**STEP3 : Sử dụng Framework Ruby on Rails , học về khái niệm quản lý version của rvm hoặc rbenv**

**(27/08/2015 – 04/09/2015)**

**Ngày 27/08/2015:**

Hôm nay nhận được schedule của sếp. Và mình còn vài ngày nữa để học Rails. Nên hôm nay mình quyết định ngồi viết lại đống tài liệu này để dễ dàng xem lại khi cần thiết.

Nhiệm vụ: Ngồi sửa lại những gì đã viết trong Step1(Ruby) và Step2(Vagrant)

**Ngày 28/08/2015**

Nhiệm vụ: Tìm hiểu sơ lược về Rails , cách rails project được sinh ra , cấu trúc của rails và tác dụng của các thành phần trong rails , mô hình MVC trong Rails là như thế nào

**Ngày 01/09/2015**

Nhiệm vụ: Hoàn thành blog sử dụng sqlite3, biết cách tạo ra controller , model , view và kiến thức xung quanh(ý nghĩa của lệnh, rails đã xây dựng nó như thế nào,…)

**Ngày 02/09/2015**

Nhiệm vụ: Sử dụng template ngoài vào Rails, sử dụng MySQL để làm 1 trang blog demo

“rmkdir

x.

+ Lki

QC:\Program Files (x86)\Git\bin

1. Rails là gì ?

David Heinemeier Hansson đã tạo ra Rails, và ông ta đặt tên cho hệ thống này là “Ruby on Rails” vì nó được viết dựa trên ngôn ngữ Ruby, dù vậy người ta vẫn thường gọi nó là “Rails”. Ruby On Rails là một framework mã nguồn mở dùng cho ứng dụng web dùng ngôn ngữ lập trình Ruby, và nó còn là một full-stack framework. Để hiểu được Rails, đầu tiên bạn nên hiểu thế nào là 1 “stack”, và thế nào là 1 “full-stack”.

Stack: là thuật ngữ dùng để chỉ một cụm các công cụ đi kèm với nhau để thực thi hoàn chỉnh một nhiệm vụ vào đó. Ví dụ như sau đây là một stack phổ biến của PHP: LAMP. Stack “LAMP” bao gồm Linux (hệ điều hành) + Apache (web server) + MySQL (hệ quản trị cơ sở dữ liệu) + PHP.

Full-stack: có nghĩa là một stack đầy đủ, không nhất thiết phải thêm công cụ khác. Và Rails là một full-stack framework (mặc dù dĩ nhiên là nó cần hệ điều hành, và người ta thường chọn các hệ điều hành có nguồn gốc Unix như Linux hay Mac). Có nghĩa là Rails bao gồm web server, các hỗ trợ kết nối cơ sở dữ liệu và nhiều thứ khác nữa.

Chính vì vậy mà Rails rất mạnh mẽ. Và nó cũng trở nên khó tự học hơn (dĩ nhiên không có nghĩa là không thể tự học :) ).

Ruby On Rails chứa đựng các “quy ước” (convention) để việc bảo trì và phối hợp giữa các lập trình viên dễ dàng hơn (collaboration and maintenance). Những quy ước được tuân theo khi viết ra các Rails API (application programming interface). Nhờ đó một lập trình viên Rails có thể dễ dàng hiểu được code của lập trình viên khác.

Đây là phần định nghĩa về Rails được mình trích từ trong chuỗi bài viết về Rails(định nghĩa và khái quát về Rails) mà mình thấy rất là kĩ.

+ <http://pingo.edu.vn/ruby-on-rails-la-gi-phan-1/> (Giới thiệu khái quát Ruby , Rails và những điểm mạnh của Rails)

+ <http://pingo.edu.vn/ruby-on-rails-la-gi-phan-2/> (Nguyên tắc hoạt động, điểm manh – yếu khi sử dụng Rails)

+ <http://pingo.edu.vn/ruby-on-rails-la-gi-phan-3/> (Nhìn Rails từ các khía cạnh khác nhau)

1. Mô hình MVC trong Rails:

Tham khảo:

+ <http://www.rubylife.jp/rails/ini/index7.html> (Nhật)

+ <http://vietonrails.com/201411-rails-va-mo-hinh-mvc/> (Việt)

Model: Đây là thành phần chứa tất cả các nghiệp vụ logic, phương thức xử lý, truy xuất database, đối tượng mô tả dữ liệu như các Class, hàm xử lý…

View: Đảm nhận việc hiển thị thông tin, tương tác với người dùng, nơi chứa tất cả các đối tượng GUI như textbox, images…Hiểu một cách đơn giản, nó là tập hợp các form hoặc các file HTML.

Controller: Giữ nhiệm vụ nhận điều hướng các yêu cầu từ người dùng và gọi đúng những phương thức xử lý chúng… Chẳng hạn thành phần này sẽ nhận request từ url và form để thao tác trực tiếp với Model.

Đây là một cách đơn giản để mô tả lại luồng sự kiện được xử lý trong MVC:

* User tương tác với View, bằng cách click vào button, user gửi yêu cầu đi.
* Controller nhận và điều hướng chúng đến đúng phương thức xử lý ở Model.
* Model nhận thông tin và thực thi các yêu cầu.
* Khi Model hoàn tất việc xử lý, View sẽ nhận kết quả từ Model và hiển thị lại cho người dùng.

1. Khởi tạo 1 project Rails

Nhớ: Thư mục làm việc chính là work/rails\_example/ là nơi chứa các project rails được tạo từ guest. Nó sẽ tương ứng với folder home/rails\_example của guest CentOS. Thư mục CentOS7 sẽ luôn đồng bộ với thư mục home/vagrant của guest.

Vậy là cài đặt môi trường Vagrant coi như đã xong (step2), giờ mình sẽ test bằng cách tạo project Rails mới. Mình sẽ tạo các project trong thư mục rails\_example được sinh ra khi sửa file Vagrantfile.

$>vagrant up # Khởi động máy ảo

$>vagrant ssh # Vào môi trường CentOS thông qua giao thức ssh

$>pwd # Kiểm tra đường dẫn

$>cd .. # Về thư mục cha

$/home/>cd rails\_example

$/home/rails\_example>rails new hellorails

Nếu muốn tạo project rails theo version thì dùng lệnh

$>rails \_x.x.x\_ new app\_name

Còn nếu ko thì dùng lệnh

$>rails new app\_name # nó sẽ tự động cài với version default

Như vậy là ta đã tạo xong 1 project rails rồi

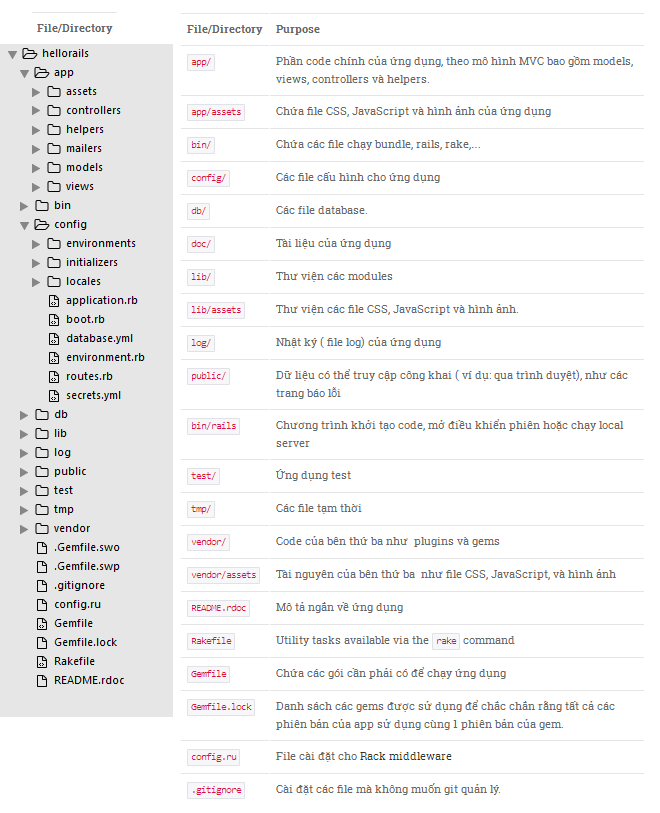
$>cd hellorails # Vào trong thư mục rails app vừa tạo

$>rails server (hoặc >rails s) # Khởi tạo server

Ra Win kiểm tra trong thư mục work/rail\_example/ có thêm folder hellorails không nhé :D

Học theo tut sau: <http://www.rubylife.jp/rails/>

1. Cấu trúc Rails Project

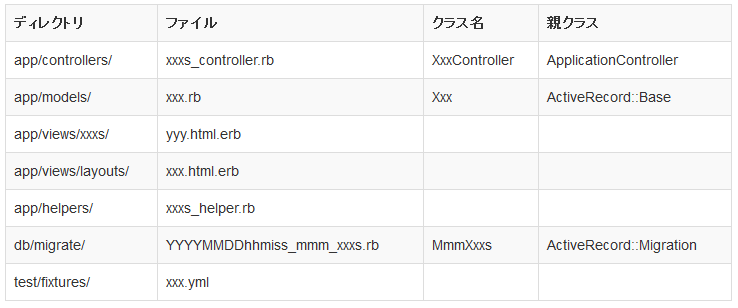


Tham khảo : <http://blog.vshtech.com/chapter1-chay-ung-dung-rails-dau-tien/>

1. Quy ước đặt tên trong Rails

Tham khảo: <http://railsdoc.com/rails_base>

Trong Rails thì có 1 số quy ước đặt tên như sau:



**Model**

Thư mục app/models là nơi chứa các model class. 1 model sẽ tuân theo quy ước :

* Tên các model class là danh từ, ở dạng số ít. Ví dụ : Entry, UserComment, …
* Tên các file chứa các model class này đặt trong thư mục app/models, tên tương ứng với model class, chuyển sang dạng viết thường có \_, ví dụ : app/models/entry.rb, app/models/user\_comment.rb, …

**Controller**

Tất cả các controller đều đặt trong thư mục app/controllers. Các quy ước về tên :

* Tên các controller ở dạng số nhiều, tương ứng với model và thêm hậu tố Controller vào phía sau. Ví dụ: EntriesController, UserCommentsController, …
* Tên các file chứa các controller này phải tương ứng, chữ thường, có \_, ví dụ: app/controllers/entries\_controller.rb, app/controllers/user\_comments\_controller.rb

**View**

Mỗi model sẽ có 1 folder giống tên của nó chứa templates trong app/views, tên các template sẽ tương ứng với tên các action trong controller, các partial sẽ có thêm tiền tố \_ ở phía trước. Ví dụ với UserComment, có các action là index, show, new, create, edit, update, destroy ( tương ứng với RESTful ), thì view sẽ là :

* app/views/user\_comments/index.html.erb
* app/views/user\_comments/show.html.erb
* app/views/user\_comments/new.html.erb
* app/views/user\_comments/edit.html.erb

create, update và destroy chỉ thực hiện lệnh thay đổi thuần tuý, không cần có view.

ERB là dạng template cơ bản và mặc định trong Rails. Nó khá đơn giản và dễ đọc với người mới. Với những người làm việc lâu, thường sẽ sử dụng HAML hoặc SLIM.

1. Chạy ví dụ đầu tiên

Tham khảo:

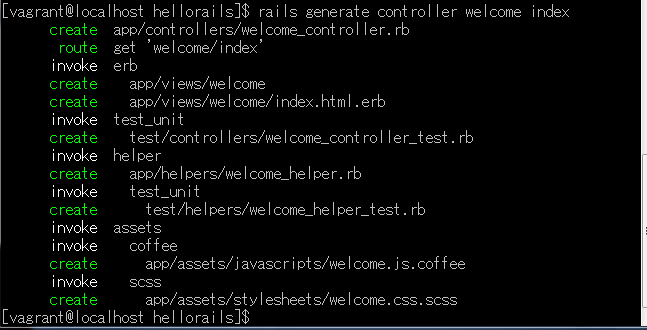
+ <https://www.youtube.com/watch?v=mlHcgeyz-CQ> (Video)

+ <http://thietkewebrubyonrails.blogspot.jp/2014/09/viet-module-dau-tien-cho-website-dung-ruby-on-rails.html> (Lý thuyết)

Thực hiện dòng lệnh để tạo controller và action

>rails generate controller welcome index

Tạo 1 controller (có tên là welcome với action index) bằng cách chạy generate và khai báo với nó rằng bạn muốn tạo 1 controller có tên là welcome và 1 action có tên index. Khi đó rails sẽ tạo ra 1 số tập tin và 1 lộ trình như sau:



Quan trong nhất trong những file này là 2 file

app/controller/welcome\_controller.rb

app/views/welcome/index.html.erb

Đây chính là 2 file trong controller và view của mô hình MVC. Tiếp theo ta sẽ đi thiết lập ứng dụng homepage

Thực thi: localhost:3000/welcome/index

Để ko cần thông qua trang chủ, ta vào config/router.rb và bỏ comment # ở dòng:

# root 'welcome#index' #=> root ‘welcome#index’

Khai báo root ‘welcome/index’ sẽ định nghĩa 1 đường dẫn ban đầu khi thực thi app. Root sẽ dẫn tới controller có tên welcome và mapping tới action tên là index.

Tạo thêm 1 action có tên create trong class WelcomeController

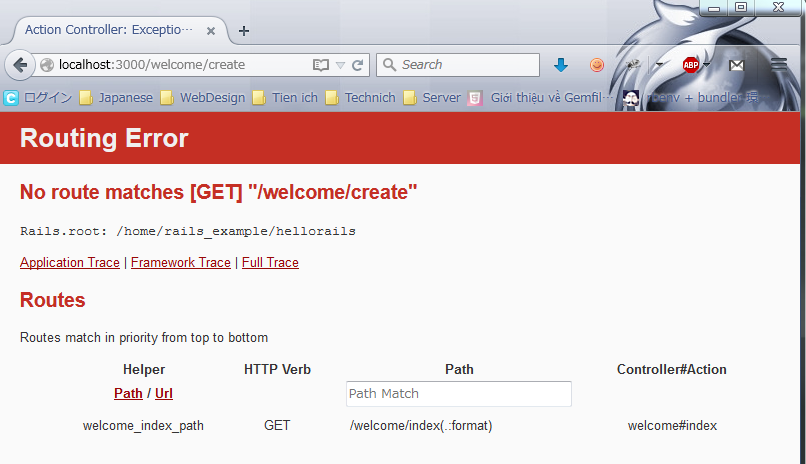
def create

end

Chạy thử: localhost:3000/welcome/create

Báo lỗi: No router matches [GET] “welcome/create”

Nguyên nhân là do nó ko có root

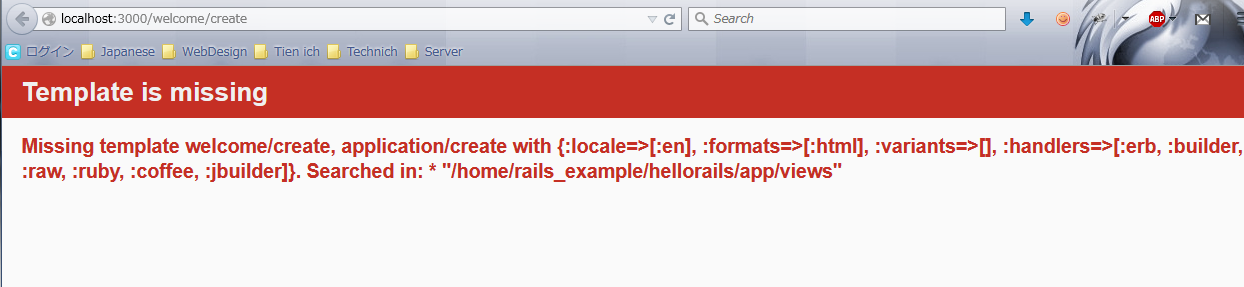


Vậy giờ ta sẽ phải vào file config và khai báo router cho nó

get ‘welcom/create’

Chạy thử: Vẫn báo lỗi thiếu Template

## Missing template articles/new, application/new with {:locale=>[:en], :formats=>[:html], :variants=>[], :handlers=>[:erb, :builder, :raw, :ruby, :coffee, :jbuilder]}. Searched in: \* "/home/rails\_example/blog/app/views"

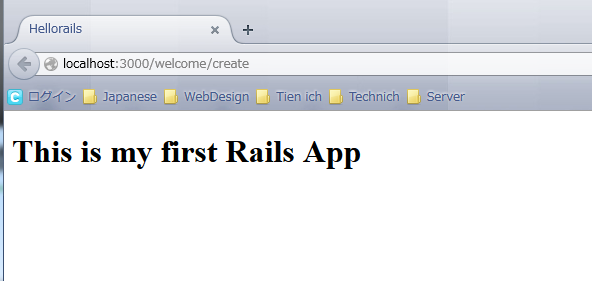


Vào view/welcome tạo thêm 1 file là create.html.erb

Nội dung

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title>Action Create Demo</title>  </head>  <body>  <h1>This is my first Rails App</h1>  </body>  </html> |

Kết quả: <http://localhost:3000/welcome/create>



Vậy là ta đã tạo được 1 model đầu tiên với Rails

Nếu trong action create có biến thì làm sao để xuất biến đó ra ngoài view ?

controllers/welcome\_controller.rb

def create

@name = “Vien Chu”

end

views/welcome/create.html.erb

<p> <%= @name %> </p>

Biến sẽ được đặt nằm trong <%= %>

Tên file đặt chữ thường tên\_controller.rb , nhưng tên của Controller thì cần đặt tên chữ hoa. Mỗi action(method) sẽ tương ứng với 1 file html.erb trong view/tên/. Nếu tên dài thì nên dùng dấu \_ để ngăn cách các từ.

Và nhớ là khi mình tạo file thì mình phải sửa router và tạo view để ko bị lỗi thiếu router và template.

OK, vậy là mình cứ hiểu qua qua như vậy nhé.

Rằng controller là nơi quyết định,xử lý mọi thứ bên trong rails app, mỗi controller sẽ có nhiều action(method), và tương ứng với nó là các view. Để controller có thể gọi đến view thì mình phải sử dụng router để điều hướng.(theo mô hình MVC)

1. Tạo blog đầu tiên với Rails

+ Nhúng mã Ruby vào trong html

<%= %>:Khai báo bên trong sẽ được hiểu là mã Ruby và được thực thi

<% %>: Khai báo bên trong cũng được hiểu là mã Ruby nhưng nó sẽ thực thi và không có output

+ Tao blog

>rails new blog

Sau khi thực thi sẽ có 1 project có tên blog được tạo ra.Bây giờ ta sẽ chú ý tới thư mục config/database.yml

Tạo database cho project

>rake db:create

Khi sử dụng SQLite thì bạn ko bắt buộc phải tạo ra database vì 1 tập tin database mới sẽ tự động được tạo ra nếu nó ko tồn tại. Vì thế mà nếu nhận thông báo “db/development.sqlite3 already exists” thì xin hãy yên tâm vì đây là dấu hiệu cho thấy tập tin sqlite đã được tìm thấy.Lệnh này khi thực thi nó sẽ tạo db theo nội dung từ file database.yml

>rake db:create:all

Nó sẽ tạo toàn bộ db được định nghĩa trong file database.yml

>rake dbconsole # Kiểm tra kết nối database

sqlitte>.exit # thoát về thư mục blog

>rake db:migrate #Apply các migration vào database của chúng ta

>rails generate model Article

Khi thực thi lệnh này thì trong folder db sẽ xuất hiện 1 folder có tên là migrate/20150831160847\_create\_articles.rb.

|  |
| --- |
| class CreateArticles < ActiveRecord::Migration  def change  create\_table :articles do |t|  t.string :title # thêm vào  t.text :body # thêm vào  t.datetime :published\_at # thêm vào  t.timestamps null: false  end  end  end |

Sau đó chạy lệnh

>rake db:migrate # Apply lại các migration vào database

Một số chú ý khi làm việc với Rails:

**1. Cách đặt têm file , class trong Rails**

Rails được phát triển theo mô hình MVC nên ta cần chú ý tới các thành phần chính trong rails app đó là: Controller , View và Model

a.Model

Tất cả các model class đều được chứa trong thư mục app/model. Và tuân thủ các quy ước đăt tên sau đây:

+ Tên các model class là các danh từ , được viết ở dạng số ít , viết hoa

vd: Friend , UserComment , Entry , ...

+ Tên file chứa model được đặt trong thư mục app/model , tên tương ứng với class và được viết thường , các từ trong tên file được phân tách bằng dấu gạch dưới "\_"

vd: friend.rb , entry.rb , user\_comment.rb , ....

b. Controller

Tất cả controller class đều được đăt trong thư mục app/controller. Và tuân thủ các quy ước đặt tên như sau:

+ Tên controller được viết ở dạng hoa , số nhiều(giống với tên model nhưng ở dạng số nhiều, còn model thì ở dạng số ít), và ta thêm hậu tố Controller vào phía sau tên.

vd: FriendsController , ItemsController , ...

+ Tên file chứa các controller này thì phải tương ứng với tên controller, nhưng ở dạng chữ thường, các từ trong tên file được tách với nhau bằng dấu gạch dưới "\_"

vd: friends\_controller.rb , items\_controller.rb

c.View

Mỗi Controller sẽ có 1 folder View tương ứng với nó chứa template bên trong thư mục app/view

Tên các template sẽ được đặt tương ứng với tên các action trong controller.

vd: Trong FriendsController có các action là: index , show , new thì view sẽ tương ứng là:

• app/views/friends/index.html.erb

• app/views/friends/show.html.erb

• app/views/friends/new.html.erb

View sẽ tương ứng với cấu trúc RESTful

Các partial(các bộ phận cục bộ) thì sẽ có thêm tiền tố "\_" đứng trước

vd: \_form.html.erb

erb là dạng template cơ bản và mặc định của Rails. Sau này nếu làm việc lâu với Rails thì còn có thêm haml và slim.

**2. Biến trong Rails**

a. Biến toàn cục (Global Variables): được đặt ở ngoài class và có thể được gọi bất cứ nơi nào,và cách viết được bắt đầu bằng dấu “$” + tên biến và tên biến thì phân biệt hoa thường.

b. Hằng số (Constrants) : được viết in toàn bộ và giá trị chỉ được gán một lần và không thể thay đổi.

c. Biến cục bộ (Local Variables): được viết thường và biến này chỉ có tác dụng trong phương thức chứa nó, nếu những phương thức không chứa nó sẽ không gọi được biến này.

d. Instance Variables: được viết bắt đầu bằng ký tự “@”, biến có tác dụng trong toàn lớp chứa nó.

**3. Trong Rails hỗ trợ chức năng có tên gọi là Scaffold**

Scaffold sẽ giúp ta tự động hình thành Controller, View và Model.

Đồng thời cũng tự động xây dựng routing giữa các thành phần ấy.

Muốn thực thi scaffold thì ta sẽ thực thi những lệnh sau đây:

>rails generate scaffold model\_name field1:data1 field2:data2 ...

vd:

rails generate scaffold friend name:string address:string

Khi thực thi lệnh này thì rails sẽ tự động sinh ra các file tương ứng với các thành phần controller , view và model

Ta chú ý đến file liên quan đến model: app/model/friend.rb và file liên quan đến db: db/migrate/20150930013715\_create\_friends.rb

Trong file db thì nó tự động khai báo 2 trường như sau:

def change

create\_table :friends do |t|

t.string :name

t.string :address

t.timestamps null: false

end

Tiếp theo để có thể build được rails app thì ta phải khởi tạo db cho nó

Khởi tạo databse

>rake db:create

Thực thi migration script. Sau khi khai báo lệnh này thì các table sẽ được tạo ra

>rake db:migrate

>rails server

**3. URL**

Đường dẫn trong brower khi triển khai rails app như sau

http://localhost:3000/controller\_name/action\_name/parametter1/parametter2

3000 là chính là port của webserver default của rails (webrick).

Nếu ta sử dụng webserver khác thì phải đổi cổng :D

**4. Controller và cách controller hoạt động**

a. Khai báo controller

Trong rails để sinh ra 1 controller ta sử dụng câu lệnh sau

>rails generate controller controller\_name

Sau khi thực thi lệnh trên thì controller sẽ tự động được sinh ra. Đồng thời folder view cũng được sinh ra cùng nó.

controller\_name có thể viết thường, viết hoa nhưng khi thực thi, rails sẽ tự động hiểu và sinh ra file\_name , controller\_class\_name theo đúng quy định.

vd:

>rails generate controller items

#=> app/items\_controller.rb , ItemsController , app/view/items/

>rails generate controller Point

#=> app/point\_controller.rb , PointController , app/view/point/

>rails generate controller rubyOnRails

#=> app/ruby\_on\_rails\_controller.rb , RubyOnRailsController , app/view/ruby\_on\_rails/

Tuy nhiên ta nên đặt tên theo đúng quy ước, tránh cho việc phát triển sau này gặp khó khăn

Ngoài ra ta có thể khai báo thêm các action

>rails generate controller controller\_name action1 action2...

vd:

>rails generate controller items new show

Như vậy khi thực thi lệnh này, ngoài việc tự động khởi tạo 1 controller có tên là items, rails cũng tự động sinh ra 2 action là new và show, và sinh ra template tương ứng với 2 action này.

b. Gọi template trong Rails

Trong Rails , các action method của controller đã tự động được định nghĩa sẽ gọi đến template tương ứng với nó sau khi action method được thực thi xong. Chính vì thế nếu thấy khai báo như làm

def index

end

mà không thấy khai báo gì nhưng nó vẫn tự động chuyển đến template app/view/friends/index

(vì bên trong nó đã được thực hiện việc render 'index' tới template index)

Nếu muốn gọi đến template của các action trong cùng 1 controller thì khai báo mã sau trong code:

render :action => 'action\_name'

hoặc

render 'action\_name'

Nếu muốn gọi đến template của các action thuộc controller khác thì khai báo mã sau trong code:

render :template => 'controller\_name/action\_name'

hoặc

render 'controller\_name/action\_name'

c. Xuất text trực tiếp từ controller

Trong trường hợp không muốn thông qua view thì ta có thể xuất text trực tiếp bằng controller với mã như sau

render :text => 'text\_muốn\_xuất\_ra'

Tuy nhiên trong 1 action method thì chỉ cho thực hiện 1 render nên nếu khai báo xuất text trực tiếp tới brower mà không thông qua view thì auto render tới view sẽ được bỏ qua(vì mặc định nó luôn nằm ở cuối method, và chỉ được thực thi sau khi thân các xử lý trong thân method đã được xử lý. Chính vì thế nếu trong thân method có khai báo render :text => "text\_output" thì nó sẽ thực hiện render này và bỏ qua render đến view )

Đó là lý do vì sao nếu ta sử dụng xuất text trực tiếp bằng controller thì sẽ ko thấy mã html trong template của view được thực hiện.

Vậy nếu ta "cố gắng" thực thi 2 render trong cùng 1 action method thì sao ?

Dĩ nhiên là Rails sẽ không cho phép việc này xảy ra và nó sẽ hiển thị lỗi "DoubleRenderError" :D

d. Không xuất gì ra màn hình

Trong trường hợp ko muốn xuất gì ra màn hình thì ta chỉ việc render nó với symbol ':nothing'- không gì cả :D

render :nothing => true

e. Xuất JSON/XML

Phần này thực ra mình cũng chẳng ứng dụng bao giờ :3

Nhưng về cơ bản thì ta cũng sẽ sử dụng mã render và thực thi như sau:

render :json => object #object chính là đối tượng json cần xuất

vd :

def index

personal = {'name' => 'Yamada', 'old' => 28}

render :json => personal

end

Tương tự như vậy, XML ta cũng có mã tương tự

render :xml => object #object là đối tượng json cần xuất dưới dạng xml

e. Redirect trong Rails

Redirect là hành động chuyển hướng đến 1 url khác. Lạ nhỉ, nếu chỉ là chuyển đến url khác thì có khác gì việc mình render nó qua template khác đâu ?

Render thì url vẫn thế, vẫn nằm ở action gọi render và nó chỉ thực thi gọi template ra, còn redirect thì sẽ chuyển url đến url được gọi hay đúng hơn là sẽ chuyển đến controller và action chứa trong url được gọi đến.

Trong trường hợp cùng controller thì ta chỉ cần gọi tên action ra như mã sau:

redirect\_to :action => 'action\_name'

Trong trường hợp khác controller thì sẽ thực thi như sau:

redirect\_to :controller => 'controller\_name'　, :action => 'action\_name'

Ngoài ra ta có thể sử dụng url để chỉ định việc redirect

redirect\_to '/url/index'

vd: redirect\_to 'http://wwww.google.com/'

Chú ý khi sử dụng redirect\_to và render đồng thời sẽ phát sinh lỗi.

Nguyên nhân thì rất dễ hiểu đó là khi redirect thì ta đã chuyển nó qua 1 action khác, nên sẽ không thể thực hiện render tới template mà lệnh render gọi đến được. Xung đột như vậy thì phát sinh lỗi là đương nhiên.

f. Truyền Parametter trong Controller

+ Lấy parametter

Khi có lời gọi parametter thì url sẽ được gọi như sau:

http://localhost:3000/controller\_name/action\_name/parametter

vd: http://localhost:3000/travels/show/10

Tuy nhiên để Rails hiểu được parametter thì ta cần khai báo tại router như sau:

match ''