JSF 2.0 (Java Server Faces) Partie 2

Université de Nice - Sophia Antipolis
Richard Grin
Version 0.9 – 21/11/12

- ❑ *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)*

- ❑ Ce support est divisé en 3 parties
- ❑ La première partie a présenté les fondamentaux de JSF
- ❑ Cette deuxième partie présente d'autres notions de base de JSF liées à l'écriture de pages JSF
- ❑ La dernière partie abordera la programmation Java en détails et des fonctionnalités plus avancées de JSF

Plan (1/2)

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

Plan (2/2)

- ❑ Quelques situations typiques
- ❑ Quelques erreurs fréquentes

Navigation

Définition

- ❑ Les applications Web contiennent le plus souvent plusieurs pages Web
- ❑ Selon les actions de l'utilisateur et l'état de l'application, une navigation est effectuée : une nouvelle page est affichée à la place de la page courante
- ❑ Une application Web simple peut aussi ne contenir qu'une seule page rafraîchie par des appels Ajax ; ce cas ne sera pas traité dans cette section

Navigation dans JSF (1/2)

- ❑ Des composants JSF standards provoquent la soumission d'un formulaire et ensuite une navigation : `<h:commandButton>`, `<h:button>`, `<h:commandLink>`, `<h:link>`
- ❑ Ils doivent tous être dans une balise `<h:form>`
- ❑ Le composant `<h:outputLink>` correspond à un simple lien `<a href>` HTML et n'a pas besoin d'être dans un formulaire

Navigation dans JSF (2/2)

- ❑ Une navigation peut aussi être initiée par un autre composant (par exemple une liste) inclus dans un formulaire si ce composant lance un code JavaScript qui contient un ordre de soumission du formulaire
- ❑ Du code Java peut aussi provoquer une navigation

Exemple

- ❑ Dans les pages JSF on indique une valeur (qui désigne un identificateur de page ou une valeur logique d'une règle de navigation) ou une méthode Java
- ❑ Avec une valeur :

```
<h:commandButton value="Texte du bouton"
  action="page" />
```
- ❑ Avec une méthode (qui retourne une valeur) :

```
<h:commandButton value="Texte du bouton"
  action="#{nomBean.nomMethode}" />
```

Quelle sera la prochaine page ?

- ❑ La valeur de l'attribut `action` ou retournée par la méthode désignée par l'attribut `action` est une valeur logique, appelée « résultat » (*outcome*) qui correspond à la prochaine vue (page dans le jargon JSF) à afficher
- ❑ Un algorithme détermine la vue à afficher à partir de ce résultat

Soumission formulaire par POST

- ❑ La navigation est le plus souvent provoquée par les balises `<h:commandButton>` ou `<h:commandLink>` utilisées pour soumettre un formulaire : les données sont envoyées au serveur par une requête POST
- ❑ La page qui est affichée à la suite de la soumission du formulaire est définie par l'attribut `action` du composant de soumission du formulaire (`h:commandButton` ou `h:commandLink`)

Soumission formulaire par GET

- La navigation peut aussi venir des composants `<h:button>` ou `<h:link>` qui soumettent le formulaire ; les données du formulaire sont envoyées au serveur par une requête GET
- La page qui est ensuite affichée est définie par l'attribut `outcome` des balises JSF

R. Grin

JSF

page 13

`<h:outputLink>`

- La navigation peut aussi venir de la balise `<h:outputLink>` qui ne soumet pas de formulaire et qui correspond à une simple balise `<a href=>`
- La page qui est ensuite affichée est définie par l'attribut `value` de la balise JSF ; l'URL de destination peut désigner une adresse en dehors de l'application, par exemple `http://google.com` ; pour les URL relatifs, il faut préfixer par `faces/` si on désigne une page JSF de l'application
- Exemple :

```
<h:outputLink value="faces/page2.xhtml" >
Page 2 </h:outputLink>
```

R. Grin

JSF

page 14

Soumettre un formulaire

- Le plus simple est d'utiliser un `<h:commandButton>` ou un `<h:commandLink>`
- Il est aussi possible d'utiliser un autre composant de type `inputText` ou liste (tout composant qui peut contenir une valeur) avec du code Javascript :

```
<h:inputText ...
onchange="this.form.submit();" >
```

R. Grin

JSF

page 15

Traitement d'un formulaire (1/2)

- La soumission génère une requête POST qui contient les valeurs saisies par l'utilisateur ; ces informations sont traitées par le serveur
- Par défaut, la page qui contient le formulaire est affichée à nouveau après le traitement par le serveur (*postback*)

R. Grin

JSF

page 16

Traitement d'un formulaire (2/2)

- Le traitement par le serveur peut aussi contenir un *forward* ou une redirection, ce qui provoque une navigation vers une autre page
- *Forward* : la réponse de la requête POST contient le code d'une autre page de l'application, sans que le navigateur n'intervienne ; cette page est affichée à l'utilisateur
- *Redirection* : le serveur envoie un ordre de redirection vers une page au navigateur ; celui-ci demande la page au serveur par une requête HTTP GET ; la page peut être extérieure à l'application

R. Grin

JSF

page 17

Attributs pour la navigation dans JSF

- Dans une page JSF, la balise `<h:form>` n'a pas d'attribut `action` comme son homologue HTML `<form>`
- La prochaine page est déterminée par l'attribut `action` de `<h:commandButton>` ou `<h:commandLink>`, l'attribut `outcome` de `<h:button>` ou `<h:link>`, ou l'attribut `value` de `<h:outputLink>`

R. Grin

JSF

page 18

Exemple

```
<h:form>
...
<h:commandButton
value="#{messages.commanderLabel}"
action="#{commandeBean.checkout}" />
</h:form>
```

Handler de navigation

- ❑ La navigation entre les pages indique quelle page est affichée quand l'utilisateur clique sur un bouton pour soumettre un formulaire ou clique sur un lien
- ❑ Cette navigation est déléguée au *handler* de navigation
- ❑ Ce *handler* peut utiliser des règles de navigation qui permettent de ne pas écrire « en dur » l'identifiant de la prochaine page à afficher, ce qui apporte plus de souplesse

2 grands types de navigation

- ❑ Navigations non préemptive : la prochaine page à afficher est déterminée à la fin de la phase « *Invoke Application* », en réponse à une requête POST
- ❑ Navigation préemptive : la prochaine page à afficher est déterminée dans la phase « *Render Response* » de la page qui contient le lien (lien qui génère une requête GET) ; l'adresse peut être une expression EL de méthode mais cette méthode sera exécutée quand la page sera affichée et pas quand l'utilisateur cliquera sur le composant

Exemples

- ❑ Navigation non préemptive :

```
<h:commandButton value="Supprimer"
action="#{bean.supprimer(item)}/>
```
- ❑ Navigation préemptive :

```
<h:button value="Ajouter un compte"
outcome="suppression"/>
<h:button value="Ajouter un compte"
outcome="#{bean.intituleAjout}"/>
```

Navigation déclarative ou implicite

- ❑ La navigation peut être
 - « déclarative » (ou définie par une règle ; *rule-based*) : définie par une valeur qui correspond à une condition d'une règle de navigation dans le fichier de configuration faces-config.xml
 - « implicite » : (depuis JSF 2.0) définie par l'identificateur de la future page, éventuellement sans l'extension .xhtml ou autre
- ❑ La navigation déclarative l'emporte : JSF regarde d'abord s'il y a une règle

Exemple de navigation déclarative

- ❑ Règle de navigation dans faces-config.xml :

```
<navigation-rule>
<from-view-id>page1.xhtml</from-view-id>
<navigation-case>
<from-outcome>page2</from-outcome>
<to-view-id>/page2.xhtml</to-view-id>
</navigation-case>
</navigation-rule>
```

- ❑ Dans la page JSF page1.xhtml :

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

Exemple de navigation implicite

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

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

Navigation statique et dynamique

- La valeur qui détermine la nouvelle page après une requête POST (navigation non préemptive) peut être
 - définie « en dur » au moment de l'écriture de l'application ; la navigation est statique
 - définie par la valeur retournée par une méthode « action » Java ; la navigation est dynamique
- La navigation préemptive est toujours statique

Exemple de navigation dynamique

- Dans la page JSF page1.xhtml :

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

- Dans un backing bean :

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

Conditions pour une méthode « action »

- Une méthode Java « action » est une méthode d'une classe Java désignée par une expression EL en valeur de l'attribut action ; par exemple la méthode allerPage2 du transparent précédent
- Elle doit remplir les conditions suivantes :
 - elle doit être public
 - elle doit retourner une instance d'objet dont la valeur de retour de la méthode toString() sera utilisée pour la navigation
 - (depuis JSF 2.0, elle peut avoir des paramètres)

Code HTML généré pour <h:commandButton>

- `<h:commandButton action="details" value="Voir détails"/>`
- génère ce genre de code HTML :

```
<input type="submit" name="j_idt8:j_idt9" value="Voir détails" />
```

Code HTML généré pour <h:commandLink>

- `<h:commandLink action="details" value="Voir détails"/>`
- génère ce genre de code HTML :

```
<a href="#" onclick="mojarra.jsfcljs(document.getElementById('j_idt13'), {'j_idt13:j_idt24':'j_idt13:j_idt24'}, ''); return false;">Voir détails</a>
```

Code HTML généré pour les composants « GET »

- Un lien `h:link` de type
`<h:link outcome="List" value="..." />`
- génère ce genre de code HTML :
`<a href="/appli/faces/List.xhtml">...</a>`
- Un lien `h:button` de type
`<h:button outcome="List" value="..." />`
- génère ce genre de code HTML :
`<input type="button"
onclick="window.location.href='/appli/faces/L
ist.xhtml'; return false;" value="..." />`

R. Grin

JSF

page 31

Navigation préemptive (1/2)

- Introduite par JSF 2.0 et attachée aux requêtes GET, c'est-à-dire aux composants `<h:link>` ou `<h:button>`
- Ces composants ont un attribut `outcome` qui désigne la page à afficher s'ils sont activés par l'utilisateur
- Cet attribut `outcome` contient une expression EL évaluée comme une `String` (pas comme une méthode) ; cette valeur désigne la future page de façon déclarative ou implicite

R. Grin

JSF

page 32

Navigation préemptive (2/2)

- La navigation est évaluée pendant la phase « Render response », dès l'affichage de la page, et pas juste après la phase « Invoke application » comme c'est le cas pour une requête POST
- Comme l'évaluation des liens se fait dès l'affichage de la page qui contient les liens, le navigateur connaît l'adresse de la future page si l'utilisateur clique sur le lien, et pourra donc afficher le bon URL pour cette page et pourra la conserver dans un marque-page (*bookmark*)

R. Grin

JSF

page 33

Algorithme qui détermine la vue à afficher

- S'il y a une règle dans le fichier `faces-config.xml`, cette règle sert à déterminer la page
- sinon le résultat désigne la page par l'algorithme suivant :
 - si le résultat n'a pas d'extension, l'extension de la page courante est ajouté au résultat (le plus souvent `.xhtml`)
 - si le résultat ne commence pas par `/`, préfixer par le chemin de la vue courante

R. Grin

JSF

page 34

Portée « View »

- Remarque importante (voir section à venir sur les beans) :
- Si le résultat renvoyé par la méthode est `null`, la vue en cours est réaffichée et la portée View est préservée
- Toute autre valeur vide la portée View, y compris une valeur qui désignerait la vue en cours

R. Grin

JSF

page 35

Règle dans le fichier de configuration

- Une règle contient la page de départ (`<from-view-id>`), des valeurs logiques (`<from-outcome>`) qui déterminent la page qui sera affichée (`<to-view-id>`)

R. Grin

JSF

page 36

Exemple (1/2)

```
<navigation-rule>
  <from-view-id>/index.xhtml</from-view-id>
  <navigation-case> Résultat
    <from-outcome>succes</from-outcome>
    <to-view-id>/succes.xhtml</to-view-id>
  <navigation-case> Vue à afficher
    <from-outcome>echec</from-outcome>
    <to-view-id>/echec.xhtml</to-view-id>
  <navigation-case>
  </navigation-rule>
```

R. Grin

JSF

page 37

Exemple (2/2)

```
<managed-bean>
  <managed-bean-name>nomBean</managed-bean-name>
  <managed-bean-class>p1.p2.C</managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
</managed-bean >
```

R. Grin

JSF

page 38

Compléments – joker

- S'il n'y a pas d'élément `<from-view-id>`, la règle de navigation s'applique à toutes les pages
- On peut mettre un joker à la fin de la valeur de `<from-view-id>` :
`<from-view-id>/truc/*</from-view-id>`
- On peut ajouter `<redirect/>` à la fin de la règle de navigation pour que l'URL de la page soit modifié (redirection à la place d'un *forward*) ; intéressant si l'utilisateur peut avoir à enregistrer un signet (*bookmark*)

R. Grin

JSF

page 39

Compléments – `<from-action>`

- Plusieurs méthodes « action » peuvent renvoyer la même valeur
- Si on veut distinguer la navigation suivant la méthode « action » qui a été exécutée, on peut ajouter un élément `<from-action>` qui désigne la méthode qui a généré la valeur logique de navigation

R. Grin

JSF

page 40

Exemple

```
public class Bean {
  public String action1(){
    return "ok";
  }
  public String action2(){
    return "ok";
  }
} Backing bean

<navigation-case>
  <from-action>#{bean.action1}</from-action>
  <from-outcome>ok</from-outcome>
  <to-view-id>page1.xhtml</to-view-id>
</navigation-case> faces-config.xml
```

R. Grin

JSF

page 41

Compléments – Navigation conditionnelle

- On peut ajouter une condition (avec le langage EL) à un cas de navigation
- La condition doit être remplie pour que le cas de navigation soit pris en compte

□ Exemple :

```
<navigation-case>
  <from-outcome>paiement</from-outcome>
  <if>#{bean.valide}</if>
  <to-view-id>payment.xhtml</to-view-id>
</navigation-case>
```

R. Grin

JSF

page 42

Redirection

- ❑ Les requêtes POST provoquent un forward du serveur vers la page de destination ; le navigateur ne sait pas quelle page va être affichée
- ❑ L'affichage direct d'une page en retour de soumission de formulaire peut aussi provoquer des soumissions multiples d'un formulaire, par exemple si l'utilisateur fait un refresh de la page affichée après le formulaire
- ❑ Pour éviter ce genre de problème il est possible d'ajouter une redirection après la soumission d'un formulaire

R. Grin

JSF

page 43

Exemple

```
<h:form>
  ...
  <h:commandButton value="..."
    action="page2?faces-redirect=true" />
</form>
```

R. Grin

JSF

page 44

Langage EL (Expression Language)

R. Grin

JSF

page 45

Utilité

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

R. Grin

JSF

page 46

Evaluation d'une expression EL

- ❑ La partie à gauche du premier « . » ou du premier « [» est la base de l'expression
- ❑ Ce qui suit est appelée la propriété et peut comporter plusieurs « . » ou « [»
- ❑ Une expression EL peut n'être composée que d'une base comme dans #{valeur}
- ❑ L'évaluation d'une expression EL commence par l'évaluation de la base, appelé aussi terme initial

R. Grin

JSF

page 47

Base, terme initial

- ❑ La base est d'abord comparée aux objets prédéfinis dans EL (voir transparents suivants)
- ❑ Si ça n'est pas un objet prédéfini, la base est cherchée dans les portées requêtes, puis vue, puis session et enfin l'application
- ❑ Si la recherche est infructueuse, JSF essaie de construire un bean géré ou un « resource bundle » (pour internationalisation)

R. Grin

JSF

page 48

Exemples

- ❑ `#{utilisateur.nom}` : si un bean géré a le nom « utilisateur », la propriété `nom` de ce bean est utilisée (par un setter ou un getter suivant le contexte de l'expression EL) ; si le bean existe déjà, il est utilisé, sinon il est créé dans sa portée
- ❑ `#{bundle.formation.message}` : dans ce cas, `bundle` correspond à un resource bundle et le message est récupéré
- ❑ `#{valeur}` : ça n'est pas un objet prédéfini ; si l'expression est la valeur de l'attribut `binding` d'un composant, celui-ci est attaché à la requête

R. Grin

JSF

page 49

Objet prédéfinis (1/2)

- ❑ `requestScope`, `viewScope`, `sessionScope`, `applicationScope` représentent les maps pour les différentes portées
- ❑ `facesContext` pour le FacesContext
- ❑ `initParam` pour les paramètres d'initialisation de l'application
- ❑ `flash` pour la mémoire flash
- ❑ `cookie` pour la map des cookies
- ❑ `cc` pour les composants composés

R. Grin

JSF

page 50

Objet prédéfinis (2/2)

- ❑ `component` pour le composant courant
- ❑ `view` pour la UIViewRoot
- ❑ `session` pour la session HTTP
- ❑ `param` pour la map des paramètres de la requête
- ❑ `header` pour la map des headers de la requête (seule la première valeur est donnée)
- ❑ `headerValues` la map des headers de la requête (toutes les valeurs sont données)

R. Grin

JSF

page 51

Exemple d'utilisation d'un objet prédéfini

- ❑ Le style CSS est « erreur » si le composant est invalide (contient une mauvaise valeur) :

```
<h:inputText value="#{bean.value}"
styleClass="#{component.valid ? '' :
'erreur'}" />
```

- ❑ Utiliser le contexte de l'application pour construire un URL :

```
<h:outputLink
value="#{facesContext.externalContext.requestContextPath}/faces/p1.xhtml">Page 1
</h:outputLink>
```

R. Grin

JSF

page 52

Syntaxe du langage

- ❑ Evaluation d'une propriété de type « a.b »
- ❑ a peut être
 - un tableau
 - une liste
 - une map
 - un objet quelconque
- ❑ Syntaxe spéciale pour les 3 premiers types

R. Grin

JSF

page 53

a.b pour un objet « ordinaire »

- ❑ Si a est un objet qui n'est ni un tableau, ni une liste, ni une map, a.b désigne une propriété b de a
- ❑ a.b peut désigner le getter `a.getB` ou le setter `a.setB` selon les cas
- ❑ `<h:inputText value="#{bean.nom}"/ >`
 - la valeur affichée au départ sera celle retournée par `getNom`
 - la valeur tapée par l'utilisateur sera rangée dans le modèle par la méthode `setNom`

R. Grin

JSF

page 54

Désigner une propriété

- `a.b`, `a["b"]` et `a['b']` sont équivalents ; les " peuvent contenir des ' et réciproquement :
`value="#{a['b']}"`
- `a.b` est plus intuitif pour un objet ordinaire
- `a[ ]` est plus intuitif pour un tableau, une liste ou une map
- `a[ ]` permet de calculer dynamiquement une propriété : `a[b.nomProp]`

R. Grin

JSF

page 55

Syntaxe spéciale pour map et liste

- Si `a` est une map, `a["cle"]` ou `a.cle` désigne la valeur de la map de clé `cle`
- Soit `a.get("cle")`, soit `a.put("cle", valeur)` sera exécuté
- Si `a` est une liste, `a[i]` ou `a.i` désigne l'élément numéro `i` (commence à 0) ; `i` doit être un nombre ou une `string` qui peut être traduite en nombre (par exemple "12")

R. Grin

JSF

page 56

Opérateurs

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

R. Grin

JSF

page 57

Exemples

- `<h:datatable ... rendered="#{(not empty liste) and (dept != 0)}"`
- `#{not empty bean.prop}`

R. Grin

JSF

page 58

Concaténation

- On peut concaténer en plaçant des expressions côte à côte :
`value="#{bundle.cliquer}, #{utilisateur.nom}"`

R. Grin

JSF

page 59

Expression de méthode

- Une expression EL peut désigner une méthode
- La méthode ne peut être surchargée dans le bean
- Exemple :
`<h:commandButton action="#{user.verifMdp}"/>`
- On peut aussi passer des paramètres à une méthode :
`<h:commandButton action="#{a.deplace(-2)}"/>`

R. Grin

JSF

page 60

Backing bean

- ❑ Dans la section « Architecture des applications JSF » on a vu que les pages JSF sont épaulées par des backing beans qui apportent la puissance du langage Java pour coder des interactions sophistiquées avec l'utilisateur
- ❑ Les possibilités sont infinies
- ❑ Dans cette section nous allons voir quelques exemples caractéristiques de l'utilisation des backing beans (appelés aussi beans gérés/managed bean car gérés par le container)

Pas de code dans les pages JSF

- ❑ Pour ceux qui ont travaillé avec JSP, il faut oublier la mauvaise habitude d'inclure de la programmation dans les pages JSF
- ❑ Tout le code (non « métier ») doit maintenant se trouver dans les backing beans (le code métier doit se trouver dans les EJB ou classes Java liées)

Backing bean dans les pages JSF

- ❑ Le code des pages JSF peuvent désigner des backing bean avec la langage EL
- ❑ Il est ainsi possible de désigner une propriété ou une méthode d'un backing bean
- ❑ Une propriété :

```
<h:inputText value="#{utilisateur.nom}" />
```
- ❑ Une méthode :

```
<h:commandButton action="#{bean.lancer}" value="Lancer les opérations" />
```

Utilité des backing bean

- ❑ Ils servent essentiellement à
 - réagir aux actions de l'utilisateur (méthodes « action » ou méthodes « écouteur »)
 - fournir l'état (modèle de données) utilisé par les pages JSF (attributs `value` des composants qui désignent des propriétés du bean)
- ❑ Moins fréquemment il peut servir à conserver des composants de la page JSF (attribut `binding` des composants)

2 types de backing bean

- ❑ On peut utiliser 2 types de beans : ceux qui sont gérés par JSF et ceux qui sont gérés par CDI
- ❑ CDI a été introduit par Java EE 6 et les beans gérés par CDI tendent à remplacer les beans gérés directement par JSF car CDI offre plus de fonctionnalités
- ❑ Dans certains cas (pour la portée « view » par exemple), il est cependant encore plus simple d'utiliser les beans gérés par JSF

@ManagedBean

- ❑ Paquetage `javax.faces.bean`
- ❑ Annote une classe Java d'un backing bean géré par JSF
- ❑ Le bean pourra être utilisé par le langage EL en le désignant par son nom
- ❑ Le nom d'un bean est par défaut le nom de sa classe, avec la première lettre en minuscule

Exemples

- ❑ Le bean suivant pourra être désigné par `employeBean` :

```
@ManagedBean
public class EmployeBean { ... }
```

- ❑ Celui-ci pourra être désigné par `employe` :

```
@ManagedBean(name="employe")
public class EmployeBean { ... }
```

eager

- ❑ Si un bean géré a la portée « application » et l'attribut `eager=true`, le bean est créé par le container dès le démarrage de l'application et pas quand il est désigné pour la première fois
- ❑ Pratique pour initialiser l'application dans une méthode `@PostConstruct` du bean
- ❑ Exemple :

```
@ManagedBean(eager=true)
@ApplicationScoped
public class GlobalBean { ... }
```

Quelles classes pour les beans ?

- ❑ N'importe quelle classe avec un constructeur qui ne prend pas de paramètres
- ❑ La classe doit respecter les conventions de nommage des Java beans pour les propriétés : une propriété `nom` correspond aux méthodes publique `getNom` et `setNom`
- ❑ Si la propriété `prop` est de type booléen, le getter peut s'appeler `isProp` à la place de `getProp`
- ❑ La propriété peut ne pas avoir de setter si elle n'est utilisée qu'en lecture

Attacher une propriété

- ❑ `<h:inputText value="#{employe.nom}" />`
- ❑ La valeur affichée au départ par le formulaire sera la valeur renvoyée par la méthode `getNom` du bean `employe`
- ❑ Lorsque une valeur sera saisie par l'utilisateur dans ce composant, cette valeur sera mise automatiquement dans la propriété `nom` par la méthode `setNom`

Association bean – page JSF

- ❑ Souvent une page – un bean : le bean gère les données associées à la page et les actions déclenchées par l'utilisateur sur cette page
- ❑ Mais on peut aussi trouver une page – plusieurs beans ou plusieurs pages – un bean

Injection de propriété dans un bean géré

- ❑ `@ManagedProperty` a été ajouté par JSF 2.0 pour injecter la valeur d'une propriété ; accepte une expression EL (utiliser plutôt `@Inject` de CDI pour faire la même chose)
- ❑ Exemple (on suppose qu'un bean nommé « utilisateur » a été créé lorsque l'utilisateur est entré dans l'application) :

```
@ManagedProperty(value="#{utilisateur}")  
private Utilisateur utilisateur;
```
- ❑ Attention, le bean doit avoir un `setter` pour la propriété ; ici `setUtilisateur`

R. Grin

JSF

page 73

Avec CDI

- ❑ Il est conseillé de préférer l'annotation des backing bean par CDI car les possibilités d'injection sont plus riches
- ❑ Le bean est annoté par `@Named` (annotation du package `javax.inject`)
- ❑ L'annotation `@Model` (stéréotype CDI) regroupe `@Named` et `@RequestScope` pour un backing bean qui a une portée « requête »

R. Grin

JSF

page 74

Exemples avec CDI

- ❑ Le bean suivant pourra être désigné par `employeBean` :

```
@Named  
public class EmployeBean { ... }
```
- ❑ Celui-ci pourra être désigné par `employe` :

```
@Named(name="employe")  
public class EmployeBean { ... }
```

R. Grin

JSF

page 75

Que signifie « géré » ?

- ❑ Le bean est géré quand c'est le container qui le crée et qui le supprime
- ❑ Par exemple, quand une page JSF désigne un bean, le container regarde s'il existe un bean du même type dans la portée (voir section suivante)
- ❑ Si c'est le cas, le bean existant est utilisé par la page JSF
- ❑ Sinon un nouveau bean est créé et utilisé par la page JSF

R. Grin

JSF

page 76

`@PostConstruct` et `@PreDestroy`

- ❑ Une méthode du bean (géré par JSF ou CDI) annotée par `@PostConstruct` est exécutée à la création du bean par le container
- ❑ Utile, par exemple, pour récupérer des données dans la base de données pour les afficher dans une page JSF (ou pour acquérir une connexion à une ressource externe)
- ❑ Une méthode du bean annotée par `@PreDestroy` sera exécutée à la suppression du bean par le container (utile pour fermer une connexion)

R. Grin

JSF

page 77

Portée (scope)

R. Grin

JSF

JSF - page 78

Types de portées

- ❑ La portée requête commence au début d'une requête et se termine à la fin de la requête
- ❑ La portée session commence quand l'utilisateur se connecte à l'application et se termine après un certain timeout (30 minutes par défaut) ou lorsqu'une commande termine la session
- ❑ La portée application commence au déploiement de l'application et dure tout le temps du fonctionnement de l'application
- ❑ Il existe aussi d'autres types de portées

R. Grin

JSF

page 79

Map des portées

- ❑ Une map est associée à chaque portée
- ❑ Cette map permet d'enregistrer des objets, « les objets de la portée »
- ❑ Par exemple, il peut exister des objets dans la portée requête ; ces objets pourront être utilisés par le code de l'application pendant toute la durée de la requête

R. Grin

JSF

page 80

Portées et backing beans

- ❑ Les beans gérés ont une portée
- ❑ Quand l'application a besoin d'un bean géré, le container regarde s'il existe déjà un bean de ce type dans la portée
- ❑ Si c'est le cas, ce bean est utilisé ; sinon un nouveau bean est créé et mis dans la portée, avec son nom comme clé de la map associée à la portée
- ❑ Le bean est supprimé par le container à la fin de la portée ; par exemple un bean de portée requête est supprimé quand la réponse est renvoyée au client

R. Grin

JSF

page 81

Utilité des portées

- ❑ Une page JSF peut contenir une expression EL qui fait référence à un backing bean
- ❑ Le bean est créé par le container la première fois qu'il est injecté
- ❑ Il va durer pendant toute la portée ; si une autre expression EL y fait référence dans la même portée, c'est le même bean qui sera utilisé, facilitant ainsi le partage d'information entre différentes parties de l'application

R. Grin

JSF

page 82

2 grands types de portée

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

R. Grin

JSF

JSF - page 83

Portées de JSF

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

R. Grin

JSF

page 84

Portées de CDI

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

R. Grin

JSF

page 85

Serializable

- ❑ Les backing beans qui ont la portée **Application**, **Session** et **Conversation** doivent implémenter `Serializable` puisqu'ils peuvent être retirés temporairement de la mémoire centrale par le serveur d'application

R. Grin

JSF

page 86

Attention aux mélanges !

- ❑ Les backing beans qui utilisent les portées CDI doivent être annotés par `@Named` ; ils utilisent les portées du paquetage `javax.enterprise.context`
- ❑ Les backing beans qui utilisent les portées JSF doivent être annotés par `@ManagedBean` ; ils utilisent les portées du paquetage `javax.faces.bean`
- ❑ Un bean annoté avec `@ManagedBean` et `@javax.enterprise.context.SessionScope` aura une portée requête (la valeur par défaut) !

R. Grin

JSF

page 87

Recherche d'un objet géré

- ❑ Lorsqu'une valeur est référencée dans une page JSF, la valeur est d'abord recherchée dans les portées les plus étroites avant d'être recherchée dans les portées plus larges

R. Grin

JSF

page 88

Portée conversation

- ❑ Cette portée permet à un bean d'exister du début à la fin d'une conversation
- ❑ Cette portée est intermédiaire entre la portée requête et la portée session puisqu'elle peut s'étendre sur plusieurs requêtes
- ❑ Le développeur indique le début et la fin de la conversation avec un objet du type `javax.enterprise.context.Conversation`
- ❑ Cette interface contient les méthodes `begin()` et `end()`

R. Grin

JSF

page 89

Utilisation

- ❑ Il faut injecter une conversation dans un bean, appeler `conversation.begin()` au début de la conversation et `conversation.end()` à la fin
- ❑ La portée est `RequestScope` tant que `begin()` n'a pas été appelé

R. Grin

JSF

page 90

Exemple

- ❑ Une page contient une liste de clients dans une table (h:dataTable)
- ❑ Chaque ligne contient un lien vers une page qui permet de modifier les informations sur le client
- ❑ Le lien lance la méthode `afficherDetails` du backing bean
- ❑ Le bouton qui soumet les modifications sur un client lance la méthode `update` du backing bean
- ❑ Une conversation est nécessaire pour conserver des informations sur une liste déroulante utilisée pour une des informations sur le client

R. Grin

JSF

page 91

Code de l'exemple

```
@Named
@ConversationScoped
public class ClientBean implements Serializable {
    @Inject private Conversation conversation;
    ...
    public String afficherDetails(Client client) {
        this.client = client;
        conversation.begin();
        return "Details?id=" + client.getClientId();
    }
    public String update() {
        client = clientManager.update(client);
        conversation.end();
        return "ListeClients";
    }
}
```

R. Grin

JSF

page 92

Difficultés

- ❑ Une difficulté vient du fait qu'une page peut être affichée de diverses façons (bouton back, rafraîchissement de la page, affichage d'une page conservée dans un marque-page) et qu'appeler plusieurs fois `begin` ou `end` lève une exception `IllegalStateException`
- ❑ `conversation.isTransient()` peut aider si on veut envisager tous les cas (renvoie `true` si la conversation n'est pas en cours)

R. Grin

JSF

page 93

Mettre une valeur dans une portée

- ❑ Le code Java de l'application peut utiliser le contexte externe pour mettre une valeur dans une portée ou pour la retrouver
- ❑ Il permet de récupérer les maps des portées requête, session et application
- ❑ Exemple en Java :

```
FacesContext.getCurrentInstance()
.getExternalContext().getSessionMap()
.set("nom", "Bernard");
```

R. Grin

JSF

page 94

Récupérer une valeur mise dans la session

- ❑ En Java :

```
FacesContext.getCurrentInstance()
.getExternalContext().getSessionMap()
.get("nom");
```
- ❑ Dans une page Web, avec le langage EL :

```
#{sessionScope['nom']}
```

R. Grin

JSF

page 95

Mémoire flash

- ❑ Il est bon de limiter la taille des informations entreposées dans la session
- ❑ Cependant la portée « request » peut ne pas convenir, en particulier s'il y a redirection de la requête (par exemple pour appliquer le modèle POST – REDIRECT – GET)
- ❑ Outre la portée conversation de CDI, la mémoire « flash » (JSF 2.0) peut convenir : un objet mis dans la mémoire flash est disponible dans la prochaine requête de la session ; il est ensuite effacé automatiquement de la mémoire flash

R. Grin

JSF

page 96

Utiliser la mémoire flash (page Web)

- Dans une page Web, avec le langage EL (**flash** est un objet implicite du langage) :
`#{flash.objet}`

Utiliser la mémoire flash (en Java)

- Il faut récupérer le contexte externe :

```
ExternalContext ec =  
    FacesContext.getCurrentInstance()  
        .getExternalContext().getFlash();
```

qui renvoie un objet de la classe **Flash** (de **javax.faces.context**) qui se manipule comme une session (comme une map) dont la clé est une **String** et la valeur est un **Object**

- Par exemple,

```
ec.getFlash().put("nom", nom);
```

Exemple

```
<h:form>  
  <h:inputText value="#{flash.nom}" .../>  
  ...  
  <h:commandButton value="Enregistrer"  
    action="confirm?faces-redirect=true" />  
</form>
```

Page `confirm.xhtml` :

```
<h:form>  
  <h:outputText value="#{flash.nom}" .../>  
  ...  
</form>
```

keep

- Il est possible de garder l'objet pour une requête de plus grâce à la méthode **keep** de la classe **Flash**
- `#{flash.keep.objet}` renvoie l'objet qui a la clé « objet » et garde l'objet pour encore une requête de plus
- Dans le code Java, il faut le faire en 2 étapes : récupérer la valeur et remettre dans le flash pour la prochaine requête en appelant la méthode **keep** qui ne renvoie rien (**void** comme type retour).

Exemple

EJB

- ❑ Pour plus de détails, se référer au cours sur les EJB
- ❑ Nous allons voir ici quelques utilisations caractéristiques des EJB par le code des backing beans

Accès à une base de données

- ❑ Un EJB de type DAO permet de manipuler les données d'une base de données (CRUD)
- ❑ Un backing bean injecte l'EJB pour ranger dans la base les valeurs saisies par l'utilisateur

Exemple de code

```

@ManagedBean
@RequestScoped
public class EmployeBean {
    private Employe courant;
    @EJB
    private ejb.EmployeFacade employeFacade;
    ...
    public String update() {
        employeFacade.update(courant);
        ...
    }
}

```

Ajax

Ajax

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

<f:ajax>

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

<f:ajax> - les attributs

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

Exemple 1

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

Exemple 2

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

Compléments sur les attributs

- ❑ Chaque type de composant a un type d'événement par défaut
- ❑ Par exemple, **valueChange** pour les **<h:inputText>** ou **action** pour les **<h:commandButton>**

Compléments sur les attributs

- ❑ **execute** et **render** doivent indiquer un ou plusieurs identificateurs de composants (le séparateur est l'espace)
- ❑ Quelques valeurs particulières : **@all**, **@this** (le composant qui englobe <f:ajax> ; défaut pour execute), **@form** (tous les composants du formulaire), **@none** (défaut pour render) ; on peut aussi donner une expression EL qui fournit une collection de String (qui désignent des identificateurs) ; expression évaluée durant le rendu de la page

Composants inclus dans <f:ajax>

- ❑ <f:ajax> peut aussi encadrer plusieurs composants JSF qui lanceront alors des appels Ajax sur les événements indiqués :

```
<f:ajax ...>
  <h:panelGrid ...>
    <h:inputText .../>
    <h:commandButton .../>
  </panelGrid ...>
</ajax>
```

Cycle de vie

- Il y a en fait 2 cycles de vie :
 - la partie qui traite des composants indiqués dans l'attribut `execute` : `apply request values`, `process validations` et `update model values` ; les convertisseurs, validateurs et les listeners sont exécutés (mais pas les listeners des `<f:event>` liés) ; les valeurs des composants `input` sont mises dans le modèle (`backing beans`)
 - la partie qui traite du rendu des composants indiqués dans l'attribut `render` : `render response`

R. Grin

JSF

page 115

Listener

- L'attribut `listener` de `<f:ajax>` désigne une méthode qui sera appelée lorsque l'événement déclencheur de l'appel Ajax surviendra :

```
<f:ajax event="keyup" render="table"
        listener="#{bb.m}"
```
- La méthode `listener` doit avoir un paramètre de type `AjaxBehaviorEvent`, interface qui hérite de `FacesEvent`

R. Grin

JSF

page 116

Exemple 1 – valeur composant

```
public void m(AjaxBehaviorEvent event) {
    UIInput input =
        (UIInput) event.getComponent();
    UnType valeur =
        (UnType) input.getValue();
    ...
}
```

R. Grin

JSF

page 117

Exemple 2 – suggestions

- Afficher toutes les valeurs qui commencent comme les caractères tapés par l'utilisateur
- ```
<h:inputText value="#{bean.input}">
 <f:ajax event="keyup" render="mots"/>
</h:inputText>
<h:panelGroup id="mots" layout="block">
 <ui:repeat var="mot" value="#{bean.mots}"
 rendered="#{bean.input != ''}"
 #{mot}

 </ui:repeat>
</h:panelGroup>
```

R. Grin

JSF

page 118

## Exemple 2 – suggestions

- Le backing bean :

```
@Named(value = "bean")
@RequestScoped
public class Bean {
 private String input;
 ...
 public List<String> getMots() {
 if (input == null) return null;
 List<String> mots = new ArrayList<>();
 for (String mot : valeursASuggerer)
 if (mot.startsWith(input)) mots.add(mot);
 return mots;
 }
}
```

R. Grin

JSF

page 119

Exemple 2 – code caractère tapé

- `AjaxBehaviorEvent` ne permet pas de récupérer directement le code du dernier caractère tapé (même des caractères non affichables comme ESC ou Ctrl-a) ; il faut ruser en utilisant un composant caché du côté du client dans lequel le code du caractère tapé est mis par du code Javascript

R. Grin

JSF

page 120

Exemple 2 – code caractère tapé

```
<h:inputText value="#{bean.input}"
onkeyup="document.getElementById('#{hiddenI.clientId}')
.value=event.keyCode">
 <f:ajax event="keyup" execute="@this keyCode"
listener="#{bean.listener}"
render="affiche"/>
</h:inputText>
<h:inputHidden id="keyCode" binding="#{hiddenI}"
value="#{bean.keyCode}" />
<h:outputText id="affiche"
value="#{bean.keyCode}" />
```

R. Grin

JSF

page 121

Mélange d'Ajax et de requête normale

□ Quand un champ de saisie lance une requête Ajax, et que l'utilisateur appuie sur la touche [Entrée] après avoir saisi la valeur, un message d'avertissement indique qu'une requête Ajax a été mélangée avec une requête « normale » car l'appui sur la touche [Entrée] déclenche la soumission du formulaire en même temps que l'action « Ajax »

□ Pour ne plus avoir cet avertissement, il suffit de passer le bouton de soumission en Ajax :

```
<h:commandButton ...><f:ajax execute="@form"
render="@form" /></h:commandButton>
```

R. Grin

JSF

page 122

Templates, modèles de présentation

R. Grin

JSF

page 123

Présentation

□ Les applications doivent utiliser des chartes graphiques pour uniformiser la présentation des pages Web

□ Les *templates* évitent la répétition de code entre les pages qui suivent un même schéma de présentation

□ Un template est un fichier XHTML qui contient des parties nommées (<ui:insert>) qui seront remplacées par du code pour chaque page particulière (appelée client du template) qui utilisera ce template

R. Grin

JSF

page 124

Exemple de template (1)

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

Valeur par défaut

R. Grin

JSF

page 125

Exemple de template (2)

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

R. Grin

JSF

page 126

Une page qui utilise le template

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

R. Grin

JSF

page 127

Utilisation

- ❑ Le développeur crée un fichier template
- ❑ Il peut ensuite indiquer qu'une page Web correspond à ce template, en précisant *****??** A COMPLETER**

R. Grin

JSF

page 128

Balises utilisées

- ❑ <ui:include>
- ❑ <ui:decorate>
- ❑ <ui:insert>
- ❑ <ui:composition>
- ❑ <ui:param>

R. Grin

JSF

page 129

Plusieurs niveaux

- ❑ Un client d'un template peut lui-même être le template de clients
- ❑ Un template peut contenir des fichiers inclus (avec <ui:include>)

R. Grin

JSF

page 130

Fichier inclus

- ❑ Ecrit avec un <ui:composition>

R. Grin

JSF

page 131

Exemple de fichier inclus

R. Grin

JSF

page 132

Paramètres

- ❑ On peut passer des valeurs de paramètres à un fichier inclus par `<ui:include>` ou à un template
- ❑ Les valeurs de ces paramètres sont donnés par des sous-balises `<ui:param>` placées à l'intérieur de la balise `<ui:include>` ou à l'intérieur de la balise `<ui:composition template="template">` du client qui utilise le template :
`<ui:param name="nomP" value="valeurP"/>`
- ❑ Les paramètres sont désignés dans le code du template par l'expression EL `#{nomP}`

R. Grin

JSF

page 133

Exemples de paramètre

- ❑ Avec fichier inclus
- ❑ Avec template

R. Grin

JSF

page 134

`<ui:decorate>`

- ❑ Sert à décorer une partie d'une page par un template
- ❑ Exemple d'utilisation :
`<ui:decorate
    template="/WEB-INF/templ/decorateur.xhtml">
    Ce qui va être décoré
</ui:decorate>`
- ❑ Le template contient des `<ui:insert>` qui désignent la partie décorée (« Ce qui va être décoré ») ; cette partie peut être complexe et comporter plusieurs composants

R. Grin

JSF

page 135

Exemple – décorateur

- ❑ Le décorateur :

R. Grin

JSF

page 136

Exemple – ce qui est décoré

R. Grin

JSF

page 137

Internationalisation

R. Grin

JSF

page 138

Quelques tâches seulement

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

R. Grin

JSF

page 139

faces-config.xml

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

R. Grin

JSF

page 140

Dans une page JSF

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

R. Grin

JSF

page 141

Exemple

- Dans la page JSF :

```
<h:outputLabel
value="#{bundle.EditLabel_nom}" for="id" />
```
- Dans le fichier bundle_en.properties :

```
EditLabel_nom=Family Name:
```
- Dans le fichier bundle_fr.properties :

```
EditLabel_nom=Nom :
```

R. Grin

JSF

page 142

Dans le code Java

- Si des textes sont générés par du code Java, il faut aussi remplacer les références aux textes par des méthodes `getString` envoyées au `ResourceBundle`
- Le code suivant récupère le texte dans la locale par défaut
- `ResourceBundle textes = ResourceBundle.getBundle(CHEMIN_BUNDLE);`
...
`texte = textes.getString(nomPropTexte);`

/Bundle dans notre cas

R. Grin

JSF

page 143

Récupérer un ResourceBundle (1/2)

1. Pour le cas où la locale a pu être modifiée dans la vue en cours la 1^{ère} ligne de code est remplacée par quelques lignes pour récupérer le `ResourceBundle` (textes) :

```
FacesContext context =
 FacesContext.getCurrentInstance();
UIViewRoot viewRoot = context.getViewRoot();
Locale locale = viewRoot.getLocale();
```
2. Récupérer le chargeur de classes

```
ClassLoader loader =
 Thread.currentThread().
 getContextClassLoader();
```

R. Grin

JSF

page 144

Récupérer un ResourceBundle (2/2)

3. Récupérer le ResourceBundle :

```
ResourceBundle bundle =
 ResourceBundle.getBundle(
 bundleName, locale, loader);
```

Pour tester

- ❑ Pour les pages JSF il suffit de configurer son navigateur pour qu'il demande en priorité certaines langues (la configuration dépend de chaque navigateur)
- ❑ Pour le code Java, on peut changer la locale de la vue en cours par

```
FacesContext context =
 FacesContext.getCurrentInstance();
context.getViewRoot().setLocale(locale);
```

Gestion des ressources

Ressources des pages Web

- ❑ Les pages HTML peuvent faire appel à des ressources externes : images, fichier CSS, code Javascript,...
- ❑ Ces ressources sont obtenues du serveur par des requêtes HTTP
- ❑ JSF 2.0 a standardisé la gestion de ces ressources avec un ResourceHandler qui gère les requêtes liées aux ressources
- ❑ Le ResourceHandler va chercher les ressources dans des endroits standardisés

Standardisation de l'accès aux ressources

- ❑ Par défaut, les ressources peuvent se situer à 2 endroits dans l'application
  - le répertoire **resources** placé sous la racine de l'application Web contient toutes les ressources
  - Les répertoires (nom par rapport à la racine de l'application) **META-INF/resources** ou **WEB-INF/classes/META-INF/resources**

Librairies

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

Identificateur de ressource

- Identifie une ressource
- Format (seul le nom de la ressource est obligatoire) :
[prefixe/][nomLibrairie/][versionLibrairie/]nomRessource[/versionRessource]
- ****??**** à expliquer, en particulier pour le numéro de version

CSS

- Pour inclure une feuille CSS pour mettre en page une page JSF,
  - placer le fichier css, par exemple *style.css*, dans un sous répertoire *css* du répertoire *resources*
  - ajouter

```
<h:outputStylesheet library="css"
 name="style.css" />
```

dans l'en-tête (<h:header>) de la page JSF

- Dans la page HTML, c'est traduit par :
<script type="text/javascript"
src="/contextAppli/faces/javax.faces.resources
/style.css?ln=css"

Javascript

- Pour inclure du code Javascript pour mettre en page une page JSF,
  - placer le fichier javascript, par exemple *jsf.js*, dans un sous répertoire *css* du répertoire *resources*
  - ajouter

```
<h:outputScript library="javascript"
 name="jsf.js" target="head" />
```

target indique l'endroit où apparaîtra le script dans la page HTML (autre possibilité : *body*) ; par défaut, le script sera là où on a mis le tag

Image

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

Ressource du code Java

- Java EE 6 a la notion de ressource utilisée par du code Java
- Par exemple, une source de données, un EJB,...
- Ces ressources sont associées à des noms JNDI
- La façon « classique » de récupérer une ressource était d'utiliser directement l'API JNDI (lookup)
- Java EE 6 permet le plus souvent de ne pas utiliser directement cette API, grâce à des annotations

Injection de ressource

- Java EE 6 offre plusieurs façon d'injecter des ressources : par des annotations spéciales, par l'annotation `@Resource` ou en utilisant l'API CDI
- L'annotation `@Resource` est une annotation générale pour injecter des ressources ; par exemple pour les sources de données, les services Web, les queues ou topic JMS, les sessions JavaMail et les UserTransaction

Injection de ressource

- ❑ Il est quelquefois indispensable d'injecter une ressource avec une annotation spéciale
- ❑ @EJB doit être utilisée pour injecter un EJB
- ❑ @PersistenceContext permet d'injecter un gestionnaire d'entités (EntityManager), cas particulier car cet objet n'est pas une ressource partagée??**??
- ❑ @Inject permet d'injecter beaucoup de choses (de type primitif ou non) et offre de très nombreuses possibilités (voir cours sur CDI)

R. Grin

JSF

page 157

Définition de ressource

- ❑ Le plus souvent il n'existe pas de manière standard de définir une ressource
- ❑ Avec GlassFish, on utilise le plus souvent la console d'administration pour cela
- ❑ @DataSourceDefinition permet de donner la définition d'une source de données de façon standard et dans le code de l'application
- ❑ Ce qui peut éviter de donner une telle définition sur le serveur d'application (avec la console d'administration de GlassFish par exemple)

R. Grin

JSF

page 158

Chemin « réel » d'un fichier

- ❑ Il peut être utile, par exemple pour lire le contenu d'un fichier texte, de connaître le nom « réel » (celui sur la machine où est placé le serveur d'application) d'un fichier, à partir de son nom dans l'application Web
- ❑ Le code pour avoir ce nom réel :

```
String cheminReel =
 FacesContext.getCurrentInstance()
 .getExternalContext().getRealPath(path);
```

R. Grin

JSF

page 159

Quelques situations typiques

R. Grin

JSF

page 160

Une liste qui dépend d'une autre

- ❑ Les items d'une liste changent suivant la valeur choisie dans une autre liste
- ❑ Exemple :
  - une liste contient des noms de pays
  - une autre liste contient les villes principales du pays choisi dans la première liste

21/10/99

Richard Grin

JSF - page 161

- ❑ **??**

R. Grin

JSF

page 162

Une table et des détails

- Une table contient une liste d'employés, en affichant les caractéristiques principales de ces employés
- L'utilisateur peut choisir un de ces employés pour supprimer cet employé, ou pour faire afficher sur une page à part (ou une fenêtre pop-up) toutes les informations sur cet employé, et éventuellement modifier les informations sur cet employé
- La page à part contient un lien pour revenir à la page qui affiche tous les employés

R. Grin

JSF

page 163

Une table avec des boîtes à cocher

- Une table dont chaque ligne contient une boîte à cocher pour dire si la ligne est sélectionnée
- Le plus simple est d'utiliser une **Map<IdLigne, Boolean>** dont le type IdLigne est le type de l'identificateur de la ligne (le plus souvent l'identificateur de l'entité qui est affichée dans la ligne)
- PrimeFaces et les autres bibliothèques de composants offrent aussi des solutions simples mais non standards

R. Grin

JSF

page 164

Exemple - dataTable

```
<h:dataTable value="#{bean.items}" var="item">
 <h:column>
 <h:selectBooleanCheckbox
 value="#{bean.checked[item.id]}" />
 </h:column>
 ...
 </h:dataTable>
<h:commandButton value="submit"
 action="#{bean.submit}" />
```

Exemple donné par BalusC
(stackoverflow.com)

R. Grin

JSF

page 165

Exemple - bean

```
public class Bean {
 private Map<Long, Boolean> checked = new HashMap<>();
 private List<Item> items;
 public void submit() {
 List<Item> checkedItems = new ArrayList<Item>();
 for (Item item : items) {
 if (checked.get(item.getId())) {
 checkedItems.add(item);
 }
 }
 checked.clear(); // Si nécessaire
 // Traitement des éléments de la liste
 ...
 }
 ...
}
```

R. Grin

JSF

page 166

Une entité avec une collection

- Page qui contient les informations sur une entité et sur une collection d'entités qui lui sont liées (association 1-N)
- Ajouter ou supprimer des entités liées dans la collection***?***

R. Grin

JSF

page 167

Quelques erreurs fréquentes

R. Grin

JSF

page 168

PWC4011

- ❑ Avertissement: PWC4011: Unable to set request character encoding to UTF-8 ... because request parameters have already been read, (ou ServletRequest.getReader() has already been called) ..."
- ❑ Pour l'enlever, il suffit d'ajouter
`<parameter-encoding default-charset="UTF-8"/>`
dans le fichier glassfish-web.xml

21/10/99

Richard Grin

JSF - page 169

PWC3999

- ❑ `java.lang.IllegalStateException: PWC3999: Cannot create a session after the response has been committed`
- ❑ Souvent rencontré avec PrimeFaces
- ❑ Contournement de ce bug : écrire un pre-render listener qui crée une session :
  - `<:event type="preRenderViewEvent" listener="#{bean.preRenderView}"/>`
  - ```
public void preRenderView() {
    HttpSession session = (HttpSession) FacesContext.
        getCurrentInstance().getExternalContext().getSession(true);
}
```

R. Grin

JSF

page 170

Table PrimeFaces ne se trie pas

- ❑ La propriété qui fournit les données à mettre dans la dataTable doit être conservée dans une variable du bean, sinon les données ne sont pas triées quand l'utilisateur clique sur l'icône de tri de la colonne
- ❑ De toute façon c'est une bonne pratique car la méthode qui fournit ces données est appelée plusieurs fois durant l'affichage de la table et cette variable permet de ne pas lancer plusieurs fois la requête SQL qui récupère les données

R. Grin

JSF

page 171

Erreur au déploiement

- ❑ Si l'erreur est incompréhensible et arrive après l'utilisation d'un EJB dans du code Java, aller voir dans web.xml si NetBeans n'a pas ajouté des balises `<ejb-ref>` ou `<ejb-local-ref>`
- ❑ Les enlever si c'est le cas

R. Grin

JSF

page 172