JSF 2.0 (*Java Server Faces*) Balises et composants standards

Université de Nice - Sophia Antipolis
Richard Grin
Version 0.5 – 31/10/12

- Ce support fait partie d'un cours sur JSF 2
- Il présente les balises et les composants standards de JSF 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.

Composants standards

- JSF fournit de nombreuses balises et 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)

Classes de base

Racines de l'arborescence

- UIComponent est la classe abstraite racine des composants JSF ; elle comporte de nombreuses méthodes abstraites
- UIComponentBase est une classe fille abstraite qui implémente toutes les méthodes abstraites de UIComponent avec un comportement par défaut ; c'est la classe de base pour les composants standards

UICommand

- UICommand représente un composant qui permet à l'utilisateur de déclencher une action (représentée par la classe ActionEvent)
- Exemple : bouton, choix de menu ou hyperlien
- Classes filles :
  - HtmlCommanButton traduit par un bouton HTML de type submit ou reset
  - HtmlCommanLink traduit par un hyperlien HTML qui agit comme un bouton submit (avec du code Javascript)

ActionEvent

- Les instances de ActionEvent générées par les UICommand sont envoyées (dans cet ordre)
  1. aux ActionListener (dans l'ordre dans lequel ils ont été enregistrés)
  2. à la méthode « actionListener » attachée au composant
  3. à la méthode « action » attachée au composant

ActionSource et ActionSource2

- Interface implémentée par les composants qui peuvent être la source de ActionEvent
- ActionCommand implémente cette interface

- **??**

Attributs des composants

- Peut avoir des attributs, des facets, des paramètres (voir façon de les utiliser…)**??**
- Autres généralités sur les composants JSF**??**

- Il est impossible dans cette introduction de passer en revue toutes les balises standards offertes par Facelets
- De nombreux sites Web en donnent des exemples
- Documentation pour ces balises : http://download.oracle.com/javaee/6/javaserverfaces/2.0/docs/pdldocs/facelets/
- Dans cette section nous donnerons quelques conseils pour débuter

Quelques références

- Le tutoriel Java EE 6 d'Oracle est un bon site pour débuter : http://download.oracle.com/javaee/6/tutorial/doc/bnarf.html
- De nombreux exemples à l'adresse http://www.mkyong.com/tutorials/jsf-2-0-tutorials/ (quelques exemples de ce support sont inspirés de ce site)

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`**

- Le composant peut ainsi être manipulé par le backing bean avec du code Java
- On n'utilise cet attribut que pour des traitements spéciaux, comme, par exemple, lier une dataTable pour pouvoir récupérer la ligne en cours par la méthode Java **`table.getRowData()`**

Librairies de balises

- La librairie d'espace de noms **`http://java.sun.com/jsf/core`** , représentée le plus souvent par l'alias **`f:`** contient les balises essentielles à la construction d'une page JSF : <f:view>, <f:event>, <f:ajax>,…
- La librairie d'espace de noms **`http://java.sun.com/jsf/html`** , représentée le plus souvent par l'alias **`h:`** est la librairie des composants standards
- Le créateur de composants peut choisir son espace de noms

- Nous allons étudier un exemple de chaque type pour en illustrer les particularités
- Reportez-vous à la documentation de JSF ou de la librairie de composants que vous utilisez pour en savoir plus
- Commençons par l'étude de balises générales de la librairie f:

Balises standards

`<f:view>`

- Balise « top level » utilisée comme container pour les composants JSF d'une page (racine de la vue)
- Peut contenir un attribut pour donner la locale (langue et pays) ou pour indiquer des méthodes pour écouter les changements de phase
- Peut contenir un tag **`<f:facet name="javax_faces_metadata">`** pour les métadonnées (ou, ce qui revient au même, le tag **`<f:metadata>`**) ou un tag **`<f:event>`** ; voir la présentation des paramètres de vue dans la section sur PRG

Exemple

```
<f:view locale="fr">
```


`<f:facet>`

- Ce tag peut se trouver dans plusieurs composants parents ; elle définit une partie du composant
- Exemple : le titre des colonnes des tables (<h:dataTable>) est donné par un tag <f:facet> dans un tag <h:column>

```
<h:column>
  <f:facet name="header">Nom</facet>
  #{item.nom}
</column>
```

Sous-balises pour un composant

- f:phaseListener enregistre un PhaseListener pour le composant
- f:actionListener enregistre un ActionListener pour un composant « action » (sous balise du composant)
- f:valueChangeListener enregistre un ChangeListener pour un composant
- f:ajax enregistre un comportement Ajax d'un ou de plusieurs composants

Sous-balises pour un composant

- f:convert, f:convertDateTime, f:convertNumber enregistre un convertisseur nommé, pour les dates, pour les nombres pour le composant
- f:validator, f:validateDoubleRange, f:validateLongRange, f:validateLength, f:validateRegex, f:validateRequired, f:validateBean **??**

Sous-balises pour un composant

- f:facet ajoute une partie du composant
- f:param ajoute un paramètre à un composant « action » (pourra être récupéré par la méthode action comme paramètre de la requête)
- f:attribute ajoute un attribut au composant
- f:setPropertyActionListener

- f:selectItem
- f:selectItems

Autres balises

- f:event
- f:metadata
- f:loadBundle

Formulaires

<h:form>

- Rendu comme un formulaire HTML
- Les composants qui servent à récupérer une information depuis l'utilisateur doivent être dans un formulaire (zones de saisie de texte, listes déroulantes, boîtes à cocher, boutons radio,…) et donc dans une balise <f:form>)

Plusieurs formulaires

- On peut avoir plusieurs formulaires dans une page
- Le cycle de vie concerne alors seulement le formulaire soumis et pas les autres

Textes

Saisie de texte

- Champ texte avec **<h:inputText>** (1 ligne)
- Saisie des mots de passe avec **<h:inputSecret>**
- Zone de texte de plusieurs lignes avec **<h:inputTextarea>**

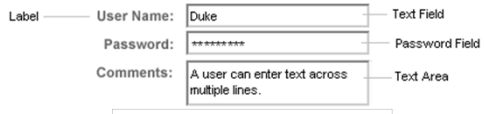

Image tutoriel Java EE 6 d'Oracle

Afficher du texte

- Il suffit d'écrire le texte dans le code de la page
- Il est même possible d'inclure directement du code EL (voir partie du cours) directement dans le code de la page :
  Nom : <h:inputText …/>
  #{msgs.nom} : <h:inputText …/>
- Pour des cas spéciaux il peut être intéressant d'utiliser la balise <h:outputText> qui a, par exemple, l'attribut escape ; si escape="true" (vrai par défaut), les caractères < > et & sont transformés en entités HTML

Exemple

- **<h:inputText id="age" label="age" size="3" value="#{bean.age} " />**
- **label** : utilisé pour les messages d'erreur lors de la validation des valeurs saisies
- **value** : la valeur affichée au moment de l'affichage de la page ; lors de la soumission du formulaire, indique où sera rangée la valeur saisie par l'utilisateur ; **bean** est un backing bean (instance d'une classe Java) qui a une propriété nommée « **age** »

Quelques attributs

- **`readonly="true"`** affiche la valeur mais ne permet pas de la modifier
- **`title`** permet d'indiquer la bulle d'aide qui sera affichée lorsque l'utilisateur laissera sa souris sur la zone de saisie

Images

Image (1/2)

- Une image peut être traitée comme un fichier ordinaire ou comme une ressource ; consultez la gestion des ressources dans la partie 2 de ce cours
- Afficher une image :

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


Image (2/2)

- Bouton ou lien avec image :

```
<h:commandLink>
  Voir les détails
  <h:graphicImage library="images"
    name="im.jpg"/>
</h:commandeLink>
```


Messages

Messages d'information ou d'erreur

- Les messages d'erreur liés à JSF, générés par JSF ou par le code Java sont affichés
  - dans une zone de la page réservée aux messages
  - ou près du composant qui a généré l'erreur (le développeur choisi l'endroit ; voir <h:message>)
- Exemples de messages :
  - indique que la saisie de l'utilisateur a provoqué une erreur de conversion ou de validation
  - message généré par le code Java pour une raison quelconque (erreur ou information)

Messages standards

- JSF génère ses propres messages pour indiquer des erreurs de conversion ou de validation
- La partie 2 de ce support (section « Messages d'erreur ou d'information ») montrera comment modifier ces messages standards ou même comment générer de nouveaux messages par programmation

Les messages affichés dépendent du stade de développement

- Attention, les messages d'erreurs sont toujours affichés dans le stade de développement « Development » mais pas au stade « Production »
- Au stade « Production », ces messages ne sont affichés que si le développeur a inclus des balises message ou messages dans ses pages

Types de messages

- JSF distingue les messages détaillés et les messages résumé (avec moins de détails ; *summary* en anglais)
- Les attributs des balises <h:messages> et <h:message> permettent de modifier le type de message affiché par défaut
- La plupart des messages sont associés à un composant mais des messages peuvent n'être associés à aucun composant (messages globaux)

<h:messages>

- Affiche les messages pour tous les composants et les messages qui ne sont pas liés à un composant particulier (par défaut, n'affiche que les messages résumés)
- De nombreux attributs permettent de définir la mise en page (messages sous la forme de table ou de liste) et d'indiquer si on veut seulement un résumé (seulement affiché par défaut) ou les détails du message, et si on veut tous les messages ou seulement ceux qui ne sont pas liés à un composant (utile si les messages liés aux composants sont déjà affichés par <h:message>

<h:message>

- Affiche un message lié à un composant (par défaut, n'affiche que les messages détaillés)
- Un attribut obligatoire **`for`** donne l'identificateur du composant qui est lié à ce message
- Habituellement placé près du composant dans la page JSF
- De nombreux attributs permettent d'indiquer si on veut seulement un résumé ou les détails (seulement affiché par défaut) du message

Format d'affichage des messages

- Les balises **`<h:message>`** et **`<h:messages>`** ont les attributs **`styleClass`** et **`style`** qui permettent de modifier le format d'affichage avec du CSS (voir section « Gestion des ressources » dans la partie 2 de ce cours pour désigner un fichier CSS) ; il existe aussi d'autres attributs qui définissent un style lié à la sévérité du message (errorStyle, errorClass, infoStyle, infoClass,…)

Exemples

- `<h:messages errorClass="erreur" infoStyle="color:green"/>`
- `<h:messages globalOnly="true"/>`
- `<h:message for="nom" />`
- `<h:message for="nom" showSummary="true" showDetail="false" />`

Mise en page

Mise en page

- **`<h:panelGrid columns="2">`** permet d'aligner sur 2 colonnes les composants internes à la balise
- **`<h:panelGroup>`** permet de regrouper plusieurs composants pour qu'ils ne soient considérés que comme un seul composant par **`<h:panelGrid>`** ; permet aussi de ne rien mettre dans une des « cases » du panelGrid en ne mettant aucun composant à l'intérieur du panelGroup

Listes

Les listes

- Il y a plusieurs composants JSF standards qui permettent de proposer un choix (ou plusieurs) parmi une liste de valeurs
- Tous les composants sont bâtis sur des classes Java communes mais se présentent différemment à l'utilisateur

Listes à choix unique

- Boutons radios avec <h:selectOneRadio>
- Boîte à cocher <h:selectBooleanCheckbox>
- Liste déroulante avec <h:selectOneMenu>
- Liste non déroulante avec <h:selectOneListbox>

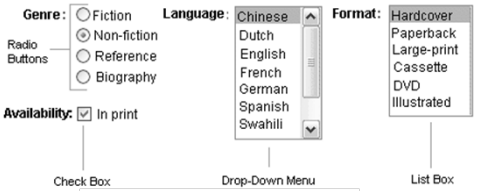

Image tutoriel Java EE 6 d'Oracle

Listes à choix multiple

- Boîtes à cocher avec <h:selectManyCheckbox>
- Liste déroulante avec <h:selectManyMenu>
- Liste non déroulante avec <h:selectManyListbox>

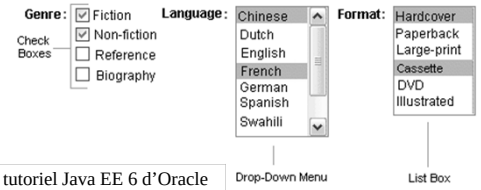

Image tutoriel Java EE 6 d'Oracle

Les listes - Exemple

```
<h:selectOneListbox value="#{bean.choix}">
  <f:selectItem itemLabel="choix"
                itemValue="#{bean.option}" />
  <f:selectItem itemLabel="autre choix"
                itemValue="false" />
  <f:selectItems value="#{bean.options}"/>
</h:selectOneListbox>
```

Ce qui est désigné par **`f:selectItem`** et **`f:selectItems`** sera affiché dans la liste. Le choix fait par l'utilisateur parmi toutes ces valeurs, est rangé dans **`bean.choix`** (attribut **`value`** de **`selectOneListBox`**)

Les entrées de la liste qui correspondent à l'attribut **`value`** sont présélectionnées dans la liste

Classe `SelecItem`

- Paquetage **`javax.faces.model`**
- Donne un label, une valeur et d'autres propriétés d'une liste de GUI
- Utilisée pour la valeur (attribut value) des tags **`<f:selectItem>`** et **`<f:selectItems>`**

`<f:selecItems>`

- Son attribut **`value`** peut désigner un tableau, une collection ou une map
- Le plus souvent, on part d'une liste d'éléments et on la transforme en tableau ou collection de **`SelectItem`**
- Le transparent suivant donne un schéma de code pour une telle transformation (souvent fourni sous la forme d'une méthode **`static`** utilitaire)

Schéma de code pour `<f:selecItems>`

```
public static SelectItem[]
       getSelectItems(List<?> entities) {
  SelectItem[] items = new SelectItem[size];
  for (Object x : entities) {
    items[i++] =
      new SelectItem(x, x.toString());
  }
  return items;
}
```


Autre façon de faire (JSF 2.0)

- Pour éviter la complication de la transformation en **`SelectItem`**, on peut utiliser les attributs **`var`**, **`itemValue`** et **`itemLabel`**
- Exemple :

```
<f:selectItems
  value="#{ecoleController.personnes()}"
  var="personne"
  itemValue="#{personne.id}"
  itemLabel="#{personne.nom}"/>
```

La variable « **`personne`** » contient un élément de la liste désignée par **`value`**

Chausse-trappes avec les listes

- Quelques chausse-trappes que l'on peut rencontrer avec les listes déroulantes sont exposées dans les transparents suivants
- En particulier avec les convertisseurs et les validateurs qui seront étudiés en détails dans la 2ème partie de ce cours sur JSF

Conversion pour les listes déroulantes

- Comment se fait la conversion entre les éléments d'une liste et ce qui est affiché ?
- Par exemple, une liste d'écoles et ce qui est affiché ?
- Il faut bien comprendre que le composant JSF va être transformé en élément(s) HTML et que toutes les valeurs vont être transformées en **`String`** ; en retour, une valeur choisie par l'utilisateur sera une **`String`** qu'il faudra éventuellement convertir en un autre type

Convertisseur

- Si le type T des éléments à choisir n'est pas **`String`** (ça peut, par exemple, être des pays de la classe **`Pays`**), il faut un convertisseur pour passer du type T à **`String`** (pour le passage en HTML) et vis-versa (pour le rangement du choix sous la forme d'une instance du type T)
- Les convertisseurs sont étudiés dans la 3ème partie du cours sur JSF

Désigner un convertisseur

```
<h:selectOneMenu
    value="#{myBean.selectedItem}">
  <f:selectItems
      value="#{myBean.selectItems}" />
  <f:converter converterId="fooConverter" />
</h:selectOneMenu>
```

- **`fooConverter`** désigne un convertisseur qui convertit des **`String`** dans le type de **`myBean.selectedItem`** et vis-versa

Convertisseur par défaut d'un type

- On verra dans le cours sur les convertisseurs qu'il est possible d'écrire un convertisseur par défaut pour un type Java :
  `@FacesConverter(forClass=fr.unice.Pays.class)`
- En ce cas, la balise <f:converter> est inutile quand le type de la propriété désignée par l'attribut value correspond à ce type Java

Entrée « fictive » de la liste

- **`<f:selectItem noSelectionOption="true" itemLabel="Choisissez une valeur" itemValue="0"/>`**
  Sera considéré par JSF comme un « non-choix » : pas de validation, et erreur si le choix est requis
- **`<f:selectItems value="`**#{**`bean.choix}`**" **`noSelectionValue=`**"#{**`bean.choix(0)}`**" **`var="choix" itemValue=`**"#{**`choix.id}`**" **`... />`**
  Un choix ayant cette valeur sera considéré comme un non-choix (JSF 2.0)

Entrée « fictive » et conversion

- On aura un message d'erreur si on veut mettre une 1ère entrée du type « Choisissez un pays » car cette première entrée n'est pas du type T, même avec l'attribut noSelectionOption à true (voir transparent suivant), si on ne traite pas ce cas particulier dans le convertisseur

Validation d'un choix dans une liste

- JSF vérifie que le choix qui est envoyé au serveur est bien un des choix proposés à l'utilisateur (la comparaison utilise la méthode equals)
- Ça implique que les choix proposés doivent être connus par JSF au moment de la validation
- Si les choix dépendent d'une propriété d'un backing bean (par exemple, un pays pour une liste de villes d'un certain pays), il faut donner la bonne portée au bean pour que les choix puissent être calculés correctement au moment de la validation (View ou Session par exemple)

Type Java pour un choix multiple

- L'attribut value pour les listes à choix multiple peut désigner un tableau ou une collection Java
- Dans le cas d'une collection, le type des objets de la collection étant perdu lors de l'exécution (*type erasure* de la généricité), le type de la collection passée par JSF au *setter* de la propriété est déterminé par un algorithme donné par la spécification JSF 2.0 (utilise un éventuel attribut **`collectionType`** des tags selectMany et le type de la valeur désignée par l'attribut **`value`** ; **`ArrayList`** en dernier recours)

Situation fréquente

- Le choix d'une entrée de la liste provoque des modifications dans l'interface utilisateur
- Par exemple, le choix d'un service de l'entreprise provoque l'affichage des informations sur les employés du service choisi
- Il ne faut pas que l'utilisateur soit obligé de cliquer sur un bouton de soumission de formulaire pour que la modification de l'interface intervienne

Situation fréquente

- Pour soumettre directement un choix dans une liste sans avoir à cliquer sur un bouton, il faut ajouter à la liste l'attribut **`onchange=submit()`** qui ajoute du code Javascript à la page HTML générée
- Dans le cas où il faut court-circuiter des validations d'autres composants, il faut aussi ajouter à la liste l'attribut **`immediate=true`** et ajouter **`context.renderResponse()`** (de **`FacesContext`**) dans une méthode valueChangeListener ; on peut aussi limiter le réaffichage des composants par Ajax

Boutons

Traitement d'un formulaire

- La balise <h:form> est traduite par la balise HTML <form>
- Une différence importante est que <h:form> ne contient pas d'attribut action ; l'action HTML est toujours l'URL de la page qui contient le formulaire
- La page qui sera affichée à la suite de la soumission du formulaire est définie par l'action associée au composant de soumission du formulaire (<h:commandButton> et <h:commandLink>)

Soumission d'un formulaire

- La soumission d'un formulaire envoie au serveur les informations contenues dans le formulaire (valeurs saisies ou choisies par l'utilisateur)
- Le plus simple est d'ajouter un bouton ou un lien de soumission dans le formulaire (<h:commandButton> ou <h:commandLink>), ce qui envoie une requête POST au serveur
- Depuis JSF 2.0 il est possible de soumettre un formulaire avec une requête GET avec les balises <h:button> et <h:link>

<h:commandButton>

- Traduit par la balise HTML <button>
- Plusieurs types indiqués par l'attribut type : submit, reset, button ; ils correspondent aux types HTML
- Le type par défaut est submit
- Le type reset permet de réinitialiser le formulaire
- Le type button ne soumet pas le formulaire ; le plus souvent on attache du code JavaScript au bouton (attribut onclick)

<h:commandButton> de type submit

- Pour soumettre un formulaire avec une requête POST
- L'action est donnée par l'attribut action
- La valeur de l'attribut action est soit une valeur fixe qui définit la prochaine page à afficher, soit, le plus souvent, une expression EL qui désigne une méthode qui est exécutée et dont la valeur retour définit la prochaine page à afficher
- La valeur fixe peut être un identificateur de page ou une valeur logique d'une règle de navigation

Exemples

```
<h:form>
  <h:commandButton action="page1"
                   value="Cliquer ici"/>
</h:form>
```

```
<h:form>
  <h:commandButton action="#{bean.page1}"
                   value="Cliquer ici"/>
</h:form>
```


<h:button>

- Pour soumettre un formulaire avec une requête GET
- La prochaine page à afficher est donnée par l'attribut outcome (adresse péemptive)
- Au contraire de h:commandButton, aucune action n'est exécutée ; le seul moyen de faire exécuter du code est l'attribut actionListener

Exemple

```
<h:button value="Ajouter de l'argent"
    outcome="ajout?idCompte=#{item.id}"/>
```


Liens

Les liens

- 3 balises JSF pour naviguer vers une autre page traduites par un lien HTML :
  - **`h:commandLink`** : soumet un formulaire (requête POST) et lance une action
  - **`h:link`** : soumet un formulaire (requête GET)
  - **`h:outputLink`** : traduite par une simple balise HTML <a href=> ; pas nécessairement dans un formulaire ; ne soumet pas de formulaire

Sous-balises

- Les 3 types de liens acceptent ces sous-balises :
  - <f:param> pour passer des paramètres qui seront ajoutés à la requête HTTP
  - <h:graphicImage> pour ajouter une image au lien

Page de navigation

- La page qui sera affichée après la page actuelle est déterminée par l'algorithme de navigation de JSF (voir partie 2 de ce support pour plus de précision) et par la valeur indiquée par la balise :
  - s'il existe une règle de navigation qui correspond à cette valeur, la règle est appliquée pour déterminer l'URL de destination
  - sinon une navigation « implicite » est effectuée ; la valeur est interprétée comme un URL

<h:commandLink>

- Nécessairement dans un formulaire
- C'est la valeur retour de la méthode définie dans l'attribut action qui déterminera la nouvelle page à afficher après la soumission du formulaire
- Traduction HTML
  - utilise JavaScript pour soumettre le formulaire
  - href="#", donc le navigateur croit que la navigation restera dans la même page (le bon URL ne sera pas affiché)

Exemple

- `<h:commandLink action="#{user.goLoginPage}" value="Login page" />`

est traduit par :

```
<script type="text/javascript"
src="/JavaServerFaces/faces/javax.faces.res
ource/jsf.js?ln=javax.faces&amp;stage=Devel
opment"> </script>
<a href="#" onclick="mojarra.jsfcljs(
document.getElementById('j_idt6'),
{'j_idt6:j_idt18':'j_idt6:j_idt18'},'');
return false">Login page</a>
```

Cette méthode soumet le formulaire

<h:link>

- `<h:link value="Lancer le traitement" outcome="adresse"/>`

  génère ce code HTML : `<a:href=/contextApplication/adresse.xhtml>` et donc envoie une requête GET
- La page pointée est déterminée au moment du rendu de la page qui contient le tag (pas ensuite quand l'utilisateur clique sur le bouton ou le lien)
- La valeur de outcome peut être un identificateur de page ou une valeur logique d'une règle de navigation

Exemple

-
```
<h:link value="Login page" outcome="login">
    <f:param name="username" value="toto"/>
</h:link>
```

est traduit en HTML par

```
<a
href="/appl/faces/login.xhtml?username=toto
">Login page</a>
```


<h:ouputLink>

- Permet de créer un lien avec une page qui n'est pas dans l'application
- Attention, si l'adresse de la page qui contient le lien ne contient pas le pattern pour les pages JSF (par exemple si la page est une page « Welcome » de l'application) et si le lien pointe vers une page JSF, il ne faut pas oublier de donner une adresse qui correspond au pattern des pages JSF

Exemple

-
```
<h:outputLink
  value="www.meubles.com">
  <h:graphicImage library="images"
                  name="sofa.png" />
</h:outputLink>
```

est traduit par

```
<a href="www.meubles.com">
  <img
src="/appl/faces/javax.faces.resource/sofa.
png?ln=images" />
</a>
```


Tables

<h:dataTable>

- Très utilisé pour représenter les données d'une base de données : chaque ligne de la table correspond à une ligne d'une table
- Génère une table HTML

Modèle de données

- Les données affichées sont indiquées par l'attribut **value**
- Elles sont représentées par un model : **javax.faces.model.DataModel**
- Souvent on n'utilise pas directement le modèle ; on fournit un tableau, une liste ou un **ResultSet** à l'attribut value de **<h:dataTable>** ; en interne une classe fille de **DataModel** enveloppera les données fournies, par exemple avec la classe **ListDataModel**

Colonnes

- La balise <h:dataTable> contient des balises <h:column> qui décrivent le contenu des colonnes
- La balise <h:column> a l'attribut value qui donne le contenu de la colonne

Attributs des colonnes

- La balise <h:column> peut contenir un facet de nom « header » qui décrit l'en-tête de la colonne ; « footer » pour le pied de colonne
- Elle peut aussi avoir des attributs pour la mise en forme de l'en-tête ou du pied de colonne en plus des attributs de base (id, styleClass, rendered, binding)

Exemple

```
<h:datatable value="#{clientBean.all}"
             var="client">
  <h:column>
    <f:facet name="header">Nom</f:facet>
    #{client.nom}
  </h:column>
  ...
</datatable>
```


Quelques mises en forme

- Titre d'une colonne sur plusieurs lignes (« < » est interdit dans l'attribut value) :

```
<f:facet name="header">
  <h:outputText escape="false"
      value="Détails &lt;br/>opérations"/>
</f:facet>
```

Quelques mises en forme

- Mise en forme des valeurs des colonnes : attribut **`columnClasses`** de **`<h:dataTable>`** avec des classes CSS pour mettre en forme les différentes colonnes
- Par exemple pour cadrer à droite la 2ème colonne (pas de format spécial pour colonnes suivantes) : **`columnClasses="texte,nombre"`**
- Dans un fichier CSS on aura la définition des classes CSS **`texte`** et **`nombre`** ; par exemple **`.nombre { text-align: right; }`**

`DataModel`

- Quelquefois il faut expliciter l'utilisation de **`DataModel`** dans le backing bean de la page pour avoir plus de facilités de manipulation des données
- Par exemple, on écrit une classe **`Donnees`** qui crée une instance de **`ListDataModel`** à partir d'une liste
- **`DataModel`** contient des méthodes pour récupérer les numéros des lignes, la valeur de la ligne « en cours » (un **`Object`**), gérer les écouteurs de la table

Récupérer le dataModel

- Au lieu de créer explicitement un **`DataModel`** dans son code, on peut aussi lier (par l'attribut **`binding`** de la balise **`<h:dataTable>`** la dataTable à une variable (appelons-la **`table`**) de type **`javax.faces.component.html.HtmlDataTable`** d'un backing bean et récupérer ensuite le dataModel par **`table.getDataModel()`**
- On peut aussi avoir des informations sur les données directement, sans passer explicitement par le modèle : **`table.getRowData()`** pour avoir le contenu de la ligne en cours

Exemple

- Si on veut récupérer la ligne en cours dans une table, on peut écrire le code suivant dans une méthode « action » :
  `modele.getRowdata()`
  renvoie le contenu de la ligne en cours (celle où l'utilisateur vient de cliquer par exemple) ; voir aussi le bas du transparent précédent
- Si les données de la table sont données par une **`List<Client>`**, le type de ce qui est renvoyé est bien **`Client`**

Compléments sur dataTable

- La balise a des attributs qui permettent d'indiquer
  - Divers mises en forme (couleur de fond, largeur des traits de séparation ou qui entourent la table, espace entre ou à l'intérieur des cellules, styles ou classes CSS divers)
  - Le numéro de la 1ère ligne qui sera affichée (0 par défaut)
  - Le nombre de lignes affichées

Exemple

- Pour bien séparer les lignes par des couleurs différentes :

```
<h:dataTable …
  columnClasses="rouge,vert"/>
```

- Classes CSS :

```
.rouge { background: #b0c4de;}
.vert { background: #00ff00;}
```

<ui:repeat>

- Permet de répéter un affichage sur une liste d'objets
- Exemple :

```
<ul>
  <ui:repeat value="#{bean.clients}"
             var="item">
     <li>#{item.nom}</li>
  </ui:repeat>
</ul>
```