

Ruby on Rails 講義 第23回 画像投稿アプリ

五十嵐邦明

twitter : igaiga555

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



2013.12.5 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>

復習

Model Migration Database(DB)

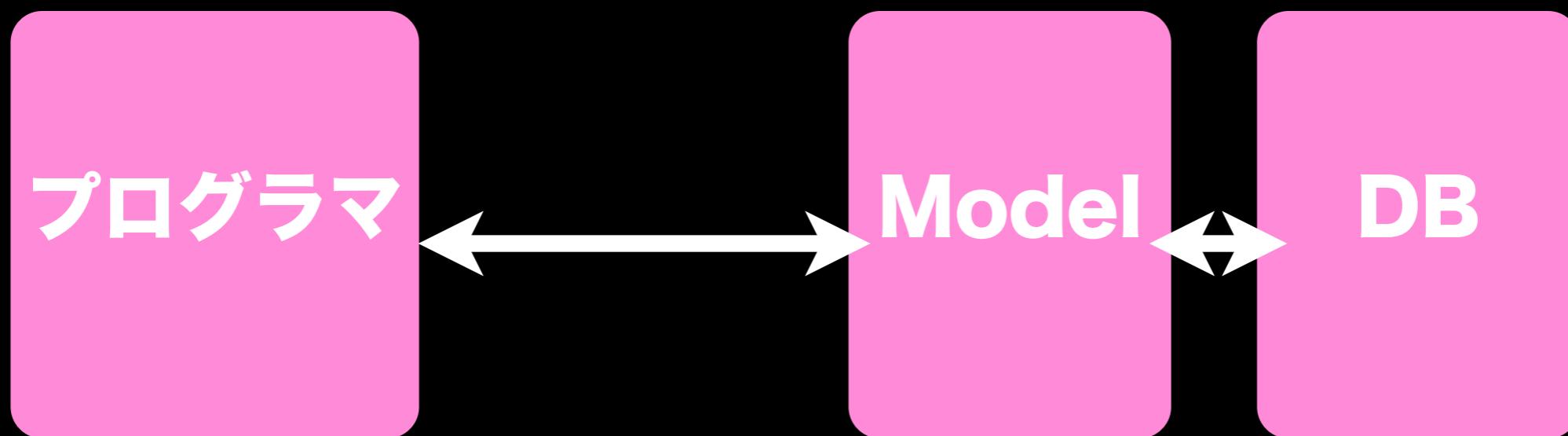
データベース(DB)とは？

データを保存したり読み出したり検索したりするために特化したプログラムです。

モデルはDBを便利につかうための仕組みでもあります。

DBは高機能で堅牢でかつ高速です。しかし、DBにアクセスするには一般に専用の言語(SQLと言います)を用いることが多く、Rubyのコードからは扱いづらい面もあります。

モデルはRubyでDBを容易に扱うことができる機能も提供します。



Modelが簡単にDBにアクセスできる機能を提供するので、プログラマはModelを通じてDBとデータをやりとりできる。

scaffoldはモデルやDBの設計図をつくり、

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

rake db:migrate で
設計図をもとにDBを作ります。

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

DB設計図 - migration

db/migrate/20121122005122_create_books.rb

```
class CreateBooks < ActiveRecord::Migration
```

```
  def change
```

```
    create_table :books do |t|
```

```
      t.string :title
```

```
      t.text :memo
```

```
      t.timestamps
```

```
    end
```

```
  end
```

```
end
```

DBの設計図を**migration**(マイグレーション)ファイルと呼びます。
migrationもRails(Ruby)のコードで書かれています。

DB設計図 - migration

db/migrate/20121122005122_create_books.rb

```
class CreateBooks < ActiveRecord::Migration
  def change
    create_table :books do |t|
      t.string :title
      t.text :memo
      t.timestamps
    end
  end
end
```

DBのテーブルは、
Excelをイメージすると
分かり易いです。

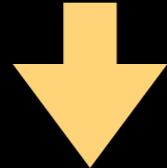
booksテーブル

新しくデータを
入れるときは
このスタイル→
で保存していく。

	A	B	C	D	E
1		title	memo	created_at	updated_at
2		ドラえもん	猫型ロボットとの冒険浪漫譚	2012/11/22 0:58	2012/11/22 0:58
3		君に届け	北海道の高校を舞台にした青春群像	2012/11/22 1:23	2012/11/25 14:12
4					
5					
6					
7					
8					
9					
10					

DB設計図(migration)からDBを作るのが rake db:migrate コマンドです

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



ファイル生成

```
db/migrate/20121122005122_create_books.rb
class CreateBooks < ActiveRecord::Migration
  def change
    create_table :books do |t|
      t.string :title
      t.text :memo
      t.timestamps
    end
  end
end
```



\$ bundle exec rake db:migrate
migrationファイル(DB設計図)からDBを作る

DB

booksテーブル

title, memo, created_at, updated_at

基本的なモデルの使い方 1 保存編

以下の2つの手順を踏むだけでデータを保存することができます。

```
book = Book.new(title: "ハチミツとクローバー",
                 memo: "美大を舞台にした青春ラブコメ")
```

Book.new で Book Model オブジェクトを作ります。
このとき、タイトルとメモの情報を渡すことができます。

```
book.save
```

Book Model オブジェクトの **save** メソッドを呼ぶと保存できます。

※尚、モデル名（クラス名）は英語の单数形にするルールがあります。

基本的なモデルの使い方 2 読み出し編

さきほど保存したデータを読み出してみましょう。

```
books = Book.all.to_a
```

Book.all で保存されているBook Model の全データを取得できます。
(以前に説明した一覧画面(indexアクション)でこのメソッドが使われています)

Book.all.to_a すると配列にBookオブジェクトが詰まって返ってきます。
(**to_a** はArrayへ変換するメソッドです。)

基本的なモデルの使い方 3 検索編

```
book = Book.where(title: "ハチミツとクローバー").first
```

```
book.title → "ハチミツとクローバー"
```

```
book.memo → "美大を舞台にした青春ラブコメ"
```

whereメソッドを使うと検索ができます。

タイトルが"ハチミツとクローバー"であるBookオブジェクトが返ります。
(検索結果が複数になることもあるので、firstメソッドで最初の1つを取得しています。)

Bookオブジェクトは titleメソッドでタイトルを、 memoメソッドでメモをそれぞれ返します。

scaffoldコマンドを打つと

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

scaffoldコマンドを打つとファイルを生成します

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

ファイル生成



ファイル生成

migrationファイル

db/migrate/20121122005122_create_books.rb

class CreateBooks < ActiveRecord::Migration

def change

 create_table :books do |t|

 t.string :title

 t.text :memo

 t.timestamps

 end

end

end

modelファイル
app/models/book.rb

```
class Book < ActiveRecord::Base  
end
```

続いて`rake db:migrate`コマンドを打つと、

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

ファイル生成

↓ ファイル生成

migrationファイル

`db/migrate/20121122005122_create_books.rb`

`class CreateBooks < ActiveRecord::Migration`

`def change`

`create_table :books do |t|`

`t.string :title`

`t.text :memo`

`t.timestamps`

`end`

`end`

`end`

↓
modelファイル
`app/models/book.rb`

`class Book < ActiveRecord::Base`
end

↓

\$ `bundle exec rake db:migrate`

続いて`rake db:migrate`コマンドを打つと、DBが作られます

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

ファイル生成

↓ ファイル生成

migrationファイル

`db/migrate/20121122005122_create_books.rb`

```
class CreateBooks < ActiveRecord::Migration
```

```
def change
```

```
  create_table :books do |t|
```

```
    t.string :title
```

```
    t.text :memo
```

```
    t.timestamps
```

```
  end
```

```
end
```

```
end
```

↓
modelファイル

`app/models/book.rb`

```
class Book < ActiveRecord::Base  
end
```

↓ **\$ bundle exec rake db:migrate**

migrationファイル(DB設計図)からDBを作る

DB

booksテーブル

title, memo, created_at, updated_at

できたmodelファイルとDBを利用してアプリは動きます。

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

ファイル生成

↓ ファイル生成

migrationファイル

db/migrate/20121122005122_create_books.rb

```
class CreateBooks < ActiveRecord::Migration
```

```
def change
```

```
  create_table :books do |t|
```

```
    t.string :title
```

```
    t.text :memo
```

```
    t.timestamps
```

```
  end
```

```
end
```

```
end
```

modelファイル
app/models/book.rb

```
class Book < ActiveRecord::Base  
end
```

↓ \$ bundle exec rake db:migrate
migrationファイル(DB設計図)からDBを作る

DB
booksテーブル

title, memo, created_at, updated_at

モデルの使い方

Model

```
app/models/book.rb
```

```
class Book < ActiveRecord::Base  
end
```

DB

booksテーブル

title, memo, created_at, updated_at

```
book = Book.new(title: "ハチミツとクローバー",  
                 memo: "美大を舞台にした青春ラブコメ")
```

Book.new で Book Model オブジェクトを作ります。

引数で title, model といった各カラムのデータを渡せます。

book.save

save メソッドを呼ぶと保存できます。

※ モデル名（クラス名）は英語の单数形にするルールがあります。

モデルの使い方 つづき

Model

```
app/models/book.rb
```

```
class Book < ActiveRecord::Base  
end
```

DB

booksテーブル

title, memo, created_at, updated_at

books = Book.all.to_a

Book.all でDBに保存されているBook Model の全データを取得できます。

Book.all.to_aするとArrayにBookオブジェクトが詰まって返ってきます。

```
book = Book.where(title: "ハチミツとクローバー")
```

```
book.title → "ハチミツとクローバー"
```

```
book.memo → "美大を舞台にした青春ラブコメ"
```

whereメソッドを使うと検索ができます。

タイトルが"ハチミツとクローバー"であるBookオブジェクトが返ります。

検索で取れたBookオブジェクトは、**title**メソッドでタイトルを、

memoメソッドでメモをそれぞれ取得できます。

画像投稿アプリつくり



sponsored by J.Sato

前準備

いつも作っている `books_app` に `picture` を加えて作ります。
(自信がある人は先週の内容を使って既存の`books_app` に
`picture:string` のカラムを追加する方法でやると練習になります。)

```
$ rails new books_picture_app
```

```
$ cd books_picture_app
```

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

```
$ 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

gem の利用と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に利用したいgem名の一文を追加します。
Gemfileは books_app フォルダの直下にあります。

gem 'carrierwave'

※追加する場所は例えば gem 'sqlite3' の次の行へ。

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。
※**rails s** が起動している場合は一度終了させてください。

bundle は **Bundler** という名のライブラリのコマンドで、**Gemfile**を設定ファイルとしてアプリで利用する複数の**gem** を管理する仕組みです。

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
```

※これらの手順は、 carrierwave を使う場合の作法だと思ってください。
gem の作者から使い方の情報が提供されているので、それに沿って実装していきます。

次にファイルを修正します。 (次のページへつづく)

次にファイルを修正します。

app/models/book.rb を修正

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

mount_uploader :picture, PictureUploader

app/views/books/_form.html.erb を修正

<%= f.text_field :picture %>

↓ 以下のように変更

<%= f.file_field :picture %>

app/views/books/show.html.erb を修正

<%= @book.picture %>

↓ 以下のように変更

<%= image_tag(@book.picture_url, width: 400) if @book.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/books`

ブラウザで上記のアドレスへアクセス

New book 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。



pry でデバッグ



pryでデバッグ

- ▶ コードが想定通り動かない
- ▶ 変数の中身を見たい
- ▶ コードを動かしながら実装したい

こんなときは `pry` を使うと便利かもしれません。

`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でデバッグ

前準備

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。

Gemfile

gem 'pry', group: [:development, :test] ←追加

※ gem 'pry' だけでも動作します。上記は開発モード(:development)とテストモード(:test)だけでpryを使い、
サービス運用モード(:productionといいます)ではpryを使わない設定にしています。

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

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のようにRubyのコードが実行できます。

```
From: /Users/igarashi/work/books_picture_app/app/controllers/books_controller.rb
@ line 8 BooksController#index:

  6: def index
  7:   @books = Book.all
=> 8:   binding.pry
  9: end
```

pryでデバッグ

```
Started GET "/books" for 127.0.0.1 at 2013-12-01 17:46:52 +0900
Processing by BooksController#index as HTML

From: /Users/igarashi/work/books_picture_app/app/controllers/books_controller.rb @ line 8 BooksController#index:

6: def index
7:   @books = Book.all
=> 8:   binding.pry
9: end

[1] pry(#<BooksController>) > p @books
Book Load (0.2ms)  SELECT "books".* FROM "books"
#<ActiveRecord::Relation [#<Book id: 2, title: "スタートアップRuby", memo: "Rubyの入門書", picture: "startup_ruby.png", created_at: "2013-12-01 06:46:53", updated_at: "2013-12-01 06:46:53">, #<Book id: 4, title: "4月は君の嘘", memo: "ピアノを巡る青春物語", picture: nil, created_at: "2013-12-01 08:46:39", updated_at: "2013-12-01 08:46:39">]
=> [#<Book id: 2, title: "スタートアップRuby", memo: "Rubyの入門書", picture: "startup_ruby.png", created_at: "2013-12-01 06:46:53", updated_at: "2013-12-01 06:46:53">,
 #<Book id: 4, title: "4月は君の嘘", memo: "ピアノを巡る青春物語", picture: nil, created_at: "2013-12-01 08:46:39", updated_at: "2013-12-01 08:46:39">]
```

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。

`params`を表示したり、`Book.where` で検索を試してみたり、工夫次第で便利な道具として使うことができます。

演習

**Bookを新規登録する際に、ブラウザから飛んでくるリクエストの
パラメータを `pry` を使って表示させてください。**

**また、`params`がChrome のデベロッパーツールでの表示と
同じになることを確認してください。**

ヒント

- ▶ 新規登録の際は `create` アクションで処理が行われます
- ▶ リクエストのパラメータは `params` という変数に格納されています
- ▶ Chrome デベロッパーツールの使い方は巻末の付録に書いています

演習回答

コード

```
class BooksController < ApplicationController
...
def create
  binding.pry
  @book = Book.new(book_params)
...
end
```

Chrome デベロッパーツール

The screenshot shows the Chrome Developer Tools Network tab with a single request listed. The request is a POST to '/books'. The Headers tab is selected. The 'Content-Type' header is set to 'multipart/form-data; boundary=----WebKitFormBoundaryln6Mdg091fr0lPyB'. The request body contains three form-data fields:

- Content-Disposition: form-data; name="book[title]"
Value: となりの閑くん
- Content-Disposition: form-data; name="book[memo]"
Value: 授業中をあらゆる手で楽しむファンタジスタ閑くん
- Content-Disposition: form-data; name="book[picture]"; filename="startup_ruby.png"
Content-Type: image/png
Value: An uploaded file (file handle @0x007f945ea32650)

At the bottom, it shows 1 requests | 13 B transferred.

pry

```
From: /Users/igarashi/work/books_picture_app/app/controllers/books_controller.rb @ line 27 BooksController#create:
26: def create
=> 27:   binding.pry
28:   @book = Book.new(book_params)
29:
30:   respond_to do |format|
31:     if @book.save
32:       format.html { redirect_to @book, notice: 'Book was successfully created.' }
33:       format.json { render action: 'show', status: :created, location: @book }
34:     else
35:       format.html { render action: 'new' }
36:       format.json { render json: @book.errors, status: :unprocessable_entity }
37:     end
38:   end
39: end

[1] pry(#<BooksController>)> params
=> {"utf8"=>"/",
"authenticity_token"=>"IuXV+wKu6pSaefvik76tq+Aa5C5rdKEGtSu/b0TKdT0=",
"book"=>
{"title"=>"となりの閑くん",
"memo"=>"授業中をあらゆる手で楽しむファンタジスタ閑くん",
"picture"=>
#<ActionDispatch::Http::UploadedFile:0x007f945ea32650
@content_type="image/png",
@headers=
"Content-Disposition: form-data; name=\"book[picture]\"; filename=\"startup_ruby.png\"\r\nContent-Type: image/png\r\n",
@original_filename="startup_ruby.png",
@tempfile=
#<File:/var/folders/cr/7xtxcnzn2ts4b4k2cw4f6gnc0000gn/T/RackMultipart20131201-2692-1al6bww>>},
"commit"=>"Create Book",
"action"=>"create",
"controller"=>"books")
```


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

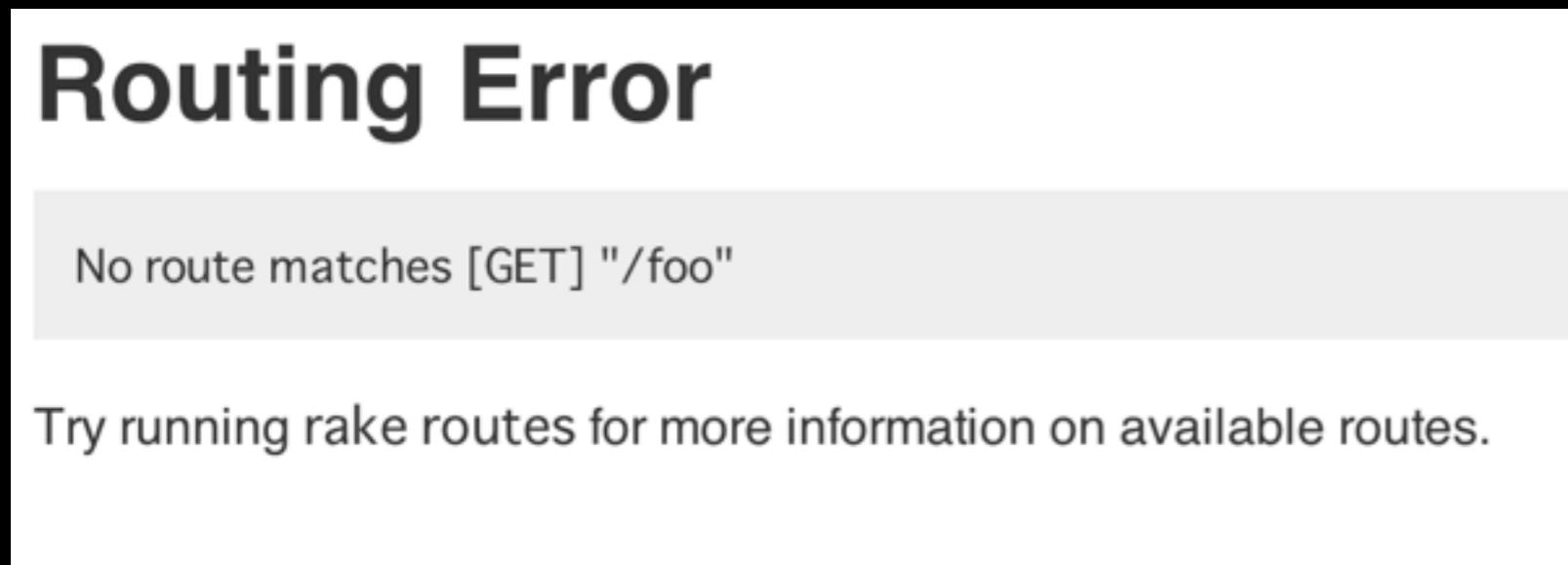
ブラウザに表示されるエラーを読む

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
```

```
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
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。

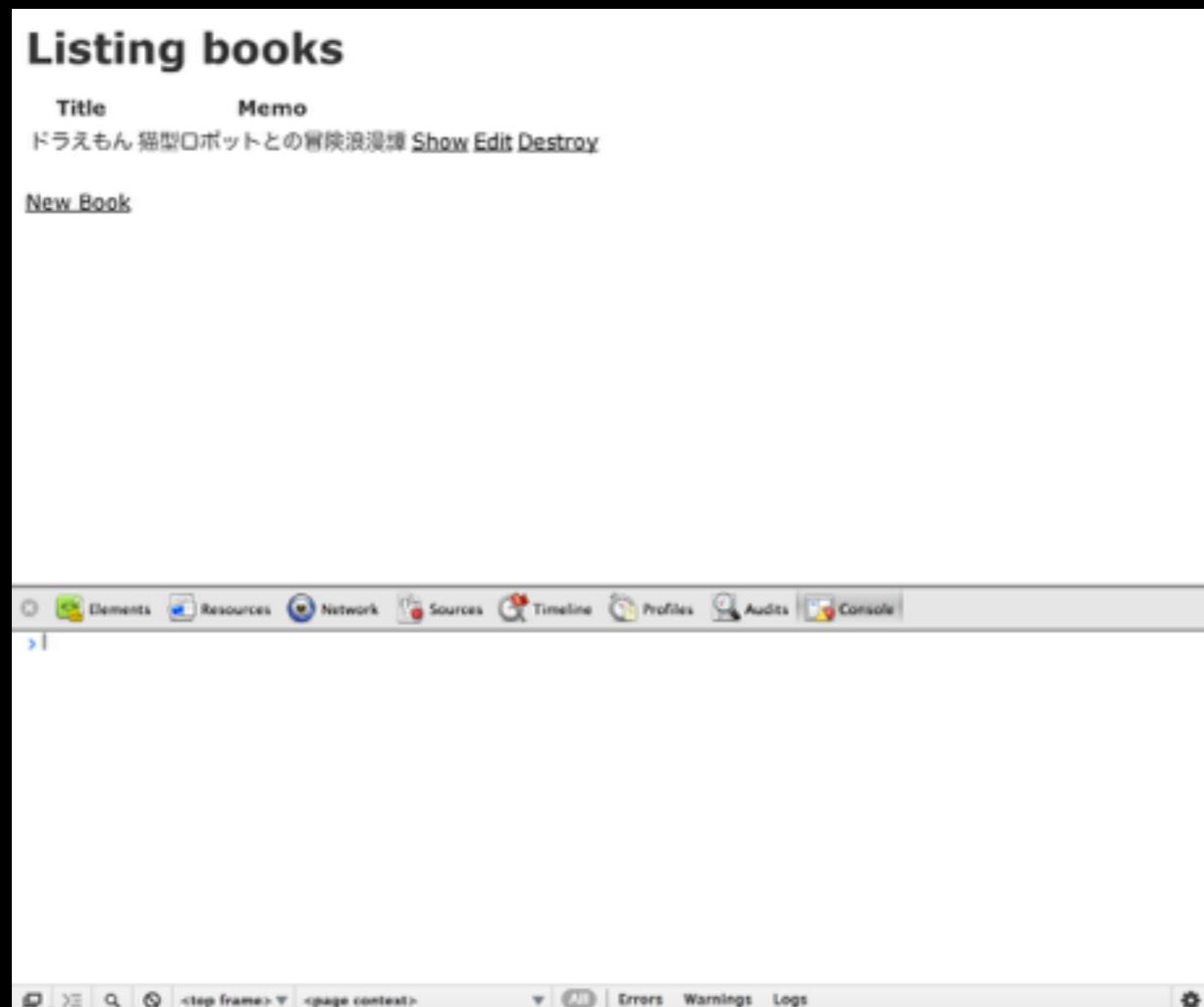
```
class BooksController < ApplicationController
  def index
    @books = Book.all
    Rails.logger.error '*****'
    Rails.logger.error @books
  ...
  end
end
```

← 目印
←@booksの中身を表示

Chromeでデバッグ

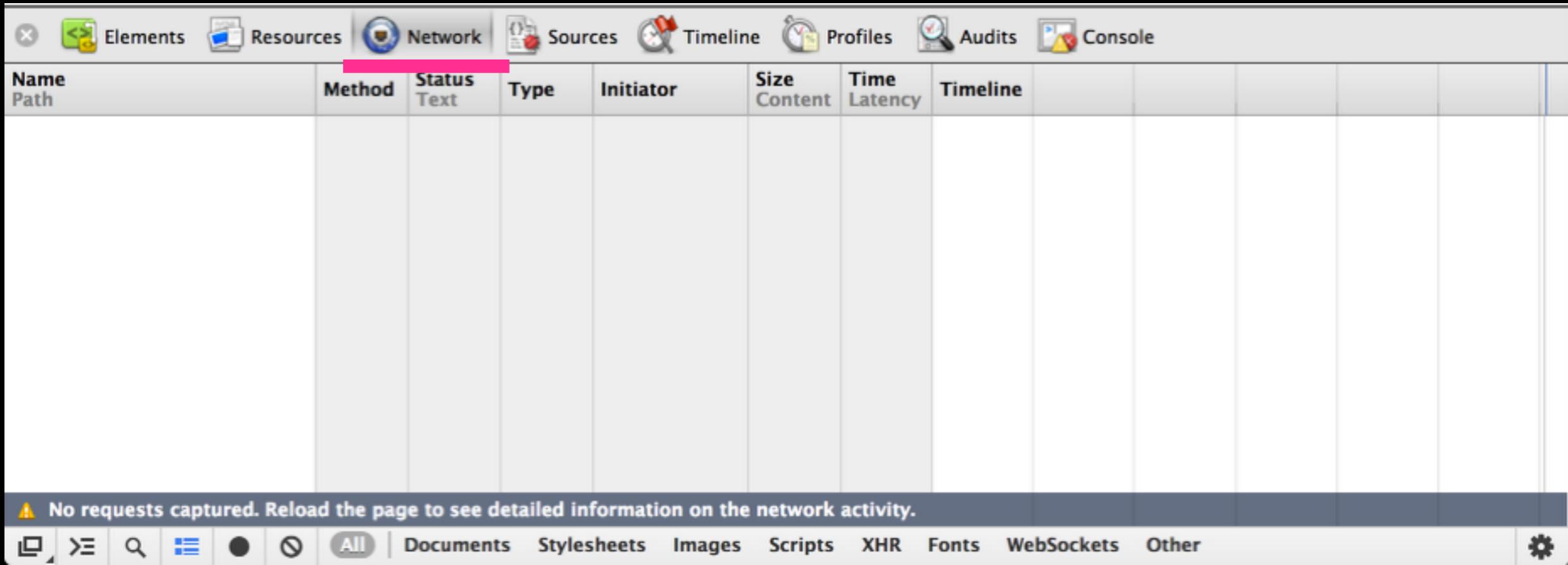
Chrome の デベロッパーツールを使うとリクエストの情報が表示できたりして便利です。

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 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--8c3c1013d86568h2f65817h00ec7c8h

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	Content
books		<!DOCTYPE html>
scaffolds.css	/assets	<html>
books.css	/assets	<head>
books.js	/assets	<title>BooksApp</title>
jquery.js	/assets	<link href="/assets/application.css?body=1" media="all" rel="stylesheet" type="text/css">
application.css	/assets	<link href="/assets/books.css?body=1" media="all" rel="stylesheet" type="text/css">
		<link href="/assets/scaffolds.css?body=1" media="all" rel="stylesheet" type="text/css">
		<script src="/assets/jquery.js?body=1" type="text/javascript"></script>
		<script src="/assets/jquery_ujs.js?body=1" type="text/javascript"></script>
		<script src="/assets/books.js?body=1" type="text/javascript"></script>
		<script src="/assets/application.js?body=1" type="text/javascript"></script>
		<meta content="authenticity_token" name="csrf-param" />
		<meta content="TdZglTcKXHnHKl7HJzX/XUr5I8z69DgmTslUL642sEE=" name="csrf-"
		</head>
		<body>
		<h1>Listing books</h1>
		<table>

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

▶ 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/>



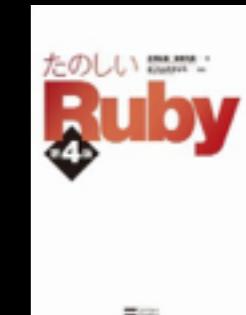
▶ 改訂新版 基礎Ruby on Rails

▶ <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