

# Ruby on Rails 講義

## 第14回

Kuniaki IGARASHI/igaiga  
2012.7.12 at 一橋大学  
社会科学における情報技術とコンテンツ作成III  
(ニフティ株式会社寄附講義)

○ 剰余金の配当に関するお知らせ

○ ニフティ、「@nifty EMOBILE LTE 定額にねんプラン」の提供を開始

○ 「@nifty温泉」で「母の日 全国一斉！100のありがとう風呂」特設サイト公開

○ 「スマブレ！」のサービス停止について

○ ニフティとサンリオウェーブ、iOS向けアプリ「Hello Kitty Worl...」

○ 平成24年3月期 決算短信

○ 特別損失の計上に関するお知らせ

○ 「シユフモ」登録会員数150万人を突破、「2012年主婦の全国節電調査（冬季...」

ニフティとなら、きっとかなう。

With Us, You Can.

# ニフティ株式会社

アット・ニフティ  
楽しいサービスがいっぱい



アクセスマップ  
大森から西新宿へ移転いたしました



@nifty Web募金  
東日本大震災復興支援  
募金受付中



2012年4月25日 IR [特別損失の計上に関するお知らせ](#)

2012年4月25日 IR [剰余金の配当に関するお知らせ](#)

2012年4月24日 IR [2012年3月期 決算短信](#)

2012年4月1日 カテゴリー [ニフティ、「@nifty EMOBILE LTE 定額にねんプラン」の提供を開始](#)

2012年4月1日 カテゴリー [ニフティとサンリオウェーブ、iOS向けアプリ「Hello Kitty World」を台湾で提供開始](#)

2012年4月10日 お知らせ [「@nifty温泉」で「母の日 全国一斉！100のありがとう風呂」特設サイト公開](#)

# 提供

講師

# 五十嵐邦明

株式会社万葉

エンジニア



GARASHI

.9.25 at 高専カンファ

Teaching Assistant 演習 健二  
クックパッド株式会社 エンジニア



おさらい

# Webアプリの基本動作

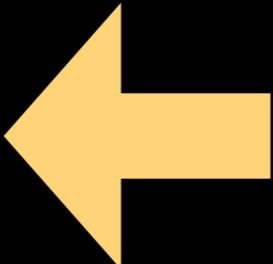
ブラウザにURLを入力してアクセス

リクエスト

URL : <http://cookpad.com/>

Web Server

Web App



Browser

<http://cookpad.com/>

# Webアプリの基本動作

レスポンスとしてHTMLが返ってくる

リクエスト

URL : <http://cookpad.com/>

Web Server

Web App

レスポンス  
HTML

Browser

<http://cookpad.com/>



# HTML

## HyperText Markup Language

### Webページを記述するための言語

```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
2
3 <html>
4 <head>
5 <script type="text/javascript">var __rendering_time = (new Date).getTime();</script>
6 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
7
8 <meta name="robots" content="noarchive" />
9
10 <link rel="alternate" media="handheld" type="text/html" href="http://m.cookpad.com/" />
11 <link rel="apple-touch-icon" href="/images/device/apple-touch-icon.png" />
12
13 <title>レシピ検索No.1／料理レシピ載せるなら クックパッド</title><meta content=",レシピ,簡単,料理,COOKPAD,くっくぱっど,recipe" name="keywords" />
content="日本最大の料理レシピサイト。125万品を超えるレシピ、作り方を検索できる。家庭の主婦が実際につくった簡単実用レシピが多い。利用者は
を公開することもできる。" name="description" />
14
15
16
17
18
19 <script src="/javascripts/jpack.js?1340964911" type="text/javascript"></script>
20
21
22 <script type="text/javascript">AsyncView.load({"specify":["kondate_ext_kondate_unit"],"assigns":"{\\\"kondate_ext_kondate_unit\\\":{}}
,"controller_name":"top","action_name":"top"});</script>
23
```

# ブラウザでHTML表示

## 右クリックからソースを表示



# ブラウザの主な機能

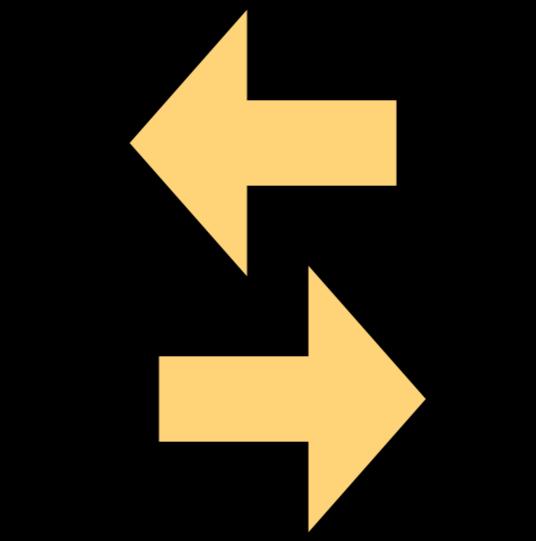
1. リクエストを飛ばす
2. レスポンスでもらったHTMLを人が見る形式で表示する

リクエスト

URL : <http://cookpad.com/>

Web Server

Web App



レスポンス  
HTML

Browser

<http://cookpad.com/>



# インターネットへのアクセス

Internet

Web Server

Web App

Browser

リクエスト



レスポンス  
HTML

WebServerはネットの向こうにある

# 開発時の構成

自分のPC

**Web Server**  
localhost:3000

**Web App**

URL  
リクエスト

**Browser**

レスポンス  
HTML

1台のマシンの中で開発、アクセス可能です。

# Rails アプリ

自分のPC

Web Server  
localhost:3000

Web App  
Rails App

URL  
リクエスト

レスポンス  
HTML

Browser

ブラウザで動作  
確認しながら進  
めます。



これから作るのはこの部分

# Railsアプリを作るため の環境セットアップ

- **RailsInstaller**  
**特に不要**

最近mac版も出ました。 <https://github.com/railsinstaller/railsinstaller-nix/downloads>

- **VM(Ubuntu)**

\$ rvm 1.9.3 --default

\$ gem install rails --no-ri --no-rdoc

\$ sudo apt-get install nodejs

※ --no-ri --no-rdoc はドキュメントインストールを省略するgemのオプション

- **Mac**

\$ sudo gem install rails --no-ri --no-rdoc

(RVMを使ってる場合はsudo 不要)

では、Railsアプリを  
つくりてみましょう

# Railsアプリをつくり動かしてみる

1. Railsアプリをつくる
2. WebServerとRailsアプリを起動する
3. ブラウザからアクセスする

# 1. Railsアプリをつくる

## \$ rails new sample\_app

```
$ rails new sample_app      [/Users/igarashi/work/0630railssampl
create README.rdoc
create Rakefile
create config.ru
create .gitignore
create Gemfile
create app
create app/assets/images/rails.png
create app/assets/javascripts/application.js
create app/assets/stylesheets/application.css
create app/controllers/application_controller.rb
create app/helpers/application_helper.rb
create app/mailers
create app/models
create app/views/layouts/application.html.erb
create app/mailers/.gitkeep
create app/models/.gitkeep
create config
```

...略...

```
Using sass-rails (3.2.5)
Using sqlite3 (1.3.6)
Installing uglifier (1.2.6)
Your bundle is complete! Use `bundle show [gemname]` to see where a bundled gem
is installed.
```

ちょっと時間がかかります。  
途中で**Enter your password to  
install the bundled RubyGems to  
your system:**  
と言わいたらパソコンのパスワードを入力し  
てください。

**rails new アプリ名**  
**でRailsアプリを作ります。**

**Railsが必要なファイルを  
たくさんgenerateします。**  
**画面に表示されるのはgenerate  
されたファイル群です。**

**その後、必要なgemを自動でセッ  
トアップします。**

## 2. WebServerとRailsアプリを起動する

```
$ cd sample_app
```

```
$ bundle exec rails server
```

```
$ bundle exec rails server
=> Booting WEBrick
=> Rails 3.2.3 application starting in development on http://0.0.0.0:3000
=> Call with -d to detach
=> Ctrl-C to shutdown server
[2012-06-30 21:32:34] INFO  WEBrick 1.3.1
[2012-06-30 21:32:34] INFO  ruby 1.9.3 (2012-04-20) [x86_64-darwin11.3.0]
[2012-06-30 21:32:34] INFO  WEBrick::HTTPServer#start: pid=2056 port=3000
```

**WebServerとRailsアプリを起動して、ブラウザからアクセス可能にします。**

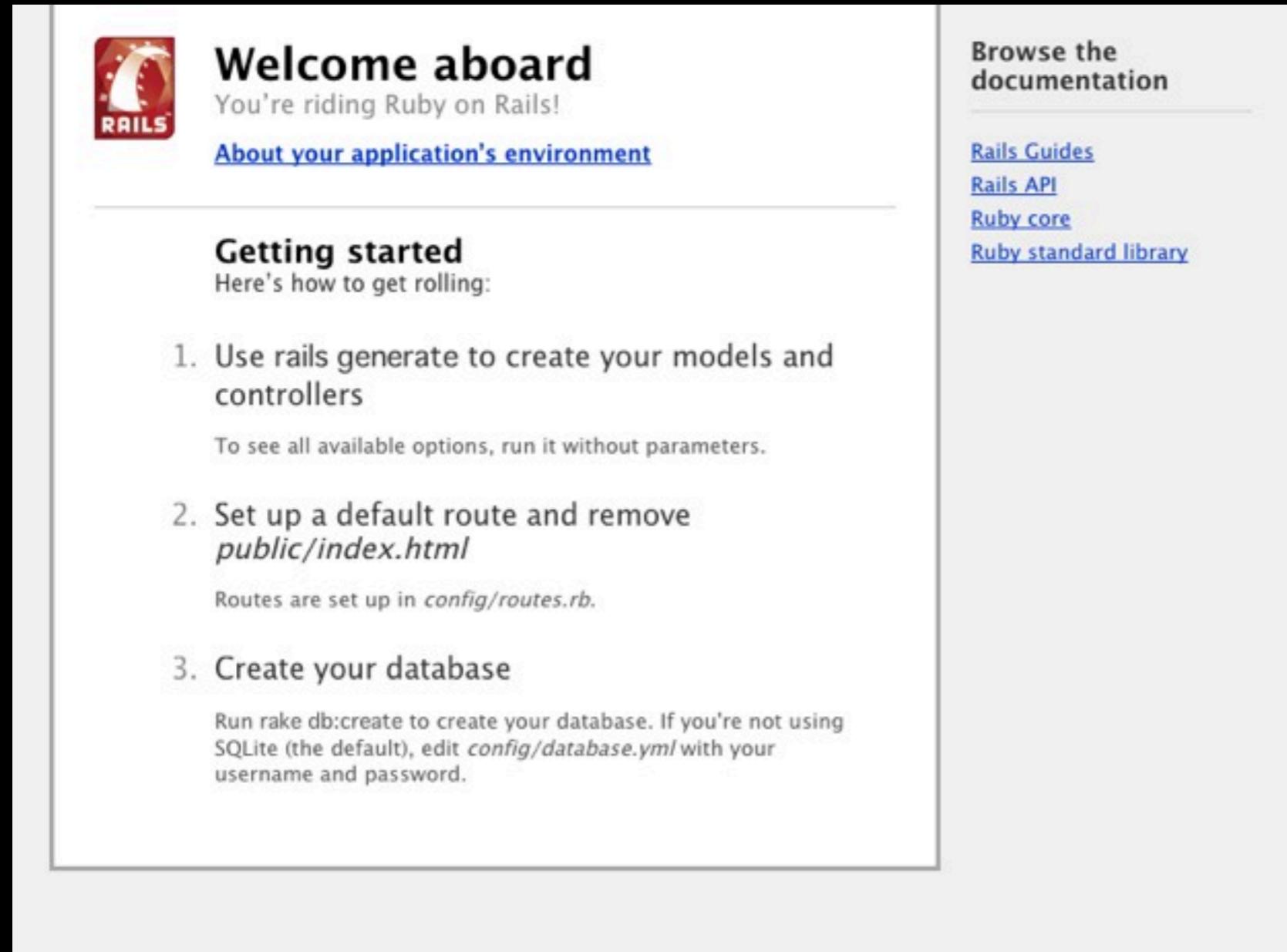
**起動中はshellからコマンドなどは入力できないので、必要な場合は別のshell画面を開きます。**

**終了は Ctrl キー と c キー を同時押しします。**

**(Windows RailsInstallerの場合はCtrlキー と Pauseキー)**

### 3. ブラウザから以下のURLにアクセスする

**http://localhost:3000**



The screenshot shows the initial 'Welcome aboard' screen of a Ruby on Rails application. On the left, there's a red 'RAILS' logo with a white train icon. To its right, the text 'Welcome aboard' is displayed in bold, followed by the subtext 'You're riding Ruby on Rails!'. Below this, a blue link reads 'About your application's environment'. A horizontal line separates this from the 'Getting started' section. Under 'Getting started', the text 'Here's how to get rolling:' is followed by three numbered steps: 1. 'Use rails generate to create your models and controllers', with a note below it stating 'To see all available options, run it without parameters.'; 2. 'Set up a default route and remove public/index.html', with a note below it stating 'Routes are set up in config/routes.rb.'; and 3. 'Create your database', with a note below it stating 'Run rake db:create to create your database. If you're not using SQLite (the default), edit config/database.yml with your username and password.' On the right side of the screen, there's a sidebar titled 'Browse the documentation' which includes links to 'Rails Guides', 'Rails API', 'Ruby core', and 'Ruby standard library'.

こんな画面が出ればRailsアプリへアクセス成功です。  
開発するための1歩目が踏み出せていることになります。

# URLの説明

**http://localhost:3000**

アドレス ポート

ポートは、同じアドレス内で通信先を特定する番号です。  
例えば、アドレスが駅の名前、ポートが駅のホーム〇番線、  
とイメージすると分かりやすいかも。

**localhost :**

特別なアドレスです。自分のマシンを指します。

**3000 :**

ポート番号です。3000番ポートを指します。

# さっきのアクセスのイメージ図

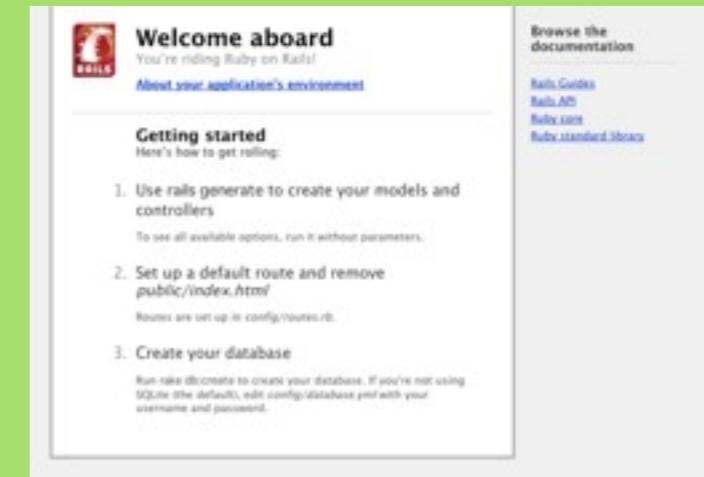
自分のPC  
リクエスト  
<http://localhost:3000>

Web Server  
localhost:3000  
\$ bundle exec rails server

Web App  
Rails App  
sample\_app

←  
レスポンス  
HTML

Browser



# 前につくったRailsアプリを動かしてみる

## 2回目以降に起動する場合の手順

### 1. Rails Root フォルダへ移動

sample\_appフォルダへ移動しておきます。

現在のフォルダは `pwd` コマンドで調べることができます。

sample\_appフォルダをRails Root と呼びます。

### 2. bundle install

```
$ bundle intall
```

Railsアプリで使うgemをセットアップしてくれるおまじない。

ほかのRailsアプリを動かした後などはこの命令が必要です。

### 3. WebServerとRailsアプリを起動する

```
$ bundle exec rails server
```

### 4. ブラウザからアクセスする

```
http://localhost:3000/books
```

# Railsはたくさんのファイルを自動生成します

```
$ rails new sample_app      [/Users/igarashi/work/0701railsample
create README.rdoc
create Rakefile
create config.ru
create .gitignore
create Gemfile
create app
create app/assets/images/rails.png
create app/assets/javascripts/application.js
create app/assets/stylesheets/application.css
create app/controllers/application_controller.rb
create app/helpers/application_helper.rb
create app/mailers
create app/models
create app/views/layouts/application.html.erb
create app/mailers/.gitkeep
create app/models/.gitkeep
create config
create config/routes.rb
create config/application.rb
create config/environment.rb
create config/environments
create config/environments/development.rb
create config/environments/production.rb
create config/environments/test.rb
create config/initializers
create config/initializers/backtrace_silencers.rb
create config/initializers/inflections.rb
create config/initializers/mime_types.rb
create config/initializers/secret_token.rb
create config/initializers/session_store.rb
create config/initializers/wrap_parameters.rb
create config/locales
create config/locales/en.yml
create config/boot.rb
create config/database.yml
create db
create db/seeds.rb
create doc
create lib/README_APP
create lib/tasks
create lib/tasks/.gitkeep
create lib/assets
create lib/assets/.gitkeep
create log
create log/.gitkeep
create log
create log/.gitkeep
create public
create public/404.html
create public/422.html
create public/500.html
create public/favicon.ico
create public/index.html
create public/robots.txt
create script
create script/rails
create test/fixtures
create test/fixtures/.gitkeep
create test/functional
create test/functional/.gitkeep
create test/integration
create test/integration/.gitkeep
create test/unit
create test/unit/.gitkeep
create test/performance/browsing_test.rb
create test/test_helper.rb
create tmp/cache
create tmp/cache/assets
create vendor/assets/javascripts
create vendor/assets/javascripts/.gitkeep
create vendor/assets/stylesheets
create vendor/assets/stylesheets/.gitkeep
create vendor/plugins
create vendor/plugins/.gitkeep
run bundle install
Fetching gem metadata from https://rubygems.org/.....
Using rake (0.9.2.2)
Using i18n (0.6.0)
Using multi_json (1.0.4)
Using activesupport (3.2.6)
Using builder (3.0.0)
Using railties (3.2.6)
Using erubis (2.7.0)
Using journey (1.0.4)
Using rack (1.4.1)
```

毎回使うような共通ファイルは自動で作られるので、  
ここにアプリ特有の処理を書いていきます

つづいて、  
簡単なページを作ってみます。

デモ用や開発のためのひな形を  
作るためのscaffoldという機  
能を利用します。

# 簡単なページをつくる

本のタイトルとメモを登録できるページをつくります。

sample\_appフォルダへ移動しておきます。  
このフォルダのことをRails Root と呼びます。

## 1. scaffoldでページをつくる

```
$ bundle exec rails generate scaffold book title:string memo:text
```

## 2. データをしまうデータベースをつくる

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

※ rails server をCtrl-cで停止していたら、また起動する

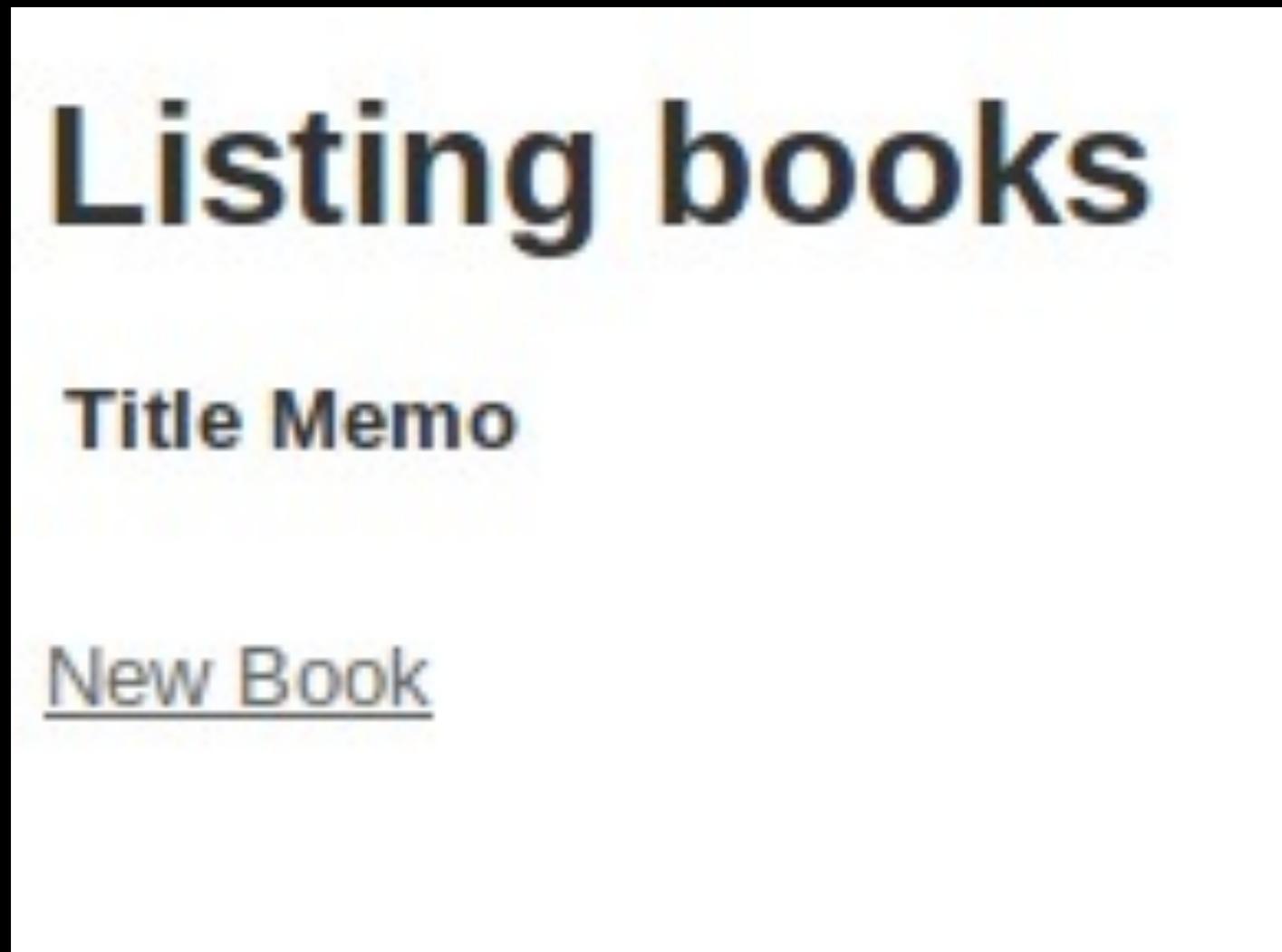
```
$ bundle exec rails server
```

## 3. ブラウザから以下のURLにアクセスする

```
http://localhost:3000/books
```

ブラウザから以下のURLにアクセスする

**http://localhost:3000/books**



こんな画面が出れば新しいページの作成に成功しています。

New Book リンクから登録したり編集したり削除したりしてみてください。

# scaffoldもたくさんのファイルを自動生成します。 これらのファイルを編集してアプリを作っていきます。

```
$ bundle exec rails generate scaffold book title:string memo:text
  invoke  active_record
  create    db/migrate/20120704140350_create_books.rb
  create    app/models/book.rb
  invoke  test_unit
  create    test/unit/book_test.rb
  create    test/fixtures/books.yml
  invoke  resource_route
  route    resources :books
  invoke  scaffold_controller
  create    app/controllers/books_controller.rb
  invoke  erb
  create    app/views/books
  create    app/views/books/index.html.erb
  create    app/views/books/edit.html.erb
  create    app/views/books/show.html.erb
  create    app/views/books/new.html.erb
  create    app/views/books/_form.html.erb
  invoke  test_unit
  create    test/functional/books_controller_test.rb
  invoke  helper
  create    app/helpers/books_helper.rb
  invoke  test_unit
  create    test/unit/helpers/books_helper_test.rb
  invoke  assets
  invoke  coffee
  create    app/assets/javascripts/books.js.coffee
  invoke  scss
  create    app/assets/stylesheets/books.css.scss
  invoke  scss
  create    app/assets/stylesheets/scaffolds.css.scss
```

# Railsアプリのフォルダ構成

`book_app`…アプリケーション名が付いたいたルートディレクトリ  
`app` …アプリケーションの主なコードの置き場  
  `assets`… アセット（注）の置き場  
    `images`… 画像ファイルの置き場  
    `javascripts`… JavaScript ファイルの置き場  
    `stylesheets`… スタイルシートの置き場  
`controllers`… コントローラクラスを格納  
`helpers`… ビュー用のヘルパークラスを格納  
`mailers`… メール用のコントローラを格納  
`models`… モデルクラスを格納  
`views`… ビューのテンプレートを格納  
  `layouts`… テンプレートの周囲のデザインを担当するレイアウトファイルを格納  
`config`… 設定ファイル類を格納  
`db`… データベースに関するファイルの置き場  
`doc`… `rdoc` などのドキュメントの置き場  
`lib`… アプリケーションが使うライブラリコード全般の置き場  
`log`… ログがこのディレクトリ以下に出力される  
`public`… Web の静的なコンテンツを配置  
`script`… ユーティリティスクリプトを格納  
`test`… Rails のデフォルトの自動テストに関するファイルを格納  
`tmp`… 一時ファイルの置き場  
`vendor`… アプリケーション外部に由来するコードの置き場  
`plugin`… Rails プラグインを格納

注) 画像、JavaScript ファイル、CSS ファイルなど、Web アプリケーションのコンテンツを作るために必要な付属的なファイルを Rails では「アセット」と言います。

**Railsでは役割ごとにフォルダが作られ、ファイルが配置されます。**

**どこにどんなファイルが置かれるかは次回以降の講義で説明していきます。**

# 復習：shellコマンド

Railsアプリ開発ではshellコマンドをよく使うので復習

カレントフォルダ（現在のフォルダ）を移動

```
$ cd フォルダ名
```

1つ上のフォルダへ移動

```
$ cd ..
```

カレントフォルダを表示

```
$ pwd
```

カレントフォルダのファイルを表示

```
$ ls
```

今週ここから

一番ちいさな  
Railsアプリづくり



<http://www.flickr.com/photos/naoyafujii/321375793/>

# 一番小さなRailsアプリつくり

## hello world を表示するアプリを作ります。

### 1. Railsアプリをつくる

```
$ rails new helloworld
```

### 2. RailsRootへ移動してページのひながたをつくる

```
$ cd helloworld
```

```
$ bundle exec rails g controller hello index
```

g は generate の略です。コマンドの意味は後半で説明します。  
打ち間違えて削除したい場合は、g の代わりに d で削除できます。

### 3. Webサーバを起動する

```
$ bundle exec rails s
```

※終了はCtrl+c(Windowsの場合は Ctrl+Pause)

s は server の略です。

### 4. ブラウザから以下のURLにアクセスする

<http://localhost:3000/hello/index>

# 一番小さなRailsアプリつくり

4. ブラウザから以下のURLにアクセスする

**http://localhost:3000/hello/index**

Hello#index

Find me in app/views/hello/index.html.erb

こんな画面が出ればOKです。

この表示を Hello world! に変えてみましょう。

# 一番小さなRailsアプリつくり

## 5. Hello world! をソースに書く

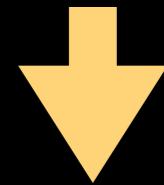
app/views/hello/index.html.erb

```
<h1>Hello#index</h1>
```

```
<p>Find me in app/views/hello/index.html.erb</p>
```

となっているところを以下のように修正

```
<h1>Hello world!</h1>
```



さきほど同様にブラウザで以下へアクセス

<http://localhost:3000/hello/index>

こんな画面が出ればOKです。

Hello world!

では、どのようにRailsアプリは  
動作しているのでしょうか。

リクエストに対して  
どのような動作をするか  
を説明していきます。

# リクエストに対する Railsアプリの動作

**Web Server**

**Web App**  
**Rails App**

リクエスト



**Browser**

URL入力欄に  
アドレスを入力して  
Enterを押すと  
リクエストが飛びます

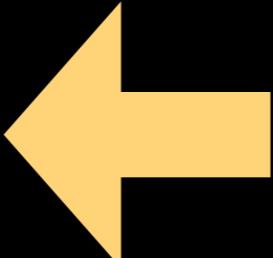
Railsアプリはリクエストに対して  
どのように動作しているのでしょうか

# リクエストについて もう少し詳しく説明します

Web Server

Web App  
Rails App

リクエスト



レスポンス  
HTML

Browser

URL入力欄に  
アドレスを入力して  
Enterを押すと  
リクエストが飛びます

Railsアプリはリクエストに対して  
どのように動作しているのでしょうか

# リクエスト URLとHTTPメソッド

リクエストは様々な情報から構成されますが、  
URLとHTTPメソッドがここでは重要な要素です

## リクエスト

- ▶ URL : `http://localhost:3000/hello/index`
- ▶ HTTPメソッド : GET

URLとHTTPメソッドについて説明していきます

## リクエスト

- ▶ URL : `http://localhost:3000/hello/index`
- ▶ HTTPメソッド : GET

## URL

URLはアドレスやポート、パスなどで構成される

URL : `http://localhost:3000/hello/index`

アドレス ポート パス

Railsアプリはパス(ここでは/hello/index)の情報を利用します。

## リクエスト

- ▶ URL : `http://localhost:3000/hello/index`
- ▶ HTTPメソッド : GET

## HTTPメソッド

Railsで扱うHTTPメソッドは下記の4種類

- ▶ GET : 取得
- ▶ POST : 作成、追加
- ▶ PUT : 変更
- ▶ DELETE : 削除

## リクエスト

- ▶ URL : `http://localhost:3000/hello/index`
- ▶ HTTPメソッド : GET

## URLとHTTPメソッド

Railsアプリにとって、  
上記のリクエストのざっくりした意味は、

パス : `/hello/index` を GET(取得) する  
になります。

# リクエストに対するRailsアプリの動作

## リクエスト

- ▶ URL : `http://localhost:3000/hello/index`
- ▶ HTTPメソッド : GET

## Web Server

### Rails App

Railsはリクエスト中の  
`/hello/index` と GET  
を使います。

## リクエスト



## レスポンス HTML

## Browser

URL入力欄に  
アドレスを入力して  
Enterを押すと  
リクエストが飛びます

# では、次はRailsアプリの中の処理を 追いかけてみましょう。

リクエスト

▶ URL : <http://localhost:3000/books>

▶ HTTPメソッド : GET

Web Server

Rails App

Railsはリクエスト中の  
`/hello/index` と GET  
を使います。

リクエスト

レスポンス  
HTML

Browser

# Railsアプリがリクエストを受けて レスポンスを返すまでの流れ

**Routes, Controller, View** の3つの部品を通過します。

## リクエスト

- ▶ URL : `http://localhost:3000/hello/index`
- ▶ HTTPメソッド : GET

## Rails App

**Routes**

**Controller**

**View**

レスポンス : HTML



# Railsアプリがリクエストを受けて レスポンスを返すまでの流れ

**Routes, Controller, View の3つの部品を通過します。**

リクエスト

- ▶ URL : `http://localhost:3000/hello/index`
- ▶ HTTPメソッド : GET

Rails App

**Routes**

Controller

View

レスポンス : HTML

# Routes

リクエストのURLとHTTPメソッドに応じて  
処理を行うコントローラとメソッドの対応表を作る仕事をします。

Rails Root フォルダで以下のコマンドを打つと対応表を表示します。

```
$ bundle exec rake routes
```

実行結果

**hello\_index GET /hello/index(.:format) hello#index**

HTTP  
メソッド

パス

処理を行う先  
コントローラ名#アクション名

ざっくりした意味

/hello/index へHTTPメソッド：GETでアクセスされた場合、

HelloControllerの indexメソッドを呼び出す

※routesからアクセスされるコントローラのメソッドのことを  
「アクション」と呼びます。

# Routes

対応表はどうやって作られているのでしょうか。  
→ソースファイルが config/routes.rb にあります。

config/routes.rb

```
Helloworld::Application.routes.draw do
```

```
  get "hello/index"
```

```
  # いろいろ書いてありますが、全てコメント文
```

```
end
```

get "hello/index" 文が対応表を作っている文です。

hello/index が GET されたらHelloControllerのindexアクションへ  
という意味です。コントローラ名とアクション名は書いてありません。  
書かれていない場合はデフォルトで / の前(hello)がコントローラ名、  
/ の後(index)がアクション名になります。

```
$ bundle exec rake routes
```

```
hello_index  GET  /hello/index(.:format)  hello#index
```

# Railsアプリがリクエストを受けて レスポンスを返すまでの流れ

**Routes, Controller, View の3つの部品を通過します。**

リクエスト

- ▶ URL : `http://localhost:3000/hello/index`
- ▶ HTTPメソッド : GET

Rails App

Routes

Controller

View

レスポンス : HTML

# Controller

さまざまな処理を行い、つぎの View に処理を渡します。

routes から、  
HelloController の index アクションが呼ばれることが分かりました。  
HelloController の ソースは  
app/controllers/hello\_controller.rb です。  
app/controllers/hello\_controller.rb

```
class HelloController < ApplicationController
  def index
  end
end
```

コントローラは ApplicationController を継承したクラスです。  
HelloController の index アクションには特に処理が書かれていません。  
その場合、Viewは app/views/コントローラ名/アクション名 のファイルを  
デフォルトで使うルールになっています。  
次はView(app/views/hello/index.html.erb)に処理が渡ります。

# Railsアプリがリクエストを受けて レスポンスを返すまでの流れ

**Routes, Controller, View の3つの部品を通過します。**

リクエスト

- ▶ URL : `http://localhost:3000/hello/index`
- ▶ HTTPメソッド : GET

Rails App

Routes

Controller

View

レスポンス : HTML

# View

ユーザーの目に届く部分（ここではHTML）をつくります。

コントローラの指示で、

**app/views/hello/index.html.erb**  
が使われます。

**app/views/hello/index.html.erb**

```
<h1>Hello#index</h1>
```

```
<p>Find me in app/views/hello/index.html.erb</p>
```

**index.html.erb** はHTMLのテンプレートファイルです。

Rubyのコードを実行した結果を埋め込むことができます。

(generateされた現在のファイルは普通のHTMLファイルで、

Rubyのコードは埋め込まれていません。)

埋め込まれたRubyのコードを実行した結果でHTMLを作ります。

Viewテンプレートファイルから作られたHTMLに

Railsがその他の加工を加えてレスポンスとして送出します。

# Railsアプリがリクエストを受けて レスポンスを返すまでの流れ

**Routes, Controller, View** の3つの部品を通過します。

## リクエスト

- ▶ URL : `http://localhost:3000/hello/index`
- ▶ HTTPメソッド : GET

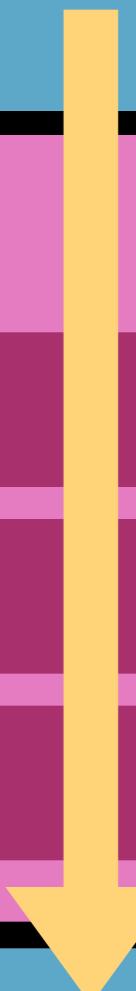
## Rails App

**Routes**

**Controller**

**View**

レスポンス : HTML



# リクエストに対するRailsアプリの動作

## リクエスト

- ▶ URL : `http://localhost:3000/hello/index`
- ▶ HTTPメソッド : GET

## Web Server

### Rails App

Routes

Controller

View

## リクエスト



## レスポンス HTML

## Browser

URL入力欄に  
アドレスを入力して  
Enterを押すと  
リクエストが飛びます

Hello#index

Find me in app/views/hello/index.html.erb

# ログ

コードがうまく動かない場合、  
ログに有用な情報が出力されていることが多いです。

**log/development.log**

ファイルにログが出力されます。

主な閲覧方法

- ▶ エディタで開く
- ▶ **\$ less log/development.log**
- ▶ **\$ tail -f log/development.log**

特にtail -f は自動でログを次々に表示してくれるので便利です。

# ログ

**log/development.log**

ActionView::Template::Error (undefined local  
variable or method `foo' for #<#<Class:  
0x007f83b9bb5398>:0x007f83b9bba898>):

1: <h1><%= foo %></h1>

app/views/hello/index.html.erb:1:in  
`\_app\_views\_hello\_index\_html\_erb\_\_313727477944  
0348102\_70101864340280'

ログの見方にはコツがあります。

たとえば、Error という文字列を探してみると問題を  
みつけやすいです。その後にファイル名と行数（下線  
部）が表示されています。

# ログ

## NameError in Hello#index

Showing /Users/igarashi/work/0701railsample/helloworld/app/views/hello/index.html.erb where line #1 raised:

```
undefined local variable or method `foo' for #<#<Class:0x007f83b9bb5398>:0x007f83b9bba898>
```

Extracted source (around line #1):

```
1: <h1><%= foo %></h1>
```

Rails.root: /Users/igarashi/work/0701railsample/helloworld

[Application Trace](#) | [Framework Trace](#) | [Full Trace](#)

```
app/views/hello/index.html.erb:1:in `__app_views_hello_index_html_erb____3137274779440348102_70101864340280'
```

また、アクセスしたブラウザの画面にもエラーログが出るので、これも参考になります。下線部がエラーのあるファイル名と行数(#1 = 1行目の意味)です。

# **rails g コマンド**

**ひながたになるファイルを生成するコマンド**

**アプリをつくる時に打ったrails g (generate)コマンドはファイルを生成します。**

```
$ bundle exec rails g controller hello index
```

generateする種類	アクション名
	コントローラ名

※打ち間違えて削除したい場合は、g の代わりに d で削除できます。

**routes、コントローラ、ビューのファイルほかを追加します。**

**config/routes.rb (get "foo/bar" の行を追加)**

**app/controllers/foo\_controller.rb**

**app/views/foo/bar.html.erb**

**できあがったファイルにコードを追記していくと楽です**

※先週やった scaffold もgenerateの種類の1つです。

では、もうすこし  
アプリをいじってみましょう。

# ViewにRubyのコードを埋め込んでみる

ViewではRubyのコードを実行し、結果を埋め込むことができます。  
コードを埋め込む場合は `<%=` と `%>` で囲みます。

`<%=` ここにRubyのコードを書くと実行結果をHTMLへ埋め込む `%>`

以下のファイルを変更して、ブラウザからアクセスしてみてください。

`app/views/hello/hello/index.html.erb`

`<h1>Hello world!</h1>`

`<p>現在時刻 <%= Time.now %> </p>`

`http://localhost:3000/hello/index`

Hello world!

現在時刻 2012-07-10 21:48:56 +0900

← `Time.now` の実行結果

# 次は、Controllerでコードを実行して、結果をViewへ渡してみましょう。

## リクエスト

- ▶ URL : `http://localhost:3000/hello/index`
- ▶ HTTPメソッド : GET

## Rails App

### Routes

Controller ここでコードを実行する

結果を渡すにはどうすればいいでしょう？

### View

ここで結果を表示する



レスポンス : HTML

# ControllerからViewへ変数を渡す

現在のコントローラのコードは何もしていません。  
ここに処理を書いて、Viewまで結果を渡してみましょう。  
Viewへ変数を渡すときは、インスタンス変数(@始まりの変数)を使います。  
以下の2ファイルを変更してブラウザからアクセスしてみてください。

**app/controllers/hello\_controller.rb**

```
class HelloController < ApplicationController
  def index
    @text = "viva summer vacation!"
  end
end
```

**app/views/hello/hello/index.html.erb**

```
<h1><%= @text %></h1>
```

**http://localhost:3000/hello/index**

viva summer vacation! ← @text の中身が表示される

# Controllerでコードを実行して、結果をViewへ渡すときはインスタンス変数を使う

## リクエスト

- ▶ URL : `http://localhost:3000/hello/index`
- ▶ HTTPメソッド : GET

## Rails App

### Routes

Controller ここでコードを実行する

結果を渡すにはインスタンス変数(@foo)を使う



### View

ここで結果を表示する



レスポンス : HTML



# 講義資料置き場

講義資料置き場をつくりました。  
過去の資料がDLできます。

<https://github.com/hitotsubashi-ruby/lecture2012>

or

<http://bit.ly/ruby-lecture>

# 雑談・質問用facebookグループ

facebookグループを作りました

<https://www.facebook.com/groups/hitotsubashi.rb>

- ・加入/非加入は自由です
- ・加入/非加入は成績に関係しません
- ・参加者一覧は公開されます
- ・書き込みは参加者のみ見えます
- ・希望者はアクセスして参加申請してください
- ・雑談、質問、議論など何でも気にせずどうぞ～
- ・質問に答えられる人は答えてあげてください
- ・講師陣もお答えします
- ・入ったら軽く自己紹介おねがいします