prerelease

Magazine

Japan Grails/Groovy User Group

G* Magazine 準備号のごあいさつ

この度、日本 Grails/Groovy ユーザーグループでは、従来の情報配信の形態を、メールマガジンから eBook 形式の本誌「G* Magazine」に変更することとなり、準備号として本号を発行するに至りました。
内容につきまして、皆様からのご意見、ご要望などお待ちしております。
今後ともどうぞよろしくお願いいたします。

G* Magazine 編集長　川原正隆

Contents

Column

Grails - JavaEE 開発をライトにする フルスタックフレームワーク 4

Series

Griffon 不定期便 10

Grails Plugin 探訪 第 1 回 14

Information

リリース情報 15

column 01

Grails - Java EE 開発をライトにするフルスタックフレームワーク

株式会社ニューキャスト 山本 剛 (やまもと つよし)

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。

Grails は、Java EE で実績のある Spring フレームワーク、Hibernate (O/R マッピング) 等をベースに Groovy DSL を活用して実装された、Java 環境で効率よく Web アプリケーション開発を行うことができるフルスタックフレームワークです。

多くのダイナミックフレームワークのように、DRY(Don't Repeat Yourself)、CoC(設定より規約) という哲学に基づいて実装されています。Grails での Web アプリケーション開発では、主に Groovy を使用した DSL(ドメイン固有言語) で記述します。DSL でプログラムを記述する事によって、その内容から動作に必要な部分は動的に生成されます。この実装によって、従来の Java 環境での Web アプリケーション開発の複雑な部分を隠蔽して軽快に開発を進めることができます。

Grails で主に使用する言語 Groovy は、JVM 上で稼働するオブジェクト指向型動的スクリプト言語です。Java とシームレスに統合されているので、Java 開発者にとっては学習コストが少なくてすみ、多くの Java 資産も活用できます。

フレームワークのベースが Spring フレームワークであることから、Grails は言わば形を変えた Spring-MVC であり、Java 資産の活用という意味では、既存の Spring ビーンを利用したい場合、特に効果を発揮すると言えるでしょう。

[フレームワークの概略図]

すぐに始められる Web 開発

Java 環境での Web 開発には、アプリケーションサーバ、データベース、ビルドツールはもちろん、アプリケーションを実装するための様々なライブラリ等が必要になります。さらには、それらのツールを使用するための設定、実装を行うための定義が必要です。Grails では、Web 開発に必要なそれらの内容を、素早く簡単に扱えるように提供してくれるので、単純な Web 開発であれば、Grails を利用することで、Grails 以外に何も準備する必要がありません。Grails で、すぐに使用できる機能を簡単にリストしてみました。

- Hibernate 上に構築された、簡単に利用できるオブジェクト・リレーショナル・マッピング (ORM) レイヤー。
- 開発時に活用できる組込 HSQLDB。※ Hibernate に対応しているデータベースは全て使用可能。
- 表現豊かなビューテクノロジー Groovy Server Pages (GSP)。
- Spring MVC を利用したコントローラレイヤー。
- ビルドなどを行うコマンドラインスクリプト環境。Groovy 版の Ant Gant を使用。
- リロード可能に設定された組込アプリケーションサーバ Tomcat。
- 組込 Spring フレームワークによる依存注入 (DI)。
- Spring フレームワークの MessageSource で実装された国際化 (i18n) 対応。
- Spring フレームワークのトランザクション実装によるサービスレイヤーのトランザクション。

Grails をインストール

Grails 公式サイトのダウンロードページ http://grails.org/Download から最新バージョンをダウンロードして、任意の場所に解凍します。そして、解凍先を GRAILS_HOME として環境変数を設定、GRAILS_HOME/bin を PATH に追加します。Grails を動作させる環境には Java が必要です。設定されていない場合は、環境変数 JAVA_HOME の設定もおこなってください。※以下は Linux での例です。

```
% export GRAILS_HOME=/opt/grails-1.3.5
% export PATH=$GRAILS_HOME/bin:$PATH
```

Grails のプロジェクトを作成する

Grails の Web 開発で一番最初に行う作業は、プロジェクトの作成です。任意のディレクトリで、「grails create-app」コマンドを実行するとプロジェクトの基礎となる内容を含んだプロジェクトディレクトリが生成されます。

```
% grails create-app myapp
% cd myapp
```

生成されたプロジェクトディレクトリ

プロジェクト作成のコマンドを実行すると、図のようなプロジェクトディレクトリが生成されます。その階層にある grails-app が Grails アプリケーションのメインとなるソースディレクトリになります。grails-app 以下の階層に、それぞれの機能・目的別に階層が準備されています。それぞれ目的別の階層以下にソースファイルを追加することで目的の動作を実装します。

```
myapp  ← 生成されたプロジェクトディレクトリ
   |-- application.properties
   |-- grails-app       ← Grailsアプリケーションディレクトリ
   |    |-- conf        ← 設定用DSL
   |    |-- controllers ← コントローラ(MVCのCにあたる)
   |    |-- domain      ← ドメインクラス(MVCのMにあたる)
   |    |-- i18         ← 国際化プロパティ
   |    |-- services    ← ビジネスロジック・サービス
   |    |-- taglib      ← タグライブラリ
   |    |-- utils       ← 文字のエンコード・デコード用Codec
   |    `-- views       ← ビュー・レイアウト
   |-- lib              ← jarファイル
   |-- scripts          ← 追加コマンドスクリプト用ディレクトリ
   |-- src              ← その他のソースコード
   |-- test             ← テストコード
   `-- web-app          ← Webリソース
```

起動とデプロイ

run-app コマンドを実行すると、コンパイル実行中などのメッセージが流れ、組込 Tomcat でアプリケーションが起動します。起動完了したら最後に表示された URL http://localhost:8080/myapp を見てみましょう。まだ何も機能はないですが Grails アプリケーションが動作したのがわかります。また停止は、コントロール +c で行います。

```
% grails run-app
Welcome to Grails 1.3.5 - http://grails.org/
.... 省略 ...
Running Grails application..
Server running. Browse to http://localhost:8080/
myapp
```

[ブラウザで確認した画面]

サーブレットコンテナがリロード可能に設定されているので、コントローラ、サービス、ビュー等は、サーバを動作させたまま開発を行うことができます。Grails で開発したアプリケーションをアプリケーションサーバにデプロイするには、grails war コマンドを実行することで、war ファイルが生成されるので、実運用時は war をアプリケーションサーバにデプロイすることになります。war ファイルを生成できるため、デプロイに関しては従来の Java 開発と変わりません。

ドメインクラス GORM

では、ここから Grails での Web 開発の基礎を順に説明していきます。

先ずはドメインクラスです。永続化するためのモデルは Grails ではドメインクラスになります。ドメインクラスには、必要なプロパティと制約、OR マッピングの定義をするマッピング DSL 等を記述します。記述した定義をもとに、Hibernate の定義、テーブルの生成等を自動で行ってくれます。本稿では割愛しますが、既存の Hibernate 定義と Java で記述したモデル等を、連携・使用することも可能です。

データ参照・クエリには、ダイナミックファインダ、クライテリアビルダ等、Groovy DSL を活用した仕組みで操作を行います。もちろん Hibernate をベースに実装されているので、HQL(Hibernate クエリ言語) も利用可能です。

これらのドメインクラスと、参照の仕組みを、GORM (ゴーム Grails O/R マッピング) といいます。本稿では、生成・参照の一部のみを紹介します。詳しくは公式リファレンスを参照しましょう。(http://grails.org/doc/latest/guide/5.%20Object%20Relational%20Mapping%20(GORM).html)

ドメインクラス生成コマンド

以下のコマンドで、パッケージ名から生成したいドメイン名称を指定します。コマンドを実行すると、grails-app/domains ディレクトリ以下に、パッケージ階層を含むベースになるソースコードが生成されます。

```
% grails create-domain-class com.example.Book
```

次のコードサンプルでは、実際の使用例に沿って、コード内で説明していきます。

Book.groovy

```
package com.example

class Book {
  // マッピング DSL：テーブル名、データ型の指定等が可能。
  static mapping = {
    id generator:'sequence',
      params:[sequence:'book_id_seq']
    columns {
      comment sqlType:'varchar(4000)'
    }
  }
  // プロパティ：テーブルのフィールド名としても使用される。
  // いちばんシンプルなドメインクラスは、この部分のみ記述すれば OK。
  String title
  String author
  String comment

  // 制約：バリデーション、ビューで自動生成される、
     フォーム定義にも使用されます。
  static constraints = {
    title nullable:false,blank:false
    author nullable:false,blank:false
    comment nullable:false,blank:false,
            maxSize:4000
  }
}
```

GORM / データ操作

データ操作用の機能は、ドメインクラスへ動的に追加されたメソッドを使用します。

```
def book = new Book() // 新規インスタンス生成
book.title = 'Grails 徹底入門 '
// .... 省略 ....
// データの保存
book.save()
// データの取得
book.get(1) // id を指定して取得
book.getAll(7,5,8) // 複数 id を指定して取得
book.list() // 全リスト

// データの削除
def book = Book.get(1)
book.delete()
```

GORM / ダイナミックファインダ

ダイナミックファインダとは、検索対象のフィールド名と条件を、メソッドにキャメルケースで記述してデータを検索する仕組みです。内部で動的にクエリが解釈され実行されます。このダイナミックファインダを使用することによって、より読みやすくわかりやすいコードを記述することが可能です。

説明用の例として、以下のドメインクラスを使用します。

```
class Book {
        String title
        Date releaseDate
        Author author
}
```

ドメインクラス Book に対しての、ダイナミックファインダの記述は、findBy、findAllBy、listOrderBy、countBy 等で始まり、フィールド名と条件などをつなげて記述します。（詳しくは公式ドキュメントも参考にしてください）

```
def book = Book.findByTitle("Grails 徹底入門 ")
def books = Book.findAllByTitleLike("Groovy%")
book = Book.findByReleaseDateBetween( firstDate, secondDate )
book = Book.findByReleaseDateGreaterThan( someDate )
book  =  Book.findByTitleLikeOrReleaseDateLessThan(
"%Something%",
          someDate )
c = Book.countByReleaseDateBetween(firstDate, new Date())
def results = Book.listOrderByTitle(max:10)
```

GORM / クライテリアビルダ

クエリを DSL で記述するもう一つの Grails でのクエリ発行方法としてクライテリアビルダがあります。この内容も詳しくは公式ドキュメントを参考にしてください。（※今後、本誌で GORM 特集を執筆する予定です。）

```
def c = Account.createCriteria()
def results = c {
    between("balance", 500, 1000)
    eq("branch", "London")
    or {
        like("holderFirstName", "Fred%")
        like("holderFirstName", "Barney%")
    }
    maxResults(10)
    order("holderLastName", "desc")
}
```

コントローラとビュー

「create-controller」コマンドを実行してコントローラのコードを生成します。同時に grails-app/view ディレクトリ以下にコントローラと同じ名称のビュー用のディレクトリが生成されます。

コントローラ生成コマンド：

パッケージ名から生成したいコントローラ名称を指定します。

```
% grails create-controller com.example.Book
```

コントローラをコマンドで生成すると、grails-app/controllers ディレクトリ以下に、BookController.groovy ファイルが生成されます。少しわかりにくいですが、コントローラを生成するときに指定する名称は、サフィックスになる Controller は追加不要です。

一先ず、簡単なコントローラへのアクション記述方法を説明します。この辺りの詳しい説明は、公式ドキュメントの http://grails.org/doc/latest/guide/6.%20The%20Web%20Layer.html を参考にしてください。

BookController.groovy

```
class BookController {
  // シンプルにテキストを返す
  def index = {
    // ここに処理を書く
    render text:"Hello"
  }

  // ビューファイル (gsp) を指定。
  def foo = {
    // ここに処理を書く
    render view:"example",models:[name:'tyama']
  }
}
```

動的スカッフォルド

コントローラに scaffold プロパティを記述する事によって、シンプルな CRUD アプリケーションを動的に動かす事ができる動的スカッフォルド機能があります。サンプルコードのように単純に scaffold プロパティに true を指定するか、CRUD アプリケーションを生成したい対象のドメインクラス名を指定します。

BookController.groovy

```
class BookController {
   // あるいは、対象のドメインクラス名
  def scaffold = true
}
```

静的スカッフォルド

動的スカッフォルドで動作する CRUD アプリケーションのソースコードを書き出して使用する事も可能です。

コードを生成したいコントローラ名称を指定

```
% grails generate-all com.example.Book
```

コマンドを実行すると、BookController.groovy ファイルとそれぞれ必要なビュー用の gsp ファイルが生成されます。

スカッフォルドを活用した開発の流れ

Grails で Web 開発をする際に、いちばんスタートしやすい流れが、スカッフォルドをベースにした開発になります。まず始めにドメインクラスを作り、動的スカッフォルドでデータを登録する等、確認調整を行いながら開発を進めます。そして、ドメインクラスの設計がまとまった時点で、静的スカッフォルドでソースコードを生成します。書き出されたソースコードをもとにして、さらに調整をします。もちろん、簡単なマスタ扱いの内容等、単純な CRUD アプリケーションで問題無い内容は、そのまま動的スカッフォルドに良いと思います。但し、どんなフレームワークも同じだと思いますが、容易に実装内容を知られてしまうため、アクセス制限をするなどセキュリティーの考慮は必要です。

GSP - Groovy Server Pages でビューを作る

Grails では、JSP と同じようなページコーディングの仕組みを Groovy で実装した GSP(Groovy Server Pages) を使用してページの実装を行います。同時に JSP を使用することも可能です。以下のサンプルコードのように HTML の中に Grails タグライブラリを記述したり、変数はエクスプレッションが使用できます。

詳しくは公式ドキュメントも参考にしてください。 http://grails.org/doc/latest/guide/6.%20The%20Web%20Layer.html#6.2%20Groovy%20Server%20Pages

GSP サンプルコード example.gsp

```
<html>
  <head>
    <title> タイトル </title>
    <meta name="layout" content="main" />
  </head>
<body>
  <div id="page">
    <h1>Hello</h1>
    <ol>
      <g:each var="item" in="${list}">
      <li>${item.name} - ${item.message}</li>
      </g:each>
    </ol>
    <g:link controller="book" action="list"> リストへ </g:link>
  </div>
</body>
</html>
```

シンプルなタグライブラリ

Grails では、タグライブラリが簡単に作成できて、JSP でのタグライブラリで必要となる、tld ファイルや web.xml の記述も不要です。必要であれば、いつでもコマンドで追加できます。簡単に作成できるだけでなく、タグライブラリの実装を、コントローラ内部でも呼び出して使用できます。

公式ドキュメントはこちらになります。http://grails.org/doc/latest/guide/6.%20The%20Web%20Layer.html#6.3%20Tag%20Libraries

パッケージ名から生成したい名称を指定します。

```
% grails create-tag-lib com.example.MyUtil
```

MyUtilTagLib サンプルコード MyUtilTagLib.groovy

```
class MyUtilTagLib {
  // タグのネームスペース
   static namespace = "myutil"
  // 返値をオブジェクトで返す場合の指定。
   static returnObjectForTags = ['content']

   // <myutil:hello name="tyama"/> タグ。
   // out ( アウトプットストリーム ) に返す
   def hello = {attrs, body ->
     out<<"Hello, ${attrs.name}"
   }

   // オブジェクトを返すタグ。
   // コントローラで myutil.content() として使用できます。
   def content = {attrs, body ->
     CmsContent.findByCode(attrs.code)?.content
   }
}
```

プラグイン

Grails での開発の醍醐味ともいえる機能プラグインを紹介します。

各種機能、開発支援ツール等様々な機能を提供してくれるのが Grails プラグインです。実際に Grails コア本体もプラグインで構成されています。そのため Grails では内部的に柔軟にモジュール化が可能に設計されています。

JavaScript のライブラリ、セキュリティ対応、クラウド支援ツール等、現在約 400 のプラグインがセントラルリポジトリ（Grails 公式が管理しているリポジトリ）存在しています。Grails プラグインは、便利な機能を提供するだけではなく、独自のプラグインを開発することも可能です。なので、プラグインを活用する事によって、Grails 本体の各機能と同じくらいにわかりやすい機能の提供や、チーム開発での効率化、再利用性の高さも提供してくれます。

プラグインは公式サイトのプラグインページより検索可能です。

Grails 公式サイトプラグイン検索 http://grails.org/plugin/home

プラグインのインストールコマンド：

セントラルリポジトリに存在するプラグインのインストール

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

プラグインの zip パッケージを指定してインストールすることも可能です。

```
% grails install-plugin /path/to/plugins/grails-view-template-0.1.zip
```

プラグイン開発

プラグイン開発方法は、Grails での通常の Web 開発とほとんど同じです。本稿では詳しい開発方法はスペースの関係で割愛しますが、プロジェクトの作成方法とパッケージ方法だけ説明しておきます。（※今後、本誌でプラグイン開発特集を執筆する予定です。）

プラグインのプロジェクト生成は create-plugin コマンドで行います。プラグインプロジェクトの内容はほとんど通常の Grails プロジェクトと同じです。違いは基本的にはプラグインディスクリプタファイルの有無だけです。

プラグインプロジェクトを生成したら、通常の Grails での開発と同じように開発を行い、package-plugin コマンドでプラグインパッケージになる zip ファイルを生成します。

プラグインプロジェクト生成コマンド：

```
% grails create-plugin mytags
```

プラグインパッケージコマンド：

```
% grails package-plugin
```

これも本稿では割愛しますが、Grails プラグインでは、セントラルプラグインリポジトリと同じように、プラグインを独自のプラグインリポジトリでも管理することが可能です。また、Maven でのプラグイン管理も可能です。

Grails で Web 開発をする際に、機能分けなどを行いプラグイン分けをして、チーム専用のプラグインリポジトリを活用する事によって、再利用可能にモジュール化した開発を行う事ができます。

また、将来的にプラグインの OSGi 対応の計画等があるので、Grails で Web 開発する際は、プラグインを活用する事をおすすめします。

Web 開発を支援する多くの機能

今回紹介しきれなかった他の機能を簡単にリストしておきます。

（※今後、本誌で順番に紹介していく予定です。）

- Apache Ivy をベースに構築された依存性管理 DSL
- リクエストフィルタ機能
- URL マッピング DSL と REST 対応
- コンテントネゴシエーション DSL

- Web フロー。Spring Web Flow の DSL 実装
- テスト駆動開発 (TDD) 支援環境。Spock プラグインを使う事で BDD も可能
- URL エンコード・デコードを行う Codec
- Ajax 機能をラップした便利なタグライブラリ

まとめ

今回の内容では、Grails での Web 開発の導入部分をかけ足で紹介しました。 Grails には、Java での Web 開発に必要な内容がほとんど揃っており、それらの技術をシームレスに活用できます。また、わかりやすいルールを基に DSL で記述しながらコードを書くという事は、見やすさ、わかりやすさを提供します。

Grails は、常に最先端の Web 開発で要求される機能をプラグインでわかりやすく DSL などで提供してくれます。しかし、その反面、不安定さも目立ったりします。ただ、欧米でコミュニティの活動が活発なので解決も速いです。

今回の執筆の時点での最新版は 1.3.5 であり、この 1.3 系の Grails は過去のリリースに比べ、スループットも良くなり、かなり安定し、機能も充実してきています。そして近々さらに向上した次期バージョン 1.4 も控えており、さらなる安定と機能向上が期待されます。

オールインワンで、エンタープライズなアジャイル Web アプリケーション開発ができる Grails を是非活用してみてはどうでしょうか。

今後、本誌において Grails の解説を執筆していく予定です。ご期待ください。

リンク

Grails 公式サイト http://grails.org
日本 Grails/Groovy ユーザーグループ http://www.jggug.org
Grails ドキュメント http://grails.org/doc/latest
Grails ドキュメント（日本語※現在翻訳中）
http://grails.jp/doc/latest

Griffon 不定期便

series 01

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：『Spring による Web アプリケーションスーパーサンプル』『Seasar2 による Web アプリケーションスーパーサンプル』

Griffon はデスクトップアプリケーションを開発するフレームワークです。Grails の機能を多く取り入れており、Grails を使って Web アプリケーションを開発した事がある人には親しみやすいフレームワークです。

Groovy では SwingBuilder を使ってデスクトップアプリケーションを作成する事も出来ますが、それなりのものを作るには少し力不足です。（ちょっとした Web アプリケーションを作るのに Groovlet だけでやってもいいですが、Grails を使うともっと色んな事ができる、そんな感じでしょうか。）今回は簡単なサンプルアプリケーションを作りながら、Griffon の概要を紹介したいと思います。

インストール

Griffon のダウンロードサイトから最新版のアーカイブをダウンロードして展開します。現時点での最新版は 0.9.2-beta-1 です。Unix 系 OS の場合、展開したディレクトリ以下に実行権限を付与しておきましょう。 展開したディレクトリを環境変数 GRIFFON_HOME に設定し、環境変数 PATH に $GRIFFON_HOME/bin を通してください。 JAVA_HOME の設定も必要なので、設定していない場合は環境変数 JAVA_HOME も設定してください。

インストールが成功したかどうか以下のコマンドで確認してみましょう。

```
$ griffon help
Welcome to Griffon 0.9.2-beta-1 - http://griffon.
codehaus.org/
Licensed under Apache Standard License 2.0
Griffon home is set to: /opt/griffon
・・・
```

Griffon のヘルプメッセージが出力されたらインストール成功です。

アプリケーションの作成

それでは早速、Griffon アプリケーションを作成してみましょう。次のコマンドを実行します。(Grails と同じですね。)

```
$ griffon create-app myapp
```

create-app コマンドを実行すると以下のようなディレクトリ構成が作成されます。Grails を知っている人は各ディレクトリの役割が想像できると思います。

```
myapp
 ├ application.properties   アプリケーションの設定情報
 ├ griffonw                 Command Wrapper(※)スクリプト
 ├ griffonw.bat             Command Wrapper(※)スクリプト(Windows用)
 ├ griffon-app/             Griffonアプリケーションディレクトリ
 |    ├ conf/               設定DSL
 |    ├ controllers/        コントローラ
 |    ├ i18n/               国際化
 |    ├ lifecycle/          ライフサイクル
 |    ├ models/             モデル
 |    ├ resources/          リソースファイル
 |    └ views/              ビュー
 ├ lib/                     依存ライブラリを格納するディレクトリ
 ├ scripts/                 スクリプト
 ├ src/                     その他のソース
 ├ test/                    テストコード
 └ wrapper/                 Command Wrapper(※)のライブラリ
```

※ Command Wrapper

Griffon のバージョン 0.9 から Command Wrapper という機能が追加されました。この機能は Gradle というビルドツールにインスパイアされて追加された機能です。 Unix 系の OS であれば griffonw、Windows であれば griffonw.bat のスクリプトを griffon コマンドの代わりに使用する事ができます。 例えば Griffon アプリをローカルで起動する場合、griffon コマンドでは以下のようになります。

```
$ griffon run-app
```

Command Wrapper を使用する場合は、Unix 系の場合以下のコマンドで griffon コマンドと同じように起動できます。

```
$ ./griffonw run-app
```

Command Wrapper は初回起動時に Griffon のアーカイブをダウンロードし、ホームディレクトリ以下の .griffon/wrapper/dists ディレクトリにインストールされます。Command Wrapper は CI サーバ上でビルドする場合などに便利な機能です。例えば Hudson で Griffon アプリをビルドする場合、Hudson を動かしているサーバに Griffon をインストールしておく必要がありますが Command Wrapper を利用すれば Griffon のインストールが不要になります。

各ディレクトリにはいくつかソースコードが生成されていま

す。この状態でもアプリケーションを起動する事ができるので、実行してみましょう。アプリケーションの実行には「run-app」コマンドを実行します。

```
$ griffon run-app
```

実行すると図のようなウィンドウが表示されます。

アプリケーションのタイトルと「Content Goes Here」と書かれたラベルが表示されるシンプルなアプリケーションです。

View

View のソースコードにあたる、griffon-app/views/myapp/MyappView.groovy を見てみましょう。

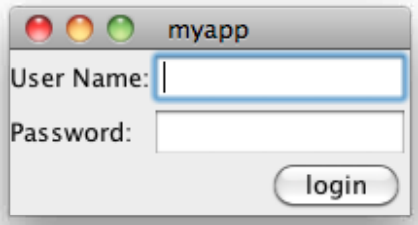
```
package myapp

application(title: 'myapp',
  //size: [320,480],
  pack: true,
  //location: [50,50],
  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]) {
    // add content here
    label('Content Goes Here') // deleteme
}
```

Griffon アプリケーションの View は SwingBuilder を利用して作成します。 MyappView.groovy の application を frame に読み替えると、Groovy の SwingBuilder を使った Swing アプリケーションと同じようなコードになります。 label メソッドは javax.swing.JLabel のインスタンスを生成するためのメソッドです。SwingBuilder の各メソッドは Groovy のユーザガイドで確認できます。

http://groovy.codehaus.org/Alphabetical+Widgets+List

今回は簡単なログインフォームを作成したいと思います。



Swing アプリケーションを作成するとき、コンポーネントのレイアウトに悩まされる事があるかと思います。デフォルトの BorderLayout だけでは図のような配置はできません。他のレイアウトマネージャを利用する必要がありますが、レイアウトマネージャの使い方に慣れていないとコンポーネントの配置にハマってしまう事があります。そんな方にお勧めするのが、SwingBuilder に追加されている TableLayout です。TableLayout は HTML の TABLE タグを記述するように簡単にコンポーネントを配置できます。TableLayout を利用した MyappView.groovy は以下のようになります。

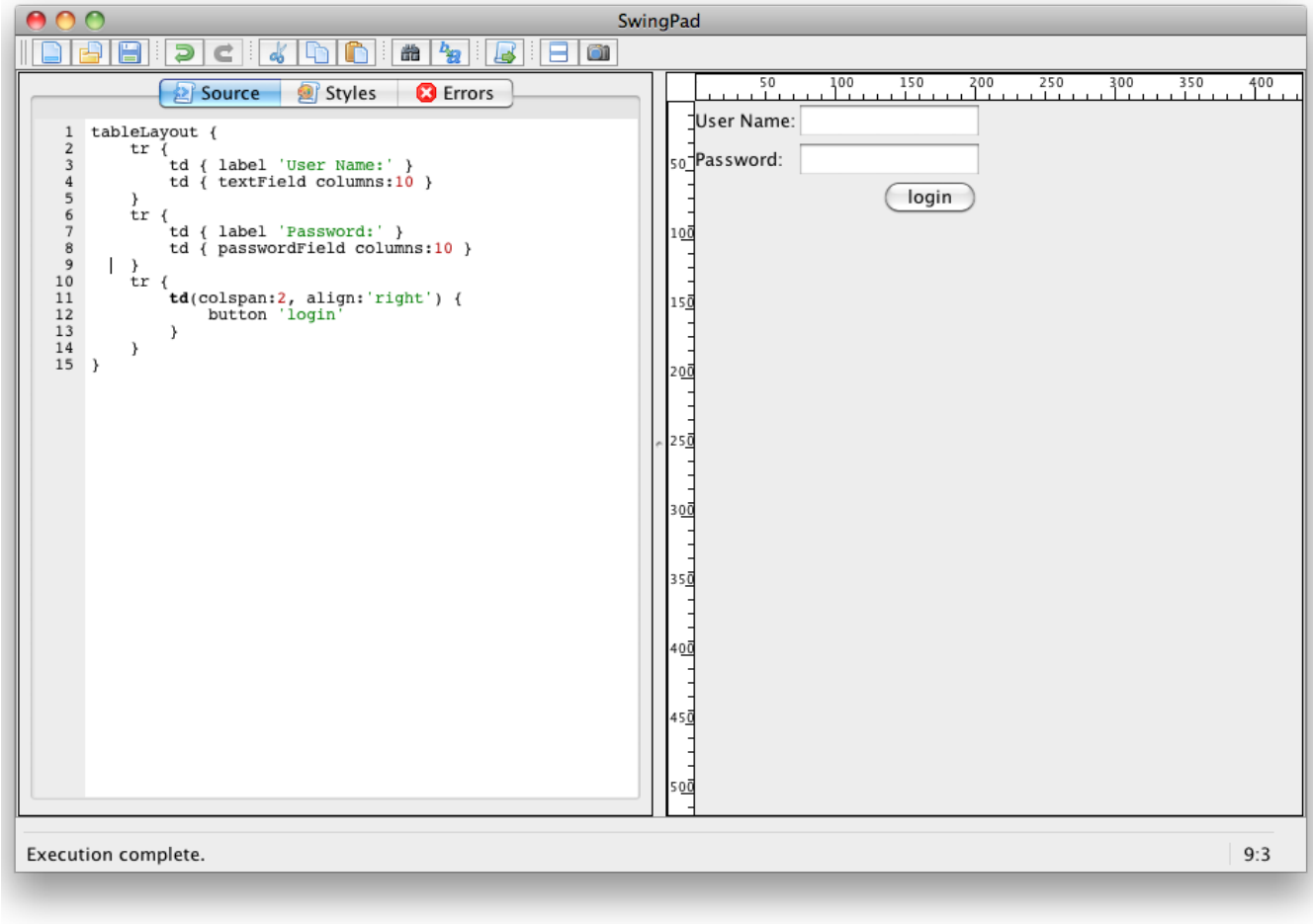
```
package myapp

application(title: 'myapp',
  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]) {
    tableLayout {
        tr {
            td { label 'User Name:' }
            td { textField columns:10 }
        }
        tr {
            td { label 'Password:' }
            td { passwordField columns:10 }
        }
        tr {
            td(colspan:2, align:'right') {
                button 'login'
            }
        }
    }
}
```

tr や td など、まるで HTML を書いているようですね。この状態でアプリケーションを実行すると図のようなログインフォームが表示されます。

run-app コマンドは実行に少し時間がかかるので、View を変更するたびに run-app するのは面倒です。 Griffon には View を作成するときに便利な SwingPad という WYSIWYG エディタがあります。 SwingPad は $GRIFFON_HOME/samples/SwingPad ディ

レクトリに格納されています。 SwingPad 自体も Griffon で作成されているので $GRIFFON_HOME/samples/SwingPad ディレクトリ上で run-app コマンドを実行すると SwingPad が起動できます。



SwingPad の左側のエディタに SwingBuilder を使ったコードを書き、メニューバーから「Script」-「Run」を選択するか、Ctrl キー (Mac の場合は Command キー) と「R」で右側に実行結果の View が表示されます。また、「Script」-「Samples」以下にサンプルコードが用意されているので書き方の参考になると思います。

Model

先ほど作成したログインフォームの View で入力した値を保持する Model を作成してみましょう。既に griffon-app/models/myapp/MyappModel.groovy ファイルがあり、MyappModel クラスが定義されています。プロパティを追加して、MyappModel クラスでユーザ名、パスワードを保持できるようにしましょう。

```
package myapp

import groovy.beans.Bindable

class MyappModel {
   @Bindable String username
   @Bindable String password
}
```

@Bindable アノテーション以外は普通のバリューオブジェクトですね。@Bindable については後ほど説明しますのでここではおまじないと思っておいてください。と、言う訳で Model が完成しました。

Controller

最後にコントローラを作成します。View で login ボタンを押したときに何か処理をしましょう。TwitterAPI を使用して Twitter にログインしたいところですが、~~解説が面倒なので~~、ページ数の都合上、ユーザ名とパスワードを標準出力に表示するだけにしたいと思います。griffon-app/controllers/myapp/MyappController.groovy に以下のコードを追加しましょう。

```
def login = { evt ->
     println "Username: [${model.username}], Password: [${model.password}]"
}
```

このままではログインボタンを押したときに何も実行されないので、View のコードを少し変更します。

```
package myapp

application(title: 'myapp',
 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]) {
   tableLayout {
       tr {
           td { label 'User Name:' }
           // 変更
           td { textField columns:10,
             text:bind('username', target:model) }
       }
       tr {
           td { label 'Password:' }
           // 変更
           td { passwordField columns:10,
             text:bind('password', target:model) }
       }
       tr {
           td(colspan:2, align:'right') {
               // 変更
               button 'login',
                actionPerformed:controller.login
           }
       }
   }
}
```

変更箇所で使用している変数 model と controller はそれぞれ、MyappModel クラスと MyappController クラスのインスタンスです。 この状態でアプリケーションを実行し、ユーザ名に「jggug」、パスワードに「griffon」と入力して login ボタンを押すと標準出力に以下の内容が表示されます。

```
Username: [jggug], Password: [griffon]
```

変更箇所の解説をしておきましょう。ユーザ名を入力するテキストフィールドは以下のようになりました。

```
textField columns:10, text:bind('username', target:model)
```

プロパティ text にはテキストフィールドに表示する文字列を指定します。「text:'jggug'」とすればユーザ名に 'jggug' という文字列が入力された状態で表示されます。サンプルでは文字列を指定する代わりに SwingBuilder の bind メソッドを使用しています。bind メソッドを使用する事でテキストフィールドに入力された値が、Model の 'username' というプロパティにバインドされるようになります。Model の方ではプロパティにバインドされるようにするためにおまじないが必要です。このおまじないが Model の作成時に指定した @Bindable アノテーションになります。@Bindable アノテーションと bind メソッドを利用する事で、Java の PropertyChangeListener を使用したバインディングが可能になります。このバインディングは Griffon 固有の仕組みではなく Groovy だけでも利用できます。@Bindable アノテーションを定義しておけば AST 変換でバインディングに必要な処理が実装されます。

login ボタンには actionPerformed プロパティにコントローラの login クロージャを設定しました。SwingBuilder の button メソッドは javax.swing.JButton のインスタンスを生成するメソッドで、JButton には actionPerformed というプロパティは存在しません。しかし、Groovy では JButton にプロパティが追加され、クロージャーを設定すると JButton に ActionListener が追加されます。他の Swing コンポーネントにも Listener 系のプロパティが追加されています。actionPerformed プロパティを設定する以外にも action プロパティを設定する方法もありますが、それはまた別の機会に紹介したいと思います。

まとめ

今回は Griffon を使って簡単な MVC を作成し、簡単なデスクトップアプリケーションの作り方を紹介しました。Griffon を利用する事で MVC をきれいに分ける事ができ、また Groovy のパワーをかりて簡潔にアプリケーションを作成できることを理解して頂けると幸いです。残念ながら今回は Griffon の入り口しか紹介できませんでしたが、Griffon には Grails と同じようにプラグインの仕組みもあります。また別の機会で Griffon の様々な魅力を紹介できればと思います。では、またお会いしましょう。

リンク

Griffon 公式サイト http://griffon.codehaus.org/
Griffon ドキュメント
http://dist.codehaus.org/griffon/guide/index.html

Grails Plugin 探訪

第 1 回

~ Application Info ~

URL: http://grails.org/plugin/app-info
Grails のバージョン : 1.2 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』

今回紹介する Grails プラグインは、Application Info です。 このプラグインを組み込むことで、実行中のアプリケーションの情報を Web ブラウザ上で確認することができます。

プラグインのインストール

プラグインのインストールは、次のとおりです。

```
$ grails install-plugin app-info
```

設定など

アプリケーションの情報を取得できるように、Config.groovy に次の設定を記述します。

```
grails.plugins.dynamicController.mixins = [
   'com.burtbeckwith.grails.plugins.appinfo.IndexControllerMixin':       'com.burtbeckwith.appinfo_test.AdminManageController',
   'com.burtbeckwith.grails.plugins.appinfo.HibernateControllerMixin':   'com.burtbeckwith.appinfo_test.AdminManageController',
   'com.burtbeckwith.grails.plugins.appinfo.Log4jControllerMixin' :      'com.burtbeckwith.appinfo_test.AdminManageController',
   'com.burtbeckwith.grails.plugins.appinfo.SpringControllerMixin' :     'com.burtbeckwith.appinfo_test.AdminManageController',
   'com.burtbeckwith.grails.plugins.appinfo.MemoryControllerMixin' :     'com.burtbeckwith.appinfo_test.AdminManageController',
   'com.burtbeckwith.grails.plugins.appinfo.PropertiesControllerMixin' : 'com.burtbeckwith.appinfo_test.AdminManageController',
   'com.burtbeckwith.grails.plugins.appinfo.ScopesControllerMixin' :     'com.burtbeckwith.appinfo_test.AdminManageController'
]
```

また、情報をグラフィカルに表示するために、GraphViz を使用します。 Mac OSX や Linux 系の OS などには標準でインストールされていると思いますが、Windows にはインストールされていませんので、GraphViz がイントールされていない場合は、GraphViz のインストールと設定を行ないます。

http://www.graphviz.org/ から GraphViz のバイナリを取得し、インストールします。 インストールされる GraphViz のコマンドのひとつに dot(Windows の場合は dot.exe) がありますので、そのパスを Config.groovy に記述します。

```
grails.plugins.appinfo.dotPath = 'c:/path/to/dot.exe'
```

使い方

アプリケーションを起動後、次の URL にアクセスします : /appname/adminManage

メニューを辿っていくことで、実行中のアプリケーションのさまざまな情報を見ることができます。 どのような情報を見ることができるかは、プラグインのページ http://grails.org/plugin/app-info、または作者の紹介ページ http://burtbeckwith.com/blog/?p=344 を参照してください。

最後に

作者の紹介ページには、プラグインをインストール済みのサンプルアプリケーションがありますので、それを動かしてみて、どのような情報が参照できるか確認してみてはいかがでしょうか。

リリース情報 2010.11

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, 1.3.5
■更新情報
- 1.2.5 では、いくつかのバグフィックスと、Quartz のコンフリクトが解消されています。
- 1.3.5 では、たくさんのバグフィックスと、いくつか新機能の追加と改良が行われています。
- 1.2.5 リリースノート : http://www.grails.org/1.2.5+Release+Notes
- 1.3.5 リリースノート : http://www.grails.org/1.3.5+Release+Notes

Groovy

Groovy は、JavaVM 上で動作する動的言語です。
URL: http://groovy.codehaus.org/
バージョン : 1.7.5, 1.8-beta-2

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 : 0.9.1a, 0.9.2-beta-1
■更新情報
- 0.9.1 は、0.9 のメンテナンスリリースで、いくつか新機能が追加されています。0.9.1a では、Windows 対応の修正が入っているようです。
- 0.9.2-beta-1 がリリースされましたが、0.9.1a からの変更内容は不明です。
- 0.9.1 リリースノート : http://griffon.codehaus.org/Griffon+0.9.1

Gant

Gant は、XML の代わりに Groovy で Ant タスクを記述し実行するビルド管理ツールです。
URL: http://gant.codehaus.org/
バージョン : 1.9.3

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-rc-2
■更新情報
- 0.9.2-rc-2 がリリースされましたが、0.9.2-rc-1 からの変更内容は不明です。
- 0.9 リリースノート : http://docs.codehaus.org/display/GRADLE/Gradle+0.9+Release+Notes

Gaelyk

Gaelyk は、Groovy で記述する Google App Engine for Java 用のライトウェイトなフレームワークです。
URL: http://gaelyk.appspot.com/
バージョン : 0.5.6
■更新情報
- GAE SDK 1.3.8 に対応し、バグフィックスや、メソッドの追加・チェック処理の追加が行われています。
- 0.5.6 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 : 1.3.8
■更新情報
- 管理コンソールからタスクキューのタスクをすぐに実行できるようになったり、複数の Google アカウントでのログインに対応したりできるようになりました。
- 1.3.8 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 : 0.10 GA

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 : 0.2.26
■更新情報
- 0.2.25 からの更新内容は不明ですが、バグフィックス対応と思われます。

Spock

Spock は、Java や Groovy のため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 : 0.4

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 : 0.4

G* Magazine Prerelease 2010.11
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