今こそッはじめよう

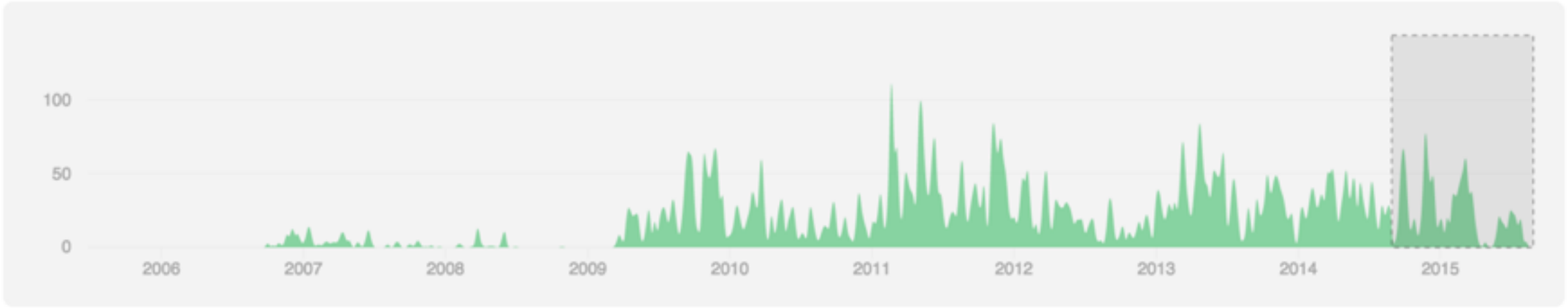
Grailsブートキャンプ!!!

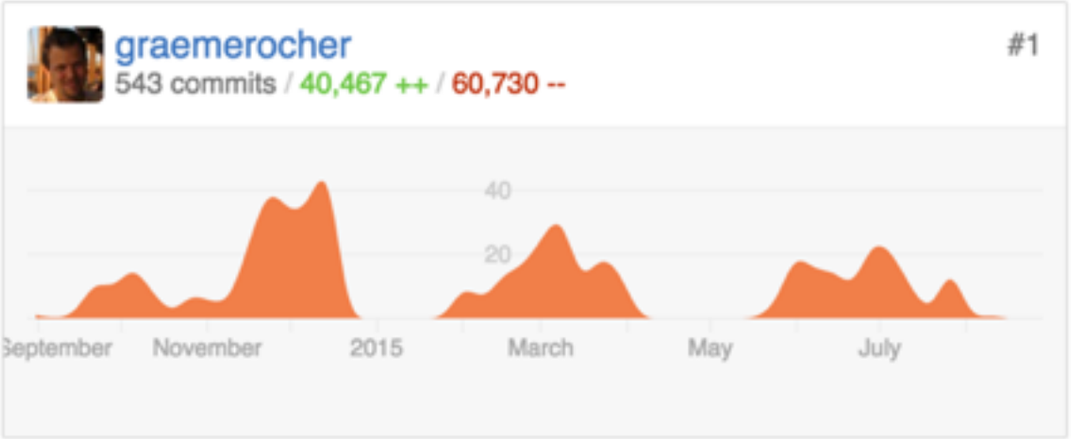
@yamkazu

#jggug

山本 和樹

- @yamkazu
- JGGUG


100
50
0
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015


graemerocher
543 commits / 40,467 ++ / 60,730 --
#1
40
20
September
November
2015
March
May
July


lhotari
363 commits / 36,118 ++ / 42,866 --
#2
40
20
September
November
2015
March
May
July


jeffbrown
238 commits / 26,672 ++ / 32,687 --
#3
40
20
September
November
2015
March
May
July


bobbywarner
11 commits / 64 ++ / 15 --
#4
40
20
September
November
2015
March
May
July


yamkazu
9 commits / 128 ++ / 17 --
#5
40
20
September
November
2015
March
May
July


candrews
8 commits / 246 ++ / 84 --
#6
40
20
September
November
2015
March
May
July

本日の内容

- Grails概要
- Grails 3概要
- Grailsハンズオン
  - Hallo World
  - TODOアプリを作ってみよう

今日やらないこと

- Groovy自体の説明
  - サンプルコードで登場するGroovyのコードについては可能なかぎり捕捉をいれていきます
- Grailsの網羅的な説明

今日は手を動かして雰囲気を掴んでもらうことを優先します！

今日のハンズオンに必要な環境

- Java
- Grails
- IDE or テキストエディタ

https://github.com/yamkazu/jggug-grails-bootcamp
を参考に準備をお願いします！

Grailsとは

- Graeme Rocher氏が開発
- フルスタックのWebフレームワーク
- Groovyベース
- Ruby on Rails、Djangoといったフレームワークに影響を受けている
    - DRY（Don't Repeat Yourself）= 同じ記述を繰り返さない
    - CoC（Convention over Configuration）= 設定よりも規約
    - スキャフォルディング
- Java EE上で動作

Grailsの歴史

1.0
2008/02

2008/11
Spring Source配下へ

2009/08
VMwareが
Spring Sourceを買収

2.0
2011/12

2013/04
Spring Sourceが
Pivotal配下へ

3.0
2015/03

2015/03
Pivotal卒業

OCIが新しいホームへ

http://www.ociweb.com/resources/news/2015/04/15/april-2015-grails-web-framework-finds-home-oci

Grails 3の概要

- Spring Bootをベースに再構築
- Gradleでビルドシステムが一新
- アプリケーションプロファイルの追加
- コントローラ、ドメインクラス等などの使い方はほとんど変更なし
- etc

とりあえずHello World!

プロジェクトの作成

コマンドラインから以下を実行する。

```
$ grails create-app sample
$ cd sample
```

作成したらIDEAで読み込む。

https://github.com/yamkazu/jggug-grails-bootcamp/blob/master/README.md
で実施済みの人は不要！

コントローラの作成

Grailsのインタラクティブモードから以下を実行しコントローラを生成する。

```
$ grails
grails> create-controller hello
```

アクションの実装

IDEAからgrails-app/controllers/sample/HelloController.groovyを開いて

以下の**index**アクションを追加する。

```groovy
package sample

class HelloController {

    def index() {
        render 'Hello Grails!'
    }
}
```

https://gist.github.com/yamkazu/08e5daed0092a24e205e#file-hellocontroller-groovy

アプリケーションの起動

Grailsのインタラクティブモードから以下を実行しアプリケーションを起動する。

```
grails> run-app
```

ブラウザで **http://localhost:8080** にアクセスする。

sample.HelloControllerのリンクをクリックする。

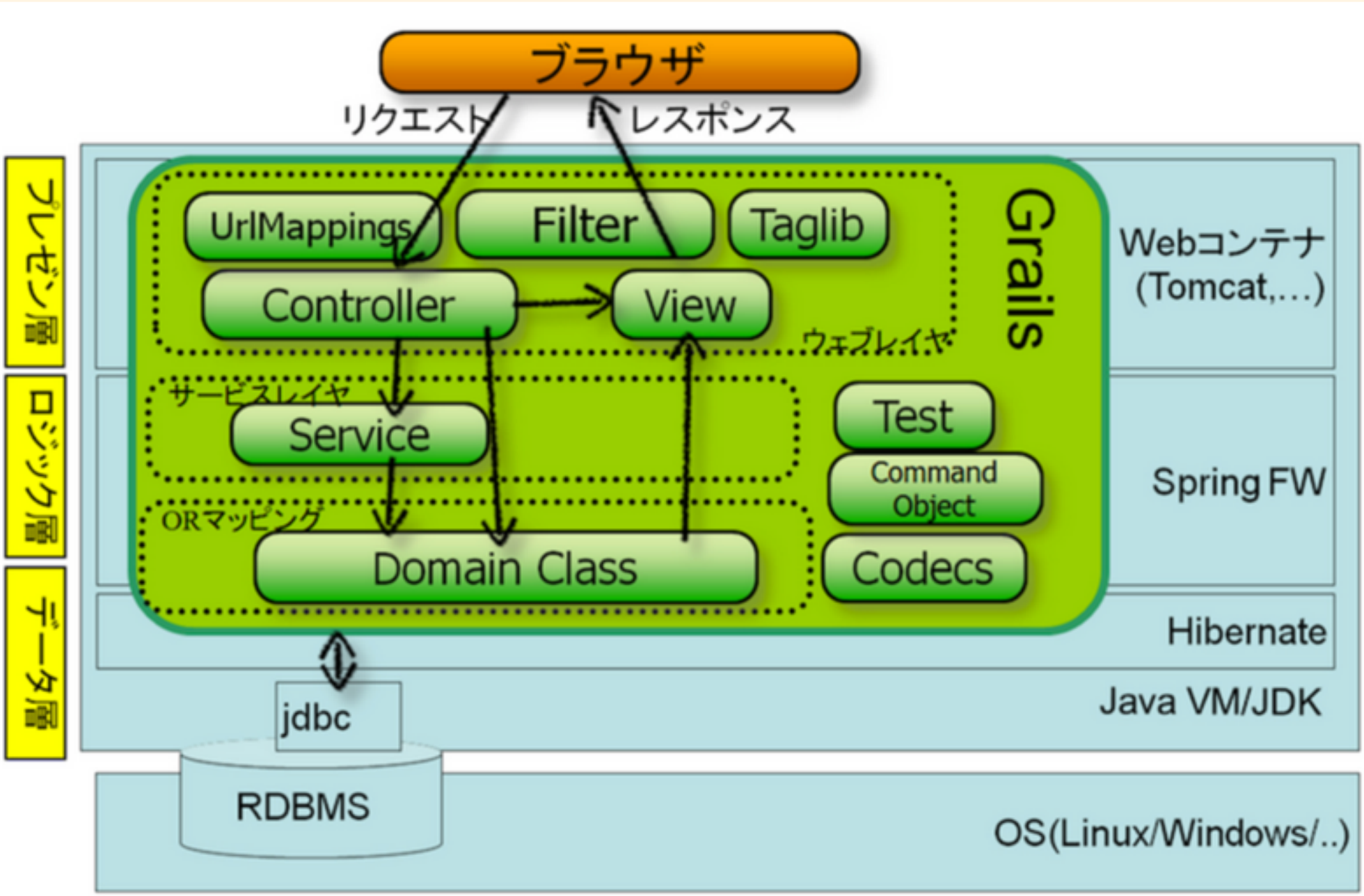
'Hello Grails!'と表示されることを確認する。

アプリケーションの停止

Grailsのインタラクティブモードから以下を実行する。

```
grails> stop-app
```

Grailsを構成する要素



アーティファクト

- Grailsによって特別扱いされるオブジェクトのこと
- 開発者が記述したままのコード以外に、Grailsによって積極的に機能が追加される
- 普通のクラスとは格納ディレクトリが分かれている
    - grails-app配下: アーティファクト
    - src配下: 普通のクラス
- 主なアーティファクトの種類 ドメインクラス
    - ドメインクラス
    - コントローラ
    - サービス
    - ビュー
    - タグライブラリ

Grailsのディレクトリ構成の作成

```
yourAppProject                  grails create-appでこの配下の構造が生成される
 ├── build.gradle               Gradleのビルド設定ファイル
 ├── gradle                     Gradle WrapperのJarファイルが格納される
 ├── gradle.properties          Gradleのプロパティファイル
 ├── gradlew                    Gradle Wrapperのスクリプトファイル
 ├── gradlew.bat                Gradle Wrapperのスクリプトファイル
 ├── grails-app                 Grailsとして特別扱いをするクラス群を格納する
 │    ├── assets                静的リソースを格納する
 │    │    ├── images           画像ファイル
 │    │    ├── javascripts      JavaScriptファイル
 │    │    └── stylesheets      CSSファイル
 │    ├── conf                  設定系
 │    ├── controllers           コントローラ
 │    ├── domain                ドメインクラス
 │    ├── i18n                  メッセージバンドル
 │    ├── init                  起動時に必要なクラス
 │    ├── services              サービス
 │    ├── taglib                タグライブラリ
 │    ├── utils                 コーデックと呼ばれる特殊クラスを配置する(滅多に使用することはない)
 │    └── views                 ビュー(GSPファイル)
 └── src
      ├── integration-test      Grailsアプリケーションを内部で起動して実行するテスト
      │    └── groovy
      ├── main                  Grailsの特別扱いを必要としない通常のクラスを格納する
      │    ├── groovy
      │    └── webapp
      └── test                  純粋なGroovy/Javaプログラムとしてのテストを格納
           └── groovy
```

スキャフォルドを
体験してみよう

ドメインクラスの作成

Grailsのインタラクティブモードから以下を実行しドメインクラスを生成する。

```
grails> create-domain-class book
```

ドメインクラスを定義する

IDEAでgrails-app/domain/sample/Book.groovyを開いて以下を記述する。

```
package sample

class Book {

    String title
    Integer price

    static constraints = {
    }
}
```

https://gist.github.com/yamkazu/08e5daed0092a24e205e#file-book-groovy

スキャフォルドの実行

Grailsのインタラクティブモードから以下を実行し画面を生成する。

```
grails> generate-all sample.Book
```

いくつかのファイルが生成される。

アプリケーションの起動

Grailsのインタラクティブモードから以下を実行しアプリケーションを起動する。

```
grails> run-app
```

ブラウザで **http://localhost:8080** にアクセスする。

sample.BookControllerのリンクをクリックする。

インタラクティブモードの終了

Grailsのインタラクティブモードから以下を実行しインタラクティブモードを終了する。

```
grails> exit
```

アプリケーションが起動中の場合は自動的に停止する。

TODOアプリを作ってみよう

今回作るアプリのイメージ

プロジェクトを作成

コマンドラインから以下を実行する。

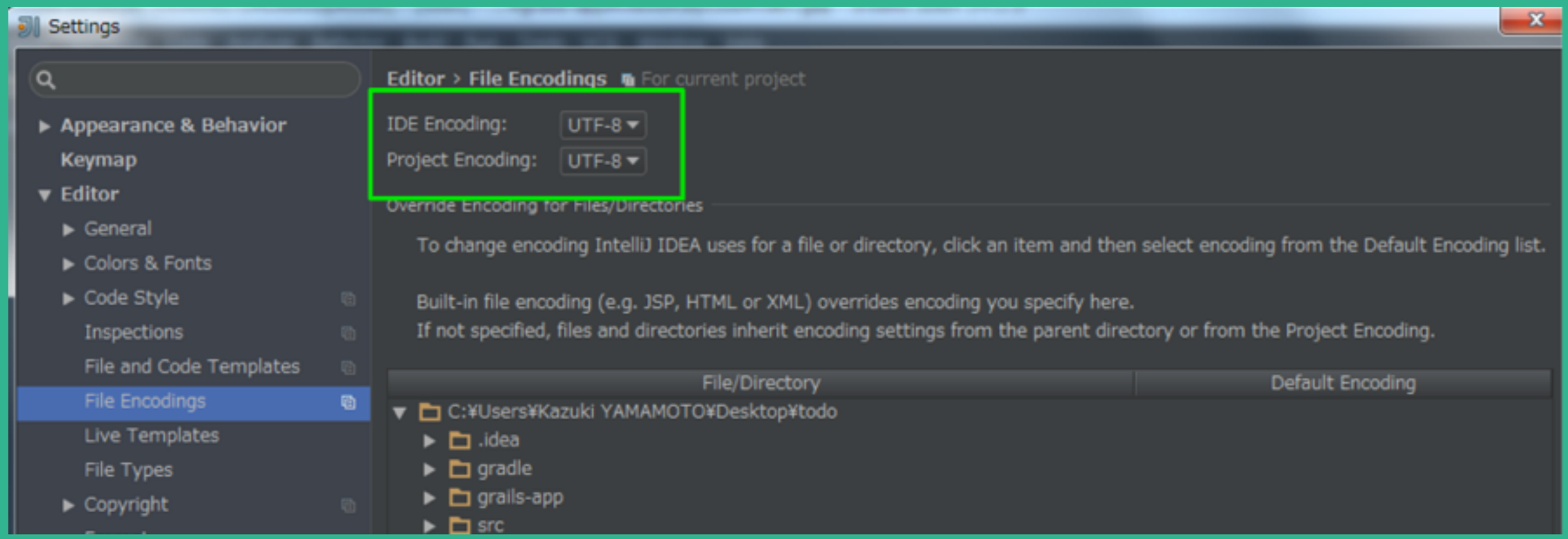
```
$ grails create-app todo
$ cd todo
```

作成したらIDEAで読み込む。

IDEAの文字コードの設定

Windowsの方は以下設定から文字コードを**UTF-8**に設定してください。

[File]-[Settings...]
[Editor]-[File Encodings]



まずはドメインクラスを作る

ドメインクラス

- いわゆるモデルを定義するクラス
    - ドメインクラス≒ Hibernate用語「エンティティ」
- GORM(Groovy Object Relational Mapping、ゴー ム、ゴルム) をドメインクラスを介して利用する
- 入力値の制約を定義できる
- ドメインクラスの定義がデータベースのマッピング定義になる
    - クラス名 -> テーブル名
    - プロパティ名 -> カラム名
    - 制約 -> カラムの制約

ドメインクラスの例

```groovy
class Person {
    // プロパティ
    String name // 氏名
    Integer age // 年齢

    // 制約
    static constraints = {
        name size: 10..30 // 10〜30文字でなければならない
        age min: 18       // 18歳以上でなければならない
    }
}
```

ドメインクラスを生成する

Grailsのインタラクティブモードから以下を実行する。

```
$ grails
grails> create-domain-class todo
```

以下のファイルが生成される。

grails-app/domain/todo/Todo.groovy
src/test/groovy/todo/TodoSpec.groovy

ドメインクラスを定義する

IDEAでgrails-app/domain/todo/Todo.groovyを開いて以下を記述する。

```
package todo

class Todo {

    String content

    static constraints = {
        content blank: false, maxSize: 20
    }
}
```

https://gist.github.com/yamkazu/08e5daed0092a24e205e#file-todo-groovy

アプリケーションを起動してデータベースを確認する

Grailsのインタラクティブモードから以下を実行する。

```
grails> run-app
```

ブラウザで **http://localhost:8080/dbconsole** にアクセスする。

以下の設定で「**Connect**」をクリック。

JDBC URL: jdbc:h2:mem:devDb
User Name: sa

ドメインクラスを使ってアプリケーション起動時にデータベースにデータを保存する

ドメインラクスを使ったCURD操作の基本

- ドメインクラスに自動的に追加されるメソッドを使う
- 保存、更新
    - domainInstance.save()
- 削除
    - domainInstance.delete()
- 一覧取得
    - DomainClass.list()
- 1件取得
    - DomainClass.get(id)

CRUD操作の例

```
// 新規作成
def person = new Person(name: "山田", age: 20)
person.save()

def person = new Person(name: "山田", age: 20).save()

// 参照
def person = Person.get(1) // IDを指定して取得
def people = Person.list() // 一覧を取得
def people = Person.list(offset: 10, max: 20) // 開始位置、件数を指定して一覧を取得
def people = Person.list(sort: "name", order: "asc") // ソート条件を指定して取得
int personCount = Person.count() // 件数を取得

// 更新
def person = Person.get(1)
person.name = "鈴木"
person.save()

// 削除
def person = Person.get(1)
person.delete()
```

ブートストラップ

- Grailsアプリケーションの起動時と終了時に、簡単に任意の処理を実行できる仕組み
- grails-app/init/BootStrap.groovyに実装する
  - init: 初期化時の処理
  - destroy: 終了時の処理
- 環境ごとの設定を記述できる

環境

- 実行環境に応じて設定値、内部の処理を切り替えるための仕組み
    - 設定ファイルやBootstrap内などでデフォルトで利用できる
- デフォルトで用意されている環境は以下の3つ
    - development
        - run-appコマンドで起動したときの環境
        - さくさくと開発できるように、自動リロードやキャッシュの無効化される
    - test
        - test-appなどでテストを実行したときの環境
    - production
        - warコマンドなどで生成されたファイルを起動したときの環境

設定ファイルでの使用例

```
environments:
    development:
        dataSource:
            dbCreate: create-drop
            url: jdbc:h2:mem:devDb;MVCC=TRUE;LOCK_TIMEOUT=10000;DB_CLOSE_ON_EXIT=FALSE
    test:
        dataSource:
            dbCreate: update
            url: jdbc:h2:mem:testDb;MVCC=TRUE;LOCK_TIMEOUT=10000;DB_CLOSE_ON_EXIT=FALSE
    production:
        dataSource:
            dbCreate: update
            url: jdbc:h2:./prodDb;MVCC=TRUE;LOCK_TIMEOUT=10000;DB_CLOSE_ON_EXIT=FALSE
```

Bootstrap.groovyでの使用例

```
class BootStrap {

    def init = { servletContext ->
        environments {
            development {
                // 開発時の初期化処理
            }
            test {
                // テスト時の初期化処理
            }
            production {
                // 本番環境での初期化処理
            }
        }
    }
    ...
}
```

起動時にテストデータを投入する

grails-app/init/BootStrap.groovyに以下を記述する。

```
import todo.Todo

class BootStrap {

    def init = { servletContext ->
        environments {
            development {
                new Todo(content: "山田さんに電話").save()
                new Todo(content: "ゴミ袋を買う").save()
                new Todo(content: "ネコに餌をやる").save()
            }
        }
    }
    …
}
```

https://gist.github.com/yamkazu/08e5daed0092a24e205e#file-bootstrap-groovy

再起動して動作を確認

Grailsのインタラクティブモードから以下を実行する。

```
grails> stop-app
grails> run-app
```

ブラウザで**dbconsole**を開く。

Todoのデータを確認する。

画面にTODOリスト
を表示する

コントローラの概要

- コントローラとはWebブラウザからのリクエストを受けて、レスポンスを返す一連の処理を定義するクラス

- メソッドごとに個別の処理を定義することができ、このメソッドをアクションと呼ぶ

コントローラとアクションの定義例

```
class BookController {

    def index() { ... }

    def list() { ... }

    def show() { ... }
}
```

アクションの中からレスポンスを返す

- renderメソッドを使う
    - 様々なタイプのレスポンスを返すことができる
        - テキスト
        - 任意のビューを指定して画面を表示したり
        - JSONを返却
        - etc
- respondメソッドを使う
    - AcceptやURLの拡張子などを使ってデータを適当なフォーマットでレンダリングしてくれる
    - http://grails.github.io/grails-doc/latest/ref/Controllers/respond.html
- アクション内でレスポンスを指定しなかった場合はコントローラ名、アクション名から自動的にビューが選択される
    - 例: BookControllerのshowアクションの場合、grails-app/views/book/show.gspがビューが使用される

renderメソッドの使用例

```groovy
class BookController {
    def action1() {
        // 何も指定しない
        // grails-app/views/book/action1.gspが使われる
    }

    def action2() {
        // grails-app/views/book/display.gspが使われる
        render(view: 'display')
    }

    def action3() {
        // grails-app/views/shared/display.gspが使われる
        render(view: '/shared/display')
    }

    def action4() {
        // 文字列を表示
        render 'Hello World!'
    }

    def action5() {
        // Bookの一覧をJSONで表示
        render Book.list() as JSON // grails.converters.JSON
    }
}
```

アクションからビューに値を渡す

- いくつかやり方があるが基本的な方法は以下の２つ
  - アクションからMapのインスタンスを返す
  - renderの引数でmodelを指定する

modelの指定例

```
// アクションからMapのインスタンスを返す
def show() {
    [message: 'hello']
}

// renderの引数でmodelを指定する
def show() {
    render(view:'display', model: [message: 'hello'])
}
```

ビュー(GSP)

- GrailsではビューのGSP(Groovy Server Pages)を使う
  - 簡単に言うとJSP(JavaServer Page)のGroovy版
  - ${expr}といった形でGSPの中でGroovyの式を書ける
  - デフォルトで用意されたタグライブラリが使える
    - タグライブラリの一覧は http://grails.github.io/grails-doc/latest/ の右に表示されているTagsを参照

GSPの例

```
<!DOCTYPE html>
<html>
<head>
  <meta name="layout" content="main"/>
  <title>Render Domain</title>
</head>
<body>
  <table>
    <tr>
      <td>Name</td>
      <td>Age</td>
    </tr>
    <g:each in="${list}" var="person">
      <tr>
        <td>${person.lastName}, ${person.firstName}</td>
        <td>${person.age}</td>
      </tr>
    </g:each>
  </table>
</body>
</html>
```

コントローラを生成する

Grailsのインタラクティブモードから以下を実行する。

```
grails> create-controller todo
```

以下のファイルが生成される。

grails-app/controllers/todo/TodoController.groovy
src/test/groovy/todo/TodoControllerSpec.groovy

コントローラを実装する

IDEAでgrails-app/controllers/todo/TodoController.groovyを開いて以下を記述する。

```groovy
package todo

class TodoController {

    def index() {
        [todos: Todo.list()]
    }
}
```

https://gist.github.com/yamkazu/08e5daed0092a24e205e#file-todocontroller1-groovy

ビューを実装する

IDEAでgrails-app/views/todoを右クリックして[New]-[File]からindex.gsp
というファイルを作成する。
以下の内容を記述する。

```
<!doctype html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>MYTODO</title>
    </head>
    <body>
        <h2>TODOリスト</h2>
        <ul>
            <g:each in="${todos}" var="todo">
                <li>${todo.content}</li>
            </g:each>
        </ul>
    </body>
</html>
```

https://gist.github.com/yamkazu/08e5daed0092a24e205e#file-index1-gsp

TODOを追加/削除
できるようにする

画面からパラメータを受け取る

- 基本的なやり方は以下
    - 暗黙の変数paramsの使用
    - リクエストパラメータをアクションの引数に指定する
    - コマンドオブジェクトをアクションの引数に指定する
        - ドメインクラスをコマンドオブジェクトとして使うこともできる
        - ドメインクラスを指定されている、かつリクエストパラメータの中にidのパラメータがあれば自動的にそのidに対応するデータを取得してくれる

paramsの例

```
def save() {
    new Person(name: params.name, age: params.int('age')).save()
    new Person(params).save()
}

def show() {
    [person: Person.get(params.id)]
}
```

リクエストパラメータをアクションの引数にする例

```
def save(String name, Integer age) {
    new Person(name, age).save()
}

def show(Long id) {
    [person: Person.get(id)]
}
```

ドメインクラスをアクションの引数にする例

```
def save(Person person) {
    person.save()
}

def show(Person person) {
    [person: person]
}
```

save/deleteアクションを実装する

grails-app/controllers/todo/TodoController.groovyに**save**アクションと**delete**アクションを追加する。

```groovy
package todo

class TodoController {
    ...
    def save(Todo todo) {
        todo.save(flush: true)
        redirect action: 'index'
    }

    def delete(Todo todo) {
        todo.delete(flush: true)
        redirect action: 'index'
    }
}
```

https://gist.github.com/yamkazu/08e5daed0092a24e205e#file-todocontroller2-groovy

追加/削除の画面を実装する

grails-app/views/todo/index.gspに以下を追加する。

```
...
    <body>
        <g:form action="save">
            <g:textField name="content" />
            <g:submitButton name="create" value="作成" />
        </g:form>
        <h2>TODOリスト</h2>
        <ul>
            <g:each in="${todos}" var="todo">
                <li>
                    ${todo.content}
                    <g:form action="delete" id="${todo.id}">
                        <g:submitButton name="delete" value="削除" />
                    </g:form>
                </li>
            </g:each>
        </ul>
...
```

https://gist.github.com/yamkazu/08e5daed0092a24e205e#file-index2-gsp

画面で入力チェックをできるよ
うにする

制約

- ドメインクラスのconstraintsブロックに定義する
    - nullable:falseだけはデフォルトで設定される
- 制約は以下の3つで主に利用される
    - バリデーション
        - 制約のメインの用途
        - Grailsが行う入力値のバリデーションに使われる
    - データベースのスキーマ
        - dbCreateがcreateなどの場合にHibernateによって自動生成されるデータベースにおいて、カラムの型やサイズ、NOT NULLなどの制約に使われる
    - スキャフォルド
        - 自動生成されるビューのフォーム要素の種類や属性値に使われる

制約の例

```
static constraints = {
    age nullable:false
    username blank: false
    type inList: ["Commercial", "Personal"]
    username unique:true
    username matches: /[a-zA-Z]/
    password validator: { value, self -> ... }
    children minSize:5, maxSize:25
    age min:0, max:120
    mailAddress email: true
    webSite url: true
    username notEqual:'root'
    age range:0..120
    price scale:2
    children size:5..25
    cardNumber creditCard: true
}
```

バリデーション

- 制約の定義を基に入力値チェックを実施する
- バリデーション関連のメソッドはドメインクラスにGrailsが自動的に追加する
- 明示的な呼び出し
    - domainInstance.validate()
- 暗黙的な呼び出し
    - domainInstance.save()の実行した場合
    - コントローラのアクションの引数にドメインクラスを指定した場合
- バリデーション実施後、エラーがある場合はdomainInstance.hasErrors()がtrueを返す

バリデーションの使用例

```
// 明示的なバリデーションの実行
def save() {
    def book = new Book(params)
    book.validate() // 明示的に実行
    if (book.hasErrors()) {
        ...
    }
    ...
}

// saveを実行した場合
def save() {
    def book = new Book(params).save()
    // 暗黙的にバリデーションを実行ずみ
    if (book.hasErrors()) {
        ...
    }
    ...
}

// コントローラのアクションの引数にドメインクラスを指定した場合
def save(Book book) {
    // 暗黙的にバリデーションを実行ずみ
    if (book.hasErrors()) {
        ...
    }
    ...
}
```

バリデーションエラーの場合は入力画面を表示する

grails-app/controllers/todo/TodoController.groovyに以下を追加する。

```
class TodoController {
    ...
    def save(Todo todo) {
        if (todo.hasErrors()) {
            render view: 'index', model: [todo: todo, todos: Todo.list()]
            return
        }

        todo.save(flush: true)
        redirect action: 'index'
    }
    ...
}
```

https://gist.github.com/yamkazu/08e5daed0092a24e205e#file-todocontroller3-groovy

入力値エラーを表示する

grails-app/views/todo/index.gspを以下のように変更する。

```
...
    <body>
        <g:hasErrors bean="${todo}">
            <g:renderErrors bean="${todo}" />
        </g:hasErrors>
        <g:form action="save">
            <g:textField name="content" value="${task?.content}" />
            <g:submitButton name="create" value="作成" />
        </g:form>
        <h2>TODOリスト</h2>
...
```

https://gist.github.com/yamkazu/08e5daed0092a24e205e#file-index3-gsp

メッセージをカスタマイズする

i18n (internationalization; 国際化)

- 様々な言語のメッセージを集約管理し、Webブラウザの言語設定やサーバ上の言語設定などを基に、適切な言語のメッセージを適用する仕組み
- grails-app/i18nディレクトリ配下にロケールごとのプロパティファイルで管理する
- タグリブのg:messageを使ってメッセージを参照できる
- バリデーションのデフォルトメッセージの定義にも使わている

メッセージファイルの例

```
my.sample.message.hello = こんにちは、i18n。
my.sample.message.withArgs = 引数1番目:{0}, 2番目:{1}, 3番目:{2}
```

メッセージの参照例

```
// GSPの内でタグライブラリを使う
<g:message code="my.sample.message.hello" />
<g:message code="my.sample.message.withArgs" args="${ ['Grails', 'Groovy', 'Advocate'] }" />

// GSPの評価式の中で使う
${message(code: 'my.sample.message.hello')}
${message(code: 'my.sample.message.hello', args=['Grails', 'Groovy', 'Advocate'])}
```

```
// コントローラで使う
def show() {
    println message(code: 'my.sample.message.hello')
    println message(code: 'my.sample.message.withArgs', args: ['Grails', 'Groovy', 'Advocate'])
    ...
}
```

バリデーションのメッセージを変更する

grails-app/i18n/messages_ja.propertiesで以下のようにメッセージを変更する。

```
default.invalid.max.size.message={1}の{0}は{3}文字以内で入力してください。
default.null.message={1}の{0}が入力されていません。
```

同じファイルに以下のメッセージを追加する。

```
todo.label=TODO
todo.content.label=内容
```

https://gist.github.com/yamkazu/08e5daed0092a24e205e#file-messages1-properties

画面のメッセージを定義する

grails-app/i18n/messages_ja.propertiesに以下のメッセージを追加する。

```
app.name=MYTODO
```

https://gist.github.com/yamkazu/08e5daed0092a24e205e#file-messages2-properties

メッセージを参照する

grails-app/views/todo/index.gspを以下のように変更する。

```
...
<head>
    <meta charset="UTF-8">
    <title><g:message code="app.name" /></title>
</head>
<body>
  ...
  <g:form action="save">
    <g:textField name="content" value="${task?.content}" />
    <g:submitButton name="create" value="${message(code: 'default.button.create.label')}" />
  </g:form>
  <h2><g:message code="default.list.label" args="${[message(code: 'todo.label')]}"/></h2>
  <ul>
    <g:each in="${todos}" var="todo">
    <li>
      ${todo.content}
      <g:form action="delete" id="${todo.id}">
        <g:submitButton name="delete" value="${message(code: 'default.button.delete.label')}" />
      </g:form>
...
```

https://gist.github.com/yamkazu/08e5daed0092a24e205e#file-index4-gsp

TODOの絞り込み機能を実装する

クエリ

- Grailsでは、データベースに検索クエリを発行するための様々な方法が提供されている
    - ダイナミックファインダ
    - クライテリア
    - whereクエリ
    - HQL (Hibernate Query Language)
    - 名前付きクエリ
    - ネイティブSQL

ダイナミックファインダ

- 命名規則に沿ったメソッド呼び出しをすると自動的にクエリを発行してくれる機能
    - 一番昔からあるクエリの機能
- 基本パターンは以下の3種
    - findBy*
        - 指定された条件に一致する最初の1件を返す
    - findAllBy*
        - 指定された条件に一致するすべてを返す
    - countBy*
        - 指定された条件に一致するレコード件数を返す
- メソッド名の「*」の部分に検索条件を指定する
    - 対応するメソッドがあらかじめ定義されている訳ではない
    - 指定されたメソッド名から動的に検索条件を判断してクエリを実行する

ダイナミックファインダの例

```
Book.findAllByTitleAndAuthor("The Hoge", "Mike Davis")
Book.findAllByReleaseDateBetween(firstDate, new Date())
Book.findAllByReleaseDateGreaterThanEquals(firstDate)
Book.findAllByTitleLike("%Hobbit%")
Book.findAllByTitleIlike("%Hobbit%") // ignore case
Book.findAllByTitleNotEqual("Harry Potter")
Book.findAllByReleaseDateIsNull()
Book.findAllByReleaseDateIsNotNull()
Book.findAllPaperbackByAuthor("Douglas Adams")
Book.findAllNotPaperbackByAuthor("Douglas Adams")
Book.findAllByAuthorInList(["Douglas Adams", "Hunter S. Thompson"])
```

クライテリア

- HibernateのCriteria APIをラップしたGroovyのDSLで、複雑な検索条件を構築できる
- 対象ドメインクラスに対するCriteriaオブジェクトを生成して、そこに条件を付与していく
  - 生成にはDomainClass.createCriteriaまたはDomainClass.withCriteriaメソッドを使う
- if文などのGroovyコードが普通に使えるため、特定の場合のみ有効な条件を指定することも簡単

クライテリアの例

```groovy
def c = Book.createCriteria()
def results = c.list {
    def now = new Date()
    between('releaseDate', now - 7, now)
    like('title', '%Groovy%')
}

def results = Book.withCriteria {
    def now = new Date()
    between('releaseDate', now - 7, now)

    // 管理者以外の場合は公開された本だけに限定する
    if (!person.isAdmin()) {
        eq('available', true)
    }

    like('title', '%Groovy%')
}
```

Whereクエリ

- GroovyのAST変換を活用したクエリの仕組み
  - コンパイル時に静的に型チェックが可能
- 条件式を書くようにクエリの条件を記述できる
- 一番新しい

Whereクエリの例

```
Person.where { name == "Bart" }.list()
Person.where { (name == "Bart") && (age == 35) }.list()
Person.where { (name == "Bart") || (age > 18) }.list()
Person.where { age in 18..65 }.list()
Person.where { name ==~ /%yamada%/ }.list()
```

キーワードが指定された場合はキーワードで検索する

grails-app/controllers/todo/TodoController.groovyに以下を追加する。

```
class TodoController {

    def index(String keyword) {
        if (keyword) {
            return [todos: Todo.where { content ==~ /%$keyword%/ }.list()]
        }

        [todos: Todo.list()]
    }
    ...
}
```

https://gist.github.com/yamkazu/08e5daed0092a24e205e#file-todocontroller4-groovy

キーワードを入力できるようにする

grails-app/views/todo/index.gspを以下のように変更する。

```
...
<h2><g:message code="default.list.label" args="${[message(code: 'todo.label')]}"/></h2>

<g:form action="index">
    <g:textField name="keyword" value="${params.keyword}"/>
    <g:submitButton name="filter" value="絞り込み"/>
</g:form>
<ul>
  <g:each in="${todos}" var="todo">
      <li>
          ${todo.content}
          <g:form action="delete" id="${todo.id}">
              <g:submitButton name="delete" value="${message(code:
'default.button.delete.label')}" />
          </g:form>
      </li>
  </g:each>
</ul>
...
```

https://gist.github.com/yamkazu/08e5daed0092a24e205e#file-index5-gsp

共通のレイアウトを定義する

SiteMesh

- GrailsではSiteMeshを使って、ビューのレイアウトを行っている
    - 使用例: ヘッダやフッタ、サイドバーなどを定義したレイアウトファイルを定義する
- レイアウトはgrails-app/views/layoutsディレクトリに配置する
- レイアウトファイルを使う側はhtmlのmetaタグを使ってlayoutsディレクトリのファイル名を指定する

レイアウトファイルの定義例

```
<!doctype html>
<html>
    <head>
        <title><g:layoutTitle default="Grails"/></title>
        <asset:stylesheet src="application.css"/>
        <asset:javascript src="application.js"/>
        <g:layoutHead/>
    </head>
    <body>
        <header>My App</header>
        <g:layoutBody/>
        <footer>Copyright 2015</footer>
    </body>
</html>
```

レイアウトファイルの指定例

```
<!doctype html>
<html>
<head>
    <meta name="layout" content="main" />
    <title>My Page</title>
</head>
<body>
    <p>Hello Grails!</p>
</body>
</html>
```

レイアウトファイルを定義する

grails-app/views/layouts/main.gspを以下のように変更する。

```
<!doctype html>
<html>
    <head>
        <meta charset="UTF-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <title><g:layoutTitle default="${message(code: 'app.name')}"/></title>
        <g:layoutHead/>
    </head>
    <body>
        <h1><g:message code="app.name" /></h1>
        <g:layoutBody/>
    </body>
</html>
```

https://gist.github.com/yamkazu/08e5daed0092a24e205e#file-main1-gsp

レイアウトファイルを指定する

grails-app/views/todo/index.gspを以下のように変更する。

```
<!doctype html>
<html>
    <head>
        <meta name="layout" content="main"/>
        <title><g:message code="app.name" /></title>
    </head>
...
```

https://gist.github.com/yamkazu/08e5daed0092a24e205e#file-index6-gsp

URLマッピングをカスタマイズする

URLマッピング

- URLマッピングをカスタマイズすることで、URLとコントローラ/アクション/ビューの関係を自由に設定できる
- grails-app/controllers/UrlMappings.groovyに定義する

URLマッピングの例

```
class UrlMappings {

    static mappings = {
        // デフォルトのマッピング設定
        "/$controller/$action?/$id?(.$format)?"{
            constraints {
                // apply constraints here
            }
        }

        // controllerとactionを指定
        "/product"(controller: "product", action: "list")
        "/product"(controller: "product")

        // viewを指定した設定
        "/"(view:"/index")

        // ステータスコードを指定した設定
        "500"(controller: "errors", exception: MyException)
        "500"(view: "/errors/serverError", exception: MyAnotherException)
        "500"(view:'/error')
        "404"(view:'/notFound')
    }
}
```

ホーム画面をTODOにする

grails-app/controllers/UrlMappings.groovyを以下のように変更する。

```
class UrlMappings {

    static mappings = {
        "/$controller/$action?/$id?(.$format)?"{
            constraints {
                // apply constraints here
            }
        }

        "/"(controller: 'todo')
        "500"(view:'/error')
        "404"(view:'/notFound')
    }
}
```

https://gist.github.com/yamkazu/08e5daed0092a24e205e#file-urlmappings-groovy

静的リソースを使う

Asset Pipeline プラグイン

- Javascript、スタイルシート、画像といった静的リソースを管理するプラグイン
  - Railsからのインスパイアされ開発された
- 以下の機能を提供する
  - アセットの結合、最小化、圧縮
  - フィンガープリント
  - CoffeeScriptやSASSといったアセットのコンパイル
- アセットはgrails-app配下のassets/javascripts、assets/stylesheets、assets/imagesディレクトリに格納する

マニフェストとディレクティブ

- Asset Pipelineを使うにはマニフェストファイルを定義して、それをビューから読み込む
    - マニフェストは複数のリソースを纏めるための定義
    - エントリーポイントのファイルのようなもの
- ディレクティブはそのマニュフェストの中でリソースを指定するための記法
    - マニフェストの中でコメントとして記述する
- よく使うディレクティブは以下の3つ
    - require
        - 指定されたリソースを読み込む
    - require_tree
        - 指定されたパスのリソースを再帰的に読み込む
    - require_self
        - 自身のリソースを読み込む

CSSのマニフェストの例

```
/*
*= require jquery
*= require main
*= require_self
*/

body {
    color: red;
}
```

ビューからの読み込み例

```
<!doctype html>
<html>
    <head>
        <title><g:layoutTitle default="Grails" /></title>
        <asset:stylesheet src="application.css" />
        <asset:javascript src="application.js" />
        <g:layoutHead/>
    </head>
    <body>
        <header>My App</header>
        <g:layoutBody/>
        <footer>Copyright 2015</footer>
    </body>
</html>
```

JavaScript、CSSのライブラリを Asset Pipelineで使う

- 以下の3つの方法がある
    - ファイルをダウンロードしてgrails-app/assets ディレクトリに手動で展開する
- Grailsのプラグインを使う
- WebJarsを使う
    - http://www.webjars.org/

WebJarsを使ってTwitter Bootstrapのリソースを取得する

build.gradleに以下を追加する。

```
...
dependencies {
    ...
    console "org.grails:grails-console"

    provided "org.webjars.bower:bootstrap:3.3.5"
}
...
```

https://gist.github.com/yamkazu/08e5daed0092a24e205e#file-build-gradle

マニフェストを定義する

grails-app/assets/stylesheets/application.cssを以下のように変更する。

```
/*
 *= require webjars/bootstrap/3.3.5/dist/css/bootstrap
 *= require_self
 */
```

https://gist.github.com/yamkazu/08e5daed0092a24e205e#file-application-css

レイアウトファイルからリソースを読み込む

grails-app/views/layouts/main.gspを以下のように変更する。

```
<!doctype html>
<html>
    <head>
        <meta charset="UTF-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <title><g:layoutTitle default="${message(code: 'app.name')}" /></title>
        <asset:stylesheet src="application.css"/>
        <asset:javascript src="application.js"/>
        <g:layoutHead/>
    </head>
    <body>
        <h1><g:message code="app.name" /></h1>
        <g:layoutBody/>
    </body>
</html>
```

https://gist.github.com/yamkazu/08e5daed0092a24e205e#file-main2-gsp

Twitter Bootstrapのスタイル適用する

grails-app/views/layouts/main.gspを以下のように変更する。

https://gist.github.com/yamkazu/08e5daed0092a24e205e#file-main3-gsp

grails-app/views/todo/index.gspを以下のように変更する。

https://gist.github.com/yamkazu/08e5daed0092a24e205e#file-index7-gsp

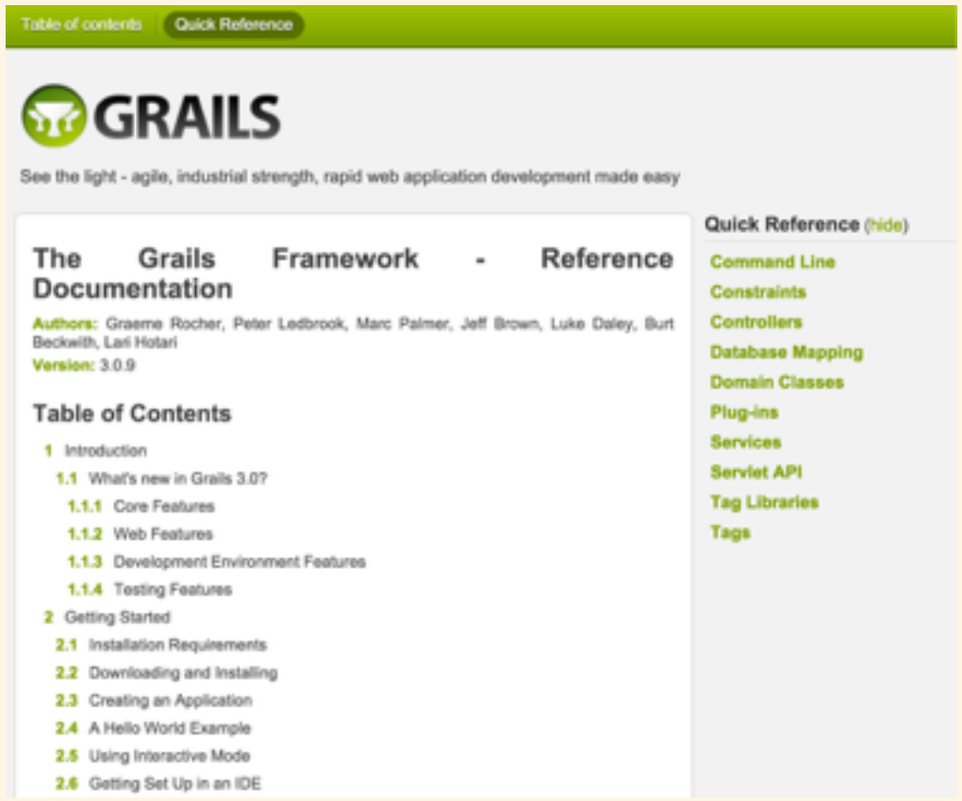
時間があまった場合のアドリブネタ候補

- テスト
- サービス
- インターセプター
- ログ
- ビルド
- プラグイン

参考情報

Grails

本家リファレンス



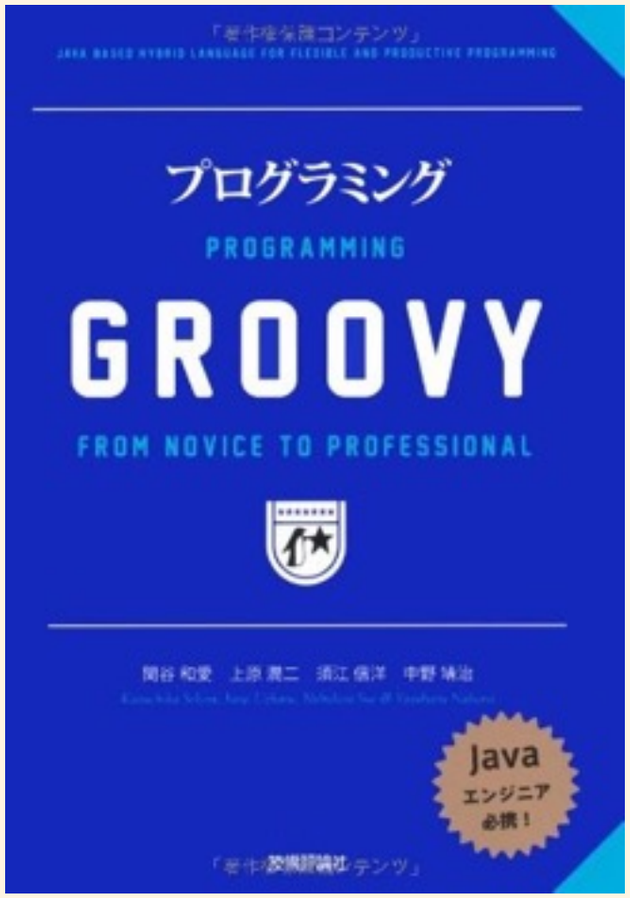
http://grails.github.io/grails-doc/latest/

日本語翻訳

http://grails.jp/doc/latest/

Groovy

プログラミングGROOVY



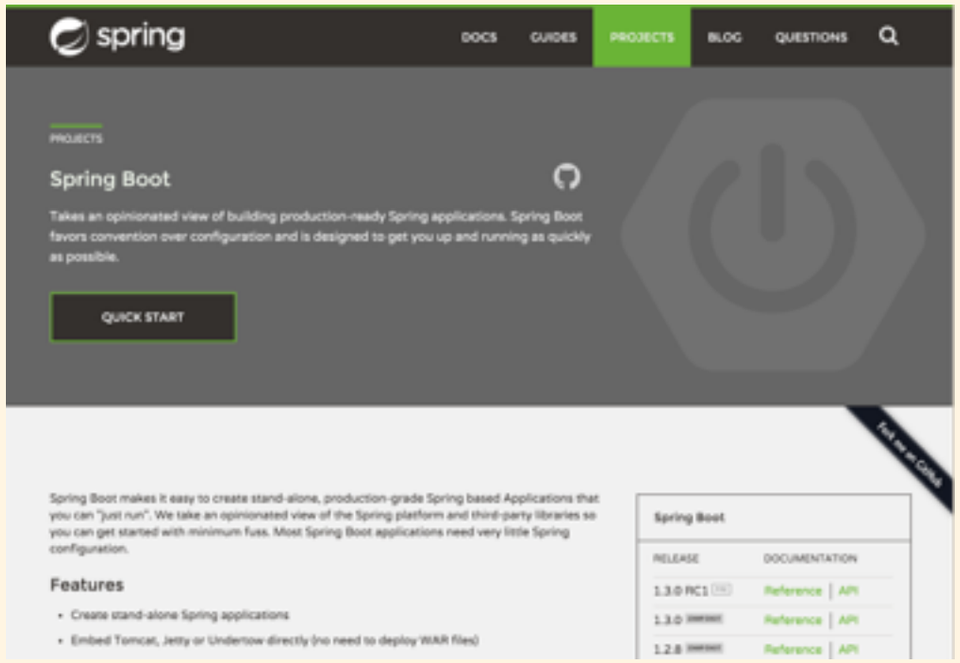
http://gihyo.jp/book/2011/978-4-7741-4727-7

Spring Boot

はじめてのSpring Boot

http://www.amazon.co.jp/dp/4777518655

本家リファレンス



http://projects.spring.io/spring-boot/

Gradle

Gradle徹底入門

http://www.amazon.co.jp/dp/4798136433/

リファレンスの翻訳

http://gradle.monochromeroad.com/docs/

Q & A