



Séance 2

Java Server Faces – Partie 1

David Vaillant – 2017/18

david.vaillant@capgemini.com

Introduction : qu'est-ce-que c'est ...

Java Server Faces est un framework de développement d'applications Web en Java permettant de respecter le modèle d'architecture MVC et basé sur des composants côté présentation

Java Server Faces permet

- une séparation de la couche présentation des autres couches (MVC)

- un mapping entre l'HTML et l'objet

- un ensemble de composants riches et réutilisables

- une liaison simple entre les actions côté client de l'utilisateur et le code Java côté serveur

- Création de nouveaux composants graphiques

Introduction : qu'est-ce-que c'est ...

Construit sur le pattern MVC

Notion de « backing bean » = objet technique associé à une page ou un composant graphique

Page : document XHTML, avec des éléments scriptés

Introduction : une specification et plusieurs implémentations

Il existe plusieurs implémentations de JSF

Oracle Mojarra : <https://javaee.github.io/javaxserverfaces-spec/>

Apache MyFaces : <http://myfaces.apache.org>

Introduction : principe pour traiter un formulaire

1. Construire le formulaire dans une page HTML en utilisant les balises JSF
2. Développer un Backed Bean qui effectue un « Mapping » avec les valeurs du formulaire
3. Modifier le formulaire pour spécifier l'action et l'associer au Bean
4. Fournir des Converters et des Validators pour traiter les données du formulaire
5. Créer les pages XHTML correspondant à chaque condition de retour

Configuration : JSF dans le web.xml 1/3

Nécessite la configuration du fichier **web.xml** de façon à ce que JSF soit pris en compte

Paramétrer le fonctionnement général de l'application : le contrôleur

Identifier la servlet principale : `javax.faces.webapp.FacesServlet`

Valider ou pas les fichiers XML

Nom du paramètre : `com.sun.faces.validateXml`

Valeurs possibles : `true` ou `false` (défaut : `false`)

Configuration : JSF dans le web.xml 2/3

Indique si les objets développés tels que les Beans, les composants, les validators et les converters doivent être créés au démarrage de l'application

Nom du paramètre : `com.sun.faces.verifyObjects`

Valeurs possibles : `true` ou `false` (défaut : `false`)

La servlet principale est le point d'entrée d'une application JSF

On trouve plusieurs manières de déclencher des ressources JSF

Préfixe `/faces/`

Suffixes `*.jsf` ou `*.faces`

Exemples (le contexte de l'application est `myAppli`)

`http://localhost/myAppli/faces/index.jsp`

`http://localhost/myAppl/index.jsf`

Configuration : JSF dans le web.xml 3/3

Exemple : paramétrer une application Web de type JSF

Utilisation de *context-param* pour paramétrer le fonctionnement des JSF

```
...  
<context-param>  
  <param-name>com.sun.faces.validateXml</param-name>  
  <param-value>>true</param-value>  
</context-param>  
<servlet>  
  <servlet-name>Faces Servlet</servlet-name>  
  <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>  
  <load-on-startup>1</load-on-startup>  
</servlet>  
<servlet-mapping>  
  <servlet-name>Faces Servlet</servlet-name>  
  <url-pattern>/faces/*</url-pattern>  
</servlet-mapping>  
<servlet-mapping>  
  <servlet-name>Faces Servlet</servlet-name>  
  <url-pattern>*.faces</url-pattern>  
</servlet-mapping>  
...
```

La Servlet qui gère les entrées au contexte JSF

Comment accéder à la Servlet « Faces Servlet »

web.xml

Configuration : balises personnalisées dans les JSP

Les composants JSF sont utilisés dans les pages HTML au moyen de balises personnalisées dédiées aux JSF

CORE : noyau qui gère les vues

HTML : composants JSF

Possibilité d'utiliser d'autres balises (Tomahawk, richfaces ...)

Les composants JSF doivent être encadrés par une vue JSF déclarée par la balise `<core:view>`

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

Utilisation des composants JSF

Le premier exemple « Hello World avec JSF »

Exemple : afficher le message « Hello World » avec JSF
welcomeJSF.xhtml du projet **HelloWorld**

```
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:core="http://java.sun.com/jsf/core"
xmlns:html="http://java.sun.com/jsf/html"
xmlns:ui="http://java.sun.com/jsf/facelets">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Hello World avec JSF</title>
  </head>
  <body>
    <core:view>
      <h1><html:outputText value="Hello World avec JSF" /></h1><br>
      La même chose avec du HTML : <h1>Hello World avec JSF</h1>
    </core:view>
  </body>
</html>
```



Le premier exemple « Hello World avec JSF »

- web.xml du projet **HelloWorld**

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/webapp_2_4.xsd">
  <servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>/faces/*</url-pattern>
  </servlet-mapping>
</web-app>
```

Bean Managé : principe

Rappelons qu'un Bean est une classe Java respectant un ensemble de directives

- Un constructeur public sans argument

- Les propriétés d'un Bean sont accessibles au travers de méthodes getXXX et setXXX portant le nom de la propriété

L'utilisation des Beans dans JSF permet

- l'affichage des données provenant de la couche métier

- le stockage des valeurs d'un formulaire

- la validation des valeurs

Bean managé : configuration par Annotations

Pour créer un Bean managé on utilise l'annotation `@ManagedBean` en déclarant dans l'attribut *name* son nom.

On utilise une autre annotation pour déclarer le scope dans lequel il sera utilisé :

`@RequestScoped`

`@SessionScoped`

`@ApplicationScoped`

`@ConversationScoped` (géré partiellement par l'application, permet la navigation multionglet)

```
@ManagedBean(name = "helloWorld")
@RequestScoped
public class HelloWorld {
```

Cycle de vie du *backing bean*

Attacher un bean au
contexte « conversation »

```
@ConversationScoped
@Named("musicianController")
public class MusicianController implements Serializable {

    @Inject
    private Conversation conversation ;

    private Musician musician ;

    // class
}
```

Cycle de vie du *backing bean*

Initialiser le bean en fonction des paramètres

Création d'un nouveau musicien

```
@PostConstruct
private void init() {

    HttpServletRequest request =
        (HttpServletRequest)FacesContext.getCurrentInstance().getExternalContext().getRequest() ;
    String sID = request.getParameter("id") ;
    this.id = sID == null ? null : Long.parseLong(request.getParameter("id")) ;

    if (id == null) {
        musician = new Musician() ;
    } else {
        musician = musicianService.findById(id) ;
    }
    conversation.begin() ;
}
```

Cycle de vie du *backing bean*

Initialiser le bean en fonction des paramètres

Création d'une méthode @PostConstruct

```
@PostConstruct
private void init() {

    HttpServletRequest request =
        (HttpServletRequest)FacesContext.getCurrentInstance().getExternalContext().getRequest() ;
    String sID = request.getParameter("id") ;
    this.id = sID == null ? null : Long.parseLong(request.getParameter("id")) ;

    if (id == null) {
        musician = new Musician() ;
    } else {
        musician = musicianService.findById(id) ;
    }
    conversation.begin() ;
}
```


Cycle de vie du *backing bean*

Initialiser le bean en fonction des paramètres

Récupération du paramètre de requête ID

```
@PostConstruct
private void init() {

    HttpServletRequest request =
        (HttpServletRequest)FacesContext.getCurrentInstance().getExternalContext().getRequest() ;
    String sID = request.getParameter("id") ;
    this.id = sID == null ? null : Long.parseLong(request.getParameter("id")) ;

    if (id == null) {
        musician = new Musician() ;
    } else {
        musician = musicianService.findById(id) ;
    }
    conversation.begin() ;
}
```

Cycle de vie du *backing bean*

Initialiser le bean en fonction des paramètres

Création d'un nouveau musicien

```
@PostConstruct
private void init() {

    HttpServletRequest request =
        (HttpServletRequest)FacesContext.getCurrentInstance().getExternalContext().getRequest() ;
    String sID = request.getParameter("id") ;
    this.id = sID == null ? null : Long.parseLong(request.getParameter("id")) ;

    if (id == null) {
        musician = new Musician() ;
    } else {
        musician = musicianService.findById(id) ;
    }
    conversation.begin() ;
}
```

Cycle de vie du *backing bean*

Initialiser le bean en fonction des paramètres

Lecture d'un musicien en base

```
@PostConstruct
private void init() {

    HttpServletRequest request =
        (HttpServletRequest)FacesContext.getCurrentInstance().getExternalContext().getRequest() ;
    String sID = request.getParameter("id") ;
    this.id = sID == null ? null : Long.parseLong(request.getParameter("id")) ;

    if (id == null) {
        musician = new Musician() ;
    } else {
        musician = musicianService.findById(id) ;
    }
    conversation.begin() ;
}
```

Cycle de vie du *backing bean*

Initialiser le bean en fonction des paramètres

Démarrage de la conversation

```
@PostConstruct
private void init() {

    HttpServletRequest request =
        (HttpServletRequest)FacesContext.getCurrentInstance().getExternalContext().getRequest() ;
    String sID = request.getParameter("id") ;
    this.id = sID == null ? null : Long.parseLong(request.getParameter("id")) ;

    if (id == null) {
        musician = new Musician() ;
    } else {
        musician = musicianService.findById(id) ;
    }
    conversation.begin() ;
}
```

Cycle de vie du *backing bean*

Fin de la conversation

Création d'un musicien, fin de la conversation

```
public void create() {  
    musicianService.createMusician(musician) ;  
    conversation.end() ;  
}
```

Cycle de vie du *backing bean*

Fin de la conversation

Édition d'un musicien, fin de la conversation

```
public String edit() {  
    musicianService.updateMusician(musician) ;  
    conversation.end() ;  
    return "list-musician.xhtml" ;  
}
```

Cycle de vie du *backing bean*

Fin de la conversation

Édition d'un musicien, fin de la conversation

```
public String edit() {  
  
    musicianService.updateMusician(musician) ;  
    conversation.end() ;  
  
    return "list-musician.xhtml" ;  
}
```

Cycle de vie du *backing bean*

Fin de la conversation

Effacement d'un musicien, fin de la conversation

```
public String delete(Musician musician) {  
  
    musicianService.deleteMusician(musician.getId()) ;  
    conversation.end() ;  
  
    return "list-musician.xhtml" ;  
}
```


Bean managé : JSP, Bean et Expression Language (EL)

Un formulaire JSF doit être construit dans un groupe défini par la balise `<html:form> ... </html:form>`

ACTION est automatiquement à SELF (URL courante)

METHOD est obligatoirement POST

Utilisation de composants JSF pour saisir des informations

`<html:inputText>` pour la balise HTML `<INPUT TYPE="Text">`

`<html:inputSecret>` pour la balise `<INPUT TYPE="PASSWORD">`

`<html:commandButton>` pour la balise `<INPUT TYPE="SUBMIT">`

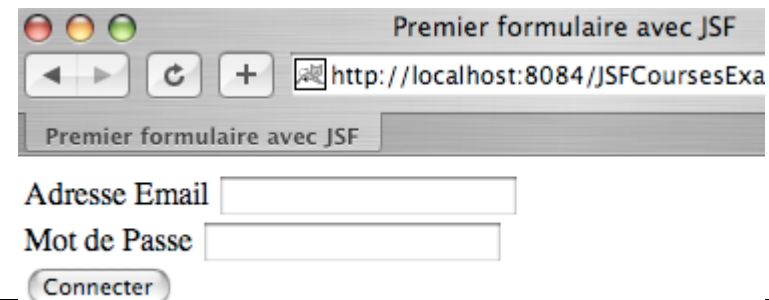
La balise `<html:commandButton>` contient un attribut `action` qui permet d'indiquer un message traité par les règles de navigation définies dans `faces-config.xml`

→ Nous verrons dans la partie Navigation le traitement de la valeur indiquée dans l'attribut `action`

Bean managé : JSP, Bean et Expression Language (EL)

Exemple : formulaire JSP utilisant des composants JSF beanform1.xhtml du projet **ManagedBean**

```
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:core="http://java.sun.com/jsf/core"
xmlns:html="http://java.sun.com/jsf/html"
xmlns:ui="http://java.sun.com/jsf/facelets">
<head>
<title>Premier formulaire avec JSF</title>
</head>
<body>
  <core:view>
    <html:form>
      <html:outputText value="Adresse Email " /><html:inputText/><br>
      <html:outputText value="Mot de Passe " /><html:inputSecret/><br>
      <html:commandButton value="Connecter" />
    </html:form>
  </core:view>
</body>
</html>
```



The screenshot shows a web browser window with the title "Premier formulaire avec JSF". The address bar displays the URL "http://localhost:8084/JSFCoursesExa". Below the browser window, the rendered form is visible, featuring two input fields: "Adresse Email" and "Mot de Passe", followed by a "Connecter" button.

Bean managé : JSP, Bean et Expression Language (EL)

Les Expressions Languages (EL) sont utilisées pour accéder aux éléments du Bean dans les pages JSP

Un EL permet d'accéder simplement aux Beans des différents scopes de l'application (page, request, session et application)

Forme d'un Expression Language JSF : `#{expression}`

Les EL JSF sont différentes des EL JSP qui utilisent la notation : `${expression}`

Une EL est une expression dont le résultat est évaluée au moment où JSF effectue le rendu de la page

Bean managé : JSP, Bean et Expression Language (EL)

L'écriture `#{MyBean.value}` indique à JSF de chercher un objet qui porte le nom de MyBean dans son contexte puis invoque la méthode `getValue()` (chercher la propriété value)

Le nom de l'objet est celui d'un Bean managé

Possibilité d'accéder à un objet contenu dans le Bean

`#{MyBean.myobject.value}` : propriété value de myobject contenu dans MyBean

Bean managé : JSP, Bean et Expression Language (EL)

JSF définit un ensemble d'objets implicites utilisables dans les Expressions Languages :

param : éléments définis dans les paramètres de la requête HTTP

param-values : les valeurs des éléments param

cookies : éléments définis dans les cookies

initParam : paramètres d'initialisation de l'application

requestScope : éléments défini dans la requête

facesContext : instance de la classe FacesContext

View : instance de UIViewRoot

Expliqué au niveau des
composants graphiques

Etudiée au niveau de la
partie cycle de vie

Bean managé : JSP, Bean et Expression Language (EL)

Exemple : objets implicites JSF dans une JSP

```
<html>
  <...>
  <body>
    <core:view>
      <p><html:outputText
        value="#{param.test}" /></p>
      <p><html:outputText
        value="#{cookie.JSESSIONID.value}" /></p>
      <p><html:outputText
        value="#{facesContext.externalContext.requestPathInfo}" /></p>
      <p><html:outputText
        value="#{facesContext.externalContext.requestServletPath}" /></p>
    </core:view>
  </body>
</html>
```

Utilisation des objets impli...

CCD5180840025B229F22EC3B0C235CAC

/converters1.jsp

/faces

Bean managé : modifier/afficher la propriété d'un Bean

Un formulaire regroupe un ensemble de composants contenant chacun une valeur (attribut value)

Chaque valeur d'un composant peut être stockée dans une propriété du Bean

Pour définir une propriété dans un Bean

- Créer l'attribut correspondant à la propriété

- Définir des méthodes set et get pour accéder en lecture et en écriture aux propriétés

Pour stocker la valeur d'un composant vers le Bean ou afficher la valeur d'une propriété dans un composant : utilisation des EL dans l'attribut value du composant JSF

JSF associe automatiquement la propriété d'un Bean à la valeur d'un composant

Bean managé : modifier/afficher la propriété d'un Bean

Exemple : gestion d'un formulaire « email et mot de passe »

```
package beanPackage;  
public class RegistrationBean {  
  
    private String email = "user@host";  
  
    private String password = "";  
  
    public String getEmail() {  
        return email;  
    }  
    public void setEmail(String t)  
        this.email = t;  
    }  
    public String getPassword() {  
        return password;  
    }  
    public void setPassword(String t) {  
        this.password = t;  
    }  
}
```

RegistrationBean.java du
projet **ManagedBean**
possède deux propriétés et
des modifieurs et accesseurs

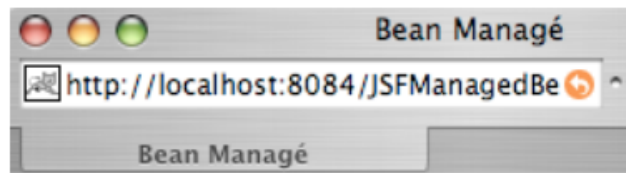
Bean managé : modifier/afficher la propriété d'un Bean

Exemple (suite) : gestion d'un formulaire ...

```
...  
<core:view>  
  <html:form>  
    <html:outputText value="Adresse Email " />  
    <html:inputText value="#{registrationbean.email}" /><br>  
  
    <html:outputText value="Mot de Passe " />  
    <html:inputSecret value="#{registrationbean.password}" /><br>  
  
    <html:commandButton value="Connecter" />  
  </html:form>  
</core:view>  
</body>  
</html>
```

EL pour accéder à la propriété *email*

EL pour accéder à la propriété *password*



Adresse Email :

Mot de Passe :

beanform2.jsp du projet
ManagedBean

Affichage de la valeur
d'initialisation du Bean

Navigation : statique ou dynamique

JSF implémente une machine à états pour gérer la navigation entre des pages JSF

Pour schématiser nous distinguons deux sortes de navigation

- Navigation statique

- Navigation dynamique

Navigation statique

La valeur de l'outcome est connue au moment de l'écriture de la JSP

Navigation dynamique

La valeur de l'outcome est inconnue au moment de l'écriture de la JSP

Elle peut être calculée par un Bean Managé ou autre chose ...

Remarques

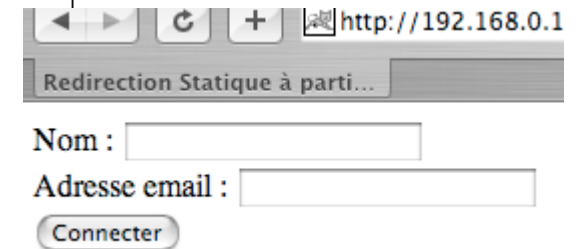
Si une valeur d'outcome est inconnue la même page JSP est rechargée

Si la valeur vaut null la page JSP est rechargée

Navigation statique : exemple

Exemple : redirection statique à partir d'un formulaire

```
...  
<html>  
  <head>  
    <title>Redirection Statique à partir d'un formulaire</title>  
  </head>  
  <body>  
    <core:view>  
      <html:form>  
        <html:outputText value="Nom : " />  
        <html:inputText value="#{beancontroller1.name}" /><br>  
        <html:outputText value="Adresse email : " />  
        <html:inputText value="#{beancontroller1.email}" /><br>  
        <html:commandButton value="Connecter" action="register" />  
      </html:form>  
    </core:view>  
  </body>  
</html>
```

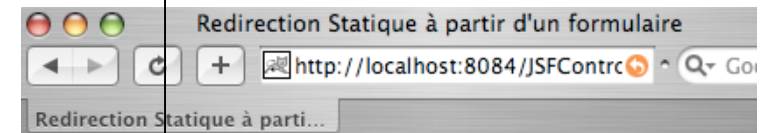


A screenshot of a web browser window. The address bar shows the URL `http://192.168.0.1`. The page title is "Redirection Statique à parti...". The form contains two input fields: "Nom :" and "Adresse email :". Below the fields is a button labeled "Connecter". A line from the code block on the left points to the "Connecter" button.

Navigation statique : exemple

Exemple (suite) : redirection statique ...

```
<%@taglib uri="http://java.sun.com/jsf/core" prefix="core"%>
<%@taglib uri="http://java.sun.com/jsf/html" prefix="html"%>
<html>
    ...
    <body>
        <core:view>
            <h1><html:outputText value="Redirection Réussie" /></h1>
        </core:view>
    </body>
</html>
```



Redirection Réussie

Navigation dynamique : exemple

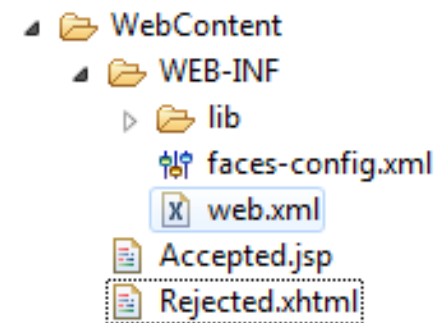
A partir de la version 2.0 JSF simplifie la gestion de la navigation en permettant de se passer du faces-config.

Si le texte reçu dans l'attribut action (statiquement ou dynamiquement) n'est pas trouvé dans le faces-config JSF recherchera dans WebContent un fichier d'affichage (jsp, xhtml,...) nommé de cette façon.

```
package beanPackage;

public class BeanController2 {
    private String email = "user@host";
    private String name = "";

    public String loginConnect() {
        if (this.email.isEmpty()) {
            return "Rejected";
        }
        if (this.name.isEmpty()) {
            return "Rejected";
        }
        return "Accepted";
    }
}
```



Navigation dynamique : exemple

Exemple (suite) : redirection dynamique à partir d'un Bean

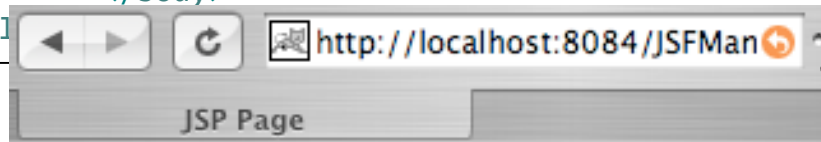
```
...  
<html>  
  <head>  
    <title>Redirection Dynamique à partir d'un formulaire</title>  
  </head>  
  <body>  
    <core:view>  
      <html:form>  
        <html:outputText value="Nom : " />  
        <html:inputText value="#{beancontroller2.name}" />  
        <br>  
        <html:outputText value="Adresse email : " />  
        <html:inputText value="#{beancontroller2.email}" />  
        <br>  
        <html:commandButton value="Connecter"  
          action="#{beancontroller2.LoginConnect}" />  
      </html:form>  
    </core:view>  
  </body>  
</html>
```

Accès au Bean identifié et à
la méthode loginConnect

Navigation dynamique : exemple

Exemple (suite) : redirection dynamique à partir d'un Bean

```
<%@taglib uri="http://java.sun.com/jsf/core" prefix="core"%>
<%@taglib uri="http://java.sun.com/jsf/html" prefix="html"%>
<html>
    <head>
        <title>Résultat du traitement du formulaire</title>
    </head>
    <body>
        <core:view>
            <h1>
                <html:outputText value="Connexion de : " />
                <html:outputText value="#{beancontroller2.name}" />
            </h1>
        </core:view>
    </body>
</html>
```



Connexion de : exemple@toto.fr

Page affichée à la suite de la soumission. Lecture de la propriété email du Bean identifié par beancontroller2

Composants graphiques : présentation

JSF fournit un ensemble de composants graphiques pour la conception de l'IHM

Un composant JSF est développé à partir de :

- classes qui codent le comportement et l'état
 - classes de « rendu » qui traduisent la représentation graphique (HTML, FLASH, XUL, ...)
 - classes qui définissent les balises personnalisées relation entre JSP et classes Java
 - classes qui modélisent la gestion des événements
- classes qui prennent en compte les Converters et les Validators

Pour exploiter les composants JSF dans les pages JSP, des balises personnalisées sont utilisées ...

Composants graphiques : balises CORE

Les balises personnalisées décrites dans CORE s'occupent d'ajouter des fonctionnalités aux composants JSF

Pour rappel pour avoir un descriptif de l'ensemble des balises CORE :

<https://docs.oracle.com/javaee/7/javaxserver-faces-2-2/vldocs-facelets/toc.htm>

La bibliothèque contient un ensemble de balises

facet : déclare un élément spécifique pour un composant

view et subview : ajoute une vue ou une sous vue

attribute et param : ajoute un attribut ou paramètre à un composant

selectionitem et selectionitems : définit un ou plusieurs éléments

convertDateTime et convertNumber : conversion de données

validator, validateDoubleRange, ... : ajoute des validators

actionListener, valueChangeListener : différents écouteurs

loadBundle : chargement de bundle (fichier de ressources ...)

Composants graphiques : balises HTML

Famille des regroupements

- form
- panelGroup
- panelGrid

Famille des saisis

- inputHidden
- inputSecret
- inputText
- inputTextArea
- selectBooleanCheckbox
- selectManyCheckbox
- selectManyListbox
- selectManyMenu

- selectOneListbox
- selectOneMenu
- selectOneRadio

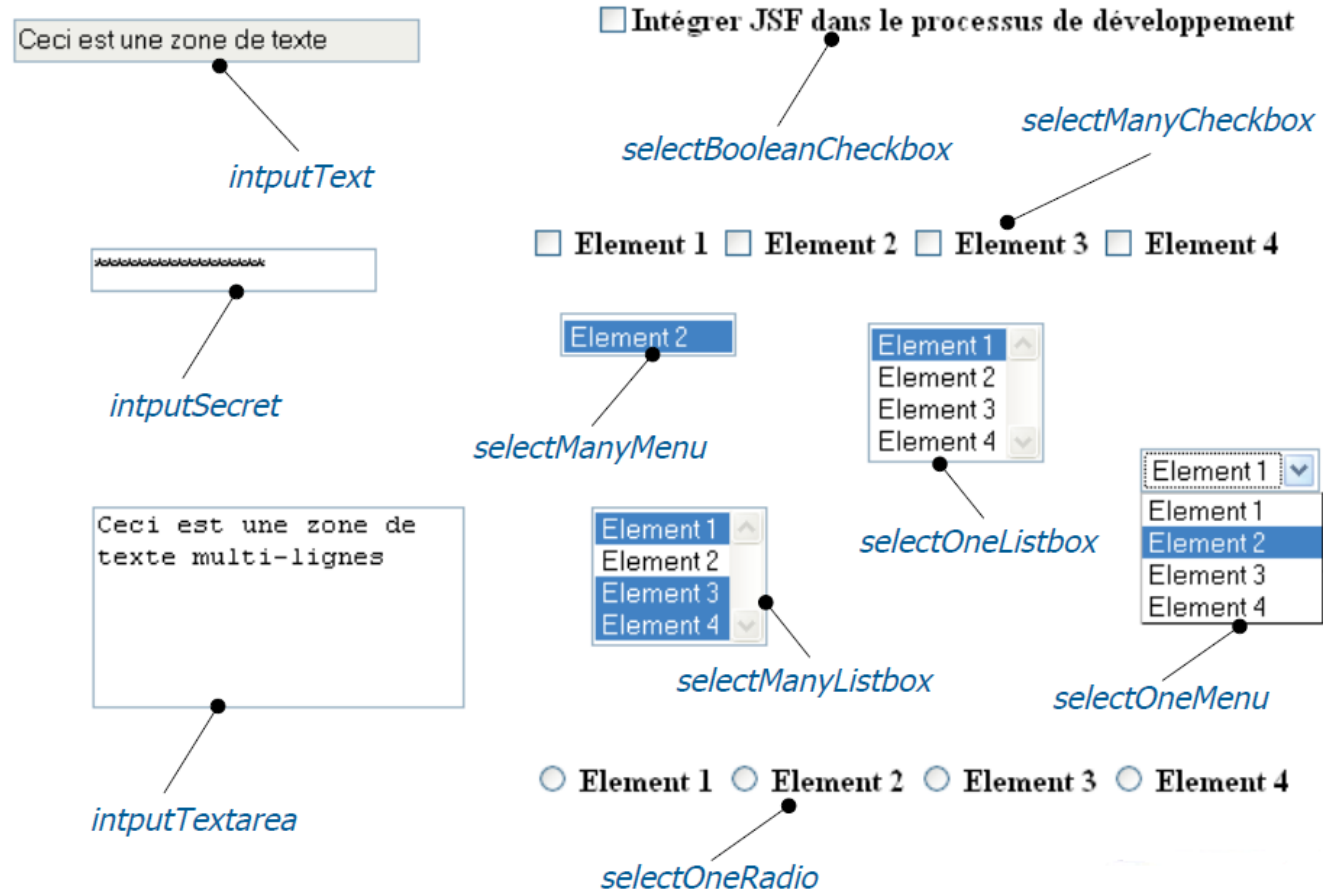
• Famille des sorties

- column
- message et messages
- dataTable
- outputText
- outputFormat
- outputLink
- graphicImage

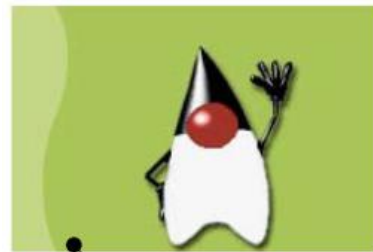
• Famille des commandes

- commandButton
- commandLink

Composants graphiques : balises HTML saisis



Composants graphiques : balises HTML sorties et commandes



graphicImage

Commande Bouton

commandButton

Commande Lien

commandLink

outputText

Du texte affiché avec la balise `outputText`

Google

outputLink

Nom	Prenom	Date de naissance	Poids
Baron	Mickael	17081976	Ingénieur d'étude et de développement
Dupont	Marcel	21041956	Boucher

dataTable

Nom :	<input type="text" value="Baron"/>
Mot de passe :	<input type="password" value="sksksksksksk"/>

panelGrid

Composants graphiques : détail balises HTML

Une balise contient généralement trois types d'attributs

- Des attributs basiques (description)

- Des attributs HTML (description de présentation)

- Des attributs DHTML (description pour la partie dynamique)

Selon la nature du composant graphique (saisie, sortie, commande et autre) le nombre d'attributs varie ...

Composants graphiques : détail balises HTML

Les attributs basiques sont utilisés pour ajouter une description spécifique à JSF

Attributs basiques

id : identifiant du composant

binding : association avec un Bean « Backing »

rendered : true composant affiché, false composant caché

styleClass : précise le nom d'une CSS à appliquer au composant

value : valeur du composant

valueChangeListener : associer à une méthode pour le changement de valeur

converter : nom de la classe pour une conversion de données

validator : nom de la classe pour une validation de données

required : true valeur obligatoire, false valeur optionnelle

Composants graphiques : détail balises HTML

Les attributs HTML sont utilisés pour modifier l'apparence graphique d'un composant JSF

Attributs HTML

alt : texte alternatif au cas où le composant ne s'affiche pas

border : bordure du composant

disabled : désactive un composant de saisie ou un bouton

maxlength : maximum de caractères pour un composant de texte

readonly : mode lecture uniquement

size : taille d'un champ de texte

style : information du style

target : nom de la frame dans lequel le document est ouvert

...

Composants graphiques : détail balises HTML

Les attributs DHTML sont utilisés pour ajouter un aspect dynamique pour les composants JSF (appel à du JavaScript)

Attributs DHTML

- onblur : élément perd le focus
- onclick : clique souris sur un élément
- ondblClick : double clique souris sur un élément
- onfocus : élément gagne le focus
- onkeydown, onkeyup : touche clavier enfoncée et relâchée
- onkeypress : touche clavier pressée puis relâchée
- onmousedown, onmouseup : bouton souris enfoncé et relâché
- onmouseout, onmouseover : curseur souris sort et entre
- onreset : formulaire est initialisé
- onselect : texte est sélectionné dans un champ de texte
- onsubmit : formulaire soumis

Composants graphiques : `<html:panelGrid>`

La balise `html:panelGrid` est utilisée pour définir l'agencement des composants visuels

Il ne s'agit donc pas d'un composant visuel mais d'un conteneur dont l'organisation de ces composants enfants est réalisée suivant une grille

Les composants enfants sont ajoutés dans le corps

Principaux attributs :

- `columns` : nombre de colonnes de la grille
- `bgcolor` : couleur du fond
- `cellpadding`, `cellspacing` : espacement entre les cellules
- `border` : border autour de la grille

Le composant `html:panelGroup` permet de regrouper des composants dans une cellule (une sorte de fusion)

Navigation dynamique : exemples

```
...  
<h4>  
  <html:panelGrid cellspacing="25" columns="2">  
    <html:outputText value="Nom : " />  
    <html:panelGroup>  
      <html:inputText size="15" required="true" />  
      <html:inputText />  
    </html:panelGroup>  
    <html:outputText value="Mot de passe : " />  
    <html:inputSecret />  
  </html:panelGrid>  
</h4>
```

Création d'une
grille avec deux
colonnes

Regroupement de
deux composants
dans une cellule

Nom :

Mot de passe :

Composants graphiques : <html:dataTable>

Le composant <html:dataTable> permet de visualiser des données sur plusieurs colonnes et plusieurs lignes

La table peut être utilisée quand le nombre d'éléments à afficher est inconnu

Les données (la partie modèle) peuvent être gérées par des Beans

Attributs de la balise :

- value : une collection de données (Array, List, ResultSet, ...)

- var : nom donné à la variable à manipuler pour chaque ligne

- border, bgcolor, width : attributs pour l'affichage

- rowClasses, headerClass : attributs pour la gestion des styles (CSS)

Pour chaque colonne de la table, la valeur à afficher est obtenue par la balise <html:column>

Composants graphiques : <html:dataTable>

Exemple : la représentation d'une table JSF

```
public class Personne {  
    private String name;  
    private String firstName;  
    private String birthName;  
    private String job;  
    public Personne(String pN, String pF, String pB, String pJ) {  
        name = pN; firstName = pF; birthName = pB; job = pJ;  
    }  
    public String getName() { return name; }  
    public void setName(String pName) { name = pName; }  
    public String getFirstName() { return firstName; }  
    public void setFirstName(String pFirstName) { firstName = pFirstName; }  
    ...  
}
```

```
public class DataTableBean {  
    private List<Personne> refPersonne;  
    public List getPersonne() {  
        if (refPersonne == null) {  
            refPersonne = new ArrayList<Personne>();  
            refPersonne.add(new Personne("Baron", "Mickael", "17081976",  
"Développeur"));  
            refPersonne.add(new Personne("Dupont", "Marcel", "21041956", "Boucher"));  
        }  
        return refPersonne;  
    }  
}
```

Composants graphiques : <html:dataTable>

Exemple (suite) : la représentation d'une table JSF

```
<core:view>
  <html:dataTable value="#{outputbean.personne}"
var="personne" border="1" cellpadding="4" width="60%" >
    <html:column>
      <html:outputText value="#{personne.name}" />
    </html:column> <html:column>
      <html:outputText value="#{personne.firstname}" />
    </html:column> <html:column>
      <html:outputText value="#{personne.birthdata}" />
    </html:column> <html:column>
      <html:outputText value="#{personne.job}" />
    </html:column>
  </html:dataTable>
</core:view>
```

Baron	Mickael	17081976	Ingénieur d'étude et de développement
Dupont	Marcel	21041956	Boucher
Martin	Alexandre	28011946	Magicien
Fox	Tracy	18021969	Sexologue

Composants graphiques : <html:dataTable>

La modification des en-tête et pied de page d'une table est obtenue en utilisant la balise <core:facet>

<core:facet> s'occupe d'effectuer une relation de filiation entre un composant et un autre

La filiation consiste à associer le composant <core:facet> est un composant défini dans le corps de <core:facet>

Attribut du composant <core:facet>

name : nom de la filiation

Pour le composant table deux filiations possibles

header : une filiation entre une colonne et le nom de la colonne

footer : une filiation entre la table et un nom

caption : une filiation entre le titre de la table et un nom

Composants graphiques : <html:dataTable>

Exemple (suite) : la représentation d'une table JSF

```
<core:view><html:dataTable value="#{outputbean.personne}" var="personne"
border="1" cellspacing="4" width="60%" >
  <html:column>
    <core:facet name="header" >
      <html:ouputText value="Nom" />
    </core:facet>
    <html:outputText value="#{personne.name}" />
  </html:column>
  <html:column>
    <core:facet name="header" >
      <html:verbatim>Prénom</verbatim>
    </core:facet>
    <html:outputText value="#{personne.firstname}" />
  </html:column>
  <html:column>
    <core:facet name="header" >
      Date de naissance
    </core:facet>
    <html:outputText value="#{personne.birthdata}" />
  </html:column>
  <html:facet name="footer">
    <html:outputText value="#{outputbean.caption}" />
  </html:facet>
  ...
</html:dataTable></core:view>
```

Nom	Prénom		Emploi
Baron	Mickael	17081976	Ingénieur d'étude et de développement
Dupont	Marcel	21041956	Boucher
Martin	Alexandre	28011946	Magicien
Fox	Tracy	18021969	Sexologue
Une Simple Table			

Ne sera jamais affiché
car <core:facet> n'est
associé à aucun autre
composant

Composants graphiques : <html:dataTable>

Pour l'instant, seul l'aspect peuplement des informations a été traité, <html:dataTable> offre la possibilité de modifier l'apparence d'une table

Les attributs :

- headerClass : style CSS pour la partie en-tête
- footerClass : style CSS pour le bas de page
- rowClasses : liste de styles CSS pour les lignes

L'attribut rowClasses permet de définir une liste de style CSS qui sera appliquée pour chaque ligne et répétée autant de fois qu'il y a de lignes (définition d'un motif)

Exemple :

- Soit la définition suivante : rowClasses="row1, row2, row2"
- Répétition du motif row1, row2, row2, row1, row2, ...

Composants graphiques : <html:dataTable>

Exemple : table JSF « stylisé »

```
<head>
<...>
<link href="output.css" rel="stylesheet" type="text/css" >
</head>
<body>
    <core:view>
        <html:dataTable value="#{outputbean.personne}" var="personne" border="1"
            cellspacing="4" width="60%" rowClasses="row1,row2"
            headerClass="heading" footerClass="footer" >
            ...
        </html:dataTable>
    </core:view>
```

Nom	Prenom	Date de naissance	Emploi
Baron	Mickael	17081976	Ingénieur d'étude et de développement
Dupont	Marcel	21041956	Boucher
Martin	Alexandre	28011946	Magicien
Fox	Tracy	18021969	Sexologue
Une Simple Table			

```
.heading {
font-family: Arial, Helvetica,
sans-serif;
font-weight: bold;
font-size: 20px;
color: black;
background-color:silver;
text-align:center;
}
.row1 {
background-color:#GFGFGF;
}
.row2 {
Background-color:#CECECE;
.footer {
background-color:#000009;
Color:white;
}
```

Composants graphiques : <html:dataTable>

Exemple : table JSF avec des composants JSF de saisie

```
<core:view>
  <html:dataTable value="#{outputbean.personne}" var="personne"
border="1" cellspacing="4" width="60%" rowClasses="..." >
    <html:column>
      <core:facet name="header" >
        <html:ouputText value="Nom" />
      </core:facet>
      <html:inputText value="#{personne.name}" />
    </html:column>
    <html:column>
      <core:facet name="header" >
        <html:verbatim>Prénom</verbatim>
      </core:facet>
      <html:outputText value="#{personne.firstname}" />
    </html:column>
  </html:dataTable>
</core:view>
```

Nom	Prenom	Date de naissance	Emploi
<input type="text" value="Baron"/>	Mickael	17081976	Ingénieur d'étude et de développement
<input type="text" value="Dupont"/>	Marcel	21041956	Boucher
<input type="text" value="Martin"/>	Alexandre	28011946	Magicien
<input type="text" value="Fox"/>	Tracy	18021969	Sexologue

Une Simple Table