



実案件で Play Frameworkを 採用してみた

比嘉 築@ECF

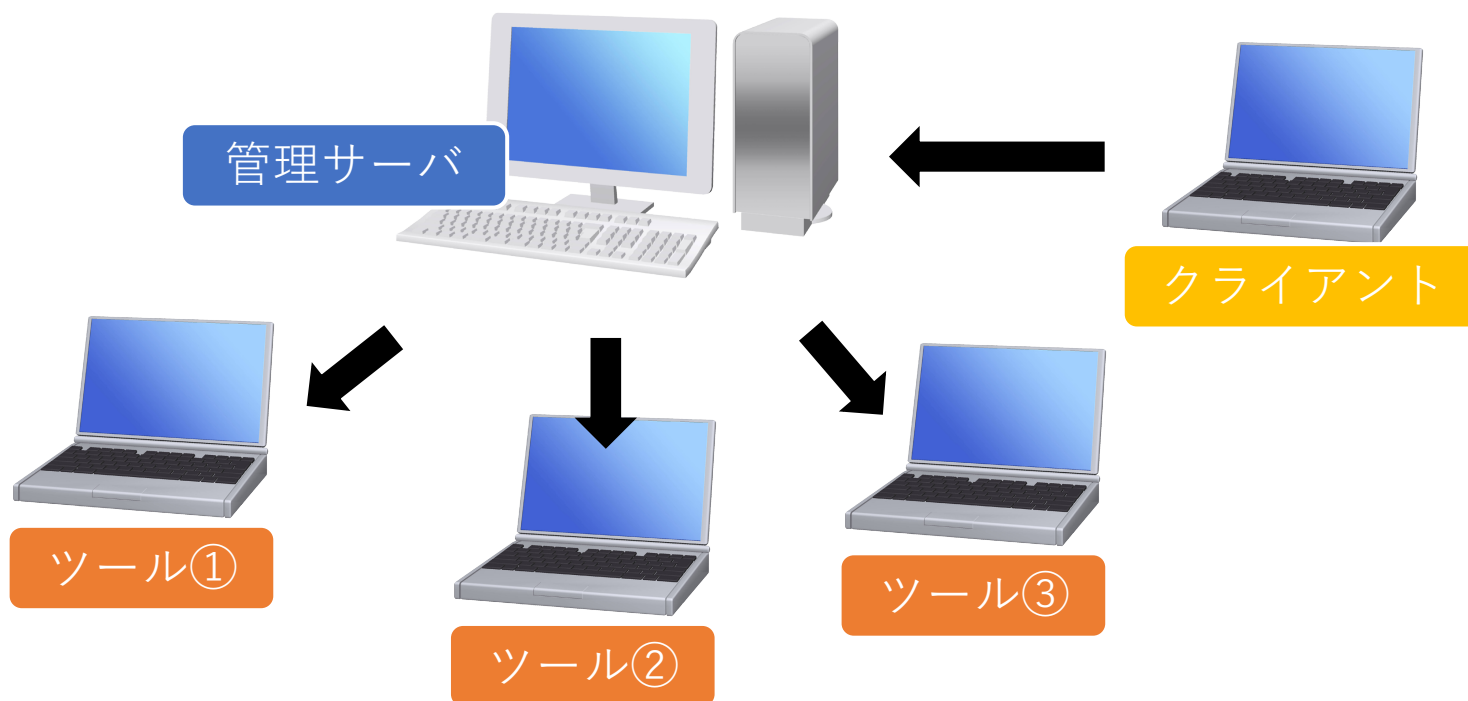


軽く自己紹介を

- **比嘉 築（ひが きずく）**
 - 合同会社イー・シー・エフ代表社員
 - プログラマー/SE
 - 技術講師
- **専門**
 - Android, Java, Python, ... ?
 - 物理学(数値計算)
 - 何でもできますよ

こんな要件が出てきました

- すでにPC上で動いているツールがある
- サーバで複数ツールの一元管理がしたい





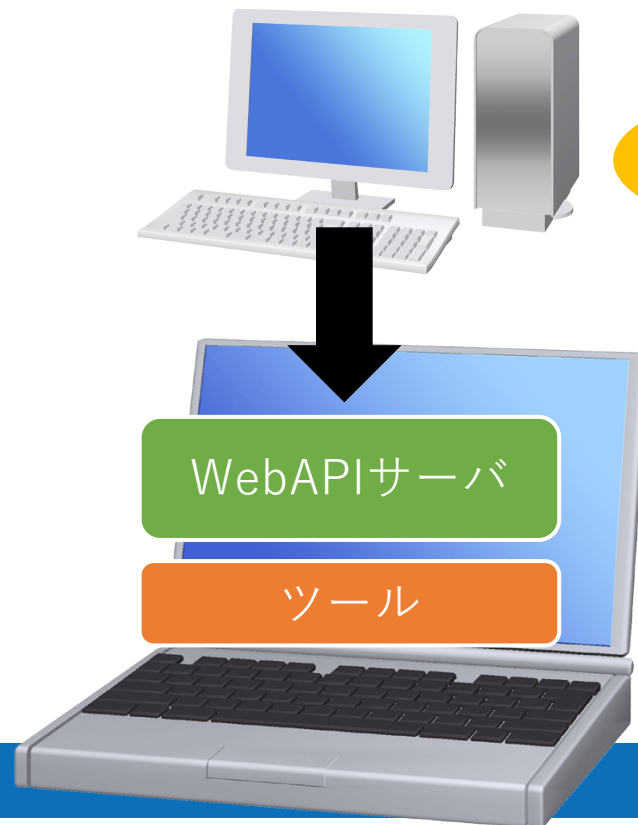
ツールにWebAPIを仕込もう

- PCにWebAPIサーバ乗っけるだけ
- 基本的に今動いているツール自体に手を加える必要なし

Before



After



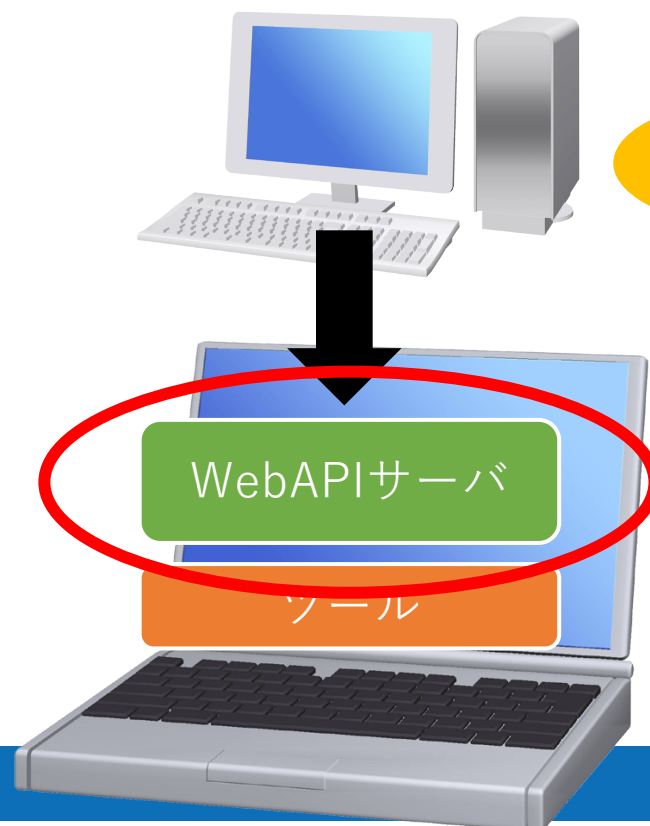
ということで

- このWebAPIサーバ作ってくれない？
- プロトタイプ的なものでいいから

Before



After





プロトタイプとな？（裏要件）

- 現プロジェクトメンバーは全員がWebの経験無し
 - CとかC#な人たちの集まり
 - 少なくとも設計/構築経験は無い
- とりあえず動くもの作って、
- 「引き継げるようにしてくださいね☆」



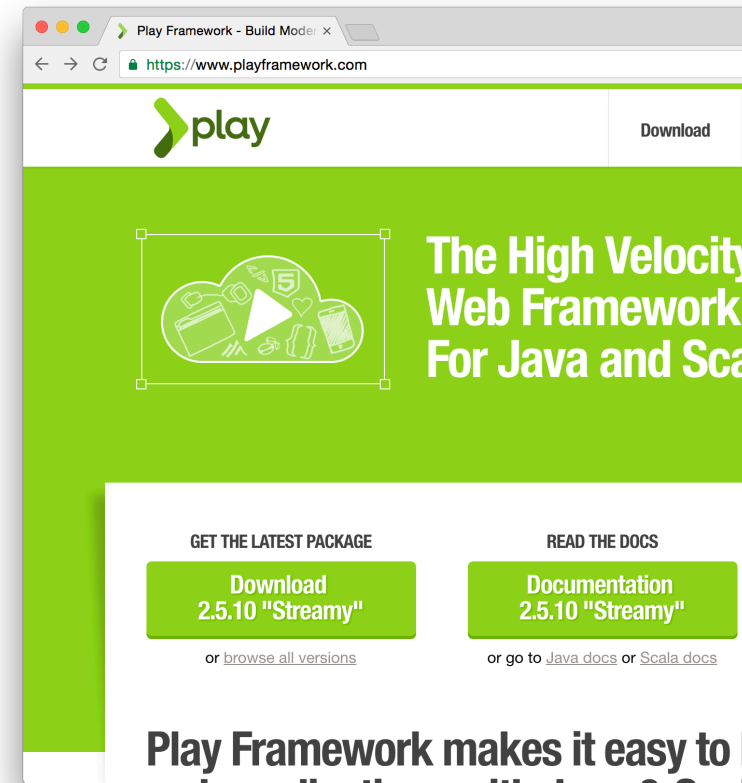
検討

- フレームワークは必須だよね
 - 開発速度重視
 - 数週間でデモ稼働させなきゃいけない
 - プロトタイプって言ってたのに・・・
 - 現メンバーの学習コストがかからないよう
- 言語は？
 - やはり学習コストが気になる
 - まあ、並のエンジニアなら何でもいけるはず？
 - でもトラブル（想定外）があると困る
 - スケジュール的に
 - それなら **Java** でいきましょう



Let's "Play Framework"

- Webアプリケーションフレームワーク
- Java (もしくは Scala)
- よくあるMVCモデル
- Java EE を使っていない
 - 軽量
 - ある意味、前提知識が不要
- Netty(Webサーバ)内蔵
 - この辺もすっとばせる





ユーティリティコマンド操作

- すべては **activator** コマンドで
- 製造の開始はこいつの入手から

The screenshot shows a web browser window at <https://www.playframework.com/download>. The page has a header with "Play Scala Intro", "Download (zip)", and "View on GitHub". The main heading is "Play with Activator". Below it, a red box highlights the link "Activator 1.3.12 including Play 2.5.10" with a download icon, followed by "(671M) — requires JDK 1.8".

Below the browser window, there is a terminal window titled "/Apps/FirstPlayApp" showing the command `activator help` and its output:

```
+ activator help
There are three ways to run activator:

1. Recommended: try `activator ui` to create a project in the UI
2. Use `activator new` to create a project on the command line
3. Load an existing project by re-running activator in a project directory
```

To the right of the terminal, the section "Activator from the command line" provides instructions:

- Add `activator` to your `PATH` to have the command available in your CLI.
- Create a new project from the command line:

```
activator new
```
- Run an existing project from its directory:

```
activator run
```
- Enter the interactive cli (in project directory):



activator new

- 新規プロジェクト作成
- プロジェクト名やJava or Scalaを選択

```
MacBookPro-E3Factory:~ e3factory$ activator new

Fetching the latest list of templates...

Browse the list of templates: http://lightbend.com/activator/templates
Choose from these featured templates or enter a template name:
  1) minimal-akka-java-seed
  2) minimal-akka-scala-seed
  3) minimal-java
  4) minimal-scala
  5) play-java
  6) play-scala
(hit tab to see a list of all templates)
> 5
Enter a name for your application (just press enter for 'play-java')
> HelloPlay
```



activator run

- サーバ起動
- 初回は時間かかるので放置

ポートを指定できる
デフォルトは9000

```
MacBookPro-E3Factory:HelloPlay e3factory$ activator "run 9009"
[info] Loading project definition from /Users/e3factory/HelloPlay/project
[info] Set current project to HelloPlay (in build file:/Users/e3factory/HelloPlay/)

--- (Running the application, auto-reloading is enabled) ---

[info] p.c.s.NettyServer - Listening for HTTP on /0:0:0:0:0:0:0:0:9009

(Server started, use Ctrl+D to stop and go back to the console...)
```



アクセス確認

- ホットデプロイ
- ビルドが入るので初回や変更後は遅い

The screenshot shows a web browser window at `localhost:9009` displaying the "Welcome to Play" page. The page has a green header with the text "Your new application is ready." and a "Browse APIs" button. Below the header, the text "Welcome to Play" is followed by a congratulatory message: "Congratulations, you've just created a new Play application. The next few steps." A yellow box indicates "You're using Play 2.5.10". Below this, the section "Why do you see this page?" explains that the `conf/routes` file defines a route to invoke the application.

Overlaid on the bottom right is a terminal window titled "HelloPlay — java — 80x24". It shows the command `MacBookPro-E3Factory:HelloPlay e3factory$ activator "run 9009"` and the subsequent output logs. The logs indicate that the application is running on `0.0.0.0:9009`, compilation is completed, and the application is started in development mode.

```
MacBookPro-E3Factory:HelloPlay e3factory$ activator "run 9009"
[info] Loading project definition from /Users/e3factory/HelloPlay/project
[info] Set current project to HelloPlay (in build file:/Users/e3factory/HelloPlay/)

--- (Running the application, auto-reloading is enabled) ---

[info] p.c.s.NettyServer - Listening for HTTP on /0.0.0.0:0.0.0.0:9009
(Server started, use Ctrl+D to stop and go back to the console...)

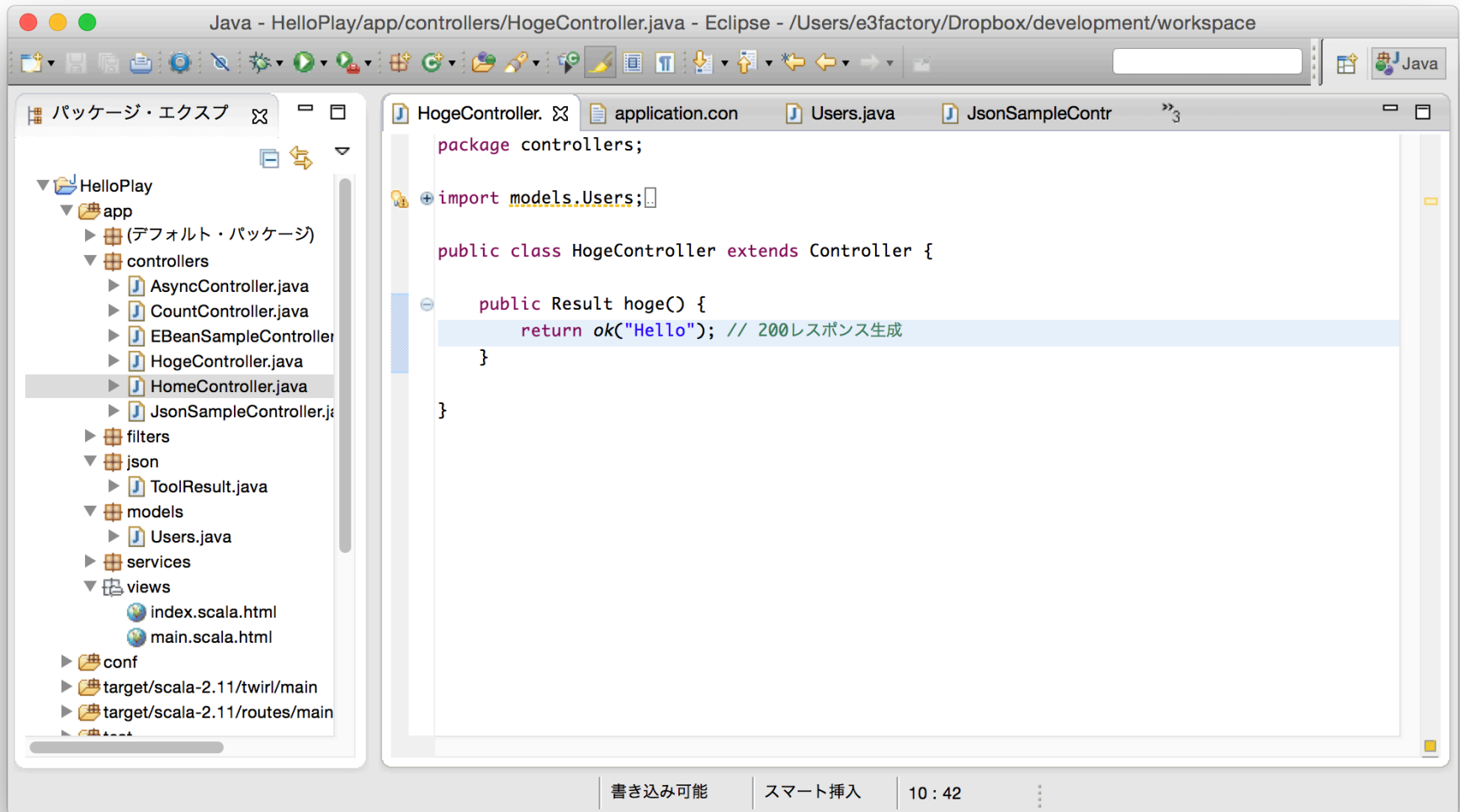
[info] Compiling 6 Scala sources and 10 Java sources to /Users/e3factory/HelloPlay/target/scala-2.11/classes...
[info] 'compiler-interface' not yet compiled for Scala 2.11.7. Compiling...
[info] Compilation completed in 10.434 s
Warning: node.js detection failed, sbt will use the Rhino based Trireme JavaScript engine instead to run JavaScript assets compilation, which in some cases may be orders of magnitude slower than using node.js.
[info] application - ApplicationTimer demo: Starting application at 2016-11-26T08:23:43.203Z
[info] play.api.Play - Application started (Dev)
```



Eclipseで開発

- 各種IDEにも対応可能
- 例 : Eclipse
 - 関連プラグインを読み込む設定
 - 「activator eclipse」
 - Eclipseプロジェクト関連ファイル生成
 - Eclipseでインポート

Eclipseで開発





ルーティング

- routesファイル
- URLとメソッドの対応づけ→アクション

Java - HelloPlay/conf/routes - Eclipse - /Users/e3factory/Dropbox/development/workspace

```
# Routes
# This file defines all application routes (Higher priority routes first)
# ~~~~

# An example controller showing a sample home page
GET    /                controllers.HomeController.index
# An example controller showing how to use dependency injection
GET    /count         controllers.CountController.count
# An example controller showing how to write asynchronous code
GET    /message       controllers.AsyncController.message

# Map static resources from the /public folder to the /assets URL path
GET    /assets/*file   controllers.Assets.versioned(path="/public", file: Asset)

# WebAPI hoge
GET    /hoge          controllers.HogeController.hoge
```



Controller

- Controllerクラスを継承したクラス
- routeで設定したアクションと対応するメソッド
 - 戻り値Resultオブジェクト

Java - HelloPlay/app/controllers/HogeController.java - Eclipse - /Users/e3factory/Dropbox/development/workspace

```
package controllers;

import models.Users;

public class HogeController extends Controller {

    public Result hoge() {
        return ok("Hello"); // 200レスポンス生成
    }

}
```



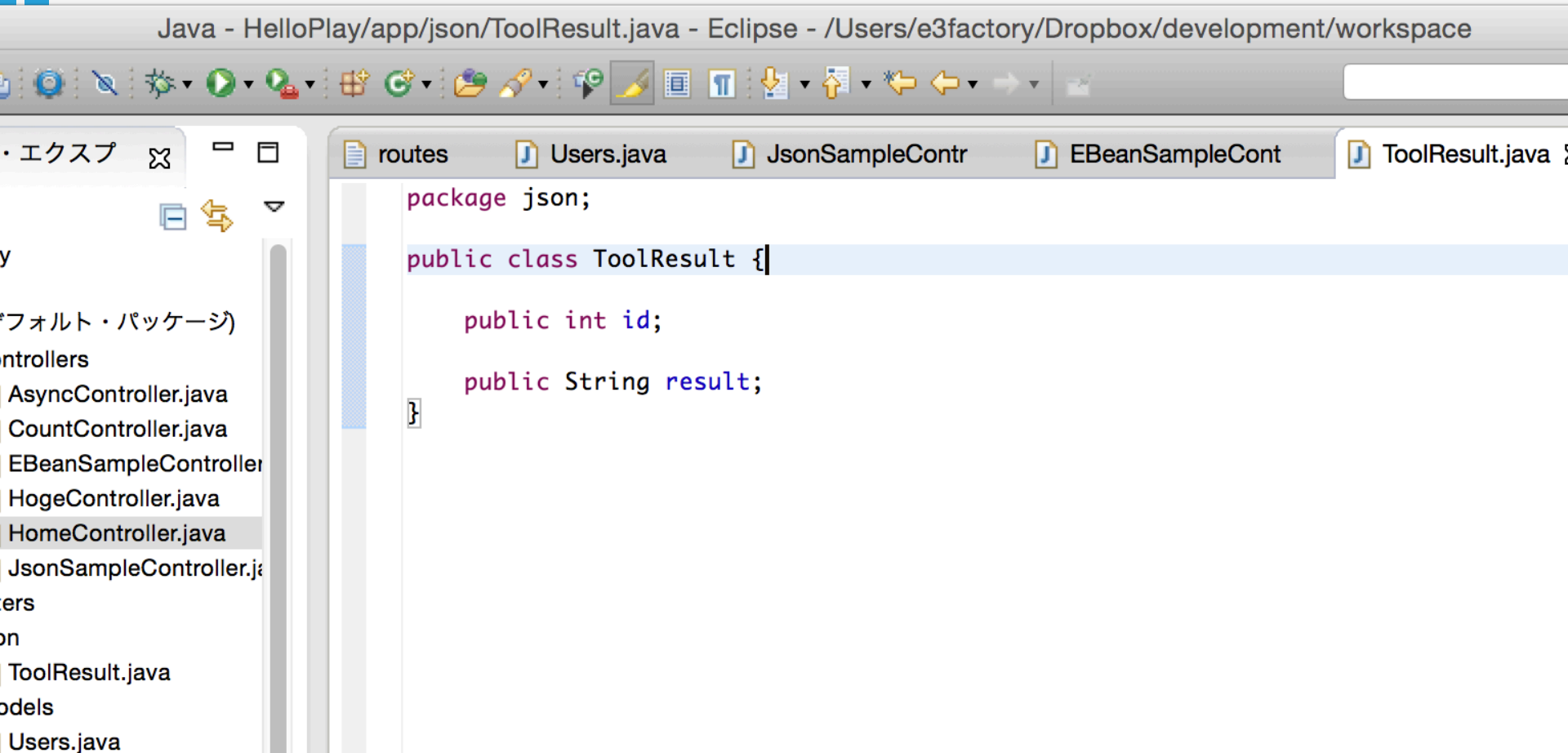

View

- **Scalaベースで記述**
 - 基本的な形式なので特に困りはしない
 - 今回の案件では使わなかったなので割愛
 - あまりわかっていない
 - JSONがビュー代わり



JSONレスポンス

•POJOなBeanを作成





JSONレスポンス

- Jsonオブジェクトに変換するだけ
- Content-Typeを自動で判別して設定

Java - HelloPlay/app/controllers/JsonSampleController.java - Eclipse - /Users/e3factory/Dropbox/development/workspace

```
package controllers;

import json.ToolResult;
import play.libs.Json;
import play.mvc.*;

public class JsonSampleController extends Controller {

    public Result run() {

        ToolResult tr = new ToolResult();
        tr.id = 1234;
        tr.result = "ALL OK";

        return ok(Json.toJson(tr));
    }
}
```

Project Explorer: HelloPlay, app, (デフォルト・パッケージ), controllers, AsyncController.java, CountController.java, EBeanSampleController.java, HogeController.java, HomeController.java, JsonSampleController.java, filters, json, ToolResult.java, models, Users.java, services, views, index.scala.html



Model

- **標準ではEBeanを利用**
 - JPAっぽい
 - 別のO/Rマッピングを使うことも可能
 - DB関連設定は application.conf へ記載
- **Modelクラスを継承したクラスを作成**
 - テーブル/カラムに対応したBean
 - アノテーションでDB関連属性を指定
- **Modelに変更があると自動的にDB変更**
 - Evolution機能

Model



Java - HelloPlay/app/models/Users.java - Eclipse - /Users/e3factory/Dropbox/development/workspace

Users.java

```
package models;

import java.util.Date;

@Entity
public class Users extends Model {

    @Id
    public Long id;

    public String name;

    public String mail;

    @Version
    public Date accessDate;
}
```



- **WebAPI程度ならすんなり**
 - エラーレスポンスやCORS対応など問題なくフレームワーク内で吸収可能
- **ドキュメントは思ったより豊富な印象**
 - 公式ドキュメントが丁寧
 - ただし一般の情報はScalaばかり
- **「Playを使いたいからScalaを勉強」はありだと思う**
 - Javaでは理由がない限りどうだろう？