

# Ruby on Rails 講義 第21回 new, create

五十嵐邦明

twitter : igaiga555

<http://www.facebook.com/kuniaki.igarashi>



2013.11.14 at 一橋大学  
ニフティ株式会社寄附講義  
社会科学における情報技術と  
コンテンツ作成IV



# 五十嵐邦明

講師

## 株式会社 spice life

twitter: igaiga555

<https://github.com/igaiga/>

<http://www.facebook.com/kuniaki.igarashi>

# 濱崎 健吾

Teaching Assistant  
fluxflex, inc(米国法人)



twitter: hmsk

<https://github.com/hmsk/>

<http://www.facebook.com/hamachang>

# 休講連絡

11月21日の講義は  
休講です。

# おしらせ

今日の講義はChromeを使う  
ので、インストールしておいて  
ください。

復習

# indexアクション 一覧画面

GET /books

## Listing books

Title	Memo	
ドラえもん 猫型ロボットとの冒険浪漫譚		<a href="#">Show</a> <a href="#">Edit</a> <a href="#">Destroy</a>
君に届け 北海道の高校を舞台にした青春群像		<a href="#">Show</a> <a href="#">Edit</a> <a href="#">Destroy</a>

[New Book](#)

登録されているBookのデータを一覧表示する画面です。

index画面が表示されるまでの流れを追いかけてみましょう。

# CRUD画面遷移図

**index**

/books

**GET**

**new**

/books/new

**GET**

削除

新規入力

**destroy**

/books/:id

**DELETE**

新規登録

**create**

/books

**POST**

編集

この図  
の見方

アクション名

url  
HTTPメソッド

画面あり

更新

アクション名

url  
HTTPメソッド

画面なし

redirect

**edit**

/books/:id/edit

**GET**

**show**

/books/:id

**GET**

**update**

/books/:id

**PUT**

redirect

redirect

画面あり

更新

画面なし

redirect

画面なし

更新

画面なし

# 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/books`
- ▶ HTTPメソッド : GET

## Rails App

Routes

routes.rb

Controller

books\_controller.rb indexアクション

View

books/index.html.erb

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メソッド : GET

## Rails App

Routes



**routes.rb**

Controller



**books\_controller.rb** indexアクション

View



**books/index.html.erb**

レスポンス : HTML

# Routes

Routesはリクエストと処理を行うコントローラの対応表です。  
以下のURLでRoutes対応表を見ることができます。  
<http://localhost:3000/rails/info/routes>

Helper	HTTP Verb	Path	Controller#Action
<u>Path / Url</u>		<u>Path Match</u>	
books_path	GET	/books(.:format)	books#index
	POST	/books(.:format)	books#create
new_book_path	GET	/books/new(.:format)	books#new
edit_book_path	GET	/books/:id/edit(.:format)	books#edit
book_path	GET	/books/:id(.:format)	books#show
	PATCH	/books/:id(.:format)	books#update
	PUT	/books/:id(.:format)	books#update
	DELETE	/books/:id(.:format)	books#destroy

下線部が今回関係する部分です。

「/books に GET でアクセスされたとき、  
BooksController の index アクションへ処理を流す」  
という意味になります。

# Routes 対応表が書かれたファイル config/routes.rb

```
Appname::Application.routes.draw do
  resources :books
end
```

routesは Appname::Application.routes.draw do から  
end の間に書きます。rake routesで表示される8行の対応表は  
resources :books の1行だけで記述可能です。  
(基本的なroutesであるため、短く書けるようになっています。)

Helper	HTTP Verb	Path	Controller#Action
<u>Path / Url</u>		<u>Path Match</u>	
books_path	GET	/books(.:format)	books#index
	POST	/books(.:format)	books#create
new_book_path	GET	/books/new(.:format)	books#new
edit_book_path	GET	/books/:id/edit(.:format)	books#edit
book_path	GET	/books/:id(.:format)	books#show
	PATCH	/books/:id(.:format)	books#update
	PUT	/books/:id(.:format)	books#update
	DELETE	/books/:id(.:format)	books#destroy

# 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メソッド : GET

## Rails App

Routes

routes.rb

Controller

books\_controller.rb indexアクション

View

books/index.html.erb

レスポンス : HTML

# Controller

## app/controllers/books\_controller.rb

```
class BooksController < ApplicationController
  def index
    @books = Book.all
  end
  ... #以下 new, edit, createなど7つのアクションが書かれている
end
```

indexアクションは **@books = Book.all** の1行です。  
**Book.all** でBookに関する全データが詰まった配列を取得して、  
@booksインスタンス変数へ代入します。

インスタンス変数(@はじまり)へ代入するのは、Viewへ表示のために渡したいからです。  
(インスタンス変数ではないと、indexアクションを抜けたところで消えてしまいます。)

次はViewに処理が移ります。

特に指定がない場合、app/views/コントローラ名/アクション名  
のViewファイルへ処理が移ります。  
(ここでは app/views/books/index.html.erb)

# 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メソッド : GET

## Rails App

Routes

routes.rb

Controller

books\_controller.rb indexアクション

View

books/index.html.erb

レスポンス : HTML

# View

app/views/books/index.html.erb

...

```
<% @books.each do |book| %> ←@booksはbookがいくつか  
<tr>入った配列  
  <td><%= book.title %></td> 初回のbook.title → "ドラえもん"  
  <td><%= book.memo %></td> 初回のbook.memo → "  
  <td><%= link_to 'Show', book %></td> ネコ型ロボットとの冒険浪漫譚"  
  <td><%= link_to 'Edit', edit_book_path(book)  
%></td>  
  <td><%= link_to 'Destroy', book, method: :delete,  
data: { confirm: 'Are you sure?' } %></td>  
</tr>  
<% end %>  
...
```

表示は@books.each do |book| ~ end で行っています。

@books内の全データ("ドラえもん", "君に届け")でブロックを繰り返し実行し、titleやmemoを表示したり、show画面やedit画面、削除ボタンのリンクを生成します。

## Listing books

Title

Memo

ドラえもん 猫型ロボットとの冒険浪漫譚

[Show](#) [Edit](#) [Destroy](#)

君に届け 北海道の高校を舞台にした青春群像

[Show](#) [Edit](#) [Destroy](#)

[New Book](#)

# 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/books`
- ▶ HTTPメソッド : GET

## Rails App

Routes

routes.rb

Controller

books\_controller.rb indexアクション

View

books/index.html.erb

レスポンス : HTML



# indexアクション 一覧画面

GET /books

## Listing books

Title	Memo	
ドラえもん 猫型ロボットとの冒険浪漫譚		<a href="#">Show</a> <a href="#">Edit</a> <a href="#">Destroy</a>
君に届け 北海道の高校を舞台にした青春群像		<a href="#">Show</a> <a href="#">Edit</a> <a href="#">Destroy</a>

[New Book](#)

以上の処理を経て一覧画面が表示されています。

**index**

**/books**

**GET**

**新規入力**

**new**

**/books/new**

**GET**

**削除**

**destroy**

**s/:id**

**DELETE**

**新規登録**

**create**

**/books**

**POST**

**編集**

**この図  
の見方**

**アクション名**

**url  
HTTPメソッド**

**画面あり**

**更新**

**アクション名**

**url  
HTTPメソッド**

**画面なし**

**redirect**

**redirect**

**edit**

**/books/:id/edit**

**GET**

**update**

**/books/:id**

**PUT**

**show**

**/books/:id**

**GET**

**いま説明した箇所**

**更新**

**edit**

**/books/:id/edit**

**GET**

**更新**

**画面なし**

**redirect**

**redirect**

# 今日の内容

## ここから

open



new & create

# 前準備

先週作った、本のタイトルとメモを管理できるアプリを題材に説明します。

```
$ rails new books_app
```

```
$ cd books_app
```

```
$ bundle exec rails g scaffold books  
title:string memo:text
```

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

```
$ bundle exec rails s
```

**<http://localhost:3000/books>**

ブラウザで上記のアドレスへアクセス

# CRUD画面遷移図

**index**  
**/books**

GET

削除 新規入力

**new**  
**/books/new**

GET

**destroy**  
**/books/:id**

DELETE

新規登録

**create**  
**/books**

POST

この図  
の見方

アクション名  
URL(パス)  
HTTPメソッド

アクション名  
URL(パス)  
HTTPメソッド

画面あり

画面なし

更新

redirect

**edit**

**/books/:id/edit**

GET

**update**  
**/books/:id**

PUT

**show**

**/books/:id**

GET

編集

redirect

redirect

scaffoldが作った  
これらの機能は  
Webアプリの基本となるため  
**CRUD（クラッド）**  
という名前がついてます。  
**Create, Read, Update, Destroy**  
の頭文字です。  
**(作成・表示・更新・削除)**

今日の内容はCRUDの  
C (Create、新規作成)  
についてです。

# CRUD画面遷移図

index

/books

GET

この部分です

削除

新規入力

redirect

destroy

/books/:id

DELETE

編集

この図  
の見方

アクション名  
url  
HTTPメソッド

画面あり

アクション名  
url  
HTTPメソッド

画面なし

edit

/books/:id/edit

GET

更新

update

/books/:id

PUT

new

/books/new

GET

新規登録

create

/books

POST

redirect

show

/books/:id

GET

# 新規作成画面

この画面でタイトルとメモを入力してCreate Book ボタンを押すと新規登録を行うことができます。

## New book

Title

Memo

Create Book

# ざっくりした流れ

ステップ1

**new** アクション (新規作成画面)

本のタイトルとメモを入力

Create Book ボタンを押す



ステップ2

**create** アクション (画面なし)

入力されたタイトルとメモで

本のデータを新規作成

**new**  
**/books/new**  
**GET**

**create**  
**/books**  
**POST**

# newアクション

GET

`/books/new`

最初に、新規作成画面が表示されるまでの処理の流れを見ていきましょう。

## New book

Title

Memo

Create Book

# new (新規入力)画面が表示されるまで

## リクエスト

- ▶ URL : `http://localhost:3000/books/new`
- ▶ HTTPメソッド : GET

## Rails App

Routes

routes.rb

Controller

books\_controller.rb newアクション

View

books/new.html.erb

レスポンス : HTML

# new (新規入力)画面が表示されるまで

リクエスト

▶ URL :

▶ HTTPメソッド : GET

## Rails App

Routes



routes.rb

Controller



books\_controller.rb newアクション

View



books/new.html.erb

レスポンス : HTML

# Routes

`http://localhost:3000/rails/info/routes`

Helper <u>Path / Url</u>	HTTP Verb	Path	Controller#Action
books_path	GET	/books(.:format)	books#index
	POST	/books(.:format)	books#create
new_book_path	GET	/books/new(.:format)	books#new
edit_book_path	GET	/books/:id/edit(.:format)	books#edit
book_path	GET	/books/:id(.:format)	books#show
	PATCH	/books/:id(.:format)	books#update
	PUT	/books/:id(.:format)	books#update
	DELETE	/books/:id(.:format)	books#destroy

上記URLへアクセスするとRoutes一覧が見れます。

下線部が今回関係する部分です。

「/books/new に GET でアクセスされたとき、  
BooksController の new アクションを呼び出す」  
という意味になります。

# new (新規入力)画面が表示されるまで

リクエスト

▶ URL :

▶ HTTPメソッド : GET

## Rails App

Routes

routes.rb

Controller

books\_controller.rb newアクション

View

books/new.html.erb

レスポンス : HTML

# Controller

## app/controllers/books\_controller.rb

```
def new  
  @book = Book.new  
end
```

newアクションは `@books = Book.new` の1行です。  
`Book.new` でBookクラスのインスタンス（データは空っぽ）を作り、  
`@book`インスタンス変数へ代入します。  
(インスタンスは夏学期の鯛焼きの話を思い出してください。クラスは型、インスタンスは焼いた鯛焼き。)  
インスタンス変数へ代入するのはViewで表示の際に使うからです。  
Bookクラスのインスタンスはタイトルとメモを格納できるようになって  
います。  
※Bookクラスには色々と便利な機能が他にもあるのですが、それは次回説明します。

# new (新規入力)画面が表示されるまで

リクエスト

▶ URL :

▶ HTTPメソッド : GET

## Rails App

Routes

routes.rb

Controller

books\_controller.rb newアクション

View

books/new.html.erb

レスポンス : HTML

# View

## app/views/books/new.html.erb

```
<h1>New book</h1>

<%= render 'form' %>

<%= link_to 'Back', books_path %>
```

3行しかありません。随分とあっさりしています。

New book

Title

Memo

Create Book

Back



# render の説明

app/views/books/new.html.erb

```
<h1>New book</h1>
```

```
<%= render 'form' %>
```

```
<%= link_to 'Back', books_path %>
```

→の枠線の部分は別のファイルに書いてあり、

`<%= render 'form' %>` で埋め込まれます。

`render` メソッドは別のファイルを埋め込みます。

埋め込む用のview ファイルをパーシャルと言います。

```
<%= render 埋め込みたいファイル名 %>
```

埋め込むファイル名には1つルールがあり、

`render` で書いた文字列の先頭に `_` を付けたファイル名にします。

つまり、`<%= render 'form' %>` で埋め込まれるファイルは  
`_form.html.erb` になります。

※なぜわざわざ別のファイルに書いてあるかというと、他の画面でも同じ部品を共用したいからです。

New book

Title

Memo

Create Book

Back

## app/views/books/new.html.erb

```
<h1>New book</h1>
<%= render 'form' %> #=> _form.html.erb を埋め込み
<%= link_to 'Back', books_path %>
```

## app/views/books/\_form.html.erb (枠線部分)

```
<%= form_for(@book) do |f| %>
<% if @book.errors.any? %>
  <div id="error_explanation">
    <h2><%= pluralize(@book.errors.count, "error") %> prohibited this book
from being saved:</h2>
    <ul>
      <% @book.errors.full_messages.each do |msg| %>
        <li><%= msg %></li>
      <% end %>
    </ul>
  </div>
<% end %>
```

```
<div class="field">
  <%= f.label :title %><br>
  <%= f.text_field :title %>
</div>
<div class="field">
  <%= f.label :memo %><br>
  <%= f.text_area :memo %>
</div>
<div class="actions">
  <%= f.submit %>
</div>
<% end %>
```

## New book

The screenshot shows a web page titled 'New book'. The page has a pink border. At the top, it says 'New book'. Below that is a 'Title' input field. Underneath it is a 'Memo' text area. At the bottom right is a 'Create Book' button and a 'Back' link.

なので、  
←この2つの  
viewテンプレートから  
ページが作られてる  
わけです。

# new (新規入力)画面が表示されるまで

## リクエスト

- ▶ URL : `http://localhost:3000/books/new`
- ▶ HTTPメソッド : GET

## Rails App

### Routes

⇨ `routes.rb`

### Controller

⇨ `books_controller.rb` newアクション

### View

⇨ `books/new.html.erb`

## レスポンス : HTML

ここまででnew画面が表示されました。  
Viewについてもう少し詳しく説明します。

## app/views/books/new.html.erb

```
<h1>New book</h1>
<%= render 'form' %> #=> _form.html.erb を埋め込み
<%= link_to 'Back', books_path %>
```

## app/views/books/\_form.html.erb

```
<%= form_for(@book) do |f| %>
<% if @book.errors.any? %>
  <div id="error_explanation">
    <h2><%= pluralize(@book.errors.count, "error") %> prohibited this book
from being saved:</h2>
    <ul>
      <% @book.errors.full_messages.each do |msg| %>
        <li><%= msg %></li>
      <% end %>
    </ul>
  </div>
<% end %>
```

この枠内部分はエラー時の表示です

```
<div class="field">
  <%= f.label :title %><br />
  <%= f.text_field :title %>
</div>
<div class="field">
  <%= f.label :memo %><br />
  <%= f.text_area :memo %>
</div>
<div class="actions">
  <%= f.submit %>
</div>
<% end %>
```

← 基本的な流れは残りの部分になります。

## New book

Title

Memo

Create Book

[Back](#)

## app/views/books/new.html.erb

```
<h1>New book</h1>
<%= render 'form' %> #=> _form.html.erb を埋め込み
<%= link_to 'Back', books_path %>
```

## app/views/books/\_form.html.erb

```
<%= form_for(@book) do |f| %>
<% if @book.errors.any? %>
  <div id="error_explanation">
    <h2><%= pluralize(@book.errors.count, "error") %> prohibited this book
from being saved:</h2>
    <ul>
      <% @book.errors.full_messages.each do |msg| %>
        <li><%= msg %></li>
      <% end %>
    </ul>
  </div>
<% end %>
```

```
<div class="field">
  <%= f.label :title %><br />
  <%= f.text_field :title %>
</div>
<div class="field">
  <%= f.label :memo %><br />
  <%= f.text_area :memo %>
</div>
```

```
<div class="actions">
  <%= f.submit %>
</div>
<% end %>
```

←この色をつけた部分が基本部分です。  
ここを詳しく見てみましょう。

## New book

Title

Memo

[Create Book](#)

[Back](#)

```
<%= form_for(@book) do |f| %>
...
<div class="field">
  <%= f.label :title %><br />
  <%= f.text_field :title %>
</div>
<div class="field">
  <%= f.label :memo %><br />
  <%= f.text_area :memo %>
</div>
<div class="actions">
  <%= f.submit %>
</div>
<% end %>
```

The diagram illustrates the mapping of an ERB template to a web form. The template on the left is color-coded by component: pink for the title field, blue for the memo field, and green for the actions (submit). The resulting form on the right shows these components: a 'Title' input field, a 'Memo' text area, and a 'Create Book' button. Arrows connect the template code to their corresponding form elements.

前のページで色づけしてあった部分を抜き出しました。  
それぞれ矢印の先の部品を作っています。  
また、全体としては **form** という名の部品になっています。  
(formはHTMLでブラウザからサーバへ情報を送信する部品の1つ)  
まずは部品の1つ、タイトルのところを見てみましょう。

## app/views/books/\_form.html.erb

```
<%= form_for(@book) do |f| %>
```

```
<div class="field">
  <%= f.label :title %><br />
  <%= f.text_field :title %>
</div>
```

```
<div class="field">
  <%= f.label :memo %><br />
  <%= f.text_area :memo %>
</div>
<div class="actions">
  <%= f.submit %>
</div>
<% end %>
```

## New book

The screenshot shows a 'New book' form. At the top, it says 'New book'. Below that is a 'Title' input field with a placeholder 'Title'. Underneath it is a 'Memo' text area. At the bottom are two buttons: 'Create Book' and 'Back'.

枠線内がタイトルの部分です。

白色の部分がHTML、`<%= %>` で囲まれた黄色の部分がRailsコードの部分です。

`<div></div>`は中のHTML要素をグルーピングするための要素です。それだけだと特に見た目を変えませんが、CSSで修飾する要素を指定するためによく使います。  
(ここでは "field" というclass名をつけてCSSで修飾できるようにしています。)

`<br />` は改行を入れるHTML要素です。 次はRailsコードの部分を見てみます。

## app/views/books/\_form.html.erb

```
<%= form_for(@book) do |f| %>
```

```
<div class="field">
```

```
  <%= f.label :title %><br />
```

```
  <%= f.text_field :title %>
```

```
</div>
```

```
<div class="field">
  <%= f.label :memo %><br />
  <%= f.text_area :memo %>
</div>
<div class="actions">
  <%= f.submit %>
</div>
<% end %>
```

`f.label :title` で "Title" という文字列を表示しています。  
その名の通り、ラベルの部分です。

`f.text_field :title` はその下にあるテキスト入力欄です。

`f` は `form` ブロック内の変数で、ここでは `@book` に関する `form` を記述するために使っています。  
(そう書くものだと思ってもらえば・・・)

## New book

Title

Memo

Create Book

Back

## New book

### app/views/books/\_form.html.erb

```
<%= form_for(@book) do |f| %>
...
<div class="field">
  <%= f.label :title %><br />
  <%= f.text_field :title %>
</div>

<div class="field">
  <%= f.label :memo %><br />
  <%= f.text_area :memo %>
</div>

<div class="actions">
  <%= f.submit %>
</div>
<% end %>
```

The screenshot shows a web form titled "New book". It has two input fields: "Title" (empty) and "Memo" (empty). Below the "Memo" field is a large text area. At the bottom are "Create Book" and "Back" buttons.

メモの部分も同様です。

**f.label :memo** が "Memo"を表示する部分です。

**f.text\_area :memo** がその下のテキスト入力欄を作ります。

(**text\_area** は先ほどの **text\_field** よりも  
広くて改行を入力できるテキスト入力欄を作るメソッドです。)

## app/views/books/\_form.html.erb

```
<%= form_for(@book) do |f| %>  
...  
<div class="field">  
  <%= f.label :title %><br />  
  <%= f.text_field :title %>  
</div>  
<div class="field">  
  <%= f.label :memo %><br />  
  <%= f.text_area :memo %>  
</div>  
  
<div class="actions">  
  <%= f.submit %>  
</div>  
<% end %>
```

form

Title

Memo

[Back](#)

f.submit は投稿ボタン(Create Book ボタン)を作ります。  
このボタンを押すとform内の情報をまとめて  
サーバへ送信（リクエストを送信）します。

つまり・・・

## app/views/books/\_form.html.erb

```
<%= form_for(@book) do |f| %>  
...  


<%= f.label :title %><br />  
  <%= f.text_field :title %>  
</div>  


<%= f.label :memo %><br />  
  <%= f.text_area :memo %>  
</div>  


<%= f.submit %>  
</div>  
<% end %>


```

New book

Title

Memo

Create Book

Back

Create Book ボタンを押すと、form内の情報、ここでは Book に関する情報、入力したタイトルとメモを送信します。

では、具体的にどんなリクエストが飛ぶのか試してみましょう。

## New book

Title  
大東京トイボックス

←入力する

Memo  
ゲーム開発会社を舞台にした熱血ストーリー

←入力する

Create Book

Back

Elements Resources Network Sources Timeline Profiles Audits Console

Name Path Method Status Type Initiator Size Content Time Latency Timeline

メニューの  
表示 - 開発管理 - デベロッパーツール  
で開発用ツールを表示  
Networkタブを選択

No requests captured. Reload the page to see detailed information on the network activity.

Documents Stylesheets Images Scripts XHR Fonts WebSockets Other

Chromeのデベロッパーツールを使うと、どんなリクエストがサーバへ送信されたかを見ることができます。

タイトル欄とメモ欄にBookの情報を入力します。

Chromeのメニューからデベロッパーツールを起動します。

Networkと書かれたタブを選択します。

つづく

Book was successfully created.

Title: 大東京トイボックス

Memo: ゲーム開発会社を舞台にした熱血ストーリー

[Edit](#) | [Back](#)

たくさん表示されました。  
一番最初の books と書かれた行が先ほどボタンを押して発行されたリクエストです。

books の行をクリックして詳細を見てみましょう。

Name Path	Method	Status Text	Type	Initiator	Size Content	Time Latency	Timeline
books	POST	302 Found	text/html	Other	909B 0B	18ms 18ms	
/books	GET	200 OK	text/html	<a href="http://localhost:3">http://localhost:3</a>	1.86KB 1.03KB	13ms 13ms	
application.css /assets	GET	200 OK	text/css	3:5 Parser	(from c...)	0ms 0ms	
books.css /assets	GET	200 OK	text/css	3:6 Parser	(from c...)	0ms 0ms	
scaffolds.css /assets	GET	200	text/css	3:7 <a href="http://localhost:3000/assets/books.css?body=1">http://localhost:3000/assets/books.css?body=1</a>	(from c...)	0ms 0ms	
jquery.js /assets	GET	200 OK	application/javascript	3:8 Parser	(from c...)	0ms 0ms	
jquery_ujs.js /assets	GET	200 OK	application/javascript	3:9 Parser	(from c...)	0ms 0ms	
books.js /assets	GET	200 OK	application/javascript	3:10 Parser	(from c...)	0ms 0ms	
application.js /assets	GET	200 OK	application/javascript	3:11 Parser	(from c...)	0ms 0ms	
nudge-icon-arrow-up.png piocplcdbaefihamjohnefbikjl	GET	200 OK	image/...	chrome-extension Script	(from c...)	0ms 0ms	

Elements Resources Network Sources Timeline Profiles Audits Console

Name Path

- books
- 3 /books
- application.css /assets
- books.css /assets
- scaffolds.css /assets
- jquery.js /assets
- jquery\_ujs.js /assets
- books.js /assets
- application.js /assets
- nudge-icon-arrow-up.png pioclipoplcbaefihamjohnnefbikjilc

Head... Previ... Respon... Cooki... Timi...

Request URL: http://localhost:3000/books  
Request Method: POST  
Status Code: 302 Found

Request Headers view source

```
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Charset: UTF-8,*;q=0.5
Accept-Encoding: gzip,deflate,sdch
Accept-Language: ja,en-US;q=0.8,en;q=0.6
Cache-Control: max-age=0
Connection: keep-alive
Content-Length: 393
Content-Type: application/x-www-form-urlencoded
Cookie: _books_app_session=BAh7B0kiD3Nlc3Npb25faWQG0gZFRkkiJTA4YwNkM2Q2ZW0NmEwNmE4ZjQ0MjQ1NjE5Mjhi0DRkBjsAVEkiEF9jc3JmX3Rva2VuBjsARkkiMVhnTW9ad3h5U0ZVZ3hEdEJLcXEwVlpWN1Fwbm1rZFRwRk1URUxWbzIxh0U9BjsARg%3D%3D--afcd8885fc9a4cdea6b6ae3200b6fc239c8460d8
Host: localhost:3000
Origin: http://localhost:3000
Referer: http://localhost:3000/books/new
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_2) AppleWebKit/537.11 (KHTML, like Gecko) Chrome/23.0.1271.64 Safari/537.11
```

Form Data view URL encoded

Documents Stylesheets Images Scripts XHR Fonts WebSockets Other

← URLとHTTPメソッド

← URLとHTTPメソッド

Elements Resources Network Sources Timeline Profiles Audits Console

Name Path

- books
- 3 /books
- application.css /assets
- books.css /assets
- scaffolds.css /assets
- jquery.js /assets
- jquery\_ujs.js /assets
- books.js /assets
- application.js /assets
- nudge-icon-arrow-up.png pioclipoplcbaefihamjohnnefbikjilc

Head... Previ... Respon... Cooki... Timi...

Host: localhost:3000  
Origin: http://localhost:3000  
Referer: http://localhost:3000/books/new  
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_8\_2) AppleWebKit/537.11 (KHTML, like Gecko) Chrome/23.0.1271.64 Safari/537.11

Form Data view URL encoded

utf8: ✓  
authenticity\_token: XgMoZwxySFUgxDtBKqq0VZV7QpnmkdTpFMTELVo21oE=  
book[title]: 大東京トイボックス  
book[memo]: ゲーム開発会社を舞台にした熱血ストーリー  
commit: Create Book

Response Headers view source

```
Cache-Control: no-cache
Connection: Keep-Alive
Content-Length: 95
Content-Type: text/html; charset=utf-8
Date: Sat, 24 Nov 2012 02:00:48 GMT
Location: http://localhost:3000/books/3
Server: WEBrick/1.3.1 (Ruby/1.9.3/2012-04-20)
X-Content-Type-Options: nosniff
```

Documents Stylesheets Images Scripts XHR Fonts WebSockets Other

↓ 下の方へスクロール

← Form Data という欄に

← book[title]とbook[memo]の情報があることが分かります。

→ 次は、飛んだこのリクエストがどのように処理されるかを見ていきましょう。

# Create Book ボタンを押したときのリクエスト

ボタンを押すと、以下のリクエストがRailsアプリへ飛ぶことが分かりました。  
次は、Railsアプリがこのリクエストをどう処理するのかを見ていきましょう。

## リクエスト

- ▶ URL : `http://localhost:3000/books`
- ▶ HTTPメソッド : POST
- ▶ パラメータ(Form data) :
  - ▶ `book[title]` : 大東京トイボックス
  - ▶ `book[memo]` : ゲーム開発会社を舞台にした熱血ストーリー

## Rails App

Routes  routes.rb

Controller  ?

View  ?

レスポンス : HTML



# Routes

## リクエスト

- ▶ URL : **http://localhost:3000/books**
- ▶ HTTPメソッド : POST
- ▶ パラメータ(Form data) :
  - ▶ book[title] : 大東京トイボックス
  - ▶ book[memo] : ゲーム開発会社を舞台にした熱血ストーリー

HTTP メソッド	URL	コントローラ#アクション
GET	/books(.:format)	books#index
POST	/books(.:format)	books#create
GET	/books/new(.:format)	books#new
GET	/books/:id/edit(.:format)	books#edit
GET	/books/:id(.:format)	books#show
PATCH	/books/:id(.:format)	books#update
PUT	/books/:id(.:format)	books#update
DELETE	/books/:id(.:format)	books#destroy

/books へPOSTなので、booksコントローラのcreateアクションが呼び出されます。

# Create Book ボタンを押したときのリクエスト

次はbooks\_controller.rb の create アクションを見てみます。

## リクエスト

- ▶ URL : <http://localhost:3000/books>
- ▶ HTTPメソッド : POST
- ▶ パラメータ(Form data) :
  - ▶ book[title] : 大東京トイボックス
  - ▶ book[memo] : ゲーム開発会社を舞台にした熱血ストーリー

## Rails App

Routes

routes.rb

Controller

books\_controller.rb createアクション

View

?

レスポンス : HTML



## リクエスト

- ▶ URL : **http://localhost:3000/books**
- ▶ HTTPメソッド : POST
- ▶ パラメータ(Form data) :
  - ▶ book[title] : 大東京トイボックス
  - ▶ book[memo] : ゲーム開発会社を舞台にした熱血ストーリー

## app/controllers/books\_controller.rb

```
def create
  @book = Book.new(book_params)

  respond_to do |format|
    if @book.save
      format.html { redirect_to @book, notice: 'Book was successfully created.' }
      format.json { render action: 'show', status: :created, location: @book }
    else
      format.html { render action: 'new' }
      format.json { render json: @book.errors, status: :unprocessable_entity }
    end
  end
end
```

createアクションへ処理が進みます。ここでやっていることは大きく3つあります。

## リクエスト

- ▶ URL : **http://localhost:3000/books**
- ▶ HTTPメソッド : POST
- ▶ パラメータ(Form data) :
  - ▶ book[title] : 大東京トイボックス
  - ▶ book[memo] : ゲーム開発会社を舞台にした熱血ストーリー

## app/controllers/books\_controller.rb

```
def create
  @book = Book.new(book_params)

  respond_to do |format|
    if @book.save
      format.html { redirect_to @book, notice: 'Book was successfully created.' }
      format.json { render action: 'show', status: :created, location: @book }
    else
      format.html { render action: 'new' }
      format.json { render json: @book.errors, status: :unprocessable_entity }
    end
  end
end
```

1. リクエストのパラメータを使って  
本のデータを作る

2. 本のデータを保存する

3a. 成功したらshow画面へ

3b. 保存失敗したらnew画面へ(さっきの画面)

では、3つの処理を順番に見ていきます。 55

# 1. リクエストのパラメータを使って本のデータを作る

## リクエスト

- ▶ URL : **http://localhost:3000/books**
- ▶ HTTPメソッド : POST
- ▶ パラメータ(Form data) :
  - ▶ book[title] : 大東京トイボックス
  - ▶ book[memo] : ゲーム開発会社を舞台にした熱血ストーリー

## app/controllers/books\_controller.rb

```
def create
  @book = Book.new(book_params) ←この行はどんな意味でしょうか

  respond_to do |format|
    if @book.save
      format.html { redirect_to @book, notice: 'Book was successfully created.' }
      format.json { render action: 'show', status: :created, location: @book }
    else
      format.html { render action: 'new' }
      format.json { render json: @book.errors, status: :unprocessable_entity }
    end
  end
end
```

# 1. リクエストのパラメータを使って本のデータを作る

## リクエスト

- ▶ URL : **http://localhost:3000/books**
- ▶ HTTPメソッド : POST
- ▶ パラメータ(Form data) :
  - ▶ book[title] : 大東京トイボックス
  - ▶ book[memo] : ゲーム開発会社を舞台にした熱血ストーリー

## app/controllers/books\_controller.rb

```
def create
  @book = Book.new(book_params)
...
end
          引数の book_paramsはメソッド呼び出しだけです。
          ファイルの後半に books_params メソッドがあります。

private
...
  def book_params
    params.require(:book).permit(:title, :memo)
  end
```

# パラメータ

app/controllers/books\_controller.rb

```
def book_params
```

```
  params.require(:book).permit(:title, :memo)
```

```
end
```

`book_params` メソッドはパラメータに関する処理を行っています。パラメータとは、ブラウザから飛んでくるリクエストの中に含まれる情報で、たとえばユーザーが入力した値が格納されています。

パラメータは `params` で取得できます。

次のページで`params`にどんな情報がどう入っているかを見てみましょう。

# パラメータの中身は？

パラメータを取得する `params` の中身を表示してみます。

`app/controllers/books_controller.rb`

```
def book_params
```

```
p "*****"
```

←試しにこの2行を追加して

```
p params
```

`params[:book]`の中身を表示して覗いてみましょう。

(最初の "\*\*\*\*\*" は見つけ易くする目印です。目立てばなんでもOK。)

```
params.require(:book).permit(:title, :memo)  
end
```

コードを変更して、ブラウザから新規登録画面で情報を入れて、  
`Create Book`のボタンを押します。  
その後、`rails s` のshellを流れた文字列から  
`*****` を探してみてください。

# パラメータの中身は？

app/controllers/books\_controller.rb

```
def book_params
```

```
  p "*****"
```

```
  p params
```

```
  params.require(:book).permit(:title, :memo)
```

```
end
```

実行結果を見ると、確かに paramsの中にブラウザで入力した値が入っていることが分かりました。(形式はHashですね。)

```
Started POST "/books" for 127.0.0.1 at 2013-09-23 06:47:19 +0900
Processing by BooksController#create as HTML
Parameters: {"utf8"=>"✓", "authenticity_token"=>"zGLyQpUxcQAlSCalAT50b0Nxr3oU7DUZgH57NWq7FmE=", "book"=>{"title"=>"大東京トイボックス", "memo"=>"ゲーム開発会社を舞台にした熱血ストーリー"}, "commit"=>"Create Book"}
"*****"
{"utf8"=>"✓", "authenticity_token"=>"zGLyQpUxcQAlSCalAT50b0Nxr3oU7DUZgH57NWq7FmE=", "book"=>{"title"=>"大東京トイボックス", "memo"=>"ゲーム開発会社を舞台にした熱血ストーリー"}, "commit"=>"Create Book", "action"=>"create", "controller"=>"books"}
(0.1ms) begin transaction
SQL (0.5ms) INSERT INTO "books" ("created_at", "memo", "title", "updated_at")
```

# パラメータの中身は？

ここで出力した `params` の値と、さきほどブラウザで表示させたパラメータの値が同じことが分かります。ブラウザがユーザーの入力データをパラメータとして送信し、私たちが作成しているアプリがそれを受け取っていることが確認できました。

ブラウザのデベロッパーツール

Elements Resources Network Sources Timeline Profiles Audits Console

Name  
Path

- books
- 3 /books
- application.css /assets
- books.css /assets
- scaffolds.css /assets
- jquery.js /assets
- jquery\_ujs.js /assets
- books.js /assets
- application.js /assets
- nudge-icon-arrow-up.png pioclpoplcdbaefihamjohnefbikjlci

Head... Previ... Respon... Cooki... Timi...

Host: localhost:3000  
Origin: http://localhost:3000  
Referer: http://localhost:3000/books/new  
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_8\_2) AppleWebKit/537.11 (KHTML, like Gecko) Chrome/23.0.1271.64 Safari/537.11

Form Data view URL encoded

utf8: ✓  
authenticity\_token: XgMoZwxySFUgxDtBKqq0VZV7QpnmkdTpFMTELVo21oE=  
book[title]: 大東京トイボックス  
book[memo]: ゲーム開発会社を舞台にした熱血ストーリー  
commit: Create Book

Response Headers view source

Cache-Control: no-cache  
Connection: Keep-Alive  
Content-Length: 95  
Content-Type: text/html; charset=utf-8  
Date: Sat, 24 Nov 2012 02:00:48 GMT  
Location: http://localhost:3000/books/3  
Server: WEBrick/1.3.1 (Ruby/1.9.3/2012-04-20)

Documents Stylesheets Images Scripts XHR Fonts WebSockets Other

コンソールのアプリログ

```
{"utf8"=>"✓", "authenticity_token"=>"zGLyQpUxcQAlSCalAT50b0Nxr3oU7DUZgH57NWq7FmI=", "book"=>{"title"=>"大東京トイボックス", "memo"=>"ゲーム開発会社を舞台にした熱血ストーリー"}, "commit"=>"Create Book", "action"=>"create", "controller"=>"books"}  
61
```

# パラメータの制限(Strong Parameters)

app/controllers/books\_controller.rb

```
def book_params
  params.require(:book).permit(:title, :memo)
end
```

params以降の文は、パラメータの内容を制限しています。意図した以外のデータが入ってくるのを防ぐためです。ここでは、book の title, memo だけを受け取るようにしています。

※なぜこれが必要かと言うと、ブラウザから飛ばすパラメータはユーザーの手によって改ざんすることも可能だからです（任意のパラメータを飛ばすことができる）。

# 1. リクエストのパラメータを使って本のデータを作る

## リクエスト

- ▶ URL : **http://localhost:3000/books**
- ▶ HTTPメソッド : POST
- ▶ パラメータ(Form data) :
  - ▶ book[title] : 大東京トイボックス
  - ▶ book[memo] : ゲーム開発会社を舞台にした熱血ストーリー

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

```
def create
  @book = Book.new(book_params)

  respond_to do |format|
    if @book.save
      format.html { redirect_to @book, notice: 'Book was successfully created.' }
      format.json { render action: 'show', status: :created, location: @book }
    else
      format.html { render action: 'new' }
      format.json { render json: @book.errors, status: :unprocessable_entity }
    end
  end
end
```

**book\_params**でパラメータの情報を取れることが分かりました。  
これを使って本のデータを作ります。

このデータは **Book.new** で作ります。  
**new**はクラスのインスタンスを作る

実はBookは「モデル」という種族に属する便利な機能を持ったクラスです。  
モデルについての説明はまた次回に・・・。

## リクエスト

- ▶ URL : **http://localhost:3000/books**
- ▶ HTTPメソッド : POST
- ▶ パラメータ(Form data) :
  - ▶ book[title] : 大東京トイボックス
  - ▶ book[memo] : ゲーム開発会社を舞台にした熱血ストーリー

## app/controllers/books\_controller.rb

```
def create
  @book = Book.new(book_params)

  respond_to do |format|
    if @book.save
      format.html { redirect_to @book, notice: 'Book was successfully created.' }
      format.json { render action: 'show', status: :created, location: @book }
    else
      format.html { render action: 'new' }
      format.json { render json: @book.errors, status: :unprocessable_entity }
    end
  end
end
```

1. リクエストのパラメータを使って  
本のデータを作る

2. 本のデータを保存する

3a. 成功したらshow画面へ

3b. 保存失敗したらnew画面へ(さっきの画面)

今日は1.の途中まで、 create アクションにパラメータ(params)が届いたのを確認したところまでやりました。

おさらい

今日の内容はCRUDの  
C (Create、新規作成)  
について説明しました。

# CRUD画面遷移図

index

/books

GET

この部分です

削除 新規入力

redirect

編集

この図  
の見方

アクション名

url  
HTTPメソッド

画面あり

更新

アクション名

url  
HTTPメソッド

画面なし

redirect

edit

/books/:id/edit

GET

update

/books/:id

PUT

new

/books/new

GET

新規登録

create

/books

POST

redirect

show

/books/:id

GET

新規作成画面は  
newアクションで  
作られて、

newアクション

GET

/books/new

新規作成画面

## New book

Title

Memo

Create Book

Back

タイトルとメモを入力して  
Create Book ボタンを  
押すと、

newアクション

GET

/books/new

新規作成画面

## New book

Title

大東京トイボックス

Memo

ゲーム開発会社を舞台にした熱血ストーリー

Create Book

←押す

Back

form内のデータを  
パラメータとして  
リクエストを飛ばします。

## New book

Title

大東京トイボックス

form

Memo

ゲーム開発会社を舞台にした熱血ストーリー

### リクエスト

- ▶ URL : <http://localhost:3000/books>
- ▶ HTTPメソッド : POST
- ▶ パラメータ(Form data) :
  - ▶ book[title] : 大東京トイボックス
  - ▶ book[memo] : ゲーム開発会社を舞台にした熱血ストーリー

/books/new

新規作成画面

Create Book

Back

# Create Book ボタンを押したときのリクエスト

## リクエスト

- ▶ URL : `http://localhost:3000/books`
- ▶ HTTPメソッド : POST
- ▶ パラメータ(Form data) :
  - ▶ `book[title]` : 大東京トイボックス
  - ▶ `book[memo]` : ゲーム開発会社を舞台にした熱血ストーリー

## Rails App

Routes	 routes.rb
Controller	 books_controller.rb createアクション
View	 ?

レスポンス : HTML

# Routes によってbooksコントローラのcreateアクションへ処理が流れ

## リクエスト

- ▶ URL : <http://localhost:3000/books>
- ▶ HTTPメソッド : POST
- ▶ パラメータ(Form data) :
  - ▶ book[title] : 大東京トイボックス
  - ▶ book[memo] : ゲーム開発会社を舞台にした熱血ストーリー

HTTP メソッド	URL	コントローラ#アクション
GET	/books(.:format)	books#index
POST	/books(.:format)	books#create
GET	/books/new(.:format)	books#new
GET	/books/:id/edit(.:format)	books#edit
GET	/books/:id(.:format)	books#show
PATCH	/books/:id(.:format)	books#update
PUT	/books/:id(.:format)	books#update
DELETE	/books/:id(.:format)	books#destroy

/books へPOSTなので、booksコントローラのcreateアクションが呼び出されます。

**books**コントローラの**create**アクション内で  
パラメータが届いていることを確認しました。

### リクエスト

- ▶ URL : **http://localhost:3000/books**
- ▶ HTTPメソッド : POST
- ▶ パラメータ(**Form data**) :
  - ▶ **book[title]** : 大東京トイボックス
  - ▶ **book[memo]** : ゲーム開発会社を舞台にした熱血ストーリー

### app/controllers/books\_controller.rb

```
def create
  @book = Book.new(book_params)
...
end

private
...
def book_params
  params.require(:book).permit(:title, :memo)
end
```



{:title => "大東京トイボックス",  
:memo => "ゲーム開発会社を舞台にした熱血ストーリー" }



# 付録

# デバッグの方法

- ▶ ブラウザに表示されるエラーを読む
- ▶ rails server の出力を確認
- ▶ ログを確認
- ▶ ログを出力
- ▶ pryを使う
- ▶ Chrome のデバッグ機能を使う

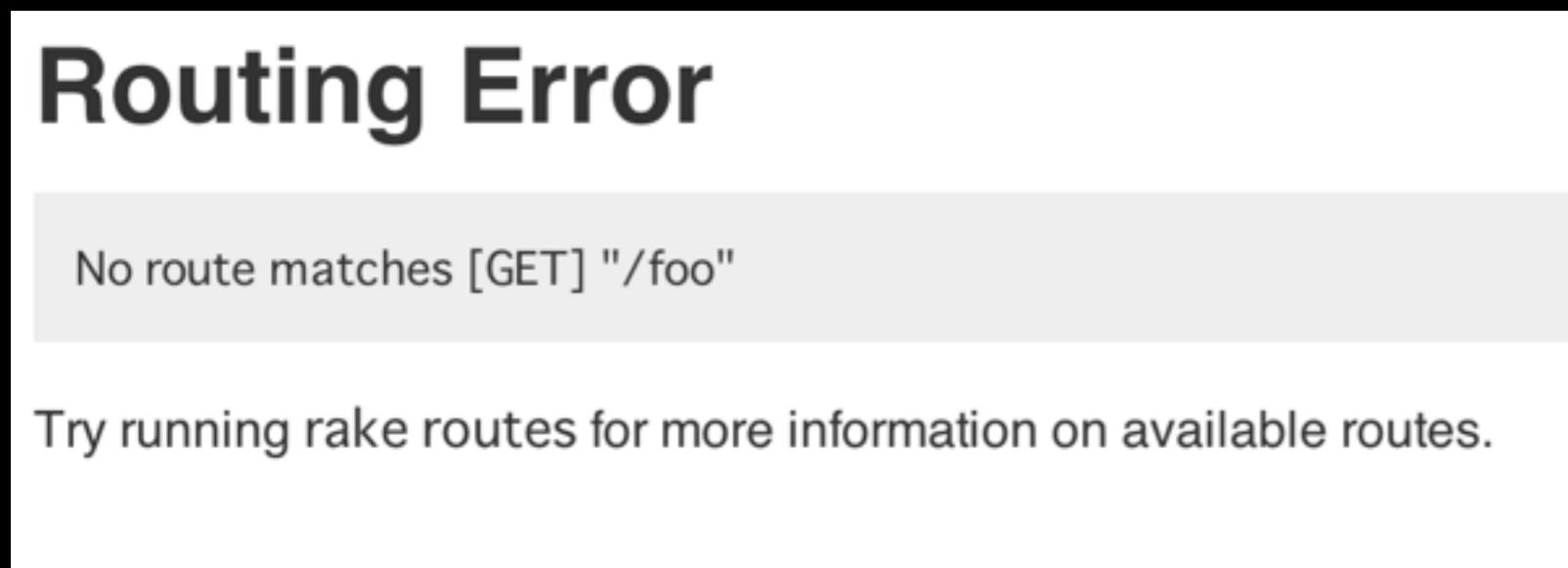
# ブラウザに表示されるエラーを読む

## エラー例：rails server が起動していない



## エラー例：そんなURL知らない

### Routing Error



# ブラウザに表示されるエラーを読む

## NameError in BooksController#index

undefined local variable or method `bar' for #<BooksController:0x007f94ddcccd28>

Rails.root: /Users/igarashi/work/scaffold1115/books\_app

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

app/controllers/books\_controller.rb:5:in `index'

### Request

#### Parameters:

None

[Show session dump](#)

[Show env dump](#)

### Response

#### Headers:

None

エラー例：

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

の5行目で bar っていう知らないものがあるのでエラーだよ。

# rails server の出力を確認

`rails s` を実行した shell を見ると、アクセスのたびに何かが  
出力されていることが分かります。

```
$ bundle exec rails s
=> Booting WEBrick
=> Rails 3.2.9 application starting in development on http://0.0.0.0:3000
=> Call with -d to detach
=> Ctrl-C to shutdown server
[2012-11-17 13:03:00] INFO  WEBrick 1.3.1
[2012-11-17 13:03:00] INFO  ruby 1.9.3 (2012-04-20) [x86_64-darwin11.3.0]
[2012-11-17 13:03:00] INFO  WEBrick::HTTPServer#start: pid=9602 port=3000
```

```
Started GET "/books" for 127.0.0.1 at 2012-11-17 13:03:05 +0900
Connecting to database specified by database.yml
Processing by BooksController#index as HTML
  Book Load (0.4ms)  SELECT "books".* FROM "books"
  Rendered books/index.html.erb within layouts/application (3.6ms)
Completed 200 OK in 76ms (Views: 37.8ms | ActiveRecord: 2.1ms)
```

```
Started GET "/assets/books.css?body=1" for 127.0.0.1 at 2012-11-17 13:03:06 +0900
Served asset /books.css - 304 Not Modified (1ms)
[2012-11-17 13:03:06] WARN Could not determine content-length of response body.
Set content-length of the response or set Response#chunked = true
```

```
Started GET "/assets/scaffolds.css?body=1" for 127.0.0.1 at 2012-11-17 13:03:06 +0900
```

# rails server の出力を確認

エラーが起こるとその旨も表示されます。

エラー例: /foo にアクセスしたけど、そんなURL知らない

```
Started GET "/foo" for 127.0.0.1 at 2012-11-17 13:16:44 +0900
```

```
ActionController::RoutingError (No route matches [GET] "/foo"):  
  actionpack (3.2.9) lib/action_dispatch/middleware/debug_exceptions.rb:21:in `call'  
  actionpack (3.2.9) lib/action_dispatch/middleware/show_exceptions.rb:56:in `call'  
  railties (3.2.9) lib/rails/rack/logger.rb:32:in `call_app'  
  railties (3.2.9) lib/rails/rack/logger.rb:16:in `block in call'  
  activesupport (3.2.9) lib/active_support/tagged_logging.rb:22:in `tagged'  
  railties (3.2.9) lib/rails/rack/logger.rb:16:in `call'  
  actionpack (3.2.9) lib/action_dispatch/middleware/request_id.rb:22:in `call'  
  rack (1.4.1) lib/rack/methodoverride.rb:21:in `call'  
  rack (1.4.1) lib/rack/runtime.rb:17:in `call'  
  activesupport (3.2.9) lib/active_support/cache/strategy/local_cache.rb:72:in `call'  
  rack (1.4.1) lib/rack/lock.rb:15:in `call'  
  actionpack (3.2.9) lib/action_dispatch/middleware/static.rb:62:in `call'  
  railties (3.2.9) lib/rails/engine.rb:479:in `call'  
  railties (3.2.9) lib/rails/application.rb:223:in `call'  
  rack (1.4.1) lib/rack/content_length.rb:14:in `call'  
  railties (3.2.9) lib/rails/rack/log_tailer.rb:17:in `call'  
  rack (1.4.1) lib/rack/handler/webrick.rb:59:in `service'  
  /Users/igarashi/.rvm/rubies/ruby-1.9.3-p194/lib/ruby/1.9.1/webrick/httpserver.  
rb:138:in `service'
```

# rails server の出力を確認

エラーが起こるとその旨も表示されます。

**app/controllers/books\_controller.rb の5行目で  
bar っていう知らないものがあるのでエラーだよ。**

```
Started GET "/books" for 127.0.0.1 at 2012-11-17 13:21:10 +0900
Connecting to database specified by database.yml
Processing by BooksController#index as HTML
Completed 500 Internal Server Error in 1ms

NameError (undefined local variable or method `bar' for #<BooksController:0x007f
940dccc28>):
  app/controllers/books_controller.rb:5:in `index'
```

```
Rendered /Users/igarashi/.rvm/gems/ruby-1.9.3-p194/gems/actionpack-3.2.9/lib/a
ction_dispatch/middleware/templates/rescues/_trace.erb (0.9ms)
Rendered /Users/igarashi/.rvm/gems/ruby-1.9.3-p194/gems/actionpack-3.2.9/lib/a
ction_dispatch/middleware/templates/rescues/_request_and_response.erb (0.7ms)
Rendered /Users/igarashi/.rvm/gems/ruby-1.9.3-p194/gems/actionpack-3.2.9/lib/a
ction_dispatch/middleware/templates/rescues/diagnostics.erb within rescues/layout (9.3ms)
```

# ログの確認

## log/development.log

上記のファイルにログが出力されます。内容は rails server が出力しているものと似ていますが、ログの方が詳しい情報が出ていることもあります。

エディタで開いて確認してもいいですが、ログが追記されるごとにリロードする必要があります。  
画面に次々と更新された内容を表示するには以下のコマンドが便利です。

```
$ tail -f log/development.log
```

※ tailはlinux, mac のみ。winではできません。

# ログを出力

ソースファイル中で変数の中身を表示したり、処理がどこまで進んだか知りたい場合にはログを出力するのが便利です。

ここでは2つの方法を説明します。

# shellに表示 p

Ruby の p メソッドがRailsでも使えます。

pメソッドは文字列を rails server の shell に表示します。

※表示だけでlog/development.logには出力しないので注意

```
class BooksController < ApplicationController
```

```
  def index
```

```
    @books = Book.all
```

```
    p '*****' ← 探し易いように目立つ文字を表示すると便利
```

```
    p @books ← p メソッドで@booksの中身を表示
```

```
...
```

```
  end
```

```
end
```

```
Started GET "/assets/application.js?body=1" for 127.0.0.1 at 2012-11-17 19:20:50  
+0900
```

```
Served asset /application.js - 304 Not Modified (25ms)
```

```
[2012-11-17 19:20:50] WARN Could not determine content-length of response body.  
Set content-length of the response or set Response#chunked = true
```

```
[2012-11-17 19:20:51] WARN Could not determine content-length of response body.  
Set content-length of the response or set Response#chunked = true
```

```
"*****"
```

```
[#<Book id: 1, title: "ドラえもん", memo: "猫型ロボットとの冒険浪漫譚", created_at: "2012-11-15 12:59:01", updated_at: "2012-11-15 12:59:01">]
```

# ログに表示 Rails.logger

**Rails.logger.error** を使うとlog/development.log へ変数の中身や文字列を出力できます。  
使い方は **p**と同じです。  
※**rails server** のshellには表示しません。

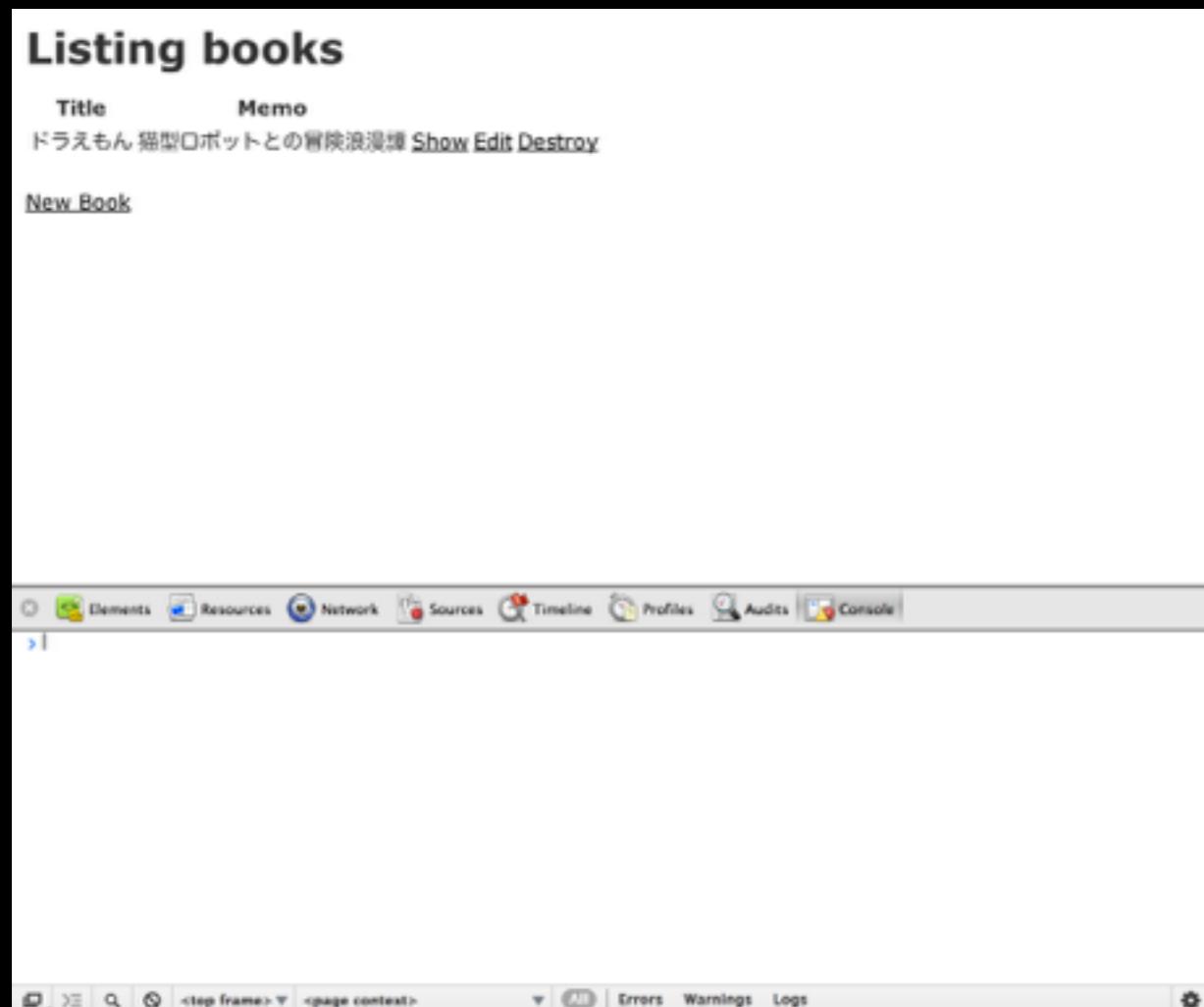
```
class BooksController < ApplicationController
  def index
    @books = Book.all
    Rails.logger.error '*****'
    Rails.logger.error @books
  ...
  end
end
```

← 目印  
←@booksの中身を表示

# Chromeでデバッグ

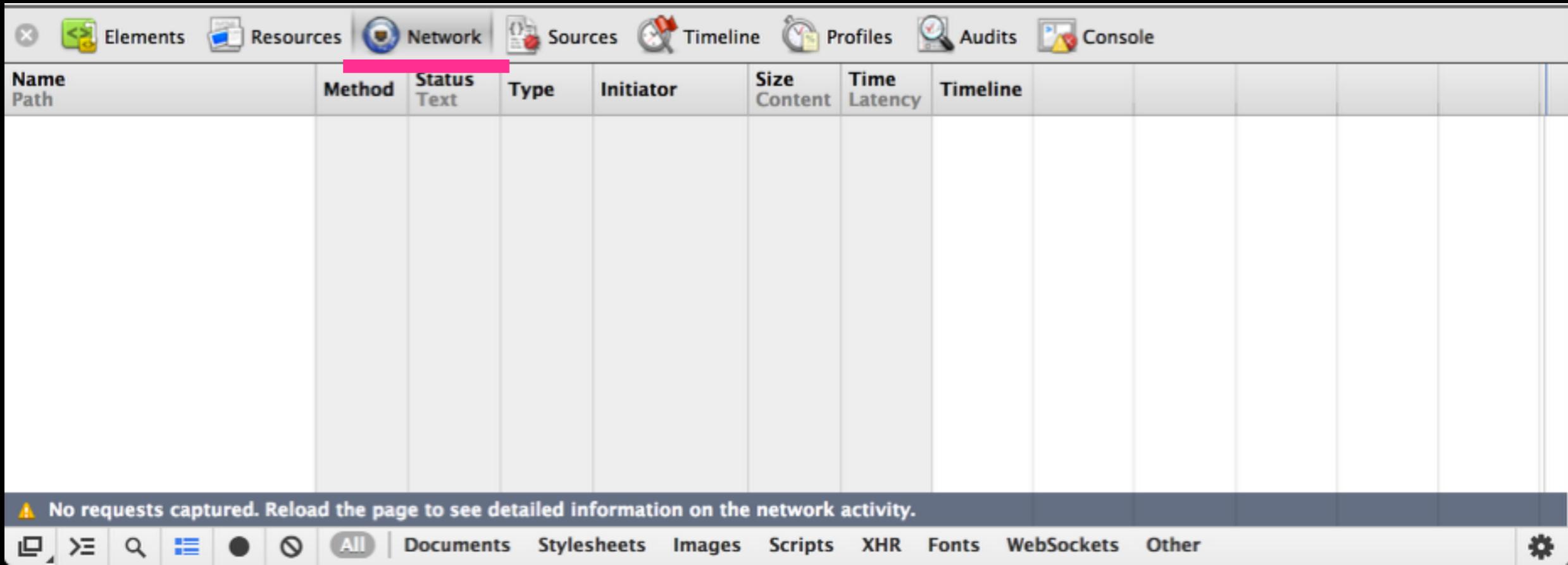
Chrome の デベロッパーツールを使うとリクエストの情報が表示できたりして便利です。

Chrome の 表示 - 開発/管理 - デベロッパーツールを選んでデベロッパーツールを起動します。



←こんなのが  
表示されます。

# Chromeでデバッグ



リクエストの情報を表示させるにはデベロッパーツールの Network タブを押して表示しておきます。  
この状態で表示させたいリクエストを行います。  
例として、/books ページへアクセスしてみましょう。

# Chromeでデバッグ

The screenshot shows the Network tab in the Chrome DevTools. It lists several requests:

Name Path	Method	Status	Type	Initiator	Size Content	Time Latency	Timeline	150ms	225ms	300ms	375ms	450ms
books	GET	304 Not Mod	text/h...	Other	321B 1.24KB	19ms 18ms	<input checked="" type="checkbox"/>					
scaffolds.css /assets	GET	(failed)	Pending	books:7 Parser	13B 0B	5ms 0.0 days						
books.css /assets	GET	304 Not Mod	text/css	books:6 Parser	252B 0B	21ms 21ms	<input checked="" type="checkbox"/>					
books.js /assets	GET	304 Not Mod	application/javascript	books:8 Parser	252B 32B	16ms 16ms	<input checked="" type="checkbox"/>					
jquery.js /assets	GET	304 Not Mod	application/javascript	books:8 Parser	252B 260.63K	28ms 25ms	<input checked="" type="checkbox"/>					
application.css /assets	GET	304 Not Mod	text/css	books:5 Parser	252B 513B	28ms 24ms	<input checked="" type="checkbox"/>					

At the bottom, there are filter buttons for All, Documents, Stylesheets, Images, Scripts, XHR, Fonts, WebSockets, and Other, along with a clear button (X) and a settings gear icon.

なんかいろいろ表示されました。一番最初のbooksをクリックしてみましょう。

(まっさらにしてもう1回実行したいときは、一番下の マークをクリックすると、表示をクリアすることができます。)

# Chromeでデバッグ

The screenshot shows the Chrome DevTools Network tab. On the left, a list of resources is shown, including 'books' (HTML), 'scaffolds.css' (CSS), 'books.css' (CSS), 'books.js' (JavaScript), 'jquery.js' (JavaScript), and 'application.css' (CSS). On the right, a detailed view of a request for 'books' is displayed. The 'Headers' tab is selected. The request URL is 'http://localhost:3003/books', the method is 'GET', and the status code is '304 Not Modified'. The 'Request Headers' section lists various HTTP headers. The 'Cookie' header contains a long session ID. At the bottom, there are filters for 'Documents', 'Stylesheets', 'Images', 'Scripts', 'XHR', 'Fonts', and 'WebSockets'.

Name  
Path

books

scaffolds.css  
/assets

books.css  
/assets

books.js  
/assets

jquery.js  
/assets

application.css  
/assets

Headers Preview Response Cookies Timing

Request URL: <http://localhost:3003/books>

Request Method: GET

Status Code: 304 Not Modified

▼ Request Headers [view source](#)

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,\*/\*;q=0.8

Accept-Charset: UTF-8,\*;q=0.5

Accept-Encoding: gzip,deflate,sdch

Accept-Language: ja,en-US;q=0.8,en;q=0.6

Cache-Control: max-age=0

Connection: keep-alive

Cookie: \_blog\_app\_session=BAh7B0kiD3Nlc3Npb25faWQG0gDg4ZjMzNjAwYjA0BjsAVEkiEF9jc3JmX3Rva2VuBjsARKkiMXcw0RGRNa3pYdC9jL2swUkU9BjsARg%3D%3D--fe120d0b40641889p\_session=BAh7B0kiD3Nlc3Npb25faWQG0gZFRkkiJTY3NjNlNBjsAVEkiEF9jc3JmX3Rva2VuBjsARKkiMVRkemdsVGNLWEhuSET17RIIIORicAR0%2D%2D--8c3c1013d86568h2f65812h0oec7cr8h

All Documents Stylesheets Images Scripts XHR Fonts WebSockets

リクエストの詳細が表示されます。リクエスト URLが <http://localhost:3003/books> であること、HTTPメソッドが GET であることが分かります。

# Chromeでデバッグ

The screenshot shows the Chrome DevTools interface with the Network tab selected. The Response tab is highlighted in pink. On the left, a list of resources is shown with icons indicating their type (e.g., CSS, JS, Images). The main pane displays the HTML code of the response. The code includes the DOCTYPE declaration, HTML tags, head content (title, links to stylesheets and scripts, meta tags for csrf), and body content (h1 and table tags).

Name	Path	Type
books		HTML
scaffolds.css	/assets	CSS
books.css	/assets	CSS
books.js	/assets	JS
jquery.js	/assets	JS
application.css	/assets	CSS

Headers Preview Response Cookies Timing

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>BooksApp</title>
5   <link href="/assets/application.css?body=1" media="all" rel="stylesheet">
6   <link href="/assets/books.css?body=1" media="all" rel="stylesheet" type="text/css">
7   <link href="/assets/scaffolds.css?body=1" media="all" rel="stylesheet" type="text/css">
8   <script src="/assets/jquery.js?body=1" type="text/javascript"></script>
9   <script src="/assets/jquery_ujs.js?body=1" type="text/javascript"></script>
10  <script src="/assets/books.js?body=1" type="text/javascript"></script>
11  <script src="/assets/application.js?body=1" type="text/javascript"></script>
12    <meta content="authenticity_token" name="csrf-param" />
13    <meta content="TdZglTcKXHnHKl7HJzX/XUr5I8z69DgmTslUL642sEE=" name="csrf-token" />
14  </head>
15  <body>
16
17  <h1>Listing books</h1>
18
19  <table>
```

All Documents Stylesheets Images Scripts XHR Fonts WebSockets Other

ちなみに、Responseタブを表示させるとリクエストの結果返ってきたHTML(=現在表示されているページ)が表示されます。

ほかにも便利な機能がいろいろありますがまたの機会に・・・

# pryでデバッグ

pry はプログラムの実行を止めて、その場所でRubyのコードを実行することができる、irbのような対話環境です。

pryを使うためには、まず前準備として  
**Gemfile** に `gem 'pry'` を追加します。  
位置は例えば `gem 'sqlite3'` の次の行あたり。

(※サービス運用も考えてちゃんと書くならば以下のようにした方が良いですが、割愛。  
`group :development, :test do gem "pry" end` )

**Gemfile**

`gem 'pry'` ←追加

追加したら、shell で `bundle` コマンドを実行してpryをインストールします。

`$ bundle`

# pryでデバッグ

次に、ソースコードで止めたい場所へ **binding.pry** と書きます。  
例として、/books へアクセスされたときに停止するように BooksController の index アクションに書いてみましょう。

```
class BooksController < ApplicationController
  def index
    @books = Book.all
    binding.pry ←追記
  ...
  end
end
```

rails server を再起動して、ブラウザでアクセスしてみましょう。  
<http://localhost:3000/books>

# pryでデバッグ

```
From: /Users/igarashi/work/scaffold1115/books_app/app/controller.rb @ line 4 BooksController#index:
```

```
4: def index
5:   @books = Book.all
=> 6:   binding.pry
7:
8:   respond_to do |format|
9:     format.html # index.html.erb
10:    format.json { render json: @books }
11:  end
12: end
```

```
[1] pry(#<BooksController>) > █
```

ブラウザでアクセス後、**rails server** のshellを見るとプログラムが停止してコマンドを打てるようになっています。  
**irb**と同じようにRubyのコードが実行できます。

# pryでデバッグ

```
From: /Users/igarashi/work/scaffold1115/books_app/app/controllers/books_
er.rb @ line 4 BooksController#index:
```

```
4: def index
5:   @books = Book.all
=> 6:   binding.pry
7:
8:   respond_to do |format|
9:     format.html # index.html.erb
10:    format.json { render json: @books }
11:  end
12: end
```

```
[1] pry(#<BooksController>) > p @books
[#<Book id: 1, title: "ドラえもん", memo: "猫型ロボットとの冒険浪漫譚",
at: "2012-11-15 12:59:01", updated_at: "2012-11-15 12:59:01">]
=> [#<Book id: 1, title: "ドラえもん", memo: "猫型ロボットとの冒険浪漫譚
ed_at: "2012-11-15 12:59:01", updated_at: "2012-11-15 12:59:01">]
[2] pry(#<BooksController>) > @books.size
=> 1
[3] pry(#<BooksController>) >
```

コードを実行できるので、変数の中身を p で表示できます。

(実は、p を打たなくても @books と打つだけでも表示できる。)

pry を終了してプログラムを再開するには exit と打ちます。

# 參考資料

# 書式

Rubyコード

`puts "abc"`

実行結果

"abc"

実行結果

文中では `#=>` で書きます

`p 1+2 #=> 3`

shellコマンド

`$ ls`

# 自習用Rails資料

特に教科書は使いませんが、自習用には以下の資料をお勧めします。

▶ RailsTutorial (web)

▶ <http://railstutorial.jp/?version=4.0>

▶ ドットインストール(web)(動画)

▶ [http://dotinstall.com/lessons/basic\\_rails\\_v2](http://dotinstall.com/lessons/basic_rails_v2)

▶ Rails Guide (web)(English)

▶ <http://guides.rubyonrails.org/>

▶ RailsによるアジャイルWebアプリケーション開発 第4版

▶ <http://www.amazon.co.jp/dp/4274068668/>



▶ 改訂新版 基礎Ruby on Rails

▶ <http://www.amazon.co.jp/dp/4844331566/>



▶ たのしいRuby

▶ <http://www.amazon.co.jp/dp/4797372273/>



# 講義資料置き場

過去の資料がDLできます。

<https://github.com/igaiga/hitotsubashi-ruby-2013>

# 雑談・質問用facebookグループ

<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
- ・参加者はスタッフ(講師・TA)と昨年、今年の受講者です
- ・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