Ruby on Rails 初學

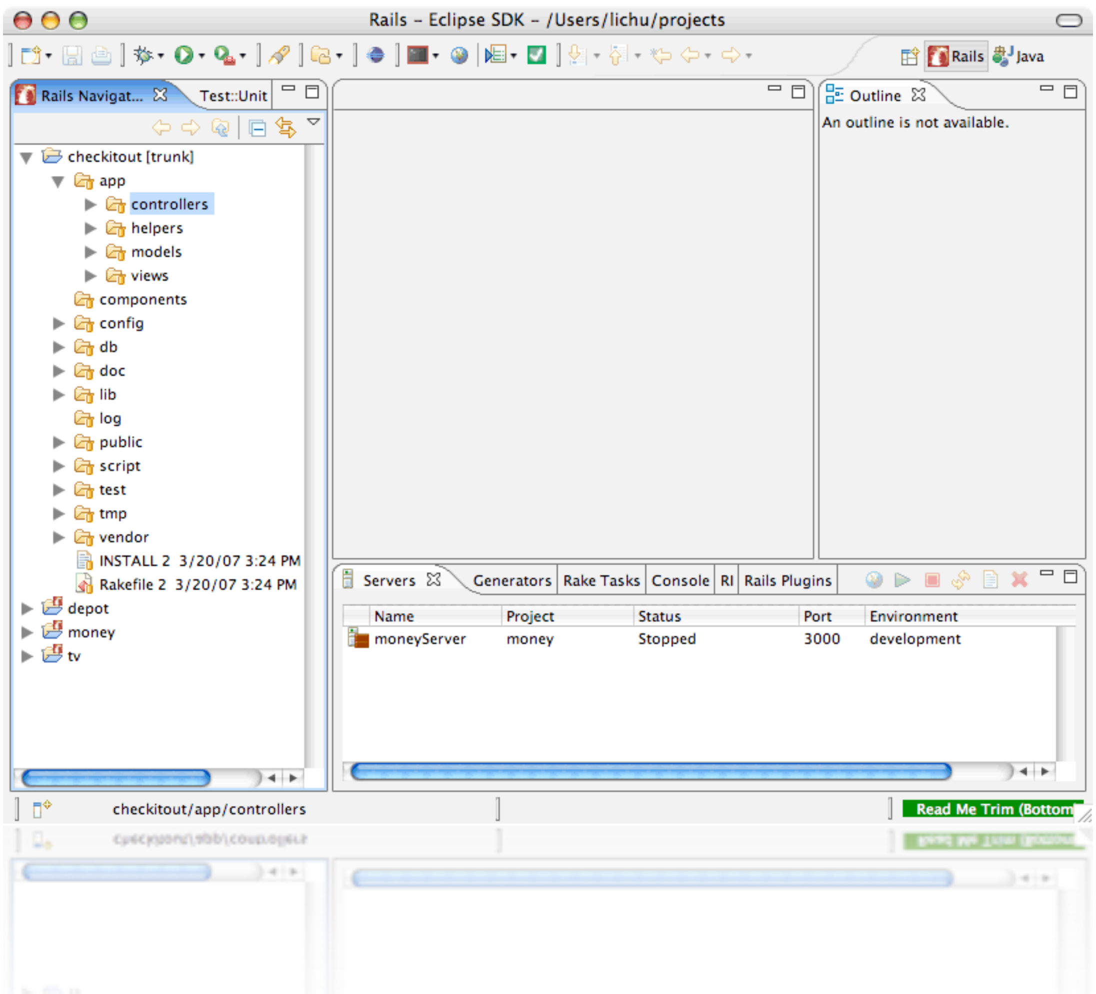
國家高速網路與計算中心 • 格網技術組 • nchcrails.blogspot.com • li@nchc.org.tw • 2007.09.06

Table of Contents

安裝 Eclipse

ECLIPSE 是什麼?

Eclipse 是一套編輯程式用的軟體工具，可用來寫很多語言的程式： Java, C++, Ruby 等等。平常寫程式會用到工具，像除錯，測試，版本控制等等，在這軟體裡就有，不用去找其他軟體。
下載 Eclipse www.eclipse.org/downloads/ (選擇 Eclipse SDK，不要選 Eclipse Distros)。



下載後,有些平台上解壓縮就可以用了，不用再安裝。要刪除的話刪整個檔案夾就可以了。
Eclipse 的使用畫面:

那一個辦法比較好呢？ 我本來是直接用 Radrails, 可是遇到了這個問題：Subclipse (一個版本管理軟體） 1.4 版 出來了，但 Radrails 獨立版包含的 Subclipse 太舊了, 使用時會出問題。要更新 Radrails 時，Radrails 那邊的人又還沒把最新的 Subclipse 包含進去他們的軟體，所以無法更新。以其等待 Radrails 的人來更新，不如自己直接用 Eclipse ，然後安裝 Subclipse 跟 Radrails 的 plugin, 如需要時再各自更新，不用等。

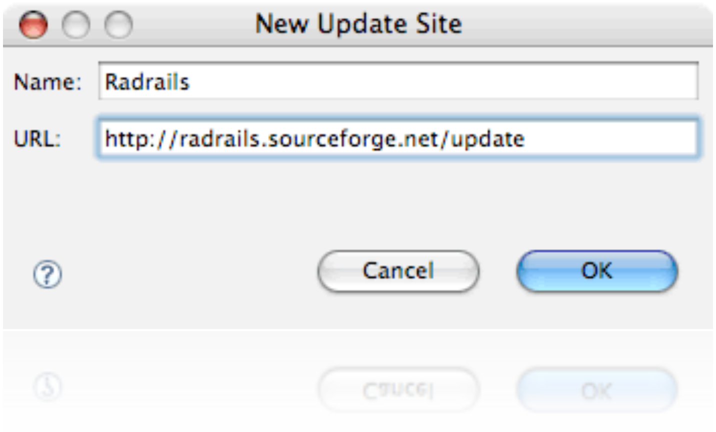
設定 ECLIPSE

打開 Eclipse 後,

Help > Software Updates > Find and Install

Search for new features to install > Next

New Remote Site



把這三個 plugin 一個一個用 New Remote Site 加入:

Radrails

http://radrails.sourceforge.net/update

RDT (Ruby)

http://updatesite.rubypeople.org/release

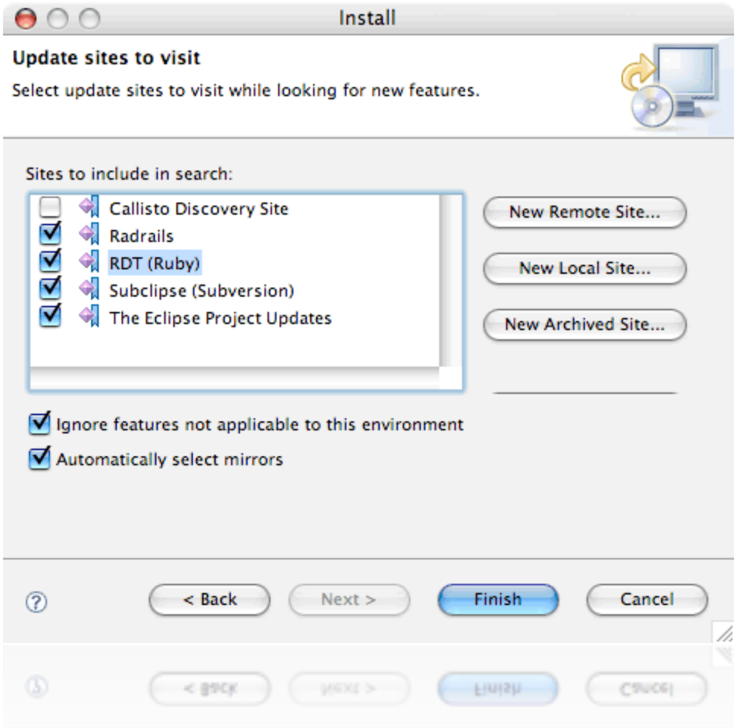
Subclipse (Subversion)

http://subclipse.tigris.org/update_1.2.x

RDT 把 Eclipse 客制化成 Ruby 語言可用的 IDE (Integrated Development Environment). Subclipse 讓 Eclipse 直接跟 Subversion 互動。Subversion 是個版本管理軟體，要深一步了解請看本書的 "Subversion 介紹" (還沒寫)。

test

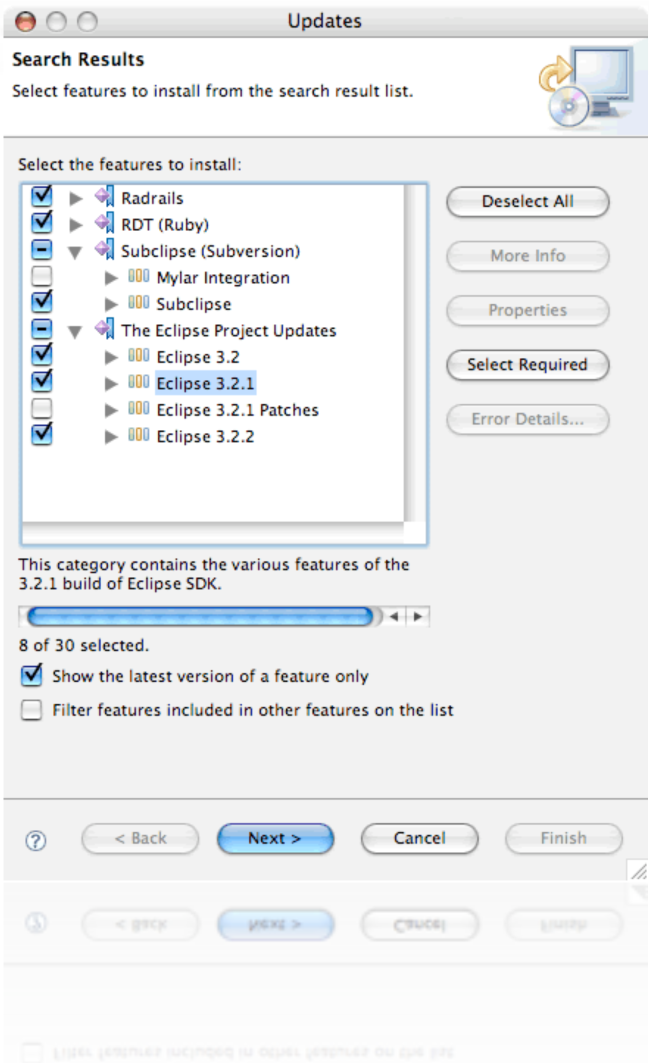
資料輸入完後選擇 "Finish"。按 Finish 前的視窗看起來應該像這樣：



Finish 按完後 Eclipse 會去找最新的軟體。有些軟體要更新前需要安裝另外一個軟體，但我們不會用到所有的軟體，所以先選擇可以 update 的軟體。不能 update 的軟體就不要管他。

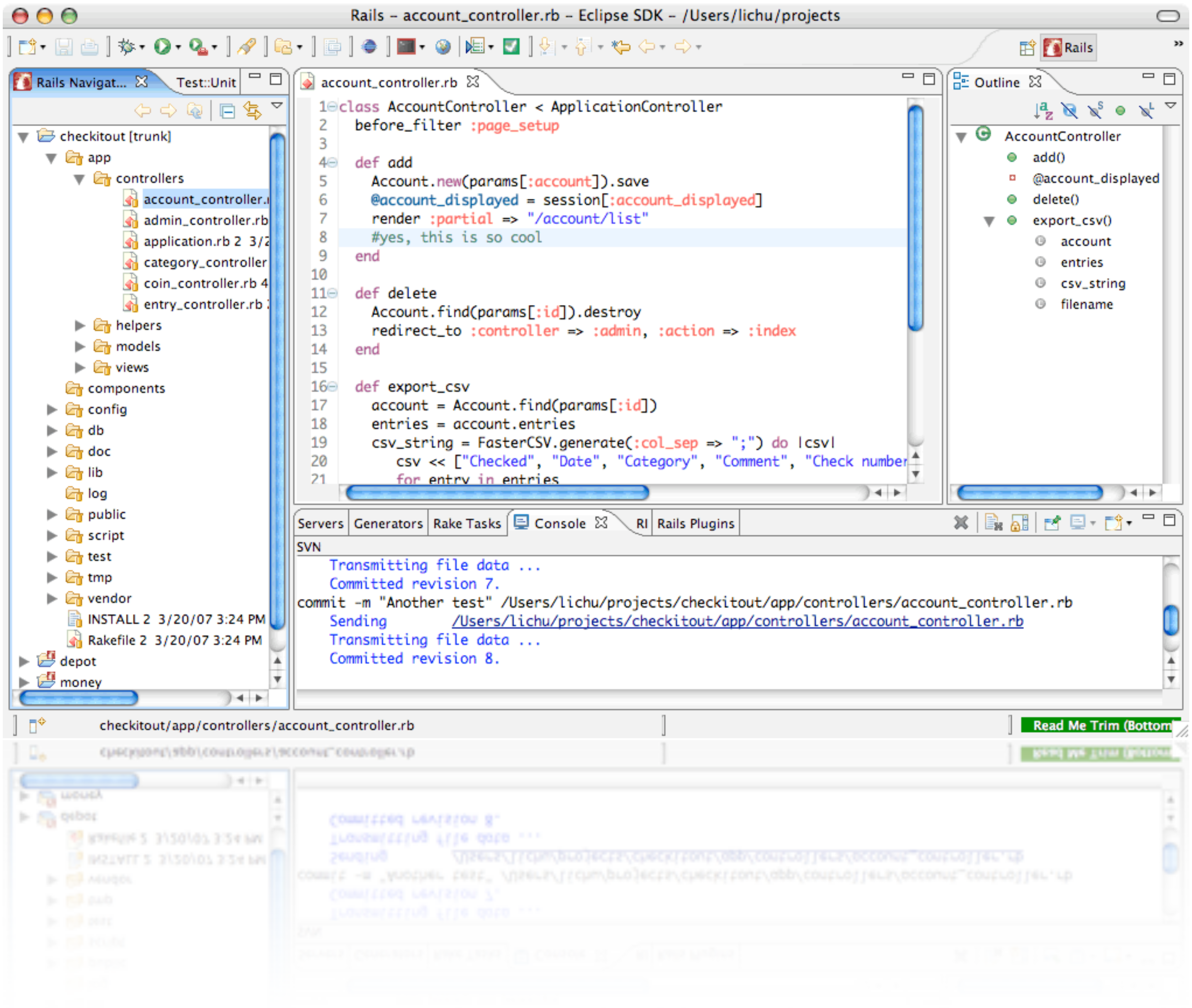
在這個範例裡，"Eclipse 3.2.1 Patches" 需要先安裝 "Eclipse 3.2.1" 才可, 所以 "Eclipse 3.2.1 Patches" 就先不要裝吧。

注意： 我們不會用到 Mylar, 所以 Mylar 也不要打勾。



下載這些 plugin 並安裝完後，到 Window > Open Perspective > Rails

這螢幕看起來就跟 Radrails 差不多一模樣囉，唯一的不同點就是在 Eclipse 上新版的 plugin 就可以自行更新。



安裝 Rails - Apple 跟 Linux

在安裝軟體之前，可以趁機考慮用蘋果的作業系統。 記得剛接觸 Rails 的時候，常常看到好多 Rails 範例裡秀的都是用 Mac OSX? 而且用的編輯器都是畫面黑黑的 Textmate? 原因是 DHH用的就是 Textmate，其他學 Rails 的也跟著用了。 Rails 社群裡常見的笑話是：
A: Textmate 好貴，一套要美金 $60 !!
B: 那才不貴呢！貴是貴在那台蘋果電腦！

因為 Textmate 只有在 OSX 上面才能跑，很多人為了要用那套軟體來寫 Rails 而跑去買了一台蘋果電腦。 會嘗試 Rails 的人，應該都是 Think Different 的人。 Think Different 當然就是蘋果電腦的座右銘。 DHH 為了用最好的工具，選擇了 Ruby 語言，也選了蘋果電腦。 在此不便開戰平台好壞論，只是要提醒，跟聰明人學習做事，自己也會變聰明喔。 我轉成使用蘋果電腦後，覺得有些作業方法在蘋果電腦上比較容易多了，用的也很開心，只是可以玩的遊戲比較少.... :)

為甚麼要自己由原始碼來 compile 呢？ Compile 的好處是，學會了這一套功夫就能打遍蘋果跟 Linux 天下，因為蘋果電腦的作業系統也是架在 unix 系統上的。 並且，安裝好後會有成就感喔。 這套安裝方法是演變 Hivelogic (http://hivelogic.com/narrative/articles/ruby-rails-mongrel-mysql-osx) 網頁上教導安裝於蘋果電腦的方法。

文章範圍

本文章會討論如何利用原始碼來安裝 Rails。 用到的電腦系統為蘋果電腦或 Linux。 安裝後的軟體有： Ruby, Ruby on Rails, Mongrel, MySQL。

/USR/LOCAL 的概念

我們會先了解 Linux 跟蘋果系統上軟體檔案應該放那裡，然後我們會安裝 Ruby 需要的工具 readline。 Readline 安裝好後就可以安裝 Ruby 了。 下一步就是安裝 RubyGems，然後利用 RubyGems 來安裝 Rails。

在安裝之前要先考慮到軟體要放哪。 由 Unix 系統來說，最好的，而且唯一的地方就是 /usr/local。 /usr/local 這個路徑是 Unix 系統指定為使用者自己編碼軟體的放置地方。 放在這裡的軟體在系統或別的軟體升級時不會被覆蓋或刪掉。

平常升級軟體時，升級程式會搜尋以前 compile 過的 binary 檔，然後把那個檔案蓋過。 這時候問題就來了。 新升級的軟體有新的功能沒錯，但是它也可能不再支援一些舊的功能了。 如果你的 Rails 程式有用到 一些 Ruby 舊的 method，但新一版的 Ruby 不支援了，那麼 Rails 程式就無法執行。 原本好好的的程式現在壞了，要從哪裡開始除錯？ 為了避免這個難搞的陷阱，還是統一把安裝的軟體放在 /usr/local 好，才不會被升級軟體隨便覆蓋。

既然 Ruby 跟 Rails 的執行檔都放在 /usr/local, 我們就得跟系統說如果要用到 Ruby 或 Rails 時，先去 /usr/local 找看有沒有 Ruby 或 Rails 的執行檔。 在你的 home folder 裡有一個隱藏的檔案叫 .bash_login。 我們修改這個檔案就可以指示系統去哪裡找了。 Unix 系統上可以用 vi 來修改:

vi ~/.bash_login

如果你要用別的軟體的話，把 vi 改成那套軟體的名字。 蘋果電腦可以用免費的軟體 TextWrangler (http://www.barebones.com/products/textwrangler/) . 打開檔案後把這行放在第一行:

export PATH="/usr/local/bin:/usr/local/sbin:$PATH"

一定要放在第一行喔，不然系統可能會跑去別的路徑找。改完後存檔並把檔案執行一次才有效:

. ~/.bash_login

/usr/local/ 這概念跟 Ruby 的檔案有什麼實際關係呢？ 當你開始寫 Rails 程式時，會看到一些檔案名稱有 .rb 結尾，這些就是 Ruby 檔案。 打開任何 Ruby 檔，就會在左上角看到 #!/usr/local/bin/ruby 。 這段字的目地是跟系統說，如要執行這一頁 script 的話，用 /usr/local/bin/ruby 目錄底下的 Ruby 程式來執行。

安裝必要的工具

XCode

先安裝 XCode。 這套軟體在蘋果系統的 DVD 可以找到。 如果沒有 DVD 的話，到 Apple Developer Connection (http://connect.apple.com/) 下載 XCode Tools。 下載後，打開 XcodeTools.mpkg 就可安裝了。 做任何軟體安裝之前都一定要先安裝這套。 現在去安裝吧，安裝好回來再繼續讀。 這個動作在 Linux 電腦上不需要。

readline

在安裝 Ruby 之前先安裝 readline。 沒安裝 readline 的話 Ruby 也安裝不起來。 我們先開個檔案夾來放原始碼。這些指令在任何的檔案夾裡都可以下：

```
sudo mkdir -p /usr/local/src
sudo chgrp admin /usr/local/src
sudo chmod -R 775 /usr/local/src
cd /usr/local/src
```

這些指令下完後你就會在 /usr/local/src 檔案夾裡。 我們會把所有的原始碼放在這個檔案夾。

Unix 上安裝軟體步驟都大致相同。 軟體原始碼可直接下指令來下載，configure 後再 compile，然後安裝。

以下的指令一行一行來輸入:

```
curl -O ftp://ftp.gnu.org/gnu/readline/readline-5.1.tar.gz
tar xzvf readline-5.1.tar.gz
cd readline-5.1
./configure --prefix=/usr/local
make
sudo make install
cd ..
```

來看看上面每一行是做什麼:

1. 下載壓縮過的 readline 的原始檔
2. 解壓縮
3. 進去解壓縮後的檔案夾
4. 把路徑設為 /usr/local
5. compile 原始檔
6. 安裝軟體。 你要有 superuser 的權限才可安裝。 sudo (SuperUser Do) 讓你輸入 root 的密碼。
7. 跳出檔案夾

這些步驟正如剛剛所說的下載，configure，compile，安裝。

在蘋果電腦上如果你有下面著些錯誤訊息就表示你沒有安裝 Xcode:

```
checking whether make sets $(MAKE)... no
checking for gcc... no
checking for cc... no
```

checking for cc... no
checking for cl... no
configure: error: no acceptable C compiler found in $PATH

安裝 RUBY

安裝 Ruby 的步驟也跟 readline 的幾乎一模一樣，只是後面有多加了一個安裝 documentation 的步驟。

```
curl -O ftp://ftp.ruby-lang.org/pub/ruby/1.8/ruby-1.8.6.tar.gz
tar xzvf ruby-1.8.6.tar.gz
cd ruby-1.8.6
./configure --prefix=/usr/local --enable-pthread --with-readline-dir=/usr/local
make
sudo make install
sudo make install-doc
cd ..
```

安裝 RUBYGEMS

RubyGems 是用來安裝 Ruby 寫出來的程式。 因為 Rails 是用 Ruby 寫出來的，所以也可以用 RubyGems 來安裝。 但安裝 Rails 之前，就得先安裝 RubyGems。 下這些指令：

```
curl -O http://files.rubyforge.mmmultiworks.com/rubygems/rubygems-0.9.2.tgz
tar xzvf rubygems-0.9.2.tgz
cd rubygems-0.9.2
sudo /usr/local/bin/ruby setup.rb
cd ..
```

這段如安裝不成功就表示 path 沒設好，請再參照上面 "/usr/local 的概念"

安裝 RAILS

RubyGems 安裝好後下這個指令就可安裝 Rails 了:

```
sudo gem install rails --include-dependencies
```

這就是有了 RubyGems 方便的地方。以後有關 Ruby 的程式用同樣的方法也可以安裝。

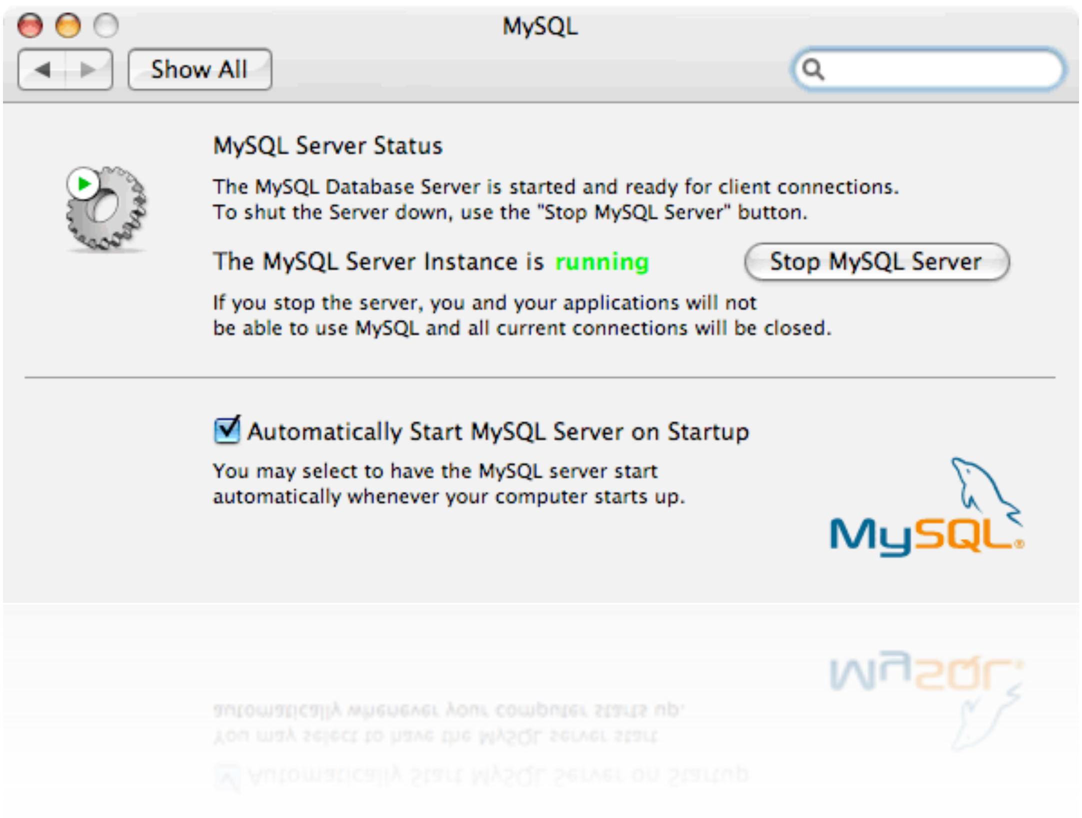
如果你有以下的錯誤訊息，就表示最前面的 path 沒設好，請回到 "/usr/local 的概念" 再重設一次。

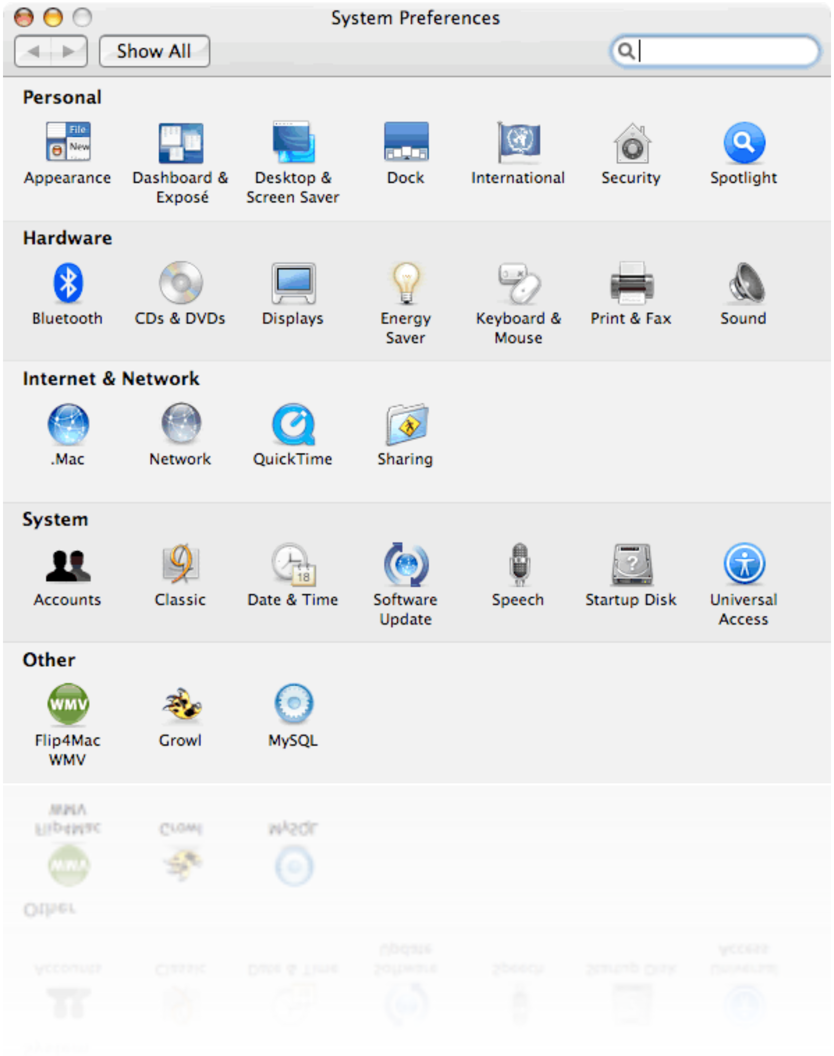
/usr/local/bin/gem:3:in `require': No such file to load -- rubygems (LoadError)
from /usr/local/bin/gem:3

安裝MYSQL 資料庫

因為 MySQL 自己有替不同的平台做最佳化，所以我們不用自己 compile. 到 http://www.mysql.org/downloads/mysql/ 來下載你的作業系統的版本後安裝。 安裝後使用者的名字是 root，密碼沒有設。 如果是蘋果電腦的話，要注意他有兩套安裝軟體: PowerPC 跟 x86. x86 指的就是最新一代用 Intel Processor 的電腦。

下載後，在蘋果電腦上除了安裝 MySQL 之外 (看起來應該像 mysql-5.0.37-osx10.4-i686.pkg)，要安裝下載檔裡的 MySQLStartupItem.pkg 和 MySQL.prefPane。 MySQLStartupItem.pkg 會讓電腦開動時就跑 MySQL。 MySQL.prefPane 會把 MySQL 加到 System Preferences 上: (看最下面那排)





安裝 MYSQL 跟 RUBY 的連結碼 (BINDINGS)

這個連結碼可以不裝，但裝了會快10-15%:

sudo gem install mysql -- --with-mysql-dir=/usr/local/mysql

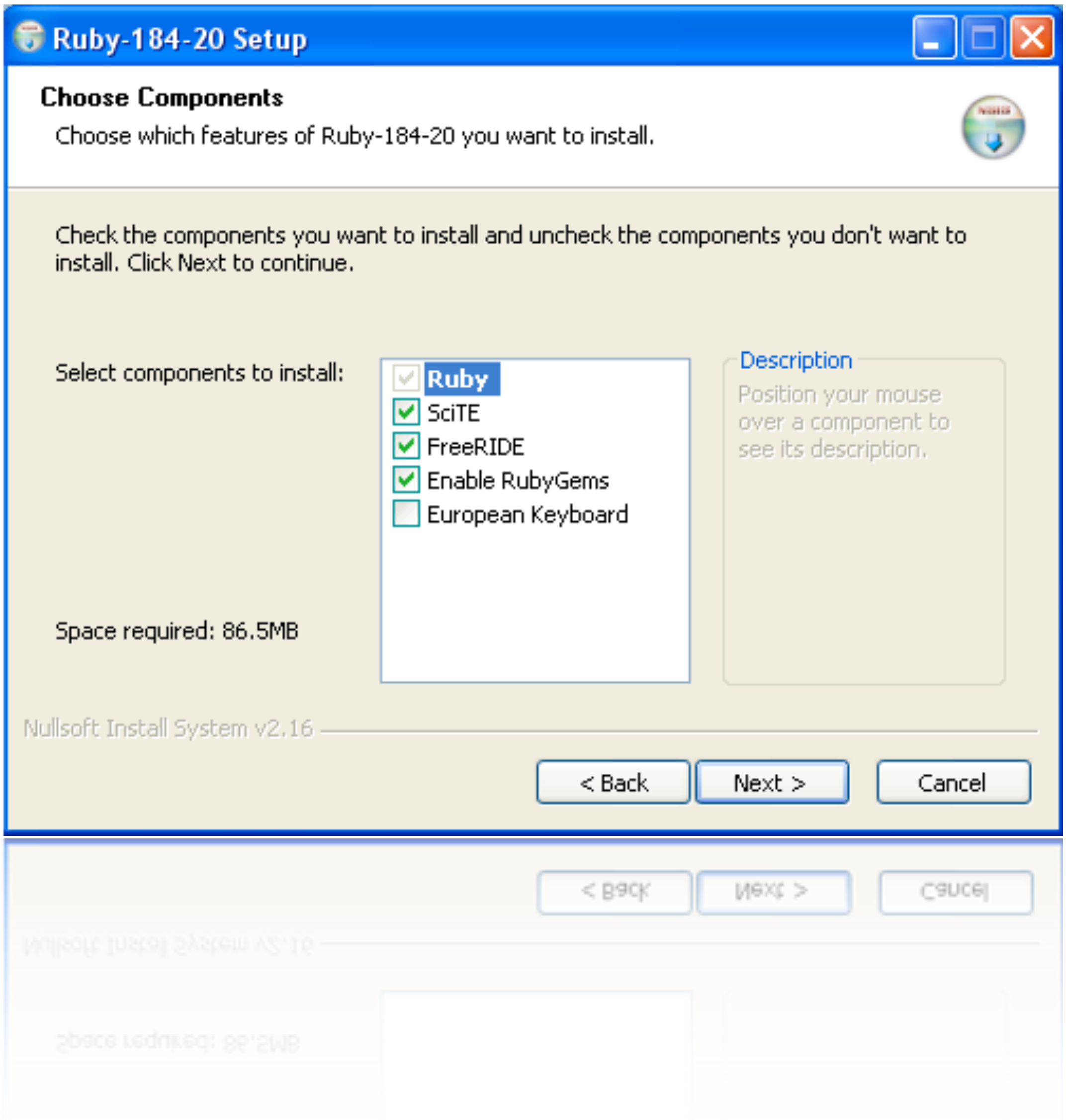
這個指令下完後就安裝好了。 如果電腦有問你要裝哪一版本的話，微軟的要選 mswin32**，其他的要選** mysql 2.7 **或是更新的版本。**

安裝 Rails - Windows 平台

要讓 Rails 跑起來需要三大軟體。 第一是 Ruby 語言軟體，第二是用 Ruby 寫出來的 Rails 程式，第三是資料庫軟體。 因為 Rails 有內建的 “mongrel” 伺服器，所以開發軟體時不用安裝伺服器。 資料庫的話，就可以選擇安裝不同的資料庫，但我會與 MySQL 為主。

安裝 RUBY

在 Windows 下安裝 Ruby 很簡單。 到 http://www.ruby-lang.org/en/downloads/ 。 目前最新版是 1.8.6。 下載 ″Ruby 1.8.6 One-Click Installer″ 後安裝。 安裝時可以選不同的模組。



″Enable RubyGems″ 一定要選，SciTE 和 FreeRIDE 是 Ruby 的開發工具，用用看也不錯。 安裝後會有個新軟體叫 ″fxri - Interactive Ruby Help and Console″。 這個工具很有用，可以用來查 Ruby Class 的 method， 也可以測試一小段的程式看他跑出來的東西符不符合你的期望。

安裝 RAILS

剛剛安裝 Ruby 時，我們有勾選 ″Enable RubyGems″，所以 RubyGems 也順便被安裝了。 RubyGems 是個很好用的工具，它的用途是安裝 Ruby 寫出來的軟體。 打開 command prompt， 直接下指令:

gem install rails --include-dependencies

整套 rails 軟體就會被下載並安裝好。 因為 rails 是由好幾個模組組成 (activesupport, activerrecord, actionpack...等等)，--include-dependencies 這字串會把所有必要的模組也安裝。 沒有打這個字串的話，他每個模組都會問你要不要裝。

測試是否安裝成功

要測試是否安裝成功都是下這樣的指令：

ruby –v

如螢幕回覆:

ruby 1.8.4 (2006-04-14) [i686-linux]

就表示安裝 Ruby 成功。 指令裡的 -v 就是 version (版本) 的意思。 螢幕回覆裡的1.8.4 就是現有安裝的版本。 [i686-linux] 就是指作業系統是 linux。 當然啦，別的系統就會顯示自己的系統名稱。

要看 Rails 否安裝成功，下此指令：

rails –v

相對的反應該是：

Rails 1.1.6

這樣的話，安裝 Ruby 跟 Rails 就大功告成了！

安裝 MYSQL 資料庫

Windows 上安裝 MySQL 超簡單的啦。 到 http://www.mysql.org/downloads/mysql/ 來下載你的作業系統的版本後安裝。 安裝後使用者的名字是 root，密碼沒有設。

安裝 Rails - 套裝軟體

一個 Rails 程式要在一台電腦上跑起，需要很多軟體，像 Ruby 語言軟體，資料庫軟體，伺服器軟體，等等。 資料庫要用什麼軟體？ 伺服器呢？ 開發套裝軟體的作者就是要解除這些煩惱，把所有 Rails 所需要的東西包在一個安裝檔裡。 使用者按兩三下後，資料庫，伺服器等等都會幫你架好。

套裝軟體雖然方便，它們可不是十全十美。 資料庫跟伺服器的軟體就得使用套裝軟體作者所決定的軟體。 而且，自己寫的程式出問題時就很難知道是套裝軟體的設定錯誤，還是自己哪裡寫錯碼。 去網上找答案時，人家給的解決方案都是要去改設定值。 但因裝的是整套軟體，設定值的位子跟人家不同，又怕改錯地方會影響套裝軟體需要的設定，所以又要花時間瞭解套裝軟體的架構。 除錯除了半天，如果是套裝軟體寫的不好，冤頭就大了。 如果已後要用的資料庫套裝軟體不支援，那還不是要自己重頭裝起？ 還是現在學會了安裝各自不同 Rails 所需的軟體，以後哪一部份需要修改或替換比較容易。

說了一大堆，如果還是堅持要用套裝軟體的話，就繼續讀囉：

文章範圍

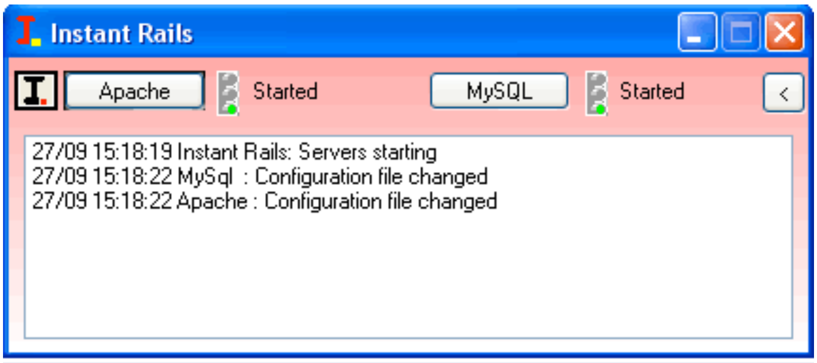
Ruby on Rails 的安裝，利用包裝好的套裝軟體。本文章討論到微軟 (instantRails) 跟蘋果 (Locomotive) 的套裝軟體。

* 注意：本人不建議用套裝軟體，有空的話自行安裝以後問題比較少 *

WINDOWS 系統： INSTANT RAILS

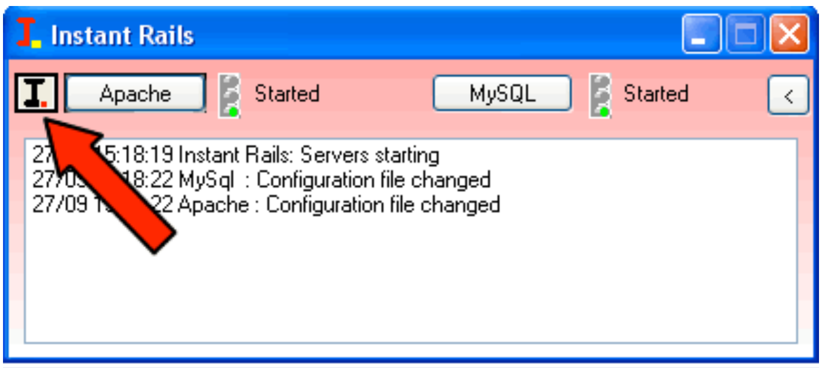
Instant Rails (http://rubyforge.org/projects/instantrails/) 是專門給 Windows 用的套裝軟體。 從官方網站下載檔案後，解壓縮到一個檔案夾裡，檔案夾的路徑不要有空格。 啟動裡面的 InstantRails.exe 就一大堆軟體可用了！ 可用的軟體有 Ruby, Rails, Apache, 和 MySQL。 另外還有一些可嘗試看看的軟體，像 Typo, SCGI, Mongrel, 和 HighLine。 InstantRails 的好處是它不會改變你的環境變數。 不要用它時把整個檔案夾刪掉就好了。

第一次啟動 InstantRails.exe 時他會問說要不要更新 path，選 ″要″。 更新完畢後會有一個視窗跑出來顯示 Apache 跟 MySQL 都已啟動。



你的電腦以前如果有裝過 Apache 的話可能會和 Instant Rails 衝突到。 因為衝突原因有好幾種，如關閉以前有的 Apache 解決不了問題只好自行上網求救了。

接下來要啟動兩個 Rails 的範例程式，以便測試是否已經成功安裝 InstantRails。 第一個是簡單的範例叫 cookbook, 另一個是很好用的部落格引擎叫 Typo。 在 instantRails 視窗的左上角，″Apache″ 按鈕左邊有一個 ″I″ 的圖形



圖形按下後選 Configure > Windows Host file。 在跳出的編輯器裡加這兩行在最後頭：

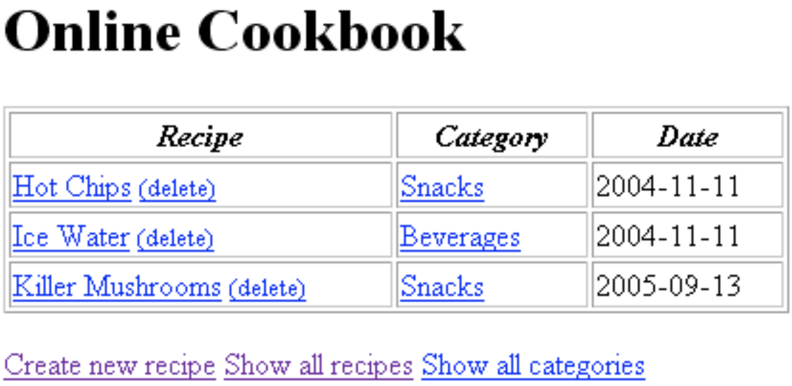
127.0.0.1 www.mycookbook.com

127.0.0.1 typo

這動作的目的是要叫 Apache 在本機找這兩個網站。

存檔後回到 instantRails 視窗，剛剛那個 i 圖樣的按鈕按一下，選擇 Rails Applications > Manage Rails Applications。 在跳出的視窗選擇 ″cookbook″ 跟 ″typo-2.6.0″，然後按 ″Start SCGI Server″。 這時後會有兩個 Command Prompt 的視窗縮小在螢幕下方。 它們不會顯示任何訊息，但他們代表著兩個在跑的網路程式。

現在到 http://www.mycookbook.com/ 會看到:



Online Cookbook

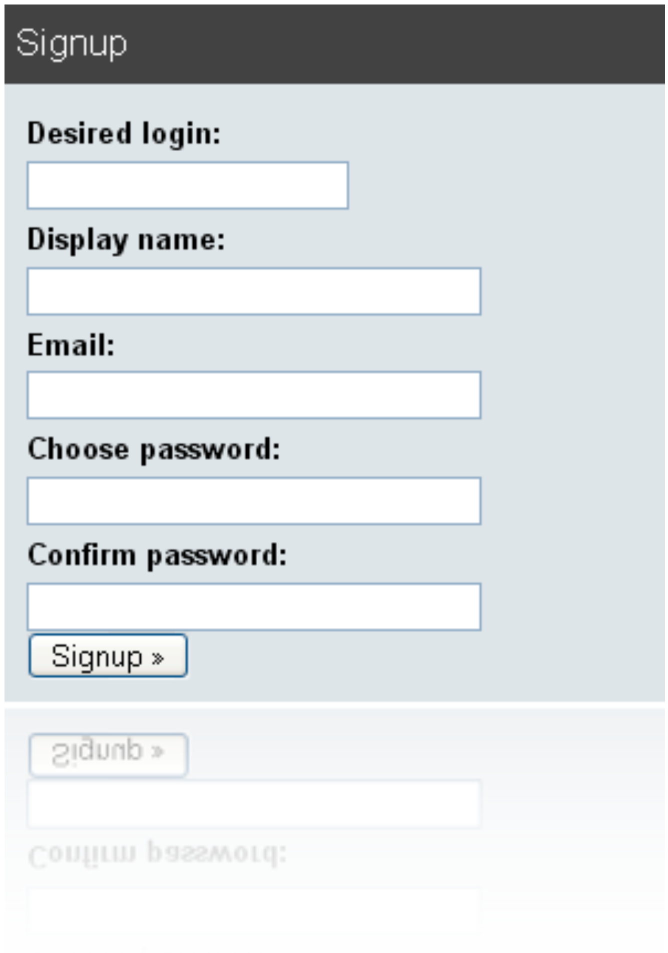
Recipe	Category	Date
Hot Chips (delete)	Snacks	2004-11-11
Ice Water (delete)	Beverages	2004-11-11
Killer Mushrooms (delete)	Snacks	2005-09-13

Create new recipe Show all recipes Show all categories

裡面可以新增，刪除食譜等等。 自己可以試試看。 網路上有篇文章一步一步來指導如何寫這個程式: http://instantrails.rubyforge.org/tutorial/index.html

另外一個範例程式叫 Typo。 如果連到 http://typo 就會到從你電腦裡伺服的 Typo 網站。 它會先要你設管理員的名稱，密碼等等。

Typo 是個很好用的部落格引擎。

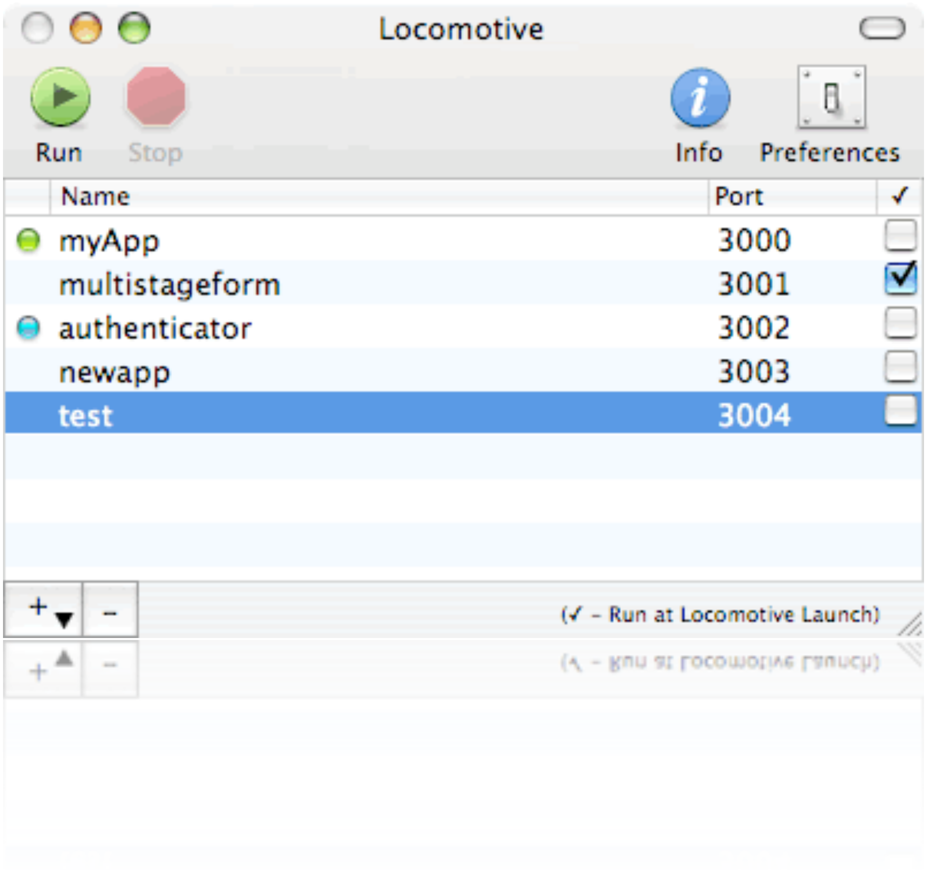


套裝軟體安裝 - 蘋果系統: LOCOMOTIVE

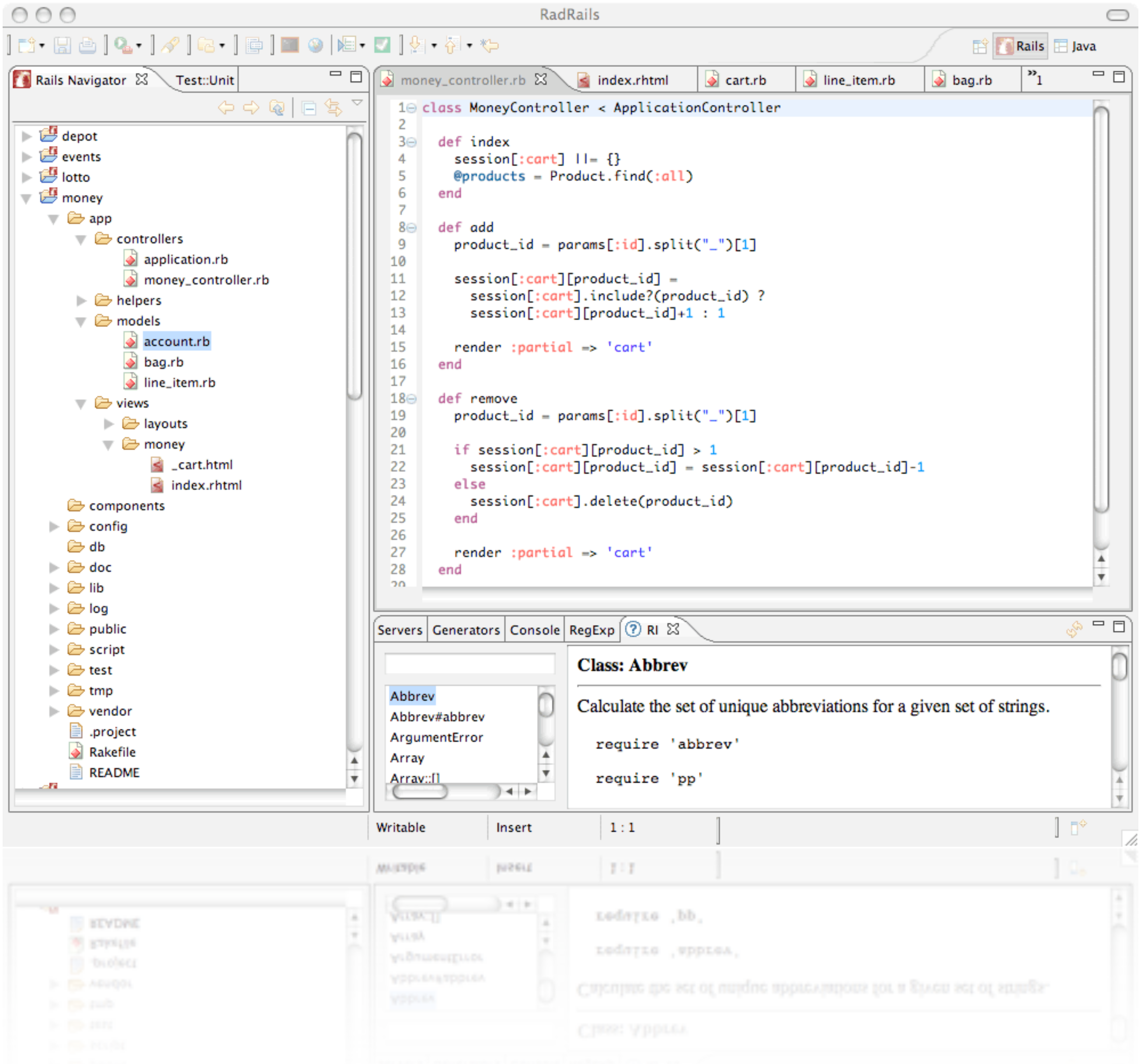
Locomotive 是一個在蘋果電腦上面架設 Ruby on Rails 的套裝軟體。 它的誕生是為了方便蘋果電腦上面 Ruby on Rails 的開發，感覺很像是 Windows 上面的 Appserv。

只要下載它的執行檔，然後直接執行，它就會在蘋果電腦上面灌好 Ruby, Rails，Lighttpd，fastcgi，SQLite，MySQL，和 PostgreSQL 的 Ruby 連結（並非資料庫本身）。

幾乎只要下載 Locomtive，點兩下安裝，它就會把整個環境設定到好，可以說是相當的方便。除此之外，它可以幫你監看所有執行中的 Rails 程式，啟動的 port 以及執行的環境模式（上線，測試，或開發模式）。 進而減少研發期的機器管理時間，讓你專心於程式開發。 如安裝成功的話，就可以看到 Locomotive 的程式監控畫面。



Radrails 簡單介紹



Radrails 的介面跟其它的 IDE 大同小異，左邊可以管理檔案，右邊可以編碼。 有些 IDE 一次只讓你管理一個專案裡的檔案，但 Radrails 可以管理在同一個目錄下不同專案的檔案。 這個好處是你可以把別人的開放軟體碼程式下載下來，放在同一個專案檔案夾裡，然後邊寫程式邊參考別人怎麼寫的。

在 Radrails 的左邊有一個視窗叫 "Rails Navigator"。 這個視窗顯示目前工作中的專案。 如要加寫到一半的專案，到 File > New > General > Project > Next 。

"use default location" 應該是打勾的，那下面顯示的路徑就是放所有專案的檔案夾。 例如我把所有的專案都放在 /Users/lichu/projects ，路徑就顯示如此。

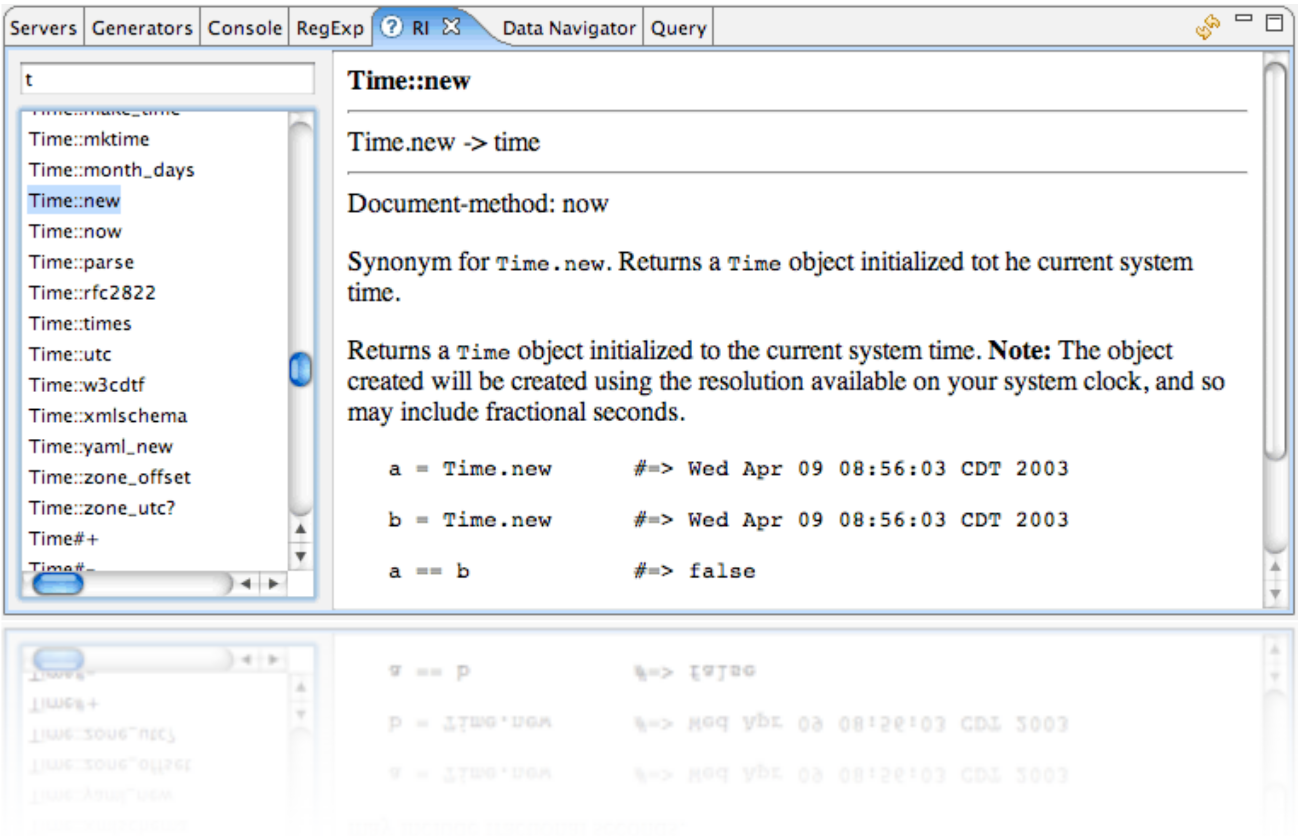
"Project name:" 這欄位要打加入的專案檔案夾名稱。 例如我的專案叫 "cats"，所以在 Project Name: 欄位打 "cats"，然後寫到一半的程式應該放在 /Users/lichu/projects/cats 裡頭。 按 "Finish" 之後檔案夾裡面的東西就會顯示在 Rails Navigator 視窗裡，不用擔心它會把你以前寫的碼給蓋過。

在 Radrails 裡如果中文字顯示是亂碼的話，到左上角的 Radrails > Preferences > General > Workspace。 然後在右手邊的 Text file encoding 裡選擇 Other: UTF-8

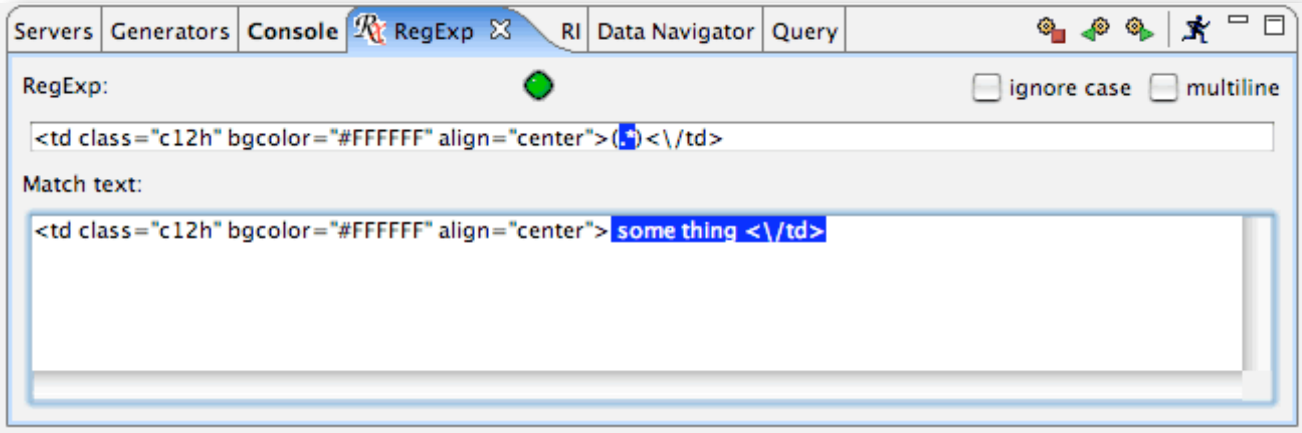
Radrails 右下角有五個可切換的小視窗，分別為 Servers, Generators, console, RegExp, RI。

最有用的應該是 RI 了。

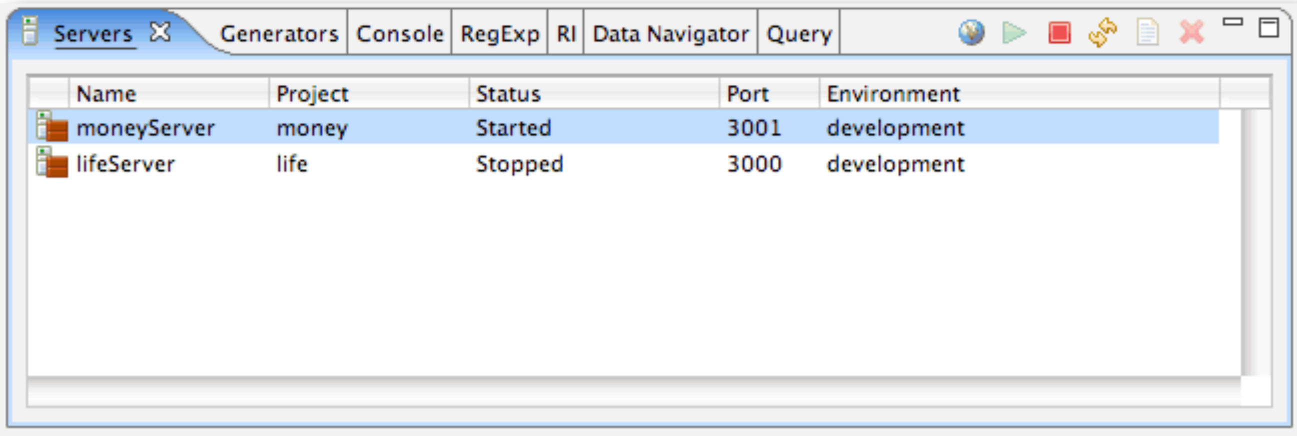
RI 讓你找 Ruby 裡的 class， 不用另外上網去找。

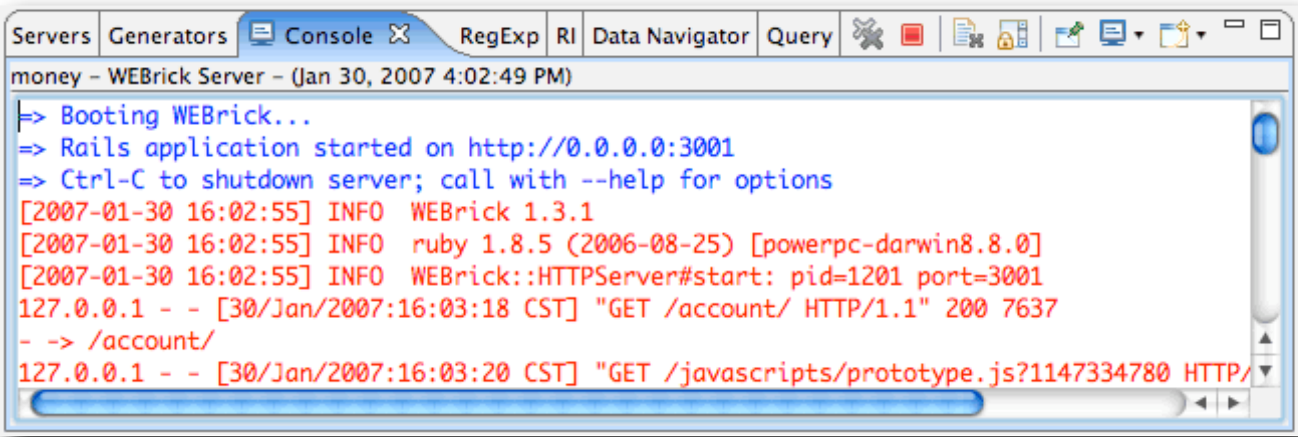


RegExp 視窗也是蠻好用的。 RegExp 是 Regular Expression 的縮寫。 Regular Expression 有時候很複雜，自己寫好了一串但不知道寫的對不對。 這時候用 RegExp 就可以先測試看看了。

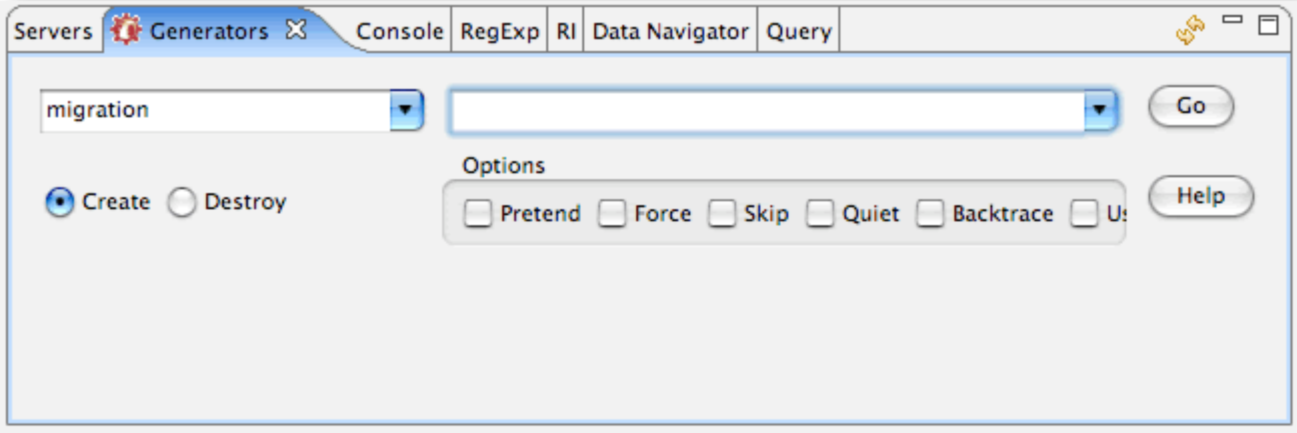


Servers 視窗可讓你啟動或停止 server. Server 在跑時它的訊息會顯示在 Console 的視窗裡。

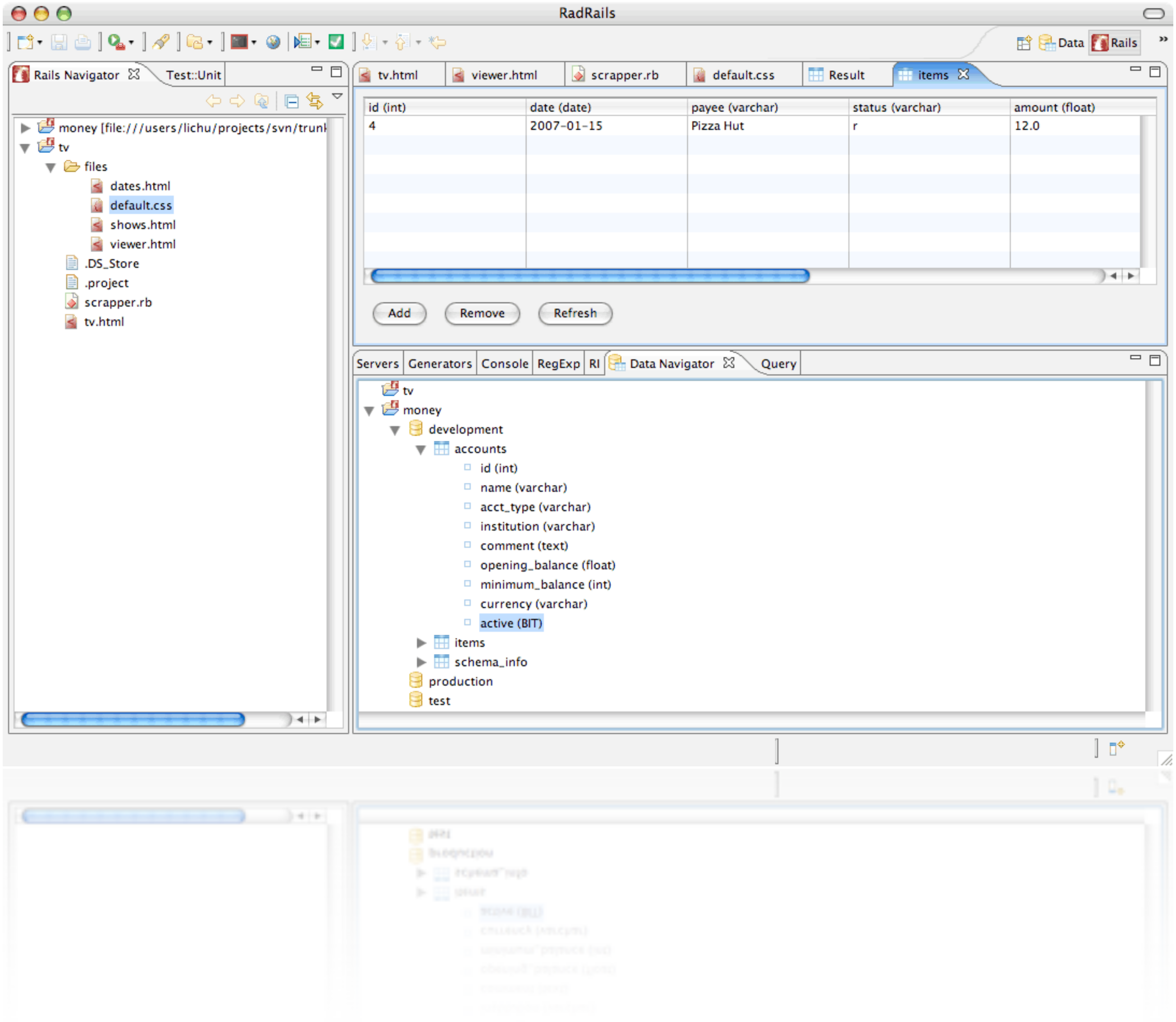




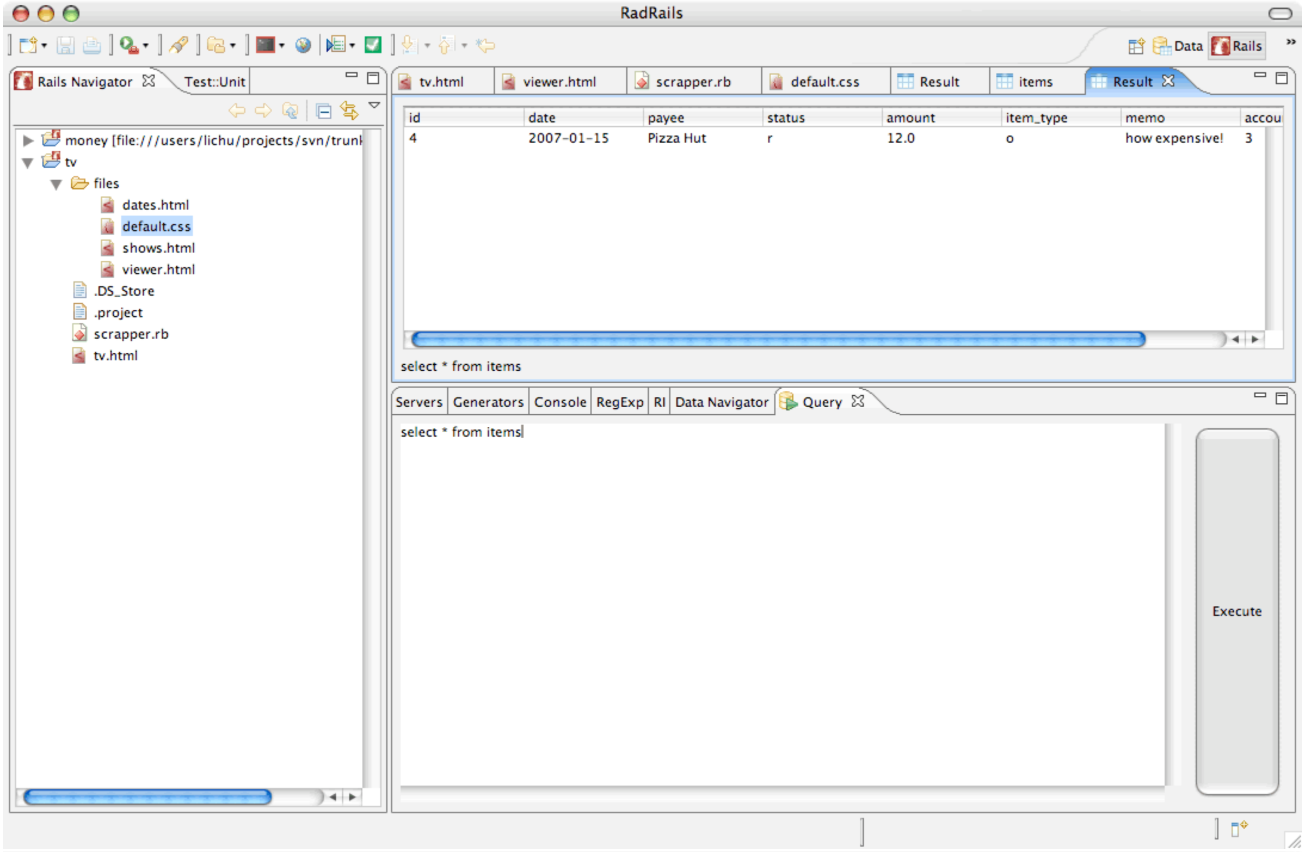
最後一個 Generators 的視窗可讓你新增/刪除 model，controller 等等。



你也可以新增別的有用的視窗。 選單裡選 Window > Show View > Other， 然後選擇 Data Navigator。 這個視窗能讓你看看不同資料庫裡的資料。



重複剛剛新增的動作，這次選 Query 的話，就會有一個視窗能讓你直接下 SQL 指令。



一個不要錢的 IDE 來說，Radrails 真是功能強阿！

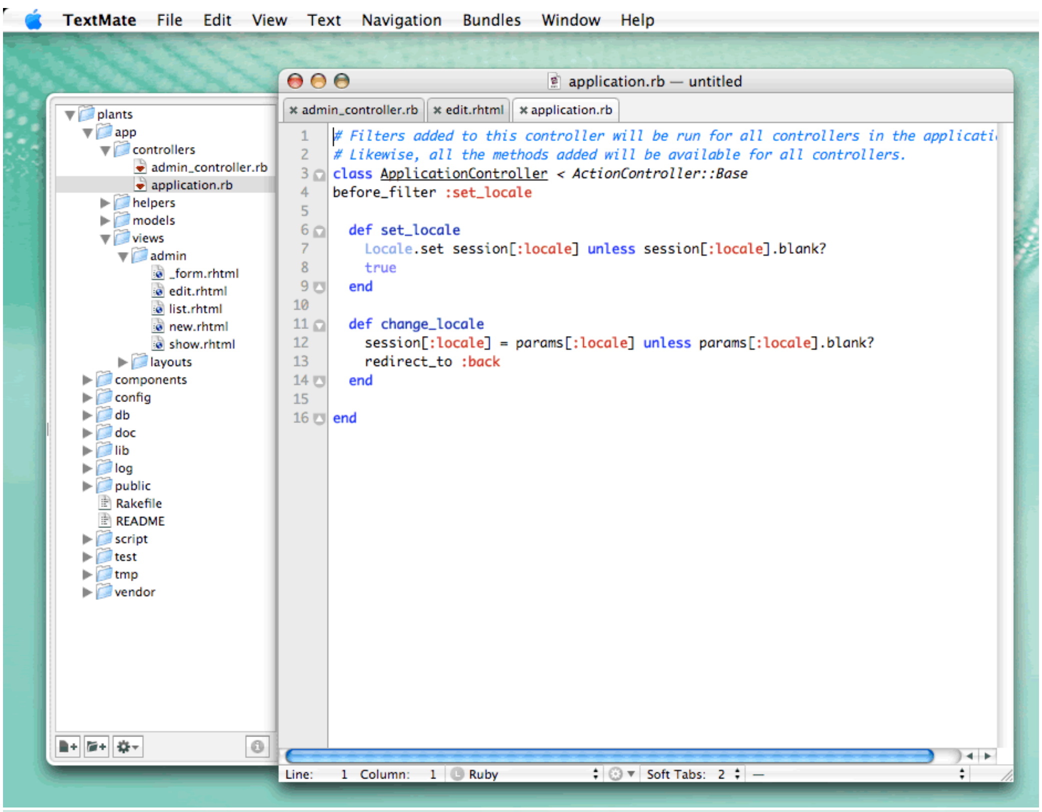
Textmate 簡單介紹

注意: Textmate 目前顯示的中文字都擠在一起，好像不知道中文字的寬度是英文的兩倍。
Textmate 軟體可以先下載來測試30天再決定要不要買。

Rails 的高手用的都是 Textmate。 在網路上看到的 screencast 用的也是 Textmate。 Textmate 到底是什麼東西為什麼會那麼紅呢？ 其實當初 Textmate 的開發者在寫那套軟體時有請教過 DHH。 DHH 邊建議邊用，用了覺得好用後就在自己的部落格上幫 Textmate 打廣告。 被 DHH 推過的東西當然就是一炮而紅了。 Textmate 不只是為了 Ruby 而寫，而是為所有的語言編輯而寫。 去看選擇語言的地方有二，三十個語言; C, Java, PHP, XML, Python, 有的沒有的，看了就覺得自己知道的語言不夠多。

要下載試用軟體的話到 http://macromates.com/ 下載。 Textmate 只能在 OSX 上跑，所以不用煩惱在別的平台上怎麼裝，煩惱怎麼湊錢買新的蘋果電腦吧。 Textmate 下載後安裝方法跟其他蘋果軟體一樣，只要把程式抓到任何的檔案夾就可以開始使用了。 既然安裝的部份談不到兩句，我們就來參觀 Textmate 的功能吧。

基本功能來說，很多 Radrails 有的功能 Textmate 也有，像專案的檔案管理，tabbed 的檔案視窗等等。

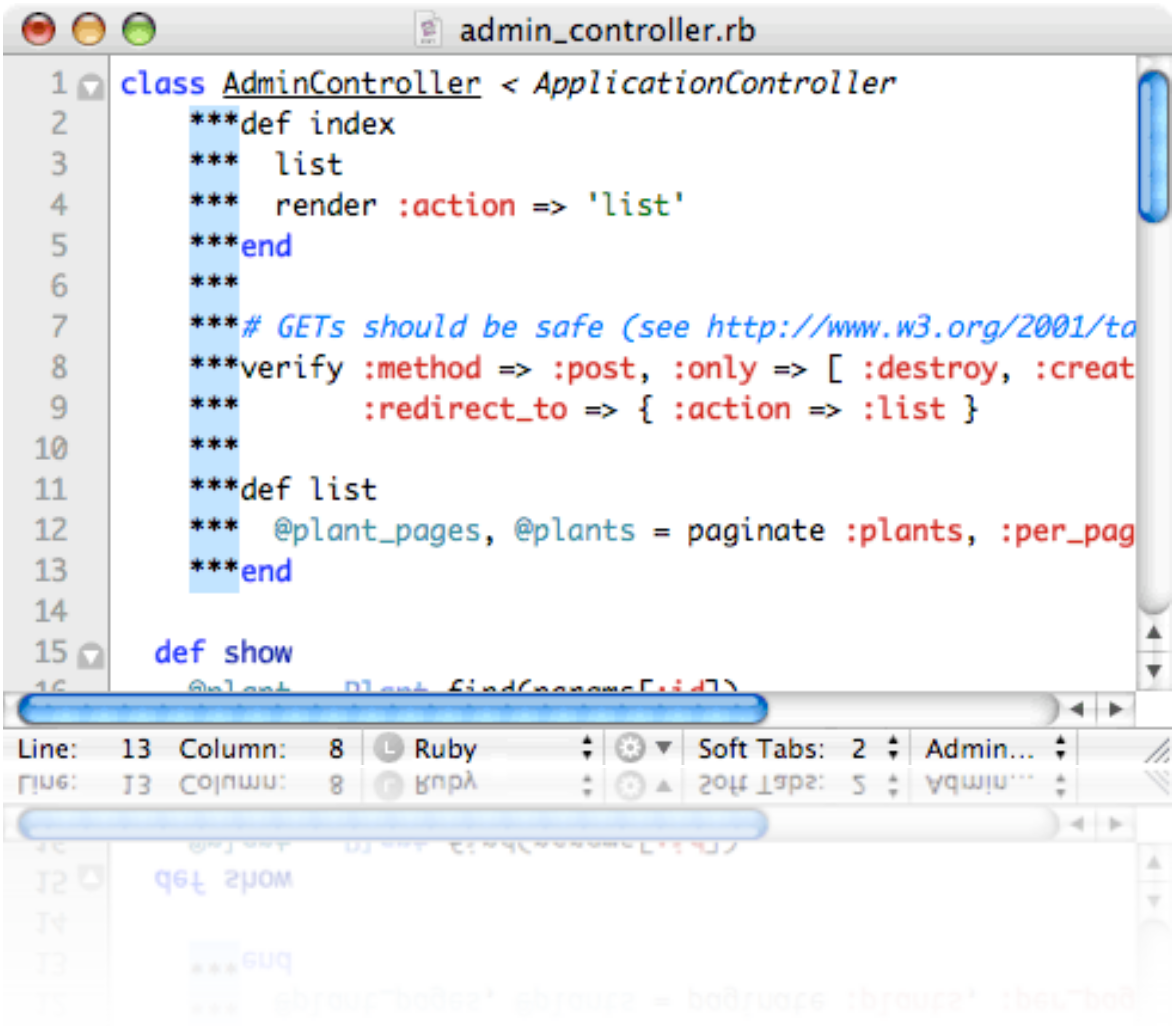


但是，Radrails 右下角那些小視窗的功能 Textmate 就沒有了。 Textmate 的重點是在文字編輯，而 Radrails 的重點是在 IDE 的整合性，讓你在同一個視窗裡做越多東西越好。

一些 Textmate 比較特殊的功能如下:

直行式的寫程式：

Textmate 可讓你一次在好幾行 code 裡寫一樣的東西



按一下 alt 鍵，滑鼠的箭頭就會變成十字。 現在選幾行的程式碼然後開始打字。 你所打的每個字將出現在剛剛選的每一行上。 這個好處就是讓你很容易的增加或刪除一堆字。

不同地點加/減字:

這跟直行式寫程式的觀念一樣，你只要事先在程式碼裡按 option+s 來 選擇不同的地點，然後開始打字，所打的字就會在所有選擇的地方出現。 （這個功能需要多加的 bundle，要加 bundle 請看下面的 ″快速鍵″)

快速鍵：

快速鍵的目地就是減少寫碼時打的字。 例如，打 reca 然後按 tab， reca 就自動伸展成 redirect_to(controller, action)。 這樣的快速鍵是別人特別為 Rails 寫的然後包裝成 bundles 來讓其他的 Textmate 使用者用。 最有用的 Textmate bundle 在 http://blog.inquirylabs.com/my-textmate-bundle/ 。 Textmate 強的功能是它的快速鍵 (macros) 可以很簡單的被客製化，所以有很多功能並不是它自己本身有的，而是別人利用它的客製化功能新加的！ 熟悉這些快速鍵後程式會寫的比較快，而且也減少拼錯指令的困擾。

版本控制：

上網下載 subversion 的 bundle 後就可以直接在 Textmate 裡 commit 最新版，或比較跟別版不同的地方。

尋找功能：

要搜尋編過的碼可以用 Regular Expression 來當收尋條件，並同時取代文字，使搜尋功能更正確並更有彈性。

快速尋找檔案：

Rails 的天性就是產生檔案。 那麼多的檔案，只要按 蘋果-t 就可馬上收尋你要的檔案名稱。

這些只是 Textmate 一部份的功能介紹，要多使用才會發覺更深奧的功能。 目前官方網站一套軟體賣台幣 $1,633。 我想還是先用 Radrails 一陣子再來考慮 Textmate 吧。

Migration 資料庫更動

為甚麼要用 MIGRATION 呢？

我如果寫了一個很好的程式放在網路上給大家使用，然後又添加了很多功能，這時要怎麼把舊資料庫的 schema 更新呢？ 如果有人想把軟體版本從第三版更新到第九版，另一個人又想把第四版更新到第六版， 又要怎麼更新呢？ 如果有一幫人用的資料庫是 mySQL, 另一幫人用的是 Postgre, 那要怎麼辦？ 別擔心，救星 Migration 來了！

寫軟體的模式通常是很快的寫出第一版，公佈給大家使用後再出功能比較強的第二，三版。但從一版到另一版之中，資料庫的欄位不斷的增加和減少。 Migration 這套系統好用的地方就是可以把所有變動的地方紀錄在文字檔裡，然後打個 ″rake migrate″ 的指令，系統就會把現在用的版本更新到最新版。 這魔術是如何變的呢？我們就一步一步看吧。

我們用從頭寫一個簡單的程式叫″dog″來紀錄不同種類的狗。

rails dog

這時候螢幕會出現一大堆自動產生在 ″dog″ 目錄下的東西。

MIGRATION 基本操作

使用 Migration 前的準備

1）資料庫設定

在開始寫 model 前，要把資料庫連線的訊息設好。 還記得要去那邊設嗎？ 沒錯，就是 dog/config/database.yml 裡。 在 development 的地方輸入你要的資料庫名稱，使用者名稱，跟密碼。 我不介意用它自動填的 dog_development 來當資料庫。我的使用者名稱也是 root, 而且也沒有密碼，所以我不用改資料。但你如果這些值有自訂的話，要記得改喔。

2）建立 dog_development 資料庫

用資料庫管理程式來建立一個叫 ″dog_development″ 的資料庫。 這個資料庫是我們現在開發用的，所以才會叫 ″development″ 如果是測試用的我們就應該用另一個新的資料庫 ″dog_testing″。 這一篇沒有講到 testing 所以先不用管 ″dog_testing″ 這個資料庫。

產生 model

設定好了後，我們可以開工了。下這個指令：

ruby script/generate model Dog

螢幕上出現的幾行字裡會看到一個

create db/migrate/001_create_dogs.rb

這表示 rails 的程式碼產生器已自動產出第一個資料庫 schema 的定義檔。 把 db/migrate/001_create_dogs.rb 打開。 第一版的資料庫裡要有什麼 table, table 裡要有什麼欄位等等都要寫在這裡面。 打開檔案之後你會看到有一個 class 叫 CreateDogs, 那個 class 有兩個 function, 叫 self.up 跟 self.down。 所有的 Migration 所產生的檔案將會有 self.up 跟 self.down。 Migration 看 self.up 的內容來決定升級的時候要做甚麼。如果要降級的話 Migration 看 self.down 的內容。

self.up

看看 self.up 的內容。 create_table :dogs ... 表示如果要從 0 版升級到 1 版的話，我們要新增一個叫做 dogs 的 table。 當初 程式碼產生器知道要把 table 命名為 dogs 是因為我們叫它產生一個 Dog 的 model。 根據 rails 的命名模式，如果 model 叫 Dog, table 就叫 dogs, controller 就叫 dog_controller.rb 等等。 在 "t.cloumn :name, :string" 的地方就是要放新增的欄位名稱跟 datatype。

```
def self.up
  create_table :dogs do |t|
    t.column :breed, :string
    t.column :avg_size, :string
    t.column :aggressive, :boolean
  end

  create_table :cats do |t|
    t.column :breed, :string
    t.column :avg_size, :string
  end
end
```

Datatype

上面寫的程式碼並不是 SQL, 原因是 Migration 可通用在不同的 database 上。如果想把 mySQL 換成 Postgre 的話，這些 Migration 檔案是不用改的。 Migration 會自動用資料庫了解的語言跟資料庫溝通。 Migration 所知道的 datatype 如下：integer, float, datetime, date, timestamp, time, text, string, binary, boolean。

id 欄位

看了上面這段碼之後你會覺得說，咦？怎麼沒有一個叫 id 的欄位？不是每個 rails 的 table 都要這個欄位嗎？ 沒錯，就是因為這個原因我們才不用自己加。 既然每個 table 都要， 我們不用寫，Migration 就會自動加入 id 這欄位了。 但如果你有特別需求不要加 id 欄位， 加 " :id => false " 就好:

```
create_table :dogs, :id => false do |t|
...
```

注意： " :id => false " 這個東西是不必要設的，除非確定不要 "id" 這個欄位。

self.down

來看看 001_create_dogs.rb 裡 self.down 的內容。 我們如果更新軟體，推出去賣後又發現問題一大堆，這時要把資料庫還原成上一版的，又該怎麼還原呢？ 依照 001_create_dogs.rb 這個檔案來說，如果 Migration 執行了這個檔，兩個 table 就會跑出來: cats 跟 dogs, 如果要還原成上一版的話，就要把 cats 跟 dogs 刪掉，所以在 self.down 程式裡，就該放 drop table 的指令。

```
def self.down
  drop_table :dogs
  drop_table :cats
end
```

001_create_dogs.rb 改好後記得要存檔喔。

執行 Migration

好了，總算可以變魔術了。下這個指令：

```
rake migrate
```

版本紀錄

下這個指令後連到資料庫裡面去看，應該會看到 dogs 跟 cats 這兩個 table，table 裡面也有我們剛剛加的欄位！好耶！不用學 SQL， Migration 就會幫我們增加欄位了。 資料庫再仔細看一

下。 咦？ 怎麼會多了一個叫 "schema_info" 的 table? 點進去看看，它只有一個欄位叫 "version"，裡面有一行資料寫著 "1"。 喔，原來這個 table 就是 Migration 用來紀錄我們目前用的版本是那版。 假如裡面登記的是 "3" 的話，我們用的就是第三版。如果 dog/db/migration/ 下有到 006_xxxxx.rb 開頭的檔案，在跑 rake migrate 的時候，Migration 就會遵照 004_xxxx.rb, 005_xxxxxxx.rb, 和 006_xxxxx.rb 裡的內容來升級資料庫。

資料庫現有的 schema

我們的資料庫 schema 會改來改去，但目前資料庫的 schema 是長的甚麼樣子？ 要看的話，就去看 dog/db/ 裡的一個檔案叫 schema.rb。 打開後你也可以看到裡面的版本是第一版:

```
ActiveRecord::Schema.define(:version => 1) do ....
```

這個檔案專門是 Migration 在用所以不能改喔。 如果這檔案不小心被你家的貓亂按鍵盤刪掉的話，不用擔心，Migration 可再生一個給你。 在 dog/ 的目錄下指令：

```
rake db_schema_dump
```

這個指令會把現有的資料庫狀態紀錄回 dog/db/shema.rb 裡。

如果你的資料庫垮了，可是你有把 dog/db/schema.rb 備份起來，要還原的話只要把備份的 schema.rb 放回 dog/db/ 目錄下，然後在 dog/ 的目錄下指令：

```
rake db_schema_import
```

注意：這動作會把舊的 table 跟其內容刪掉。

升級到第二版

我們的 database 用了一陣子之後，發覺到貓的體型都差不多，所以不需要 "avg_size" 這個欄位。 還有，我們想紀錄不同種類貓毛的長度，因為有人覺得長毛貓的毛很難清，所以我們要多加一個欄位叫 hair_length。 要改變的話如下：

在 dog/ 目錄下產生一個 migration 檔：

```
ruby script/generate migration change_cat_table
```

下這指令後你會看到 dog/db/migration/ 檔案夾裡有多一個檔案叫 002_change_cat_table.rb。 把這個檔案打開後你會看到一個 class 叫 ChangeCatTable。除了前面 002 的編碼跟 "_" 之外，"ChangeCatTable" 這個 class 的名稱 一定要跟 "002_change_cat_table.rb" 拼的一樣。 "change_cat_table" 這個名字是隨便我們取的，但注意以下的規定：

a) 名字中不要有 "-"。 例如 change-cat-table 是不對的。

b）名字不要跟 model 的名字一樣，像 002_dog 是不對的，因為我們已有一個 model 叫 Dog.

c) Migration 的檔名不要一樣，像 002_change_cat_table.rb 跟 003_change_cat_table.rb 是不能同時存在的。

修改 002_change_cat_table.rb 的內容

```
class ChangeCatTable < ActiveRecord::Migration
 def self.up
  add_column :cats, :hair_length, :integer, :null => false
  remove_column :cats, :avg_size
 end

 def self.down
  remove_column :cats, :hair_length
  add_column :cats, :avg_size, :string
  Cat.find(:all).each { |cats| cats.update_attribute :avg_size, "unknown" }
 end
end
```

哇！這個檔案是在搞什麼？一行一行來看吧。

add_column 是增加 column 的指令。
:cats 是 table 的名字。
:hair_length 是我們要新增的欄位。
:integer 是 datatype。
:null => false 表示這個欄位不能空白。 除了 :null => false, 其它可用的設定值有：
:limit => 34 表示欄位的寬度最多只能有 34 個字母

:default => "gymnasium" 會把 "gymnasium" 當為預設值，也就是說每一筆新的資料在這欄位裡一剛開始的資料會是 "gymnasium" 這個字串。

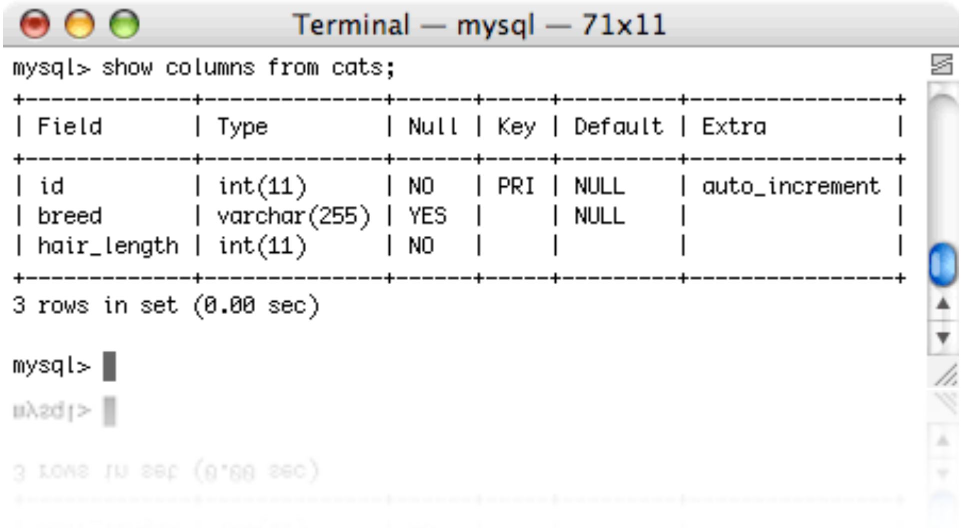
回到上面的範列，remove_column 是刪除 column 的指令。 self.down 裡的內容我們等一下在討論，因為升級時用不到它。

執行 rake migrate

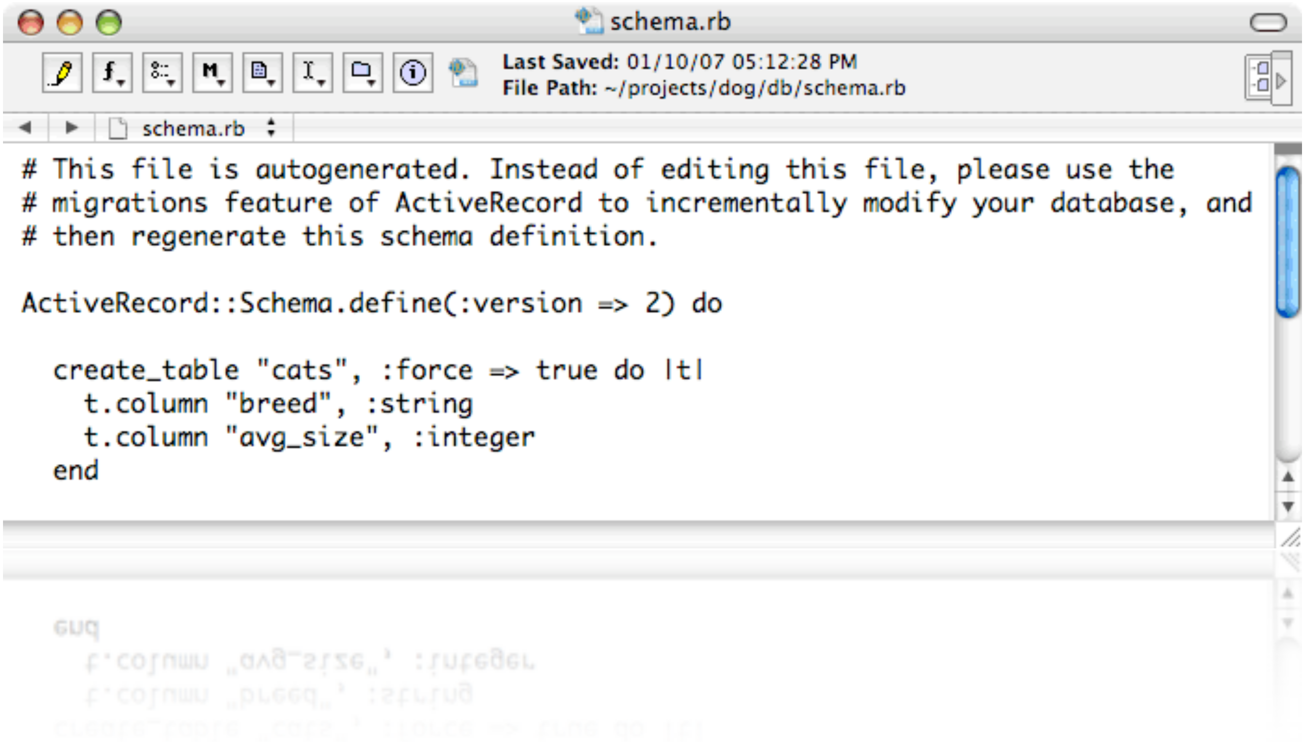
在 dog/ 目錄下

rake migrate

指令之後，我們去看資料庫，果真 cats 的 table 裡現在只有三個欄位: id, breed, 跟 hair_length。



我們再來看看 dog/db/schema.rb 檔。 耶！ ″version =>″ 變成 2 了！



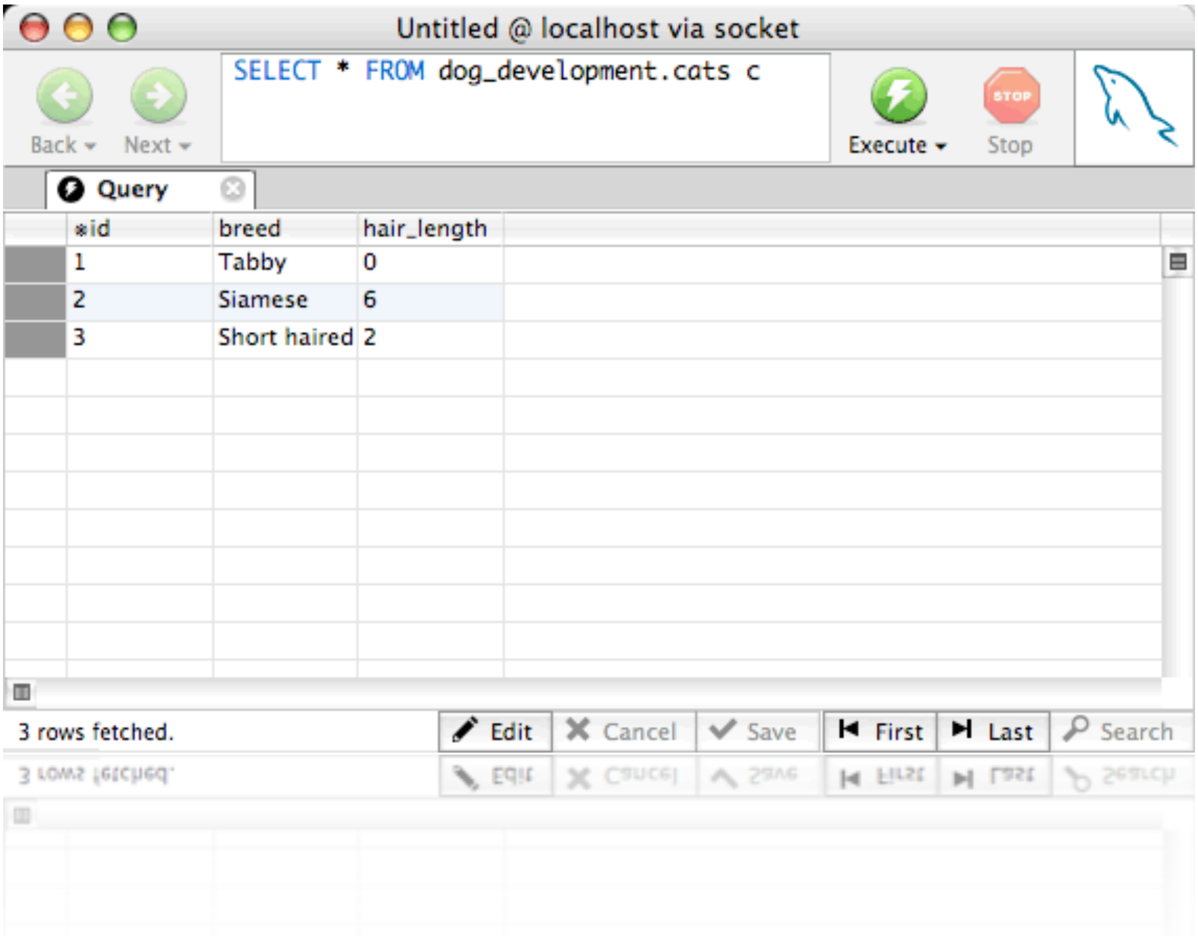
降級版本

在降級版本之前我們要先建一個 Cat 的 model 才能展現 migration 的功能。請在 db/ 下此指令：

ruby script/generate model Cat --skip-migration

-skip-migration 是叫 Migration 不要自動生出一個 003_xxx.rb 檔，因為我們已經有 cats 這個 table 了，不需要用 Migration 來生另外一個。

model 產生好之後，記得去 cats 的 table 裡加幾行假資料，加完後準備手續就完畢了！



我們現在用的資料庫版本是二，以後如果有陸續寫出三，四，五版的話，就可以任意的從第二版跳到任何版本。 下指令時只要指定您要的版本就可以。

因為我們現在只有寫到第二版，所以只能從第二版跳回第一版。要怎麼跳呢？下這指令吧：

rake migrate VERSION=1

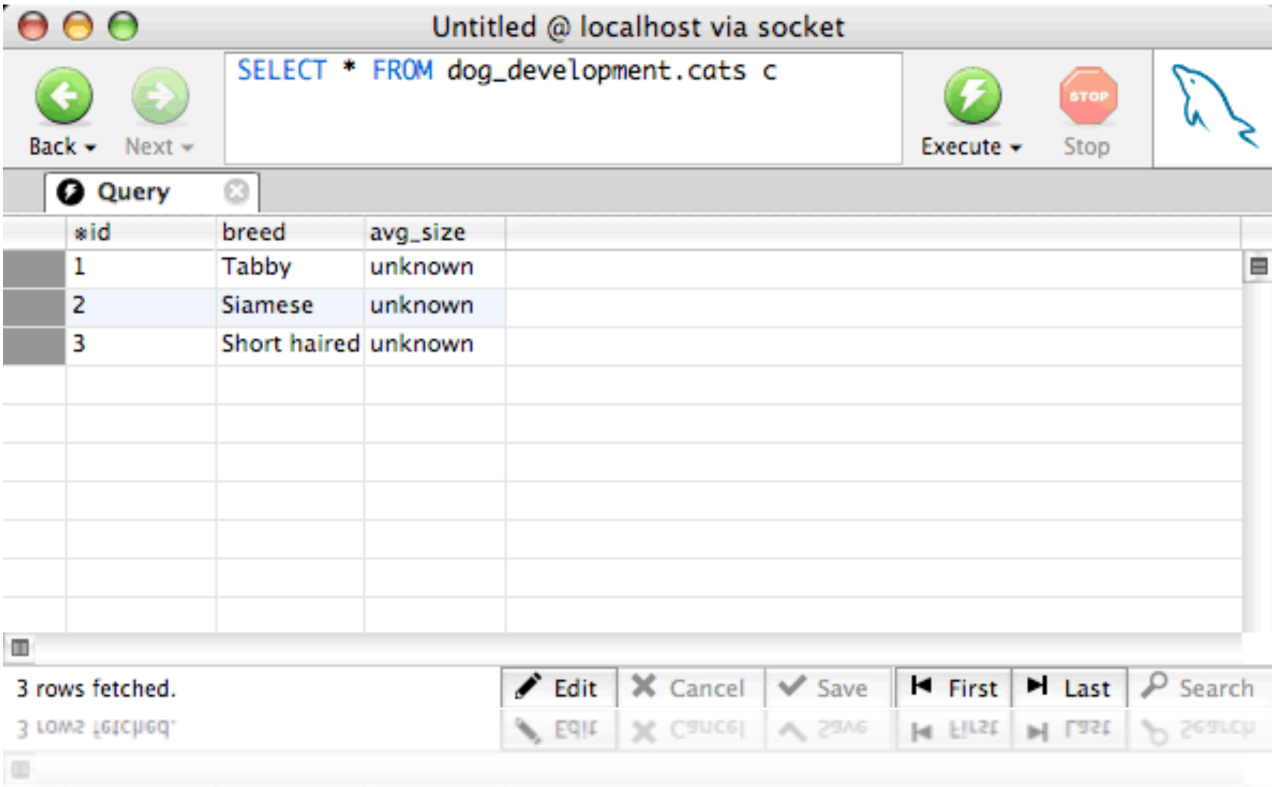
下這指令後，Migration 會去資料庫找出 schema_info 的 table. 還記得那裡面只有一個欄位吧？那個欄位裡有個 "2"，所以 Migration 知道它必須從第二版降級到第一版。 降級的步驟當然是要去 002_change_cat_table.rb 裡的 self.down 找啦。 好，那我們現在就來看 self.down 裡面到底是在做甚麼。

第一行 "remove_column..." 是移除 "hair_length" 這個欄位。 為啥要移除呢？ 因為這個欄位是從第一版升到第二版時加的，所以要從第二版降回第一版時就應該把這欄位刪掉。

第二行 add_column... 是把以前刪掉的 "avg_size" 欄位加回來。

第三行 ″Cat.find....″ 就比較複雜了。 我們要加回 ″avg_size″ 這個欄位，但如果資料庫裡已經有資料的話，我們並不知道每行以前的 ″avg_size″ 裡的值是甚麼，所以只好放 ″unkown″ 值進去。 Cat.find(:all) 就是去 cats 的 table 裡把所有的資料撈出來。 Cat.find(:all).each 就是把撈出來的資料一筆一筆的放進去 { } 裡面的程式來處理。 | cats | 就代表每一筆放進去的資料。 每筆放進去的資料它會叫 ″update_attribute″ 的功能。 :avg_size 是要作業的欄位。 ″unkown″ 是要放入作業欄位的值。 一執行後，:avg_size 這欄位會被加回去，但所有紀錄裡的值都是 ″unknown″。

解釋了那麼多，來看看資料庫有沒有被還回第一版。有耶！cats 的 table 裡面 ″hair_length″ 不見了，″avg_size″ 回來了，而且每行放進去的假資料裡都有 ″unkown″。



其他功能

介紹到此，基本的 Migration 功能都有碰到了，如要特殊的客製化，看看下面不同的話題。

不要自動產出 Migration 檔

剛剛看過了，如果不要讓電腦自己產生 001_xxx.rb 等的 Migration 檔的話，在下 generate 指令時加個 ″--skip-migration″ 指令

```
ruby script/generate model Cat --skip-migration
```

或是把它生出來的檔案刪掉也可以。

設定 table 的 encoding

Table 裡存的資料不會每次都存英文的，如要存中文的話，可以在建 table 時把整個 table 的編碼改成 UTF8

```
create_table(:dogs, :options => 'DEFAULT CHARSET=UTF8') do |t|
```

多欄位的索引

如要加多欄位的索引 (multiple-column index), 傳入欄位名稱即可。

```
def self.up
  add_index :cars, [:license_plate, :state]
end
```

在這個例子我們的 table 是 "cars", 索引是由 "license_plate" 跟 "state" 的欄位組成。

自己加或刪除 foreign key

目前 Migration 還沒支援 foreign key，所以你如果有用到 foreign key 的話要自己下 SQL 指令。 把 SQL指令放在 self.up 或 self.down 裡頭就好:

```
execute 'ALTER TABLE employees ADD CONSTRAINT fk_employees_unions FOREIGN KEY ( union_id ) REFERENCES unions(id);
```

上面這一行純粹是 SQL 指令。 它在 employees 的 table 裡加了一個 foreign key 叫 union_id，對印 unions table 裡的 id 欄位。

加了 foreign key 之後，要刪除 table 時資料庫一定會抱怨，這時只要先下指令叫資料庫不要查是否有 foreign key:

```
def self.down
  execute "SET FOREIGN_KEY_CHECKES=0"
  drop_table "employees"
  ...
end
```

強行刪掉舊的 table

如果資料庫裡已有你要新增的 table, Migration 跑一半會失敗，並抱怨說 table 已經有在資料庫了。 要強制把舊的 table 刪掉並叫資料庫不要抱怨的話，加個 :force => true

```
create_table "toys", :force => true do
...
```

可通用的 Database

除了 DB2 之外，寫出來的 Migration 檔案可通用在以下的資料庫：MySQL, PostgreSQL, SQLite, SQL Server, Sybase, Oracle。

常用到的 Migration 指令

* 新增 table: create_table(table 的名稱, 其他變數)
* 刪除 table: drop_table(table 的名稱)
* 改 table 名稱: rename_table(舊 table 名稱, 新 table 名稱)
* 新增 column: add_column(table 的名稱, column 的名稱, type, 其他變數)
* 改 column 名稱: rename_column(table 的名稱, 舊 column 的名稱, 新 column 的名稱)
* 改 column 格式: change_column(table 的名稱, column 的名稱, type, 其他變數)
* 刪除 column: remove_column(table 的名稱, column 的名稱)
* 增加 index: add_index(table 的名稱, column 的名稱, index_type)
* 移除 index: remove_index(table 的名稱, column 的名稱)

對這些功能的用法如果還有問題的話可以去看看 Migration 的 API 喔。

http://api.rubyonrails.org/classes/ActiveRecord/Migration.html

Sessions

SESSION 的概念

一個網站利用 session 來記得它跟某個使用者的互動。 例如，阿福登入 Gmail， Gamil 就會建立一個它跟阿福之間的 session。 每當阿福按他的收信籃或垃圾桶連結時，session 就會告訴 Gmail 阿福要什麼樣的資訊，然後 Gamil 就會把此資訊傳回給阿福。 如阿福登出 Gmail 或關閉瀏覽器，session 就會消失，Gmail 將不知道阿福是誰，也不會把阿福的資訊傳出去。 Session 只有在需要知道使用者是誰時才會建立。 如果阿福只要用 Google 的收尋功能，不用 Gmail， Google 就不需要知道他是誰，所以也不用建 session。

在 Rails 裡，session 是怎麼建立的呢？ 答案就是 cookie：當你登入一個網站時，網站的伺服器就即時產生一個 id 給你，當作 cookie 放在你的瀏覽器裡。 這 id 其實只是一個長度 32 位數的英文字母字串。 每次你要新的資訊時，你的瀏覽器會把那串字給伺服器，這樣伺服器就知道你是誰了。

在我們的示範裡頭，有時會把一些變數存在 session 裡，方便紀錄一些有的沒有的。 這又是怎麼辦到的呢？ 每個人登入網站之後，Rails 會利用剛剛說的 32 位數的字串建一個文字檔，存放在你程式目錄下的 tmp\sessions 檔案夾。 你如果有執行過用到 session 的程式，去看看那個檔案夾你就會看到一堆檔案命名類似 ruby_sess.bb97ac9253f7ce87。 打開其中一個，就會看到一些資料。 這些資料就是當初那個使用者的 session 物件被伺服器處理後存放的資料。 一個使用者會產生一個 session，一個 session就會產生一個 session 檔案。跟伺服器溝通時，伺服器會用你給它的字串找出你的 session 檔案，然後把這檔案裡的值灌回你的 session。 Session 裡的值依照你的需求被更改後，又會被存回 tmp\sessions 檔案夾。 在 tmp\sessions 檔案夾裡看到的就是不同的使用者產生的 session 檔案，如果沒定時把這些檔案清掉的話，檔案夾會爆掉。 清除的方法等一下再談。

在程式裡要把變數存入 session 很簡單，直接用 session 陣列就可以：

```
session[:current_phone] = user.phone
```

要取回存入的值的話也是一樣：

```
The phone number currently on file is <%= session[:current_phone]%>
```

儲存 MODEL 在 SESSION 裡頭

除了簡單變數的儲存外你也可以存整個 model 物件 (object)。 如要存物件，記得要先在 controller 裡頭宣告，這樣 Rails 才知道物件的 class 定義。 宣告的方式如下：

```
class CustomerController < ApplicationController
  model :paycheck
...
```

這個範例裡，我們的 controller 是 Customer， model 是 paycheck。

如果你的程式裡有好幾個 controller, 而且有兩個以上的 controller 會用到 session 裡存的物件，那麼你就應該在 application_controller.rb 裡宣告這物件。 存物件的壞處是，如果改版本的話，新的物件裡的值就跟舊的不相容了，所以要記得把 tmp\sessions 裡的檔案刪掉。

SESSION 的儲存方法

Session 資料其實有好幾種儲存的方式。上一小節 "Session 的概念" 裡提到的方法是 PStore。 其他方法有的是存在資料庫，有的是記憶體。 各有好有壞，以下從最簡單的到最複雜的大致介紹：

MemoryStore 跟 FileStore

MemoryStore 把 session 資料記在程式的 memory 裡面。 同時如果有很多人登入的話，記憶體將不夠用，所以這個方法不實用。 FileStore 把資料記在檔案裡，但資料的型態只能存 string, 不能存 object, 所以對 Rails 程式不實用。

PStore

PStore 是系統的預設值。 什麼都不用設系統就自動會用這個方式了。 每一個 session 的檔案就存在 tmp\sessions 裡頭。這個的壞處是如果同時有很多個 session 的話 （一萬個人以上同時登入你的系統）整個程式的效能就會變差。 效能變差的原因是 tmp\sessions 裡的檔案會越來越多，有些系統遇到檔案夾裡有好幾千萬的檔案時就會慢下來。 用 PStore 的話記得要定時把 tmp\sessions 裡的舊檔案刪掉，不然系統有一天可能因為一個檔案夾裡太多檔案而拒絕建立新的 session。 檔案要怎麼刪？總不能人工去刪吧？如果你是用 Linux/unix 的話，可以增加一個 cron 工作檔來刪除一個禮拜內沒碰過的 session 檔。 雖然有這個麻煩，但如果你的網站只需要一台伺服器, 而且同時不會有一萬個人來登入，那用 PStore 最方便了。

ActiveRecordStore

比 PStore 更進一層的就是 ActiveRecordStore。如果只有少數的人登入的話，它的效能不會比 PStore 好，但是如果越來越多人同時用，它的效能不會變得差，反而比 Pstore 好。會用到 ActiveRecordStore，就表示你的程式有分散到好幾台伺服器，或是同時有一萬人以上來登入。

ActiveRecordStore 會把 session 的資料存在資料庫裡，所以你要在資料庫裡建一個叫做 "sessions" 的 table，裡面要有四個欄位: id, sessid, data, 跟 updated_at. id 的屬性是 integer。 sessid 就是我們常說的 session 的 id, 欄位的長度至少要 32 個字母。 要記得把這個欄位設為 index，這樣資料庫找資料時才會快。 data 就是放 session 資料，屬性為 text 才夠用。 updated_at 這欄位不是一定要的，但有了它 Active Record 會自動把上次變動過的時間填進去。此欄位的 datatype 是 timestamp。

DrbStore

DrbStore 又比 ActiveRecordStore 更好。 如果只是幾個 session 在跑的話，DrbStore 的效能跟 PStore 一樣好。 越來越多人登入時效能也不會下降。 DrbStore 可以透過區域網路來分享物件 (objects)。 它的 session 資料是由 DRb 伺服器來管的。 你的程式如果分散到不同的伺服器上，每個程式的 instance 都能跟同一個 DrbStore 溝通。 Rails 有包含一個簡單的 DRb 伺服器。 列如我的就在

/usr/local/lib/ruby/gems/1.8/gems/actionpack-1.12.1/lib/action_controller/session

依照你灌 Ruby 的地方，你的路徑跟版本可能跟以上有點不同。

MemCacheStore

MemCacheStore 是最厲害也是最難的一個 session 管理方法。 有些大的網站，像 Wikipedia, LiveJournal 等等，都是用 MemCacheStore。 LiveJournal 的網站一天要伺服兩千萬客製化的網頁，並有一百萬使用者。 除非你的網站跟這些網站有的比，否則最複雜的用到 DrbStore 就可以了。 更多資訊請到 www.danga.com/memcached/ 。

變更 SESSION 的儲存方法

看了這幾個不同的方法後，如想變更要怎麼做呢？ 開啟程式裡的 config/environment.rb。 在檔案的最底部你會看到一行字

```
# Include your application configuration below
```

在這行字下，貼上

```
ActionController::CgiRequest::DEFAULT_SESSION_OPTIONS[:database_manager] = CGI::Session::ActiveRecordStore
```

這行就會把儲存方法改為 ActiveRecordStore 了。 如要 MemCacheStore, 把 ActiveRecordStore 改為 MemCacheStore, 依此類推。 如要 PStore, 把整行刪掉，因為預設值是 PStore.

不想用 SESSION

如果不要用 session，打開 config/environment.rb 然後變更或加入下面這行：

```
ActionController::CgiRequest::DEFAULT_SESSION_OPTION = false
```

要注意的是，沒有 session 的話，就不能用使用者登錄等等功能喔。

SESSION 管理

除了在 PStore 裡提到的定時刪除 session 檔案之外，程式裡面在使用者登出時多加個 reset_session， 這樣伺服器就會把剛剛用過的 session 檔案刪掉，檔案夾也比較不會爆掉:

```
class MyOwnStoreController < ApplicationController
 ...
 def logout
  reset_session
 end
end
```

一台伺服器多個 RAILS 程式

伺服器上如果有跑兩個或兩個以上的 rails 程式的話，你要叫每一個程式把他們用的 cookie 前面加上程式的名稱，這樣程式才不會使用互相的 cookie，造成一個程式登入後，在另一個程式沒有登入就可以進去了。 打開 config/environment.rb 然後變更或加入下面這行。

```
ActionController::CgiRequest::DEFAULT_SESSION_OPTIONS[:session_key] = 'my_mouse_diary'
```

把 my_mouse_diary 改成你要加的名字。

限制只有 HTTPS 能用 SESSION

程式如果要說只有透過 https 才可啟動 session 的話，打開 config/environment.rb 然後變更或加入下面這行

```
ActionController::CgiRequest::DEFAULT_SESSION_OPTIONS[:session_secure] = true
```

Cookies

COOKIES 概念

常常聽說有些網站會在我的電腦上放 cookies，這是怎麼回事？其實 cookies 只是個放在使用者端的文字檔，裡面記一些變數。 當你去瀏覽一個網站時，那台伺服器可能會把 cookie 放在你的電腦。你再跟它要資料時，它會跟你的瀏覽器要 cookie。 要到之後它就可以看以前儲存的變數了。rails 就是利用 cookies 來放 session 需要用的辨識碼。有些使用者不允許 cookies, 這樣的話 rails 就沒辦法儲存 session，不允許 cookies 的人也就無法登入網站了。

存/取 COOKIES

你自己也可以寫一些變數存在使用者的電腦上，但是要記得使用者可以隨時用自己的瀏覽器把你放的 cookie 丟掉。 要存 cookies 的話很簡單。如果我要存的 cookie 叫做 ″rewards″ 的話：

```
cookies[:rewards] = ″teddy bear″
```

之後要用的方式：

```
earned_rewards = cookies[:rewards]
```

就這樣啦。 cookie 的值一定要是 string，不然會有錯誤訊息。

COOKIES 參數

在設 cookies 時，如果沒加其他的參數，你放的 cookie 就永遠不會過期，會一直待在使用者的電腦上 (除非他把 cookies 清掉)。以上面的範列來說，我們可以多加一些參數：

```
cookies[:rewards] = { :value   => ″teddy bear″,
            :expires => 5.weeks.from_now,
            :path    => ″/my_first_app″,
            :domain  => ″somesite.com″,
            :secure  => true}
```

:expires 是設定這個 cookie 的有效期限。

:path 是設定 request 的路徑。也就是說，使用者要瀏覽的路徑裡，在 domain 的名稱後一定要由 "/my_first_app" 開始 cookie 才可以用。可以用的意思是使用者的瀏覽器會把你所要的 cookie 傳回你的 server。

使用者如果瀏覽

http://somesite.com/my_first_app/ 的話，存在電腦裡的 cookie 會被傳回到你的伺服器，
http://sm1.somesite.com/my_first_app/homework/page1.html 也會，但
http://sm1.somesite.com/shop/homework/page1.html 就不行了。

:domain 就是設定說網站名稱的最後一部份要跟設定值相同，不然 cookie 不能使用。 像

http://somesite.com/ 可以，
http://sm1.somesite.com/ 也可以，可是
http://somesite.org/ 就不行了。

:secure 如果是 true 的話，只有透過 "https://" cookie 才會被傳回來。

除錯

先試試你的 CODE

應用 Ruby 的 functions 時，有時後記不太得正確的寫法是什麼。 如用猜的話又不知道猜的對不對。 這時候有兩種可試的地方。如果不確定寫法的那段碼純粹是 ruby 的話，你可以用 irb。 在任何的路徑打 ″irb″ 就可以開始測試 ruby 的語法了：

```
wrl88-083:~ lichu$ irb
irb(main):001:0> 1 + 2
=> 3
irb(main):002:0> exit
wrl88-083:~ lichu$
```

wrl88-083:~ lichu$ 是我的 command prompt, 只要打 ″irb″ 就好，不用打 wrl88-083:~ lichu$ 。 記得， irb 純粹只是測試 ruby 碼而已喔，不會測試 Rails 碼。 如果要測試的程式碼有用到 Rails 的 class 的話，那就要去現在在開發中的 Rails 程式目錄下打指令。 例如，我的一個程式叫 ″dog″ 放在 ″projects″ 的目錄下，而且我想測試一行程式碼 ″20.days.from_now″：

```
wrl88-083:~ lichu$ cd projects
wrl88-083:~/projects lichu$ cd dog
wrl88-083:~/projects/dog lichu$ ruby script/console
Loading development environment.
>> 20.days.from_now
=> Mon Sep 04 17:14:56 CST 2006
```

上面的例子中，我們先進到 dog 的檔案夾，然後再啟動 ruby script/console。 這時， Rails 的 class 會自動被載入記憶體當中，電腦才知道要怎麼處理 days.from_now 這個 method。 相同的，我自己在 ″dog″ 這個程式裡所新寫的 class 也會被載入 memory, 所以我也可以測試我自己寫出來的 class。

輸入 ruby script/console 之後，螢幕上出現了一串字 ″Loading development environment″。 意思是 ″載入開發的環境″。還記得 Rails 有三個主要的環境吧：開發，上線，跟測試。如果我們要的環境是″上線″環境的話，可以打

ruby script/console production

20.days.from_now, 也就是 20 天之後，就是 9/4 了。 如果是在 irb 裡頭打 20.days.from_now 的話，它會不知道你要做什麼:

```
wrl88-083:~/projects/dog lichu$ irb
irb(main):001:0> 20.days.from_now
NoMethodError: undefined method `days' for 20:Fixnum
    from (irb):1
```

這是因為 days.from_now 這個功能是 Rails 的 class 裡才有。 irb 只能接觸到 Ruby 的 class，所以無法執行 days.from_now。

看 PARAMS 裡的變數

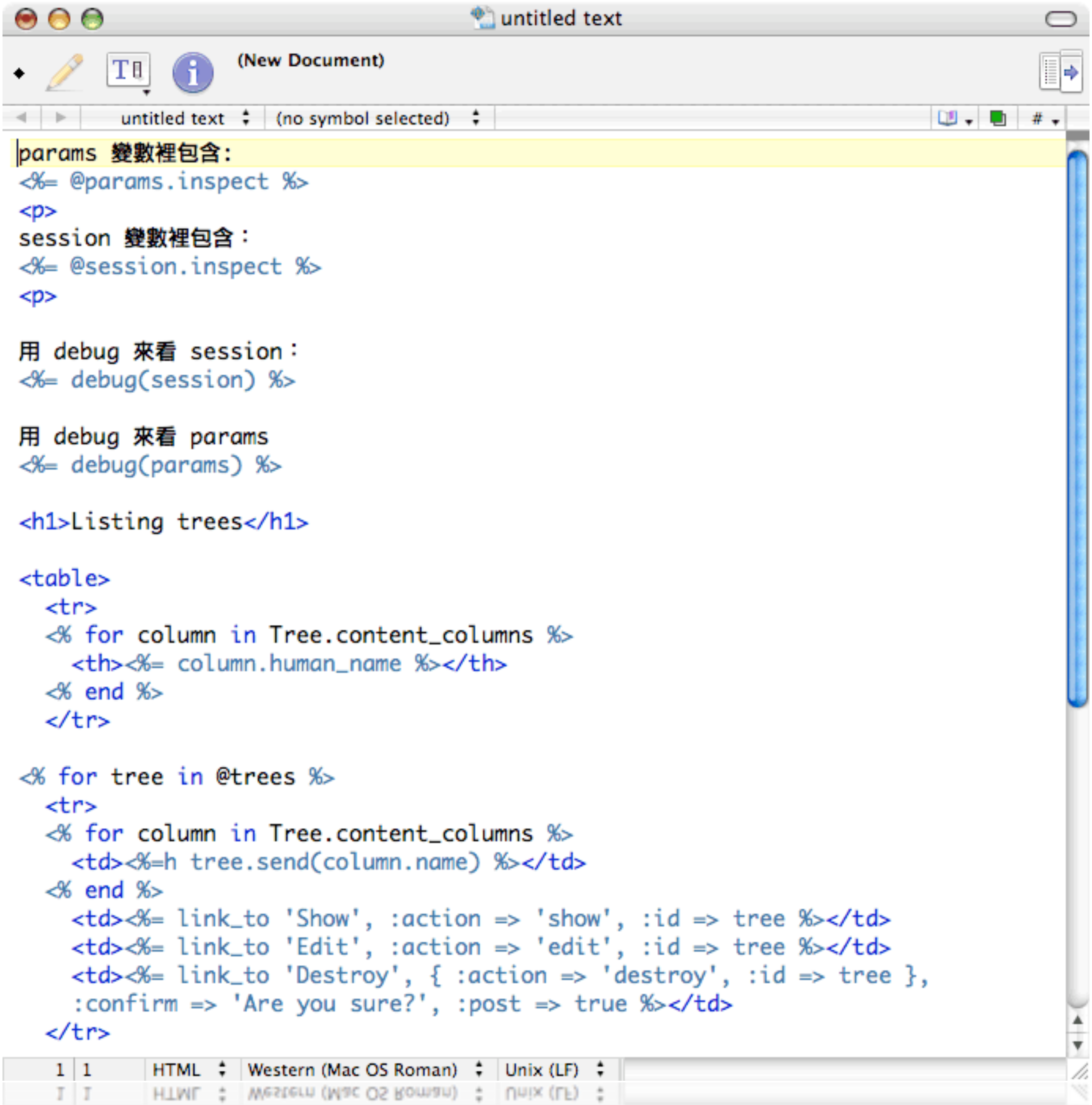
除錯的步驟之一可看看傳送的變數是否正確。

如要看某一頁傳到下一頁的參數的話，可以在 view 裡打

<%= @params.inspect %>

或要看 session 的參數的話:

<%= @session.inspect %>




```
params 變數裡包含:
<%= @params.inspect %>
<p>
session 變數裡包含：
<%= @session.inspect %>
<p>

用 debug 來看 session：
<%= debug(session) %>

用 debug 來看 params
<%= debug(params) %>

<h1>Listing trees</h1>

<table>
  <tr>
  <% for column in Tree.content_columns %>
    <th><%= column.human_name %></th>
  <% end %>
  </tr>

<% for tree in @trees %>
  <tr>
  <% for column in Tree.content_columns %>
    <td><%=h tree.send(column.name) %></td>
  <% end %>
    <td><%= link_to 'Show', :action => 'show', :id => tree %></td>
    <td><%= link_to 'Edit', :action => 'edit', :id => tree %></td>
    <td><%= link_to 'Destroy', { :action => 'destroy', :id => tree },
    :confirm => 'Are you sure?', :post => true %></td>
  </tr>
```

在下面的圖樣裡有一個 scaffold **產生的** view **檔叫** list.rhtml。 **裡面有放** <%= @params.inspect %> **和** <%= @session.inspect %>:

Tree was successfully created.

params 變數裡包含: {"action"=>"list", "controller"=>"admin"}

session 變數裡包含： #, @hash={"flash"=>{:notice=>"Tree was successfully created."}}>], @dbman=#, @hash={"flash"=>{:notice=>"Tree was successfully created."}}>, @new_session=false, @data={"flash"=>{:notice=>"Tree was successfully created."}}>

用 debug 來看 session：

```
--- !ruby/object:CGI::Session
data: &id001
  flash: !map:ActionController::Flash::FlashHash
    :notice: Tree was successfully created.
dbman: &id002 !ruby/object:CGI::Session::PStore
  hash: *id001
  p: !ruby/object:PStore
    abort: false
    filename: script/../config/../tmp/sessions//ruby_sess.b9e3095af1d5ac87
    rdonly: false
    table:
    transaction: false
dbprot:
- *id002
new_session: false
session_id: dc1e83d7a5cbe5134607139956ace9ad
```

用 debug 來看 params

```
--- !map:HashWithIndifferentAccess
action: list
controller: admin
```

Listing trees

Breed	Avg size	Aggressive			
Ficus	Large	false	Show	Edit	Destroy
阿樹	5	false	Show	Edit	Destroy

New tree

伺服器在跑時網頁上就會出現:

圖樣裡也有顯示 debug 指令的功能。 要看某物件裡的參數，可用 debug 這著指令:

```
<%= debug(params) %>
```

其他可看的系統變數有 headers, params, request, 和 response

debug 也可以用來看任何物件裡的內容:

```
<%= debug(@order) %>
```

寫程式要解除常抓錯的困擾就是要常常執行寫過的碼，一出錯時才能很快的知道那裡要改喔。

國際化

準備一個示範的程式

Rails 如要中文化要在每個 Rails 程式裡單獨設定，所以我們來做一個 ″花草記錄簿″ 當示範程式。

首先，建一個叫 plants_development 的資料庫。 建好了之後到你習慣放程式的檔案夾裡建一個新的 rails 程式:

```
rails plants
cd plants
```

把 Ruby 的編碼方式改成 UTF-8

Ruby 的編碼方式一開始只用英語。 我們要改用 UTF-8 才能使用中文。 打開檔案

plants/config/environment.rb，

在最後一行 ″# Include your application configuration below″ 後面加上這兩行：

```
$KCODE = 'u'
require 'jcode'
```

儲存並關閉檔案。

設定跟資料庫溝通時用的編碼

每種不同的程式，不管是 PHP, Rails, 或 JSP 等等，要跟任何的資料庫溝通時都要先決定溝通時用的編碼。 在 Rails 跟 MySQL 裡頭，溝通的編碼是在 database.yml 裡面設的。 打開 config/database.yml，在每個設定區裡加一個 ″encoding: utf8″。

像 development 應該看起來像這樣 (你的 username 跟 password 可能會不同）:

```
development:
  adapter: mysql
  database: plants_development
```

```
 username: root
 password:
 host: localhost
 encoding: utf8
```

這個範例是 development 的設定，"test" 跟 "production" 也要加最後一行的 encoding: utf8。

你用的資料庫如果是 PostgreSQL 的話，把 "encoding: utf8" 改成 "encoding: unicode"。

SQLite 資料庫不用設這個值，但是要安裝 SQLite 時要啟動 utf8。 例如，在 Linux 系統安裝前要給 ./configure 一個變數 "--enable-utf8":

```
./configure --enable-utf8
```

建立 model

建立一個 model 才有可以修改資料的東西:

```
ruby script/generate model Plant
```

這時候 Migration 會建一個叫 db/migrate/001_create_plants.rb 的檔案。 把這個檔案打開，在

def self.up 裡放入：

```
  create_table (:plants, :options => 'DEFAULT CHARSET=UTF8') do |t|
   t.column "name", :string
   t.column "planted_at", :datetime
   t.column "flower", :boolean
  end
```

"DEFAULT CHARSET=UTF8" 是要讓資料庫儲存各國不同的語言。 存檔後回到 plants/ 的目錄裡下這個指令來產生新的 "plants" table:

```
rake migrate
```

有了這個 table 之後就可以自動產生 CRUD (Create, Read, Update, Delete) 的架構了:

```
ruby script/generate scaffold Plant Admin
```

設定 HTML 的 encoding

雖然我們設定了 Ruby 跟資料庫溝通時用的編碼 ， 每一網頁的 <head> </head> 裡頭不要忘了加

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
```

真的要每頁每頁加嗎？ 當然不用啦。直接放在 layout 裡頭就會在每頁上面顯示出來了。 打開

app/views/layouts/admin.rhtml

在這個檔案裡的東西會出現在所有 admin controller 管理的網頁上。 在 <head> 跟 </head> 中間貼上剛剛的那行字串：

<meta http-equiv="Content-Type" content="text/html; charset=utf-8">

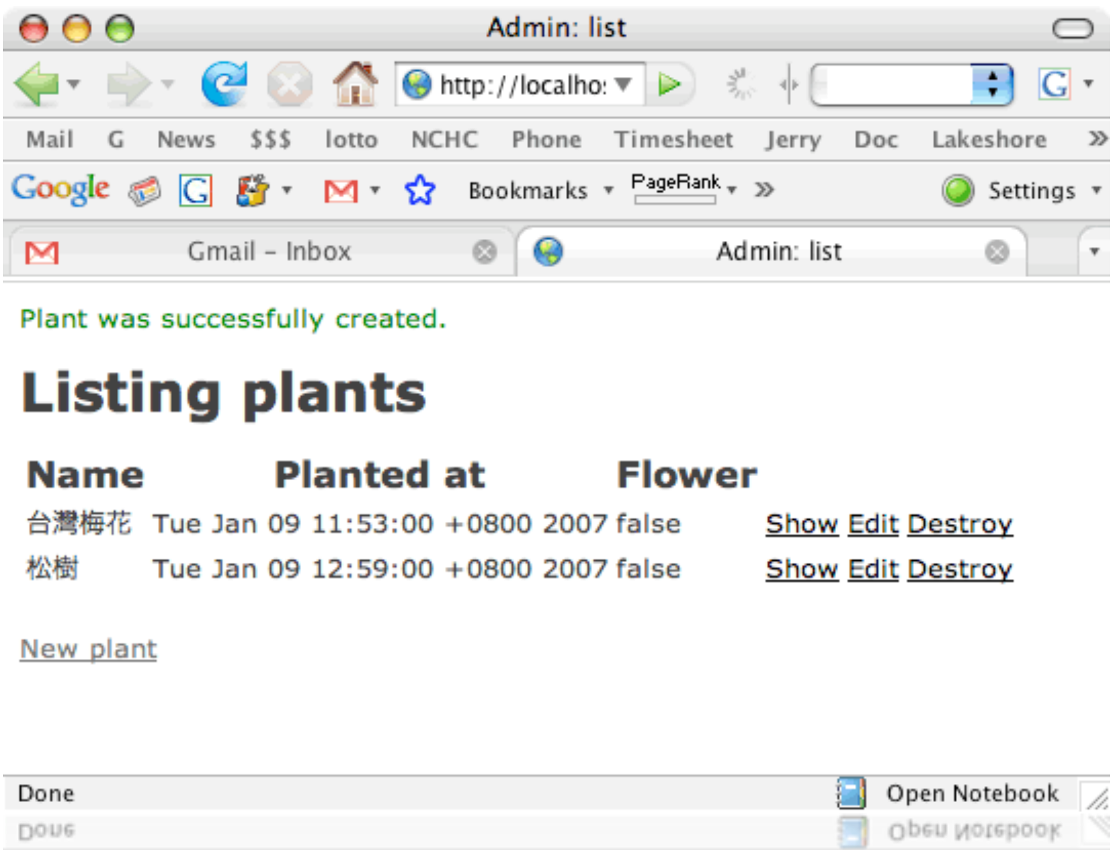
測試中文讀寫

到現在設定的部份總算完成，但要測試真的會不會跑。

啟動伺服器:

ruby script/server

到 http://localhost:3000/admin/ 增加一些中文名字的植物吧！



到這個步驟你如果還是看到亂碼的話就要看看瀏覽器上設的 encoding 是否為 utf-8。

現在程式已經可以讀/寫中文了。 如果要的功能只有這樣，就不用繼續讀下去。 可是如果想要按一個連結就把整個網頁切換成中/英文的話，就繼續讀吧。

國際化

Rails 有很多別人已寫好的軟體。 我們要用的工具是寫的比較完整的 plugin 叫 Globalize (www.globalize-rails.org)。 這一套軟體把你翻譯過的字串存在資料庫裡。 如果改變語言時，它會把翻譯過的字串叫出來。 除此之外，各國不同的數字，金錢顯示方法也可以存在資料庫裡。 我們邊做邊解釋吧。

安裝 Globalize Plugin

到程式的資料夾裡 (plants/) 下此指令:

```
ruby script/plugin install svn://svn.globalize-rails.org/globalize/branches/for-1.1
```

整個指令不要分兩行打入喔。 這會安裝 Globalize 在程式裡。 打開 /vendor/plugins 檔案夾，裡面會看到一個 trunk 的檔案夾。 這裡面放的就是 Globalize 的軟體了。 在 plants/ 的目錄裡再下此指令來準備 Globalize:

```
ruby script/generate globalize
```

這個指令會透過 Migration 產生一個檔，叫 db/migrate/002_globalize_migration.rb。 打開來看它是在做啥麼的吧！ 裡面喵了一下，它會產生 3 個 table (globalize_countries

globalize_translations, 和 globalize_languages)，然後它會做一些看不太懂的東西，然後加了一大堆有關世界各國的資料。 仔細的看看，它有加一些 index，原來加 index 的語法是這樣寫的。 以後自己寫 Migration 檔案時可以拿這個來做參考。 不知道為什麼，這個檔案在新增 table 的時候，並不會自己把 table 的編碼用成 UTF-8。 我們只好自己去修改。 在每一個 create_table 的指令裡多放 :options => 'DEFAULT CHARSET=UTF8' 如下:

```
create_table (:globalize_countries, :force => true, :options => 'DEFAULT CHARSET=UTF8') do |t|
```

檔案裡有三個 create_table, 所以要改三個地方。 改好後就可以執行這個檔案了:

```
rake migrate
```

跑了 35.2505 秒之後，去看資料庫，果真多了三個 table.

設定基本語言

在運用這些 table 之前，我們還得設定基本語言。 打開 config/environment.rb，在最後面加這幾行：

```
include Globalize
Locale.set_base_language 'zh-TW'
Locale.set 'zh-TW'
```

第一行就是說此程式需要包含 Globalize plugin 的程式碼。 第二行把基本語言設成 zh-TW， zh-TW 就是中文-台灣。 你的基本語言如要是英語的話，把 zh-TW 改成 en-US （英文-美國）。 第三行把地點設成台灣。

加入翻譯字串

回到 http://localhost:3000/admin/ 的網頁上，你應該看到以前加入的花名。 在每個花名後有三個連結: 'Show' 'Edit' 跟 'Destroy'。 現在的目的就是把這三個連結翻譯成中文。 翻譯的第一步驟是輸入翻譯字串。

要加入翻譯字串最快的方法就是用 script/console 。 在 plants/ 目錄裡下此指令:

```
ruby script/console
```

這會把程式裡有用到的 class 載入記憶體裡，讓你方便測試。打一個指令：

```
Locale.language
```

它應該會跟你說 "Chinese"。 這是因為我們剛剛把基本語言設成 zh-TW。如果你的不是 Chinese 的話，用此指令讓它變成 Chinese:

```
Locale.set 'zh-TW'
```

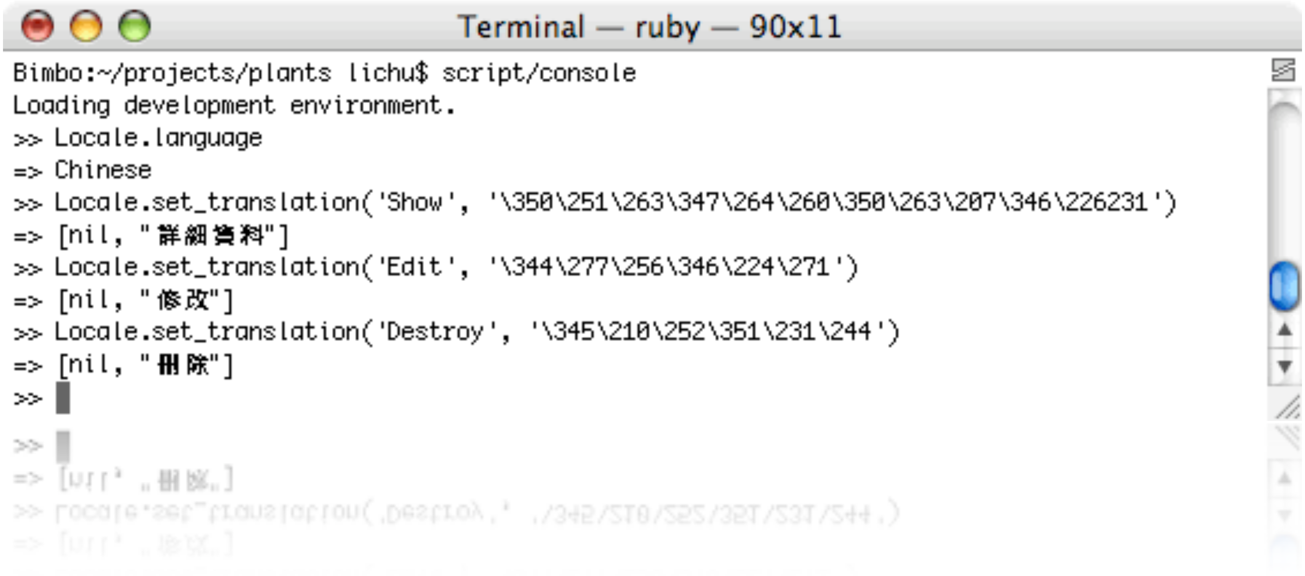
加入下面幾個翻譯字串。 加的時候你的中文碼在螢幕上可能會變成數字字串: \207\346\226\231 這不用擔心，存入資料庫時它會轉成 UTF-8 碼。 下這些指令來輸入翻譯字串：

```
Locale.set_translation('Show', '詳細資料')
Locale.set_translation('Edit', '修改')
Locale.set_translation('Destroy', '刪除')
```

每輸入一行它會回覆類似:

```
[nil, "詳細資料"]
```

注意大小寫是有差的喔。 Show 跟 show 是兩個不一樣的字串。



設定那些字串可被翻譯

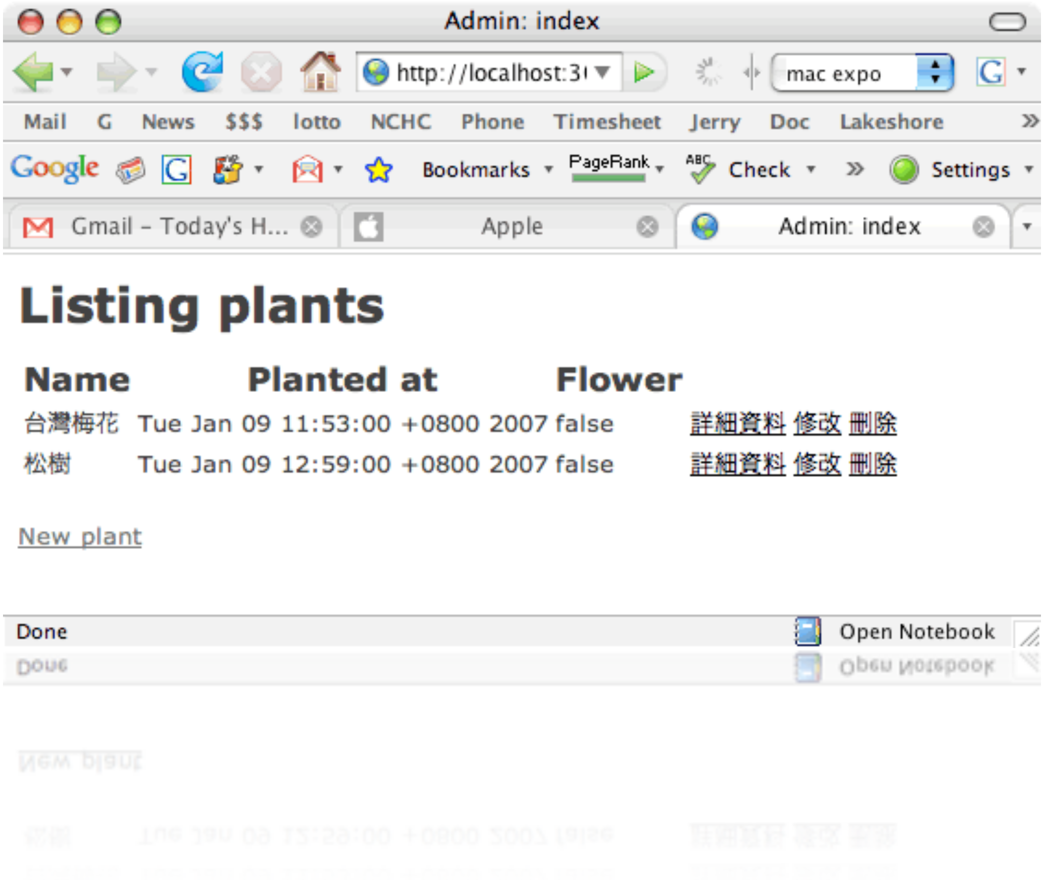
翻譯字串輸入了，可是這並不代表每次程式遇到 "Show" 時會自動換入 "詳細資料"。 必須要在想翻譯的那串字後加一個 .t 才可以。 打開

app/views/admin/list.rhtml 。

這個檔案是 scaffold 自動產生的。 使用者到 http://localhost:3000/admin/ 時其實叫出來的就是 list.rhtml。 檔案裡面，在 'Show' 的後面加個 .t，如下:

<td><%= link_to 'Show'.t, :action => 'show', :id => plant %></td>

'Edit' 跟 'Destroy' 後面也要加。 現在瀏覽 http://localhost:3000/admin/ 的話， Show, Edit, 跟 Destroy 應該就是用中文顯示了。



切換中英文

那要怎麼切換中英文呢？ 以下三個步驟大致說明一下，再來仔細看看:

1) 在我們的網頁上，我們要放兩個連結。 一個可以把英文改成中文，另一個可以把中文改成英文。 中文的連結裡有個叫 locale 的變數，變數的值是 'zh-TW'。 使用者按了連結後，這個變數會透過 params 傳到下一頁去。

2) 在下一頁還沒出現時，把變數裡的值複製到 session 裡一個也是叫做 locale 的變數。

3) 下一頁要出現時，先去檢查 session 裡頭的 locale 變數。 查到了 'zh-TW' 就知道要用中文了。

英文要換成中文也是一樣的原理。 現在一步一步來看：

步驟一: View 裡的連結

我們的連結可以放在 app/views/layouts/admin.rhtml 裡，因為放這裡面的東西會顯示在所有網頁上。

```
<%= link_to "中文", :action => "change_locale", :locale => "zh-TW"%>
```

這一段碼會產生一個連結。 連結上會寫 "中文"，按了之後 controller 會把控制權轉給一個叫 "change_locale" 的 method。 同時，'zh-TW' 也放入了 :locale 的變數裡。 :locale 這個名字是我們自己取的，我們也可以隨便放個名字，像 :booboo => "zh-TW" 也可以。 不管是什麼名字，我們不用事先宣告，直接把值給任何的名字，然後用同樣的名字去 params 裡面找，就能把裡面的值叫出來了。

我們多加一點功能吧。 如果 session[:locale] 裡面已經有了 'zh-TW'，就表示我們現在已經在顯示中文了，所以 "中文" 這個連結就不用顯示。 把第一行跟第三行加入：

```
<% unless session[:locale] == 'zh-TW' %>
 <%= link_to "中文", :action => "change_locale", :locale => "zh-TW"%>
<% end %>
```

這段讀起來就是: 顯示 "中文" 這連結，除非 session[:locale] 裡已有 'zh-TW'。 英文的程式碼也要：

```
<% unless session[:locale] == 'en-US' %>
 <%= link_to "English", :action => "change_locale", :locale => "en-US"%>
<% end %>
```

步驟二: 傳 param 的值進去 session 裡

"中文" 跟 "English" 這兩個連結按了之後的 action 是 change_locale。 現在就來寫 change_locale 的內容吧。 這個 method 應放在 app/controllers/application.rb 裡頭:

```
def change_locale
  session[:locale] = params[:locale] unless params[:locale].blank?
  redirect_to :back
end
```

這個 method 的作法就是 "除非 params[:locale] 是空白，把 params[:locale] 裡的變數放入 session[:locale] 裡"。 以我們這個程式來說，裡面放的值不是 'zh-TW' 就是 'en-US'。

步驟三: 設定 Locale

每次一有人要瀏覽網頁時，我們就要設 locale。 因為設 locale 的 method 是整個程式都可用到的，所以最適當的地方是放在 app/controllers/application.rb 裡。 我們把這個 method 叫 set_locale。

剛剛看過改 Locale 的方法是用 Locale.set。 要 set 成什麼呢？就是 set 成我們存在 session[:locale] 的變數了。 所以我們改變 locale 的方法如下:

```
def set_locale
  Locale.set session[:locale] unless session[:locale].blank?
  true
end
```

這個 method 的作法就是 ″除非 session[:locale] 是空白，把 Locale 設為 session[:locale]″。

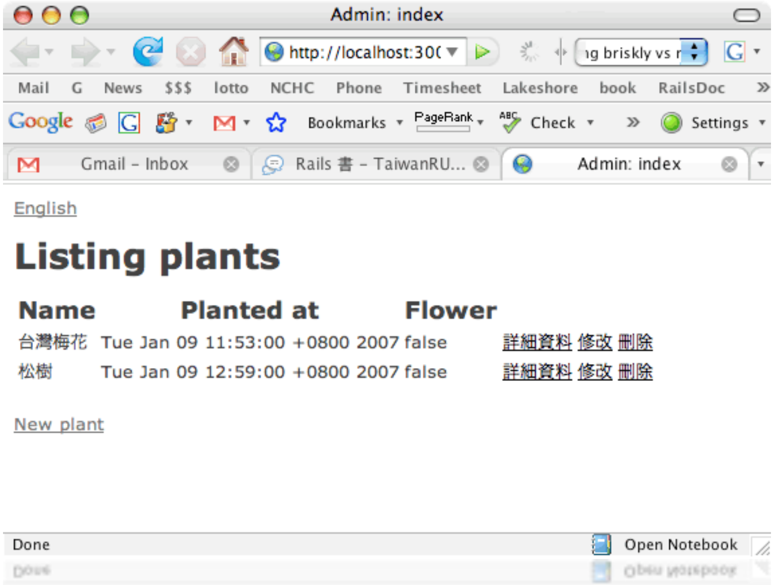
這個 method 要在每次程式要跑之前就要叫，所以要放在 before_filter:

```
class ApplicationController < ActionController::Base
before_filter :set_locale
...
```

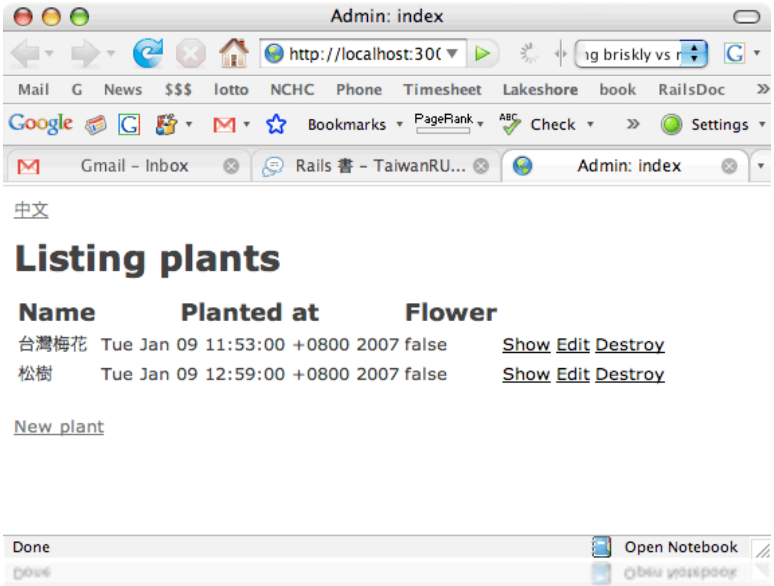
在 set_locale 方法的第二行有一個 ″true″，這是為什麼呢？ 因為 before_filter 所叫的方法一定要回傳 true, 整個程式才會繼續執行，所以我們就回傳一個 true。

現在來瀏覽 http://localhost:3000/admin/ 吧。 一開始 ″中文″ 跟 ″English″ 兩個都會顯現，因為 session 裡面沒有 zh-TW 或 en-US 的參數。 ″詳細資料″， ″修改″， 跟 ″刪除″ 也會顯示，因為我們在 ″設定基本語言″ 裡有把預設值設為 ′zh-TW′。 按一下 ″English″ 的連結吧。 ″Show″, ″Edit″, 跟 ″Destroy″ 果真出現了！ 其他的如果要翻譯的話就照這些方法去翻譯。

中文版



英文版



如果你的程式支持個人客製化(使用者可以 login)，你可以在使用者的 table 裡加一個 ″locale″ 的欄位。 在使用者登入時，把那欄位裡的值叫出來，放入 session[:locale] 裡頭。 使用者在改變語言時要把新的 locale 值寫回資料庫裡，這樣你的程式就會記住每個使用者的愛好了。

Ajax

AJAX 大開眼界

Ajax 主要的特色就是在網頁上改完資料後新的資料會馬上顯現，不需要跳到別頁再顯現。 這樣的話網站看起來反應會很快，不像以前每一個動作後要等整個網頁更新後才好。 Ajax 這個技術在很久以前就有了，但直到 Gmail 跟 Google Maps 開始大量使用時大家才發覺它的好處。 但是，看別的網站用的好跟在自己的網站上套用是兩回事。 Ajax 本身並不是很好用，當初少人用就是這個問題。 直到許多高手寫好的 library 出來時才越來越多人使用。

還記得 scaffold 嗎？ 就是 Rails 自動產生的 CRUD (create, read, update, delete) 介面。 我們之前架的 scaffod 雖然可用，長相並不是怎麼好看，而且沒有 Ajax 功能。 不用擔心，網路上有一位叫 Richard White 的先生把 Ajax 功能加到 scaffold 裡面去了。 要了解他的作品可到 http://www.ajaxscaffold.com/ 參考一下，或是在這裡寫個簡單的程式來體驗比較快。 這程式裡會有基本的 CRUD 功能，也會有 先進的 Ajax 使用者介面。 我們把這個程式叫 flowers。 寫這程式的步驟跟在 Localization 文章裡的 plants 差不多，但為了簡單化，flowers 將不會有中文的功能。 要中文的話請到 Localization 看要怎麼加。

動工了！ 首先，在 MySQL 裡建一個叫 flowers_development 的資料庫。 然後建一個新的 rails 程式:

rails flowers

cd flowers

在這新的 rails 程式裡安裝 Richard White 的 scaffold 程式碼產生器：

gem install ajax_scaffold_generator

產生新的 model 叫做 ″Flower″。

ruby script/generate model Flower

來複習一下以前學過的概念。 在寫新的程式時，我們都會先產生 model，然後再修飾自動產生的 migration 檔案，然後再利用那個檔案產生真正的 table。 這些步驟如果不熟悉的話請看 “Migration 版本管理” 章節。

剛剛 model 有產生了，所以現在要修飾自動產生的 migration 檔。 把 db/migrate/001_create_flowers.rb 打開，在 def self.up 裡放入下面這幾行：

```
create_table (:flowers) do |t|
  t.column "name", :string
  t.column "planted_at", :datetime
  t.column "seed", :boolean
end
```

檔案儲存後下此指令：

```
rake migrate
```

這時 table 已建好了，我們就可以啟動剛剛安裝的 scaffold 程式碼產生器：

```
ruby script/generate ajax_scaffold Flower Admin
```

這個指令會產生一個控制 Flower 的 controller。 Controller 的名字叫作 Admin。 這指令也會產生一些以前 rails 已自動產生過的檔案，新的檔案就有 ajax 的功能。 它會問你確定要覆蓋舊的檔案嗎？

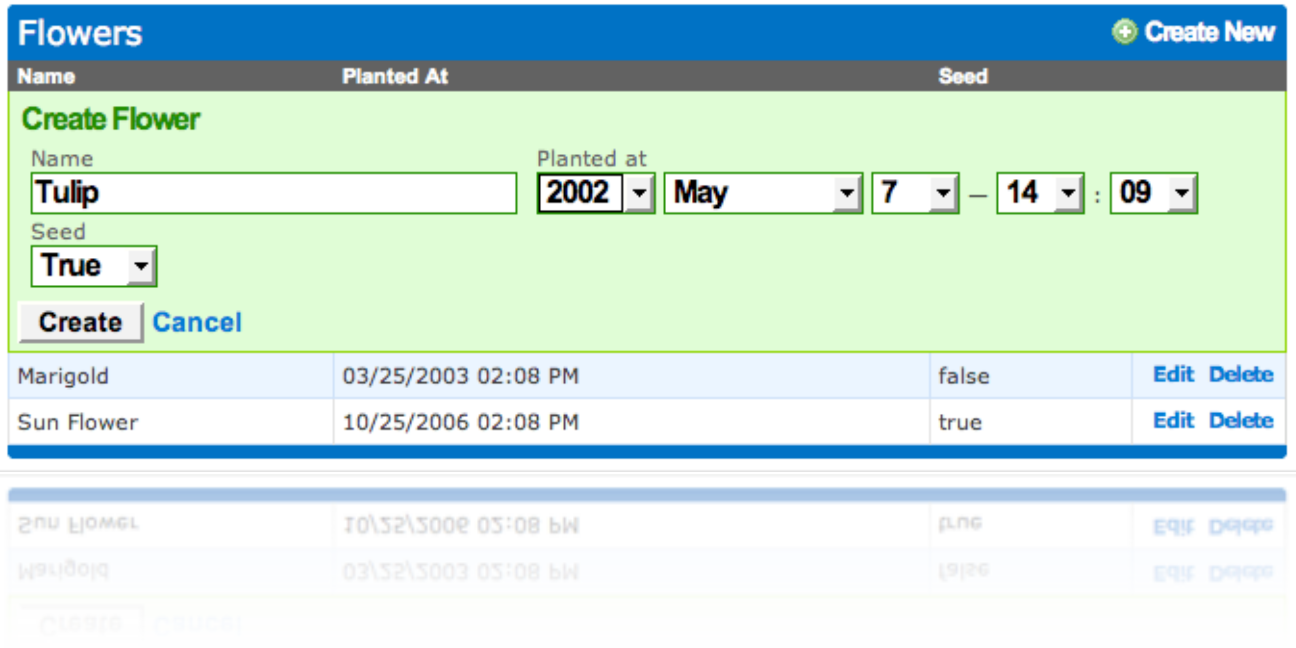
```
overwrite app/models/flower.rb? [Ynaq]
```

按一下 "a" 來全部覆蓋舊的檔案。

來看看新的檔案長的什麼樣子。 打開 app/controllers/admin_controller.rb 哇！多出了好多 method。 原本的 scaffold 只有七個 method (index, list, show, new, create, edit, update, destroy) 現在多了五個 method (update_params_filter, return_to_main, component_update, component, cancel) 而且原本的 method 裡也新增了很多程式碼。 這麼多的訊息真是消化不良。 再來看看 app/views/admin 裡面，也多了好幾個檔案，這麼多東西，是要解釋到那時候？ 還是先不管，來看看多出那麼多的程式碼是增加了那些功能。 啟動伺服器：

```
ruby script/server
```

然後到 http://localhost:3000/admin/ 看看。



嗯！使用者介面顏色配的不錯。 加幾筆假資料吧。 按 ″Create New″ 之後，在同一頁裡會出現新增資料的表格。 新增之後的紀錄也顯示在同一頁中，網頁都不用更新。 用這網站感覺就是操作流程快了許多。 螢幕沒常在更新的話使用者也覺得安全一點，東西不會怕說消失後跑不出來。 現在按一下左邊 ″name″ 這個字，整個資料就自動幫你照那個欄位排了！ 真好。

從頭來看 - 自己做 AJAX

這些這麼酷的功能是怎麼做的呢？ 壞消息是，ajax scaffold 做的太複雜，沒辦法在一篇文章內解釋完。 好消息是，我們會再做一個比較簡單的 ajax 範例叫 horses，從那個範例裡你就可以了解 ajax 的概念，這樣再回來看 flowers 裡的 ajax scaffold 程式碼就看的懂了。

現在我們要建立一個新的 Rails 程式叫 horses。 在這程式裡，我們不會用 ajax scaffold,而是手動增加 ajax 功能。

在這本書裡我們常常建立新的 Rails 程式，現在程序應該背熟了吧？ 首先建一個叫 horses_development 的資料庫。 建好了之後建一個新的 rails 程式:

```
rails horses
cd horses
ruby script/generate model Horse
```

這時候 Migration 會產生一個檔案叫 db/migrate/001_create_horses.rb。 把這個檔案打開，在 def self.up 裡放入：

```
create_table (:flowers) do |t|
  t.column "name", :string
  t.column "height", :integer
  t.column "weight", :integer
  t.column "born_on", :date
  t.column "died_on", :date
end
```

檔案儲存後下此指令：

```
rake migrate
ruby script/generate scaffold Horse Admin
ruby script/server
```

去看看 http://localhost:3000/admin/ 哇！回到以前的 scaffold 了，真懷念。 剛剛這些步驟都是家常便飯，跟 ajax 沒關係，現在就要開始加 ajax 功能了！

步驟一: 包含控制 JavaScript 的 class

Rails 的 ajax 功能是包含在一個叫做 JavascriptHelper 的 class 裡頭。 JavascriptHelper 讓你用 ruby 來控制 JavaScript。 要用 JavascriptHelper 前 要把這小段程式碼放在

app/views/layouts/admin.rhtml 裡的 <head></head> 中間：

```
<%= JavaScript_include_tag :defaults %>
```

這一小段碼會把所需要用到的 ruby class 包裝進來。

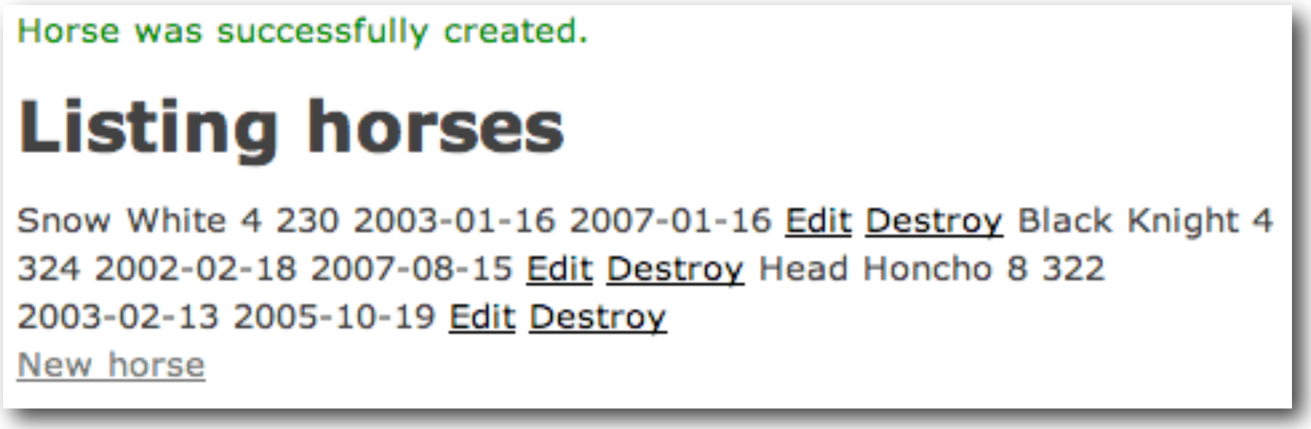
> 小提示
>
> 你可能會想說直接把這段程式碼放在 app/views/layouts/application.rhtml 檔案裡頭，不就不用在每個 controller 的 layout 檔放了嗎？ 這個想法是錯的喔。 一個 controller 如果沒在 app/views/layouts/ 裡設定 layout 檔的話，application.rhtml 才會被用到。 如今我們的 admin controller 已在 app/views/layouts/ 裡有自己的 admin.rhtml，application.rhtml 檔案裡的東西是不會被用到的。

第一步驟其實這樣就好了，但為了要更清楚的展示如何寫 ajax 程式碼， 我們要把 app/views/admin/list.rhtml 簡單化。 把此檔案打開，並把裡面有關 table 的 html code 移除。 產生欄位標題的 code 也移除，還有 <%= link_to 'Show'... %> 也移除。 剩下來的程式碼如下：

```
<h1>Listing horses</h1>
<% for horse in @horses %>
  <% for column in Horse.content_columns %>
    <%=h horse.send(column.name) %>
  <% end %>
    <%= link_to 'Edit', :action => 'edit', :id => horse %>
    <%= link_to 'Destroy', { :action => 'destroy', :id => horse }, :confirm => 'Are you sure?', :post => true
%>
<% end %>

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

啟動伺服器後加一些假資料進去。 現在所有的資料顯示時都擠在一起:



沒關係，我們繼續修改，利用顏色區分不同的資料。

把 <%=h horse.send(column.name) %> **改成**

```
<font color="blue"><%= column.human_name %>: </font><%=h horse.send(column.name) %><br />
```

然後在 <%= link_to 'Destroy'....%> 下面加個 <p> 這樣每一匹馬的資料會一塊一塊的區分開來。

步驟二: 使用 <div><div>

第一步驟所做的動作只有把操控 JavaScript 的程式碼匯入我們的程式裡。 第二步驟要做的就是告知 ajax 資料處理後網頁的那個部份需要更新。 當我們在一個網頁上修改資料時，Ajax 會利用 JavaScript 來送一小段修改資料的指令給伺服器，伺服器處理那段指令後會傳回來一段回覆碼, JavaScript 再利用那段回覆碼把網頁中的一小段內容改掉。 傳送這一小段的指令跟回覆碼總比把整個網頁的內容再傳送一次還快。

雖然 Asynchronous Javascript and XML (Ajax) 裡的 x 是代表 XML, 但伺服器傳回的可以是 html，或 JavaScript, 或純文字等等。 當伺服器傳回 html 回覆碼時， JavaScript 怎麼知道這段 html 要放那邊呢？ 放的地方就是要用 <div><div> 來指定了。

我們把每一匹馬的紀錄當做一個區塊，要改變時只要那個區塊有變動，其他同一網頁的東西不必要動。 所以在 <% for horse in @horses %> 之後要放 <div> ，而 div 的結束點 <div> 要放在 <% end %> 前面。 <div><div> 的用途就是把每匹碼的資料區分開來。 但雖然分成了不同的區塊，我們不同的區塊要編不同的識別碼，這樣伺服器回傳 html 碼時才知道是要代替那個區塊的內容。 區塊怎麼添加識別碼呢？就是把 <div> 裡加一個 id:

把 <div> 改成 <div id="myDiv<%= horse.id %>">

這裡的識別碼我們用了 horse 的 id, 因為 horse 的 id 一定不會重複的。

myDiv 是我自己加的，這只是一個卡位的名字，可以隨便取。 這時你如果把網頁 reload 然後去看它的網頁編碼 (page source)，你就可以看到每匹馬的紀錄會有 <div id="myDiv1"> 或 <div id="myDiv2"> 等等包著。 重點是每個 <div> 有不同的識別碼:

Source of: http://localhost:3000/admin/list

```html
<html>
<head>
  <title>Admin: list</title>
  <link href="/stylesheets/scaffold.css?1161775364" media="screen" rel="Stylesheet"
  <script src="/javascripts/prototype.js?1161775180" type="text/javascript"></script
<script src="/javascripts/effects.js?1161775180" type="text/javascript"></script>
<script src="/javascripts/dragdrop.js?1161775180" type="text/javascript"></script>
<script src="/javascripts/controls.js?1161775180" type="text/javascript"></script>
<script src="/javascripts/application.js?1161775180" type="text/javascript"></script
</head>
<body>

<p style="color: green"></p>

<h1>Listing horses</h1>


  <div id="myDiv1">

      <font color="blue">Name: </font>Snow White<br />

      <font color="blue">Height: </font>4<br />

      <font color="blue">Weight: </font>230<br />

      <font color="blue">Born on: </font>2003-01-16<br />

      <font color="blue">Died on: </font>2007-01-16<br />

      <a href="/admin/edit/1">Edit</a>
      <a href="/admin/destroy/1" onclick="if (confirm('Are you sure?')) { var f = dc
  </div>

  <div id="myDiv2">

      <font color="blue">Name: </font>Black Knight<br />

      <font color="blue">Height: </font>4<br />

      <font color="blue">Weight: </font>324<br />

      <font color="blue">Born on: </font>2002-02-18<br />

      <font color="blue">Died on: </font>2007-08-15<br />

      <a href="/admin/edit/2">Edit</a>
      <a href="/admin/destroy/2" onclick="if (confirm('Are you sure?')) { var f = dc
  </div>

  <div id="myDiv3">

      <font color="blue">Name: </font>Head Honcho<br />

      <font color="blue">Height: </font>8<br />

      <font color="blue">Weight: </font>322<br />

      <font color="blue">Born on: </font>2003-02-13<br />

      <font color="blue">Died on: </font>2005-10-19<br />

      <a href="/admin/edit/3">Edit</a>
      <a href="/admin/destroy/3" onclick="if (confirm('Are you sure?')) { var f = dc
  </div>
```

步驟三: link_to_remote

從步驟一到現在我們已把 JavaScript 的控制檔匯到程式裡，也有用 <div> 指定了可變動的區塊，剩下來的就是指定那個連結會變動那個區塊。

把

```
<%= link_to 'Edit', :action => 'edit', :id => horse %>
```

改成

```
<%= link_to_remote image_tag("/images/edit.png", {:alt => 'edit'}),
        :update => "myDiv#{horse.id}",
        :url => { :action => 'edit', :id => horse.id }
%>
```

原始的 link_to 會產生一個連結，連結按了後整個頁面會跳到 http://localhost:3000/admin/edit/ 那裡。 現在把 link_to 改成 link_to_remote 後， link_to_remote 就會在幕後用 JavaScript 跟伺服器溝通。 link_to_remote 是 Rails 用來包裝 ajax 指令的 method。 我們要給它一些變數它才知道有那些東西要處理。

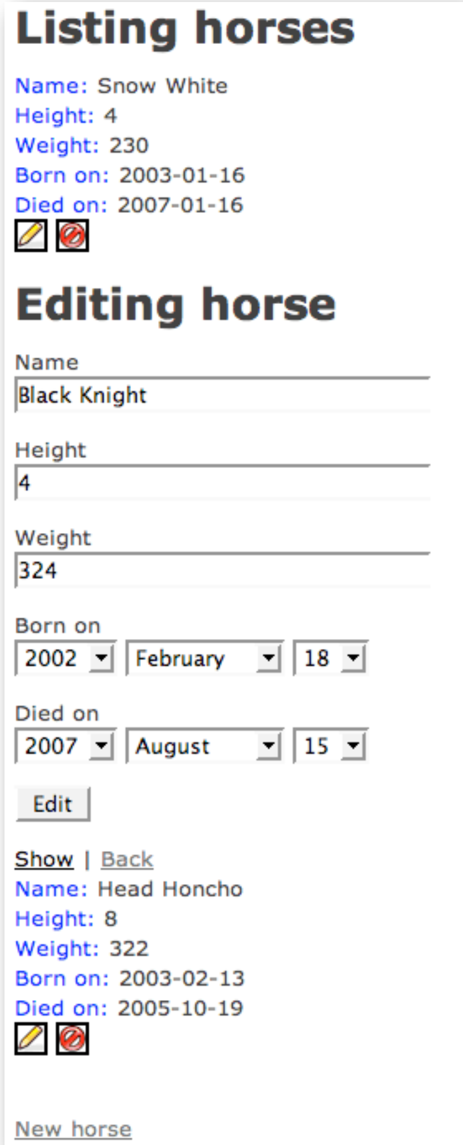
image_tag 裡指定這個連結的圖案檔放那裡。 記得要放一張 edit.png 在 public/images/ 裡頭。 別的圖片格式也可以，不用一定是 png 檔。 圖片名字也可以改，只要檔案名稱跟 image_tag 裡一樣就好。 :alt 是圖片檔如果找不到時要顯示的文字。 如果不要用圖片來當連結，image_tag("/images/edit.png", {:alt => 'edit'}) 這段字換成 “Edit” 它就會顯示文字了:

```
<%= link_to_remote ("Edit",
        :update => "myDiv#{horse.id}",
        :url => { :action => 'edit', :id => horse.id })
%>
```

:update 這變數包含著資料傳回來時要取代區塊的辨識碼。

:url 指定要用那個 method 來處理傳給伺服器的碼。 上面指定的是 Admin controller 裡的 method, 叫 edit。 edit 這 method 在跑時也需要知道處理那匹馬，所以我們也有傳入馬的 id。

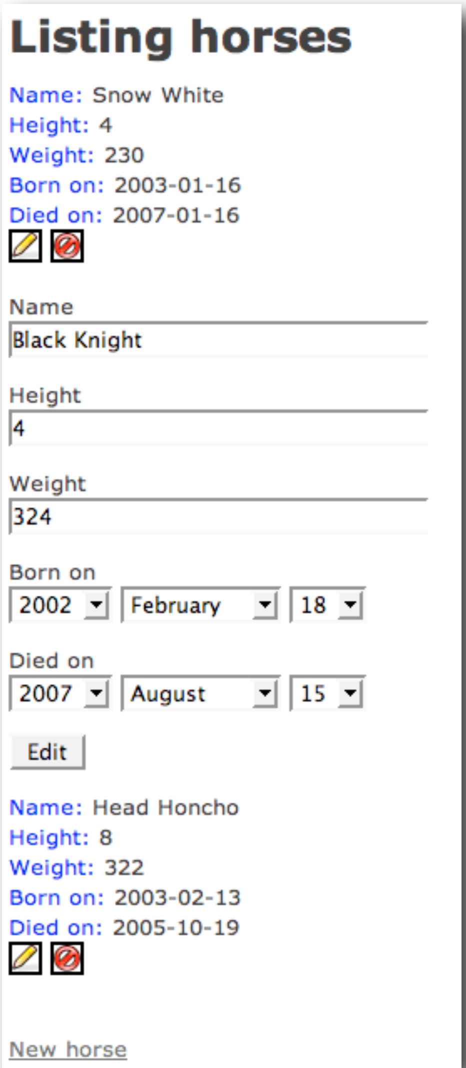
回到 http://localhost:3000/admin/ 看看。 按一下任何資料的 "edit", 螢幕上會出現一些可以改的欄位，但也會出現 "Show" 跟 "Back":



因為我們整個修改的流程都會在同一頁上，這兩個連結就不需要了。 把 app/views/admin/edit.rhtml 檔案打開，移除

```
<%= link_to 'Show', :action => 'show', :id => @horse %> |
<%= link_to 'Back', :action => 'list' %>
```

<h1>Editing horse</h1> 也移掉。



剩下來還有一個問題。 當我們在網頁上改完一筆資料後，

它會跳到 http://localhost:3000/admin/show/ 去。 我們不希望它自動跳去 show 那裡，所以把 app/controllers/admin_controller.rb 開啟，把 ″update″ method 裡的

```
redirect_to :action => 'show', :id => @horse
```
改成
```
redirect_to :action => 'list'
```

現在回網頁再試試看，改完後沒跳到別的網頁了！

增加 ajax 的方法就只需這三步驟， 以後你會接觸到各式各樣的 ajax 功能，但應用的概念還是這三個基本的步驟喔。

AJAX 特效

Ajax 的另一個好處是它可加特效。 不用特別下載軟體。 使用者如果按一份資料的 destroy 連結時，我們要讓那份資料像″煙″一樣消失掉。 把 app/views/admin/list.rhtml 裡的

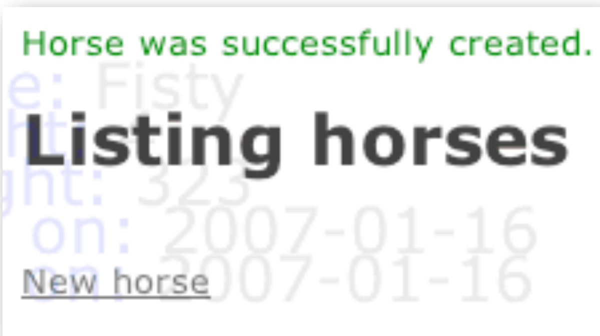
```
<%= link_to 'Destroy', { :action => 'destroy', :id => horse }, :confirm => 'Are you sure?', :post => true %>
```

換成

```
<%= link_to_remote image_tag("/images/delete.png", {:alt => 'delete'}),
        :url => { :action => 'destroy', :id => horse.id },
        :complete => visual_effect(:puff, "myDiv#{horse.id}"),
        :confirm => 'Are you sure you want to delete?', :post => true %>
```

圖片要記得放喔。 這個 destroy 連結的 link_to_remote 跟 edit 連結的 link_to_remote 不同的地方是它多個 :complete 跟 :confirm。

:confirm 是再確認的視窗。 按了 ″delete″ 之後，視窗會問說 ″Are you sure you want to delete?″



:complete 是 action 處理後要做的動作，像增加特別效果。 這些特效是由 http://script.aculo.us/ 網站上的 Thomas Fuchs 寫的。 script.aculo.us 本身是架在 prototype 上。 Prototype 是一個 JavaScript library，把一些 Ajax 的功能寫好包裝成 class 來讓大家用。 Script.aculo.us 在這 library 上多建了動畫功能，drag and drop，同欄位修改，測試功能等等。 這個 library 不只是 RoR 可用，其他語言像

pearl, php, java, .net 等等都可用。 因為 Rails 本身就有把 script.aculo.us 內建，要使用時需要做剛剛所學過的 "步驟一: 包含控制 JavaScript 的 class"就好，不要費時間去他的網站查看在 RoR 怎麼套用喔。 還沒讀到這裡之前，你應該已迫不及待的去看看 "特效" 是什麼了吧？ 沒的話先去看看。 嗯，資料像煙一樣消失的確是很酷！

除了 "煙" 之外，其他效果多得很呢：
到
http://wiki.script.aculo.us/scriptaculous/show/CombinationEffectsDemo
來看，此頁所有特效都一目了然，讓你很快的決定你要的特效是什麼。

* 消失/出現 的特效
appear / fade　出現或消失。 div 要先沒看到，這個效果才會有效
blindup / blinddown 像窗簾一樣拉上或拉下
slideup / slidedown 整個 div 滑上或滑下
grow / shrink　由中間往外膨脹或由外往中間縮小

* 消失 的特效
dropout 往下掉然後消失
fold　由下縮到上，然後由右縮到左
squish 同時由下縮到上，由右縮到左
puff　煙化掉 (讓一切煩惱隨著風吹而散)
fade　淡化掉

* "注意!" 的特效

highlight 銀光筆突顯。 如果資料有 update 的話，用這個功能來突顯會很好
pulsate　消失，出現，消失，出現。。。像心跳一樣

我們也可以進一步的控制這些特效，像顏色，特效長度，等等。 以剛剛 destroy 的範例來說，來做一個客製化的特效吧。

```
:complete => visual_effect(:highlight, "myDiv#{horse.id}",
    :delay    => 5,
    :duration => 6,
    :from     => 0.3,
    :to       => 0.7,
    :queue    => 'front')
```

:highlight 我們要客製化的特效名稱，這個特效是”銀光筆突顯”
:delay 等 x 秒後再開始特效
:duration 特效重頭到尾要跑 x 秒
:from 從特效的那階段開始，可選由 0.0 到 1.0. 預設是 0.0
:to 到特效的那階段結束，可選由 0.0 到 1.0. 預設是 1.0
:queue 特效跟別的特效同時跑的排程。 有三種選擇，front (跑在別的特效前), parallel (同時跑), back (跑在別的特效後)

從剛剛的範例來看，這個”銀光筆突顯”特效會在資料刪除後五秒開始，從頭到尾會跑六秒，如同時有別的特效的話，會在那些特效後跑，跑的時候會在特效平常的 30% 效果開始跑，跑到 70% 的效果後停止。

這些可客製化的變數叫做 ”options”。 :delay， :duration， :from， :to， :queue 等都是 options。 其他的 options 還有很兩三個，像 :fps, :transition, :sync, :direction 等等。 這些不常用到的東西如果要了解的話到此網站看看：
http://wiki.script.aculo.us/scriptaculous/show/CoreEffects

自己探索

script.aculo.us 網站上還有欄位原地修改的功能 (in-place editing)，不用按 ”submit” 就可以改資料了。 還有，<div> 裡包含的東西可以一整塊拖放，也就是drag and drop。 很有用的。 可惜是這些功能就要自己上它的網站去研究了！ 要寫這些先進的功能之前，還是先回到 ajaxscaffold.com，看看它的 ajax 的 scaffold 是怎麼寫的吧。

想參考看看別人 ajax 功夫練到了什麼程度嗎？ 雖然以下這些網站沒有開放原始碼，但 blog.wired.com/monkeybites/ 網站上的讀者覺得他們的 Ajax 功能設計的不錯。 來參考看看，以後自己的程式也用的到：

del.icio.us , Local.Live.com , GMail , Google Calendar , Google Spreadsheets , Digg , YouTube , Yelp , Blinklist , Bloglines , Basecamp , Writely , Dimewise , Kayak , Spurl , LibraryThing , Last.fm , NetVibes, Meebo, Mint, Shopify, Traineo。

資源

http://script.aculo.us/ - Javascript 特效

http://prototype.conio.net/ - Ajax library

http://mir.aculo.us/ - Thomas Fuchs 的 blog, script.aculo.us 的作者

Logging

寫 LOG

在 rails 的程式裡隨時都可以做 logging:

logger.debug("debug: current username is #{user.name}")

“ ” 裡可以隨便放任何的資訊。 #{} 裡可放程式的變數，像使用者的名字 user.name，方便我們紀錄誰登入過。

Logging 有分五種不同的嚴重性。 從最輕微到最嚴重的排列如下：

debug: 除錯

info: 訊息

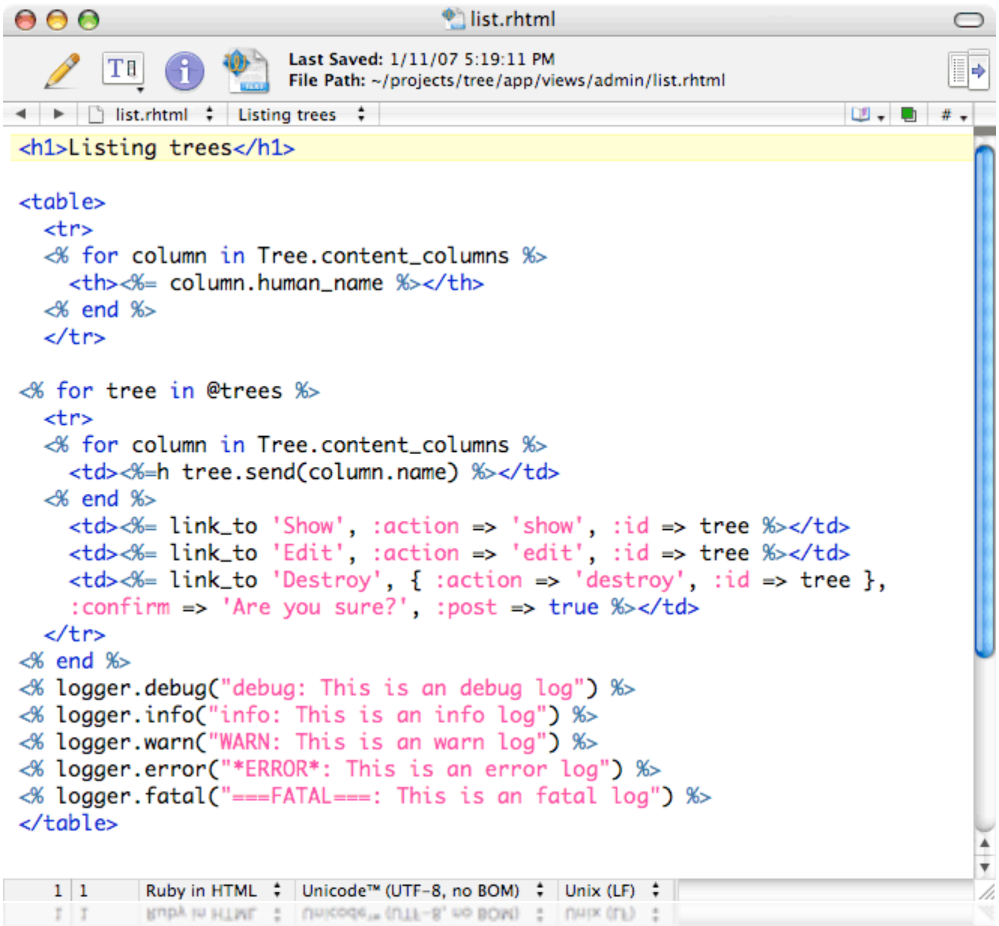
warn: 警告

error: 錯誤訊息

fatal: 嚴重錯誤

剛剛的例子就是除錯訊息的紀錄方法， 如要紀錄警告資訊的話，在程式裡寫：

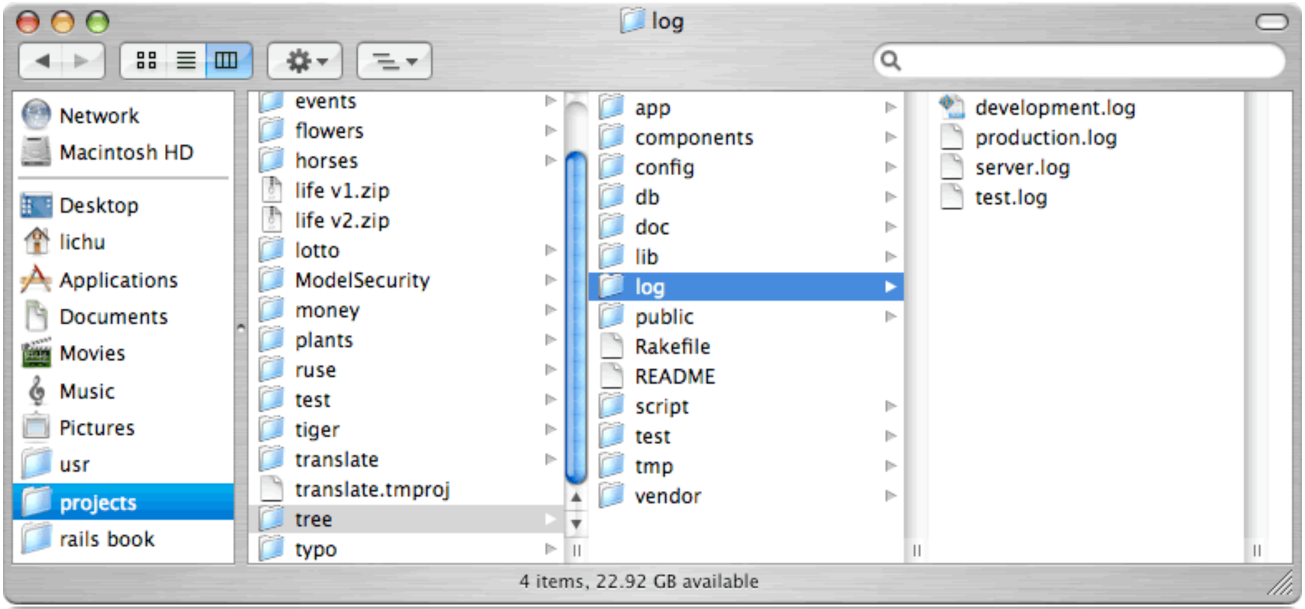
logger.warn("WARN: This transaction is irreversible")



圖片裡顯示的是在一個 scaffold 產生出來的檔案 (list.rhtml) 裡寫入五個 logger。 每當使用者瀏覽 list.html 時，這五個訊息會被寫入 log 檔裡。

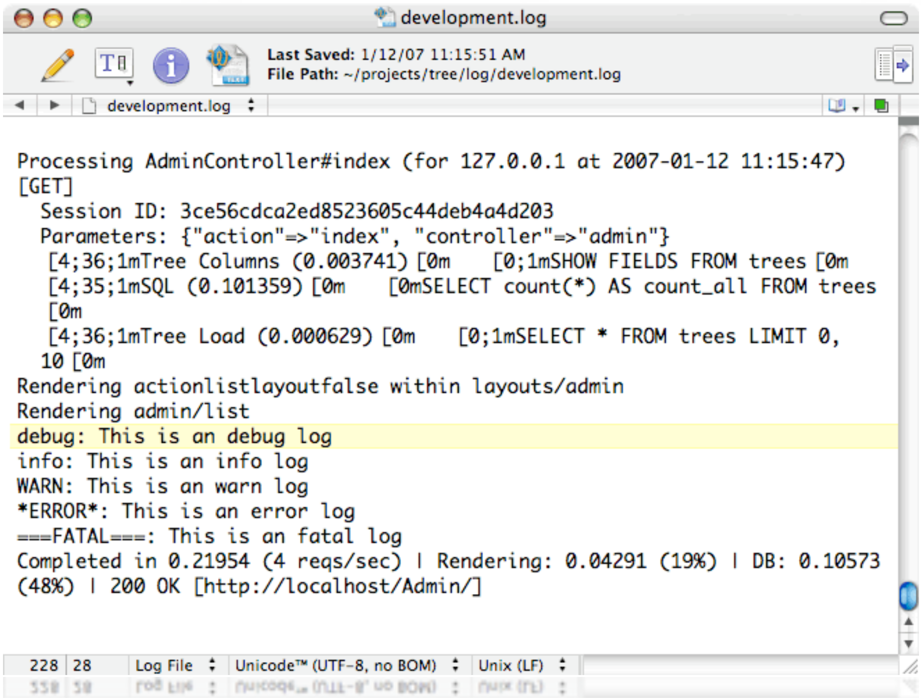
讀 LOG

紀錄起來的訊息可以在程式下的 log/ 目錄裡看到。如果在開發的模式下跑，所有的 log 會出現在 log/development.log。 在測試模式的話就會在 log/test.log。 在上線模式的話就會在 log/production.log。



每次寫新的 Rails 程式時，預設是在 development （開發）模式下跑，所以啟動 伺服器後，log/development.log 裡有資料寫入了，其他的 test.log 和 production.log 目前是空白的。

Log/development.log 的內容如右：



要看 log 不用每次都把檔案打開。 伺服器的系統如是 U/linux 的話，可

以用 tail 來即時顯示在螢幕上。 下面這個範例會把開發模式下的 log 顯示在螢幕上，一有新的紀錄就會顯示:

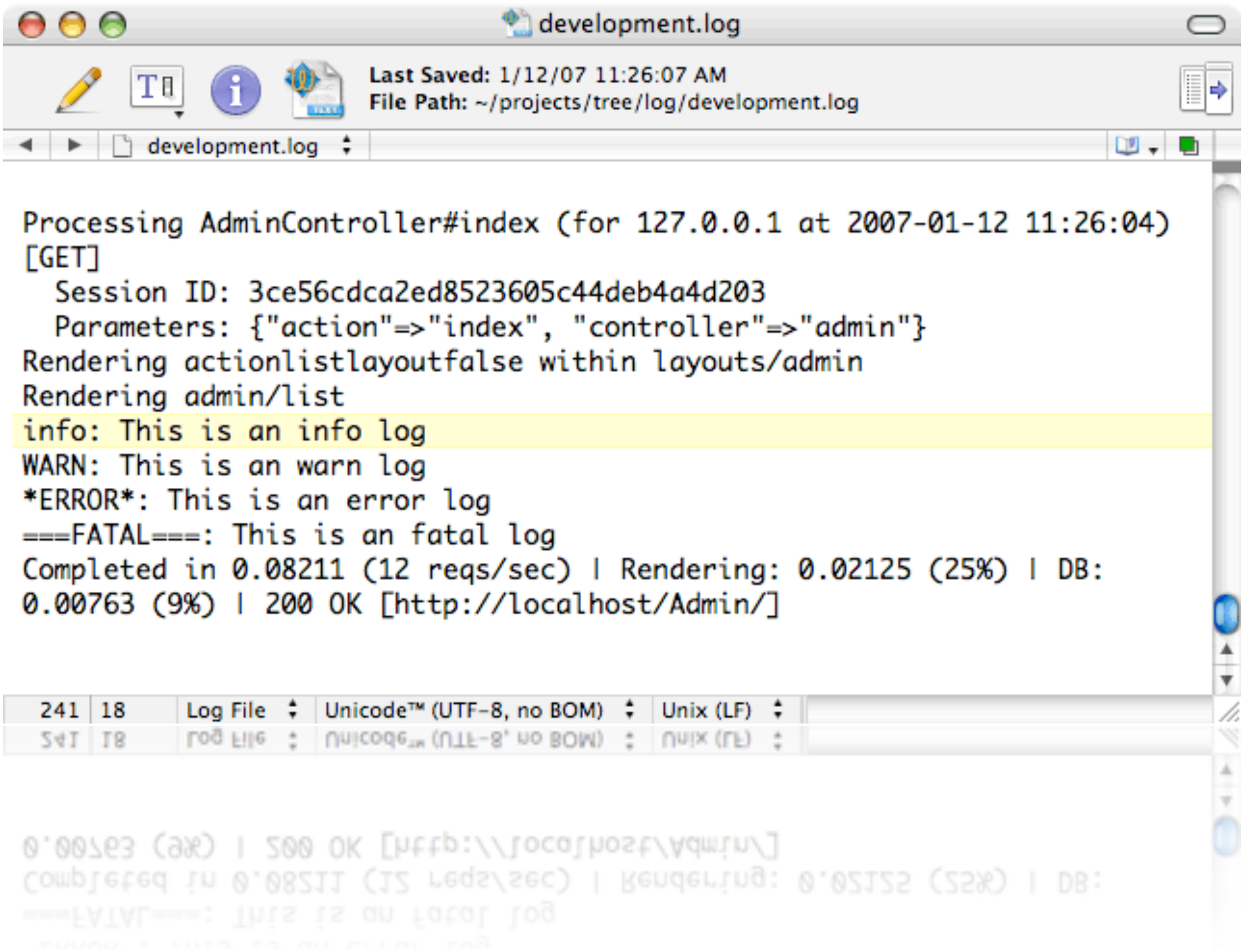
```
cd /myRailsApp/log
tail -f development.log
```

設定那種類的 LOG 要紀錄

你也可以設定在那一種模式下要紀錄那種類的訊息。 例如，開發模式 (development) 的預設值是紀錄所有的 logger，我們如果覺得那太囉唆，只想紀錄比 info 還嚴重訊息，就把 /config/environments/development.rb 打開，把這行字加在最後面：

```
config.log_level = :info
```

加完後伺服器重開，網頁重整 (reload)，去 log/development.log 裡就會看到新的 log:

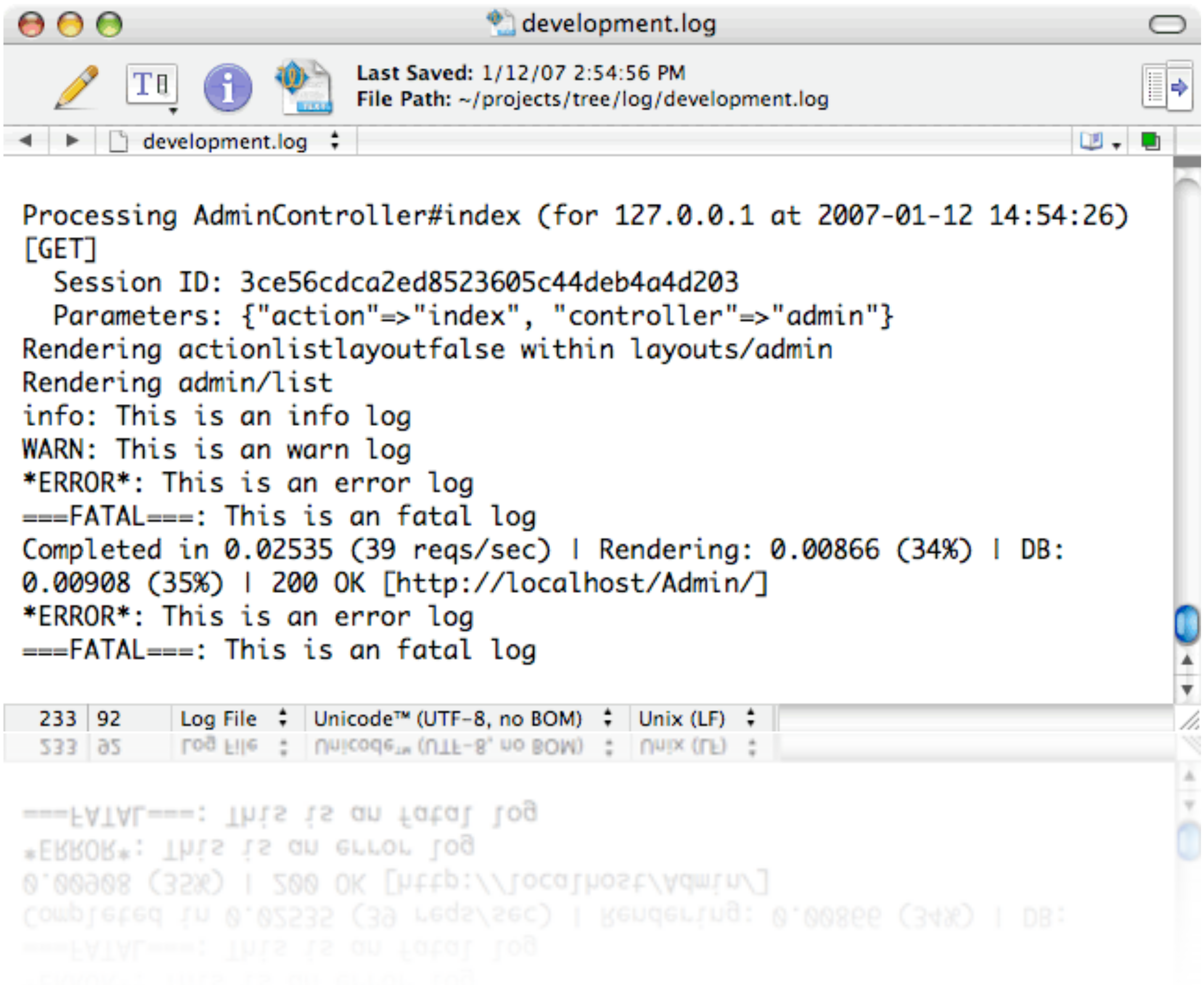


新的 log 裡所紀錄的就沒包括 debug 的類型了。

那如果嚴重性 error 以下的都不要呢？ 把剛剛的 config.log_level 那行改成

config.log_level = :error

重開 server 後，新的 log 就只有 error 跟 fatal 了：



大部份的人在開發環境裡會紀錄所有類型的 log，在上線環境裡才把 config.log_level 設成 :error，來紀錄比較嚴重的錯誤訊息。

不要紀錄敏感的資料

Rails 1.1.6 版本有新增一個可以不要紀錄敏感資料的功能。 敏感資料為密碼或信用卡號等等。 使用方法如下： 把工作中程式的 app/controllers/application.rb 打開，在

class ApplicationController < ActionController::Base

下面加一行

filter_parameter_logging "creditcard"

這就會叫 logger 不要紀錄變數名字裡有含 "creditcard" 這字串的資料，像 [customer][primary_creditcard] 跟 [customer][secondary_creditcard] 這兩個變數裡含的資料都不會被記下來。

更好的 LOGGING 方法

Rails 用的 logger 是 ruby 內建的，雖然好用可是功能有點簡單。如要更好的話，就研究 log4r 看看：http://log4r.sourceforge.net/ log4r 目前跟 rails 的 benchmark 模組有衝突，所以先看可不可用再細部研究。

http://wiki.rubyonrails.org/rails/pages/logger

如果 logger 已能滿足你的需求，深一步了解可到 http://wiki.rubyonrails.org/rails/pages/logger