

Ruby on Rails 講義

第26回

コードを書くための道具箱

Kuniaki IGARASHI/igaiga

2012.12.20 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



コードを 書くための 道具箱



バージョン管理システム



thanks to ryopeko, the photo is <http://www.flickr.com/photos/martindale/59445824/>

こんなことありませんか？

あれ、コードを書き換えたら
動かなくなっちゃった・・・

動いてたときへ戻したいけど
分かなくなっちゃった・・・

そんなときに役に立つのが
バージョン管理システム
です。

もしも、こんな風に履歴が残っていて

book.rb 2012.12.15 14:10
authorメソッド作った

book.rb 2012.12.15 14:34
メモ機能を追加した

book.rb 2012.12.15 15:02
現在。素晴らしい機能を実装中。

時間の流れ

うまく動かなくなっちゃった・・・ってときに

■ book.rb 2012.12.15 14:10
authorメソッド作った

■ book.rb 2012.12.15 14:34
メモ機能を追加した

■ book.rb 2012.12.15 15:05
現在。素晴らしい機能は失敗に終わった。

時間の流れ

過去のある時点へ戻れたら便利ですよね？

book.rb 2012.12.15 14:10
authorメソッド作った

book.rb 2012.12.15 14:34
メモ機能を追加した

book.rb 2012.12.15 15:05
現在。素晴らしい機能は失敗に終わった。

ここへ
戻りたい

時間の流れ

バージョン管理システム
なら、できちゃいます！

では、使い方を見ていきましょう。

バージョン管理システム
にはいくつか種類があるのでですが、
今回は **git** という
バージョン管理システムを使います。

Rubyistがよく使うバージョン管理システムです。
Linuxを作ったLinusさんが作者です。

※gitとRubyとは特に関係はありません。
独立して別々に使われます。

git インストール確認

git はshellでコマンドとして使います。

Win, Mac

RailsInstaller でインストールした人はすでにgitがインストールされています。

shell で以下のコマンドを打ってバージョンが表示されればインストール済みです。

\$ git version

git version 1.7.5.2

※1.7.5.2の部分は異なります。

※Macの人でインストールされていない場合は、
以下からDLしてインストールしてください。

<http://code.google.com/p/git-osx-installer/>

git 設定確認

git は使用者の名前とメールアドレスを履歴に残します。
以下のコマンドで設定を確認できます。

\$git config --global user.name

Kuniaki IGARASHI

\$git config --global user.email

igaiga@gmail.com

設定ができない場合は以下で設定できます。

git config --global user.name "Kuniaki IGARASHI"

git config --global user.email igaiga@gmail.com

※これらの情報はPC内に保存されます。公開されることは今回の使い方だとありませんが、心配な場合は仮のものでも大丈夫です。

git 初期化

git はフォルダ単位で履歴（バージョン）を管理します。

試しに、新しいフォルダを作ってそこをgitでバージョン管理してみましょう。

前準備

```
$ mkdir フォルダ名
```

```
$ cd フォルダ名
```

フォルダを新規作成します。
作ったフォルダへ移動します。

git 初期化

```
$ git init
```

Initialized empty Git repository in フォルダ名

git init すると、そのフォルダをバージョン管理できるようになります。

※豆知識：バージョン管理されている場所を「リポジトリ」と呼びます。

履歴のポイントを作る

歴史に残したい状態でポイントを作ります。

このポイントを**コミット**と呼びます。

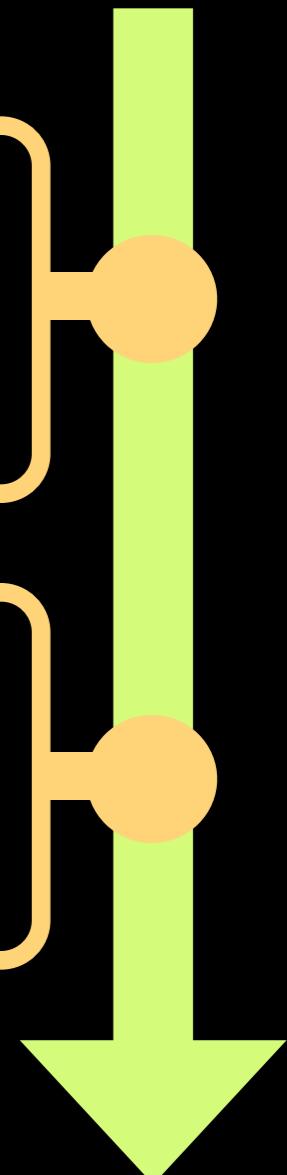
さきほどの図で言う●の部分です。

book.rb 2012.12.15 14:10

authorメソッド作った

book.rb 2012.12.15 14:34

メモ機能を追加した



コミットの方法

コミットしてみましょう。

前準備

エディタでファイルを作って、git init したフォルダへ保存しましょう。

hello.rb

puts "Hello world!"

ファイル名もファイルの内容も
自由でかまいません。
RubyのコードでなくてもOKです。

git コミット

\$ git add hello.rb

\$ git commit -m "hello.rb を追加"

hello.rb の部分はファイル名
"hello.rb を追加"の部分は
コメントです。自由に書けます。

```
[master (root-commit) ebcb9df] hello.rb を追加
```

```
1 files changed, 1 insertions(+), 0 deletions(-)
create mode 100644 hello.rb
```

コミットするには、git add した後で、git commit します。

履歴を見る

履歴を見てみましょう。先ほど作ったコミットがあるはずです。

履歴を見る

\$ git log

```
commit ebeb9df1625ef74856a2724cca54e87abf72dfbc
Author: Kuniaki IGARASHI <igaiga@gmail.com>
Date:   Sat Dec 15 21:44:22 2012 +0900
```

hello.rb を追加

さきほどのコミットがありました。
なにやら色々と書いてあります。（次のページへ続く）

履歴の見方

```
$ git log
```

```
commit ebcb9df1625ef74856a2724cca54e87abf72dfbc ←コミットハッシュ  
Author: Kuniaki IGARASHI <igaiga@gmail.com> ←コミットした人  
Date: Sat Dec 15 21:44:22 2012 +0900 ←コミットした日時
```

```
hello.rb を追加 ←コミットメッセージ
```

各種情報が記録されています。

コミットハッシュとはナニモノでしょうか？

コミットハッシュはそのコミットを一意に示す英数字です。

このコミットの名前と言ってもいいでしょう。

このコミットを指すときは ebcb9d... と指定します。

もう1つコミットを作ってみましょう

前準備

エディタでさっきのファイルを変更して保存しましょう。

hello.rb

```
puts "foo"
```

git コミット

```
$ git add hello.rb
```

```
$ git commit -m "puts文を変更"
```

hello.rb の部分はファイル名
"puts文を変更"の部分は
コメントです。自由に書けます。

```
[master 6ba048b] puts文を変更
```

```
1 files changed, 1 insertions(+), 1 deletions(-)
```

コミットしたらgit log を見てみましょう。(つづく)

履歴を確認してみましょう

\$ git log

```
commit 6ba048b2967d1c04f271e12988e43f19f3f65def ←新しいコミット  
Author: Kuniaki IGARASHI <igaiga@gmail.com>  
Date: Sun Dec 16 09:56:42 2012 +0900  
が増えました
```

puts文を変更

```
commit ebeb9df1625ef74856a2724cca54e87abf72dfbc  
Author: Kuniaki IGARASHI <igaiga@gmail.com>  
Date: Sat Dec 15 21:44:22 2012 +0900
```

hello.rb を追加

このようにコミットが積み上がっていきます。
変更の内容をもう少し詳しく見るにはどうすればいいでしょうか。 (つづく)

履歴を詳しく見てみましょう

\$ git log -p

```
commit 6ba048b2967d1c04f271e12988e43f19f3f65def
Author: Kuniaki IGARASHI <iigaiga@gmail.com>
Date:   Sun Dec 16 09:56:42 2012 +0900
```

puts文を変更

```
diff --git a/hello.rb b/hello.rb  ↪hello.rb が変更された
index 08b85bf..a32c710 100644
--- a/hello.rb
+++ b/hello.rb
@@ -1 +1 @@
-puts "Hello world!" ← - から始まる行は削除されたことを意味します。
+puts "foo"           ← + から始まる行は追加されたことを意味します。
```

```
commit ebeb9df1625ef74856a2724cca54e87abf72dfbc
Author: Kuniaki IGARASHI <iigaiga@gmail.com>
Date:   Sat Dec 15 21:44:22 2012 +0900
```

コミットで変わった差分(diff)が表示されます。何を変えたか調べたいときに便利です。
また、以下の設定文を打つと結果がカラーになって見易いです。(1回だけ打てばずっと有効)

\$git config --global color.ui true

gitの現在状態を確認する

git 状態確認

```
$ git status
```

ファイルに変更がないとき

```
# On branch master  
nothing to commit (working directory clean)
```

ファイルが変更されているとき

```
# On branch master  
# Changes not staged for commit:  
#   (use "git add <file>..." to update what will be committed)  
#   (use "git checkout -- <file>..." to discard changes in  
working directory)  
#  
#       modified:   hello.rb  ←変更されているファイル名  
#  
no changes added to commit (use "git add" and/or "git commit -a")
```

gitの現在状態を確認する(つづき)

git 状態確認

```
$ git status
```

過去にコミットしていないファイルが存在するとき

```
# On branch master
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       iga.rb ←過去にコミットされていないけど存在してるファイル名
nothing added to commit but untracked files present (use "git add"
to track)
```

歴史を巻き戻す

過去のコミットまで戻してみましょう。

\$ git checkout [コミットハッシュ]

コミットハッシュは
git log で調べておきます。

例) **\$ git checkout ebbe9d**

コミットハッシュは全部打たなくても
一意に決まる長さで大丈夫です。

Note: checking out 'ebbe9d'.

You are in 'detached HEAD' state. You can look around, make experimental changes and commit them, and you can discard any commits you make in this state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may do so (now or later) by using -b with the checkout command again. Example:

```
git checkout -b new_branch_name
```

HEAD is now at ebbe9df... hello.rb を追加

hello.rb をエディタで開くと、ebbe9d 時点の状態に戻ってることが分かります。
puts "Hello world!"

巻き戻した歴史を元に戻す

元に戻すには `checkout` で `master` を指定します。
※`hello.rb`のファイルも最新に戻るので、必要ならば別の名前でコピーしておきます。

```
$ git checkout master
```

```
Previous HEAD position was ebeb9df... hello.rb を追加  
Switched to branch 'master'
```

`hello.rb` をエディタで開くと、最新のコミット時点の状態に戻っています。

```
puts "foo"
```

參考：git 練習場 <http://try.github.com>



1.1 • Got 15 minutes and want to learn Git?

Git allows groups of people to work on the same documents (often code) at the same time, and without stepping on each other's toes. It's a distributed version control system.

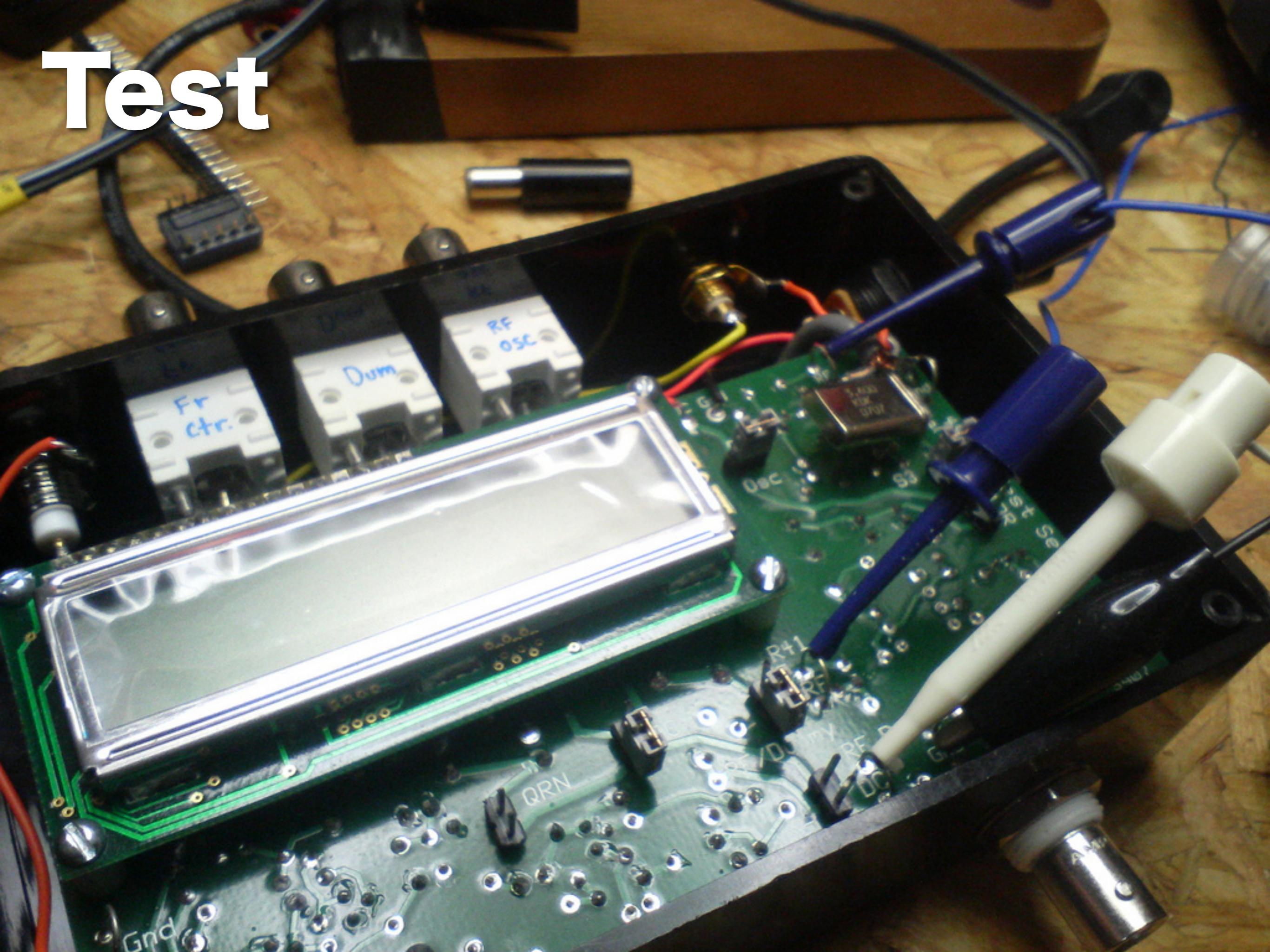
Our terminal prompt below is currently in an **octobox** directory. To initialize a Git repository here, type the following command:



➔ **git init**

```
Press enter to submit commands
$
```

Test



ぶっちゃけ、ブラウザからぽちぽち動作確認するの、面倒なんですね。rails s も打たないといけないし。

そこで、動作確認(Test)もプログラムにやらせてしまうことにします。開発を速く巧く進める技です。

RSpec

プログラムでテストを書く仕組みの1つがRSpecです。
RSpecはGemになっているので簡単に使い始めることができます。

RailsアプリでRSpecを書く方法を見てみましょう。
例によっていつものアプリを作ります。
(既にできているアプリを使ってもOK)

前準備

```
$ rails new books_app
```

```
$ cd books_app
```

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

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

次の行も加えて実行してください。 (既にあるアプリを使う場合もこれだけ実行してください。)

```
$ RAILS_ENV=test bundle exec rake db:migrate
```

RSpec インストール

Gemfileに以下を追加します。
(たとえばsqlite3の下)

```
gem 'rspec-rails'
```

bundle コマンドで変更したGemfileを反映させます。

```
$ bundle
```

Rspecをインストールします。
shellで以下のコマンドを打ちます。

```
$ bundle exec rails g rspec:install
```

RSpec プロダクトコード実装

たとえばBookモデルが以下のようになっていて、`deco_title`というタイトルをデコる素敵なメソッドを実装していたとしましょう。

`app/models/book.rb`

```
class Book < ActiveRecord::Base
  attr_accessible :memo, :title
  def deco_title
    "*** " + title + " ***"
  end
end
```

たとえば、`book = Book.new(:title => "3月のライオン")` のとき
`book.deco_title` と実行して `"*** 3月のライオン ***"` を得たいわけです。
テストなしだとうまく動くか不安で夜も眠れません。
テストコードを書いてみましょう！

※このBookクラスのように、アプリで実際の動作に使われるコードをプロダクトコードと読んだりもします。私は本番コードと読んじゅうことが多いです。

RSpec テストコード実装

テストコードは以下のように書けます。

spec/book_spec.rb

```
# -*- coding: utf-8 -*-
```

```
require 'spec_helper'
```

```
describe "Book" do ←Bookクラスの
```

```
  describe "#deco_title" do ←deco_titleメソッドについて(#はインスタンスメソッドの意)
```

```
    it "*** *** で囲まれたタイトルを返す" do ←こう動作すべき
```

```
      book = Book.new(:title => "3月のライオン") ←前準備
```

```
      book.deco_title.should == "*** 3月のライオン ***" ←テストコード コア部分
```

```
    end
```

```
  end
```

```
end
```

ポイントはこのコードです。 "should" がテストのためのメソッド。

book.deco_title.should == "* 3月のライオン ***"**

「book.deco_title は "*** 3月のライオン ***" と == (同じ) になるべき」
というテストコードです。

book.deco_title の結果が "*** 3月のライオン ***" の場合にテストを通過します。
異なる場合やうまく動かない場合はテストが失敗します。

RSpec テストコード実行

さきほど書いたテストコードを実行してみましょう。
実行には rspec コマンドを使い、
実行したいテストコードを指定します。

```
$ rspec spec/book_spec.rb
```

```
$ rspec spec/book_spec.rb
```

```
.
```

```
Finished in 0.0124 seconds
1 example, 0 failures
```

```
Randomized with seed 23837
```

テストが無事に通過すると、
こんな感じのスッキリしたグリーンの表示になります。

RSpec テストコード実行

```
$ rspec spec/book_spec.rb
```

```
$ rspec spec/book_spec.rb
F
```

Failures:

```
1) Book#deco_title *** *** で囲まれたタイトルを返す
Failure/Error: book.deco_title.should == "*** 3月のライオン ***"
TypeError:
  can't convert nil into String
# ./app/models/book.rb:4:in `+'
# ./app/models/book.rb:4:in `deco_title'
# ./spec/book_spec.rb:8:in `block (3 levels) in <top (required)>'
```

```
Finished in 0.01352 seconds
1 example, 1 failure
```

Failed examples:

```
rspec ./spec/book_spec.rb:6 # Book#deco_title *** *** で囲まれたタイトルを返す
Randomized with seed 38530
```

なにか問題があるとこんな感じの残念なレッドの表示になります。
プロダクトコード、もしくはテストコードを見直しましょう。

Cosmo

A preview of changes in this swatch.

[Typography](#) [Navbar](#) [Buttons](#) [Forms](#)

Cosmo

An ode to Metro.

[Preview](#)

[Download](#)



Twitter Bootstrap

[Bootswatch](#) [News](#) [Gallery](#)

Readable

A preview of changes in this swatch.

[Typography](#) [Navbar](#) [Buttons](#)

Readable

Optimized for legibility.

[Preview](#)

[Download](#)



Cyborg

A preview of changes in this swatch.

[Typography](#) [Navbar](#) [Buttons](#) [Forms](#)

Cyborg

Jet black and electric blue.

[Preview](#)

[Download](#)



Bootswatch

[Bootswatch](#) [News](#) [Gallery](#) [Preview](#)

Simplex

A preview of changes in this swatch.

[Typography](#) [Navbar](#) [Buttons](#) [Forms](#) [Misc](#)

Simplex

Minimal and minimalist.

[Preview](#)

[Download](#)



Journal

A preview of changes in this swatch.

[Typography](#) [Navbar](#) [Buttons](#) [Forms](#)

Journal

Crisp like a new sheet of paper.

[Preview](#)

[Download](#)



Bootswatch

[Bootswatch](#) [News](#) [Gallery](#) [Preview](#)

Slate

A preview of changes in this swatch.

[Typography](#) [Navbar](#) [Buttons](#) [Forms](#) [Misc](#)

Slate

Shades of gunmetal gray.

[Preview](#)

[Download](#)



Twitter Bootstrapは
なんかええ感じの
デザインにしてくれる
お手軽ツールです。

前準備

例によっていつものscaffoldで作るアプリを題材にします。
前に作ったものがあればそれをそのまま使ってもOKです。

```
$ rails new books_app
```

```
$ cd books_app
```

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

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

Listing books

Title

Memo

路地恋花 京都の路地を舞台にした恋愛短編集

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

ちはやふる 高校の競技カルタ部を描く火花散らす青春

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

[New Book](#)

地味～

New book

Title

路地恋花

Memo

京都の路地を舞台にした恋愛短編集

地味～

Create Book

Back

そこで Twitter Bootstrap

<http://twitter.github.com/bootstrap/>

Twitter Bootstrap インストール

Gemfile を2箇所編集します。

1. group :assets do ... end の中に以下を追加で書いてください。
(例えば group :assets do の下の行とか)

```
gem 'twitter-bootstrap-rails'  
gem 'less-rails'
```

2. 以下のコメント行先頭の#を削って有効にしてください。
(環境によって platforms の部分が異なるかもしれません。)

```
# See https://github.com/sstephenson/execjs#readme for more supported runtimes  
# gem 'therubyracer', :platforms => :ruby
```

↑コレ削除



```
# See https://github.com/sstephenson/execjs#readme for more supported runtimes  
gem 'therubyracer', :platforms => :ruby
```

編集したらbundleコマンドで変更を反映させます。

```
$ bundle
```

Twitter Bootstrap インストール

以下の2つのコマンドを打って必要なファイルを生成します。

```
$ bundle exec rails g bootstrap:install
```

```
$ bundle exec rails g bootstrap:layout application fluid
```

※以下のように上書きするか聞かれるのでYと答えて上書きします。

```
conflict app/views/layouts/application.html.erb  
Overwrite app/views/layouts/application.html.erb?  
(enter "h" for help) [Ynaqdh]
```

Twitter Bootstrap 動作確認

では、動作確認してみましょう。

いつも通りrails s を起動してブラウザでアクセスしてみてください。

```
$ bundle exec rails s
```

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



Listing books

Title	Memo	Show	Edit	Destroy
路地恋花	京都の路地を舞台にした恋愛短編集	Show	Edit	Destroy
ちはやふる	高校の競技カルタ部を描く火花散らす青春	Show	Edit	Destroy

New Book

Sidebar

SIDE BAR

Link1

Link2

Link3

© Company 2012

ちょっといい感じ～



New book

Title

路地恋花

Memo

京都の路地を舞台にした恋愛短
編集

Sidebar

SIDE BAR

Link1

Link2

Link3

ちょっといい感じ～

Twitter Bootstrap More

もっと使ってみたい方は、
ドットインストールの解説動画がお勧めです。

http://dotinstall.com/lessons/basic_twitter_bootstrap_v3

付録

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。

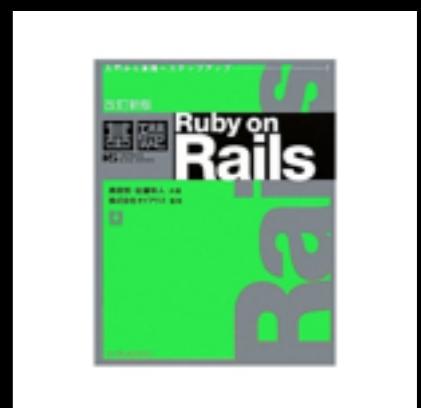
http://guides.rubyonrails.org/getting_started.html

(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

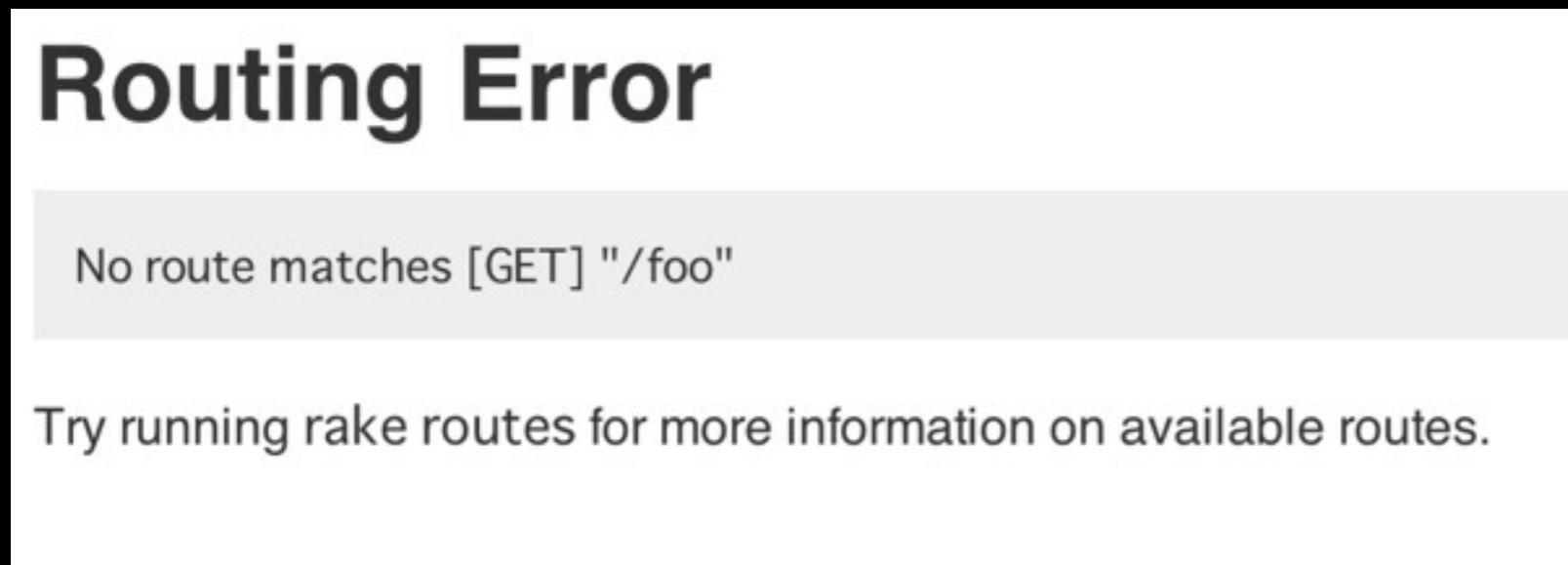
ブラウザに表示されるエラーを読む

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。

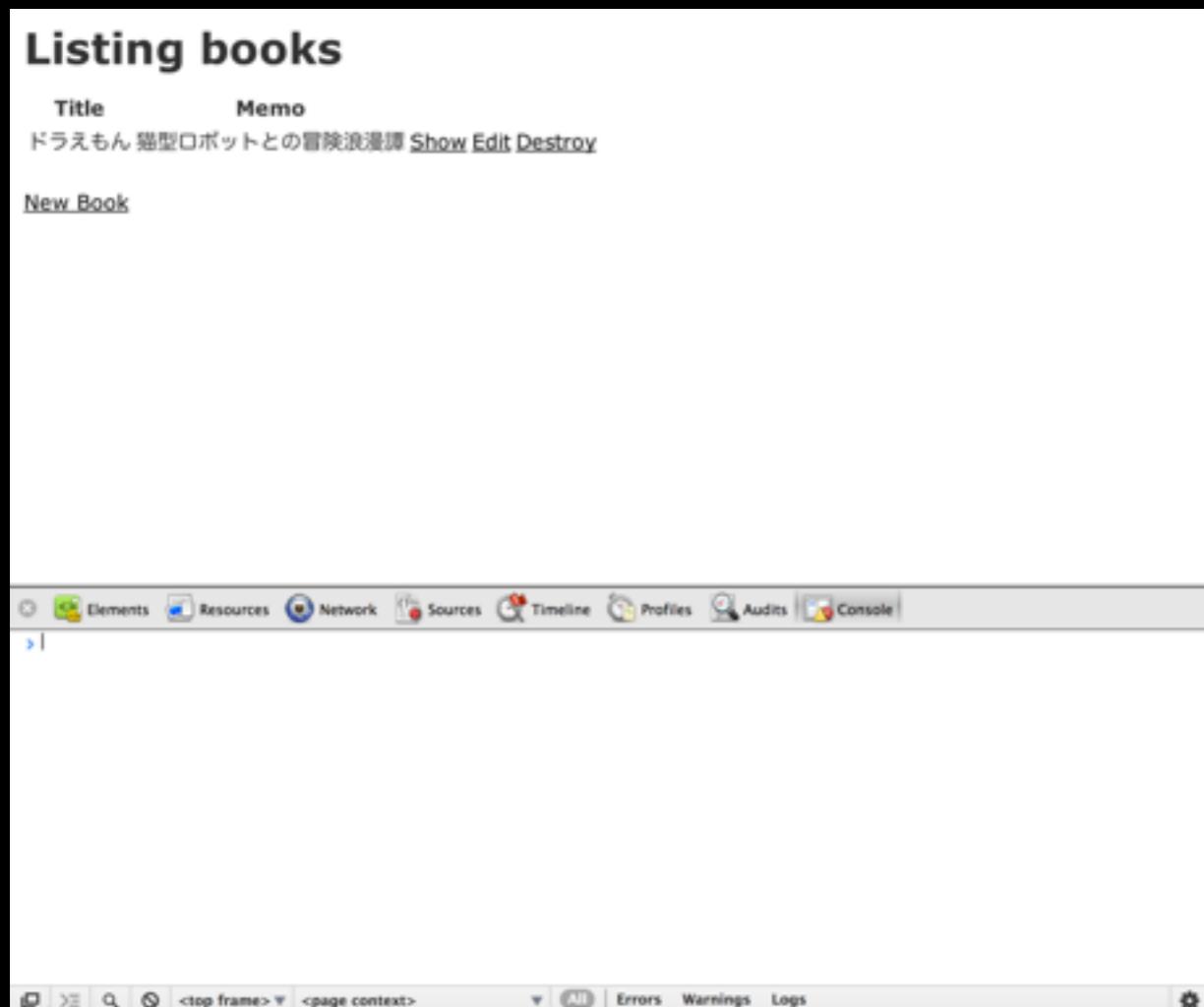
```
class BooksController < ApplicationController
  def index
    @books = Book.all
    Rails.logger.error '*****'
    Rails.logger.error @books
  ...
  end
end
```

← 目印
←@booksの中身を表示

Chromeでデバッグ

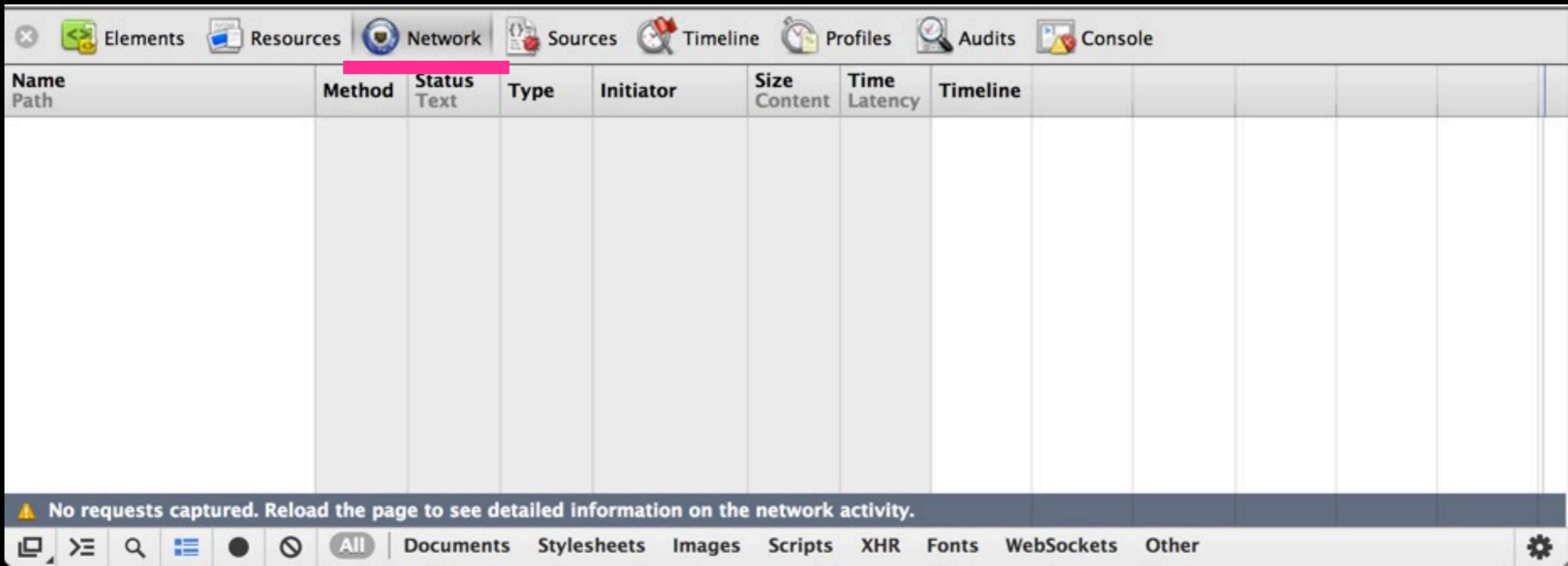
Chrome の デベロッパーツールを使うとリクエストの情報が表示できたりして便利です。

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 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	<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, there are filter buttons: All, Documents, Stylesheets, Images, Scripts, XHR, Fonts, WebSockets, Other, and a clear button (X) with a count of 1.

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: Accept, Accept-Charset, Accept-Encoding, Accept-Language, Cache-Control, Connection, and Cookie. The cookie value is partially visible as a long string of characters.

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

□ ⌂ ⌂ ⌂ 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 Network tab interface. On the left, a tree view lists resources: books (HTML), scaffolds.css (CSS), books.css (CSS), books.js (JS), jquery.js (JS), and application.css (CSS). The 'books' resource is expanded. On the right, the Response tab is selected, displaying the HTML source code of the page. The code includes the DOCTYPE, HTML structure, head content (title, links to CSS and JS files, meta tags for csrf token), and body content (h1 and table tags). The code is numbered from 1 to 19.

Name	Path	Content
books		<!DOCTYPE html> <html> <head> <title>BooksApp</title> <link href="/assets/application.css?body=1" media="all" rel="stylesheet" type="text/css"> <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-token" /> </head> <body> <h1>Listing books</h1> <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