JSF
Java Server Faces

L'AWT-SWING du WEB

Yves Bekkers

Introduction

JSF Qu'est-ce que c'est ?

- Une technologie de développement d'application WEB (*framework*) qui dispense d'écrire du code Java dans les interfaces
- Qui offre
  - Deux librairies de composants graphiques
  - Mécanismes d'évènements et de validation
  - Mécanisme de navigation de page en page
  - Gestion automatique de d'objets en arrière plan
- Conçue aussi pour faciliter la fabrication d'outils de développements adéquates (plus de 6 outils de développement connus)

Deux librairies

- Librairie HTML (composants graphiques)

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

- Librairie Core (validation, gestion des évènements, conversions)

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


Librairie HTML

- **Inputs** `<h:inputText>, <h:inputTextarea>`
- **Outputs** `<h:outputText>, <h:outputLabel>`
- **Commands** `<h:commandButton>`
- **Selections** `<h:selectOneRadio>, <h:selectOneListbox>, <h:selectOneMenu>`
- **Layouts** `<h:panelGrid>`
- **Data table** `<h:dataTable>`
- **Errors and messages** `<h:message>, <h:messages>`

Librairie core

- **`<f:view>`** fenêtre principale
- **`<f:subview>`** fenêtre secondaire au sein d'une fenêtre principale.
- **`<f:validator>`** ajouter une validation à un composant.
- **`<f:converter>`** ajouter un convertisseur à un composant.
- **`<f:actionListener>`** ajouter un écouteur d'évènement de type action.
- **`<f:valueChangeListener>`** ajouter un écouteur d'évènement de type changement de valeur.

Composants graphiques et rendu

- Deux niveaux de vues
  - la vue abstraite ou modèle, la vue concrète (rendu visuel)
  - Un même composant abstrait peut avoir plusieurs rendus.
- le composant `UISelectOne` possède les 3 rendus
  - Bouton radio
  - Case à cocher
  - Élément d'une liste déroulante
- Le composant `UICommand` possède 2 rendus
  - Le lien hypertexte `<commandLink>`
  - Le bouton `<commandButton>`

Le composant abstrait `UIInput`

- `IUInput` possède 4 rendus
  - `<h:inputHiden>` Paramètres cachés
  - `<h:inputSecret>` Ligne de texte sans espace rendu par des *
  - `<h:inputText>` Une ligne de texte
  - `<h:inputTextarea>` Plusieurs lignes de texte

Rendus du composants `UIInput`

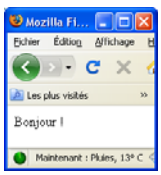
- `<h:inputext>`
- `<h:inputsecret>`
- `<h:inputhidden>`
- `<h:inputTextArea>`

Listeners, validateur, convertisseurs

- On peut nommer les composants graphiques abstraits et leur associer des
  - Listeners
  - Validateurs
  - Convertisseurs
  - Des objets (java beans)

La page bonjour

```
<body>
   <f:view>
      <h:outputText value="Bonjour !" />
   </f:view>
</body>
```



Second exemple – un formulaire


```
<body>
  <f:view>
    <h:form>
      <h:commandButton action= "vert"
                       value="GoToGreenPage" />
      <h:commandButton action= "bleu"
                       value="GoToBluePage" />
    </h:form>
  </f:view>
</body>
```

Exemples de tags JSF

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

`<f:view>`	Conteneur des éléments graphiques : obligatoire dans toute page JSF
`<h:form>`	Formulaire
`<h:commandButton>`	Bouton de formulaire

Technologies concurrentes

- Servlets et pages JSP
  - Pas de technologie d'interface à base de composants graphiques
- Struts
  - Pas de technologie d'interface à base de composants graphiques
  - Pas de modèle d'évènements
  - Pas de gestion de l'état de composants graphiques
  - Struts est lié à HTML pour son rendu
- Spring
  - Très récent, à voir …

Bibliothèques requises

- **`jsf-api.jar`**
  - librairies `javax.faces.*`
- **`jsf-imp.jar`** (ou **`jsf-ri.jar`**)
  - l'implementation des composants `com.sun.faces.*`
- **`jstl.jar`** et **`standard.jar`**
  - nécessaires pour utiliser les tags JSTL tags et pour certaines références faites par l'api JavaServer Faces (internationalisation …)
- **`commons-beanutils.jar`**
  - manipulation des java beans
- **`commons-digester.jar`**
  - Manipulation des documents XML
- **`commons-collections.jar`**
  - extensions de Java 2 SDK (`Collections`)
- **`commons-logging.jar`**
  - des facilités génériques pour créer des fichiers de "*log*"

Fichier `web.xml`

- Déclaration de la servlet controleur JSF

```
<!-- 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>
```


Fichier de configuration

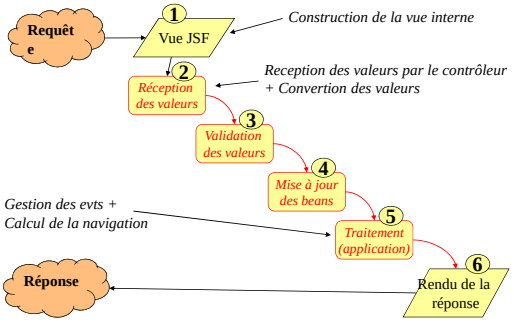
- Fichier de configuration **faces-config.xml**
  - Définir la navigation de paga en page
  - Définir les instances d'objets d'arrière plan (les *backing beans*)
  - Définir des classes de validation de données
  - Définir des classes d'écoute d'évènements

Fonctionnement

JSF comment ça marche

- Chaque page est caractérisée par un arbre de composants graphiques
- Une **servlet contrôleur** construit les pages avec leurs composants graphiques (*Input type, label, table...*) puis la servlet déclenche des évènements relatifs à chacun des composants de la page.
- Les valeurs affichées ou saisies dans les composants graphiques peuvent être associées à des attributs d'instances d'objets gérés automatiquement en arrière plan de l'application « les *backing beans* »

Cycle de vie d'une requête JSF



Étapes 1-2

- 1 - **Constrution de la vue d'entrée** : recréer l'arborescence des composants qui composent la page d'origine. Cette arborescence est stockée dans un objet de type *FacesContext* et sera utilisée tout au long du traitement de la requête.
- 2 - **Réception des valeurs** trouvées dans le FaceContext. Durant cette phase des opérations de *convertions* sont réalisées pour permettre de transformer les valeurs stockées sous forme de chaîne de caractères dans la requête HTTP en un type utilisé pour le stockage des données.

Étapes 3 - 4

- 3 - **Validation** : les données extraites sont validées en appliquant des *validateurs* enregistrés auprès de chaque composant. Les éventuelles erreurs de conversions sont stockées dans le FaceContext. Dans ce cas d'erreur, l'étape suivante est directement l'étape 6 pour réafficher la même page avec les valeurs saisies et signifier les erreurs
- 4 - **Synchronisation du modèle** : Cette étape permet de stocker dans les composants du FaceContext leur valeur locale validée respective. Les éventuelles erreurs de conversions sont stockées dans le FaceContext. Dans ce cas, l'étape suivante est directement l'étape 6

Étapes 5-6

- 5 - **Invoquer l'application** : dans cette étape, le ou les événements émis dans la page sont traités. On détermine quelle sera la page résultat qui sera renvoyée dans la réponse en utilisant les règles de navigation définie dans l'application. L'arborescence des composants de cette page résultat est créée.
- 6 - **Rendu de la réponse** : cette étape se charge de créer le rendue de la page de la réponse.

Identification des 6 phases

- Possibilité de spécifier des `listerners` déclanchés aux instants significatifs des phases d'exécution du contrôleur
- La classe `javax.faces.event.PhaseId` définit des constantes qui identifient chacune des 6 phases
  - `PhaseId.RESTORE_VIEW,`
  - `PhaseId.APPLY_REQUEST_VALUES,`
  - `PhaseId.PROCESS_VALIDATIONS,`
  - `PhaseId.UPDATE_MODEL_VALUES,`
  - `PhaseId.INVOKE_APPLICATION`
  - `PhaseId.RENDER_RESPONSE`
- La constante `PhaseId.ANY_PHASE` permet de demander l'application du `listener` à toutes les phase (debuggage)

Exemple : `Ecouteur` (traceur)

- Un écouteur implémente la classe `javax.faces.event.PhaseListener`

```
import javax.faces.event.PhaseEvent;
import javax.faces.event.PhaseId;
import javax.faces.event.PhaseListener;
public class Ecouteur implements PhaseListener {
  public void afterPhase(PhaseEvent pe) {
      System.out.println("Apres " + pe.getPhaseId());
  }
  public void beforePhase(PhaseEvent pe) {
      System.out.println("Avant " + pe.getPhaseId());
  }
  public PhaseId getPhaseId() {
      return PhaseId.ANY_PHASE;
  }
```


Enregistrement d'un `PhaseListener`

- Fichier de configuration « server Faces »

```
<faces-config>
   ...
   <lifecycle>
   <phase-listener>lib.PhasesEcouteur
                          </phase-listener>
   </lifecycle>
   …
</faces-config>
```


Exemple d'exécution de la classe `Ecouteur`

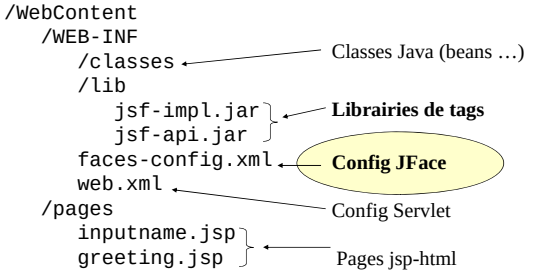
```
Avant RESTORE_VIEW 1
Apres RESTORE_VIEW 1
Avant APPLY_REQUEST_VALUES 2
Apres APPLY_REQUEST_VALUES 2
Avant PROCESS_VALIDATIONS 3
Apres PROCESS_VALIDATIONS 3
Avant UPDATE_MODEL_VALUES 4
Apres UPDATE_MODEL_VALUES 4
Avant INVOKE_APPLICATION 5
Apres INVOKE_APPLICATION 5
Avant RENDER_RESPONSE 6
Apres RENDER_RESPONSE 6
```


Toutes les phases ne sont pas toujours exécutées

- Exemple d'exécution avec une erreur de validation

```
Avant RESTORE_VIEW 1
Apres RESTORE_VIEW 1
Avant APPLY_REQUEST_VALUES 2
Apres APPLY_REQUEST_VALUES 2
Avant PROCESS_VALIDATIONS 3
Apres PROCESS_VALIDATIONS 3
Avant RENDER_RESPONSE 6
Apres RENDER_RESPONSE 6
```


Structure d'une application JSF



`web.xml` : Un contrôleur «*faces*»

```
<web-app>
    <!- Contrôleur 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>/faces/*</url-pattern>
    </servlet-mapping>

</web-app>
```

web.xml + faces-config.xml

```
<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>

  <listener>
    <listener-
  class>com.sun.faces.config.ConfigureListener
                  </listener-class>
  </listener>
...
```


Fichier de configuration JFace

- Structure du document

```
<?xml version="1.0"?>
<!DOCTYPE faces-config PUBLIC
  "-//Sun Microsystems, Inc.//DTD
  JavaServer Faces Config 1.1//EN"
  "http://java.sun.com/dtd/web-
  facesconfig_1_1.dtd">

<faces-config>
  <navigation-rule> … </navigation-rule>
  <managed-bean> … </managed-bean>
</faces-config>
```


Fichier de configuration JFace

`<application>`	Configuration de l'application
`<factory>`	Remplacement de certaines fabriques (`FacesContextFactory`, …)
`<component>`	Définition de composants graphiques
`<converter>`	Définition de convertisseurs
`<managed-bean>`	Définition d'objets autogérés
`<navigation-rule>`	Définition des règles de navigation
`<render-kit>`	Définition d'un kit de rendu
`<validator>`	Définition de validateurs

Enregistrement de la classe `Ecouteur`

- Déclaration dans le fichier de configuration « *Faces* » **`faces-config.xml`**

```
<faces-config>
...
  <lifecycle>
    <phase-listener>myLib.Ecouteur</phase-listener>
  </lifecycle>
...

</faces-config>
```


Navigation

Soumettre un formulaire
Lancer une action

- Gestion par JSF : deux méthodes
  - Le bouton

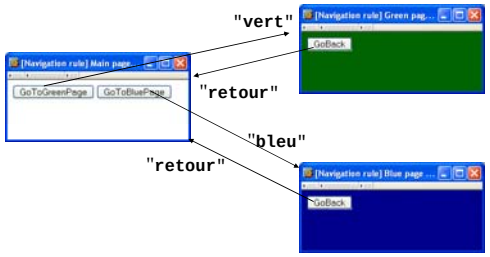
    Next

    ```
    <h:commandButton id="submit" value=
      "Next" action="nextPage"/>
    ```
  - Le lien (utilise Javascript …)

    Next Page

    ```
    <h:commandLink id="link"
      action="goto">
       <h:outputText value="Next page"/>
    </h:commandLink>
    ```

Navigation dirigée par des chaînes



index.jsp

```
<body>
 <f:view>
  <h:form>
    <h:commandButton action= "vert" value="GoToGreenPage" />
    <h:commandButton action= "bleu" value="GoToBluePage" />
  </h:form>
 </f:view>
</body>
```


green.jsp

```
<body bgcolor="darkgreen">
 <f:view>
  <h:form>
    <h:commandButton action="back" value="GoBack" />
  </h:form>
 </f:view>
</body>
```

blue.jsp

```
<body bgcolor="darkblue">
<f:view>
 <h:form>
    <h:commandButton action="back" value="GoBack" />
 </h:form>
</f:view>
</body>
```

faces-config.xml règles de navigation (1)

```
<navigation-rule>
<from-view-id>/index.jsp</from-view-id>
<navigation-case>
   <from-outcome>vert</from-outcome>
   <to-view-id>/green.jsp</to-view-id>
</navigation-case>
<navigation-case>
   <from-outcome>bleu</from-outcome>
   <to-view-id>/blue.jsp</to-view-id>
</navigation-case>
</navigation-rule>
```

faces-config.xml règles de navigation (2)

```
<navigation-rule>
<from-view-id>/green.jsp</from-view-id>
<navigation-case>
   <from-outcome>back</from-outcome>
   <to-view-id>/index.jsp</to-view-id>
</navigation-case>
</navigation-rule>

<navigation-rule>
<from-view-id>/blue.jsp</from-view-id>
<navigation-case>
   <from-outcome>back</from-outcome>
   <to-view-id>/index.jsp</to-view-id>
</navigation-case>
</navigation-rule>
```

Navigation rules (2) simplification

```
<navigation-rule>
  <navigation-case>
    <from-outcome>back</from-outcome>
    <to-view-id>/index.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
```


Navigation calculée : langage d'expression

- Expression dans l'attribut *action* du bouton

```
<h:commandButton value="annuler"
action="#{myBean.eval}"/>
```

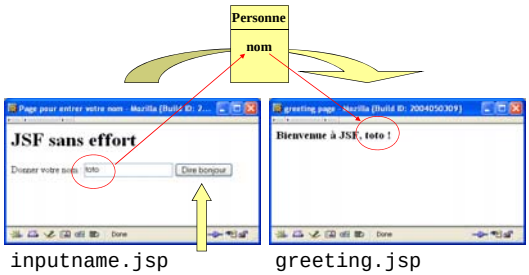
- Fait appel à une méthode d'un bean qui rend un résultat `String`

```
public String eval() {
   if (test) return "false"
   return "true"
}
```


JavaBeans Supervisés

Deux pages

JavaBean



`inputname.jsp` `greeting.jsp`

Lier l'interface aux Javabeans

inputName.jsp

```
<h:inputText value="#{laPersonne.nom}"/>
```

faces-config.xml

```
<managed-bean>
  <managed-bean-name>
   laPersonne
  </managed-bean-name>
  <managed-bean-class>
   lib.PersonBean
  </managed-bean-class>
```

PersonBean.java

```
public class PersonBean {
   private String nom;
   public String getNom() {…
   public void setNom(String nom) {…
```


JavaBean

```
public class PersonBean {
   private String nom;
   public String getNom() {
      return nom;
   }
   public void setNom(String nom) {
      this.nom = nom;
   }
}
```

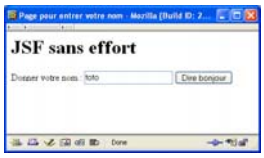
Déclaration des instances de bean

Au sein du fichier `faces-config.xml`

```
<faces-config>
...
<managed-bean>
  <managed-bean-name>laPersonne</managed-bean-name>
  <managed-bean-class>lib.PersonBean</managed-bean-class>
  <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
...
</faces-config>
```


`inputName.jsp` : formulaire

```
<h1><h:outputText value="JSF sans effort" /></h1>
<h:form id="helloForm">
   <h:outputText value="Donner votre nom"/>
   <h:inputText value="#{laPersonne.nom}"/>
   <h:commandButton action="greeting" value="Dire
  bonjour" />
</h:form>
```



La valeur saisie dans le composant `inputText` est transférée automatiquement Dans le Champ `nom` du Javabean `laPersonne`

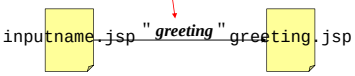
`greeting.jsp`

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<html>
  <head><title>greeting page</title></head>
  <body>
     <f:view>
       <h3>
    <h:outputText value="Bienvenue à JSF"/>
    <h:outputText value="#{laPersonne.nom}" />
      <h:outputText value="!" />
       </h3>
     </f:view>
  </body>
</html>
```

Configuration JFace (1)

- Règles de navigation

```
<navigation-rule>
  <from-view-id>/pages/inputname.jsp</from-view-id>
  <navigation-case>
    <from-outcome>greeting</from-outcome>
    <to-view-id>/pages/greeting.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
```



Configuration JFace (2)

- Gestion des JavaBeans

```
<managed-bean>
  <managed-bean-name>laPersonne</managed-bean-name>
  <managed-bean-class>lib.PersonBean</managed-bean-class>
  <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
```

Valeur par défaut d'une propriété de bean

```
<managed-bean>
  <managed-bean-name>laPersonne</managed-bean-name>
  <managed-bean-class>lib.PersonBean</managed-bean-class>
  <managed-bean-scope>request</managed-bean-scope>
  <managed-property>
     <property-name>nom</property-name>
     <value>Dupont</value>
  </managed-property>
</managed-bean>
```

Lire en Java des beans

Lire un bean en java (1)

- Sans création du bean (peut rendre null)

```
FacesContext context =
  FacesContext.getCurrentInstance();
ExternalContext
   ctx=context.getExternalContext();
Map map = ctx.getApplicationMap();
Message mess = (Message)map.get("mess");
```


Lire un bean en java (2)

- Avec création du bean si nécessaire

```
FacesContext context =
  FacesContext.getCurrentInstance();
Application app = context.getApplication();
VariableResolver resolver =
  app.getVariableResolver();
Message mess = (Message)resolver.
  resolveVariable(context, "mess");
```


Evaluer une expression en Java

- Avec création du bean si nécessaire

```
FacesContext context =
  FacesContext.getCurrentInstance();
Application app = context.getApplication();
ValueBinding binding =
  app.createValueBinding("#{" + expr + "}");
Object value = binding.getValue(context);
```

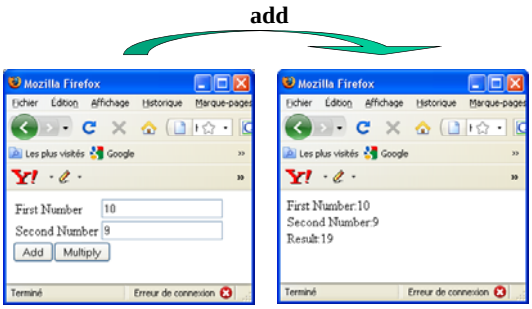

Message de log

```
FacesContext.getCurrentInstance().
     getExternalContext().log("un message");
```


Modèle MVC

Un exemple

Calculateur



Modèle

```
public class Calculator {

    public int add(int a, int b) {
        return a + b;
    }

    public int multiply(int a, int b) {
        return a * b;
    }

}
```


Controleur

Controleur 1

```
public class CalculatorController {

    private Calculator calculator = new Calculator();

    private int firstNumber = 0;
    private int secondNumber = 0;
    private int result = 0;

    public CalculatorController() {
        super();
    }
…
```


Controleur 2

```
public void setFirstNumber(int aFirstNumber) {
    this.firstNumber = aFirstNumber;
}
public int getFirstNumber() {
    return firstNumber;
}

public int getResult() {
    return result;
}

public void setSecondNumber(int aSecondNumber) {
    this.secondNumber = aSecondNumber;
}
public int getSecondNumber() {
    return secondNumber;
}
```


Controleur 3

```
public String add() {
    result = calculator.add(firstNumber,
                secondNumber);
    return "success";
}
public String multiply() {
    result = calculator.multiply(firstNumber,
                secondNumber);
    return "success";
}
```

Bean supervisé

```
<managed-bean>
  <managed-bean-name>
    CalcBean
  </managed-bean-name>
  <managed-bean-class>
    fr.ifsic.test.CalculatorController
  </managed-bean-class>
  <managed-bean-scope>
    session
  </managed-bean-scope>
</managed-bean>
```


Règle de navigation

```
<navigation-rule>
  <from-view-id>
     /calculator.jsp
  </from-view-id>
  <navigation-case>
   <from-outcome>success</from-outcome>
   <to-view-id>/result.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
```


calculator.jsp

calculator.jsp 1

```
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:h="http://java.sun.com/jsf/html"
xmlns:f="http://java.sun.com/jsf/core">
  <f:view>
    <h:form id="calcForm">
…
    </h:form>
  </f:view>
</html>
```


calculator.jsp 2

```
<h:panelGrid columns="3">
  <h:outputLabel value="First Number"/>
  <h:inputText id="firstNumber"
  value="#{CalcBean.firstNumber}"
  required="true" />
  <h:message for="firstNumber" />

  <h:outputLabel value="Second Number"/>
  <h:inputText id="secondNumber"
  value="#{CalcBean.secondNumber}"
  required="true" />
  <h:message for="secondNumber" />
</h:panelGrid>
```


calculator.jsp 3

```
<h:panelGroup>
    <h:commandButton id="submitAdd"
  action="#{CalcBean.add}"
  value="Add" />
    <h:commandButton
  id="submitMultiply"
  action="#{CalcBean.multiply}"
  value="Multiply" />
</h:panelGroup>
```

result.jsp

result.jsp 1

```
<f:view>
  First Number: <h:outputText
  id="firstNumber"
  value="#{CalcBean.firstNumber}"/>
  <br />
  Second Number: <h:outputText
  id="secondNumber"
  value="#{CalcBean.secondNumber}"/>
  <br />
  Result: <h:outputText id="result"
  value="#{CalcBean.result}"/>
  <br />
</f:view>
```


Langage d'expression

Langage d'expression « *EL* »

Valeur	Expression
Booléen	`cart.numberOfItems>0`
Élément de tableau	`books[2]`
Élément dans une collection	`books["fiction"]`
Propriété d'un Java bean	`laPersonne.nom`
Propriété d'un objet dans un tableau	`books[3].prix`
Propriété initialisée d'un paramètre `init` du contexte	`initParam.quantite`

Liste des objets implicites

`applicationScope`	Attributs de l'application (`Map`)
`cookie`	Cookies (`Map`)
`faceContext`	Instance courante du contexte
`header`	HTTP headers (`Map`)
`headerValues`	HTTP headers (`Map of String[]`)
`initParam`	Paramètres d'initialisation de l'appli
`param`	Paramètres de la requête (`Map`)

Liste des objets implicites (2)

`requestScope`	Attributs de la requête (`Map`)
`sessionScope`	Attributs de la session (`Map`)
`view`	La racine de l'arborescence des composants pour cette requête

Opérateurs

`+ - * / % div mod`	opérateurs arithmétique
`< <= > >= == != lt le gt ge eq ne`	opérateurs de comparaisons
`&& || ! and or not`	opérateurs logiques
`Empty`	un objet null, une chaîne vide, un tableau ou une collection sans élement,
`…? … : …`	opérateur ternaire de test

Internationalisation

Gestion de la langue
Gestion des dates/heures
….

Que peut-on internationaliser

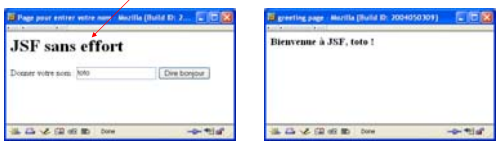
- Messages (d'état ou d'erreur)
- Labels (simples ou sur les composants graphiques)
- Dates et heures
- Nombres, valeurs monétaires
- Numéros de téléphone, adresses
- Formules de politesse
- Graphiques, images, couleurs, sons, …
- Et même la disposition des pages !

Internationaliser les chaînes ?

En anglais **En français**

Messages_fr.properties

```
entete_champNom = JSF sans effort
prompt = Donner votre nom :
signe = !
texteDeBienvenue = Bienvenue à JSF
texteDuBouton = Dire bonjour
```



Messages_en.properties

```
entete_champNom = JSF KickStart
prompt = Tell us your name
signe = !
texteDeBienvenue = Welcome to JSF,
texteDuBouton = Say Hello
```

<f:loadBundle> <h:outputText>

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<f:loadBundle basename="lib.messages" var="msg"/>

<html>
 <head>
  <title>Page pour entrer votre nom</title>
 </head>
 <body>
   <f:view>
     <h1><h:outputText value="#{msg.entete_champNom}"/></h1>

      ... fomulaire ...

   </f:view>
 </body>
</html>
```

Lecture des messages

Utilisation d'un des messages

Fichier de configuration

- Les langues dans le fichier de configuration

```
<faces-config>
...
  <application>
    <locale-config>
      <default-locale>fr</default-locale>
      <supported-locale>en</supported-locale>
    </locale-config>
  </application>
...
</faces-config>
```


Recherche d'un fichier de propriétés

- Lorsque l'utilisateur donne un ordre de préférences (dans son navigateur)
  - Pour chaque préférence prise dans l'ordre de l'utilisateur java cherche le bon fichier dans l'ordre :
  1. Resources_**language_country_variant**.properties
  2. Resources_**language_country**.properties
  3. Resources_**language**.properties
  4. Resources.properties

<h:outputFormat> Message paramètré

- Déclaration dans le fichier de ressources

```
mess = bonjour Mr {0}
```

- Utilisation dans un composant JSF

```
<h:outputFormat value="#{msg.mess}">
  <f:param value="#{myBean.nom}"/>
</h:outputFormat>
```


Chargement d'un fichier de `properties` par programme

- Les fichiers de messages `/games/messsages.properties` peuvent être captés par progamme comme suit : :

```
FacesContext context =
  FacesContext.getCurrentInstance();
ResourceBundle msg =
  ResourceBundle.getBundle("game.messages",
     context.getViewRoot().getLocale());
String texte = msg.getString("how_to_play");
```


Sortie conditionnelle

- Attribut `rendered`

```
<h:outputText value="Je vous ai
  reconnu"
  rendered="#{PersonneBean.user
  Name=='bekkers'}"/>
```

Librairie « core » (1)

`f:view, f:subview`	Vue principale, sous-vue
`f:facet`	Qualifier un élément
`f:attribute`	Ajouter un attribut, à un composant
`f:param`	Passer un paramètre à un composant
`f:loadBundle`	charger un fichier de properties
`f:verbatim`	ajouter un texte brut à la vue
`f:selectItem`	Un élément dans un composant à choix
`f:selectItems`	Groupe d'éléments dans un composant à choix

Librairie « core » (2)

`f:actionListener`	Ajouter un « *listener d'action* » à composant
`f:convertDateTime`	Conversion date/heure
`f:convertNumber`	Conversion numérique
`f:validator`	Ajouter un validateur spécifique
`f:validateDoubleRange`	Validation « *plage de réels* »
`f:validateLength`	Validation « *longueur de valeur* »
`f:validateLongRange`	Validation « *plage d'entiers* »
`f:valueChangeListener`	Associer un « *listener de changement de valeur* » à un composant

`<f:setPropertyActionListener>`

- Il doit obligatoirement être défini comme fils d'un composant d'action `<h:commandLink>`,`<h:commandButton>`.
- Affecter une valeur à une cible avant l'exécution d'une action donnée
- Deux attributs
  - **`target`** Le champ cible (à qui la valeur sera affectée)
  - **`value`** La valeur à affecter au champ cible

Ajouter des objets aux composants html

- La plupart des tags de la librarie « core » ajoutent des objets aux composants « html »
  - Attribute
  - Listener
  - Converter
  - Validator
  - Facet
  - Parameter
  - Select item

Librairie html

`<h:outputText>` `<h:graphicImage>`

```
<h:outputText
value="#{form.testString}"/>
```

1234567890123456789O

```
<h:outputText value="Number
#{form.number}"/>
```

Number 1000

```
<h:outputText value="<input type='text'
value='hello'/>"/>
```

hello

```
<h:outputText escape="true"
value="<input type='text'
value='hello'/>"/>
```

<input type="text" value="hello">

```
<h:graphicImage
value="/tjefferson.jpg"/>
```

```
<h:graphicImage value="/tjefferson.jpg"
style="border: thin solid black"/>
```

<h:outputFormat> <f:param>
Texte paramètré

```
<h:outputFormat value="{0} est
  agé de {1} ans">
    <f:param value="Jaques"/>
    <f:param value="25"/>
</h:outputFormat>
```


<h:panelGroup>

Regroupements

```
<h:panelGroup>
  <h:outputText value="one row"  />
  <h:outputText value=" "  />
  <h:outputText
       value="grouped with panelGroup"/>
</h:panelGroup>
```

one row grouped with panelGroup

Affichage conditionnel d'un groupe

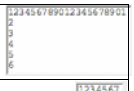
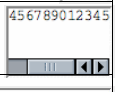
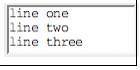
- Attribut **@renderer**

```
<h:panelGroup rendered="#{myBean.number>10}">
  <h:outputLabel id="isl2" for="is2">
      <h:outputText value="blabla"/>
  </h:outputLabel>
  <h:inputText id="is2"/>
  <h:commandButton value="Go!"/>
</h:panelGroup>
```


<h:inputText> <h:inputSecret>

`<h:inputText value="#{form.testString}" readonly="true"/>`	
`<h:inputSecret value="#{form.passwd}" redisplay="true"/>`	
`<h:inputSecret value="#{form.passwd}" redisplay="false"/>`	
`<h:inputText value="inputText" style="color: Yellow; background: Teal;"/>`	
`<h:inputText value="1234567" size="5"/>`	
`<h:inputText value="1234567890" maxlength="6" size="10"/>`	

<h:inputTextarea>

`<h:inputTextarea rows="5"/>`	
`<h:inputTextarea cols="5"/>`	
`<h:inputTextarea value="123456789012345" rows="3" cols="10"/>`	
`<h:inputTextarea value="#{form.dataInRows}" rows="2" cols="15"/>`	

<h:commandButton> <h:commandLink>

Navigation JSF	
`<h:commandButton value="submit" type="submit"/>`	
`<h:commandLink> <h:outputText value="register"/> </h:commandLink>`	
`<h:commandLink> <h:outputText value="#{msgs.link}"/> <h:graphicImage value="/regis.jpg"/> </h:commandLink>`	

<h:outputLink>

Liens html simples

```
<h:outputLink value="#introduction">
   <h:outputText value="Introduction"
style="font-style: italic"/>
</h:outputLink>
```

Introduction

```
<h:outputLink value="#toc"
   title="Go to the table of contents">
  <f:verbatim>
   <h2>Table of Contents</h2
  </f:verbatim>
</h:outputLink>
```

Table of Contents

<h:panelGrid> Tableaux

Classes CSS

```
<h:panelGrid columns="4" footerClass="subtitle"
 headerClass="subtitlebig" styleClass="medium"
 columnClasses="subtitle,medium">
 <f:facet name="header">
   <h:outputText value="Table with numbers"/>
 </f:facet>
 <h:outputText value="1" /><h:outputText value="2" />
 <h:outputText value="3" /><h:outputText value="4" />
 <h:outputText value="5" /><h:outputText value="6" />
 <h:outputText value="7" />
 <f:facet name="footer">
    <h:outputText value="blabla" />
   </h:panelGroup>
 </f:facet>
</h:panelGrid>
```


<h:panelGrid> rendu

Table with numbers			
blabla			
1	2	3	4
5	6	7	

<h:dataTable> <h:column>

```
<h:dataTable id="books" value="#{BookStore.items}"
  var="store">
 <h:column>
   <f:facet name="header">
     <h:outputText  value="#{msg.storeNameLabel}"/>
   </f:facet>
   <h:outputText value="#{store.name}"/>
 </h:column>
 <h:column>
   <f:facet name="header">Subject</f:facet>
   <h:outputText value="#{store.subject}"/>
 </h:column>
 <h:column>
   <f:facet name="header">
     <h:outputText  value="#{msg.storePriceLabel}"/>
   </f:facet>
    <h:outputText value="#{store.price}"/>
 </h:column>
</h:dataTable>
```

List, tableau …

<h:dataTable> <h:column> rendu

Title	Subject	Price ($)
JSF For Dummies	JSF	25.0
Struts For Dummies	Struts	22.95

Attribut @value d'un élément <h:dataTable>

- @value représente une collection sur la quelle l'itération sur les lignes du tableau porte
- @var contient le nom de la variable courante d'itération

```
<h:dataTable id="books"
    value="#{BookStore.items}"
    var="store">
…
 </h:dataTable>
```

Collections autorisées

- Doit être un des types suivant
  - Un tableau (*ex* `String[], Integer[], …`)
  - `java.util.List`
  - `java.sql.ResultSet`
  - `java.servelt.jsp.jstl.sql.Result`
  - `javax.faces.model.DataModel`

`<f:verbatim>` spécifier du texte comme contenu d'un élément JSF

```
<h:column>
  <f:verbatim>Mr </f:verbatim>
  <h:outputText value="#{name.last}"/>
</h:column>
```


`<f:facet name="header">`

- Entêtes de colonne

```
<h:column>
  <f:facet name="header">
    <h:outputText value="#{msg.nom}"/>
  </f:facet>
  <h:commandLink id="link"
                 action="#{p.go}">
    <h:outputText value="#{p.nom}"/>
  </h:commandLink>
</h:column>
```


Lien dans une ligne de tableau

```
<h:column>
  ...
  <h:commandLink id="link" action="#{p.go}">
    <h:outputText value="#{p.nom}"/>
  </h:commandLink>
</h:column>
```

- Récupération par programme de la ligne
  - utiliser le type `DataModel`

```
javax.faces.model.DataModel values;
public String go() {
   ...
   Couleur c = values.getRowData();
   ...
}
```

Couleur
bleu
blanc
rouge

Initialisation d'un `DataModel`

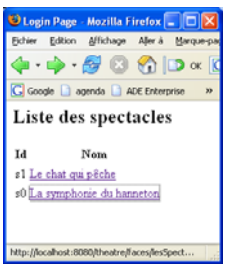
- Création à partir d'une liste

```
List<Couleur> liste =
      new ArrayList<Couleur>();
liste.add(new Couleur("bleu"));
liste.add(new Couleur("blanc"));
liste.add(new Couleur("rouge"));
DataModel values = new DataModel(liste);
public DataModel getValues() {…}
```

- Liaison dans la page JSF

```
<h:dataTable id= "couleurs"
   value="#{lesCouleurs.values}"
   var="couleur">
…
 </h:dataTable>
```


Lien direct dans une table



Lien direct dans une table
<h:outputLink>

```
<h:dataTable id="spectacles" var="spectacle"
                    value="#{lesSpectacles.valuesModel}">
  <h:column>
    <f:facet name="header">
      <h:outputText value="#{msg.id}" />
    </f:facet>
    <h:outputText value="#{spectacle.id}" />
  </h:column>
  <h:column>
    <f:facet name="header">
      <h:outputText value="#{msg.nom}" />
    </f:facet>
    <h:outputLink  id="link" value="dir/#{spectacle.id}.html">
      <h:outputText value="#{spectacle.nom}" />
    </h:outputLink>
  </h:column>
</h:dataTable>
```


Gestion de collections

<h:selectManyCheckbox>
<f:selectItem>

<h:selectManyCheckbox>
<f:selectItem>

- Valeurs `String[]`

```
<h:form id="form">
   <h:messages style="color:red;" />
   <h:selectManyCheckbox id="couleur"
              value="#{selection.values}">
      <f:selectItem itemValue="bleu" itemLabel="Bleu"/>
      <f:selectItem itemValue="blanc" itemLabel="Blanc"/>
      <f:selectItem itemValue="rouge" itemLabel="Rouge"/>
   </h:selectManyCheckbox>
   <br />
   <h:commandButton value="Soumettre" id="submit"
  action="show" />
</h:form>
```

Valeurs de retour

- Bean Java

```
private String[] values;
public void setValues(String[] vals) {values = vals;}
public String[] getValues() {return values;}
```


<h:selectManyCheckbox>
<f:selectItems>

- Valeurs `String[]`

```
<h:selectManyCheckbox id="couleur"
              value="#{selection.values}">
   <f:selectItems value="#{selection.couleurs}"/>
</h:selectManyCheckbox>
```

- Bean Java

```
private SelectItem[] couleurs = {
   new SelectItem("bleu", "Bleu"),
   new SelectItem("blanc", "Blanc"),
   new SelectItem("rouge", "Rouge")
};
public SelectItem[] getCouleurs() {return couleurs;}
public void setCouleurs(SelectItem[] couleurs) {
   this.couleurs = couleurs;
}
```


Composants de l'interface

```
<f:view>
<h:form id="form">
   <h:messages style="color:red;" />
   <h:selectManyCheckbox id="couleur"
              value="#{selection.values}">
      <f:selectItem itemValue="bleu" itemLabel="Bleu"/>
      <f:selectItem itemValue="blanc" itemLabel="Blanc"/>
      <f:selectItem itemValue="rouge" itemLabel="Rouge"/>
   </h:selectManyCheckbox>
   <br />
   <h:commandButton value="Soumettre" id="submit"
  action="show" />
</h:form>
</f:view>
```

`f:view`

`h:form`

`h:message` `h:selectManyCheckbox` `h:commandButton`

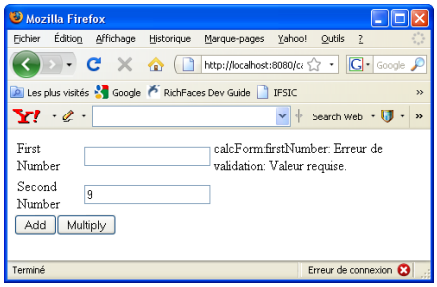
Parcourir l'interface par programme

```
private UISelectMany many;
private UISelectMany getMany() {
   if (many == null) {
      UIViewRoot root =FacesContext.
               getCurrentInstance().getViewRoot();
      UIForm form = (UIForm)root.getChildren().get(0);
      many = (UISelectMany)form.getChildren().get(1);
   }
   return many;
}
...
String[] selected = (String[]) getMany().getSelectedValues();
for (int i = 0; i < selected.length; i++) {
   System.out.println(selected[i]);
}
```

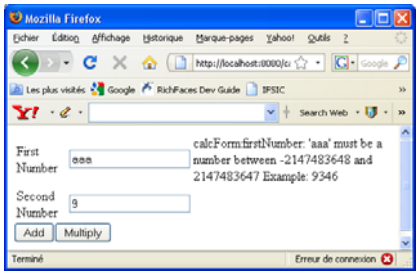
f:view
h:form
h:message h:selectManyCheckbox h:commandButton

Erreurs

Valeur requise



Erreur de conversion



Validation – récupération d'erreurs

```
<f:view>
  <h1>
    <h:outputText value="#{msg.entete_champNom}"/>
  </h1>
  <br/><h:messages style="color: red"/><br/>

  <h:form id="helloForm">
    <h:outputText value="#{msg.prompt}"/>
    <h:inputText value="#{laPersonne.nom}"
              required="true"/>
    <h:commandButton action="greeting"
              value="#{msg.texteDuBouton}" />
  </h:form>
</f:view>
```

<h:messages> <h:message>

- Avertissements collectifs
  `<h:messages style="color:red"/>`
- Avertissements individuels (utilise les identifications)
  - Attributs *id* et *for*

```
<h:inputText id="password" size="15"
  required="true"
    value="#{LoginServer.password}" />
<h:message style="color:red"
     for="password"/>
```

Conversion et validation des données saisies

<f:convertNumber> (1)

- *Valeur monétaire*

```
<h:outputText value="#{convert.prix}">
    <f:convertNumber type="currency"/>
</h:outputText>
```

  En interne `float getPrix() : 1.23f`

  Affichage `1,23 €`

- *Numérique* `<f:convertNumber type="number"/>`

  En interne `int getPoids() : 12`

  Affichage `12`

<f:convertNumber> (2)

- *Pourcentage* `<f:convertNumber type="percent"/>`

  En interne `float getRatio() : 0.5f`

  Affichage `0,5 %`

- *Entier* nombre maximum de chiffres

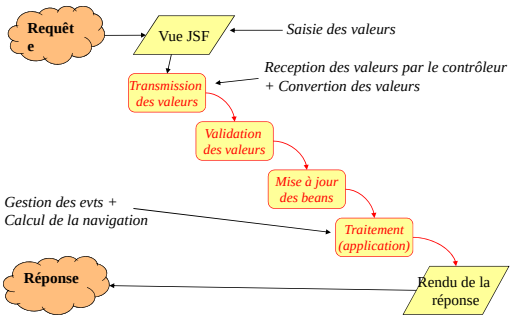
```
<f:convertNumber integerOnly="true"
  maxIntegerDigits="2"/>
```

  En interne `float getPrix() : 1234.56f`

  Affichage `34.56`

- `<f:convertNumber pattern="#.#"/>`

Cycle de vie d'une requête JSF



Différentes méthodes de validation

- Validateurs prédéfinis
  - validateDoubleRange
  - validateLength
  - validateLongRange
- Validateurs définis par l'usager
  - Implémenter l'interface `Validator`
- Validation dans les beans gérés
  - Implémenter des méthodes de validation

Validateurs prédéfinis

- Longueur d'une chaîne

```
<h:inputText value="#{laPersonne.nom}"
      required="true">
   <f:validateLength minimum="2" maximum="10" />
</h:inputText>
```

- Valeurs numériques bornées

```
<h:inputText value="#{laPersonne.age}"
      required="true">
   <f:validateLongRange minimum="18" maximum="120" />
</h:inputText>
```

Validateur défini par l'utilisateur

- Dans le formulaire

```
<h:inputText value="#{laPersonne.nom}"
      required="true">
   <f:validator validatorId="nameValidator"/>
</h:inputText>
```

- Dans le fichier de configuration

```
<validator>
  <validator-id>nameValidator</validator-id>
  <validator-class>
    fr.ifsic.tpJsf.NameValidator</validator-class>
</validator>
```


Implémenter l'interface `Validator`

```
public class NameValidator implements Validator {
   private static final String NOM_REGEX =
      "([a-z]|[A-Z]){1}(('|\\ |\\-)?([a-z]|[A-Z])+)*";
   public void validate(FacesContext arg0,
      UIComponent arg1, Object value)
            throws ValidatorException {
       Pattern mask = Pattern.compile(NOM_REGEX);
       String nom = (String)value;
       Matcher matcher = mask.matcher(nom);
       if (!matcher.matches()){
          FacesMessage message = new FacesMessage();
          message.setDetail("Name not valid");
          message.setSummary("Name not valid");
          message.setSeverity(
            FacesMessage.SEVERITY_ERROR);
          throw new ValidatorException(message);
       }
   }
}
```


Validation dans un bean (1)

- **Au sein du formulaire**

String

```
<h:inputText id="email"
   value="#{UserRegistration.user.email}"
   validator="#{UserRegistration.validateEmail}"
   required="true">
</h:inputText>
```

- **Au sein du bean `UserRegistration`**
  - Programmer une méthode

```
public void validateEmail(FacesContext context,
                          UIComponent toValidate,
                          Object value)
```


Validation dans un bean (2)

- **Au sein du bean**

```
public void validateEmail(FacesContext context,
                          UIComponent toValidate,
                          Object value) {
   String email = (String) value;
   if (email.indexOf('@') == -1) {
      ((UIInput)toValidate).setValid(false);
      FacesMessage message =
         new FacesMessage("Invalid Email");
      context.addMessage(
         toValidate.getClientId(context), message);
    }
}
```


Instancier un attribut de bean

- Instancier un attribut de bean sur un click de bouton

```
<h:commandLink action="edit"
  styleClass="editButton">
    <f:setPropertyActionListener
  target="#{fxContentViewBean.content}"

  value="#{fxSystemBean.content[row[0]]}"/>
    Edit...
</h:commandLink>
```


Mélanger éléments JSF et non JSF

- Interdit de placer un élément JSF dans un élément non JSF qui effectue des itérations

```
<ul>
  <c:forEach items="${books}" var="b">
    <li><h:outputText value="#{b}"/></li>
  </c:forEach>
</ul>
```

- Autorisé dans un `<c:if>` ou `<c:choose>`
  - Mais : les composants JSF internes doivent être identifiés !
  - D'autres problèmes peuvent apparaître …

Mélange d'éléments JSF et de texte

- Premier cas

```
<h:outputText value="Some text" />
Some more text
```

Some text
Some more text

- Second cas

```
<h:panelGroup>
  <h:outputText value="Some text" />
  Some more text
</h:panelGroup>
```

Some more text
Some text

Fragments de pages

- Fabriquer une page à l'aide de fragments agglomérés
  - modularité, réutilisation de code
- Chargement dynamique
  - `<jsp:include>, <c:import>`
- Chargement statique (méthode préférée)
  - `<%@ include file="relative url"%>`

Chargement dynamique

- Page principale

```
<f:view>
   ...
   <jsp:include page="foo.jsp"/>
   ...
</f:view>
```

La page inclue doit être dans un élément `<f:subview>`

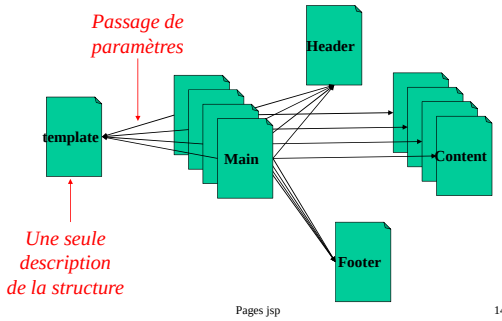
- Page `Foo.jsp`

```
<f:subview>
   <h:outputText value="heyah!"/>
   ...
   <f:verbatim>
      <b>Template text.</b>
      <customtag:dothis/>
   </f:verbatim>
</f:subview>
```

Les éléments non JSF doivent être inclus dans des éléments `<f:verbatim>`

Facelets

Structurer un site WEB



facelets

- Composition (appel de procédure)

```
<ui:composition
  template="myTemplate.jsp">
```

- Paramêtres effectifs

```
<ui:define>
```

- Paramêtres formels (modèle de page)

```
<ui:insert>
```

Modèle de page

```
<div id="header">
    <ui:insert name="header">
      <ui:include src="header.xhtml"/>
    </ui:insert>
</div>


<div id="left">
  <ui:insert name="navigation" >
    <ui:include src="navigation.xhtml"/>
  </ui:insert>
</div>
```


Composition

```
<ui:composition template="layout.xhtml">
  <ui:define name="title">CD form</ui:define>
  <ui:define name="content">
     <h:form id="cdForm">
   …
     </h:form>
  </ui:define>
</ui:composition>
```


Outils

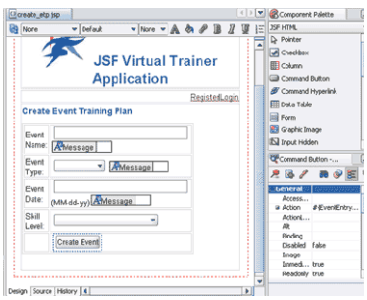
Des librairies JSF …

- La dernière version officielle de Sun
  http://java.sun.com/j2ee/javaserverfaces
- Myfaces d'Apache
  http://myfaces.apache.org/
- Oracle ADF Faces
  http://www.oracle.com/technology/products/jdev/htdocs/partners/addins/exchange/jsf/index.html
- JScape WebGalileo Faces
  http://www.jscape.com/webgalileofaces/dload.html

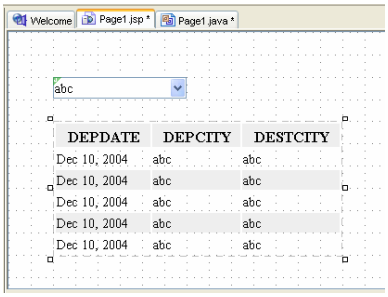
Editeurs JSF graphiques

- Oracle JDeveloper 10.1.3
  - http://www.oracle.com/technology/products/jdev/index.html
- Sun Java Studio Creator's JSF Visual Editor
  - http://developers.sun.com/prodtech/javatools/jscreator/
- IBM Websphere Application Developer's JSF enabled JSP Visual editor
  - http://www-128.ibm.com/developerworks/websphere/techjournal/0401_barcia/barcia.html
- MyEclise's JSF Visual Editor
  - http://www.myeclipseide.com/ContentExpress-display-ceid-54.html
- JBuilder de Borland
  - http://www.borland.com/us/products/jbuilder/index.html
- JavaServer Faces Tooling Project (*gratuit, à venir*)
  - http://www.eclipse.org/proposals/eclipse-jsf/

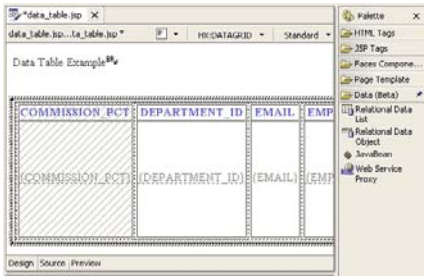
Oracle JDeveloper 10.1.3



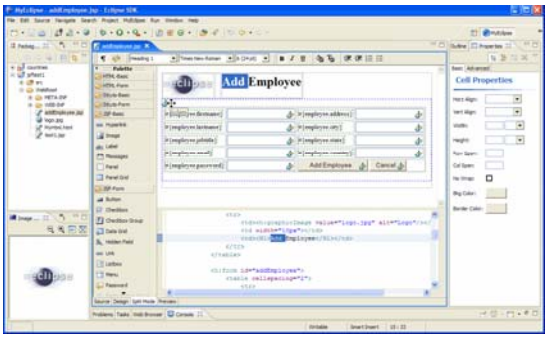
Sun Java Studio Creator's JSF Visual Editor



IBM Websphere Application Developer



MyEclise



Inconvénients de JSF

- JSF utilise POST uniquement.
  - Impossibilité de garder la référence d'une page
- Les noms de fichiers sont différents des URL
  - Noms fichier se terminent par *.JSP
  - URL se terminent par *.JSF
- Pas de possibilité d'utiliser javascript
  - Validation coté client impossible

Technologies MVC concurrentes

- Struts (le plus ancien) tend à être remplacé par les autres – JSF est dit plus *flexible*
  - http://struts.apache.org/
- Spring (le tout dernier) à voir …
- pour la petite histoire, JSF est fortement inspiré des *WebForms* du framework Microsoft *ASP.NET*

Références

Un tableau de référence sur les composants graphiques JSF

Tableau de référence des composant JSF html

- http://www.exadel.com/tutorial/jsf/jsftags-guide.htmlréférence

Quelques Articles

- “A first look at JavaServer Faces” by David Geary
  http://www.javaworld.com/javaworld/jw-11-2002/jw-1129-jsf.html
- “Developing Web Applications with JavaServer Faces” by Qusay H. Mahmoud
  http://developer.java.sun.com/developer/technicalArticles/GUI/JavaServerFaces/
- “JavaServer Faces: A standard-based solution for Java Web applications” by Murali Kaundinya & Jamiel Sheikh
  http://sys-con.com/java/source.cfm?id=1991
- "Putting a New Face on Web Interfaces" by Peter Varhol
  http://www.fawcette.com/javapro/2003_04/magazine/columns/weblication/
- "Introducing JavaServer Faces" by Budi Kurniawan
  http://www.onjava.com/pub/a/onjava/2003/07/30/jsf_intro.html

Autres références

- Rappel : Tutorial de sun (1500 pages)
  - (pdf) http://java.sun.com/j2ee/1.4/docs/tutorial-update2/doc/J2EETutorial.pdf
- Un tutorial Exadel
  - http://www.exadel.com/tutorial/jsf/jsftutorial-kickstart.html
- James Holmes Java Server Faces Links
  - http://www.jamesholmes.com/JavaServerFaces/
- Java Server Faces central
  - http://www.jsfcentral.com/
- La page officielle JSF chez Sun
  - http://java.sun.com/j2ee/javaserverfaces/