Java Server Faces

Youssef Saadi

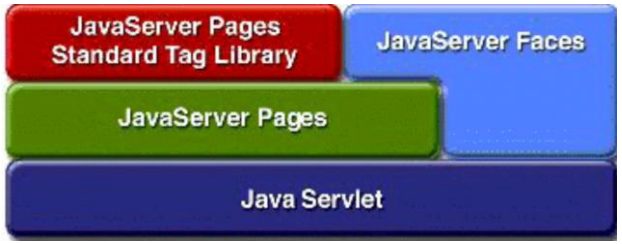
Master Informatique Décisionnelle

Faculté Des Sciences Et Techniques
Université Sultan Moulay Slimane Béni-Mellal

AU: 2019/2020

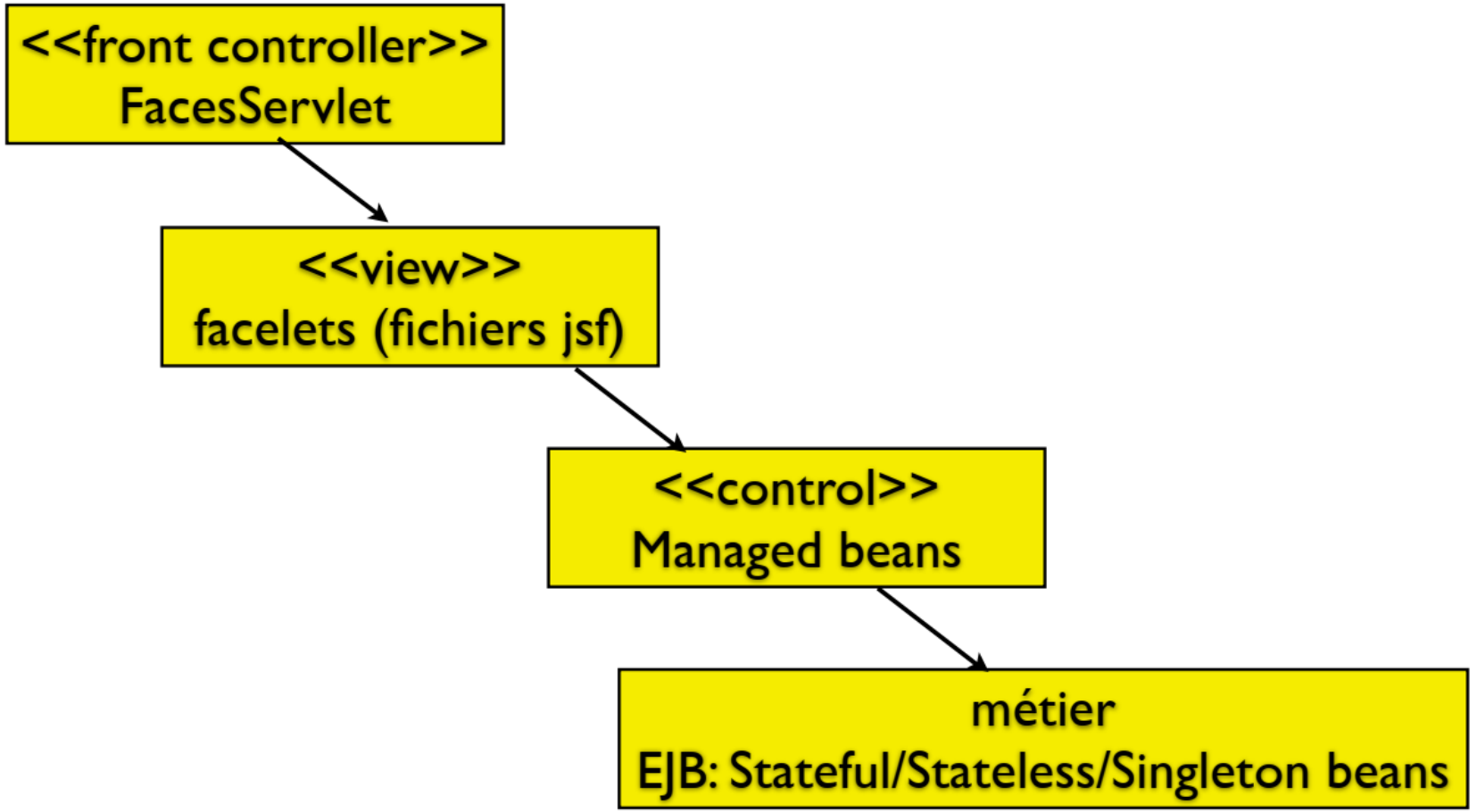
Java Server Faces

- Java Server Faces est un framework développé par Sun pour la création des interfaces utilisateur coté serveur pour les applications web .
- Framework standard pour gérer la « couche présentation » des applications Web JEE.
- Spécification JEE (**JSR-127**)
- Principales caractéristiques :



  - JSF conserve l'état des composants graphiques d'une page Web sur le serveur (UI components)
  - Modèle « orienté événements » (event listeners, validators, …)
  - JSF utilise toutes les technologies liées aux JSP (taglibs, Expression Language, etc…)

Architecture JSF



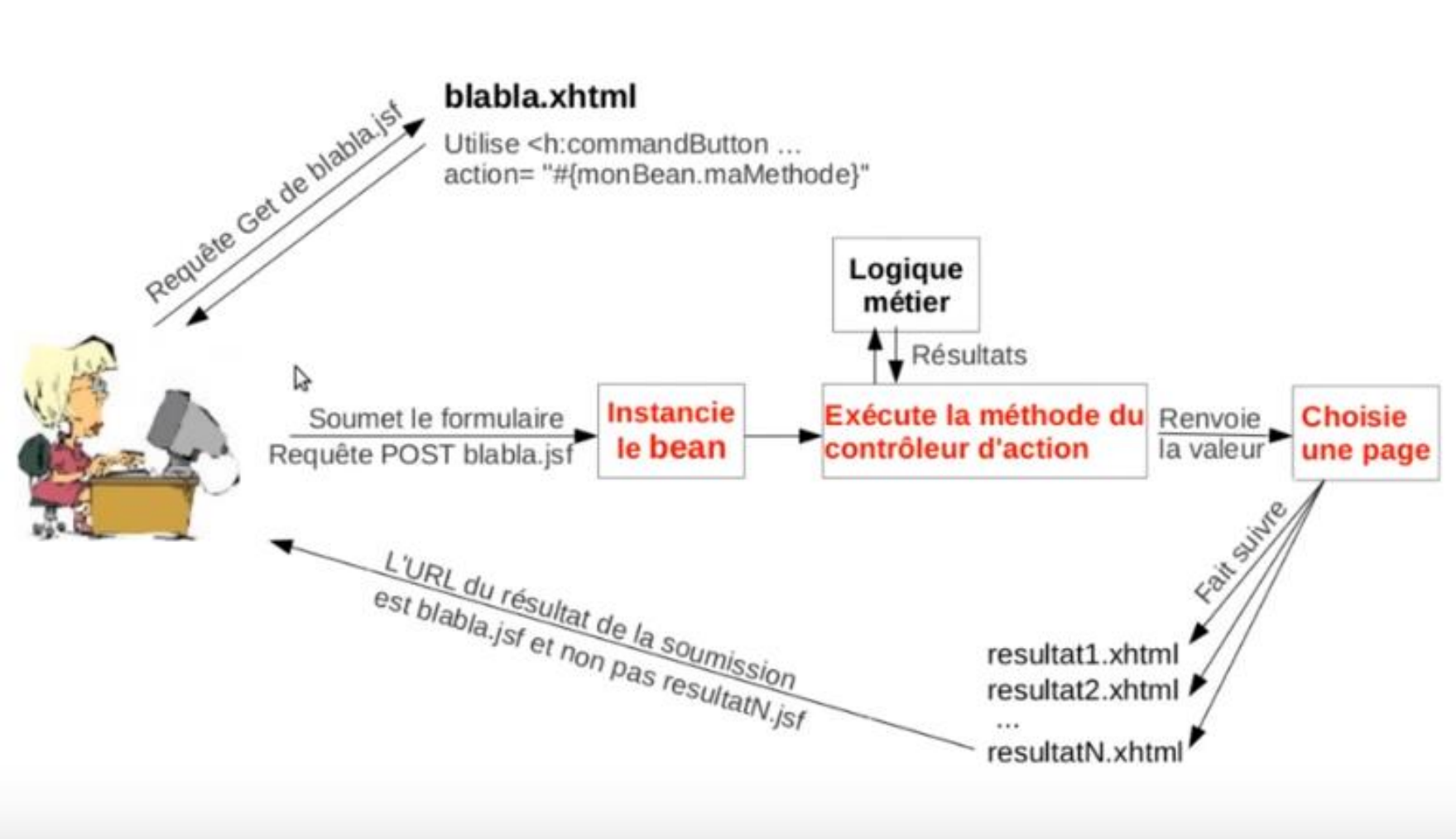
Une application JSF typique

- Des pages web
- Des tags
- Des managed Bean
- Un descripteur de déploiement (**web.xml**)
- Des fichiers de configuration de l'application (**faces-config.xml**)
- D'autres objets – validateurs, convertisseurs, écouteurs
- Composants personnalisés et leurs tags associés

JSF et MVC

- Modèle :
  - Couplé à JSF par un Bean géré.
- Vue :
  - JSP + balises JSF
- Contrôleur :
  - Servlet (FaceServlet)
  - Règles définies dans un fichier xml

Flux de contrôle de JSF (simplifié)



Cycle de vie

- **Restore view**: construire l'arbre des composants
- **Apply request values**: fixe les valeurs des paramètres
- **Validation** : vérification des valeurs pour la validation jsf
- **Update model** : copie des valeurs dans les beans. validation lors de la copie
- **Invoque application** : exécution de la commande
- **Render response**: création du résultat
- Après la plupart des phases, les événements peuvent être traités.
- **Interruption du traitement**:
  - appel de FacesContext.renderResponse : saute directement à la dernière phase
  - appel de FacesContext.responseComplete() : production directe du résultat final (par exemple PDF, image…)

Cycle de vie

- L’exécution :
  - La vue de l’application est construite ou restaurée.
  - Les valeurs de paramètre de requête sont appliquées.
  - Les conversions et les validations sont effectuées pour les valeurs des composants.
  - Les Managed Bean sont mis à jour avec les valeurs des composants.
  - La logique de l’application est appelée.
- Le rendu :
  - Génère une sortie, telle que HTML ou XHTML.

Mise en place d'une application JSF 2.x

- Installation des librairies JSF:
  - Fichier JAR de JSF sont requis
    - À omettre dans Glassfish 3, Jboss 6 et d'autres serveurs JEE 6
    - Les JAR JSTL 1.2 sont recommandés
- **faces-config.xml** contenant la configuration de l'application. Peut être vide dans le cas d'utilisation des annotations.
- **web.xml** doit déclarer un pattern url pour *.jsf et la faces servlet.

Application JSF : les fichiers XML

```xml
<?xml version="1.0" encoding="UTF-8"?>

<faces-config
    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-facesconfig_2_1.xsd"
    version="2.1">

</faces-config>
```

Application JSF : les fichiers XML

- On déclare la **faces servlet** « le Front Controller » dans le fichier web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
   <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>
```

Facelets

- **VUE**
- Ressemblent à des JSP, mais :
  - Le fichier XML est analysé, et **une représentation objet du document** est construite (JSP= texte) ;
  - Le contenu HTML final est engendré par cette représentation, appliquée aux valeurs contenues dans les Managed Bean (pas de représentation objet intermédiaire en JSP).
  - Il faut utiliser '**#**' à la place de '**$**' pour accéder aux objets Bean.

Facelet?

- **Langage de déclaration permettant la conception des vues JSF.**
- **Les Facelets sont utilisées par défaut dans JSF 2.x**
- **Les Facelets permettent le templating, la création de composants composites, la réutilisation de contenu, etc…**

```
<!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">
<h:head>
...
</h:head>
<h:body>
...
<h:form>
...
</h:form>
...
</h:body>
</html>
```

Managed Bean

- **Contrôle (donc, font partie de l'UI)**
- Annotés avec **@ManagedBean** ou **@Named**
- **Portées** (scope):
  - @SessionScoped,
  - @RequestScoped,
  - @ApplicationScoped,
  - @ViewScoped.
- Accès possible aux EJB par injection.

Exemple : Managed Bean

```
import java.io.Serializable;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;

@ManagedBean (name="counter")
@SessionScoped
public class CounterBean implements Serializable{
    private int countValue = 0;

    public int getCountValue() {
        return countValue++;
    }

    public void setCountValue(int countValue) {
        this.countValue = countValue;
    }

}
```

- **L'utilisation des Bean dans JSF permettent :**
  - **l'affichage des données provenant de la couche métier**
  - **le stockage des valeurs d'un formulaire**
  - **la validation des valeurs**
- **Java Bean dont le cycle de vie va être contrôlé par le framework JSF en fonction des besoins.**

Exemple : Facelet

```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "ht
2 <html xmlns="http://www.w3.org/1999/xhtml"
3       xmlns:ui="http://java.sun.com/jsf/facelets"
4       xmlns:f="http://java.sun.com/jsf/core"
5       xmlns:h="http://java.sun.com/jsf/html"
6       xmlns:c="http://java.sun.com/jsp/jstl/core">
7
8 <h:head></h:head>
9 <h:body>
10 <h1>test</h1>
11
12 Count value : <h:outputText value="#{counter.countValue}" />
13
14
15 </h:body>
16 </html>
17
```

Formulaire en JSF

- Inclus dans une balise <**h:form**>,
- Gérés par un **Managed bean**,
- Les champs du formulaire utilisent des balises JSF (et non HTML),
- Le managed bean va automatiquement récupérer les valeurs des champs et réaliser la ou les actions du formulaire.

Exemple de formulaire JSF

```
<h:head></h:head>
<h:body>
    <h:form>
        <h:panelGrid columns="2">
            <h:outputLabel value="premier operande" />
            <h:inputText value="#{formb.op1}" />
            <h:outputLabel value="2eme operande" />
            <h:inputText value="#{formb.op2}" />
            <h:commandButton action="#{formb.somme()}" value="Ajouter" />
            <h:outputLabel value="#{formb.res}" />
        </h:panelGrid>
    </h:form>

</h:body>
</html>
```

Lie le contrôle à une propriété du bean

La méthode qui est exécutée quand le bouton est pressé

Bean de support

```
@ManagedBean(name = "formb")
@SessionScoped
public class FormBean {
    private int op1;
    private int op2;
    private int res;

    public int getRes() {
        return res;
    }

    public void setRes(int res) {
        this.res = res;
    }
    ...
```

```
    public void setOp2(int op2) {
        this.op2 = op2;
    }
    public void somme() {
        res= op1+op2;
    }
}
```

Pas de valeur de retour
On reste sur la même page

Formulaire

- Les données du formulaire sont ***injectées*** dans le bean.
- Validation automatique ; affichage de messages d'erreurs si la validation n'est pas correcte.
- Le formulaire est réaffiché s'il n'est pas valide.

Formulaire et messages d'erreur

- id: identifie le champ
- label : nom utilisé dans le message d'erreur
- for : réfère à l'identifiant du champ

```
<h:inputText label="a" id="a" value="#{addControl.first}"/>
<h:message for="a"/>
```

Exemple

```
<h:head></h:head>
<h:body>
    <h:form>
        <h:inputText id="txt" size="10" label="Txt" required="true">
            <f:validateLength for="txt" minimum="5" max="20" />
        </h:inputText>
        <h:message for="txt" />
        <h:commandButton action="#{formb.stayHere()}" value="Valider" />
    </h:form>

</h:body>
</html>
```

Navigation entre les pages

- La navigation entre les pages indique quelle page est affichée quand l'utilisateur clique sur un bouton pour soumettre un formulaire ou sur un lien.
- La navigation peut être définie par des règles dans le fichier de configuration **faces- config.xml** ou par des valeurs écrites dans le code Java ou dans la page JSF.
- La navigation peut être statique : définie « en dur » au moment de l'écriture de l'application.
- La navigation peut aussi être dynamique : définie par l'état de l'application au moment de l'exécution (en fait, par la valeur retournée par une méthode).

Navigation en JSF

- Les actions peuvent renvoyer une String, qui dit quelle est la prochaine vue.
- On peut aussi, dans un lien de la JSF, donner le nom d'une vue au lieu d'une action.
- La chaîne renvoyée par l'action ou le lien peut être le nom d'une vue, ou un résultat abstrait (comme « success » ou « error ») qui sera utilisé par le fichier faces-config.xml pour décider de la prochaine vue.

Exemple simple

- Crée un lien avec le texte «exemple» :
  - saute à la vue «add», si elle existe (s'il y a un fichier add.xhtml dans les *facelets*).
  - **ou** : cherche une règle de navigation pour «add»

(sur la page page index.xhtml)

```
<h:link value="exemple" outcome="add"/>
```

Exemple avec une règle de navigation

dans add2.xhtml

```
<h:form>
a <h:inputText label="a" id="a" value="#{addControl.first}"/>
b <h:inputText label="b" id="b"  value="#{addControl.second}"/>
 <h:commandButton value="add" action="#{addControl.add2()}"/>
 </h:form>
```

dans addResult.xhtml

```
<h:body>
 Addition result :
  <h:outputText value="#{addControl.result}"/>
  <h:link value="back to index" outcome="index"/>
 </h:body>
```

managed bean:

```
@ManagedBean
@RequestScoped
public class AddControl {
    private int first, second, result;
     ....
     public String add2() {
        result= first+ second;
        return "success";
     }
}
```

Exemple avec une règle de navigation

- Quand une action appelée depuis add2.xhtml retourne «**success**», alors on montre **addResult.xhtml**.
- Permet de séparer la navigation du résultat des actions.

faces-config.xml

```
<navigation-rule>
    <from-view-id>/add2.xhtml</from-view-id>
    <navigation-case>
        <from-outcome>success</from-outcome>
        <to-view-id>/addResult.xhtml</to-view-id>
    </navigation-case>
</navigation-rule>
```

Les composants JSF

- <h:form>: délimite un formulaire
- <h:panelGrid columns="2"> présente son contenu sur deux colonnes

Affichage

```
<h:outputText value="#{bean.texte}"/>
```

un texte

```
<h:outputLabel value="Nom" for="idNom"/>
<h:inputText id='idNom' value="#{bean.value}"/>
```

Nom

```
<h:graphicImage value='toto.png'/>
```

Boutons et liens

- Deux types: liés aux actions (command) ou non.

```
<h:commandButton action="#{bean.faire()}" value="allez!"/>
```

allez!

```
<h:commandLink action="#{bean.faire()}" value="allez!"/>
```

allez!

```
<h:outputLink value="http://www.google.com/search">
   Google
<f:param name="q" value="glg203"/>
</h:outputLink>
```

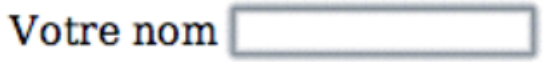
Google

```
<h:button outcome="index" value="page d'accueil"/>
```

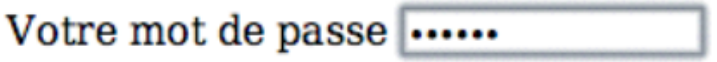
page d'accueil

Entrées

```
<h:panelGrid columns="2">
  Votre nom <h:inputText value="#{bean.value}"/>
</h:panelGrid>
```

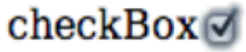



```
Votre mot de passe
<h:inputSecret  value="#{bean.value}"/>
```




```
<h:inputTextarea  value="#{bean.value}" cols="80" rows="10"/>
```

```
<h:selectBooleanCheckbox value="#{bean.value}">
checkBox
</h:selectBooleanCheckbox>
```



Listes

- Avec entrée en dur

```
<h:selectOneMenu value="#{bean.value}">
  <f:selectItem itemDescription="lundi" itemValue="0"/>
  <f:selectItem itemDescription="mardi" itemValue="1"/>
</h:selectOneMenu>
```


```
<h:selectOneMenu value="#{bean.value}">
  <f:selectItems value="#{bean.choixPossibles}"/>
</h:selectOneMenu>
```

De même : selectOneListBox, selectOneRadio

choixPossibles retourne une liste.
le toString() des éléments est utilisé pour l'affichage

Affichage de tableau

```
<h:dataTable var="p" value="#{personControl.personList}">
        <h:column>
            <f:facet name="header">
                <h:outputText value="Surname"/>
            </f:facet>
            <h:outputText value="#{p.surname}"/>
        </h:column>
        <h:column>
            <f:facet name="header">
                <h:outputText value="Name"/>
            </f:facet>
            <h:outputText value="#{p.name}"/>
        </h:column>
</h:dataTable>
```

Affichage conditionnel en JSF

- Facile: les balises JSF ont un attribut booléen «**rendered**».
- Pour un contenu complexe, on peut le mettre dans un **panelGroup**

```
<h:outputText rendered="#{empty personControl.personList}"
              value="The person list is empty"/>
<h:panelGroup
      rendered="#{not empty personControl.personList}">
  The person list contains
  <h:outputText value="#{personControl.personList.size()}"/>
  persons.
</h:panelGroup>
```

Injection des paramètres de requête dans un Bean

- Par exemple pour des pages qui sont chargées en mode GET.
- Pour les Bean de scope « **request** » uniquement.
- Annoter la propriété dans le bean avec @ManagedProperty(value="#{param.*NAME*}") où « **name** » est le nom du paramètre.

Méthode GET en JSF

- Pour forcer l'usage de la méthode GET, utilisez h:button ou h:link.

```
<h:link value="view"  outcome="viewMessage">
    <f:param name="messageId" value="#{m.id}"/>
</h:link>
```

appellera http://adresse/viewMessage?messageId=**N**

Traitement des paramètres GET
JSF 2.x: Paramètres de vue

- Les paramètres d'une vue sont définis par des balises <f:viewParam>incluses dans une balise <f:metadata>(à placer au début de la page destination de la navigation, avant les <h:head>et <h:body>) :
- <f:metadata>
  <f:viewParamname="n1" value="#{bean1.p1}"/>
  <f:viewParamname="n2" value="#{bean2.p2}" />
  <f:metadata>

<f:viewParam>

- L'attribut **name** désigne le nom d'un paramètre HTTP de requête GET,
- L'attribut value désigne (par une expression du langage EL) le nom d'une propriété d'un bean dans laquelle la valeur du paramètre est rangée,
- Important : il est possible d'indiquer une conversion ou une validation à faire sur les paramètres, comme sur les valeurs des composants saisis par l'utilisateur.
- Un URL vers une page qui contient des balises **<f:viewParam>**contiendra tous les paramètres indiqués par les **<f:viewParam>** s'il contient **«includeViewParams=true»**
- Exemple **:<h:commandButton value=...action="page2?faces-redirect=true &includeViewParams=true**«
  - Dans le navigateur on verra l'URL avec les paramètres HTTP.

Fonctionnement de includeViewParams

1. La page de départ a un URL qui a le paramètre includeViewParams,
2. Elle va chercher les <f:viewParam>de la page de destination. Pour chacun, elle ajoute un paramètre à la requête GET en allant chercher la valeur qui est indiquée par l'attribut value du <f:viewParam>,
3. A l'arrivée dans la page cible, la valeur du paramètre est mise dans la propriété du bean indiquée par l'attribut value.

Cela revient à faire passer une valeur d'une page à l'autre.

Si la portée du bean est la requête et qu'il y a eu redirection, cela revient plus précisément à faire passer la valeur d'une propriété d'un bean dans un autre bean (de la même classe).

Attribuer une valeur à un paramètre

- Il y a plusieurs façons de donner une valeur à un paramètre de requête GET ; les voici dans l'ordre de priorité inverse (la dernière façon l'emporte)
- Dans la valeur du outcome
  <h:link outcome="page?p=4&p2='bibi' " ...>
  <h:link outcome="page?p=#{bean.prop+ 2} " ...>
- Avec les paramètres de vue<h:link outcome="page" includeViewParams="true"...>
- Avec un <f:param>
  <h:link outcome="page" includeViewParams="true"...>
  <f:param name="p" value.../> </h:link>

Exemple : page.xhtml

```
<!DOCTYPE …>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns=...>
<f:metadata>
  <f:viewParam name="param1"
               value="#{bean.prop1}"/>
  <f:viewParam name="param2" value="#{…}"/>
</f:metadata>
<h:head>…</h:head>
<h:body>
  Valeur : #{bean.prop1}
```

Fonctionnement

- Si page.xhtml est appelé par la requête GET suivante: page.xhtml?param1=v1&param2=v2, la méthode setProp1du bean est appelée avec l'argument v1 (idem pour param2 et v2)
- Si un paramètre n'apparait pas dans le GET, la valeur du paramètre de requête est null et le setter n'est pas appelé (la propriété du bean n'est donc pas mise à null).

FacesContext

- Permet l'accès bas niveau aux données de la requête, à la session, etc.
- Se récupère dans un ManagedBean avec:

  FacesContext ctx= FacesContext.getCurrentInstance();

- ctx.getExternalContext() donne accès à la requête http, à la session…

Validation

- Possible à plusieurs niveaux:
  - **Validation JSF** : installation de validateurs associés à des champs
  - **Validation de bean** : règles associées aux propriétés des Managed Bean.
- Il est possible de contraindre la validation des champs dans le textes des JSF, soit en spécifiant :
  - un ***validateur*** comme argument,
  - soit en fournissant un ***validateur*** dans ***l'annotation*** <f:validator…>

Validation : Exemple

```
@FacesValidator("net.youssadi.emailV")
public class EmailValidateur implements Validator{
    private static final String EMAIL_PATTERN = "^[_A-Za-z0-9-]+(\\." +
            "[_A-Za-z0-9-]+)*@[A-Za-z0-9]+(\\.[A-Za-z0-9]+)*" +
            "(\\.[A-Za-z]{2,})$";

    private Pattern pattern;
    private Matcher matcher;
    public EmailValidateur() {
        pattern = Pattern.compile(EMAIL_PATTERN);
    }
```

Validation : Exemple

```
@Override
public void validate(FacesContext context, UIComponent component, Object value)
        throws ValidatorException {
    matcher = pattern.matcher(value.toString());
    if(!matcher.matches()){

        FacesMessage msg =
            new FacesMessage("E-mail validation failed.",
                    "Invalid E-mail format.");
        msg.setSeverity(FacesMessage.SEVERITY_ERROR);
        throw new ValidatorException(msg);

    }

}
```

Validation : Exemple

```java
@ManagedBean(name="user")
@SessionScoped
public class Userbean {
    String email;

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

}
```

Managed Bean

page.xhtml

```
<h:form>

    <h:panelGrid columns="3">

      Enter your email :

      <h:inputText id="email" value="#{user.email}" size="20"
            required="true" label="Email Address">

            <f:validator validatorId="net.youssadi.emailV" />

      </h:inputText>

      <h:message for="email" style="color:red" />

    </h:panelGrid>

    <h:commandButton value="Submit" action="result" />

</h:form>
```

Validation : Exemple

Utilisation d'un validateur

- Le validateur doit implémenter l'interface javax.faces.validator.Validator
- On doit le déclarer et lui donner un ID, **soit** dans faces-config.xml, **soit** en l'annotant avec @FaceValidator("**ID**")
- On l'emploie avec <f:validator id="…">

```
<?xml version='1.0' encoding='UTF-8'?>
<faces-config version="2.2" …>
    <!-- déclaration du valideur (en dehors de « application »!!!)->
    <validator>
        <validator-id>hexValidator</validator-id>
        <validator-class>glg203.ui.HexValidator</validator-class>
    </validator>

    <application>
        <locale-config>
            <default-locale>fr</default-locale>
        </locale-config>
    </application>
</faces-config>
```

c'est optionnel, les annotations étant plus simples

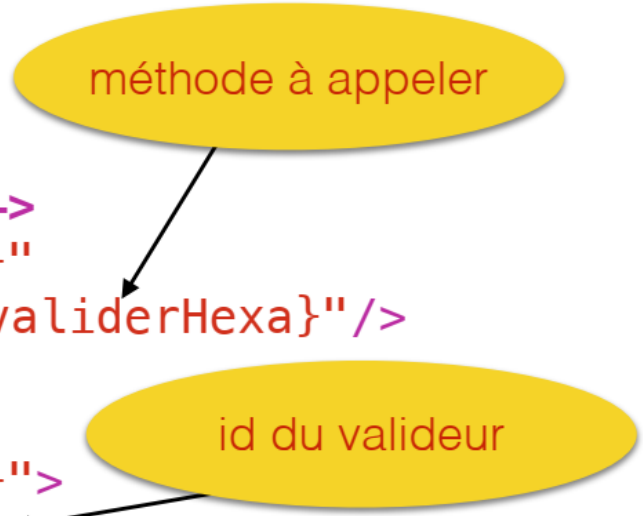
Utilisation dans la JSF

```
<h:form>

  <!– champ validé par un appel de méthode –>
  <h:inputText value="#{demoValidation.val1}"
               validator="#{demoValidation.valideurHexa}"/>

  <!– champ validé par un objet valideur –>
  <h:inputText value="#{demoValidation.val2}">
     <f:validator validatorId="hexValidator"/>
  </h:inputText>

  <h:outputText value="#{demoValidation.resultat}"/>
  <h:commandButton action=« #{demoValidation.action()} »
                   value="Somme"/>
</h:form>
```



Validation par méthode

```
@Named(value = "demoValidation")
@RequestScoped
public class DemoValidation {
   private String val1="0", val2="0";
   private int resultat;

   public void validerHexa(FacesContext context,
            UIComponent toValidate,
            Object value) {
        String string = (String) value;
        if (! string.matches("^[0-9A-Fa-f]+$")) {
            ((UIInput) toValidate).setValid(false);
            FacesMessage message =
                 new FacesMessage("Mauvais format de nombre");
            context.addMessage(toValidate.getClientId(context),
                 message);
        }
    }
  //… reste des méthodes
}
```

Validation des propriétés

- Automatique pour le tapage
  - Plus précis et général : annotations de javax.validation (JSR 303)

```
@Named("helloBean")
@RequestScoped
public class HelloBean {

    @Size(min = 1, max = 100,message = "{erreur.longueur}")
    private String nom;

    @Pattern(regexp = "(\\p{L}| )+", message = "{erreur.prenom}")
    private String prenom;

    @Min(0)
    @Max(200)
    private int age;
```

Personnalisation des messages d'erreurs

- Méthode 1:
- Localiser le fichier Messages.properties situé dans le jar jsf-api
  - javax.faces. Messages.properties
- Sélectionner les messages d'erreurs à personnaliser et les copier dans un autre fichier avec l'extension .properties
  - Les réécrire

MyMessage.properties

```
javax.faces.component.UIInput.REQUIRED = {0}: Erreur de validation
javax.faces.component.UIInput.REQUIRED_detail = {0}: Une donn\u00e9e est obligatoire ici.
```

- Ajouter une déclaration au niveau application de ce fichier.

```
<application>
    <message-bundle>MyMessage</message-bundle>
</application>
```

Personnalisation des messages d'erreurs

- Méthode 2

```
@Named
@RequestScoped
public class HelloController {
    @Size(min=5, message="Name must be at least 5 letters long")
    private String name;

    //Getter setter etc...

}
```

- Méthode 3

```
<x:inputXxx ... required="true" requiredMessage="Value is required" />
```

Internationalisation des messages

- Dans JSF 2.0, nous pouvons changer par programmation la locale de l'application.
- La locale permet à JSF de supporter plusieurs langues [Internationalisation]

```
//this example change locale to france
FacesContext.getCurrentInstance().getViewRoot().setLocale(new Locale('fr');
```

Internationalisation des messages

- Méthode
- On crée deux fichier de propriété : par exemple « *welcome.properties » et welcome_fr.properties*
- *On suppose que chaque fichier contiendra une propriété dans son langage:*
  - jsf.welcome = Happy learning JSF 2.0 → *welcome.properties*
  - jsf.welcome = Bon apprentissage de JSF 2.0 → *welcome_fr.properties*

Dans le faces-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<faces-config
    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-facesconfig_2_0.xsd"
    version="2.0">
     <application>
           <locale-config>
                <default-locale>en</default-locale>
           </locale-config>
       <resource-bundle>
        <base-name>com.mkyong.welcome</base-name>
        <var>msg</var>
       </resource-bundle>
     </application>
</faces-config>
```

- La locale par défaut est en anglais.
- On définit une ressource bundle qui va comporter les valeurs d'affichage en multi langue.

Exemple d'utilisation : ManagedBean

```java
@ManagedBean(name="language")
@SessionScoped
public class LanguageBean implements Serializable{

    private static final long serialVersionUID = 1L;

    private String localeCode;

    private static Map<String,Object> countries;
    static{
        countries = new LinkedHashMap<String,Object>();
        countries.put("English", Locale.ENGLISH); //label, value
        countries.put("Chinese", Locale.SIMPLIFIED_CHINESE);
    }

    public Map<String, Object> getCountriesInMap() {
        return countries;
    }


    public String getLocaleCode() {
        return localeCode;
    }
```

```java
public void setLocaleCode(String localeCode) {
    this.localeCode = localeCode;
}

//value change event listener
public void countryLocaleCodeChanged(ValueChangeEvent e){

    String newLocaleValue = e.getNewValue().toString();

    //loop country map to compare the locale code
            for (Map.Entry<String, Object> entry : countries.entrySet()) {

           if(entry.getValue().toString().equals(newLocaleValue)){

            FacesContext.getCurrentInstance()
                .getViewRoot().setLocale((Locale)entry.getValue());

          }
           }
}
```

Exemple d'utilisation : la Facelet

```
<h:body>

    <h1>JSF 2 internationalization example</h1>

 <h:form>

<h2>
    <h:outputText value="#{msg['welcome.jsf']}" />
</h2>

<h:panelGrid columns="2">

    Language :
    <h:selectOneMenu value="#{language.localeCode}" onchange="submit()"
        valueChangeListener="#{language.countryLocaleCodeChanged}">
        <f:selectItems value="#{language.countriesInMap}" />
    </h:selectOneMenu>

</h:panelGrid>

  </h:form>

</h:body>
```

default.xhtm

On utilise la variable « **msg** » et on accède à la valeur de la propriété « **jsf.welcome** »

Exemple d'utilisation : le rendu

Phase de conversion

- Mise à jour du model
- Parcours de l'arbre des composants, de l'IHM interne par JSF
- Affectation des propriétés côté serveur
  - Des conversions sont peut-être à faire… JJ/MM/AAAA …
  - Et inversement ….
- Deux fonctions : String -> Object et Object -> String
  - Bijection réciproque ?
- Mise à jour des propriétés du Bean
- Conversion standard
- Ou personnalisée

Exemple de convertisseurs standards

```
<h:inputText value="#{demoConverterBean.dateEtHeure}">
                <f:convertDateTime/>
</h:inputText>
```

```
<h:inputText value="#{demoConverterBean.valeur}">
      <f:convertNumber pattern="#000.00" />
</h:inputText>
```

Format d'affichage

Conversion des nombres

Si l'on assume que **receipt.amount** contient la valeur 0.1

```
<h:outputText value="#{receipt.amount}" >
    <f:convertNumber minFractionDigits="2" />
</h:outputText>
```

Affiche : 0.10

```
<h:outputText value="#{receipt.amount}" >
    <f:convertNumber pattern="#0.000" />
</h:outputText>
```

Affiche : 0.100

```
<h:outputText value="#{receipt.amount}" >
    <f:convertNumber currencyCode="GBP" type="currency" />
</h:outputText>
```

Affiche : GBP0.10

```
<h:outputText value="#{receipt.amount}" >
    <f:convertNumber type="percent" />
</h:outputText>
```

Affiche : 10%

Conversion des dates

```
 Receipt Date :
<h:inputText id="date" value="#{receipt.date}"
   size="20" required="true"
   label="Receipt Date" >

   <f:convertDateTime pattern="d-M-yyyy" />
 </h:inputText>

 <h:message for="date" style="color:red" />
```

```
Receipt Date :
<h:outputText value="#{receipt.date}" >
    <f:convertDateTime pattern="d-M-yyyy" />
</h:outputText>
```

Le format de date dans l'attribut pattern est défini dans java.text.SimpleDateFormat.

Conversion personnalisée

```
@FacesConverter("net.youssadi.URLCV")
public class URLConverter implements Converter {

    @Override
    public Object getAsObject(FacesContext context, UIComponent component, String value) {
        StringBuilder url = new StringBuilder();

        if (!value.startsWith("http://", 0)) {
            url.append("http://");
        }
        url.append(value);

        try {
            new URI(url.toString());
        } catch (URISyntaxException e) {
            FacesMessage msg = new FacesMessage("Error converting URL", "Invalid URL format");
            msg.setSeverity(FacesMessage.SEVERITY_ERROR);
            throw new ConverterException(msg);
        }

        URLBookMark urlBookmark = new URLBookMark(url.toString());

        return urlBookmark;
    }
```

Conversion personnalisée

```
@Override
public String getAsString(FacesContext context, UIComponent component, Object value) {

    if (value == null)
        return "";

    return value.toString();
}
```

Conversion personnalisée

```java
@ManagedBean(name = "user")
@SessionScoped
public class UserBean {
    String bookmarkURL;

    public String getBookmarkURL() {
        return bookmarkURL;
    }

    public void setBookmarkURL(String bookmarkURL) {
        this.bookmarkURL = bookmarkURL;
    }

}
```

Conversion personnalisée

```
<h:form>

    <h:panelGrid columns="3">

    Enter your bookmark URL :

    <h:inputText id="bookmarkURL" value="#{user.bookmarkURL}" size="20"
            required="true" label="Bookmark URL" rendered="true">
            <f:converter converterId="net.youssadi.URLCV" />
    </h:inputText>

    <h:message for="bookmarkURL" style="color:red" />

    </h:panelGrid>

    <h:commandButton value="Submit" action="result" />

</h:form>
```

Conversion personnalisée

```
<h:head></h:head>
<h:body>

    <h1>Custom converter in JSF 2.0</h1>

    <h:panelGrid columns="2">

        Bookmark URL :
        <h:outputText value="#{user.bookmarkURL}" />

    </h:panelGrid>

</h:body>
```

Binding

- Permet d'avoir accès à toutes les caractéristiques d'un composant dans un Managed Bean, et pas seulement à sa valeur.
- Exemple : un composant qui change de couleur selon la valeur qu'il affiche.


```
<h:form>
<h:outputText binding="#{compteurBinding.component}"
value="#{compteurBinding.value}"/>
<h:commandButton action="#{compteurBinding.augmenter()}"
value="Augmenter"/>
<h:commandButton action="#{compteurBinding.reduire()}"
value="Reduire"/>
</h:form>
```

Binding

```
@Named(value = "compteurBinding")
@SessionScoped
public class CompteurBinding implements Serializable{
   private int value= 0;
   private HtmlOutputText component;

   public int getValue() {return value;}

   public void augmenter() {
        value++;
        if (value <0) {
            component.setStyle("color: red");
        } else {
            component.setStyle("color: black");
        }
    }
  …
    public void setComponent(HtmlOutputText component) {
        this.component = component;
    }

    public HtmlOutputText getComponent() {return component;}
}
```

Evènements

- Utilité : les gestionnaires d'événement ont plus d'information sur l'interface que les « actions » normales.
- Ils sont appelés **avant** les actions.
- Utilisation possible: préparer le terrain pour une action (par exemple: action « éditer » dans une liste; le listener peut déterminer quelle entrée à éditer).
- Autre utilisation: modifier les éléments de ***l'interface!***
- Souvent l'utilisation d'ajax est une alternative.
  - Exemple : http://docs.oracle.com/javaee/7/tutorial/doc/jsf-pagecore002.htm

Binding par Méthode

```
@ManagedBean(name = "country")
@SessionScoped
public class CountryBean implements Serializable {
    private static final long serialVersionUID = 1L;
    private static Map<String, String> countries;
    private String localeCode = "en"; // default value

    static {
        countries = new LinkedHashMap<String, String>();
        countries.put("United Kingdom", "en"); // label, value
        countries.put("French", "fr");
        countries.put("German", "de");
        countries.put("China", "zh_CN");
    }
    public void countryLocaleCodeChanged(ValueChangeEvent e) {
        localeCode = e.getNewValue().toString();
    }
    public Map<String, String> getCountryInMap() {
        return this.countries;
    }
    public String getLocaleCode() {
        return localeCode;
    }
    public void setLocaleCode(String localeCode) {
        this.localeCode = localeCode;
    }
```

Binding par Méthode

```
<h1>JSF 2 valueChangeListener example</h1>

<h:form>

    <h:panelGrid columns="2">

    Selected country :
    <h:inputText id="country" value="#{country.localeCode}" size="20" />

    Select a country {method binding}:
    <h:selectOneMenu value="#{country.localeCode}" onchange="submit()"
            valueChangeListener="#{country.countryLocaleCodeChanged}">
            <f:selectItems value="#{country.countryInMap}" />
        </h:selectOneMenu>

    </h:panelGrid>

</h:form>
```

Utilisation des interface des écouteurs

```
public class CountryValueListener implements ValueChangeListener {

    @Override
    public void processValueChange(ValueChangeEvent event) throws AbortProcessingException {
        // access country bean directly
        CountryBean country =
                (CountryBean) FacesContext.getCurrentInstance().getExternalContext()
                .getSessionMap().get("country");

        country.setLocaleCode(event.getNewValue().toString());

    }

}
```

Utilisation des interface des écouteurs

```
<h:form>

    <h:panelGrid columns="2">

    Selected country :
    <h:inputText id="country" value="#{country.localeCode}" size="20" />

    Select a country {ValueChangeListener class}:
    <h:selectOneMenu value="#{country.localeCode}" onchange="submit()">
            <f:valueChangeListener type="com.mkyong.CountryValueListener" />
            <f:selectItems value="#{country.countryInMap}" />
        </h:selectOneMenu>

    </h:panelGrid>

</h:form>
```

Template

- Système permettant de factoriser la présentation du site.
- Les JSF précisent quelles templates elles utilisent, et quelle partie de la template elles remplacent.

La template

template.xhtml

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">

    <h:head>
    </h:head>

    <h:body>
        <div id="top" class="top">
            <ui:insert name="top">My Nice forum</ui:insert>
        </div>
        <div>
             <div id="content" class="left_content">
                <ui:insert name="content">Content</ui:insert>
            </div>
        </div>
    </h:body>
</html>
```

ui:insert définit une zone dont le contenu peut être remplacé par une facelet

Utilisation de la template

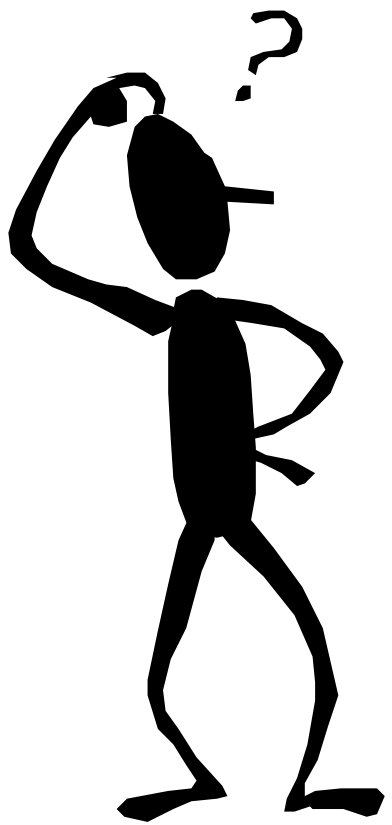
```
<?xml version='1.0' encoding='UTF-8' ?>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:ui="http://java.sun.com/jsf/facelets">

    <ui:composition template="template.xhtml">
        <ui:define name="content">
            <h:form>
                <h:panelGrid columns="2">
                    <h:outputText value="Login:"/>
                    <h:inputText value="#{login.login}" title="Login"/>
                    <h:outputText value="Password:"/>
                    <h:inputSecret value="#{login.password}" title="Password"/>
                </h:panelGrid>
                <h:commandButton type="submit" action="#{login.doLogin()}"
label="login" value="login"/>
            </h:form>
            <h:messages/>
        </ui:define>
    </ui:composition>
</html>
```

utiliser le fichier template.xhtml

dont on redéfinit la partie « content »

Des Questions ??



Démo3

Prise en main de la technologie JSF