JSF 2.0
(*Java Server Faces*)
Partie 1

Université Française d'Egypte
Richard Grin
Version O 0.8 – 3/10/12

- Ce support est une introduction à JSF 2.0 utilisé dans un cadre Java EE 6, CDI (*Contexts and Dependency Injection*) compris, avec un serveur d'application de type Glassfish

Parties du support

- Cette première partie présente les fondamentaux de JSF ainsi qu'un survol des éléments qu'on peut rencontrer dans une application JSF
- La deuxième partie présente des notions de base de JSF, plus attachées à l'écriture de pages JSF
- La dernière partie aborde la programmation Java en détails et des fonctionnalités plus avancées de JSF
- Les composants JSF standards sont présentés sur un support à part

Plan de cette partie

- Présentation
- Application Web
- Cycle de vie
- Développement d'une application Web
- Rappels sur HTTP
- Page JSF
- Fichiers de configuration
- Implémentations de JSF (librairies JSF)
- Code d'une page JSF
- Architecture des applications JSF

Présentation

Java EE 6

- JSF 2.0 est intégré dans Java EE 6 ; il facilite l'écriture de l'interface utilisateur des applications Web
- JSF 2.0 peut (c'est conseillé) utiliser CDI (*Contexts and Dependency Injection*)

Utilité de JSF

- Créer des pages Web dynamiques
- Une grande partie de la programmation liée à la validation des données saisies par l'utilisateur et de leur transmission au code Java est automatisée

Services rendus par JSF (1/2)

- Permet de bien séparer l'interface utilisateur, la couche de persistance et les processus métier
- Conversion des données entre le texte de l'interface utilisateur et les valeurs du serveur
- Validation des données saisies par l'utilisateur
- Automatisation de l'affichage des messages d'erreur si problèmes de conversion ou validation
- Internationalisation
- Support d'Ajax sans programmation

Services rendus par JSF (2/2)

- Fournit des composants standards pour l'interface utilisateur
- Possible d'utiliser des librairies de composants tierce et d'ajouter ses propres composants
- Templates graphiques

Page JSF

- Code XHTML qui contient des balises qui décrivent les composants sur le serveur
- Sera traduit en XHTML « pur » pour être envoyé au client Web

Exemple de page JSF

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 ..."
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1- ...">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
 <h:head>
   <title>Hello</title>
 </h:head>
 <h:body>
   <h1>Hello #{utilisateur.nom}</h3>
 </h:body>
</html>
```

Lien avec un bean Java qui a une propriété « **nom** »

Les traitements

- JSF permet une séparation très nette entre la présentation des pages de l'interface utilisateur (pages JSF) et les traitements
- Même les traitements directement liés à l'interface utilisateur sont effectués en dehors par du code Java (dans des *backing beans*)

Exemple de backing bean

```
@Named
@SessionScoped
public class Utilisateur implements Serializable {
  private String nom;
  ...
  public String getNom() { return nom; }
  public void setNom(String nom) {
    this.nom = nom;
  }
  ...
}
```

Une instance de cette classe est automatiquement créée par le container de page JSF quand c'est nécessaire

Facelets

- Langage de description préféré pour décrire les pages JSF
- Utilise XHTML, des librairies de balises standards pour JSF et le langage EL (*Expression Language*) pour faire le lien entre des valeurs ou des actions utilisées par des pages JSF et des propriétés ou des méthodes de classes Java
- Facilite l'utilisation de *templates* pour uniformiser l'aspect des pages affichées par l'application
- Ne plus utiliser JSP

Architecture des applications JSF

Composants de l'architecture

- Pages JSF
- Ressources (images, fichiers CSS ou JavaScript)
- Backing beans
- EJBs
- Entités (JPA)
- Classes java ordinaire (POJO)
- Parfois des servlets

Répartition des tâches

- Les pages JSF sont utilisées pour l'interface avec l'utilisateur ; elles ne contiennent pas de traitements
- Les backing beans s'occupe des traitements liés directement à l'interface utilisateur
- Les backing beans peuvent aussi faire appel à des EJB ou POJO pour les traitements qui ne sont pas liés directement à l'interface utilisateur (traitements métier ou accès aux sources de données)
- Les accès aux bases de données utilisent le plus souvent JPA et donc les classes Java entités

Backing bean

- Souvent, mais pas obligatoirement, un backing bean par page JSF
- Fait la liaison entre les valeurs affichées ou saisies dans les pages JSF et le code Java qui effectue les traitements métier

Code d'un backing bean

- Contient des valeurs à afficher ou saisies par l'utilisateur (attribute **`value`**)
- Lance un traitement initié par l'utilisateur (clic bouton par exemple)
- Calcule une expression qui indique si un composant ou un groupe de composant doit être affiché (attribut **`rendered`**)
- Contient un lien vers un composant JSF (attribut **`binding`**), pour manipuler ce composant ou les attributs de ce composant par programmation

EJB

- Lorsqu'un processus métier ou un accès aux bases de données doit être déclenché, le backing bean cède la main à un EJB
- Un EJB est totalement indépendant de l'interface graphique

Templates

- Les pages d'un site Web respectent souvent une charte graphique
- Avec les facelets il est très simple de définir l'aspect commun aux pages dans un fichier XHTML à part, appelé *template*
- Il suffit alors de référencer ce template dans le code d'une page JSF pour qu'elle ait cet aspect

Convertisseurs et validateurs

- Les données affichées ou saisies dans l'interface Web sont des chaînes de caractères
- Pour effectuer la correspondance avec les valeurs manipulées par le code Java des beans il faut les convertir dans les bons types Java
- Il faut aussi ne lancer les traitements Java qu'après avoir validé les données (par exemple un email tapé par l'utilisateur)
- Les implémentations JSF fournissent des convertisseurs et validateurs standards

Convertisseurs et validateurs personnalisés

- Le développeur doit écrire ses propres convertisseurs et validateurs lorsque l'implémentation JSF ne les fournit pas
- Ce sont des classes ou des méthodes Java dont l'interface est définie par la spécification JSF

Composants standards

- JSF fournit de nombreux composants standards
- Ces composants correspondent aux balises HTML des pages Web : zone de saisie des données, listes déroulantes, boutons, formulaires, tables,…
- Pour améliorer ces composants ou pour en avoir de nouveaux le développeur peut utiliser une ou plusieurs bibliothèques de composants (le plus souvent gratuites)

Composants personnalisés

- Le développeur peut aussi créer ses propres composants

Container pour les JSF

- Pour qu'il sache traiter les pages JSF, le serveur Web doit disposer d'un container pour ces pages
- On peut utiliser pour cela un serveur d'application de type Glassfish

Description d'une application

- L'architecture d'une application et les relations entre les différents composants sont décrits par des annotations incluses dans le code Java ou dans des fichiers de configuration XML
- Plus simple d'utiliser les annotations
- Les fichiers de configuration l'emportent sur les annotations

Page JSF

Librairies de balises

- Plusieurs librairies de balises peuvent être utilisées dans une page JSF (avec Facelets) :
  - JSF Facelets, de préfixe ui
  - JSF HTML, de préfixe h
  - JSF core tag, de préfixe f
  - JSTL core tag (version 1.1), de préfixe c
  - JSTL functions tag (version 1.1), de préfixe fn
- Ces préfixes correspondent à des espaces de nommage XML

Exemple – en-tête

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">
<h:head>
  <title>Inscription</title>
  <h:outputStylesheet library="css" name="styles.css"/>
</h:head>
```

Exemple – liste déroulante

```
<h:body>
  <h3>Inscription</h3>
  <h:form>
    <h:panelGrid columns="3"
       columnClasses="rightalign,leftalign,leftalign">
      <h:outputLabel value="Titre : " for="titre"/>
      <h:selectOneMenu id="titre" label="Titre"
          value="#{inscriptionBean.titre}" >
        <f:selectItem itemLabel="" itemValue=""/>
        <f:selectItem itemLabel="M." itemValue="M"/>
        <f:selectItem itemLabel="Mme" itemValue="Mme"/>
      </h:selectOneMenu>
      <h:message for="titre"/>
```


Exemple – nom et email (avec validateur)

```
      <h:outputLabel value="Nom :" for="nom"/>
      <h:inputText id="nom" label="Nom"
            required="true"
            value="#{inscriptionBean.nom}" />
      <h:message for="prenom" />
      <h:outputLabel value="Email :" for="email"/>
        <h:inputText id="email" label="Email"
            required="true"
            value="#{inscriptionBean.email}">
          <f:validator validatorId="validateurEmail"/>
        </h:inputText>
        <h:message for="email" />
```


Exemple – fin de la page

```
      <h:panelGroup/>
      <h:commandButton id="enregistrement"
         value="Enregistrer"
         action="#{inscriptionBean.confirmer}" />
    </h:panelGrid>
  </h:form>
</h:body>
</html>
```


Commentaires

- Par défaut les commentaires HTML (**<!-- -->**) ne fonctionnent pas pour JSF
- Il faut entourer ce que l'on veut commenter par le tag **<ui:remove>**

Le cycle de vie

- Pour bien utiliser JSF il est indispensable de bien comprendre tout le processus qui se déroule entre le remplissage d'un formulaire par l'utilisateur et la réponse du serveur sous la forme de l'affichage d'une nouvelle page

Le servlet « Faces »

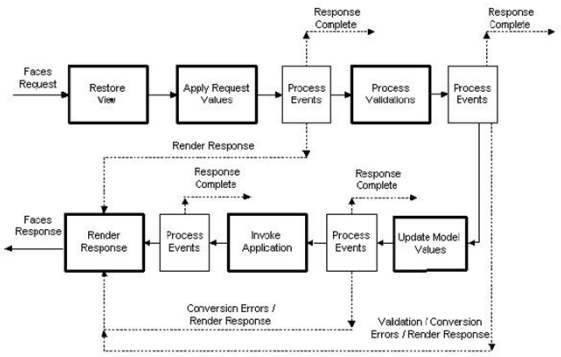
- Toutes les requêtes vers des pages « JSF » sont interceptées par un servlet défini dans le fichier **`web.xml`** de l'application Web
- Avec JSF 2.0, il n'est plus nécessaire d'indiquer ce servlet dans le fichier **`web.xml`**, ni les mappings /faces/*, *.jsf et *.faces pour les URL des pages JSF

Codage - décodage

- Les pages HTML renvoyées par une application JSF sont représentées sur le serveur par un arbre de composants Java
- Le codage est la génération d'une page HTML à partir de l'arbre des composants
- Le décodage est l'utilisation des valeurs renvoyées par une requête HTTP pour les répercuter dans les composants Java qui correspondent aux éléments HTML

Cycle de vie

- Le cycle de vie est composé de 6 phases gérées par le servlet « Faces »



Demande page HTML

- Étudions tout d'abord le cas simple d'une requête d'un client qui demande l'affichage d'une page JSF, sans passer de paramètres dans la requête

La vue

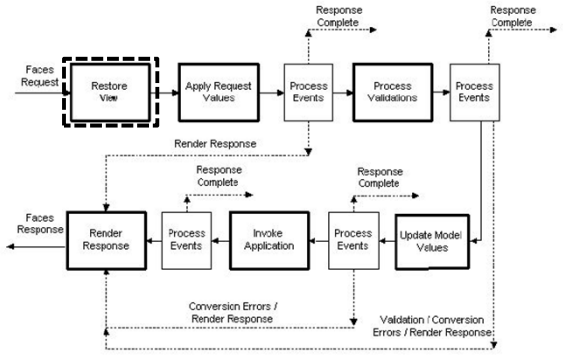
- Cette requête HTTP correspond à une page JSF est interceptée par le servlet Faces
- La page HTML correspondant à la page JSF doit être affichée à la suite de cette requête HTTP
- La page HTML qui sera affichée est représentée sur le serveur par une « vue »
- Cette vue va être construite sur le serveur et transformée sur le serveur en une page HTML qui sera envoyée au client

Construction vue et page HTML

- Une vue formée des composants JSF (en Java) est construite sur le serveur (ou restaurée si elle avait déjà été affichée) : phase « Restore View »
- Puisqu'il n'y a pas de données ou d'événements à traiter, la vue est immédiatement rendue ; le code HTML est construit à partir des composants de la vue et envoyé au client dans la phase « Render Response »

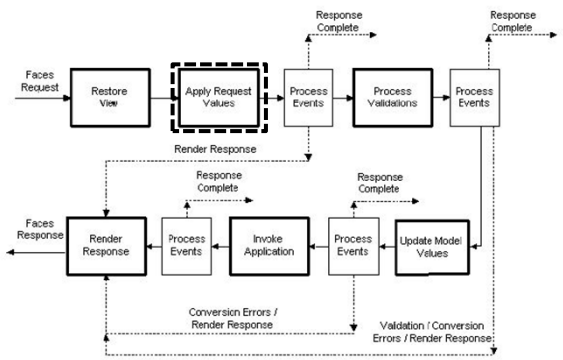
Traitement d'un formulaire

- Nous allons cette fois-ci partir d'une requête HTTP générée à partir d'une page HTML qui contient un formulaire (page construite à partir d'une page JSF)
- L'utilisateur a saisi des valeurs dans ce formulaire
- Ces valeurs sont passées comme des paramètres de la requête HTTP ; par exemple, http://machine/page.xhtml?nom=bibi&prenom=bob si la requête est une requête GET, ou dans l'entité d'un POST



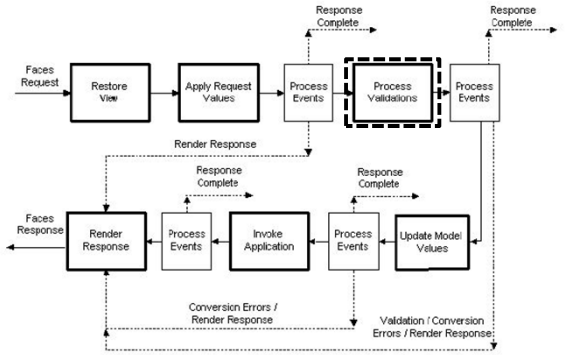
Phase de restauration de la vue

- La vue qui correspond à la page qui contient le formulaire est restaurée (phase « Restore View »)
- Tous les composants reçoivent la valeur qu'ils avaient avant les nouvelles saisies de l'utilisateur



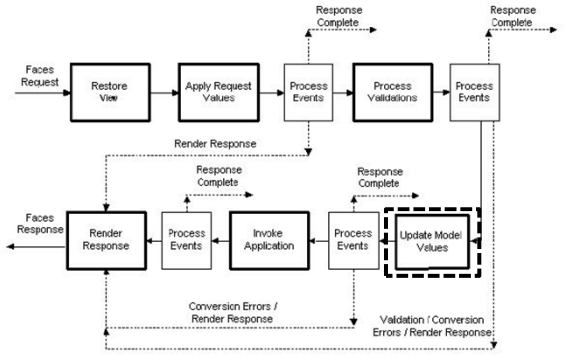
Phase d'application des paramètres

- Tous les composants Java de l'arbre des composants reçoivent les valeurs qui les concernent dans les paramètres de la requête HTTP (ces paramètres contiennent les valeurs saisies par l'utilisateur dans le formulaire) : phase « Apply Request Values »
- Par exemple, si le composant texte d'un formulaire contient un nom, le composant Java associé conserve ce nom dans une variable



Phase de validation

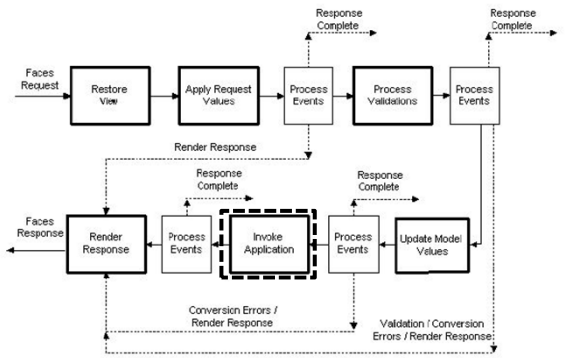
- Toutes les validations des données traitées à l'étape précédentes sont exécutées
- Les données sont converties dans le bon type Java ; elles sont ensuite validées pour voir si elles respectent les contraintes indiquées dans la page JSF
- Si une validation échoue, la main est donnée à la phase de rendu de la réponse



Phase de mise à jour du modèle

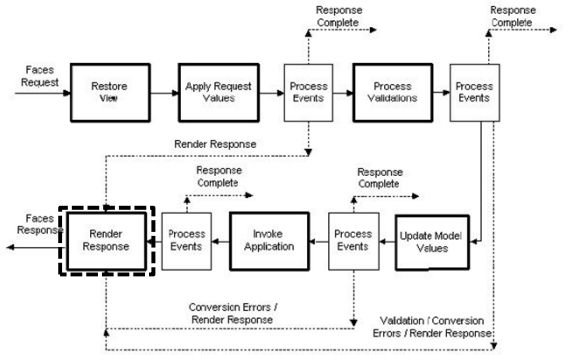
- Si les données ont été validées, elles sont mises dans les variables d'instance des *backing beans* associés aux composants de l'arbre des composants
- Exemple :

```
<h:inputText id="nom"
  value="#{backingBean.utilisateur.nom}" />
```

Phase d'invocation de l'application

- Les actions associées aux boutons ou aux liens sont exécutées
- Le plus souvent le lancement des processus métier se fait par ces actions
- La valeur de retour de ces actions va déterminer la prochaine page à afficher (navigation)



Phase de rendu de la réponse

- La page déterminée par la navigation est encodée en HTML et envoyée vers le client HTTP

Sauter des phases

- Il est quelquefois indispensable de sauter des phases du cycle de vie

`immediate=true` sur un `UICommand`

- Cet attribut peut être ajouté à un bouton ou à un lien (**`<h:commandButton>`**, **`<h:commandLink>`**, **`<h:button>`** et **`<h:link>`**) pour faire passer immédiatement de la phase « Apply request values » à la phase « Invoke Application » (en sautant donc les phases de validation et de mise à jour du modèle)
- Exemple : bouton d'annulation d'un formulaire ; on ne veut pas que la validation des champs de saisie soit effectuée, ni que les valeurs actuelles soient mises dans le modèle

`immediate=true` sur un `EditableValueHolder`

- Cet attribut peut être ajouté à un champ de saisie, une liste déroulante ou une boîte à cocher pour déclencher immédiatement après la phase « Apply request values » la validation et la conversion de la valeur qu'il contient, avant la validation et la conversion des autres composants de la page
- Utile pour effectuer des modifications sur l'interface utilisateur sans valider toutes les valeurs du formulaire

Exemple (1/2)

- Formulaire qui contient champ de saisie du code postal et un champ de saisie de la ville
- Lorsque le code postal est saisi, un écouteur (voir partie 2 de ce cours, section « Événements ») met automatiquement à jour la ville :

```
<h:inputText valueChangeListener="#{bean.m}"
  ... onChange = "this.form.submit()" />
```

- La soumission du formulaire déclenchée par la modification du code postal ne doit pas lancer la validation de tous les composants du formulaire qui ne sont peut-être pas encore saisis

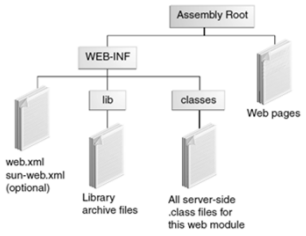
Exemple (2/2)

- La solution est de mettre « **`immediate=true`** » sur le champ code postal
- et, dans la méthode « écouteur » m,
  - de mettre automatiquement à jour la ville,
  - et aussi de mettre ce code à la fin de la méthode (pour éviter la validation des autres champs) :

```
FacesContext context =
  FacesContext.getCurrentInstance();
context.renderResponse();
```


Fichiers de configuration

Structures des fichiers



Distribution de l'application

- L'application Web (conforme au Web profile) doit être distribuée sous la forme d'un fichier « war » (*Web Application Archive*)
- Le fichier war peut contenir plusieurs fichiers de configuration

web.wml

- **`WEB-INF/web.xml`** fichier global de configuration d'une application Web Java

Exemple (1/2)

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
  <context-param>
    <param-name>
      javax.faces.PROJECT_STAGE
    </param-name>
    <param-value>Development</param-value>
  </context-param>
```

Exemple (2/2)

```
  <session-config>
    <session-timeout>
      30
    </session-timeout>
  </session-config>
  <welcome-file-list>
    <welcome-file>faces/index.xhtml</welcome-file>
  </welcome-file-list>
</web-app>
```

Pas de « / » avant faces !

Stade de développement

- Attention, le stade de développement **`Development`** ajoute automatiquement l'affichage des messages d'erreurs JSF s'ils n'y sont pas déjà
- Ne pas oublier de mettre un zone de messages dans la page pour que l'utilisateur les voit lors quand l'application sera en production

faces-config

- WEB-INF/faces-config.xml : configuration de la partie JSF de l'application, par exemple pour indiquer la navigation entre les pages ou pour déclarer les backing beans utilisés par les pages
- Pas indispensable avec JSF 2.0 grâce aux annotations Java
- Le fichier XML l'emporte sur les annotations

beans.xml

- Fichier WEB-INF/beans.xml, indispensable si CDI est utilisé
- Un fichier beans.xml vide suffit à déclencher l'utilisation de CDI (attention, si ce fichier n'existe pas, CDI n'est pas utilisé, sans aucun message d'avertissement, et donc les @Inject ne fonctionnent pas)