

Ruby on Rails 講義 第22回 model

五十嵐邦明

twitter : igaiga555

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



2013.11.28 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>

model

前準備

scaffoldで作った、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

今日はデータを長期保存する
方法についてお話しします。

プログラムの中で変数に入れたデータは、
変数の有効範囲（スコープと言います）
が終わると消えてしまいます。

ローカル変数

そのメソッド内が有効範囲です。

```
def foo  
  x = "Hello world!"  
end
```



インスタンス変数(@はじまりの変数)

インスタンスがなくなると一緒に消えます。Railsの場合は、1つのリクエスト内が有効範囲です。@book のようにインスタンス変数に入れるとControllerからViewまで使うことができますが、その後なくなります。

でも、つくったRailsアプリは 別のリクエストでも情報が見れますよね？

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

自分で入力したデータがブラウザから何回アクセスしても表示されます。
※ブラウザで1回の表示が1回のリクエストです。

でも、つくったRailsアプリは 別のリクエストでも情報が見れますよね？

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](#)

データがずっと残っているのは「データを保存する」仕事をしている「何かの仕組み」があるからなのです。

その仕組みが今日の主役 **Model(モデル)** です。

今日は、ここを説明します

index

/books

GET

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

edit

/books/:id/edit

GET

update

/books/:id

PUT

削除 新規入力

destroy

/books/:id
DELETE

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

BooksController のcreate アクション

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)

BooksController のcreate アクション

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)
  ↑ BookクラスはModelという種類に属しています。
  respond_to do |format|
    if @book.save
      format.html ↑ここで保存しています。 1. リクエストのパラメータを使って
      ↑redirect_to book, notice: 'Book was successfully created.' } 2. 本のデータを保存する
      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
```

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、モデルの仕事について
見ていきましょう。

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。

実習 rails console でモデルを使ってみる

RailsにはRubyでいうirbのような、1行ずつコードを実行する機能があります。
前準備でつくったアプリへ移動して、`rails console`を使ってみましょう。

```
$ cd books_app          ← Rails Root フォルダへ移動  
$ bundle exec rails c    ← cはconsoleの略  
1.9.3-pXXX :001 >      ← ここにコードを打ちます。
```

以下のコードを実行してみてください。

```
book = Book.new(title: "some title", memo: "some memo")
```

↑文字列はなんでもOK

<code>book.save</code>	←保存します
<code>Book.last</code>	←最後のデータを取得します(上で保存した値の確認)

また、ここで保存したデータはブラウザからも見ることができます。

`rails`サーバを起動

```
$ bundle exec rails s
```

ブラウザでアクセス

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

※これができた人は、前で説明した `where` を使って検索もしてみてください。

次は、モデルのコードを見て、
モデルの仕組みを解説していきます。

モデルのコードは `app/models/` 以下にあります。

Book モデルのコード

app/models/book.rb

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

Bookモデルにはコードがほとんどありません。

では、`save` や `all` といったメソッドが使えるのはなぜでしょうか？
また、`title` や `memo` といった要素があることをどこで知るのでしょうか？

Book モデルのコード

問. `save` や `all` といったメソッドが使えるのはなぜでしょうか？

`app/models/book.rb`

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

答え. `ActiveRecord::Base` クラスを継承しているから
`ActiveRecord::Base` クラスがモデルの仕事に必要な機能を持っていま
す。それを継承している `Book` クラスも同じ機能を持つことができます。

Book モデルのコード

問. title や memo といった要素があることをどこで知るのでしょうか？

答え. データベースから情報を得ます

ActiveRecord::Base はデータベースから情報を得て、
Bookモデルに title, memo という要素があることを知っています。
(book.title や book.memo というメソッドを提供します。)

データベースとは何でしょうか？

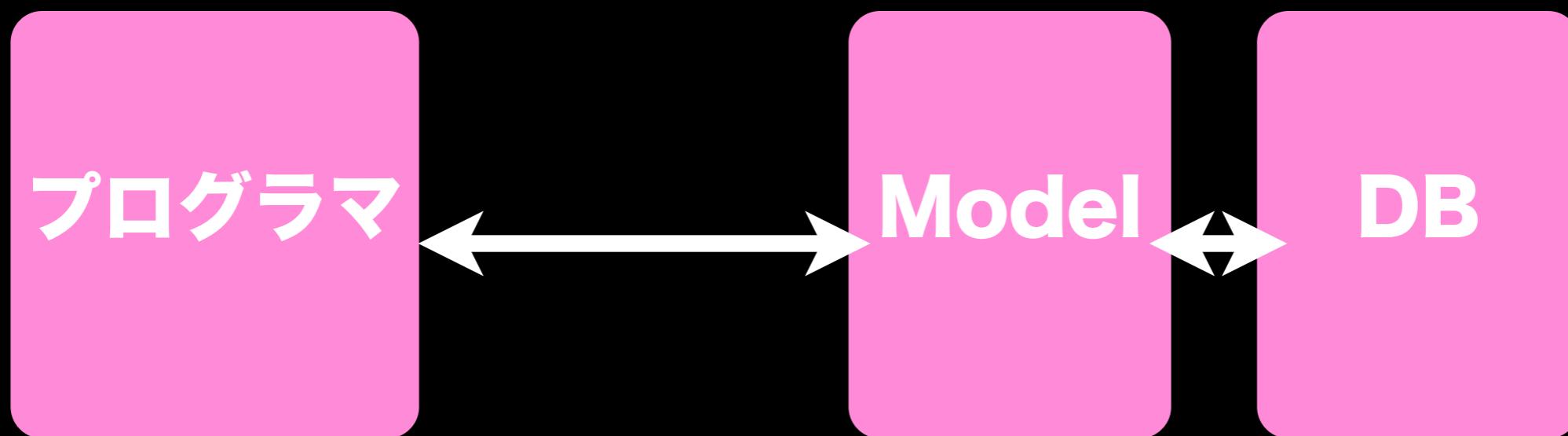
データベース(DB)とは？

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。

では、DBはいつのまに
作られたのでしょうか？

実はここで作ってました。

```
$ 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
```

scaffoldはモデルとDB設計図などを作ります。

```
$ 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
```

作られるファイルの中で関係するもの

```
create    test/fixtures/books.yml
```

```
invoke  resource_route
```

モデル

```
model   resources :books
```

```
invoke  scaffold_controller
```

▶ **app/models/book.rb**

DB設計図

▶ **db/migrate/20121122005122_create_books.rb**

```
create   app/views/books/edit.html.erb
```

```
create   app/views/books/show.html.erb
```

DB設計図とはどんなものか、見てみましょう。

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
```

create_table :books で books という名前のテーブルを作ります。
DBはテーブルという単位でデータを管理します。このアプリの本に関する
データを保存するために、books という名前のテーブルを作っています。
※テーブル名はモデル名の複数形にするという原則があります。

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
```

booksテーブルはtitleという要素と、 memo という要素を持ちます。

(この要素のことをDB用語でカラムといいます。)

string はデータの型の1つです。 文字列を格納します。

text もデータの型の1つで(stringよりも多く)文字列を格納します。

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
```

また、`created_at`(作成日時)や`updated_at`(更新日時)を記録する要素を持ちます。`(t.timestampsが作成します。)`

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

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
```

ところで下線部の部分、見覚えがありませんか？

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
```

実はscaffoldで指定していたのはこの部分です。

```
$ bundle exec rails g scaffold books
```

```
title:string memo:text
```

DB設計図 - migration

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

scaffold で指定していたのはテーブル名とカラム、データの型でした。

string はデータの型です。文字列を格納できます。

text もデータの型で(stringよりも多く)文字列を格納できます。

```
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
```

scaffoldはDB設計図(migration)を作ることを説明しました

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

↓ ファイル生成

db/migrate/20121122005122_create_books.rb

※ファイル名先頭の20121122005122は他のファイルと重複しないようにRailsが自動でつけます。

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

```
  def change
```

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

```
      t.string :title
```

```
      t.text :memo
```

```
      t.timestamps
```

```
    end
```

```
  end
```

```
end
```

scaffoldはDB設計図(migration)を作ることを説明しました

```
$ 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
```

では、migrationから
実際にDBを作るには
どうすれば
いいでしょう？

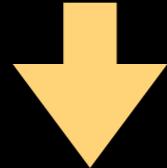


???

DB

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

では、 scaffoldで作られる
model、 migrationの
まとめです。

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。

応用編：

**既存のDB(Model)に
カラムを増やすには？**

既存のDBテーブル(Model)にカラムを増やすには?

rails g コマンドに**migration**を指定すると**migration**ファイルだけを生成できます。

たとえば、**books**テーブルに**string**型の**author**を加えるには以下のようにします。

```
$ bundle exec rails g migration AddAuthorToBooks author:string
```

結果

```
invoke  active_record
create    db/migrate/20131126231116_add_author_to_books.rb
```

rails g migrationコマンドの基本形は以下になります。

```
$ bundle exec rails g migration Addカラム名Toテーブル名 カラム名:型名
```

既存のDBテーブル(Model)にカラムを増やすには?

rails g コマンドにmigrationを指定するとmigrationファイルだけを生成できます。

たとえば、booksテーブルにstring型のauthorを加えるには以下のようにします。

```
$ bundle exec rails g migration AddAuthorToBooks author:string
```

↓ migrationファイル生成

```
db/migrate/20121206124543_add_author_to_books.rb
```

```
class AddAuthorToBooks < ActiveRecord::Migration
  def change
    add_column :books, :author, :string
  end
end
```

←生成されたmigrationファイルには
booksテーブルへauthorをstring型で追加
する指示が書かれている

↓ \$ bundle exec rake db:migrate
migrationからDBを作る

DB

booksテーブル

title, memo, author, created_at, updated_at

ちなみに、Railsのgenerate機能は開発をアシストする機能なので、使わないので0から手でコードを書いても同様に動きます。たとえば、以下の方法Aと方法B、どちらも同じように動作します。

方法A: rails g を使う方法

```
$ bundle exec rails g migration AddAuthorToBooks author:string
```

↓ migrationファイル生成

```
db/migrate/20121206124543_add_author_to_books.rb
class AddAuthorToBooks < ActiveRecord::Migration
  def change
    add_column :books, :author, :string
  end
end
```

方法B: 0から手でコードを書く方法

```
db/migrate/20121206124543_add_author_to_books.rb
class AddAuthorToBooks < ActiveRecord::Migration
  def change
    add_column :books, :author, :string
  end
end
```

エディタでこのファイル名で
←この内容を書く

ほかのrails gコマンド、scaffold や controllerなども全て同じことが言えます。

応用編： 新しいモデルとmigrationを 作るには？

Modelとmigrationを生成するには

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

rails g コマンドに**model**を指定すると**model**と**migration**を生成できます。

db/migrate/20121208060032_create_books.rb
app/models/books.rb

rails g あれこれ

migration だけ

\$ bundle exec rails g migration AddAuthorToBooks author:string

model + migration

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

routes + controller + view

\$ bundle exec rails g controller books index

scaffold =

model + migration + routes + controller + view

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

応用編：

**scaffoldでつくった
Migration, Model, Controllerへ
カラムを追加するには？**

booksテーブルにstring型のauthorを加え、 ブラウザから入力できるようにしてみましょう

1. booksテーブルにstring型のauthorを加えます。

```
$ bundle exec rails g migration AddAuthorToBooks author:string
```

結果

```
invoke  active_record
create    db/migrate/20131126231116_add_author_to_books.rb
```

2. migrationからDBを作る

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

結果

```
--> 0.0028s
==  AddAuthorToBooks: migrated (0.0028s) ==
```

3. viewを修正(1)

app/views/books/_form.html.erb (new, edit 入力画面の部品)

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

修正後はこんな感じになります→

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

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

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

既にある↑のコードの下に以下のコードを追記します。

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

app/views/books/show.html.erb (show 詳細表示画面)

```
<p>
  <strong>Memo:</strong>
  <%= @book.memo %>
</p>
```

修正後はこんな感じになります→

```
<p>
  <strong>Memo:</strong>
  <%= @book.memo %>
</p>

<p>
  <strong>Author:</strong>
  <%= @book.author %>
</p>

<%= link_to 'Edit', edit_book_path(@book) %> |
<%= link_to 'Back', books_path %>
```

既にある↑のコードの下に以下のコードを追記します。

```
<p>
  <strong>Author:</strong>
  <%= @book.author %>
</p>
```

3. viewを修正(2)

app/views/books/index.html.erb (index 一覧表示画面)

```
<th>Title</th>
```

```
<th>Memo</th>
```

既にある↑のコードの下に以下のコードを追記します。

```
<th>Author</th>
```

```
<td><%= book.title %></td>
```

```
<td><%= book.memo %></td>
```

既にある↑のコードの下に以下のコードを追記します。

```
<td><%= book.author %></td>
```

修正後はこんな感じになります

(略)

```
<table>
```

```
<thead>
```

```
<tr>
```

```
<th>Title</th>
```

```
<th>Memo</th>
```

```
<th>Author</th>
```

```
<th></th>
```

```
<th></th>
```

```
<th></th>
```

```
</tr>
```

```
</thead>
```

```
<tbody>
```

```
<% @books.each do |book| %>
```

```
<tr>
```

```
<td><%= book.title %></td>
```

```
<td><%= book.memo %></td>
```

```
<td><%= book.author %></td>
```

```
<td><%= link_to 'Show', book %></td>
```

```
<td><%= link_to 'Edit', edit_book_path(book) %></td>
```

```
<td><%= link_to 'Destroy', book, method: :delete, data: { confirm: 'Are you sure?' } %></td>
```

```
</tr>
```

```
<% end %>
```

```
</tbody>
```

```
</table>
```

(略)

4. controllerを修正

app/controllers/books_controller.rb

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

上記の `permit` メソッドの引数に `:author` を追加します。修正後は以下のようになります。

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

5. rails serverを起動して動作を確認

\$ bundle exec rails s

New book

Title

Memo

Author

[Create Book](#)

[Back](#)

Title: ちはやふる

Memo: 热血竞技かるた漫画

Author: 末次由纪

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

Listing books		
Title	Memo	Author
ちはやふる	热血竞技かるた漫画	末次由纪

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

各画面にAuthor欄が追加されて、登録できるようになりました。

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

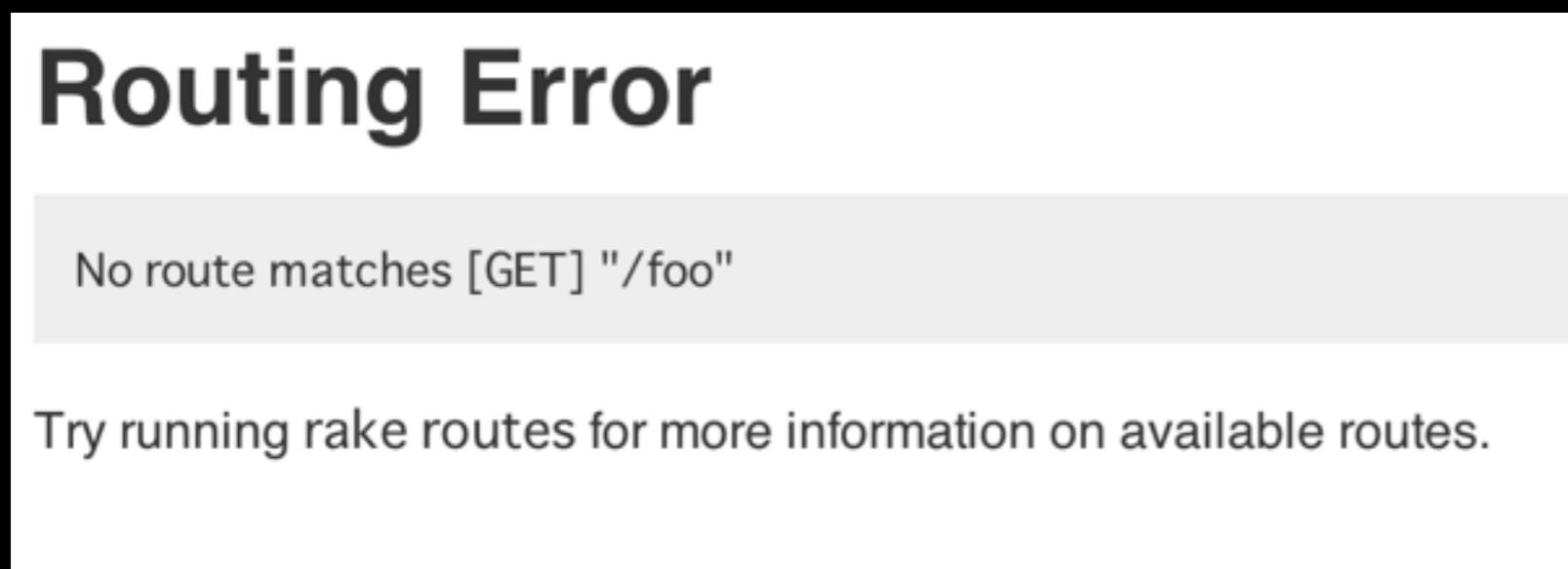
ブラウザに表示されるエラーを読む

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。

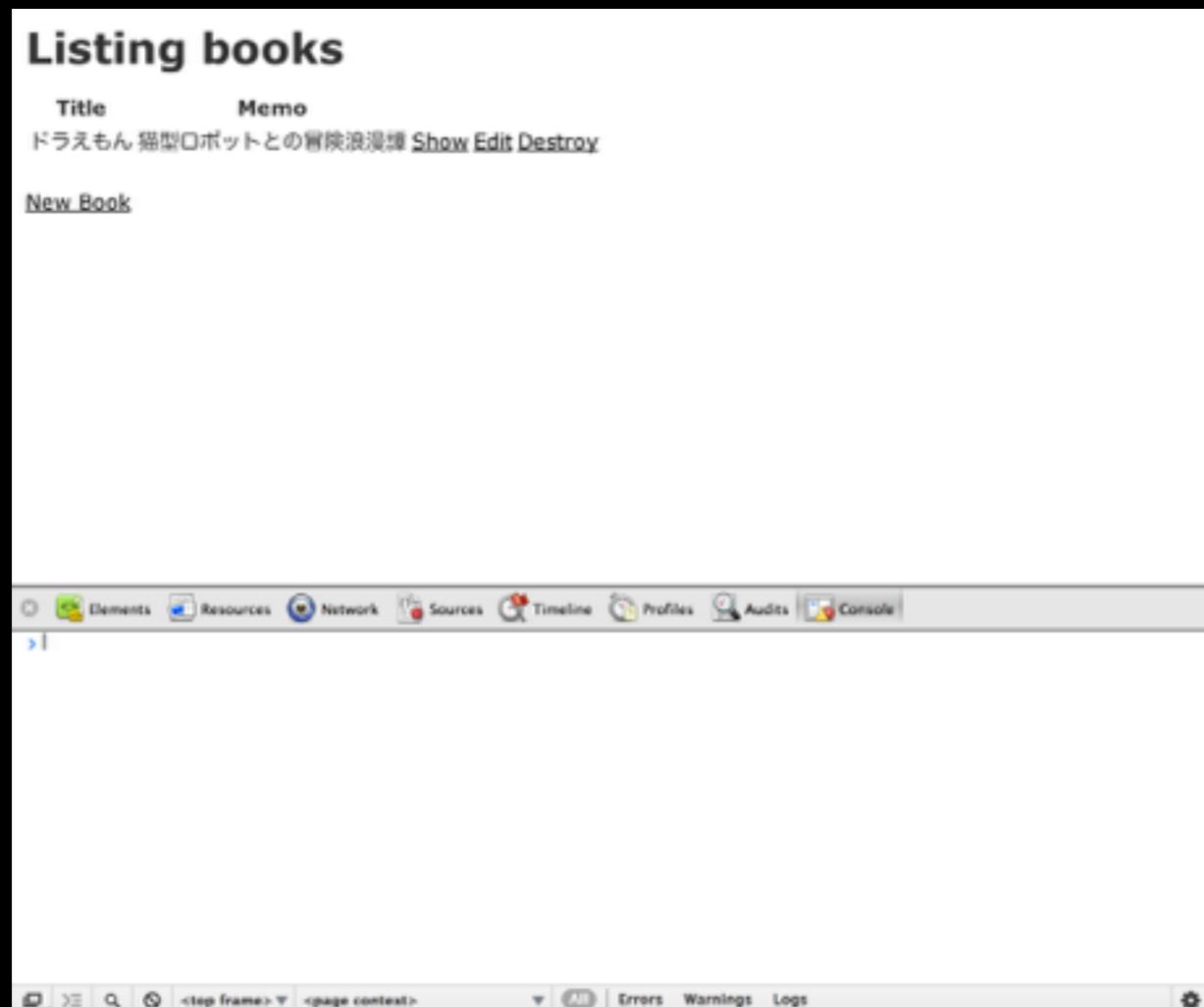
```
class BooksController < ApplicationController
  def index
    @books = Book.all
    Rails.logger.error '*****'
    Rails.logger.error @books
  ...
  end
end
```

← 目印
←@booksの中身を表示

Chromeでデバッグ

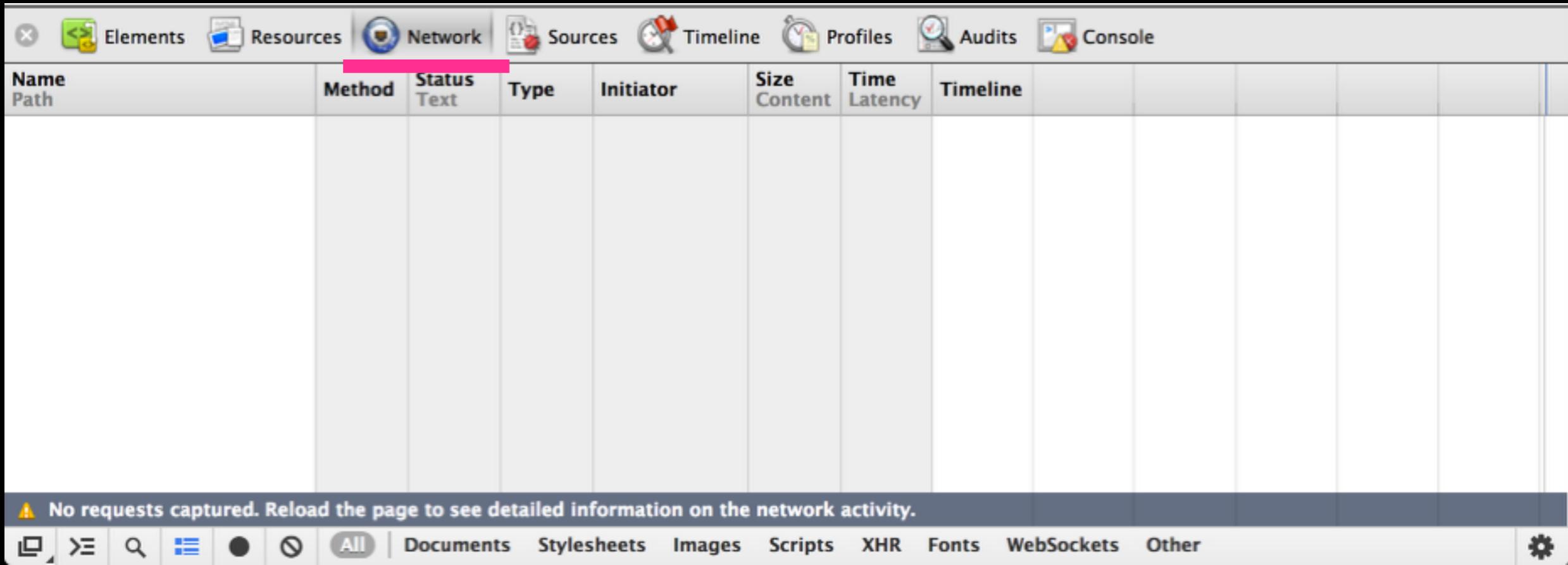
Chrome の デベロッパーツールを使うとリクエストの情報が表示できたりして便利です。

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 developer tools. It lists several requests made by the browser:

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	<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 timeline, there is a red 'All' button and a clear icon (a circle with a minus sign).

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	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

▶ 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