PlayとRails、どうして差がついたのか

2016/5/13 オプト社内勉強会

自己紹介

渋谷　充宏 @m4buya

- サーバーサイドプログラマ
- Ruby / Scala
- https://github.com/mshibuya
- RailsAdmin committer

本日のテーマ

オプト開発２部のプロダクトで多く使われているPlay Frameworkと自分が慣れ親しんだRailsを独断と偏見(と些少な知識)をもって比較することで、
得手不得手など見えてきたらいいな…という感じです

Play Framework

Play Framework

https://www.playframework.com/documentation/ja/2.3.x/Home

Play は、現代の web アプリケーション開発に
必要なコンポーネント及び API を統合した生産性の高い
Java と Scala の web アプリケーションフレームワークです。

Play の特徴は、ライトウェイト、ステートレス、webフレンドリーな
アーキテクチャであること、機能予測のしやすさです。また、
Iteratee IO をベースにしたリアクティブモデルのおかげで、
スケーラブルなアプリケーションでも
CPU、メモリ、スレッドなどのリソース消費が最小限になっています。

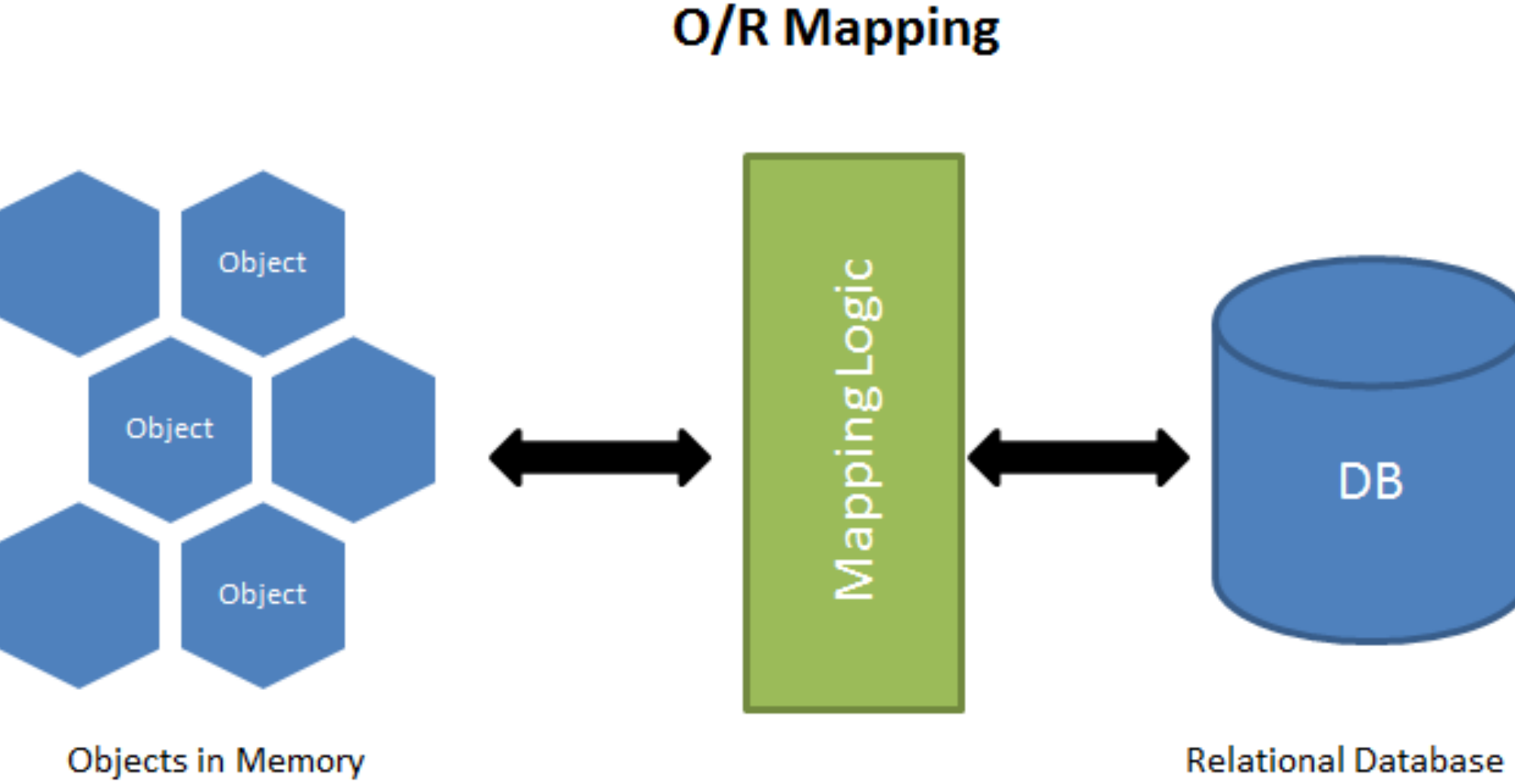
Ruby on Rails

Ruby on Rails

https://ja.wikipedia.org/wiki/Ruby_on_Rails

> Railsの基本理念は
> 「同じことを繰り返さない」（DRY:Don't Repeat Yourself）と
> 「設定より規約」（CoC:Convention over Configuration）である。

ORM


O/R Mapping
Object
Object
Object
Mapping Logic
DB
Objects in Memory
Relational Database

Play

- 標準ではEBean
  - 知らない。。
- Slick
- Skinny
- ScalikeJDBC
  - ORMなの?

Rails

- ActiveRecord
- Mongoid(MongoDB)
- DataMapper(開発止まってる)
- Sequel

Scaffold

Play

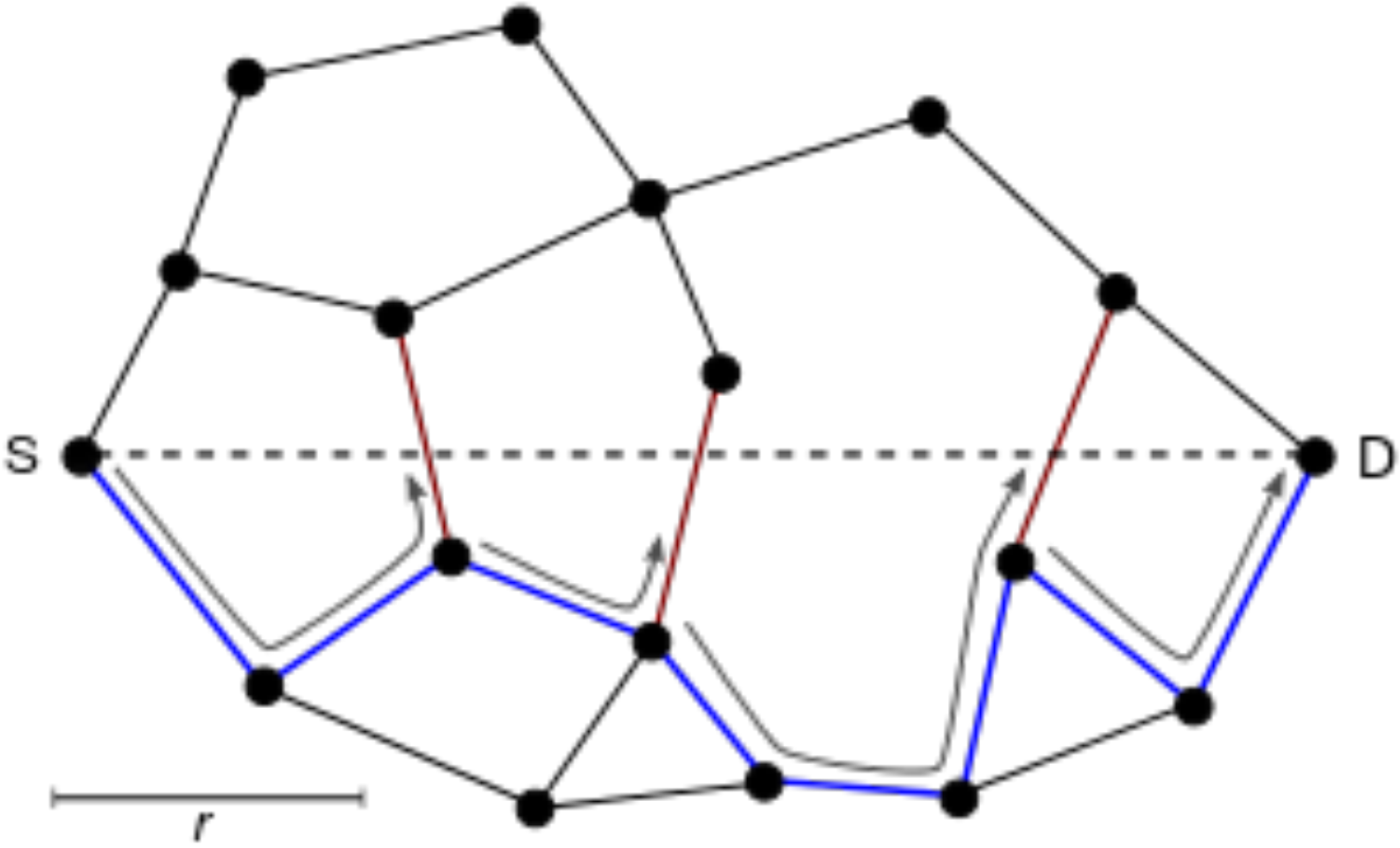
- Scalaだとなさそう？
- Javaならある
  - https://github.com/lmcalpin/Play–Scaffold

Rails

強力

```
  create  app/controllers/welcome_controller.rb
   route  get 'welcome/index'
  invoke  erb
  create    app/views/welcome
  create    app/views/welcome/index.html.erb
  invoke  test_unit
  create    test/controllers/welcome_controller_test.rb
  invoke  helper
  create    app/helpers/welcome_helper.rb
  invoke  assets
  invoke    coffee
  create      app/assets/javascripts/welcome.js.coffee
  invoke    scss
  create      app/assets/stylesheets/welcome.css.scss
```

Routing



Play

```
GET   /clients/:id          controllers.Clients.show(id: Long)
GET   /clients/all          controllers.Clients.list()
POST  /clients              controllers.Clients.create()
PUT   /clients/:id          controllers.Clients.update(id: Long)
```

- 外部DSL
- マッピングのみ

Rails

```
resources :clients
namespace :admin do
  resources :users
end
```

- 内部DSL
- 高機能

View

Play

引数に型がある！

```
@(customer: Customer, orders: List[Order])

<h1>Welcome @customer.name!</h1>

<ul>
@for(order <- orders) {
  <li>@order.title</li>
}
</ul>
```

Rails

```
<h1>Names of all the people</h1>
<% @people.each do |person| %>
  Name: <%= person.name %><br>
<% end %>
```

Form

Play

case classを作っておき、マッピング

```
case class User(name: String, age: Long)

val userForm = Form(
  mapping(
    "name" -> text,
    "age" -> number
  )(UserData.apply)(UserData.unapply)
)
```

Rails

マッピングとかない。ARオブジェクトにつっこんで終わり

```
Person.create(params[:person])
```

並行処理


EAST
INTERSTATE
40
NORTH
BUSINESS
85
NORTH
29
EAST
70
NORTH
220
SOUTH
421

Play

- 非同期・イベントドリブン
- スレッドプールは低め
  - ブロッキングなIOをたくさんするとパフォーマンスに影響がでる

Rails

- マルチプロセス or マルチスレッド
- たくさん並べればなんとかなる
  - メモリとかたくさん食いがち

まとめ

- Play
  - Scalaいい
    - 型
- Rails
  - Ruby素敵

ご静聴ありがとうございました