Ruby 初級者向けレッスン第 23 回

okkez @ Ruby 関西

2008 年 08 月 16 日

今回の内容

- Rack 入門

今回のゴール

- Rack で簡単な web アプリケーションを作成する

Rack とは

Rack: a Ruby Webserver Interface

http://rack.rubyforge.org/

Rack provides an minimal interface between webservers supporting Ruby and Ruby frameworks.

Rack はウェブサーバの Ruby サポートと Ruby で書かれたフレームワークの間に最小限のインターフェイスを提供する。(意訳)

Rack の目的とか動機とか

以下にわかりやすくまとめられているので未読の人は読むべし。

Greenbear Diary - 5 分でわかる Rack，シュレーディンガーの猫たち

http://mono.kmc.gr.jp/~yhara/d/?date=20080716

まとめると、「要チェックや！」ということです。

インストール

以下で OK です。

```
$ sudo gem install rack
```

確認は

```
$ rackup --version
```

とかでできます。

Hello, Rack!

以下のファイルを用意します。
hello.rb

```
01: require 'rubygems'
02: require 'rack'
03:
04: class Hello
05:   def call(env)
06:     Rack::Response.new{|res|
07:       res['Content-Type']= 'text/html;charset=utf-8'
08:       res.status = '200'
09:       res.write('Hello, Rack!')
10:     }.finish
11:   end
12: end
```

hello.ru

```
01: require 'hello'
02: run Hello.new
```

そして以下のコマンドを実行してウェブブラウザで http://localhost:9292/ にアクセスします。

```
$ rackup hello.ru
```

これだけです。

Rack が対応しているサーバ

標準

- CGI
- FastCGI
- LSWS
- Mongrel
- SCGI
- WEBrick

サードパーティ

- Ebb (http://repo.or.cz/w/ebb.git)
- Fuzed (http://fuzed.rubyforge.org/)
- Thin (http://code.macournoyer.com/thin/)

色々なサーバで動かす

rackup コマンドを使って色々なサーバで動かすことができます。rackup コマンドで起動するウェブサーバはデフォルトでは webrick ですが mongrel がインストールされている場合は mongrel が起動します。

- webrick

  $ rackup hello.ru -s webrick -p 9292

- mongrel

  $ rackup hello.ru -s mongrel -p 9292

- thin

  $ thin -R hello.ru -p 3000 start

- CGI

CGI が動くようにウェブサーバを設定しておき、以下のようなファイルを用意する。

```
01: #!/usr/bin/ruby
02:
03: require 'hello'
04:
05: Rack::Handler::CGI.run(Rack::ShowExceptions.new(Rack::Lint.new(Hello.new)))
06:
```

- FastCGI

FastCGI が動くようにウェブサーバを設定しておき、以下のようなファイルを用意する。

```
01: #!/usr/bin/ruby
02:
03: require 'hello'
04:
05: Rack::Handler::FastCGI.run(Rack::Builder.new{ run Hello.new }.to_app)
06:
```

Rack に対応しているウェブアプリケーションフレームワーク

- Rack::Adapter::Camping
- Coset
- Halcyon
- Maveric
- Merb
- Racktools::SimpleApplication
- Ramaze
- Sinatra
- Vintage
- Waves

サンプル: 一行掲示板

Rack のみを使用した簡単なアプリケーションのサンプルとして一行掲示板を作ってみます。仕様や制限は以下のような感じです。

- 最小のファイル数で実装する
- Handler は限定しない
- エラー処理はしない
- 最低限のエスケープ処理はする
- データは DB に保存する (ActiveRecord 使用)
- View には ERB を使う

解説

まずは linebbs.ru の説明。

```
01: require 'linebbs'
02:
03: use Rack::Lint
04: use Rack::ShowExceptions
05: use Rack::Reloader
06: use Rack::CommonLogger
07:
08: run LineBbs.new
```

今までと違って、3 つのミドルウェアを使用している。

Rack::Lint
　このアプリケーションが Rack の仕様を満たしていることをチェックする

Rack::ShowExceptions
　例外発生時のエラー表示を見やすくする。開発時は必須。

Rack::Reloader
　プログラムが変更されたらファイルをリロードする。開発時は必須。

Rack::CommonLogger
　ロガー。Apache の CommonLog 形式でログを出力する。

テーブル定義はこんな感じです。

```
01: CREATE TABLE "comments" (
02:   "id" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
03:   "name" varchar(255) DEFAULT NULL NULL,
04:   "comment" varchar(255) DEFAULT NULL NULL,
05:   "created_at" datetime DEFAULT NULL NULL,
06:   "updated_at" datetime DEFAULT NULL NULL
07: );
```

ActiveRecord を使用しているけど、migration ファイルは使わなかった。テーブルが一つだったので直接 SQL を書いて定義しました。
　本体はシンプルなので Model, View, Controller を全て一つのファイルにまとめました。

```
01: require 'rubygems'
02: require 'rack'
03: require 'activerecord'
04: require 'erb'
05:
06: class LineBbs
07:   include ERB::Util
08:
09:   def call(env)
10:     request = Rack::Request.new(env)
11:     if request.get?
12:       res = get(request)
13:     elsif request.post?
14:       res = post(request)
15:     end
16:     res.finish
17:   end
18:
```

```
19:   def get(request)
20:     comments = Comment.find(:all, :order => "id DESC")
21:     Rack::Response.new{|res|
22:       res['Content-Type']= 'text/html;charset=utf-8'
23:       res.status = '200'
24:       res.write ERB.new(::TEMPLATE, nil, '-').result(binding)
25:     }
26:   end
27:
28:   def post(request)
29:     Comment.new{|c|
30:       c.name    = request.params['name']
31:       c.comment = request.params['comment']
32:     }.save!
33:     Rack::Response.new{|res|
34:       res['Content-Type']= 'text/html;charset=utf-8'
35:       res.status = '302'
36:       res['Location'] = "#{request.scheme}://#{request.host}:#{request.port}/"
37:     }
38:   end
39: end
40:
41: ActiveRecord::Base.establish_connection(
42:   :adapter => 'sqlite3',
43:   :database => 'linebbs.sqlite3'
44: )
45: ActiveRecord::Base.logger = Logger.new(STDOUT)
46:
47: class Comment < ActiveRecord::Base
48: end
49:
50: TEMPLATE=<<EOD
51: <html>
52:   <head>
53:     <title>LineBBS</title>
54:   </head>
55:   <body>
56:     <h3>LineBBS</h3>
57:     <div class="form">
58:       <form action="/" method="post">
59:         <input type="text" name="name" size="20" /> :
60:         <input type="text" name="comment" size="60" />
61:         <input type="submit" name="submit" />
```

```
62:       </form>
63:     </div>
64:     <hr />
65:     <div class="comments">
66:     <%- comments.each do |c| -%>
67:       <p>
68:          <%=h c.created_at.strftime('%Y-%m-%d %H:%M:%S') %> :
69:          <%=h c.name %> : <%=h c.comment %>
70:       </p>
71:     <%- end -%>
72:     </div>
73:   </body>
74: </html>
75: EOD
```

- line. 7 : View 内部で h メソッドを使用したかったので。
- line. 9-17 : リクエストをオブジェクト化して get, post に処理を委譲するだけです。
- line. 19-26 : 全てのコメントを表示するための処理をしている
- line. 28-38 : POST されたデータを DB に登録して GET へリダイレクトしている。
- line. 41-48 : ActiveRecord の設定とモデルクラスの定義。
- line. 50-72 : View のテンプレートを文字列で定義している部分。

その他のミドルウェア

Rack::Auth::Basic
HTTP のベーシック認証を使用するためのミドルウェア。サンプル有り。

Rack::Auth::Digest
HTTP のダイジェスト認証を使用するためのミドルウェア。

Rack::Auth::OpenID
OpenID による認証を扱うためのミドルウェア。ruby-openid に依存。

Rack::Cascade
一つのリクエストをいくつかのアプリケーションにたらい回しするためのミドルウェア。

Rack::File
クライアントへファイルを送信するためのミドルウェア。

Rack::ForwardRequest
他のアプリケーションなどへリクエストをフォワードするためのミドルウェア。

Rack::MockRequest
テスト用。

Rack::MockRequest::FatalWarner
テスト用。

Rack::MockRequest::FatalWarning
テスト用。

Rack::MockResponse
テスト用。

Rack::Session::Cookie
クッキーベースのセッション管理を使用するためのミドルウェア。

Rack::Session::Pool
クッキーベースのセッション管理を使用するためのミドルウェア。

Rack::ShowStatus
空のレスポンスをラップしてサイトの説明などを表示してくれるミドルウェア。

Rack::Static
静的なファイルを表示するために使用するミドルウェア。

Rack::URLMap
URL とアプリケーションのマッピングを行うためのミドルウェア。

演習

一行掲示板を写経

一行掲示板を写経してください。わからないところはウェブで調べたり TA に質問したり、まわりの人と相談するなどして理解してみてください。
また、余裕のある人はよりよい一行掲示版になるように改善してみてください。

ちょっと変更

ちょっとだけ変更してみてください。

- 発言の表示順を古い発言が上になるようにしてみてください。
- 日付の表示をカスタマイズしてみてください。

ミドルウェアを使う

その他のミドルウェアで紹介したいずれかのミドルウェアを一行掲示版に組み込んでください。

機能追加

以下の機能を追加してください。

- 入力時に削除キーを記録しておき、削除キーを入力したら発言を削除できる機能。
  - 入力時に削除キーを入力していない場合は削除できません。
- 管理用のキーを入力したらどの発言でも削除できる機能。

その他、色々といじってみてください。

参考文献

Rack: a Ruby Webserver Interface
http://rack.rubyforge.org/

Greenbear Diary - 5 分でわかる Rack , シュレーディンガーの猫たち
http://mono.kmc.gr.jp/~yhara/d/?date=20080716

Greenbear Laboratory - Rack 日本語リファレンス
http://mono.kmc.gr.jp/~yhara/w/?RackReferenceJa

ウノウラボ Unoh Labs: Rack で Web アプリの Web サーバー依存を無くす
http://labs.unoh.net/2007/05/rackweb.html

今後の情報源

公式 Web サイト
http://www.ruby-lang.org/

Ruby リファレンスマニュアル刷新計画
http://doc.loveruby.net/

日本 Ruby の会
http://jp.rubyist.net/

Rubyist Magazine
http://jp.rubyist.net/magazine/

okkez weblog
http://typo.okkez.net/