Manuel PARENT, Benjamen LECOMTE (LP SIL IDSE Parcours JV)

Ruby on Rails

Un framework qui vise la productivité

Présentation générale

Ruby... le langage.

Ruby est un langage de programmation open source distribué sous licence Ruby/GNU GPL créé par le japonais Yukihiro Matsumoto (surnommé « Matz ») en 1995. Multiparadigme, il supporte tout aussi bien la programmation impérative que la programmation orientée objet à typage dynamique et met l'accent sur la simplicité de sa syntaxe et de son fonctionnement.

Comme beaucoup de langages web, c'est un langage de script et doncson exécution passe nécessairement par une machine virtuelle. Parmi celles-ci, on peut notamment mentionner *Ruby* (la VM « officielle »), *Jruby* (une VM écrite en Java) ou encore *IronRuby* (basée sur la technologie .NET de Microsoft). L'avantage qui en découle est naturellement la portabilité du code : une application Ruby peut potentiellement s'exécuter sur n'importe quel OS ou processeur tant qu'une VM adaptée existe.

Ruby on Rails... le framework.

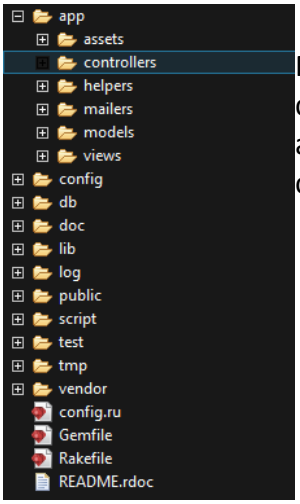
Créé en 2005, Ruby on Rails, également appelé Rails ou RoR au sein de la communauté est un framework libre distribué sous licence MIT destiné au prototypage et au développement d'applications web diverses et variées. A l'instar de Ruby avec lequel il est développé, il est multiplateforme.

La philosophie de Ruby on Rails est on ne peut plus claire : il faut au maximum s'abstraire de contraintes trop techniques et trop éloignées de la réalité du client pour se focaliser avant tout sur les fonctionnalités. Ce modèle étant réputé très efficace, d'autres frameworks comme CakePHP s'en sont depuis clairement inspirés.

Tout ceci étant pour l'instant très flou, explorons Rails à travers quelques unes de ses fonctionnalités majeures afin de voir comment ce framework simplifie le développement par rapport à des solutions classique

Ruby on Rails pour simplifier le développement web...

Une architecture MVC imposée par Rails



Ruby on Rails implémente nativement la très répandue architecture Modèle-Vue-Contrôleur. Cela se voit dès l'arborescence d'un projet (voir ci-contre) : dans le dossier app d'une application rails apparaissent entre autres trois sous dossiers appelés respectivement models, views et controlers. Chacun de ces dossiers va donc accueillir respectivement :

- Les modèles : une description logique des entités → est-elle associée à une autre, quels attributs ?
- Les vues : des fichiers html dans lesquels on insère du code Ruby pour gérer les parties dynamiques.
- Les controleurs : un script qui contient entre autres les méthodes CRUD pour accéder aux différents éléments de la base de donnée (voir ci-dessous).

Rails et le CRUD

Pour développer une application web dynamique, il est absolument nécessaire de conserver des données relatives aux utilisateurs dans des fichiers ou des bases de donnée pour pouvoir les réutiliser ultérieurement. Pour que ces données soient exploitables, il faut impérativement qu'on puisse les créer, les lire, les mettre à jour et les supprimer. On parle de CRUD (Create, Read, Update, Delete).

Produire un CRUD n'est pas spécialement difficile mais c'est assez frustrant pour le programmeur qui doit manuellement créer la base de donnée et coder de nombreuses fonctions pour répondre à chacun de ces besoins de création, de modification... Pour pallier ce problème, Ruby on Rails propose un mécanisme simple qui permet de générer un CRUD en un temps record : le **scaffolding**, littéralement « échafaudage ». Prenons par exemple la commande suivante :

```
rails generate scaffold post title:string body:text
```

Cette commande génère le scaffold d'une entité post ayant un attribut title de type texte et un body de type texte. Cette commande génère entre autres un modèle qui décrit cette entité, un contrôleur qui donne accès à tout ses attributs, des fichiers de migration qui sont chargés de générer la table.

Par exemple, dans le cas de la commande ci-dessus, pour insérer la table post, il suffirait d'utiliser la commande `rake db:migrate`.

Certains éléments déjà intégrés...

Un avantage de Ruby on Rails est qu'il fournit d'entrée de jeu un certain nombre de

méthodes pour simplifier des tâches courantes mais non moins nécessaires comme par exemple la sécurité d'un site.

Prenons des failles de sécurité courantes comme les failles XSS (Cross-Site Scripting) qui peuvent amener un utilisateur sur des sites peu recommandables par injection de code JavaScript dans un commentaire par exemple. Comment faire pour pallier ce problème ? Ruby on Rails propose une solution clé en main. Ci-après un bout de code de vue :

```
<%= @post.title  %>
```

Ce bout de code insère dans la vue l'attribut title d'un objet @post et tout cela de manière sécurisée. En effet, dans Ruby on Rails 3, le framework va échapper les caractères à risque pour éviter l'insertion de code malicieux par défaut.

Des gems pour tout...

Les **gems** ne sont pas à proprement parler une fonctionnalité de Ruby on Rails mais une fonction du langage Ruby très utilisée avec Rails. Alors qu'est-ce qu'une Gem ? Eh bien ce sont de petits programmes ou librairies qui permettent de répondre à des besoins plus ou moins spécifiques : gestion avancée des utilisateurs, gestion de médias, échappement de balises HTML etc. Il est bon de préciser que Rails lui-même est une gem.

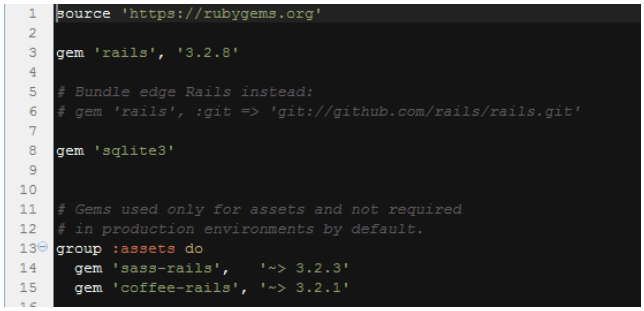
Si le concept de librairie n'a absolument rien de particulier, ce qui est extrêmement pratique avec Ruby est la gestion même des Gem grâce à un gestionnaire de paquet appelé RubyGems.

Dans le principe, cet utilitaire est très proche des gestionnaires de paquet que l'on peut retrouver sur Linux comme aptitude, apt-get, yum ou encore RPM. Il permet ainsi via un petit utilitaire en ligne de commande de lister les gems installées, de les mettre à jour, d'en installer de nouvelles... Par exemple pour installer une gem il suffit de faire :

```
gem install rails
```

Mais l'utilité des gems ne se limite pas à fournir de nouvelles fonctionnalités à l'utilisateur final. Supposons par exemple qu'on veuille migrer une application Rails sur une autre machine et que cette application utilise des gems spécifiques ? Il faudrait vérifier les dépendances de l'application et réinstaller les gems une par une, ce qui n'est pas très commode. Bundler est une gem qui répond à ce besoin.

En spécifiant toutes les dépendances du projet dans un fichier Gemfile situé à la racine de celui-ci, il suffira d'exécuter un bundle install pour gérer automatiquement toutes les dépendances de l'application, soit une seule ligne contre beaucoup plus si on le faisait à la main.



```
source 'https://rubygems.org'

gem 'rails', '3.2.8'

# Bundle edge Rails instead:
# gem 'rails', :git => 'git://github.com/rails/rails.git'

gem 'sqlite3'


# Gems used only for assets and not required
# in production environments by default.
group :assets do
  gem 'sass-rails',   '~> 3.2.3'
  gem 'coffee-rails', '~> 3.2.1'
```

Le fichier Gemfile tel qu'il est généré à la création du projet

Conclusion

Au final, Ruby on Rails se révèle être un framework particulièrement adapté au prototypage et au développement rapide dans la mesure où il fournit un ensemble d'outil clé en main aux développeurs pour éviter de « perdre » du temps.

En effet, Rails permet au développeur de ne pas se poser la question de l'architecture logicielle vu qu'il impose dès le départ un cadre strict avec le pattern MVC. Par la suite, le framework offre un certain nombre d'outils qui permettent au développeur d'économiser beaucoup de temps sur les aspects purement techniques afin de se concentrer sur des problématiques de conception plus proche de la réalité du client.

Sources :

- http://rubyonrails.org/
- http://www.ruby-lang.org/fr/
- http://www.camilleroux.com/
- http://sixrevisions.com/
- http://fr.wikipedia.org/
- http://stackoverflow.com/
- http://www.developpez.net/