Ruby on Rails

Une (courte) introduction

Olivier Gutknecht

(Ruby on) Rails ?

- Un *framework* pour les applications Web
- Écrit avec et grâce à Ruby
- Inventé par **D**avid **H**einemeier **H**ansson
- Souple, intégré, dynamique, élégant, productif... agile

Ruby (on Rails) ?

Un langage de script objet, très dynamique, très souple, avec une syntaxe élégante, qui reprend les bonnes idées de Smalltalk, Lisp, ou Perl

Quelques petites indications sur la syntaxe de Ruby, suffisantes pour ce qui va suivre

Le rajout d'une fonctionnalité tracer *(définie autre part) dans cette classe.* :tout *indique un symbole (à voir un peu comme une chaîne constante)*

On crée un objet de classe Bonjour. @ *désigne une variable d'instance (nul besoin de les déclarer avant)*

Envoie le message saluer *à l'objet* @message *(avec le paramètre* a*). Envoyer un message sur un objet ou sur une classe utilise la même syntaxe.*

Ruby étant un langage très dynamique, on a souvent tendance à parler d'envoi de message plutôt que d'appel de méthode.

```
class Essai < AutreClasse
  tracer :tout

  def exemple
    @message = Bonjour.new
    amis = Copains.liste

    for a in amis
      @message.saluer(a)
    end
  end
end
```

Définition d'une nouvelle classe Essai *héritant de la classe* AutreClasse

Définition d'une nouvelle méthode exemple

Quand un appel de méthode n'a pas de paramètre, on omet d'habitude le ()

amis *et* a *sont des variables locales (nul besoin de les déclarer avant usage)*

Cette introduction ?

- Un coup d'œil général
- Les 3 piliers de Rails
  - Le **Modèle :** ActiveRecord
  - La **Vue :** ActionView
  - Le **Contrôleur :** ActionController
- Développer avec Rails

Pourquoi Rails ?

Prendre le meilleur des deux grandes façons de faire des applications Web

Interactif, rapide, facile à apprendre, peu de contraintes, très souple

Peu structuré, dur à maintenir, pas de séparation entre logique et rendu

Pour caricaturer: le monde PHP

Rails concilie souplesse de développement et attention portée à l'architecture des applications

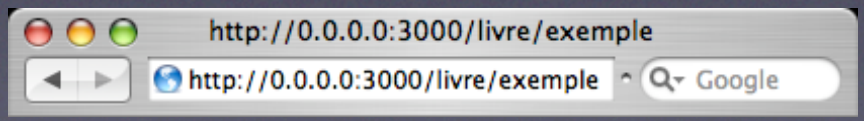
Très structuré, cadre les architectures, infrastructures puissantes

Complexe, lourd, long à maîtriser, difficile à adapter, multiples couches

Pour caricaturer: le monde J2EE

Un petit exemple (fonctionnel) pour commencer

Afficher tous les livres d'une bibliothèque stockés dans une base de données quand l'utilisateur navigue sur...



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

id	titre	auteur	isbn
1	L'usage du monde	Nicolas Bouvier	222889401
2	Le seigneur des porcheries	Tristan Egolf	2070414736

```
class LivreController
        < ApplicationController
  def exemple
    @etagere = Livre.find(:all)
  end
end
```

```
<html><body>
 <h1>Tous les livres...</h1>
  <% for l in @etagere %>
    <%= l.titre %>: <%= l.auteur %><br>
  <% end %>
</body></html>
```

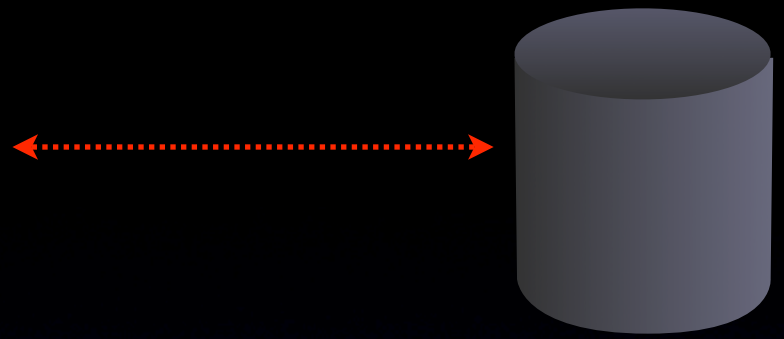
3 fichiers, 12 lignes de code, réalisé sans trucage

(4 lignes de plus si l'on compte le fichier configurant l'accès à la base de données)

Modèle(s)

(des fichiers contenant des classes Ruby)

Représente les données de l'application, masque le dialogue avec la base



Contrôleur(s)

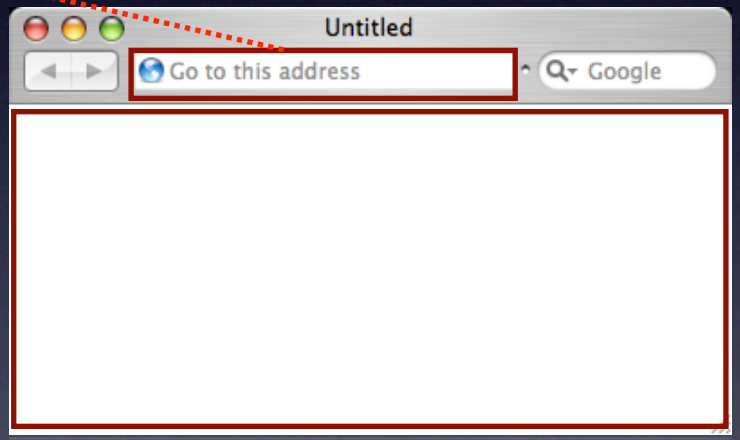
(des fichiers contenant des classes Ruby)

Gére la logique de la navigation, choisit quelles informations transmettre à la vue

Vue(s)

(des fichiers 'rhtml' : des pages HTML avec du code Ruby)

Permet d'élaborer des modèles de pages web et d'y inclure les données reçues



MVC : un modèle classique...

... pour les applications clientes, mais assez rarement mis en œuvre d'une manière aussi élégante dans les systèmes de développement Web

Modèle(s)

"ActiveRecord"

+

Contrôleur(s)

"ActionController"

+

Vue(s)

"ActionView"

+

Outils supplémentaires
WebServices, Mail, Intégration AJAX...

=

Ruby on Rails

Définir le **Modèle**

“ActiveRecord”

"ActiveRecord"

livres

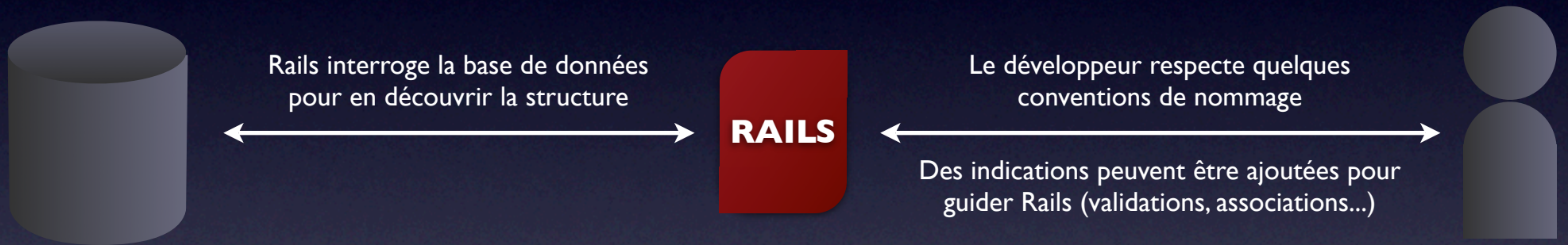
id	titre	auteur	isbn
1	L'usage du monde	Nicolas Bouvier	222889401
2	Le seigneur des porcheries	Tristan Egolf	2070414736

ActiveRecord est un *"ORM" (Object/Relational Mapper)* : il assure le lien entre le monde objet de Ruby et le monde relationnel de la base de données

Cela couvre à la fois la **structure** (comment représenter une table en une classe ?) et le **comportement** : les problèmes "non triviaux" de la gestion des caches, la validation des données, l'optimisation des requêtes, la portabilité entre plusieurs moteurs de base de données...

Où est l'information ?

Objectif : ne jamais avoir besoin de dupliquer la même information à deux endroits différents (dans la structure de la base et dans le code)



Rails essaie de naviguer entre deux écueils classiques

Les moteurs Web où la structure des données n'est jamais explicite et se retrouve disséminée dans tout le code de l'application

Les moteurs Web où la structure des données de la base doit être réexplicitée dans des fichiers de configuration du moteur

Exemple

id	titre	auteur	isbn
...	...	...	...

livres

SQL

```
CREATE TABLE livres (
 id int(11) NOT NULL auto_increment,
 titre varchar(255),
 auteur varchar(255),
 isbn varchar(13),
 PRIMARY KEY (id));
```

Rails

```
class Livre < ActiveRecord::Base


end
```

La classe de base de tous les modèles ActiveRecord

Nul besoin de se répéter

Rails interrogera la base pour obtenir la structure de la table et enrichir la classe.

Tirer parti de conventions

Une colonne *"id"* sera supposée être l'index principal.
Une classe *"Livre"* correspondra à la table *"Livres"* (au pluriel).
Les colonnes *"titre"*, *"auteur"*, *"isbn"* deviendront des attributs de la classe.

Sous le capot

Ruby

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

```
a = Livre.new
a.titre = “L’usage du monde”
a.auteur = “Nicolas Bouvier”
a.isbn = “222889401X”
a.save
```

SQL généré

```
INSERT INTO livres ("isbn", "auteur",
"titre") VALUES('222889401X', 'Nicolas
Bouvier', 'L''usage du monde')
```

```
b.find(1)
puts b.auteur
```

Affichera “Nicolas Bouvier”

```
SELECT * FROM livres WHERE (livres.id
= '1') LIMIT 1
```

```
b.destroy
```

```
DELETE FROM livres WHERE id = 1
```

Rien de magique

Les méthodes *save* ou *find* ne font que masquer le dialogue avec la base de données.

On peut aussi utiliser des méthodes *find_by_auteur* ou *find_by_titre*. Rails reconnaîtra les formes en *find_by_xxx* et les traduira en requêtes sur le champ correspondant : le dynamisme de Ruby en action.

Associations

```
class Usager < ActiveRecord::Base
  has_many :emprunts
end
```

```
class Livre < ActiveRecord::Base
   has_many :emprunts
end
```

```
class Emprunt < ActiveRecord::Base
   belongs_to :usager
   belongs_to :livre
end
```

Indicateurs et conventions

Ces marqueurs permettent de clarifier un modèle complexe, de donner à Rails de quoi construire et optimiser les requêtes SQL sous-jacentes, en supposant quelques conventions d'écriture (ici *usager*_id et *livre*_id pour représenter les associations dans la table).

Sous le capot (bis)

Rails

```
u = Usager.find_by_nom("Olivier Gutknecht")

u.emprunts[0].sortie
u.emprunts[0].retour
```

SQL généré

```
SELECT * FROM usagers WHERE (usagers."nom" = 'Olivier Gutknecht' )
SELECT * FROM emprunts WHERE (emprunts.usager_id = 1)
 (rien)
```

```
u = Usager.find_by_nom("Enoch Root",
                       :include => :emprunts)

u.emprunts[0].sortie
```

```
SELECT usagers.id, usagers.nom, usagers.adresse,
emprunts.id, emprunts.sortie, emprunts.retour,
emprunts.livre_id, emprunts.usager_id FROM usagers
LEFT OUTER JOIN emprunts ON emprunts.usager_id =
usagers.id WHERE (usagers.nom = 'Enoch Root')
(rien)
```

À Rails la tâche d'optimiser les requêtes, cacher les résultats

Au développeur la responsabilité de donner suffisamment d'indications sur l'usage voulu (par exemple ici le *:include*) pour guider Rails

Oui, mais si...

... ma base existait déjà ?

... et toutes mes tables ont un préfixe ?

... et mes index ont un nom spécifique ?

... je dois faire une requête bien particulière ?

lib2_book

bookID	titre	auteur	isbn
...	...	...	...

```
class Livre < ActiveRecord::Base
   set_primary_key "bookID"
end
```

```
ActiveRecord::Base.table_name_prefix = "lib2_"
ActiveRecord::Base.pluralize_table_names = false
```

```
Livre.find_by_sql("SELECT * FROM livres WHERE auteur = 'Nicolas Bouvier'")
```

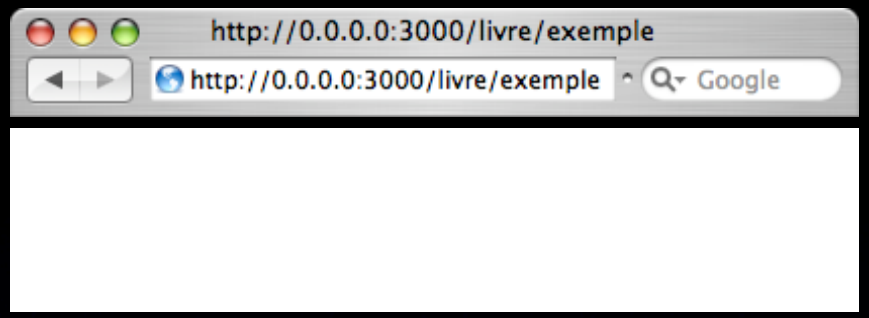
Convention n'est pas interdiction !

Il reste possible de préciser des comportements différents table par table, globalement, de revenir à des requêtes construites à la main, sans perdre les autres avantages de Rails.

Les conventions sont là pour faciliter le cas général, et non rendre impossible le cas particulier.

Mettre en place les **Contrôleurs**

“ActionController”



Serveur Web

Rails - Aiguillage Initial

L'URL est 'disséquée' pour trouver le contrôleur et sa méthode, ainsi que la vue qui en fera le rendu final

views/**livre**/**exemple**.rhtml

```
<html>
  <body>
    <%= @info ... >
  </body>
</html>
```

controllers/**livre_controller**.rb

```
class LivreController < ApplicationController
  def exemple
    @info = ...
  end
end
```

Encore des conventions !

Cette façon d'aiguiller une requête permet de simplifier le travail du concepteur de l'application : il y a juste à respecter des règles simples de nommage de fichiers et de méthodes.

Là encore, il est possible de passer outre ces conventions, moyennant un peu de code supplémentaire.

http://exemple.com/**usager**/**liste**

http://exemple.com/**usager**/**fiche**/**4212**


```
class UsagerController < ApplicationController
    def liste
        @usagers = Usager.find_all
    end

    def fiche
        @usager = Usager.find(@params["id"])
    end
end
```

app/controllers/**usager_controller**.rb

Paramètres

Une fois extraits le contrôleur et l'action, le reste de l'URL - si présent - sera vu comme des paramètres supplémentaires stockés dans un dictionnaire et prêt à être utilisés par le contrôleur

On peut ainsi exploiter de la même manière des requêtes complexes *POST* ou *GET* (de la forme *?a=b)*

Que trouver dans un contrôleur ?

- La logique de l'application...
  - ... préparer des données pour une page, et le faire différemment pour un utilisateur identifié ou anonyme
  - ... recevoir les résultats d'un formulaire HTML pour enregistrer des données, et immédiatement rediriger vers une autre page
  - ... vérifier qu'un utilisateur est identifié et si non, le rediriger vers la page idoine

Une navigation

http://exemple.com/**usager**/**liste**

Contrôleur: **usager** Action: **liste**

obtention de la liste des utilisateurs

Vue: **usager/liste.rhtml**

Usagers

Olivier Gutknecht [Fiche] [Nouvel Emprunt]
Enoch Root [Fiche] [Nouvel Emprunt]

http://exemple.com/**emprunt**/**nouveau**/**2**

Contrôleur: **emprunt** Action: **nouveau**
Paramètre: **2** (ID usager)

Obtient les informations sur l'utilisateur et ses emprunts courants, vérifie s'il a le droit d'emprunter

Vue: **emprunt/nouveau.rhtml**

Nouvel emprunt
Enoch Root
2 livres empruntés
Snow Crash - le 12/02/45 (retard)
Quicksilver - le 02/05/06

Nouvel emprunt: Ubik OK

http://exemple.com/**emprunt**/**creer**/**...**

Contrôleur: **emprunt** Action: **creer**
Paramètres: **Ubik** (titre), **2** (ID usager)

Récupère les informations de formulaire, vérifie la validité (obtient l'objet livre), instancie et enregistre un objet emprunt. Redirige ensuite vers un autre contrôleur

http://exemple.com/**usager**/**fiche**/**2**

Contrôleur: **usager** Action: **fiche**
Paramètre: **2** (ID usager)

Obtient les informations sur l'utilisateur et ses emprunts courants

Vue: **usager/fiche.rhtml**

Enoch Root - 12, Bletchley Park [Editer]
3 livres empruntés
Snow Crash - le 12/02/45 (retard)
Quicksilver - le 02/05/06
Ubik - le 05/08/06

[Sommaire] [Emprunt] [Usagers] [Livres]

L'enchaînement d'un contrôleur à l'autre se fait suite à des actions de l'utilisateur, ou selon la logique d'un contrôleur

Quelques autres outils

Rails

```
class UsagerController < ApplicationController
  before_filter :autoriser, :only => :nouveau
  after_filter :compression
  caches_action :liste

  def nouveau
   if request.get?
     @usager = Usager.new
   else
     @usager = Usager.new(params[:usager])
     @usager.save
     cookies[:nom] = @usager.nom
     redirect_to :action => "liste"
   end
  end

  def liste
   ...
```

On peut se servir des "filtres" réutilisables pour tout ou partie d'un contrôleur, appliqués avant ou après le traitement de la requête

Une politique de cache peut être définie comme ici sur une action précise, mais aussi au niveau d'une page complète ou d'un fragment

Rails transmet la requête complète au contrôleur, on peut donc regrouper dans la même action deux traitements différents, selon que l'on examine un GET ou un POST: typiquement pour générer un formulaire, ou en traiter le résultat

Les cookies sont gérés par Rails et transmis prêt à l'usage au contrôleur. La session de travail de l'utilisateur est gardée et manipulable exactement de la même manière

Même ces aspects très transversaux s'expriment encore dans ce même principe de contrôleur/action

Créer des **Vues**

“ActionView”

“ActionView”

- Utilise le principe de “gabarits” de pages HTML avec un peu de code inclus *(fichiers .rhtml)*
- Les données du contrôleur sont automatiquement transmises à la vue
- Les liens entre les pages, les formulaires s’expriment là encore en tant que contrôleurs et actions

Inclure du code

```
<html><body>
 <h1>Usager: <%= @usager.nom %> </h1>

 <h1>Livres empruntés</h1>
  <% for l in @usager.emprunts %>
    <%= l.titre %>: <%= l.auteur %> <br>
  <% end %>

</body></html>
```

<% ... %>

Le code Ruby à l'intérieur de ce tag sera exécuté, mais le résultat ne sera pas placé dans la page HTML

Parfait pour les tests, les boucles, et tous les petits traitements

<%= ... %>

Le code Ruby à l'intérieur de ce tag sera exécuté, et le résultat sera inclus dans la page HTML

Parfait pour afficher les informations transmises par le contrôleur

Afficher l'information

```
<html><body>
 <h1>Usager: <%= @usager.nom %> </h1>

 <h1>Livres empruntés</h1>
  <% for l in @usager.emprunts %>
    <%= l.titre %>: <%= l.auteur %> <br>
  <% end %>

</body></html>
```

Les variables commençant par @ sont celles définies dans le contrôleur, et Rails va automatiquement les rendre accessibles à la vue.

On parcourt simplement ici les emprunts d'un usager, mais ce qui est manipulé ici est bel et bien un *modèle* ActiveRecord qui rend transparent l'accès à la base de données, comme vu précédemment

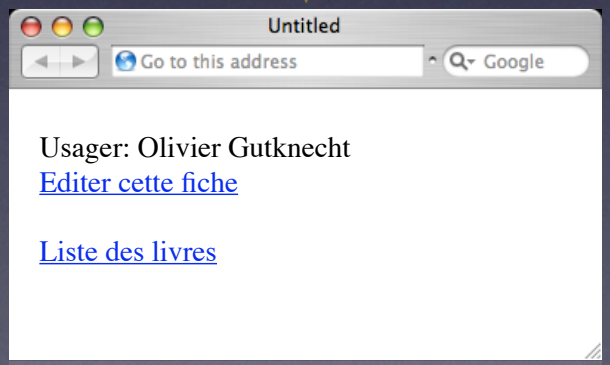
Générer un lien

Vue RHTML

```
Usager: <%= @usager.nom %> <br>
<%= link_to("Editer cette fiche", :action => "editer", :id => @usager) %><p>
<%= link_to("Liste des livres", :controller => "livre", :action => "liste")%>
```

HTML généré

```
Usager: Olivier Gutknecht <br>
<a href="/usager/editer/1">Editer cette fiche</a> <p>
<a href="/livre/liste">Liste des livres</a>
```



Les outils de génération de liens

Même dans une vue, Rails permet de continuer à travailler dans ce même univers de contrôleur / action / paramètres.

On peut ainsi exprimer les liens à un niveau plus abstrait et s'affranchir de la gestion des URL. Cela permet également de simplifier l'écriture (par exemple, le contrôleur courant peut être omis).

Le périple d'un formulaire

http://exemple.com/**livre**/**editer**/**42**

@params = {:id => 42}

```
def editer
  @livre = Livre.find(params[:id])
end
```

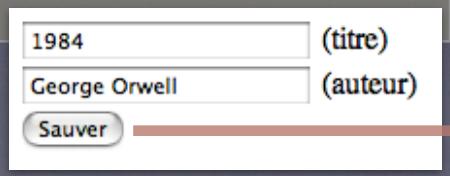
controllers/**livre_controller**.rb

```
<%= form_tag :action => 'sauver', :id => @livre %>
 <%= text_field 'livre', 'titre' %>
 <%= text_field 'livre', 'auteur' %>
 <%= submit_tag "Sauver"%>
<%= end_form_tag %>
```

views/**livre**/**editer**.rhtml

```
<form action="/livre/sauver/42" method="post">
 <input name="livre[titre]" value="1984">
 <input name="livre[auteur]" value="George Orwell">
 <input type="submit" value="Sauver">
</form>
```

Code HTML générée



@params = {:id => 42,
:livre => {:titre => "1984",
:auteur => "George Orwell"}}

```
def sauver
  l = Livre.find(params[:id])
  l.update_attributes(params[:livre])
  l.save
end
```

controllers/**livre_controller**.rb

Une belle coopération entre modèle, contrôleur et vue

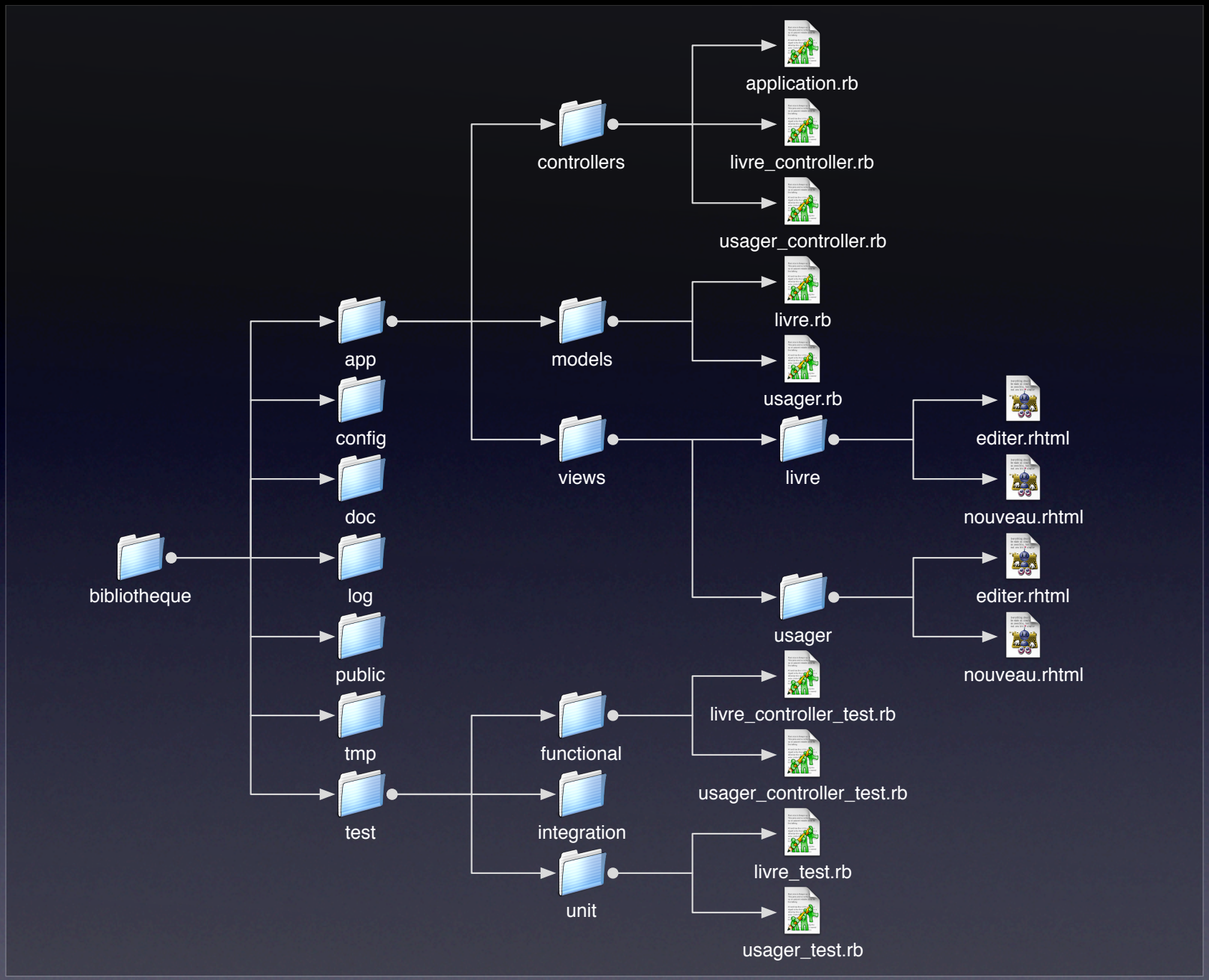
text_field, *form_tag*... : ces méthodes spécialisées (*helpers*) permettent de construire facilement des formulaires pré-remplis selon les données du modèle et d'en transmettre les modifications au contrôleur sous une forme immédiatement exploitable.

Développer
avec Ruby on Rails

L'importance des à-côtés

- Structurer le plus possible l'application
  - Même via l'organisation du projet sur le disque
- Encourager les bonnes pratiques
  - Migration de schémas, tests unitaires, nommage et placement des fichiers
- Faciliter un développement rapide et itératif
  - Mode de développement "live", échafaudages

Les outils de Rails (générateurs de code, serveur de test) sont conçus dans cet esprit.



Structure d'un projet

La séparation de l'application entre *modèles, vues, contrôleurs*, tests, plugins, configurations, logs, etc. est immédiatement visible.

Guider le développeur

Rails fournit de nombreux scripts permettant la génération de squelettes de code. Tous ces outils se basent sur les conventions de nommage et d'organisation de fichiers de Rails.

Script de génération de projet
(structure de répertoires, fichiers de configuration initiaux...)

Script de génération de modèle
(squelette de code applicatif, squelette de code de test, configuration du test...)

Script de génération de contrôleur
(Squelette de code applicatif, squelette du code de test, répertoire des vues...)

$ rails bibliotheque

```
      create
      create  app/controllers
      create  app/helpers
      create  app/models
      create  app/views/layouts
      create  config/environments
[...]
      create  doc
      create  log
      create  public/images
      create  public/javascripts
      create  public/stylesheets
      create  script/performance
      create  script/process
      create  test/functional
      create  test/integration
      create  test/unit
[...]
      create  tmp/sessions
      create  tmp/cache
      create  Rakefile
      create  README
```

$ script/generate model usager

```
      exists  app/models/
      exists  test/unit/
      exists  test/fixtures/
      create  app/models/usager.rb
      create  test/unit/usager_test.rb
      create  test/fixtures/usagers.yml
      exists  db/migrate
      create  db/migrate/002_create_usagers.rb
```

$ script/generate controler usager

```
      exists  app/controllers/
      exists  app/helpers/
      create  app/views/usager
      exists  test/functional/
      create  app/controllers/usager_controller.rb
      create  test/functional/usager_controller_test.rb
      create  app/helpers/usager_helper.rb
```

Rails fournit par défaut un certain nombre de générateurs de code (modèle, contrôleur, webservice, mailer...) mais permet aussi d'intégrer facilement un grand nombre de plugins: authentification, statistiques, graphes, outils AJAX...

Les “échafaudages”

Au début du développement d’une application Web, il est fréquent d’avoir besoin de créer/afficher/modifier/détruire des éléments du modèle (*CRUD: Create/Read/Update/Delete*)

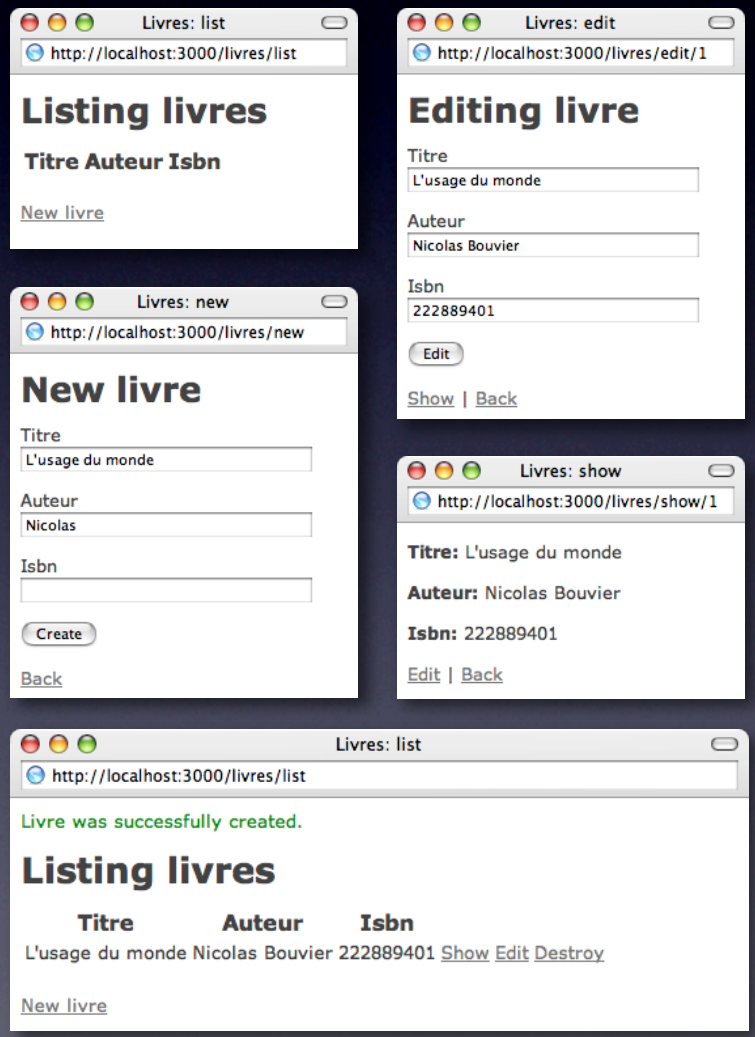
Le générateur d’échafaudages (*scaffolding*) de Rails produit automatiquement un contrôleur et des vues - simplistes - mais immédiatement fonctionnelles pour un modèle donné.

```
$ script/generate scaffold Livre
      exists  app/controllers/
      exists  app/helpers/
      create  app/views/livres
      exists  test/functional/
      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  app/views/livres/_form.rhtml
      create  app/views/livres/list.rhtml
      create  app/views/livres/show.rhtml
      create  app/views/livres/new.rhtml
      create  app/views/livres/edit.rhtml
      create  app/controllers/livres_controller.rb
      create  test/functional/livres_controller_test.rb
      create  app/helpers/livres_helper.rb
      create  app/views/layouts/livres.rhtml
      create  public/stylesheets/scaffold.css
```

Itérer, toujours

Rails incite à avoir des étapes de développement rapprochées, où l’application reste le plus possible fonctionnelle

L’échafaudage *(i.e. les différentes actions du contrôleurs et les vues)* sera retiré au fur et à mesure, au profit du code définitif.



En développement

Générer un squelette de code

Éditer/compléter les contrôleurs, modèles ou vues

Ruby
ou
RHTML

Tester l'application

Corriger une erreur

En développement, Rails fonctionne avec son propre mini-serveur Web: WEBrick.

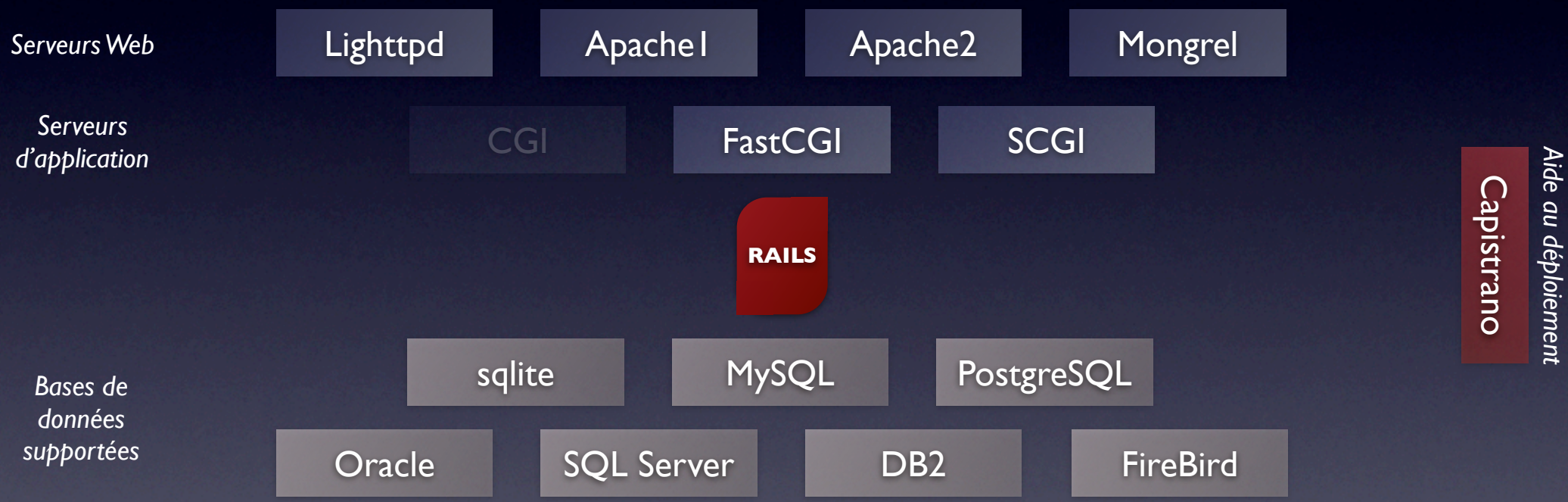
Dans ce mode, changer quelque chose dans l'application ne demande que deux choses pour être pris en compte :
sauver le fichier, recharger la page Web.

RAILS

En traitant les requêtes, Rails vérifiera que les fichiers sont à jour, et si besoin, rechargera dynamiquement le code, sans avoir à relancer l'application ou le serveur

En déploiement

De nombreuses options possibles



Faciliter l'expérimentation

Rails permet de travailler dans des *environnements* (test, production, développement...) isolés où les comportements de cache et les configurations de bases de données peuvent rester distincts.

Assembler le puzzle

Il est tout à fait possible de combiner ces solutions, par exemple un serveur Apache pour délivrer le contenu statique de l'application, et un modproxy vers un lighttpd ou un mongrel pour l'aspect dynamique

Les 2 clés de Rails

Ne pas se répéter !

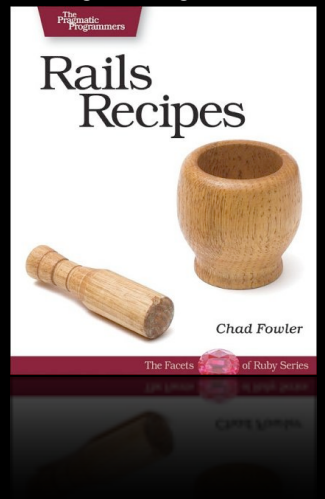
Rails fournit au développeur l'architecture et les outils nécessaires pour éviter de dupliquer inutilement du code. À lui de jouer le jeu jusqu'au bout.

Plutôt convention que configuration !

Rails part du principe que le comportement le plus courant doit être celui qui ne nécessite aucun code ou configuration, quitte à pouvoir le changer s'il ne convient pas.

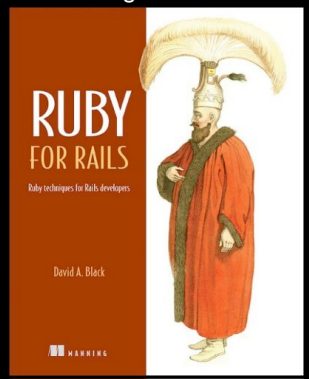
Références

PragmaticProgrammers

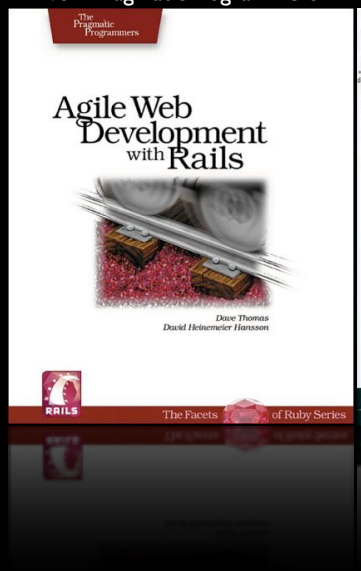


O'Reilly

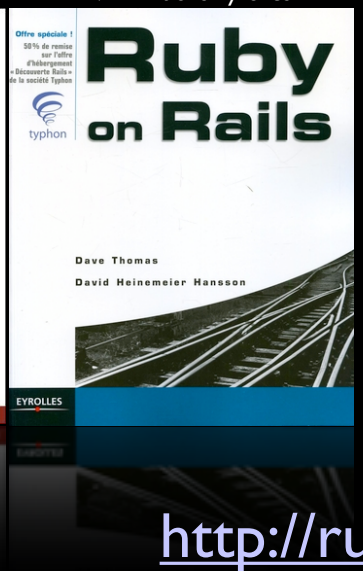
Manning Publications



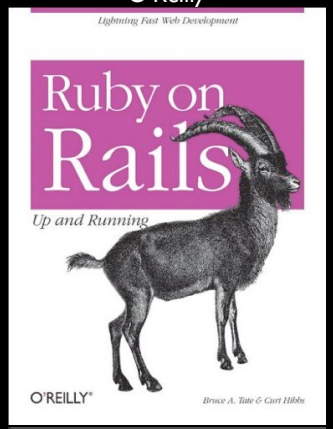
vo - PragmaticProgrammers



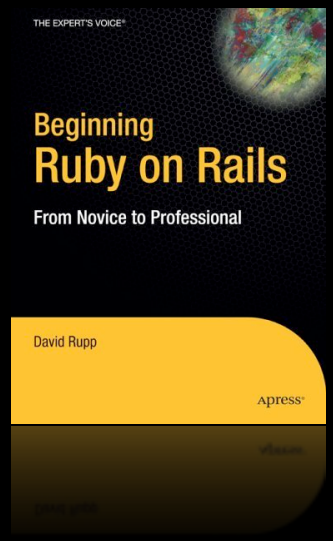
vf - Editions Eyrolles



O'Reilly



APress

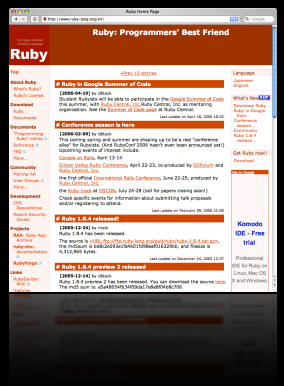


http://rubyonrails.org/books

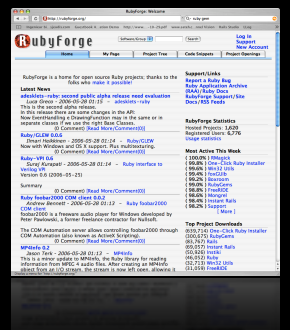
Références

http://www.rubyonrails.org/
Site officiel de Rails : documentation, weblog, downloads, wiki et howtos, APIs, bugtracker...

http://www.railsfrance.org/
Site Rails francophone : forums, liens, traductions, tutoriaux, actualité...



http://www.ruby-lang.org/
Site principal sur Ruby : FAQ, documentation, tutoriaux, projets Ruby...



http://www.rubyforge.org/
Repository de projets Ruby Open Source, base de données de projets

Creative Commons