-files

logs/                                    -
tomcat/                                  -

$ grails cf-list-files logs

stderr.log                             1.2K
stdout.log                               0B

$ grails cf-list-files tomcat

bin/                                     -
conf/                                    -
lib/                                     -
logs/                                    -
stacktrace.log                          0B
temp/                                    -
webapps/                                 -
work/                                    -
```

●cf-get-file

ファイルをダウンロード、またはファイルの内容を表示します。格納先のファイル名を指定すればダウンロード、指定しなければファイル内容の表示となります。

テキスト形式のファイルが対象のようです。

▼使用方法

```
$ grails cf-get-file ファイルパス [格納先ファイル名]
[--appname=アプリケーション名] [--instance=インス
タンス番号]
```

▼実行例

```
$ grails cf-get-file logs/stderr.log ./stderr.log

Wrote logs/stderr.log to ./stderr.log

$ grails cf-get-file logs/stderr.log

Aug 27, 2011 4:07:07 AM org.apache.coyote.http11.Http11Protocol init
INFO: Initializing Coyote HTTP/1.1 on http-16924
Aug 27, 2011 4:07:07 AM org.apache.catalina.startup.Catalina load
INFO: Initialization processed in 368 ms
(省略)
Aug 27, 2011 4:07:13 AM org.apache.coyote.http11.Http11Protocol start
INFO: Starting Coyote HTTP/1.1 on http-16924
Aug 27, 2011 4:07:13 AM org.apache.catalina.startup.Catalina start
INFO: Server startup in 5440 ms
```

●cf-stats

アプリケーションのリソース使用状況を表示します。

▼使用方法

```
$ grails cf-stats [--appname=アプリケーション名]
```

▼実行例

```
$ grails cf-stats

+----------+------------+---------------+-------------+--------------+
| Instance | CPU (Cores) | Memory (limit) | Disk (limit) | Uptime       |
+----------+------------+---------------+-------------+--------------+
| 0        | 1.3% (4)    | 309.8M (512M) | 38.0M (2G)   | 0d:0h:19m:34s |
+----------+------------+---------------+-------------+--------------+
```

●cf-show-instances

インスタンスの状況を表示します。

▼使用方法

```
$ grails cf-show-instances [--appname=アプリケーション名]
```

▼実行例

```
$ grails cf-show-instances

+-------+---------+--------------------+
| Index | State   | Start Time         |
+-------+---------+--------------------+
| 0     | RUNNING | 08/27/2011 01:36午後 |
+-------+---------+--------------------+
| 1     | RUNNING | 08/27/2011 01:59午後 |
+-------+---------+--------------------+
```

■アプリケーション環境

●cf-env

アプリケーション用環境変数の一覧を表示します。

▼使用方法

```
$ grails cf-env [--appname=アプリケーション名]
```

▼実行例

```
$ grails cf-env

+----------+----------+
| Variable | Value    |
+----------+----------+
| LOCATION | Yokohama |
+----------+----------+
```

●cf-env-add

アプリケーション用環境変数を追加します。

▼使用方法

```
$ grails cf-env-add 環境変数名 値 [--appname=アプリケーション名]
```

▼実行例

```
$ grails cf-env-add LOCATION Yokohama

Adding Environment Variable [LOCATION=Yokohama]: OK

Application 'store' stopped.


Trying to start Application: 'store'.
...........

Application 'store' started at http://jggugstore.cloudfoundry.com
```

●cf-env-del

アプリケーション用環境変数を削除します。

▼使用方法

```
$ grails cf-env-del 環境変数名 [--appname=アプリケーション名]
```

▼実行例

```
$ grails cf-env-del LOCATION

Deleting Environment Variable [LOCATION]: OK


Application 'store' stopped.


Trying to start Application: 'store'.
.

ERROR - Application 'store' failed to start, logs information below.
```

■サービス

●cf-services

利用可能なサービスの一覧と、作成したサービスのインスタンスを表示します。

▼使用方法

```
$ grails cf-services
```

▼実行例

```
$ grails cf-services

============== System Services ==============

+------------+---------+------------------------------+
| Service    | Version | Description                  |
+------------+---------+------------------------------+
| redis      | 2.2     | Redis key-value store service |
| mongodb    | 1.8     | MongoDB NoSQL store          |
| rabbitmq   | 2.4     | RabbitMQ messaging service   |
| postgresql | 9.0     | PostgreSQL database service  |
| mysql      | 5.1     | MySQL database service       |
+------------+---------+------------------------------+


=========== Provisioned Services ============

+--------------+---------+
| Name         | Service |
+--------------+---------+
| mysql-99afdca | mysql   |
+--------------+---------+
```

●cf-create-service

サービスのインスタンスを作成します。

▼使用方法

```
$ grails cf-create-service ベンダー名(サービス名)
[サービスのインスタンス名] [--bind]
```

▼実行例

```
$ grails cf-create-service mongodb

Service 'mongodb-2c8f174' provisioned.
```

●cf-delete-service

サービスのインスタンスを削除します。

▼使用方法

```
$ grails cf-delete-service サービスのインスタンス名
```

▼実行例

```
$ grails cf-delete-service mongodb-2c8f174

Successfully removed service: mongodb-2c8f174
```

●cf-bind-service

サービスのインスタンスをアプリケーションにバインドします。

▼使用方法

```
$ grails cf-bind-service サービスのインスタンス名
[--appname=アプリケーション名]
```

▼実行例

```
$ grails cf-bind-service mongodb-49ecee4

Creating new service binding to 'mongodb-49ecee4' for 'store'.

Application 'store' updated
Service 'mongodb-49ecee4' added


Application 'store' stopped.


Trying to start Application: 'store'.
.....

Application 'store' started at http://jggugstore.cloudfoundry.com

atropos:store rhapsody$
```

●cf-unbind-service

サービスのインスタンスをアプリケーションからアンバインドします。

▼使用方法

```
$ grails cf-unbind-service サービスのインスタンス名
[--appname=アプリケーション名]
```

▼実行例

```
$ grails cf-unbind-service mongodb-49ecee4

Removing service binding 'mongodb-49ecee4' from 'store'.

Application 'store' updated
Service 'serviceName' removed


Application 'store' stopped.


Trying to start Application: 'store'.
........

Application 'store' started at http://jggugstore.cloudfoundry.com
```

●cf-clone-services

あるアプリケーションにバインドされているサービスのインスタンスを別のアプリケーションにクローンします。

▼使用方法

```
$ grails cf-clone-services クローン元アプリケーション名 クローン先アプリケーション名
```

▼実行例

```
$ grails cf-clone-services store jggugstore

Creating new service binding to 'mongodb-4d4697f' for 'jggugstore'.
Creating new service binding to 'mysql-99afdca' for 'jggugstore'.
Application 'jggugstore' stopped.
Trying to start Application: 'jggugstore'.
.....
Application 'jggugstore' started at http://jggugstore.cloudfoundry.com
$ grails cf-apps
+-------------+----+--------+---------------------------+-------------------------------+
| Application | #  | Health | URLs                      | Services                      |
+-------------+----+--------+---------------------------+-------------------------------+
| store       | 2  | 0%     | jggugstore.cloudfoundry.com | mongodb-4d4697f, mysql-99afdca |
+-------------+----+--------+---------------------------+-------------------------------+
| jggugstore  | 1  | 0%     | jggugstore.cloudfoundry.com | mongodb-4d4697f, mysql-99afdca |
+-------------+----+--------+---------------------------+-------------------------------+
```

■管理

●cf-user

ログインしているユーザ名を表示します。

▼使用方法

```
$ grails cf-user
```

▼実行例

```
$ grails cf-user
[your.email@yourserver.com]
```

■システム

●cf-runtimes

サポートしている実行環境の一覧を表示します。

▼使用方法

```
$ grails cf-runtimes
```

▼実行例

```
$ grails cf-runtimes

+--------+-------------+-----------+
| Name   | Description | Version   |
+--------+-------------+-----------+
| java   | Java 6      | 1.6       |
| node   | Node.js     | 0.4.5     |
| ruby18 | Ruby 1.8    | 1.8.7     |
| ruby19 | Ruby 1.9    | 1.9.2p180 |
+--------+-------------+-----------+
```

●cf-frameworks

サポートしているフレームワークの一覧を表示します。

▼使用方法

```
$ grails cf-frameworks
```

▼実行例

```
$ grails cf-frameworks

+---------+
| Name    |
+---------+
| grails  |
| lift    |
| node    |
| rails3  |
| sinatra |
| spring  |
+---------+
```

おわりに

Cloud Foundryプラグインを使えば、Grailsアプリケーションのデプロイやサービスとのバインドを、簡単に実行することができます。

制約はいろいろとありますが、Cloud FoundryでGrailsアプリケーションを公開することを考えていらっしゃる場合、是非Cloud Foundryプラグインを使用することを検討してください。

▼ぐるーびーたん　第2話までのあらすじ

プログラマを目指して熊本から上京したぐるーびーたん。IT業界の現実にも少しずつ慣れ、先輩社員からお仕事を頼まれるようにもなったのだが。。。

※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。

リリース情報 2011.08.25

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, 2.0.0.M2

■更新情報

- 2.0.0.M2では、エラーレポートと分析表示が強化されたり、付属のDBがH2に変更されたり、HTML5スカッフォルドとなる等、1.3系から大幅に変更されています。
- 2.0.0.M2リリースノート: http://grails.org/2.0.0.M2+Release+Notes
- 山本さんのブログ: http://d.hatena.ne.jp/mottsnite/20110907/1315412925

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, 1.8.1, 1.9-beta-1

■更新情報

- 1.8.1では、JSONサポートの強化や、takeメソッドの追加、いくつかのバグフィックスが行われています。
- 1.8.1リリースノート: http://jira.codehaus.org/secure/ReleaseNote.jspa?projectId=10242&version=17223
- 1.9-beta-1では、Project Coinのバイナリリテラルや例外のマルチキャッチの対応、いくつかのバグフィックスが行われています。
- 1.9-beta-1リリースノート: http://jira.codehaus.org/secure/ReleaseNote.jspa?projectId=10242&version=17153
- 関谷さんのブログ: http://d.hatena.ne.jp/ksky/20110722/p1

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

■更新情報

- 0.9.3では、Groovy 1.8.1対応や、新しいArchetypeが利用可能となったり、デフォルトパッケージ名の付け方が変わったりと、いろいろと追加・変更がなされています。
- 0.9.3リリースノート: http://griffon.codehaus.org/Griffon+0.9.3

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: 1.9.6

■更新情報

- 1.9.6では、Groovy 1.8.1および1.9.0-beta-1対応が行われています。
- 1.9.6リリースコメント: http://groovy.329449.n5.nabble.com/Gant-1-9-6-released-td4623268.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

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: 1.0

■更新情報

- 1.0では、GAE SDK 1.5.2に対応したり、テンプレートプロジェクトをGradleでビルドできるようになったり、データストアにgetメソッドが追加されたりと、様々な変更が行われています。
- 1.0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: 1.5.3

■更新情報

- 1.5.3では、Blobstore APIを使ったblobアップロードのサイズ制限がなくなったり、DatastoreServiceにgetIndexesメソッドが追加されたり、Remote APIが追加になったりと、様々な変更が行われています。
- 1.5.3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: 0.12GA

■更新情報

- 0.12GAでは、ActiveObjectのパフォーマンスが改良されたり、非同期なメソッドを作成するためにAST変換が追加されたり、いくつかバグフィックスが行われています。
- 0.12GAリリースノート: http://jira.codehaus.org/secure/ReleaseNote.jspa?projectId=12030&version=16994

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: 0.4.300

■更新情報

- 0.4.300では、多くのパフォーマンスの改善が行われているようです。

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: 0.9

■更新情報

- 0.9では、PWD環境変数が使えるようになったり、パフォーマンスの改善、バグフィックスなどが行われています。
- 0.9チェンジログ: http://kobo.github.com/groovyserv/changelog.html#ref-changelog

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: 0.6.0

■更新情報

- 0.6では、TestNGのサポートが追加されたり、Grails 1.3ベースとなったり、いくつかバグフィックスが行われています。
- 0.6リリースノート: http://jira.codehaus.org/secure/ReleaseNote.jspa?projectId=12101&version=16999

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: 0.15

■更新情報

- 0.15では、新しいルールが23追加されたり、いくつかバグフィックスが行われています。
- 0.15リリースノート: http://groovy.329449.n5.nabble.com/ANN-Announcing-CodeNarc-0-15-td4677503.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: 0.3

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: 1.2.4

■更新情報

- 1.2.4では、staticメソッドがサポートされたり、いくつかバグフィックスが行われています。
- 1.2.4リリースノート: http://blog.andresteingress.com/2011/06/13/gcontracts-1-2-4-released/

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://groovy.codehaus.org/GroovyFX
バージョン: 1.0 Alpha

■企業スポンサー

■個人サポーター

須江 信洋 様

関谷 和愛 様

田中　明 様

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

G* Magazine vol.3 2011.09
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

© 2011 JGGUG 掲載記事の再利用については
Creative Commons ライセンスによります。