OpenSourceJava-Troisième partie : MVC2 avec Struts

Auteurs
Didier GIRARD est à la tête de la Direction Technique Technologies & Solutions d'Improve. Expert J2EE et XML, créateur du site portail www.application-servers.com et animateur du site de littérature francophone abu.cnam.fr. (Didier.Girard@improve.fr)

Jean-Noël RIBETTE a rejoint IMPROVE en 2001. C'est un développeur actif sur le projet Struts. (Jean-Noel.Ribette@improve.fr)

LES APPLICATIONS WEBS SONT DE PLUS EN PLUS COURANTES. NEANMOINS, LEUR DEVELOPPEMENT RESTE UNE CHOSE DIFFICILE, LES METHODES ACTUELLES PRESENTENT ENCORE CERTAINS DEFAUTS. LES TECHNOLOGIES S'AMELIORENT, UN NOUVEAU TYPE DE FRAMEWORK PROPOSANT DE SUIVRE UN MODELE EFFICACE DE PROGRAMMATION, EMERGE : LES FRAMEWORKS MVC2. DANS CET ARTICLE STRUTS, PROJET OPENSOURCE APACHE, EST PRESENTE.

RETROUVEZ CET ARTICLE DANS « DEVELOPPEUR REFERENCE V1.2 »

CGI, servlet, JSP, ASP et PHP3

Différentes technologies existent pour développer des applications web : les CGI, les Servlets, ou encore les scripts (ASP, PHP et JSP). Les CGI, ou Common Gateway Interface, ont permis de développer les premiers sites webs dynamiques. Cette technologie permet d'appeler un programme externe au serveur web lors d'une requête. Ce programme peut accéder à toutes les ressources nécessaires, comme une base de données, et construire la page en fonction de la requête. Néanmoins, les CGI ont certains inconvénients : mixage de code HTMlL et de code de programmation qui rend la maintenance difficile, surcharge mémoire avec le lancement d'un nouveau processus pour chaque requête etc. Les CGI ne sont donc pas adaptés à la création d'applications webs importantes. Avec le langage Java est apparue une nouvelle technologie: les servlets. Ces petits 'serveurs' ou 'services' sont écrits en langage Java et utilisent une API spécifique. Ils corrigent certaines faiblesses des CGI. Initialisé à la création, il n'en existe qu'une seule instance. Les performances sont donc améliorées. Cependant, le développeur doit toujours mixer le code Java et HTML. De plus, la moindre modification oblige à recompiler le servlet et à le recharger. Les JSP, ou Java Server Pages, viennent résoudre ces problèmes de recompilation. Ici, c'est le code Java que l'on incorpore dans la page HTML avec des techniques de scriptings. Le serveur compile automatiquement la page en un servlet et l'exécute ensuite. Les asp et le php3 sont d'autres variantes de scripting, le php3 étant sans doute la solution la plus en vogue actuellement.

Les approches à base de scripting nécessitent l'incorporation importante de code applicatif dans le HTML. Ces techniques limitent aussi la réutilisation de code. Pour ce qui est du monde java il a donc été proposé de faire collaborer les servlets et les JSP dans les applications. Les servlets sont utilisées par les développeurs pour gérer les aspects programmations d'une application web et les JSP sont utilisés par les infographistes pour effectuer l'affichage.

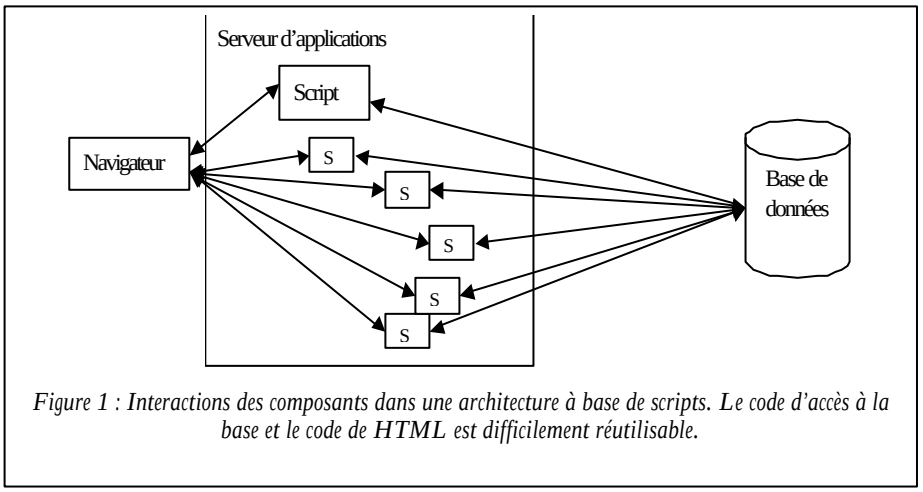

Figure 1 : Interactions des composants dans une architecture à base de scripts. Le code d'accès à la base et le code de HTML est difficilement réutilisable.

Servlet/CGI ou JSP/ASP/PHP3 ?

Une équipe développant un site web en servlet/CGI est composé d'informaticiens (donc de mauvais infographistes !). Une équipe développant un site web en JSP/ASP/PHP3 est composé d'infographistes (donc de mauvais informaticiens !). Le RAD implique des équipes réduites, dans ce cas, le choix de la technologie aura un impact sur le site : en servlet/CGI le site aura un look commun mais pourra avoir une complexité métier importante. En JSP/ASP/PHP3 un site sera joli mais aura une complexité métier faible.

Hors des projets RAD il est important d'avoir deux équipes et donc deux technologies : les servlets (pour les informaticiens) et les JSP (pour les infographistes).

On retrouve ainsi un servlet et un jsp par requête possible sur le site web. Le servlet ne contient plus de HTML, et le jsp contient juste le code nécessaire à l'affichage. Ce style de programmation respecte le paradigme MVC

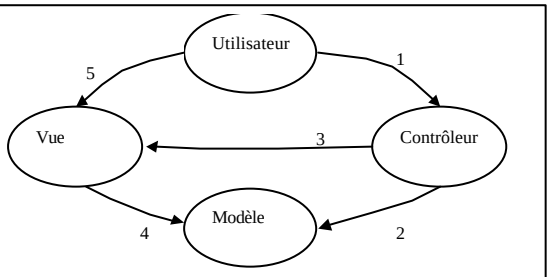

Figure 2 : Cheminement d'une requête lorsqu'un utilisateur agit sur une interface

Paradigme MVC

L'objectif de MVC est de faire collaborer une équipe à consonance infographie et une équipe à consonance informatique. Attention toutefois ceci augmente les interactions et nécessite une gestion de projet plus élaborée.

Le paradigme MVC est un schéma de programmation qui propose de séparer une application en 3 parties :

- Le modèle, qui contient la logique et l'état de l'application.
- La vue, qui représente l'interface utilisateur.
- Le contrôleur, qui gère la synchronisation entre la vue et le modèle. Le contrôleur réagit aux actions de l'utilisateur en effectuant les actions nécessaires sur le modèle. Le contrôleur surveille les modifications du modèle et informe la vue des mises à jour nécessaires.

La figure 2 décrit le traitement d'une requête par e modèle MVC.

1. L'utilisateur manipule l'interface homme/machine. Un événement est envoyé. Cet événement est récupéré par le contrôleur.
2. Le contrôleur effectue l'action demandée par l'utilisateur en appelant les méthodes nécessaires sur le modèle.
3. Le contrôleur informe la vue d'un changement d'état du modèle.
4. La vue interroge le modèle afin de connaître son état.
5. L'utilisateur voit le résultat de son action.

Au final, une telle séparation favorise le développement et la maintenance des applications :

- Le modèle étant séparé des autres composants, il est développé indépendamment. Le développeur du modèle se concentre sur le fonctionnel et le transactionnel de son application.
- Le modèle n'est pas lié à une interface, il peut donc être réutilisé (passage d'une application avec interface en java à une application avec interface web)

Dans les applications J2EE le modèle est assuré par un EJB, le contrôleur est assuré par des servlets et la vue par des JSP. Pour plus d'information sur l'implémentation en java du paradigme MVC se reporter à l'article du mois précédent (*servlet/JSP/MVC avec Tomcat*).

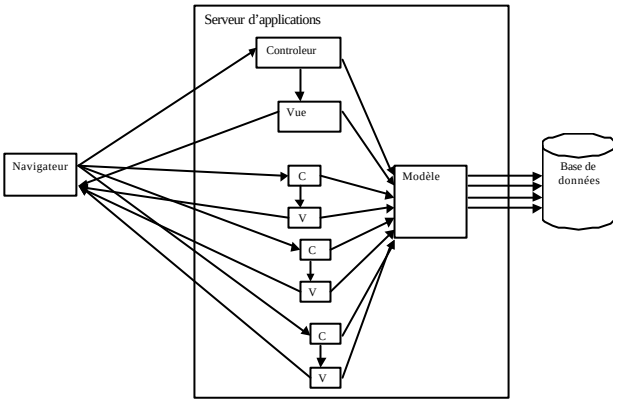

Figure 3 : Interaction des composants dans une architecture MVC. Le code d'accès aux données et le code HTML sont intégrés dans des composants séparés. Il est donc possible de réutiliser le code

Le paradigme MVC est une avancée importante en terme d'architecture d'applications web. Elle n'est cependant pas encore idéale : elle oblige à écrire une multitude de servlets, qui sont autant de points d'entrée dans l'application. Pour palier cet inconvénient des frameworks ont été développés. Ces frameworks qui sont composés d'un seul servlet (ie un seul controleur) sont regroupés sous l'étiquette "Model 2" encore appelé "MVC2".

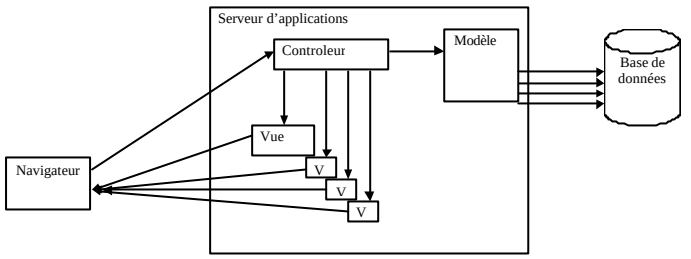

Figure 4 Interaction des composants dans une architecture MVC2. Le navigateur agit sur un seul composant, le contrôleur, l'architecture est simplifiée. Dans un framework évolué, le contrôleur intègre des services : gestion des erreurs, habilitation, autorisation, diffusion multi-terminaux,…

Les frameworks MVC2

Il existe actuellement un certain nombre de projet plus ou moins avancé visant à la création de framework MVC2. Ces projets tous basés sur les servlets progressent très vite. En voici une rapide présentation :

Barracuda

Ce framework qui se veut très complet est développé par la communauté Open Source Enhydra. Il est prévu qu'il comporte :

- Un modèle événementiel complet,
- Un modèle et une librairie de composants MVC pour l'interface utilisateur
- Un système de validation des formulaires et de mappings avec des objets Java.
- Un contrôleur MVC2
- Une librairie Javascript.

Hammock

Ce framework permet de développer une application web de la même façon qu'une application Swing. Pour cela, Hammock utilise le même modèle : système d'événements identique, présence de composants IHM 'Panel', et de layout 'FlowLayout ' ou 'TableLayout' etc. Ce système fonctionne sans JSP. Signalons qu'il n'est pas Open Source mais développé par OOP.

Tapestry

Ce framework est axé sur les composants : la création d'une page se fait par le développement de composants qui vont définir cette page. Il propose des concepts intéressants comme la séparation de l'état de la page et de la page elle-même afin de disposer d'un pool de pages ou encore des possibilités d'internationalisation via des templates. Tapestry facilite la gestion des erreurs et de la charge.

Tapestry a d'abord été développé par Primix et est maintenant disponible en Open source.

Webwork

Ce framework utilise le concept du Pull Hierarchical Model View Controller. Il est similaire techniquement au framework Struts dont la présentation suit, mais son API est plus réduite. Il pourrait être utilisé avec d'autres technologies que les servlets.

Pourquoi choisir Struts comme framework MVC2

Struts fait parti du projet Apache. Or les projets Apache ont tendance à monopoliser toute l'attention des utilisateurs. Sachant que la qualité des projets Open Source est apportés par le nombre d'utilisateurs, on peut penser que Struts sera de meilleure qualité que les autres projets MVC2.

	Version	url	Puissance	Complexité
Barracuda	PR2	http://barracuda.enhydra.org/	+++	+++
Hammock	1.0	http://www.oop.com/TechnologiesHammock.jsp	+++	+++
Tapestry	0.2.9	http://sourceforge.net/projects/tapestry	++	+++
Webwork	0.95	http://sourceforge.net/projects/webwork/	+	+
Struts	1.0 beta 1	http://jakarta.apache.org/struts	++	++

Tableau 1 Projets de framework de type MVC2

Struts

Struts est un projet Open Source développé par la communauté Jakarta d'Apache. Il a débuté en mai 2000 sous la direction de Craig R Mc Clanahan, qui participe également au développement de Tomcat. Aujourd'hui, Struts est géré par plusieurs committers. Sa mailing-list comporte un millier de personnes. C'est un projet très actif.

Struts fournit un framework MVC comprenant les composants suivants :

- Un contrôleur facilement configurable permettant d'associer des actions (méthode d'un objet Java) à des requêtes HTTP.
- Des librairies de tags spécifiques pour créer facilement une vue.
- Un Digester, permettant de parser un fichier XML et d'en récupérer seulement les informations voulues.
- Des utilitaires permettant de remplir automatiquement des champs et de créer des applications supportant plusieurs langages.

Le modèle

Le modèle d'une application Struts est complètement standard : suivant le paradigme MVC, Struts et le modèle sont indépendant. Le modèle peut très bien accéder directement à une base donnée relationnelle, XML ou utiliser des EJB. Il faudra juste veiller à permettre son initialisation.

La vue

La vue sous Struts est constituée de pages JSP. Afin de rendre la création de ces pages aisées par un designer et d'y éviter l'introduction de code Java, Struts propose l'utilisation de 4 librairies de tags ou *taglib* spécifiques : *html*, *bean*, *logic* et *template*.

La taglib bean

Les tags de cette librairie peuvent être vue comme des améliorations des tags JSP <jsp :useBean> <jsp :getProperty> et <jsp :setProperty>. En effet, ils permettent de rendre accessible des objets dans les contextes standards (page, session, application) via des variables de scriptings. Il est ensuite possible de récupérer, de modifier ou d'afficher les valeurs des propriétés de ces objets.

Par exemple si la session contient un objet nommé 'item' disposant d'une méthode getPrize() et que l'on veut afficher son prix, il suffira d'écrire :

```
<bean:write name='item' property='prize'/>
```

Struts supporte également les nested et indexed properties. Par exemple si la session contient un objet nommé 'items' et a une méthode getItem(int item) qui renvoie un item. il est possible d'écrire :

```
<bean:write name='items'
property='item[2].prize'/>
```

La taglib bean propose également un tag permettant d'afficher un message en fonction de la langue de l'utilisateur. Dans l'optique de créer une application supportant plusieurs langues, les pages JSP ne doivent pas contenir de texte mais faire appel à ce tag. Dans Struts, les messages destinés à l'utilisateur sont associés à une clé et définis dans des fichiers spéciaux. Chaque fichier contient les messages dans une langue donnée. Le tag

```
<bean :message key='titre.index'/>
```

affichera le texte sous la clé titre.index dans la langue principale définie par le browser si disponible ou sinon dans la langue par défaut.

La taglib html

Cette librairie contient les tags nécessaires à la création d'interface utilisateur dynamique, et notamment des formulaires.

Les tags permettent de remplir automatiquement les champs à afficher et de retourner les valeurs en vues de leur gestion par le contrôleur. Ceci se fait en spécifiant dans le tag le nom de la propriété qui doit être affichée / modifiée. Cette propriété correspond à une propriété Javabeans d'un objet Formulaire qui aura été spécialement écrit par le développeur pour servir d'interface entre la vue et le contrôleur.

Les autres tags de la librairie permettent :

- D'afficher les messages d'erreur relatifs à l'application ou au remplissage d'un champ donné.
- De gérer plus facilement les liens et les sessions.

Voici par exemple comment créer un formulaire permettant de se loguer :

```
<html:form action= '/logon' focus='username'>
Username : <html:text property='username'/>
<br>
Password: <html:password property='password'/>
<html:submit property="submit" value="Submit"/>
</html:form>
```

L'utilisation de ce formulaire est plus détaillée dans la section exemple.

La taglib logic

Cette librairie définit tout un ensemble de tags permettant d'inclure ou de ne pas inclure le corps des tags en fonction de critères logiques portant sur la valeur des propriétés d'un objet. Il est également possible d'effectuer des itérations.

Par exemple, pour afficher un message en fonction de l'âge de l'utilisateur :

```
<logic:greaterThan name='utilisateur'
       property='age' value='17'>
   Vous êtes majeurs !
</logic:greaterThan>
```

Où pour afficher tous les prénoms des enfants d'une personne :

```
<logic:iterate name='user' property='enfants'
               id='enfant'>
<bean:write name='enfant'/><br>
</logic :iterate>
```

La taglib template

Cette dernière librairie est destinée à faciliter la création de pages suivant un modèle. Pour un exemple courant, imaginons la barre de navigation en haut de la page, une zone centrale et un copyright en bas. Les templates permettent de définir la barre de navigation et le copyright, puis ensuite de les inclure aisément dans toutes les pages.

Taglibs de Struts

Les taglibs de Struts sont très complètes et permettent de faire beaucoup de chose sans écrire de code Java. Des librairies répondant à des besoins plus spécifiques - création de composants, accès à une base de données, mises en page - sont en projet pour les prochaines versions de Struts.

Le contrôleur

Le contrôleur est la partie du framework qui fait le lien entre la vue et le modèle. C'est elle qui permet de gérer l'enchaînement de la navigation. Dans Struts, le point central du contrôleur est un servlet de la classe *ActionServlet*. Toutes les requêtes y aboutissent.

Du point de vue du développeur, la programmation du contrôleur passe par :

- L'écriture d'un fichier de configuration : *struts-config.xml*.
  Ce fichier décrit en particulier quelles sont les actions possibles et à quelles classes Actions il faut les associer (mapping).
- L'écriture des objets formulaires qui vont servir à la transmission des données de la vue au modèle.
  Ces objets étendent *ActionForm*. Ils contiennent toutes les propriétés des formulaires, ainsi qu'une méthode permettant de valider ou non les valeurs de ces propriétés. Lorsqu'un formulaire HTML est renvoyé, Struts instancie automatiquement l'*ActionForm* correspondant et initialise ses propriétés avant de les valider.
- L'écriture des *Action* à effectuer pour chaque requête.

  Une fois que Struts a validé les données envoyées, il appelle la méthode *perform*() de l'*Action* correspondante. Dans cette méthode, le développeur peut récupérer les données rentrées dans le formulaire via l'*ActionForm*, et effectuer les opérations nécessaires sur le modèle. Il peut pour cela s'aider des utilitaires de Struts permettant de recopier des propriétés d'un bean à un autre.

Struts fournissant le servlet cœur du contrôleur et des classes Action et ActionForm qu'il n'y a plus qu'à étendre, la création d'actions est très rapide.

Essayer Struts

Essayer Struts est très facile: la distribution binaire de Struts est fournie avec des fichiers .war parmi lesquels une application exemple. Il suffit avec Tomcat de copier ses fichiers dans le répertoire webapps après avoir installé un parseur XML JAXP (l'implémentation de référence de Sun ou Xerces par exemple) et de redémarrer le serveur pour faire fonctionner Struts.

Les personnes désirant recompiler Struts ou utiliser d'autres serveurs trouveront les informations nécessaires sur le site de Struts.

Exemple d'application Struts

Struts est fourni avec une application exemple très simple. Celle-ci montre comment effectuer des taches courantes comme afficher un formulaire, récupérer les données, les réafficher, afficher des listes dynamiques etc.

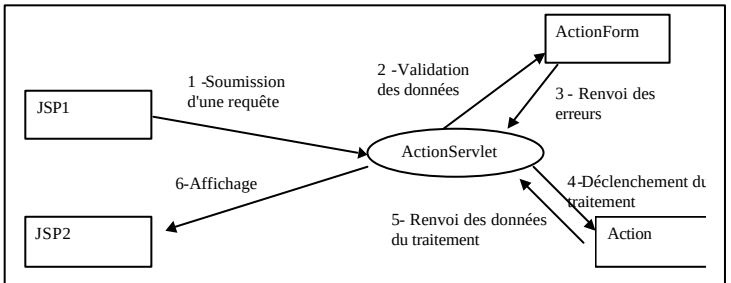

Figure 5 Traitement d'une requête par le contrôleur de Struts

Fonctionnalités

L'application exemple permet à un utilisateur de s'enregistrer et de maintenir une liste des différents comptes mails qu'il a sur Internet.

Les actions qu'il est possible d'effectuer avec cette application sont :

- Se loguer (associée par le contrôleur à la classe LogonAction)

- Se déloguer (classe LogoffAction)
- Enregistrer ses coordonnées (classe SaveRegistrationAction)
- Enregistrer les propriétés d'un compte mail (classe SaveSubscriptionAction)
- Afficher ses coordonnées (classe EditRegistrationAction)
- Afficher les propriétés d'un compte mail (classe EditSubscriptionAction)

L'application est composée de cinq pages :

- la page d'accueil, index.jsp,
- la page de login, logon.jsp,
- le menu principal, mainMenu.jsp,
- la page permettant d'afficher les propriétés de l'utilisateur, registration.jsp,
- la page permettant d'afficher la configuration d'un compte mail de l'utilisateur, subscription.jsp.

Grâce à l'emploi de la taglib logic, les pages servant à l'affichage des propriétés de l'utilisateur et de ses comptes servent aussi pour la création, la modification et la suppression. Les pages contenant des formulaires (logon.jsp, registration.jsp, subscription.jsp) sont chacune associées à un objet Form (on trouve respectivement les classes LogonForm, RegistrationForm et SubscriptionForm)

Ensemble des composants formant un traitement

Examinons en détail le processus permettant de se loguer :

La page logon.jsp contient les tags suivants :

```
…
<html:form action= '/logon' focus='username'>
…
 <html:text property='username'/>
…
 <html:password property='password'/>
…
<html:submit property="submit" value="Submit"/>
…
</html:form>
…
```

se qui provoque l'affichage d'un formulaire avec deux champs login / password et un bouton submit.

Voici ce qui se passe lorsqu'on clique sur submit :

1. La requête est envoyée à l'adresse logon.do à laquelle va répondre le contrôleur de Struts.
2. Le contrôleur utilise le fichier struts-config.xml pour déterminer quel est le formulaire qui va permette la validation des données (dans notre exemple il s'agit de LogonForm) et quelle est l'action associée à cette requête (dans notre exemple il s'agit de LogonAction).

Extraits du fichier struts-config.xml :

```
…
<!-- Logon form bean -->
    <form-bean      name="logonForm"

type="org.apache.struts.example.LogonForm"/>
…
<!-- Process a user logon -->
    <action    path="/logon"

type="org.apache.struts.example.LogonAction"
               name="logonForm"
              scope="request"
              input="/logon.jsp">
    </action>
…
```

1. Le servlet va donc instancier un LogonForm et l'initialiser avec les valeurs des attributs fournis dans la requête (password et username). Pour se faire LogonForm propose les méthodes setUsername et setPassword :

```
public class LogonForm extends ActionForm {
   …
   public void setPassword(String password) { …}
   public void setUsername(String username) {…}
   public String getUsername() {…}
   public String getPassword() {…}
   …
}
```

2. Le servlet va ensuite appeler la méthode validate() de LogonForm. Celle-ci renvoie une erreur si au moins un des deux champs est vide.

```
public class LogonForm extends ActionForm {
   …
   public ActionErrors validate(
         ActionMapping mapping,
         HttpServletRequest request) {
  ActionErrors errors = new ActionErrors();

  if ((username == null) ||
      (username.length() < 1))
    errors.add("username",
              new
ActionError("error.username.required"));

if ((password == null) || (password.length() < 1))
  errors.add("password",
      new ActionError(
        "error.password.required"));
             return errors;
      }
}
```

3. Si il n'y a pas d'erreur, le controleur va appeler la méthode perform() de LogonAction. Sinon il va renvoyer à la page de login.
4. La méthode perform() contient le code métier associé à une opération d'authentification.

```
public class LogonAction extends Action {
  …
  public ActionForward perform(
       ActionMapping mapping,
       ActionForm form,
       HttpServletRequest request,
       HttpServletResponse response) throws … {
…
// récupération des paramètres
String username =
     ((LogonForm) form).getUsername();
String password =
     ((LogonForm) form).getPassword();

// recuperation de la base de données
Hashtable database = (Hashtable)
servlet.getServletContext().getAttribute(..) ;
…

// création de l'utilisateur
// (utilisation du modèle)
User user = (User) database.get(username);
if (user!=null &&
   (!user.getPassword().equals(password)))
   user=null;
if (user==null)
  errors.add(ActionErrors.GLOBAL_ERROR,
    new ActionError("error.password.mismatch"));

// si on a des erreurs, retour à la page de login
if (!errors.empty()) {
  saveErrors(request, errors);
  return (new ActionForward(mapping.getInput()));
}

// ok, sauvegarde de l'utilisateur
HttpSession session = request.getSession();
session.setAttribute(Constants.USER_KEY, user);
…

// renvoie à la page suivante
return (mapping.findForward("success"));
}
```

Le code source complet de l'application, écrit par Craig R. McClanahan, est fourni avec Struts.

Extension du framework

Struts a été conçu de façon à pouvoir être aisément modifié par les développeurs pour répondre à leurs besoins particuliers. Cela se voit dans l'application exemple, qui redéfinit la classe ActionMapping pour introduire la notion de 'success' et de 'failure' pour une action. Des tags spécifiques sont également introduits pour vérifier si un utilisateur est logué ou non et pour écrire plus facilement certains liens.

Conclusion

Struts est un framework qui permet de développer des applications web en utilisant le modèle MVC2. Les applications conçues ainsi sont plus facilement maintenables et évolutives, de part la séparation des fonctionnalités du modèle, du contrôle, et de la vue.

L'utilisation du framework permet au développeur de se concentrer sur le modèle, Struts fournissant le cadre nécessaire au contrôle et à l'interface de l'application, permettant un développement rapide de ces parties.

Struts préfigure une nouvelle méthode de développement, sucession de la programmation par servlets.

Néanmoins, il convient d'ajouter que l'écriture d'une application utilisant Struts ne doit pas se décider à la légère : un certain temps est nécessaire pour prendre en main le concept et les API.

Références

Le projet Struts
http://jakarta.apache.org/struts

Un autre site sur Struts
http://www.husted.com/about/struts/

Le serveur Tomcat
http://jakarta.apache.org/tomcat

Le parseur XML de Sun
http://java.sun.com/xml

La mailing-list de Struts
struts-user@jakarta.apache.org

Les archives de la mailing-list :
http://www.mail-archive.com/struts-user@jakarta.apache.org/

D'autres articles sur Struts :
http://www-106.ibm.com/developerworks/library/j-struts/?dwzone=java
http://www.us-eh.com/craiger/articles/struts