Java Server Faces

Claude Duvallet

Université du Havre
UFR Sciences et Techniques
25 rue Philippe Lebon - BP 540
76058 LE HAVRE CEDEX
Claude.Duvallet@gmail.com
http://litis.univ-lehavre.fr/∼duvallet/

Java Server Faces

Présentations de JSF (1/7)

- Les technologies permettant de développer des applications web avec Java ne cessent d'évoluer :
  1. Servlets.
  2. JSP.
  3. MVC Model 1 : servlets + JSP.
  4. MVC Model 2 : un seule servlet + JSP.
  5. Java Server Faces.
- Java Server Faces (JSF) est une technologie dont le but est de proposer un framework qui facilite et standardise le développement d'applications web avec Java.
- Son développement a tenu compte des différentes expériences acquises lors de l'utilisation des technologies standards pour le développement d'applications web (servlet, JSP, JSTL) et de différents frameworks (Struts, ...).

Présentations de JSF (2/7)

- Le grand intérêt de JSF est de proposer un framework qui puisse être mis en œuvre par des outils pour permettre un développement de type RAD pour les applications web et ainsi faciliter le développement des applications de ce type.
- Ce type de développement était déjà courant pour des applications standalone ou client/serveur lourd avec des outils tel que Delphi de Borland, Visual Basic de Microsoft ou Swing avec Java.
- Ce concept n'est pourtant pas nouveau dans les applications web puisqu'il est déjà mis en œuvre par WebObject d'Apple et plus récemment par ASP.Net de Microsoft mais sa mise en œuvre à grande échelle fût relativement tardive.

Présentations de JSF (3/7)

- L'adoption du RAD pour le développement web trouve notamment sa justification dans le coût élevé de développement de l'IHM à la "main" et souvent par copier/coller d'un mixe de plusieurs technologies (HTML, Javascript, ...), rendant fastidieux et peu fiable le développement de ces applications.
- Plusieurs outils commerciaux intègrent déjà l'utilisation de JSF notamment Studio Creator de Sun, WSAD d'IBM, JBuilder de Borland, JDeveloppper d'Oracle, ...
- Même si JSF peut être utilisé par codage à la main, l'utilisation d'un outil est fortement recommandée pour pouvoir mettre en œuvre rapidement toute la puissance de JSF.
- Ainsi de par sa complexité et sa puissance, JSF s'adapte parfaitement au développement d'applications web complexes en facilitant leur écriture.

Présentations de JSF (4/7)

- Les pages officielles de cette technologie sont à l'url : http ://java.sun.com/j2ee/javaserverfaces/
- La version 1.0 de Java Server Faces, développée sous la JSR-127, a été validée en mars 2004.
- JSF est une technologie utilisée côté serveur dont le but est de faciliter le développement de l'interface utilisateur en séparant clairement la partie "interface" de la partie "métier" d'autant que la partie interface n'est souvent pas la plus compliquée mais la plus fastidieuse à réaliser.

Présentations de JSF (5/7)

- Cette séparation avait déjà été initiée avec la technologie JSP et les bibliothèques de tags personnalisés. JSF va plus loin en reposant sur le modèle MVC et en mettant en œuvre :
  - l'assemblage de composants serveur qui génèrent le code de leur rendu avec la possibilité d'associer certains composants à une source de données encapsulée dans un bean.
  - l'utilisation d'un modèle de développement standardisé reposant sur l'utilisation d'événements et de listener.
  - la conversion et la validation des données avant leur utilisation dans les traitements.
  - la gestion de l'état des composants de l'interface graphique.
  - la possibilité d'étendre les différents modèles et de créer ces propres composants.
  - la configuration de la navigation entre les pages.
  - le support de l'internationalisation.
  - le support pour l'utilisation par des outils graphiques du framework afin de faciliter sa mise en œuvre.

Présentations de JSF (6/7)

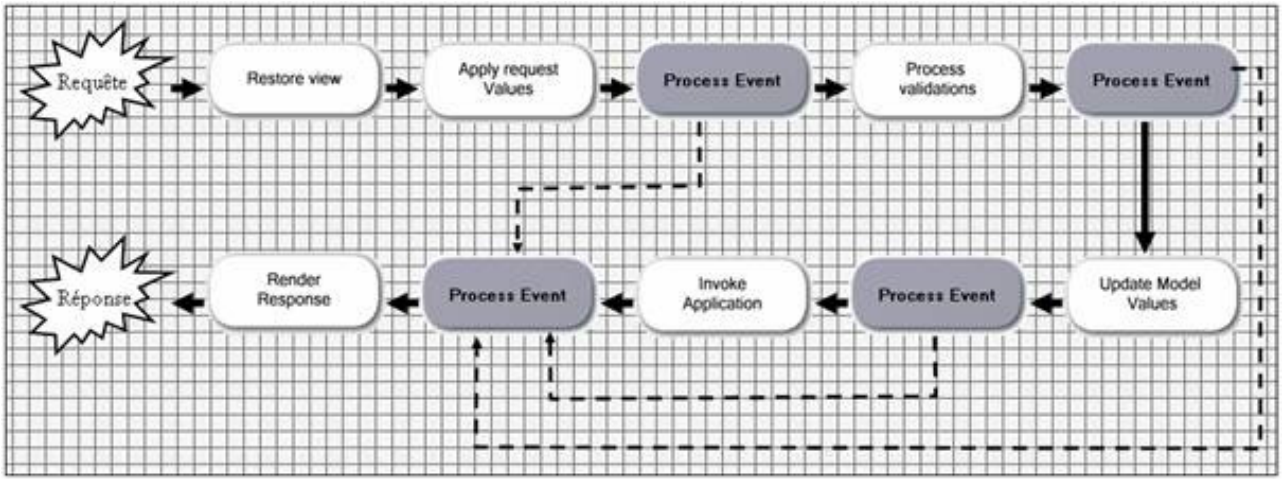
- JSF se compose :
  - d'une spécification qui définit le mode de fonctionnement du framework et une API : l'ensemble des classes de l'API est contenu dans les packages javax.faces.
  - d'une implémentation de référence
  - de bibliothèques de tags personnalisés fournies par l'implémentation pour utiliser les composants dans les JSP, gérer les événements, valider les données saisies, ...
- Le rendu des composants ne se limite pas à une seule technologie même si l'implémentation de référence ne propose qu'un rendu des composants en HTML.

Présentations de JSF (7/7)

- Le traitement d'une requête traitée par une application utilisant JSF utilise un cycle de vie particulier constitué de plusieurs étapes :
  - Création de l'arbre de composants.
  - Extraction des données des différents composants de la page.
  - Conversion et validation des données.
  - Extraction des données validées et mise à jour du modèle de données (javabean).
  - Traitements des événements liés à la page.
  - Génération du rendu de la réponse.
- Ces différentes étapes sont transparentes lors d'une utilisation standard de JSF.

Le cycle de vie d'une requête (1/4)

- JSF utilise la notion de vue (view) qui est composée d'une arborescence ordonnée de composants inclus dans la page.
- Les requêtes sont prises en charge et gérées par le contrôleur d'une application JSF (en général une servlet).



- Celle ci va assurer la mise en œuvre d'un cycle de vie des traitements permettant de traiter la requête en vue d'envoyer une réponse au client.
- JSF propose pour chaque page un cycle de vie pour traiter la requête HTTP et générer la réponse.

Le cycle de vie d'une requête (2/4)

- Ce cycle de vie est composé de plusieurs étapes :
  - Restore view ou Reconstruct Component Tree :
    - Cette première phase permet au serveur de recréer l'arborescence des composants qui composent la page.
    - Cette arborescence est stockée dans un objet de type FacesContext et sera utilisée tout au long du traitement de la requête.
  - Apply Request Value :
    - Dans cette étape, les valeurs des données sont extraites de la requête HTTP pour chaque composant et sont stockées dans leur composant respectif dans le FaceContext.
    - Durant cette phase des opérations de conversions sont réalisées pour permettre de transformer les valeurs stockées sous forme de chaîne de caractères dans la requête http en un type utilisé pour le stockage des données.

Le cycle de vie d'une requête (3/4)

- Étapes suivantes :
  - Perform validations :
    - Une fois les données extraites et converties, il est possible de procéder à leur validation en appliquant les validators enregistrés auprès de chaque composant. Les éventuelles erreurs de conversions sont stockées dans le FaceContext.
    - Dans ce cas, l'étape suivante est directement "Render Response" pour permettre de réafficher la page avec les valeurs saisies et afficher les erreurs
  - Synchronize Model ou update model values :
    - Cette étape permet de stocker dans les composants du FaceContext leur valeur locale validée respective.
    - Les éventuelles erreurs de conversions sont stockées dans le FaceContext.
    - Dans ce cas, l'étape suivante est directement "Render Response" pour permettre de réafficher la page avec les valeurs saisies et afficher les erreurs.

Le cycle de vie d'une requête (4/4)

- Dernières étapes :
  - Invoke Application Logic :
    - Dans cette étape, le ou les événements émis dans la page sont traités.
    - Cette phase doit permettre de déterminer quelle sera la page résultat qui sera renvoyée dans la réponse en utilisant les règles de navigation définie dans l'application.
    - L'arborescence des composants de cette page est créée.
  - Render Response : cette étape se charge de créer le rendue de la page de la réponse.

Le contenu d'une application (1/4)

- Les applications utilisant JSF sont des applications web qui doivent respecter les spécifications de J2EE.
- En tant que telle, elles doivent avoir la structure définie par J2EE pour toutes les applications web :

```
/
/WEB-INF
/WEB-INF/web.xml
/WEB-INF/lib
/WEB-INF/classes
```

- Le fichier web.xml doit contenir au minimum certaines informations notamment, la servlet faisant office de contrôleur, le mapping des url pour cette servlet et des paramètres.

Le contenu d'une application (2/4)

- Exemple de fichier web.xml :

```
<!DOCTYPE web-app PUBLIC
     "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
     "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
    <display-name>Test JSF</display-name>
    <description>Application de tests avec JSF</description>
    <context-param>
<param-name>javax.faces.STATE_SAVING_METHOD</param-name>
<param-value>client</param-value>
    </context-param>
    <!-- Faces Servlet -->
    <servlet>
<servlet-name>Faces Servlet</servlet-name>
<servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
<load-on-startup> 1 </load-on-startup>
    </servlet>
    <!-- Faces Servlet Mapping -->
    <servlet-mapping>
<servlet-name>Faces Servlet</servlet-name>
<url-pattern>*.jsf</url-pattern>
    </servlet-mapping>
</web-app>
```

Le contenu d'une application (3/4)

- Chaque implémentation nécessite un certain nombre de bibliothèques tiers pour leur bon fonctionnement.
- Par exemple, pour l'implémentation de référence, les bibliothèques suivantes sont nécessaires :

```
jsf-api.jar
jsf-ri.jar
jstl.jar
standard.jar
common-beanutils.jar
commons-digester.jar
commons-collections.jar
commons-logging.jar
```

- Remarque : avec l'implémentation de référence, il n'y a aucun fichier .tld à copier car ils sont intégrés dans le fichier jsf-impl.jar.
- Les fichiers nécessaires dépendent de l'implémentation utilisée.

Le contenu d'une application (4/4)

- Les bibliothèques peuvent être mises à disposition de l'application selon plusieurs modes :
  - incorporées dans le package de l'application dans le répertoire /WEB-INF/lib
  - incluses dans le répertoire des bibliothèques partagées par les applications web des conteneurs web s'ils proposent une telle fonctionnalité.
  - Par exemple avec Tomcat, il est possible de copier ces bibliothèques dans le répertoire shared/lib.
- L'avantage de la première solution est de faciliter la portabilité de l'application sur différents conteneur web mais elle duplique ces fichiers si plusieurs applications utilisent JSF.
- Les avantages et inconvénients de la première solution sont exactement l'opposé de la seconde solution.
- Le choix de l'une ou l'autre est donc à faire en fonction du contexte de déploiement.

La configuration de l'application (1/6)

- Toute application utilisant JSF doit posséder au moins deux fichiers de configuration qui vont contenir les informations nécessaires à la bonne configuration et exécution de l'application.
- Le premier fichier est le descripteur de toute application web J2EE : le fichier `web.xml` contenu dans le répertoire WEB-INF.
- Le second fichier est un fichier de configuration particulier au paramétrage de JSF au format XML nommé `faces-config.xml`.

La configuration de l'application (2/6)

- Le fichier `web.xml` doit contenir au minimum certaines informations notamment, la servlet faisant office de contrôleur, le mapping des urls pour cette servlet et des paramètres pour configurer JSF.

```
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
    <display-name>Test JSF</display-name>
    <description>Application de tests avec JSF</description>
    <context-param>
      <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
      <param-value>client</param-value>
    </context-param>
    <!-- Servlet faisant office de controleur-->
    <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup> 1 </load-on-startup>
    </servlet>
    <!-Le mapping de la servlet -->
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>*.faces</url-pattern>
    </servlet-mapping>
</web-app>
```

La configuration de l'application (3/6)

- Le tag `<servlet>` permet de définir une servlet et plus particulièrement dans ce cas de préciser la servlet qui sera utilisée comme contrôleur dans l'application.
- Le plus simple est d'utiliser la servlet fournie avec l'implémentation de référence `javax.faces.webapp.FacesServlet`.
- Le tag `<load-on-startup>` avec comme valeur 1 permet de demander le chargement de cette servlet au lancement de l'application.
- Le tag `<servlet-mapping>` permet de préciser le mapping des urls qui seront traitées par la servlet. Ce mapping peut prendre deux formes :
  - mapping par rapport à une extension : exemple `<url-pattern>*.faces</url-pattern>`.
  - mapping par rapport à un préfixe : exemple `<url-pattern>/faces/*</url-pattern>`.

La configuration de l'application (4/6)

- Les URL utilisées pour des pages mettant en œuvre JSF doivent obligatoirement passer par cette servlet. Ces urls peuvent être de deux formes selon le mapping défini.
- Exemple :
  - `http://localhost:8080/nom_webapp/index.faces`
  - `http://localhost:8080/nom_webapp/faces/index.jsp`
- Dans les deux cas, c'est la servlet utilisée comme contrôleur qui va déterminer le nom de la page JSP à utiliser.
- Le paramètre de contexte `javax.faces.STATE_SAVING_METHOD` permet de préciser le mode d'échange de l'état de l'arbre des composants de la page. Deux valeurs sont possibles :
  - client.
  - server.

La configuration de l'application (5/6)

- Il est possible d'utiliser l'extension .jsf pour les fichiers JSP utilisant JSF à condition de correctement configurer le fichier `web.xml` dans ce sens.
- Pour cela deux choses sont à faire :
  - il faut demander le mapping des url terminant par .jsf par la servlet :

```
<servlet-mapping>
    <servlet-name>jsp</servlet-name>
    <url-pattern>*.jsf</url-pattern>
</servlet-mapping>
```

  - il faut préciser à la servlet le suffix par défaut à utiliser :

```
<context-param>
    <param-name>javax.faces.DEFAULT_SUFFIX</param-name>
    <param-value>.jsf</param-value>
</context-param>
```

La configuration de l'application (6/6)

- Le démarrage d'une application directement avec une page par défaut utilisant JSF ne fonctionne pas correctement.
- Il est préférable d'utiliser une page HTML qui va effectuer une redirection vers la page d'accueil de l'application
- Exemple :

```
<html>
    <head>
        <meta http-equiv="Refresh" content= "0; URL=index.faces"/>
        <title>Demarrage de l'application</title>
    </head>
    <body>
        <p>D&eacute;marrage de l'application ...</p>
    </body>
</html>
```

- Il suffit alors de préciser dans le fichier `web.xml` que cette page est la page par défaut de l'application.
- Exemple :

```
<welcome-file-list>
    <welcome-file>index.html</welcome-file>
</welcome-file-list>
```

Le fichier `faces-config.xml` (1/2)

- Il faut placer ce fichier dans le répertoire WEB-INF de l'application Web.
- Il est aussi possible de préciser son emplacement dans un paramètre de contexte nommé `javax.faces.application.CONFIG_FILES` dans le fichier `web.xml`.
- Il est possible par ce biais de découper le fichier de configuration en plusieurs morceaux.
- Ceci est particulièrement intéressant pour de grosses applications car un seul fichier de configuration peut dans ce cas devenir très gros.

Le fichier `faces-config.xml` (2/2)

- Il suffit de préciser chacun des fichiers séparés par une virgule dans le tag `<param-value>`.
- Exemple :

```
<context-param>
    <param-name>javax.faces.application.CONFIG_FILES</param-name>
    <param-value>
        /WEB-INF/ma-faces-config.xml, /WEB-INF/navigation-faces.xml,
        /WEB-INF/beans-faces.xml
    </param-value>
</context-param>
```

- Ce fichier au format XML permet de définir et de fournir des valeurs d'initialisation pour des ressources nécessaires à l'application utilisant JSF.
- Ce fichier doit impérativement respecter la DTD proposée par les spécifications de JSF : `http://java.sun.com/dtd/web-facesconfig_1_0.dtd`.
- Le tag racine du document XML est le tag `<face-config>`.

Les beans

- Les beans sont largement utilisées dans une application utilisant JSF notamment pour permettre l'échange de données entre les différentes entités et le traitement des événements.
- Les beans sont des classes qui respectent une spécification particulière notamment la présence :
  - de getters et de setters qui respectent une convention de nommage particulière pour les attributs.
  - un constructeur par défaut sans arguments.
- Différents beans :
  - Les beans managés (managed beans).
  - Les backing beans.

Les beans managés (managed bean) (1/4)

- Les beans managés sont des javabeans dont le cycle de vie va être géré par le framework JSF en fonction des besoins et du paramétrage fourni dans le fichier de configuration.
- Dans le fichier de configuration, chacun de ces beans doit être déclaré avec un tag `<managed-bean>`. Ce tag possède trois tags fils obligatoires :
  - `<managed-bean-name>` : le nom attribué au bean (celui qui sera utilisé lors de son utilisation)
  - `<managed-bean-class>` : le type pleinement qualifié de la classe du bean
  - `<managed-bean-scope>` : précise la portée dans laquelle le bean sera stockée et donc utilisable

Les beans managés (managed bean) (2/4)

- La portée peut prendre les valeurs suivantes :
  - request : cette portée est limitée entre l'émission de la requête et l'envoi de la réponse.
    - Les données stockées dans cette portée sont utilisables lors d'un transfert vers une autre page (forward).
    - Elles sont perdues lors d'une redirection (redirect).
  - session : cette portée permet l'échange de données entre plusieurs échanges avec un même client.
  - application : cette portée permet l'accès à des données pour toutes les pages d'une même application quelque soit l'utilisateur.
- Exemple :

```
<managed-bean>
    <managed-bean-name>login</managed-bean-name>
    <managed-bean-class>LoginBean</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

Les beans managés (managed bean) (3/4)

- Il est possible de fournir des valeurs par défaut aux propriétés en utilisant le tag `<managed-property>`.
- Ce tag possède deux tags fils :
  - `<property-name>` : nom de la propriété du bean
  - `<value>` : valeur à associer à la propriété
- Exemple :

```
<managed-bean>
    <managed-bean-name>login</managed-bean-name>
    <managed-bean-class>LoginBean</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
    <managed-property>
        <property-name>nom</property-name>
        <value>test</value>
    </managed-property>
</managed-bean>
```

Les beans managés (managed bean) (4/4)

- Lorsque que le bean sera instancié, JSF appellera automatiquement les setters des propriétés identifiées dans des tags `<managed-property>` avec les valeurs fournies dans leur tag `<value>` respectif.
- Pour initialiser la propriété à null, il faut utiliser le tag `<null-value>`
- Exemple :

```
<managed-property>
    <property-name>nom</property-name>
    <null-value>
</managed-property>
```

- Ces informations seront utilisées par JSF pour automatiser la création ou la récupération d'un bean lorsque celui ci sera utilisé dans l'application.

Les backing Beans (1/3)

- Les beans de type backing bean sont spécialement utilisés avec JSF pour encapsuler tout ou partie des composants qui composent une page et ainsi faciliter leur accès notamment lors des traitements.
- Ces beans sont particulièrement utiles durant des traitements réalisés lors de validations ou de traitements d'événements car ils permettent un accès aux composants dont ils possèdent une référence.

Les backing Beans (2/3)

- Exemple :

```
import javax.faces.component.UIInput;

public class LoginBean {

    private UIInput composantNom;
    private String nom;
    private String mdp;

    public UIInput getComposantNom() {
        return composantNom;
    }

    public void setComposantNom(UIInput input) {
        composantNom = input;
    }

    public String getNom() {
        return nom;
    }
}
```

Les backing Beans (3/3)

- Dans la vue, il est nécessaire de lier un composant avec son attribut correspondant dans le backing bean. L'attribut binding d'un composant permet de réaliser cette liaison.
- Exemple :

```
<h:inputText value="#{login.nom}" binding="#{login.composantNom}" />
```

Les composants pour les interfaces graphiques (1/4)

- JSF propose un ensemble de composants serveurs pour faciliter le développement d'interfaces graphiques utilisateur.
- Pour les composants, JSF propose :
  - un ensemble de classes qui gèrent le comportement et l'état d'un composant
  - un modèle pour assurer le rendu du composant pour un type d'application (par exemple HTML)
  - un modèle de gestion des événements émis par le composant reposant sur le modèle des listeners
  - la possibilité d'associer à un composant un composant de conversion de données ou de validation des données
- Tous ces composants héritent de la classe abstraite UIComponentBase.

Les composants pour les interfaces graphiques (2/4)

- JSF propose 12 composants de base :
  - **UICommand :** Composant qui permet de réaliser une action qui lève un événement.
  - **UIForm :** Composant qui regroupe d'autres composants dont l'état sera renvoyé au serveur lors de l'envoi au serveur.
  - **UIGraphic :** Composant qui représente une image.
  - **UIInput :** Composant qui permet de saisir des données.
  - **UIOutput :** Composant qui permet d'afficher des données.
  - **UIPanel :** Composant qui regroupe d'autres composants à afficher sous la forme d'un tableau
  - **UIParameter :** Composant qui permet de spécifier des paramètres.
  - **UISelectItem :** Composant qui représente un élément sélectionné parmi un ensemble d'éléments.
  - **UISelectItems :** Composant qui représente un ensemble d'éléments.

Les composants pour les interfaces graphiques (3/4)

- Trois derniers composants de base :
  - **UISelectBoolean :** Composant qui permet de sélectionner parmi deux états.
  - **UISelectMany :** Composant qui permet de sélectionner plusieurs éléments parmi un ensemble.
  - **UISelectOne :** Composant qui permet de sélectionner un seul élément parmi un ensemble.
- Ces classes sont des javabeans qui définissent les fonctionnalités de base des composants permettant la saisie et la sélection de données.
- Chacun de ces composants possède un type, un identifiant, une ou plusieurs valeurs locales et des attributs.
- Ils sont extensibles et il est même possible de créer ces propres composants.

Les composants pour les interfaces graphiques (4/4)

- Le comportement de ces composants repose sur le traitement d'événements respectivement le modèle de gestion des événements de JSF.
- Ces classes ne sont pas utilisées directement : elles sont utilisées par la bibliothèque de tags personnalisés qui se charge de les instancier et de leur associer le modèle de rendu adéquat.
- Ces classes ne prennent pas en charge le rendu du composant.
- Par exemple, un objet de type UICommand peut être rendu en HTML sous la forme d'un lien hypertexte ou d'un bouton de formulaire.

Le modèle de rendu des composants (1/2)

- Pour chaque composant, il est possible de définir un ou plusieurs modèles qui se chargent du rendu d'un composant dans un contexte client particulier (par exemple HTML).
- L'association entre un composant et son modèle de rendu est réalisée dans un RenderKit : il précise pour chaque composant quel est le ou les modèles de rendu à utiliser.
- Par exemple, un objet de type UISelectOne peut être rendu sous la forme d'un ensemble de bouton radio, d'une liste ou d'une liste déroulante.
- Chacun de ces rendus est définis par un objet de type Renderer.
- L'implémentation de référence propose un seul modèle de rendu pour les composants qui propose de générer de l'HTML.
- Ce modèle favorise la séparation entre l'état et le comportement d'un composant et sa représentation finale.

Le modèle de rendu des composants (2/2)

- Le modèle de rendu permet de définir la représentation visuelle des composants.
- Chaque composant peut être rendu de plusieurs façons avec plusieurs modèles de rendu.
- Par exemple, un composant de type UICommand peut être rendu sous la forme d'un bouton ou d'un lien hypertexte.
- Dans cet exemple, le rendu est HTML mais il est possible d'utiliser d'autre système de rendu comme XML ou WML.
- Le modèle de rendu met un œuvre un plusieurs kits de rendus.

L'utilisation de JSF dans une JSP (1/2)

- Pour une utilisation dans une JSP, l'implémentation de référence propose deux bibliothèques de tags personnalisés :
  - core : cette bibliothèque contient des fonctionnalités de bases ne générant aucun rendu.
    L'utilisation de cette bibliothèque est obligatoire car elle contient notamment l'élément view
  - html : cette bibliothèque se charge des composants avec un rendu en HTML
- Pour utiliser ces deux bibliothèques, il est nécessaire d'utiliser une directive taglib pour chacune d'elle au début de page jsp.
- Exemple :

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
```

- Le préfix est libre mais par convention ce sont ceux fournis dans l'exemple qui sont utilisés.

L'utilisation de JSF dans une JSP (2/2)

- Le tag `<view>` est obligatoire dans toutes pages utilisant JSF.
- Cet élément va contenir l'état de l'arborescence des composants de la page si l'application est configurée pour stocker l'état sur le client.
- Le tag `<form>` génère un tag HTML form qui définit un formulaire.
- Exemple :

```
<html>
    <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
    <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
    <f:view>
        <head>
            <title>Application de tests avec JSF</title>
        </head>
        <body>
            <h:form>
            ...
            </h:form>
        </body>
    </f:view>
</html>
```

La bibliothèque de tags `Core` (1/2)

- Cette bibliothèque est composée de 18 tags :
  - **actionListener :** ajouter un listener pour une action sur composant.
  - **attribute :** ajouter un attribut à un composant.
  - **convertDateTime :** ajouter un convertisseur de type DateTime à un composant.
  - **convertNumber :** ajouter un convertisseur de type numérique à un composant.
  - **facet :** définit un élément particulier d'un composant.
  - **loadBundle :** charger un fichier contenant les chaînes de caractères d'une locale dans une collection de type Map.
  - **param :** ajouter un paramètre à un composant.
  - **selectitem :** définir l'élément sélectionné dans un composant permettant de faire un choix.
  - **selectitems :** définir les éléments sélectionnés dans un composant permettant de faire un choix.

La bibliothèque de tags `Core` (2/2)

- Suite des tags :
  - **subview :** définir une sous vue.
  - **verbatim :** ajouter un texte brut à la vue.
  - **view :** définir une vue.
  - **validator :** ajouter un valideur à un composant.
  - **validateDoubleRange :** ajouter un valideur de type "plage de valeurs réelles" à un composant.
  - **validateLength :** ajouter un valideur de type "taille de la valeur" à un composant.
  - **validateLongRange :** ajouter un valideur de type "plage de valeurs entières" à un composant.
  - **valueChangeListener :** ajouter un listener pour un changement de valeur sur un composant.
- La plupart de ces tags permettent d'ajouter des objets à un composant.

Quelques références pour programmer

- **Les Tags JSF de la bibliothèque HTML** :

  `http://www.exadel.com/tutorial/jsf/jsftags-guide.html`

- **La FAQ du site developpez.com** :

  `http://javaweb.developpez.com/faq/jsf/`

- **Les Tags JSF pour HTML et CORE** : `http://www.jsftoolbox.com/documentation/help/12-TagReference/index.jsf`

- **La javadoc MyFACES** :

  `http://myfaces.apache.org/core20/myfaces-api/apidocs/index.html`

- **La documentation pour les Tag JSF** :

  `http://myfaces.apache.org/core20/myfaces-impl/tlddoc/index.html`

- **La documentation pour les Tag Facelet** : `http://myfaces.apache.org/core20/myfaces-impl/tlddoc-facelets/index.html`

- **Les Tags de la librairie JSTL** :

  `http://adiguba.developpez.com/tutoriels/j2ee/jsp/jstl/`