

2016年度 Ruby on Rails勉強会

2016年4月18日
乃村研究室
江見圭祐, 吉田尚史

今日の流れ

- (1) Ruby on Railsの紹介
- (2) Ruby on Railsの環境構築
- (3) アプリケーションの作成
- (4) 作ってみよう商品管理システム

Ruby on Railsの紹介

はじめに

<Ruby on Railsとは>

Rubyで書かれたWebアプリケーションフレームワーク

II

Webアプリケーションの開発を支援する
クラスやライブラリの集まり

<Railsを学ぶ前に>

そもそもWebアプリケーションってどういう仕組みで動くの？
なんでRailsを使うの？

Railsの事前知識としてWebアプリケーションについて学習

Webサーバとブラウザの関係

例) <http://www.okayama-u.ac.jp/index.html> にアクセス

通信プロトコル: HTTP
通信先(サーバ): www.okayama-u.ac.jp
サーバへの要求: /index.html を GET する

Webブラウザ
(クライアント)



(1) リクエスト送信

Webサーバ



(3) レスポンス返却

(4) 返却内容を整形して表示

(2) 要求に対応した処理

index.html を返却

Webサーバとブラウザの関係

例) <http://www.okayama-u.ac.jp/index.html> にアクセス

通信プロトコル: HTTP

通信先(サーバ): www.okayama-u.ac.jp

サーバへの要求: /index.html を GET する

Webブラウザ
(クライアント)



(1) リクエスト送信

Webサーバ



(3) レスポンス受信

Railsを用いて実装

(2) 要求に対応した処理

(4) 返却内容を整形して表示

index.html を返却

Ruby on Rails

Ruby製のWebアプリケーションフレームワーク

Webアプリケーション開発に必要な機能を用意
例: Webサーバの立ち上げ
データベース管理

Railsの基本理念

- (1) DRY -Don't Repeat Yourself (同じことを繰り返さない)
重複を排除する
- (2) CoC -Convention over Configuration (設定より規約)
規約に従う事で面倒な設定を減らす

乃村研究室のRuby on Rails利用例

(1) LastNote



(2) camome

(3) jay

(4) 各自の研究プロジェクト

Rails で日付を返すページを作ろう

- **Controller** を使ってページを作る

```
class WelcomeController < ApplicationController
  def index
    render text: Date.today.to_s
  end
end
```

今日の日付をHTTP Responseに包んで返す

- このページに menu をつけたい

menu 付きページを作ろう

- Controller のコードを修正して menu をつける

```
class WelcomeController < ApplicationController
  def index
    render text: "<ul><li>menu1</li>\n
                  <li>menu2</li>\n
                </ul>" + Date.today.to_s

  end
end
```

- Controller だけでページを全て生成するのは大変
- 表示を調整するためのコードを分離したい

表示のためのコードの分離

- View は Controller で処理した情報をどう表示するかを担当
- Controller は情報の処理だけを担当
- 先ほどのコードを Controller と View に分離

– Controller

```
class WelcomeController < ApplicationController
  def index
    @date = Date.today.to_s
  end
end
```

– View

```
<ul><li>menu1</li><li>menu2</li><ul><%= @date %>
```

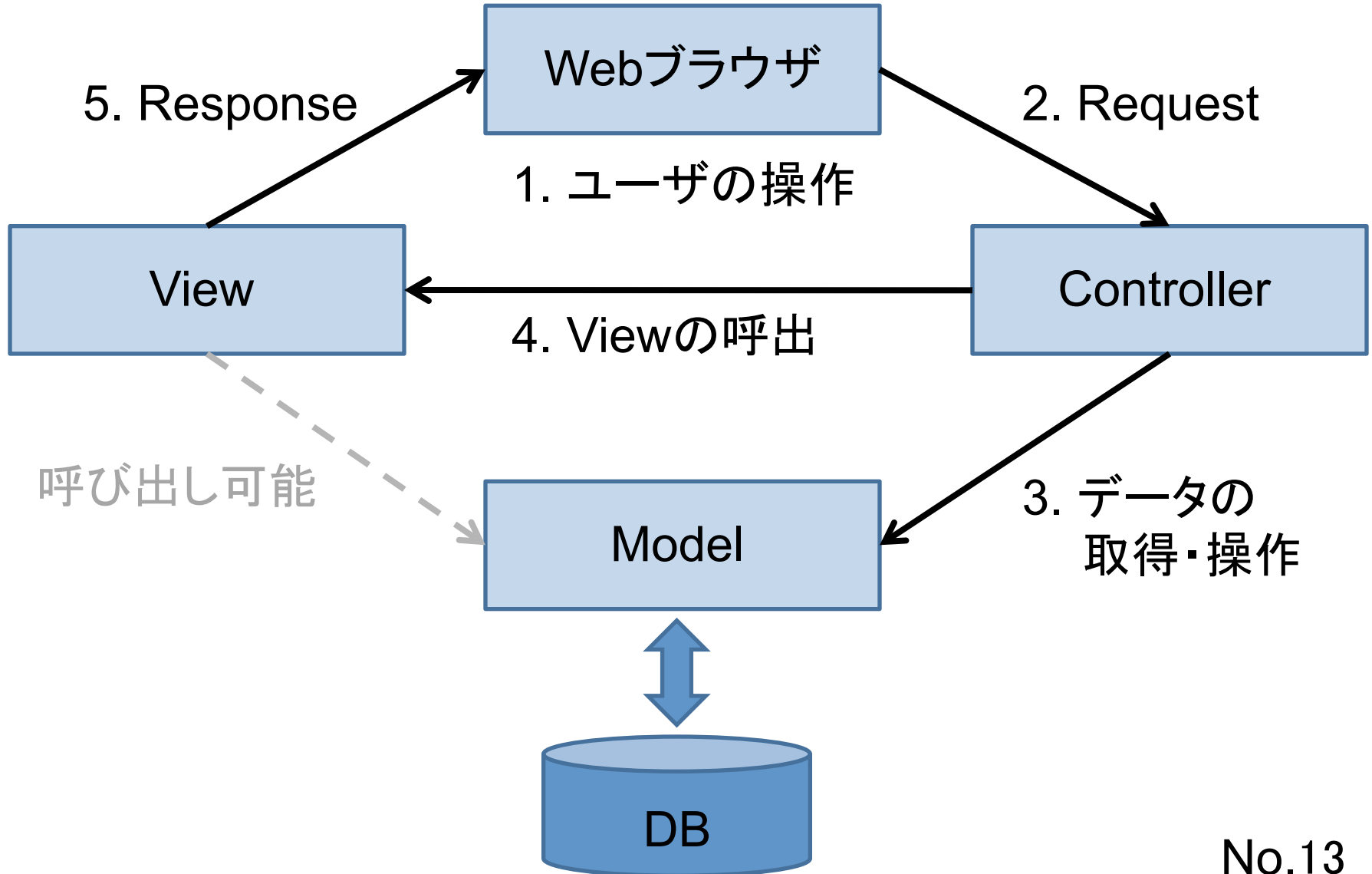
DB の情報でページ生成

- DB の情報を利用する場合
 - **View:** Controller から受け取った情報を基にページ生成
 - **Controller:** DB の情報を扱うオブジェクトから情報を取得
= Model

取得した情報を加工して View に渡す

- Model ≡ 構造体配列
データ永続化のために DB に情報を保存しているイメージ

MVC



Ruby on Railsの環境構築

railsのインストール

bundlerを用いてrails(gem)をインストールする

bundler: gemを管理するgem

(1) 作業ディレクトリでRuby2.2.1を利用

```
$ rbenv local 2.2.1
```

(2) bundlerインストール

```
$ gem install bundler
```

(3) gemをインストールする準備

```
$ bundle init  
$ echo "gem 'rails'" >> Gemfile
```

(4) gemをvendor/bundle以下にインストール

```
$ bundle install --path vendor/bundle
```

アプリケーションの作成

手順1: アプリケーションの作成

(1) アプリケーションを作成

```
$ bundle exec rails new . --skip-bundle
```

作業ディレクトリ以下にアプリケーションのファイルが生成

生成される主要なディレクトリ

- └ app アプリケーション本体を格納
 - └ controllers コントローラを格納
 - └ helpers ビューを支援するメソッド群を格納
 - └ models モデルを格納
 - └ views ビューを格納
- └ config 設定ファイルを格納
- └ db データベースファイルを格納

(2) gemをvendor/bundle以下にインストール

```
$ bundle install --path vendor/bundle
```

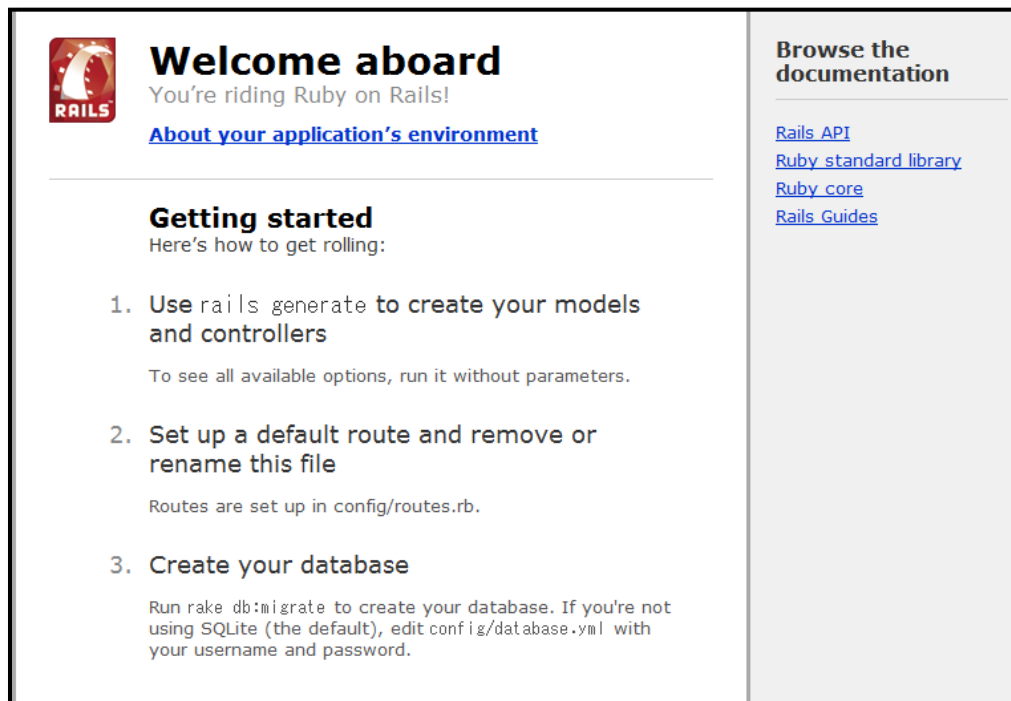
手順2: アプリケーションの動作確認

(1) アプリケーションを起動

```
$ bundle exec rails server -p [任意のポート番号]
```

(2) 起動を確認

webブラウザで<http://localhost:任意のポート番号/>にアクセス



The screenshot shows the default Rails application landing page. On the left, there's a 'Welcome aboard' section with the Rails logo and a 'Getting started' list. On the right, there's a sidebar with links to documentation.

Welcome aboard
You're riding Ruby on Rails!

[About your application's environment](#)

Getting started
Here's how to get rolling:

1. Use rails generate to create your models and controllers
To see all available options, run it without parameters.
2. Set up a default route and remove or rename this file
Routes are set up in config/routes.rb.
3. Create your database
Run rake db:migrate to create your database. If you're not using SQLite (the default), edit config/database.yml with your username and password.

Browse the documentation

- [Rails API](#)
- [Ruby standard library](#)
- [Ruby core](#)
- [Rails Guides](#)

手順3: 静的なページ作成

(1) 表示させているページについて確認
public/を見る

(2) Hello Worldページ作成
public/hello.htmlを作成
以下を記述する

```
<html>
  <head>
    <title>Hello world !!</title>
  </head>
  <body>
    Hello world !!
  </body>
</html>
```

(3) <http://localhost:3000/hello>にアクセス

手順4: HelloWorldページを作成

- (1) app/controllersにhello_controller.rbを作成
以下を記述する

```
class HelloController < ApplicationController
  def world
  end
end
```

- (2) app/views以下で次のことをする
- (A) app/views/helloディレクトリを作成
 - (B) app/views/hello/にworld.html.erbを作成
 - (C) world.html.erbの中身は先に作成したpublic/hello.htmlと同じ
- (3) config/routes.rbに以下を追記

```
get ':controller(/:action(/:id))(.:format)'
```

- (4) <http://localhost:3000/hello/world>にアクセス

手順5: ビューへのプログラムの埋め込み

- (1) app/views/hello/world.html.erbにプログラムを埋め込む
world.html.erbに赤の部分を追記

```
<html>
  <head>
    <title>Hello world !</title>
  </head>
  <body>
    Hello world ! <%= Time.now %>
  </body>
</html>
```

- (2) <http://localhost:3000/hello/world>にアクセス
赤の部分<% %>はRubyのプログラムとして実行される

手順6: コントローラに変数を渡す

(6) controllerから値を渡す

(A) app/controllers/hello_controller.rbに赤の部分を追記

```
class HelloController < ApplicationController
  def world
    @time = Time.now
  end
end
```

(B) app/views/hello/world.html.erbの赤の部分を変更

```
<body>
  Hello world ! <%= @time %>
</body>
```

(7) <http://localhost:3000/hello/world>にアクセス

controllerで定義した変数はviewで参照可能

作ってみよう商品管理システム

商品管理システム概要

<機能>

商品管理機能

- (1) 商品の一覧表示
- (2) 商品の詳細表示
- (3) 商品の新規登録
- (4) 商品の登録内容変更
- (5) 商品の削除

Listing products

Title	Description	Image url
RailsによるアジャイルWebアプリケーション 第3版	Railsを始めるならこれ！ ただし、第3版ではRails3系まで未対応。	Show Edit Destroy
Rubyレシピブック 第2版 268の技	良く使います。	Show Edit Destroy
Railsレシピブック 183の技	Rails3には未対応。	Show Edit Destroy

[New Product](#)

商品管理システムで作るもの

商品 = Product

app/controllers

products_controller.rb

Productを操作するコントローラ

app/models

product.rb

Productのデータを扱うモデル

app/views

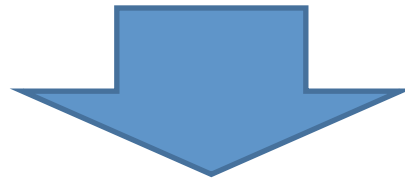
products/index.html.erb

products/show.html.erb

products/new.html.erb

products/edit.html.erb

各種操作に対応するView



一気に作ってくれるscaffold

Scaffoldの概要

<Scaffoldとは>

Webアプリケーションの骨格となるファイルを自動生成するコマンド

基本的な機能を持つアプリケーションの雛形を生成する

基本的な機能とは以下の4つ（CRUD）

- (1) Create (生成)
- (2) Read (読み取り)
- (3) Update (更新)
- (4) Delete (削除)

<Scaffoldの利点>

命名規則に則ったファイル名で生成

開発を効率的に進められる

Scaffoldを使ってみる

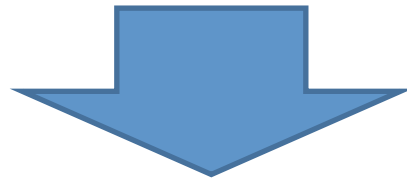
rails generate: railsに関するファイルを生成するコマンド

scaffold は基本的な機能の実現に必要なファイルをすべて生成
モデル名

```
$ bundle exec rails generate scaffold Product  
title:string description:text image_url:string  
price:decimal
```

カラム名:データ型

モデル名, カラム名, およびデータ型を設定し, 実行



productに関するファイル一式を自動生成

Scaffoldで生成されるファイル(1/2)

(A)コントローラ, ビュー, モデルのファイル

app/controllers/products_controller.rb ... controller

app/models/product.rb ... model

app/views/products/index.html.rb

app/views/products/show.html.rb

app/views/products/new.html.rb

app/views/products/edit.html.rb

app/views/products/_form.html.rb

} view

Scaffoldで生成されるファイル(2/2)

(B) マイグレーションファイル

db/migrate/[時刻]_create_products.rb

データベースにproductのスキーマを追加するファイル

(C) ヘルパファイル

app/helpers/products_helper.rb

productに関するビューの共通処理を記述するファイル

(D) テストファイル

test/unit/product_test.rb

test/unit/helpers/products_helper_test.rb

productに関するテストを記述するファイル

Scaffoldした内容を反映・確認

(1) データベースにproductを反映

Scaffoldで生成したマイグレーションファイルを利用

```
$ bundle exec rake db:migrate
```

(2) 自動生成された商品管理機能を触ってみる

```
$ bundle exec rails s -p 3000
```

<http://localhost:3000/products>にアクセスする

Listing products

Title Description Image url

[New Product](#)

商品登録をやってみよう！

モデルの仕組みについて

(1) app/models/product.rbをしてみる

Productクラスについてほとんど何も書かれていない
ただし, ProductはActiveRecordを継承している

(2) Productクラスを使ってみる

(A) コンソールを利用

```
$ bundle exec rails console
```

または

```
$ bundle exec rails c
```

アプリケーションをコンソールで操作可能

モデルの仕組みについて

(2) Productモデルを使ってみる

(A) コンソールを利用

```
$ bundle exec rails console
```

(B) Productモデルを操作

(a) 保存されているProductモデルのデータを一覧表示

```
irb(main) > Product.all
```

(b) 特定の条件に合うProductモデルを表示

```
irb(main) > Product.where(:title => “[タイトル]”)
```

(c) 特定の項目を表示

```
irb(main) > product = Product.first  
irb(main) > product.title
```

データベースをオブジェクトとして扱える

モデルの仕組みについて

データベースの中身をオブジェクトとして見る仕組み

データベースとオブジェクトの連携が名前によって決まっている

(a) `Product.all`

`products`テーブルの各行から`Product`オブジェクトを生成

(b) `Product.where(:title => “[タイトル]”)`

`products`テーブルに対して、`title` 列が[タイトル]である行を検索し、一致した行の`Product`オブジェクトを生成

(c) `product = Product.first`

`product.title`

`products`テーブルの最初の行の`title`列を返す

RailsのO/RマッパであるActiveRecordがデータベースに応じて動的にメソッドを生成する

マイグレーション

データベースの構造変更を管理する機能

例: 今回のマイグレーションファイル

```
class CreateProducts < ActiveRecord::Migration
  def change
    create_table :products do |t|
      t.string :title
      t.text :description
      t.string :image_url
      t.decimal :price

      t.timestamps
    end
  end
end
```

このマイグレーションファイルを適用した時にデータベースに加わる変更

すべてのマイグレーションファイルを適用 = 最終データベースの状態