

インテリジェンスIT派遣サービス presents
**WebSocketに触れるハンズオン
Play framework 2.1 Javaでつくる
リアルタイムコンテンツ**

<https://github.com/karad/playhandson1>

Creative

原 一浩

@kara_d

自己紹介



ハラ カズヒロ

原 一浩 (@kara_d)

グレーティブ合同会社代表

<http://creative.jp/>

Playはじめて&もくもく会主宰

Scala conferenceスポンサー

日本Play frameworkユーザー会参加



配布物のご案内

➔ **play_20130325.pdf**

このスライド

➔ **slide.txt**

補足テキスト、コードつき

➔ **wsPractice1**

はじめてのPlay完成版

➔ **wsPractice2**

WebSocketアプリ完成版

➔ **wsPractice2_src**

WebSocketアプリハンズオン版

このハンズオンについて

- まず、つくってみる
- そのあと、ちょこっと解説
- できた人はどんどん改造する
- わからないところは助けあう
- 明日火曜日19:00から行うもくもく会でサポートします（最後に告知します）

本日の内容

- Play frameworkの概要
- はじめてのPlayアプリケーション
- WebSocketサンプルを見てみる
- WebSocketアプリケーションの構築
- 終わった人向けの課題

Play frameworkの概要

Play frameworkとは？

- Zenexity(フランス)
- Typesafe(Scala創始者の作った会社)

特徴

- スケーラブル
- ステートレス
- 軽量
- ウェブ向き
- 型安全
- IDEサポート
- Java/Scalaで開発

Play frameworkのサイト

→ 英語版（本家）

<http://www.playframework.com/>

→ 日本語版（有志）

<http://www.playframework-ja.org/>

→ モジュール一覧（暫定）

<https://github.com/garbagetown/playdocja/blob/master/documentation/2.0.4/manual/Modules.md>

Play バージョン 2.xについて

→ ～2012/2

Play 1.x : Javaベース、Groovyテンプレート、Hibernate独自拡張、Pythonによるビルドシステム

→ 2012/2

Play 2.0 : Scalaベース、Java/Scala両方で開発可能、JavaのORMはEBean、Scalaテンプレート、SBTによるビルド

→ 2013/2

Play 2.1 : フォルダやビルド構成の見直し

PlayとScalaとJava

- Playは、ScalaでもJavaでも開発ができる
- Scala版とJava版では、DB周りが大きく違う
- Scala版もJava版も、同じScalaテンプレート
- Java版は、Scala版のラッピング的要素が強い
- Java版にも、関数型プログラミングを補助するライブラリなどが入っている

Playインストールその後

playのパスは通っていますか？

→ UNIX系

```
export PATH=$PATH:/relativePath/to/play
```

→ Windows系

XPは、

;C:\play-2.0

とか、インストール先のディレクトリを入れる

Playコマンドを入力してみよう

→ きちんとPlayが起動することを確認

```
$ play
```

→ 困ったら、play clean-all

```
$ play clean-all
```

はじめてのPlayアプリケーション(1)

初期設定

新規アプリケーション作成

→ 作成先のディレクトリに移動し、下記入力

```
$ play new wsPractice1
```

→ Playコンソールが立ち上がる

What is the application name? [wsPractice1]

> そのままEnterを押す

Which template do you want to use for this new application?

1 - Create a simple Scala application

2 - Create a simple Java application

> 2を選ぶ

アプリのコンソールを立ち上げてみる

→ Playコンソールに入る

```
$ cd wsPractice1
```

```
$ play
```


Playアプリの起動

→ 起動して、 <http://localhost:9000/> へアクセス

```
[wsPractice1] $ run
```

→ 無事スタート画面が表示されればOK

IDEサポート

→ IntelliJ IDEAの場合

`[wsPractice1] $ idea`

→ Eclipseの場合

`[wsPractice1] $ eclipse`

Editorサポート

→ SublimeTextがオススメ

Play 2.0のプラグインがある

フォルダ構成の話

→ 重要なもの

app

app/controllers

app/views

app/models

conf

conf/application.conf

conf/routes

public

project

はじめてのPlayアプリケーション(2)

ソースの編集

世界で一番簡単なHello World

- ステップ1：とりあえず表示してみる
- ステップ2：独自のテンプレートで表示してみる
- ステップ3：Playのセッションを使ってみる

controllers/Application.java

→ 中身をこんな風に変更してみよう

```
return ok(index.render("Hello world"));
```

→ 無事タイトル欄に出たらOK.

views/index.scala.html

- まず中身見てみましょう
- Scalaテンプレートで書かれています

```
@(message: String)
```

```
@main("Welcome to Play 2.1") {
```

```
    @play20.welcome(message, style = "Java")
```

```
}
```

views/main.scala.html

→ こちらが、レイアウトに相当

```
@(title: String)(content: Html)
```

```
<!DOCTYPE html>
```

```
<html>
```

```
  <head>
```

```
    <title>@title</title>
```

```
    .....
```

```
  </head>
```

```
  <body>
```

```
    @content
```

```
  </body>
```

```
</html>
```


views/index.scala.html

→ コメントアウトをしてみよう

```
@*@play20.welcome(message, style = "Java")*@
```

→ 続いて、下記ソースを追加

```
@message
```

controllers/Application.java

→ もう一つメソッド追加

```
public static Result hello() {  
    return ok(index.render("Hello world"));  
}
```

conf/route

→ 7行目に追加

GET /hello

controllers.Application.hello()

Routesファイルについて

- routesはURLを定義する
- コントローラのアクションと1対1になる

GET /hello/:hoge controllers.Application.hello(hoge)

controllers/Application.java

- 先ほどのメソッドにて、
セッションへの書き込みを試みる

```
public static Result hello() {  
    session("username", "kara_d");  
    return ok(index.render("Hello world"));  
}
```

views/index.scala.html

→ @messageの下に追加

@session.get("username")

ビューテンプレートはコンパイルされる

→ Scalaの関数になる

`target/scala-2.10/src_managed/main/views/html/`以下をしてみる

WebSocketサンプルをしてみる

WebSocketのサンプルを起動させてみる

```
$ cd PLAY/samples/java/websocket-chat  
$ play  
[websocket-chat] $ run
```

WebSocketサンプルを読み解くキーワード

- WebSocketにはinとoutがある
- 通信は非同期で行われる
- JSONで通信をしている

WebSocketアプリケーションの構築

wsPractice2_src/

→ フォルダの中身を見てみよう

controllers/Application.java

→ #1を編集

```
public static Result draggable(String username) {  
    session("username", username);  
    return ok(draggable.render("ドラッグアプリ", username));  
}
```

→ #2を編集

```
public static WebSocket<JsonNode> ws() {  
    final String username = session("username");  
    return new WebSocket<JsonNode>() {  
        public void onReady(final WebSocket.In<JsonNode> in, final  
WebSocket.Out<JsonNode> out) {  
            try {  
                WebSocketActor.join(username, in, out);  
            } catch (Exception e) {  
                Logger.error("Can't connect WebSocket");  
                e.printStackTrace();  
            }  
        }  
    }  
};  
}
```

Eventについて

- `events/Event.java`
- `events/Message.java`
- `events/EventUtil.java`
- `events/WebSocketEvent.java`

イベントシステムについて

→ Event.java

イベントのインターフェース定義

→ Message.java

今回のアプリで送受信されるイベントオブジェクト
Eventインターフェースを実装

→ EventUtil.java

イベントオブジェクトかどうかの判定。Optionで返す

→ WebSocketEvent.java

イベントオブジェクトが持つイベントの種類。enumで定義

ビューの解説

→ **views/draggable.scala.html**

ドラッグ可能な矩形が描画されている

→ **views/main.scala.html**

jQuery UIなどを読みこんだり、WebSocketのURLなどを出力したり

フロントエンド実装の解説

→ **public/jquery-ui-1.10.2.custom.min.js**

jQuery UI

→ **public/swfobject.js**

未実装ブラウザ用WebSocket通信用Flash読み込みライブラリ

→ **public/web_socket.js**

未実装ブラウザ用WebSocket通信用JavaScriptライブラリ

→ **public/WebSocketMain.swf**

未実装ブラウザ用WebSocket通信用Flash

→ **public/site.js**

今回使うJavaScript

models/WebSocketActor.java

→ #3を編集

```
private final static ActorRef ref =  
Akka.system().actorOf(new Props(WebSocketActor.class));
```

→ #4を編集

```
Map<String, WebSocket.Out<JsonNode>> members =  
new HashMap<String, WebSocket.Out<JsonNode>>();
```

→ #5を編集

ソースはテキストを参照

Akkaってなに？

- アクターモデル
- 非同期、スケジューリング処理
- 耐障害性 (let-it-crash)
- 位置透過性
- ソフトウェアトランザクショナルメモリ

ActorRefってなに？

- Akkaのアクターへの参照
- アクターモデル

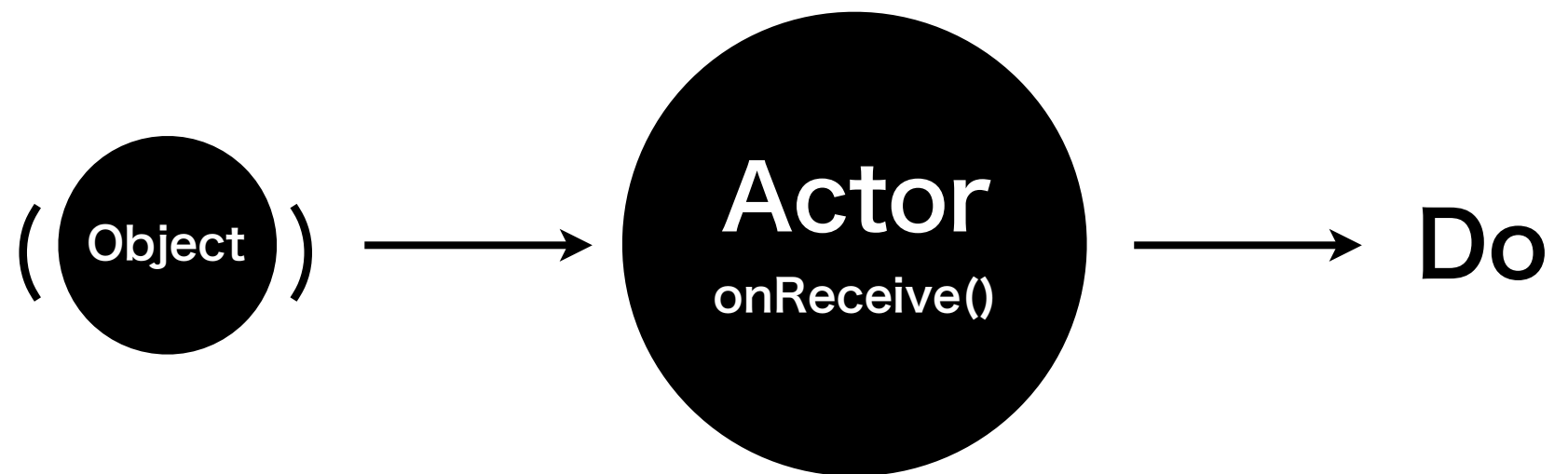
並行計算の数学的モデルの一種

アクターモデルの基本は「全てのものはアクターである」という哲学

アクターは並行的に受信するメッセージに対応した以下のような振る舞いを備えた計算実体（Computational Entity）である

他のアクターとの通信は非同期に発生する

- wikipedia



WebSocket<JsonNode>

WebSocket.In<JsonNode>>とか

WebSocket.Out<JsonNode>>ってなに？

→ **WebSocketはAbstract**

無名クラスにて実装

→ **WebSocket.InはClass**

WebSocket.javaにて定義

→ **WebSocket.OutはInterface**

JavaWebSocket.scalaにて定義

Awaitについて

- `Await.result()`にて、非同期実行された処理の結果を受け取る

Akkaのaskについて

- `ActorRef`に対して、メッセージオブジェクトを投げ、その結果を非同期にもらう。Future型でかえってくる

models/WebSocketActor.java

→ #6を編集

テキストを参照

models/WebSocketMessenger.java

→ #7を編集

テキストを参照

終わった人向けの課題

- ドラッグできるオブジェクトを増やすには？
- 色を変えられるようにしたいんだけど？
- 中身を書き換えられるようにするには
どうしたらいいんだろう？



おつかれさまでした!!
明日26日火曜日19:00より
Playはじめて&もくもく会します!

[Play部屋] 第10回 Play 2.1 はじめて&もくもく会
日本Playframeworkユーザー会
<http://playframeworkja.doorkeeper.jp/events/3380>
参加料：会場利用代 ¥1,000