JSF 2.0 (*Java Server Faces*) Partie 1

Université de Nice - Sophia Antipolis
Richard Grin
Version 0.9 – 8/11/12

- 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 du type de Glassfish
- *Avertissement : les supports sur JSF sont en cours d'écriture. Il peuvent comporter des pages vides ou des points d'interrogation (le plus souvent uniquement à cause du manque de temps, pas d'un problème réel)*

Parties du support

- Ce support est divisé en 3 parties
- 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
- 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 (1/2)

Plan (2/2)

Bibliographie

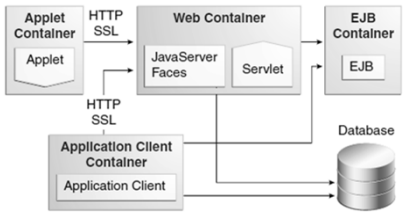
- Core JSF, 3ème édition mise à jour pour JSF 2.0. David Geary et Cay Horstmann. Prentice Hall. Le meilleur livre pour débuter.
- The Complete Reference JSF 2.0. Ed Burns et Chris Schalk. Quelques compléments intéressants par rapport au livre Core JSF.
- JSF 2.0 Cookbook. Anghel Leonard. Packt Publishing. Descriptions détaillées de quelques recettes pour résoudre des points précis.

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*)

Containers de Java EE 6



Utilité de JSF

- Créer des pages Web dynamiques
- Par exemple, une page Web qui est construite à partir de données enregistrées dans une base de données
- Une grande partie de la programmation liée à la validation des données saisies par l'utilisateur et de leur transmission au code Java de l'application est automatisée ou grandement facilitée
- Ajax sans programmation

Modèle MVC

- Framework MVC qui reprend le modèle des interfaces utilisateurs locales (comme Swing)
- Le modèle est représenté par des classes Java sur le serveur
- Les vues sont représentées par des pages JSF écrites avec un langage de description de page proche de XHTML
- Le contrôleur est le servlet FacesServlet, fourni par JSF, qui intercepte les requêtes HTTP liées aux applications JSF et qui organise les traitements JSF

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
- JSF est intégré à Java EE 6 et peut donc profiter des facilités offertes par Java EE 6 pour lancer des traitements métier (transactions, persistance, sécurité,…)

Composants sur le serveur

- Les vues sont représentées par des composants Java sur le serveur, transformées en pages HTML sur le client
- Par exemple, un composant java **`UIInputText`** du serveur sera représenté par une balise **`<INPUT>`** dans la page HTML
- Une page Web sera représentée par une vue, **`UIViewRoot`**, hiérarchie de composants JSF qui reflète la hiérarchie de balises HTML

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
- L'ancien langage de description JSP ne devrait plus être utilisé pour de nouvelles applications JSF

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** »

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

Services rendus par JSF (1/2)

- Architecture MVC pour séparer l'interface utilisateur, la couche de persistance et les processus métier, utilisant la notion d'événement ; liaisons facilitées entre les couches (backing bean, injection)
- Conversion des données entre le texte de l'interface utilisateur et les valeurs du serveur
- Validation des données (par exemple, des tailles minimales pour les chaînes de caractères)
- Automatisation de l'affichage des messages d'erreur si problèmes de conversion ou validation

Services rendus par JSF (2/2)

- Internationalisation
- Support d'Ajax sans programmation (communication en arrière-plan et mise à jour partielle de l'interface utilisateur)
- Fournit des composants standards simples pour l'interface utilisateur
- Possible d'utiliser des librairies de composants tierce et d'ajouter ses propres composants
- Adaptable à d'autres langages de balise que HTML (WML par exemple pour les téléphones portables)

Téléchargements

- Pour utiliser JSF il suffit de télécharger la version Java EE (*Enterprise Edition*) ; gratuit
- Il est fortement conseillé d'utiliser un IDE (NetBeans, Eclipse, IntelliJ IDEA ; les 2 premiers sont gratuits ; JDeveloper si on utilise la librairie de composants ADF Faces)
- Il faut aussi un serveur d'applications pour héberger les applications (GlassFish par exemple est gratuit ; il sera utilisé dans ce support)

Application Web

- Cette section est un survol rapide des fonctionnalités d'une application Web
- Plus particulièrement d'une application Web qui sépare l'interface utilisateur, le modèle de données et les traitements métier

Application Web

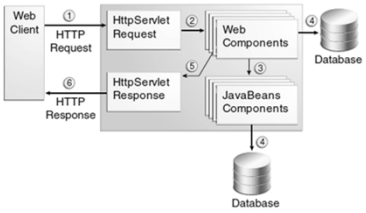
- Une application dont l'interface utilise les standards du Web, HTTP et HTML
- L'interface utilisateur est constitué de pages HTML affichées par un navigateur Web
- Les traitements sont effectués sur un serveur distant (différent de la machine qui affiche les pages Web)

Déroulement d'une application Web (1/2)

- Une page d'accueil est affichée
- L'utilisateur peut cliquer sur des boutons ou faire des choix dans des menu pour choisir un certain traitement (par exemple afficher les employés d'une entreprise)
- La page qui concerne le traitement choisi est affichée
- Les pages Web affichées contiennent des données calculées par l'application ou qui proviennent de bases de données distantes

Déroulement d'une application Web (2/2)

- Le plus souvent le traitement nécessite la saisie d'information par l'utilisateur
- Celui-ci saisit les données dans un formulaire HTML et soumet le formulaire en cliquant sur un bouton
- Le traitement est effectué par l'application; selon le cas, le formulaire est réaffiché ou l'application affiche une nouvelle page pour afficher les résultats du traitement ou pour aller plus loin dans le traitement



Types de traitement

- Le traitement peut être directement lié à l'interface utilisateur, par exemple, validation des données saisies ou affichage d'une liste liée à un choix effectué par l'utilisateur
- Le traitement peut être lié au métier de l'application, par exemple calculer le chiffre d'affaire du mois dernier, faire des statistiques ou lancer la commande d'un produit

JSF dans une application Web

- JSF facilite la vie du développeur pour l'écriture de l'interface utilisateur d'une application Web
- Par exemple pour lier les données affichées par les pages Web aux données qui proviennent du serveur distant, ou pour la validation des données saisies par l'utilisateur et l'affichage d'éventuels messages d'erreur
- JSF s'intègre dans Java EE 6 pour lancer des traitements métier et profite donc de toutes ses possibilités (transactions, persistance, sécurité,…)

Page JSF

Facelets

- Les pages JSF sont au format XHTML
- Elles sont écrites en utilisant un langage de description ; Facelets est le langage standard depuis JSF 2.0
- Le code des pages contient des balises qui décrivent les composants qui représenteront la page sur le serveur

Une page JSF

- Une en-tête (header) qui précise la version, du code Javascript ou CSS et les librairies de balises et de composants utilisées
- Le corps de la page contient du code HTML ou JSF, avec des parties en EL (*Expression Language*) **#{…}** qui permettent de référencer des propriétés et méthodes d'un bean Java
- Cette page JSF sera traduite en HTML pour être envoyée au client Web

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>
```

Extrait traduit du livre
« Java EE 6 Development with NetBeans 7 »
de David R. Heffelfinger, Packt Publishing

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>
```


Composants sur le serveur

- La page JSF est représentée par un arbre de composants Java sur le serveur
- Cet arbre des composants est créé la première fois que la page est utilisée ; il est ensuite conservé sur le serveur pour les autres accès à la page
- L'arborescence des composants est transformée en page HTML pour être envoyée au client HTTP

Commentaires (1/2)

- Par défaut les commentaires HTML (`<!-- -->`) ne fonctionnent pas pour JSF
- Une solution est de configurer l'application pour que ces commentaires soient pris en compte par Facelets en ajoutant ceci dans web.xml :

```
<context-param>
  <param-name>
     javax.faces.FACELETS_SKIP_COMMENTS
  </param-name>
  <param-value>true</param-value>
</context-param>
```


Commentaires (2/2)

- Cependant, dans certains cas, des informations mises dans des commentaires HTML peuvent être utilisés par l'application
- Le plus sûr est donc d'entourer ce que l'on veut commenter par le tag **`<ui:remove>`**
- On peut mettre ce tag dans tous les emplacements où on pourrait mettre un composant JSF

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
- En particulier, il faut bien situer où se passent les choses, entre le poste client et le poste serveur et comprendre la chaîne de traitement des données

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`**)

```
<servlet>
  <servlet-name>Faces Servlet</servlet-name>
  <servlet-class>
    javax.faces.webapp.FacesServlet
  </servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
```


URL des pages JSF

- Les pages JSF sont traitées par le servlet parce que le fichier web.xml contient une configuration telle que celle-ci :

```
<servlet-mapping>
  <servlet-name>Faces Servlet</servlet-name>
  <url-pattern>/faces/*</url-pattern>
</servlet-mapping>
```

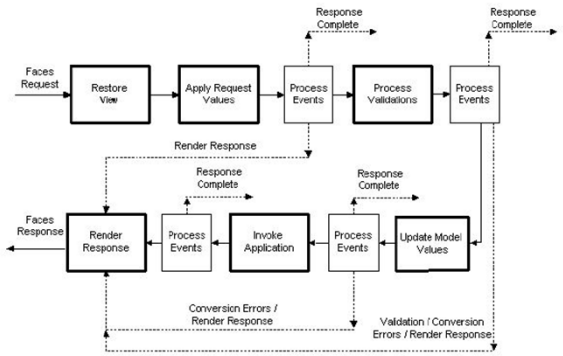
Le pattern est souvent aussi de la forme **`*.faces`** (les pages dont l'URL se termine par .faces sont considérées comme des pages JSF)

Codage - décodage

- Les pages HTML renvoyées par une application JSF sont représentées sur le serveur par un arbre de composants Java
- L'encodage est la génération d'une page HTML à partir de l'arbre des composants
- Le décodage est l'utilisation des valeurs renvoyées par un POST HTML pour donner des valeurs aux variables d'instance des composants Java, et le lancement des actions associées aux « UICommand » JSF (boutons ou liens)

Cycle de vie

- Le codage/décodage fait partie du cycle de vie des pages JSF
- Le cycle de vie est composé de 6 phases
- Ces phases sont gérées par le servlet « Faces » qui est activé lorsque qu'une requête demande une page JSF (correspond au pattern indiqué dans le fichier web.xml ; le plus souvent /faces/* ou *.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 requêtes

La vue

- Cette requête HTTP correspond à une page JSF (par exemple *.faces) et est donc 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

Contenu de la vue

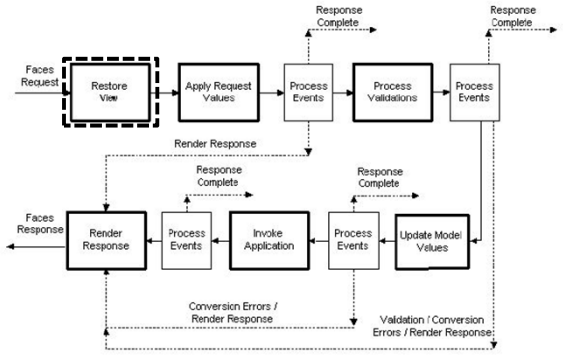
- Cette vue est un arbre dont les éléments sont des composants JSF (composants Java) qui sont sur le serveur (des instances de classes qui héritent de **`UIComponent`**)
- Sa racine est de la classe **`UIViewRoot`**

Construction vue et page HTML

- Une vue formée des composants JSF est donc 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 »

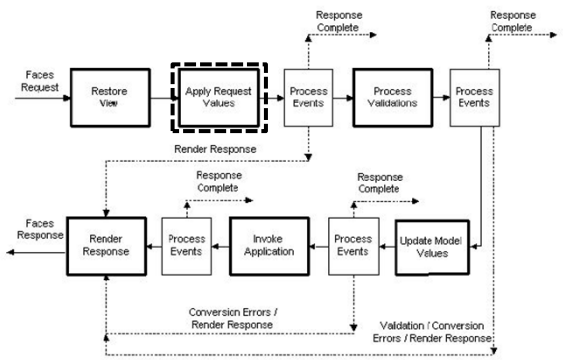
Traitement d'un formulaire

- 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



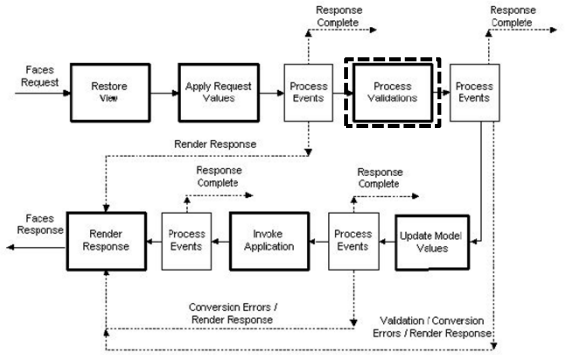
Phase de restauration de la vue

- 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
- En fait, chaque composant de la vue récupère ses propres paramètres dans la requête HTTP

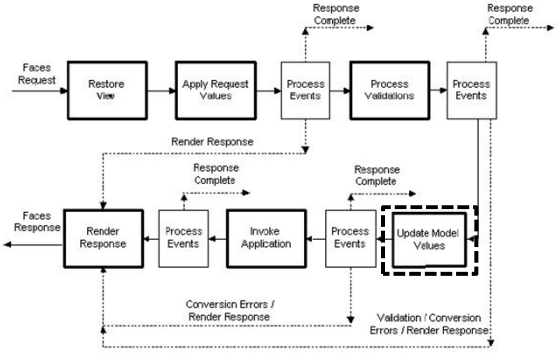


Phase de validation

- 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

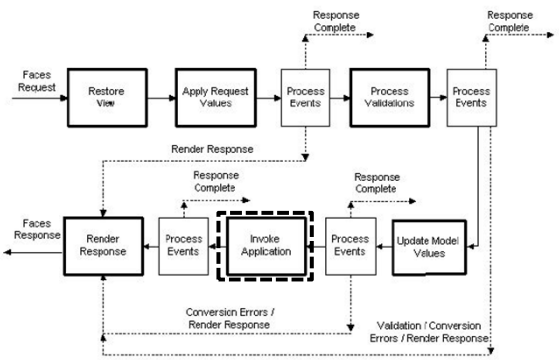
Phase de validation - implémentation

- Chaque composant valide et convertit les données qu'il contient
- Si un composant détecte une valeur non valable, il met sa propriété « **`valid`** » à false et il met un message d'erreur dans la file d'attente des messages d'erreur et les phases suivantes sont sautées pour aller directement dans la phase de rendu « render response » (dans laquelle les messages d'erreur seront affichés)



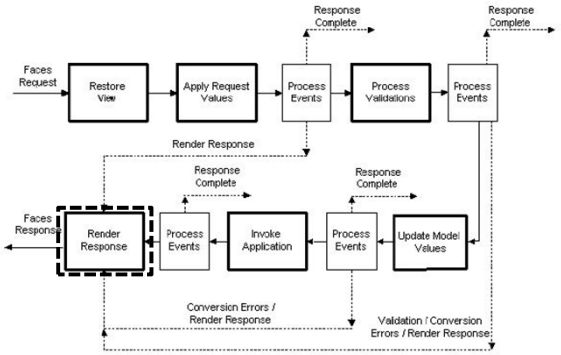
Phase de mise à jour du modèle

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



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
- Par exemple, si l'utilisateur clique sur un bouton d'annulation, on ne veut pas que la validation des champs de saisie soit effectuée, ni que les valeurs actuelles soient mises dans le modèle
- Autre exemple : si on génère un fichier PDF à renvoyer à l'utilisateur et qu'on l'envoie à l'utilisateur directement sur le flot de sortie de la réponse HTTP, on ne veut pas que la phase de rendu habituelle soit exécutée

`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 directement (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)
- En effet, en ce cas, les événements « action » sont lancés dès la phase « Apply request values », ce qui lance l'exécution des écouteurs « action » et donc la navigation éventuelle vers une autre page

Exemple

- Soit un formulaire qui a des champs requis (**`required`**) et un bouton d'annulation
- Si l'utilisateur clique sur le bouton d'annulation, le plus souvent il n'aura pas rempli les champs requis
- Il est nécessaire de mettre la valeur de l'attribut **`immediate`** du bouton d'annulation à **`true`** pour éviter que JSF ne génère un message d'erreur pour les champs requis

Autre exemple

- Une page pour saisir ses login et mot de passe ; un lien **<h:commandeLink>** pour créer un nouveau compte dans le formulaire
- On ne pourra aller créer un nouveau compte si on n'a pas déjà tapé un login et un mot de passe (car ils sont « **required** »)
- Si on ajoute « **immediate=true** » sur le lien qui envoie vers la page de création du nouveau compte, on indique à JSF qu'il doit effectuer immédiatement la navigation (phase « Invoke Application »), sans valider les autres composants du formulaire

immediate=true sur un **EditableValueHolder** (1/2)

- 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

immediate=true sur un **EditableValueHolder** (2/2)

- Ce qu'il faut faire pour cela :
  - mettre « **immediate = true** » sur tous les composants dont les valeurs doivent être converties, validées et traitées tout de suite
  - si on ne veut pas que les autres champs soient ensuite validés, il faut mettre l'instruction « **FacesContext.renderResponse()** » dans l'écouteur de type **ValueChangeListener** d'un champ qui contient « **immediate = true** »

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)

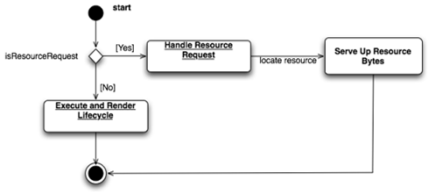
- La solution est de mettre « **immediate=true** » sur le champ code postal et, dans la méthode « écouteur » m, de mettre automatiquement à jour la ville de mettre ce code à la fin de la méthode (pour éviter la validation des autres champs) :

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


Phase listeners

- Il est possible de lancer l'exécution d'un code Java aux changements de phase du cycle de vie
- Sera étudié dans la 3ème partie de ce cours sur JSF (section « Evénements »)

Pas de cycle de vie pour les ressources



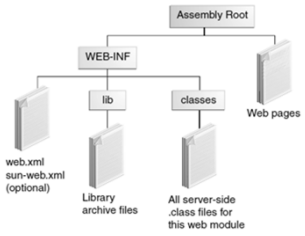
Ressource : image, fichier CSS,…

Développement d'une application Web

Étapes du développement

- Écrire le code des pages JSF et des beans (classe Java associées à l'interface graphique ou classes métier)
- Écrire le code du fichier descripteur de déploiement web.wml (et éventuellement d'autres fichiers comme glassfish-web.xml avec Glassfish)
- Tester en local en déployant sur un container Web
- Déployer sur le container Web final
- Tester

Structures des fichiers



Fichier de distribution

- L'application doit être distribuée sous la forme d'un fichier « war » (*Web Application Archive*), ou « ear » (*Enterprise Application Archive*) si on préfère séparer les EJB ou si l'application contient des composants de Java EE qui ne peuvent être contenus dans un fichier war (pas contenus dans le « Web profile » de Java EE 6)
- Les fichiers war et ear sont des fichiers au format jar avec des fichiers de configuration et des structures particulières

Rappels sur HTTP

- Un minimum de connaissances sur le protocole HTTP est requise pour comprendre certains aspects de JSF

Requête HTTP

- Le protocole HTTP permet à un client HTTP d'envoyer plusieurs types de requêtes à un serveur
- Les 2 types de base principaux sont les types GET et POST
- GET permet au client de demander une page HTML
- POST permet au client de passer des données au serveur

Réponse du serveur HTTP

- En réponse à une requête d'un client HTTP, un serveur HTTP renvoie au client une réponse

Format d'une requête

- La première ligne contient le type de la requête (GET par exemple), suivi de l'URI qui désigne le destinataire de la requête, suivi du numéro de la version HTTP utilisée
- Ensuite peuvent venir des lignes facultatives (en-têtes) pour indiquer la configuration du client ou des formats de documents acceptés en réponse
- En dernier la requête peut contenir une « entité » composée de données qui pourront être utilisées par le serveur (essentiellement réservé à la requête POST) ; ligne vide avant l'entité

Format d'une réponse

- Une première ligne d'état qui contient la version HTTP utilisée, le code d'état (nombre ; par exemple, 200 indique que tout s'est bien passé) et la description du code (« OK » pour 200)
- Viennent ensuite des lignes d'en-têtes (comme pour la requête du client)
- Finalement, le corps de la réponse est envoyé ; il peut s'agir du contenu d'un fichier HTML ou d'un contenu généré par un programme du serveur

GET

- La fonctionnalité de base de HTTP est de permettre à un navigateur (client HTTP) de récupérer et d'afficher une page Web
- Pour cela le client envoie une requête GET au serveur
- Cette requête contient l'URL de la page Web avec, éventuellement, des paramètres (couples clé-valeur)
- La réponse contient le plus souvent un corps qui est le code d'une page HTML

Exemple de requête GET

```
GET page.html?nom=Toto&Prenom=Pierre HTTP/1.1
Accept: */*
Host: www.unice.fr
User-Agent: Generic
```


POST

- Des pages Web contiennent des questionnaires remplis par l'utilisateur du navigateur
- Lorsque l'utilisateur clique sur le bouton « Submit », les données qu'il a tapées sont envoyées au serveur par une requête POST
- Cette requête contient l'URI du code qui va traiter les données, ainsi que les données
- La réponse du serveur contient le code de la prochaine page à afficher par le navigateur

Exemple de POST

POST /appl/client.xhtml HTTP/1.1
Accept: */*
Host: www.unice.fr

nom=Toto
Prenom=Pierre

Formulaires HTML

- Une page HTML peut contenir un formulaire (<FORM>)
- Lorsque ce formulaire est soumis au serveur (bouton de type « submit »), les données entrées dans le formulaire sont transmises au serveur par une requête POST ou une requête GET
- Avec GET, les valeurs saisies sont intégrées à l'URI ; avec POST elles sont mises dans l'entité
- C'est l'attribut name des balises qui donne le nom des paramètres de la requête

Postback

- Un terme souvent utilisé par les utilisateurs du Web, et en particulier de JSF
- Il s'agit d'une requête POST qui rend au client la même page que la page de départ
- La page de départ contient un formulaire (et plusieurs autres composants) ; lorsque l'utilisateur soumet les données qu'il a saisies, une action est exécutée pour les traiter (par exemple les enregistrer), et la même page est à nouveau affichée, avec les informations les plus récentes

Architecture des applications JSF

JSF, servlets et beans

- Dans l'architecture des applications qui utilisent JSF, on trouve
  - des pages JSF écrites en XHTML
  - des ressources (images, fichiers CSS ou JavaScript)
  - des backing beans
  - des EJB
  - des classes entités (JPA)
  - des classes POJO écrits en Java
  - parfois des servlets (par exemple pour exporter des données vers un tableur)

Répartition des tâches (1/2)

- Les pages JSF sont utilisées pour l'interface avec l'utilisateur
- Les pages JSF ne contiennent pas de traitements ; pas de code Java ou autre code comme dans les pages JSP anciennement utilisées par JSF
- Les traitements liés directement à l'interface utilisateur sont écrits dans des classes Java souvent appelées *backing beans* (car elles soutiennent les pages JSF)

Répartition des tâches (2/2)

- Ces backing beans font appels à des EJB ou des classes Java ordinaire pour effectuer les traitements qui ne sont pas liés directement à l'interface utilisateur
- Les EJB sont chargés des traitements métier et des accès aux bases de données
- Les accès aux bases de données utilisent le plus souvent JPA et donc les classes Java entités

Backing bean (1/2)

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

Backing bean (2/2)

- Conserve, par exemple,
  - les valeurs à afficher ou saisies par l'utilisateur (attribut **`value`** des composants JSF)
  - une expression qui indique si un composant ou un groupe de composant doit être affiché (attribut rendered)
  - un lien vers un composant JSF (attribut **`binding`**), ce qui permet de 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 ; il exécute les processus métier ou s'occupe de la persistance des données

Templates

- Les pages d'un site Web respectent souvent une charte graphique pour que les pages aient un même aspect visuel et des fonctionnalités communes (ce qui facilite le travail de l'utilisateur)
- Avec les facelets il est très simple de définir l'aspect commun aux pages dans un fichier XHTML à part, appelé *template* (modèle, calibre, gabarit en français)
- 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
- Il peut simplement créer un nouveau composant en réunissant plusieurs composants, sans écrire de code Java (« *composite component* »)
- Il peut aussi définir un composant tout à fait nouveau mais il doit alors écrire du code Java pour indiquer comment s'affichera ce composant sur le client Web et comment seront traités les données qui viennent du client Web (saisies par l'utilisateur par exemple)

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 du type de Glassfish ou TomEE
- Les requêtes des clients Web vers les pages JSF sont alors dirigées vers le conteneur JSF
- Tomcat (projet Apache) est une solution plus légère pour traiter les pages JSF (mais il n'a pas toutes les facilités des serveurs d'application, par exemple pour traiter les transactions)

Injection

- Le container peut injecter des beans lorsque c'est nécessaire
- Par exemple, le container peut fournir une instance d'une classe Java pour valider une donnée saisie par l'utilisateur, ou une instance d'un EJB pour accéder à une base de données
- Le lien entre les différents composants Java d'une architecture JSF utilise en grande partie cette injection de code

CDI

- JSF fournit des facilités pour injecter des beans mais il est recommandé d'utiliser plutôt CDI (*Contexts and Dependency Injection*), spécification Java EE pour l'injection de code (le cours sur CDI est à part de ce support)
- En effet, CDI est plus souple et plus puissant et il peut être utilisé ailleurs que dans les applications JSF

Description d'une application

- L'architecture d'une application et les relations entre les différents composants sont décrits dans des fichiers de configuration XML ou par des annotations incluses dans du code Java
- La tendance est à l'utilisation de conventions ou d'annotations pour réduire la taille des fichiers de configuration

Fichiers de configuration

web.wml

- **`WEB-INF/web.xml`** fichier global de configuration d'une application Web Java
- Il peut contenir de nombreuses informations (ou très peu d'information si on utilise les annotations)
- L'ordre des balises n'a pas d'importance

web.xml pour JSF

- Pour JSF, il contient le plus souvent
  - le servlet « JSF » et le mapping des pages JSF (ordinairement /faces/* ou *.jsf) ; optionnel avec JSF 2.0
  - un timeout pour la session
  - un paramètre indiquant le stade du développement (Production, Development, UnitTest, SystemTest, Extension)
  - des balises pour la protection des pages

Stade de développement (1/3)

- Indiqué par le paramètre de contexte **`javax.faces.PROJECT_STAGE`** :

```
<context-param>
  <param-name>
    javax.faces.PROJECT_STAGE
  </param-name>
  <param-value>Development</param-value>
</context-param>
```


Stade de développement (2/3)

- 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
- Ne pas oublier aussi de changer **`Development`** en **`Production`** pour éviter l'affichage d'informations qui pourrait le piratage du site

Stade de développement (3/3)

- C'est rare de l'utiliser, mais on peut obtenir le stade de développement depuis du code Java :

```
FacesContext fc =
  FacesContext.getCurrentIntance();
Application appl = fc.getApplication();
ProjectStage ps = appl.getProjectStage();
switch(ps) {
  case ProjectStage.Production : ...
```


Exemple (1/3)

```
<?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 (2/3)

```
<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>
```


Exemple (3/3)

```
  <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 !

Quelques autres balises (1/2)

- Par défaut, les commentaires HTML `<!-- -->` sont interprétés par JSF ; pour qu'ils soient ignorés :

```
<param-name>
javax.faces.FACELETS_SKIP_COMMENTS
</param-name>
<param-value>true</param-value>
```


Quelques autres balises (2/2)

- Par défaut, l'état des composants d'une page JSF est conservé sur le serveur ; pour les sites avec de nombreux accès clients, on peut souhaiter conserver cet état sur le client :

```
<param-name>
javax.faces.STATE_SAVING_METHOD
</param-name>
<param-value>client</param-value>
```


glassfish-web.xml

- Fichier de configuration propre à GlassFish
- Tous les autres serveurs d'application ont aussi leurs propres fichiers de configuration qui leur permet de configurer des propriétés qui leur sont propres (consultez leur documentation)
- Une propriété bien utile pour GlassFish :

```
<parameter-encoding default-charset="UTF-8"/>
```

qui permet d'indiquer le codage utilisé pour les valeurs saisies par un utilisateur dans un formulaire et pour éviter les avertissements « PWC4011: Unable to set request character encoding to UTF-8 from context … »

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

Autres fichiers

- Fichiers nommés faces-config.xml ou dont le nom se termine par « .faces-config.xml », placés dans le répertoire WEB-INF des fichiers jar utilisés par l'application
- Fichiers listés dans le paramètre d'initialisation de l'application Web (dans web.xml) :

```
<web-app>
  <context-param>
    <param-name>javax.faces.CONFIG-FILES</param-name>
    <param-value>WEB-INF/navig.xml,...</param-value>
  </context-param>
</web-app>
```


Annotations et fichier faces-config.xml

- JSF 2.0 a développé l'emploi des annotations des classes pour configurer son fonctionnement
- Le fichier XML l'emporte sur les annotations, ce qui permet de changer la configuration même si on n'a pas le code source (ou d'éviter de recompiler si on a le code source)

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)

Exemple de beans.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-
  instance"
  xsi:schemaLocation="http://java.sun.com/xml/n
  s/javaee
  http://java.sun.com/xml/ns/javaee/beans_1_0.x
  sd">
</beans>
```


Fichiers de description de balises

- Dans le cas où des balises personnalisées sont utilisées (voir la section sur les composants personnalisés dans la partie 3 de ce cours)

Implémentations de JSF – librairies JSF

Implémentations

- Implémentation de référence de JSF
- Librairies de composants

Implémentation de référence

- Projet Mojarra
- Fournit les composants de base et avec des fonctionnalités minimales
- MyFaces est un autre projet « concurrent » de Mojarra (pas officiellement soutenu par Oracle)
- De nombreuses librairies fournissent d'autres composants ou enrichissent les fonctionnalités des composants de l'implémentation de référence
- Elles s'appuient sur Mojarra mais sont aussi compatibles avec MyFaces

Implémentation de référence

- Le projet Mojarra est l'implémentation de référence de JSF
- Fournit les balises et les composants de base
- Le projet MyFaces d'Apache est une autre implémentation de JSF

Librairies de composants

- Les composants de base ont des fonctionnalités minimales
- De nombreuses librairies fournissent d'autres composants ou enrichissent les fonctionnalités des composants de l'implémentation de référence
- Principales librairies : PrimeFaces, RichFaces, OpenFaces, IceFaces, ADF Faces

Compatibilité des librairies de composants

- L'idéal serait que toutes les librairies de composants soient compatibles entre elles, ce qui permettrait de choisir ses composants dans plusieurs librairies
- Dans la réalité ça n'est pas toujours le cas et il vaut mieux bien choisir la librairie et s'y tenir

Composant JSF

- Cette section donne des généralités sur les composants JSF
- Un support de cours à part décrit les composants JSF standards fournis avec JSF

Composant JSF

- Un composant JSF est représenté par une balise dans le code d'une page JSF
- Il peut avoir des attributs, des facets, des paramètres (voir façon de les utiliser…)**??**
- Autres généralités sur les composants JSF**??**

- **??**

Balises et composants JSF standards

Types de composants (1/2)

- 5 grands types de composants ; ceux qui
  - contiennent une valeur (implémentent ValueHolder ou EditableValueHolder pour ceux qui permettent de modifier la valeur) ; un champ de saisie <h:inputText> par exemple
  - proposent un choix à l'utilisateur parmi plusieurs propositions (implémentent EditableValueHolder) ; une liste déroulante <h:selectOneMenu> par exemple

Types de composants (2/2)

- lancent une requête POST, une action (ActionSource2) ; un bouton <h:commandButton> par exemple
- lancent une requête GET (ValueHolder de type OutcomeTarget) ; un lien (<h:link>) par exemple
- correspondent à une structure itérative (hérite de UIData) ; une datatable <h:dataTable> par exemple

Types JSF

- JSF a sa propre notion de type de composant qui sert comme identifiant du composant
- Par exemple, le type de <h:selectOneMenu> est javax.faces.HtmlSelectOneMenu
- Le type ne correspond pas nécessairement à la classe d'implémentation du composant ; par exemple, <h:column> a le type javax.faces.Column mais est implémenté par la classe javax.faces.component.UIColumn

Attributs communs aux composants

- **`id`** donne un identifiant au composant ; si le développeur ne le donne pas, JSF en génèrera un
- **`rendered`** indique si le composant doit être affiché (la valeur est le plus souvent le résultat booléen d'une méthode Java du *backing bean*, donné par une expression EL)
- **`styleClass`** donne le nom de la classe CSS pour la mise en forme du composant
- **`binding`** permet de lier le composant à une propriété d'un *backing bean* (classe Java)

Attribut **`binding`**

- Utilisé assez rarement
- Le composant peut ainsi être manipulé par le backing bean avec du code Java
- Ainsi le code peut changer les attributs du composant, ou même l'instancier
- On peut aussi utiliser cet attribut pour des traitements spéciaux, comme, par exemple, lier une dataTable pour pouvoir récupérer la ligne en cours par la méthode Java **`table.getRowData()`**