Rails 初心者レッスン 第1回

かずひこ@（株）ネットワーク応用通信研究所，あゆ@（株）イーサポート

2006年6月17日

今日のお題

- Railsアプリってどうなってるの？
- 簡単な「ブックマーク」を作りましょう。

1-1 最初のセットアップ

- railsコマンドで作ります。まずはヘルプの表示。

```
$ rails --help
    -d, --database=name データベースを指定する
                        (mysql/oracle/postgresql/sqlite2/sqlite3)
```

- いよいよrailsコマンドを実行します。

（SQLite3の場合）
```
$ rails -d sqlite3 bookmark
```

（MySQLがデフォルトなので-dなしでよい）
```
$ rails bookmark
```

（PostgreSQLの場合）
```
$ rails -d postgresql bookmark
```

1-2 ディレクトリ構成

- bookmarkディレクトリに移動して、中を確認しましょう。
  - app - アプリケーション本体が入る場所
  - config - 設定ファイルが入る場所
  - db - データベースの構造ファイルが入る場所
  - public - ウェブサーバで公開される場所

1-3 データベースの設定（SQLite3）

- config/database.ymlで設定します

```
development:
  adapter: sqlite3
  database: db/development.sqlite3

test:
  adapter: sqlite3
  database: db/test.sqlite3

production:
  adapter: sqlite3
  database: db/production.sqlite3
```

1-3 データベースの設定（MySQL）

- まずデータベースを作成し（bookmark_development, bookmark_test, bookmark_productionの三つ）、適切な権限を設定します。作成するデータベースの文字コードはutf8にしましょう。以下はコマンドラインからの実行例です。

```
$ mysql -u root mysql
mysql> create database bookmark_development default character set utf8;
mysql> create database bookmark_test default character set utf8;
mysql> create database bookmark_production default character set utf8;
mysql> grant all on bookmark_development.* to 'rails'@'localhost';
mysql> grant all on bookmark_test.* to 'rails'@'localhost';
mysql> grant all on bookmark_production.* to 'rails'@'localhost';
```

- config/database.ymlで設定します。databaseの項は、さきほど作成したデータベース名を指定します。接続のための設定（username, password, host）は、環境にあわせて設定してください。また、日本語で使う場合はencoding: utf8を指定しておきましょう。

```
development:
  adapter: mysql
  database: bookmark_development
  username: root
  password:
  host: localhost
  encoding: utf8

test:
```

```
  adapter: mysql
  database: bookmark_test
  username: root
  password:
  host: localhost
  encoding: utf8

production:
  adapter: mysql
  database: bookmark_production
  username: root
  password:
  host: localhost
  encoding: utf8
```

1-3 データベースの設定（PostgreSQL）

- **まずデータベースを作成し（**bookmark_development, bookmark_test, bookmark_production **の三つ）、適切な権限を設定します。作成するデータベースの文字コードは** utf8 **にしましょう。以下はコマンドラインからの実行例です。**

```
$ createdb -U bookmark -E utf8 bookmark_development
$ createdb -U bookmark -E utf8 bookmark_test
$ createdb -U bookmark -E utf8 bookmark_production
```

- config/database.yml **で設定します。**database **の項は、さきほど作成したデータベース名を指定します。接続のための設定（**username, password, host**）は、環境にあわせて設定してください。また、日本語で使う場合は** encoding: utf8 **を指定しておきましょう。**

```
development:
  adapter: postgresql
  database: bookmark_development
  username: bookmark
  password:
  encoding: utf8

test:
  adapter: postgresql
  database: bookmark_test
  username: bookmark
  password:
  encoding: utf8
```



```
production:
  adapter: postgresql
  database: bookmark_production
  username: bookmark
  password:
  encoding: utf8
```

1-4 文字コードの設定

- Ruby **の文字コードを** UTF-8 **に設定します。**config/environment.rb **の先頭に以下を追加してください。**

```
$KCODE = 'u'
```

1-4 とりあえず起動

- script/server **で簡易サーバを起動します。**

```
$ ruby script/server --help

$ ruby script/server
=> Booting WEBrick...
=> Rails application started on http://0.0.0.0:3000
=> Ctrl-C to shutdown server; call with --help for options
[2006-06-16 00:52:02] INFO  WEBrick 1.3.1
[2006-06-16 00:52:02] INFO  ruby 1.8.4 (2005-12-24) [i486-linux]
[2006-06-16 00:52:02] INFO  WEBrick::HTTPServer#start: pid=17087 port=3000
```

1-5 ブラウザで確認

- http://127.0.0.1:3000/**をアクセスします。**
  - public/index.html **がそのまま表示されます。**
- http://127.0.0.1:3000/robots.txt **をアクセスします。**
  - public/robots.txt **がそのまま表示されます。**

1-6 まとめ

- rails **コマンドで最初の準備**
- config/database.yml **でデータベースの設定**
- ruby script/server **で簡易サーバを起動（**http://127.0.0.1:3000/**）**

2-1 モデルを作る

- **モデルというのは、アプリケーションで扱うオブジェクトのことです。**

```
$ ruby script/generate model --help

$ ruby script/generate model Item
      exists  app/models/
      exists  test/unit/
      exists  test/fixtures/
      create  app/models/item.rb
      create  test/unit/item_test.rb
      create  test/fixtures/items.yml
      create  db/migrate
      create  db/migrate/001_create_items.rb

# app/models/item.rb
class Item < ActiveRecord::Base
end
```

2-2 テーブルの定義（migration）

- db/migrate/001_create_items.rb **を編集します。**

```
class CreateItems < ActiveRecord::Migration
  def self.up
    create_table(:items) do |t|
      t.column(:url, :string)
      t.column(:title, :string)
    end
  end

  def self.down
    drop_table(:items)
  end
end
```

- **先頭の** 001 **がバージョン番号です。**
- **データベースの状態をバージョンごとに管理します。**
- **そのファイルのバージョンに上げると** self.up **が実行され、そのファイルのバージョンから下げると** self.down **が実行されます。**

2-3 テーブルの作成

- rake db:migrate **を実行します（**Rails 1.0 **以前のバージョンでは** rake migrate **を実行します）。**

```
$ rake db:migrate
== CreateItems: migrating ==================================
-- create_table(:items)
   -> 0.1141s
== CreateItems: migrated (0.1154s) =========================
```

- rake **のタスクの一覧は、**rake -T **で表示できます。**
- rake **のヘルプは** rake –help **で表示できます。**

2-4 データベースの確認

- sqlite3 **コマンドで、データベースを確認しましょう。**

```
$ sqlite3 db/development.sqlite3
SQLite version 3.3.5
Enter ".help" for instructions
sqlite> .schema
CREATE TABLE items ("id" INTEGER PRIMARY KEY NOT NULL, "url" varchar(255),
  "title" varchar(255));
CREATE TABLE schema_info (version integer);
sqlite> select version from schema_info;
2
sqlite> .quit
```

- schema_info **というのは、**migration **のバージョン情報を記録するためのテーブルです。**

2-5 モデルの操作

- script/console **で、モデルの操作をしてみましょう。**

```
$ ruby script/console --help

$ ruby script/console
（まだ空っぽ）
>> Item.count
=> 0
（新しいモデルを作って保存）
>> bm = Item.create(:url => 'http://www.rubyonrails.org/', :title => 'Ruby on Rails')
```

```
=> #<Item:0xb7460570 @new_record=false, @errors=#<ActiveRecord::Errors:0xb73ec24c
   @errors={}, @base=#<Item:0xb7460570 ...>>, @attributes={"title"=>"Ruby on Rails",
   "url"=>"http://www.rubyonrails.org/", "id"=>1}>
```

（モデルの数は一つ）

```
>> Item.count
=> 1
```

（新しいモデルを作る（保存はまだ））

```
>> bm2 = Item.new(:url => 'http://www.ruby-lang.org/', :title => 'Ruby')
=> #<Item:0xb73d5f60 @new_record=true, @attributes={"title"=>"Ruby",
   "url"=>"http://www.ruby-lang.org/"}>
```

（モデルの数は増えていない）

```
>> Item.count
=> 1
```

（モデルを保存）

```
>> bm2.save
=> true
```

（モデルの数が増えた）

```
>> Item.count
=> 2
```

（全てのモデルの取得）

```
>> Item.find(:all)
=> [#<Item:0xb73c6c7c @attributes={"title"=>"Ruby on Rails",
   "url"=>"http://www.rubyonrails.org/", "id"=>"1"}>, #<Item:0xb73c6c40
    @attributes={"title"=>"Ruby", "url"=>"http://www.ruby-lang.org/", "id"=>"2"}>]
```

2-6 データベースの確認

- sqlite3 **コマンドで、データベースを確認しましょう。**

```
$ sqlite3 db/development.sqlite3
sqlite> .header on
sqlite> select * from items;
id|url|title
1|http://www.rubyonrails.org/|Ruby on Rails
2|http://www.ruby-lang.org/|Ruby
```

2-7 まとめ

- Ruby **のオブジェクトとデータベースのテーブルが対応する**
- ruby script/generate model **でモデルの雛型を作成する**
- **データベースのテーブルの定義も** Ruby **で書ける**

- rake db:migrate **で実際に更新する**
- ruby script/console **でモデルを操作できる**

3-1 とりあえずアクセスしてみる

- http://127.0.0.1:3000/item/**をアクセスしてみると**...

```
Routing Error

  Recognition failed for "/item/"
```

- **ブラウザからのアクセスを受け付けて処理をするのはコントローラの仕事です。**

3-2 コントローラを作る

- **コントローラとは、外部からのアクセスに応じて処理をする部分のことです。**

```
$ ruby script/generate controller --help

$ ruby script/generate controller Item
      create  app/views/item
      create  app/controllers/item_controller.rb
      create  test/functional/item_controller_test.rb
      create  app/helpers/item_helper.rb

# app/controllers/item_controller.rb
class ItemController < ApplicationController
end
```

3-3 もう一度アクセスしてみる

- http://127.0.0.1:3000/item/**をアクセスしてみると**...

```
Unknown action

  No action responded to index
```

3-4 scaffold を使ってみる

- app/controllers/item_controller.rb **を編集します。**

```
class ItemController < ApplicationController
  scaffold(:item)
end
```

3-5 アクセスしてみる

- http://127.0.0.1:3000/item/から、以下の操作が行えます。
  - 一覧の表示
  - 新規作成
  - 個別に表示
  - 個別に編集
  - 個別に削除

3-6 まとめ

- ruby script/generate controller でコントローラの雛型を作成する
- URI の始めのディレクトリ名とコントローラの名前が対応している
- scaffold と書くだけでモデルに対するひととおりの操作ができる

4-1 もうひとつの scaffold

- 「おまじない」ではなく、コードを生成してみましょう。

```
$ ruby script/generate scaffold --help

$ ruby script/generate scaffold Item Item
      create  app/views/item/_form.rhtml
      create  app/views/item/list.rhtml
      create  app/views/item/show.rhtml
      create  app/views/item/new.rhtml
      create  app/views/item/edit.rhtml
overwrite app/controllers/item_controller.rb? [Ynaq]
       force  app/controllers/item_controller.rb
overwrite test/functional/item_controller_test.rb? [Ynaq]
       force  test/functional/item_controller_test.rb
      create  app/views/layouts/item.rhtml
      create  public/stylesheets/scaffold.css
```

4-2 できたファイルを見る（コントローラ）

- app/controllers/item_controller.rb に以下のメソッドが追加されました。
  - index
  - list
  - show
  - new
  - create
  - edit
  - update
  - destroy

- index メソッドを見ましょう。

```
def index
  list
  render :action => 'list'
end
```

- list メソッドを呼んで、list というアクションのビューを表示させています。
- ItemController の list アクションのビューのファイルは app/views/item/list.rhtml です。

4-3 できたファイルを見る（ビュー）

- ビューとは、結果をブラウザなどに出力するためのファイルのことです。
- app/views/item/list.rhtml を見てみましょう。

```
<tr>
<% for column in Item.content_columns %>
  <th><%= column.human_name %></th>
<% end %>
</tr>
```

- ここで、Item モデルのフィールド名をテーブルの見出しとして表示しています。

```
<% for item in @items %>
  <tr>
  <% for column in Item.content_columns %>
    <td><%=h item.send(column.name) %></td>
  <% end %>
 (略)
  </tr>
<% end %>
```

- ここで、@items の各要素に対して、各フィールドの内容を表示しています（例:item.send('url') → item.url）。
- h は HTML 上ヤバい文字（&"><）をエスケープするメソッドです。外部からの入力（を処理したもの）を表示させる時には指定するようにしましょう。

4-4 できたファイルを見る（コントローラ）

- app/controllers/item_controller.rb の new メソッドを見てみましょう。

```
def new
  @item = Item.new
end
```

- render を指定していないときは、メソッド名と同じアクションで render が呼ばれたのと同じになります（この例では render(:action => 'new') と同じ）

4-5 できたファイルを見る（ビューの部品）

- app/views/item/new.rhtml を見てみましょう。

```
<h1>New item</h1>

<%= start_form_tag :action => 'create' %>
  <%= render :partial => 'form' %>
  <%= submit_tag "Create" %>
<%= end_form_tag %>

<%= link_to 'Back', :action => 'list' %>
```

- ItemController の create メソッド、つまり/item/create という URI を呼ぶフォームです。
- render(:partial => 'form') は、_form.rhtml というビューのファイルを埋め込む書き方です。
- 続いて、app/views/item/_form.rhtml を見てみましょう。

```
<%= error_messages_for 'item' %>

<!--[form:item]-->
<p><label for="item_url">Url</label><br/>
<%= text_field 'item', 'url' %></p>

<p><label for="item_title">Title</label><br/>
<%= text_field 'item', 'title' %></p>
<!--[eoform:item]-->
```

- URL とタイトルの入力欄が text_field メソッドで書いてあります。

4-6 まとめ

- ruby script/generate scaffold で、一通りの操作ができるコントローラとビューを作成できる
- render メソッドで、どのビューを表示するか指定する。
- render メソッドを省略すると、コントローラのメソッドと同じ名前のビューを表示する。
- ビューの部品は_ではじまるファイル名をつけて、render メソッドの:partial オプションで指定する。

Tips メソッドの仕様の調べかた

- gem_server を起動すれば、ドキュメント用のウェブサーバが起動します。

```
$ gem_server
[2006-06-16 20:10:40] INFO  WEBrick 1.3.1
[2006-06-16 20:10:40] INFO  ruby 1.8.4 (2005-12-24) [i486-linux]
[2006-06-16 20:10:40] INFO  WEBrick::HTTPServer#start: pid=22101 port=8808
```

- port=8808 と書いてあるように、http://127.0.0.1:8808/にアクセスすればドキュメントを読むことができます。
- ネットワークにつながっていなくても見ることができるので便利です。

5-1 モデルの変更

- description フィールドを追加します。まずは migration スクリプトの雛型を作ります。

```
$ ruby script/generate migration add_description_to_items
      create  db/migrate/002_add_description_to_items.rb
```

- 次に、db/migrate/002_add_description_to_items.rb を編集します。

```
class AddDescriptionToItems < ActiveRecord::Migration
  def self.up
    add_column(:items, :description, :text)
  end

  def self.down
    remove_column(:items, :description)
  end
end
```

- では、rake db:migrate を実行します。

```
== AddDescriptionToItems: migrating ======================
-- add_column(:items, :description, :text)
   -> 0.2277s
== AddDescriptionToItems: migrated (0.2289s) =============
```

5-2 データベースの確認

- sqlite3 コマンドで、データベースを確認しましょう。

```
$ sqlite3 db/development.sqlite3
sqlite> .schema
CREATE TABLE items ("id" INTEGER PRIMARY KEY NOT NULL, "url" varchar(255),
  "title" varchar(255), "description" text);
CREATE TABLE schema_info (version integer);
sqlite> select version from schema_info;
2
sqlite> .quit
```

5-3 ブラウザで確認

- http://127.0.0.1:3000/item/やhttp://127.0.0.1:3000/item/show/1 を見ると、description フィールドが追加されています。
  - Item.content_columns を順に表示しているからですね。
- 一方、http://127.0.0.1:3000/item/new は変化ありません。

5-4 入力フォームの追加

- app/views/item/_form.rhtml を編集しましょう。

```
<%= error_messages_for 'item' %>

<!--[form:item]-->
<p><label for="item_url">Url</label><br/>
<%= text_field 'item', 'url' %></p>

<p><label for="item_title">Title</label><br/>
<%= text_field 'item', 'title' %></p>

<p><label for="item_description">Description</label><br/>
<%= text_area 'item', 'description' %></p>
<!--[eoform:item]-->
```


- 長い文字列を入力できるように、text_field メソッドのかわりに text_area メソッドを使います。

5-5 ブラウザで確認

- もう一度 http://127.0.0.1:3000/item/new を見てみましょう。
- http://127.0.0.1:3000/item/edit/1 も見てみましょう。

5-6 URI とコントローラの関係

- config/routes.rb に書いてあります。

```
ActionController::Routing::Routes.draw do |map|
  map.connect ':controller/:action/:id'
end
```

- アクセスされた URI は、/コントローラ名/アクション名/id という風に解釈されて、アプリケーションが動作します。

5-7 まとめ

- テーブルの変更も migration で行えます。
- 長い文字列の入力には text_area タグを使います。
- 簡易サーバは、ファイルを変更したらすぐに反映します（一部のファイルを除く）。
- URI とコントローラの関係は config/routes.rb に記述します。

改変履歴

- 1.0 (20060617)
  - 当日に配布したもの
- 1.1 (20060618)
  - 最初のウェブ公開版。見出し番号の修正。
- 1.2 (20060619) ←いまここ
  - MySQL と PostgreSQL の場合のデータベースの作成と確認について追記。

参考文献

はじめよう Ruby on Rails
ISBN:4756147739

Rails によるアジャイル Web アプリケーション開発
ISBN:4274066401

今後の情報源

Rails Wiki
http://wiki.fdiary.net/rails/

ふぇみにん日記
http://kazuhiko.tdiary.net/