MVC
Modèle vue controleur

GUINKO Tonguim Ferdinand

14 novembre 2011

Sommaire

Introduction : Modèle architectural

Jusqu'à présent, tous les exemples devéloppés dans le cadre de ce cours l'ont été en utilisant un modèle architectural ;

- Dans ce type d'architecture, les requêtes HTTP sont gérées par les composants Web (soit servlet, soit JSP), qui reçoivent ces requêtes, créent les réponses et les retournent aux clients. Un seul composant est donc responsable de la logique d'affichage, de la logique métier et de la manipulation des requêtes ;
- La logique métier, la logique d'affichage et la manipulation des requêtes sont mélangés dans un même composant. Cette architecture conduit à placer du code Java dans des JSP ou du code HTML dans les servlets ;

Introduction : Modèle architectural

- Lorsqu'il s'agit d'une petite application, cela ne pose pas de problème, et le sujet est correctement traité. Mais imaginez une application réelle avec des pages Web très sophistiquées et un traitement complexe des données. Une telle application est d'une maintenance très délicate.

Conclusion : Les données, la présentation et les traitements étaient donc mélangés jusqu'à présent dans des servlets ou des pages JSP.

Introduction : Modèle architectural : Exemple

Soit l'exemple suivant :

Considérons l'application web qui consiste en un formulaire de type QCM utilisé pour évaluer les connaissances des étudiants.

Pour avoir accès au formulaire, un étudiant doit nécessairement s'authentifier sur une fenêtre de connexion (page login.jsp) prévue à cet effet. Un étudiant se connecte donc au système à travers la page connexion.jsp en donnant un nom et un mot de passe. La soumission se connecte sur la page auth.jsp qui vérifie l'identité des utilisateurs.

- En cas d'erreur, un forward est effectué sur la page login.jsp ;
- En cas de réussite, le bean est chargé depuis la base et un forward est effectué vers la page form.jsp ;

Introduction : Modèle architectural : Exemple (... suite)

suite de l'exemple :

La page form.jsp ; se charge d'afficher un formulaire de modification. Le formulaire de validation se connecte sur la page save.jsp . cette dernière page enregistre les données du formulaire dans le bean et valide ce dernier en vérifiant la conformité des données entrées par l'utilisateur.

- Si les données sont valides, la page save.jsp sauvegarde le bean et affiche un message indiquant la réussite ;
- Si les données ne sont pas valides, la page save.jsp effectue un forward vers la page form.jsp afin que l'utilisateur corrige les erreurs.

Introduction : Modèle architectural : Exemple (... suite)

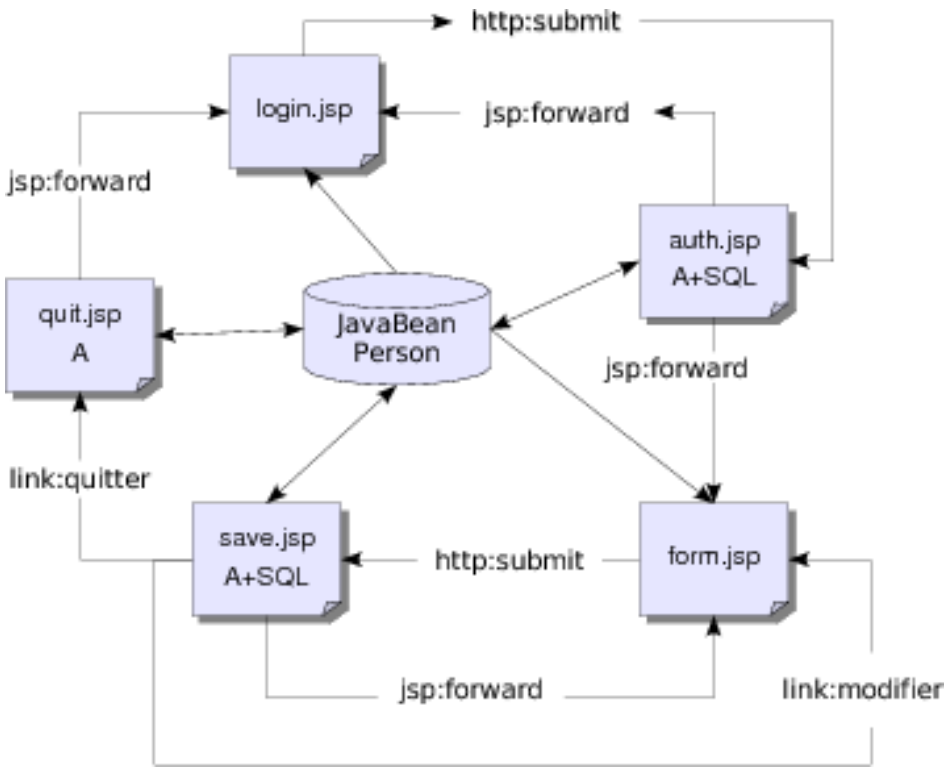

FIGURE: Illustration de l'architecture de l'application web de connexion à un formulaire QCM

Introduction : Modèle architectural : Exemple (... suite)

Dans cet exemple :

- le symbole A indique une action (du code Java qui agit sur les données) ;
- SQL des accès à la couche de stockage ;
- Les flèches qui arrivent sur le JabaBean indiquent une modification et celles qui partent une utilisation.

Introduction : Modèle architectural : Exemple (... suite)

Remarquons que :

- Les actions sont réparties dans trois pages ; cela ne facilite pas la maintenance du logiciel et son évolutivité (éparpillement de la logique applicative) ;
- Les pages JSP ne sont pas homogènes : certaines se contentent de construire une page HTML alors que d'autres sont responsables de la logique applicative ;
- Il existe deux points d'entrée (auth.jsp et save.jsp) ; Chacun doit vérifier avec soin la nature des données fournis en entrée (paramètres de la requête). Un seul point d'entrée aurait facilité la conception d'un code sécurisé.

Introduction : Modèle MVC

Remarque : il est très difficile de développer des applications complexes et/ou importantes en adoptant cette structure éclatée, dans laquelle les données, la présentation et les traitements étaient donc mélangés. Quelle solution ?

Solution : il existe une autre architecture de développement d'application web, qui partage les responsabilités, et qui gère de façon disctincte les données, la présentation et les traitements. Ce modèle est appelé Modèle - Vue - Contrôleur ou MVC .

Introduction : Modèle MVC

Modèle-Vue-Contrôleur : patron de conception inventé par Trygve Reenskaug en 1979 pour séparer les trois aspects d'une IHM (données, représentation et interaction).

Dans le modèle MVC :

- un composant est chargé de traiter les données,
- un autre de préparer l'affichage et
- un troisième de recevoir les requêtes.

Patron de conception : solution standardisée et réutilisable, à un problème récurrent, indépendante des langages de programmation.

Introduction : Modèle Vue Contrôleur

Dans le modèle MVC, les 3 composants qui permettent de séparer clairement les trois activités sont :

Le modèle

- Représente le comportement de l'application : traitements des données, interactions avec une BD, etc.
- Représentation des données manipulées par le programme ;
- Assure la gestion des données (texte, fichier mp3, terrain de jeu et joueurs, etc.) et garantit leur intégrité etc.
- Fournit les accès aux données ;
- Fournit les traitements applicables aux données
- Expose les fonctionnalités de l'application.

Le modèle représente le noyau fonctionnel de l'application

Introduction : Modèle Vue Contrôleur

La vue

- Représente l'interface avec laquelle l'utilisateur interagit : présente l'état du modèle et gère les requêtes utilisateurs ;
- Représentation(s) visuelle des données à l'utilisateur ;
- Vue 3D, tableau, texte, etc.
- Assure la consistance entre la représentation qu'elle donne et l'état du modèle/le contexte de l'application.

La vue représente les sorties de l'application

Introduction : Modèle Vue Contrôleur

Le contrôleur

- Gestion de l'interaction ; gestion des entrées de l'utilisateur ;
- Fournit la traduction des actions de l'utilisateur en actions sur le modèle ;
- Fournit la vue appropriée par rapport aux actions de l'utilisateur et des réactions du modèle ;
- N'effectue aucun traitement, ni ne modifie aucune donnée ;
- Gère les événements : met à jour la vue et/ou le modèle ;
- Effets de la souris, du clavier, etc.

Le contrôleur représente le comportement et la gestion des entrées de l'application.

Introduction : Modèle Vue Contrôleur

Si les interfaces entre ces 3 composants que sont le modèle, la vue et le contrôleur sont clairements définies, il devient facile d'en modifier un sans toucher aux 2 autres. Dans ce contexte, il est d'ailleurs possible de prévoir plusieurs affichages, par exemple, un pour les ordinateurs de type PC et un autre pour les PDA.

Introduction : Modèle Vue Contrôleur : Avantages

- Structure ≪propre≫ de l'application ;
- Indépendance données–représentation–comportements ;
- Modulaire et réutilisable ;
  - Vues interchangeables ;
  - Contrôleurs interchangeables ;
- Facilite les vues et contrôleurs multiples
  - Synchronisation ≪quasi-implicite≫

Introduction : Modèle Vue Contrôleur : Inconvénients

- Mise en place complexe dans le cas d'applications importantes ;
- Mises à jour potentiellement trop nombreuses :
  - ≪Spaghettis≫ dans le code ;
  - Temps d'exécution ;
- Contrôleur et Vue restent souvent fortement liés au Modèle.

Exemple : affichage d'une note

Affichage graphique et interactif d'une note (patron Note)

1. Modèle :
   - représentation informatique d'une note
   - un objet d'une classe adaptée
2. Vues :
   - représentation(s) graphique(s) de la note
   - une classe par représentation
3. Contrôleur(s) :
   - contrôle du modèle (changer la valeur de l'entier) et/ou de la vue (changer la représentation de l'entier)
   - une classe par grande catégorie de contrôles

Exemple : Modèle d'une note

```
package ca.uqar;
public class Note
{
    private int value;
    private boolean initialized;

    public int getValue()
    {
return value;
    }
    public void setValue(int value)
    {
if (value < 0 || value > 5)
{
    throw new RuntimeException("Out of bounds: " + value);
}
this.value = value;
initialized = true;
    }
    public boolean isInitialized()
    {
return initialized;
    }
}
```

Exemple : Vue texte

```
package ca.uqar;
import java.awt.Color;
import javax.swing.JLabel;

public class VueNoteTexte extends JLabel
{
    private Note note;

    public VueNoteTexte(Note note)
    {
this.note = note;
setHorizontalTextPosition(CENTER);
setOpaque(true);
update();
    }
    public void update()
    {
if(note.isInitialized())
{
    setForeground(Color.BLACK);
    setText(String.valueOf(note.getValue()));
}
else
{
    setForeground(Color.LIGHT_GRAY);
    setText("aucune");
}
```

Exemple : Vue texte

```
package ca.uqar;
import java.awt.Color;
import javax.swing.JLabel;

public class VueNoteTexte extends JLabel
{
    private Note note;

    public VueNoteTexte(Note note)
    {
this.note = note;
setHorizontalTextPosition(CENTER);
setOpaque(true);
update();
    }
    public void update()
    {
if(note.isInitialized())
{
    setForeground(Color.BLACK);
    setText(String.valueOf(note.getValue()));
}
else
{
    setForeground(Color.LIGHT_GRAY);
    setText("aucune");
}
```

Exemple : Programme principal

```
package ca.uqar;
import java.awt.FlowLayout;

import javax.swing.JFrame;
import javax.swing.JLabel;
public class DemoV1
{
    public static void main(String[] args)
    {
Note note = new Note (); //le modèle
VueNoteTexte vue = new VueNoteTexte(note); //la vue
//la fenêtre principale
JFrame cadre = new JFrame("Une note");
cadre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
cadre.getContentPane().setLayout(new FlowLayout());
cadre.getContentPane().add(new JLabel("Note :"));
cadre.getContentPane().add(vue);
cadre.pack();
cadre.setSize(200, 50);
// affichage
cadre.setVisible(true);
    }
}
```

Leçons

1. Modèle :
   - une ou plusieurs classes relativement arbitraires pour l'instant
   - accès au ≪ contenu ≫ par des méthodes set/get
2. Vue :
   - construite à partir des composants Swing de Java
   - liée au modèle (contient une référence vers le modèle)
   - un ou plusieurs objets :
   - contenant des instances de composants Swing
   - ou/et dont la classe hérite de celle d'un composant Swing
3. le patron est souple : nombreuses variantes possibles

Sites web ayant servis à la rédaction de ce cours

- Approche modèle vue contrôleur, Fabrice Rossi, Télécom ParisTech
- Java et le pattern MVC, F. Michel, LIRMM, Avignon
- Applications interactives-Modèle Vue-Contrôleur (MVC), Stéphane HUOT, IUT Orsay
- Application WEB : les bonnes pratiques, Jean-Luc Massat, Université de la Méditerranée, Marseille
-
-