Java Server Faces
Traitements et Navigation

Java EE 6 - Sommaire

- Introduction
- Le pattern MVC
  - FacesServlet
  - FacesContext
  - FacesConfig
- Managed Beans
  - Modèle de développement
  - Navigation
  - Gestion des messages

Java EE 6 - Sommaire

- Conversion et validation
  - Converters
  - Converters personnalisés
  - Validators
  - Validators personnalisés
- Ajax
  - Concepts généraux
  - Support dans JSF
  - Exemple

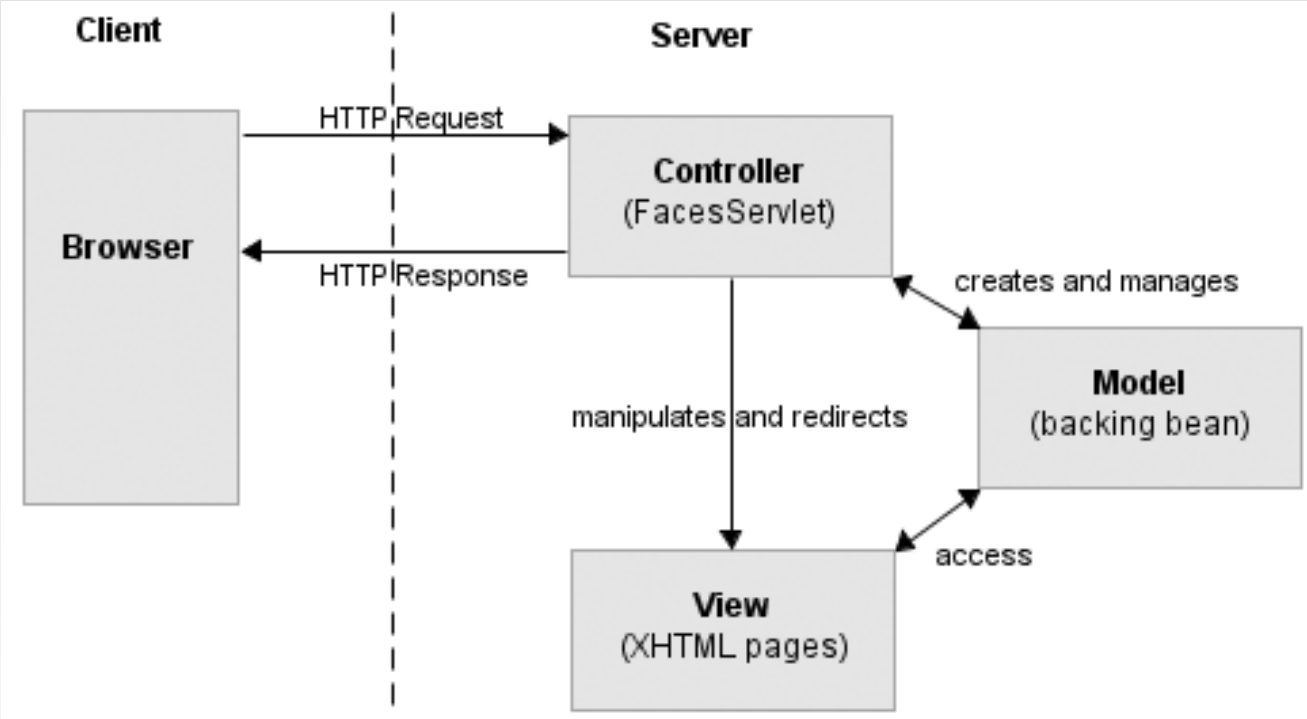
Introduction

- JSF est la technologie conseillée de Java EE 6 pour réaliser des applications web
- La création de pages avec des composants graphiques n'est pas suffisant, il faut faire interagir ces pages avec le back-end
- Dans JSF, ce sont les managed beans qui permettent d'invoquer la couche métier et de naviguer dans l'application
- JSF supporte aussi nativement Ajax

Le pattern MVC

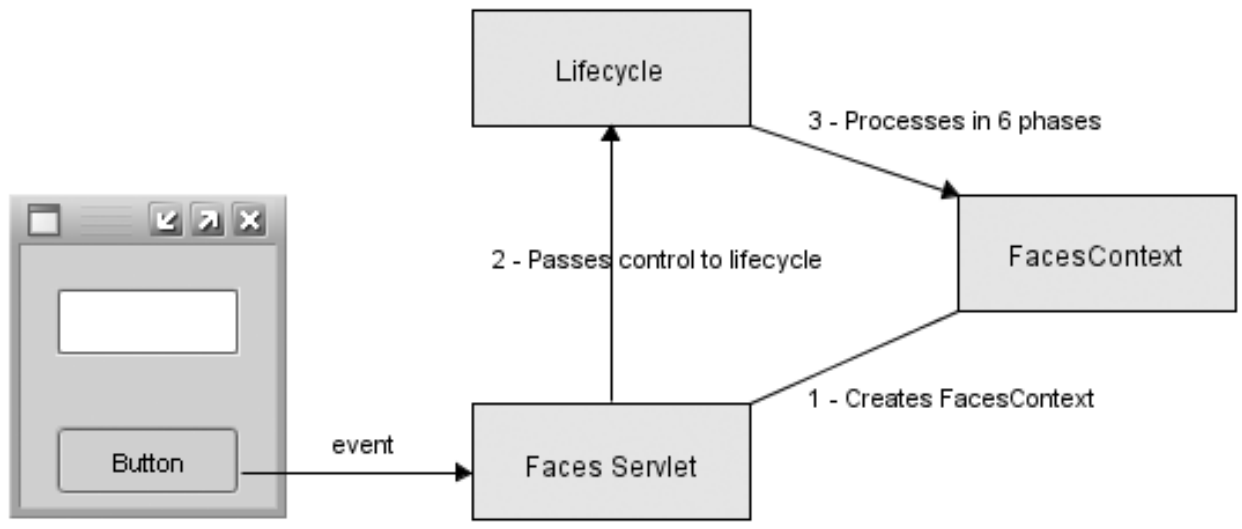
- JSF comme la plupart des frameworks encourage la séparation des tâches en utilisant une variation du pattern MVC
- MVC est un pattern d'architecture visant à isoler la logique métier des l'interface utilisateur
- Quand on mélange les deux, l'application devient très difficile à maintenir et plus difficile à faire monter en charge

Le pattern MVC



Le pattern MVC

- FacesServlet
  - C'est une implémentation de javax.servlet.Servlet qui joue le rôle de controleur
  - Toutes les requêtes passent par cette servlet



Le pattern MVC

- Le fichier web.xml permet d'associer la servlet

```
<?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>
    <context-param>
        <param-name>javax.faces.PROJECT_STAGE</param-name>
        <param-value>Development</param-value>
    </context-param>
</web-app>
```

Le pattern MVC

Parameter	Description
javax.faces.CONFIG_FILES	Defines a comma-delimited list of context-relative resource paths under which the JSF implementation will look for resources.
javax.faces.DEFAULT_SUFFIX	Allows the web application to define a list of alternative suffixes for pages containing JSF content (e.g., .xhtml).
javax.faces.LIFECYCLE_ID	Identifies the Lifecycle instance to be used when processing JSF requests.
javax.faces.STATE_SAVING_METHOD	Defines the location where state is saved. Valid values are server, which is the default (typically saved in HttpSession) and client (saved as a hidden field in the subsequent form submit).
javax.faces.PROJECT_STAGE	Describes where this particular JSF application is in the software development life cycle (Development, UnitTest, SystemTest, or Production). This could be used by a JSF implementation to cache resources in order to improve performance in production, for example.
javax.faces.DISABLE_FACELET_JSF_VIEWHANDLER	Disables Facelets as the default page declaration language (PDL) if set to true.
javax.faces.LIBRARIES	Interprets each file found in the semicolon-separated list of paths as a Facelets tag library.

Le pattern MVC

- FacesContext
  - JSF définit la classe abstraite javax.faces.context.FacesContext pour garder les informations contextuelles liées au traitement d'une requête entrante et à la création de la réponse
  - Pour récupérer une références, on peut utiliser l'objet implicite facesContext dans les pages ou dans les managed beans utiliser la méthode statique FacesContext.getCurrentInstance()

Le pattern MVC

Method	Description
addMessage	Appends an error message.
getApplication	Returns the Application instance associated with this web application.
getAttributes	Returns a Map representing the attributes associated with the FacesContext instance.
agetCurrentInstance	Returns the FacesContext instance for the request that is being processed by the current thread.
getMaximumSeverity	Returns the maximum severity level recorded on any FacesMessage that has been queued.
getMessages	Returns a collection of FacesMessage.
getViewRoot	Returns the root component that is associated with the request.
release	Releases any resources associated with this FacesContext instance.
renderResponse	Signals the JSF implementation that, as soon as the current phase of the request-processing life cycle has been completed, control should be passed to the *Render response* phase, bypassing any phases that have not been executed yet.
responseComplete	Signals the JSF implementation that the HTTP response for this request has already been generated (such as an HTTP redirect), and that the request-processing life cycle should be terminated as soon as the current phase is completed.

Le pattern MVC

- Faces Config
  - Pour configurer la servlet FacesServlet, il est possible comme d'habitude, d'utiliser les annotations ou un fichier de configuration xml : le fichier faces-config.xml
  - Avec JSF 2.0, on peut utiliser les annotations avec les managed beans, converters, event listeners, renderers et validators

Managed Beans

- Dans le modèle MVC, les managed beans sont les passerelles vers le modèle
- Ce sont de simples classes Java annotées
- Ils contiennent la logique métier ou délèguent le travail à des EJB, gèrent la navigation entre pages et contiennent les données
- Une application JSF contient généralement un ou plusieurs managed beans utilisés par plusieurs pages
- Les données sont portées sous forme d'attributs du managed bean et référencées avec EL

Modèle de développement

- Les managed beans sont des classes java gérées par la FacesServlet
- Les composants graphiques sont liés à des propriétés du managed bean et peuvent appeler des méthodes

```
@ManagedBean
public class BookController {
    private Book book = new Book();
    public String doCreateBook() {
        createBook(book);
        return "listBooks.xhtml";
    }
    // Constructors, getters, setters
}
```

Modèle de développement

- Un managed bean est une classe java qui :
  - Doit être annotée par @javax.faces.model.ManagedBean ou l'equivalent xml
  - Doit avoir un ”scope” (par défaut RequestScoped)
  - Doit être publique et ne peut pas être final ou abstract
  - Doit avoir un constructeur par défaut sans paramètre
  - Ne doit pas définir de méthode finalize()
  - Doit avoir des accesseurs publics pour les liser aux composants
- Par défaut le nom d'un managed bean est le nom de sa classe (première lettre minuscule)

Modèle de développement

```
@ManagedBean(name = "myManagedBean")
public class BookController {
    private Book book = new Book();
    public String doCreateBook() {
        createBook(book);
        return "listBooks.xhtml";
    }
    // Constructors, getters, setters
}
```

```
<h:outputText value="#{myManagedBean.book.isbn}"/>
<h:form>
    <h:commandLink action="#{myManagedBean.doCreateBook}">
        Create a new book
    </h:commandLink>
</h:form>
```

Modèle de développement

- Scopes : les Managed beans peuvent avoir une portée (durée de vie et accès) différentes suivant l'annotation :
  - @ApplicationScoped : disponibles toutes la durée de l'application
  - @SessionScoped : disponibles pendant tous les échanges avec un client
  - @ViewScoped : disponibles pendant l'affichage d'une même vue
  - @RequestScoped : disponible pendant la durée de la requête jusqu'à la réponse. **Valeur par défaut**
  - @NoneScoped : managed beans non visibles par les pages JSF

Modèle de développement

```
@ManagedBean(eager = true)
@ApplicationScoped
public class InitController {

   private Book defaultBook;

   @PostConstruct
   private void init() {
      defaultBook=new Book("default title", 0, "default descritpion",➥
                        "0000-000", 100, true);
   }
     // Constructors, getters, setters
}
```

Modèle de développement

- @ManagedProperty
  - Cette balise permet de demander au système d'injecter une valeur dans une propriété

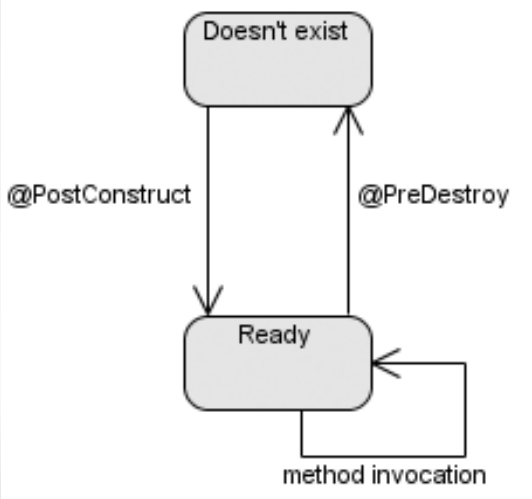
```
@ManagedBean
public class BookController {
    @ManagedProperty(value = "#{initController.defaultBook}")
    private Book book;

    @ManagedProperty(value = "this is a title")
    private String aTitle;

    @ManagedProperty(value = "999")
    private Integer aPrice;
    // Constructors, getters, setters & methods
}
```

Modèle de développement

- Cycle de vie et callback
  - Les managed beans ont aussi un cycle de vie (similaire à celui des Session beans)
  - Les méthodes d'un managed bean peuvent être annotées par @PostConstruct et @PreDestroy



Navigation

- Suivant l'application, on peut avoir des modes de navigation plus ou moins sophistiqués
- JSF propose plusieurs options pour contrôler le flux de l'application
- Pour simplement aller d'une page à l'autre, il est possible de spécifier directement le nom de la page cible :

```
<h:commandButton value="Create" action="listBooks.xhtml"/>
```

Navigation

- Mais la plupart du temps, il faut accéder à la couche métier pour lancer des traitements, c'est là qu'il faut utiliser des managed beans

newBook.xhtml

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

listBooks.xhtml

```
<h:commandLink
  action="#{bookController.doNewBookForm}">
    Create a new book
</h:commandLink>
```

```
@ManagedBean
public class BookController {
    @EJB
    private BookEJB bookEJB;
    private Book book = new Book();
    private List<Book> bookList = new ArrayList<Book>();
    public String doNewBookForm() {
        return "newBook.xhtml";
    }
    public String doCreateBook() {
        book = bookEJB.createBook(book);
        bookList = bookEJB.findBooks();
        return "listBooks.xhtml";
    }
    // Constructors, getters, setters
}
```

Navigation

- La valeur renvoyée par les méthodes du managed bean peut prendre plusieurs formes :
  - Le nom de la page (avec ou sans l'extension .xhtml)
  - Une valeur définie dans faces-config.xml dans la balise <navigation-rule>. Cette balise contient la page de départ, une condition et la page si le résultat est positif

Navigation

```
@ManagedBean
public class BookController {
    // ...
    public String doNewBookForm() {
        return "sucess";
    }
    public String doCreateBook() {
        book = bookEJB.createBook(book);
        bookList = bookEJB.findBooks();
        return "sucess";
    }
    // Constructors, getters, setters
}
```

```
<?xml version='1.0' encoding='UTF-8'?>
<faces-config xmlns="http://java.sun.com/xml/ns/javaee"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
              http://java.sun.com/xml/ns/javaee/web-facesconfig_2_0.xsd"
              version="2.0">
   <navigation-rule>
         <from-view-id>newBook.xhtml</from-view-id>
         <navigation-case>
               <from-outcome>sucess</from-outcome>
               <to-view-id>listBooks.xhtml</to-view-id>
         </navigation-case>
      </navigation-rule>
      <navigation-rule>
         <from-view-id>listBooks.xhtml</from-view-id>
         <navigation-case>
               <from-outcome>sucess</from-outcome>
               <to-view-id>newBook.xhtml</to-view-id>
         </navigation-case>
      </navigation-rule>
   </faces-config>
```

Navigation

- Le fichier XML est utile pour la navigation quand on veut centraliser les enchainements de pages ou quand on a des liens communs sur plusieurs pages (par exemple login et logout)

```
<navigation-rule>
    <from-view-id>*</from-view-id>
    <navigation-case>
        <from-outcome>logout</from-outcome>
        <to-view-id>logout.xhtml</to-view-id>
    </navigation-case>
</navigation-rule>
```

Navigation

- On peut aussi avoir des pages différentes affichées :

```
public String doNewBookForm() {
    switch (value) {
    case 1: return "page1.xhtml"; break;
    case 2: return "page2.xhtml"; break;
    case 3: return "page3.xhtml"; break;
    default: return null; break;
  }
}
```

NB : La valeur de retour null renvoie vers la page courante

Gestion des messages

- Quand les choses se passent mal, on peut afficher des messages sur la page de l'utilisateur. Il y a 2 types de messages :
  - Les erreurs applicatives (logique métier, problème de base, problème réseau, …)
  - Les erreurs de saisie (champs vides, …)
- Les erreurs applicatives sont généralement affichées dans des pages dédiées
- Les erreurs de saisie sont plutôt affichées dans la même page

Gestion des messages

- Les balises <h:message> et <h:messages> permettent d'afficher un ou plusieurs messages
- Pour produire les messages affichés, JSF permet d'empiler les messages en appelant la méthode FacesContext.addMessage() depuis le managed bean

```
void addMessage(String clientId, FacesMessage message)

FacesMessage(Severity severity, String summary, String detail)
```

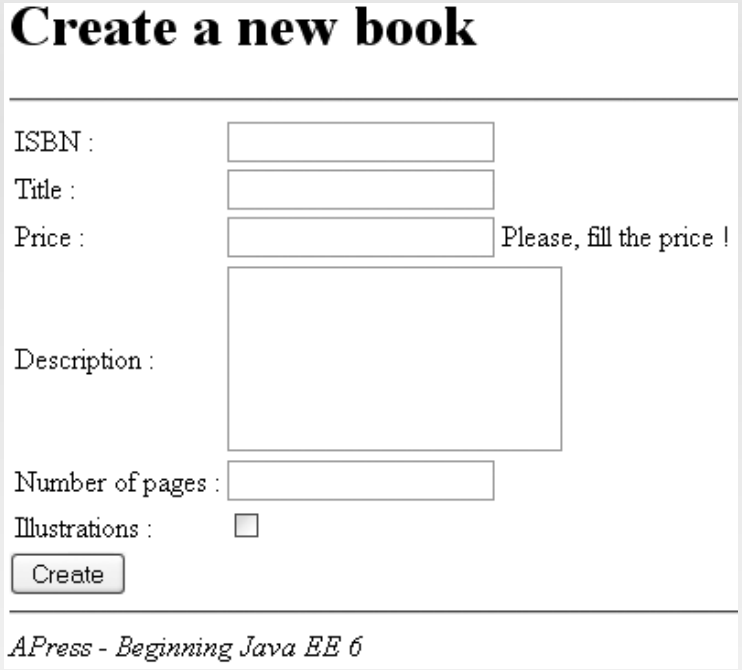
Gestion des messages

```
FacesContext ctx = FacesContext.getCurrentInstance();
try {
  book = bookEJB.createBook(book);
  ctx.addMessage(null, new FacesMessage(FacesMessage.SEVERITY_INFO, ➥
            "Book has been created", "The book" + book.getTitle() + ➥
            " has been created with id=" + book.getId()));
} catch (Exception e) {
  ctx.addMessage(null, new FacesMessage(FacesMessage.SEVERITY_ERROR,➥
              "Book hasn't been created", e.getMessage()));
}
```

Gestion des messages

```
<h:inputText id="priceId" value="#{bookController.book.price}"/>
<h:message for="priceId"/>
```

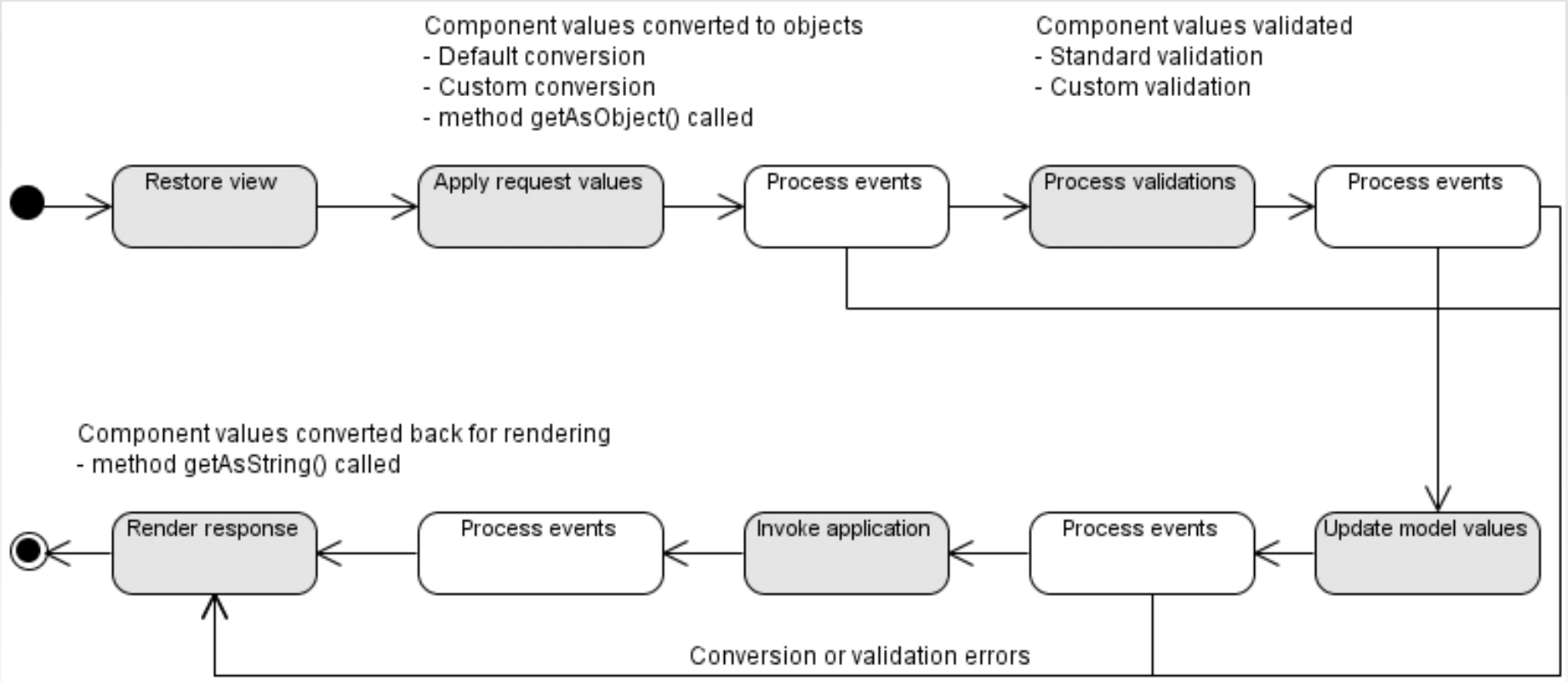
```
if (book.getPrice() == null || "".equals(book.getPrice())) {
    ctx.addMessage("priceId", new FacesMessage(SEVERITY_WARN, ➥
    "Please, fill the price !", "Enter a number value"));
```



Conversion et Validation

- JSF fournit un mécanisme standard de conversion et de validation des entrées utilisateur
- Ces traitements sont faits avant l'appel des méthodes métier (sur le managed bean)
- Ceci permet au développeur de se concentrer sur le traitement métier plutôt que sur la vérification des données

Conversion et Validation



Converters

- Dans le navigateur, les données sont affichées sous forme de chaîne de caractères. Quand le formulaire est envoyé au serveur, les converters doivent transformer ces chaînes en objet (Float, Integer, BigDecimal, Date, …)
- De la même façon le processus inverse se produit quand la réponse doit être renvoyée au navigateur

Converters

- JSF fournit une liste de converters pour tous les types simples (Integer, int, Float, float, …) et convertit automatiquement dans un sens et dans l'autre
- Pour les autres types, vous devez ajouter votre propre converter

Converters

- Converters standards (javax.faces.convert)

Converter	Description
BigDecimalConverter	Converts a String to a java.math.BigDecimal and vice versa
BigIntegerConverter	Converts a String to a java.math.BigInteger and vice versa
BooleanConverter	Converts a String to a Boolean (and boolean primitive) and vice versa
ByteConverter	Converts a String to a Byte (and byte primitive) and vice versa
CharacterConverter	Converts a String to a Character (and char primitive) and vice versa
DateTimeConverter	Converts a String to a java.util.Date and vice versa
DoubleConverter	Converts a String to a Double (and double primitive) and vice versa
EnumConverter	Converts a String to an Enum (and enum primitive) and vice versa
FloatConverter	Converts a String to a Float (and float primitive) and vice versa
IntegerConverter	Converts a String to an Integer (and int primitive) and vice versa
LongConverter	Converts a String to a Long (and long primitive) and vice versa
NumberConverter	Converts a String to an abstract java.lang.Number class and vice versa
ShortConverter	Converts a String to a Short (and short primitive) and vice versa

Converters

- Si la conversion automatique n'est pas correcte, on peut la configurer avec les balises convertNumber et convertDateTime

```
<h:inputText value="#{bookController.book.price}">
    <f:convertNumber currencySymbol="$" type="currency"/>
</h:inputText>

<h:inputText value="#{bookController.book.publishedDate}">
    <f:convertDateTime pattern="MM/dd/yy"/>
</h:inputText>
```

Converters personnalisés

- Pour écrire un converter personnalisé, il suffit d'implémenter les 2 méthodes de l'interface javax.faces.convert.Converter puis de la déclarer dans faces-config.xml ou avec l'annotation @FacesConverter

```
Object getAsObject(FacesContext ctx, UIComponent component, String value)
String getAsString(FacesContext ctx, UIComponent component, Object value)
```

Converters personnalisés

```
@FacesConverter(value = "euroConverter")
public class EuroConverter implements Converter {
   @Override
   public Object getAsObject(FacesContext ctx, UIComponent component,➥
                                        String value) {
      return value;
   }
   @Override
   public String getAsString(FacesContext ctx, UIComponent component,➥
                                         Object value) {
      float amountInDollars = Float.parseFloat(value.toString());
      double ammountInEuros = amountInDollars * 0.8;
      DecimalFormat df = new DecimalFormat("###,##0.##");
      return df.format(ammountInEuros);
   }
}
```

```
<h:outputText value="#{book.price}">
    <f:converter converterId="euroConverter"/>
</h:outputText>

ou

<h:outputText value="#{book.price}" converter="euroConverter"/>
```

Validators

- La vérification des données saisies peut se faire côté client en JavaScript ou côté serveur en utilisant les validators
- JSF simplifie la validation des données avec des validators standards
- Les composants gèrent souvent une première validation avec le côté obligatoire ou non

```
<h:inputText value="#{bookController.book.title}" required="true"/>
```

Validators

Converter	Description
DoubleRangeValidator	Checks the value of the corresponding component against specified minimum and maximum double values
LengthValidator	Checks the number of characters in the string value of the associated component
LongRangeValidator	Checks the value of the corresponding component against specified minimum and maximum long values
RegexValidator	Checks the value of the corresponding component against a regular expression

```
<h:inputText value="#{bookController.book.title}" required="true">
    <f:validateLength minimum="2" maximum="20"/>
</h:inputText>
<h:inputText value="#{bookController.book.price}">
    <f:validateLongRange minimum="1" maximum="500"/>
</h:inputText>
```

Validators personnalisés

- Pour ajouter des validators personnalisés, il faut implémenter l'interface javax.faces.validator.Validator et enregistrer le validator dans faces-config.xml ou avec l'annotation @FacesValidator

```
void validate(FacesContext context, UIComponent component, Object value)
```

Validators personnalisés

```
@FacesValidator(value = "isbnValidator")
public class IsbnValidator implements Validator {
  private Pattern pattern;
  private Matcher matcher;
  @Override
  public void validate(FacesContext context, UIComponent component, Object value) throws ValidatorException {
    String componentValue = value.toString();
    pattern = Pattern.compile("(?=[-0-9xX]{13}$)");
    matcher = pattern.matcher(componentValue);
    if (!matcher.find()) {
      String message = MessageFormat.format("{0} is not a valid isbn format", componentValue);
      FacesMessage facesMessage = new FacesMessage(SEVERITY_ERROR, message, message);
      throw new ValidatorException(facesMessage);
    }
  }
}
```

```
<h:inputText value="#{book.isbn}" validator="isbnValidator"/>

ou

<h:inputText value="#{book.isbn}">
    <f:validator validatorId="isbnValidator" />
</h:inputText>
```

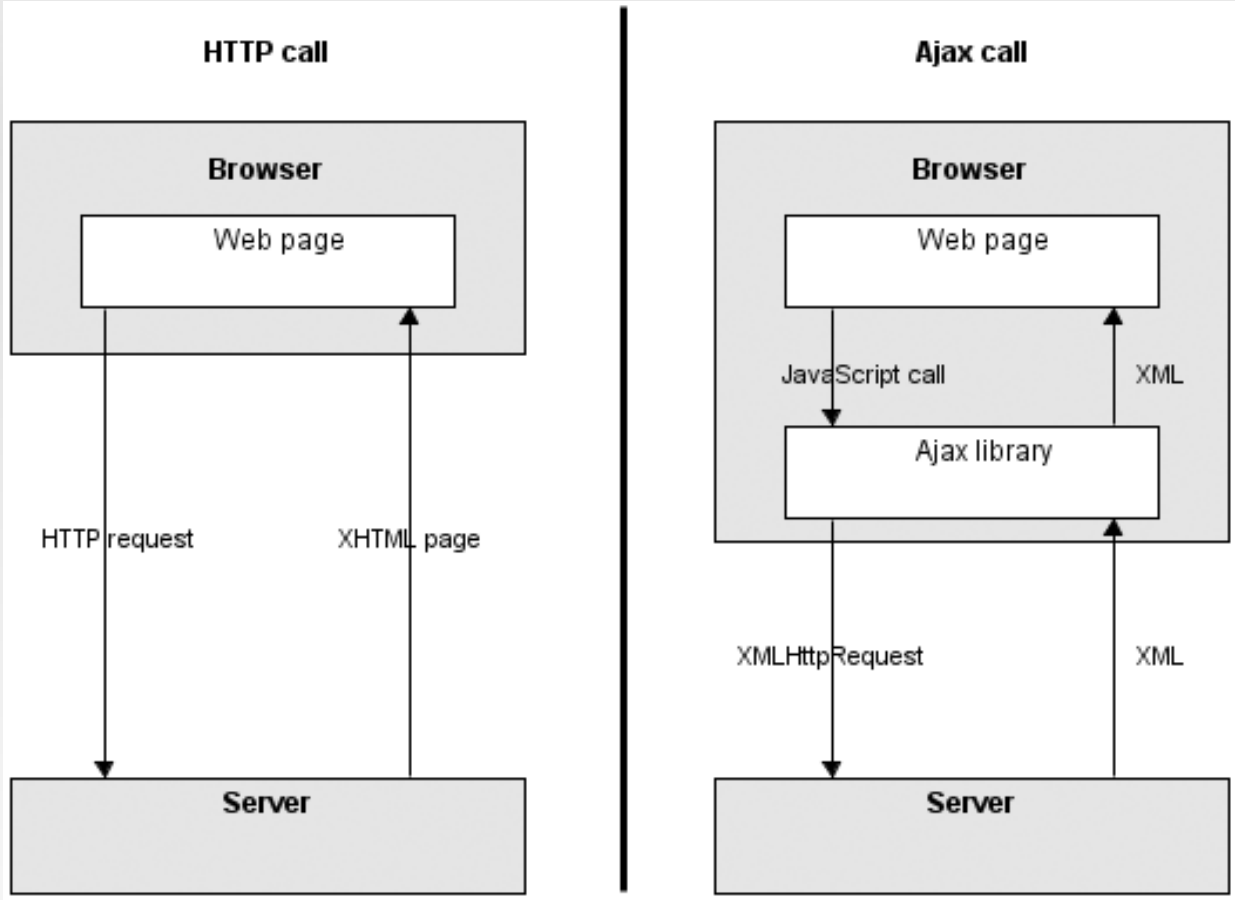
Ajax

- Le protocole HTTP est basé sur un mécanisme de requête/réponse initié par le client
- Quand le client a besoin d'une donnée (image, page, …) il interroge le serveur
- Pour augmenter la réactivité des pages, il est possible avec Ajax de ne pas demander toute une page si une partie seulement doit être mise à jour
- Ajax signifie Asynchronous JavaScript and XML

Ajax

- Une fois les données récupérées avec Ajax, les portions du DOM de la page concernée sont mises à jour
- Il y a aussi un mécanisme appelé Reverse Ajax (ou programmation Comet) pour ”pousser” des données du serveur vers le client
- Ces mécanismes font partie du quotidien sur le web et ont été intégrés à JSF 2.0

Concepts généraux



Concepts généraux

- En principe, Ajax est basé sur :
  - XHTML et CSS pour la présentation
  - DOM pour l'affichage dynamique et l'intéraction avec les données
  - XML et XSLT pour la mise à jour et la manipulation de données XML
  - L'objet XMLHttpRequest pour les communications asynchrones
  - JavaScript pour assembler ces technologies

Support dans JSF

- Les précédentes versions de JSF n'offraient aucune intégration native de Ajax et il fallait utiliser des librairies tierces, augmentant la complexité du code et pénalisant parfois les performances
- Dans JSF 2.0, les choses sont plus simples puisque Ajax a été intégré sous forme d'une librairie JavaScript (jsf.js)

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

Support dans JSF

- La fonction utilisée dans les pages web est :

jsf.ajax.request(ELEMENT, |event|, { |OPTIONS| });

  - ELEMENT : un composant JSF ou XHTML
  - event : événement supporté par l'élément (onmousedown, onclick, onblur, …)
  - OPTIONS : un tableau qui peut contenir :
    - execute : liste d'IDs de composants à envoyer au serveur
    - render : liste d'IDs de composants à mettre à jour

```
<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}" />
```

Exemple

Create a new book

ISBN : 256-6-56

Title : Dune

Price : 23.25

Description : The trilogy

Number of pages : 529

Illustrations : ☐

Create a book

List of the books

ISBN	Title	Price dollar	Description	Number Of Pages	Illustrations
1234-234	H2G2	12.0	Scifi IT book	241	false
564-694	Robots	18.5	Best seller	317	true

APress - Beginning Java EE 6

Exemple

```
<!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>Create a new book</title>
  </h:head>
  <h:body>
    <h:outputScript name="jsf.js" library="javax.faces" target="head"/>
    <h1>Create a new book</h1>
    <hr/>
    <h:form id="form" prependId="false">
     <table border="0">
      <tr>
        <td><h:outputLabel value="ISBN : "/></td>
        <td><h:inputText id="isbn" value="#{bookCtrl.book.isbn}"/></td>
      </tr>
      <tr>
          <td><h:outputLabel value="Title :"/></td>
          <td><h:inputText id="title" value="#{bookCtrl.book.title}"/></td>
      </tr>
      <tr>
        <td><h:outputLabel value="Price : "/></td>
        <td><h:inputText id="price" value="#{bookCtrl.book.price}"/></td>
      </tr>
      <tr>
        <td><h:outputLabel value="Description : "/></td>
        <td><h:inputTextarea id="description" ➥
              value="#{bookCtrl.book.description}" cols="20" rows="5"/></td>
      </tr>
      <tr>
        <td><h:outputLabel value="Number of pages : "/></td>
        <td><h:inputText id="nbOfPage" ➥
                    value="#{bookCtrl.book.nbOfPage}"/></td>
      </tr>
      <tr>
        <td><h:outputLabel value="Illustrations : "/></td>
        <td><h:selectBooleanCheckbox id="illustrations" ➥
                    value="#{bookCtrl.book.illustrations}"/></td>
      </tr>
    </table>
    <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="#{bookCtrl.doCreateBook}" />
  </h:form>
```

Exemple

```
<hr/>
<h1>List of the books</h1>
<hr/>
<h:dataTable id="booklist" value="#{bookCtrl.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>
  <h:column>
    <f:facet name="header">
      <h:outputText value="Price dollar"/>
    </f:facet>
    <h:outputText value="#{bk.price}"/>
  </h:column>
  <h:column>
    <f:facet name="header">
      <h:outputText value="Description"/>
    </f:facet>
    <h:outputText value="#{bk.description}"          />
  </h:column>
  <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>
 <i>APress - Beginning Java EE 6</i>
</h:body>
</html>
```