Développons en Java v 2.00

68. Struts

Niveau : Supérieur

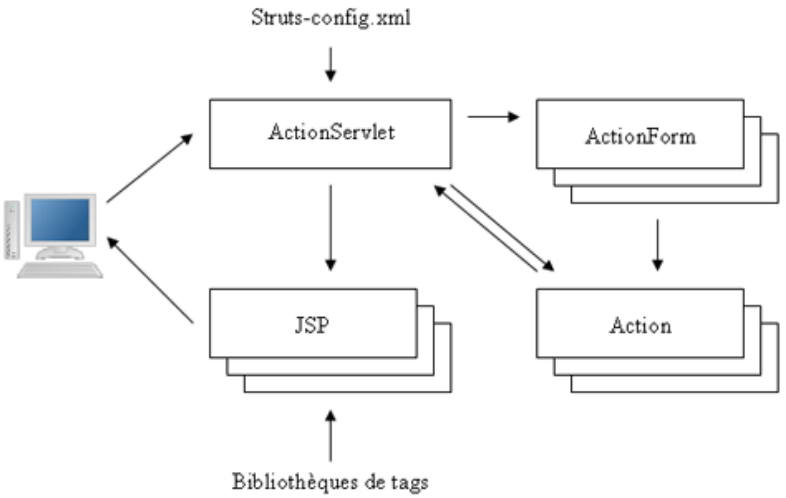
Struts est un framework pour applications web développé par le projet Jakarta de la fondation Apache. C'est le plus populaire des frameworks pour le développement d'applications web avec Java.

Il a été initialement développé par Craig Mc Clanahan qui l'a donné au projet Jakarta d'Apache en mai 2000. Depuis, Struts a connu un succès grandissant auprès de la communauté du libre et des développeurs à tel point qu'il sert de base à de nombreux autres framework open source et commerciaux et que la plupart des grands IDE propriétaires (Borland, IBM, BEA, ...) intègrent une partie dédiée à son utilisation.

Struts met en oeuvre le modèle MVC 2 basé sur une seule servlet faisant office de contrôleur et des JSP pour l'IHM. L'application de ce modèle permet une séparation en trois parties distinctes de l'interface, des traitements et des données de l'application.

Struts se concentre sur la vue et le contrôleur. L'implémentation du modèle est laissée libre aux développeurs : ils ont le choix d'utiliser des JavaBeans, un outil de mapping objet/relationnel, des EJB ou toute autre solution.

Pour le contrôleur, Struts propose une unique servlet par application qui lit la configuration de l'application dans un fichier au format XML. Cette servlet de type ActionServlet reçoit toutes les requêtes de l'utilisateur concernant l'application. En fonction du paramétrage, elle instancie un objet de type Action qui contient les traitements et renvoie une valeur particulière à la servlet. Celle-ci permet de déterminer la JSP qui affichera le résultat des traitements à l'utilisateur.



Les données issues de la requête sont encapsulées dans un objet de type ActionForm. Struts va utiliser l'introspection pour initialiser les champs de cet objet à partir des valeurs fournies dans la requête.

Struts utilise un fichier de configuration au format XML (struts-config.xml) pour connaître le détail des éléments qu'il va gérer dans l'application et comment ils vont interagir lors des traitements.

Pour la vue, Struts utilise par défaut des JSP avec un ensemble de plusieurs bibliothèques de tags personnalisés pour faciliter leur développement.

Struts propose aussi plusieurs services techniques : pool de connexions aux sources de données, internationalisation, ...

La dernière version ainsi que toutes les informations utiles peuvent être obtenues sur le site http://struts.apache.org/.

Il existe plusieurs versions de Struts : 1.0 (publiée en juin 2001), 1.1 et 1.2

Ce chapitre contient plusieurs sections :

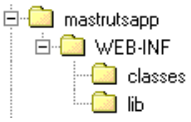
- L'installation et la mise en oeuvre
- Le développement des vues
- La configuration de Struts
- Les bibliothèques de tags personnalisés
- La validation de données

68.1. L'installation et la mise en oeuvre

Il faut télécharger la dernière version de Struts sur le site du projet Jakarta. La version utilisée dans cette section est la version 1.2.4.

Il suffit de décompresser le fichier jakarta-struts-1.2.4.zip dans un répertoire quelconque du système d'exploitation.

Il faut créer une structure de répertoires qui va accueillir l'application web, nommée par exemple mastrutsapp :



En utilisant Tomcat, une mise en oeuvre possible est de créer le répertoire de base de l'application dans le répertoire webapps.

Pour pouvoir utiliser Struts dans une application web, il faut copier les fichiers *.jar contenus du répertoire lib de Struts dans le répertoire WEB-INF/lib de l'application :

- commons-beanutils.jar
- commons-collection.jar
- commons-digester.jar
- commons-fileupload
- commons-logging.jar
- commons-validator.jar
- jakarta-oro.jar
- struts.jar

Il faut aussi copier les fichiers .tld (struts-bean.tld, struts-html.tld, struts-logic.tld, struts-nested.tld, struts-tiles.tld) dans le répertoire WEB-INF ou un de ses sous-répertoires.

Dans le répertoire WEB-INF, il faut créer deux fichiers :

- web.xml : le descripteur de déploiement de l'application
- struts-config.xml : le fichier de configuration de Struts

Le fichier web.xml minimal est le suivant :

Exemple :

```
01. <?xml version="1.0" encoding="UTF-8"?>
02. <web-app xmlns="http://java.sun.com/xml/ns/j2ee"
03.    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
04.    version="2.4">
05.   <display-name>Mon application Struts de tests</display-name>
06.
07.   <!-- Servlet controleur de Struts -->
08.   <servlet>
09.     <servlet-name>action</servlet-name>
10.     <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
11.     <init-param>
12.       <param-name>config</param-name>
13.       <param-value>/WEB-INF/struts-config.xml</param-value>
14.     </init-param>
15.     <init-param>
16.       <param-name>debug</param-name>
17.       <param-value>2</param-value>
18.     </init-param>
19.     <init-param>
20.       <param-name>detail</param-name>
21.       <param-value>2</param-value>
22.     </init-param>
23.     <load-on-startup>2</load-on-startup>
24.   </servlet>
25.
26.   <!-- Mapping des url avec la servlet -->
27.   <servlet-mapping>
```

```
    <servlet-name>action</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>

  <!-- page d'accueil de l'application -->
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>

  <jsp-config>
    <!-- Descripteur des bibliotheques personnalisees de Struts -->
    <taglib>
      <taglib-uri>/struts-bean</taglib-uri>
      <taglib-location>/WEB-INF/struts-bean.tld</taglib-location>
    </taglib>

    <taglib>
      <taglib-uri>/struts-html</taglib-uri>
      <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
    </taglib>

    <taglib>
      <taglib-uri>/struts-logic</taglib-uri>
      <taglib-location>/WEB-INF/struts-logic.tld</taglib-location>
    </taglib>

    <taglib>
      <taglib-uri>/struts-nested</taglib-uri>
      <taglib-location>/WEB-INF/struts-nested.tld</taglib-location>
    </taglib>

    <taglib>
      <taglib-uri>/struts-tiles</taglib-uri>
      <taglib-location>/WEB-INF/struts-tiles.tld</taglib-location>
    </taglib>
  </jsp-config>

</web-app>
```

Le mapping des URL de l'application prend généralement une des deux formes suivantes :

- préfixer chaque URL
- suffixer chaque URL avec une extension

Exemple de prefixe d'url :

```
<servlet-mapping>
  <servlet-name>action</servlet-name>
  <url-pattern>/do/*</url-pattern>
</servlet-mapping>
```

Exemple de suffixe d'url :

```
<servlet-mapping>
  <servlet-name>action</servlet-name>
  <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

Les exemples fournis sont de simples : n'importe quel préfixe ou extension peut être utilisé avec sa forme respective.

Le fichier struts-config.xml minimal est le suivant :

Exemple :

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts-config PUBLIC
   "-//Apache Software Foundation//DTD Struts Configuration 1.2//EN"
   "http://jakarta.apache.org/struts/dtds/struts-config_1_2.dtd">
<struts-config>

</struts-config>
```

Ces deux fichiers seront complétés au fur et à mesure des sections suivantes.

Comme Struts met en oeuvre le modèle MVC, il est possible de développer séparément les différents composants de l'application.

68.1.1. Un exemple très simple

L'exemple de cette section va simplement demander le nom et le mot de passe de l'utilisateur et le saluer si ces deux données saisies ont une valeur précise.

Cet exemple est particulièrement simple et sera enrichi dans les autres sections de ce chapitre : son but est de proposer un exemple d'enchaînement de deux pages et de récupération des données d'un formulaire.

Le fichier struts-config.xml va contenir la définition des entités utilisées dans l'exemple : le Form Bean et l'Action.

Exemple :

```
01. <?xml version="1.0" encoding="UTF-8"?>
02. <!DOCTYPE struts-config
03. PUBLIC "-//Apache Software Foundation//DTD Struts Configuration 1.0//EN"
04. "http://jakarta.apache.org/struts/dtds/struts-config_1_2.dtd">
05. <struts-config>
06.
07.   <form-beans type="org.apache.struts.action.ActionFormBean">
08.     <form-bean name="loginForm" type="com.jmd.test.struts.data.LoginForm" />
09.   </form-beans>
10.
11.   <action-mappings type="org.apache.struts.action.ActionMapping">
12.     <action path="/login" parameter="" input="/index.jsp" scope="request"
13.       name="loginForm" type="com.jmd.test.struts.controleur.LoginAction">
14.       <forward name="succes" path="/accueil.jsp" redirect="false" />
15.       <forward name="echec" path="/index.jsp" redirect="false" />
16.     </action>
17.   </action-mappings>
18.
19. </struts-config>
```

Il faut écrire la page d'authentification.

Exemple : la page index.jsp

```
01. <%@ page language="java" %>
02. <%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
03. <%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
04. <%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>
05. <html:html locale="true">
06.   <head>
07.     <title>Authentification</title>
08.     <html:base/>
09.   </head>
10.   <body bgcolor="white">
11.     <html:form action="login" focus="nomUtilisateur">
12.       <table border="0" align="center">
13.         <tr>
14.           <td align="right">
15.             Utilisateur :
16.           </td>
17.           <td align="left">
18.             <html:text property="nomUtilisateur" size="20" maxlength="20"/>
19.           </td>
20.         </tr>
21.         <tr>
22.           <td align="right">
23.             Mot de Passe :
24.           </td>
25.           <td align="left">
26.             <html:password property="mdpUtilisateur" size="20" maxlength="20"
27.                            redisplay="false"/>
28.           </td>
29.         </tr>
30.         <tr>
31.           <td align="right">
32.             <html:submit property="submit" value="Submit"/>
33.           </td>
34.           <td align="left">
35.             <html:reset/>
36.           </td>
37.         </tr>
38.       </table>
39.     </html:form>
40.   </body>
41. </html:html>
```

Il faut aussi définir la page d'accueil qui sera affichée une fois l'utilisateur authentifié.

Exemple : la page accueil.jsp

```
01. <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
02.     pageEncoding="ISO-8859-1"%>
03. <%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
04. <%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
```

```
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html:html locale="true">
  <head>
    <title>Accueil</title>
    <html:base/>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
  </head>
  <body bgcolor="white">
     <h1> Bienvenue <bean:write name="loginForm" property="nomUtilisateur"/></h1>
  </body>
</html:html>
```

Il faut définir l'objet de type ActionForm qui va encapsuler les données saisies par l'utilisateur dans la page d'authentification.

Exemple : la classe LoginForm

```
package com.jmd.test.struts.data;

import org.apache.struts.action.*;
import javax.servlet.http.HttpServletRequest;

public class LoginForm extends ActionForm {
  String nomUtilisateur;

  String mdpUtilisateur;

  public String getMdpUtilisateur() {
    return mdpUtilisateur;
  }

  public void setMdpUtilisateur(String mdpUtilisateur) {
    this.mdpUtilisateur = mdpUtilisateur;
  }

  public String getNomUtilisateur() {
    return nomUtilisateur;
  }

  public void setNomUtilisateur(String nomUtilisateur) {
    this.nomUtilisateur = nomUtilisateur;
  }

  public ActionErrors validate(ActionMapping mapping, HttpServletRequest request) {
    ActionErrors errors = new ActionErrors();
    return errors;
  }

  public void reset(ActionMapping mapping, HttpServletRequest request) {
    this.mdpUtilisateur = null;
    this.nomUtilisateur = null;
  }

}
```

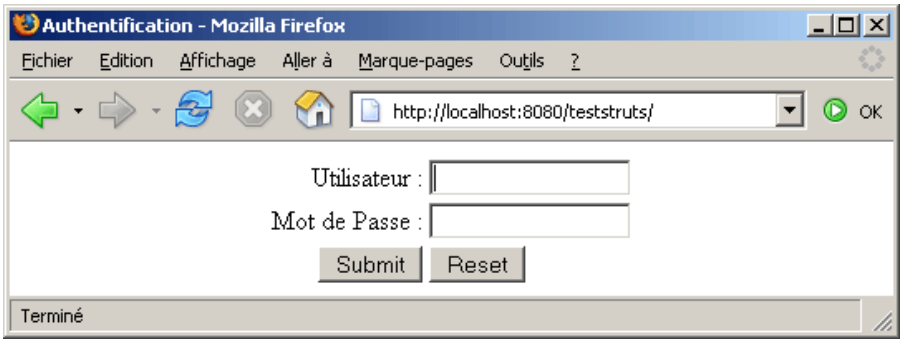
Enfin, il faut définir un objet de type Action qui va encapsuler les traitements lors de la soumission du formulaire.

Exemple : la classe LoginAction

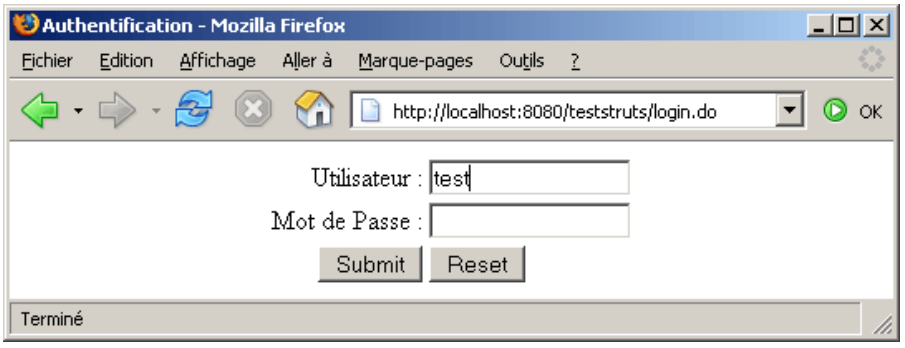
```
package com.jmd.test.struts.controleur;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

import com.jmd.test.struts.data.LoginForm;

public final class LoginAction extends Action {

  public ActionForward execute(ActionMapping mapping,
                               ActionForm form,
                               HttpServletRequest req,
                               HttpServletResponse res) throws Exception {
    String resultat = null;
    String nomUtilisateur = ((LoginForm) form).getNomUtilisateur();
    String mdpUtilisateur = ((LoginForm) form).getMdpUtilisateur();

    if (nomUtilisateur.equals("xyz") && mdpUtilisateur.equals("xyz")) {
      resultat = "succes";
    } else {
      resultat = "echec";
    }

    return mapping.findForward(resultat);
```

```
30.     }
31.
32. }
```

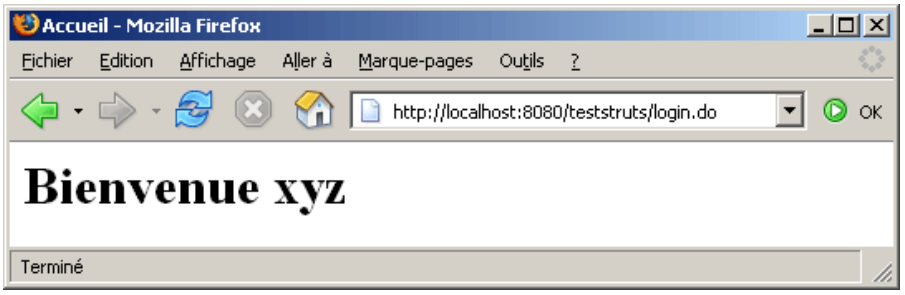
Pour exécuter cet exemple, il faut le déployer dans un conteneur web (par exemple Tomcat)



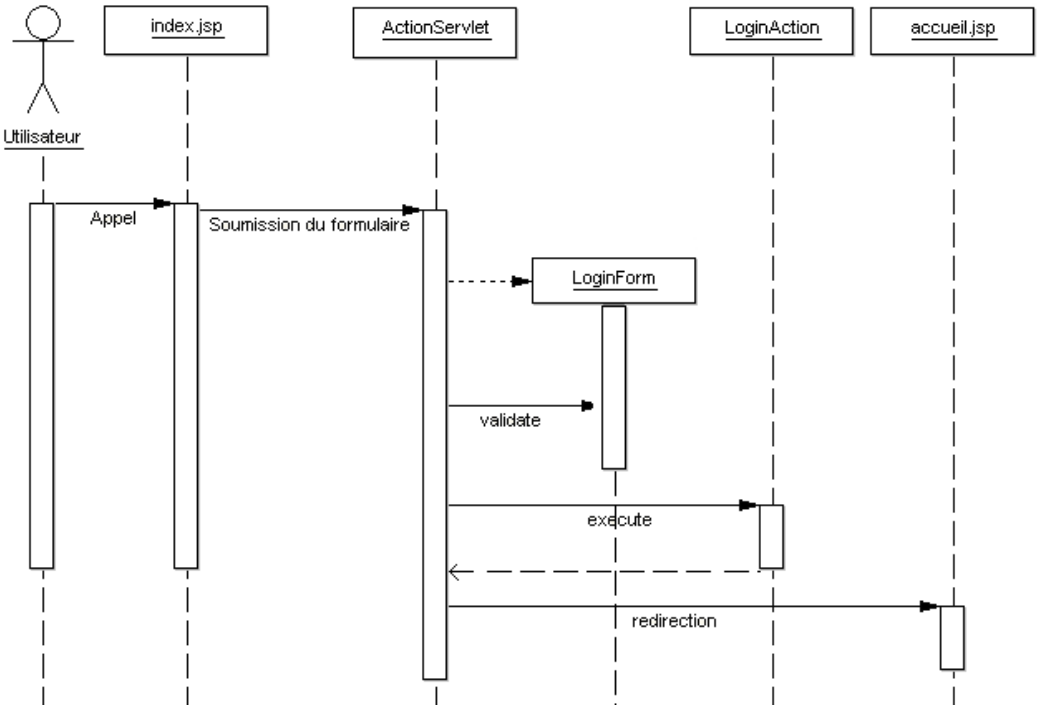
Si le nom d'utilisateur et le mot de passe saisis ne valent pas « xyz » alors la page d'authentification est réaffichée.



Si le nom d'utilisateur et le mot de passe saisie valent « xyz » alors la page d'accueil s'affiche.



Le diagramme de séquence ci-dessous résume les principales actions de cet exemple.



L'utilisateur appelle la page d'authentification index.jsp, saisit son nom d'utilisateur, son mot de passe et valide le formulaire.

L'ActionServlet intercepte la requête pour la traiter en effectuant les actions suivantes :

- Instancie un objet de type LoginForm et alimente ses données avec celles correspondantes dans la requête
- Appel de la méthode validate de la classe LoginForm pour valider les données saisies par l'utilisateur
- Détermination de l'Action à utiliser en fonction des informations contenues dans le fichier struts-config.xml. Dans l'exemple, c'est un objet de type LoginAction.
- Appel de la méthode execute() de la classe LoginAction qui contient les traitements à effectuer pour répondre à la requête. Elle renvoie un objet de type ActionForward
- En fonction de la valeur renvoyée par la méthode execute() et des informations du fichier de configuration, l'ActionServlet détermine la page à présenter à l'utilisateur
- La page déterminée est retournée au navigateur de l'utilisateur pour être affichée

68.2. Le développement des vues

Les vues représentent l'interface entre l'application et l'utilisateur. Avec le framework Struts, les vues d'une application web sont constituées par défaut de JSP et de pages HTML.

Pour faciliter leur développement, Struts propose un ensemble de nombreux tags personnalisés regroupés dans plusieurs bibliothèques possédant chacune un thème particulier :

- HTML : permet de faciliter le développement de pages Web en HTML
- Bean : permet de faciliter l'utilisation des Javabeans
- Logic : permet de faciliter la mise en oeuvre de la logique des traitements d'affichage
- Tiles : permet la gestion de modèles (templates)

Struts propose aussi au travers de ses tags de nombreuses fonctionnalités pour faciliter le développement : un formatage des données, une gestion des erreurs, ...

68.2.1. Les objets de type ActionForm

Un objet de type ActionForm est un objet respectant les spécifications des JavaBeans qui permet à Struts de mapper automatiquement les données saisies dans une page HTML avec les attributs correspondants dans l'objet. Il peut aussi réaliser une validation des données saisies par l'utilisateur.

Pour automatiser cette tâche, Struts utilise l'introspection pour rechercher un accesseur correspondant au nom du paramètre contenant la donnée dans la requête HTTP.

C'est la servlet faisant office de contrôleur qui instancie un objet de type ActionForm et alimente ses propriétés avec les valeurs contenues dans la requête émise à partir de la page.

Pour chaque page contenant des données à utiliser, il faut définir un objet qui hérite de la classe abstraite org.apache.struts.action.ActionForm. Par convention, le nom de cette classe est le nom de la page suivi de "Form".

Pour chaque donnée, il faut définir un attribut private ou protected qui contiendra la valeur, un getter et un setter public en respectant les normes de développement des Java beans.

Exemple :

```
01. package com.jmd.test.struts.data;
02.
03. import org.apache.struts.action.*;
04. import javax.servlet.http.HttpServletRequest;
05.
06. public class LoginForm extends ActionForm {
07.   String nomUtilisateur;
08.
09.   String mdpUtilisateur;
10.
11.   public String getMdpUtilisateur() {
12.     return mdpUtilisateur;
13.   }
14.
15.   public void setMdpUtilisateur(String mdpUtilisateur) {
16.     this.mdpUtilisateur = mdpUtilisateur;
17.   }
18.
19.   public String getNomUtilisateur() {
20.     return nomUtilisateur;
21.   }
22.
23.   public void setNomUtilisateur(String nomUtilisateur) {
24.     this.nomUtilisateur = nomUtilisateur;
25.   }
26.
27. ...
28.
```

```
29. }
```

La méthode reset() doit être redéfinie pour initialiser chaque attribut avec une valeur par défaut. Cette méthode est appelée par l'ActionServlet lorsqu'une instance de l'ActionForm est obtenue par la servlet et avant que cette dernière ne valorise les propriétés.

Exemple :

```
1. public void reset(ActionMapping mapping, HttpServletRequest request) {
2.   this.mdpUtilisateur = null;
3.   this.nomUtilisateur = null;
4. }
```

La signature de cette méthode est la suivante :
public void reset(ActionMapping mapping, HttpServletRequest request);

La méthode validate() peut être redéfinie pour permettre de réaliser des traitements de validation des données contenues dans l'ActionForm.

La signature de cette méthode est la suivante :
public ActionErrors validate(ActionMapping mapping, HttpServletRequest request);

Elle renvoie une instance de la classe ActionErrors qui encapsule les différentes erreurs détectées ou renvoie null si aucune erreur n'est rencontrée.

Exemple :

```
01. public ActionErrors validate(ActionMapping mapping, HttpServletRequest request) {
02.   ActionErrors errors = new ActionErrors();
03.
04.   if ((nomUtilisateur == null) || (nomUtilisateur.length() == 0))
05.     errors.add("nomUtilisateur", new ActionError("erreur.nomutilisateur.obligatoire"));
06.
07.   if ((mdpUtilisateur == null) || (mdpUtilisateur.length() == 0))
08.     errors.add("mdpUtilisateur", new ActionError("erreur.mdputilisateur.obligatoire"));
09.
10.   return errors;
11. }
```

Comme les objets de type ActionForm sont des éléments de la vue du modèle MVC, les objets de type ActionForm ne doivent contenir aucun traitement métier. La méthode validate() ne doit contenir que des contrôles de surface (présence de données, taille des données, format des données, ...).

Il faut compiler cette classe et la placer dans le répertoire WEB-INF/classes suivi de l'arborescence correspondant au package de la classe.

Il faut aussi déclarer pour chaque ActionForm, un tag <form-bean> dans le fichier struts-config.xml. Ce tag possède plusieurs attributs :

Attribut	Rôle
Name	le nom sous lequel Struts va connaître l'objet
Type	le type complètement qualifié de la classe de type ActionForm

Exemple :

```
1. <form-beans type="org.apache.struts.action.ActionFormBean">
2.   <form-bean name="loginForm" type="com.jmd.test.struts.data.LoginForm" />
3. </form-beans>
```

Chaque objet de type ActionForm doit être défini dans un tag <form-beans> et <form-bean> dans le fichier de description struts-config.xml.

Pour demander l'exécution des traitements de validation des données, il est nécessaire d'utiliser l'attribut validate dans le fichier struts-config.xml.

Remarque : pour assurer un découplage entre la partie IHM et la partie métier, il n'est pas recommandé de passer à cette dernière une instance de type ActionForm. Il est préférable d'utiliser un objet dédié respectant le modèle de conception Data Transfert Object (DTO).

68.2.2. Les objets de type DynaActionForm

Le développement d'un objet de type ActionForm pour chaque page peut s'avérer fastidieux à écrire (même si des outils peuvent se charger de générer les getters et les setters nécessaires) et surtout à maintenir dans le cas d'une évolution. Ceci est d'autant plus vrai si cet objet n'est utilisé que pour obtenir les données du formulaire.

Struts propose les objets de type DynaActionForm qui permettent, après déclaration dans le fichier de configuration, d'obtenir dynamiquement les données sans avoir à développer explicitement un objet dédié.

Les DynaActionForm doivent donc obligatoirement être déclarés dans le fichier de configuration struts-config.xml comme les ActionForm.

Exemple :

```
<form-beans>
  <form-bean name="saisirProduitActionForm"
             type="org.apache.struts.action.DynaActionForm">
    <form-property name="reference" type="java.lang.String"/>
    <form-property name="libelle" type="java.lang.String"/>
    <form-property name="prix" type="java.lang.String" initial="0"/>
  </form-bean>
</form-beans>
```

Par défaut la méthode validate() de la classe DynaActionForm ne réalise aucun traitement. Pour pouvoir l'utiliser, il est nécessaire de créer une classe fille qui va hériter de DynaActionForm et dans laquelle la méthode validate() va être redéfinie. C'est cette classe fille qui devra alors être précisée dans l'attribut type du tag <form-bean>.

68.3. La configuration de Struts

L'essentiel de la configuration de Struts se fait dans le fichier de configuration struts-config.xml.

68.3.1. Le fichier struts-config.xml

Ce fichier au format XML contient le paramétrage nécessaire à l'exécution d'une application utilisant Struts.

Il doit se nommer struts-config.xml et il doit être dans le répertoire WEB-INF de l'application.

Le tag racine de ce document XML est le tag <struts-config>.

Ce fichier se compose de plusieurs parties :

- la déclaration des beans de formulaire (ActionForm) dans un tag <form-beans>
- la déclaration des redirections globales à toute l'application dans un tag <global-forwards>
- la déclaration des Action dans un tag <action-mappings>
- la déclaration des ressources dans un ou plusieurs tags <message-ressources>
- la déclaration des plugins dans un ou plusieurs tags <plug-in>

Exemple :

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts-config
PUBLIC "-//Apache Software Foundation//DTD Struts Configuration 1.0//EN"
"http://jakarta.apache.org/struts/dtds/struts-config_1_2.dtd">
<struts-config>

  <form-beans type="org.apache.struts.action.ActionFormBean">
    <form-bean name="loginForm" type="com.jmd.test.struts.data.LoginForm" />
  </form-beans>

  <action-mappings type="org.apache.struts.action.ActionMapping">
    <action path="/login" parameter="" input="/index.jsp" scope="request"
      name="loginForm" type="com.jmd.test.struts.controleur.LoginAction">
      <forward name="succes" path="/accueil.jsp" redirect="false" />
      <forward name="echec" path="/index.jsp" redirect="false" />
    </action>
  </action-mappings>

</struts-config>
```

Le tag <form-beans> permet de définir les objets de type ActionForm et DynaActionForm utilisée dans l'application.

Les DynaActionForm sont déclarés grâce à un tag <form-bean> fils du tag <form-beans>. Comme pour les ActionForm, le paramètre name permet de préciser le nom qui va faire référence au bean. L'attribut type doit avoir comme valeur org.apache.struts.action.DynaActionForm ou une classe pleinement qualifiée qui en hérite.

Chaque attribut du bean doit être déclaré dans un tag fils <form-property>. Ce tag possède plusieurs attributs :

- name : nom de la propriété
- type : type pleinement qualifié de la propriété suivi de [] pour un tableau
- size : taille si le type est un tableau
- initial : permet de préciser la valeur initiale de la propriété

Exemple :

```
<form-beans>
  <form-bean name="saisirProduitActionForm"
             type="org.apache.struts.action.DynaActionForm">
    <form-property name="reference" type="java.lang.String"/>
    <form-property name="libelle" type="java.lang.String"/>
    <form-property name="prix" type="java.lang.String" initial="0"/>
  </form-bean>
</form-beans>
```

Le tag <global-exception> permet de définir des handlers globaux à l'application pour traiter des exceptions.

Le tag <action-mappings> permet de définir l'ensemble des actions de l'application. Celles-ci sont unitairement définies grâce à un tag <action>.

Le tag Action permet d'associer une URL (/login.do dans l'exemple) avec un objet de type Action (LoginAction dans l'exemple). Ainsi, à chaque utilisation de cette URL, l'ActionServlet utilise la classe Action associée pour exécuter les traitements.

La propriété path permet d'indiquer l'URI d'appel de ce mapping : c'est cette valeur qui sera par exemple indiquée (suffixée ou préfixée selon le paramétrage du fichier web.xml) dans l'attribut action d'un formulaire ou href d'un lien.

La propriété type permet d'indiquer le nom pleinement qualifié de la classe Action qui sera utilisée par ce mapping.

La propriété name permet d'indiquer le nom d'un bean de type ActionForm associé à ce mapping. Cet objet encapsulera les données contenues dans la requête http.

La propriété scope permet de préciser la portée de l'objet ActionForm instancié par l'ActionServlet précisé par l'attribut name :

- request : la durée de vie des données ne concerne que la requête
- session : les données concernent un utilisateur
- application : les données sont communes à tous les utilisateurs de l'application

Il est préférable d'utiliser la portée la plus courte possible et d'éviter l'utilisation de la portée application.

L'attribut validate permet de préciser si les données de l'ActionForm doivent être validées en faisant appel à la méthode validate(). La valeur par défaut est true.

La propriété input permet de préciser l'URI de la page de saisie des données qui sera réaffichée en cas d'échec de la validation des données.

Le tag fils <forward> permet de préciser avec l'attribut path l'URI d'une page qui sera affichée lorsque l'Action renverra la valeur précisée dans l'attribut name. L'attribut redirect permet de préciser le type de redirection qui sera effectuée (redirect si la valeur est true sinon c'est un forward qui sera effectué). L'URI fournie doit être relative dans le cas d'un forward et relative ou absolue dans le cas d'un redirect.

Les informations contenues dans ce tag seront utilisées lors de l'instanciation d'objets de type ActionForward

Le tag <global-forward> permet de définir des redirections communes à toute l'application. Ce tag utilise des tags fils de type <forward>. Les redirections définies localement sont prioritaires par rapport à celles définies de façon globales.

Le tag <message-ressources> permet de définir les ressources nécessaires à l'internationalisation de l'application.

Le tag <plug-in> permet de configurer des plugins de Struts tels que Tiles ou Validator.

Le tag <data-sources> permet de définir des sources de données. Chaque source de données est définie dans un tag <data-source>.

68.3.2. La classe ActionMapping

La classe ActionMapping encapsule les données définies dans un tag <Action> du fichier de configuration.

Chacune de ces ressources est définie dans le fichier de configuration struts-config.xml dans un tag <action> regroupé dans un tag <action-mappings>.

La méthode findForward() permet d'obtenir une redirection définie dans un tag <forward> de l'action ou dans un tag <global-forward>.

La classe ActionMappings encapsule une collection d'objets de type ActionMapping.

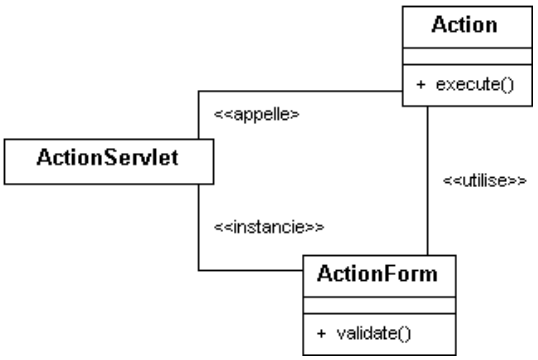
68.3.3. Le développement de la partie contrôleur

Basée sur le modèle MVC 2, la partie contrôleur de Struts se compose donc de deux éléments principaux :

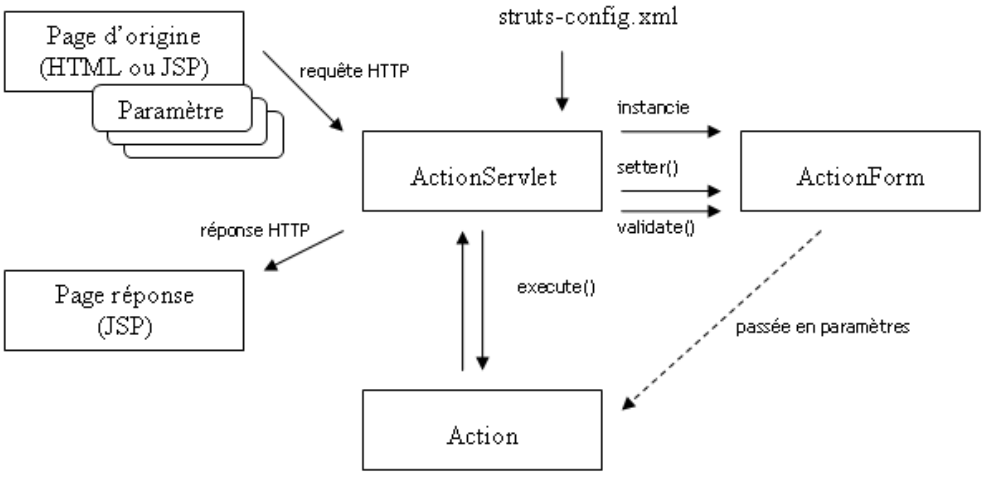
- une servlet de type org.apache.struts.action.ActionServlet
- plusieurs classes de type org.apache.struts.action.Action

La partie contrôleur est implémentée en utilisant une seule et unique servlet par application. Cette servlet doit hériter de la classe org.apache.struts.action.ActionServlet.

Cette servlet possède des traitements génériques qui utilisent les informations contenues dans le fichier struts-config.xml et dans des objets du type org.apache.struts.action.Action



Une instance de la classe RequestProcessor est utilisée par l'ActionServlet en appelant sa méthode process() pour initialiser un objet de type ActionForm associé à l'action liée à la requête en cours de traitement.



L'ActionServlet vérifie la présence d'une instance du type de l'ActionForm dans la session : dans la négative, une nouvelle instance est créée et ajoutée à la session. La clé associée au bean dans la session est définie par l'attribut attribute du tag <Action>.

La requête est ensuite analysée : pour chaque attribut présent dans la requête, la servlet recherche dans l'ActionForm une propriété dont le nom correspond en utilisant l'introspection : si elle est trouvée, la servlet appelle son setter pour lui associer la valeur contenue dans la requête. La correspondance des noms doit être exacte en respectant la casse.

Si la validation est positionnée dans le fichier de configuration, la servlet appelle la méthode validate() de l'ActionForm. Si la validation réussie et ou n'est pas demandée, l'ActionForm est passé en paramètre de la méthode execute() de l'instance d'Action.

68.3.4. La servlet de type ActionServlet

Le coeur d'une application Struts est composé d'une servlet de type org.apache.struts.action.ActionServlet.

Cette servlet reçoit les requêtes HTTP émises par le client et en fonction de celles-ci, elle appelle un objet du type Action qui lui est associé dans le fichier struts-config.xml. Le traitement d'une requête par une application Struts suit plusieurs étapes :

1. le navigateur client envoie une requête
2. réception de la requête par la servlet de type ActionServlet
3. en fonction de l'URI et du fichier de configuration struts-config.xml, la servlet instancie ou utilise l'objet de type ActionForm précisé. La servlet utilise l'introspection pour appeler les setters des propriétés dont les noms correspondent
4. la servlet instancie un objet de type Action associé à l'URI de la requête
5. la servlet appelle la méthode execute() de la classe Action. En retour de cet appel un objet de type ActionMapping permet d'indiquer à la servlet la page JSP qui sera affichée en réponse
6. la JSP génère la réponse HTML qui sera affichée sur le navigateur client

Pour respecter les spécifications J2EE, cette servlet doit être définie dans le fichier de déploiement web.xml de l'application web.

68.3.5. La classe Action

Un objet de type Action contient une partie spécifique de la logique métier de l'application : il est chargé de traiter ses données et de déterminer la page à afficher en fonction des traitements effectués.

Cet objet doit étendre la classe org.apache.struts.action.Action. Par convention, le nom de cette classe est le nom de la page suivi de "Action".

Il est important de développer ces classes de façon thread-safe : le contrôleur utilise une même instance pour traiter simultanément plusieurs requêtes. Il n'est donc pas recommandé d'utiliser des variables d'instances pour stocker des données sur une requête.

La méthode la plus importante de cette classe est la méthode execute(). C'est elle qui doit contenir les traitements qui seront exécutés. Depuis la version 1.1 de Struts, elle remplace la méthode perform() qui est deprecated mais toujours présente pour des raisons de compatibilité. La différence majeure entre la méthode perform() et execute() est que cette dernière déclare la possibilité de lever une exception.

La méthode execute() attend plusieurs paramètres :

- Un objet de type ActionMapping
- Un objet de type ActionForm
- Un objet de type HttpServletRequest
- Un objet de type HtppServletResponse

Il existe une autre surcharge de la méthode execute() qui attend les mêmes paramètres sauf pour les deux derniers qui sont de types ServletRequest et ServletResponse.

Les traitements typiquement réalisés dans cette méthode sont les suivants :

- Utiliser un cast vers l'objet de type ActionForm à utiliser pour l'objet fourni en paramètre de la méthode : ceci permet un accès aux données spécifiques de l'objet de type ActionForm
- Réaliser les traitements requis sur ces données
- Déterminer la page de retour en fonction des traitements réalisés sous la forme d'un objet de type ActionForward.

Une bonne pratique de développement consiste à faire réaliser les traitements par des objets métiers dédiés indépendant de l'API Struts. Ces objets peuvent par exemple être des Javabeans ou des EJB.

Pour obtenir un objet de type ActionForward encapsulant la page réponse, il faut utiliser la méthode findForward() de l'objet de type ActionMapping passé en paramètre de la méthode execute(). La méthode findForward() attend en paramètre le nom de la page tel qu'il est défini dans le fichier struts-config.xml.

Cet objet est retourné au contrôleur qui assurera la redirection vers la page concernée.

Pour stocker les éventuelles erreurs rencontrées, il est nécessaire de créer une instance de la classe ActionErrors

Exemple :

```
1. ActionErrors erreurs = new ActionErrors();
```

Pour extraire les données issues de l'objet ActionForm, il est nécessaire d'effectuer un cast vers le type de l'instance fournie en paramètre

Exemple :

```
1. String nomUtilisateur = "";
2. String mdpUtilisateur = "";
3.
4. if (form != null) {
5.   nomUtilisateur = ((LoginForm) form).getNomUtilisateur();
6.   mdpUtilisateur = ((LoginForm) form).getMdpUtilisateur();
7. }
```

Pour extraire les données issues d'un objet de type DynaActionForm, il est nécessaire d'effectuer un cast vers le type DynaActionForm de l'instance fournie en paramètre.

Comme les objets de type DynaActionForm ne possèdent pas de getter et setter, pour obtenir la valeur d'une propriété d'un tel objet il est nécessaire d'utiliser la méthode get() en passant en paramètre le nom de la propriété et de caster la valeur retournée.

Exemple :

```
1. DynaActionForm daf = (DynaActionForm)form;
2.
3. String reference = (String)daf.get("reference");
4. String libelle = (String)daf.get("libelle");
5. int prix = Integer.parseInt( (String)daf.get("prix")  );
```

Si une erreur est détectée dans les traitements, il faut instancier un objet de type ActionError et le fournir en paramètre avec le type de l'erreur à la méthode add() de l'instance de type ActionErrors.

Exemple :

```
1. if (nomUtilisateur.equals("xyz") && mdpUtilisateur.equals("xyz")) {
2.   resultat = "succes";
3. } else {
4.   erreurs.add(ActionErrors.GLOBAL_ERROR, new ActionError("erreur.login.invalide"));
5.   resultat = "echec";
```

```
6. }
```

A la fin des traitements de la méthode execute(), si des erreurs ont été ajoutées il est nécessaire de faire appel à la méthode saveErrors() pour les enregistrer.

Exemple :

```
1. if (!erreurs.isEmpty()) {
2.   saveErrors(req, erreurs);
3. }
```

Pour permettre un affichage des erreurs, il faut faire renvoyer à la méthode une instance de la classe ActionForward() qui encapsule la page émettrice de la requête.

Exemple :

```
1. return (new ActionForward(mapping.getInput()));
```

Sans erreur, le dernier traitement à réaliser est la création d'une instance de type ActionForward qui désignera la page à afficher en réponse à la requête.

Il y a deux façons d'obtenir cette instance :

- instancier directement un objet de type ActionForward
- utiliser la méthode findForward() de l'instance de type ActionMapping fournie en paramètre de la méthode execute()

Il existe plusieurs constructeurs pour la classe ActionForward dont les deux principaux sont :

- ActionForward(String path)
- ActionForward(String path, boolean redirect)

Le paramètre direct est un booléen qui ,avec la valeur true, fera procéder à une redirection vers la réponse (Response.sendRedirect()) et qui autrement provoquera le transfert vers la page réponse (RequestDispatcher.forward()).

L'utilisation de l'instance de type ActionMapping est sûrement la façon la plus pratique. Un appel à la méthode findForward() en précisant en paramètre le nom logique défini dans le fichier struts-config.xml permet d'obtenir un objet de type ActionForward pointant vers la page associée au nom logique.

Exemple :

```
1. return mapping.findForward(resultat);
```

A partir de l'objet de type HttpRequest, il est possible d'accéder à la session en utilisant la méthode getSession().

Exemple :

```
1. HttpSession session = request.getSession();
2. session.setAttribute(" key ", user);
```

68.3.6. La classe DispatchAction

La classe DispatchAction permet d'associer plusieurs actions à un même formulaire. Cette situation est assez fréquente par exemple lorsqu'une page propose l'ajout, la modification et la suppression de données.

Elle va permettre en une seule action de réaliser une des opérations supportées par l'action. L'opération à réaliser selon l'action qui est sélectionnée par l'utilisateur doit être fournie dans la requête http sous la forme d'un champ caché de type Hidden ou en paramètre dans l'URL.

Exemple :

```
01. <%@ page contentType="text/html;charset=windows-1252"%>
02. <%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean"%>
03. <%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html"%>
04. <html:html locale="true">
05.   <html>
06.     <head>
07.       <meta http-equiv="Content-Type" content="text/html; charset=windows-1252"/>
08.       <title>untitled</title>
09.       <SCRIPT language="javascript" type="text/javascript">
10.       function setOperation(valeur){
```

```
        document.forms[0].operation.value=valeur;
      }
    </SCRIPT>
    </head>
    <body>
      <html:form action="operations.do" focusIndex="reference">
        <html:hidden property="operation" value="aucune"/>
        <table>
          <tr>
            <td>
              <bean:message key="app.saisirproduit.libelle.reference"/>:
            </td>
            <td>
              <html:text property="reference"/>
            </td>
          </tr>
          <tr>
            <td colspan="2" align="center">
              <html:submit onclick="setOperation('ajouter');">Ajouter</html:submit>
              <html:submit onclick="setOperation('modifier');">Modifier</html:submit>
              <html:submit onclick="setOperation('supprimer');">Supprimer</html:submit>
            </td>
          </tr>
        </table>
      </html:form>
    </body>
  </html>
</html:html>
```

L'implémentation de l'action doit hériter de la classe DispatchAction. Il est inutile de redéfinir la méthode execute() mais il faut définir autant de méthodes nommées avec les valeurs possibles des opérations.

L'introspection sera utilisée pour déterminer dynamiquement la méthode à appeler en fonction de l'opération reçue dans la requête.

Exemple :

```
package test.struts.controleur;

import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.actions.DispatchAction;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class OperationsAction extends DispatchAction
{

  public ActionForward ajouter(
    ActionMapping       mapping,
    ActionForm          form,
    HttpServletRequest  request,
    HttpServletResponse response) throws IOException, ServletException
  {
    System.out.println("Appel de la methode ajouter()");
    return (mapping.findForward("succes"));
  }

  public ActionForward modifier(
    ActionMapping       mapping,
    ActionForm          form,
    HttpServletRequest  request,
    HttpServletResponse response) throws IOException, ServletException
  {
    System.out.println("Appel de la methode modifier()");
    return (mapping.findForward("succes"));
  }

  public ActionForward supprimer(
    ActionMapping       mapping,
    ActionForm          form,
    HttpServletRequest  request,
    HttpServletResponse response) throws IOException, ServletException
  {
    System.out.println("Appel de la methode supprimer()");
    return (mapping.findForward("succes"));
  }
}
```

Dans le fichier de configuration strut-config.xml, il faut déclarer l'action en précisant dans un attribut parameter le nom du paramètre de la requête qui contient l'opération à réaliser.

Exemple :

```
<struts-config>
...
  <form-beans>
...
    <form-bean name="operationsForm"
               type="org.apache.struts.action.DynaActionForm">
      <form-property name="operation" type="java.lang.String"/>
      <form-property name="reference" type="java.lang.String"/>
    </form-bean>
...
  </form-beans>
  <action-mappings>
...
    <action path="/operations" type="test.struts.controleur.OperationsAction"
            name="operationsForm" scope="request" validate="true" parameter="operation">
      <forward name="succes" path="/operations.jsp"/>
    </action>
...
  </action-mappings>
...
</struts-config>
```

Si la méthode à invoquer n'est pas définie dans la classe de type DispatchAction alors une exception est levée.

Exemple :

```
03-juil.-2006 13:14:43 org.apache.struts.actions.DispatchAction dispatchMethod
GRAVE: Action[/operations] does not contain method named supprimer
java.lang.NoSuchMethodException: test.struts.controleur.OperationsAction.supprimer(
org.apache.struts.action.ActionMapping, org.apache.struts.action.ActionForm,
javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse)
    at java.lang.Class.getMethod(Class.java)
```

Il est aussi possible d'utiliser plusieurs boutons avec pour valeur l'opération à réaliser. Ceci évite d'avoir à écrire du code JavaScript. Dans ce cas, chaque bouton doit avoir comme valeur de l'attribut property la valeur fournie à l'attribut parameter du tag <action>.

Exemple :

```
<%@ page contentType="text/html;charset=windows-1252"%>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean"%>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html"%>
<html:html locale="true">
  <html>
    <head>
      <meta http-equiv="Content-Type" content="text/html; charset=windows-1252"/>
      <title>untitled</title>
    </head>
    <body>
      <html:form action="operations.do" focusIndex="reference">
        <table>
          <tr>
            <td>
              <bean:message key="app.saisirproduit.libelle.reference"/>:
            </td>
            <td>
              <html:text property="reference"/>
            </td>
          </tr>
          <tr>
            <td colspan="2" align="center">
              <html:submit property="operation">ajouter</html:submit>
              <html:submit property="operation">modifier</html:submit>
              <html:submit property="operation">supprimer</html:submit>
            </td>
          </tr>
        </table>
      </html:form>
    </body>
  </html>
</html:html>
```

Attention cependant, la valeur du bouton est aussi son libellé : il est donc nécessaire de synchroniser le nom du bouton dans la vue et la méthode correspondante dans l'action. Ceci empêche l'internationalisation du libellé du bouton.

68.3.7. La classe LookupDispatchAction

Pour contourner le problème de l'internationalisation des opérations avec DispatchAction sans JavaScript, il est possible d'utiliser une action de type LookupDispatchAction.

Dans ce cas, le mapping ne se fait pas sur une valeur en dur mais sur la valeur d'une clé extraite des RessourcesBundles en fonction de la Locale courante.

La déclaration dans le fichier de configuration est similaire à celle nécessaire pour l'utilisation d'une action de type DispatchAction.

Exemple :

```
...
<struts-config>
  <form-beans>
..
    <form-bean name="operationsLookupForm"
               type="org.apache.struts.action.DynaActionForm">
      <form-property name="operation" type="java.lang.String"/>
      <form-property name="reference" type="java.lang.String"/>
    </form-bean>
...
  </form-beans>
  <action-mappings>
...
    <action path="/operationslookup" type="test.struts.controleur.OperationsLookupAction"
            name="operationsLookupForm" scope="request" validate="true" parameter="operation">
      <forward name="succes" path="/operationslookup.jsp"/>
    </action>
...
  </action-mappings>
...
</struts-config>
```

Dans la vue, le libellé des boutons de chaque action doit être défini dans les RessourcesBundles.

Exemple :

```
<%@ page contentType="text/html;charset=windows-1252"%>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean"%>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html"%>
<html:html locale="true">
  <html>
    <head>
      <meta http-equiv="Content-Type" content="text/html; charset=windows-1252"/>
      <title>Test LookupDispatchAction</title>
    </head>
    <body>
      <html:form action="operationslookup.do" focusIndex="reference">
        <table>
          <tr>
            <td>
              <bean:message key="app.saisirproduit.libelle.reference"/>:
            </td>
            <td>
              <html:text property="reference"/>
            </td>
          </tr>
          <tr>
            <td colspan="2" align="center">
              <html:submit property="operation">
                <bean:message key="operation.ajouter"/>
              </html:submit>
              <html:submit property="operation">
                <bean:message key="operation.modifier"/>
              </html:submit>
              <html:submit property="operation">
                <bean:message key="operation.supprimer"/>
              </html:submit>
            </td>
          </tr>
        </table>
      </html:form>
    </body>
  </html>
</html:html>
```

La valeur de chaque bouton doit être identique et précisée dans l'attribut property.

Il faut définir dans les ResourceBundles les libellés des boutons de chaque opération.

Exemple : ApplicationResources.properties

```
...
operation.ajouter   = Ajouter
operation.modifier  = Modifier
operation.supprimer = Supprimer
...
```

Exemple : ApplicationResources_en.properties

```
...
operation.ajouter   = Add
operation.modifier  = Modify
operation.supprimer = Delete
...
```

L'action doit hériter de la classe LookupDispatchAction. Il faut redéfinir la méthode getKeyMethodMap() pour qu'elle renvoie une collection de type Map dont chaque clé corresponde à la clé du ResourceBundle du bouton et chaque valeur à la méthode qui doit être invoquée.

La définition des méthodes de chaque opération est identique à celle utilisée avec une action de type DispatchAction.

Exemple :

```
package test.struts.controleur;

import java.util.HashMap;
import java.util.Map;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.actions.LookupDispatchAction;
import org.apache.struts.util.MessageResources;

public class OperationsLookupAction extends LookupDispatchAction
{
  public static final String OPERATION_AJOUTER = "operation.ajouter";
  public static final String OPERATION_MODIFIER = "operation.modifier";
  public static final String OPERATION_SUPPRIMER = "operation.supprimer";


 public Map getKeyMethodMap() {
  Map map = new HashMap();
  map.put(OPERATION_AJOUTER, "ajouter");
  map.put(OPERATION_MODIFIER, "modifier");
  map.put(OPERATION_SUPPRIMER, "supprimer");
  return map;
}

  public ActionForward ajouter(
    ActionMapping       mapping,
    ActionForm          form,
    HttpServletRequest  request,
    HttpServletResponse response) throws IOException, ServletException
  {
    System.out.println("Appel de la methode ajouter()");
    return (mapping.findForward("succes"));
  }

  public ActionForward modifier(
    ActionMapping       mapping,
    ActionForm          form,
    HttpServletRequest  request,
    HttpServletResponse response) throws IOException, ServletException
  {
    System.out.println("Appel de la methode modifier()");
    return (mapping.findForward("succes"));
  }

  public ActionForward supprimer(
    ActionMapping       mapping,
    ActionForm          form,
    HttpServletRequest  request,
    HttpServletResponse response) throws IOException, ServletException
  {
    System.out.println("Appel de la methode supprimer()");
    return (mapping.findForward("succes"));
  }
}
```

Grâce à la méthode getKeyMethodMap(), la valeur de chaque opération est déterminée dynamiquement en fonction de la Locale.

68.3.8. La classe ForwardAction

Cette action permet uniquement une redirection vers une page sans qu'aucun traitement ne soit exécuté.

L'intérêt est de centraliser ces redirections dans le fichier de configuration plutôt que de les laisser en dur dans la ou les pages qui en ont besoin.

Il suffit de définir une action dans le fichier struts-config.xml en utilisant les attributs :

- path : l'URI de l'action
- type : org.apache.struts.actions.ForwardAction
- parameter : la page vers laquelle l'utilisateur va être redirigé

Exemple :

```
<action path="/redirection"
        type="org.apache.struts.actions.ForwardAction"
        parameter="/test.jsp">
</action>
```

Pour utiliser cette action, il suffit de faire un lien vers le path de l'action.

Exemple :

```
<html:link action="redirection.do">Page de test</html:link>
```

68.4. Les bibliothèques de tags personnalisés

L'utilisation des bibliothèques de tags de Struts nécessite leur définition dans le fichier de déploiement web.xml et leur déclaration dans chaque page qui les utilise.

Exemple :

```
<%@ taglib uri="/WEB-INF/struts-html.tld"  prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld"  prefix="bean" %>
<%@ taglib uri="/WEB-INF/struts-logic.tld"  prefix="logic" %>
```

Les vues sont aussi composées selon le modèle MVC d'objets de type ActionForm ou DynaActionForm qui encapsulent les données d'une page. Ils permettent l'échange de données entre la vue et les objets métiers par le contrôleur.

68.4.1. La bibliothèque de tags HTML

Cette bibliothèque permet de faciliter le développement de page Web en HTML.

Pour utiliser cette bibliothèque, il faut, comme pour toute bibliothèque de tags personnalisés, réaliser plusieurs opérations :

1. copier le fichier struts-html.tld dans le répertoire WEB-INF de la webapp
2. configurer le fichier WEB-INF/web.xml pour déclarer la bibliothèque de tag

```
<taglib>
  <taglib-uri>struts-html.tld</taglib-uri>
  <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
</taglib>
```

3. ajouter dans chaque page JSP qui va utiliser cette bibliothèque un tag de directive taglib précisant l'utilisation de la bibliothèque

```
<%@ taglib uri="struts-html.tld" prefix="html" %>
```

La plupart de ces tags encapsulent des tags HTML notamment pour les formulaires mais ils assurent aussi des traitements particuliers à Struts.

Exemple :

Un lien vers une URL absolue avec HTML doit intégrer le nom de la webapp :
```
<a href="/testwebapp/index.jsp">
```

La balise Struts correspondante sera indépendante de la webapp : elle tient compte automatiquement du contexte de l'application
```
<html:link page="/index.jsp">
```

Il est cependant préférable d'utiliser un mapping défini dans le fichier struts-config.xml plutôt que d'utiliser un lien vers la page JSP correspondante. Ceci va permettre l'exécution de l'Action correspondante.

Exemple :

<html:link page="/index.do">Accueil</html:link>

Tag	Description
base	Encapsule un tag HTML <base>
button	Encapsule un tag HTML <input type="button">
cancel	Encapsule un tag HTML <input type="submit"> avec la valeur Cancel
checkbox	Encapsule un tag HTML <input type="checkbox">
errors	Affiche les messages d'erreurs stockés dans la session
file	Encapsule un tag HTML <input type="file">
form	Encapsule un tag HTML <form>
frame	Encapsule un tag HTML <frame>
hidden	Encapsule un tag HTML <input type="hidden">
html	Encapsule un tag HTML <html>
image	Encapsule une action affichée sous la forme d'une image
img	Encapsule un tag HTML <img>
javascript	Assure la génération du code JavaScript requis par le plug-in Validator
link	Encapsule un tag HTML <A>
messages	Affiche les messages stockés dans la session
multibox	Assure le rendu de plusieurs checkbox
option	encapsule un tag HTML <option>
options	Assure le rendu de plusieurs options
optionsCollection	Assure le rendu de plusieurs options
password	Encapsule un tag HTML <input type="password">
radio	Encapsule un tag HTML <input type="radio">
reset	Encapsule un tag HTML <input type="reset">
rewrite	Le rendu d'une URI
select	Encapsule un tag HTML <select>
submit	Encapsule un tag HTML <input type="submit">
text	Encapsule un tag HTML <input type="text">
textarea	Encapsule un tag HTML <input type="textarea">
xhtml	Le rendu des tags HTML est au format XHTML

Les tags les plus utilisés seront détaillés dans les sections suivantes.

68.4.1.1. Le tag <html:html>

Ce tag génère un tag HTML <html>.

Il possède plusieurs attributs dont les principaux sont :

Attribut	Rôle
lang	génère un attribut lang en accord avec celui stocké dans la session ou la requête, ou encore, selon la Locale par défaut
locale	utilise la valeur true pour forcer le stockage dans la session de la Locale correspondant à la langue de la requête Ce tag est deprecated depuis la version 1.2 car il crée automatiquement une session : utiliser l'attribut lang à la place
xhtml	utilise la valeur true pour assurer un rendu au format xhtml des tags

Ce tag doit être inclus dans un tag <html:form>.

68.4.1.2. Le tag <html:form>

Ce tag génère un tag HTML <form>.

Il possède de nombreux attributs correspondant aux attributs du tag html <form> dont les principaux sont :

Attribut	Rôle
action	URL à laquelle le formulaire sera soumis
enctype	type d'encodage du formulaire lors de la soumission
focus	nom de l'élément qui aura le focus au premier affichage de la page
method	méthode de soumission du formulaire
name	nom associé à la classe ActionForm
scope	portée de la classe ActionForm
target	cible d'affichage de la réponse
type	type de la classe ActionForm

68.4.1.3. Le tag <html:button>

Ce tag génère un tag HTML <input> de type button.

Il possède de nombreux attributs dont les principaux sont :

Attribut	Rôle
alt	correspond à l'attribut alt du tag HTML
altKey	clé du ResourceBundle dont la valeur sera affectée à l'attribut alt du tag HTML
bundle	nom du bean qui encapsule le ResourceBundle (utilisé lorsque plusieurs ResourceBundles sont définis)
disabled	true pour rendre le bouton non opérationnel
property	nom du paramètre dans la requête http lors de la soumission du formulaire : correspond à l'attribut name du tag HTML
title	correspond à l'attribut title du tag HTML
titleKey	clé du ResourceBundle dont la valeur sera affectée à l'attribut title du tag HTML
value	libellé du bouton : correspond à l'attribut value du tag HTML

Ce tag doit être inclus dans un tag <html:form>

Exemple :

<html:button property="valider" value="Valider" title="Valider les données" />

Résultat

<input type="button" name="valider" value="Valider" title="Valider les données">

68.4.1.4. Le tag <html:cancel>

Ce tag génère un tag HTML <input> de type button avec une valeur spécifique pour permettre d'identifier ce bouton comme étant celui de type "Cancel".

Il possède des attributs similaires au tag <html:button>.

Il n'est pas recommandé d'utiliser l'attribut property : il faut laisser la valeur par défaut de Struts pour lui permettre d'identifier ce bouton. La valeur par défaut de l'attribut property permet à Struts de déterminer la valeur de retour de la méthode Action.isCancelled.

Exemple :

<html:cancel />

Résultat

<input type="submit" name="org.apache.struts.taglib.html.CANCEL" value="Cancel" onclick="bCancel=true;">

68.4.1.5. Le tag <html:submit>

Ce tag génère un tag HTML <input type="submit"> permettant la validation d'un formulaire.

Il possède des attributs similaires au tag <html:button>.

Ce tag doit être inclus dans un tag <html:form>

Exemple :

```
<html:submit />
```

Résultat

```
<input type="submit" value="Submit">
```

68.4.1.6. Le tag <html:radio>

Ce tag génère un tag HTML <input type="radio"> permettant d'afficher un bouton radio.

Exemple :

```
<html:radio property="sexe" value="femme" />Femme<br>
<html:radio property="sexe" value="homme" />Homme<br>
```

Résultat :

```
<input type="radio" name="sexe" value="femme">Femme<br>
<input type="radio" name="sexe" value="homme">Homme<br>
```

68.4.1.7. Le tag <html:checkbox>

Ce tag génère un tag HTML <input type="checkbox"> permettant d'afficher un bouton de type case à cocher.

Exemple :

```
<html:checkbox property="caseACocher"> Une case à cocher</html:checkbox>
```

Résultat :

```
<input type="checkbox" name="caseACocher" value="on">Une case à cocher
```

68.4.2. La bibliothèque de tags Bean

Cette bibliothèque fournit des tags pour faciliter la gestion et l'utilisation des javabeans.

Pour utiliser cette bibliothèque, il faut, comme pour toute bibliothèque de tags personnalisés, réaliser plusieurs opérations :

1. copier le fichier struts-bean.tld dans le répertoire WEB-INF de la webapp
2. configurer le fichier WEB-INF/web.xml pour déclarer la bibliothèque de tag

```
<taglib>
  <taglib-uri>struts-bean.tld</taglib-uri>
  <taglib-location>/WEB-INF/struts-bean.tld</taglib-location>
</taglib>
```

3. ajouter dans chaque page JSP qui va utiliser cette bibliothèque un tag de directive taglib précisant l'utilisation de la bibliothèque

```
<%@ taglib uri="struts-bean.tld" prefix="bean" %>
```

68.4.2.1. Le tag <bean:cookie>

Le tag <bean:cookie> permet d'obtenir la ou les valeurs d'un cookie.

Il possède plusieurs attributs :

Attribut	Rôle
id	identifiant du cookie
name	nom du cookie
multiple	précise si toutes les valeurs ou seulement la première valeur du cookie sont retournées
value	valeur du cookie; si celui-ci n'existe pas alors il est créé

68.4.2.2. Le tag <bean:define>

Le tag <bean:define> permet de définir une variable.

Il possède plusieurs attributs :

Attribut	Rôle
id	nom de la variable qui va être créée
name	nom du bean qui va fournir la valeur
property	propriété du bean qui va fournir la valeur
scope	portée du bean
toScope	portée de la variable créée
type	type de la variable créée
value	

Exemple :

```
1. <jsp:useBean id="utilisateur" scope="page" class=" com.jmd.test.struts.data.Utilisateur"/>
2. <bean:define id="nomUtilisateur" name="utilisateur" property="nom"/>
3. Bienvenue <%= nomUtilisateur %>
```

Cet exemple permet de définir un bean de type Utilisateur qui est stocké dans la portée page. Une variable nomUtilisateur est définie et initialisée avec la valeur de la propriété nom du bean de type Utilisateur.

68.4.2.3. Le tag <bean:header>

Le tag <bean:header> est similaire au tag <bean:cookie> mais il permet de manipuler des données contenues dans l'en-tête de la requête HTTP.

68.4.2.4. Le tag <bean:include>

Le tag <bean:include> permet d'évaluer et d'inclure le rendu d'une autre page. Son mode de fonctionnement est similaire au tag JSP <jsp:include> excepté que le rendu de la page n'est pas inclus directement dans la page mais dans une variable.

Il possède plusieurs attributs :

Attribut	Rôle
id	nom de la variable créée qui va contenir le résultat de la page. Cette variable sera stockée dans la portée page.
forward	nom d'une redirection globale définie dans le fichier de configuration
href	URL de la page
page	URI relative au contexte de l'application de la page

Exemple :

```
1. <bean:include id="barreNavigation"  page="/navigation.jsp"/>
```

```
2. <bean:write name="barreNavigation"  filter="false" />
```

Ce tag est utile notamment pour obtenir un document XML qu'il sera alors possible de manipuler.

68.4.2.5. Le tag <bean:message>

Le tag <bean:message> permet d'obtenir la valeur d'un libellé contenu dans un ResourceBundle.

Il possède plusieurs attributs :

Attribut	Rôle
arg0	valeur du premier paramètre de remplacement
arg1	valeur du second paramètre de remplacement
arg2	valeur du troisième paramètre de remplacement
arg3	valeur du quatrième paramètre de remplacement
arg4	valeur du cinquième paramètre de remplacement
bundle	nom du bean qui encapsule le ResourceBundle (utilisé lorsque que plusieurs ResourceBundle sont définis)
key	clé du libellé à obtenir
locale	nom du bean qui stocke la Locale dans la session
name	nom du bean qui encapsule les données
property	propriété du bean précisé par l'attribut name contenant la valeur du libellé
scope	portée du bean précisé par l'attribut name

68.4.2.6. Le tag <bean:page>

Le tag <bean:page> permet d'obtenir une variable implicite définie par l'API JSP contenue dans la portée page.

Il possède plusieurs attributs :

Attribut	Rôle
id	nom de la variable à créer
property	variable implicite à extraire. Les valeurs possibles sont : application, config, request, response ou session

68.4.2.7. Le tag <bean:param>

Le tag <bean:param> est similaire au tag <bean:cookie> mais il permet de manipuler des données contenues dans les paramètres de la requête HTTP.

68.4.2.8. Le tag <bean:resource>

Le tag <bean:resource> permet d'obtenir la valeur d'une ressource sous la forme d'un objet de type java.io.InputStream ou String.

Il possède plusieurs attributs :

Attribut	Rôle
id	nom de la variable à créer
name	URI de la ressource relative à l'application à utiliser
input	permet d'obtenir la ressource sous la forme d'un objet de type java.io.InputStream. Sinon c'est un objet de type String qui est retourné

68.4.2.9. Le tag <bean:size>

Le tag <bean:size> permet d'obtenir le nombre d'éléments d'une collection ou d'un tableau. Ce tag crée une variable de type java.lang.Integer.

Il possède plusieurs attributs :

Attribut	Rôle
id	nom de la variable à créer
collection	expression renvoyant la collection à traiter
name	nom du bean qui encapsule la collection
property	propriété du bean qui encapsule la collection
scope	portée du bean

Exemple :

```
1. <bean:size id="count" name="elements" />
```

68.4.2.10. Le tag <bean:struts>

Le tag <bean:struts> permet de copier un objet Struts (FormBean, Mapping, Forward) dans une variable.

Il possède plusieurs attributs :

Attribut	Rôle
id	nom de la variable à créer (attribut obligatoire)
formBean	nom du bean de type ActionForm
forward	nom de l'objet global de type ActionForward
mapping	nom de l'objet de type ActionMapping

68.4.2.11. Le tag <bean:write>

Le tag <bean:write> permet d'envoyer dans le JspWrite courant la valeur d'un bean ou d'une propriété d'un bean.

Il possède plusieurs attributs :

Attribut	Rôle
bundle	nom du bean qui encapsule le ResourceBundle (utilisé lorsque plusieurs ResourceBundle sont définis)
filter	la valeur true permet de remplacer les caractères spécifiques d'HTML par leur entité correspondante
format	format de conversion en chaîne de caractères
formatKey	clé du ResourceBundle qui précise le format de conversion en chaîne de caractères
ignore	la valeur true permet d'ignorer l'inexistence du bean dans la portée. La valeur false lève une exception si le bean n'est pas trouvé dans la portée. Par défaut, false
locale	nom du bean qui stocke la Locale dans la session
name	nom du bean qui encapsule les données (attribut obligatoire)
property	propriété du bean
scope	portée du bean

Exemple :

```
1. <jsp:useBean id="utilisateur" scope="page" class=" com.jmd.test.struts.data.Utilisateur"/>
2. <bean:write name="utilisateur" property="nom"/>
```

L'attribut format du tag permet de formater les données restituées par le bean.

Exemple :

```
1. <p><bean:write  name="monbean" property="date" format="dd/MM/yyyy  HH:mm"/></p>
```

L'attribut formatKey du tag permet de formater les données restituées par le bean à partir d'une clé des ResourceBundle : ceci permet d'internationaliser le formatage.

Exemple :

```
1. <p><bean:write  name="monbean" property="date"  formatKey="date.format"/></p>
```

Dans le fichier ApplicationResources.properties

date.format=dd/MM/yyyy HH:mm

Dans le fichier ApplicationResources_en.properties

date.format=MM/dd/yyyy HH:mm

Il est important que le format précisé soit compatible avec la Locale courante sinon une exception est levée

Exemple :

```
1. javax.servlet.jsp.JspException: Wrong format string: '#.##0,00'
2.            at org.apache.struts.taglib.bean.WriteTag.formatValue(WriteTag.java:376)
3.            at org.apache.struts.taglib.bean.WriteTag.doStartTag(WriteTag.java:292)
```

68.4.3. La bibliothèque de tags Logic

Cette bibliothèque fournie des tags pour faciliter l'utilisation de logiques de traitements pour l'affichage des pages.

Pour utiliser cette bibliothèque, il faut, comme pour toute bibliothèque de tags personnalisés, réaliser plusieurs opérations :

1. copier le fichier struts-logic.tld dans le répertoire WEB-INF de la webapp
2. configurer le fichier WEB-INF/web.xml pour déclarer la bibliothèque de tags

```
<taglib>
 <taglib-uri>struts-logic.tld</taglib-uri>
 <taglib-location>/WEB-INF/struts-logic.tld</taglib-location>
</taglib>
```

3. ajouter dans chaque page JSP qui va utiliser cette bibliothèque un tag de directive taglib précisant l'utilisation de la bibliothèque

```
<%@ taglib uri="struts-logic.tld" prefix="logic" %>
```

La plupart de ces tags encapsulent des tags de conditionnement des traitements ou d'exécution d'opérations sur le flot des traitements.

L'utilisation de ces tags évite l'utilisation de code Java dans les JSP.

Exemple :

```
01. <jsp:useBean id="elements"  scope="request" class="java.util.List" />
02.   ...
03. <%
04.   for (int i = 0; i  < elements.size(); i++)
05.   {
06.     MonElement monElement =  (MonElement)elements.get(i);
07. %>
08. <%=monElement.getLibelle()%>
09. <%
10.   }
11. %>
```

Tout le code Java peut être remplacé par l'utilisation de tag de la bibliothèque struts-logic.

Exemple :

```
1. <jsp:useBean  id="elements" scope="request" class="java.util.List" />
2. <logic:iterate id="monElement" name="elements" type="com.jmd.test.struts.data.MonElement">
```

```
  <bean:write name="monElement" property="libelle"/>
</logic:iterate>
```

Cette bibliothèque définit une quinzaine de tags.

Dans différents exemples de cette section, le bean suivant sera utilisé

Exemple :

```
package test.struts.data;

import java.util.Date;

public  class MonBean {
  private String libelle;
  private Integer valeur;
  private Date date;

  public MonBean() {
    libelle="libelle de test";
    valeur = new Integer(123456);
    date = new Date();
  }

  public void setLibelle(String libelle) {
    this.libelle = libelle;
  }

  public String getLibelle() {
    return libelle;
  }

  public void setDate(Date date) {
    this.date = date;
  }

  public Date getDate() {
    return date;
  }

  public void  setValeur(Integer valeur) {
    this.valeur =  valeur;
  }

  public Integer  getValeur() {
    return valeur;
  }
}
```

L'intérêt de cette bibliothèque a largement diminué depuis le développement de la JSTL qui intègre en standard des fonctionnalités équivalentes. Il est d'ailleurs fortement recommandé d'utiliser dès que possible les tags de la JSTL à la place des tags de Struts.

68.4.3.1. Les tags <logic:empty> et <logic:notEmpty>

Le tag <logic:empty> permet de tester si une variable est null ou vide. Le tag <logic:notEmpty> permet de faire le test opposé.

Il possède plusieurs attributs :

Attribut	Rôle
name	nom de la variable à tester si l'attribut property n'est pas précisé sinon c'est le nom de l'entité à tester (attribut obligatoire)
property	Nom de la propriété de la variable à tester
scope	Portée contenant la variable

68.4.3.2. Les tags <logic:equal> et <logic:notEqual>

Le tag <logic:equal> permet de tester l'égalité entre une variable et une valeur. Le tag <logic:notEqual> permet de faire le test opposé.

Ils possèdent plusieurs attributs :

Attribut	Rôle
	contient la valeur : celle-ci peut être une constante ou être déterminée dynamiquement par exemple avec le tag JSP <%= ... %> (attribut

Value	obligatoire)
cookie	nom du cookie dont la valeur doit être testée
header	nom de l'attribut de l'en-tête http dont la valeur doit être testée
name	nom de la variable dont la valeur doit être testée
property	nom de la propriété de la variable à tester
parameter	nom du paramètre http dont la valeur doit être testée
scope	portée contenant la variable

Exemple :

```
1.   <% int valeurReference = 123456; %>
2. ...
3.   <logic:equal name="monbean"
4.   property="valeur"
5.   value="<%= valeurReference %>">
6.   <p>La valeur est égale</p>
7.   </logic:equal>
```

68.4.3.3. Les tags <logic:lessEqual>, <logic:lessThan>, <logic:greaterEqual>, et <logic:greaterThan>

Ils sont similaires au tag <logic:equal> mais permettent respectivement de tester les conditions inférieur ou égal, strictement inférieur, supérieur ou égal et strictement supérieur.

68.4.3.4. Les tags <logic:match> et <logic:notMatch>

Le tag <logic:match> permet de tester si une valeur est contenue dans une variable. Le tag <logic:notMatch> permet de faire le test opposé.

Ils possèdent plusieurs attributs :

Attribut	Rôle
value	contient la valeur : celle-ci peut être une constante ou déterminée dynamiquement par exemple avec le tag JSP <%= ... %> (attribut obligatoire)
cookie	nom du cookie dont la valeur doit être testée
header	nom de l'attribut de l'en-tête http dont la valeur doit être testée
name	nom de la variable dont la valeur doit être testée
property	nom de la propriété de la variable à tester
parameter	nom du paramètre http dont la valeur doit être testée
scope	portée contenant la variable
location	permet de préciser la localisation de la valeur à rechercher dans la variable. Les valeurs possibles sont start et end pour une recherche respectivement en début et en fin. Sans préciser cet attribut, la recherche se fait dans toute la variable

68.4.3.5. Les tags <logic:present> et <logic:notPresent>

Le tag <logic:present> permet de tester l'existence d'une entité dans une portée donnée. Le tag <logic:notPresent> permet de faire le test opposé.

Ils possèdent plusieurs attributs :

Attribut	Rôle
cookie	nom du cookie dont la valeur doit être testée
header	nom de l'attribut de l'en-tête http dont la valeur doit être testée
name	nom de la variable dont la valeur doit être testée

property	nom de la propriété de la variable à tester
parameter	nom du paramètre http dont la valeur doit être testée
scope	portée contenant la variable

68.4.3.6. Le tag <logic:forward>

Le tag <logic:forward> permet de transférer le traitement de la requête vers une page définie dans les redirections globales de l'application.

Il ne possède qu'un seul attribut name qui permet de préciser le nom de la redirection globale définie dans le fichier de configuration struts-config.xml

Exemple : dans une JSP

```
1. <logic:forward name=">strong<login>/strong<" />
```

Exemple : dans le fichier de configuration

```
1. <global-forwards>
2.   <forward name=">strong<login>/strong<" path="/login.jsp"/>
3. </global-forwards>
```

68.4.3.7. Le tag <logic:redirect>

Le tag <logic:redirect> permet de rediriger l'affichage vers une autre page en utilisant la méthode HttpServletResponse.sendRedirect().

Il possède plusieurs attributs :

Attribut	Rôle
forward	nom de la redirection globale définie dans le fichier de configuration struts-config.xml à utiliser
href	URL de la ressource à utiliser
page	URL de la ressource relative au contexte de l'application (doit obligatoirement commencer par /)
name	collection de type Map qui contient les paramètres à passer à la ressource
paramId	nom de l'unique paramètre passé à la ressource
paramName	nom d'une variable dont la valeur sera utilisée comme valeur du paramètre
paramProperty	propriété de la variable paramName dont la valeur sera utilisée comme valeur du paramètre

L'avantage de ce tag est de permettre de modifier les paramètres fournis à la ressource.

68.4.3.8. Le tag <logic:iterate>

Ce tag permet de réaliser une itération sur une collection d'objets. Le corps du tag sera évalué pour chaque occurrence de l'itération.

Il possède plusieurs attributs :

Attribut	Rôle
id	nom de la variable qui va contenir l'occurrence courante de l'itération (attribut obligatoire)
name	nom de la variable qui contient la collection à parcourir
property	nom de la propriété de la variable name qui contient la collection à parcourir
scope	portée de la variable qui contient la collection
type	type pleinement qualifié des occurrences de la collection
indexId	nom de la variable qui va contenir l'index de l'occurrence courante

length	nombre maximum d'occurrences à traiter. Par défaut toute la collection est parcourue
offset	index de la première occurrence de l'itération. Par défaut c'est la première occurrence de la collection

68.5. La validation de données

La méthode validate() de la classe ActionForm permet de réaliser une validation des données fournies dans la requête.

Elle est appelée par l'ActionServlet lorsque l'attribut validate est positionné à true dans le tag <action>.

Exemple :

Exemple :

```
<action  path="/validerproduit"
         type="test.struts.controleur.ValiderProduitAction"
        name="saisirProduitForm"
        validate="true">
  <forward name="succes"  path="/listeproduit.jsp"/>
  <forward name="echec"  path="/saisirproduit.jsp"/>
</action>
```

Pour définir ses propres validations, il faut redéfinir la méthode validate() pour y coder les règles de validation. Si une erreur est détectée lors de l'exécution de ces règles, il faut instancier un objet de type ActionError et l'ajouter à l'objet ActionErrors retourné par la méthode validate(). Cet ajout se fait en utilisant la méthode add().

68.5.1. La classe ActionError

Cette classe encapsule une erreur survenue lors de la validation des données. C'est dans la méthode validate() de la classe ActionForm que les traitements doivent créer des instances de cette classe.

Le constructeur de cette classe attend en paramètre une chaîne de caractères qui précise le nom d'une clé du message d'erreur correspondant au message de l'erreur défini dans le fichier ressource bundle de l'application.

La méthode validate() de la classe ActionForm possède deux surcharges :

public ActionErrors validate(ActionMapping mapping, javax.servlet.http.HttpServletRequest request)

public ActionErrors validate(ActionMapping mapping, javax.servlet.ServletRequest request)

La première version est essentiellement mise en oeuvre car elle est utilisée pour les applications web.

Elle renvoie un objet de type ActionErrors qui va contenir les éventuelles erreurs détectées lors de la validation. Si la collection est vide ou nulle cela précise que la validation a réussie. Ceci permet à l'ActionServlet de savoir si elle va pouvoir appeler la méthode execute() de l'Action.

Par défaut, la méthode validate() de la classe ActionForm renvoie systématiquement null. Il est donc nécessaire de sous-classer la classe ActionForm et de redéfinir la méthode validate().

Exemple :

Exemple :

```
public ActionErrors validate(ActionMapping  mapping, HttpServletRequest request) {
  ActionErrors errors = new ActionErrors();
  if ((nomUtilisateur == null) ||  (nomUtilisateur.length() == 0))
    errors.add("nomUtilisateur",  new ActionError("erreur.nomutilisateur.obligatoire"));
  if ((mdpUtilisateur == null) ||  (mdpUtilisateur.length() == 0))
    errors.add("mdpUtilisateur",  new ActionError("erreur.mdputilisateur.obligatoire"));
  return errors;
}
```

Ce mécanisme peut aussi être mis en oeuvre dans la méthode execute() de la classe Action.

68.5.2. La classe ActionErrors

Cette classe encapsule une collection de type HashMap d'objets ActionError générés lors d'une validation.

C'est la méthode validate() de la classe ActionForm qui renvoie une instance de cette classe. Les traitements qu'elle contient se chargent de créer une instance de cette classe et d'utiliser la méthode add() pour ajouter des instances de la classe ActionError pour chaque erreur rencontrée.

Il est aussi possible de définir des erreurs dans la méthode : il faut créer un objet de type ActionErrors, utiliser sa méthode add() pour chaque erreur à ajouter et appeler la méthode saveErrors de la classe Action pour sauvegarder les erreurs.

Exemple :

```
01. public ActionForward execute(
02.   ActionMapping       mapping,
03.   ActionForm          form,
04.   HttpServletRequest  request,
05.   HttpServletResponse response) throws  Exception {
06.   DynaActionForm daf       = (DynaActionForm) form;
07.   ActionForward  resultat  =  mapping.findForward("succes");
08.   String         reference = (String)  daf.get("reference");
09.   String         libelle   = (String) daf.get("libelle");
10.   int            prix      = Integer.parseInt((String)  daf.get("prix"));
11.
12.   System.out.println("reference=" +  reference);
13.   System.out.println("libelle=" +  libelle);
14.   System.out.println("prix=" +  prix);
15.
16.   if ((reference == null) ||  (reference.equals(""))) {
17.     ActionErrors errors = new ActionErrors();
18.     errors.add("reference", new  ActionError("app.saisirproduit.erreur.reference"));
19.     saveErrors(request, errors);
20.
21.     resultat =  mapping.findForward("echec");
22.   }
23.   return resultat;
24. }
```

Remarque : dans cet exemple, la validation des données est effectuée dans la méthode execute. Il est préférable d'effectuer cette tâche grâce à une des fonctionnalités proposées par Struts (validation par l'ActionForm ou le plug-in Validator).

68.5.3. L'affichage des messages d'erreur

Le tag <html:errors> permet d'afficher les erreurs contenues dans l'instance courante de la classe ActionErrors.

Exemple :

```
1. <html:errors/>
```

Le plus simple est d'utiliser ce tag en début du corps de la page. Il se charge d'afficher toutes les erreurs (les erreurs globales et celles dédiées à un élément du formulaire) pour permettre leur gestion de façon globale.

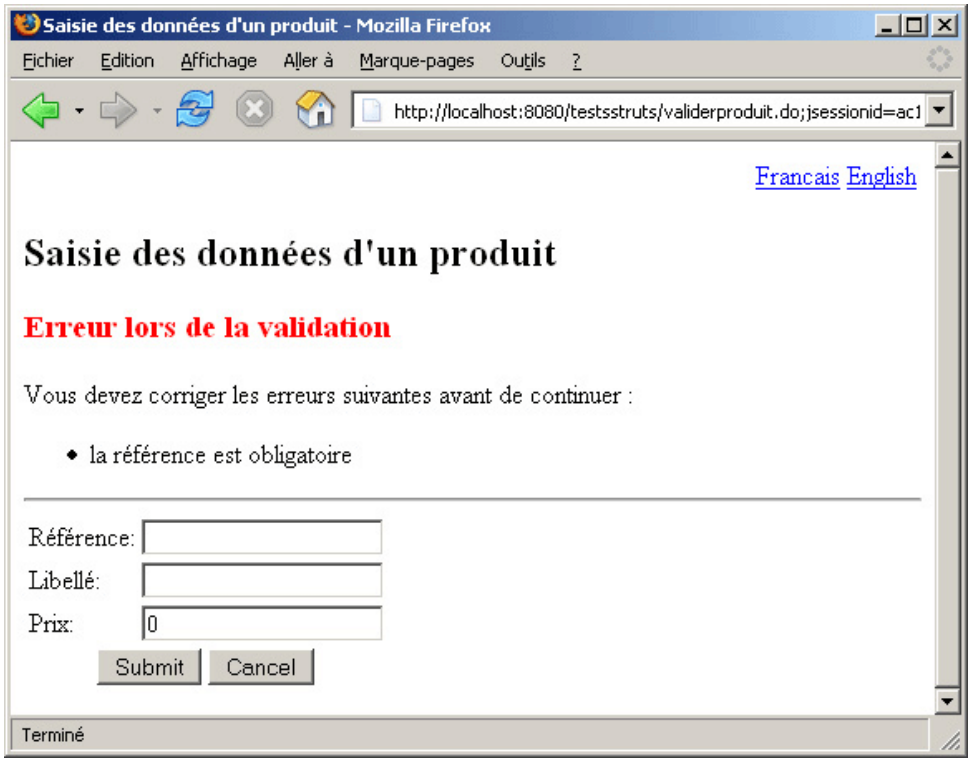
Ce tag recherche dans les ResourceBundle les deux clés errors.header et errors.footer dont les valeurs seront affichées avant les messages. A partir de la version 1.1 de Struts, les clés errors.prefix et error.suffix sont recherchées dans les ResourceBundles et ajoutées respectivement avant et après chaque message.

Exemple :

```
1. errors.prefix=<li>
2. errors.suffix=</li>
3. errors.header=<h3><font  color\="red">Erreur lors de la  validation</font></h3>
4. Vous devez corriger les erreurs suivantes avant de continuer \:<ul>
5. errors.footer=</ul><hr>
```

L'utilisation de tags HTML dans les ResourceBundle peut paraître choquante mais c'est la solution utilisée par Struts.

Exemple :



Avec Struts 1.1, il est aussi possible d'utiliser le tag <html:errors> pour afficher des messages d'erreurs liés à un composant du formulaire. Dans ce cas, l'approche est légèrement différente.

L'exemple ci-dessous va afficher un message personnalisé pour un composant et un message d'erreur général.

Exemple : ApplicationResources.properties

```
...
app.saisirproduit.erreur.reference=la référence saisie est erronée
app.saisirproduit.erreur.libelle=le libelle saisi est erroné
app.saisirproduit.erreur.globale=une ou plusieurs erreurs sont survenues

errors.prefix=
errors.suffix=
errors.header=
errors.footer=
...
```

Comme les clés préfixées par errors sont utilisées pour chaque affichage d'erreur, leur contenu est laissé vide.

L'action instancie des objets de type ActionError si une erreur est détectée sur les données et l'associe au composant correspondant. Lors de l'ajout d'une erreur, il faut préciser l'identifiant du composant correspondant à son attribut property dans le tag de la page.

Si au moins une erreur est détectée sur une donnée alors une erreur globale est ajoutée à la liste des erreurs. Pour cela, il faut utiliser la constante ActionErrors.GLOBAL_ERROR lors de l'ajout de l'erreur dans la collection ActionErrors.

Exemple Struts 1.1 :

```
...
public  ActionForward execute(
   ActionMapping       mapping,
   ActionForm          form,
   HttpServletRequest  request,
   HttpServletResponse response) throws Exception {
  DynaActionForm daf       = (DynaActionForm) form;
  ActionForward  resultat  = mapping.findForward("succes");
  ActionErrors   errors    = new ActionErrors();
  String         reference = (String) daf.get("reference");
  String         libelle   = (String) daf.get("libelle");
  int            prix      = Integer.parseInt((String)  daf.get("prix"));

  if (reference.equals("test")) {
    errors.add("reference", new  ActionError("app.saisirproduit.erreur.reference"));
  }
  if (libelle.equals("test")) {
    errors.add("libelle", new  ActionError("app.saisirproduit.erreur.libelle"));
  }

  if (!errors.isEmpty())
  {
    errors.add(ActionErrors.GLOBAL_ERROR,
      new  ActionError("app.saisirproduit.erreur.globale"));
    saveErrors(request, errors);
    resultat = mapping.findForward("echec");
```

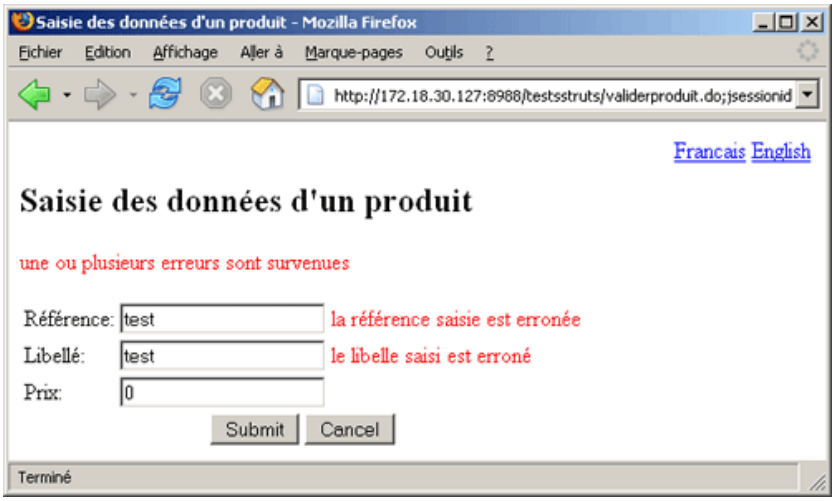
```
  }

  return resultat;
}
...
```

Il ne reste plus qu'à assurer l'affichage des messages d'erreurs dans la page. Pour le message associé à un composant il faut utiliser l'attribut property du tag <html:errors> en précisant comme valeur le nom du composant dont les messages doivent être affichés.

Pour afficher les messages d'erreurs globaux, il faut préciser dans l'attribut property la valeur de la constante ActionErrors.GLOBAL_ERROR.

Exemple Struts 1.1 :

```
<%@ taglib  uri="/WEB-INF/struts-bean.tld" prefix="bean"%>
<%@ taglib  uri="/WEB-INF/struts-html.tld" prefix="html"%>
<%@ taglib  uri="/WEB-INF/struts-logic.tld" prefix="logic"%>
<%@ page  contentType="text/html;charset=windows-1252"%>
<%@ page  import  ="org.apache.struts.action.*"    %>
<html:html locale="true">
  <head>
    <meta http-equiv="Content-Type" content="text/html;  charset=windows-1252"/>
    <title>
      <bean:message key="app.saisirproduit.titre"/>
    </title>
  </head>
  <body>
    <table width="100%">
      <tr>
        <td align="right">
          <html:link  href="changerlangue.do?langue=fr">Francais</html:link>
          <html:link  href="changerlangue.do?langue=en">English</html:link>
        </td>
      </tr>
    </table>
    <h2>
      <bean:message key="app.saisirproduit.titre"/>
    </h2>
    <html:form action="validerproduit.do"  focusIndex="reference">
      <logic:present name="<%=Action.ERROR_KEY%>">
      <P style="color:red;"><html:errors  property="<%=ActionErrors.GLOBAL_ERROR%>"/></P>
      </logic:present>
       <table>
        <tr>
          <td>
            <bean:message  key="app.saisirproduit.libelle.reference"/>:
         </td>
          <td>
            <html:text property="reference"/>
          </td>
          <td style="color:red;"><html:errors  property="reference"/></td>
        </tr>
        <tr>
          <td>
            <bean:message key="app.saisirproduit.libelle.libelle"/>:
          </td>
          <td>
            <html:text property="libelle"/>
          </td>
          <td style="color:red;"><html:errors  property="libelle"/></td>
        </tr>
        <tr>
          <td>
            <bean:message key="app.saisirproduit.libelle.prix"/>:
          </td>
          <td>
            <html:text property="prix"/>
          </td>
          <td></td>
        </tr>
        <tr>
          <td colspan="3" align="center">
            <html:submit/>
            <html:cancel/>
          </td>
        </tr>
      </table>
    </html:form>
  </body>
</html:html>
```



68.5.4. Les classes ActionMessage et ActionMessages

La classe ActionMessage, apparue avec Struts 1.1, fonctionne de la même façon que la classe ActionError mais elle encapsule des messages d'information qui ne sont pas des erreurs.

Ce type de message est pratique notamment pour afficher des messages de confirmation ou d'information aux utilisateurs.

La classe ActionMessages encapsule une collection d'ActionMessage.

Exemple :

```
1. ActionMessages  actionMessages = new ActionMessages();
2. actionMessages.add(ActionMessages.GLOBAL_MESSAGE,
3.   new  ActionMessage("liste.incomplete"));
4. saveMessages(request,actionMessages);
```

La méthode add() permet d'ajouter des messages dans la collection.

La méthode clear() permet de supprimer tous les messages de la collections.

La méthode isEmpty() permet de savoir si la collection est vide et la méthode size() permet de connaître le nombre de messages stockés dans la collection.

68.5.5. L'affichage des messages

Le tag <html:messages> permet d'afficher les messages contenus dans l'instance courante de la classe ActionMessages.

Exemple :

```
1. <logic:messagesPresent message="true">
2.  <html:messages  id="message" message="true">
3.    <bean:write  name="message"/>
4.   </html:messages>
5. </logic:messagesPresent>
```

La suite de ce chapitre sera développé dans une version future de ce document