Ruby on Rails

Dave Thomas
David Heinemeier Hansson

ISBN : 2-212-11746-9.

EYROLLES

1

Introduction

En déchargeant le cerveau de tout travail inutile, une bonne notation le laisse libre de se concentrer sur des problèmes plus avancés...

Alfred North Whitehead

Ruby on Rails est un framework qui rend le développement, le déploiement et la maintenance d'applications Web plus facile.

Bien sûr, c'est ce que prétendent tous les frameworks Web. Alors qu'est-ce qui rend Rails si différent ? Plusieurs réponses peuvent être apportées à cette question.

La première réside dans l'architecture de Rails. Au fil du temps, la plupart des développeurs d'applications Web conséquentes ont opté pour une architecture modèle-vue-contrôleur (MVC). L'approche MVC les aide à structurer leurs applications plus proprement (nous couvrirons MVC dans le chapitre suivant). Les frameworks Java comme Tapestry et Struts sont basés sur MVC. Rails fait aussi partie de cette famille. Quand vous développez en Rails, chaque bout de code a sa place, et tous les morceaux de votre application interagissent de façon standardisée. C'est comme si vous commenciez un projet d'application avec un canevas déjà en place.

Une autre façon de répondre à la question est de considérer le langage de programmation. Les applications Rails sont écrites en Ruby, langage de scripting moderne et orienté objet. Ruby est concis sans être abscons et permet d'exprimer des idées de façon claire et naturelle. Ceci

rend les programmes faciles à écrire et, de façon toute aussi importante, faciles à lire plusieurs mois après.

Ruby s'inspire d'un style de programmation familier aux programmeurs Lisp, mais qui paraîtra plutôt exotique aux autres[1]. Le langage permet de créer facilement des méthodes qui se comportent quasiment comme des extensions de la syntaxe. Certains appellent cela la « métaprogrammation » ; nous dirons simplement que c'est utile. La métaprogrammation rend en effet les programmes plus compacts et plus lisibles. Cela permet aussi d'effectuer dans le code lui-même certaines tâches qui sont normalement rejetées dans des fichiers de configuration externes. Il est ainsi plus facile de voir ce qui se passe dans l'application. L'exemple de code qui suit définit la classe modèle pour un projet. Ne vous souciez pas des détails pour le moment. Pensez simplement à la quantité d'information exprimée à la lecture de ces quelques lignes de code.

```
class Project < ActiveRecord::Base
  belongs_to              :portfolio
  has_one                 :project_manager
  has_many                :milestones
  has_and_belongs_to_many :categories
  validates_presence_of   :name, :description
  validates_acceptance_of :non_disclosure_agreement
  validates_uniqueness_of :key
end
```

Enfin, on peut aussi considérer la différence de Rails sous l'angle philosophique. Quelques concepts clés ont présidé à la conception de Rails : *DRY* et *convention plutôt que configuration*.

DRY signifie Don't Repeat Yourself[2] : chaque élément de connaissance d'un système ne devrait être exprimé qu'à un seul endroit. Rails utilise la puissance de Ruby pour donner vie à ce principe. Vous ne trouverez que très peu de duplication dans une application Rails ; vous dites ce que vous avez besoin de dire à un seul endroit – un endroit souvent suggéré par les conventions de l'architecture MVC – puis vous passez à la suite.

Convention plutôt que configuration est un principe capital lui aussi. Il signifie que Rails adopte des comportements par défaut pour à peu près tous les aspects de votre application. En suivant ces conventions vous pouvez écrire une application Rails avec moins de code que n'en utilise une application Web Java typique dans ses fichiers de configuration XML. Et rassurez-vous : si vous avez besoin de passer outre ces conventions, la chose est facile aussi.

1. NDT : le traducteur de ce livre ayant lui-même un long passé de programmeur C, C++, Perl et un peu moins de Lisp... se doit d'ajouter que cette assertion est un peu exagérée. Ne vous laissez pas impressionner, Ruby est vraiment très facile à apprendre.
2. NDT : qu'on pourrait traduire par "Ne vous répétez pas".

Nous pourrions aussi mentionner toutes les fonctionnalités alléchantes que propose Rails comme le support de services Web intégrés, la réception d'e-mails entrants, AJAX (pour des applications Web hautement interactives), un framework complet de test unitaire (y compris le support transparent pour les *mock objects*) et enfin, des environnements indépendants pour les phases de développement, de test et de mise en production. Nous pourrions également parler des générateurs de code livrés avec Rails (ou d'autres qui sont disponibles sur Internet), qui permettent de créer des canevas de code Ruby, en vous laissant le soin de vous concentrer sur la logique de l'application.

Enfin, Rails est différent de par ses origines : il a « émergé » d'une application commerciale réelle. Il s'avère que le meilleur moyen de créer un framework consiste à trouver les thèmes centraux dans une application spécifique et, ensuite, de les encapsuler à l'aide d'une couche de code générique apte à servir de fondation. C'est pour cette raison que lorsque vous développez une application Rails, vous démarrez sur de bonnes bases.

Mais il y a encore autre chose à propos de Rails, quelque chose qu'il est difficile de décrire. D'une certaine façon, Rails paraît tout simplement bien. Bien sûr, vous devez nous croire sur parole jusqu'à ce que vous ayez écrit une application Rails par vous-même, ce qui devrait être fait dans les 45 prochaines minutes... Et nous aurons ensuite tout le reste du livre pour finir de vous convaincre.

Les 10 bonnes raisons d'aimer Rails selon Dave

1. Il apporte de l'agilité dans le développement Web.
2. Je peux créer des pages avec des effets visuels sympathiques, comme le font les petits jeunes...
3. Il me permet de me concentrer sur mon application et non pas de « nourrir » le framework.
4. Mes applications restent maintenables en grossissant.
5. Il me permet de dire « oui » à mes clients plus fréquemment.
6. Le test est intégré (et facile), donc il est utilisé.
7. Retour instantané : éditer le code, appuyer sur Rafraîchir, et le changement apparaît aussitôt dans mon navigateur.
8. La métaprogrammation signifie que je peux programmer à un niveau vraiment haut.
9. Les générateurs de code me permettent de démarrer rapidement.
10. Pas de XML !

Rails est agile

Le titre de l'édition américaine de ce livre est *Agile Web Development with Rails*. Vous êtes peut-être donc surpris de découvrir que nous n'avons pas de sections spécifiques sur l'application de la méthode agile X, Y ou Z à la programmation en Rails.

La raison de cette absence est à la fois simple et subtile : l'agilité est partie intégrante de Rails.

Attardons-nous un moment sur les valeurs exprimées dans le manifeste du développement agile[1] :

- les individus et l'interaction plutôt que les procédures et les outils ;
- du logiciel en ordre de marche plutôt qu'une documentation exhaustive ;
- la collaboration avec le client plutôt que la négociation de contrat ;
- répondre aux demandes de changement plutôt que de suivre un plan.

Rails est avant tout une histoire d'individus et d'interactions. Ici pas d'outils lourds, pas de configuration complexe et pas de procédures élaborées. Il n'y a que des petits groupes de développeurs, leur éditeur favori et des bouts de code Ruby. Cela amène une grande transparence ; ce que fait le développeur se reflète immédiatement dans ce que le client voit. Il s'agit là, intrinsèquement, d'un processus interactif.

Rails ne rejette pas la documentation. Au contraire, Rails permet de créer très facilement de la documentation HTML pour la totalité de votre code. Mais le processus de développement de Rails n'est pas dirigé par la documentation. Vous ne trouverez pas un document de spécification de 500 pages au cœur d'un projet Rails. Vous y verrez plutôt un groupe composé d'utilisateurs et de développeurs explorant ensemble une série de besoins et les différentes façons de les satisfaire. Vous y trouverez des solutions qui évoluent au fur et à mesure que les développeurs et les utilisateurs progressent dans leur compréhension du problème à résoudre. Vous y découvrirez aussi un framework qui produit du code opérationnel très tôt dans le cycle de développement. Dans sa première version, votre application sera peut-être un peu brute de fonderie mais elle permettra aux utilisateurs de se faire une première idée de ce qui leur sera délivré.

Par ce biais, Rails encourage la collaboration avec le client. En effet, quand celui-ci voit à quelle vitesse un projet Rails peut répondre aux demandes de modifications, il prend rapidement conscience que l'équipe saura délivrer ce qui doit l'être et non pas uniquement ce qui leur a été demandé. Les habituelles séances de confrontation avec les utilisateurs sont remplacées par des sessions d'exploration des différents scénarios possibles ; ce qu'on appelle les sessions « Et si... ? ».

L'idée centrale de Rails est qu'il faut être capable de répondre au changement. La façon acharnée, presque obsessionnelle, dont Rails se conforme au principe DRY signifie que les

1. http://agilemanifesto.org/. Dave Thomas est l'un des 17 auteurs de ce document.

changements apportés à une application Rails concernent beaucoup moins de code que dans un autre framework Web. Et puisque les applications Rails sont écrites en Ruby, langage où les concepts sont exprimés de façon précise et concise, les modifications ont tendance à être très localisées et faciles à écrire. Le fort accent mis sur les tests unitaires et fonctionnels ainsi que le support des garnitures de test (*fixtures*) et des objets factices (*mock objects*) donnent aux développeurs le filet de sécurité nécessaire lorsqu'ils opèrent des changements dans le code. Avec un bon jeu de tests en place, les changements sont moins éprouvants pour les nerfs.

Plutôt que de chercher à relier constamment le processus de développement Rails aux principes du développement agile, nous avons décidé de laisser le framework parler de lui-même. Lorsque vous lirez le didacticiel, essayez de vous imaginer personnellement en train de développer une application de cette façon : vous travaillez littéralement aux côtés de vos clients et vous établissez ensemble les priorités et les solutions à apporter aux problèmes posés. Ensuite, lorsque vous lirez la documentation de référence de Rails dans la seconde moitié de l'ouvrage, vous verrez à quel point la structure sous-jacente de Rails vous permet de satisfaire les besoins de vos clients plus rapidement et sans cérémonie.

Un dernier point à propos de l'agilité et de Rails : bien que ça ne fasse pas très professionnel de le mentionner, il faut aussi souligner que programmer en Rails est réellement excitant.

Se repérer dans le livre

Ce livre s'est finalement avéré un peu plus volumineux que prévu. Rétrospectivement, il est clair que, poussés par notre enthousiasme, nous avons en fait écrit deux livres en un : un didacticiel et un guide de référence détaillé de Rails.

Les deux premières parties de ce livre proposent une introduction aux concepts de Rails suivi d'un exemple approfondi d'une application : une boutique en ligne. C'est par là qu'il vous faudra démarrer si vous cherchez à vous faire une idée de ce qu'est la programmation en Rails. En fait la plupart des lecteurs semblent apprécier de construire l'application au fur et à mesure de leur lecture. Si vous ne souhaitez pas taper tout le code de l'application, vous pouvez « tricher » et télécharger le code source[1].

La troisième partie de l'ouvrage, qui débute page 193, est une étude détaillée de l'ensemble des fonctions et facilités de Rails. Cette partie vous sera utile pour découvrir comment utiliser l'ensemble des composants de Rails et comment déployer vos applications Rails de façon efficace et sûre.

Au fil de votre lecture, vous rencontrerez un certain nombre de conventions adoptées lors de la rédaction de cet ouvrage.

1. Le code source est disponible sur le site http://www.editions-eyrolles.com.

Extrait de code source

La plupart des extraits de code figurant dans le texte proviennent d'exemples complets et fonctionnels que vous pouvez télécharger. Pour vous guider, et si le code est disponible en téléchargement, vous verrez apparaître un marqueur dans la marge (exactement comme représenté ci-dessous).

Fichier 1.1

```
class SayController < ApplicationController
end
```

Reportez-vous à la table d'index qui débute page 581, cherchez le numéro correspondant, et vous trouverez en vis-à-vis le nom du fichier contenant l'extrait de code en question. Si vous lisez la version PDF de cet ouvrage et si votre visualiseur PDF supporte les hyperliens, vous pourrez cliquer sur le numéro dans la marge et le code apparaîtra automatiquement dans la fenêtre de votre navigateur Web. Certains navigateurs (tels que Safari sur Mac OS X) essaient à tort d'interpréter certains extraits de code comme du HTML. Si cela vous arrive, demandez à votre navigateur de vous montrer le code source de la page pour voir le code source réel.

Astuces Ruby

Bien que vous deviez connaître Ruby pour écrire une application Rails, nous sommes conscients que nombre de nos lecteurs vont apprendre Ruby et Rails en même temps. C'est pourquoi l'annexe A est une (très) brève introduction au langage Ruby qui se trouve en annexe de cet ouvrage. Lorsque nous utiliserons une instruction spécifique à Ruby pour la première fois, nous indiquerons un pointeur vers l'annexe. Par exemple, ce paragraphe contient un usage totalement gratuit de `:name`, qui est un symbole Ruby. Dans la marge, figure l'indication que les symboles Ruby sont traités en page 515. Si vous ne connaissez pas Ruby ou si vous avez besoin d'une petite réactualisation, vous pouvez choisir de lire l'annexe A avant d'aller plus loin. Il y a beaucoup de code dans ce livre...

Cet ouvrage n'est pas un manuel de référence Rails. Nous passons en revue la plupart des modules et leurs méthodes, soit au travers d'exemples soit dans le texte du livre, mais nous n'avons pas inclus les centaines de pages des interfaces de programmation (API). Il y a une bonne raison à cela : vous disposez de cette documentation lorsque vous installez Rails et vous êtes assuré qu'elle est plus à jour que ne pourrait l'être le contenu de ce livre. Si vous installez Rails en utilisant RubyGems (ce que nous recommandons vivement), démarrez simplement le serveur de documentation Gem (en utilisant la commande `gem_server`) et vous pourrez accéder à toutes les interfaces de programmation Rails en pointant votre navigateur Web sur `http://localhost:8808`.

Les versions de Rails

La version 1.0 de Rails, sortie en décembre 2005, a apporté certaines modifications par rapport à ce que décrit l'édition américaine du livre, basée sur Rails 0.13. Étant donné que l'édition française est parue après la publication de Rails 1.0, le traducteur et les relecteurs en ont profité pour mettre à jour le contenu de l'ouvrage. En conséquence, la version française de l'ouvrage n'est pas totalement identique à la version anglaise d'origine. Les interfaces de programmation décrites dans ce livre sont bien celles de Rails 1.0 et le code fourni en exemple a été testé avec la version 1.0 de Rails.

Remerciements

La rédaction de ce livre s'est avérée être une tâche de grande ampleur. Elle n'aurait jamais pu être menée à bien sans l'aide considérable apportée par les communautés Ruby et Rails. Il est difficile de mentionner tous ceux qui ont contribué, donc si votre nom n'apparaît pas dans la liste qui suit, soyez assuré qu'il ne s'agit que d'un oubli de notre part.

Ce livre a aussi bénéficié d'un groupe de relecteurs qui a généré plus de 6 méga-octets de commentaires. Nous les remercions donc de tout cœur :

> Alan Francis, Amy Hoy, Andreas Schwarz, Ben Galbraith, Bill Katz, Carl Dearmin, Chad Fowler, Curt Micol, David Rupp, David Vincelli, Dion Almaer, Duane Johnson, Erik Hatcher, Glenn Vanderburg, Gunther Schmidl, Henri ter Steeg, James Duncan Davidson, Johannes Brodwall, John Harechmak, John Johnson, Justin Forder, Justin Gehtland, Kim Shrier, Krishna Dole, Leon Breedt, Marcel Molina Jr., Michael Koziarski, Mike Clark, Miles K. Forrest, Raymond Brigleb, Robert Rasmussen, Ryan Lowe, Sam Stephenson, Scott Barron, Stefan Arentz, Steven Baker, Stian Grytøyr, Tait Stevens, Thomas Fuchs, Tom Moertel, and Will Schenk.
>
> Rails lui-même évoluait pendant la rédaction de cet ouvrage. En conséquence, les développeurs de Rails ont passé de nombreuses heures à répondre aux questions de l'auteur et aussi à sympathiser (ils ont aussi passé de non moins nombreuses heures à tourmenter l'auteur pour lui faire changer quelque chose qui venait juste d'être documenté, mais nous n'en parlerons pas davantage ici...). Un *grand merci* donc à : Jamis Buck (minam), Jeremy Kemper (bitsweat), Marcel Molina Jr, (noradio), Nicholas Seckar (Ulysses), Sam Stephenson (sam), Scott Barron (htonl), Thomas Fuchs (madrobby), and Tobias Lütke (xal).

Justin Forder a fait un travail remarquable en améliorant et en corrigeant les feuilles de style anémiques initialement développées par l'auteur pour l'application Dépôt.

Des milliers de personnes ont participé au programme de béta-testeurs de cet ouvrage. Merci d'avoir pris ce risque. Des centaines d'entre eux ont pris le temps de nous faire parvenir leurs commentaires et leurs correctifs suite à leur lecture. Le livre s'en est trouvé largement amélioré.

Enfin nous souhaitons remercier les personnes qui ont rédigé les chapitres plus spécialisés de ce livre : Leon Breedt, Mike Clark, Thomas Fuchs et Andreas Schwarz.

De Dave Thomas

Ma famille ne m'a pas vu durant les huit derniers mois. Je leur suis infiniment reconnaissant de leur patience, de leur soutien et de leur amour. Merci Juliet, Zachary et Henry.

De David Heinemeier Hansson

Marianne : pour sa patience devant les interminables nuits passées à hacker Rails.