Java Server Faces

Java EE 6 - Sommaire

- Introduction
- Comprendre JSF
  - FacesServlet et faces-config.xml
  - Pages et composants
  - Renderers
  - Converters et validators
  - Managed beans et navigation
  - Support d'Ajax
- Les nouveautés de JSF 2.0

Java EE 6 - Sommaire

- Implémentation de référence
- Exemple complet

Introduction

- L'information traitée par le back end doit être présentée aux utilisateurs
- L'interface utilisateur peut être de différents types : applications de bureau, application web dans un navigateur, application mobile embarquée, …
- Au départ, le World Wide Web a été pensé comme un mécanisme pour échanger des documents écrits en Hypertext Markup Language (HTML)

Introduction

- Hypertext Transfer Protocol (HTTP) est le protocole d'échange des documents
- Les documents sont statiques ou dynamiques, c'est à dire assemblés ou non à la volée
- Pour créer un contenu dynamique, il faut parser la requête HTTP, la comprendre et répondre dans un format approprié pour le navigateur
- L'API Servlet a simplifié ce processus en fournissant une vision objet (HttpRequest, HttpResponse, ...)

Introduction

- Le modèle à base de Servlets s'est révélé trop bas niveau
- JSP a été introduit pour simplifier le processus de création dynamique de pages. Les JSPs sont en fait des Servlets, mais principalement écrites en HTML avec du code Java
- En réponse à des limitations des JSP, JavaServer Faces (JSF ou Faces) a été créé avec un nouveau modèle : amener au web la notion de composant graphique

Introduction

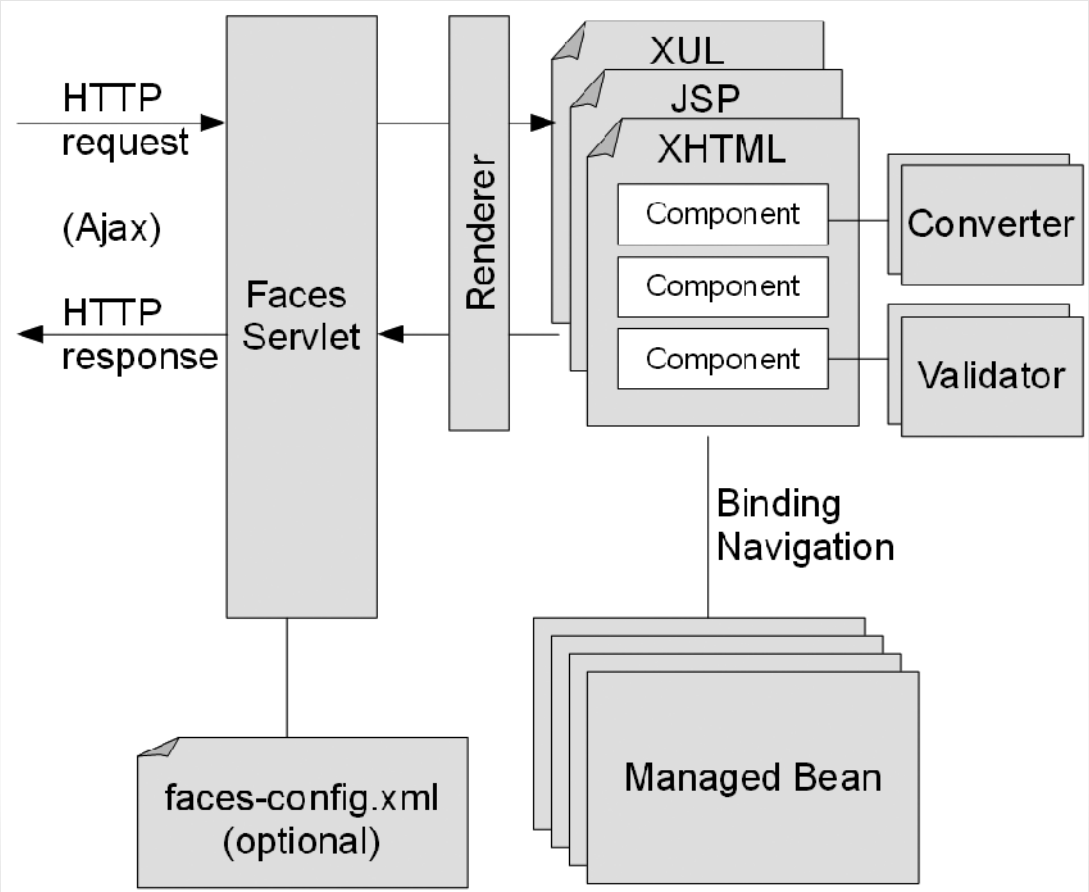
- Inspiré du modèle de composants de Swing et d'autres frameworks, JSF permet de penser en terme de composants, d'événements et d'interactions plutôt que de requêtes, réponses et langage de balises
- Le but de JSF est de simplifier et d'accélérer le temps de développement de la couche web en grâce à des composants d'interface (comme les champs de texte, les listes, les onglets et les grilles)

Comprendre JSF

- Les applications JSF sont des applications web standard qui interceptent les requêtes HTTP via la servlet FacesServlet et répondent de l'HTML
- Sous le capot, l'architecture permet de brancher différents PDL (Page Definition Language), de rendre la page pour différents clients (navigateur web, appareils portables, ...), d'utiliser les événements, listeners et composants à la Swing

Comprendre JSF

- L'architecture de JSF est facile à comprendre quand on est habitué aux frameworks web



Comprendre JSF

- FacesServlet et faces-config.xml : Servlet principale de l'application et fichier de configuration optionnel
- Pages et composants : JSF permet différents PDL comme JSP ou Facelets
- Renderers : ils sont responsables du rendu des composants et de transformer les valeurs utilisateurs en valeurs de composants

Comprendre JSF

- Converters : convertissent une valeur de composant (Date, Boolean, …) en valeur de markup (String)
- Validators : vérifient les données entrées par l'utilisateur
- Managed bean et navigation : la logique métier et la navigation sont gérées par les managed beans
- Support d'Ajax : JSD 2.0 support Ajax

FacesServlet et faces-config.xml

- JSF comme la plupart des frameworks web suivent le design pattern MVC (Model-View-Controller)
- Le pattern permet de découpler la vue (la page) du modèle (les données à afficher)
- Le controleur gère les actions utilisateurs pour mettre à jour le modèle ou les vues
- Dans JSF, le controleur est FacesServlet
- On utilise faces-config.xml ou les annotations pour configurer la le controleur FacesServlet

Pages et composants

- Le framework JSF doit envoyer une page au client en utilisant une technologie pour la PDL
- Une application JSF peut utiliser plusieurs technologies PDL, dont JSP et Facelets
- JSP était le PDL par défaut dans JSF 1.1 et 1.2, Facelets est le PDL par défaut de JSF 2.0

Pages et composants

- Les pages JSP et Facelets sont organisées comme un arbre de composants (aussi appelés widgets) qui fournissent une fonctionnalité spécifique pour interagir avec l'utilisateur (bouton, champ de saisie, combo box, ...)
- JSF a un ensemble de composants standards et permet d'en développer ou d'en brancher d'autres
- La page a un cycle de vie complexe pour gérer cet arbre de composants (initialisation, événements, rendu, …)

Pages et composants

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:h="http://java.sun.com/jsf/html">
   <h:head>
      <title>Creates a new book</title>
   </h:head>
   <h:body>
      <h1>Create a new book</h1>
      <hr/>
      <h:form>
         <table border="0">
            <tr>
               <td><h:outputLabel value="ISBN : "/></td>
               <td><h:inputText value="#{bookController.book.isbn}"/></td>
            </tr>
            <tr>
               <td><h:outputLabel value="Title :"/></td>
               <td><h:inputText value="#{bookController.book.title}"/></td>
            </tr>
         </table>
         <h:commandButton value="Create a book" action="#{bookController.doCreateBook}"
               styleClass="submit"/>
      </h:form>
      <hr/>
   </h:body></html>
```

Renderer

- JSF permet deux modes de développement pour rendre les composants : direct ou délégué
- Le mode direct consiste à laisser les composants à gérer leur rendu et inversement
- Dans le mode délégué le travail est délégué à un renderer. Cela permet à un composant d'être indépendant de la technologie de rendu et d'avoir plusieurs représentations différentes
- Un renderer doit gérer le rendu d'un composant et de traduire les entrées utilisateur pour le composant

Renderer

- On peut penser au renderer comme à un traducteur entre le client et le serveur : il décode la requête utilisateur pour définir les valeurs du composant et encode la réponse pour créer une représentation d'un composant
- Les renderers sont organisés en kits spécialisés pour un type particulier de sortie. Par défaut, JSF inclut un kit avec des renderers pour HTML 4.01
- Il pourrait y en avoir d'autres pour SVG, WML, ...

Converters et Validators

- Une fois un page rendue, l'utilisateur peut intéragir avec et entrer des données
- Les converters convertissent les objets (Integer, Date, Enum, Boolean, …) en String pour l'affichage et inversement
- JSF fournit un ensemble de converters pour les types courants (javax.faces.convert)

Converters et Validators

- Parfois il faut valider les données avant de les faire traiter par le back-end. Les validators vérifient que les valeurs rentrées par l'utilisateur sont acceptables
- On peut associer un ou plusieurs validator à un même composant
- JSF fournit par défaut un certain nombre de validateurs (LengthValidator, RegexValidator, ...)

Managed beans et navigation

- Tous les concepts vus jusqu'à présent étaient lié à une seule page : ce qu'est une page, un composant, comme ils sont rendus, convertis et validés
- Les applications web sont faites de plusieurs pages et doivent réaliser de la logique métier (en appelant la couche EJB par exemple)
- Passer d'une page à l'autre, invoquer des EJBs et synchroniser les données entre composants est géré par les managed beans

Managed beans et navigation

- On associe un composant avec une propriété ou action d'un managed bean en utilisant l'Expression Language (EL)

```
<h:inputText value="#{bookController.book.isbn}"/>
<h:commandButton value="Create" action="#{bookController.doCreateBook}"/>
```

```
@ManagedBean
public class BookController {
    @EJB
    private BookEJB bookEJB;
    private Book book = new Book();

    public String doCreateBook() {
        book = bookEJB.createBook(book);
        return "listBooks.xhtml";
    }
    // Getters, setters
}
```

Support d'Ajax

- Une application doit fournir une interface à la fois riche et réactive
- La réativité peut être obtenue en ne mettant à jour qu'un petite partie de la page de façon asynchrone, c'est tout l'objet de Ajax
- Les versions précédentes de JSF ne géraient pas Ajax et il fallait utiliser des librairies tiers comme a4jsf.
- Avec JSF 2.0, le support d'Ajax a été ajouté sous la forme d'une librairie JavaScript jsf.js

Support d'Ajax

```
<h:commandButton id="submit" value="Create a book"
    onclick="jsf.ajax.request(this, event,
        {execute:'isbn title price description nbOfPage illustrations',
         render:'booklist'}); return false;"
    actionListener="#{bookController.doCreateBook}" />
```

Les nouveautés de JSF 2.0

- JSF est une évolution de JSF 1.2 avec de nombreuses nouveautés
- Les nouveautés incluent :
  - Une technologie de rendu alternative basée sur Facelets
  - Un nouveau système de gestion des ressources (pour les images, CSS, fichiers javascript, …)
  - De nouveaux scopes (view et component scope)
  - Un développement facile avec les annotations pour les managed beans, renderers, converters, ...

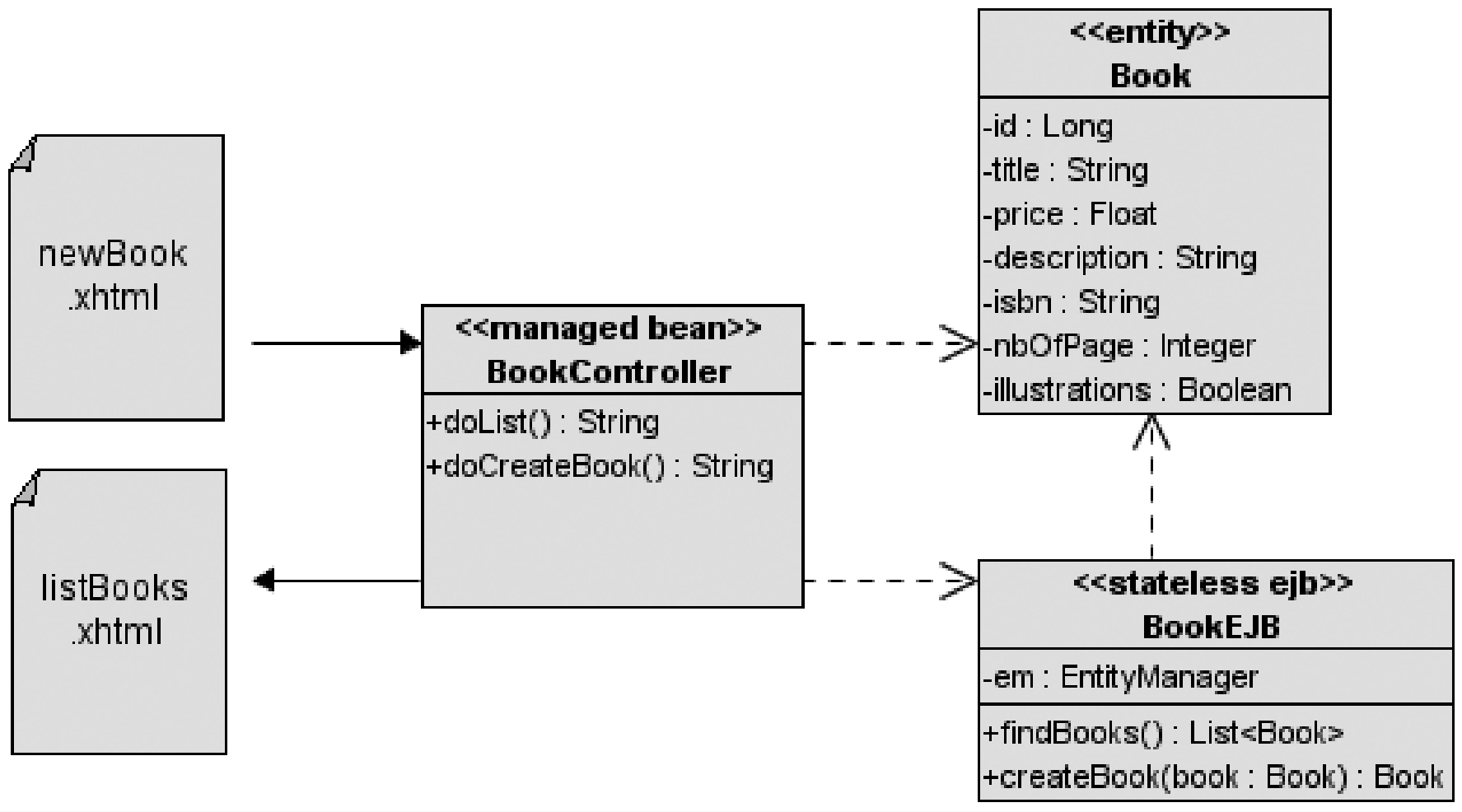
Les nouveautés de JSF 2.0

- La réduction de la configuration XML en utilisant les annotations et la configuration par l'exception (faces-config.xml devenant optionnel)
- Le support d'Ajax
- Un modèle de développement facile de composants

Implémentation de référence

- L'implémentation de référence s'appelle Mojarra, du nom d'un poisson de la côte Caraïbe et sud américaine
- Mojarra est disponible depuis le centre de mise à jour de GlassFish v3 et permet le développement d'applications JSF 2.0

Exemple complet



Exemple complet

- L'application web suit la structure de répertoires Maven :
  - src/main/java : contient Book entité, BookEJB, et BookController managed bean
  - src/main/resources : persistence.xml pour la persistence de l'entité en base
  - src/webapp : contient les deux pages newBook.xhtml et listBooks.xhtml
  - src/webapp/WEB-INF : fichier web.xml qui déclare FacesServlet
  - pom.xml : la description du projet Maven

Book entité

```
@Entity
@NamedQuery(name = "findAllBooks", query = "SELECT b FROM Book b")
public class Book {
    @Id @GeneratedValue
    private Long id;
    @Column(nullable = false)
    private String title;
    private Float price;
    @Column(length = 2000)
    private String description;
    private String isbn;
    private Integer nbOfPage;
    private Boolean illustrations;

    // Constructors, getters, setters
}
```

BookEJB

```
@Stateless
public class BookEJB {
    @PersistenceContext(unitName = "chapter10PU")
    private EntityManager em;

    public List<Book> findBooks() {
        Query query = em.createNamedQuery("findAllBooks");
        return query.getResultList();
    }

    public Book createBook(Book book) {
        em.persist(book);
        return book;
    }
}
```

BookController managed bean

```
@ManagedBean
@RequestScoped
public class BookController {
    @EJB
    private BookEJB bookEJB;

    private Book book = new Book();
    private List<Book> bookList = new ArrayList<Book>();

    public String doNew() {
        return "newBook.xhtml";
    }
    public String doCreateBook() {
        book = bookEJB.createBook(book);
        bookList = bookEJB.findBooks();
        return "listBooks.xhtml";
    }
    // Getters, setters
}
```

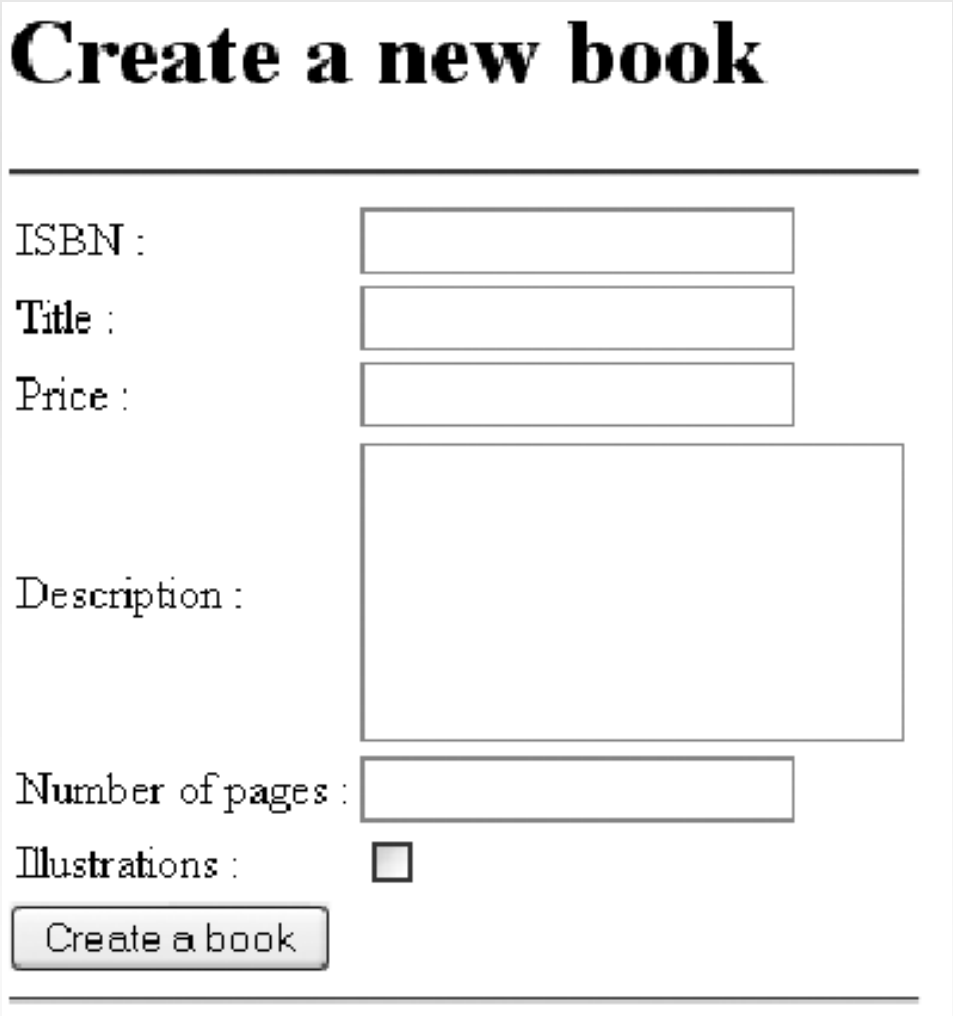
newBook.xhtml

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
<h:head>
    <title>Creates a new book</title>
</h:head>
<h:body>
<h1>Create a new book</h1>
<hr/>
<h:form>
  <table border="0">
    <tr>
      <td><h:outputLabel value="ISBN : "/></td>
      <td><h:inputText value="#{bookController.book.isbn}"/></td>
    </tr>
    <tr>
      <td><h:outputLabel value="Title :"/></td>
      <td><h:inputText value="#{bookController.book.title}"/></td>
    </tr>
    <tr>
      <td><h:outputLabel value="Price : "/></td>
      <td><h:inputText value="#{bookController.book.price}"/></td>
    </tr>
```

newBook.xhtml

```
    <tr>
      <td><h:outputLabel value="Description : "/></td>
      <td><h:inputTextarea value="#{bookController.book.description}"
                           cols="20" rows="5"/></td>
    </tr>
    <tr>
      <td><h:outputLabel value="Number of pages : "/></td>
      <td><h:inputText value="#{bookController.book.nbOfPage}"/></td>
    </tr>
    <tr>
      <td><h:outputLabel value="Illustrations : "/></td>
      <td><h:selectBooleanCheckbox
                value="#{bookController.book.illustrations}"/></td>
    </tr>
  </table>
  <h:commandButton value="Create a book" ➥
                   action="#{bookController.doCreateBook}"/>
</h:form>
<hr/>
</h:body>
</html>
```

newBook.xhtml



listBooks.xhtml

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">
<h:head>
    <title>List of the books</title>
</h:head>
<h:body>
<h1>List of the books</h1>
<hr/>
<h:dataTable value="#{bookController.bookList}" var="bk">
  <h:column>
    <f:facet name="header">
      <h:outputText value="ISBN"/>
    </f:facet>
    <h:outputText value="#{bk.isbn}"/>
  </h:column>
  <h:column>
    <f:facet name="header">
      <h:outputText value="Title"/>
    </f:facet>
    <h:outputText value="#{bk.title}"/>
  </h:column>
```

listBooks.xhtml

```
…
  <h:column>
    <f:facet name="header">
      <h:outputText value="Number Of Pages"/>
    </f:facet>
       <h:outputText value="#{bk.nbOfPage}"/>
    </h:column>
    <h:column>
      <f:facet name="header">
        <h:outputText value="Illustrations"/>
      </f:facet>
      <h:outputText value="#{bk.illustrations}"/>
    </h:column>
    </h:dataTable>
    <h:form>
      <h:commandLink action="#{bookController.doNew}">Create a new book
      </h:commandLink>
    </h:form>
  <hr/>
  </h:body>
  </html>
```

listBooks.xhtml

List of the books

ISBN	Title	Price	Description	Number Of Pages	Illustrations
1234-234	H2G2	12.0	Scifi IT book	241	false
564-694	Robots	18.5	Asimov Best seller	317	true
256-6-56	Dune	23.25	The trilogy	529	false

Create a new book

web.xml

- Dans la spécification Servlet 3.0, web.xml est devenu optionnel. Cependant JSF 2.0 est basé sur Servet 2.5, ce qui explique qu'il faut toujours le fichier web.xml

```
<?xml version='1.0' encoding='UTF-8'?>
<web-app version="2.5"
        xmlns="http://java.sun.com/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
        http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
    <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>*.faces</url-pattern>
    </servlet-mapping>
</web-app>
```

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" ➥
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"➥
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 ➥
        http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.apress.javaee6</groupId>
    <artifactId>chapter10</artifactId>
    <packaging>war</packaging>
    <version>1.0</version>

    <dependencies>
        <dependency>
            <groupId>javax</groupId>
            <artifactId>javaee-api</artifactId>
            <version>6.0</version>
            <scope>provided</scope>
        </dependency>
```

pom.xml

```
<dependency>
    <groupId>org.eclipse.persistence</groupId>
    <artifactId>eclipselink</artifactId>
    <version>2.0.1</version>
    <scope>provided</scope>
</dependency>

<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <inherited>true</inherited>
            <configuration>
                <source>1.6</source>
                <target>1.6</target>
            </configuration>
        </plugin>
    </plugins>
</build>
</project>
```

mvn package

Déploiement sur GlassFish

- Démarrer Derby puis GlassFish, puis taper :

  asadmin deploy chapter10-1.0.jar

- Pour vérifier le déploiement :

  asadmin list-components

Lancement de l'application

- Dans son navigateur préféré, taper l'URL :

  http://localhost:8080/chapter10-1.0/newBook.faces