JAVA ENTERPRISE EDITION.

O .ND

Introduction et Objectif

Java Enterprise Edition, ou **Java EE** (anciennement **J2EE**), est une spécification pour la technique Java d'Oracle plus particulièrement destinée aux applications d'entreprise. Ces applications sont considérées dans une approche multi-niveaux.

Cette édition est dédiée à la réalisation d'applications pour entreprises.

JEE, une plate–forme fortement orientée serveur pour le développement et l'exécution, est basée sur J2SE (Java 2 Standard Edition) qui contient les API de bases de java.

JEE permet une grande flexibilité dans le choix de l'architecture de l'application en combinant les différents composants. Ce choix dépend des besoins auxquels doit répondre l'application mais aussi des compétences dans les différentes API de JEE.

L'architecture d'une application se découpe en au moins trois tiers :

la partie cliente(Présentation) : c'est la partie qui permet le dialogue avec l'utilisateur. Elle peut être composé d'une application standalone, d'une application web ou d'applets, …

la partie métier : correspondant à la mise en œuvre de l'ensemble des règles de gestion et de la logique applicative .

la partie données : correspondant aux données qui sont destinées à être conservées sur la durée, voire de manière définitive.

Les acteurs d'une application J2EE

La réalisation d'une application basée sur l'architecture JEE fait appel à différents types de compétences allant de la conception jusqu'à la supervision de l'application en passant par le développement et le déploiement.

Afin de pouvoir maîtriser ce processus JEE adopte l'approche des partages des responsabilités.

Plus spécifiquement pour les EJB, cette approche définit plusieurs niveaux de responsabilité :

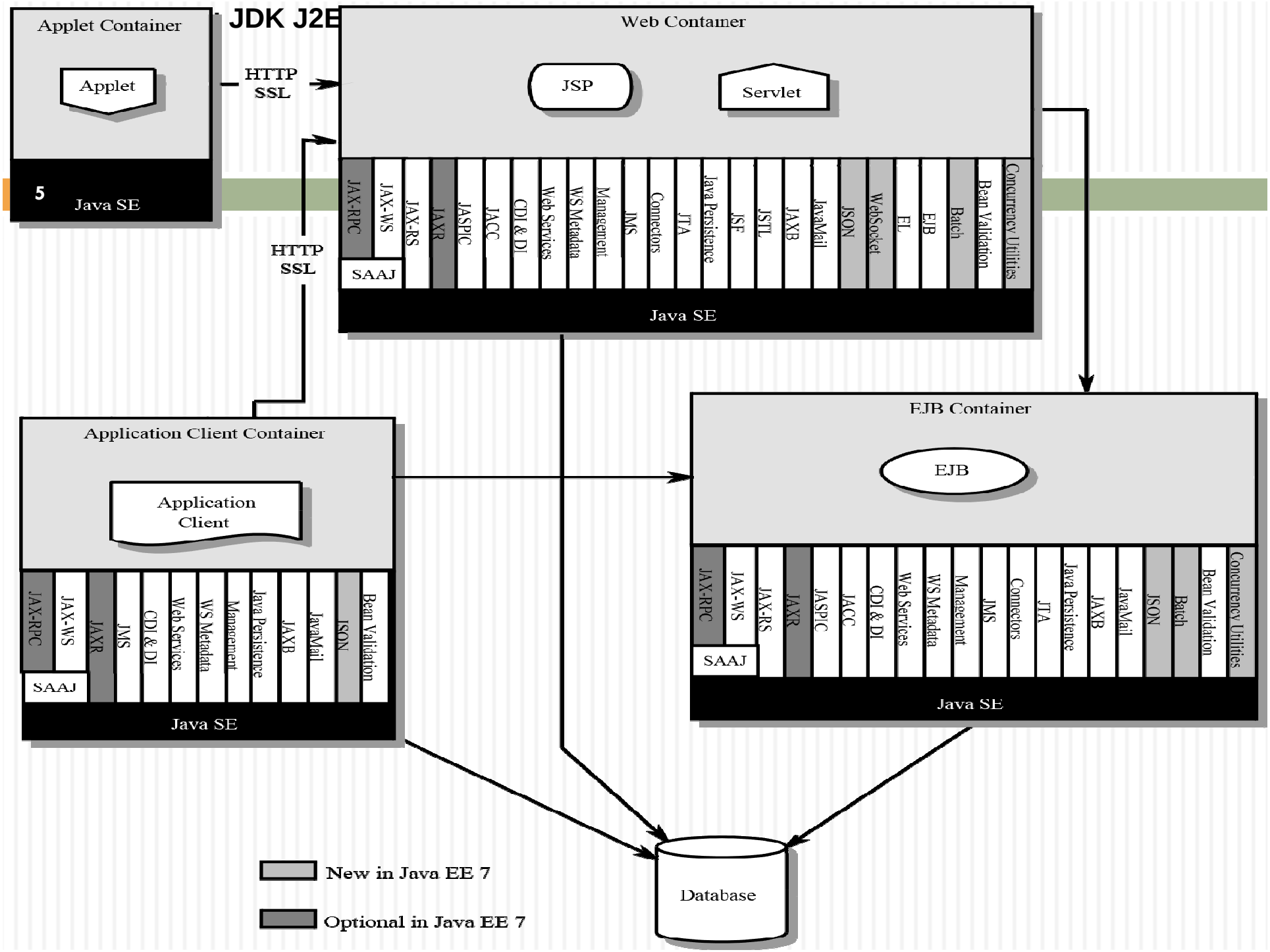
- **Le fournisseur de services ou d'EJB :** c'est l'acteur qui fournit des composants métiers réutilisables soit par l'achat à un fournisseur de composants, soit par développement interne ;
- **L'assembleur d'applications :** l'acteur qui est capable de construire une application à partir d' EJB existants ;
- **Le déployer :** l'acteur qui récupère l'application et s'occupe de son déploiement dans un serveur d'applications ;
- **L'administrateur :** l'acteur qui contrôle le fonctionnement du serveur d'application et assure la

supervision des applications ;

- **Le fournisseur de conteneur :** l'éditeur qui commercialise un conteneur web ou un conteneur EJB;

cet éditeur commercialise souvent un serveur d'application incluant ces conteneurs ;

- **Le fournisseur de serveur :** c'est l'éditeur qui commercialise un serveur d'application (BEA, IBM etc...)


JDK J2E
Applet Container
Applet
HTTP
SSL
5
Java SE
Web Container
JSP
Servlet
JAX-RPC
JAX-WS
JAX-RS
JAXR
JASPIC
JACC
CDI & DI
Web Services
WS Metadata
Management
JMS
Connectors
JTA
Java Persistence
JSF
JSTL
JAXB
JavaMail
JSON
WebSocket
EL
EJB
Batch
Bean Validation
Concurrency Utilities
SAAJ
Java SE
HTTP
SSL
Application Client Container
Application
Client
JAX-RPC
JAX-WS
JAXR
SAAJ
JMS
CDI & DI
Web Services
WS Metadata
Management
Java Persistence
JAXB
JavaMail
JSON
Bean Validation
Java SE
EJB Container
EJB
JAX-RPC
JAX-WS
JAX-RS
JAXR
SAAJ
JASPIC
JACC
CDI & DI
Web Services
WS Metadata
Management
JMS
Connectors
JTA
Java Persistence
JAXB
JavaMail
JSON
Batch
Bean Validation
Concurrency Utilities
Java SE
Database
New in Java EE 7
Optional in Java EE 7

Les principaux blocs de Java EE

- **Servlets**

Les servlets sont des composantes web basés sur java, et qui génèrent du contenu **dynamique**. Elles sont gérées par un container de servlets et permettent d'interagir avec des clients web via requête/réponse.

La dernière version Servlets 3.1 est structurée et décrite dans la **JSR 340**.

Dans la plupart des cas, on utilise une Servlet pour générer dynamiquement des pages appelées par un client dans un navigateur web. Chaque Servlet est associée à un URI, et l'appel de ce dernier depuis le navigateur déclenchera le contenu d'une méthode de votre Servlet.

Si les servlets sont très utiles pour faire du déclenchement de processus métier, elles sont en revanche moins utilisables pour faire de la présentation (retourner du code HTML). En effet, ceci impliquerait d'écrire du code HTML dans des objets String depuis votre classe, et de les renvoyer au client. Peu ergonomique, très complexe à maintenir. C'est là qu'on utilisera les JSP

Les JSP

La **JSP** (Java Server Page) est un composant permettant de créer de manière textuelle des pages web dynamiques. Plus simplement, une JSP permet d'écrire du code de **présentation** (en général du code **HTML**) et d'y affecter des comportements et données dynamiques de façon simple. Cette page sera convertie et compilée par votre web container en… une **servlet**!

Le grand avantage longtemps plébiscité par les créateurs de la JSP est qu'elle permet un bon découpage des rôles. Un développeur frontend sera capable de créer les pages, le look&feel de l'application avec des JSP sans connaitre java, et le développeur java pourra s'occuper de la logique métier en amont sans toucher à la JSP.

Les JSP facilitent et encouragent l'utilisation de composants réutilisables, que ce soit des **JavaBeans** ou des **tags**, pour faciliter la création de widgets, et le passage de données (**data binding**).

La dernière version JSP 2.3 est structurée et décrite dans la **JSR 245**.

La JSTL

La **JSTL** (Java Standard Tag Library) est simplement une librairie de tags (comme son nom l'indique!), destinée à fournir les composants de base pour enrichir vos JSP.

Des exemples? Faire une boucle, des conditions, afficher des attributs passés dans la requête ou dans des variables de session, créer des urls, internationaliser les chaines de caractère, et bien d'autres choses.

La dernière version JSTL 1.2 est structurée et décrite dans la **JSR 52**.

JSF

JSF (Java Server Faces) est un framework Java pour créer des web UI (User Interface) de façon intuitive. Il propose des composants graphiques et un ensemble d'outils qui diminueront grandement le temps de développement.

Le framework (JSF 1.0) permet de faire du quick-start, mais est en réalité ultra **complexe** et requiert une réelle expertise lorsque l'application atteint une taille critique, souvent plus complexe que d'écrire le côte HTML soi même.

En 2009, JSF sort en version 2.0 et adresse certains des points critiques de la version précédente, notamment:

- Une intégration en natif de fonctionnalités AJAX
- La notion de **partial processing** (pas de rafraichissement complet de la page, juste du composant)

- Des déclarations simplifiées par **annotations**
- Création de composants externes simplifiée
- Meilleure gestion des ressources statiques (CSS, js …)
- Cette version redonne espoir à ceux qui utilisaient déjà le framework, et améliore grandement les possibilité de scalabilité. Mais la mauvaise réputation de JSF ne lui permet pas encore de revenir de ses déboires.
- La version 2.2 (**JSR 344**) sortie en 2013 apporte surtout deux nouveautés:
- Un meilleur support de l'HTML5
- Des vues stateless (sans état)

- **Les EJB**

EJB (Enterprise Java Bean) est une spécification qui facilite la distribution de composants (appelés Beans) sur le réseau. EJB sont de trois types différents, qu'il est important de distinguer:

- **Les Session-Beans**: Qu'ils soient avec (**stateful**) ou sans (**stateless**) état, un Session-Bean est un objet qui contient du code métier exécutable par un ou des appelants. Ce sont typiquement des **services** hébergés sur le serveur d'application qui seront appelables par les clients de façon transparente via le réseau. Les clients posséderont uniquement une interface de ce dernier, permettant au code métier d'être conservé et protégé.

Les Entity Beans: Les EJB Entité permettent de gérer des objets ayant vocation à être**persistés**. Utilisés avec des **ORM** (Object Relational Mapping), ils sont un moyen efficace de représenter les **entités** en base de données sous forme d'**objets Java**.

Les Message-Driven Beans: Utilisés pour la publication de messages asynchrones avec JMS. Tous les types d'EJB peuvent évoluer dans un contexte **transactionnel**, et leur cycle de vie est géré de manière **autonome** par le serveur d'application, au sein du conteneur d'EJB (on ne les instancie donc pas nous même). Ils permettent au développeur de s'affranchir de certains aspects techniques complexes, comme la publication de **services** pour des clients distants, la **gestion des connexions** à la base de données ou la **gestion de transactions**.

EJB 3.1 est la dernière version sortie avec JavaEE 7 en 2013. Elle est décrite par la **JSR 345**

Java Persistence

Java Persistence API, aussi appelé JPA, est une interface sortie avec JavaEE 5 en 2006. Elle repose essentiellement sur l'utilisation des annotations (introduites avec Java 5 en 2002), et simplifie grandement le développement des entités qui représentent les objets du monde relationnel (des base de données). JPA permet de définir des tables, colonnes, clés primaires, jointures, et de les apposer sur des attributs de classes pour automatiser toute une partie du mapping entre le monde objet et le monde relationnel (ORM).

Note: JPA s'utilise généralement avec des Entity Bean.

La dernière version 2.1 de JPA date de 2013, est décrite par la JSR 338

Environnement de Développement

JEE fournit un ensemble d'API permettant de construire des sites WEB avec la technologie JAVA.

Pour cela, on utilise un serveur d'application JEE:

Un serveur d'application permet de charger et d'exécuter les servlets, EJB, …dans une JVM.

C'est une extension du serveur web.

Ce serveur d'application contient entre autre un moteur de servlets qui se charge de manager les servlets qu'il contient.

L'utilisation d'un IDE (Integrated Development Environment) est obligatoire comme Eclipse ou JBuilder, ou NetBeans.

Dans ce cours, on utilisera NetBeans, vu sa gratuité, sa robustesse et son extensibilité.

- Serveur d'applications

Le serveur d'application est l'environnement d'exécution des applications côté serveur. Il prend en charge l'ensemble des fonctionnalités qui permettent à N clients d'utiliser une même application :

Les serveurs d'application se sont développés depuis la création de J2EE. On peut distinguer principalement 2 grandes catégories de serveurs :

- Open Source : évolue grâce à la communauté
- Propriétaire : évolue selon l'éditeur

Chaque catégorie a ses avantages et ses inconvénients. Nous allons décrire les serveurs les plus connus afin d'avoir une vision globale des solutions disponibles.

Gratuit: Tomcat : Apache, Jonas : ObjectWeb, JBoss : Jboss, Glassfish

- WebSphere : IBM
- WebLogic : BEA
- WebObject : Apple
- Oracle Application Server : Oracle

- *Oracle GlassFish Server*

GlassFish est le nom du serveur d'applications Open Source Java EE avec la version 4 qui sert de socle au produit *Oracle GlassFish Server.*

GlassFish est certifié Java EE (EJB 3.1, JPA2, CDI, JSF 2.0, JAX-RS 1.1, ...)

L'un des meilleurs serveurs d'applications Open Source du marché à l'heure actuelle!

- TP 1: Installation et configuration de l'environnement de travail :

- JDK , configuration JAVA_HOME
- Moteur de BD MySQL(EASYPHP v12 +)
- NetBeans (compatible avec JDK choisi)
- Glassfish Server
- « Hello World web Application »

Déploiement d'application web

Généralement, le déploiement d'une application est basé sur un ensemble de règles (en format XML) enregistrées

dans un fichier appelé descripteur de déploiement (web.xml) . il contient les caractéristiques et paramètres de l'application, ce qui inclut la description des servlets utilisées et leurs différents paramètres d'initialisation.

Chaque servlet doit être déclarée dans l'élément XML

<servlet> de la manière suivante :

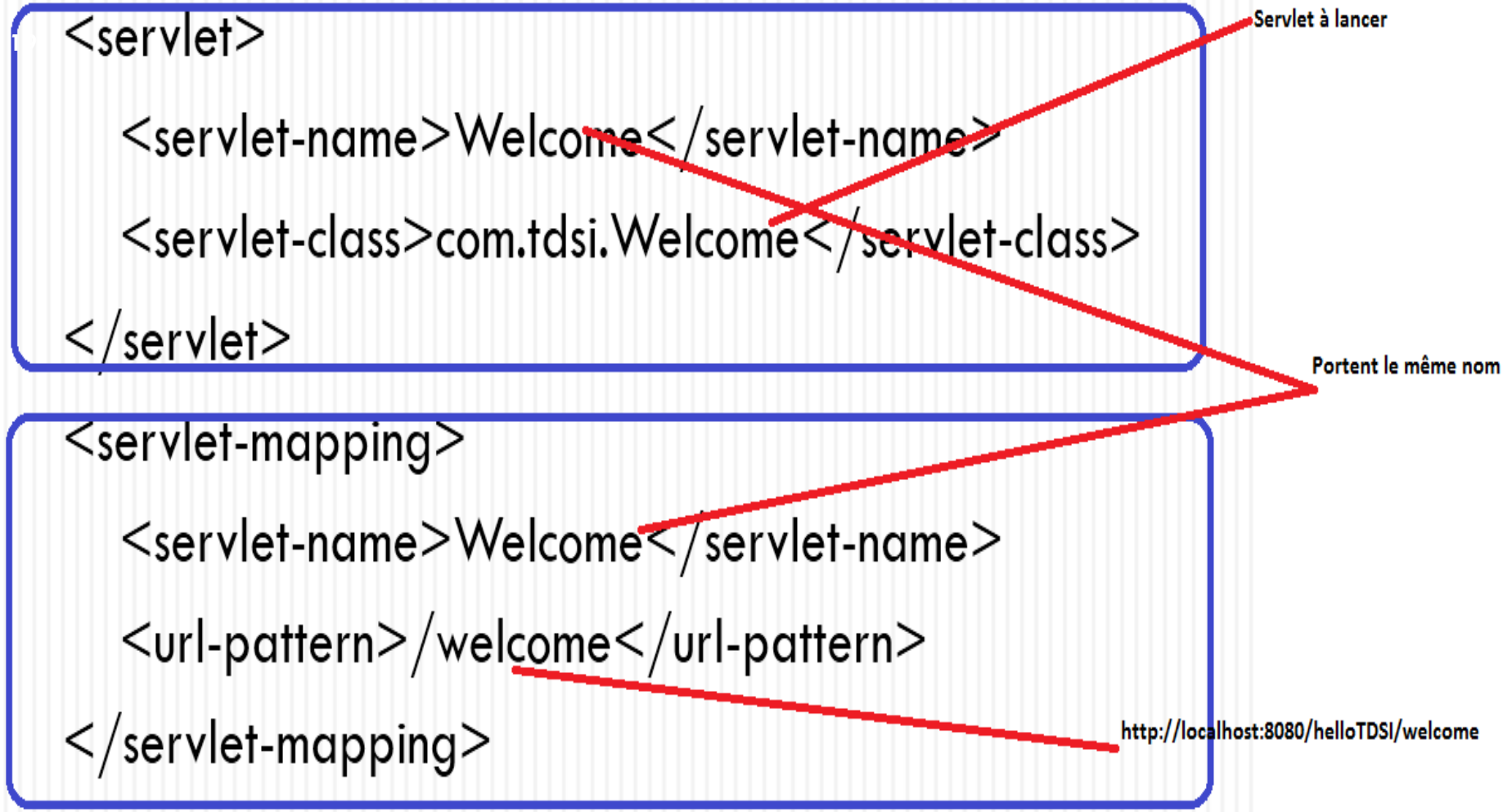
- <servlet-name> est le nom interne de la servlet, nom qui l'identifie de façon unique.
- <servlet-class> est la classe Java associée à la servlet.
- <load-on-startup> demande que la servlet soit chargée dès le démarrage du serveur.

<servlet-mapping> effectue le lien entre l'URL et la servlet.

- <url-pattern> est l'URL permettant de faire le lien avec la servlet.

Exemple

```
<web-app version="3.1">
    <servlet>
        <servlet-name>Welcome</servlet-name>
        <servlet-class>com.tdsi.Welcome</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>Welcome</servlet-name>
        <url-pattern>/welcome</url-pattern>
    </servlet-mapping>
</web-app>
```



- descripteur de déploiement.

Chaque application web possède son propre fichier de configuration, le fichier web.xml, appelé descripteur de déploiement, situé dans le répertoire WEB-INF de l'application. Ref: https://docs.oracle.com/cd/E13222_01/wls/docs81/webapp/web_xml.html

- Les servlets

Une servlet est un programme qui s'exécute côté serveur en tant qu'extension. Elle reçoit une requête du client, elle effectue des traitements et renvoie le résultat. La liaison entre la servlet et le client peut être directe ou passer par un intermédiaire comme par exemple un serveur http.

Ecrite en Java, une servlet en retire ses avantages : la portabilité, l'accès à toutes les API de Java dont JDBC pour l'accès aux bases de données, ...

Une servlet fonctionne selon un modèle client/serveur ou requête/réponse, par conséquent tous

les protocoles utilisant ce modèle pourront être utilisés (http, ftp, etc.)

- Le fonctionnement d'une servlet (cas d'utilisation de http)

Une servlet est chargé par le serveur d'application et exécuté dans la JVM.

- Pour exécuter une servlet, il suffit de saisir une URL qui désigne la servlet dans un navigateur.

1) Le serveur reçoit du navigateur la requête http qui a recours à une servlet
2) Si c'est la première sollicitation de la servlet, le serveur l'instancie. Les servlets sont stockées (sous forme de fichiers .class) dans un répertoire particulier du serveur. Ce répertoire dépend du serveur d'applications utilisé. La servlet reste en mémoire jusqu'à l'arrêt du serveur. Certains serveurs d'applications permettent aussi d'instancier des servlets dès le lancement du serveur.
   Au fil des requêtes, la servlet peut être appelée par plusieurs threads lancés par le serveur. Ce principe de fonctionnement évite d'instancier un objet de type servlet à chaque requête et permet de maintenir un ensemble de ressources actives telles qu'une connexion à une base de données.
3) Le serveur crée un objet qui représente la requête http ainsi que l'objet qui contiendra la réponse et les envoie à la servlet
4) La servlet crée dynamiquement la réponse sous forme de page html transmise via un flux dans l'objet contenant la réponse. La création de cette réponse utilise bien sûr la requête du client mais aussi un ensemble de ressources incluses sur le serveur telles de que des fichiers ou des bases de données.
5) Le serveur récupère l'objet réponse et envoie la page html au client.

- L'API servlet

L'API servlet regroupe un ensemble de classes dans deux packages :

- javax.servlet : contient les classes pour développer des serlvets génériques indépendantes d'un protocole

- javax.servlet.http : contient les classes pour développer des servlets qui reposent sur le protocole http utilisé par les serveurs web.

- Envoyer un contenu HTML à un client Web
- une servlet dérive de la classe HttpServlet

24 public class HelloWorld extends HttpServlet {

- Une requête GET faite à la servlet est traitée par la méthode doGet

public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException

- De même une requête POST faite à la servlet est traitée par la méthode doPost

public void doPost(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException

- Une requête GET ou POST faite à la servlet peut être traitée par la méthode processRequest
- protected void processRequest(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException

- L'objet HttpServletRequest request est l'objet qui nous donne accès à la requête faite par le client Web. La réponse de la servlet sera faite via l'objet HttpServletResponse response
- L'objet response nous permet de fixer les entêtes http qui seront envoyés au client. Par exemple l'entête Content-type: text/html est ici fixé par :

```
response.setContentType("text/html");
```

- Pour envoyer la réponse au client, la servlet utilise un flux de sortie que lui délivre l'objet response :

```
PrintWriter out = response.getWriter();
```

- Récupérer les paramètres envoyés par un client web

On suppose que le client nous envoie par GET , son nom par le paramètre « nom » et son prénom par «prenom»

http://localhost:8080/helloTDSI/welcome?nom=ouz&prenom=Ndiaye

- Les paramètres envoyés par le navigateur sont récupérés de la façon suivante :
  - String nom= request.getParameter("nom");
  - String prenom = request.getParameter("prenom");
- La méthode request.getParameter("nomParamètre") rend le pointeur null, si le paramètre nomParamètre ne fait pas partie de ceux envoyés par le client web.

javax.servlet.http.HttpSession

- public interface HttpSession
- Utilisées pour mémoriser les actions (requêtes HTTP) d'un utilisateur unique au cours du temps

- **Accès à l'objet session dans une Servlet**
  - appel à request.getSession(true|false) pour recevoir un objet HttpSession
- **Extraire les informations associées à une session**
  - appel à Object getAttribute("cle") sur l'objet HttpSession, récupérer la valeur, vérifier si le résultat est nul
- **Mémoriser les informations de la session**
  - utiliser Object setAttribute("cle","valeur") avec une clé et une valeur
- Supprimer les données de la session
  - removeAttribute supprime une valeur
  - invalidate pour annuler une session
  - logout pour déloguer le client de l'application Web

javax.servlet.RequestDispatcher

- public interface RequestDispatcher

Définit un objet qui reçoit les requêtes du client et les renvoie à une ressource(Servlet, page HTML, JSP, …) du server .

- Un objet RequestDispatcher s'obtient grâce à l'appel de la méthode getRequestDispatcher (String path);
  - RequestDispatcher rd = request.getRequestDispatcher ("index").

L'interface RequestDispatcher contient deux méthodes:

- forward(request, response)

Forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.

- include(request, response)

Includes the content of a resource (servlet, JSP page, HTML file) in the response. In essence, this method enables programmatic server-side includes.

- **Personne** p = new **Personne** (matricule, prenom, nom, tel);
- request.setAttribute ("personnel", p);
- javax.servlet.RequestDispatcher disp = request.getRequestDispatcher("/forward_2.html");
- disp.forward(request,response);

Dans "/forward_2.html" :

Personne pers = (**Personne**) request.getAttribute ("personnel");

La classe HttpServlet

Une servlet est une classe qui implémente l'interface Servlet, laquelle définit 5 méthodes qui permettent au conteneur web de dialoguer avec la servlet.

- public void init (ServletConfig config): appelé par le serveur juste après l'instanciation de la servlet, cela permet de charger des ressources tel qu'un fichier ou une connexion vers une base de données
- public void service (ServletRequest req, ServletResponse resp): appelée pour le traitement d'une requête, mais ne peut être invoquée tant que la méthode init ( ) n'a pas terminée. Cette méthode appelle doGet ( ) ou doPost ( )
- public void destroy( ): est appelée juste avant que le serveur ne détruise la servlet, cela permet de libérer des ressources allouées dans la méthode init ( ) tel qu'un fichier ou une connexion vers une base de données

Exercice 1

L'objectif de cet exercice est de mettre en place un Système d'authentification et déconnexion des utilisateurs en s'appuyant sur une BD (login et password).

1. Donner les différentes étapes de l'authentification (algorithme), Faites le diagramme de Séquence
2. Donner les différentes étapes de la déconnexion (algorithme), Faites le diagramme de Séquence
3. Mettre en place la base test , contenant une table **users** avec 3 colonnes (id, login, password(sha1))

Java Server Page(JSP)

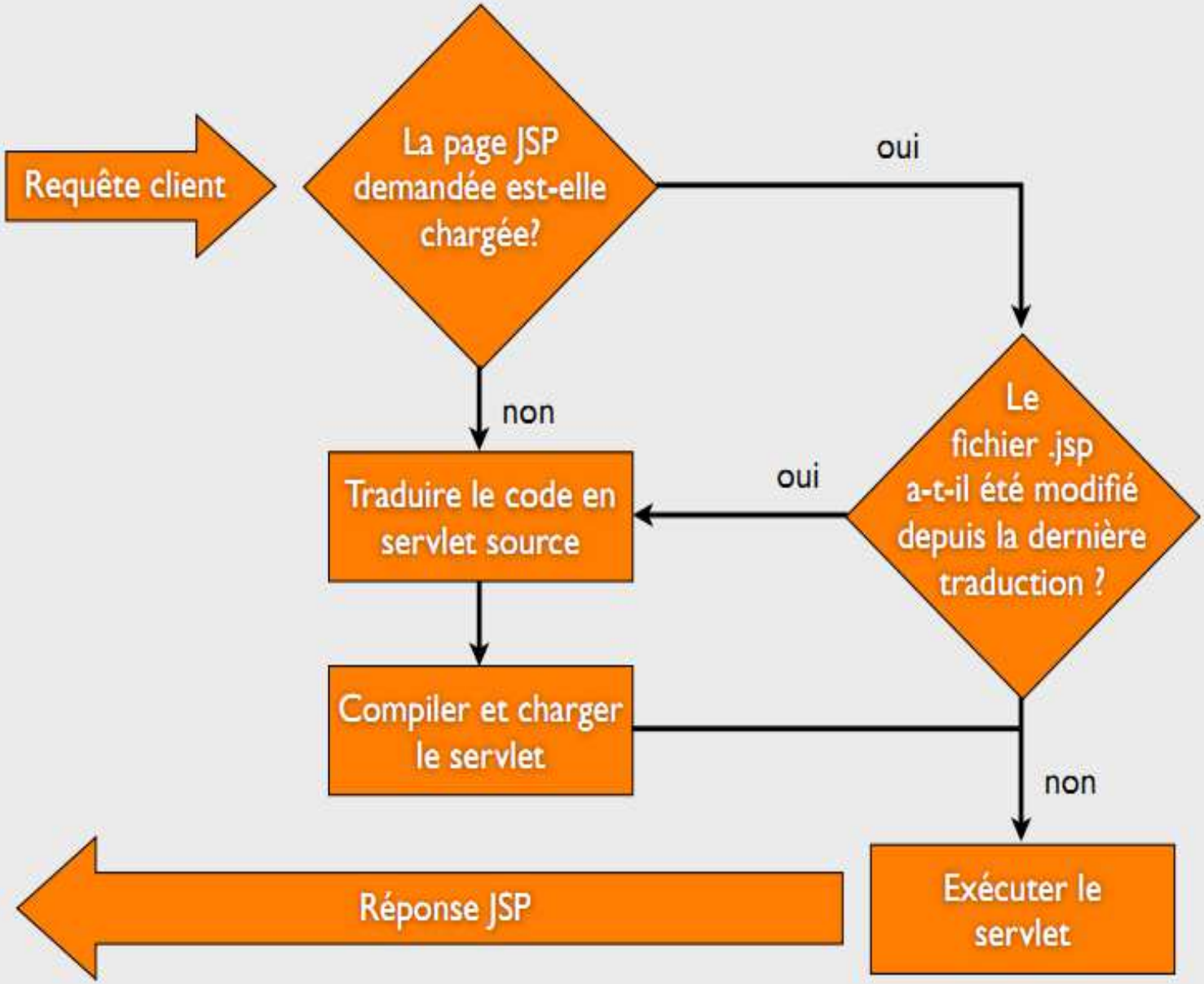
- Les JSP permettent d'introduire du code Java dans des tags prédéfinis à l'intérieur d'une page HTML. La technologie JSP mélange la puissance de Java côté serveur et la facilité de mise en page d'HTML côté client.

Une JSP est habituellement constituée :

- de données et de tags HTML
- de tags JSP
- de scriptlets (code Java intégré à la JSP)

Les fichiers JSP possèdent par convention l'extension .jsp.

Java Server Page(JSP)



Java Server Page(JSP)

- Syntaxe des éléments de base

Texte HTML

- <H1>Le titre</H1>
- Les commentaires HTML
- <!-- commentaire HTML -->
- Les commentaires JSP
- <%-- commentaire JSP --%>

Les Tags JSP

Trois types de tags

- <%@ … %> Tags de directives: contrôlent la structure de la servlet générée
- <% … %> Tags de scripting: permettent insertion de code java dans la servlet
- <jsp:… > Tags d'actions: facilitent l'utilisation de composants

Tags de directives

- Directives: <%@…%>
  - Permettent de spécifier des informations globales sur la page
  - instruction pour le moteur JSP (page et include)

<%@ page ... %>	permet de définir des options de configuration de la page
<%@ include ... %>	permet d'inclure des fichiers statiques dans la JSP avant la génération de la servlet
<%@ taglib ... %>	permet de définir des tags (actions) personnalisés

La directive page

Principaux attributs de la directive page

- <%@page import="java.util.*,java.sql.Connection" %>
- <%@page contentType="text/html;charset=ISO-8859-1" %>
- <%@page session="true|false" %>
  - Indique si la page est incluse ou non dans une session. Par défaut true, ce qui permet d'utiliser un objet de type HttpSession pour gérer des données de session
- <%@page errorPage="relativeURL" %>
  - Précise la JSP appelée au cas où une exception est levée
    - URL relative par rapport au répertoire qui contient la page JSP ou relative par rapport au contexte de l'application Web si elle débute par /
- <%@page isErrorPage=" true|false" %>
  - Précise si la page JSP est une page de gestion d'erreur (dans ce cas l'objet exception peut être utilisée dans la page), false par défaut.
- <%@page isThreadSafe=" true|false" %>
  - Précise si la servlet générée est multithreadée ou non.
- …

La directive include

- Syntaxe : <%@include file="chemin relatif du fichier" %>
  - chemin relatif par rapport au répertoire qui contient la page JSP ou relatif par rapport au contexte de l'application Web si il débute par /
- Inclus le fichier dans le source JSP avant que celui-ci ne soit interprété (traduit en servlet) par le moteur JSP
- Le fichier peut être un fragment de code JSP, HTML ou Java
- Tag utile pour insérer un élément commun à plusieurs pages (en-tête, pied de page): **Templating**

Tags de scripting

- Permettent d'insérer du code Java qui sera inclus dans la servlet générée

- Trois types de tags
- <%! …%> tag de déclaration
  - Code inclus dans le corps de la servlet (déclaration de membres, variable ou méthodes)
- <%=expression%> tag d'expression
  - L'évaluation de l'expression est insérée dans le flot de sortie dans la méthode service() de la servlet <==> out.println(expression)
- <%...%> tag de scriptlet
  - Le code Java est inclus dans la méthode service() de la servlet

Exemples

- Expressions: <%=…%>
  - date: <%= new java.util.Date() %>
  - hôte: <%= request.getRemoteAddr()%>

- Déclarations: <%!…%>:
  - <%! int i = 0; %>

- Scriplets: <%…%>
  - <%

```
        out.println( request.getRemoteAddr());
```

    %>

Variables Implicites

- les spécifications des JSP définissent plusieurs objets implicite utilisables directement dans le code Java

Variable	Classe	Rôle
out	javax.servlet.jsp.JspWriter	Flux en sortie de la page HTML générée
request	javax.servlet.http.HttpServletRequest	Contient les informations de la requête
response	javax.servlet.http.HttpServletResponse	Contient les informations de la réponse
session	javax.servlet.http.HttpSession	Gère la session
exception	java.lang.Throwable	L'objet exception pour une page d'erreur

JSP et Java Beans

- Un des objectif des JSP / Servlets
  - Ne pas mélanger du code HTML au code Java des

- Un point important dans la conception de pages JSP est de minimiser le code Java embarqué dans les pages
- Déporter la logique métier dans des composants objets qui seront accédés depuis les pages JSP
  - Simplification des traitements inclus dans la JSP
  - Possibilité de réutilisation des composants depuis d'autres JSP ou d'autres composants
- Les spécifications JSP définissent une manière standard d'intérargir avec des composants Java Beans

Java Bean

- Qu'est-ce qu'un composant Java Bean ?

- Un classe Java Bean
  - Classe publique
  - Possède un constructeur public sans arguments(constructeur par defaut)
  - Regroupe un ensemble de propriétés (private)
  - accessibles par des méthode de la forme getXXX() où XXX est le nom de la propriété
  - éventuellement modifiables par une méthode setXXX() où XXX est lenom de la propriété
  - Implémente en option l'interface java.io.Serializable

Exemple de Java Bean


```
Package test;
 public class Personne  {
        private String nom;
         public Personne() {
                ...
        }
        public String getNom() {
        return this.nom;
        }
        public void setNom(String nom) {
        this.nom = nom;
        }
}
```

JSP et Java Beans

Le respect de ces règles d'écriture permet ensuite d'utiliser ces composants de manière standard

- Composants Java Beans depuis JSP (à voir tout de suite)
- Composants graphiques pour la construction d'interface utilisateurs
- …

Utiliser un bean depuis une JSP

- Le tag <jsp:useBean>Permet de localiser une instance ou bien d'instancier un bean pour l'utiliser dans la JSP


```
<jsp:useBean
    id="beanInstanceName"
    class="package.class"
    type="package.class"
    scope="page | request | session | application"
/>
```

Nom utilisé pour la variable qui servira de référence sur le bean

La classe du bean

Optionnel, le type de la référence beanInstanceName si ce n'est pas le type défini par l'attribut class

Optionnel, détermine la portée durant laquelle le bean est défini et utilisable

- Si le bean demandé est déjà instancié pour la portée précisée:
  - renvoi de sa référence
- S'il n'est pas déjà présent pour la portée précisée
  - instanciation du bean
  - ajout de celui-ci à la portée spécifiée

Utiliser un bean depuis une JSP

L'attribut scope	Description
page	**valeur par défaut** bean utilisable dans toute la page JSP ainsi que dans les fichiers statiques inclus.
request	bean accessible durant la durée de vie de la requête. La méthode getAttribute() de l'objet request permet d'obtenir une référence sur le bean.
session	bean utilisable par toutes les JSP qui appartiennent à la même session que la JSP qui a instanciée le bean. Le bean est utilisable tout au long de la session par toutes les pages qui y participent. La JSP qui créé le bean doit avoir l'attribut session = "true" dans sa directive page
application	bean utilisable par toutes les JSP qui appartiennent à la même application que la JSP qui a instanciée le bean. Le bean n'est instancié que lors du rechargement de l'application

Exemple


```
Package test;
public class Personne  {
        private String nom;
         public Personne() {
                this.nom= "TDSI";
        }
        public String getNom() {
        return this.nom;
        }
        public void setNom(String nom) {
        this.nom = nom;
        }
}
```

```
<jsp:useBean
        id="utilisateur"
        class="test.Personne"
         scope="session" />
```

```
<HTML>
    <BODY>
        <UL>
            <LI>NOM : <%=utilisateur.getNom()%></LI>
        </UL>
    </BODY>
</HTML>
```

Fixer les propriétés d'un bean depuis une JSP

- Le tag <jsp:setProperty>
  - Permet de mettre à jour la valeur de un ou plusieurs attributs d'un bean

- Syntaxe

```
<jsp:setProperty name="beanInstanceName"
                 property="propertyName"
                 value="string|<%=expression%>"/>
```

```
<jsp:useBean id="utilisateur" class="test.Personne" scope="session"/>
...
<jsp:setProperty name="utilisateur" property="nom" value="Toto"/>
```

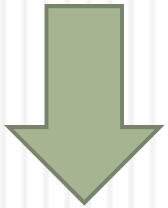
Fixer les propriétés d'un bean depuis une JSP

- Possibilité de fixer les propriétés du bean à l'aide des paramètres de la requête

Syntaxe

- <jsp:setProperty name="beanInstanceName" property="propertyName"/>
  - Si le nom de la propriété est le même que le nom du paramètre de la requête
- <jsp:setProperty name="beanInstanceName" property="propertyName" param="paramétrage"/>
  - Si Le nom de la propriété est différent du nom du paramètre de la requête

MaPageJSP?name=Toto



```
<jsp:useBean id="utilisateur" class="test.Personne" scope="session"/>
<jsp:setProperty name="utilisateur" property="nom" param="name"/>
```

Accéder aux propriétés d'un bean depuis une JSP

- Le tag <jsp:getProperty … >

  - Permet d'obtenir la valeur d'un attribut d'un bean
- Syntaxe
  - <jsp:getProperty name="beanInstanceName" property="propertyName" />
- Exemple

```
<jsp:useBean id="utilisateur" class="test.Personne" scope="session"/>
…
<body>
<UL>
        <LI>Nom : <jsp:getProperty name="utilisateur" property="nom"/></LI>
</UL>
```

Tag de redirection

- Le tag <jsp:forward>

  - Permet de rediriger la requête vers un fichier HTML, une autre page JSP ou une Servlet

- Syntaxe
- <jsp:forward page="relativeURL|<%=expression%>"/>

Exemple

```
<jsp:forward page="uneAutrePage.jsp" />
```

Si URL commence par un / elle est absolue (contexte de l'application) sinon elle est relative à la JSP

Ce qui suit l'action forward est ignoré, et tout ce qui a été généré dans cette page JSP est perdu

Tag de redirection

- Possibilité de passer un ou plusieurs paramètres vers la ressource appelée


```
<jsp:forward page="relativeURL|<%=expression%>">
<jsp:param name="parametre" value="string|<%=expression%>">
...
</jsp:forward>
```

Exemple réalisation d'un Module de connexion

- L'objectif est de reprendre l'exercice (page 31),

1. Un javaBean qui represente la table user
2. Une servlet capable de recupérer le login et password, capable d'intéroger une BD
3. Une page JSP contenant un formulaire de saisi de login et password
4. Une page JSP de Bienvenue qui présente l'utlisateur

Une javaBean qui represente la table user

```
package com.tdsi;
import java.io.Serializable;
public class Users implements Serializable {
    private Integer idusers;
    private String nom;
    private String prenom;
    private String login;
    private String password;
    private String email;
    private String tel;
    public Users() {
    }
```

```
public Users(Integer idusers, String nom, String prenom) {
        this.idusers = idusers;
        this.nom = nom;
        this.prenom = prenom;
    }
 // Générer les Getter et Setter
}
```

Fin de la Classe Users

Une servlet capable de recupérer le login et password, capable d'intéroger une BD

```
public class Auth extends HttpServlet {
    private Connection conn;
    private Statement stmt;
    protected void processRequest(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException{
    try {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        Users user1=null;
        HttpSession session = request.getSession();
       String username = request.getParameter("username");
       String password = request.getParameter("password");
       Class.forName("com.mysql.jdbc.Driver").newInstance();
       conn=DriverManager.getConnection("jdbc:mysql://localhost/conf_tdsi?user=root");
       PreparedStatement prep1 = conn.prepareStatement("SELECT * FROM users WHERE LOGIN = ? AND PASSWORD=?");
        prep1.setString(1, username);
        prep1.setString(2, password);
        ResultSet rs = prep1.executeQuery();
        while (rs.next()) {
              user1 = new Users(rs.getInt("IDUSERS"),rs.getString("NOM"),rs.getString("PRENOM"));
              session.setAttribute("user", user1);
              break;
          }
```

```
            if(session.getAttribute("user")!=null){
                        request.getRequestDispatcher("accueil.jsp"). forward( request,response);
            }else {
                        request.getRequestDispatcher("index.jsp").forward( request,response);
             }
       } catch (Exception ex) {
            ex.printStackTrace();
       }
 }
@Override
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
         processRequest(request, response);
   }
   @Override
   protected void doPost(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      processRequest(request, response);
   }
}
```

Page Login.jsp

```
<form action="auth" id="lg-form" name="lg-form" method="post">
        <label for="username">Username:</label>
        <input type="text" name="username" id="id_username" placeholder="your username"/>
        <label for="password">Password:</label>
        <input type="password" name="password" id="id_password" placeholder=" Your password" />
        <button type="submit" id="login">Login</button>
</form>
```

La Page: accueil.jsp

```
<%@ page language="java" %>
<!DOCTYPE HTML PUBLIC "-//w3c//dtd html 4.0 transitional//en">
<jsp:useBean id="user" class=" com.tdsi.Users" scope="session"/>
<jsp:setProperty name="user" property="password" value=""/>
<HTML>
<HEAD><TITLE>accueil</TITLE></HEAD>
<BODY>
Bonjour <%= user.getPrenom ( ) %> <%= user.getNom( ) %>
</BODY>
</HTML>
```

Exercice 1: calcul de factorielles de nombres entiers.

Création d'une servlet pour le calcul de factorielle.

Demande à l'utilisateur de saisir un nombre positif sur un champ de texte et renvoie le factorielle de nombre par validation d'un bouton VALIDER du formulaire.

Exercice 2: gestions de plusieurs boutons de formulaires et JDBC

On dispose d'une base de données Mysql Employe contenant une table Personnel (Matle, Prenom, Nom, Sexe, Adresse, Pays).

Créer une servlet qui permet soit de sauver de nouveaux employés dans la base soit de modifier les données concernant un employé soit d'afficher sur une table tous les employés.

On se servira d'un formulaire de saisie avec trois boutons.

Exercice 3: JDBC et transfert de contrôle à un autre servlet.

On dispose d'une base de données Mysql Banque contenant une table Compte (NumCompte, prénom, Nom, Sexe, adresse, pays, solde, decouvert).

Créer une servlet qui demande à l'utilisateur de saisir le numéro de compte d'un client, de faire un forward vers une deuxième servlet qui se chargera de vérifier si ce compte existe et d'afficher le solde et le decouvert du compte.

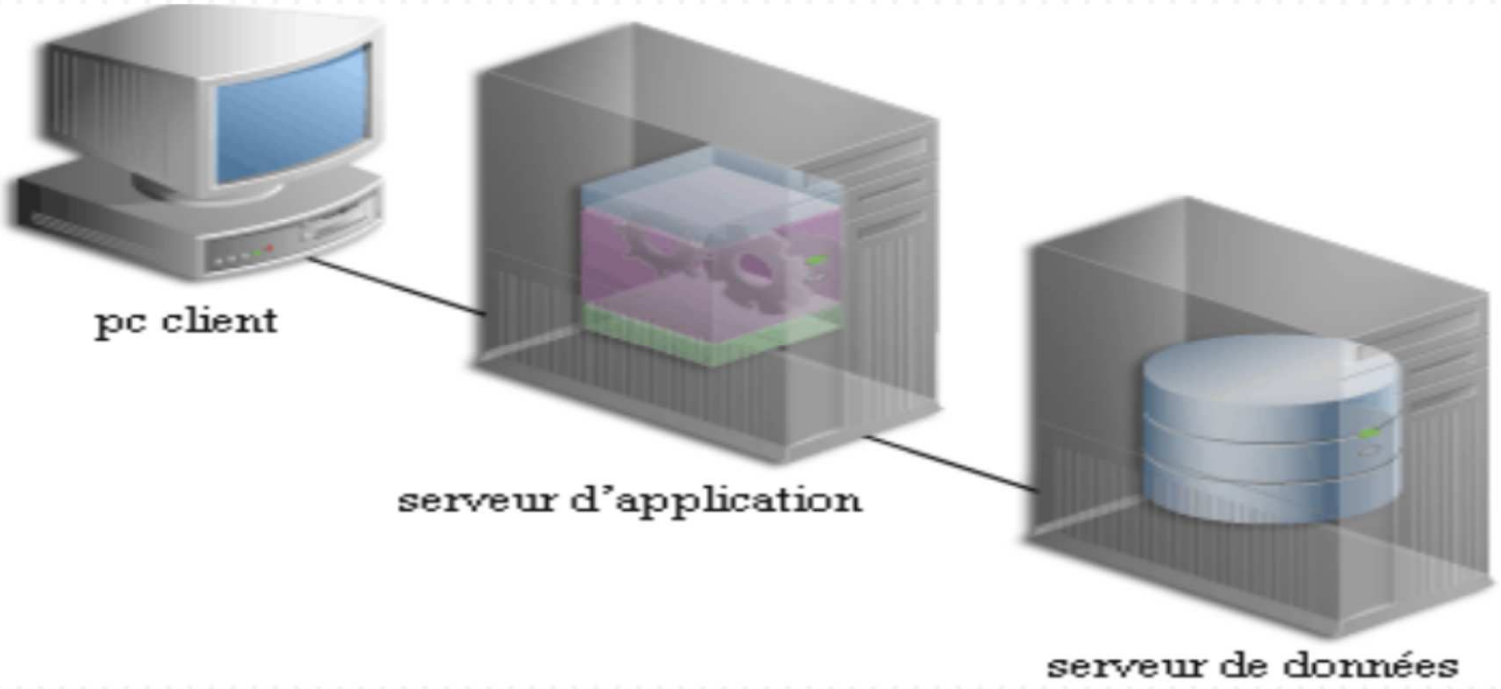
Si le compte n'existe, pas, elle l'indique aussi. On se sert aussi de formulaire.

architectures n-tiers

- Une Architecture **n-tiers (**à trois niveaux) est une architecture client-serveur dans lequel

  - l'interface utilisateur (**présentation ou Vue ou IHM**),
  - la logique de processus fonctionnel (**Métier ou Service**)
  - Le Moteur de stockage de données (**l'accès aux données**)

sont développés et maintenus en tant que modules indépendants, le plus souvent sur des plates-formes distinctes.



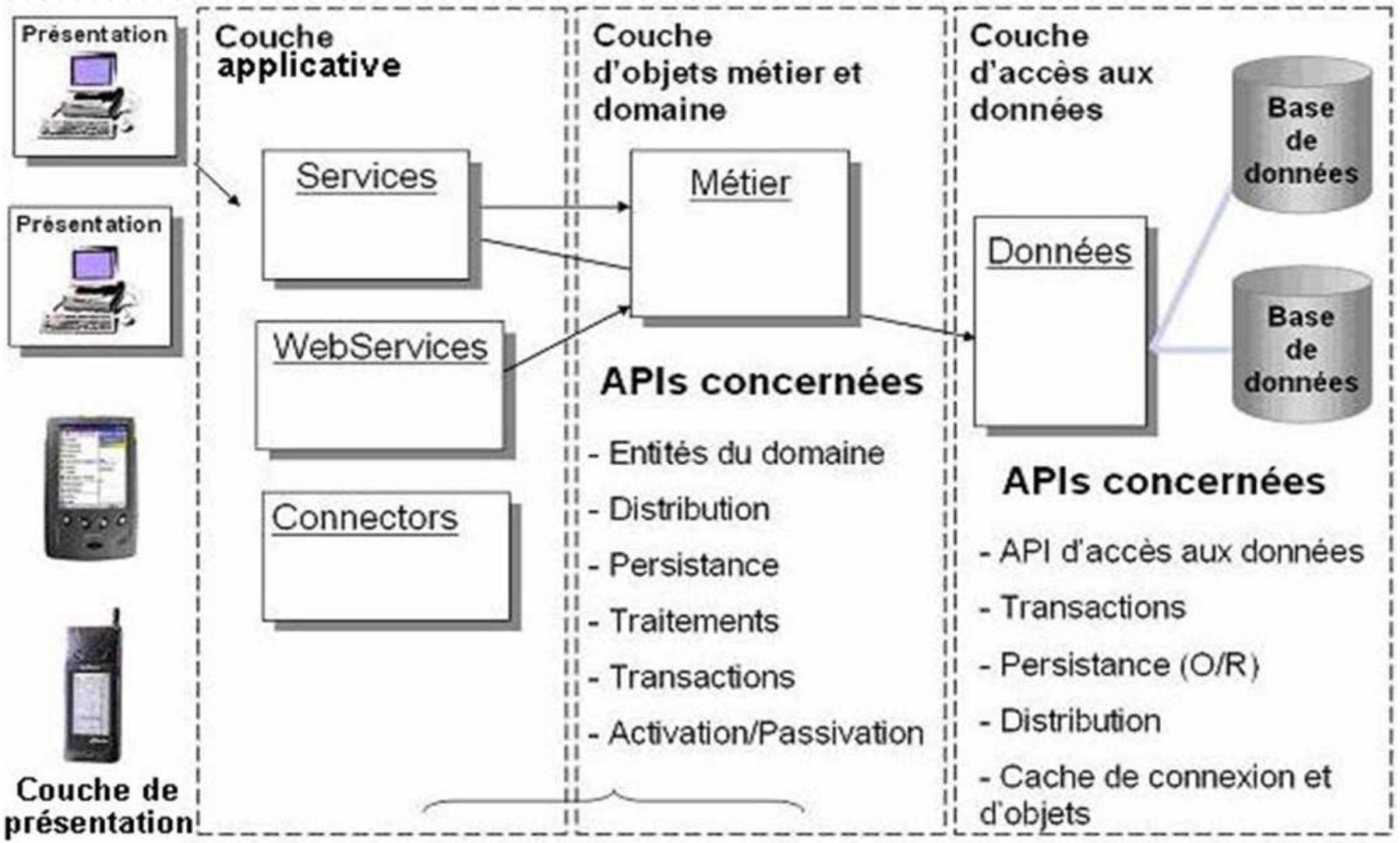
architectures n-tiers

- Motivation pour les serveurs d'application

- Simplifier le développement
- Séparer clairement les différentes couches
- se concentrer sur la logique de l'application (le reste est pris en charge par la structure d'intégration)
- Obtenir des gains de productivité
- Faciliter l'intégration avec l'existant
- Optimiser la maintenance et la sécurité
- Permettre la liberté de choix d'Outils
- Fournir une réponse adaptée à la demande
- Proposer une architecture orienté Services (SOA)
- Proposer une architecture Répartie (distribuée)

architectures n-tiers

Exemple : 4-tiers



Les différentes couches d'une architecture 4-tier :

- **La couche de présentation** contient les différents types de clients, léger (HML) ou lourd (Applet, Software)
- **La couche applicative** contient les traitements représentant les règles métier: (créer un compte de facturation, calculer un amortissement ...)
- **La couche d'objets métier** est représentée par les objets du domaine, c'est à dire l'ensemble des entités persistantes de l'application (Facture, Client ...)
- **La couche d'accès aux données** contient les usines d'objets métier, c'est à dire les classes chargées de créer des objets métier de manière totalement transparente, indépendamment de leur mode de stockage (SGBDR, Objet, Fichiers, ...)

Les Composants dans J2EE

- La **programmation orientée composant** (POC) consiste à utiliser une approche modulaire de l'architecture d'un projet informatique, ce qui permet d'assurer au logiciel une meilleure lisibilité et une meilleure maintenance.
- Les développeurs se servent de briques réutilisables.

Cf: cours UML

- **J2EE est une architecture de composants :**

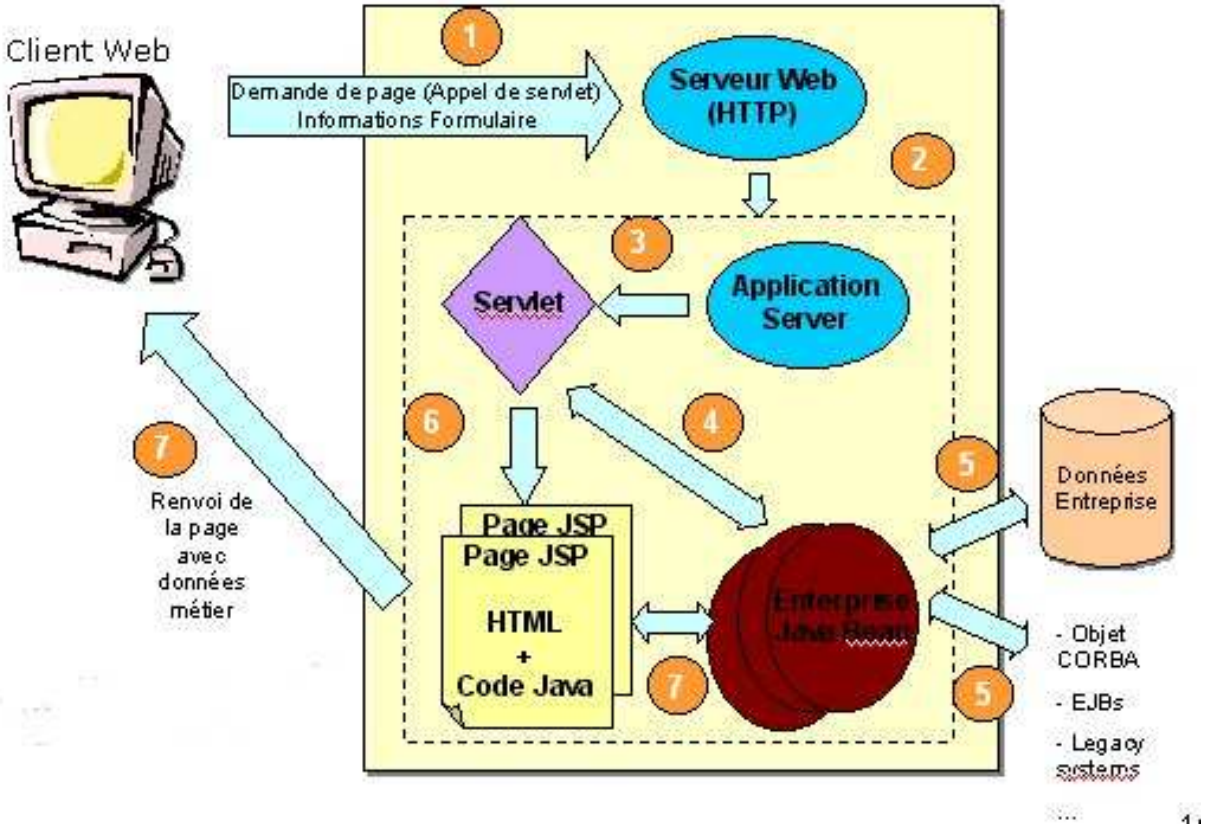
- Objectif des composants :
  - avoir des briques de bases réutilisables.
- Définition d'un composant :
  - **module logiciel,**
  - **exporte différents attributs, propriétés et méthodes,**
  - **est prévu pour être configuré,**
  - **est prévu pour être installé,**
  - **fournit un mécanisme lui permettant de s'auto-décrire.**
- Composant = objet + configurateur + installateur.

ASSEMBLAGE ET DEPLOIEMENT D'APPLICATIONS J2EE 1/6

- **Le développement d'applications Web repose sur trois composants J2EE principaux :**
  - **Les servlets** : ce sont des programmes Java exécutés sur un serveur (par sa JVM). Ils permettent d'étendre le comportement du serveur dynamiquement.
  - **Les JSP** : ce sont des pages HTML incluant du code JAVA (stocké à l'intérieur de balises).
  - **Les EJB** : ce sont des entités de traitement s'exécutant dans un environnement adapté (conteneur) et possédant des mécanismes de configuration et d'installation.

ASSEMBLAGE ET DEPLOIEMENT D'APPLICATIONS J2EE 2/6

- **Mécanisme d'une application Web J2EE :**



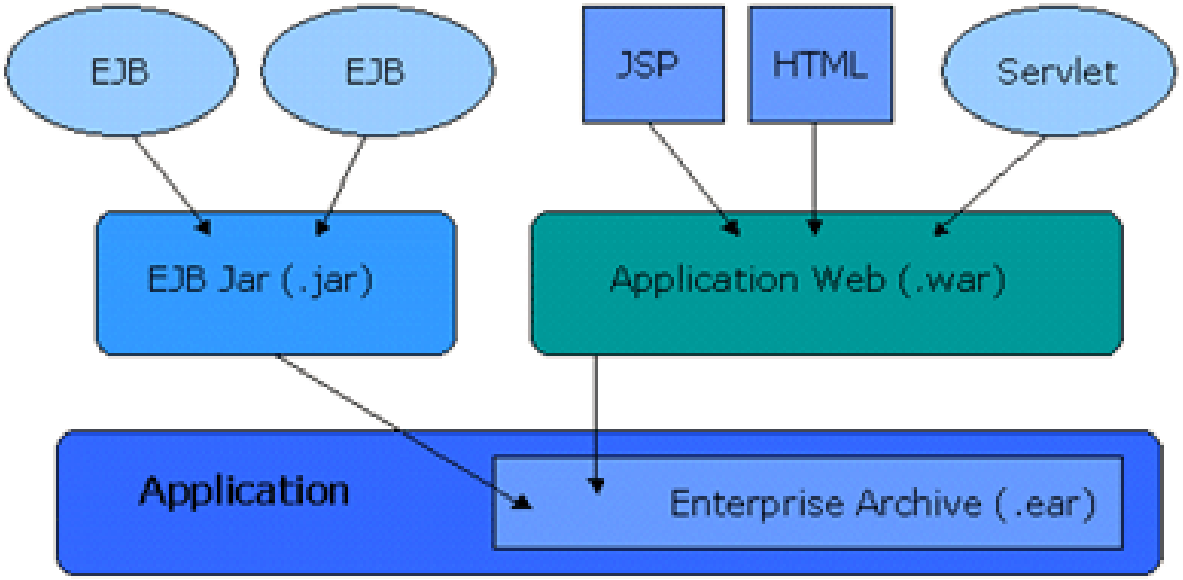
3 - La servlet contrôle la validité de la requête HTTP.

4 - Elle instancie les beans de données pour accéder aux données.

6 - Elle invoque la JSP pour générer la page HTML qui contient le résultat de la requête.

ASSEMBLAGE ET DEPLOIEMENT D'APPLICATIONS J2EE 3/6

Architecture d'une application J2EE :



- 3 couches :
  - *Les composants.*
  - *Les modules regroupant les composants*
  - *Les applications regroupant les modules*

- Les modules et les applications correspondent physiquement à des fichiers d'archives : archive EJB JAR (.jar) pour un module EJB, archive WAR pour un module web, archive EAR pour une application.

ASSEMBLAGE ET DEPLOIEMENT D'APPLICATIONS J2EE 4/6

Module Web (.war). Selon la spécification J2EE, une application Web doit avoir la structure suivante:

- un répertoire racine public contenant les pages HTML, les pages JSP, les images...
- un répertoire *WEB-INF* situé dans le répertoire racine de l'application web.
- un fichier *web.xml* situé à la racine de *WEB-INF* : c'est le descripteur de déploiement de l'application web.
- un répertoire *WEB-INF/classes* contenant les classes compilées de l'application (servlets, classes auxiliaires...).
- un répertoire *WEB-INF/lib* contenant les fichiers JAR de l'application (drivers JDBC, frameworks empaquetés...).

Le tout peut être empaqueté dans une archive sous la forme d'un fichier WAR (réalisé avec l'utilitaire jar du JDK).

Module EJB (.jar). Selon la spécification J2EE 1.2, un fichier JAR doit avoir la structure suivante :

- un répertoire *META-INF/* contenant un descripteur de déploiement XML du module EJB, nommé ejb-jar.xml
- les fichiers .class correspondant aux interfaces locale (home interface) et distante (remote interface), à la classe d'implémentation, et aux classes auxiliaires (classes d'exception par exemple) des EJBs, situées dans leur package.

Le tout peut être empaqueté dans une archive sous la forme d'un fichier JAR.

ASSEMBLAGE ET DEPLOIEMENT D'APPLICATIONS J2EE 6/6

Application d'entreprise (.ear). Selon les spécifications J2EE, une application d'entreprise doit avoir la structure suivante :

- un répertoire *META-INF/* contenant le descripteur de déploiement XML de l'application J2EE nommé application.xml. C'est dans ce descripteur que l'on définit les modules web et EJB qui constituent l'application d'entreprise. On y précise par exemple sur quelle racine du serveur web (placé en frontal devant le serveur d'application) doit résider l'application web.
- les fichiers archives .JAR et .WAR correspondant aux modules EJB et aux modules Web de l'application d'entreprise.

Le tout peut être empaqueté dans une archive sous la forme d'un fichier EAR.

Le Pattern MVC dans une architecture 3-thiers

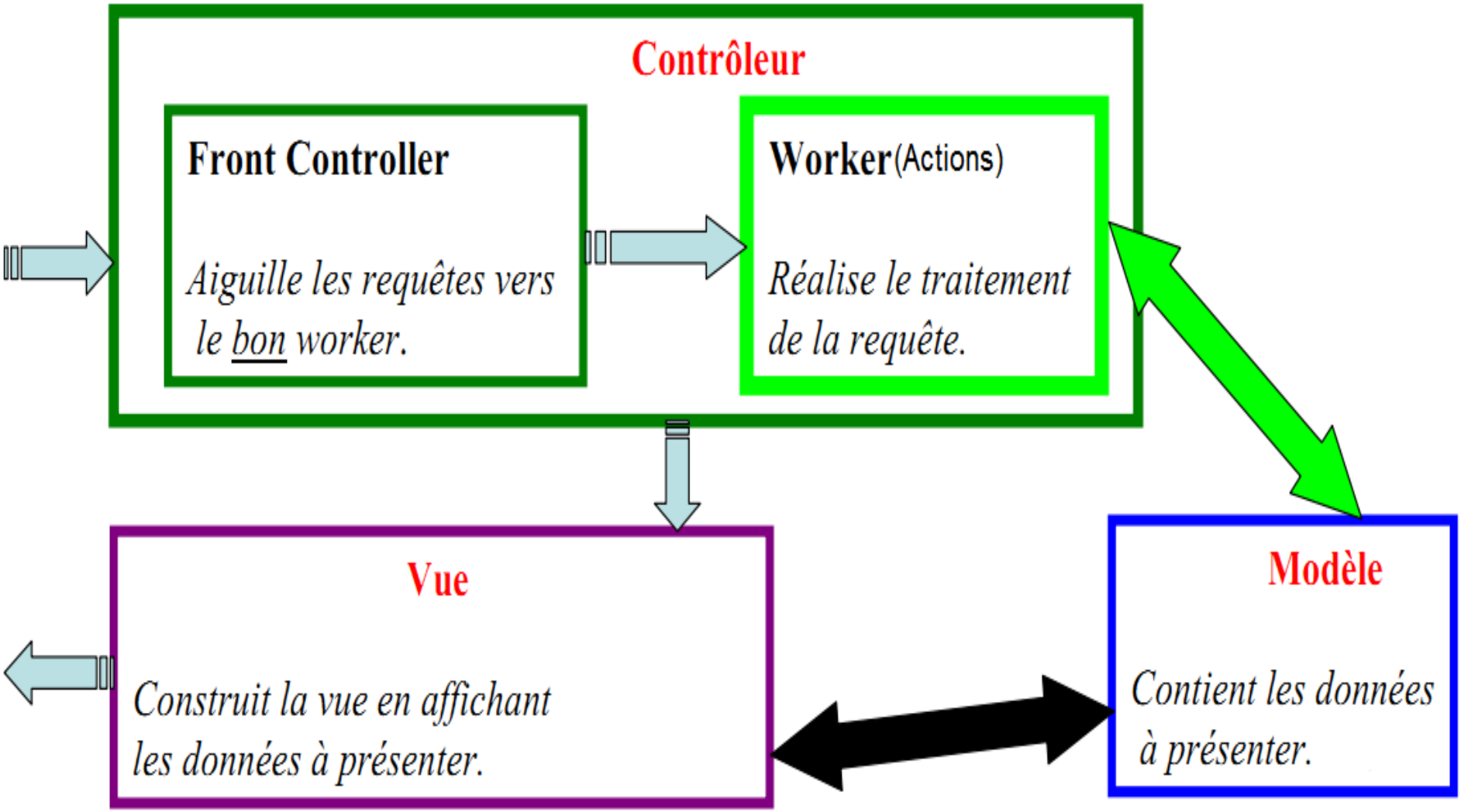
- Le pattern MVC est communément utilisé dans les applications Java/JEE pour réaliser la couche présentation des données aussi bien pour les applications Web que pour les clients lourds.

Lorsqu'il est utilisé dans le cadre de JEE, il s'appuie généralement sur l'API servlet et des technologies comme JSP/JSTL.

- Le modèle de conception MVC met en œuvre trois composants: un modèle, une vue, un contrôleur.
  - Le modèle contient les objets représentant les données. Ce sont des objets qui sont manipulés afin de produire les éléments qui seront présentés à l'utilisateur.(JavaBeans, EJB)
  - La vue est la représentation (en général à l'écran) du modèle. La vue représente l'état courant du modèle.(HTML, JSP, SWING,…)
  - le contrôleur gère les interactions de l'utilisateur avec l'interface. Il manipule le modèle et les objets représentant les données. (une seule servlet dans l'application)

Le design pattern MVC

- le pattern MVC dit de type 2, plus récent qui possède un contrôleur unique.



Le contrôleur gère les interactions avec le client tout en déclenchant les traitements appropriés.

Cette entité interagit avec les composants de la couche service métier et a pour responsabilité la récupération des données mises à la disposition dans le modèle.

Ce point d'entrée unique traite la requête et dirige les traitements vers l'entité Worker appropriée.

Pour cette raison, l'entité Worker est habituellement appelée contrôleur.

Le Front Controller ou « contrôleur façade » est intégré au framework MVC, et seuls les workers sont spécifiques.

Un framework MVC implémente le contrôleur façade, les mécanismes de gestion du modèle ainsi que ceux de sélection et de construction de la vue.

L'utilisateur d'un tel framework a en charge le développement et la configuration des workers.

Frameworks MVC

- JavaServer Faces :
  - Controller: javax.faces.webapp.FacesServlet
- Struts
  - Controller : org.apache.struts2.dispatcher.FilterDispatcher
- Spring MVC
  - Controller : org.springframework.web.servlet.DispatcherServlet

- Exercice :

- Reprendre le dernier exercice sur l'authentification et proposer une architecture 3-tiers reposant sur le pattern MVC.

Java Server Faces

Java Server Faces

Java Server Faces permet de développer des applications web en bénéficiant de concepts déjà éprouvés par Java et Java EE (composants graphiques Swing, modèle de programmation événementiel, JSP, servlets,

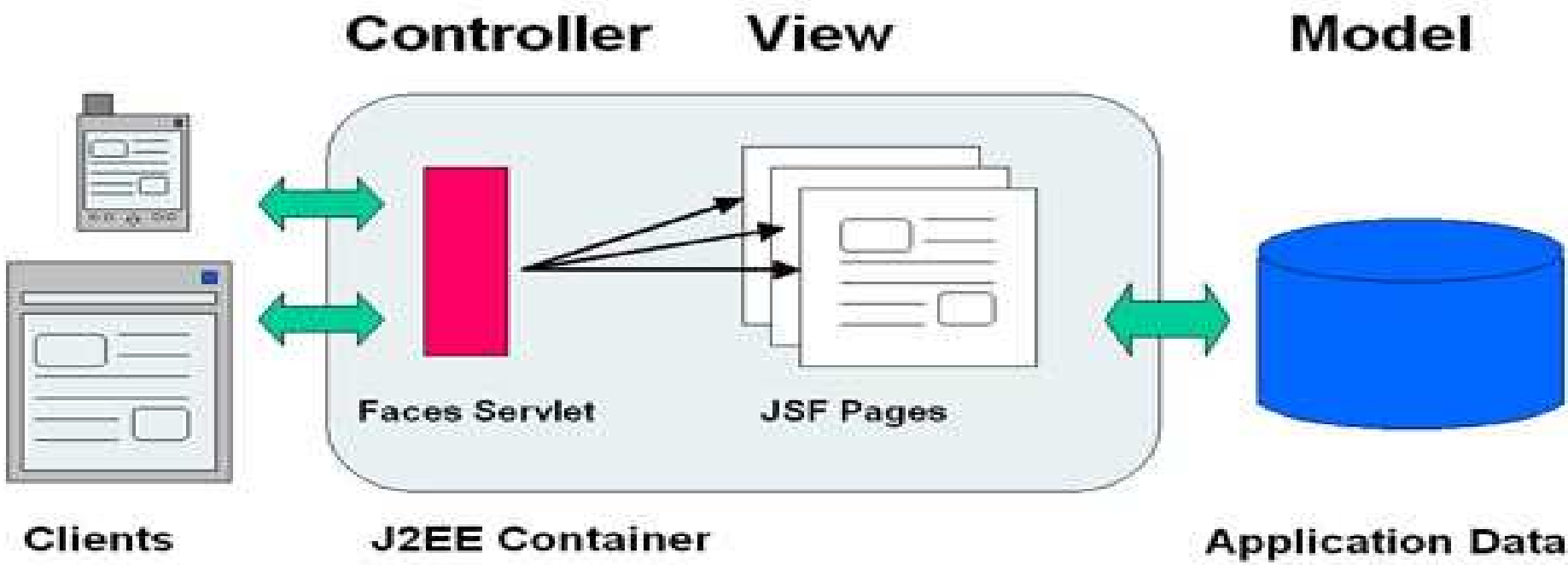
JSTL, langage d'expression), et par les apports d'autres framework Open Source tel que Struts.

JSF ne remplace pas les autres technologies web, il les utilise et les complète.

JSF apporte plusieurs fonctionnalités destinées à résoudre les problèmes inhérents à la programmation web. JSF se compose d'un ensemble d'API fournissant notamment :

- Une séparation nette entre la couche de présentation et les autre couches ;
- Une librairie de composants graphiques ;
- Un mode de programmation événementiel pour ces composants ;
- La navigation entre pages ;
- Le traitement des formulaires et leur validation ;

- La gestion des exceptions et l'affichage de messages d'erreur ;
- La conversion des types primitifs provenant des données d'applications vers des objets de plus haut niveau (String vers Objets) ;
- La gestion de clients hétérogènes (HTML, WML, XML...) ;
- L'indépendance des protocoles (HTTP, WAP...) ;
- La création ou l'enrichissement de composants graphiques (custom) ;
- La gestion de l'état des composants entre requêtes ;
- La différence de comportement entre navigateurs.
- Ajax sans programmation


Controller
View
Model
Faces Servlet
JSF Pages
Clients
J2EE Container
Application Data

Une page JSF

- Code XHTML,
- Contient des parties en EL(**Expression Language**) : #{…},
- Utilise souvent une (ou plusieurs) bibliothèque de composants,
- Sera traduite en XHTML « pur » pour être envoyée au client Web,
- Les pages JSF sont utilisées pour l'interface avec l'utilisateur.

- Les pages JSF ne contiennent pas de traitements (pas de code Java ou autre code comme dans les pages JSP, « ancien langage » de JSF pour créer les pages affichées à l'utilisateur),
- Les traitements liés directement à l'interface utilisateur sont écrits dans les backing beans qui offrent des beans managés (managed bean),
- Ces backing beans font appels à des EJB ou des classes Java ordinaires pour effectuer les traitements qui ne sont pas liés directement à l'interface utilisateur

Backing beans

Ils sont associés aux pages JSF qui font la liaison entre les composants JSF de l'interface, les valeurs affichées ou saisies dans les pages JSF et le code Java qui effectue les traitements métier.

- Souvent, mais pas obligatoirement, un backing bean par page JSF,
- Conserve, par exemple, les valeurs saisies par l'utilisateur (attribut « value »), ou une expression qui indique si un composant ou un groupe de composant doit être affiché ; peut aussi conserver un lien vers un composant de l'interface utilisateur (attribut « binding »), ce qui permet de manipuler ce composant ou les attributs de ce composant par programmation: on parle de **managed bean**

- JSF utilise deux librairies de balises, HTML et Core, définies dans les pages JSP par les directives suivantes :
  - <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
  - <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>

Ces déclarations impliquent que les balises Core seront préfixées par f (<f:view>) et les balises HTML par h (<h:form>).

Les balises HTML

Balise JSF	Rendu graphique
`<h:outputText value="#{cart.customer.lastname}"/>` ❶	Smith
`<h:inputText value="#{cart.customer.firstname}" />` ❷	

`<h:inputSecret value="#{account.password}"` ❸
`  maxlength="10"`
`  size="12"/>`	

`<h:message style="color: red"/>` ❹
 	
`Login: <h:inputText value="#{account.login}"/>`
`Pwd: <h:inputSecret value="#{account.pwd}"/>`
`<h:commandButton value="Sign In"`
`  action="#{account.doSignIn}" type="submit"/>`
	Invalid login/password .
	Login: job
	Pwd: •••
	Save

Balise JSF	Rendu graphique

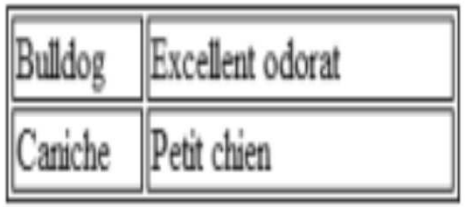
`<h:commandButton value="Sign In"` ❺
`  action="#{account.doSignIn}"`
`  type="submit"/>`
	Sign In

`<h:commandLink action="#{account.doSignIn}">` ❻
`  <h:outputText value="Sign In"/>`
`</h:commandLink>`
	Sign In

`<h:graphicImage url="images/bird1.jpg"/>` ❼	

`<h:commandLink action="#{catalog.doShowItem}">`
`  <h:graphicImage url="images/bird1.jpg"/>` ❽
`</h:commandLink>`
	

Balise JSF	Rendu graphique
<h:selectOneMenu value="#{cart.creditCard.type}"> ⑨ <f:selectItem itemLabel="Visa"/> <f:selectItem itemLabel="Visa Gold"/> <f:selectItem itemLabel="Master Card"/> </h:selectOneMenu>	
<h:panelGrid columns="4"> ⑩ <h:outputText value="1" /> <h:outputText value="2" /> <h:outputText value="3" /> <h:outputText value="4" /> <h:outputText value="5" /> <h:outputText value="6" /> <h:outputText value="7" /> </h:panelGrid>	

Balise JSF	Rendu graphique
<h:dataTable value="#{catalog.products}" var="product"> <h:column> ⑪ <h:outputText value="#{product.name}"/> </h:column> <h:column> <h:outputText value="#{product.description}"/> </h:column> </h:dataTable>	

JSF Core Tags:

Ces tags JSF permettent de gérer les validations, évènements, conversions, etc.…

La librairie Core est compatible (supporte) la librairie html.

La librairie Core contient des tags pour les vues et les sous vue, elle contient aussi des tags pour le chargement de ressources (resource bundle), pour l'ajout de texte dans les pages.

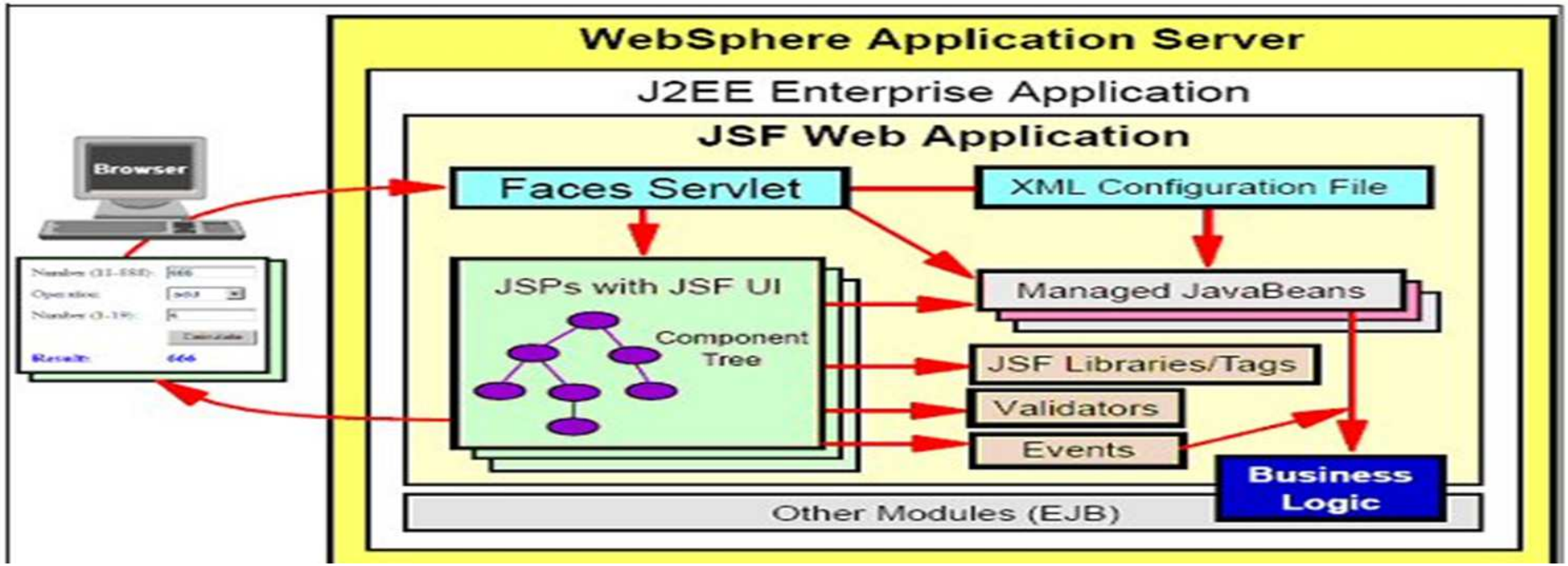
Exemple:

f:view	utilisé pour construire la vue parente
f: subview	utilisée pour créer une sous vue d'une vue.
f: validator	pour ajouter un validateur (validator) pour un composant.
f:converter	pour ajouter un converter à un component.
f:actionListener	pour ajouter un listener à un component.
f:valueChangeListener	pour ajouter un valuechange listener à un component

JSF mise en Route

Traitements et navigation

il n'y a qu'un contrôleur unique (le FacesServlet) qui intercepte les requêtes HTTP , une navigation configurable par fichier XML , des traitements délégués à des managed beans qui manipulent le modèle et un rendu graphique utilisant les balises JSF .



La FacesServlet

La FacesServlet reçoit les requêtes de la part des clients web, et exécute un certain nombre d'étapes pour préparer la réponse qui sera affichée par le navigateur. Les actions et la navigation entre pages sont paramétrables par fichier XML (faces-config.xml).

Pour que cette servlet puisse intercepter toutes les requêtes concernant JSF, elle doit être déclarée dans le fichier web.xml de l'application.

```
<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>
```

un managed bean

La FacesServlet délègue les traitements et les appels aux managed beans. En fait, JSF instancie le bean en fonction de son champ d'application (scope), le stocke dans le contexte JSF, et l'invoque lorsque nécessaire.

Une Classe pour représenter les données de formulaires

```
@Named(value="userManagedBean")
@SessionScoped
public class UserManagedBean implements Serializable{

    private Users utilisateur;
    public UserManagedBean() {
    }
    // Getters & setters des attributs
}
```

Transforme le javaBean en Managed Bean (@ManagedBean).
Les pages peuvent via EL (#{userManagedBean}) accéder aux méthodes et propriétés (champs) de la classe UserManagedbean

NB: ici on prévoit de mettre au niveau d'une page JSF des formulaires qui concerne les « Users » : login page, inscription, mot de passe oublié, …

un managed bean

- D'une manière équivalente on pouvait remplacer les annotations par :


```
<managed-bean>
 <managed-bean-name>userManagedBean</managed-bean-name>
 <managed-bean-class>com.controller.UserManagedBean</managed-bean-class>
 <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

Dans **faces-config.xml**

Chaque tag : <managed-bean> possède trois tags fils obligatoires :

<managed-bean-name> : le nom attribué au bean utilisé lors de son utilisation

<managed-bean-class> : le type pleinement qualifié de la classe du bean

<managed-bean-scope> : portée dans laquelle le bean sera stocké et donc utilisable

La portée peut prendre les valeurs suivantes :

- request : portée 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.

Contexts and Dépendance Injection

- Les services proposes par le CDI:

- **Contexts**: la possibilité de lier le cycle de vie et les interactions avec les composants
- **Dependency injection**: La possibilité d'injecter (créer dynamiquement) un composant à l'intérieur d'un autre, (*Inversion of Control*, IoC) .

NB: il est fortement conseillé d'utiliser le CDI en JSF , d'ailleurs la méthode de dépendance forte sera déprécié à la prochaine version majeur:
Remplacer :
import javax.faces.bean.*; ----------> import javax.enterprise.context.*;
import javax.inject.*;
import javax.faces.bean.ManagedBean; ---------- >import javax.inject.Named;

Initialisation et destruction des Managed Beans

- Des actions et traitements peuvent être enclenchés juste après l'initialisation et juste avant la destruction du Managed beans par le conteneur. (see CDI)
- @javax.annotation.PostConstruct
- @javax.annotation.PreDestroy

```
@PostConstruct
public void init(){
    utilisateur=new Users();
}
@PreDestroy
public void dest(){
    utilisateur=null;
}
```

La navigation

La navigation

- La navigation entre les pages indique quelle page est affichée quand l'utilisateur clique sur un bouton pour soumettre un formulaire ou sur un lien,
- Cette navigation est déléguée au handler de navigation,
- La navigation peut être définie par des règles dans le fichier de configuration faces-config.xml ou par des valeurs écrites dans le code Java ou dans la page JSF
- La navigation peut être statique : définie « en dur » au moment de l'écriture de l'application,
- La navigation peut aussi être dynamique : définie au moment de l'exécution par l'état de l'application (en fait par la valeur retournée par une méthode)

Redirection des les backing beans

- Cette redirection est dynamique et la route est définie par la valeur de retour de la fonction exécuter


```
public String login(){
    Users user = UserAction.connect(utilisateur);
    if(user!=null){
        utilisateur=user;
        System.out.println(utilisateur);
        return "accueil.xhtml?faces-redirect=true";
    }

    else return "index.xhtml";
}
```

Dans la page index.xhtml

```
<h:form>
    <h:outputLabel value="Login:"/>
    <h:inputText value="#{userManagedBean.utilisateur.login}" />
    <h:outputLabel value="Password: "/>
    <h:inputSecret value="#{userManagedBean.utilisateur.password}"/>
    <h:commandButton action="#{userManagedBean.login()}" value="Valider"/>
</h:form>
```

Dans accueil.xhtml

```
Bonjour <h:outputText value="#{userManagedBean.utilisateur.prenom}"/>
```

- Nb: Ici l'EL nous permet d'acceder aux champs et méthodes du Managed bean.

- Dans le bean: @Named(value="userManagedBean")
- Dans la page: #{userManagedBean.utilisateur.login} qui correspend à New UsermanagedBean().getUtilisateur().getLogin();

Le Langage d'Expression

EL

Le Language d'Expression fournit un important mécanisme permettant la couche présentation (les vues) de communiquer avec la logique application (managed Bean).

- Lire dynamiquement, les données gardées dans les composants JavaBean
- Ecrire dynamiquement les données, comme les entrées de formulaire, dans les composants JavaBean
- Invoquer d'une manière arbitraire les méthodes « static » et « public » des Managed Bean
- Traiter dynamiquement les opérations arithmétiques

- EL accepte deux forme d'évaluation de la syntaxe :

- Évaluation immédiate (read-Only)
  - L'expression est évaluée une fois lorsque et le résultat retourné au premier appelle de la page.
    - <c:if test="${userManagedBean.utilisateur.age > 12}"></c:if>
- Évaluation différée (read-Write)
  - Elles signifie que c'est à la technologie qui utilise EL d'utiliser ses propres mécanismes pour évaluer l'expression
    - <h:inputText id="name" value="#{userManagedBean.utilisateur.age}" />
- NB: Il est ainsi recommandé d'utiliser le dernier en JSF

Converters, Actions, Listeners, and Validators

Converters, Actions, Listeners, and Validators

- **Les Actions** représentent les actions de l'application, à invoquer quand le composant est activé par l'utilisateur(sumbit de formulaire).
  - L'expression doit évaluer une méthode "public" qui ne prend pas d'arguments et retourne un "String " permettant de rediriger vers une autre page (ou la meme)

- Les **Converters** sont utilisés pour convertir les données qui sont reçus à partir des composantes d'entrée.

- Les **Listeners** sont utilisés pour écouter les événements qui se passent dans la page et effectuer des actions telles que définies.

- **Validators** sont utilisées pour valider les données qui sont reçues à partir des composantes d'entrée.

Les Actions et ActionListeners

Les Actions:

La navigation peut être directe (appel à la page XHTML sans passer par le Managed Bean) ou indirecte en passant par le Managed bean.

- Attributs 'action' et 'outcome'
- Les composants h:commandButton et h:commandLink utilisent l'attribut 'action' car ils sont en général dans des formulaire ;
- Les composants h:button et h:link utilisent l'attribut 'outcome' pour la navigation.

```
<h:commandButton  action="#{userManagedBean.login()}" value="Valider"/>
```

```
<h:commandLink action="#{userManagedBean.login()}" value="Cliquer"/>
```

```
<h:commandLink action="nextpage"  value="Cliquer"/>
```

Les Actions et ActionListeners

Les ActionListeners

- Traitement asynchrone de tâches à l'activation des composants : AJAXX
- S'exécutent avant les actions
  - <h:commandXxx ... actionListener="#{bean.actionListener}" />
  - **Dans le bean: comme dans les SWING**

```
public void actionListener(ActionEvent event) {
        // ...
}
```

les techniques de passage de paramètres

- Method expression (JSF 2.0)
  - <h:commandButton action="#{user.editAction(15)}" />

- f:param : via request parameter
  - <h:commandButton action="#{user.editAction}">
    <f:param name="action" value="15" />
    </h:commandButton>
  - String action =FacesContext.getExternalContext().getRequestParameterMap().get("action");
- f:attribute :via action listener
  - <h:commandButton action="#{user.editAction}"
    actionListener="#{user.attrListener}">
    <f:attribute name="action" value="15" />
    </h:commandButton>
  - action = (String)event.getComponent().getAttributes().get("action");
- f:setPropertyActionListener bean property(setXxx())
  - <h:commandButton action="#{user.editAction}" >
    <f:setPropertyActionListener target="#{user.action}" value="15" />
    </h:commandButton>

Converters

- Ils sont utilisés pour convertir les données reçues des formulaires :

Package javax.faces.convert

111

Class Summary

Class	Description
BigDecimalConverter	Converter implementation for java.math.BigDecimal values.
BigIntegerConverter	Converter implementation for java.math.BigInteger values.
BooleanConverter	Converter implementation for java.lang.Boolean (and boolean primitive) values.
ByteConverter	Converter implementation for java.lang.Byte (and byte primitive) values.
CharacterConverter	Converter implementation for java.lang.Character (and char primitive) values.
DateTimeConverter	Converter implementation for java.util.Date values.
DoubleConverter	Converter implementation for java.lang.Double (and double primitive) values.
EnumConverter	Converter implementation for java.lang.Enum (and enum primitive) values.
FloatConverter	Converter implementation for java.lang.Float (and float primitive) values.
IntegerConverter	Converter implementation for java.lang.Integer (and int primitive) values.
LongConverter	Converter implementation for java.lang.Long (and long primitive) values.
NumberConverter	Converter implementation for java.lang.Number values.
ShortConverter	Converter implementation for java.lang.Short (and short primitive) values.

- **<h:inputText converter="javax.faces.convert.IntegerConverter" />**

f:convertNumber

S.N.	Attribute & Description
1	**type** number (default), currency , or percent
2	**pattern** Formatting pattern, as defined in java.text.DecimalFormat
3	**maxFractionDigits** Maximum number of digits in the fractional part
4	**minFractionDigits** Minimum number of digits in the fractional part
5	**maxIntegerDigits** Maximum number of digits in the integer part
6	**minIntegerDigits** Minimum number of digits in the integer part
7	**integerOnly** True if only the integer part is parsed (default: false)
8	**groupingUsed** True if grouping separators are used (default: true)
9	**locale** Locale whose preferences are to be used for parsing and formatting
10	**currencyCode** ISO 4217 currency code to use when converting currency values
11	**currency Symbol** Currency symbol to use when converting currency values

```
<h:outputText value="100.12345" >
        <f:convertNumber pattern="#000.000" />
</h:outputText>
```

→ 100.123

f:convertDateTime

S.N.	Attribute & Description
1	**type** date (default), time, or both
2	**dateStyle** default, short, medium, long, or full
3	**timeStyle** default, short, medium, long, or full
4	**pattern** Formatting pattern, as defined in java.text.SimpleDateFormat
5	**locale** Locale whose preferences are to be used for parsing and formatting
6	**timeZone** Time zone to use for parsing and formatting

```
<h:inputText id="dateInput" value="#{userData.date}" label="Date" >
        <f:convertDateTime pattern="dd-mm-yyyy" />
 </h:inputText>
```

Custom Converter

- Créer un Converter personnalisé en 3 étapes:

1. Créer une Classe converter en implémentant l'interface *javax.faces.convert.Converter*
2. *Y Implémenter les méthodes* getAsObject() and getAsString() de l'inteface *javax.faces.convert.Converter*
3. *Utiliser une annotation* @FacesConvertor pour assigner une valeur unique au converter

Exemple:


```
@FacesConverter(value = "userConverter")
public class UsersConverter implements Converter {
@Override
  public Object getAsObject(FacesContext facesContext, UIComponent component, String value) {
        Users user= new Users ();
       User.setIdusers(Integer.valueOf(value));
        return user;
  }
@Override
 public String getAsString(FacesContext facesContext, UIComponent component, Object value) {
        Users user = (Users) object;
       Return user.getIdusers().toString();
   }
}
```

```
<h:inputText id="idinput" value="#{userManagedBean.utilisateur}" label="User" >
            <f:converter converterId="userConverter" />
 </h:inputText>
```

Validators

- f:validateLength

  Valider la longueur en chaine de caratères

  - <f:validateLength minimum="5" maximum="8" />
- f:validateLongRange

  Valider la valeur entire (long) comprise en minimum et maximum
  - <f:validateLongRange minimum="5" maximum="200" />
- f:validateDoubleRange

  Valider la valeur réelle (Double) comprise en minimum et maximum
  - <f:validateDoubleRange minimum="1000.50" maximum="10000.50" />
- f:validateRegex

  Validate JSF component with a given regular expression.
  - <f:validateRegex pattern="((?=.*[a-z]).{6,})" />

```
<h:inputText id="nameInput" value="#{userData.name}" label="name" >

<f:validateLength minimum="5" maximum="8" />

</h:inputText>
```

Custom Validator

- Créer un validator personnalisé en 3 étapes:

1. Créer une classe validator en implémentant l'interface *javax.faces.validator.Validator*
2. *Implémenter la méthode validate() de l'interface*
1. *Utiliser une annotation* @Facesvalidator pour assigner une valeur unique au validator ainsi créé

Templating: Facelets

Facelets est un langue puissant de déclaration la page qui est utilisé pour construire des vues JavaServer Faces en utilisant des modèles de style de HTML et de construire des arbres de composants.

Ses caractéristiques sont les suivantes:

- XHTML pour créer des pages web
- Supporte de bibliothèques de balises Facelets en plus de celles de JavaServer Faces et JSTL
- Supporte EL
- Templating pour les composants et pages

Templating: Facelets

- Avantages de Facelets pour des projets de développement à grande échelle sont les suivants:
  - Supporte la réutilisation du code travers les templates et les composites composants
- Extensibilité fonctionnelle des composants et autres objets côté serveur grâce à la personnalisation
- Plus rapide temps de compilation et de validation du EL
- Rendu haute performance

Tag Libraries Supported by Facelets

Tag Library	URI	Prefix	Example	Contents
Facelets	http://java.sun.com/jsf/facelets	ui:	ui:component ui:insert	templating
JSF HTML	http://java.sun.com/jsf/html	h:	h:head h:body h:outputText h:inputText	All UIComponent objects
JSF Core	http://java.sun.com/jsf/core	f:	f:actionListener f:attribute	actions that are independent of any particular render kit
JSTL Core	http://java.sun.com/jsp/jstl/core	c:	c:forEach c:catch	JSTL 1.2 Core Tags
JSTL Functions	http://java.sun.com/jsp/jstl/functions	fn:	fn:toUpperCase fn:toLowerCase	JSTL 1.2 Functions Tags

template tags

S.N.	Tag & Description
1	**ui:insert** Used in template file. It defines contents to be placed in a template. ui:define tag can replaced its contents.
2	**ui:define** Defines the contents to be inserted in a template.
3	**ui:include** Includes contents of one xhtml page into another xhtml page.
4	**ui:composition** Loads a template using **template** attribute. It can also define a group of components to be inserted in xhtml page.

1 –un template est formé de plusieurs composites(Header, Footer, content, …)

Exemple: header.xhtml

```
<?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:ui="http://java.sun.com/jsf/facelets">
        <body>
              <ui:composition>
                                    <h1>texte from header composite</h1>
              </ui:composition>
       </body>
</html>
```

2- inserer les composites dans le template

```
<ui:include src="/header.xhtml" />
```

3- Définir dans le template, des zones (HTML) à remplacer par la page qui chargera le Template

```
<ui:insert name="header" >
 <ui:include src="header.xhtml" />
</ui:insert>
```

le Template: template.xhtml

```
<?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:ui="http://java.sun.com/jsf/facelets">
        <h:head>
       </h:head>
       <h:body>
              <div style="border-width:2px; border-color:green; border-style:solid;">
                    <ui:insert name="header" >
                                  <ui:include src="header.xhtml" />
                    </ui:insert>
             </div>
       </h:body>
</html>
```

Chargement du Template et redéfinition des zone d'insertion

```
<?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:ui="http://java.sun.com/jsf/facelets">
   <h:body>
     <ui:composition template="template.xhtml">
        <ui:define name="header">
                <h2>remplacement du contenu par défaut</h2>
          </ui:define>
     </ui:composition>
   </h:body>
</html>
```

Chargement du Template

redéfinition des zone d'insertion

Y'avait <ui:insert name="header" > dans le template.xhtml

L'internationalisation

L'internationalisation est une technique pour laquelle les messages d'état, les étiquettes des composants de l'interface graphique, la monnaie, la date ne sont pas codés en dur dans le programme.

Ils sont stockés en dehors du code source dans des regroupements de ressources et récupérés dynamiquement.

JSF fournir un moyen très pratique pour gérer ces ressources.

Création des fichiers propriétés

« messages_<LANG>.properties »

Lors de la connexion d'un visiteur, il est possible de connaître la langue de celui-ci grâce à la variable LANG dans les entêtes de la requête HTTP.

La langue du visiteur se retrouve codée LANG = :

- fr Français
- en Anglais
- fr_FR pour la France
- fr_CA pour le Canada
- en_US pour les USA
- en_UK pour la Grande Bretagne
- wo_SN pour le wolof Sénégal

login= Identifiant
pass = mot de passe

messages_fr.properties

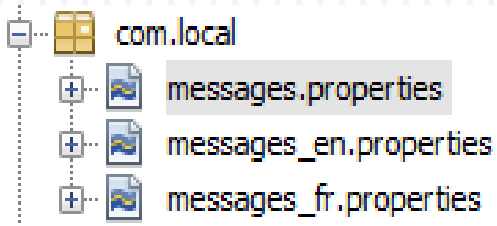
login= Login
pass = PassWord

messages_en.properties

login= Mandarga
pass = Sa thiabi

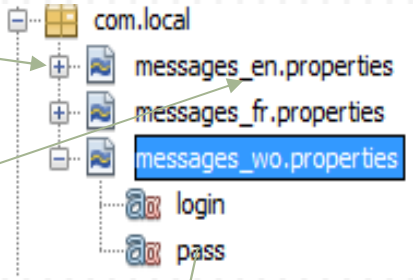
messages.properties

Créer ces fichiers dans dans le package com.local



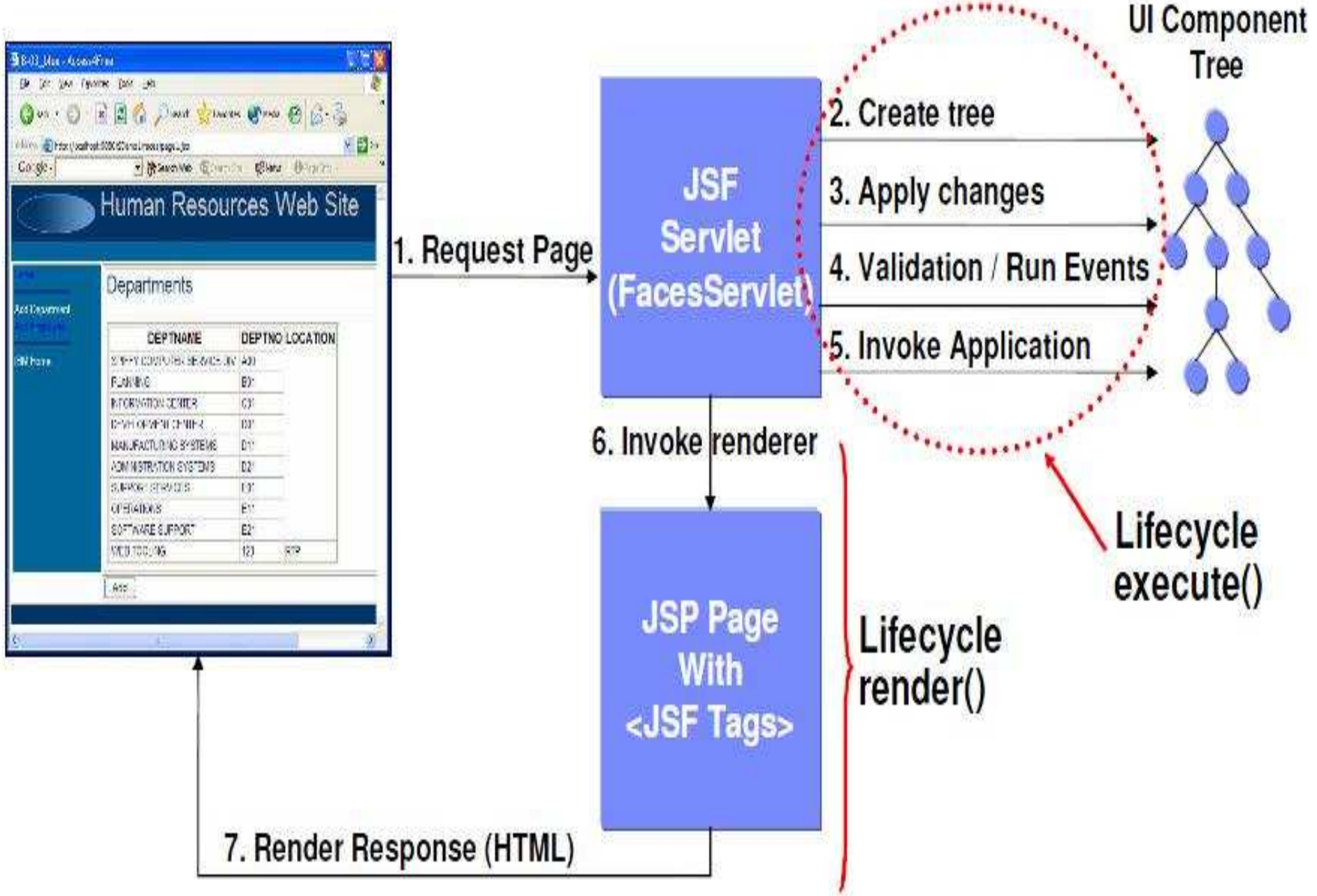
Déclaration des fichiers propriétés dans faces-config.xml


```
<application>
        <resource-bundle>
            <base-name>com.local.messages</base-name>
            <var>msg</var>
        </resource-bundle>
    <!-- Ajout de la langue par défaut et des langues supportées -->
        <locale-config>
            <default-locale>wo</default-locale>
            <supported-locale>fr</supported-locale>
            <supported-locale>en</supported-locale>
        </locale-config>
    </application>
```



Dans les JSF page:
#{msg.pass}

```
<h:form >
        <h:outputLabel value="#{msg.login}"/><br/>
        <h:inputText  value="#{userManagedBean.utilisateur.login}" /><br/>
        <h:outputLabel  value="#{msg.pass}"/><br/>
        <h:inputSecret  value="#{userManagedBean.utilisateur.password}"/><br/>
        <h:commandButton  action="#{userManagedBean.login()}" value="Valider"  />
</h:form>
```


Human Resources Web Site
Departments
DEPTNAME
DEPTNO
LOCATION
1. Request Page
JSF
Servlet
(FacesServlet)
2. Create tree
3. Apply changes
4. Validation / Run Events
5. Invoke Application
UI Component
Tree
6. Invoke renderer
JSP Page
With
<JSF Tags>
7. Render Response (HTML)
Lifecycle
execute()
Lifecycle
render()

- Cours Sur Primefaces (Nous y reviendrons)
- Documentation
- DEMO page
- PF Mobile

Enterprise Java Bean: EJB3

Enterprise Java Bean: EJB3

- Les spécifications se basent sur l'expérience accumulée des développements J2EE précédents.
- Ces spécifications ont pour objectif de simplifier au maximum le développement des EJB et éviter les facteurs bloquants des anciennes spécifications.

Enterprise Java Bean

- Les EJB

  facilitent la création d'applications distribuées pour les entreprises.

  - S'occupent du traitement métier de l'application
  - Permettent aux développeurs de se concentrer sur les traitements orientés métiers
  - Sont réutilisables
  - Sont assemblables

Enterprise Java Bean

- Composant serveur qui encapsule une logique métier, qui peut être déployé dans un serveur d'application
- Composé de un ou plusieurs objets
- Les appels aux méthodes par les clients de l'EJB sont interceptés par le conteneur d'EJB

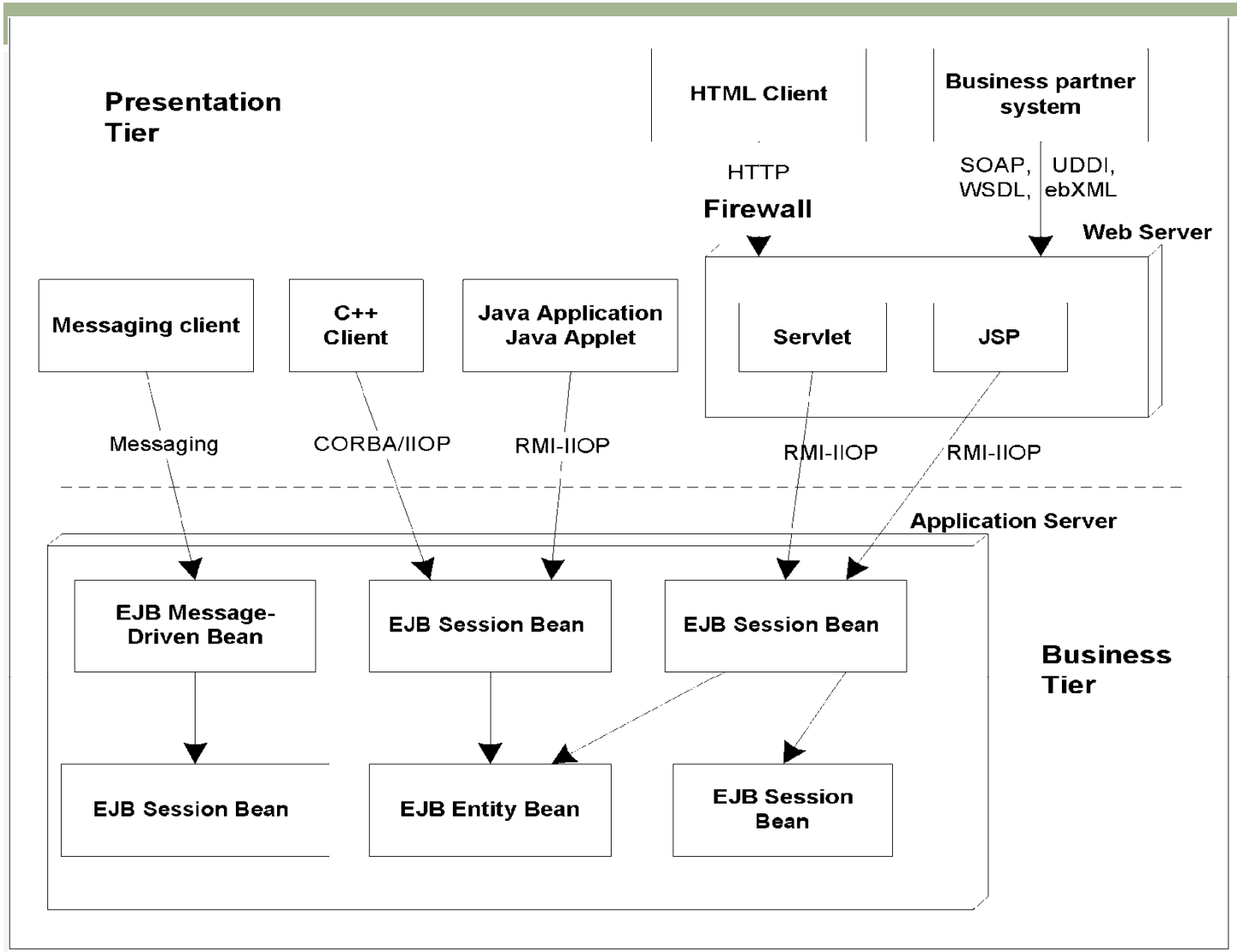
Rôle du conteneur

- Le conteneur d'EJB s'occupe de certains traitements
  - Cycle de vie du bean
  - Injection de dépendance
  - Accès au bean, communication à distance
  - Sécurité d'accès
  - Accès concurrents
  - Transactions, …

2 Types d'EJB

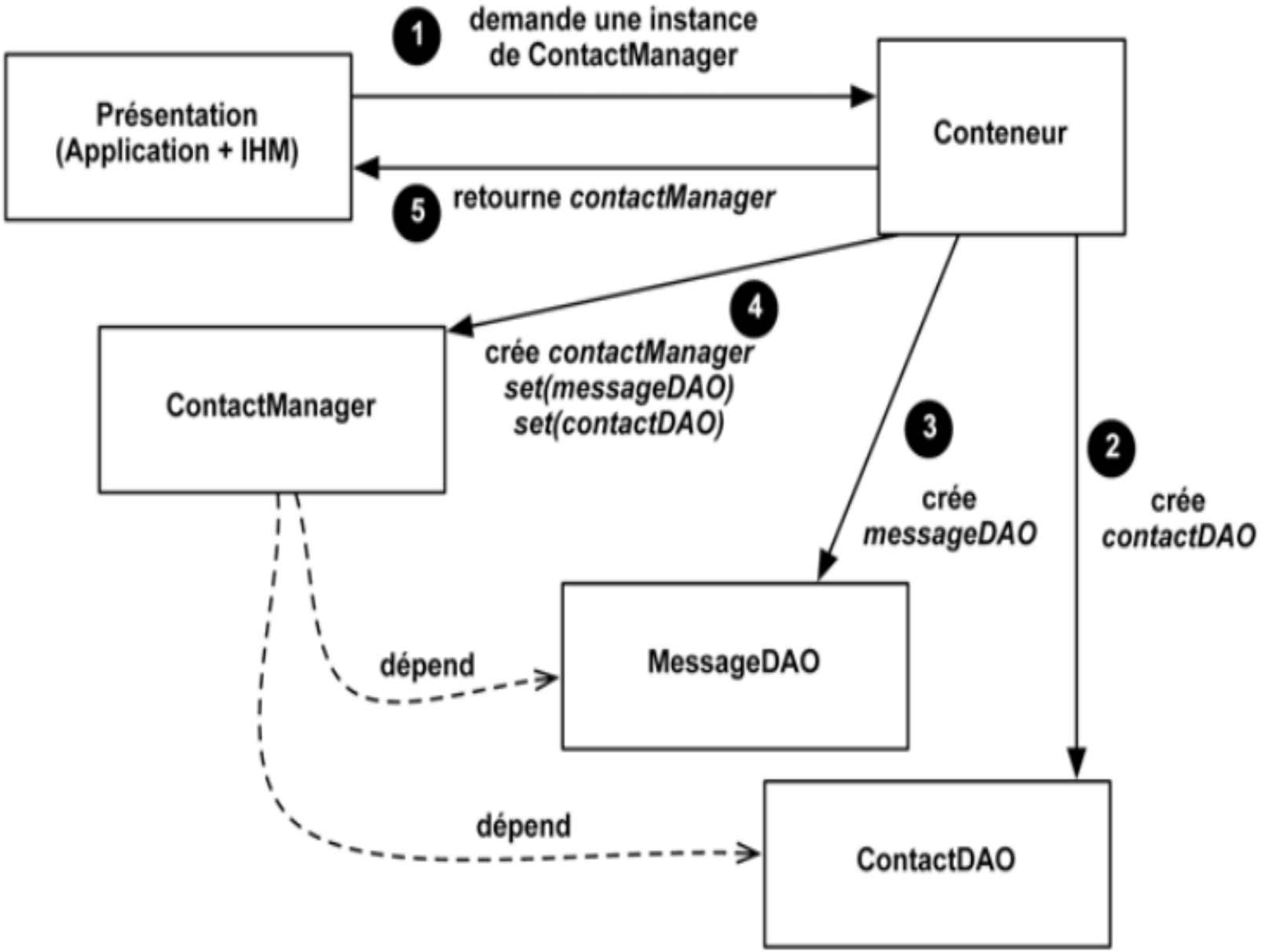
- **Session Bean**
  - Un Session Bean est une application côté serveur permettant de fournir un ou plusieurs services à différentes applications clientes.
  - Modélise un traitement
  - Représenté par une classe Java et une interface qui expose certaines méthodes
- **Message Driven Bean (MDB)**
  - Consomme des messages asynchrones envoyés par des clients
  - Permettent l'interconnexion avec des systèmes différents (non Java EE)

Clients interagissant avec un serveur à base d'EJBs



Injection de dépendances

- Le couplage des composants est un problème récurrent au sein d'une application.
- Pour remédier à ces difficultés de développement, le concept du design pattern « IoC » (Inversion of Control) a été directement intégré depuis Java EE 5.
  - Celui-ci permet de réduire plus facilement le couplage d'une architecture grâce à un conteneur spécialisé :
  - Le conteneur est responsable de la création des objets ; il effectue lui-même l'instanciation (l'opération « new »).
  - Le conteneur résout les dépendances entre les objets qu'il gère.
  - C'est à lui de définir le cycle de vie des objets qu'il construit


1
demande une instance
de ContactManager
Présentation
(Application + IHM)
Conteneur
5
retourne contactManager
4
crée contactManager
set(messageDAO)
set(contactDAO)
ContactManager
3
crée
messageDAO
2
crée
contactDAO
dépend
MessageDAO
dépend
ContactDAO

Session Bean

- Modélise un traitement (business process)
- Correspond à un *verbe*, à une *action*
- Ex : gestion de compte bancaire, affichage de catalogue de produit, vérifieur de données bancaires, gestionnaire de prix…
- Les actions impliquent des calculs, des accès à une base de données, consulter un service externe (appel téléphonique, etc.)
- Les Session Beans sont des services qui jouent le rôle d'intermédiaires entre les applications clientes et l'accès aux données.

3 types de Session Bean

- Bean sans état (*stateless*)
  - Pour traiter les requêtes de plusieurs clients,
  - sans garder un état entre les différentes requêtes
  - Exemple : obtenir la liste de tous les produits
- Bean avec état (*stateful*)
  - Pour tenir une conversation avec un seul client,
  - en gardant un état entre les requêtes
  - Exemple : gestion des objets appartenant à l'utilisateur connecté

3 types de Session Bean

- Bean singleton
  - Garantie de n'avoir qu'une seule instance du bean dans tout le serveur d'application
  - Supporte les accès concurrents (configurable)
  - Exemple : bean qui « cache » une liste de pays, utilisé par les classes de l'application pour éviter d'interroger la BD

Comment développer un stateless bean

- Un stateless bean peut avoir une interface distante et/ou locale.
- L'interface distante (@javax.ejb.Remote) permet aux clients distants d'invoquer des méthodes de l'EJB. Les paramètres des méthodes sont ainsi copiés, sérialisés, puis transmis à l'EJB. On appelle ce mécanisme l'appel par valeur (call-by-value).
- L'interface locale (@javax.ejb.Local) est utilisée par les objets résidants dans la même JVM que l'EJB. Les paramètres des méthodes ne sont pas recopiés mais passés par référence (call-by-reference).

```
@Remote
public interface UsersRemote {
  Users login(string login, String pass) ;
}
```



```
@Local
public interface UsersLocal {
  Users login(string login, String pass) ;
}
```

```
@Stateless(name = “UsersSB",
        mappedName = "ejb/stateless/Users")

public class UsersBean implements UsersRemote,UsersLocal {

}
```

L'injection

En Java EE 5, ces références sont instanciées par le conteneur lui-même, puis injectées dans les objets managés. Ce mécanisme est appelé injection (en anglais Dependency Injection, ou Inversion of Control - IoC):

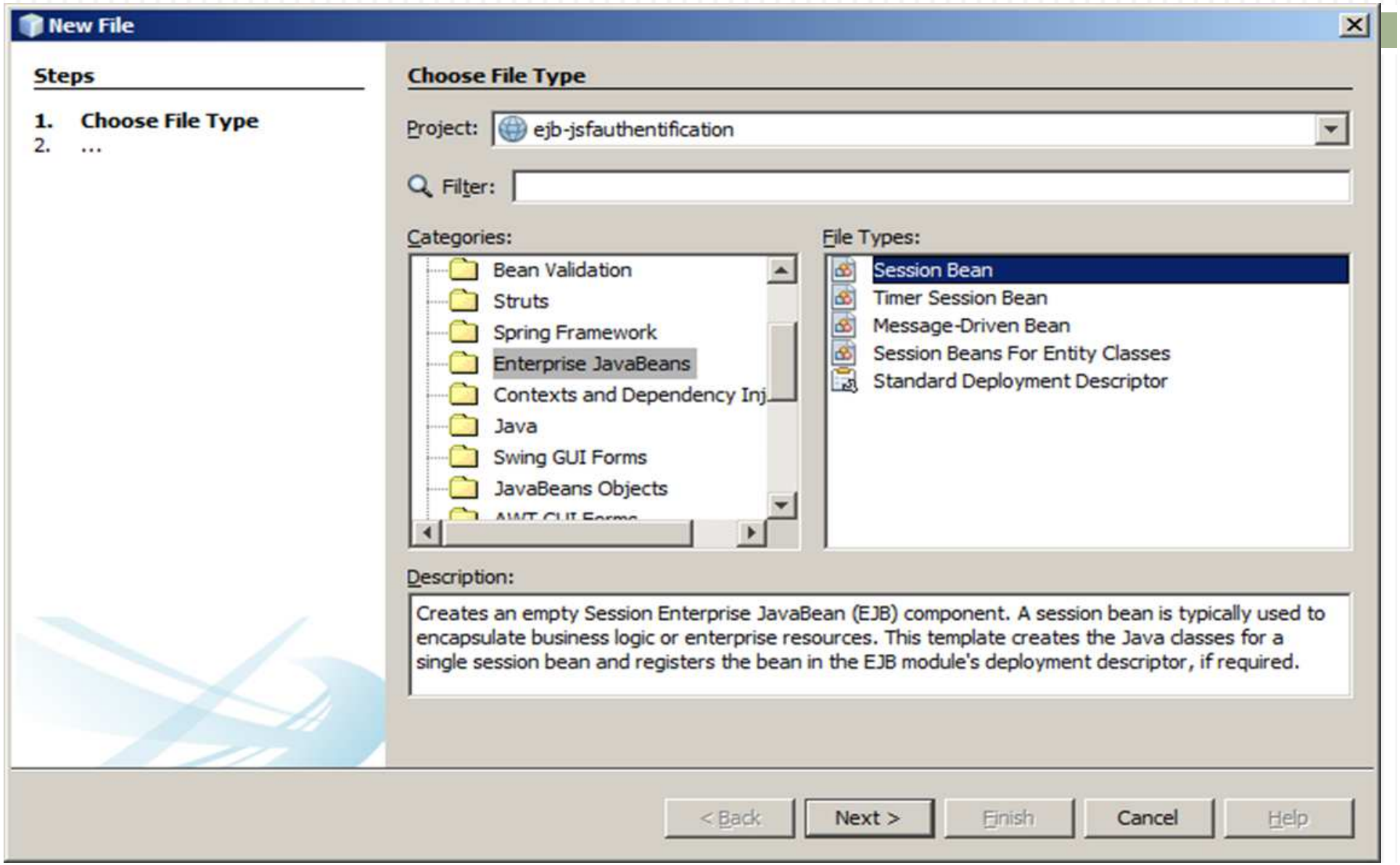
```
@EJB(beanName=«ejb/stateless/Users»)
UsersLocal usersFacade;
```

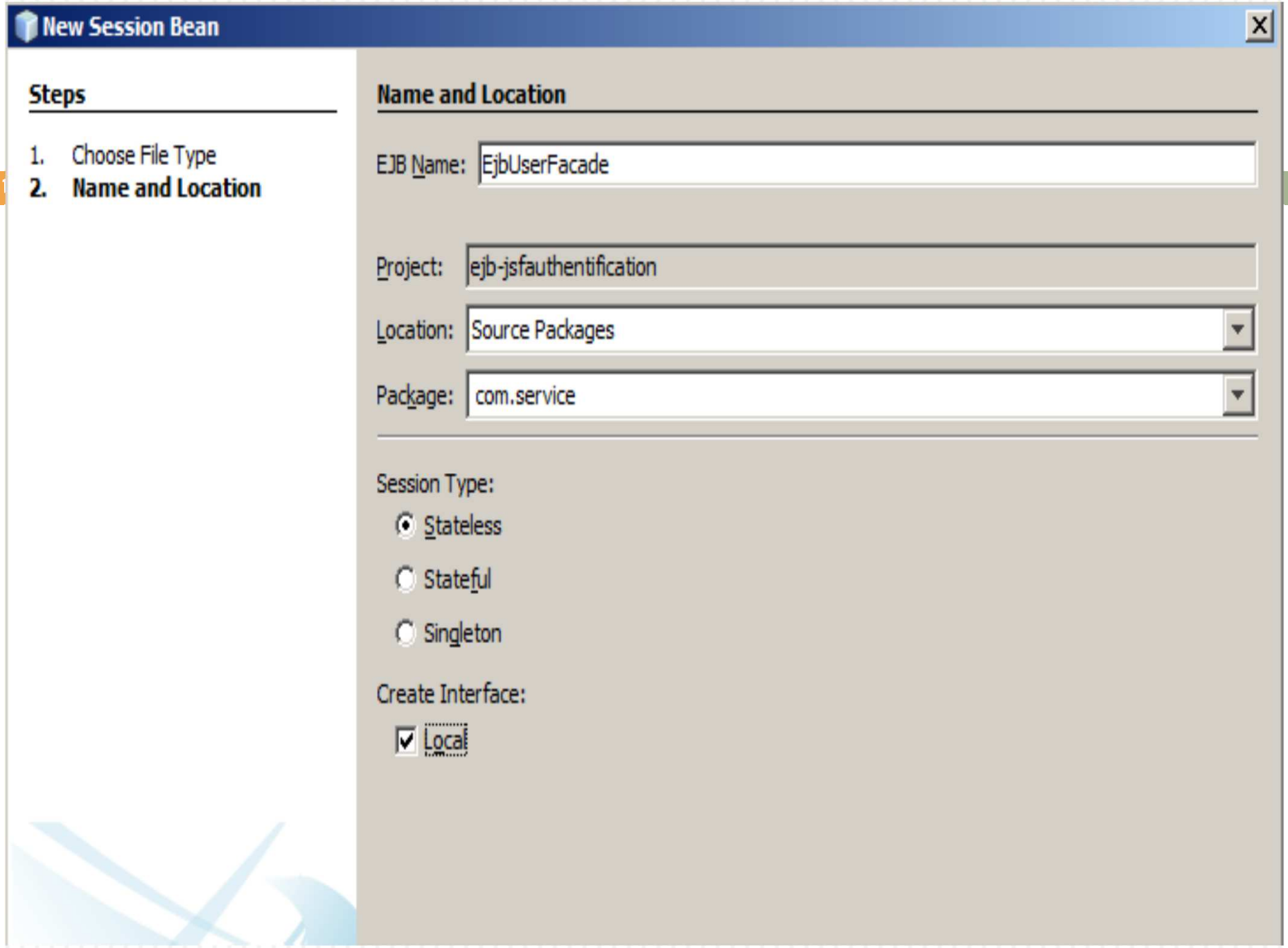
Cette injection de dépendance ne peut être utilisée que par des objets pris en compte par le conteneur.

- Dans les autres cas on fait recours aux lookups JNDI (voir EJB2 ou RMI)

- Les managed beans, quant à eux, s'exécutent à l'intérieur du conteneur web et peuvent donc utiliser le mécanisme d'injection. JNDI est, bien sûr, toujours sollicité mais de façon plus discrète. Le développeur n'a plus besoin de manipuler directement l'API JNDI.
- L'injection de l'EJB se fait grâce à l'annotation @javax.ejb.EJB.

Un clic-droit sur le package ad-hoc




New Session Bean
Steps
1. Choose File Type
2. Name and Location
Name and Location
EJB Name: EjbUserFacade
Project: ejb-jsfauthentification
Location: Source Packages
Package: com.service
Session Type:
Stateless
Stateful
Singleton
Create Interface:
Local

```java
@Local
public interface EjbUserFacadeLocal {
    /**
     * cette méthode permettra de connecter un user partir de son login et passwor
     * l'objet user est un objet générique dont les champs login et password sont r
     * 1. Pour faire le travail demandé , aller dans la classe d'implémentation
     * @param user
     * @return le user si cela reussi , sinon il retourne "null"
     */
    public  Users connect(Users user);
}
```

```
@Stateless
public class EjbUserAction implements EjbUserActionLocal {

    @Override
    public Users connect(Users user) {
        try {
            Class.forName("com.mysql.jdbc.Driver").newInstance();
            Connection conn = DriverManager.getConnection("jdbc:mysql://localhost



            PreparedStatement prep1 = conn.prepareStatement("SELECT * FROM users
            prep1.setString(1, user.getLogin());
            prep1.setString(2, user.getPassword());
            ResultSet rs = prep1.executeQuery();
```

```java
@Named(value = "userManagedBean")
@SessionScoped
public class UserManagedBean implements Serializable {

    @EJB
    EjbUserFacadeLocal ejbUserAction;
```