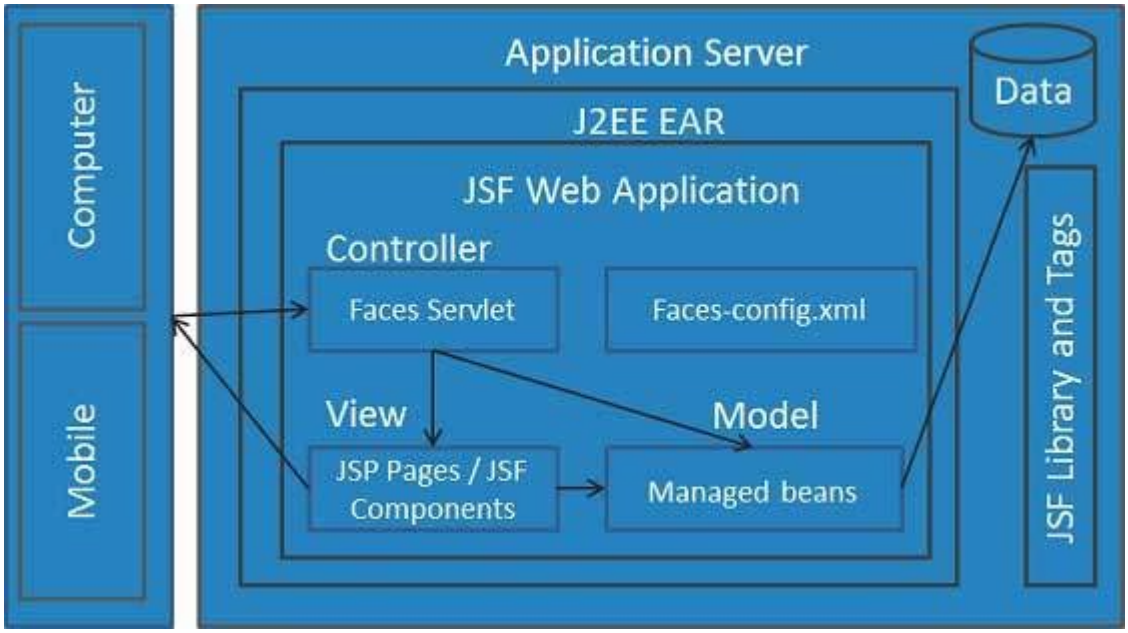
I. Programmation Java JS2

Table des matières

II. Présentation et architecture

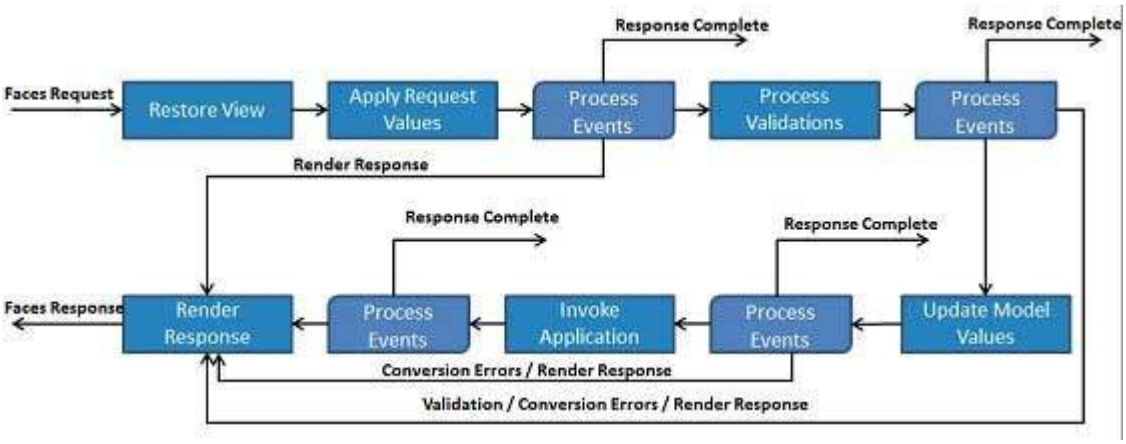
1. Architecture MVC et implémentation JSF

L'architecture Model View Controller (MVC) sépare la logique metier de la présentation. On parle de MVC Design Pattern.



Phases :

- Restoration de la vue: au click, une requete est émise et les validateurs+événements sont mis dans le FacesContext
- Application des valeurs de requete : les valeurs sont extraites de la requête par les composants
- Traitement de la validation : application des validateurs enregistrés
- Mises à jour des valeurs du model : les composants sont parcourues et les beans mises à jour
- Invocation application : submit + liaison à une autre page
- Rendu en réponse : Le contenu est ajouté au JSP



2. Etat actuel de la norme JSF

Versions de JSF / JSR 3445

- JSF 1.0
- JSF 1.1
- JSF 1.2
- JSF 2.0
- JSF 2.1
- JSF 2.2

3. Environnement de développement

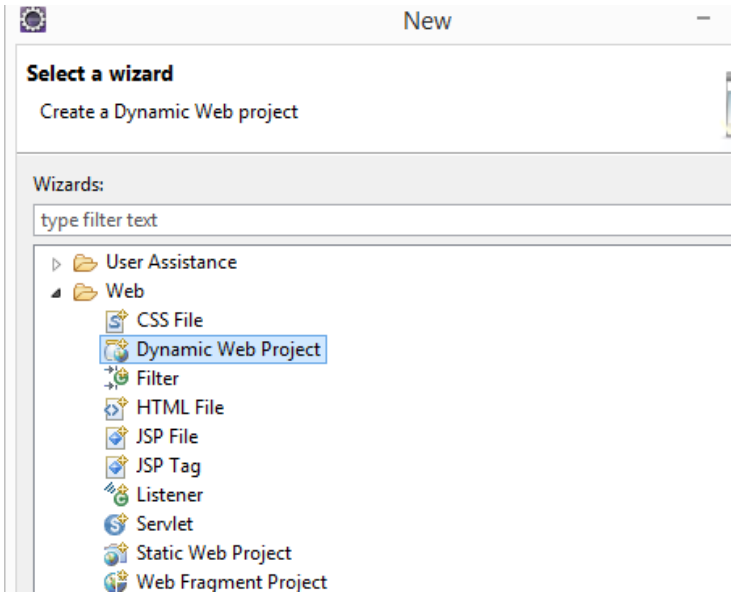
Prérequis

- Java JDK 1.8
- Tomcat 8.3
- Librairie JSF 2.0
- Elipse Oxygen

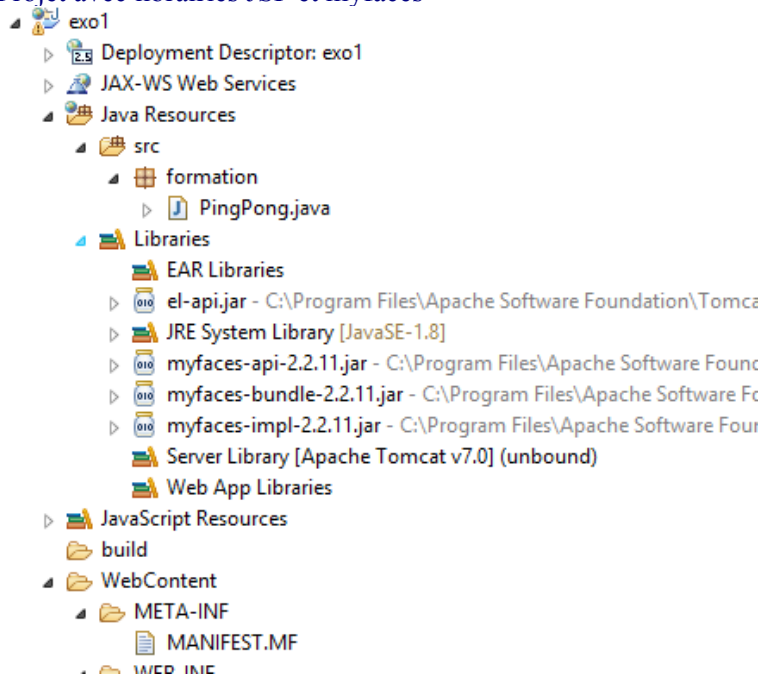
La librairie Jsf2 est importée simplement dans les projets dynamiques Eclipse

- On télécharge la librairie Jsf2 puis l'importation se fait dans le buidpath
- Ou bien utiliser maven/POM (.m2)

Projet Eclipse



Projet avec librairies JSF et myfaces



Contenu du pom.xml

Pom.xml si on utilise maven

```
<dependencies>
<dependency>
    <groupId>com.sun.faces</groupId>
    <artifactId>jsf-api</artifactId>
    <version>2.1.7</version>
</dependency>
<dependency>
```



```
        <groupId>com.sun.faces</groupId>
        <artifactId>jsf-impl</artifactId>
        <version>2.1.7</version>
    </dependency>
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jstl</artifactId>
        <version>1.2</version>
    </dependency>
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>servlet-api</artifactId>
        <version>2.5</version>
    </dependency>

    <dependency>
        <groupId>javax.servlet.jsp</groupId>
        <artifactId>jsp-api</artifactId>
        <version>2.1</version>
    </dependency>
                <!-- Tomcat 6 need this -->
    <dependency>
        <groupId>com.sun.el</groupId>
        <artifactId>el-ri</artifactId>
        <version>1.0</version>
    </dependency>
    </dependencies>
```

4. Tour d'horizon des constituants d'une application JSF

Constituants de la framwork

- Un ensemble d'APIs pour la représentation et la gestion des composants, de leur état, des évènements, de la validation des entrées
- Deux jeux de composants standards : html et core
- Intégration JSP/JSF pour que jsp sert dans la technologie MVC
- Un modèle évènementiel côté serveur
- Managed-Beans : couche contrôle de JSF
- Unified Expression Language : EL

Composants additionnels et intégration de Ajax:
- Primefaces opensource : composants
- Apache Tomahawk : composants
- RCFaces : composants

5. Utilisation des annotations JSF 2.0

XML vs Annontations

Les configurations XML ont été remplacées par des annotations

Parmétrage avec XML

```
<managed-bean>
    <managed-bean-name>userBean</managed-bean-name>
    <managed-bean-class>com.demo.bean.UserBean</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

Parmétrage avec @

Bean controleur
- @ManagedBean

Annotations de Scope :
- @RequestScoped
- @NoneScoped
- @ViewScoped
- @SessionScoped
- @ApplicationScoped

6. Configuration et déploiement dans un conteneur de servlet

La framwork JSF est intégrée à tomcat (conteneur Servlet) au moyen de web.xml au travers d'un Servlet spécifique. Tomcat aiguillera le traffic http vers cette Servlet à la détection dans l'URL des termes

- /faces
- .xhtml
- .jsf

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
  http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  id="WebApp_ID" version="2.5">

  <welcome-file-list>
    <welcome-file>faces/index.xhtml</welcome-file>
  </welcome-file-list>
  <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>

  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.jsf</url-pattern>
  </servlet-mapping>

  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.faces</url-pattern>
  </servlet-mapping>

  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.xhtml</url-pattern>
  </servlet-mapping>

</web-app>
```

III. Cycle de vie

7.La servlet FacesServlet

Pour les applications JSF, le point d'entrée est la servlet nommée FacesServlet.

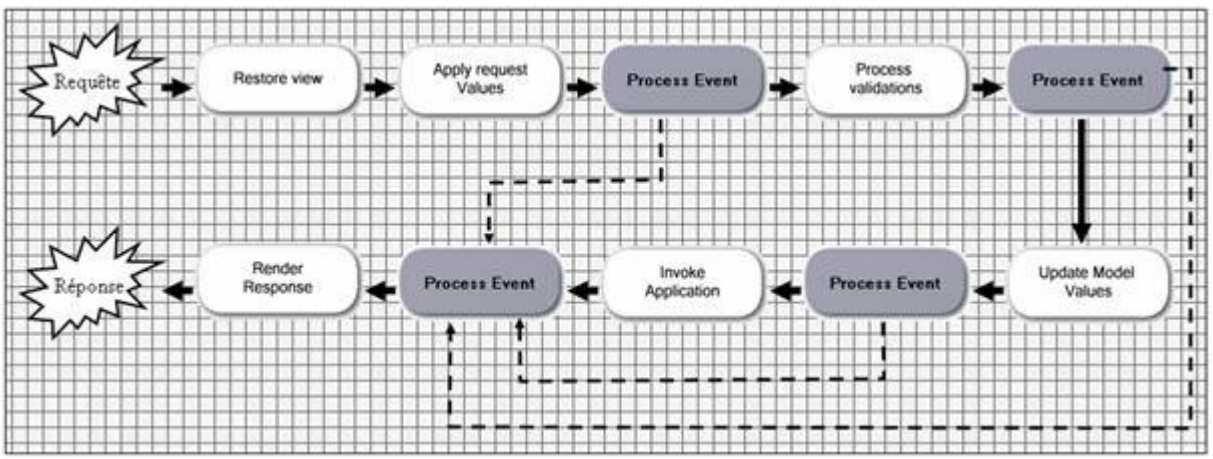
Le fichier web.xml doit être configuré avec le mappage adequate

```
<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>
```

8.Cycle de traitement des pages JSF

Phases , Cycle

- Restore view phase
- Apply request values phase :
- Process validations phase
- Update model values phase
- Invoke application phase
- Render response phase



Apply request values

les valeurs des données sont extraites de la requête HTTP pour chaque composant et sont stockées dans leur composant respectif dans le FaceContext. Les données sont converties en chaines pour rejoindre http

Process validations

Les validators sont appliqués

Update model values

Les valeurs relatives aux composants sont stockées dans le FaceContext

Invoke application

Les événements émis dans la page sont traités. Cette phase doit permettre de déterminer la page résultat qui sera renvoyée dans la réponse en utilisant les règles de navigation définies dans l'application

Render response

cette étape se charge de créer le rendu de la page de la réponse.

9.Les managed-beans

Les managed-beans sont des contrôleurs dans le modèle MVC. Elles assument les échanges avec la View.

Les données et les méthodes du managed-beans sont accessibles dans le view.

Les échanges entre la vue et le controleur se font au moyen de getter/setter.

Expression Language. Appel de pingpongcontroler.getEg()

```
#{pingpongcontroler.eg}
```

Bean managed pour le jeux de ping/pong

```
package formation;

import java.io.Serializable;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;

@ManagedBean(name = "pingpong", eager = true)
@SessionScoped
public class PingPong {
  private String eg = "AAA";
  private String ed = "";
     public PingPong () {
    System.out.println("Instanciation pingpong!");
  }
   public String getEg() {
     return eg;
   }
   public void setEg(String eg) {
     this.eg = eg;
   }
  public String getEd() {
```



```
        return ed;
        }
        public void setEd(String ed) {
        this.ed = ed;
        }
    }
```

Html pour la vue

```
<?xml version="1.0" encoding="UTF-8"?>
<!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:f="http://java.sun.com/jsf/core"
xmlns:h="http://java.sun.com/jsf/html">
<h:head>
<title>Formulaire</title>
</h:head>
<h:body>
<h1>Deplacer le contenu</h1>
<h:form>
GEdit :<h:outputText value="#{pingpongcontroler.eg}" /><br/>
DEdit :<h:outputText value="#{pingpongcontroler.ed}" /><br/>
<h:commandButton type="submit" action="#{pingpongcontroler.gd}"
value="&gt;&gt;&gt;&gt;&gt;"/><br/>
<h:commandButton type="submit" action="#{pingpongcontroler.dg}"
value="&lt;&lt;&lt;&lt;&lt;"/><br/>
</h:form>
</h:body>
</html>
```

10. Les Backing beans(BB)

Les managed beans sont appelés "Backing beans". Ils servent de façade au viewer et implémentent des get/set et des actions que l'ont peut invoquer depuis le viewer

backing beans sont des JavaBeans assurant la logique servant à gérer les datas entre web tier et business tier.

Les backing bean contiennent les propriétés des composants UI .

Pour transmettre des paramètres aux BB on peut utiliser :

- f:param
- f:atribute

param

```
<h:commandButton action="#{user.editAction}">
    <f:param name="action" value="delete" />
</h:commandButton>
```

attribute

```
<h:commandButton action="#{user.editAction}" actionListener="#{user.attrListener}">
    <f:attribute name="action" value="delete" />
</h:commandButton>
```

```
@ManagedBean(name="user")
@SessionScoped
public class UserBean{

  String action;
  //action listener event
  public void attrListener(ActionEvent event){
     action = (String)event.getComponent().getAttributes().get("action");
  }
  public String editAction() {
     //...
  }
}
```

11. Règles de navigation

Les règles de navigations permettent d'enchainer les formulaire par rapport au retour

Configuration pour un formulaire

```
<navigation-rule>
    <from-view-id>/formulaire.jsp</from-view-id>
    <navigation-case>
            <from-outcome>Ok</from-outcome>
            <to-view-id>/ok.jsp</to-view-id>
    </navigation-case>
    <navigation-case>
            <from-outcome>Erreur</from-outcome>
            <to-view-id>/erreur.jsp</to-view-id>
    </navigation-case>
</navigation-rule>
```

Action

```
<h:commandButton value="Valider" action="#{mon_bean.action}" />
```

Dans le Bean

```
public void action() {
    if(this.rempli) return "Ok";
    else return "Erreur";
}
```

12. FacesContext

L'objet FacesContext est créé par JSF avant le traitement des requêtes côté serveur.

L'objet FacesContext contient l'arborescence des composants jsf comme une DOM

```
FacesContext facesContext = arg0.getFacesContext();
    UIViewRoot uiViewRoot = facesContext.getViewRoot();
    if (getCancelButton(uiViewRoot) == null)
            oldViewId = uiViewRoot.getViewId();
}
```

Ajouter un message au contexte

```
FacesContext context = FacesContext.getCurrentInstance();
context.addMessage( null, new FacesMessage( "The message to display in client" ));
```

```
FacesContext.getCurrentInstance().addMessage("newPassword1",
            new FacesMessage(FacesMessage.SEVERITY_ERROR, "Error Message"));
```

13. Validateurs et convertisseurs de données

Convertisseurs

- f:convertNumber
- f:convertDateTime

14. Validateurs standards et spécifiques

- f:validateLength
- f:validateLongRange
- f:validateDoubleRange
- f:validateRegex

Personnalisé

```
public class UrlValidator implements Validator {
    @Override
    public void validate(FacesContext facesContext,
      UIComponent component, String value) throws ValidatorException {
      ...
    }
}
```

Annotation

```
@FacesValidator("formation.UrlValidator")
public class UrlValidator implements Validator {
}
```

15. Evénements JSF

Les événements permettent de provoquer des traitements côté serveur depuis l'interface xhtml

Evénements :
- valueChangeLstener
- actionListener
- Application Events

16. Listener et PhaseListener

Tracer les phases listener

```
public class LogPhaseListener implements PhaseListener {
   public long startTime
   private static final LogProvider log = Logging.getLogProvider(LogPhaseListener.class);
   public void afterPhase(PhaseEvent event) {
      if (event.getPhaseId() == PhaseId.RENDER_RESPONSE) {
         long endTime = System.nanoTime();
         long diffMs = (long) ((endTime - startTime) * 0.000001);
         if (log.isDebugEnabled()) {
            log.debug("Execution Time = " + diffMs + "ms");
         }
      }
      if (log.isDebugEnabled()) {
         log.debug("Executed Phase " + event.getPhaseId());
      }
   }

   public void beforePhase(PhaseEvent event) {

      if (event.getPhaseId() == PhaseId.RESTORE_VIEW) {
         startTime = System.nanoTime();
      }
   }

   public PhaseId getPhaseId() {
      return PhaseId.ANY_PHASE;
   }
}
```

LogPhaseListener

faces-config.xml

```
<lifecycle>
  <phase-listener>package.name.LogPhaseListener</phase-listener>
```

```
</lifecycle>
```


IV. Composants et affichage

17. Facelets

Eléments :
- Librairie de composants
- jsf-facelets.jar
- web.xml
- faces-config.xml

web.xml

```
<web-app>
 :
 <context-param>
     <param-name>javax.faces.DEFAULT_SUFFIX</param-name>
     <param-value>.xhtml</param-value>
 </context-param>
 ..
</web-app>
```

faces-config.xml

```
<faces-config>
 ..
 <application>
   <view-handler>com.sun.facelets.FaceletViewHandler</view-handler>
 </application>
</faces-config>
```

18. Evaluations avec EL

Permet d'évaluer des opérations côté client ou bien évoquer des setter/getter ainsi que de méthodes côté managed-bean

Syntaxe

```
#{operation-expression}
```

```
<h:form>
GEdit :<h:outputText value="#{pingpongcontroler.eg}" /><br/>
DEdit :<h:outputText value="#{pingpongcontroler.ed}" /><br/>
<h:commandButton type="submit" action="#{pingpongcontroler.gd}"
value="&gt;&gt;&gt;&gt;&gt;"/><br/>
<h:commandButton type="submit" action="#{pingpongcontroler.dg}"
value="&lt;&lt;&lt;&lt;&lt;"/><br/>
```

```
</h:form>
```

Managed Bean

```
@ManagedBean(name = "pingpong", eager = true)
@SessionScoped
public class PingPong {
  private String eg = "AAA";
  private String ed = "";
     public PingPong () {
    System.out.println("Instanciation pingpong!");
  }
     public String getEg() {
              return eg;
     }
     public void setEg(String eg) {
              this.eg = eg;
     }
     public String getEd() {
              return ed;
     }
     public void setEd(String ed) {
              this.ed = ed;
     }
}
```

19. Templating avec facelets

On crée un template ui:composition

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://xmlns.jcp.org/jsf/html"
    xmlns:ui="http://java.sun.com/jsf/facelets">

   <ui:composition template="/template/layout.xhtml">

     <!--redéfinition du titre-->
     <ui:define name="title">Connexion</ui:define>
     <!--redéfinition du header pour le faire disparaitre -->
     <ui:define name="header"/>
     <!--idem pour le footer-->
     <ui:define name="footer"/>
     <!--ajout d'une feuille de style pour cette page uniquement-->
     <ui:define name="css">
        <h:outputStylesheet name="css/login.css"/>
     </ui:define>
```

```
        <!--redéfinition du contenu-->
        <ui:define name="content">
            <h:outputText value="Le contenu de la page login" class="login-content"/>
        </ui:define>
    </ui:composition>
</html>
```

20. Composition de composants

Tags :
- <ui:insert>
- <ui:define>
- <ui:include>
- <ui:composition>

21. Les composants JSF de base ("JSF Core Tags")

- UICommand
- UIForm
- UIGraphic
- UIInput
- UIOutput
- UIPanel
- UIParameter
- UISelectItem
- UISelectItems
- UISelectBoolean
- UISelectMany
- UISelectOne

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

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

22. Internationalisation

faces-config.xml

```
<application>
  <locale-config>
    <default-locale>en</default-locale>
    <supported-locale>fr</supported-locale>
  </locale-config>
  <resource-bundle>
    <base-name>formation.messages</base-name>
    <var>msg</var>
  </resource-bundle>
</application>
```

Interface

```
    <h:outputText value = "#{msg['greeting']}" />
```


V. Composants et affichage avancés

23. Les principaux Frameworks de composants JSF

24. La librairie PrimeFaces

Primefaces est une librairie open source de composants graphiques pour les applications JSF (Java Server Faces)

- Il s'agit d'un ensemble de plus de 100 composants graphiques
- Réaliser des applications temps réel avec l'utilisations de l'API WebSocket
- les composants intègrent pleinement les fonctionnalités AJAX et HTML5, sont responsives et compatibles avec la plupart des navigateurs modernes
- Primefaces existe aussi en version mobile

25. L'intégration native d'Ajax avec JSF 2.0

Index.html

```
<?xml version="1.0" encoding="UTF-8"?>
<!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:f="http://java.sun.com/jsf/core"
xmlns:h="http://java.sun.com/jsf/html">
<h:head>
<title>Formulaire</title>
</h:head>
<h:body>
<h1>Deplacer le contenu</h1>
<h:form>
Compteur : <h:inputText id='idcompteur' value="#{compteur.valeur}" /><br/>
<h:commandButton value="Ok">
<f:ajax execute="@form" render="@form" listener="#{compteur.incrementer}"/>
</h:commandButton>
</h:form>
</h:body>
</html>
```

Traitement asynchrone avec Ajax

```
package formation;

import javax.faces.bean.ManagedBean;
```

```
import javax.faces.bean.SessionScoped;
import javax.faces.event.AjaxBehaviorEvent;

@ManagedBean (name = "compteur")
@SessionScoped
public class Compteur{
   private int valeur = 0;

      public int getValeur() {
               return valeur;
      }

      public void setValeur(int valeur) {
               this.valeur = valeur;
      }

      public void incrementer (AjaxBehaviorEvent event)
      {
                System.out.println (++valeur);
      }
}
```

26. Personnalisation de composants

```
<?xml version = "1.0" encoding = "UTF-8"?>
<!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"
  xmlns:composite = "http://java.sun.com/jsf/composite">

  <composite:interface>
    <composite:attribute name = "usernameLabel" />
    <composite:attribute name = "usernameValue" />
    <composite:attribute name = "passwordLabel" />
    <composite:attribute name = "passwordValue" />
    <composite:attribute name = "loginButtonLabel" />
    <composite:attribute name = "loginButtonAction"
      method-signature = "java.lang.String login()" />
  </composite:interface>
```

```
    <composite:implementation>
      <h:form>
        <h:message for = "loginPanel" style = "color:red;" />
        <h:panelGrid columns = "2" id = "loginPanel">
          #{cc.attrs.usernameLabel} :
          <h:inputText id = "username" value = "#{cc.attrs.usernameValue}" />
          #{cc.attrs.passwordLabel} :
          <h:inputSecret id = "password" value = "#{cc.attrs.passwordValue}" />
        </h:panelGrid>
        <h:commandButton action = "#{cc.attrs.loginButtonAction}"
          value = "#{cc.attrs.loginButtonLabel}"/>
      </h:form>
    </composite:implementation>
</html>

package formation;

import java.io.Serializable;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {
  private static final long serialVersionUID = 1L;
  private String name;
  private String password;

  public String getName() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }

  public String getPassword() {
    return password;
  }

  public void setPassword(String password) {
    this.password = password;
```

```
    }

    public String login() {
      return "result";
    }
  }
```

```
<?xml version = "1.0" encoding = "UTF-8"?>
<!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"
  xmlns:tp = "http://java.sun.com/jsf/composite/formation">

  <h:head>
    <title>JSF tutorial</title>
  </h:head>
  <h:body>
    <h2>Custom Component Example</h2>
    <h:form>
    <tp:loginComponent
      usernameLabel = "Enter User Name: "
      usernameValue = "#{userData.name}"
      passwordLabel = "Enter Password: "
      passwordValue = "#{userData.password}"
      loginButtonLabel = "Login"
      loginButtonAction = "#{userData.login}" />
    </h:form>
  </h:body>
</html>
```

27. Création de composants

Pour créer un composant , il faut coder en Java .

- Annotation @FacesComponent
- Héritage de UIComponentBase
- Redéfinition de traitements avec @Override

```
import java.io.IOException;
import javax.faces.context.FacesContext;
import javax.faces.component.FacesComponent;
import javax.faces.component.UIComponentBase;
import javax.faces.context.ResponseWriter;
```

```
@FacesComponent("OutputTitle")
public class OutputTitle extends UIComponentBase {

	private String label;

	public String getLabel() {
		return label;
	}

	public void setLabel(String label) {
		this.label = label;
	}

	@Override
	public String getFamily() {
		return "fr.mon.projet";
	}

	@Override
	public void encodeBegin(FacesContext context) throws IOException {
		ResponseWriter writer = context.getResponseWriter();
		writer.write("<h1>");
		if (label != null) {
			writer.write(label);
		}
		writer.write("</h1>");
	}
 }

}
```

Déploitement avec web.xml

```
<context-param>
	<param-name>javax.faces.FACELETS_LIBRARIES</param-name>
	<param-value>/WEB-INF/monprojet-taglib.xml</param-value>
</context-param>
```

```
 <?xml version="1.0" encoding="UTF-8"?>
<facelet-taglib 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-facelettaglibrary_2_0.xsd"
   version="2.0">
```

```
    <namespace>http://fr.mon.projet/lib</namespace>
    <tag>
      <tag-name>outputTitle</tag-name>
      <component>
        <component-type>OutputTitle</component-type>
      </component>
    </tag>
    <tag>
      <tag-name>simpleOutputTitle</tag-name>
      <source>simpleOutputTitle.xhtml</source>
    </tag>
  </facelet-taglib>
```

fichier xhtml

```
  <html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:t="http://fr.mon.projet/lib">
    <f:view>
      <t:outputTitle label="hello"/>
      <t:simpleOutputTitle label="hello"/>
    </f:view>
  </html>
```

VI. Conception avec JSF

28. Gestion d'état avec JSF

JasperReports une librairie de reporting open source

Utilise des données en entrées sous divers formats et produit un reporting sous divers formats (HTML, PDF, RTF, EXCEL).

- title
- pageHeader
- columnHeader
- detail
- columnFooter
- pageFooter

```
InputStream reportTemplate =
this.getClass().getClassLoader().getResourceAsStream("quote_orderform.jrxml");
Map<String, Object> parameters = new HashMap<String, Object>();
//
parameters.put(JRJpaQueryExecuterFactory.PARAMETER_JPA_ENTITY_MANAGER,entityManager;

JasperReport jasperReport;
JasperPrint jasperPrint;
JasperDesign jasperDesign;

parameters.put("param_01", "value of your param");
parameters.put("param_02", "value of your param");
parameters.put("param_03", "value of your param");

byte[] pdf = null;

try {
    jasperDesign = JRXmlLoader.load(reportTemplate);
    jasperReport = JasperCompileManager.compileReport(jasperDesign);
    jasperPrint = JasperFillManager.fillReport(jasperReport,
    parameters, new JREmptyDataSource());
    pdf = JasperExportManager.exportReportToPdf(jasperPrint);
} catch (JRException e) {
    //handle error
}
```

```
FacesContext context = FacesContext.getCurrentInstance();
HttpServletResponse response = (HttpServletResponse) context.getExternalContext().getResponse();

try {
    OutputStream os = response.getOutputStream();
    response.setContentType("application/pdf"); // fill in
    // contentType
    response.setContentLength(pdf.length);
    response.setHeader("Content-disposition", "attachment; filename=\""+ "_nameOFile.pdf\"");
    os.write(pdf); // fill in bytes
    os.flush();
    os.close();
    context.responseComplete();
} catch (IOException e) {
    //handle error
}
```

Optimisation de la gestion d'état serveur

```
public void prepareJasperReport(){
   Map<String, String> columnNameNTypeMap = new HashMap<String, String>();
   Connection connection = null;
   try {
      Class.forName("com.mysql.jdbc.Driver");
      connection = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/DB_NAME", "USER", "PASSWORD");
      ResultSet rsColumns = null;
      DatabaseMetaData meta = connection.getMetaData();
      rsColumns = meta.getColumns(null, null, "TABLE_NAME", null);
      while (rsColumns.next()) {
         columnNameNTypeMap.put(rsColumns.getString("COLUMN_NAME"),
rsColumns.getString("TYPE_NAME"));
      }} catch (SQLException e) {
      e.printStackTrace();
      return;
   } catch (ClassNotFoundException e) {
      e.printStackTrace();
      return;
   }

   // a new report
   JasperReportBuilder report = DynamicReports.report();
   // populating new report with TABLE object
   report.setDataSource("select * from TABLE_NAME;", connection);
```

```
    // creating COLUMNS // add extra datatypes if your table have ex. long, float etc
    for (Map.Entry<String, String> entry : columnNameNTypeMap.entrySet()){
      if(entry.getValue().equalsIgnoreCase("int")){
        report.columns(Columns.column(entry.getKey(), entry.getKey(), DataTypes.integerType()));
      }else if(entry.getValue().equalsIgnoreCase("varchar")){
        report.columns(Columns.column(entry.getKey(), entry.getKey(), DataTypes.stringType()));
      }else if(entry.getValue().equalsIgnoreCase("bit")){
        report.columns(Columns.column(entry.getKey(), entry.getKey(), DataTypes.booleanType()));
      }else if(entry.getValue().equalsIgnoreCase("datetime") || entry.getValue().equalsIgnoreCase("date"))
{
        report.columns(Columns.column(entry.getKey(), entry.getKey(), DataTypes.dateType()));
      }

    }

    report.title(Components.text("Summary Report").setHeight(40)
        .setStyle(DynamicReports.stl.style()
        .setBold(true).setFontSize(16).setForegroundColor(Color.BLUE)
        .setAlignment(HorizontalAlignment.CENTER, VerticalAlignment.MIDDLE)));

    report.setColumnTitleStyle(DynamicReports.stl.style().setBold(true));

  report.setColumnStyle(DynamicReports.stl.style().setHorizontalAlignment(HorizontalAlignment.LEFT)
  );
    report.setHighlightDetailEvenRows(true);
    report.pageFooter(Components.pageXofY());

    try {
      // show the report
      report.show(false);
      // export the report to a pdf file
      //report.toPdf(new FileOutputStream("d://report.pdf"));
    } catch (DRException e) {
      e.printStackTrace();
    } /*catch (FileNotFoundException e) {
      e.printStackTrace();
    }*/
  }
```

View

```
  <?xml version='1.0' encoding='UTF-8' ?>
  <!DOCTYPE html>
  <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>Report</title>
  </h:head>
  <h:body>
    <f:view>
      <h:form id="form" target="_blank">
        <h:outputLabel>Export Report</h:outputLabel>
        <h:selectOneRadio value="#{reportsBean.exportOption}">
          <f:selectItem itemValue="PDF" itemLabel="PDF"/>
          <f:selectItem itemValue="HTML" itemLabel="HTML"/>
          <f:selectItem itemValue="EXCEL" itemLabel="EXCEL"/>
          <f:selectItem itemValue="RTF" itemLabel="RTF"/>
        </h:selectOneRadio>
        <h:commandButton  action="#{reportsBean.execute}" value="Get Report" />
      </h:form>
    </f:view>
  </h:body>
</html>
```

29. Optimisation du client avec HTML5

JSF ne prend pas en charge directement les nouveaux attributs de HTML5

Dans la version JSF 2.2 il est possible d'ajouter les attributs libres en direction du client au moyen de pass-through attributions

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:p="http://xmlns.jcp.org/jsf/passthrough">

  <head>
    <title>JSF 2.2 HTML5 Support</title>
  </head>
  <body>
    <h:form>
      <h:inputText value="#{myBean.value}" p:placeholder="Enter text" p:autofocus="true"/>
    </h:form>
  </body>
</html>
```

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://xmlns.jcp.org/jsf/core">
```

```
    <head>
      <title>JSF 2.2 HTML5 Support</title>
    </head>

    <body>
      <h:form>
        <h:inputText value="#{myBean.value}">
          <f:passThroughAttribute name="placeholder" value="Enter text" />
        </h:inputText>
      </h:form>
    </body>
</html>
```

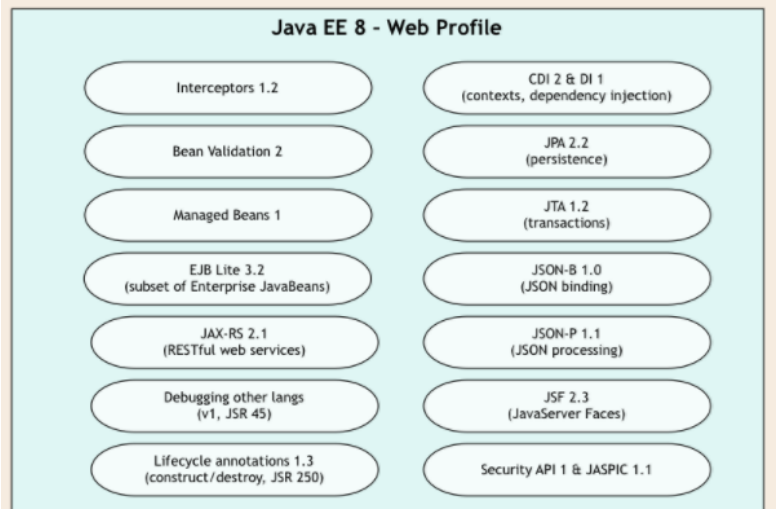
```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:jsf="http://xmlns.jcp.org/jsf">

    <head>
      <title>HTML(5) Friendly Markup in JSF 2.2</title>
    </head>
    <body>
      <form jsf:id="myForm" jsf:prependId="false">
        <h:messages/>
        <table border="0">
          <tr>
            <td><label jsf:for="email">E-Mail</label></td>
            <td>
              <input jsf:id="email" type="text" jsf:value="#{myBean.email}"
jsf:validator="myValidator" jsf:size="30" />
            </td>
          </tr>
        </table>
        <input type="submit" value="Submit"/>
      </form>
    </body>
</html>
```

30. JSF et Web Profile

WebProfile est une version allégée des API de la JEE qui est considérée comme “Full Java EE”.

WebProfile définit une version allégée des EJB et d’autres composants de la JEE



31. Intégration avec CDI

Introduction

CDI (Context and Dependency Injection) est la spécification qui standardise le conception de l'injection de dépendance et de context utilisé depuis longtemps avec Spring. Initialement l'injection été faite depuis une configuration XML maintenant de plus en plus les intentions d'injection sont effectuées par annotations.

Le CDI fait partie de la spécification Java EE 6 : annotation : @Inject

Pour lier deux instances d'objets de classes, on le fait par annotation et non plus par programme.

@inject

@Inject peut être mise sur :
- Un champs
- Un setter
- Un constructeur

```
public class A {
   @Inject
   private B b;
}

public class B {
}
```

On trouve également d'annotations pour enrichir les dépendances d'injection
- @Qualifier
- @Retention(RUNTIME)
- @Target({ TYPE, METHOD, FIELD, PARAMETER })
- @interface
- @Named("XX")

```
public class A {
```



```
    @Inject
  @Named("B")
    private B b;
  }

  @Named
  public class B {
  }
```

Les annotations peuvent être cumulées

```
  @Qualifier
  @Retention(RUNTIME)
  @Target({ TYPE, METHOD, FIELD, PARAMETER })
  public class A {
    @Inject
    private B b;
  }
```

Portée

Les dépendances ont une portée Scope

- @RequestScoped
- @SessionScoped
- @ApplicationScoped
- @Dependent

Evénements

Pour qu'un Bean sache quand il est détruit ou créé, on utilise les événements

- @PostConstruct
- @PreDestroy
- @Start

```
  class B {
        @PostConstruct
        void init()
        {
                ...
        }
        @PreDestroy
        void clean()
        {
                ...
        }
  }
```

intercepteurs AOP

On peut mettre en places des événements autour d'une méthode. Pour intercepter les instants d'avant l'appel et après l'appel de la méthode.

- @InterceptorBinding
- @Target({TYPE, METHOD})
- @Retention(RUNTIME)
- public @interface Audited {}

