Ruby on Rails

2e édition

Dave Thomas

David Heinemeier Hansson

Avec la collaboration de Leon Breedt, Mike Clark,
James Duncan Davidson, Justin Gehtland et Andreas Schwarz

ISBN : 978-2-212-12079-0

EYROLLES

1

Introduction

Ruby on Rails est un framework qui rend le développement, le déploiement et la maintenance d'applications Web plus faciles. Pendant les douze mois qui ont suivi la sortie de sa première version, Rails est passé du statut de jouet inconnu à celui de phénomène mondial. Il a remporté plusieurs récompenses et, plus important encore, il est devenu le framework de choix pour l'implémentation de toute une gamme d'applications dites *Web 2.0*. Rails n'est pas uniquement « tendance » dans le monde des hackers purs et durs : de nombreux grands comptes utilisent aujourd'hui Rails pour créer leurs applications Web.

Il existe plusieurs raisons à cet engouement. Tout d'abord, il est très vite apparu qu'un nombre important de développeurs étaient véritablement frustrés par les outils qu'ils utilisaient pour créer leurs applications Web. Qu'ils utilisent Java, PHP ou .NET, le sentiment général semblait être que les choses étaient véritablement trop compliquées. Et puis, tout à coup, Rails est arrivé et sa simplicité a immédiatement séduit.

Par exemple, toutes les applications Rails sont implémentées autour de l'architecture modèle-vue-contrôleur (MVC). Les développeurs Java sont familiers des frameworks tels que Tapestry ou Struts qui sont aussi basés sur MVC. Mais Rails va au-delà de MVC : lorsque vous développez, chaque bout de code a une place bien définie et tous les morceaux de votre application interagissent de façon standardisée. C'est comme si vous débutiez un projet avec un canevas déjà en place.

Comme chacun sait, les programmeurs professionnels écrivent des tests. Et là encore, Rails tient ses promesses. Toutes les applications Rails offrent directement dans leur structure tout le support nécessaire aux tests. Au fur et à mesure de l'ajout de nouvelles fonctionnalités dans le code, Rails crée automatiquement des squelettes de tests. Le framework facilite grandement le test d'applications et, de fait, les applications Rails ont une nette tendance à être testées.

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.

Rails tire pleinement parti de Ruby en l'étendant de façon très novatrice, facilitant ainsi le travail du développeur qui produit alors des programmes plus compacts et plus lisibles. Cela permet aussi d'effectuer dans le code lui-même certaines tâches qui sont habituellement rejetées dans des fichiers de configuration externes. Il est ainsi plus aisé de voir ce qui se passe dans l'application. L'exemple de code qui suit définit la classe de modèle pour un projet. Ne vous souciez pas des détails pour le moment. Pensez simplement à la quantité d'informations exprimée par ces quelques lignes de code.

```
class Project < ActiveRecord::Base
  belongs_to              :portfolio
  has_one                 :project_manager
  has_many                :milestones
  has_many                :deliverables, :through => :milestones
  validates_presence_of   :name, :description
  validates_acceptance_of :non_disclosure_agreement
  validates_uniqueness_of :short_name
end
```

Les développeurs découvrent également que Rails s'appuie sur une base philosophique forte. Quelques concepts clés ont présidé à la création de Rails : *DRY* et *convention plutôt que configuration*. *DRY* signifie *Don't Repeat Yourself*[1] : 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 – ce dernier est souvent suggéré par les conventions de l'architecture MVC – puis vous passez à la suite. Pour les développeurs habitués à d'autres frameworks où le moindre changement dans le schéma de la base de données peut se traduire par plus d'une demi-douzaine de modifications dans le code, Rails a constitué une véritable révélation.

Convention plutôt que configuration est lui aussi un principe fondamental. Il signifie que Rails adopte des comportements par défaut pour presque 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 en avez besoin, il est aisé de passer outre ces conventions.

Les développeurs qui viennent à Rails y trouvent aussi autre chose. Rails est nouveau et l'équipe de développement comprend parfaitement le Web nouvelle génération. Rails ne passe pas son temps à courir après les nouveaux standards de fait du Web : il aide à leur défi-

1. NDT : qu'on pourrait traduire par « Ne vous répétez pas ».

nition. C'est ainsi que Rails permet aux développeurs d'intégrer facilement dans leur code des technologies comme Ajax ou des interfaces conformes au standard REST car tout est déjà là (si vous n'êtes pas familier avec Ajax ou REST, ne craignez rien, nous y viendrons en temps voulu).

Les développeurs se soucient également du déploiement de leurs applications. Ils se rendent compte qu'avec Rails on peut déployer les versions successives d'une application sur un nombre quelconque de serveurs en une ligne de commande (et revenir en arrière tout aussi aisément si la nouvelle version se révèle moins parfaite que prévue).

Rails est aussi 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 est un environnement dans lequel on se sent 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.

Rails est agile

Le titre de l'édition américaine de ce livre est *Agile Web Development with Rails.* Vous êtes donc peut-être 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 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 de petits groupes de développeurs, leur éditeur de texte 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.

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

Rails ne rejette pas la documentation. Au contraire, il 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 spécifications 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. L'acharnement, presque obsessionnel, avec lequel Rails se conforme au principe DRY signifie que les 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 (*fixtures*) et des souches (*stubs*) de tests 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 qu'il ne soit pas très professionnel de le mentionner, il faut aussi souligner que programmer en Rails est réellement excitant.

Se repérer dans le livre

Les deux premières parties de ce livre proposent une introduction aux concepts de Rails suivie d'un exemple approfondi d'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 intitulée *Le framework Rails* est une étude détaillée de l'ensemble des fonctions et outillages de Rails. Elle vous permettra de découvrir comment utiliser les différents 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.

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 au-dessus du bloc de code le chemin permettant d'accéder au fichier correspondant, une fois le code source téléchargé et décompressé sur votre disque dur (exactement comme représenté ci-après).

```
code/work/demo1/app/controllers/say_controller.rb
class SayController < ApplicationController
  def hello
  end
end
```

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. Lorsque nous utiliserons une instruction spécifique à Ruby pour la première fois, nous indiquerons un pointeur vers cette annexe. Si vous découvrez 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

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

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` (l'encadré *Créer votre propre documentation des API Rails*, page 30, décrit une autre façon d'installer la documentation complète de l'API).

Les versions de Rails

Ce livre couvre Rails version 1.2 qui est disponible depuis janvier 2007.

Si ce n'est pas la version installée sur votre machine, vous devez la mettre à jour avant de tester les exemples de code de cet ouvrage.

Remerciements

On pourrait croire que rédiger la seconde édition d'un ouvrage est chose aisée. Après tout, le texte est déjà écrit et il suffit juste d'ajuster les exemples de code par-ci, de reformuler le texte de façon mineure par-là et le tour est joué. On pourrait le croire...

Il est difficile de dire pourquoi, mais j'ai l'impression qu'écrire la seconde édition m'a demandé à peu près autant d'efforts que pour la première[1]. Pendant la rédaction, Rails était en constante évolution et le livre devait suivre. Certaines parties de l'application Dépôt ont dû être réécrites trois ou quatre fois et le texte d'explication a dû être mis à jour. L'importance donnée à REST ajoutée au mécanisme naturel d'obsolescence ont changé la structure du livre, des aspects auparavant essentiels devenant juste accessoires.

Ainsi, cet ouvrage n'existerait pas sans l'aide massive des communautés Ruby et Rails. Comme pour la première édition, cet ouvrage a d'abord été publié en version bêta : des versions préliminaires ont été publiées en PDF et les lecteurs pouvaient faire des commentaires en ligne. Et ils ne s'en sont pas privés : plus de 1 200 suggestions et rapports d'anomalies ont été postés. La grande majorité ont été insérés dans l'ouvrage, rendant ce livre incommensurablement plus utile. Merci à tous à la fois pour votre support du programme *beta book* des éditions Pragmatic Programmers et pour vos précieuses contributions.

Encore une fois, l'équipe de développement de Rails a été d'un très grand secours, répondant aux questions, vérifiant les exemples de code et corrigeant les bogues. Un grand merci à :

Scott Barron (htonl), Jamis Buck (minam), Thomas Fuchs (madrobby), Jeremy Kemper (bitsweat), Michael Koziarski (nzkoz), Marcel Molina Jr, (noradio), Rick Olson (technoweenie), Nicholas Seckar (Ulysses), Sam Stephenson (sam), Tobias Lütke (xal) et Florian Weber (csshsh).

1. NDT: c'est aussi l'avis des traducteurs ☺

J'aimerais aussi remercier ceux qui ont écrit certains chapitres très spécialisés de l'ouvrage : Leon Breedt, Mike Clark, James Duncan Davidson, Justin Gehtland et Andreas Schwarz.

À chaque fois que j'écris un ouvrage, je me promets que ce sera le dernier, ne serait-ce que parce que cela m'éloigne de ma famille pendant des mois. Une fois de plus : Juliet, Zachary et Henry, merci pour tout.