Maintaianable Rails View

Organize templates in good way

by xdite

Maintainable Rails View

Organize templates in good way

xdite

This book is for sale at http://leanpub.com/rails-view-book

This version was published on 2013-11-12

Contents

前言

相對於 View，在一個 Project 裡面，設計出乾淨的 Model 與 Controller，是相對簡單的。

但程式碼一攪和跑到 View 的層級，維護的方法就會變得相當複雜。很難有一個基礎簡單的思路去整理這些糾結的線條。

所以長久以來，我一直想要撰寫這方面的主題。從 View 的角度切入，介紹整理程式碼的技巧。

閱讀這本書前應該知道的事

要讀這本書之前，首先希望讀者知道這是在幾個「事實前提」之下撰寫的，這也是這些「整理方法」之所以被發明的原因:

- 在 View 裡面有 Logic 糾纏 (if / else & other syntax) 是不好的，這會導致 View 很難修改以及維護
- 在 View 裡面有 Logic 糾纏是不好的，這會導致 View Performance 下降 (pure logic)。
- 在 View 裡面有 Logic 糾纏是不好的，這會導致 View Performance 嚴重下降 (with data query)。而這包含在 Helper 裡面 perform data query。

這本書會包含以下幾個主題:

- Helper 使用時機
- Helper Best Pratices
- Partial 使用時機
- Partial Best Pratices
- Helper 與 Partial 之外的整理武器
- Object-Oriented View

大致上會有 18 個整理手法。

值得注意的是，這些手法是「循序漸進」的，也就是前面的手法未必是「最好」的，而是在「初期整理階段」是一個好的手法，而事情變得複雜的時候，才需要越後面的技巧去協助整理。

Helper 與 Partial

Helper 與 Partial 是 Rails 開發者在整理 View 時最常用到的工具。但卻也是開發者容易混淆使用時機的兩個主題。

什麼是 Partial

Partial 指的是局部樣板。而 Helper 指的卻是在樣板中的一些輔助方法 (Ruby Method)。這兩種都是整理又臭又長的 HTML 版面時的好工具。

一般而言, 我們會使用 Partial 去處理大段且重複的程式碼。或者是經常使用到的局部程式碼。

大段程式碼:(如 new / edit 會複用到的表單)

```
<%= form_for @post do |f| %>
  <%= render :partial => "form", :locals => { :f => f} %>
<% end %>
```

經常使用到的局部程式碼:(如 sidebar 內的區塊)

```
<div id="sidebar">
  <%= render :partial => "recent_posts" %>
  <%= render :partial => "recent_comments" %>
</div>
```

Partial 的優點

- Don’t repeat yourself(DRY) 程式碼不重複
- 程式修改會比較清楚
- Partial 樣板比較容易被重複使用

什麼是 Helper

Partial 的定位多半是被用來處理「大片 HTML 」的工具，而 Helper 卻是比較屬於需要邏輯性輸出 HTML 時用的整理工具。

一般學 Rails 常見的：

- stylsheet_link_tag
- link_to
- image_tag
- form_for 中的 f.text_field…etc

都屬於 Helper 的範疇。

你該知道的 Helper 知識

Helper 是屬於需要邏輯性輸出 HTML 時用的整理工具。這個名詞不限定於自己撰寫的 Ruby 方法，Rails 也內建許多 Helper。可以加速專案的開發。

在開始自己撰寫 Helper 之前，閱讀 Rails 已經內建哪些 Helper 也是相當重要的一件事。

它們可以：

省下開發者重造輪子的功夫

Rails 最令其他 Ruby Web Framework 羨慕的就是內建的很多方便 Helper。舉幾個省下很多重造輪子功夫的 Helper：

simple_format

可以處理使用者的內容中 \r\n 自動轉 br 和 p 的工作

auto_link

可以處理使用者的內容中若有連結就自動 link 的工作。

truncate

使用者輸入的內容若過長可以指定多少字後就自動砍掉並加入“....”

html_escape

使用者輸入的內容若有 html tag 為了怕使用者輸入惡意 tag 進行 hack。會進行自動過濾。

(Rails 3 之前要手動加 h 過濾，在 3.0.0+ 後預設 escap，不想被 escape 可以用 `raw` unescape)

這些東西若自己寫 parser 處理不知道要花費多少精力還不一定濾的徹底。

卻都是 Rails 預設內建 Helper。

綁定內建基礎建設 (最佳實踐)

Rails 內建的某些 Helper 是為了與其他更棒的基礎建設整合

stylesheet_link_tag 與 image_tag

有些開發者覺得，這東西還要用 Helper 嗎？直接貼 HTML 不是也一樣會動嗎? 有什麼差別。

差別可大了。

```
1  <%= stylesheet_link_tag "abc","def",:cache => true %>
```

這一個 Helper 可以在 production 環境時自動幫你將兩支 CSS 自動壓縮成一支 all.css。

直接實現了 Yahoo Best Practices for Speeding Up Your Web Site 中 minify HTTP reqesut 的建議。而在 Rails 3.1 之後甚至還會自動幫你 trim 與 gzip。

完全不需要去在 deploy process 中 hook 另外的 compressor 就可以達到。

image_tag

至於 image_tag 有什麼特別的地方?

```
1  <%= image_tag("xxx.jpg") %>
```

Rails 可以幫忙在 asset 自動在後面上 query string。

若網站有整合 CDN 架構時，以自動處理 invalid cache 的問題。而 Rails 也有選項可以實作 asset parallel download 的機制，一旦打開，站上的 asset 也會配合設定，亂數吐不同來源的 asset host 實做平行下載。

輕鬆就可以把網站 Scale 上去。

解決資料庫映射問題

form_for

form_for 也是 Rails 相當為人稱讚的一個利器。

Rails 的表單欄位是綁 model(db 欄位)。

開發方便 (`Post.new(params[:post]` 直接收參數做 mapping) 之外，也內建了防 CSRF (protect_from_forgery) 的防禦措施。

開發者可以在一眨眼的工夫借助框架的力量實作出業界 (performace / security) 最佳實踐。而卻不需要先修煉成架構大師。