

# Ruby on Rails 簡介:

吳孟勳 撰寫 胡崇偉 修訂

這篇簡介是以 Agile Web Development With Rails (2nd Edition) 裡的範例為主，簡介的內容主要是如何使用 ruby on rails (ror / rails) "從無到有"的快速開發網頁應用程式。

## install ruby and rails

首先，確定你的環境有安裝 ruby，

```
c:\ ruby -v [enter]
ruby 1.8.4 (2006-04-14) [i386-mswin32]
```

-----

```
> ruby -v [enter]
ruby 1.8.5 (2006-08-25) [i386-freebsd6]
```

沒有的話，可上 ruby 官方網站下載(<http://www.rubyonrails.org/download>)，或是 port 有(lang/ruby18)

接著確定有安裝 gem，這是一套管理 ruby 套件的程式，可以用來升級 ruby 的套件

```
c:\ gem -v [enter]
0.9.0
```

-----

```
> gem -v [enter]
0.9.0
```

沒有的話也請至<http://www.rubyonrails.org/download>下載，打ruby setup.rb

最後確定有沒有裝 ror:

```
c:\ rails [enter]
[出現使用說明...]
```

-----

```
> rails [enter]
[出現使用說明...]
```

沒有的話用 gem 安裝，打 `gem install rails`，它提示安裝 dependency 套件與否，加上 `--include-dependencies` 就不會問了

## 讓 `ror` 動起來：

新增一個空的資料夾 `\myrails` 當作 web app 的 home，接著打 `rails c:\myrails` 或 `> rails /usr/local/myrails`

`rails` 會再放其他功能進去，那些功能會在稍後解釋...

你可以試著跑看看 `ruby myrails\script\server`，它會啟動一個簡單的 web service，加上 `--help` 有詳細參數的說明，這時候你可以用 browser 看看，`ror is running!!`

連到 database:

在 `myrails\config\database.yml` 放 `ror` 連接資料庫的設定，一開始的呈現方式為：

```
# MySQL (default setup).  Versions 4.1 and 5.0 are recommended.
#
# Install the MySQL driver:
#   gem install mysql
# On MacOS X:
#   gem install mysql -- --include=/usr/local/lib
# On Windows:
#   There is no gem for Windows.  Install mysql.so from RubyForApache.
#   http://rubyforge.org/projects/rubyforapache
#
# And be sure to use new-style password hashing:
#   http://dev.mysql.com/doc/refman/5.0/en/old-client.html
development:
  adapter: mysql
  database: _ruby_development
  username: root
  password:
  host: localhost

# Warning: The database defined as 'test' will be erased and
# re-generated from your development database when you run 'rake'.
# Do not set this db to the same as development or production.
```

test:

```
adapter: mysql
database: _ruby_test
username: root
password:
host: localhost
```

production:

```
adapter: mysql
database: _ruby_production
username: root
password:
host: localhost
```

ror 啟動的模式有三種: development, test, production, 未提及的為 development...也就是在開發用的 database, ror 會開啟 debug, 速度比較慢  
test 模式 database 每次都會被清空, 要注意...  
production 模式則是正式上線用的, ror 有最佳化, 比較快

最好是 3 個模式都用不同的 database, 更改 database 設定之後, 以手動方式測試連結。

## the first page...

所有的 page 都是從 view 來的...而 view 都是 controller 決定的...  
所以新增 view, 我們先要有 controller, 執行 `ruby script\generate controller greeting` 產生第一個 controller 叫 greeting, 如下

```
c:\ ruby script\generate controller greeting
exists app/controllers/
exists app/helpers/
create app/views/greeting
exists test/functional/
create app/controllers/greeting_controller.rb
create test/functional/greeting_controller_test.rb
create app/helpers/greeting_helper.rb
```

ror 會檢查一些檔案存不存在, 有的時候會問要不要覆蓋舊檔...  
注意到 ror 新增了 3 個 .rb : controller, controller\_test, helper  
controller 對應到網址列的關係如下:

如果在 browser 打  
http://mysite.go.where.sky/greeting/hello  
那麼 greeting 的部份會對應到 controller  
hello 則對應到該 controller 的 method

而 browser 所看到的網頁則是 app/views/greeting/hello.rhtml, 如果  
hello.rhtml 不存在, ror 將出現錯誤訊息

以上所提的對應規則也都是 ror 預設的, 想要改成更複雜的規則, 可以參考  
config/下的檔案...

接著編輯 app/controllers/greeting\_controller.rb, 新增一些 method

```
class GreetingController < ApplicationController
  def hello
  end
end
```

還有該 method view, app/views/greeting/hello.rhtml

```
<h1> Hello, Welcome to RoR World!!</h1>
```

新增了 method, view 之後, 再用 browser 連連看, 立刻就看得到改變  
ps. 但是用 production mode 的時候, script/server 要重新起動.

相信看到這裡, 各位一定覺得新增 action 真是麻煩, 要重新寫一個 function, 還  
都是一個一個對應, 要是有人隨便打 action name, ror 還是會 error 嗎??  
其實 ruby 的物件裡有個 function, 專門處理不存在的 method call, 另外 action  
的對應方式也不一定是一一對應, 範例如下:

```
app/controllers/greeting_controller.rb
class GreetingController < ApplicationController
  def hello
  end
  def method_missing(mId)
    @name=mId.id2name;
    render :template => "greeting/hello"
  end
end
```

app/views/greeting/hello.rhtml

```
<h1> Hello
  <font color='#777777'>
    <%= @name %>
  </font>
, Welcome to RoR World!!</h1>
```

接著在 browser 打入 `http://127.0.0.1/greeting/<your name>` 試試看!

下一個範例就是 roR 網站應用程式

首先, 為了在線上開店, 把產品當作是 model 的一種存在 database 以供查詢, 建立名為 product 的 model -> `ruby script/generate model product`

```
c:\ ruby script\generate model product
exists  app/models/
exists  test/unit/
exists  test/fixtures/
create  app/models/product.rb
create  test/unit/product_test.rb
create  test/fixtures/products.yml
create  db/migrate
create  db/migrate/001_create_products.rb
```

model 的繼承關係是 ActiveRecord::Base 過來的, ActiveRecord 是可以把 database 的 table 轉換成物件, 所以接著要做的事情就是在 database 加一些 table:

一種方式就是直接到 database 裡去 create table, 但是那樣不會有 revision...

還有另一種方式是用 rake, 可以把它想成是 ruby 版的 make... like this:

編輯 db/migrate/001\_create\_products.rb, 新增欄位 title description image\_url 三個...

```
class CreateProducts < ActiveRecord::Migration
  def self.up
    create_table :products do |t|
      t.column :title, :string
      t.column :description, :text
      t.column :image_url, :string
    end
  end
end
```

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

接著執行 `rake db:migrate`，它會把你的 database scheme "升級"到有 product model 的樣子，也就是你可以一點一點的改變網站的 database，而且通用於 test 跟 production mode

(in mysite/)

```
== CreateProducts: migrating =====
-- create_table(:products)
-> 0.0625s
== CreateProducts: migrated (0.0656s) =====
```

要是發現加錯欄位或是想要修改 table，可以 generate 產生一個 migrate 就行了~  
(`ruby script/generate migrate add_column`)

有了 product，還需要一個 controller，所以新增 controller admin:

`c:\ruby script/generate controller admin`

編輯 `app/controller/admin_controller.rb`，加入 product model 的 scaffold

scaffold 是一種好用的管理 model 的方式，內建簡單的 CRUD (Create Read Use Delete)，以 <http://127.0.0.1:3000/admin> 這個位址測試

接著你會發現一個明顯的錯誤...就是 product 沒有標價，其實是為了示範新增 table 欄位所留的。

那用 generate 產生一個 db migration: `ruby generate migrate add_price`

`mysite> ruby script/generate migration add_price`

`exists db/migrate`

`create db/migrate/002_add_price.rb`

修改 `db/migrate/002_add_price.rb`

```
class AddPrice < ActiveRecord::Migration
```

```
  def self.up
```

```
    add_column :products, :price, :integer, :default => 0
```

```
  end
```

```

    def self.down
      remove_column :products, :price
    end
  end
end

```

再執行一次 rake db:migrate  
(in mysite/)

```

== AddPrice: migrating =====
-- add_column(:products, :price, :integer, {:default=>0})
-> 0.0443s
== AddPrice: migrated (0.0472s) =====

```

你可以試試看是不是多了 price 這個欄位

相信還是有人會覺得不夠，對那些空白欄位有許多疑惑  
對於這個問題，你可以：

編輯 app\model\product.rb

```

class Product < ActiveRecord::Base
  validates_presence_of :title, :description, :image_url      #檢查不能空白
  validates_numericality_of :price, :only_integer => true    #只能放數字
  validates_uniqueness_of :title
#title 不重複
  validates_format_of :image_url,
#match regexp, 還有錯誤訊息
                                :with => %r{\.(gif|jpg|png)$}i,
                                :message => "must be a URL for a GIF,
JPG, or PNG image"
end

```

你也許覺得 ror 內建的功能不夠豐富或是有問題，你就是堅持自己寫一個，也可以~

ror 會自動呼叫 model 的 validate method, 所以寫成這樣：

ps. 寫成 protected 以防從 url 呼叫到，所以定 controller 的 function 時要注意

```

class Product < ActiveRecord::Base
  validates_presence_of :title, :description, :image_url      #檢查不能空白
  validates_numericality_of :price, :only_integer => true    #只能放數字
  validates_uniqueness_of :title
#title 不重複

```

```

    validates_format_of :image_url,
#match regexp, 還有錯誤訊息
    :with => %r{\.(gif|jpg|png)$}i,
    :message => "must be a URL for a GIF,
JPG, or PNG image"
    protected
#自行定義 validate function,
    def validate
      errors.add(:price, "should be positive" ) if price.nil? || price <= 0
    end
  end
end

```

最後, end user 的意見還是最重要的, 他們覺得那個 product listing 實在是太難看了, 不能 show 的比較 fancy 一點嗎??

那 ror 還有一招: ruby script\generate scaffold product admin

```
mysite> ruby script/generate scaffold product admin
```

```
exists app/controllers/
```

```
exists app/helpers/
```

```
exists app/views/admin
```

```
exists test/functional/
```

```
dependency model
```

```
exists app/models/
```

```
exists test/unit/
```

```
exists test/fixtures/
```

```
skip app/models/product.rb
```

```
identical test/unit/product_test.rb
```

```
identical test/fixtures/products.yml
```

```
create app/views/admin/_form.rhtml
```

```
create app/views/admin/list.rhtml
```

```
create app/views/admin/show.rhtml
```

```
create app/views/admin/new.rhtml
```

```
create app/views/admin/edit.rhtml
```

```
overwrite app/controllers/admin_controller.rb? [Ynaq] y
```

```
force app/controllers/admin_controller.rb
```

```
overwrite test/functional/admin_controller_test.rb? [Ynaq] y
```

```
force test/functional/admin_controller_test.rb
```

```
identical app/helpers/admin_helper.rb
```

```
create app/views/layouts/admin.rhtml
```

create public/stylesheets/scaffold.css

意思是為 product，以 admin 當作 controller 產生專用的 scaffold

注意到他為 admin 新增了 5 個 view, \_from list show new edit,以及一個 layout, 跟.css

這些就是我們可以些改的部分。

到目前為止，已經來到了電子書的第 7 章(catalog display)，接下來要講的是網頁的 layout, 使用者的 session, AJAX, Authorization , email 等...

## 新增 layout

layout 有點像是網頁的排版，因為 ror 可以讓網頁的頁面更新只更新某一塊，所以其他部分就是用 layout 來管理不用更新的部分，或是同時更新也可以...