

Ruby on Rails 講義

第22回 画像投稿アプリと CRUDの基礎

Kuniaki IGARASHI/igaiga

2012.11.22 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IV
(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19日 IR [「@nifty EMOBILE LTE 定額にねんプラン」の提供を開始](#)
- 2012年4月19日 IR [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

五十嵐邦明

講師 株式会社万葉 エンジニア



画像投稿アプリつくりと CRUDの基礎



sponsored by J.Sato

下準備

以前の講義でも使ったscaffoldを利用してブログを作ります。

```
$ rails new blog_app
```

```
$ cd blog_app
```

```
$ bundle exec rails g scaffold story title:string  
description:text address:string picture:string
```

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

```
$ bundle exec rails s
```

```
http://localhost:3000/stories
```

ブラウザで上記のアドレスへアクセスして動作確認してみましょう。確認できたらrails s を止めて次ぎへ進みます。

ここまででは以前の講義でやったものと同じですね。

gem の利用(Gemfile)

画像を投稿する機能を付けるために、
carrierwave というgemを使います。
Railsアプリでgemを利用する場合は、
Gemfileに以下を足します。
(Gemfileは blog_app フォルダの直下にあります。)
追加する場所は例えば gem 'sqlite3' の次の行へ。

```
gem 'carrierwave'
```

Gemfileを編集したら、bundle コマンドでインストールを行います。次のページで説明します。

bundle install

```
$ bundle
```

bundle は **bundle install** の略で、**Gemfile**に新しく追加した **gem** をインストールするコマンドです。

[豆知識]たびたび出てくる **bundle exec ~** というコマンドは、**Gemfile**に書いてあるgem情報をつかって実行しなさいというコマンドです。

ファイルアップロードの実装

carrierwave gem を利用するための手続きをします。

最初に必要なファイルを生成します。

```
$ bundle exec rails g uploader Picture
```

次にファイルを修正します。

app/models/story.rb

class Story < ActiveRecord::Base の次の行へ以下追加

```
mount_uploader :picture, PictureUploader
```

次のページへつづく

つづき

app/views/stories/_form.html.erb を2箇所修正

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

↓ 以下のように変更

```
<%= form_for(@story, :html => {:multipart => true}) do |f| %>
```

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

↓ 以下のように変更

```
<%= f.file_field :picture %>
```

app/views/stories/show.html.erb を1箇所修正

```
<%= @story.picture %>
```

↓ 以下のように変更

```
<%= image_tag(@story.picture_url, :width => 600) if @story.picture.present? %>
```

ブラウザから動作確認

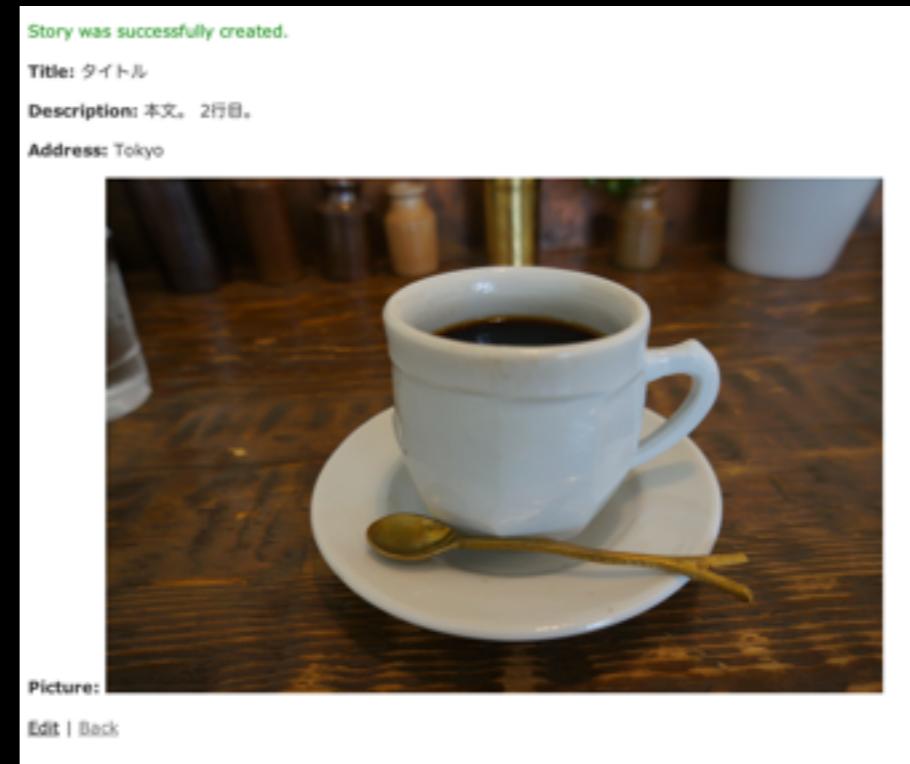
`rails s` を再起動して動作確認してみましょう。

```
$ bundle exec rails s
```

`http://localhost:3000/stories`

ブラウザで上記のアドレスへアクセス

New story ページで Picture に
ファイルを選択して保存すると、
画像ファイルをuploadできます。



では、ここまで何度か登場した
scaffold
について少し説明します。

※画像を投稿する部分は説明しません。gemを使うと便利なことができる、というパワーを感じてもらえばOKです。詳しく知りたい人は自習してみてください。

Scaffold



このタイトルとメモを管理できるアプリ

scaffoldを利用して書籍メモアプリを作った場合を説明します。

```
$ 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

scaffold

かんたんにブログページのような
投稿機能を生成する機能

作成、表示、更新、削除
の4つの機能を持つページ群を
動かすコードを生成します。

※実はscaffoldを活用するというのは実際の開発では
そう多くないです。しかし、基本の勉強にはとても良い
ので、scaffoldで生成したページがどう動くかを中心
に解説を進めていきます。

生成されるコードファイル群

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

```
invoke  active_record
create    db/migrate/20121029112448_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
        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
```

scaffold で生成されるファイルの一部

.rb はRubyコード、.html.erb はHTMLテンプレート

- ▶ **db/migrate/20121029112448_create_books.rb**
- ▶ **app/models/book.rb**
- ▶ **resource_route**
 resources :books
- ▶ **app/controllers/books_controller.rb**
- ▶ **app/views/books**
- ▶ **app/views/books/index.html.erb**
- ▶ **app/views/books/edit.html.erb**
- ▶ **app/views/books/show.html.erb**
- ▶ **app/views/books/new.html.erb**
- ▶ **app/views/books/_form.html.erb**

この後の説明に主に関係してくるのは以下のファイル群

- ▶ **db/migrate/20121029112448_create_books.rb**
- ▶ **app/models/book.rb**
- ▶ **resource_route**
 resources :books
- ▶ **app/controllers/books_controller.rb**
- ▶ **app/views/books**
- ▶ **app/views/books/index.html.erb**
- ▶ **app/views/books/edit.html.erb**
- ▶ **app/views/books/show.html.erb**
- ▶ **app/views/books/new.html.erb**
- ▶ **app/views/books/_form.html.erb**

scaffoldは次のページに示す
作成、表示、更新、削除のための
4つの画面と、
画面のない3つの機能を生成します。

index

一覧

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](#)

new

新規 入力

New book

Title

Memo

[Create Book](#)

[Back](#)

edit

編集

Editing book

Title

ドラえもん

Memo

猫型ロボットとの冒険浪漫譚

[Update Book](#)

[Show](#) | [Back](#)

show

詳細

Title: ドラえもん

Memo: 猫型ロボットとの冒険浪漫譚

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

scaffoldで作られる画面と機能一覧(後で説明していきます)

index
/books

GET

destroy
/books/:id

DELETE

new
/books/new

GET

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

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

redirect

show

/books/:id

GET

redirect

削除 新規入力

新規登録

更新

redirect

redirect

edit

update

show

/books/:id/edit

/books/:id

PUT

GET



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。
(新規作成・表示・更新・削除)

各画面と機能は対応する
アクション(controllerのメソッド)と
viewで処理が行われ表示されます。

scaffoldは
画面のある4つのアクションと、
画面のない3つのアクション、
あわせて7つのアクションを作ります

scaffoldで作成されるアクション

```
class BooksController < ApplicationController
  # GET /books
  # GET /books.json
  def index
    @books = Book.all

    respond_to do |format|
      format.html # index.html.erb
      format.json { render json: @books }
    end
  end

  # GET /books/1
  # GET /books/1.json
  def show
    @book = Book.find(params[:id])

    respond_to do |format|
      format.html # show.html.erb
      format.json { render json: @book }
    end
  end

  # GET /books/new
  # GET /books/new.json
  def new
    @book = Book.new

    respond_to do |format|
      format.html # new.html.erb
      format.json { render json: @book }
    end
  end

  # GET /books/1/edit
  def edit
    @book = Book.find(params[:id])
  end

  # POST /books
  # POST /books.json
  def create
    @book = Book.new(params[:book])

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

  # PUT /books/1
  # PUT /books/1.json
  def update
    @book = Book.find(params[:id])

    respond_to do |format|
      if @book.update_attributes(params[:book])
        format.html { redirect_to @book, notice: 'Book was successfully updated.' }
        format.json { head :no_content }
      else
        format.html { render action: "edit" }
        format.json { render json: @book.errors, status: :unprocessable_entity }
      end
    end
  end

  # DELETE /books/1
  # DELETE /books/1.json
  def destroy
    @book = Book.find(params[:id])
    @book.destroy

    respond_to do |format|
      format.html { redirect_to books_url }
      format.json { head :no_content }
    end
  end
end
```

app/books_controller.rb

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

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

redirect

show

/books/:id

GET

redirect



では、それぞれのアクションと
対応する画面を見ていきましょう。

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

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

redirect

show

/books/:id

GET

redirect



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

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
POST

show
/books/:id
GET

redirect

redirect

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		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](#)

CRUD画面遷移図

index

/books

GET

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

edit

/books/:id/edit

GET

update

/books/:id

PUT

削除 新規入力

destroy

/books/:id

DELETE

new

/books/new

GET

新規登録

create

/books

POST

アクション名
url
HTTPメソッド

画面なし

redirect

show

/books/:id

GET

newアクション

GET

/books/new

新規入力画面

New book

Title

Memo

Create Book

Back

CRUD画面遷移図

index

/books

GET

編集

edit

/books/:id/edit

GET

destroy

/books/:id
DELETE

画面あり

更新

update

/books/:id
PUT

この図
の見方

アクション名
url
HTTPメソッド

アクション名
url
HTTPメソッド

画面なし

redirect

new

/books/new
GET

新規登録

create

/books
POST

show

/books/:id
GET

削除 新規入力

redirect

redirect

editアクション

GET

/books/:id/edit

編集画面

Editing book

Title

ドラえもん

Memo

猫型ロボットとの冒険浪漫譚

[Update Book](#)

[Show](#) | [Back](#)

CRUD画面遷移図

index

/books

GET

削除 新規入力

new

/books/new

GET

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

update

/books/:id

PUT

show

/books/:id

GET

showアクション

詳細画面

GET /books/:id

Title: ドラえもん

Memo: 猫型ロボットとの冒険浪漫譚

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

CRUD画面遷移図

index

/books

GET

編集

この図
の見方

アクション名
url
HTTPメソッド

画面あり

edit

/books/:id/edit

GET

update

/books/:id

PUT

削除 新規入力

destroy

/books/:id

DELETE

new

/books/new

GET

新規登録

create

/books

POST

画面なし

redirect

show

/books/:id

GET

redirect

更新

redirect

redirect

edit

/books/:id/edit

GET

update

/books/:id

PUT

削除 新規入力

destroy

/books/:id

DELETE

new

/books/new

GET

新規登録

create

/books

POST

画面なし

redirect

show

/books/:id

GET

redirect

更新

redirect

redirect

edit

/books/:id/edit

GET

update

/books/:id

PUT

削除 新規入力

destroy

/books/:id

DELETE

new

/books/new

GET

新規登録

create

/books

POST

画面なし

redirect

show

/books/:id

GET

redirect

更新

redirect

redirect

edit

/books/:id/edit

GET

update

/books/:id

PUT

削除 新規入力

destroy

/books/:id

DELETE

new

/books/new

GET

新規登録

create

/books

POST

画面なし

redirect

show

/books/:id

GET

redirect

更新

redirect

redirect

edit

/books/:id/edit

GET

update

/books/:id

PUT

削除 新規入力

destroy

/books/:id

DELETE

new

/books/new

GET

新規登録

create

/books

POST

画面なし

redirect

show

/books/:id

GET

redirect

更新

redirect

redirect

edit

/books/:id/edit

GET

update

/books/:id

PUT

削除 新規入力

destroy

/books/:id

DELETE

new

/books/new

GET

新規登録

create

/books

POST

画面なし

redirect

show

/books/:id

GET

redirect

更新

redirect

redirect

edit

/books/:id/edit

GET

update

/books/:id

PUT

削除 新規入力

destroy

/books/:id

DELETE

new

/books/new

GET

新規登録

create

/books

POST

画面なし

redirect

show

/books/:id

GET

redirect

更新

redirect

redirect

edit

/books/:id/edit

GET

update

/books/:id

PUT

削除 新規入力

destroy

/books/:id

DELETE

new

/books/new

GET

新規登録

create

/books

POST

画面なし

redirect

show

/books/:id

GET

redirect

更新

redirect

redirect

edit

/books/:id/edit

GET

update

/books/:id

PUT

削除 新規入力

destroy

/books/:id

DELETE

new

/books/new

GET

新規登録

create

/books

POST

画面なし

redirect

show

/books/:id

GET

redirect

更新

redirect

redirect

edit

/books/:id/edit

GET

update

/books/:id

createアクション 新規登録

POST /books

CRUD画面遷移図

index

/books

GET

編集

この図
の見方

アクション名
url
HTTPメソッド

画面あり

edit

/books/:id/edit

GET

update
/books/:id
PUT

削除 新規入力

destroy

/books/:id
DELETE

new

/books/new
GET

新規登録

create

/books
POST

画面なし

redirect

show

/books/:id
GET

redirect

更新

redirect

redirect

updateアクション **更新**

PUT /books/:id

CRUD画面遷移図

index

/books

GET

削除 新規入力

new

/books/new

GET

新規登録

create

/books

POST

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

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

show

/books/:id

GET

destroyアクション 削除

DELETE /books/:id

ここまで説明した部分はこのファイルに書かれています

- ▶ db/migrate/20121029112448_create_books.rb
 - ▶ app/models/book.rb
 - ▶ resource_route
 - resources :books
 - ▶ app/controllers/books_controller.rb
 - ▶ app/views/books
 - ▶ app/views/books/index.html.erb
 - ▶ app/views/books/edit.html.erb
 - ▶ app/views/books/show.html.erb
 - ▶ app/views/books/new.html.erb
 - ▶ app/views/books/_form.html.erb
- url+HTTPメソッドと
アクションの対応表
- 7つの
アクション
- 対応する
画面表示

では、処理の流れを追いかけてみ
ましょう。

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		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](#)

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

update

/books/:id

PUT

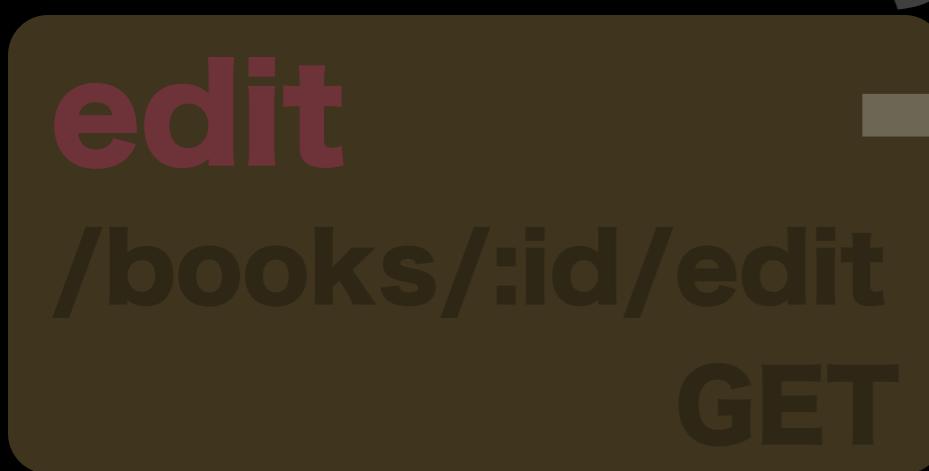
show

/books/:id

GET

redirect

redirect



redirect

redirect

redirect

(復習)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://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

Routes

```
$ bundle exec rake routes
```

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

`rake routes` コマンドを使うと、URL+HTTPメソッドに対するコントローラアクションの対応表が表示されます。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7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。)

\$ bundle exec rake routes

books	GET	/books(.:format)	books#index
	POST	/books(.:format)	books#create
new_book	GET	/books/new(.:format)	books#new
edit_book	GET	/books/:id/edit(.:format)	books#edit
book	GET	/books/:id(.:format)	books#show
	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://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

Controller

app/controllers/books_controller.rb

```
class BooksController < ApplicationController
```

```
  def index
```

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

```
    respond_to do |format|
```

```
      format.html # index.html.erb
```

```
      format.json { render json: @books }
```

```
    end
```

```
  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アクションで行っていることで重要なのは

@books = Book.all の行です。

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。

インスタンス変数へ代入するのはViewで表示の際に使うからです。

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/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

View

app/views/index.html.erb

...

```
<% @books.each do |book| %> ←@booksはbookがいくつか  
<tr>  
  <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

ドラえもん 猫型ロボットとの冒険浪漫譚

Memo

君に届け 北海道の高校を舞台にした青春群像

[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](#)

(復習)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		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](#)

以上の処理を経て一覧画面が表示されています。

index

/books

GET

いま説明した箇所

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

edit

/books/:id/edit

GET

新規入力

削除

destroy

DELETE

new

/books/new

GET

新規登録

create

/books

POST

**アクション名
url
HTTPメソッド**

画面なし

redirect

show

/books/:id

GET



redirect

index

/books

GET

新規入力

new

/books/new

GET

削除

destroy

/books/:id

DELETE

新規登録

redirect

編集

のこりはまた次回以降で。

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

redirect

show

/books/:id

GET

削除

destroy

/books/:id

DELETE

新規登録

redirect

create

/books

POST

redirect

のこりはまた次回以降で。

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

更新

update

/books/:id

PUT

redirect

show

/books/:id

GET

削除

destroy

/books/:id

DELETE

新規登録

redirect

create

/books

POST

redirect

のこりはまた次回以降で。

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

更新

update

/books/:id

PUT

redirect

show

/books/:id

GET

削除

destroy

/books/:id

DELETE

新規登録

redirect

create

/books

POST

redirect

のこりはまた次回以降で。

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

更新

update

/books/:id

PUT

redirect

show

/books/:id

GET

削除

destroy

/books/:id

DELETE

新規登録

redirect

create

/books

POST

redirect

のこりはまた次回以降で。

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

更新

update

/books/:id

PUT

redirect

show

/books/:id

GET

削除

destroy

/books/:id

DELETE

新規登録

redirect

create

/books

POST

redirect

のこりはまた次回以降で。

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

更新

update

/books/:id

PUT

redirect

show

/books/:id

GET

より詳しく知りたい人は

以下のページが参考になります。

http://guides.rubyonrails.org/getting_started.html

(日本語の資料が良い方は)

この本が易しめです。

改訂新版 基礎Ruby on Rails

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



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

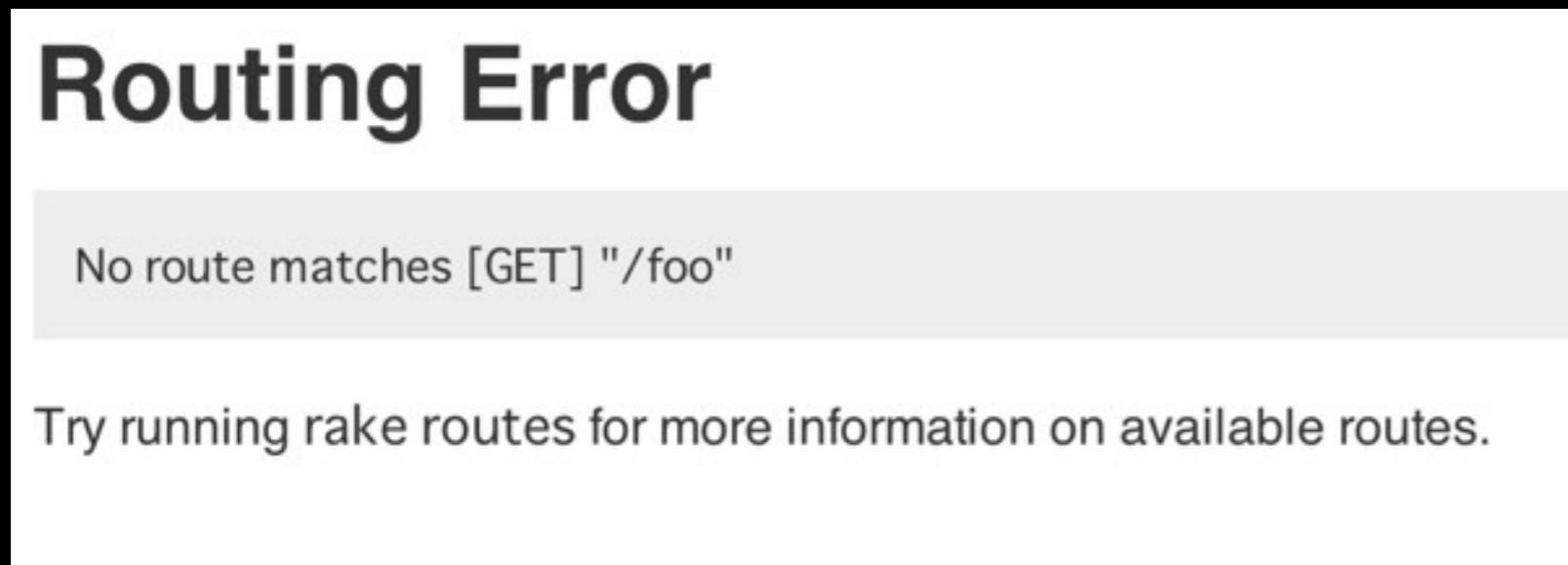
ブラウザに表示されるエラーを読む

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
```

```
0
```

```
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
```

```
0
```

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
94daccda28>):
  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。

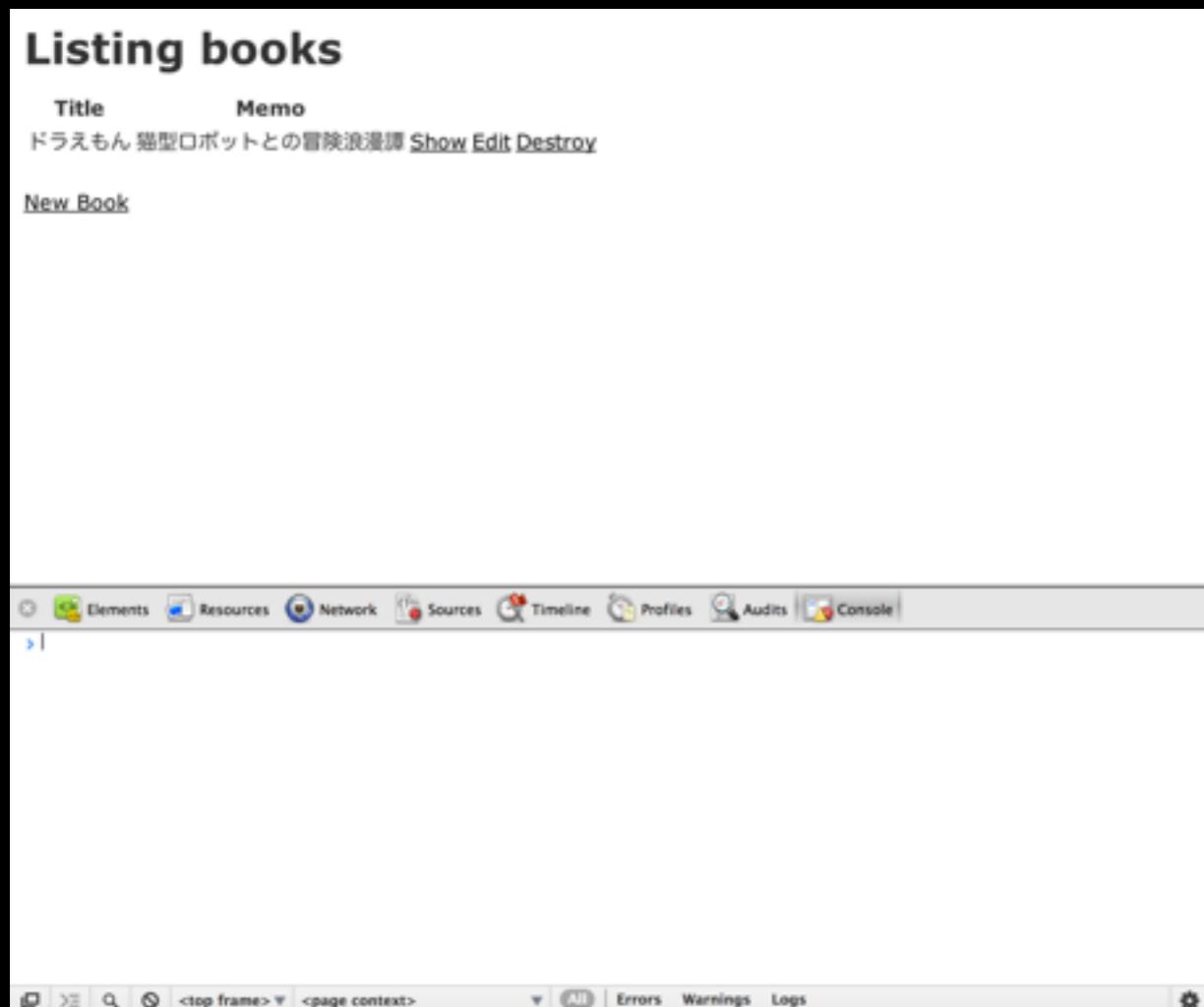
```
class BooksController < ApplicationController
  def index
    @books = Book.all
    Rails.logger.error '*****'
    Rails.logger.error @books
  ...
  end
end
```

← 目印
←@booksの中身を表示

Chromeでデバッグ

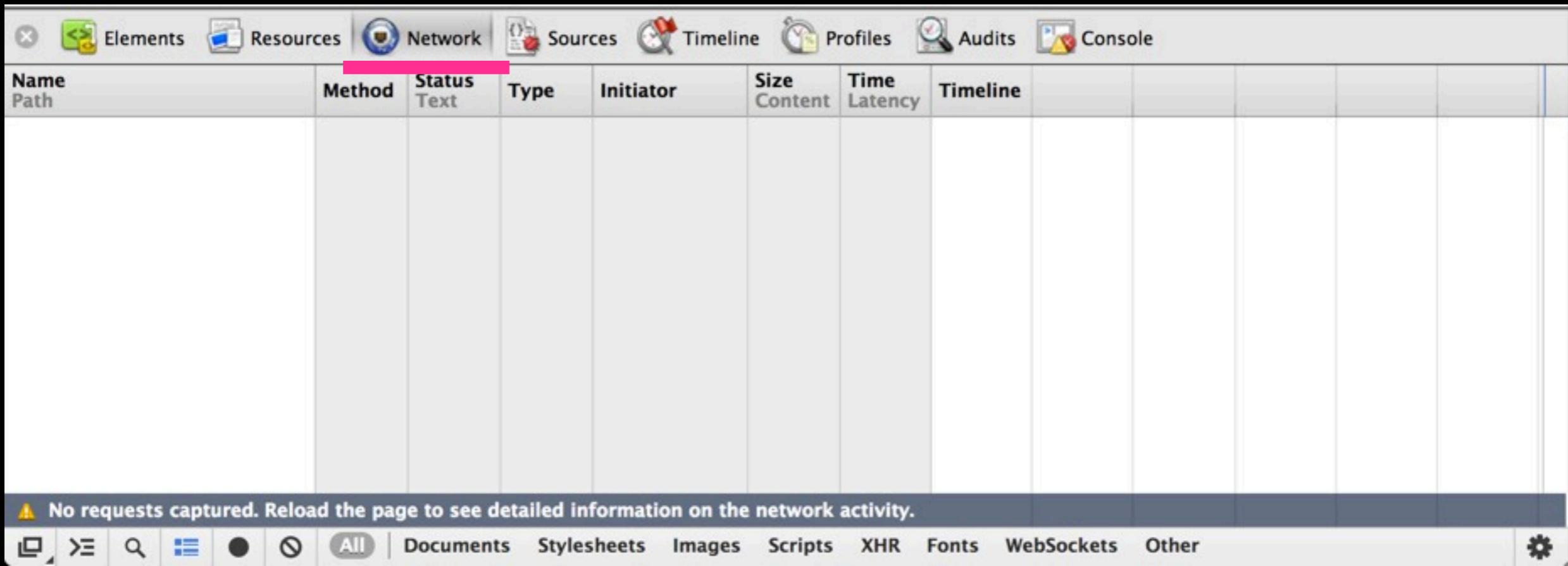
Chrome の デベロッパーツールを使うとリクエストの情報が表示できたりして便利です。

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 Chrome DevTools Network tab with the following details:

Name Path	Method	Status Text	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	<input type="checkbox"/>					
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 of the Network tab, there are several filter icons: All, Documents, Stylesheets, Images, Scripts, XHR, Fonts, WebSockets, and Other. There is also a red 'X' icon with the number '1' and a 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.

Name
Path

Elements Resources Network Sources Timeline Profiles Audits Console

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=BAh7B0kiD3Nlc3Npb25faWQG0gZFRkkiJTY3NjNlNBjsAVEkiEF9jc3JmX3Rva2VuBjsARKkiMVRkemdsVGNLWEhuSEt17DIIIORic1R0%2D%2D--8c3c1013d86568h7f65812h00ec7c8h

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 with a pink background. On the left, a list of resources is shown with their names and paths. On the right, the response content is displayed as a numbered code block.

Name	Path	Content
books		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>
scaffolds.css	/assets	
books.css	/assets	
books.js	/assets	
jquery.js	/assets	
application.css	/assets	

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。

講義資料置き場

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

<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