Applications orientées données (NSY135)

4 – Applications Web Dynamiques

Auteurs: Raphaël Fournier-S'niehotta et Philippe Rigaux
(philippe.rigaux@cnam.fr,fournier@cnam.fr)

Département d'informatique
Conservatoire National des Arts & Métiers, Paris, France

Plan du cours

S3 Un embryon de MVC

Notre implémentation et les principes du MVC

Dans la séquence S2, on a respecté les principes :

- le modèle, sous forme de classes **Plain Old Java**, est bien indépendant du contexte (une application Web)
- il se charge de l'implantation de la logique "métier"; les classes du modèle pourraient très bien être développées par un programmeur totalement ignorant de la programmation Web;
- le contrôleur est implanté par des servlets qui font parfaitement l'affaire car elles sont justement conçues pour être au centre des échanges HTTP gérés par le serveur;
- il n'y a pas de texte "en dur" : le contrôleur, c'est de la coordination de composants
- les vues, c'est le point le plus délicat car rien dans le langage Java de base ne correspond au besoin d'intégrer des éléments dynamiques dans des fragments HTML;
- nous avons intégré directement les **tag libraries** les JSP pour obtenir une syntaxe moderne, assez propre, qui n'inclut aucune programmation java.

Lacunes actuelles et améliorations

Contrôleurs, actions, sessions

- un contrôleur se décompose en **actions**, ce qui donne une structure à notre application MVC
- chaque contrôleur gère une partie bien définie des fonctionnalités (ex. : gestion des utilisateurs)
- il est constitué d'actions correspondant aux tâches élémentaires, par exemple le formulaire d'identification, le formulaire de création d'un compte utilisateur, l'édition d'un compte.

- Problème : Les servlets ne sont pas nativement équipées pour une décomposition en actions
- notre contournement avec “doGet“ et “doPost“ ne se généralisera pas pour davantage d'actions
- nous n'avons pas vu le paramétrage de la requête, de la réponse, et surtout gestion des sessions

Lacunes actuelles et améliorations

Vues

- nous n'avons que des choses élémentaires avec les JSTP (JSP étendues aux **tag libraries**)
- nous ne traitons ni les tableaux par exemple,
- et l'on n'utilise pas de conditions ("si cette variable vaut plus que 2, j'affiche ceci, sinon cela")
- on aimerait aussi pouvoir combiner des fragments HTML, par exemple pour utiliser un **template** pour notre site, que nous appliquerions à chaque contrôleur.

Lacunes actuelles et améliorations

Modèle

- autre lacune : nous ne savons pas rendre les instances d'un modèle **persistantes**
- la persistance est la capacité d'une information à survivre à l'interruption de l'application ou à l'arrêt de la machine
- en pratique, la persistance repose sur le stockage sur support non volatil (ie.: pas en mémoire vive) : SSD, disque dur
- c'est la **base de données** qui se charge de ce stockage

- ici, notre convertisseur n'a pas besoin de persistance car la règle de conversion est immuable.
- dans la majorité des applications, le recours à une base de données est indispensable
- c'est un des points de conception et de réalisation les plus délicats

- pour chaque modèle, certains objets doivent être rendus persistants, d'autres non
- pour chacun d'entre eux, cela doit être un service, que les contrôleurs actionnent (sans y toucher en détail)
- **les vues ne doivent rien savoir de la persistance**

Framework

- pour répondre à ces besoins, on utilisera évidemment des **frameworks**
- c'est une boîte à outil, reposant sur des standards technologiques
- offrant la possibilité d'appliquer des **bonnes pratiques**
- dans notre contexte, ils reposeront l'environnement JEE
- il faut donc consacrer un peu de temps à l'étude de JEE

Organisation du code

- nous avons presque toutes les connaissances nécessaires pour étudier comment combiner une application Web et un framework de persistance

Quelques règles organiser notre code :

- Répertoires : deux **packages**
  - “controleurs“ contiendrait tous les contrôleurs, chacun implanté par une **servlet**;
  - “modeles“ contiendra toutes les classes du modèle.
  - un répertoire “vues“ dans “WebContent“, avec **un sous-répertoire par contrôleur** (du nom du contrôleur, en minuscules)
  - ce répertoire des vues d’un contrôleur contiendra au moins une vue, nommée “index.jsp“, qui présentera le contrôleur et ses actions.
  - Contrôleurs et actions
    - on passera un paramètre HTTP “action“, en mode “GET“, au contrôleur, pour déclencher l’action correspondante
    - si ce paramètre est absent ou s’il ne correspond pas à une action, le contrôleur affichera “index.jsp“

Exemple de squelette de contrôleur

```java
package controleurs;

/**
* Servlet implementation class Convertisseur
*/
@WebServlet("/convertisseur")
public class Convertisseur extends HttpServlet {

  private static final String  VUES="/vues/convertisseur/", DEFVUE="index.jsp";

  protected void doGet(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException {

  // On devrait récupérer l'action requise par l'utilisateur
  String action = request.getParameter("action");

  // La vue par défaut
  String maVue = VUES + DEFVUE;

  try {
    if (action == null) {
      // Rien à faire, la vue est celle par défaut
    } else if (action.equals("formulaire")) {
      // Action + vue formulaire de saisie
      maVue = VUES + "formulaire.jsp";
    } else if (action.equals("conversion")) {
      maVue = VUES + "conversion.jsp";
    } else if (action.equals(...)) {
      // Etc.
    }
  } catch (Exception e) {
    maVue = VUES + "exception.jsp";
    request.setAttribute("message", e.getMessage());
  }

  // On transmet à la vue
  RequestDispatcher dispatcher = getServletContext().getRequestDispatcher(maVue);
  dispatcher.forward(request, response);
  }
}
```

Résumé de la séquence

- les contrôleurs sont les composants qui réagissent aux requêtes HTTP, ils ne contiennent ni code métier, ni aucun élément de présentation;
- le modèle implante les fonctionnalités métiers; une classe du modèle doit être totalement indépendante d'un contexte d'exécution (application web ou autre) ou de critères de présentation;
- les vues se chargent de l'affichage, et c'est tout. On va les aborder plus en détail dans le chapitre suivant