

Sommaire

Ruby On Rails.....	1
Avant propos.....	1
Mise en place de l'environnement.....	1
Configuration.....	2
Modèle MVC.....	2
Création des modèles.....	3
Génération des fichiers relatifs à la table.....	3
Fichiers de migration.....	3
Génération des tables.....	4
Edition des modèles.....	5
Création d'un contrôleur.....	5
Mise en place des vues.....	6
Le layout.....	6
La vue.....	7
WebRick.....	7
Pour aller un peu plus loin.....	7
Les environnements.....	9
 Copyright.....	 10

Ruby On Rails

Ruby On Rails

par GEDsismik

Monter un projet Ruby On Rails (RoR) pas à pas.

Avant propos

Ruby On Rails (RoR) est un framework web écrit en ruby. Ce document vous présente comment faire un petit projet Ruby On Rails.

On supposera installé et correctement configuré :

- un SGBD supporté par RoR (personnellement, j'utilise MySQL)
- Ruby
- Rails et ses dépendances

Il vous faudra aussi des notions de ruby.

Dans ce document, je mettrai en place un gestionnaire de livre rudimentaire. Le but est de présenter les bases d'un projet Rails. Il est évident que ces informations sont loin d'être exhaustives vue que RoR peut remplir tout un livre de 500 pages.

Mise en place de l'environnement

```
$ rails monsite
  create
  create  app/controllers
  create  app/helpers
  create  app/models
  create  app/views/layouts
[...]
```

Le répertoire monsite vient d'être créé ainsi que la structure initiale du projet. Entrez maintenant dans ce répertoire. Normalement, il devrait contenir les fichiers et répertoires suivants :

```
app
components
config
db
doc
lib
log
public
Rakefile
README
script
test
tmp
vendor
```

Configuration

Si vous n'avez pas de base de donnée, créez-en :

```
mysql> create database monsite_development;
mysql> create database monsite_test;
mysql> create database monsite_production;
```

Note : J'en ai créé 3 pour pouvoir utiliser 3 environnements différents. Une seule peut suffire, ça dépend de ce que vous voulez faire.

Nous allons ensuite éditer `./config/database.yml`. Vous avez 3 environnements de pré-configurées de la sorte :

```
production:
  adapter: mysql
  database: monsite_production
  username: root
  password:
  host: localhost
```

Avec :

- adapter le type de base de donnée (mysql, postgres, sqlite...)
- database le nom de la base de donnée
- username le nom de l'utilisateur de la base de donnée
- password le mot de passe à la base de donnée
- host le serveur de base de donnée

Dans le cas de sqlite, on a :

- adapter le type de base de donnée (sqlite)
- dbfile nom du fichier de base de donnée

Editez-le en fonction des paramètres de votre base de donnée. Exemple :

```
production:
  adapter: mysql
  database: monsite_production
  username: gedsismik
  password: mypassword
  host: localhost
  socket: /var/run/mysql/mysql.sock
```

Note : Sous Slackware, j'ai du ajouter la ligne socket: `/var/run/mysql/mysql.sock` sinon il ne trouve pas la socket MySQL.

Modèle MVC

Ruby On Rails utilise le motif de conception MVC (modèle-vue-contrôleur). Je m'explique : dans un projet RoR, vous séparez :

- les modèles (répertoire `./app/models`) : ce qui concerne les données.

Dev-rails

Dans notre cas, c'est donc principalement les tables des bases de données.

- les vues (répertoire ./app/views) : ce qui concerne l'interface.

Dans notre cas, il s'agit de la présentation des pages (en HTML et Ruby embarqué, nous verrons ça plus tard).

- les contrôleurs (répertoire ./app/controllers) : concerne la gestion des événements.

Dans notre cas, ce sont les classes qui feront le lien entre les tables (models) et le html (views). Ce qui revient aux appels aux tables via les modèles et la création de variables dont le contenu sera affiché dans les vues.

Pour palier à certaines fonctions qui ne peuvent pas suivre le modèle MVC à la lettre, Rails dispose d'une partie appelée helpers (./app/helpers) qui contient des classes accessibles aux vues et qui contient du code ayant accès aux modèles sans passer par les contrôleurs.

Création des modèles

Génération des fichiers relatifs à la table

On va utiliser le script ./script/generate pour générer la première table.

```
$ ./script/generate model livre
  exists  app/models/
  exists  test/unit/
  exists  test/fixtures/
  create  app/models/livre.rb
  create  test/unit/livre_test.rb
  create  test/fixtures/livres.yml
  create  db/migrate
  create  db/migrate/001_create_livres.rb
$ ./script/generate model auteur
  exists  app/models/
  exists  test/unit/
  exists  test/fixtures/
  create  app/models/auteur.rb
  create  test/unit/auteur_test.rb
  create  test/fixtures/auteurs.yml
  exists  db/migrate
  create  db/migrate/002_create_auteurs.rb
```

Fichiers de migration

Les fichiers de migration/création de table se trouvent dans ./db/migrate. Il va falloir les éditer pour ajouter les champs que vous voulez.

Note : ne faites pas de champ "id", ils sont automatiquement créés et gérés par Rails.

create_livres.rb

Fichier ./db/migrate/001_create_livres.rb

```
class CreateLivres < ActiveRecord::Migration
  def self.up
    create_table :livres do |t|
```

Dev-rails

```
        t.column :titre, :string, :null => false
        t.column :auteur_id, :integer, :null => false
        t.column :quantite, :integer, :default => "1"
      end
    end

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

create_auteurs.rb

Fichier ./db/migrate/002_create_auteurs.rb

```
class CreateAuteurs < ActiveRecord::Migration
  def self.up
    create_table :auteurs do |t|
      t.column :nom, :string
      t.column :prenom, :string
    end
  end

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

Notez que vous pouvez réunir les deux fichiers en un seul :

./db/migrate/001_initial.rb

```
class Initial < ActiveRecord::Migration
  def self.up
    create_table :livres do |t|
      t.column :titre, :string, :null => false
      t.column :auteur_id, :integer, :null => false
      t.column :quantite, :integer, :default => "1"
    end
    create_table :auteurs do |t|
      t.column :nom, :string, :null => false
      t.column :prenom, :string
    end
  end

  def self.down
    drop_table :livres
    drop_table :auteurs
  end
end
```

Génération des tables

Toujours dans le repertoire monsite, exécutez rake migrate.

```
$ rake migrate
```

Fichiers de migration

Dev-rails

```
(in /home/gedsismik/monsite)
== CreateLivres: migrating =====
-- create_table(:livres)
--> 0.0441s
== CreateLivres: migrated (0.0442s) =====

== CreateAuteurs: migrating =====
-- create_table(:auteurs)
--> 0.0457s
== CreateAuteurs: migrated (0.0458s) =====
```

Si vous regardez dans votre base de donnée, vous trouverez donc 3 tables :

```
- auteurs
- livres
- schema_info
```

Cette dernière table contient en fait la version de la base de donnée. Les versions sont incrémentées lors du `./script/generate model <nom table>` et sont contenus dans le nom du fichier `./db/migrate/001_initial.rb` est donc la version 1 de la base.

Edition des modèles

Intéressons-nous aux fichiers `./app/models/auteur.rb` et `./app/models/livre.rb`. Ces classes héritant de `ActiveRecord::Base`, presque tout est déjà fait. Nous allons juste ajouter la relation : "un auteur écrit plusieurs livres". Cette relation se traduit dans `./app/models/auteur.rb` par :

```
class Auteur < ActiveRecord::Base
  has_many :livre
end
```

En contrepartie, on peut avoir besoin d'accéder à l'auteur d'un livre. Modifions donc `./app/models/livre.rb` en :

```
class Livre < ActiveRecord::Base
  belongs_to :auteur
end
```

Et voilà ! Ca devrait suffir pour les modèles. En effet, Rails fera tout seul le lien entre les deux tables grâce à `auteur_id`.

Création d'un contrôleur

Tout d'abord, utilisons, comme pour les modèles, le script `./script/generate` :

```
$ ./script/generate controller bibliotheque
  exists  app/controllers/
  exists  app/helpers/
  create  app/views/bibliotheque
  exists  test/functional/
  create  app/controllers/bibliotheque_controller.rb
  create  test/functional/bibliotheque_controller_test.rb
  create  app/helpers/bibliotheque_helper.rb
```

Editons maintenant `./app/controllers/bibliotheque_controller.rb`. Par défaut, vous avez :

Dev-rails

```
class BibliothequeController < ApplicationController
end
```

Nous allons créer des actions. Un contrôleur peut comporter plusieurs actions.

```
class BibliothequeController < ApplicationController
  def index
    listeAuteur
    render_action 'listeAuteur'
  end

  def listeAuteur
    @auteurs = Auteur.find(:all, :order => "nom,prenom")
  end

  def ajoutAuteur
    param = @params['auteur']
    auteur = Auteur.new(param)
    if auteur.save
      flash[:ok] = "L'auteur a été ajouté;"
    else
      flash[:error] = "Problème à l'ajout de l'auteur"
    end
    # ajoutAuteur n'a pas besoin de vue. Une fois l'auteur ajouté
    # on réaffiche la liste
    redirect_to :action => 'listeAuteur'
  end
end
```

L'action index est l'action par défaut. L'action listeAuteur donnera la liste des auteurs. On constate ici qu'on peut largement se passer d'écrire du SQL dans Rails. @auteurs est une variable qui contiendra tous les auteurs.

Mise en place des vues

Le layout

Tout d'abord, créons une layout. Le layout est la structure commune à toutes les pages. Ils se trouvent dans ./app/views/layouts/.

```
$ touch app/views/layouts/bibliotheque.rhtml
```

Editons maintenant ce fichier :

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//FR" "http://www.w3.org/TR/html4/loose.d
<html>
<head>
  <title>Bibliothèque - <%= controller.action_name %></title>
  <meta name="author" content="GEDsismik">
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-15">
</head>
<body>
  <h1>Bibliothèque</h1>
  <br/>
  <div class="all">
  <p style="color: green"><%= flash[:notice] %></p>
  <%= @content_for_layout %>
```

Dev-rails

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

C'est du Ruby embarqué dans du HTML. Pour ceux qui connaissent, c'est un peu le même principe que le PHP. On écrit du HTML et aux endroits où on veut mettre du ruby, on place le code dans une balise `<% #code %>`. Si on veut afficher la sortie du code, on place le code dans une balise `<%= #code %>`.

`<%= @content_for_layout %>` affichera la partie propre à l'action.

La vue

Les vues sont dans un répertoire de la forme `./app/views/<contrôleur>/` et porte le nom `<action>.rhtml`

Créons donc le fichier `./app/views/bibliotheque/listeAuteur.rhtml` :

```
<h2>Liste des auteurs</h2>
<ul>
<%
  @auteurs.each do |auteur|
    %><li><%= auteur.nom + " " + auteur.prenom %></li><%
  end
%>
</ul>

<h2>Ajout d'un auteur</h2>
<%= start_form_tag :action => "ajoutAuteur" %>
  <p><label>Pr&eacut;nom : <%= text_field 'auteur', 'prenom' %></label></p>
  <p><label>Nom : <%= text_field 'auteur', 'nom' %></label></p>
  <%= submit_tag("Ajouter") %>
<%= end_form_tag %>
```

WebRick

Vous disposez d'un serveur de test appelé WebRick. Pour le lancer, tapez `./script/server`

```
$ ./script/server
=> Booting WEBrick...
=> Rails application started on http://0.0.0.0:3000
=> Ctrl-C to shutdown server; call with --help for options
[2006-10-19 18:32:57] INFO WEBrick 1.3.1
[2006-10-19 18:32:57] INFO ruby 1.8.5 (2006-08-25) [i686-linux]
[2006-10-19 18:32:57] INFO WEBrick::HTTPServer#start: pid=26166 port=3000
```

Par défaut, ce serveur écoute le port 3000. Ouvrez donc un navigateur et tapez : `http://localhost:3000`. Normalement, vous devriez obtenir une page de test. Les url de Rails par défaut se décompose de la sorte `http://localhost:3000/<contrôleur>/<action>[/<parametres>]`. Tapez donc dans votre navigateur `http://localhost:3000/bibliotheque/listeAuteur`.

Pour aller un peu plus loin

Ajoutons le support des livres dans `./app/controllers/bibliotheque_controller.rb` :

```
class BibliothequeController < ApplicationController
```

Dev-rails

```
def index
  listeLivres
  render_action 'listeLivres'
end

def listeAuteur
  @auteurs = Auteur.find(:all, :order => "nom,prenom")
end

def listeLivres
  @livres = Livre.find(:all, :order => "titre")
  @auteurs = Auteur.find(:all, :order => "nom,prenom").map{ |u| [ u.nom + " (" + u.
end

def ajoutAuteur
  param = @params['auteur']
  auteur = Auteur.new(param)
  if auteur.save
    flash[:ok] = "L'auteur a été ajouté;"
  else
    flash[:error] = "Problème à l'ajout de l'auteur"
  end
  redirect_to :action => 'listeAuteur'
end

def ajoutLivres
  param = @params['livre']
  livre = Livre.new(param)
  if livre.save
    flash[:ok] = "Le livre a été ajouté;"
  else
    flash[:error] = "Problème à l'ajout du livre"
  end
  redirect_to :action => 'listeLivres'
end
end
```

Créons du même coup la vue correspondante à listeLivres ./app/views/bibliotheque/listeLivres.rhtml :

```
<% if @livres.blank? %>
  <h2>Liste des ouvrages</h2>
  <ul>
  <%
    @livres.each do |livre|
      %><li><%= livre.titre %> - <%= livre.auteur.prenom + " " + livre.auteur.n
    end
  %>
  </ul>
<% else %>
  <i>La bibliothèque ne contient aucun livre.</i>
<% end %>

<% unless @auteurs.blank? %>
  <h2>Ajout d'un livre</h2>
  <%= start_form_tag :action => "ajoutLivres" %>
  <p><label>Titre : <%= text_field 'livre', 'titre' %></label></p>
  <p><label>Nombre d'exemplaire : <%= text_field 'livre', 'quantite' %></label></p>
  <p><label>Auteur : <%= select('livre', 'auteur_id', @auteurs) %></label></p>
  <%= submit_tag("Ajouter") %>
  <%= end_form_tag %>
<% end %>
```

Les environnements

Pour utiliser les différents environnements, vous devez manipuler la variable `RAILS_ENV`. Par défaut, vous êtes dans le premier environnement du fichier `./config/database.yml` (à savoir en général `development`). Pour un rake migrate en production par exemple :

```
RAILS_ENV=production rake migrate
```

notez que l'environnement `test` efface les tables à chaque rake et que l'environnement `production` n'affiche aucune information en cas d'exception (vous devez alors vous reporter aux fichiers de log `./log/<environnement>.log`).

Copyright

Copyright © 19/10/2006, GEDsismik



*Ce document est publié sous licence Creative Commons
Attribution, Partage à l'identique 2.0 :*
<http://creativecommons.org/licenses/by-sa/2.0/>