A

Architecture logicielle MVC

Exemple d'application multi-présentation

- Vente de produits en ligne
  - Panier
  - Facture HTML
  - Facture PDF

Architecture initiale

- Partie Facturation
- Pas de séparation Présentation-Contenu
  - Une servlet pour la facture (aurait pu être une page JSP)
  - Une servlet pour la facture PDF

Facture

```
HttpSession session = request.getSession(true) ;
Vector panier = (Vector) session.getAttribute("panier") ;
        ......
for (int i=0; i<panier.size(); i++) {
       int refId = ((Integer) panier.get(i)).intValue() ;

       ResultSet result = stmt.executeQuery("select * from produits where id="+refId) ;
       result.next() ;

       String description = result.getString(2) ;
       int prix = result.getInt(3) ;

       out.println("<tr>") ;
       out.println("<td>  " + description +"  </td><td>" + prix + "</td>") ;
       prixTotal += prix ;
       out.println("</tr>") ;
}
            ....
out.println("<h3>Prix total : " + prixTotal + " Euros</h2>") ;
```

Facture PDF

```
 HttpSession session = request.getSession(true) ;
 Vector panier = (Vector) session.getAttribute("panier") ;

 Paragraph titre = new Paragraph(...) ;
...
  int prixTotal = 0 ;
  PdfPTable table = new PdfPTable(2) ;

  table.addCell(new Paragraph(...)) ;
.....
 for (int i=0; i<panier.size(); i++) {
 int refId = ((Integer) panier.get(i)).intValue() ;

 ResultSet result = stmt.executeQuery("select * from produits where id="+refId) ;
 result.next() ;

 String description = result.getString(2) ;
 int prix = result.getInt(3) ;

 table.addCell(" " + description+" ") ;
 table.addCell(" " + prix+ " Euros") ;
 prixTotal += prix ;
```

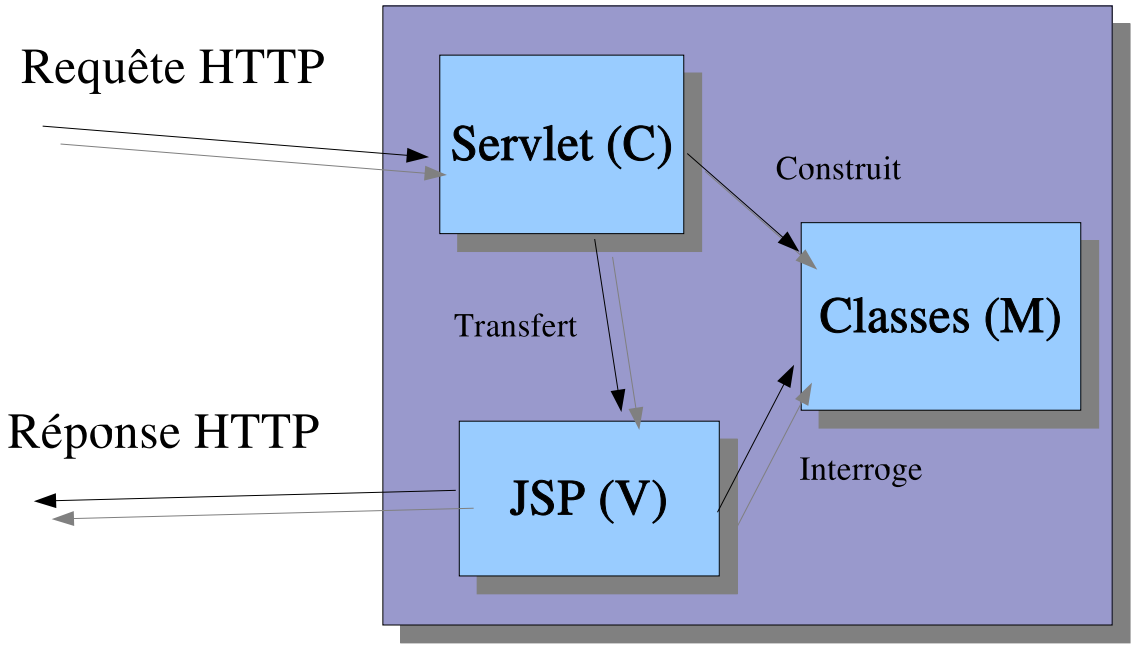
Servlet (2)

- Mélange des trois aspects : analyse, calcul, présentation
- Complexité du code
- Duplication :
  - des accès à la base de données
  - de la logique (calcul de la somme totale).
- Difficulté de maintenance
- Peu de possibilités d'évolution (en particulier pour la présentation)

Architecture MVC

- Séparation des trois aspects en unités (classes ?) distinctes
  - Pouvoir modifier un des aspects sans toucher aux autres
- **M**odèle, **V**ue, **C**ontrôle :
  - *Contrôle* :
    - décodage du formulaire,
    - mise en place du calcul.
    - mise en place de la présentation.
  - *Modèle* : calcul
  - *Vue* : présentation du résultat.

Architecture MVC (2)



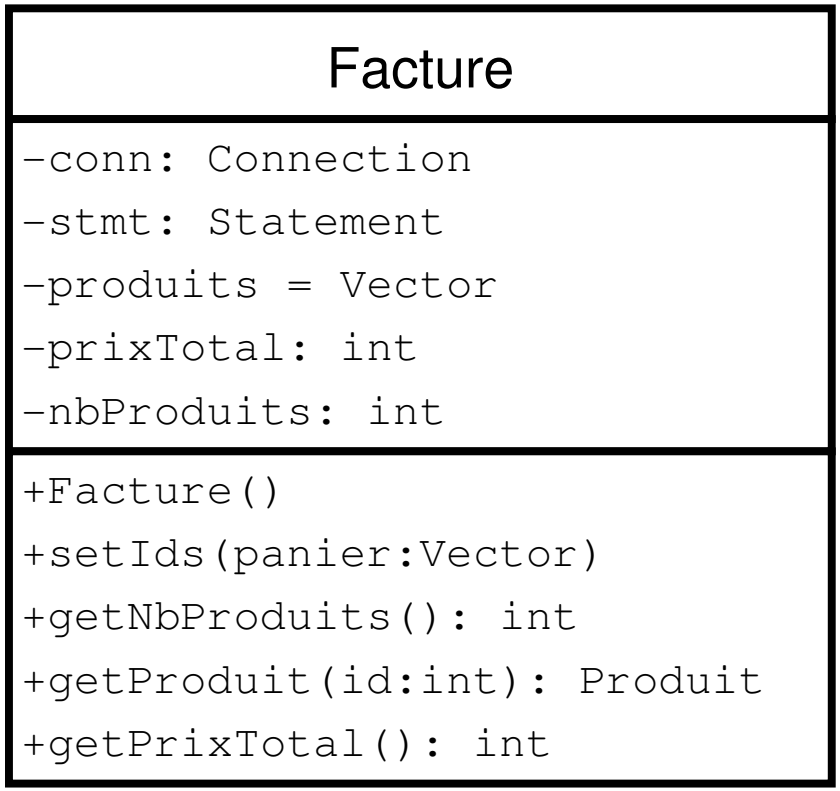
Contrôle

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
    throws IOException, ServletException
{
    //-------------------------------------------------------------------
    //  Analyse de la requête HTTP
    //-------------------------------------------------------------------
        matiere = request.getParameter("matiere") ;
        .....
    //-------------------------------------------------------------------
    // Construction du modèle
    //-------------------------------------------------------------------
         ......
    //-------------------------------------------------------------------
    // Génération de la présentation
    //-------------------------------------------------------------------
        try {
            getServletContext().
                getRequestDispatcher("/tableau.jsp").forward(request, response);
        } catch (Exception e)
            {
                response.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR,
                                   "erreur à la création de la présentation tableau") ;
                return ;
            }
}
```

Modèle

- Ensemble de classes Java :
  - Encapsulent l'ensemble des traitements (accès BD . . .)
  - Objets construits par le contrôle.
  - Interrogés par la vue.
- Variable de session ou de requête

Exemple de modèle



Construction du modèle (variable de session)

```
Facture facture = null ;
HttpSession session = request.getSession(true) ;
facture = (Facture) session.getAttribute("facture") ;
if (facture == null)
  // on cree le modele et on l'enregistre dans la session courante
   {
      facture = new Facture() ;
      session.setAttribute("facture", facture) ;
   }
```

Construction du modèle (variable de requête)

```
Facture facture = new Facture() ;
request.setAttribute("facture", facture) ;
```

Construction du modèle (2)

```
try {
   facture.setIds(panier) ;
} catch (SQLException e)
   {
     response.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR,
           e.getMessage()) ;
   }
```

Vue

```
<html>
    <body>
       <jsp:useBean id="facture" scope="session" class="model.Facture" />
       ....
        <table border>
        <tr><th>Produit</th><th>Prix</th></tr>
        <% for (int i=0; i<facture.getNbProduits(); i++)
        { %>
        <tr><td><%=facture.getProduit(i).getDescription()%></td>
              <td><%=facture.getProduit(i).getPrix()%></td>
        </tr>
     <%}%>
        </table>
        </center>

        <p>
        Prix total : <%=facture.getPrixTotal()%>
    </body>
</html>
```

Remarque

- Mélange de code et de balises HTML
- Rappel :
  - JSP = écriture de balises
  - Génération automatique du code java de la servlet

  $\Rightarrow$ écrire le moins de code soi-même.
  - Remarque : toute la partie traitement et SQL n'est déjà plus là.

Tags JSP

- Récupération du modèle depuis la session :

```
HttpSession session = request.getSession(true) ;
Facture facture = (Facture) session.getAttribute("facture") ;
```

Devient :

```
<jsp:useBean id="facture" class="model.Facture" scope="session"/>
```

Tags JSP (2)

- Récupération du modèle depuis la requête :

```
Facture facture = (Facture) request.getAttribute("facture") ;
```

Devient :

```
<jsp:useBean id="facture" class="model.Facture" scope="request"/>
```

Tags JSP (3)

- Utilisation des java beans
- Accès aux attributs de l'objet :

```
<%= facture.getNbProduits() %>
```

Devient :

```
<jsp:getProperty name="facture" property="nbProduits"/>
```

- Possibilité d'utiliser des bibliothèques de Tags.

Conclusion

- Séparation claire des rôles
- Comparaison avec XML - XSLT :
  - XML : structure des données = structure du fichier XML
  - MVC : structure des données = structure des classes Java

B

Struts

Motivations

- Développement d'une application MVC :
  - Architecture nécessaire mais "un peu" lourde
  - Aide à la mise en oeuvre.

  $\Rightarrow$ Support d'un Framework

Modèle

- Modèle : très fortement couplé aux données
- Fait très souvent appel à une base de données.
- Ensemble de classes Java reflètant la structure de données :
  - Note, Etudiant, ...
- Pont entre ces classes et leur représentation dans le base de données :
  - NoteDAO, EtudiantDAO, ...
- Framework de mapping Object $\leftrightarrow$ Relationnel (Hibernate, iBatis etc)

Vue

- Utilisation de mécanismes communs à un grand nombre d'applications
  - Accès à des variables de session ou de requête
  - Accès aux attributs de ces variables
  - Parcours de listes
  - etc

$\Rightarrow$ Utilisation de bibliothèques de Tags :
  - JSTL (JavaServer Pages Standard Tag Library)
  - `http://java.sun.com/products/jsp/jstl/index.jsp`

Motivations (suite)

- Vérification des champs d'un formulaire
- Gestion des valeurs dans les zones de saisies
  - Affichage d'un formulaire pour mise à jour de données.
  - Ré-affichage d'un formulaire suite à une erreur de saisie.

$\Rightarrow$ Struts.

B1 - Concepts

Présentation générale

- Struts = Servlet :
  - ensemble de jars à déposer dans `WEB-INF/lib`
- Prise en charge d'une partie des URLs (`*.do`)
- Laisse le container gérer les autres ($\neq$ Cocoon)

web.xml

```
</servlet>
  <servlet-name>action</servlet-name>
  <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
  <init-param>
    <param-name>config</param-name>
    <param-value>/WEB-INF/struts-config.xml</param-value>
  </init-param>
</servlet>
<servlet-mapping>
  <servlet-name>action</servlet-name>
  <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

Concepts (suite)

- Deux grandes catégories de classes :
  - **Action :** Contrôleur générique
  - **ActionForm** : Représentation ”objet” :
    - du contenu d’un formulaire.
    - des paramètres d’une requête.
- Un fichier de configuration :
  - Décrit l’enchaînement des pages
  - `struts-config.xml`

B2 - Actions

Première exemple : pas d'action

- Passage par le contrôleur de Struts
- Redirection vers une page JSP
- `struts-config.xml` :

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<!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>
<action-mappings>
    <action path="/test" forward="/hello.jsp"/>
</action-mappings>
</struts-config>
```

Première Action

- Association URL $\Leftrightarrow$ Contrôleur
- Présentation en fonction de la valeur retournée par le Contrôleur
- Contrôleur :
  - Dérive de la classe `Action`
  - Redéfinit la méthode `execute`

MonControleur

```
public class MonControleur extends Action {

    public ActionForward execute(ActionMapping mapping, ActionForm form,
                                         HttpServletRequest request,
                                         HttpServletResponse response)
                  throws Exception {

                Date date = java.util.Calendar.getInstance().getTime() ;

                if (date.getHours() < 12)
                    return mapping.findForward("matin") ;
                else
                    return mapping.findForward("soir") ;
    }
}
```

struts-config.xml

```
<struts-config>

<action-mappings>
    <action path="/go" type="MonControleur">
       <forward name="soir" path="/bonsoir.jsp"/>
       <forward name="matin" path="/bonjour.jsp"/>
    </action>
</action-mappings>

</struts-config>
```

Remarque

- Redéfinition de la méthode execute

$\Rightarrow$ un object par type d'action

- Exemple : gestion d'une boutique
  - Ajout d'un article
  - Suppression d'un article
  - Modification du prix d'un article

  $\Rightarrow$ trois classes : *AddArticle, RemoveArticle, UpdateArticle*

Remarque (suite)

- Eviter la multiplication des classes
- Une classe *Article*
  - Une méthode par type d'action.

  $\Rightarrow$ `DispatchAction`.
- Choix de la méthode :
  - Un paramètre de l'URL.

Exemple

- Un contrôleur avec deux méthodes.
- Le contrôleur forward vers une seule JSP
- Transmission d'une chaîne de caractère entre le controleur et la JSP :
  - Permet d'identifier la méthode par laquelle on est passé.
  - Transmission par la requête.

MonControleur

```
public class MonControleur extends DispatchAction {
  public ActionForward methode1(ActionMapping mapping,
                                ActionForm form,
                                HttpServletRequest request,
                                HttpServletResponse response)
  {
      String data = "Je suis passe par methode 1" ;
      request.setAttribute("data", data) ;
      return mapping.findForward("success") ;
  }

  public ActionForward methode2(ActionMapping mapping,
                                ActionForm form,
                                HttpServletRequest request,
                                HttpServletResponse response)
  {
      String data = "Je suis passe par methode 2" ;
      request.setAttribute("data", data) ;
      return mapping.findForward("success") ;
  }
}
```

La page JSP

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Affichage</title>
    </head>
    <body>

    <h1>Resultat</h1>

    <p>
    <jsp:useBean id="data" scope="request" type="String"/>

    Valeur transmise par la requete : <%= data %>
    </body>
</html>
```

strut-config.xml

- Lien entre le Contrôleur et la JSP

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<!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>

<action-mappings>
    <action path="/go" parameter="choix" type="MonControleur">
       <forward name="success" path="/affichage.jsp"/>
    </action>
</action-mappings>

</struts-config>
```

B3 - Présentation

Concepts

- Simplifier l'écriture des pages JSP
- Minimiser l'écriture de code Java
- Plusieurs aspects :
  - Accès aux variables (session, requête)
  - Parcours de listes
  - Internationalisation

  $\Rightarrow$ *JSTL* (ne fait pas partie de Struts)
  - Gestion des formulaires (valeurs par défaut)

  $\Rightarrow$ *HTML* (Struts)

i18n

- Principe :
  - *Code de langue* : 2 lettres en minuscule (ex: fr, en)
  - *Ressources* : un fichier par langue (suffixé par le code langue) contenant des associations *clé* $\Leftrightarrow$ *texte correspondant*
- Déclaration des ressources :
  - web.xml :

```
<context-param>
      <param-name>javax.servlet.jsp.jstl.fmt.localizationContext</param-name>
      <param-value>MessageResources</param-value>
  </context-param>
```

  - struts-config.xml :

```
<message-resources parameter="MessageResources" null="false"/>
```

- Création des fichiers `MessageResources_fr ...` à la racine des sources

Principe

- MessageRessource_fr :

```
#-- titles --
title.welcome=Essai de I18N

#-- messages
message.welcome=Bonjour
```

- Balises JSP :
  - `<fmt:message key=".."/>` : fournit le message
  - `<fmt:setLocale value=".."/>` : fixe la langue par défaut si la langue demandée n'est pas fournie.

Hello.jsp

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

   <fmt:setLocale value="en"/>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    </head>
    <body>
    <h1 align="center"><fmt:message key="title.welcome"/></h1>
    <fmt:message key="message.welcome"/>
    </body>
</html>
```

Accès aux variables

- JSTL = ensemble de tag libs
- Possibilité d'utiliser des objets Java en paramètres de ces tags
  - Objets transmis dans la session ou la requête
  - Instances référencées par un nom symbolique.
- Accès par la notation `${nomDeLobjet}`
- Accès aux attributs de l'objet (JavaBeans)

  `${objet.attribut}`

Exemple de tags

```
<c:out value="${...}"/>
```

- affiche dans la page la valeur de la variable

```
<c:forEach  var="..." items="${..}" />
.....
</c:forEach>
```

- Parcours la variable définit dans *items* (variable de type *Collection*).
- Les valeurs successives sont affectées à la variable définie dans *var*.

Exemple de tags (suite)

```
<c:url var=".." scope=".." value=".."/>
    <c:param name=".." value=".."/>
    ......
</c:url>
```

- Crée un variable de type URL,
  - `var` : nom de la variable
  - `scope` : domaine de validité (page, request, session)
  - `value` : valeur de l'url
- `<c:param/>` : permet de rajouter une requête à l'URL.

Exemple

- Faire une page affichant une liste de produits
- Un objet *Produit* (JavaBean)
- Un object d'accès à la BD (ProduitDAO)
- Un contrôleur pour construire la liste de produits
- Une page JSP pour afficher le vecteur

Produit (JavaBean)

```
public class Produit {
    private String nom ;
    private int prix ;

    public String getNom() {
        return nom;
    }

    public void setNom(String nom) {
        this.nom = nom;
    }

    public int getPrix() {
        return prix;
    }

    public void setPrix(int prix) {
        this.prix = prix;
    }
}
```

ProduitDAO

```
public class ProduitDAO {

    protected DataSource dataSource ;
    protected Connection conn = null ;
    protected Statement stmt = null ;

    public ProduitDAO() {
        try {
            Context init = new InitialContext();
            Context ctx = (Context) init.lookup("java:comp/env");
            dataSource = (DataSource) ctx.lookup("jdbc/magasin");
        } catch (NamingException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
```

ProduitDAO (suite)

```
    public List getListOfProduits() throws SQLException {
        conn = dataSource.getConnection();
        stmt = conn.createStatement() ;
        ResultSet resultSet = stmt.executeQuery("select * from article") ;
        Vector resultat = new Vector() ;

        while (resultSet.next()) {
            Produit produit = new Produit() ;
            produit.setNom(resultSet.getString(1)) ;
            produit.setPrix(resultSet.getInt(2)) ;
            resultat.add(produit) ;
        }
        conn.close() ;
        return resultat ;
    }
}
```

Contrôleur

```
public class ProduitsAction extends DispatchAction {

    public ActionForward getProduits(ActionMapping mapping,
                                     ActionForm form,
                                     HttpServletRequest request,
                                     HttpServletResponse response) throws Exception
    {
        ProduitDAO produitDAO = new ProduitDAO() ;
        List produits = produitDAO.getListOfProduits() ;

        request.setAttribute("listOfProduits", produits) ;

        return mapping.findForward("success") ;
    }
}
```

Présentation

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

<html>
    <body>

    <h1>Liste de produits</h1>

    <table border=1>
        <tr><th>Nom</th><th>Prix</th></tr>

        <c:forEach var="produit" items="${listOfProduits}">
        <tr>
           <td><c:out value="${produit.nom}"/></td>
           <td><c:out value="${produit.prix}"/></td>
        </tr>
        </c:forEach>

    </table>
    </body>
</html>
```

Taglib : HTML

- Redéfinir sous forme de Tag Lib tous les tags HTML (formulaire)
- Objectif :
  - Contrôler facilement les valeurs par défaut (section suivante)
- Exemples :

`<form action="..." method=".."/>`	`<html:form action=".." method=".."/>`
`<input type=text name="x">`	`<html:text property="x"/>`
`<input type="submit" value="envoyer">`	`<html:submit>envoyer</html:submit>`
...	...

B4 - Validation de formulaires

Problème

- Contrôleur :
  - Accéder aux valeurs d'un formulaire
  - Accéder aux paramètres d'une URL
- Formulaire :
  - Pré-remplir les champs avec des valeurs fournies par le contrôleur

ActionForm

- Mise en place d'un objet (JavaBean):
  - Reflet "objet" du formulaire
  - Reflet "objet" des paramètres d'une URL
  - Remarque : tous les champs peuvent ne pas être présents dans le formulaire ou l'URL
- Attention :
  - Formulaire : champs de type chaine de caractères

  $\Rightarrow$ les attributs du JavaBean sont de type *String*
- JavaBean : dérive d'*ActionForm*

ActionForm

```
public class ProduitForm extends ActionForm {

    private String nom ;
    private String prix ;

    public String getNom() {
        return nom;
    }

    public void setNom(String nom) {
        this.nom = nom;
    }

    public String getPrix() {
        return prix;
    }

    public void setPrix(String prix) {
        this.prix = prix;
    }
}
```

produitForm.jsp

```
<html>
    <body>

    <h1>Produit</h1>

    <c:url var="url" scope="page" value="addProduit.do">
        <<c:param name="action" value="addProduit"/>
    </c:url>

    <html:form action="${url}">

    <table>
    <tr><td>Nom</td><td><html:text property="nom"/></td>
    <tr><td>Prix</td><td><html:text property="prix"/></td>
    </table>

    <p>
    <html:submit>Ajouter</html:submit>

    </html:form>
    </body>
</html>
```

ProduitsAction

```
public ActionForward addProduit(ActionMapping mapping,
                                ActionForm form,
                                HttpServletRequest request,
                                HttpServletResponse response) throws Exception
{
    ProduitForm produitForm = (ProduitForm) form ;
    Produit produit = new Produit() ;

    BeanUtils.copyProperties(produit, produitForm) ;

    ..............
    return mapping.findForward("success") ;
}
```

Struts-config.xml

```
<struts-config>
    <form-beans>
        <form-bean name="produitForm" type="form.ProduitForm"/>
    </form-beans>

    <action-mappings>
        <action path="/enterProduit" forward="/produitForm.jsp"/>

        <action path="/addProduit" validate="false" name="produitForm" parameter="action"
                        type="action.ProduitsAction">
            <forward name="success" path="/success.jsp"/>
            <forward name="fail" path="/error.jsp"/>
        </action>

       <action path="/showProduits" parameter="action" type="action.ProduitsAction">
            <forward name="success" path="/showProduits.jsp"/>
        </action>
    </action-mappings>

    <message-resources parameter="MessageResources" null="false"/>
</struts-config>
```

Struts-config.xml

- envoie du formulaire = appel d'une URL :
  - Déclenchant la méthode *addProduit* de *ProduitAction*
  - Association du Java Bean au contrôleur

Passage de données du contrôleur vers le formulaire

- Contrôleur :

```
public  ActionForward setUpProduit(ActionMapping mapping,
         ActionForm form,
         HttpServletRequest request,
         HttpServletResponse response) throws Exception {
     ProduitForm produitForm = (ProduitForm) form ;

     Produit produit = new Produit() ;
     produit.setNom("Livre") ;
     produit.setPrix(40) ;

     BeanUtils.copyProperties(produitForm, produit) ;

     return mapping.findForward("success") ;
 }
```

- struts-config.xml :

```
<action path="/modifyProduit" name="produitForm" validate="false"
          parameter="action" type="action.ProduitsAction">
   <forward name="success" path="/produitForm.jsp"/>
</action>
```

Validation des données

- Deux aspects :
  - Vérification des champs (champs obligatoires, type des données etc)
  - Communication des erreurs à l'utilisateur
  - Prise en charge (optionnelle) par Struts
- Vérification des champs :
  - Redéfinir la méthode *validate* de l'ActionForm.
  - Renvoie *null* ou une liste vide si il n'y a pas d'erreur
  - La liste des messages d'erreur sinon
- Communication avec l'utilisateur
  - Par la liste des messages d'erreur
  - Struts boucle automatiquement tant qu'il reste des erreurs

validate (dans ActionForm)

```
public ActionErrors validate(ActionMapping mapping, HttpServletRequest request ) {
     ActionErrors errors = new ActionErrors() ;
     if (getNom().equals(""))
         errors.add("nom", new ActionMessage("errors.required", "name")) ;

     try {
         Integer.parseInt(getPrix()) ;
     } catch (NumberFormatException e) {
         errors.add("prix", new ActionMessage("errors.integer", "price")) ;
     }

     return errors ;
 }
```

- Méthode *add*
  - Association d'une clé à un message
  - Permet d'accéder aux messages depuis le formulaire

ActionMessage

- Définition d'un message à partir :
  - D'une clé provenant du fichier de ressources
  - De 1, 2, 3, ou 4 éléments à remplacer dans le texte associé.
- Exemple :
  - `new ActionMessage("errors.required", "name")) ;`
  - errors.required : `{0} is required.`
  - Résultat final : `name is required.`

Struts-config.xml

```
<struts-config>
    <form-beans>
        <form-bean name="produitForm" type="form.ProduitForm"/>
    </form-beans>

    <action-mappings>
        <action path="/enterProduit" validate="false" name="produitForm"
                forward="/produitForm.jsp"/>

        <action path="/addProduit" validate="true" input="/produitForm.jsp"
                name="produitForm" parameter="action" type="action.ProduitsAction">
            <forward name="success" path="/success.jsp"/>
            <forward name="fail" path="/error.jsp"/>
        </action>

   <message-resources parameter="MessageResources" null="false"/>

</struts-config>
```