

Introduction à JSF « Java Server Faces »



ATOL Conseils et Développements

3 bd Eiffel
21600 LONGVIC
contact@atolcd.com



www.atolcd.com

Tel : 03 80 68 81 68

Fax : 03 80 68 81 61

version 1.2

Introduction à JSF

Table des matières

1	PRÉSENTATION DE JSF.....	3
1.1	Introduction.....	3
1.2	Pourquoi utiliser JSF ?.....	3
2	CONFIGURATION TOMCAT/MyFACES.....	3
3	QU'EST CE QU'UN FICHIER JSF	5
3.1	Premier exemple JSF : Hello World.....	5
3.2	Cycle de vie d'une page JSF.....	7
3.3	Composants JSF les plus utilisés.....	8
4	JSF ET LES JAVA BEANS.....	8
4.1	Qu'est ce qu'un JavaBean.....	8
4.2	Les JavaBeans dans les JSF.....	9
5	JSF ET LE MODÈLE MVC.....	14
6	BIBLIOGRAPHIE.....	21

1 PRÉSENTATION DE JSF

1.1 INTRODUCTION

JSF (Java Server Faces) est un framework permettant la création d'interfaces WEB, la mise en place du Design Pattern MVC. Il bénéficie des apports du framework STRUTS et des concepts JAVA-J2EE (Swing, modèle événementiel, JSP, Servlets) tout en allant beaucoup plus loin, par l'apport de nouveaux concepts.

1.2 POURQUOI UTILISER JSF ?

JSF permet au développeur d'accroître la productivité du développement d'interfaces « client léger » tout en permettant une maintenance assez facile. JSF est un standard **J2EE**.

JSF permet :

- ➔ une séparation entre la couche présentation et les autres couches d'une application web
- ➔ une mise en place d'un **mapping HTML/OBJET**
- ➔ la **ré-utilisation** de composants graphiques
- ➔ une gestion de l'état de l'interface entre les différentes requêtes
- ➔ une **liaison** entre les actions coté « Client » et les actions des objets Java coté « Serveur »
- ➔ création de **composants customs** grâce à une API
- ➔ le **support de différents clients** (HTML, WML, XML, ...) grâce à la séparation des problématiques de construction de l'interface et du rendu de cette interface

2 CONFIGURATION TOMCAT/MYFACES

Il existe différentes implémentations de JSF : **SUN et MyFaces**. Au cours de la présentation des JSF, nous utiliserons l'implémentation **MyFaces**, couplée au serveur d'application **TOMCAT 5.5.12 (vous devez avoir installé le JDK 5.0)**.

Tout d'abord télécharger le serveur d'application TOMCAT :

<http://tomcat.apache.org/download-55.cgi>

Télécharger les bibliothèques nécessaires au développement d'une application WEB :

<http://archive.apache.org/dist/jakarta>

Télécharger les bibliothèques de l'implémentation de MyFaces:

<http://myfaces.apache.org/binary.cgi>

Les **.jar** doivent être copiés dans le répertoire **WEB-INF/lib** de votre application web :

Fichier Jar	Commentaires
commons-beanutils-1.6.1.jar	
commons-codec-1.2.jar	
commons-collections-3.0.jar	
commons-digester-1.5.jar	
commons-el.jar	
commons-fileupload-1.0.jar	Utilisé, seulement si vous « UPLOADER » des fichiers
commons-validator.jar	
commons-lang.jar	
jakarta-oro.jar	
jstl.jar	
log4j-1.2.8.jar	Utilisé, seulement si vous avez des « commons-logging » configurés pour utilisés log4j
myfaces-api.jar	
myfaces-impl.jar	
portlet-api-1.0.jar	Utilisé, seulement si vous travaillez avec des Portlets.
struts.jar	Utilisé, seulement si vous désirez utiliser la librairie tile.jar.
tomahawk.jar	Si vous désirez utiliser l'extension « Tomahawk » de MyFaces
myfaces-all.jar	Regroupe les 3 .jar suivants : <ul style="list-style-type: none"> • myfaces-api • myfaces-impl • tomahawk Utiliser plutôt ce fichier si vous voulez utiliser MyFaces et son extension Tomahawk

ATTENTION : vérifiez qu'il n'y aie pas de fichier **jsf-api.jar**, c'est à dire l'implémentation de SUN dans le CLASSPATH ou dans les répertoires partagés de TOMCAT \$CATALINA_HOME/common/lib et \$CATALINA_HOME/shared/lib.

3 QU'EST CE QU'UN FICHIER JSF

3.1 PREMIER EXEMPLE JSF : HELLO WORLD

Comme dans tous les tutoriels que vous avez pu rencontrer, vous avez pu constater que l'exemple « Hello World » est récurrent. C'est pour cela que nous n'écharperons pas à la règle et nous vous présenterons notre premier exemple via une sortie vers l'écran du message « HELLO WORLD ! ».

Toute application WEB JSF, nécessite la configuration de deux fichiers :

- ➔ web.xml
- ➔ faces-config.xml

Dans un premier temps éditer un fichier de nom **web.xml** et ajouter le contenu ci-dessous :

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC
    "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>

    <context-param>
        <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
        <param-value>server</param-value>
    </context-param>

    <context-param>
        <param-name>javax.faces.CONFIG_FILES</param-name>
        <param-value>/WEB-INF/faces-config.xml</param-value>
    </context-param>
    <!-- Faces Servlet -->
    <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup> 1 </load-on-startup>
    </servlet>

    <!-- Faces Servlet Mapping -->
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>*.jsf</url-pattern>
    </servlet-mapping>

</web-app>
```

Placer le fichier *web.xml* dans le répertoire **WEB-INF** de votre application WEB.

Créer un fichier de nom ***faces-config.xml*** et ajouter le contenu ci-dessous :

```
<?xml version="1.0"?>
<!DOCTYPE faces-config PUBLIC
"-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.0//EN"
"http://java.sun.com/dtd/web-facesconfig_1_1.dtd">
<faces-config>
  <application>
    <locale-config>
      <default-locale>fr</default-locale>
    </locale-config>
  </application>
</faces-config>
```

Placer le fichier *faces-config.xml* dans le répertoire WEB-INF de votre application WEB.

Maintenant, nous allons créer notre premier fichier JSF. Il porte l'extension .jsp et non .jsf comme on pourrait le croire.

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>

<%@ taglib uri="http://myfaces.apache.org/tomahawk" prefix="t"%>
<%@ taglib uri="http://myfaces.apache.org/wap" prefix="wap" %>

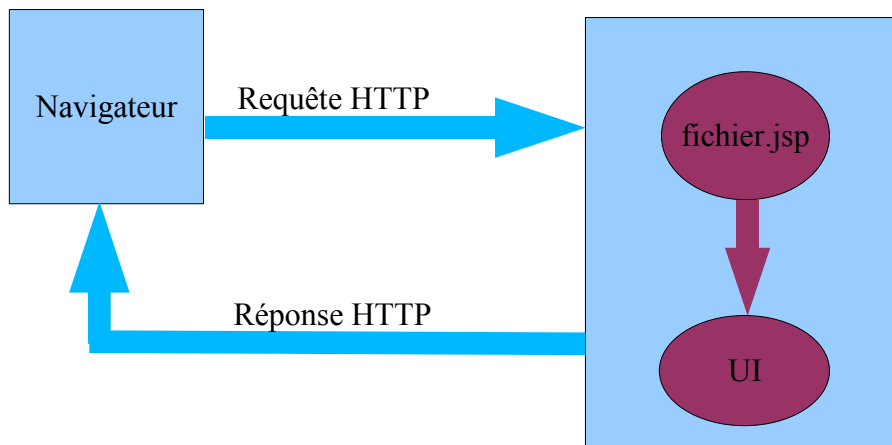
<f:view>
  <h:outputText value="Hello World ! (en JSF !)" />
</f:view>
```

Les deux premiers *imports* correspondent à l'importation des **TagLib** de JSF.

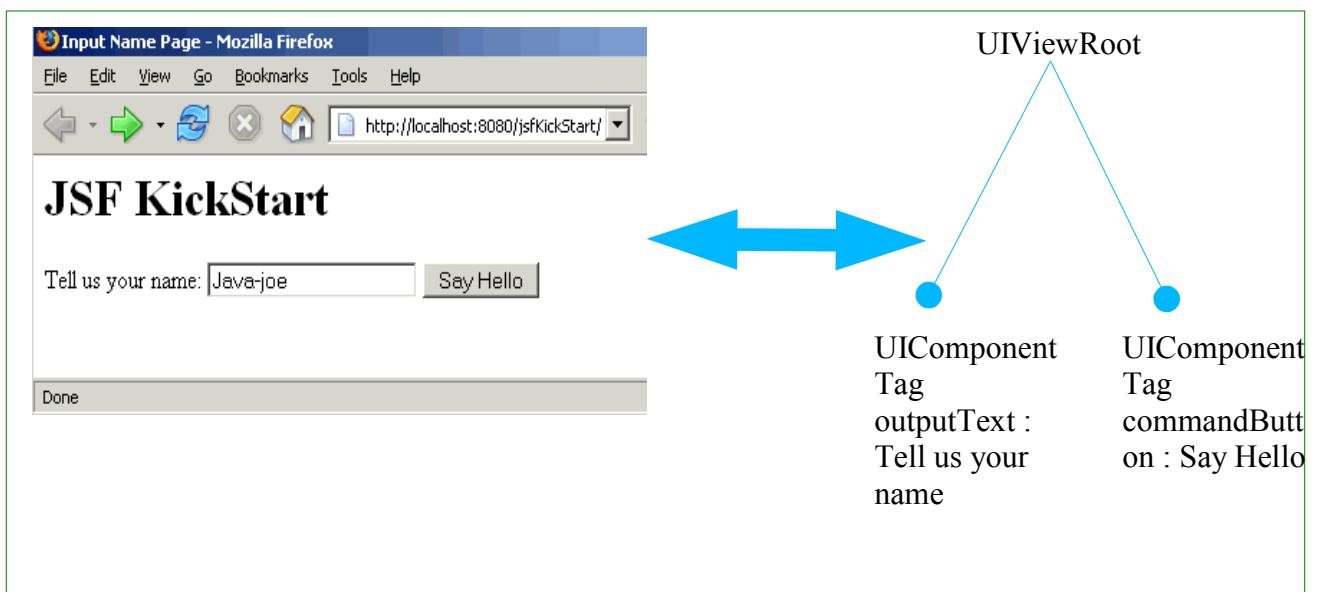
Les deux autres *imports* correspondent à l'importation des **TagLib** pour l'utilisation de l'extension **Tomahawk** de **MyFaces**.

3.2 CYCLE DE VIE D'UNE PAGE JSF

Voici le fonctionnement d'une page JSF, vu à travers le schéma ci-dessous :



Le navigateur accède à la page *jsp*, le serveur crée les composants et les renvoie au navigateur. Lors de l'exécution d'une page JSP, les *tags* de la page sont interprétés et exécutés. Le moteur JSF va alors créer un arbre de composants Java côté serveur dont la racine est un objet de type **javax.faces.component.UIViewRoot**. A chaque composant JSF du fichier, est associé un objet qui est de type **javax.faces.component.UIComponent**.



3.3 COMPOSANTS JSF LES PLUS UTILISÉS

Voici un tableau montrant les composants les plus utilisés dans les pages JSF :

Nom du composant	Tag JSF	Description
Output Text	outputText	Zone de texte
Hyperlink	outputLink	Lien
Image	graphicImage	Image
Text Field	inputText	Zone de saisie de texte
Multi Line Text Area	inputTextarea	Zone de saisie de texte sur plusieurs lignes
Button	commandButton	Bouton
Link Action	commandLink	Bouton de lien
DropDown List	selectOneMenu	Liste déroulante
List Box	selectOneListbox	TextBox permettant le choix de plusieurs option
Multi Select ListBox	selectManyListbox	Idem de que List Box
Data Table	dataTable	Tableau de zone de texte
CheckBox	selectBooleanCheckbox	Case à cocher
Hidden Field	InputHidden	Champ caché

4 JSF ET LES JAVA BEANS

Le lien entre le Client et le Serveur se fait grâce aux JavaBeans. En effet JSF permet de séparer la couche présentation de la logique métier. Le lien entre les deux se fait par la liaison des tags JSF avec des JavaBeans

4.1 QU'EST CE QU'UN JAVA BEAN

Un bean est un objet Java qui respecte la spécification JavaBean. Un bean doit respecter les spécification suivantes :

- ➔ la classe doit posséder au moins un constructeur vide
- ➔ les attributs de la classe ne doivent pas être publics
- ➔ chaque attribut de la classe doit posséder les méthodes getter et setter (getXXX et SetXXX), sachant que la première lettre suivant le **get** ou le **set** doit être en majuscules

4.2 LES JAVABEANS DANS LES JSF

Nous allons vous présenter l'interaction entre JSF et les Beans. Pour cela, créons une classe de nom `Personne`.

```
public class Personne {
    private String nom = "";
    private String prenom = "";

    public Personne() {
    }

    public String getNom() {
        return this.nom;
    }
    public void setNom(String nom) {
        this.nom = nom;
    }
    public String getPrenom() {
        return this.prenom;
    }
    public void setPrenom(String prenom) {
        this.prenom = prenom;
    }
}
```

Maintenant, nous allons créer deux fichiers : `saisieDonnees.jsp` et `afficheDonnees.jsp`, pour lesquels nous allons respectivement saisir un nom et un prénom de personne, et en ensuite afficher les données saisies.

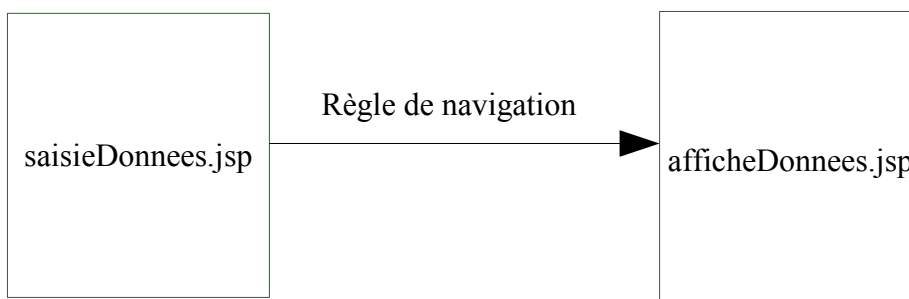
Création de *saisieDonnees.jsp* :

```
<%@ taglib uri="http://java.sun.com/jsp/html" prefix="h"%>
<%@ taglib uri="http://java.sun.com/jsp/core" prefix="f"%>
<f:view>
  <h3>Saisie des données personnelles</h3>
  <h:form>
    </BR>
    <h:outputText value= "Nom      " />
    <h:inputText value="#{personne.nom}" />
    </BR>
    <h:outputText value= "Prénom  " />
    <h:inputText value="#{personne.prenom}" />
    </BR>
    </BR>
    <h:commandButton value="Afficher les données saisies"
                     action="AfficherDonnees"/>
  </h:form>
</f:view>
```

Création de *afficheDonnees.jsp*:

```
<%@ taglib uri="http://java.sun.com/jsp/html" prefix="h"%>
<%@ taglib uri="http://java.sun.com/jsp/core" prefix="f"%>
<f:view>
  <h3>Affichage des données saisies</h3>
  </BR>
  <h:outputText value= "Nom :  #{personne.nom}" />
  </BR>
  <h:outputText value= "Prénom : #{personne.prenom}"/>
  </BR>
</f:view>
```

Pour que le lien se fasse lorsque vous appuyez sur le bouton d'envoi du formulaire, il est nécessaire de définir **une règle de navigation** au sein de l'application :



La règle de navigation doit être déclarée dans le fichier **faces-config.xml** de cette façon :

```
<navigation-rule>
  <!-- Indique pour quelle vue courante la regle s applique -->
  <from-view-id>/saisieDonnees.jsp</from-view-id>
  <navigation-case>
    <!-- Si l outcome renvoie est AfficherDonnees
    alors JSF passe a la page /afficheDonnees.jsp -->
    <from-outcome>AfficherDonnees</from-outcome>
    <to-view-id>/afficheDonnees.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
```

Egalement pour que le bean *Personne* puisse être reconnu par les vues JSF, il faut le déclarer dans le fichier **faces-config.xml** de cette manière :

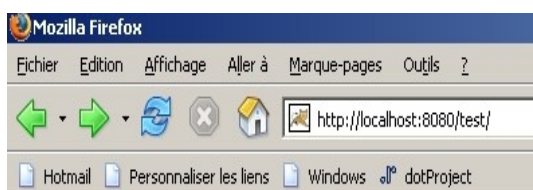
```
<managed-bean>
  <managed-bean-name>personne</managed-bean-name>
  <managed-bean-class>Personne</managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

Le **tag <managed-bean-name>** déclare le nom du bean, la manière dont il sera appelé dans les fichiers JSF.

Le **tag <managed-bean-class>** spécifie la classe du bean.

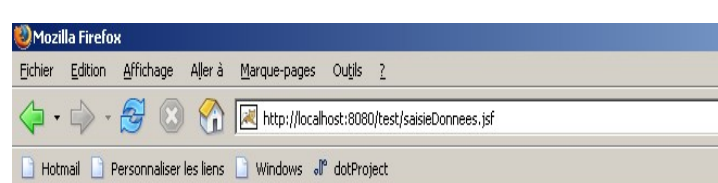
Le **tag <managed-bean-scope>** spécifie la portée du bean au sein de l'application, les différentes valeurs possibles sont : application, session ou request.

Voici le résultat des l'exécution des fichiers :



Saisie des données personnelles

Nom
 Prénom



Affichage des données saisies

Nom : Luis
 Prénom : Figo



Voici un exemple JSF, **ne respectant pas** le principe de Tags JSF « binder » avec un Bean. Cette exemple montre un arbre contenant des données extraites d'une base de données. Pour la connexion à la base de données nous utiliserons, un pont JDBC.

Voici le squelette du code permettant de créer un arbre, avec des données extraites d'une base de données.

```

<%@ page import="org.apache.myfaces.custom.tree.DefaultMutableTreeNode,
    org.apache.myfaces.custom.tree.model.DefaultTreeModel"
    import = "java.sql.Connection"
    import = "java.sql.DriverManager"
    import = "java.sql.ResultSet"
    import = "java.sql.SQLException"
    import = "java.sql.Statement" %>
<%@ page session="true" contentType="text/html;charset=utf-8"%>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@ taglib uri="http://myfaces.apache.org/tomahawk" prefix="t"%>
<html>
<body>

<%
if (pageContext.getAttribute("treeModel", PageContext.SESSION_SCOPE) == null) {

    DefaultMutableTreeNode root = new DefaultMutableTreeNode("Nom_Racine");
    //DECLARER D AUTRES DefaultMutableTreeNode en fonction des besoins ...

    //OBJET POUR LA CONNEXION
    Connection conn = null;
    try {
        try {
            //CHARGEMENT du Driver de la base de données
            Class.forName("Driver_Base_Donnees");
        } catch(ClassNotFoundException e1) {
            System.out.println("ERREUR du Chargement du DRIVER");
            e1.printStackTrace();
        }
        //CONNEXION à la base de données
        conn = DriverManager.getConnection("Chaine_de_Connexion","Login","Password");
    } catch(SQLException e2) {
        System.out.println("ERREUR de CONNEXION");
        e2.printStackTrace();
    }
    System.out.println("Connexion établie ...");

    ...
    //AJOUTER ICI LE CODE POUR LA CONSTRUCTION DE L ARBRE
    ...

    pageContext.setAttribute("treeModel", new DefaultTreeModel(root), PageContext.SESSION_SCOPE);
}
%>

```

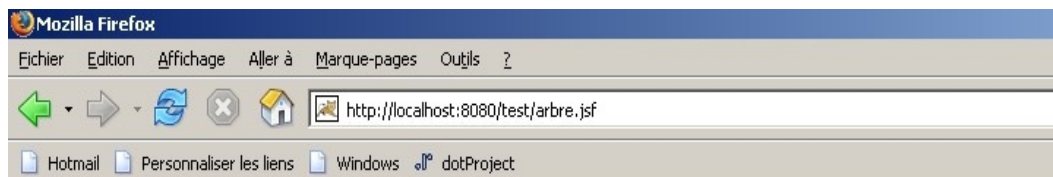
```
</f:view>
// composant TREE issu de l'extension TOMAHAWK
<t:tree id="tree" value="#{treeModel}"
  styleClass="tree"
  nodeClass="treenode"
  selectedNodeClass="treenodeSelected"
  expandRoot="true">
</t:tree>

</f:view>

</body>

</html>
```

Voici le résultat d'un arbre, dont les données sont extraites du domaine de la comptabilité analytique hospitalière.



Arbre des comptes referentiel de Lettre Budgetaire P et de code commençant par 326

- [-] Comptes referentiel
 - [-] 326-Fournitures consommables
 - 3261-Combustibles et carburants
 - 3262-Produits d'entretien
 - 3263-Fournitures d'atelier
 - 3264-Fournitures scolaires, éducatives et de loisirs
 - 3265-Fournitures de bureau et informatiques
 - [-] 3266-Fournitures hôtelières
 - 32662-Petit matériel hôtelier
 - 32663-Linge et habillement
 - 32668-Autres fournitures hôtelières
 - 3268-Autres fournitures consommables

5 JSF ET LE MODÈLE MVC

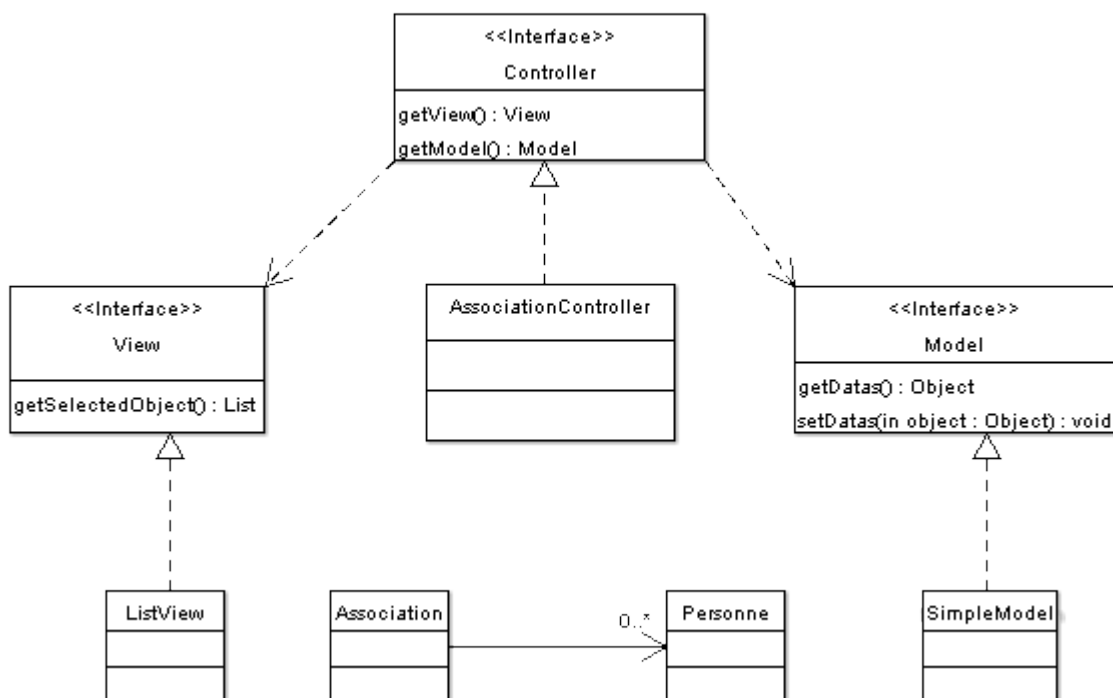
Le framework JSF permet de mettre en place une architecture **MVC** (Modèle View Controller). Pour cela, nous allons vous présenter comment mettre en place une telle architecture basée sur un exemple. Notre exemple va s'appuyer sur une liste de personnes faisant partie d'une association. Nous allons créer une interface permettant d'ajouter et de supprimer des personnes.

Notre exemple se divisera en deux pages JSF:

- suppression d'adhérents
- ajout d'un adhérent

La suppression d'adhérents, va se faire en sélectionnant dans un tableau, le ou les adhérents à supprimer, et la validation se faire par l'action sur un bouton.

L'ajout d'un adhérent se fera en saisissant son nom et prénom et en validant par l'action sur un bouton. Pour les informations sur l'adhérent nous nous sommes limités à des données assez simples, car le but de cet exemple n'étant pas de mettre en place une application riche en données, mais de montrer le fonctionnement global d'une architecture MVC.



La classe **Controller** permet de faire le lien entre la vue et le modèle de données.

La vue représente les composants JSF . Dans notre cas, les composants JSF seront un tableau et une checkbox. Cette vue permettra de faire le lien avec la page JSF.

Le modèle représente les données manipulées par l'utilisateur via la vue. Dans notre exemple, nos données seront une liste de personnes. La classe **Association** et **Personne** modélisent le domaine.

La classe **Association** :

```
package beans;

import java.util.ArrayList;
import beans.Personne;

public class Association {

    private ArrayList adherents = null;
    public Association() {
        this.adherents = new ArrayList();
        this.adherents.add(new Personne("RONALDO","Cristiano"));
        this.adherents.add(new Personne("ZIDANE","Zinedine"));
        this.adherents.add(new Personne("FIGO","Luis"));
        this.adherents.add(new Personne("MOREIRA","Ronaldo"));
    }
    public ArrayList getAdherents() {
        return this.adherents;
    }
    public void setAdherents(ArrayList adherents) {
        this.adherents = adherents;
    }
    public void removeAdherent(Personne adhe){
        this.adherents.remove(adhe);
    }
    public void addAdherent(Personne adhe){
        this.adherents.add(adhe);
    }
}
```

Les classes **View** et **ListView** :

```
package beans;

import java.util.List;

public interface View {
    public List getSelectedObjects();
}
```

```
package beans;
import java.util.ArrayList;
import java.util.List;
import javax.faces.component.UIData;
import javax.faces.component.UISelectBoolean;

public class ListView implements View {

    private UIData dataTable;
    private UISelectBoolean checkbox;

    public ListView() {
    }
    public UIData getDataTable(){
        return this.dataTable;
    }
    public void setDataTable(UIData dataTable){
        this.dataTable = dataTable;
    }
    public UISelectBoolean getCheckbox(){
        return this.checkbox;
    }
    public void setCheckbox(UISelectBoolean checkbox){
        this.checkbox = checkbox;
    }
    public List getSelectedObjects(){
        int size = this.dataTable.getRowCount();
        List datas = (List) this.dataTable.getValue();
        List selectedObjects = new ArrayList();
        for(int i=0; i < size; i++){
            this.dataTable.setRowIndex(i);
            if(this.checkbox.isSelected()){
                selectedObjects.add(datas.get(i));
            }
        }
        return selectedObjects;
    }
}
```

Remarquer, la déclaration des deux objets de types **UIData** et **UISelectBoolean**, objets qui vont permettre de faire le *binding* avec les *tags* JSF.

Les classes **Model** et **SimpleModel** :

```
package beans;

public interface Model {

    public Object getDatas();
    public void setDatas(Object object);
}
```

```
package beans;
import java.util.ArrayList;
import java.util.List;

public class SimpleModel implements Model {
    private Object datas;
    public SimpleModel() {
    }
    public Object getDatas(){
        return this.datas;
    }
    public void setDatas(Object object){
        this.datas = object;
    }
}
```

Les classes **Controller** et **AssociationListController** :

```
package beans;
import beans.Model;
import beans.View;
public interface Controller {
    public Model getModel();
    public View getView();
}
```

```
package beans;

import beans.Association;
import beans.Personne;

public class AssociationListController implements Controller {
    private Model model;
    private View view ;

    private Personne personne;

    public Personne getPersonne() {
        return this.personne;
    }
    public void setPersonne(Personne pers) {
        this.personne = pers;
    }
    public AssociationListController() {
        this.personne = new Personne();
        this.model = new SimpleModel();
        this.view = new ListView();
        this.model.setDatas(new Association());
    }

    public View getView(){
        return this.view;
    }

    public void setView(View view){
        this.view = view;
    }

    public Model getModel(){
        return this.model;
    }

    public void setModel(Model model){
        this.model = model;
    }

    public void removeSelectedPersonnes(){
        System.out.println("Suppression d'une personne ...");
        Association assoc = (Association) getModel().getDatas();
        assoc.getAdherents().removeAll(getView().getSelectedObjects());
    }
    public void addPersonne(){
        System.out.println("Insertion d'une nouvelle personne ...");
        Association assoc = (Association) getModel().getDatas();
        assoc.getAdherents().add(this.personne);
        System.out.println("Adhérents : "+assoc.getAdherents());
    }
}
```

Voici le code la page *association.jsp* :

```
<%@ page contentType="text/html" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<f:view>
  <h3>Association des footballeurs</h3>
  <h:form>
    <h:dataTable binding="#{assocCtrl.view.dataTable}"
      value="#{assocCtrl.model.datas.adherents}" var="personne" border="1">
      <h:column>
        <h:selectBooleanCheckbox binding="#{assocCtrl.view.checkbox}"/>
      </h:column>
      <h:column>
        <f:facet name="header">
          <f:verbatim>Nom</f:verbatim>
        </f:facet>
        <h:outputText value="#{personne.nom}"/>
      </h:column>
      <h:column>
        <f:facet name="header">
          <f:verbatim>Prénom</f:verbatim>
        </f:facet>
        <h:outputText value="#{personne.prenom}"/>
      </h:column>
    </h:dataTable>
    <br>
    <h:commandButton value="Supprimer adhérent"
      action="#{assocCtrl.removeSelectedPersonnes}"/>
    <h:commandButton value="Ajouter adhérent"
      action="Ajout"/>
  </h:form>
</f:view>
```

Le tag **dataTable** définit un tableau. L'attribut **value** permet de spécifier où récupérer les données. Le tag **var** va permettre lorsque le composant **dataTable** va itérer sur lui-même de stocker à chaque itération le contenu du tag **value**.

Le tag **facet** permet d'associer un composant à un autre sans qu'il y ait relation de filiation entre les deux composants.

Le tag **<f:verbatim> Nom </f:verbatim>** permet de déclarer le nom de la colonne. Le fait de déclarer un composant **facet** évite que la table, à chaque itération affiche le nom des colonnes. En outre il faut afficher le nom des colonnes avant d'afficher les lignes du tableau et pour cela rendre le nom des colonnes indépendant du tableau.

Le tag **binding** va permettre de lier le composant JSF **dataTable** à la propriété **UIData** du bean **ListView**.

Le tag **action** permet lors du clic sur le bouton, d'exécuter une action. Dans notre cas, d'exécuter la méthode **removeSelectedPersonnes** du bean **AssociationListController** ou de retourner une chaîne de caractères « Ajout » qui sera utilisée pour la navigation.

Voici le code de la page *ajout.jsp* :

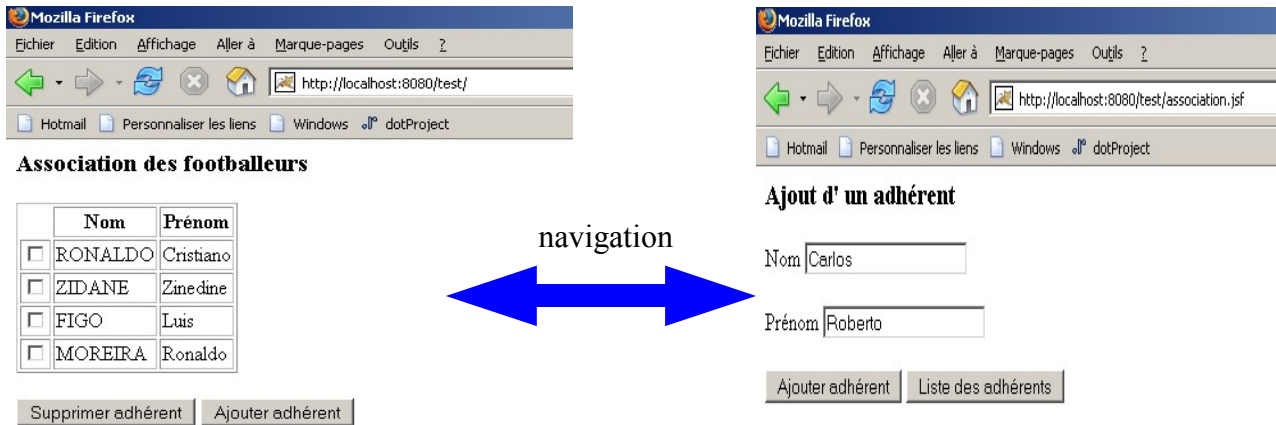
```
<%@ page contentType="text/html" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<f:view>
  <h3>Ajout d' un adhérent</h3>
  <h:form>
    <h:outputText value = "Nom " />
    <h:inputText value = "#{assocCtrl.personne.nom}"/>
    <br>
    <br>
    <h:outputText value = "Prénom " />
    <h:inputText value = "#{assocCtrl.personne.prenom}"/>
    <br>
    <br>
    <h:commandButton value="Ajouter adhérent" action= "#{assocCtrl.addPersonne}"/>
    <h:commandButton value="Liste des adhérents" action= "liste"/>
  </h:form>
</f:view>
```

Afin de permettre un bon fonctionnement de notre exemple, il faut ajouter ces déclarations dans le fichier *faces-config.xml*. Ces déclarations décrivent la règle de navigation entre les deux pages *jsp* et les beans utilisés dans l'application.

```
<navigation-rule>
  <from-view-id>/association.jsp</from-view-id>
  <navigation-case>
    <from-outcome>Ajout</from-outcome>
    <to-view-id>/ajout.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
<navigation-rule>
  <from-view-id>/ajout.jsp</from-view-id>
  <navigation-case>
    <from-outcome>liste</from-outcome>
    <to-view-id>/association.jsp</to-view-id>
  </navigation-case>
</navigation-rule>

<managed-bean>
  <managed-bean-name>personne</managed-bean-name>
  <managed-bean-class>beans.Personne</managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
<managed-bean>
  <managed-bean-name>assoc</managed-bean-name>
  <managed-bean-class>beans.Association</managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
<managed-bean>
  <managed-bean-name>assocCtrl</managed-bean-name>
  <managed-bean-class>beans.AssociationListController</managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

Voici donc le résultat de l'exécution des pages *JSF* :



Association des footballeurs

	Nom	Prénom
<input type="checkbox"/>	RONALDO	Cristiano
<input type="checkbox"/>	ZIDANE	Zinedine
<input type="checkbox"/>	FIGO	Luis
<input type="checkbox"/>	MOREIRA	Ronaldo

Supprimer adhérent Ajouter adhérent

Ajout d' un adhérent

Nom

Prénom

Ajouter adhérent Liste des adhérents

navigation

conclusion

JSF permet une production accrue d'interfaces web, tout en les séparant de la logique métier. JSF permet d'utiliser un grand nombre de composants standards ou de nouveaux via l'extension Tomahawk. JSF permet également la création de composants customs (personnalisés). Les composants JSF n'atteignent pas la richesse des composants des clients lourds comme Java Swing. Cette présentation visait à vous montrer le fonctionnement global du framework JSF.

6 BIBLIOGRAPHIE

Sites :

<http://myfaces.apache.org>

<http://myfaces.apache.org/tomahawk/overview.html>

<http://java.sun.com/j2ee/javaxserverfaces/index.html>

<http://java.sun.com/j2ee/javaxserverfaces/docs/tutorial.html>