vol.5

Magazine

Japan Grails/Groovy User Group

Contents

Series 07

JGGUG 合宿 2011 企画アプリ
GDK48 投票アプリの作り方（後編）…… 4

Series 08

GroovyFX で遊ぼう（第 2 回） …… 8

Series 04

Geb ではじめる Web テスト
～新機能活用編～…… 14

Series 05

Grails Plugin 探訪
～第 6 回 Build Test Data プラグイン～ …… 19

JGGUG 4 コマ漫画「ぐるーびーたん」第 5 話 …… 25

Information

リリース情報…… 26

series 07

JGGUG合宿2011企画アプリ

GDK48 投票アプリの作り方（後編）

奥 清隆（おく きよたか）

仕事でもがっつり 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』

前編から少し時間が経ちましたが、今回も後編ということでJGGUG合宿2011で作成したGDK48投票アプリの作り方を紹介していきます。今回は、Twitter連携と投票の仕組みについてです。

Twitterとの連携

前回はTwitterからハッシュタグ検索してツイートを拾ってくる処理を実装しましたが、今回はTwitter Authenticationプラグインを使って認証できるようにします。このプラグインは、Spring Security Coreプラグインの認証にTwitter認証を利用できるプラグインです。Twitter認証をアプリケーションで実装することを考えると大変そうに思えますが、このプラグインを使うと簡単にTwitter認証をアプリケーションに組み込むことができます。次のような手順でTwitter認証を実装します。

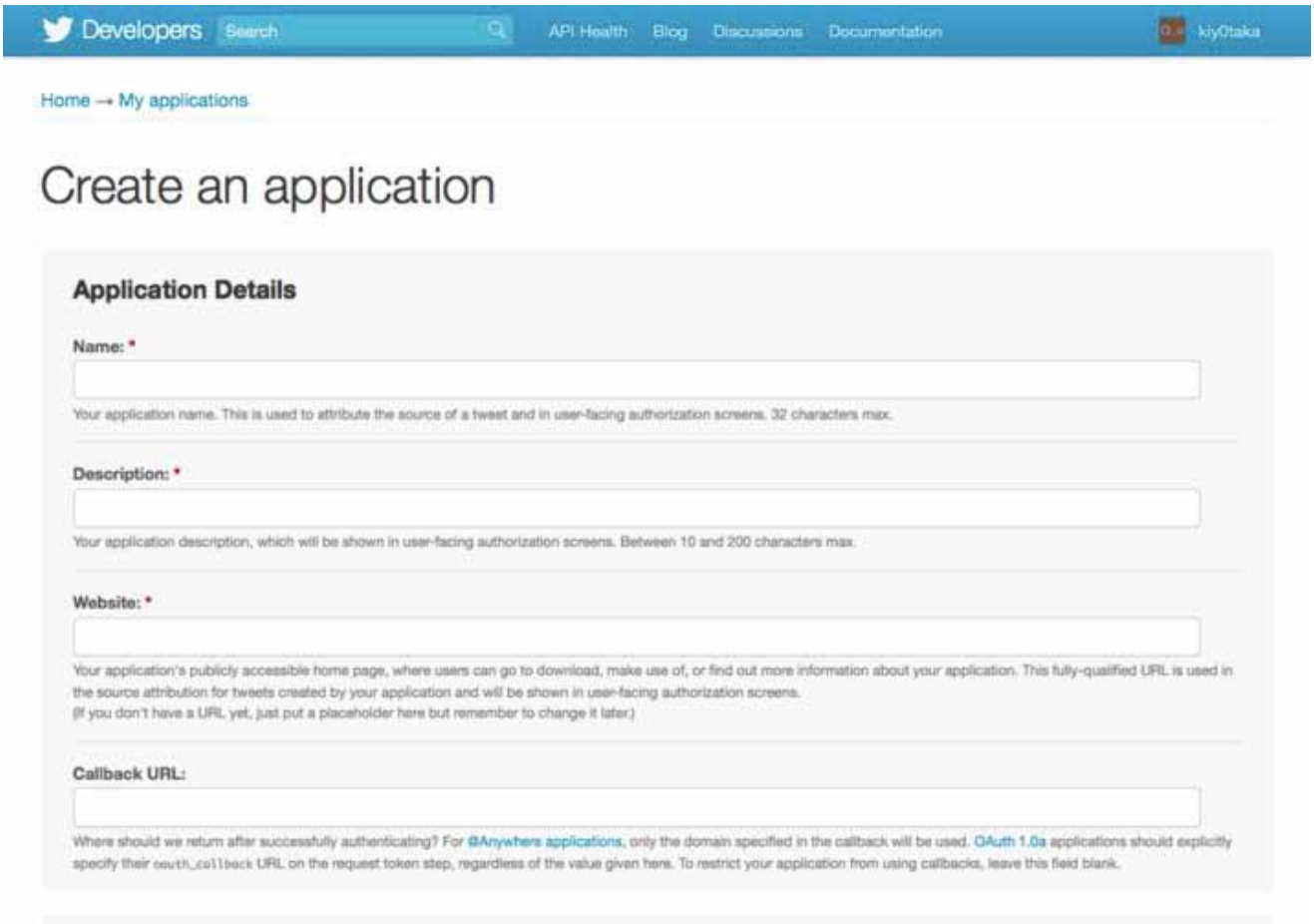
1. Twitter側にアプリケーションを登録
2. プラグインのインストール＆初期化
3. Twitter連携ボタンの設置

Twitterやプラグインの設定のみで、ほとんどコードを書く必要はありません。それではやってみましょう。

■Twitterアプリケーションの登録

アプリケーション申請画面にアクセスしてTwitterにアプリケーションを登録します。

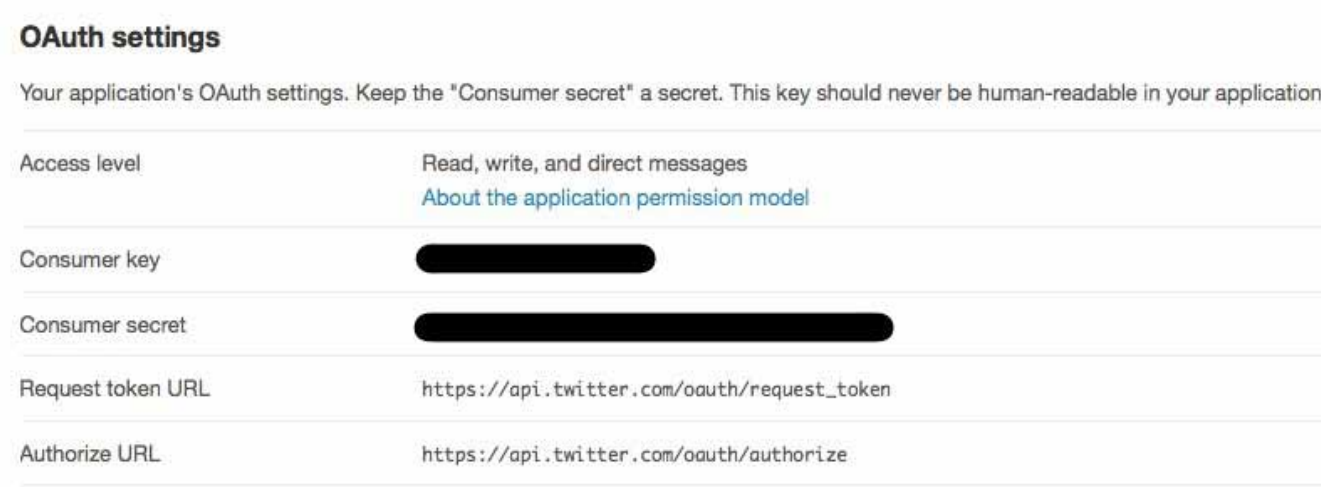
https://dev.twitter.com/apps/new



次の内容を入力しましょう。

Name	アプリケーション名
Description	アプリケーションの説明
Website	アプリケーションのWebサイト
Callback URL	Twitter認証後にコールバックするURL

Callback URLは、Webアプリケーションの場合に入力する必要がありますが、Twitter Authenticationが実行時にパラメータとしてURLを渡してくれるので、適当なURLで構いません。登録が完了するとConsumer KeyとConsumer Secretが発行されます。後ほど使うのでコピーしておきましょう。



■プラグインのインストール

GrailsでTwitter認証するためには、Spring Security Coreプラグインと Twitter Authenticationプラグインが必要です。まずは、Spring Security Coreプラグインをインストールしましょう。

```
$ grails install-plugin spring-security-core
```

次に、s2-quickstartコマンドを使って必要なドメインの生成とプラグインの設定をします。

```
$ grails s2-quickstart gdk48 User Role
```

これでSpring Security Coreプラグインがインストールできました。それではTwitter Authenticationのインストールと設定をしていきましょう。

```
$ grails install-plugin spring-security-twitter
```

s2-init-twitterコマンドを使って、Twitter連携の設定をします。

```
$ grails s2-init-twitter
```

このコマンドを実行すると次の入力を求められます。

- Twitter API Key
- Twitter API Consumer Key
- Twitter API Consumer Secret

Consumer KeyとConsumer Secretは登録したTwitterアプリケーションの値をコピペしましょう。Twitter API KeyはTwitter Authenticationプラグイン独自の設定項目ですので任意の値を設定しておきましょう。今回の投票アプリでは「gdk48」としておきました。

■Twitter連携ボタンの設置

Twitterの認証を使ってアプリケーションにログインするボタンをViewに追加します。最低限必要なのはCSSと認証用ボタンのタグだけです。

- CSS

```
<link rel="stylesheet" href="${resource(dir:'css',
file:'twitter-auth.css')}" />
```

- 認証用ボタンのタグ

```
<twitterAuth:button/>
```

今回はどの画面からでもログインできるようにgrails-app/views/layouts/main.gspを次のようにします。

```
<!DOCTYPE html>
<html>
  <head>
    <title><g:layoutTitle default="Grails" /></title>
    <link rel="stylesheet" href="${resource(dir:'css',file:'main.css')}" />
    <link rel="shortcut icon" href="${resource(dir:'images',file:'favicon.ico')}" type="image/x-icon" />
    <link rel="stylesheet" href="${resource(dir:'css',file:'twitter-auth.css')}" />
    <g:layoutHead />
    <g:javascript library="application" />
  </head>
  <body>
    <sec:ifNotLoggedIn>
      <twitterAuth:button/>
    </sec:ifNotLoggedIn>
    <sec:ifLoggedIn>
     Logged in as <sec:loggedInUserInfo field="username"/>
    </sec:ifLoggedIn>
     <g:layoutBody />
  </body>
</html>
```

これで次のようなボタンが表示されます。試しにボタンをクリックし、Twitter認証できれば成功です。初めてTwitter連携を実装しようとすると、難しそうなイメージがありますが、とても簡単に実装できました。

投票の実装

Twitterでログインできるようになったので、投票機能を実装していきましょう。まずは投票を表すドメインクラスを作成します。

▼投票ドメインクラス

```
package gdk48

class Vote {

  Gist gist
  User voter

  static constraints = {
    gist nullable:false, unique:'voter'
    voter nullable:false
    }
}
```

このドメインはGistと投票者でユニークになります。つまり、投票は同じGistに対して1人1票しか投票できないものとします。

某48の総選挙とは仕様が異なりますが、何票でも投票できてしまうのは個人的にどうかと思いますのでGDK48の総選挙はこの仕様で行きたいと思います。

それでは、このドメインを使って投票できるように、コントローラと画面を実装します。

画面の方は前回作っていたlist.gspの内容を変更し、Gistごとにボタンを設置します。ボタンには現在の獲得票数も表示して、投票と取消しを切り替えられるようにします。次のように変更します。

```
<div class="gist">
  <div>
    <sec:ifLoggedIn>
      <button
       onclick="${remoteFunction(
          controller:'vote', action:'vote',
          params:[gistNo:gist.gistNo],
          update:'span_' + gist.gistNo)}">
        <span id="span_${gist.gistNo}">
         ${gist.voted ? '取消' : '投票'}
        (${gist.votedCount})</span>
      </button>
    </sec:ifLoggedIn>
  </div>
  <div>
    <script src="https://gist.github.com/${gist.
gistNo}.js"></script>
  </div>
</div>
```

投票と取消しはAjaxで出来るようにしてみました。このボタンをクリックしたときに呼び出されるコントローラは次のようになります。

```
package gdk48

class VoteController {

  def springSecurityService

  def vote = {
    def gist = Gist.findByGistNo(params.long('gistNo'))
    if (gist) {
      def vote = Vote.findByVoterAndGist(
         springSecurityService.currentUser, gist)
      if (vote) {
        vote.delete(flush:true)
        render text:"投票(${Vote.countByGist(gist)})"
      } else {
        new Vote(
                  voter:springSecurityService.currentUser,
                  gist:gist).save(flush:true)
        render text:"取消(${Vote.countByGist(gist)})"
        }
    } else {
       render status:404
    }
  }
}
```

投票と投票の取消しはAjaxで処理されますが、ここではJavaScriptは1行も書かずに実装できました。ポイントは投票ボタンのonclick属性に使用しているremoteFunctionです。

remoteFunctionを使用した個所が次のようにjQueryを使ったAjaxに展開されます。

```
jQuery.ajax({
  type: 'POST',
  data: {'gistNo': '1261979'},
  url: '/gdk48/vote/vote',
  success: function(data,textStatus){
    jQuery('#span_1261979').html(data);
  },
    error: function(XMLHttpRequest,
                    textStatus, errorThrown) {}
});"
```

remoteFunctionで指定したコントローラのアクションにパラメータを付けてPOSTし、指定したIDをもつHTML要素をレスポンスで上書きしてくれます。remoteFunctionには他にも指定できるオプションがいくつかあるので、詳しくはドキュメントを参照してください。

http://grails.org/doc/latest/ref/Tags/remoteFunction.html

■最後に

さて、前後編と2回に分けてGDK48投票アプリの作り方を紹介しました。去年の合宿0日目に急遽作ることになり、寝る時間を削って作っていたのが少し懐かしいですが、そろそろ今年も合宿の季節が近づいてきましたね。年を重ねる毎に1年が短くなってきているのをひしひしと感じています。JGGUG合宿2012はまだ未定ですが、今年はゆっくりと温泉につかれるようにしたいと思います。

series 08

GroovyFX で遊ぼう！（第 2 回）

関谷 和愛（せきや かずちか）
Groovy、Java、Apple 製品を愛するおっさんエンジニア。日本 Grails/Groovy ユーザーグループ運営委員長。
共著『プログラミング GROOVY』、共訳『Groovy イン・アクション』

前回の導入編では、JavaFXの概要、GroovyFXの入手やビルド方法、そして簡単なサンプルコードを使ってGroovyFXの特徴や、シーングラフ構築の基本を紹介しました。今回は、GroovyFXのもう一つの主要機能であるタイムライン構築を含む、アニメーションの基本について説明したいと思います。

その前に、前回からほぼ半年がたっており、GroovyFXとJavaFXにもいくつか大きなアップデートがありましたので、本稿の前半ではまずそれらについてまとめておきます。

JavaFX/GroovyFXアップデート

■JavaFX 2.1 リリース！

JavaFXの新バージョン、2.1の正式版がリリースされました。なんと言っても重要なのは、このバージョンからMac OS Xが正式サポートされたことです。G*界隈にはMacユーザが多いので、これは非常に大きな一歩と言えるでしょう。それ以外にも以下のような大きな機能追加がありました：

- H.264+AACを格納したMPEG-4コンテナの再生をサポート（これまではVP6+MP3/FLVだけ）
- WebView/WebEngineでJavaScriptからのJavaの呼び出しをサポート（これで双方向に！）
- UIコントロールやチャートの追加：ComboBox、StackedAreaChart、StackedBarChart
- Mac OS風システムメニューバー対応：MenuBar.setUseSystemMenuBar(true)
- OS標準のDirectoryChooser
- シーングラフをビジュアルに構築するツール、Scene Builderのベータ版リリース(Win/Mac)

WindowsでもMacでも最新のJDK7にJavaFX 2.1がバンドルされるようになり、インストールがより簡単になっています。また、次のバージョンである2.2のベータ版も公開されました。

■GroovyFX正式リリース！

JavaFXの進化にあわせ、GroovyFXもプレビュー段階を終え、ついに正式リリースとなりました！

新しいWebサイトが用意され、ソースはsvnからgithub上に移され、jarファイルでのダウンロードも可能になりました（もう自分でビルドする必要はありません）。何より大きいのはライブラリがmavenのcentralリポジトリに登録されたことです。これにより、@GrabでGroovyFXを利用することができるようになりました（後述のサンプルソースを参照）。また、Gradleを使う場合、build.gradleで次のように依存関係を記述できます：

```
// build.gradle (抜粋)
repositories { mavenCentral() }

dependencies {
  groovy 'org.codehaus.groovy:groovy-all:1.8.6'
  compile 'org.codehaus.groovyfx:groovyfx:0.2'
  compile files(<JavaFXのjfxrt.jarへのパス>)
}
```

GroovyFXのバージョン0.1はJavaFX 2.0に、0.2はJavaFX 2.1に対応しています。ベータ公開中のJavaFX 2.2にはGroovyFX 0.3が対応予定です(SNAPSHOT利用可)。本稿のサンプルではJavaFX 2.1/GroovyFX 0.2を使っています。

なお、GroovyFX 0.2では、一つ大きな変更がありました。GroovyFXの中核であるSceneGraphBuilderは、0.2からは明示的に生成する必要はなくなりました。startメソッドに渡すクロージャ内では、SceneGraphBuilderが自動的に生成され、クロージャのdelegateに設定されます。これによって、クロージャ内のメソッド呼び出しはこのSceneGraphBuilderオブジェクトに対して行われます。具体的なコードは後半のサンプルを参照してください。

アニメーション入門

さて、本題のアニメーションに進みましょう。JavaFXでのアニメーションには、トランジション(javafx.animation.Transition)と、タイムライン(javafx.animation.Timeline)を使ったアニメーションの2種類があります。どちらもjavafx.animation.Animationインタフェースを継承していて、以下のような共通の特性があります：

- play()/stop()などのメソッドで再生や停止を制御する
- onFinishedにクロージャを渡して終了時の処理を指定できる
- 自動反転のオン・オフ(autoReverse)、繰り返し回数(cycleCount)などのプロパティを持つ

では、トランジションとタイムラインアニメーションのそれぞれについて、使い方や具体例をみていきます。

■トランジション

トランジションは特定のシーングラフノードに適用する、あらかじめ用意された視覚効果のことです。

トランジションを定義するには、シーングラフ構築の記述で、対象ノードの子ノードとして（後続のクロージャ内に）適用するトランジションを記述します。下記の例では、rectangleノードを3秒間で360度回転させるrotateTransitionを定義しています：

```
rectangle(x: 100, y: 100, width: 50, height: 50) {
  transition = rotateTransition(3.s, from: 0, to: 360)
}
transition.play()
```

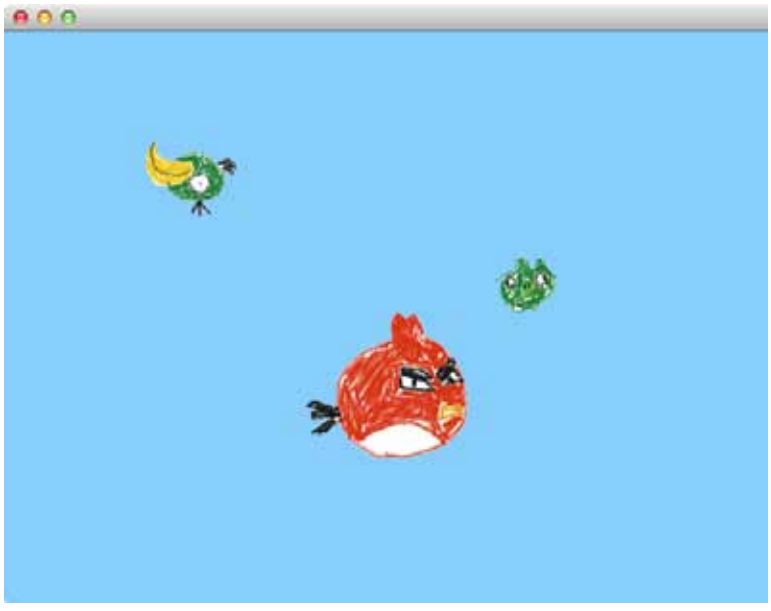
トランジションは種類によらず、最初の引数には必ず持続時間(javafx.util.Duration)を指定します。GroovyFXでは、Durationを指定する場合、500.ms、3.s、3.min、1.hといったリテラル風表現が可能です。

実際にこのトランジションを動作させるときは、play()メソッドを呼びます。

トランジションには次のようなものがあります：

名前	効果	主な引数	備考
fadeTransition	フェードイン／アウト	from/toで透明度指定(0:透明～1:不透明)	
rotateTransition	回転	from/toで角度指定(0～360度)	
scaleTransition	拡大／縮小	from/toで倍率指定	
pathTransition	移動（経路指定）	pathで経路(Shape)指定	
translateTransition	移動（終点/差分指定）	fromX/Y/Z（始点）からtoX/Y/Z（終点）またはbyX/Y/Z（差分）	
fillTransition	塗りつぶし色の変更	from/toで色指定	Shapeノードにのみ適用可能
strokeTransition	枠線色の変更	from/toで色指定	Shapeノードにのみ適用可能
pauseTransition	休止	なし	SequentialTransitionの中で間をとるためだけに存在

複数のトランジションを合成することもできます。SequentialTransitionは複数のトランジションを順番に実行し、ParallelTransitionは複数のトランジションを同時に実行します（回転しながらフェードアウトするなど）。



●トランジションの例：怒れる鳥たち

具体的なトランジションの例として、某人気ゲームのオープニングをイメージしたアニメーションを作ってみました。

```
@Grab('org.codehaus.groovyfx:groovyfx:0.2')
import static groovyx.javafx.GroovyFX.start

start {
  stage(visible: true) {
    scene(width: 640, height: 480, fill: lightskyblue) {

      // 緑の鳥：回転しながらpathに沿って飛ぶ（周期4秒）
      imageView {
        image("file:green_bird.png", width: 80, height: 80)
        greenAnim = parallelTransition {
          pathTransition(4.s,
                         path: quadCurve(startX: 0, startY: 200,
                                         controlX: 320, controlY: 0,
                                         endX: 640, endY: 200),
                         interpolator: linear, cycleCount: indefinite)
          rotateTransition(2.s, from: 0, to: 360,
                           interpolator: linear, cycleCount: indefinite)
        }
      }
      // 豚：拡大・縮小を繰り返しながらpathに沿って飛ぶ（周期3秒）
      imageView {
        image("file:pig.png", width: 120, height: 120)
        pigAnim = parallelTransition {
          pathTransition(3.s,
                         path: quadCurve(startX: 0, startY: 300,
                                         controlX: 320, controlY: 100,
                                         endX: 640, endY: 300),
                         interpolator: linear, cycleCount: indefinite)
          scaleTransition(1.s, from: 1.0, to: 0.5,
                          autoReverse: true, cycleCount: indefinite)
        }
      }
      // 赤い鳥：pathに沿って飛ぶ（周期2秒）
      imageView {
        image("file:red_bird.png", width: 160, height: 160)
        redAnim = pathTransition(2.s,
                                 path: quadCurve(startX: 0, startY: 400,
                                                 controlX: 320, controlY: 200,
                                                 endX: 640, endY: 400),
                                 interpolator: linear, cycleCount: indefinite)
      }

    }
  }
  // 各トランジションを再生
  redAnim.play()
  pigAnim.play()
  greenAnim.play()
}
```

（イメージファイルはこちらから入手してください:
https://dl.dropbox.com/u/132573/birds_images.zip）
実行は簡単です（@Grabを使っているのでGroovyFXをインストールしておく必要すらありません）:

```
groovy -classpath <JavaFXのjfxrt.jarへのパス>
birdsAnim.groovy
```

この例では、３つのimageViewノードを用意し、それぞれを別々の周期のpathTransitionで動かしています。１つのノードにはさらにrotateTransitionをかけて回転させ、別の１つにはscaleTransitionを適用して拡大・縮小を繰り返させています。トランジションの複合にはparallelTransitionを使っています。

コード中、cycleCountプロパティに指定しているindefiniteは無限回を表す疑似定数です。interpolatorはアニメーションの加減速を指定するプロパティです。トランジションの場合、デフォルト値はease_both（ゆっくりスタートして加速してゆき、終了に向けてまた減速する）ですが、このサンプルでは多くの箇所でlinearに設定して加減速のない一定なテンポに設定しています。

なお、この例はGroovyFX 0.2で書いているので、startメソッドのクロージャの冒頭で、SceneGraphBuilderの生成をすることなく、いきなりstage以下のシーングラフの構築を行っていることにも注意してください。

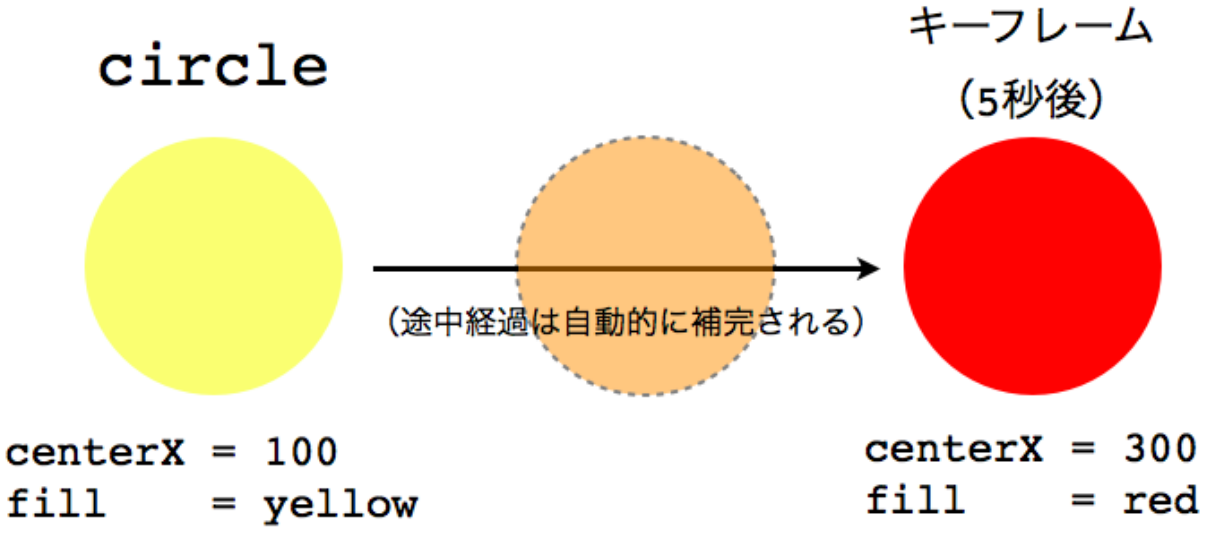
■タイムライン

タイムラインは、時間軸にそってオブジェクトのプロパティの変化を定義するものです。タイムラインはアニメーション以外にも利用可能ですが、オブジェクトの視覚的なプロパティを変化させればアニメーションを実現できます。オブジェクトの位置を変化させればオブジェクトは移動し、透明度を変化させればフェードイン・アウトし、大きさを変化させれば拡大・縮小する、といった具合です。

タイムラインは、複数のオブジェクト／プロパティを操ったり、独自のタイミング制御を定義したりできるので、トランジションに比べ、より汎用的なアニメーション手法と言えます。また、トランジションや別のタイムラインと複合させて、より複雑なアニメーションを実現することも可能です。

タイムラインは、一連のキーフレームで定義されます。キーフレームとは、ある時刻におけるオブジェクトの状態（プロパティ値）を定義したものです。キーフレームとキーフレームの間のオブジェクトの状態は自動的に補完され、連続した変化として再生されます。

次の例では、まずcircleノードを作成し、キーフレームで５秒後のcenterXとfillプロパティ値を定義することで、タイムラインアニメーションを作成しています。circleは５秒かけて中心座標が100から300になるよう水平に移動しつつ、黄色から赤に段階的に色が変わっていきます。



```
circle = circle(
  centerX: 100, centerY: 200, radius: 50,
  fill: yellow)
timeline = timeline {
  at (5.s) {
    change (circle, 'centerX') {
      to 300; tween ease_both  }
    change (circle, 'fill') { to red }
  }
}
timeline.play()
```

タイムラインを定義するとき、GroovyFX 0.1までは明示的にTimelineBuilderを生成する必要がありましたが、0.2からは（startメソッドのクロージャ内であれば）timelineノードをいきなり記述することができます。

timelineノードは子ノードとして1個以上のatノードを持ち、またautoReverse、cycleCount、onFinishedなどの引数をとることもできます。

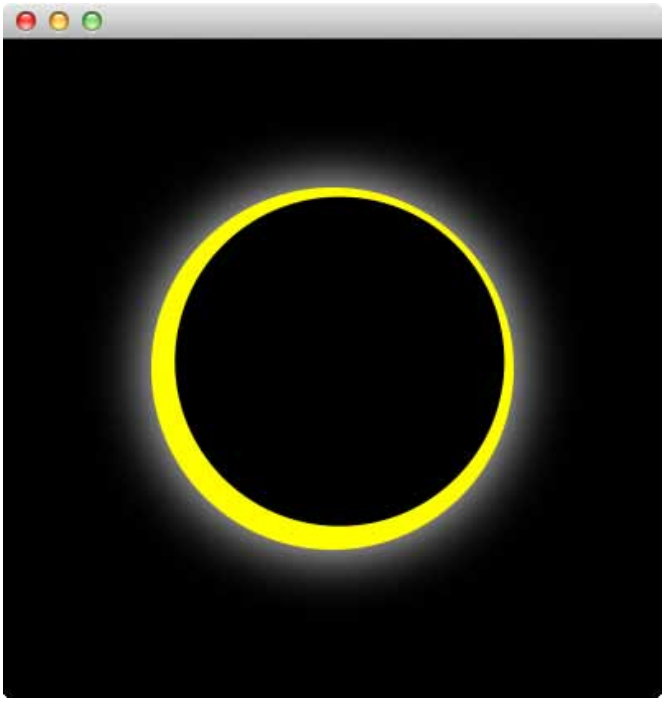
atノードはキーフレームを定義し、最初の引数には必ず時刻を指定します。onFinishedクロージャで任意の処理を指定することも可能です。atは任意個のchangeノードを持つことができます。

changeノードでは、どのオブジェクトの、どのプロパティが、どんな値になるのかを定義します。クロージャ内には1個のtoノード（必須）に加え、オプションでtweenノードを指定することもでき、これはプロパティ変化のinterpolatorとして働きます。タイムラインでのtweenのデフォルトはlinearです（トランジションとは異なるので注意）。

タイムラインを実際に動作させるときはトランジション同様、play()メソッドを呼ぶ必要があります。

●タイムラインの例：金環日食

ごく簡単なタイムラインアニメーションの例として、金環日食シミュレータを作ってみました。



```
@Grab('org.codehaus.groovyfx:groovyfx:0.2')
import static groovyx.javafx.GroovyFX.start

start {
  stage(visible: true) {
    scene(width: 400, height: 400, fill: black) {
      circle(centerX: 200, centerY: 200, radius:
110, fill: yellow) {
        effect dropShadow(color: white, radius:
50)
      }
      moon = circle(centerX: 400, centerY: 0,
radius: 100, fill: black)
    }
  }

  eclipse = timeline {
    at (60.s) {
      change (moon, 'centerX') { to 0 }
      change (moon, 'centerY') { to 400 }
    }
  }
  eclipse.play()
}
```

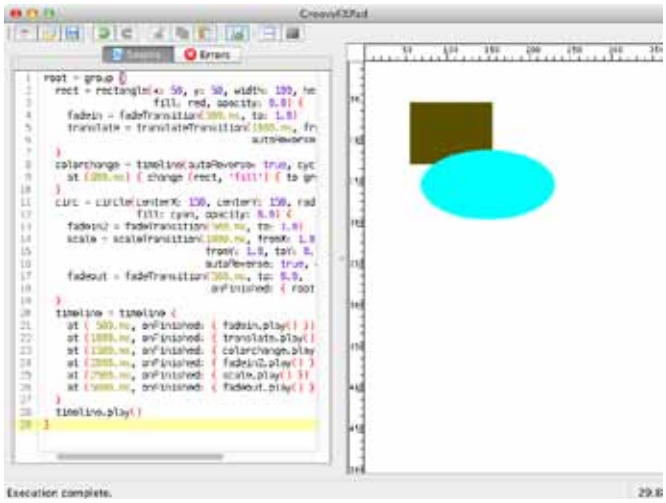
この例では、キーフレームとして60秒後のmoonオブジェクト（月を表す黒い円）の位置を定義することにより、月をゆっくりと動かしています。「どう動かすか」ということはまったくプログラムしていませんが、月の初期位置とキーフレーム定義から、JavaFXのランタイムが中間状態を補完し、自動的にアニメーションを作り出しているのです。

コラム - GroovyFxPad

G*なGUIアプリケーションフレームワーク、Griffonも先日ついに1.0がリリースされました！

このリリースには、GroovyFxPadという面白いサンプルアプリが同梱されています。左側のペインにGroovyFXなコード（シーングラフノードを返すように書く）を入力して実行すると、右側のペインに実行結果が即座に表示されます。シーングラフやアニメーションのパラメータをいろいろ変えながら出力結果をチェックできるので試行錯誤や勉強にはなかなか便利です。ぜひダウンロードして試してみてください：

http://griffon.codehaus.org/Download



まとめ

今回はJavaFX/GroovyFXのアップデート情報と、アニメーションの基本を説明しました。より複雑なアニメーションやGroovyFxPadなどについては下記リンクのブログ記事（およびそのリンク先）なども参考にしてください。

これでGroovyFXの主要機能であるシーングラフとタイムラインの構築はひとめぐりしたので、今後は個々の機能についてより深く掘り下げるか、あるいはもう少し本格的なアプリケーションに取り組んでみようと思います。感想やリクエストなどがあればTwitter: @kazuchika までぜひお知らせください。

■リンク

- JavaFX http://www.oracle.com/technetwork/java/javafx/overview/
- GroovyFX http://groovyfx.org/
- サンプルコード（怒れる鳥たち）https://gist.github.com/2963721
- サンプルコード（金環日食）https://gist.github.com/2963743
- GroovyFX 初の正式リリース（翻訳）- Groovy ラボ
  http://d.hatena.ne.jp/ksky/20120422/p1
- 複雑なアニメーション（GroovyFX版）- Groovy ラボ
  http://d.hatena.ne.jp/ksky/20120613/p1

series 04

GebではじめるWebテスト ～第３回 ～新機能活用編～

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。前回の Geb0.6 編からだいぶ時間が経ったこともあって、Geb はこの間に 4 度のバージョンアップを重ね、新たな API や振る舞いが変わるような変更が多く取り込まれました。そこで今回は新機能活用編として、バージョン 0.6 から今年 4 月にリリースされたバージョン 0.7.0 までに追加された主要な新機能を中心に解説したいと思います。

Geb 0.6.1 - 0.7.0 までの新機能・変更点

0.6.1 以降からは依存するライブラリのバージョンアップに伴う対応から、様々な新機能の追加、既存の振る舞いの見直しなど多くの変更が入りました。Geb のマニュアルの変更履歴には、全部で 31 もの新機能と重要な変更が記載されていますが、その中から主要なものを以下にまとめました。

カテゴリ	概要
依存ライブラリ	Selenium 2.9.0 に対応
	Spock 0.6 に対応
	select 要素を使用する場合は、selenium-support.jar が必須
新機能	暗黙のアサーション
	マウスやキーボード操作のインタラクション支援
	繰り返しのコンテンツを扱うモジュールリスト
	ウィンドウフレームへの対応
	複数ウィンドウへの対応
新機能	ブラウザのキャッシュ機構
	waitFor() のためのカスタムタイムアウトメッセージ
	Content DSL での別名パラメータ指定
振る舞い変更	Navigator#classes() の戻り値が Set からアルファベット順のソートが保証された List になった
	Page#convertToPath() は必要に応じて接頭辞のスラッシュを追加するようになった
	チェックボックスがチェックされていない状態は null の替わりに false を返すようになった

これらの中から、使用頻度が高いと思われるウィンドウ関連の 2 機能、モジュールリストと 0.7.0 から導入された暗黙のアサーションについて見ていきます。

■ウィンドウフレーム

一般的な Web のデザインではウィンドウをフレームで分割するような構造のものは少なくなってきましたが、エンタープライズ系のシステムではまだまだフレーム分割を行っている画面もあるかと思います。Geb 0.6.2 以降ではフレームを扱うことができるようになり、フレーム内のコンテンツへのアクセスが可能になりました。

次のサンプル画面では、分割された上部フレームの「update action」ボタンを押下すると、下部フレームのテキストが約 5 秒後に更新されます。

ここでのボタン押下からテキスト更新までの動作をテストしてみましょう。トップ画面とそれに対する Page クラスのソースコードは次のようになります。

```
<html>
<head>
</head>
<frameset rows="20%,*">
    <frame name="headerFrame" id="header"
                        src="header.html"></frame>
    <frame name="mainFrame" id="main"
                        src="main.html"></frame>
</frameset>
</html>
```

```
class NewFunctionsPage extends Page {

    static url =
'http://localhost:8080/hello/samples/frameTop.html'
    static content = {
        header { $('#header') }
        main { $('#main') }
    }
}
```

Spockを利用したテストコードは以下のようになります。

```
class NewFunctionsSpec extends GebReportingSpec {

  def "[update action]ボタンを押すと、5秒後に下フレームが更新される。"() {

    given:
    to NewFunctionsPage

    when:
    withFrame(header) {
              $('#updateActionButton').click() }

    then:
    withFrame(main) {
      assert waitFor(
        message:'タイムアウト！', 6
      ){ $('div') }.text() == 'update text!'
    } == null
  }
}
```

フレーム内の画面を扱う場合は、withFrame() を使用します。まず、when: ブロックで header を引数にして上部フレームのボタンを押下させます。そして、次に then: ブロックで下部フレームの検証を行います。

しかし、このサンプルでは更新タイミングを5秒後に遅らせているため、そのまま assert してしまうと即時検証され、テストが失敗してしまいます。このような場合は waitFor() を使用して更新されるまで待機させる必要があります。Geb のデフォルトの待機時間は5秒となっていますが、今回の更新タイミングも5秒後のため、場合によってはタイムアウトしてしまうこともあります。そのため、ここでは待機時間を6秒として設定しています。

ちなみに、withFrame() {…} == null としているのは、withFrame() の戻り値（ null となる）を評価しないと、テスト失敗となってしまうためです。これは Spock のバグらしく※、一時的な対応として、このような実装としています。

▼実行結果

また、わざと待機時間を短くしてテストを実行してみると、以下のように WaitTimeoutException がスローされ、waitFor() の引数に設定したカスタムメッセージが出力されます。

※ Geb の同様のテストコードに「can't put this in an expect block, some spock bug」とコメントがありました。

■複数ウィンドウ

別ウィンドウやタブに参照情報を表示させるような場面は、よくあるケースだと思いますが、Geb 0.6.2 から複数ウィンドウに対する制御も行えるようになりました。

先ほどのサンプル画面の上部フレームの「window action」ボタンを押下すると子画面が、「tab window action」ボタンを押下するとタブ画面が表示されます。これらの別画面に対するテストコードは以下のようになります。

```
  def "[window action]ボタンを押すと、子画面が表示される。"() {

    given:
    to NewFunctionsPage

    when:
    withFrame(header) { $('#windowActionButton')
                                        .click() }

    then:
    withWindow('childWindow') {
      assert $('title').text() == 'child window'
    } == null
  }
```

```
  def "[tab window action]ボタンを押すと、タブ画面が表示される。"() {

    given:
    to NewFunctionsPage

    expect:
    withNewWindow({ withFrame(header) {
      $('#tabWindowActionButton').click()
    } }) {
      assert $('title').text() == 'tab window'
    } == null
  }
```

別ウィンドウを扱う際には、withWindow() か withNewWindow() を使用します。子画面の検証には ウィンドウ名を指定して withWindow() を呼び、クロージャ内で検証をします。タブ画面ではウィンドウ名を指定していないため、withNewWindow() を使用しています。withNewWindow() はウィンドウを開く処理を行うクロージャを引数とするため、上部フレームのボタン操作を記述しています。

▼実行結果

■モジュールリスト

Web での検索結果や確認画面での表は、概ね同じ構成の繰り返しになります。そのような内容を含んだ画面をテストする際の仕組みとしてモジュールリスト機能が Geb 0.6.2 から導入され、さらに 0.7.0 からモジュールリストにインデックスや範囲を指定して操作できるようになりました。

次の単純なテーブル要素の画面をモジュールリストを使用してテストしてみましょう。

画面とそれに対する Page クラスは以下のようになります。

```
<html>
<head><title>G* ProductList</title>
</head>
<body>
<div>G* なプロダクト</div>
<table border="1">
  <tr><th>Product</th><th>Current Version</th>
  </tr>
  <tr><td>Groovy</td><td>1.8.6</td></tr>
  <tr><td>Grails</td><td>2.0.4</td></tr>
  <tr><td>Griffon</td><td>1.0.0</td></tr>
  <tr><td>Gradle</td><td>1.0</td></tr>
  <tr><td>Spock</td><td>0.6</td></tr>
  <tr><td>Geb</td><td>0.7.0</td></tr>
</table>
</body>
</html>
```

```
class RepeatingContentPage extends Page {

    static url =
'http://localhost:8080/hello/samples/repeating.html'
    static content = {
        gProducts { moduleList TableContent,
            $('table tr').tail() }
    }
}

class TableContent extends Module {

    static content = {
        cell { $('td', it) }
        productName { cell(0).text() }
        currentVersion { cell(1).text() }
    }
}
```

ここでのポイントは、RepeatingContentPage クラスの gProducts 内で $('table tr').tail() としているところです。これにより、テーブルのヘッダー行を除いてリスト化しています。

テストコードは以下のようになります。

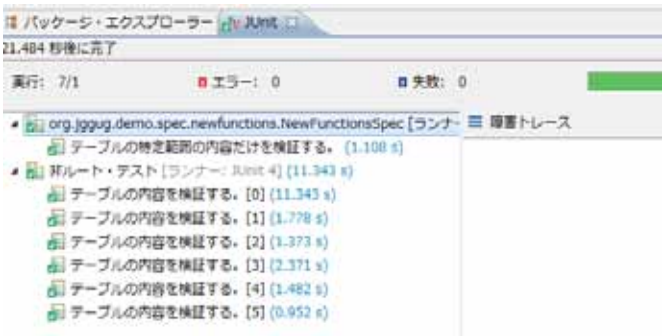
```
@Unroll
def "テーブルの内容を検証する。"() {

  when:
  to RepeatingContentPage

  then:
  gProducts[index].productName == productName
  gProducts[index].currentVersion == currentVersion

  where:
  index | productName | currentVersion
  0     | 'Groovy'    | '1.8.6'
  1     | 'Grails'    | '2.0.4'
  2     | 'Griffon'   | '1.0.0'
  3     | 'Gradle'    | '1.0'
  4     | 'Spock'     | '0.6'
  5     | 'Geb'       | '0.7.0'
}
```

ここでは Spock ではお馴染みのデータドリブンテストを利用して全ての内容について検証しています。範囲指定をしたテストは次のようになります。

```
class RepeatingContentPage070 extends Page {

  static url =
'http://localhost:8080/hello/samples/repeating.html'
  static content = {
    gProducts { index -> moduleList TableContent,
    $('table tr').tail(), index }
  }
}
```

```
def "テーブルの特定範囲の内容だけを検証する。"() {
    when:
    to RepeatingContentPage070

    then:
    gProducts(4..5)*.productName == ['Spock', 'Geb']
    gProducts(4..5)*.currentVersion == ['0.6', '0.7.0']
}
```

▼実行結果



G*Magazine vol.5

■暗黙のアサーション

0.7.0 から導入された機能の内、注目すべきなのが、暗黙のアサーション（implicit assertions）です。これは特定のブロックのコードを自動的にアサーションするように置き換える仕組みです。現時点では waiting 式と at 式に対して使用されています。

具体的にどのような動きをするのかを2つのバージョンの at 式で比較してみましょう。以下のコードは assert at(HelloPage) の行で失敗するテストコードです。

```
class HelloPage extends Page {
    static url = 'http://localhost:8080/hello'
    static at = { title == 'hello' }

    static content = {
        username { $('form').username() }
        greetButton(to:GreetingPage) {
            $('input', name:'greet')
        }
    }
}
```

```
@Test
void helloJGGUG() {
    to HelloPage
    assert at(HelloPage)

    username = 'JGGUG'
    greetButton.click()

    assert at(GreetingPage)

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

これを 0.6.3 と 0.7.0 で実行すると、次のようになります。

▼バージョン 0.6.3 での実行結果

障害トレース

Assertion failed:

```
assert at(HelloPage)
       |
       false
```

▼バージョン 0.7.0 での実行結果

障害トレース

Assertion failed:

```
title == 'hello'
|     |
Hello false
```

バージョン 0.6.3 ではテストコードの assert で判定されるため、ただ false としか出力されませんが、暗黙のアサーションが行われるバージョン 0.7.0 では、HelloPage クラスの at 式の中で assert title == 'hello' が実行されるため、具体的な値が出力されるようになります。さらに、テストコード内で明示的に assert at() と記述する必要もなくなるため、テストコードも読み易くなります。この暗黙のアサーションは AST 変換によって実装されており、開発者は全く意識することなく、恩恵を享受することができます。

まとめ

以上、実用的な新機能を中心にお伝えしましたが、如何でしたでしょうか？ウィンドウ機能の強化により、フレーム構造の画面や子画面を出すような場合にも対応することができるようになりましたし、waiting 式と併用することで、テスト実行時にも実際の画面遷移を想定した動作をさせることが可能です。また、モジュールリストや暗黙のアサーションは、テストコードをシンプルにし、失敗時の結果も分かり易くなりました。

さらに高機能で簡潔になった Geb に興味を持たれた方は、是非とも試してみてください。

■リンク

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

Grails Plugin 探訪

第 6 回

~ Build Test Data プラグイン ~

URL: http://grails.org/plugin/build-test-data
プラグインのバージョン: 2.0.2
対応するGrailsのバージョン: 2.0.0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、Build Test Data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_31
- Grails : 2.0.4

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

Build Test Dataプラグインとは

Build Test Dataプラグインは、ドメインクラスの制約から自動的にテストデータを作成するためのプラグインです。ドメインクラスが持つプロパティの制約を解析し、その制約に合う値を持つインスタンスを生成します。

プラグインのインストール

Build Test Dataプラグインのインストールは、grailsのinstall-pluginコマンドで行います。

```
$ grails create-app testdata
$ cd testdata
$ grails install-plugin build-test-data
```

ドメインクラスの作成

今回のサンプルのドメインクラスとして、本(Book)クラスと著者(Author)クラスを使用します。

```
package testdata

class Book {
    String title
    int price
    Date published
    String isbn13

    static belongsTo = [author: Author]

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

```
package testdata

class Author {
    String firstName
    String lastName

    static hasMany = [books: Book]

    static constraints = {
        firstName(blank: false)
        lastName(blank: false)
    }
}
```

上述のドメインクラスをベースに、必要に応じ変更します。

ドメインクラスのインスタンス生成

Build Test Dataプラグインをインストールすることで、ドメインクラスにbuildメソッド、およびbuildLazyメソッドがインジェクションされます。

■grails consoleコマンドの場合

grails consoleコマンドでは、そのままbuild/buildLazyメソッドが使用できます。ドメインクラスのインスタンスを生成、プロパティに値を設定し、保存します。

```
$ grails console
...
import testdata.Author
def author = Author.build()
println "First Name = '${author.firstName}'"
println "Last Name = '${author.lastName}'"
println "ID = '${author.id}'"
```

実行結果は次のとおりです。

```
First Name = 'firstName'
Last Name = 'lastName'
ID = '1'
```

■integration testの場合

integration testでも、そのままbuild/buildLazyメソッドが使用できます。

```
package testdata

import static org.junit.Assert.*
import org.junit.*

class BookServiceTests {

  @Before
  void setUp() {
  }

  @After
  void tearDown() {
  }

  @Test
  void testSomething() {
    def book = Book.build()
    dump(book)
  }

  private dump(book) {
    println "Title = '${book.title}'"
    println "Price = '${book.price}'"
    println "Published = '${book.published}'"
    println "ISBN13 = '${book.isbn13}'"
    println "Book's ID = '${book.id}'"
    println "First Name = '${book.author.firstName}'"
    println "Last Name = '${book.author.lastName}'"
    println "Author's ID = '${book.author.id}'"
  }
}
```

実行結果は次のとおりです。

```
Title = 'title'
Price = '0'
Published = 'Sun Jun 03 02:11:06 JST 2012'
ISBN13 = 'isbn13'
Book's ID = '1'
First Name = 'firstName'
Last Name = 'lastName'
Author's ID = '1'
```

■unit testの場合

unit testの場合、そのままではbuild/buildLazyメソッドが使用できません。テストクラスに、@Buildアノテーションでbuild/buildLazyメソッドを使用したいドメインクラスを指定します。複数クラスを対象にする場合、リストで指定します。

```
package testdata

import grails.test.mixin.*
import org.junit.*
import grails.buildtestdata.mixin.Build

@TestFor(BookController)
@Build(Book)
class BookControllerTests {

  void testSomething() {
    def book = Book.build()
    dump(book)
  }

  private dump(book) {
    println "Title = '${book.title}'"
    println "Price = '${book.price}'"
    println "Published = '${book.published}'"
    println "ISBN13 = '${book.isbn13}'"
    println "Book's ID = '${book.id}'"
    println "First Name = '${book.author.firstName}'"
    println "Last Name = '${book.author.lastName}'"
    println "Author's ID = '${book.author.id}'"
  }
}
```

実行結果は次のとおりです。

```
Title = 'title'
Price = '0'
Published = 'Sun Jun 03 02:11:06 JST 2012'
ISBN13 = 'isbn13'
Book's ID = '1'
First Name = 'firstName'
Last Name = 'lastName'
Author's ID = '1'
```

■プロパティに設定する値を変更したい場合

インスタンスのプロパティに設定する値を変更したい場合、build/buildLazyメソッドにマップ形式で変更したい値を指定します。

```
def book = Book.build(title: 'Grails Programming',
price: 3500)
```

実行結果は次のとおりです。

```
Title = 'Grails Programming'
Price = '3500'
Published = 'Sun Jun 03 02:57:41 JST 2012'
ISBN13 = 'isbn13'
Book's ID = '1'
First Name = 'firstName'
Last Name = 'lastName'
Author's ID = '1'
```

■buildLazyメソッドについて

buildメソッドは実行ごとにドメインクラスのインスタンスを生成しますが、buildLazyメソッドはbuild/buildLazyメソッドで生成した同じ値を持つインスタンスがあれば、そのインスタンスを再利用します。

```
    void testSomething() {
        def book = Book.build(
            title: 'Grails Programming',
            price: 3500,
            author: Author.build()
        )
        println '[book]'
        dump(book);

        def book2 = Book.build(
            title: 'Grails Programming 2',
            price: 3600,
            author: Author.build()
        )
        println '[book2]'
        dump(book2);
    }
}
```

実行結果は次のとおりです。IDがbookとbook2で違うことから、BookとAuthorのインスタンスがbuildメソッドの呼び出しごとに生成されるのがわかると思います。

```
[book]
Title = 'Grails Programming'
Price = '3500'
Published = 'Sun Jun 03 03:27:09 JST 2012'
ISBN13 = 'isbn13'
Book's ID = '1'
First Name = 'firstName'
Last Name = 'lastName'
Author's ID = '1'

[book2]
Title = 'Grails Programming 2'
Price = '3600'
Published = 'Sun Jun 03 03:27:09 JST 2012'
ISBN13 = 'isbn13'
Book's ID = '2'
First Name = 'firstName'
Last Name = 'lastName'
Author's ID = '2'
```

次に、Bookインスタンスは別々に、Authorインスタンスを再利用します。Authorのbuildメソッド呼び出しをbuildLazyメソッド呼び出しに変更します。

```
    void testSomething() {
        def book = Book.buildLazy(
            title: 'Grails Programming',
            price: 3500,
            author: Author.buildLazy()
        )
        println '[book]'
        dump(book);

        def book2 = Book.buildLazy(
            title: 'Grails Programming 2',
            price: 3600,
            author: Author.buildLazy()
        )
        println '[book2]'
        dump(book2);
    }
```

実行結果は次のとおりです。BookインスタンスのIDが異なるため、Bookインスタンスは別々に生成され、AuthorインスタンスのIDは同じのため、Authorインスタンスは再利用されていることがわかります。

```
[book]
Title = 'Grails Programming'
Price = '3500'
Published = 'Sun Jun 03 03:30:34 JST 2012'
ISBN13 = 'isbn13'
Book's ID = '1'
First Name = 'firstName'
Last Name = 'lastName'
Author's ID = '1'

[book2]
Title = 'Grails Programming 2'
Price = '3600'
Published = 'Sun Jun 03 03:30:34 JST 2012'
ISBN13 = 'isbn13'
Book's ID = '2'
First Name = 'firstName'
Last Name = 'lastName'
Author's ID = '1'
```

■インスタンスを保存したくない場合

build/buildLazyメソッドはインスタンスの保存を行いますが、インスタンスの保存をしたくない場合、buildWithoutSaveメソッドを使用します。

```
    void testSomething() {
        def book = Book.buildWithoutSave(
            title: 'Grails Programming',
            price: 3500,
            author: Author.buildWithoutSave()
        )
        println '[book]'
        dump(book);

        def book2 = Book.buildLazy(
            title: 'Grails Programming 2',
            price: 3600,
            author: Author.buildLazy()
        )
        println '[book2]'
        dump(book2);
    }
```

実行結果は次のとおりです。bookの方のIDがnullとなっていることで保存されていないことがわかります。

```
[book]
Title = 'Grails Programming'
Price = '3500'
Published = 'Sun Jun 03 03:53:50 JST 2012'
ISBN13 = 'isbn13'
Book's ID = 'null'
First Name = 'firstName'
Last Name = 'lastName'
Author's ID = 'null'

[book2]
Title = 'Grails Programming 2'
Price = '3600'
Published = 'Sun Jun 03 03:53:51 JST 2012'
ISBN13 = 'isbn13'
Book's ID = '1'
First Name = 'firstName'
Last Name = 'lastName'
Author's ID = '1'
```

デフォルト動作の変更

引数なしでbuild/buildLazyメソッドを呼び出した場合、プロパティに設定される値はプロパティ名やConstraintsをもとに設定されますが、固定の値を設定したい場合があります。

まず、Configファイルのテンプレートをインストールします。次のコマンドを実行すると、grails-app/conf/TestDataConfig.groovyが作成されます。

```
$ grails install-build-test-data-config-template
```

testDataConfig/sampleDataの中に、ドメインクラスごとにプロパティに設定したい値を指定します。

```
testDataConfig {
    sampleData {
        'testdata.Author' {
            firstName = 'Ichiro'
            lastName = 'Suzuki'
        }
        'testdata.Book' {
            title = 'Easy Grails Programming'
            price = 3000
        }
    }
}
```

引数を指定していないbuildメソッドを呼び出した実行結果は次のとおりです。TestDataConfig.groovyで指定した値がインスタンスに設定されているのがわかります。

```
Title = 'Easy Grails Programming'
Price = '3000'
Published = 'Sun Jun 03 10:24:50 JST 2012'
ISBN13 = 'isbn13'
Book's ID = '1'
First Name = 'Ichiro'
Last Name = 'Suzuki'
Author's ID = '1'
```

■設定する値を動的に変更したい場合

設定する値をインスタンスごとに動的に設定したい場合もあると思います。その場合、クロージャを使用することで指定可能です。

```
testDataConfig {
  sampleData {
    'testdata.Author' {
      firstName = 'Ichiro'
      lastName = 'Suzuki'
    }
    'testdata.Book' {
      def no = 1
      def delta = 1
      title = { "Easy Grails Programming ${no++}" }
      price = { 3000 + 100 * delta++ }
    }
  }
}
```

Bookクラスのインスタンスを3つ生成します。

```
    void testSomething() {
        def book1 = Book.build()
        println '[book1]'
        dump(book1);

        def book2 = Book.build()
        println '[book2]'
        dump(book2);

        def book3 = Book.build()
        println '[book3]'
        dump(book3);
    }
```

実行結果は次のとおりです。title, priceがインスタンスごとに違う値が設定されていることがわかります。

```
[book1]
Title = 'Easy Grails Programming 1'
Price = '3100'
Published = 'Sun Jun 03 10:42:03 JST 2012'
ISBN13 = 'isbn13'
Book's ID = '1'
First Name = 'Ichiro'
Last Name = 'Suzuki'
Author's ID = '1'

[book2]
Title = 'Easy Grails Programming 2'
Price = '3200'
Published = 'Sun Jun 03 10:42:03 JST 2012'
ISBN13 = 'isbn13'
Book's ID = '2'
First Name = 'Ichiro'
Last Name = 'Suzuki'
Author's ID = '2'

[book3]
Title = 'Easy Grails Programming 3'
Price = '3300'
Published = 'Sun Jun 03 10:42:03 JST 2012'
ISBN13 = 'isbn13'
Book's ID = '3'
First Name = 'Ichiro'
Last Name = 'Suzuki'
Author's ID = '3'
```

■設定する値をリセットしたい場合

先の設定はインスタンスの生成ごとに値が変わりますので、異なるテストメソッド中でインスタンスを生成した場合、異なる値となります。

```
    void testSomething1() {
        def book1 = Book.build()
        println '[book1]'
        dump(book1);

        def book2 = Book.build()
        println '[book2]'
        dump(book2);
    }

    void testSomething2() {
        def book1 = Book.build()
        println '[book1]'
        dump(book1);

        def book2 = Book.build()
        println '[book2]'
        dump(book2);
    }
```

実行結果は次のとおりです。

```
--Output from testSomething1--
[book1]
Title = 'Easy Grails Programming 1'
Price = '3100'
Published = 'Sun Jun 03 11:24:55 JST 2012'
ISBN13 = 'isbn13'
Book's ID = '1'
First Name = 'Ichiro'
Last Name = 'Suzuki'
Author's ID = '1'

[book2]
Title = 'Easy Grails Programming 2'
Price = '3200'
Published = 'Sun Jun 03 11:24:55 JST 2012'
ISBN13 = 'isbn13'
Book's ID = '2'
First Name = 'Ichiro'
Last Name = 'Suzuki'
Author's ID = '2'

--Output from testSomething2--
[book1]
Title = 'Easy Grails Programming 3'
Price = '3300'
Published = 'Sun Jun 03 11:24:55 JST 2012'
ISBN13 = 'isbn13'
Book's ID = '1'
First Name = 'Ichiro'
Last Name = 'Suzuki'
Author's ID = '1'

[book2]
Title = 'Easy Grails Programming 4'
Price = '3400'
Published = 'Sun Jun 03 11:24:55 JST 2012'
ISBN13 = 'isbn13'
Book's ID = '2'
First Name = 'Ichiro'
Last Name = 'Suzuki'
Author's ID = '2'
```

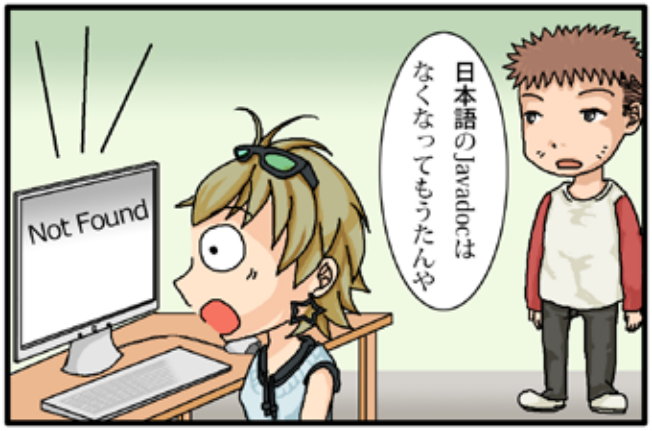
設定をリセットすることで、プロパティに設定する値を振り直すことができます。例えばテストメソッドごとに値を振り直したい場合、setUpメソッドで次のコードを実行します。

```
  void setUp() {
    grails.buildtestdata.TestDataConfigurationHolder.reset()
  }
```

```
--Output from testSomething1--
[book1]
Title = 'Easy Grails Programming 1'
Price = '3100'
Published = 'Sun Jun 03 11:26:09 JST 2012'
ISBN13 = 'isbn13'
Book's ID = '1'
First Name = 'Ichiro'
Last Name = 'Suzuki'
Author's ID = '1'

[book2]
Title = 'Easy Grails Programming 2'
Price = '3200'
Published = 'Sun Jun 03 11:26:10 JST 2012'
ISBN13 = 'isbn13'
Book's ID = '2'
First Name = 'Ichiro'
Last Name = 'Suzuki'
Author's ID = '2'

--Output from testSomething2--
[book1]
Title = 'Easy Grails Programming 1'
Price = '3100'
Published = 'Sun Jun 03 11:26:10 JST 2012'
ISBN13 = 'isbn13'
Book's ID = '1'
First Name = 'Ichiro'
Last Name = 'Suzuki'
Author's ID = '1'

[book2]
Title = 'Easy Grails Programming 2'
Price = '3200'
Published = 'Sun Jun 03 11:26:10 JST 2012'
ISBN13 = 'isbn13'
Book's ID = '2'
First Name = 'Ichiro'
Last Name = 'Suzuki'
Author's ID = '2'
```

終わりに

unitテストやintegrationテストで大量のデータが必要になる場合があると思いますが、特定の値である必要がなければ、Build Test Dataプラグインを使用してみるのはいかがでしょうか。



※現在は「http://docs.oracle.com/」ドメインで日本語javadocが復活しています。

▼ぐるーびーたん　第4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。今度は翻訳にチャレンジ!?

※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 2012.06.03

Grails

Grailsは、GroovyやHibernateなどをベースとしたフルスタックのWebアプリケーションフレームワークです。
URL: http://grails.org/
バージョン: 1.3.9, 2.0.4, 2.1.0.RC2

■更新情報

- 1.3.9では、コマンドオブジェクトのバリデーションでインジェクトされたBeanが使えないバグに対応しました。
- 1.3.9リリースノート: http://grails.org/1.3.9+Release+Notes
- 山本さんのブログ: http://d.hatena.ne.jp/mottsnite/20120524/1337881977
- 2.0.4では、GrailsLog4jLoggerAdapterのパフォーマンス改善や、コントローラを持たないビューやリソースのパフォーマンス改善、いくつかのバグ対応が行われています。
- 2.0.4リリースノート: http://grails.org/2.0.4+Release+Notes
- 山本さんのブログ: http://d.hatena.ne.jp/mottsnite/20120523/1337787556
- 2.1.0.RC2では、コントローラを持たないビューやリソースのパフォーマンス改善や、いくつかのバグ対応が行われています。
- 2.1.0.RC2リリースノート: http://grails.org/2.1.0.RC2+Release+Notes
- 山本さんのブログ: http://d.hatena.ne.jp/mottsnite/20120523/1337776401

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, 1.8.6, 2.0.0-rc-2

■更新情報

- 1.8.6では、配列にcontainsメソッドが追加されたり、リストにcollateメソッドが追加されたり、いくつかのバグ対応が行われています。
- 1.8.6リリースノート: http://jira.codehaus.org/secure/ReleaseNote.jspa?projectId=10242&version=18245
- 2.0.0-rc-2では、rc1からjarの依存関係などのバグ対応が行われています。
- 2.0.0-rc-2リリースノート: http://jira.codehaus.org/secure/ReleaseNote.jspa?projectId=10242&version=18550

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: 0.9.5

■更新情報

- 0.9.5では、コンパイル前にプロジェクトをクリーンするオプションが追加されたり、いくつかのバグ対応が行われています。
- 0.9.5リリースノート: http://docs.codehaus.org/display/GRIFFON/Griffon+0.9.5#Griffon095-095

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: 1.9.7

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: 1.0-rc-3

■更新情報

- 1.0-rc-3では、Groovyソースのコンパイル時に発生する例外に対応されています。
- 1.0-rc-3リリースノート: http://wiki.gradle.org/display/GRADLE/Gradle+1.0-rc-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: 1.2

■更新情報

- 1.2では、Groovy 1.8.6、App Engine SDK 1.6.6にアップグレードし、Mavenから利用可能となりました。
- 1.2リリースノート: http://gaelyk.appspot.com/download

Google App Engine SDK for Java

Google App Engine SDK for Javaは、JavaでGoogle App Engine用のWebアプリケーションを開発するためのSDKです。
URL: http://code.google.com/intl/ja/appengine/
バージョン: 1.6.6

■更新情報

- 1.6.6では、管理コンソールのQuota DetailsにSearch APIの項目が追加されたり、Search APIに関するクラスや属性が一部非推奨になったり、またいくつかのバグ対応が行われています。
- 1.6.6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

Groovy++

Groovy++は、Groovy言語に対して静的な機能を拡張します。
URL: http://code.google.com/p/groovypptest/
バージョン: 0.9.0

Spock

Spockは、JavaやGroovy用のテストと仕様のためのフレームワークです。
URL: http://code.google.com/p/spock/
バージョン: 0.6

■更新情報

- 0.6では、Groovy 1.8, 2.0, Grails 2.0のサポート、Mockや状況表示の改良、またいくつかのバグ対応が行われています。
- 0.6リリースメール: https://groups.google.com/group/spockframework/browse_thread/thread/c720a3871114666

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: 0.10

■更新情報

- 0.10では、-vオプションでGroovyServのバージョン表示や、–allow-fromオプションでループバックアドレスを除いた'localhost'の解決ができるようになり、いくつかバグ対応が行われています。
- 0.10リリースノート: http://kobo.github.com/groovyserv/changelog.html#version-0-10-2012-03-30

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: 0.7.0

■更新情報

- 0.7.0では、フォームコントロールのショートカットがページやモジュールコンテンツでも動作するようになり、waitFor()メソッド呼び出しでのタイムアウトメッセージをカスタマイズできるようになったりと、いくつか新機能の追加や改良がされています。
- 0.7.0リリースノート: http://www.gebish.org/manual/current/project.html#070

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: 0.8.2

■更新情報

- 0.8.2では、Groovyのバージョンが1.8.4になり、Grails 2.0対応といくつかのバグ対応がされています。
- 0.8.2リリースブログ: http://grails.io/post/15568051467/gmock-0-8-2-released-with-grails-2-0-support

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: 0.5.2

■更新情報

- 0.5.2では、エスケープされたURIクエリーパラメータの対応や、AsyncHTTPBuilderのバグ対応、HttpClient生成のカスタマイズ対応がされています。
- 0.5.2チェンジログ: http://groovy.codehaus.org/modules/http-builder/changes.html

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: 0.17

■更新情報

- 0.17では、新しいルールの追加や更新・拡張、Groovy 2.x系のサポートや、いくつかバグ修正されています。
- 0.17リリースノート: http://sourceforge.net/projects/codenarc/files/codenarc/CodeNarc%20Version%200.17/

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: 0.5

■更新情報

- 0.5では、新しいメトリクスやメトリクスセットDSLのの追加、メソッドレベルのメトリクスの拡張などされています。
- 0.5リリースノート: http://sourceforge.net/projects/gmetrics/files/gmetrics-0.5/

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: 1.2.5

■更新情報

- 1.2.5では、Groovy 1.8対応や、Spockと一緒に使用した際のコンパイルエラー対応されています。
- 1.2.5リリースノート: http://blog.andresteingress.com/2012/01/16/gcontracts-1-2-5-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://groovyfx.org/
バージョン: 0.1

■更新情報

- 0.1では、Grab対応や、TableCellやListCellがより簡単に作成できるようになり、またいくつかバグ修正されています。
- 0.1リリースノート: http://jira.codehaus.org/secure/ReleaseNote.jspa?version=18233&styleName=Text&projectId=12531

GBench

GBenchは、Groovyのためのベンチマーク・フレームワークです。
URL: http://code.google.com/p/gbench/
バージョン: 0.3

■更新情報

- 0.3では、ベンチマークの自動化とマイクロベンチマークを計算するBenchmarkBuilderの追加や、@Benchmarkのシステムプロパティの変更が行われています。
- 0.3リリースノート: http://code.google.com/p/gbench/wiki/ReleaseNotes030

Betamax

Betamaxは、HTTP通信の内容を保存し再生するテストツールです。
URL: https://github.com/robfletcher/betamax
バージョン: 1.0

Caelyf

Caelyfは、Groovyで記述するCloud Foundry用のライトウェイトなツールキットです。
URL: http://caelyf.cloudfoundry.com/
バージョン: 0.1

Vert.x

Vert.xは、非同期アプリケーション開発のためのフレームワークです。
URL: http://vertx.io/
バージョン: 1.0.1

■企業スポンサー

NTTソフトウェア

Delight Technologies

NEWCAST

metabolics

■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,昨年に引き続き2012年度も個人サポータを募集いたします。.

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.5 2012.06
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