2006年12月7日 かずひこ・香西利衣

はじめに

Ruby on Railsの概要

- Rubyで書かれたウェブアプリケーションフレームワークです。
- 自動化プログラムやアプリケーションサーバも含まれます。

Ruby on Railsのポリシー

- DRY (Don't Repeat Yourself)「重複を避けましょう」
  - 重複する部分を一つにまとめることで、プログラムのメンテナンスがしやすくなります。
- CoC (Convention over Configuration)「設定より規約」
  - 積極的にデフォルト値を利用することで、プログラムの記述量を減らせます。

想定する環境

OS
　Linux
データベース
　PostgreSQL
Ruby
　バージョン1.8.5
Ruby on Rails
　バージョン1.1.6をgemコマンドでインストール

環境によっては、ファイルのパスやSQLのログなどが若干異なりますが、適宜読みかえてください。

> [MEMO] 各種環境でのインストール方法は http://wiki.rubyonrails.org/rails/pages/HowtosInstallation を参照してください。

Step1 さあ始めよう

アプリケーションの雛型の作成

その名も「rails」コマンドで作成します。

```
$ rails --help
Usage: /usr/bin/rails /path/to/your/app [options]
（略）
$ rails bookmark -d postgresql
      create
      create  app/controllers
      create  app/helpers
      create  app/models
      create  app/views/layouts
 （略）
```

> [MEMO] Railsに含まれる各種スクリプトは--helpオプションでヘルプを表示させることができます。

ディレクトリの中を確認します。

```
$ cd bookmark
$ ls
README   app/       config/  doc/  log/    script/  tmp/
Rakefile  components/  db/     lib/  public/  test/    vendor/
```

app/

アプリケーションの本体であるモデルやコントローラ、ビューのファイルが入ります。

app/models/
モデル用のrbファイルが入ります。

app/controllers/
コントローラ用のrbファイルが入ります。

app/views/
ビュー用のrhtmlファイルが入ります。

app/helpers/
ビューヘルパー用のrbファイルが入ります。

config/
設定ファイルを入れるディレクトリです。Railsの環境設定やデータベースの接続パラメータなどを設定するファイルが入ります。

db/
データーベースの構造を記述したファイルが入ります。

lib/
アプリケーションで使うライブラリなどが入ります。

public/
HTTPサーバーのルートディレクトリにあたるディレクトリで、ブラウザから直接参照できる場所です。

script/
アプリケーション作成を支援するスクリプトがあらかじめ入っています。

test/
テストのためのファイルが入ります。

vendor/
プラグインなどが入ります。

文字コードの設定

Ajaxとの相性からいっても、Railsアプリケーションの文字コードはUTF-8をお薦めします。

データベースの文字コードをUTF-8に

config/database.ymlを以下のように編集します。

config/database.yml

```
development:
 （略）
 encoding: UTF8

test:
 （略）
 encoding: UTF8

production:
 （略）
 encoding: UTF8
```

つづいて、データベースを作成します。

```
$ createdb -U bookmark -E UTF8 bookmark_development
$ createdb -U bookmark -E UTF8 bookmark_test
$ createdb -U bookmark -E UTF8 bookmark_production
```

rubyの文字コードをUTF-8に

以下をconfig/environment.rbの先頭に追加します。

```
$KCODE = "u"
```

[MEMO] $KCODEの変更はrubyスクリプトの読み込みにも影響をあたえるので、できるだけ早い段階で変更するために、先頭に追加しましょう。

ウェブサーバのcharsetをUTF-8に

app/controllers/applications.rbを以下のように編集します。

```
class ApplicationController < ActionController::Base
  after_filter :set_charset

  private

  def set_charset
    content_type = @headers["Content-Type"] || "text/html"
    if %r!¥Atext/! =~ content_type
      @headers["Content-Type"] = "#{content_type}; charset=utf-8"
    end
  end
end
```

[MEMO] ApplicationControllerは、各コントローラクラスのスーパークラスです。after_filterで指定されたメソッドは、コントローラによる処理の最後で呼ばれます。

アプリケーション・サーバの起動

script/serverで、WEBrickを利用したアプリケーション・サーバが起動します。

```
$ ruby script/server --help
（略）
$ 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
[2006-10-16 22:37:25] INFO  WEBrick 1.3.1
[2006-10-16 22:37:25] INFO  ruby 1.8.5 (2006-08-25) [i486-linux]
[2006-10-16 22:37:25] INFO  WEBrick::HTTPServer#start: pid=6074 port=3000
```

script/serverは、デフォルトでは3000/tcpで起動します。

[MEMO] Railsで使えるscript/server以外の主なアプリケーションサーバには以下のものがあります。

- Apache HTTPD + FastCGI
- LightTPD + FastCGI http://www.lighttpd.net/
- Mongrel http://mongrel.rubyforge.org/

アプリケーション・サーバの確認

ウェブブラウザで http://localhost:3000/ にアクセスします。この時に表示されているのは、public/index.htmlです。

[MEMO] public/がApacheのDocumentRootにあたるディレクトリで、その中にあるファイルへのアクセスは、アプリケーションを介さずに返されます（例：/robots.txtや/javascripts/prototype.jsなど）。

リファレンスマニュアル

gemコマンドでRailsをインストールすると、Railsを構成する各パッケージのHTML形式のマニュアルもインストールされます。このマニュアルは、gem_serverを起動することで、ブラウザからアクセスすることができます。

```
$ gem_server --help
（略）
$ gem_server &
[2006-11-06 11:46:45] INFO  WEBrick 1.3.1
[2006-11-06 11:46:45] INFO  ruby 1.8.5 (2006-08-25) [i486-linux]
```

```
[2006-11-06 11:46:45] INFO  WEBrick::HTTPServer#start: pid=14875 port=8808
```

デフォルトでは8808/tcpで起動しますので、ブラウザで http://localhost:8808/ にアクセスしてください。

モデリング

「○○さん」が「○○というページ」に「○月○日に○○というコメントでブックマークする」というのが、今回のアプリケーションのモデリングです。順にそれぞれ、User、Page、Bookmarkというモデルを割り当てます。

各モデルの持つべき情報

各モデルが最低限持つべき情報は以下のようになります。

User
　　ログインID, パスワード
Page
　　URI, タイトル
Bookmark
　　ユーザID, ページID, コメント, 作成日時

ユーザとページとブックマークの関係

あるユーザは複数のブックマークを持ち、あるページも複数のブックマークを持つので、モデル間には以下のような関係があります。

```
User ------ Bookmark ------ Page
    1:多          多:1
```

また、あるユーザは複数のページをブックマークし、あるページも複数のユーザにブックマークされるので、以下のような関係もあります。

```
User ------- Page
    多:多
```

Step2 モデル作成（1）

最初に「ページ」のモデルを作成します。

雛型の作成

"script/generate model"でモデルの雛型を作成します。

```
$ ruby script/generate model --help
（略）
$ ruby script/generate model Page
    create  app/models/page.rb
    create  test/unit/page_test.rb
    create  test/fixtures/pages.yml
    create  db/migrate
    create  db/migrate/001_create_pages.rb
```

テーブル定義ファイルの編集

db/migrate/001_create_pages.rbを編集して、uriとtitleという二つのカラムをPageテーブルに設定します。

db/migrate/001_create_pages.rb

```
class CreatePages < ActiveRecord::Migration
  def self.up
    create_table :pages do |t|
```

```
      t.column :uri, :string, :limit => 1024
      t.column :title, :string, :limit => 1024
    end
  end

  def self.down
    drop_table :pages
  end
end
```

self.upは、migrationのバージョンが上がる時に実行され、逆にself.downはmigrationのバージョンが下がる時に実行されます。

[MEMO] 詳しくは「マニュアル：ActiveRecord::Migration」を参照してください。

テーブル定義の実行

"rake db:migrate"で、テーブル定義が実行されテーブルが作成されます。

```
$ rake --help
（略）
$ rake db:migrate
(in /home/rails/bookmark)
== CreatePages: migrating ====================================================
-- create_table(:pages)
   -> 0.5079s
== CreatePages: migrated (0.5272s) ===========================================
```

[MEMO] Rakeはmakeやantのような自動化のためのツールです。

Rakeの具体的な各処理はタスクと呼ばれ、Rakeはタスク間の依存関係に従ってタスクを実行します。以下のコマンドでタスクの一覧を見ることが出来ます。

```
$ rake -T
```

[MEMO] Railsで使える標準のrakeのタスクは、/usr/lib/ruby/gems/1.8/gems/rails-1.1.6/lib/tasks/*.rakeで定義されています。なお、タスクを指定せずに実行すると、testタスクを実行します。

Step3 コントローラ作成（1）

ページのコントローラの雛型を題材に、コントローラの基本を学びます。

ページ操作の雛型の作成

```
$ ruby script/generate scaffold --help
（略）
$ ruby script/generate scaffold Page Page
  identical    app/models/page.rb
  identical    test/unit/page_test.rb
  identical    test/fixtures/pages.yml
     create  app/views/page/_form.rhtml
     create  app/views/page/list.rhtml
     create  app/views/page/show.rhtml
     create  app/views/page/new.rhtml
     create  app/views/page/edit.rhtml
     create  app/controllers/page_controller.rb
     create  test/functional/page_controller_test.rb
     create  app/helpers/page_helper.rb
     create  app/views/layouts/page.rhtml
```

```
    create  public/stylesheets/scaffold.css
```

ソースを読もう

では、作成されたファイルをそれぞれ見てみましょう。

コントローラ

コントローラ中の一つの処理に、同名のビューファイルが一つ対応して表示されます。

Rails はリクエストの情報に基づいて、コントローラ名とそれに続くアクション名を取得し、コントローラのクラスの中から対象のアクションのメソッドを実行します。

例えば、http://localhost:3000/page/index（または http://localhost:3000/page/ ）にアクセスすると、PageControllerクラス中のindexというアクションメソッドが呼ばれます。

app/controllers/page_controller.rb

```
class PageController < ApplicationController
  def index
    list # ←(1)
    render :action => 'list' # ←(2)
  end
(略)
end
```

indexメソッドでは、(1)で同クラスのlistメソッドを呼び出した後、(2)でlistという名前のビューファイル（app/views/page/list.rhtml）を出力しています。

もし http://localhost:3000/page/list にアクセスすると、PageControllerクラス中のlistというアクションメソッドが呼ばれます。

app/controllers/page_controller.rb（一部）

```
  def list
    @page_pages, @pages = paginate :pages, :per_page => 10
  end
```

ここでは、さきほどのindexメソッドの定義と違って、renderメソッドが呼ばれていませんが、その場合はアクションメソッドと同名のビューが呼ばれます。つまり、

```
render :action => 'list'
```

と書くのと同じ挙動になります。

ビュー

indexメソッドやlistメソッドでrenderされるビューのファイルを見ましょう。

app/views/page/list.rhtml

```
<h1>Listing pages</h1>

<table>
  <tr>
  <% for column in Page.content_columns %>
    <th><%= column.human_name %></th> # ←(1)
  <% end %>
  </tr>

  <% for page in @pages %>
  <tr>
  <% for column in Page.content_columns %>
```

```
    <td><%=h page.send(column.name) %></td>    # ←(2)
  <% end %>
    <td><%= link_to 'Show', :action => 'show', :id => page %></td> # ←(3)
    <td><%= link_to 'Edit', :action => 'edit', :id => page %></td>
    <td><%= link_to 'Destroy', { :action => 'destroy', :id => page }, :confirm> 'Are you sure?', :post =>
true %></td>
  </tr>
<% end %>
</table>

<%= link_to 'Previous page', { :page => @page_pages.current.previous } if @pagpages.current.previous
%>
<%= link_to 'Next page', { :page => @page_pages.current.next } if @page_pages.rrent.next %> <br />
<%= link_to 'New page', :action => 'new' %>
```

ビューで動的コンテンツを可能にしているのはERB(Embedded Ruby)です。ERBでは、以下のタグを使って、Rubyのコードを HTMLなどのテキストファイルに埋め込むことができます。

<% ... %>
　　「...」の部分をRubyで実行する。
<%= ... %>
　　「...」の部分をRubyで実行し、その値をその場に挿入する。
<%# ... %>
　　「...」の部分をコメントアウトする。

まず、Pageモデルの各カラムの名前を見出しとして出力しています(1)。つぎに、@pagesに含まれている各Pageごとにテーブルの行を作成し、各カラムの値を出力しています(2)。(3)ではShowというリンク文字列に対して、コントローラのshowアクションを指定して http://localhost:3000/page/show/1 のようなURIへのハイパーリンクを作成しています。showアクションに扱う項目が何かを指定するためにidを指定しています。

> [MEMO] この例のように、「:id => page」とモデルのオブジェクトを指定すると、自動的に「:id => page.id」と書くのと同じ意味になります。

さまざまなアクションの出力に共通する部分は、レイアウトとよばれるビューファイルに書きます。

app/views/layouts/page.rhtml

```
<html>
<head>
  <title>Page: <%= controller.action_name %></title>
  <%= stylesheet_link_tag 'scaffold' %>
</head>
<body>

<p style="color: green"><%= flash[:notice] %></p>

<%= yield  %> # ←(1)

</body>
</html>
```

実際にHTMLが出力される際は、(1)のyieldの部分が、個別のrender結果に置き換わって出力されます。

テスト

Railsでは、モデルに対するテストをユニットテスト、コントローラを対象とするテストを機能テストと呼ばれています。

ユニットテスト用のフォルダとしては、test/unit、機能テスト用のフォルダとしてはtest/functionalが用意されています。そして、テスト実行に必要なデータを書いたymlファイルは、test/fixturesに入れます。

コントローラのテスト(test/functional/page_controller_test.rb)

```
def test_index
```

```
  get :index # ←(1)
  assert_response :success # ←(2)
  assert_template 'list' #   ←(3)
end
```

ここではindexメソッドをGETで呼び出し(1)、HTTPレスポンスがSuccessである事を確認し(2)、list.rhtmlがrenderされている事を確認します(3)。

```
def test_list
  get :list
  assert_response :success
  assert_template 'list'
  assert_not_nil assigns(:pages) # ←(1)
end
```

このlistアクションのテストは、基本的にtest_indexと同じですが、最後に、renderする際に@pagesが定義されているかテストしています(1)。

test/fixtures/pages.ymlは、テスト用のモデルのデータで、YAML形式で記述します。今回のアプリケーションのテスト用に、以下に変更します。

```
ruby:
  id: 1
  uri: http://www.ruby-lang.org/
  title: Ruby Programming Language
ruby_no_kai:
  id: 2
  uri: http://jp.rubyist.net/
  title: Nihon Ruby no Kai
```

[MEMO] YAMLについての詳細は、Rubyist Magazineの記事「プログラマーのためのYAML入門」
http://jp.rubyist.net/magazine/?0009-YAML を参照してください。

Step4 モデル修正

validationの追加

Pageのuriというカラムは、ブックマークするウェブサイトのURIを記録するためのものですので、以下をチェックするようにしましょう。

- URIとして正しいか?
- URIのスキーマはhttpまたはhttpsか?

ActiveRecordオブジェクトの値が、期待した値かどうか判定する機能がvalidationです。モデルのクラス内でvalidateメソッドを定義することで機能します。

またvalidateメソッドを定義する他に、ActiveRecordにはいくつかのvalidateを行うメソッドも用意されています。

モデルのvailidationのテスト

テスト駆動開発(Test Driven Development)という、テストを先に書き、その後に目的のコードを書くスタイルで開発します。

まず初めに、validateが失敗する例から始めて、その後に成功する例を見ていきましょう。では、test/unit/page_test.rbにユニットテストを書きます。

test/unit/page_test.rb

```
class PageTest < Test::Unit::TestCase
  fixtures :pages

  def test_validate
    # 不正なURI
```

```
    assert(Page.create.errors.invalid?(:uri))
    assert(Page.create(:uri => "-").errors.invalid?(:uri))
    assert(Page.create(:uri => "http:///").errors.invalid?(:uri))
    # http/https以外のスキーム
    assert(Page.create(:uri => "ftp://example.com/").errors.invalid?(:uri))
    # 正しいURI
    assert(Page.create(:uri => "http://example.com/").errors.empty?)
  end
end
```

次に、ユニットテストのみを指定してrakeコマンドを実行します。

```
$ rake test:units
（略）
  1) Failure:
test_validate(PageTest) [./test/unit/page_test.rb:8]:
<false> is not true.

1 tests, 1 assertions, 1 failures, 0 errors
```

テストの失敗を確認したら、実際にapp/models/page.rbにvalidateメソッドを使ってモデルを実装します。

app/models/page.rb

```
require "uri"

class Page < ActiveRecord::Base

  private

  def validate
    begin
      parsed_uri = URI.parse(uri)
      raise unless parsed_uri.host
      raise unless %w(http https).include?(parsed_uri.scheme)
    rescue
      errors.add(:uri, "invalid URI")
    end
  end
end
```

[MEMO] validationの詳細は「マニュアル：ActiveRecord::Validations」「マニュアル：ActiveRecord::Validations::ClassMethods」を参照してください。

また同様にrakeコマンドを使ってテストを実行します。

```
$ rake test:units
（略）
1 tests, 5 assertions, 0 failures, 0 errors
```

テストの成功を確認して、モデルのvailidationのテストは終わりです。

次にコントローラのvalidationのテストを行います。

```
$ rake test:functionals
（略）
  1) Failure:
test_create(PageControllerTest) [./test/functional/page_controller_test.rb:56]:
Expected response to be a <:redirect>, but was <200>

8 tests, 25 assertions, 1 failures, 0 errors
```

機能テストで新規ページを作成する際に失敗しているので、そこでも正しいURIで作成を試みるように修正します。

test/functional/page_controller_test.rb（一部）

```
 def test_create
  （略）
  post :create, :page => {
   :uri => "http://www.rubyonrails.org/",
   :title => "Ruby on Rails"
  }
  （略）
 end
```

rakeコマンドでテストを実行します。

```
$ rake test:functionals
（略）
8 tests, 28 assertions, 0 failures, 0 errors
```

テストの成功を確認して、コントローラのvalidationテストも終わりです。

アクセサのオーバライド

Pageモデルのtitleというカラムは、値が空の場合はかわりにURIをそのまま表示するようにしてみましょう。

ここでもテスト駆動開発のスタイルで先にtest/unit/page_test.rbから書いていきます。

test/unit/page_test.rb

```
class PageTest < Test::Unit::TestCase
 （略）
 def test_title
  page = Page.new(:uri => "http://example.com/")
  assert_equal("http://example.com/", page.title)
 end
end
```

rakeコマンドを使ってユニットテストを実行し、失敗することを確認します。

```
$ rake test:units
（略）
  1) Failure:
test_title(PageTest) [./test/unit/page_test.rb:19]:
<"http://example.com"> expected but was
<nil>.

2 tests, 6 assertions, 1 failures, 0 errors
```

カラムを取得するメソッドをオーバーライドする際は、元になる値をself.column_nameではなく、self[:column_name]のように取得します。テストが失敗する事を確認してから、app/models/page.rbにtitleメソッドを定義します。

app/models/page.rb

```
class Page < ActiveRecord::Base
  def title
   self[:title].blank? ? self[:uri] : self[:title]
 end
 （略）
end
```

rakeコマンドを使って、ユニットテストおよび機能テストが通ることを確認します。

```
$ rake
（略）
2 tests, 6 assertions, 0 failures, 0 errors
（略）
8 tests, 28 assertions, 0 failures, 0 errors
```

Step5 モデル作成（2）

つづいて「ユーザ」のモデルを作成します。ここでは、認証の実装が鍵になります。

ユーザ認証の実装

ユーザ認証を支援する方法はいろいろありますが、ここではacts_as_authenticatedプラグインを使います。

- Railsプラグインによる実装
- モデルとコントローラの雛型を作成することが可能
- 自動サインアップが可能
- クッキーによる自動ログインが可能
- 非ログイン時に、一旦ログイン画面でログインさせた後で、本来の目的のページにリダイレクトすることが可能
- メール認証を使ったアクティベーションも可能

[MEMO] Acts as Authenticatedのホームページ：
http://technoweenie.stikipad.com/plugins/show/Acts+as+Authenticated

acts_as_authenticatedプラグインのインストール

script/pluginでインストールします。

```
$ ruby script/plugin --help
（略）
$ ruby script/plugin source http://svn.techno-weenie.net/projects/plugins
$ ruby script/plugin install acts_as_authenticated
```

generateで自動生成

script/generateでauthenticatedジェネレータを呼び出します。ここでは、モデル名：user、コントローラ名：accountで作成します。

```
$ ruby script/generate authenticated --help
（略）
$ ruby script/generate authenticated User Account
      create  app/views/account
      create  app/models/user.rb
      create  app/controllers/account_controller.rb
      create  lib/authenticated_system.rb
      create  lib/authenticated_test_helper.rb
      create  test/functional/account_controller_test.rb
      create  app/helpers/account_helper.rb
      create  test/unit/user_test.rb
      create  test/fixtures/users.yml
      create  app/views/account/index.rhtml
      create  app/views/account/login.rhtml
      create  app/views/account/signup.rhtml
      create  db/migrate/002_create_users.rb
```

- userモデル
- migrationファイル
- accountコントローラ

などが自動生成されます。

migrateを実行してusersテーブルを作成します。

```
$ rake db:migrate
(in /home/rails/bookmark)
== CreateUsers: migrating ====================================================
-- create_table("users", {:force=>true})
   -> 1.0104s
== CreateUsers: migrated (1.0358s) ===========================================
```

rakeでテストを実行します。

```
$ rake
（略）
12 tests, 23 assertions, 0 failures, 0 errors
（略）
22 tests, 54 assertions, 0 failures, 0 errors
```

試してみましょう

http://localhost:3000/account/ にアクセスしてみましょう。今はまだユーザがいないので、自動的にサインアップ画面にリダイレクトされます。

Step6 モデル作成（3）

ブックマーク

```
$ ruby script/generate model Bookmark
      create  app/models/bookmark.rb
      create  test/unit/bookmark_test.rb
      create  test/fixtures/bookmarks.yml
      create  db/migrate/003_create_bookmarks.rb
```

db/migrate/003_create_bookmarks.rb

```
class CreateBookmarks < ActiveRecord::Migration
  def self.up
    create_table :bookmarks do |t|
      t.column :user_id, :integer, :null => false
      t.column :page_id, :integer, :null => false
      t.column :comment, :string, :limit => 1024
      t.column :created_at, :datetime
    end
  end

  def self.down
    drop_table :bookmarks
  end
end
```

```
$ rake db:migrate
(in /home/rails/bookmark)
== CreateBookmarks: migrating ================================================
-- create_table(:bookmarks)
   -> 0.7434s
== CreateBookmarks: migrated (0.7661s) =======================================
```

モデル間のリレーション

モデリングの章で記したように、Page、User、Bookmarkの三つのデータテーブルは、UserとBookmark、PageとBookmarkが1:多で、UserとPageはBookmarkを介して多:多の関係になっています。

app/models/page.rb

```
class Page < ActiveRecord::Base
  has_many :users, :through => :bookmarks # ←(1)
  has_many :bookmarks, :order => "created_at desc" # ←(2)
  (略)
end
```

Pageから見るとUserは複数あり、PageはBookmarkを中間テーブルとして持ちます(1)。

そして、Pageから見たbookmarkは複数あり、bookmarkモデルをcreated_atの降順で並べます(2)。

app/models/user.rb

```
class User < ActiveRecord::Base
  has_many :pages, :through => :bookmarks # ←(1)
  has_many :bookmarks, :order => "created_at desc" # ←(2)
  (略)
end
```

Userから見たPageは複数あり、UserはBookmarkを中間テーブルとして持ちます(1)。

(2)では、Userから見るとBookmarkは複数あり、Bookmarkモデルがcreated_atの降順で並ぶようにしています。

app/models/bookmark.rb

```
class Bookmark < ActiveRecord::Base
  belongs_to :user
  belongs_to :page # ←(1)

  validates_uniqueness_of :page_id, :scope => :user_id # ←(2)
end
```

BookmarkはUserとPageを参照しています(1)。また、同じuser_idを持つブックマークの中でpage_idの重複がないかどうか、validates_uniqueness_ofを用いて確認しています(2)。

モデル間のリレーションは、以下のように参照することができます。

あるユーザがブックマークしている全ページ
　　a_user.pages
あるユーザの全ブックマーク
　　a_user.bookmarks

あるページをブックマークしている全ユーザ
　　a_page.users
あるページの全ブックマーク
　　a_page.bookmarks

あるブックマークのユーザ
　　a_bookmark.user
あるブックマークのページ
　　a_bookmark.page

テスト

モデル間のリレーションのテストは、それぞれ以下のように書きます。

test/fixtures/bookmarks.yml

```
bookmark1:
  id: 1
  user_id: 1
  page_id: 1
```

```
  comment: cool
  created_at: 2006-01-02 12:34:56 +09:00
bookmark2:
  id: 2
  user_id: 2
  page_id: 1
  comment: nice
  created_at: 2006-01-03 12:34:56 +09:00
bookmark3:
  id: 3
  user_id: 2
  page_id: 2
  comment: great
  created_at: 2006-01-04 12:34:56 +09:00
```

test/unit/page_test.rb

```
class PageTest < Test::Unit::TestCase
  fixtures :bookmarks, :users, :pages

  def test_users
    assert_equal(2, pages(:ruby).users.size)
    assert_equal(1, pages(:ruby_no_kai).users.size)
  end

  def test_bookmarks
    assert_equal(2, pages(:ruby).bookmarks.size)
    assert_equal(1, pages(:ruby_no_kai).bookmarks.size)
  end
  （略）
```

test/unit/user_test.rb

```
class UserTest < Test::Unit::TestCase
  （略）
  fixtures :bookmarks, :users, :pages

  def test_pages
    assert_equal(1, users(:quentin).pages.size)
    assert_equal(2, users(:aaron).pages.size)
  end

  def test_bookmarks
    assert_equal(1, users(:quentin).bookmarks.size)
    assert_equal(2, users(:aaron).bookmarks.size)
  end
  （略）
```

test/unit/bookmark_test.rb

```
class BookmarkTest < Test::Unit::TestCase
  fixtures :bookmarks, :users, :pages

  def test_user
    assert_equal(users(:quentin), bookmarks(:bookmark1).user)
  end

  def test_page
    assert_equal(pages(:ruby), bookmarks(:bookmark1).page)
  end
end
```

[MEMO] fixturesの引数は、関連するモデルについても追加します。

rakeでテストを実行します。

```
$ rake
（略）
18 tests, 33 assertions, 0 failures, 0 errors
（略）
22 tests, 54 assertions, 0 failures, 0 errors
```

Step7 コントローラ作成（2）

ページのブックマーク一覧の表示

すでにStep3でPageControllerにshowメソッドはありますので、それを変更します。

- ページのタイトルを見出しにして、ページのURI宛のリンクにする
- ページへのブックマークの一覧を表示し、各ユーザのページへのリンクにする

app/views/page/show.rhtml

```
<h1><%= link_to(h(@page.title), h(@page.uri)) %></h1>
<ul>
<% for bookmark in @page.bookmarks -%>
  <li>
    <%= bookmark.created_at.strftime("%Y/%m/%d") %>
    <%= link_to(h(bookmark.user.login), :controller => "user",
      :action => "show", :id => bookmark.user) %> <%= h(bookmark.comment) %>
  </li>
<% end -%>
</ul>
```

[MEMO] hはHTML上特別な文字（<, >, &, "）をエスケープするメソッドです。外部から入力された値を表示する際には必ず使うようにしましょう。

機能テストを実行します。

```
$ rake test:functionals
（略）
22 tests, 54 assertions, 0 failures, 0 errors
```

ユーザのブックマーク一覧の表示

同じ要領で、UserControllerにshowメソッドを追加します。

ユーザのコントローラの雛型を作成します。

```
$ ruby script/generate controller --help
（略）
$ ruby script/generate controller User
      create  app/controllers/user_controller.rb
      create  test/functional/user_controller_test.rb
      create  app/helpers/user_helper.rb
```

機能テストを書きます。

test/functional/user_controller_test.rb

```
class UserControllerTest < Test::Unit::TestCase
```

```
  fixtures :users
  （略）
  def test_show
    get :show, :id => 1

    assert_response :success
    assert_template 'show'

    assert_not_nil assigns(:user)
    assert assigns(:user).valid?
  end
end
```

コントローラとビューを書きます。

app/controllers/user_controller.rb

```
class UserController < ApplicationController
  def show
    @user = User.find(params[:id])
  end
end
```

app/views/user/show.rhtml

```
<h1><%= h(@user.login) %>さんのブックマーク</h1>
<ul>
<% for bookmark in @user.bookmarks -%>
  <li>
    <%= bookmark.created_at.strftime("%Y/%m/%d") %>
    <%= link_to(h(bookmark.page.title), :controller => "page",
      :action => "show", :uri => bookmark.page.uri) %> <%= h(bookmark.comment) %>
  </li>
<% end -%>
</ul>
```

機能テストを実行します

```
$ rake test:functionals
（略）
23 tests, 58 assertions, 0 failures, 0 errors
```

ブックマークの追加

あるページへのブックマークの追加は、以下のような処理になります。

- すでにデータベースに存在するページへのブックマーク → ブックマークのみ作成
- データベースに存在しないページへのブックマーク → ページとブックマークを同時に作成

つまり、Pageモデルのオブジェクトの管理は、Bookmarkモデルのオブジェクトを介して行います。

コントローラの雛型の作成

```
$ ruby script/generate controller Bookmark
      create  app/controllers/bookmark_controller.rb
      create  test/functional/bookmark_controller_test.rb
      create  app/helpers/bookmark_helper.rb
```

関連するモデルの保存

あるモデルを保存する際に、関連するモデルが新規オブジェクトの場合、同時に保存されます。例えば今回のアプリケーションでブックマークを保存する場合は、そのブックマークに関連するページが新規オブジェクトであれば、ブックマークとページの両方が

同時にデータベースに書き込まれます（ユーザは、すでにログインしているユーザに関連づけられているので、新規オブジェクトになることはありません）。

実行例：

```
$ ruby script/console --help
（略）
$ ruby script/console
>> b = Bookmark.new
>> b.page = Page.new(:uri => "http://notexisting.example.com/") # ← 新規ページ
>> b.user = User.find(1) # ← 既存ユーザ
>> b.save
```

[MEMO] script/consoleで対話的にモデルの操作が行えます。

トランザクション

- 関連するオブジェクトを保存する際は、自動的にトランザクションにラップされます（今回はこちら）。
- そうでない場合はActiveRecord::Base#transactionメソッドで明示的にトランザクションを使います。

実行例

```
$ ruby script/console
>> b = Bookmark.new
>> b.page = Page.new(:uri => "http://notexisting.example.com/") # ← validなページ
>> b.save # ←ユーザなしのブックマークなので例外が起きる
ActiveRecord::StatementInvalid: PGError: ERROR:  null value in column "user_id" violates not-null
constraint
```

SQLログの抜粋

```
BEGIN
INSERT INTO pages ("uri", "title") VALUES('http://notexisting.example.com/',
'http://notexisting.example.com/')
PGError: ERROR: null value in column "user_id" violates not-null constraint
: INSERT INTO bookmarks ("page_id", "user_id", "comment", "created_at") VALUES(14, NULL, NULL,
'2006-10-16 05:55:15')
ROLLBACK
```

機能テスト

BookmarkControllerのaddメソッドで、GETによるアクセスの場合は確認画面を出し、POSTによるアクセスの場合は保存するようにします。

test/functional/bookmark_controller_test.rb

```
class BookmarkControllerTest < Test::Unit::TestCase
  def setup
    @controller = BookmarkController.new
    @request    = ActionController::TestRequest.new
    @response   = ActionController::TestResponse.new

    @request.session = ActionController::TestSession.new({:user => 1}) # ←（1）
  end

  def test_add_new_page
    get :add, :uri => "http://example.com/", :title => "example"
    assert_template "add"
    assert_equal("http://example.com/", assigns(:bookmark).page.uri)
    assert_equal("example", assigns(:bookmark).page.title)

    post :add, :uri => "http://example.com/", :title => "example"
```

```
    assert_redirected_to :controller => "user", :action => "show", :id => 1
  end

  def test_add_existing_page
    get :add, :uri => "http://jp.rubyist.net/", :title => "overrided title"
    assert_template "add"
    assert_equal("http://jp.rubyist.net/", assigns(:bookmark).page.uri)
    assert_equal("overrided title", assigns(:bookmark).page.title)

    post :add, :uri => "http://jp.rubyist.net/", :title => "overrided title"
    assert_redirected_to :controller => "user", :action => "show", :id => 1
  end

  def test_add_invalid_page
    get :add, :uri => "mailto:foo@example.com", :title => "test"
    assert_template "add"
    assert_equal("mailto:foo@example.com", assigns(:bookmark).page.uri)
    assert_equal("test", assigns(:bookmark).page.title)

    post :add, :uri => "mailto:foo@example.com", :title => "test"
    assert_template "add"
    assert(assigns(:page).errors.invalid?(:uri))
  end
end
```

ブックマークの追加は、ログインしたユーザごとの処理になるので、setupメソッドであらかじめセッション情報を設定しておきます（1）。

> [MEMO] テストのsetupメソッドは、各メソッドの実行前に毎回呼ばれます。

コントローラとビューの実装

app/controllers/bookmark_controller.rb

```
class BookmarkController < ApplicationController
  include AuthenticatedSystem
  before_filter :login_required

  def add
    @page = Page.find_by_uri(params[:uri]) || Page.new(:uri => params[:uri]) # ←（1）
    @page.title = params[:title]
    @bookmark = Bookmark.new
    @bookmark.user = current_user # ←（2）
    @bookmark.page = @page # ←（2）
    @bookmark.comment = params[:comment]
    if request.post? && (@bookmark.save! rescue false) # ←（3）
      redirect_to :controller => "user", :action => "show", :id => current_user
    else
      render(:action => "add")
    end
  end
end
```

ブラウザから受け取るパラメータのうち、uriを元にページがあるか探し、なければ新規オブジェクトを作ります（1）。つづいて、新規のブックマークオブジェクトに対して、そのページとログインしているユーザを関連付けます（2）。そして、POSTによるアクセスの場合は、ブックマークオブジェクトの保存を試み、成功したらユーザのページにリダイレクトし、失敗した場合は再度同じページを表示します（3）。

app/views/bookmark/add.rhtml

```
<h1>ブックマークの追加</h1>
```

```
<% if request.post? -%> # ←(1)
<%= error_messages_for "page" %>
<%= error_messages_for "bookmark" %>
<% end -%>

<%= secure_form_tag %>
<dl>
  <dt>URI</dt>
  <dd><%= text_field_tag "uri", @page.uri, :size => 40 %></dd> # ←(2)
  <dt>タイトル</dt>
  <dd><%= text_field_tag "title", @page.title, :size => 40 %></dd>
  <dt>コメント</dt>
  <dd><%= text_field_tag "comment", @bookmark.comment, :size => 40 %></dd>
</dl>
<p><%= submit_tag %></p>
<%= end_form_tag %>
```

(1)では、POSTアクセス時のみエラーメッセージを表示させています。(2)では、text_field_tagメソッドを使って、パラメータ名とデフォルト値と、幅に関するオプションを指定しています。

では、機能テストを実行します。

```
$ rake test:functionals
(略)
26 tests, 73 assertions, 0 failures, 0 errors
```

Step8 コントローラ作成(3)

不要なアクションメソッドとビューの削除

ページの管理はすべてブックマークを介して行いますから、Step3のscaffoldで作られたapp/controllers/page_controller.rbのindex、list、new、create、edit、update、destroyメソッドは不要なので削除します。

test/functional/page_controller_test.rbの該当するテストも削除します。

app/views/page/以下の_form.rhtml、edit.rhtml、list.rhtml、new.rhtmlも削除します。

削除したら、機能テストを実行します。

```
$ rake test:functionals
(略)
19 tests, 49 assertions, 0 failures, 0 errors
```

人気順ページ一覧の表示

PageControllerのtopメソッドで、人気順のページ一覧を出しましょう。

テストを書く

「PageControllerのtopメソッドにアクセスすると、top.rhtmlが表示され、その中に含まれるページの数は二つ」というのをテストしましょう。

test/functional/page_controller_test.rb

```
class PageControllerTest < Test::Unit::TestCase
  fixtures :pages
(略)
  def top
    get :top
    assert_template "top"
    assert_equal(2, assigns(:items).size)
  end
```

```
end
```

findのオプション

人気順のページ一覧を取得するので、bookmarksテーブルからuser_idの個数が多いpage_idを、user_idの個数の多い順に取得します。つまり、SQLで書けば以下のようになります。

```
SELECT page_id, count(user_id) AS count
  FROM bookmarks
  GROUP BY page_id
  ORDER BY count DESC
```

これを、ActiveRecordのfindメソッドで書くと、以下のようになります。

```
Bookmark.find(:all,
              :select => "page_id, count(user_id) as count",
              :group => "page_id",
              :order => "count desc")
```

このように、SQLのSELECTやGROUP BYやORDER BYに相当する:select、:group、:orderといったオプションを指定することができます。

[MEMO] 詳しくは「マニュアル:find (ActiveRecord::Base)」を参照してください。

ですので、コントローラとビューは以下のようになります。

app/controllers/page_controller.rb

```
class PageController < ApplicationController
（略）
  def top
    @items = Bookmark.find(:all,
                           :select => "page_id, count(user_id) as count",
                           :group => "page_id",
                           :order => "count desc")
  end
end
```

app/views/page/top.rhtml

```
<h1>人気順ページ一覧</h1>
<ul>
<% for item in @items -%>
  <li><%= link_to(h(item.page.title), :action => "show", :uri => item.page.uri)
%> <%= item.count %> users</li>
<% end -%>
</ul>
```

確認

では、rakeコマンドでテストを実行し、ブラウザでも確認しましょう（ http://localhost:3000/page/top ）。

Step9 パフォーマンスの改善

インデックスの追加

findする際の検索条件や並び替えによく使われるカラムについて、インデックスを追加することで、findの速度を向上させることができます。今回のアプリケーションでは、以下のカラムにインデックスを追加します。

pages
　　uri

users
　　login
bookmarks
　　user_id, page_id, created_at

インデックスの追加も、migration機能でできます。まずは、migrationスクリプトの雛型を作成します。

```
$ ruby script/generate migration add_index
     create  db/migrate/004_add_index.rb
```

作成されたファイルを以下のように編集します。

db/migrate/004_add_index.rb

```
class AddIndex < ActiveRecord::Migration
  def self.up
    add_index :pages, :uri
    add_index :users, :login
    add_index :bookmarks, :user_id
    add_index :bookmarks, :page_id
    add_index :bookmarks, :created_at
  end

  def self.down
    remove_index :pages, :uri
    remove_index :users, :login
    remove_index :bookmarks, :user_id
    remove_index :bookmarks, :page_id
    remove_index :bookmarks, :created_at
  end
end
```

では、rake db:migrateを実行します。

```
$ rake db:migrate
(in /home/rails/bookmark)
== AddIndex: migrating ========================================================
-- add_index(:pages, :uri)
   -> 0.3602s
-- add_index(:users, :login)
   -> 0.1894s
-- add_index(:bookmarks, :user_id)
   -> 0.1422s
-- add_index(:bookmarks, :page_id)
   -> 0.1597s
-- add_index(:bookmarks, :created_at)
   -> 0.1453s
== AddIndex: migrated (1.0936s) ===============================================
```

log/development.logを確認すると、以下のような行が出力されています。

```
CREATE INDEX "pages_uri_index" ON pages ("uri")
CREATE INDEX "users_login_index" ON users ("login")
CREATE INDEX "bookmarks_user_id_index" ON bookmarks ("user_id")
CREATE INDEX "bookmarks_page_id_index" ON bookmarks ("page_id")
CREATE INDEX "bookmarks_created_at_index" ON bookmarks ("created_at")
```

変更後、rakeコマンドでテストを実行しましょう。

関連するオブジェクトを同時に取得する

UserControllerのshowメソッドのこれまでの実装では、まずparams[:id]からユーザを取得し、user.bookmarksでそのユーザ

のブックマークの一覧を取得、つぎに各ブックマークに対してbookmark.pageでページを取得しています。そのため、もしあるユーザが100個のブックマークを持っている場合、1+1+100の計102回SQLが発行されます。

しかし、ActiveRecordのfindメソッドの:includeオプションを使えば、あらかじめ関連するオブジェクトを同時に取得することができますので、SQLの発行回数を減らせます。

- :includeなしにUserをfindする

```
user = User.find(1) → 1回SQL発行
user.bookmarks → 1回SQL発行
user.pages → 1回SQL発行
user.bookmarks.collect{|e| e.page} → user.bookmarks回SQL発行
```

- 「:include => [:bookmarks]」付きでUserをfindする

```
user = User.find(1) → 1回SQL発行
user.bookmarks → SQL発行なし
user.pages → 1回SQL発行
user.bookmarks.collect{|e| e.page} → user.bookmarks回SQL発行
```

- 「:include => [:pages]」付きでUserをfindする

```
user = User.find(1) → 1回SQL発行
user.bookmarks → 1回SQL発行
user.pages → SQL発行なし
user.bookmarks.collect{|e| e.page} → user.bookmarks回SQL発行
```

- 「:include => [:bookmarks, :pages]」付きでUserをfindする

```
user = User.find(1) → 1回SQL発行
user.bookmarks → SQL発行なし
user.pages → SQL発行なし
user.bookmarks.collect{|e| e.page} → user.bookmarks回SQL発行
```

- 「:include => [{:bookmarks => :page}]」付きでUserをfindする

```
user = User.find(1) → 1回SQL発行
user.bookmarks → SQL発行なし
user.pages → 1回SQL発行
user.bookmarks.collect{|e| e.page} → SQL発行なし
```

- 「:include => [:pages, {:bookmarks => :page}]」付きでUserをfindする

```
user = User.find(1) → 1回SQL発行
user.bookmarks → SQL発行なし
user.pages → SQL発行なし
user.bookmarks.collect{|e| e.page} → SQL発行なし
```

[MEMO] 詳しくは「マニュアル:find (ActiveRecord::Base)」を参照してください。

ここでは、ユーザからブックマークを取得する際に、関連するページも同時に取得するように変更します。

app/views/user/show.rhtml（一部）

```
<% for bookmark in @user.bookmarks -%>
```

↓

```
<% for bookmark in @user.bookmarks.find(:all, :include => [:page]) -%>
```

同様に、ページからブックマークを取得する際に、関連するユーザも同時に取得するように変更します。

app/views/page/show.rhtml（一部）

```
<% for bookmark in @page.bookmarks -%>
```

↓

```
<% for bookmark in @page.bookmarks.find(:all, :include => [:user]) -%>
```

変更後、rakeコマンドでテストを実行しましょう。

Step10 URIのルーティングの変更

これまでの例では、以下のようなURIにルーティングされています。

```
http://localhost:3000/コントローラ名/アクション名
http://localhost:3000/コントローラ名/アクション名/idの値
```

Railsでは、config/routes.rbを変更することで、URIのルーティングを変更することができます。

デフォルトでは、以下のようなファイルになっています（コメントは省略）。

```
ActionController::Routing::Routes.draw do |map|
  map.connect ':controller/service.wsdl', :action => 'wsdl'
  map.connect ':controller/:action/:id'
end
```

上から順に見てマッチングした最初のルールにしたがってルーティングが決定されます。つまり、"map.connect ':controller/:action/:id'"というの指定のため、上記のようなルーティングになっています。

[MEMO] 詳しくは http://wiki.rubyonrails.com/rails/pages/Routes を参照してください。

トップページを人気順ページ一覧に

まず、トップページ（ http://localhost:3000/ ）を、人気順ページ一覧に変更します。

デフォルトでは、トップページにアクセスするとpublic/index.htmlが返ります。つまり、public/以下の該当する場所に静的ファイルが存在すると、アプリケーションを介さずにそのファイルがそのまま返ってしまいますので、まずはそれを削除します。

```
$ rm public/index.html
```

つづいて、トップページへのアクセスがPageControllerのtopメソッドにルーティングされるように、config/routes.rbを変更します。

```
ActionController::Routing::Routes.draw do |map|
  map.connect "", :controller => "page", :action => "top" # ←（1）
  map.connect ':controller/:action/:id' # ←（2）
```

これで、トップページへのアクセスだけ（1）のルールが適用され、それ以外は従来どおり（2）のルールが適用されます。

ブックマーク追加のURIを短くする

ブックマーク追加のURIを、/bookmark/addから、はてなブックマークのように/addに変更します。

config/routes.rb

```
ActionController::Routing::Routes.draw do |map|
  map.connect "", :controller => "page", :action => "top"
```

```
  map.connect "add", :controller => "bookmark", :action => "add" # ←追加
  map.connect ':controller/:action/:id'
end
```

ユーザのブックマーク一覧のURIを、ID番号ではなくログイン名にする

ユーザのブックマーク一覧を表示するURIを、/user/show/1ではなく/user/quentinのように変更します。

まずテストを変更します。

test/functional/user_controller_test.rb（一部）

```
  def test_show
    get :show, :login => "quentin" # ←変更
  （略）
```

test/functional/bookmark_controller_test.rb（一部）

```
  def test_add_new_page
    （略）
    assert_redirected_to :controller => "user", :action => "show", :login => "quentin"
  end

  def test_add_existing_page
    （略）
    assert_redirected_to :controller => "user", :action => "show", :login => "quentin"
  end
```

次に、config/routes.rbを以下のように変更します。

config/routes.rb

```
ActionController::Routing::Routes.draw do |map|
  map.connect "", :controller => "page", :action => "top"
  map.connect "add", :controller => "bookmark", :action => "add"
  map.connect "user/:login", :controller => "user", :action => "show" # ←追加
  map.connect ':controller/:action/:id'
end
```

最後に、コントローラとビューを変更します。

app/controllers/user_controller.rb（一部）

```
  def show
    @user = User.find_by_login(params[:login]) # ←変更
  end
```

app/controllers/bookmark_controller.rb（一部）

```
  def add
    （略）
    if request.post? && (@bookmark.save rescue false)
      redirect_to :controller => "user", :action => "show", :login => current_user.login # ←変更
    else
      render(:action => "add")
    end
    （略）
  end
```

app/views/page/show.rhtml（一部）

```
  <li>
    <%= bookmark.created_at.strftime("%Y/%m/%d") %>
    <%= link_to(h(bookmark.user.login), :controller => "user",
      :action => "show", :login => bookmark.user.login) %> <%= h(bookmark.comment) %> # ←変更
  </li>
```

このように、redirect_toやlink_toの引数を、:id指定から:login指定に変更しています。

ページのブックマーク一覧のURIを、ID番号ではなくページのURIにする

ページのブックマーク一覧のURIを、/page/show/1ではなく、はてなブックマークのように/entry/ http://example.com のように変更します。

まずテストを変更します。

test/functional/bookmark_controller_test.rb（一部）

```
  def test_show
    get :show, :uri => "http://www.ruby-lang.org/" # ←変更
（略）
```

次に、config/routes.rbを以下のように変更します。「/」が含まれる内容をパラメータとして渡したい場合は、*uriのように「*」を付ける必要があります。

config/routes.rb

```
ActionController::Routing::Routes.draw do |map|
  map.connect "", :controller => "page", :action => "top"
  map.connect "add", :controller => "bookmark", :action => "add"
  map.connect "user/:login", :controller => "user", :action => "show"
  map.connect "entry/*uri", :controller => "page", :action => "show" # ←追加
  map.connect ':controller/:action/:id'
end
```

最後に、コントローラとビューを変更します。

app/controllers/page_controller.rb（一部）

```
  def show
    @page = Page.find_by_uri(decode(request.path.sub(%r!¥A/entry/!, ''))) # ←変更
  end
  （略）
  private

  def decode(str) # ←追加
    (str || "").gsub(/%([0-9a-f]{2})/i){|a| [$1].pack("H*")}
  end
end
```

params[:uri]でパラメータを取得しようとすると、「/」で区切った配列になってしまいますので、URIのような文字列を取得するのには使えません。そこで、request.pathでパス全体を取得してから、必要な部分を取り出して、「%xx」という表記をデコードしています。

app/views/user/show.rhtml（一部）

```
  <li>
    <%= bookmark.created_at.strftime("%Y/%m/%d") %>
    <%= link_to(h(bookmark.page.title), :controller => "page",
      :action => "show", :uri => bookmark.page.uri) %> <%= h(bookmark.comment) %> # ←変更
  </li>
```

このように、link_toの引数を、:id指定から:uri指定に変更しています。

Step11 その他の変更

AccountControllerのリダイレクト先の変更

ログイン、サインアップ、ログアウトした際のリダイレクト先が、AccountControllerのindexメソッドになっているのを、それぞれユーザのページ、ユーザのページ、トップページに変更します。

app/controllers/account_controller.rb（一部）

```
def login
  （略）
    redirect_back_or_default(:controller => '/user', :action => 'show', :login => current_user.login)
  （略）
end

def signup
  （略）
    redirect_back_or_default(:controller => '/user', :action => 'show', :login => current_user.login)
  （略）
end

def logout
  （略）
    redirect_back_or_default(:controller => '/page', :action => 'top')
  （略）
end
```

共通レイアウトの作成

HTMLの<head>タグや、ページのヘッダやフッタのように、さまざまなページで共通の部分は、app/views/layout/以下にテンプレートを置くことで共通化することができます。

デフォルトでは、app/views/layout/コントローラ名.rhtmlがあればそれを使い、なければapp/views/layout/application.rhtmlが使われます。

Step3のscaffoldで作成されたapp/views/layout/page.rhtmlがありますので、それを参考にapplication.rhtmlを削除し、元のpage.rhtmlは削除します。

app/views/layout/application.rhtml

```
<html>
<head>
  <title>Bookmark: <%= controller.action_name %></title>
  <%= stylesheet_link_tag 'scaffold' %>
</head>
<body>
<ul>
<li><%= link_to_unless_current("トップ", :controller => "page", :action => "top") %></li> # ←(1)
<% if session[:user] %> # ←(2)
<li><%= link_to("ホーム", :controller => "user", :action => "show", :login =>
User.find(session[:user]).login ) %></li>
<li><%= link_to("追加", :controller => "bookmark", :action => "add") %></li>
<li><%= link_to("ログアウト", :controller => "account", :action => "logout") %></li>
<% else %>
<li><%= link_to("ログイン", :controller => "account", :action => "login") %></li>
<li><%= link_to("サインアップ", :controller => "account", :action => "signup") %></li>
<% end %>
</ul>

<p style="color: green"><%= flash[:notice] %></p>

<%= yield  %>
```

```
</body>
</html>
```

(1)のlink_to_unless_currentメソッドは、基本的にはlink_toと同じですが、現在のページとリンク先が同じ場合はリンクにせずに文字列だけを表示します。ログインしているかどうかをif文で分岐して、表示するメニューを変えています(2)。

> [MEMO] コントローラやアクションメソッドによって使うレイアウトを変えたり、レイアウトを使わないようにしたりすることもできます。詳しくは「マニュアル：layout (ActionController::Layout::ClassMethods)」を参照してください。

Javascriptによるブックマークレット

外部から、http://localhost:3000/add?title=this_is_title;uri=http://example.com/page.rhtml のようなURIにアクセスすると、あらかじめパラメータに値をセットした状態でブックマークの追加画面に行くことができます。そこで、ブラウザ上で今見ているページからJavascriptで動的に上記のようなURIを生成するブックマークレットを作ります。

ページの移動
　　window.location=' http://...'
titleの取得
　　encodeURIComponent(document.title)
uriの取得
　　encodeURIComponent(location.href)

以上より、ブックマークレットのリンクは以下になります(一行につなげてください)。

```
javascript:window.location='http://localhost:3000/add?title='+
encodeURIComponent(document.title)+';uri='+
encodeURIComponent(location.href);
```

> [MEMO] Javascriptでエスケープする関数はescape()やencodeURI()もありますが、前者は%uXXXXのような文字列になって、WEBrickを含む一部の環境で正しく受け付けられません。また、後者は「&」や「?」をエスケープしないので、今回のようにクエリーの一部に使うのには問題があります。

セキュリティ

XSS

動的にページを生成するシステムを利用し、サイト間を横断して悪意のあるスクリプトが混入される事をXSS(Cross Site Scripting)といいます。

意図しないサイトからスクリプトが混入され、混入したスクリプトが実行されてしまう様な攻撃を防ぐためには、外部からの入力値をもとに動的にHTMLを出力する際に、適切にエスケープ処理をする必要があります。

<script>～</script>等の文字列を実行できないように、入力された値を表示するビューのコードで以下のヘルパーメソッドを使ってエスケープします。

hまたはhtml_escape
　　HTML上特別な文字(<, >, &, ")をエスケープします。
uまたはurl_encode
　　URIに使えない文字をエスケープします。

> [MEMO] XSSについての詳細は http://ja.wikipedia.org/wiki/XSS を参照してください。

CSRF

外部のページからのHTTPリクエストを受け付けるよう仕向け、ユーザーの意図しない操作をWebアプリケーション上で行わせる事をCSRF(Cross Site Request Forgeries)といいます。

意図したサイトとは異なるサイトからのリクエストの受信を拒否して防ぐには、security_extensionsというプラグインが便利です

インストール方法：

```
$ ruby script/plugin install security_extensions
```

使い方は、以下のとおりです。

- ビューで、start_form_tagメソッドの変わりにsecure_form_tagメソッドを使う
- コントローラで、verify_form_posts_have_security_tokenメソッドで検証するメソッドを指定する

今回は、BookmarkControllerの全てのPOSTリクエストを検証するために、以下のように変更します。

app/controllers/bookmark_controller.rb

```
class BookmarkController < ApplicationController
  verify_form_posts_have_security_token
  （略）
```

app/views/bookmark/add.rhtml（一部）

```
<%= secure_form_tag %> # ←変更
<dl>
（略）
```

[MEMO] CSRFについての詳細は、http://e-words.jp/w/CSRF.html を参照してください。

SQLインジェクション

SQL文を含む引数を持つHTTPリクエストを使って、データーベースシステムを不正に操作することをSQLインジェクションと言います。

SQLの中に直接パラメータを入れると、SQLインジェクション脆弱性につながりますので、ActiveRecordが提供するプレースフォルダの機能を使います。

ActiveRecordのfindメソッドでは、以下の二つの書き方ができます。

- User.find(:all, :conditions => ["name = ?", params[:name]])
- User.find(:all, :conditions => ["name = :name", :name => params[:name]])

なお、find_by_nameのようなメソッドでは、内部で自動的にエスケープされますので、使えるケースでは積極的に使うのがいいでしょう。

- User.find_by_name(params[:name]) ← OK

パラメータの改竄

```
@page = Page.new(params[:page])
```

のようなコードだと、本来フォームから入力されないはずのフィールドまで設定されてしまう可能性があります。その対策として、以下のような方法があります。

- ActiveRecord::Base.attr_protectedメソッドで、保護したいフィールドを指定します。
- ActiveRecord::Base.with_scopeメソッドで、あらかじて制限をかけます。

今回は、

```
@page.title = params[:title]
```

のように、個別に設定しているので、パラメータの改竄による問題はありません。

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）
- ライド・オン・Rails（ISBN:4797335750）
- かんたんRuby on RailsでWebアプリケーション開発（ISBN:4798111570）
- Ruby on Rails入門ー優しいRailsの育て方（ISBN:4798013951）
- 実践Ruby on Rails（ISBN:4881665413）

今後の情報源

Ruby on Railsホームページ
　　http://rubyonrails.org/
Rails Wiki
　　http://wiki.fdiary.net/rails/
Rails to you
　　http://rails2u.com/
Rubyホームページ
　　http://www.ruby-lang.org/
日本Rubyの会
　　http://jp.rubyist.net/
お問い合わせ
　　kazuhiko-iw2006@fdiary.net