JSF 2.0 (*Java Server Faces*) Partie 2

Université de Nice - Sophia Antipolis
Richard Grin
Version O 0.6 – 15/10/12

- Cette deuxième partie présente des notions de base de JSF attachées à l'écriture de pages JSF

Plan

- Navigation
- Langage EL
- Backing Beans
- Portées (requête, session,…)
- Utilisation des EJB
- Ajax
- Templates (patrons de mise en page)
- Internationalisation
- Gestion des ressources

Navigation

Navigation dans JSF

- Navigation = affichage d'une nouvelle page
- Des composants JSF standards provoquent une navigation : **`<h:commandButton>`**, **`<h:button>`**, **`<h:commandLink>`**, **`<h:link>`**, **`<h:ouputLink>`**
- A part le dernier, ils doivent tous être dans une balise **`<h:form>`**

Soumission de formulaire

- **`<h:commandButton>`** et **`<h:commandLink>`** soumettent le formulaire par une requête POST et affichent la nouvelle page indiquée par l'attribut **`action`**
- **`<h:button>`** et **`<h:link>`** soumettent le formulaire par une requête GET et affichent la nouvelle page indiquée par l'attribut **`outcome`**

Adresse affichée par le navigateur

- Avec **<h:commandButton>** et **<h:commandLink>** le navigateur affiche toujours l'URL du formulaire quand la nouvelle page est affichée
  si on veut voir le bon URL, il faut ajouter une redirection
- Avec **<h:button>** et **<h:link>** le navigateur affiche le bon URL

Exemple

Dans formulaire.xhtml

```
<h:form>
  ...
  <h:commandButton
    value="Envoyer la commande"
    action="#{commandeBean.checkout}" />
</h:form>
```

Si la méthode **checkout** retourne "page2" , page2.xhtml sera affichée, mais le navigateur affichera toujours l'URL de formulaire.xhtml

Navigation statique et dynamique

- La valeur qui détermine la nouvelle page peut être
  - définie « en dur » au moment de l'écriture de l'application ; la navigation est statique
  - définie par la valeur retournée par une méthode Java ; la navigation est dynamique
- Dans les 2 cas, la valeur peut déterminer la prochaine page à afficher de façon déclarative ou implicite

Exemple de navigation statique

- Dans la page JSF page1.xhtml :

```
<h:commandButton
  action="page2"
  value="Aller à page2" />
```


Exemple de navigation dynamique

- Dans la page JSF page1.xhtml :

```
<h:commandButton
  action="#{bean.allerPage2}"
  value="Aller à page2" />
```

- Dans un backing bean :

```
@Named
@SessionScoped
public class Bean implements Serializable {
  public String allerPage2(){
    if (...) return "page2";
    else return "autrePage";
  }
  ...
```


Navigation déclarative ou implicite

- Si la valeur qui détermine la nouvelle page correspond à une règle de navigation, la règle détermine la prochaine page à afficher ; c'est la navigation déclarative
- Sinon, la valeur est considérée comme le nom du fichier de la future page à afficher ; c'est la navigation implicite

Exemple de navigation déclarative

- Règle de navigation dans faces-config.xml :

```
<navigation-rule>
  <from-view-id>page1.xhtml</from-view-id>
  <navigation-case>
    <from-outcome>ok</from-outcome>
    <to-view-id>/succes.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
```

- Dans la page JSF page1.xhtml :

```
<h:commandButton action="ok"
                 value="Valider" />
```


Exemple de navigation implicite

- Dans la page JSF page1.xhtml (pas de règle de navigation dans faces-config.xml) :

```
<h:commandButton action="page2"
    value="Aller à page2" />
```

.xhtml ajouté automatiquement **`page2.xhtml`**

Redirection

- Si on veut une redirection,
  - dans le cas d'une navigation statique, il suffit d'ajouter **`?faces-redirect=true`** à l'URL de la page à afficher ; par exemple, **`action="page2?faces-redirect=true"`**
  - dans le cas d'une navigation dynamique, il suffit d'ajouter cette même chaîne à la valeur retournée par la méthode

- Exercice 1 du TP sur JSF

Langage EL (Expression Language)

Utilité

- Sert à relier des propriétés ou des méthodes de beans avec des valeurs de pages JSF
- Une expression EL est entourée par #{…}
- Exemple :
  `<h:inputText value="#{utilisateur.nom}"/>`
  lie la valeur du champ de saisie de texte avec la propriété « nom » du bean « utilisateur »
- Nous ne verrons qu'un aperçu de la syntaxe

a.b pour un objet « ordinaire »

- Si a est un objet qui n'est ni un tableau, ni une liste, ni une map, a.b désigne une propriété b de a
- a.b peut désigner le getter a.getA() ou le setter a.setA(A ap) selon les cas
- <h:inputText value="a.b" / >
  - la valeur affichée au départ sera celle de a.getA()
  - la valeur tapée par l'utilisateur sera rangée dans le modèle par la méthode a.setA

Opérateurs

- Arithmétiques : + - * / (ou div) % (ou mod)
- De comparaison : < <= > >= == != (ou lt le gt ge eq ne)
- Logiques : && || ! (ou and or not)
- empty ; « empty a » est vrai si a est null, un tableau ou une chaîne vide, une collection ou une map de longueur 0
- Ternaire ? : (comme en Java)
- Les priorités sont les mêmes qu'en Java

Expression de méthode

- Une expression EL peut désigner une méthode
- Exemple :

```
<h:commandButton action="#{user.verifMdp}" />
```

- On peut aussi passer des paramètres à une méthode :

```
<h:commandButton action="#{a.deplace(-2)}" />
```


Backing bean

Bean géré par JSF

- Le backing bean doit être annoté par **`@ManagedBean`** pour être géré par JSF
- Ainsi il sera utilisé par les pages JSF et créé automatiquement si besoin est
- Depuis JSF 2.0 il est plutôt conseillé d'utiliser CDI pour gérer le bean

Avec CDI

- Les possibilités d'injection sont plus riches
- Pour qu'un bean puisse être référencé dans une page JSF il doit être annoté par **`@Named`**

Exemple

```
@Named
@SessionScoped
public class Bean implements Serializable {
  ...
}
```


Utilité des backing bean

- Ils servent essentiellement à
  - réagir aux actions de l'utilisateur (méthodes « action » ou méthodes « écouteur »)
  - fournir l'état (modèle de données) utilisé par les pages JSF (contenu dans les propriétés du bean)

Attacher une propriété

- `<inputText value="#{employe.nom}">`

Association bean – page JSF

- Souvent une page – un bean
- Mais on peut aussi trouver une page – plusieurs beans ou plusieurs pages – un bean

Portée (scope)

Utilité des portées

- Le backing bean est créé par le container la première fois qu'il est utilisé et sa durée de vie dépend de sa portée
- Si la portée est la session, il va durer pendant toute la session de travail de l'utilisateur ; si une autre expression EL y fait référence pendant le même session, c'est le même bean qui sera utilisé

2 types de portée

- JSF a ses portées mais il est recommandé d'utiliser plutôt CDI (Contexts and Dependency Injection)
- JSF et CDI ont des portées qui ont le même nom terminal et il faut faire attention à importer le bon paquetage ; c'est souvent une source d'erreur

Portées de JSF

- ApplicationScoped
- SessionScoped
- RequestScoped (portée par défaut)
- ViewScoped (vue)
- CustomScoped (personnalisée)
- NoneScoped (instancié mais mis dans une portée ; utile pour un bean qui est un propriété d'un autre bean)
- Les annotations sont dans le paquetage **`javax.faces.bean`**

Portées de CDI

- ApplicationScoped
- SessionScoped
- RequestScoped
- ConversationScoped
- Dependent (portée par défaut ; nouvel objet créé à chaque référence et supprimé lorsque le but de l'injection est supprimé)
- Les annotations sont dans le paquetage **`javax.entreprise.context`**

Ajax

Ajax

- Asynchronous Javascript and XML
- Permet des échanges entre le client et le serveur Web sans affichage d'une page entière
- Les requêtes envoyées par le client sont lancées « en arrière-plan » ; l'utilisateur peut continuer à travailler avec la page en cours
- La réponse du serveur à la requête peut ne modifier qu'une partie de la page en cours

<f:ajax>

- Permet d'ajouter des fonctionnalités « Ajax » au composant JSF dans lequel il est inclus
- Par exemple, un bouton de soumission de formulaire peut envoyer une requête et la réponse du serveur mettra à jour une partie de la page sur laquelle l'utilisateur a continué à travailler

<f:ajax> - les attributs

- **event** : événement déclencheur de la requête
- **execute** : composants pris en compte pour l'envoi de la requête (espace séparateur)
- **render** : composants mis à jour au moment de la réponse du serveur (espace séparateur)

Exemple 1

```
<h:form>
  <h:input value= />
  <h:commandButton>
    <f:ajax event="click" execute="@form"
            render="out1 out2" />
  </h:commandButton>
  <h:input id="out1" …/>
  <h:input id="out2" …/>
</h:form>
```


Exemple 2

```
<h:form>
  <h:input value="#{bean.prop" />
  <h:selectOneMenu value="#{bean.selected}">
    <f:selectItems value="#{bean.items}" />
    <f:ajax execute="@form" render="@form" />
  </h:selectOneMenu>
  ...
</h:form>
```


Templates, modèles de présentation

Présentation

- Les applications doivent utiliser des chartes graphiques pour uniformiser la présentation des pages Web
- Les *templates* évitent la répétition de code entre les pages qui suivent un même schéma de présentation
- Un template est un fichier XHTML qui contient des parties nommées (<ui:insert>) qui seront remplacées par du code pour chaque page particulière qui utilisera ce template

Exemple de template (1)

```
<!DOCTYPE html ...>
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:ui="http://java.sun.com/jsf/facelets" ...>
<h:head>
    <title>
        <ui:insert name="titreFenetre">
            #{msgs.titreFenetre}
        </ui:insert>
    </title>
</h:head>
```

Valeur par défaut

Exemple de template (2)

```
<h:body> ...
  <div class="enTetePage">
      <ui:insert name="heading">
          #{msgs.enTetePage}
      </ui:insert>
  </div>
  <div class="menuEtContenu">
    <div class="menuGauche">
      <ui:insert name="menuGauche"/>
    </div>
  ...
</h:body>
</html>
```


Une page qui utilise le template

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
xmlns:ui="http://java.sun.com/jsf/facelets"
template="/templates/masterLayout.xhtml">
    <ui:define name="menuGauche">
      <ui:include src="/sections/login/menuLeft.xhtml"/>
    </ui:define>
    <ui:define name="contenu">
      <ui:include src="/sections/login/content.xhtml"/>
    </ui:define>
</ui:composition>
```


Internationalisation

Quelques tâches seulement

- Il est simple d'internationaliser son application si on le prévoit dès le début :
  - Prévoir un fichier par langue prise en compte pour ranger les messages
  - Dans les pages JSF, remplacer les messages par un code simple
  - Dans le code Java, ajouter quelques lignes de code
  - Ajouter quelques lignes dans faces-config.xml

`faces-config.xml`

```
<application>
  <resource-bundle>
    <base-name>/Bundle</base-name>
    <var>bundle</var>
  </resource-bundle>
  <locale-config>
    <default-locale>en</default-locale>
    <supported-locale>fr</supported-locale>
  </locale-config>
</application>
```


Dans une page JSF

- Pour afficher dans la locale préférée du navigateur de l'utilisateur, il suffit de remplacer les textes par des appels du type **`#{bundle.UnTexte}`**
  et de prévoir un fichier de propriétés par langue prise en compte

Exemple

- Dans la page JSF :
  `<h:outputLabel value="#{bundle.EditLabel_nom}" for="id" />`
- Dans le fichier bundle_en.properties :
  `EditLabel_nom=Family Name:`
- Dans le fichier bundle_fr.properties :
  `EditLabel_nom=Nom :`

Gestion des ressources

Ressources des pages Web

- Les pages HTML peuvent faire appel à des ressources externes : images, fichier CSS, code Javascript,…
- Ces ressources sont obtenues du serveur par des requêtes HTTP
- JSF 2.0 a standardisé la gestion de ces ressources

Standardisation de l'accès aux ressources

- Par défaut, les ressources peuvent se situer à 3 endroits dans l'application
  - le répertoire **`resources`** placé sous la racine de l'application Web contient toutes les ressources
  - Les répertoires (nom par rapport à la racine de l'application) **`META-INF/resources`** ou **`WEB-INF/classes/META-INF/resources`**

Librairies

- Les ressources sont regroupées en librairies
- Une librairie correspond à un sous-répertoire des répertoires des ressources
- css, javascript et images sont les noms usuellement utilisés pour les fichiers CSS, javascript et pour les images

Exemple avec image

```
<h:graphicImage library="images"
   name="photo.jpg" />
```