

Anatomie d'un framework de développement d'application web

Les différents services apportés au développeur par un framework:

- interface bas niveau HTTP Requête/Réponse
- gestion de session
- routage : appel de code à partir de propriétés de la requête
- génération de vue : moteur de template
- persistance des informations

Interface HTTP

- gestion du codage/décodage des messages de HTTP
- gestion des contenus (corps des messages) : encodage, binaire/texte
- gestion des en-têtes (calcul automatique de la longueur du corps pour Content-Length)
- abstraction des en-têtes les plus courants (cookie, contenu, longueur du corps ...)

e.g. Rack

Gestion de session

- abstraction de la notion de session : la session est un conteneur

Interface type

- session = tableau associatif : positionner/lire valeur
- débiter une session
- signaler la fin d'une session

Ce qui est caché

- génération efficace et sécurisé des identifiants de session
- gestion automatique des identifiants de session (par cookie ou paramètre HTTP)
- gestion du stockage des sessions : fichier, BD ...
- Attention au côté multiprocessus d'un serveur web, une même session peut être partagée par plusieurs processus (ou threads).

Routage

sous réserve de suivre la norme HTTP ...

Éléments d'une requête HTTP paramétrant le routage :

- Path (chemin)
- type de la requête : GET, POST, PUT, DELETE, PATCH
- en-têtes, en particulier les en-têtes de négociation pour le format du contenu désiré

Redirection vers ...

- des pages statiques : le chemin indique un fichier
- un script : le chemin indique un script qui est exécuté pour renvoyer un résultat (type PHP/ASP)
- un ensemble de chemins capturés par une expression régulière est traité par le même code sur le serveur

Exemples

- GET /index.html web statique
- GET/POST /users.php?id=42 PHP/ASP
- GET/POST /users/show/42
- GET/POST /:controller/:action/:id le premier élément du chemin indique le code à exécuter (une classe par exemple), le reste donne des arguments (une méthode dans la classe)
- GET/PUT/DELETE /:type_ressource/:id, POST /:type_ressource : la méthode HTTP indique l'action (interface REST), l'en-tête Accept indique le format de représentation souhaité.

Moteur de template

Un template est un schéma de document, qui comporte :

- une partie fixe
- une partie variable

Un moteur de template permet de produire un document à partir d'un template et d'un ensemble d'arguments en remplaçant la partie variable dans le template par une transformation des arguments.

Erb

Erb est un moteur de template disponible dans la librairie standard de Ruby.

```
<pre 'code'>
require 'erb'

x = 42
template = ERB.new <<-EOF
  La valeur de x est: <%= x %>
  x est : <% if (x % 2) == 0 %>
    <%= "pair" %>
  <%>
</pre>
```

```
<% else %>  
<%= "impair" %>  
<% end %>
```

EOF

```
# binding : renvoie le contexte d'exécution  
# courant, variables, valeur de self, les méthodes  
puts template.result(binding)  
#La valeur de x est: 42  
#x est :  
#pair
```

Pour les applications web

- la partie fixe est du HTML/XML
- la partie variable est remplacée par du contenu stocké au niveau serveur, issue par exemple d'une base de données
- sécurité : attention si la partie variable est engendrée à partir de données utilisateurs (attaques par injection de script, vols de session)

Persistence des informations

- sauvegarder les données du domaine de l'application
- créer, consulter, mettre à jour, détruire : Create Read Update Delete
- gestion des accès concurrents
- typiquement des BD (BD relationnelles, BD documents), mais pas seulement (utilisation d'un web service)
- sécurité : prévention des attaques par injection de script

MVC pour le web

- Modèle: abstraction de la persistance des informations (e.g. remplacer le SQL par une couche objet)
- Vue : moteur de template
- Contrôleur : routage, pilotage des couches métier et vue.

Une application web

- une configuration de routage
- un ou plusieurs contrôleur
- un ou plusieurs modèles
- de nombreux templates de vues

Schéma de cycle

En amont : réception par la couche HTTP d'une requête.

1. Routage : à la réception de la requête, décision du contrôleur devant la traiter
2. Contrôleur : pilotage du Modèle

3. Le Modèle manipule la couche de persistance
4. Contrôleur : pilotage de la couche Vue : rendu d'un template à partir de données issues de la couche Modèle

En aval : codage par la couche HTTP de la réponse, le corps étant constitué du rendu du template

Exemple : Sinatra

Sinatra est un DSL permettant de rapidement créer des applications web en ruby :

```
<pre 'code'>
# myapp.rb
require 'rubygems'
require 'sinatra'

get '/' do
  'Hello world!'
end
```

Installation

```
<pre 'code'>
gem install sinatra
```

Exécution

```
<pre 'code'>
ruby myapp.rb # aller voir http://localhost:4567
```

Anatomie de Sinatra

- basé sur Rack
- DSL exprimant le routage

```
<pre 'code'>
get|post|put|delete|patch|options '/' do
  ... do stuff ...
  [status, headers, body] | [status, body] | string | status | whatever.respond_to? :each
end
```

- rien d'autre de fixé
- intégration facilitée de couche de persistance et de moteur de templates.

```
<pre 'code'>
get '/' do
  erb :index # views/index.erb
end
```