

Développement avec Java EE

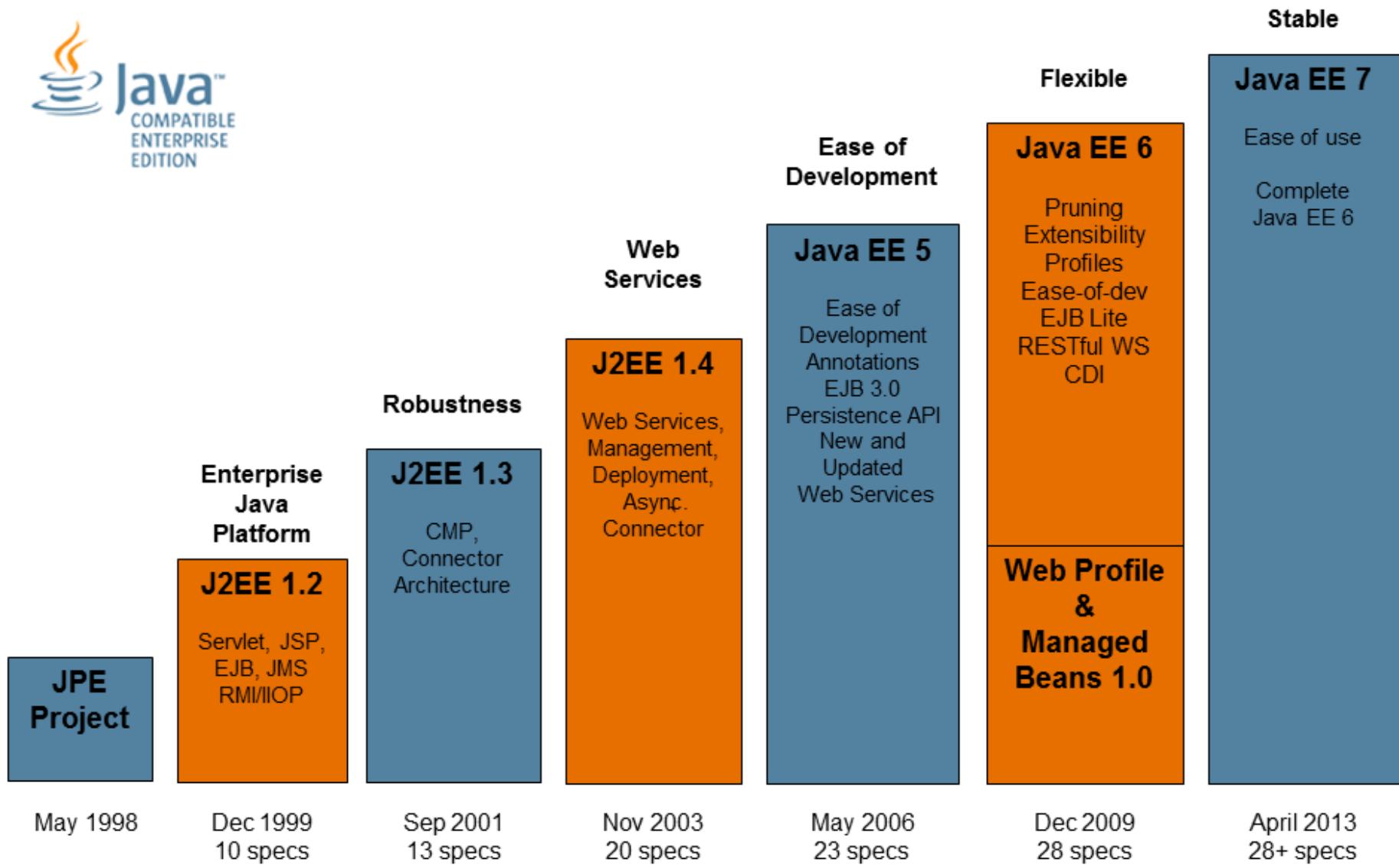
- Servlets, JSP, Taglibs -
- Fonctions avancées -

Nantes – 6 et 7 juillet 2015

- Qui êtes-vous ?
- Pourquoi avoir choisi cette formation ?
- Quels conseils, quelles informations et compétences souhaitez-vous obtenir grâce à cette formation ?
- Les objectifs cités correspondent-ils à vos attentes ?
- Quelles sont les compétences qui vous semblent nécessaires pour suivre ce cours ?
- Êtes vous venu(e) avec des questions ?

- 1 - Présentation de la technologie JEE**
- 2 - Conception et développement des Servlets**
- 3 - Conception et développement de pages JSP**
- 4 - Taglibs, EL et JSTL**
- 5 - Fonctionnalités avancées en JEE**

1 - Présentation de la technologie JEE**2 - Conception et développement des Servlets****3 - Conception et développement de pages JSP****4 - Taglibs, EL et JSTL****5 - Fonctionnalités avancées en JEE**



- JEE : Java Enterprise Edition (ex J2EE)

**Une technologie
outils liés au langage Java
+ des spécifications**

ET

**Un modèle de développement
applications découpées en tiers**

- **Une technologie:**
 - Le langage Java
 - La machine virtuelle (JVM)
 - Des APIs (le JDK + APIs applicatives)
 - Des serveurs respectant le standard JEE (JSR)



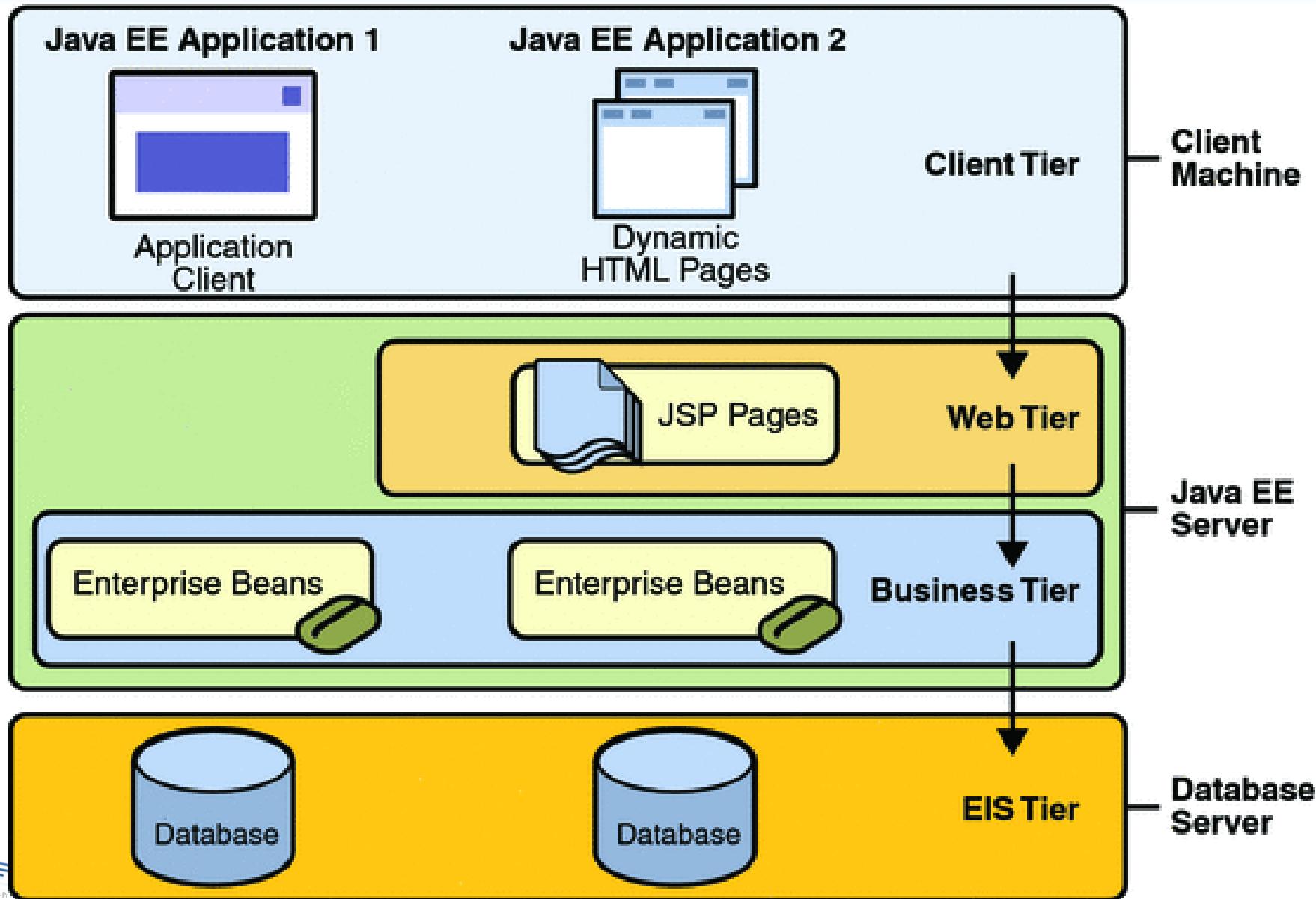
Plate-forme JEE

- **Un modèle de développement:**

Développement en tiers (multitiers) : applications découpées logiquement (correspondance avec le déploiement : clients, serveurs, SGBDs,...)

Ce modèle partitionne le travail en 2 parties :

- Les aspects métiers/présentation, à la charge du développeur
- Les services standards fournies par la plate-forme JEE



Les composants d'une application Java EE sont considérées suivant 4 tiers différents :

- **Le tier client (Client-tier)** : la partie tournant sur le client.
- **Le tier Web (Web-tier)** : des composants qui s'exécutent sur le serveur Java EE.
- **Le tier Métier (Business-tier)** : des composants qui s'exécutent sur le serveur Java EE.
- **Le tier système d'information (Enterprise information system (EIS)-tier)** : le logiciel appartenant au système d'information et s'exécutant sur le serveur correspondant (EIS server).

- Les applications JEE sont considérées comme des **applications 3-tiers** car elles sont distribuées sur **3 localisations** (virtuelles) différentes :
 - les machines clientes
 - le serveur JEE
 - les systèmes d'informations (Bds, etc.)
- C'est donc une extension du modèle 2-tiers classique client/serveur : *ajout d'une couche applicative entre client et SIs*

Terminologie JEE

JEE-components et JEE Containers

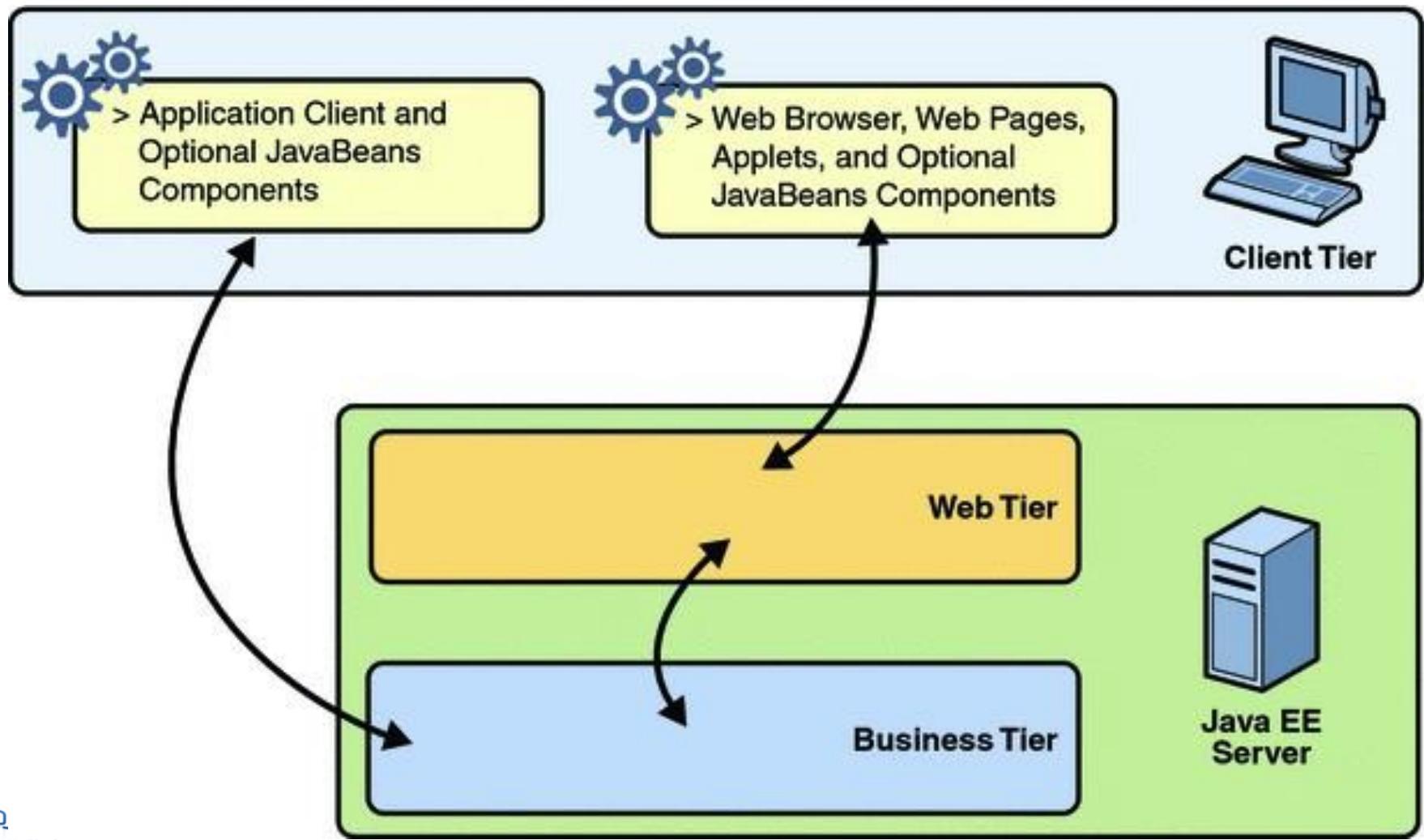
JEE-components

- Les applications JEE sont faites de *composants JEE (Java EE-components)*
- **Java EE-component :**
 - Une unité de programmation autonome, un composant, pouvant communiquer avec d'autres composants

- **La spécification JEE distingue 3 types de composants:**
 - 1 - Les applications clientes et les applets : des composants qui tournent sur le client.
 - 2 - Java Servlet, JavaServer Faces, et les JavaServer Pages (JSP) : des composants qui tournent sur le serveur.
 - 3 - Les Enterprise JavaBeans (EJB) : également sur le serveur

Différence avec des classes classiques :

- Vérifient la spécification JEE
- Déployées sur un serveur JEE



. On distingue 2 types de clients JEE :

1 - les clients Web

2 - les applications clientes

Un Client Web est considéré suivant 2 parties distinctes:

- 1) des pages web dynamiques générées par le Web-tier
- 2) un navigateur qui affiche les pages générées

On parle de client léger (thin client) : toutes les opérations complexes sont exécutées par le serveur

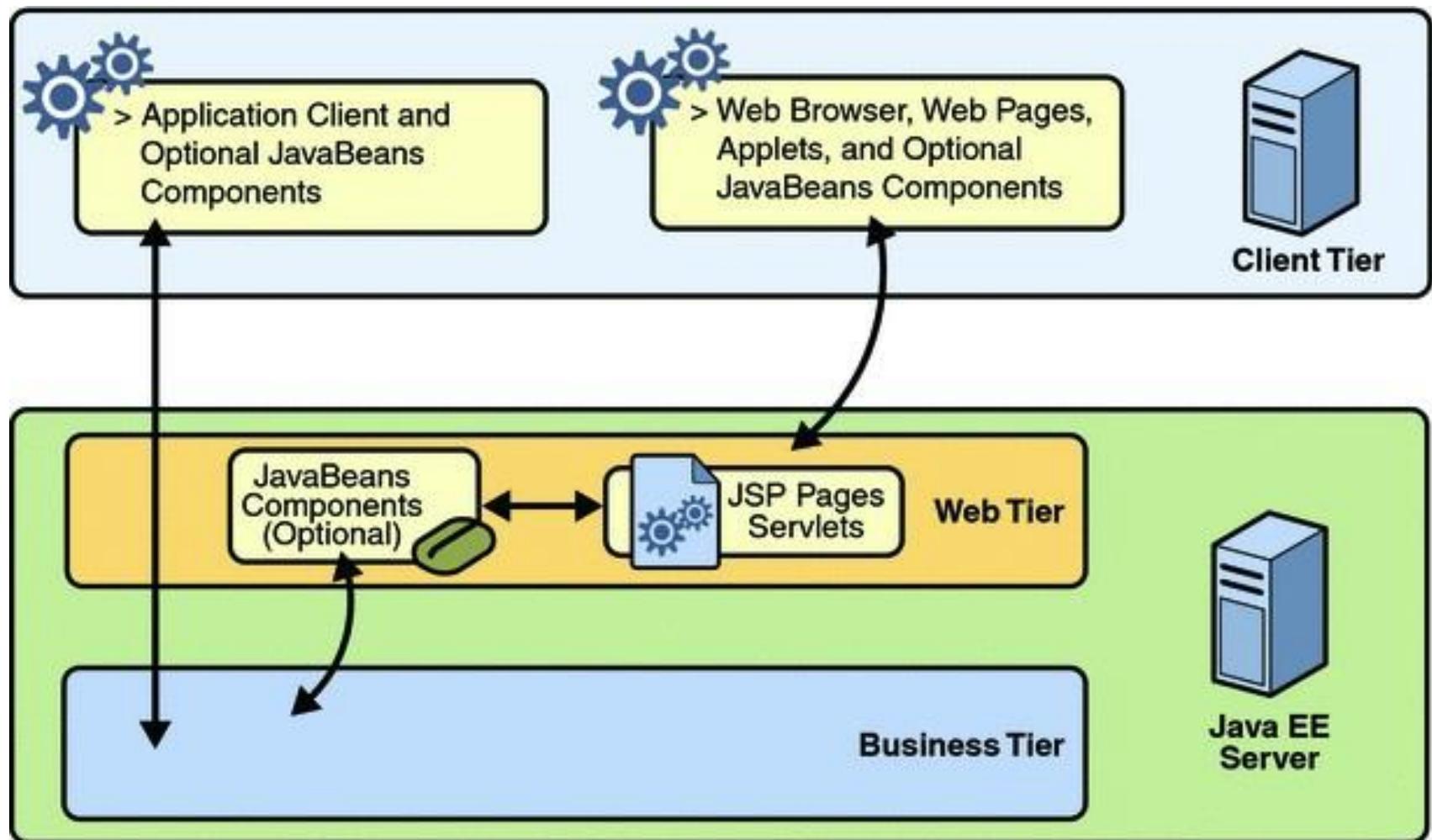
- **Une page web reçue depuis le web-tier peut contenir une applet** : petite application cliente écrite en Java exécutée par le navigateur
 - Nécessite un plugin contenant une JVM
 - Et parfois un fichier contenant des règles de sécurité

On leur préfère aujourd'hui les composants web :

- pas besoin de plugin ou de fichier particulier sur le client
- Séparation nette entre le design de la page et les couches applicatives.

- **Applications clientes (exécutées sur la machine cliente) :**
 - plus riches en terme d'interface utilisateur (swing, etc.)
 - accèdent directement au business-tier
 - mais, peut aussi accéder à des services fournis par le web-tier et ainsi interagir avec d'autres composants web
- **On parle de clients lourds (ou thick clients)**

- Choisir entre client léger ou lourd dépend du type d'applications :
- client léger :
 - facilite la distribution, le déploiement et la maintenance de l'application
 - mais limite l'expérience utilisateur (latence réseau, GUI limité, etc.)



- Les composants Web JEE sont soit :

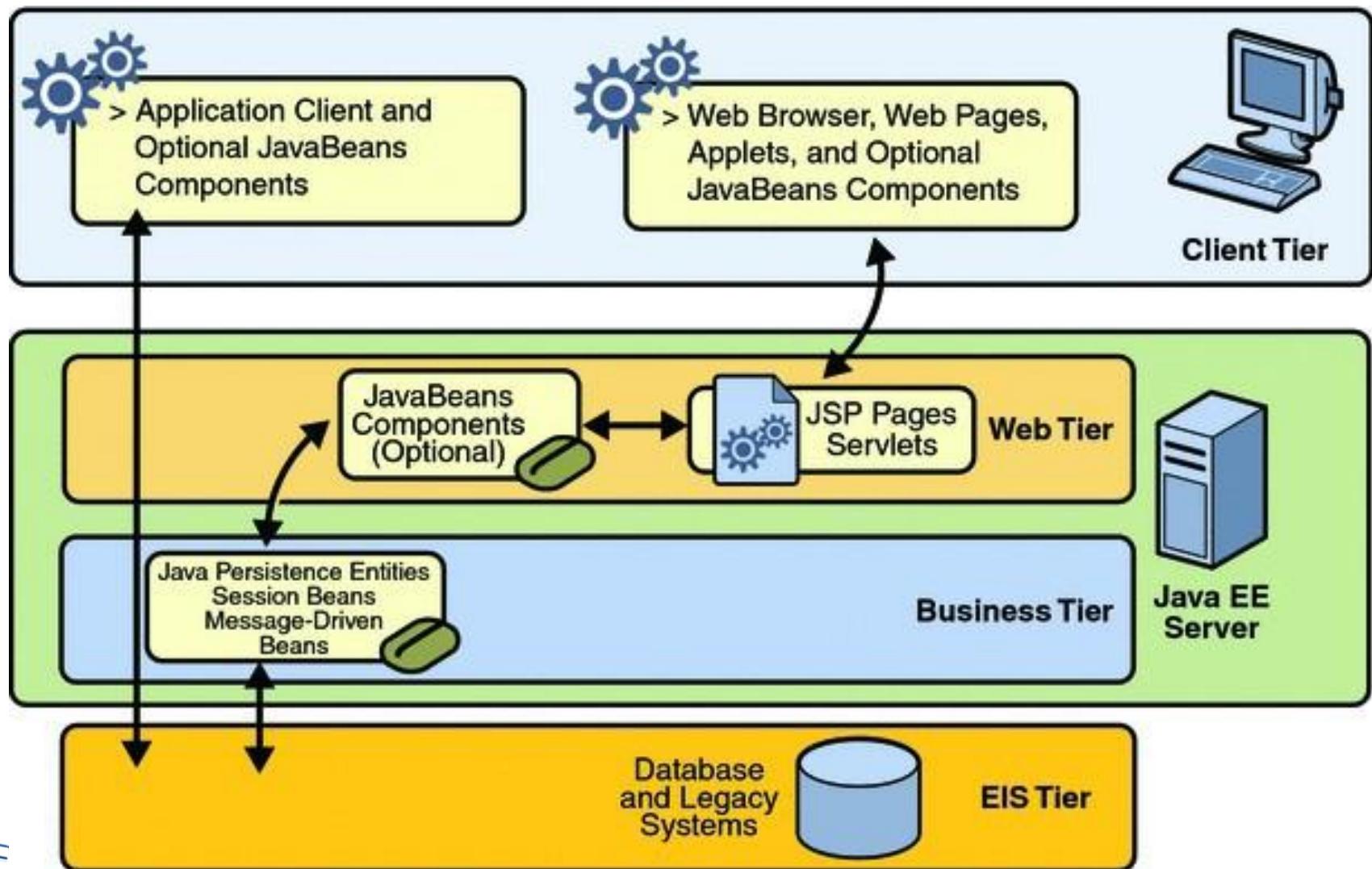
- Des Servlets (des classes Java générant des pages web)
- Soit des pages créées avec la technologie JSP

Les pages JSP : code html statique et des appels à des objets Java (« à la PHP »)

La technologie JavaServer Faces JSF (MVC) : utilise les servlets/JSP et fournit des outils liés à la gestion des interfaces web (analogie composants swing).

Note : Les pages html et les applets ne sont pas considérées comme des composants Web par la spécification JEE

Business-tier et EIS-tier



Business-tier et EIS-tier

Business tier

Les parties liées au domaine d'application (les composants métiers) sont appelés les Business components (ils sont localisés dans le Business-tier)

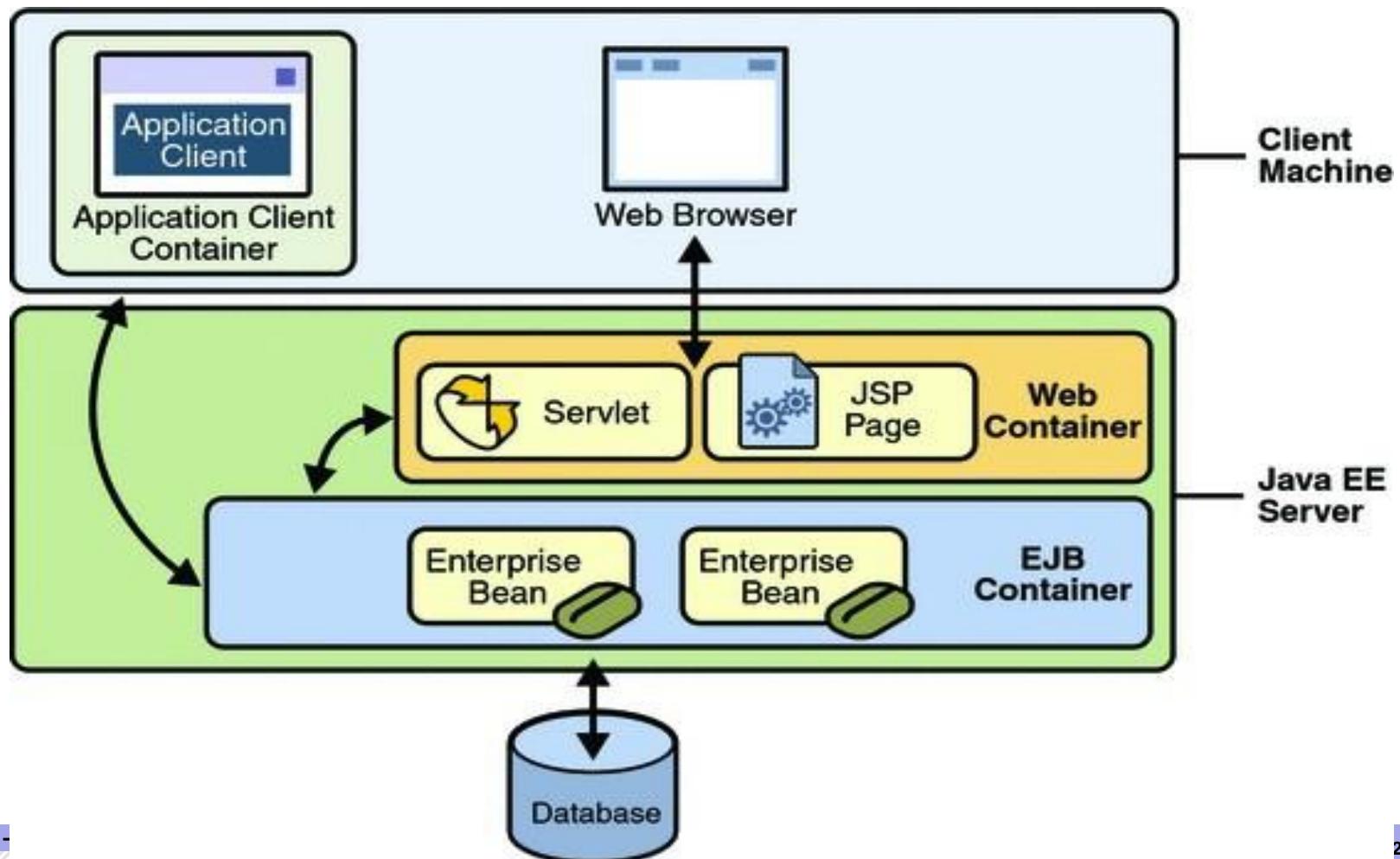
EIS-tier

Le tier lié au système d'information (de l'entreprise) concerne tout le logiciel lié à l'infrastructure de l'entreprise (ERP, système transactionnel, base de données, etc.)

JEE-Containers

- **Le découpage en composants est le premier point fort de la technologie JEE.**
- **Le deuxième est la gestion par conteneur des composants de l'application**
 - pour chaque type de composants, le serveur JEE fournit des services associés si celui-ci est placé dans un conteneur adapté

- Le déploiement va donc consister à placer les composants dans les conteneurs adaptés :

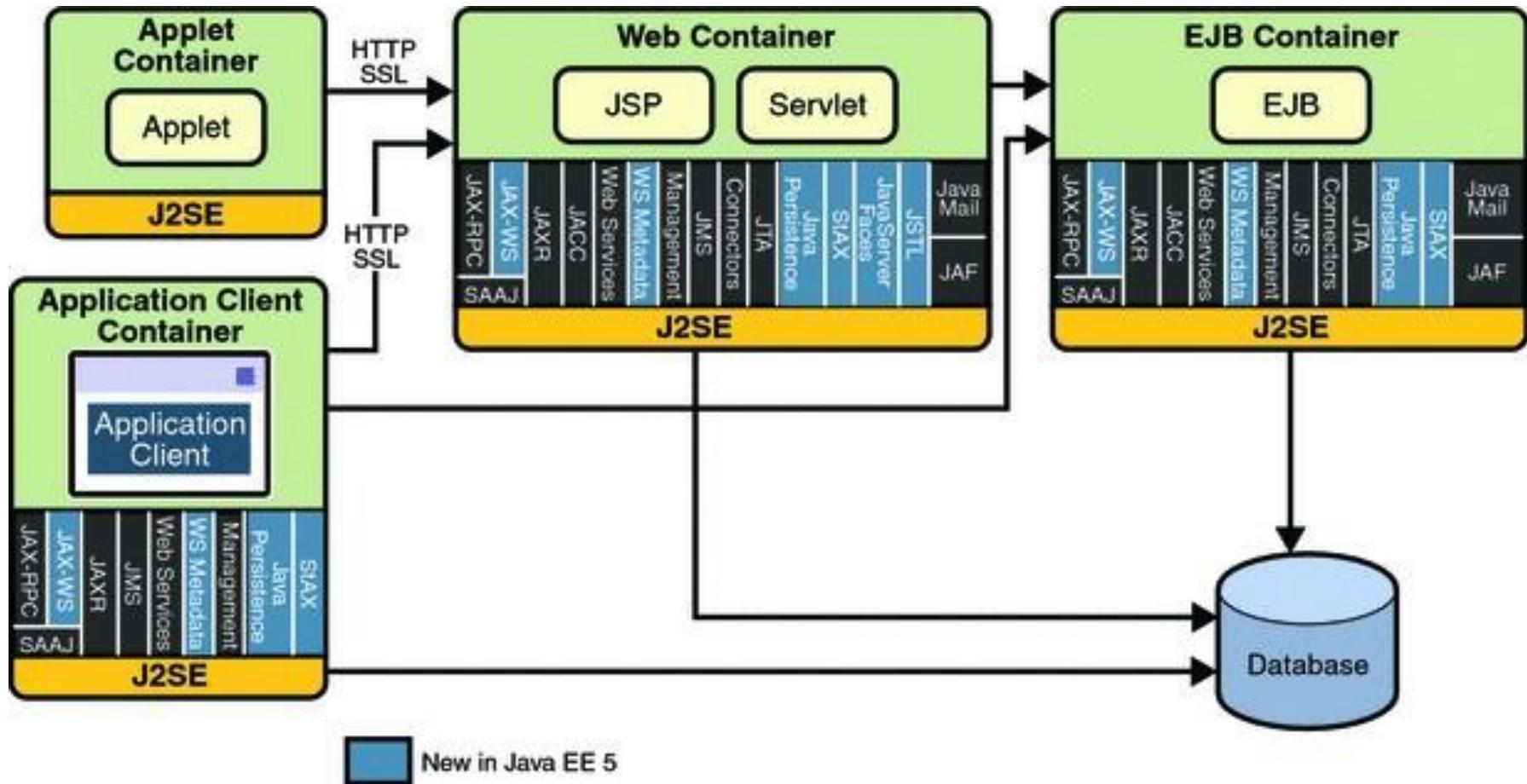


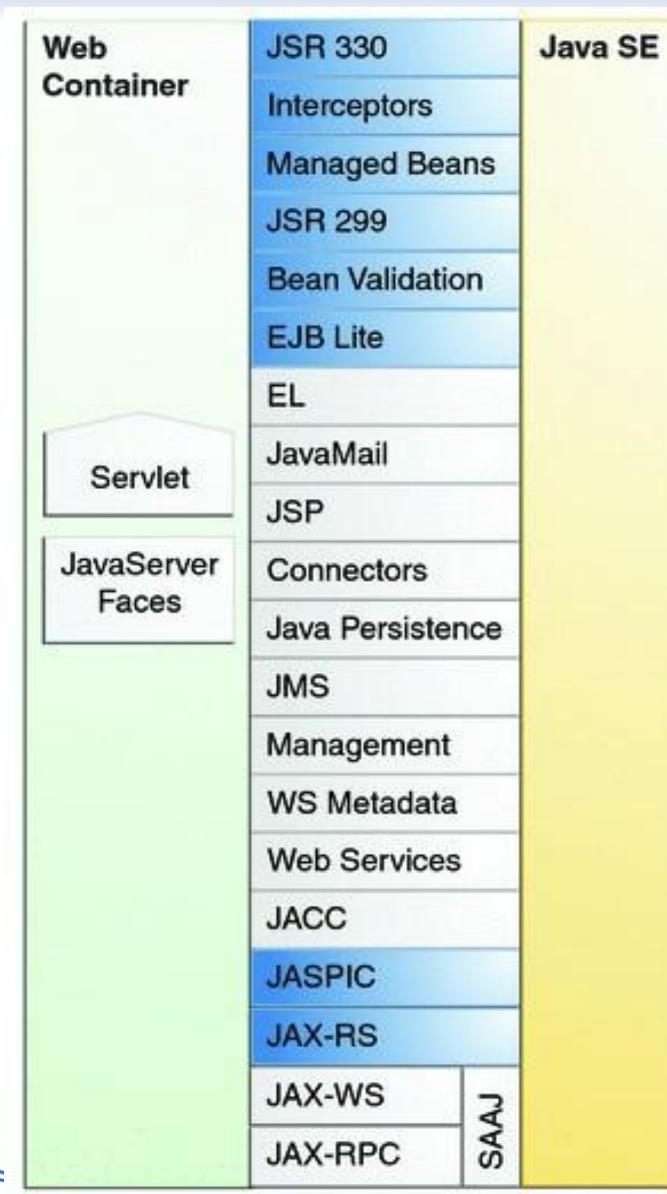
- Un **conteneur (container)** est l'interface entre le composant et les services de bas niveaux dont il peut avoir besoin
- Pour pouvoir être exécutés, un composant ou une application web doit être :
 - 1 - assemblé dans un module JEE et**
 - 2 - déployé dans son conteneur.**

Le processus d'assemblage consiste principalement à paramétrer les services fournis par le conteneur :

- Pour chaque composant
- Et pour l'application elle-même.

Par ex., il s'agit de définir les services de sécurité (login par ex.), d'espace de nommage, etc.







→ Implémentations Java EE 6

Serveurs open source



Tomcat
(implém. partielle)



Serveurs commerciaux



Websphere



Weblogic



Application
Server



- Présentation des composantes JEE**
- Présentation détaillée des composantes Web**
 - Servlets et pages JSP
- Présentation des EJB**
- Les formats de déploiement JEE**

□ JNDI

- Accès unifié à des services d'annuaire ou de répertoire (e.g. LDAP)
- Récupérer toute sorte de ressources, à distance, à partir d'un nom ou d'une référence
 - Permet d'isoler le code client d'informations spécifiques à l'environnement d'exploitation (chaîne de connexion à une base, ...)

□ RMI

- Communication entre objets répartis (objets Java uniquement)
 - Protocole sous-jacent : JRMP ou IIOP

□ JDBC

- Accès aux bases de données relationnelles

□ **JTA (Java Transaction Api)**

- pilotage de transaction, généralement masqué par l'utilisation des EJB

□ **JMS (Java Messaging Service)**

- Interaction avec des middlewares orientés messages (MQSeries...)
- Utilisé notamment par un nouveau type d'EJB : les Message Beans

□ **JAAS (Java Authentication and Authorization Service)**

- Gestion de l'authentification et des droits des utilisateurs

□ **JDBC Extension**

- JDBC 2.0 définit la notion de DataSource :
- Package javax.sql
- Support des pools de connexions de façon portable
- Utilisation de JNDI pour retrouver la DataSource

❑ JavaMail

- Envoi et récupération de mails
- Support de SMTP, POP3, IMAP, pièces jointes...

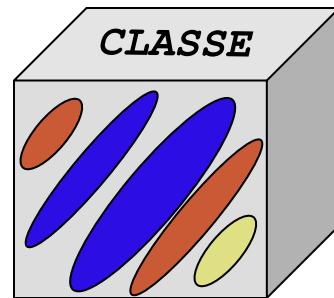
❑ JAXP (Java Api for Xml Parsing)

- Manipulation générique de parsers XML de type SAX et DOM
- Support de XSLT

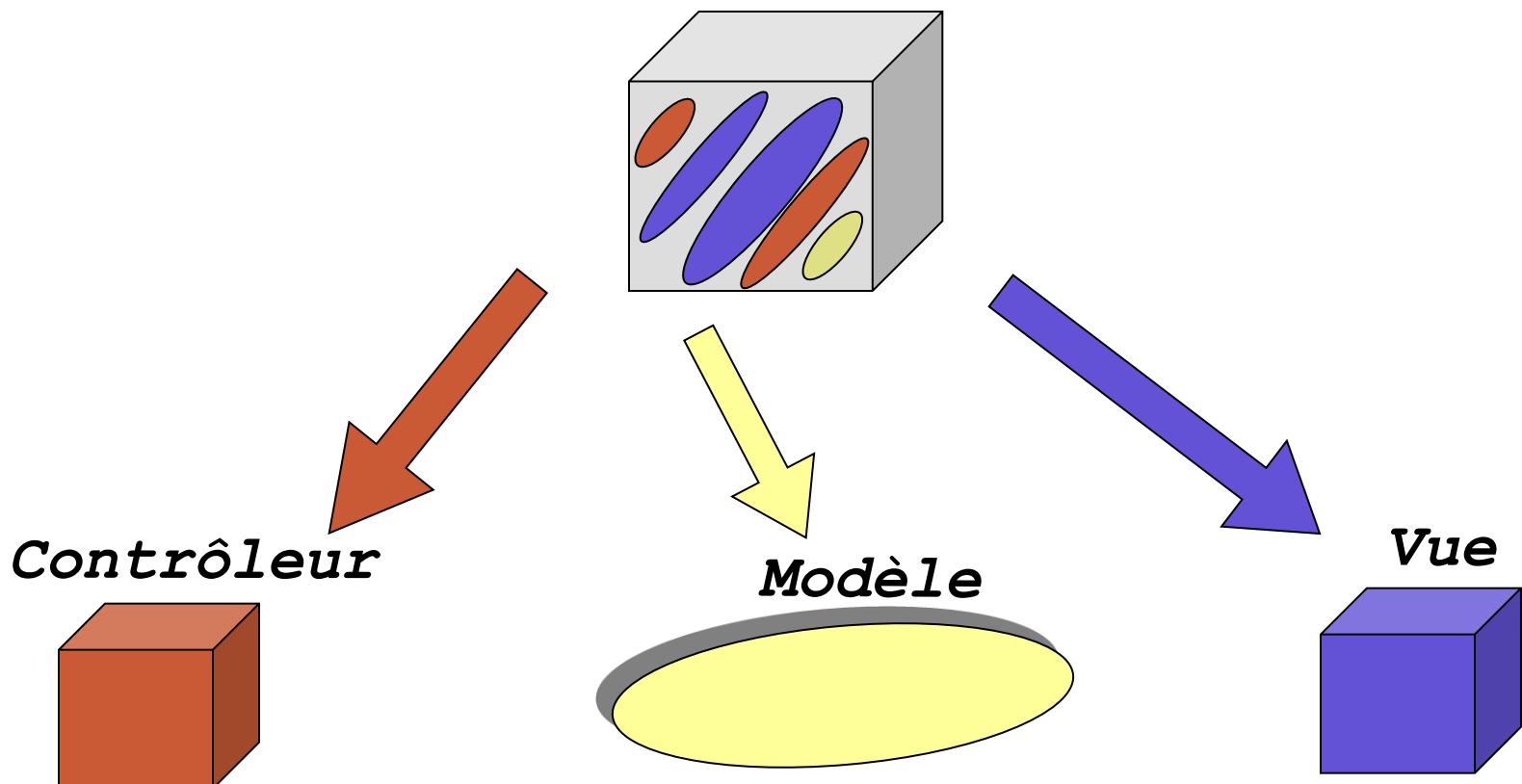
❑ JCA : JEE Connector Architecture

- Connexion à CICS, SAP, PeopleSoft...
- une API standardisée, indépendante du système ciblé, qui gère :
 - la gestion des connexions
 - les transactions
 - la sécurité
 - l'invocation et la manipulation des résultats

- Présentation des composantes JEE
- Présentation des composantes Web
 - Servlets et pages JSP
- Présentation des EJB
- Les formats de déploiement JEE



- **Dans une structure qui n'est pas modulaire, toutes les couches sont contenues dans un même bloc de code.**
 - Réception d'une action utilisateur
 - Identification du traitement à invoquer
 - Traitement
 - Accès en base de données
 - Formatage des données à afficher
 - ...



Réceptionne une action utilisateur
Déclenche les traitements appropriés
Retourne des données non-formatées

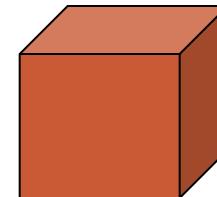
- Traitements à invoquer
- Javabeans (classes portant les données résultat avec des méthodes get et set sur ces données).

Formatage graphique des données

Utilisateur

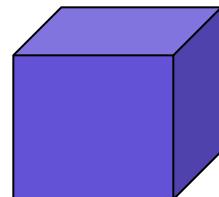


Contrôleur



5. La page est affichée

1. Effectue action



Vue

2. Déclenche 1 ou plusieurs traitements

Modèle

3. Transmet la partie 'JavaBean' du modèle

- Le contrôleur est une servlet.**
- La vue est une page JSP**
- Le modèle sont les objets métiers (POJO)**
- Le fichier web.xml permet (entre autre) de déterminer la servlet à appeler**

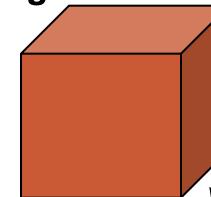
Utilisateur



Après consultation du web.xml, le serveur d'application déclenche l'exécution de LoginServlet.service(...)

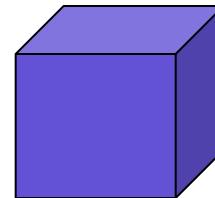
Servlet

LoginServlet



Modèle

Classes d'accès métier



Page jsp
login.jsp

- ❑ **Classe Java dont le rôle est de déclencher un traitement suite à une action utilisateur.**
- ❑ **Une servlet est associée à une URI donnée.**
 - ex : <http://localhost:80/bankonet/loginServlet>
 - La correspondance entre l'URI et la servlet est paramétrée dans le fichier web.xml (vu plus tard)
- ❑ **Une instance unique de chaque servlet répond à tous les appels**
 - Mécanisme multi-threads : à chaque client est affectée une thread.
- ❑ **Le conteneur Web**
 - L'ensemble des servlets sont gérées par un service du serveur d'application appelé "conteneur Web".

□ Les pages JSP (Java Server Page) :

- portent la partie "affichage graphique" d'une application.
- utilisent une syntaxe mixte
 - Généralement : Java / Html.
 - Il est possible d'utiliser d'autres langages de sortie (ex : WML)

□ Exemple de page JSP

```
<HTML>

<HEAD>

    <TITLE>Accueil BANKONET</TITLE>

</HEAD>

...

<BODY>

    <h1>Bienvenue sur Bankonet</h1>
    <P>Voici la date du jour : <%=new Date()%></P>
    <P><A href="<%=request.getContextPath() + "/Connexion"%>">
        Entrer</A></P>

</BODY>

</HTML>
```

□ **Les pages JSP (Java Server Page) ne sont qu'une simplification d'écriture pour le développeur**

- Au premier appel, une Servlet est générée à partir de chaque page JSP.
- Par la suite, la servlet générée est gérée par le conteneur Web au même titre que les autres servlets.

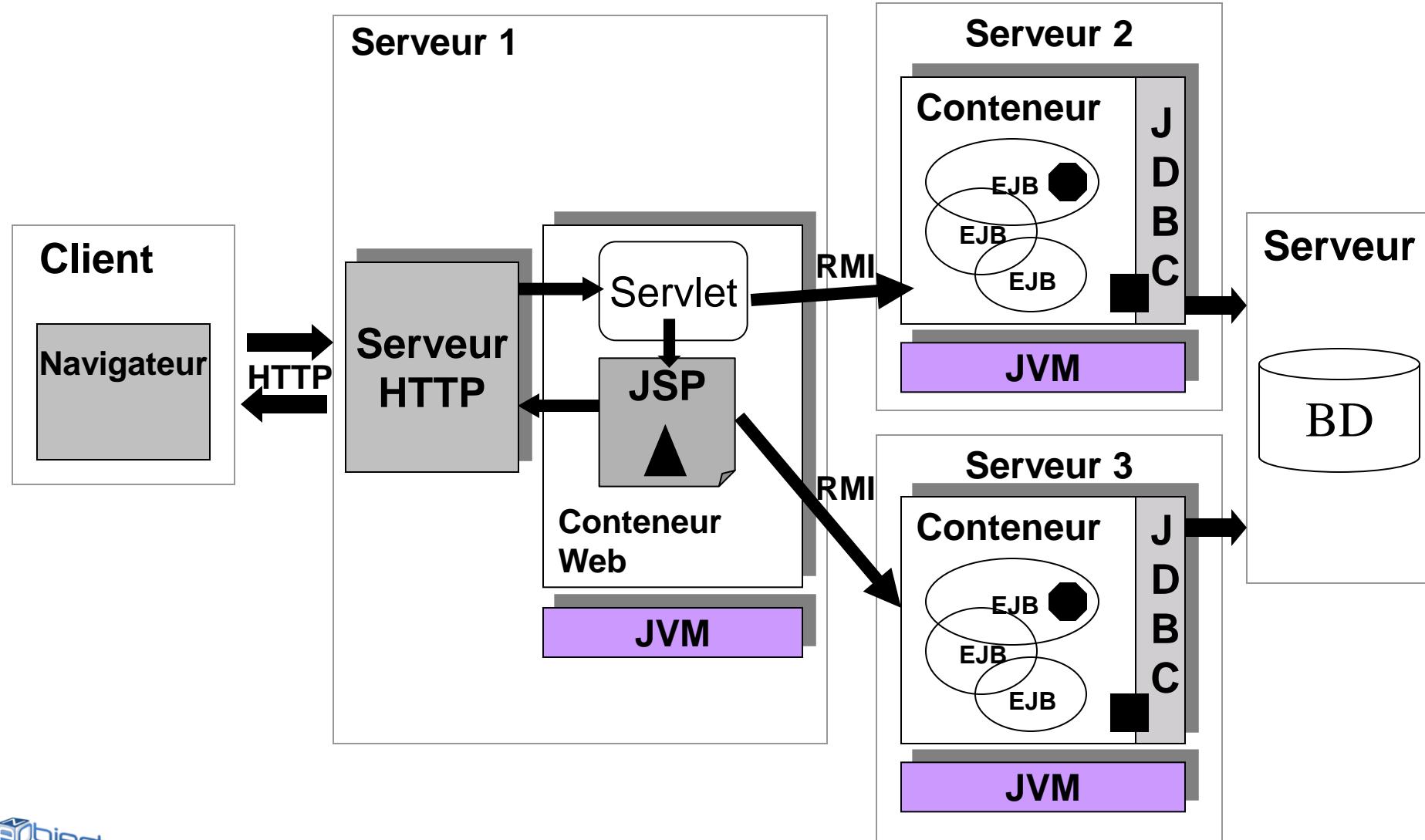
- Présentation des composantes JEE
- Présentation des composantes Web
 - Servlets et pages JSP
- Présentation des EJB
- Les formats de déploiement JEE

❑ Les EJB sont des composants métiers écrits en Java

- Ils s'exécutent au sein d'un conteneur

❑ Ils ont les propriétés suivantes :

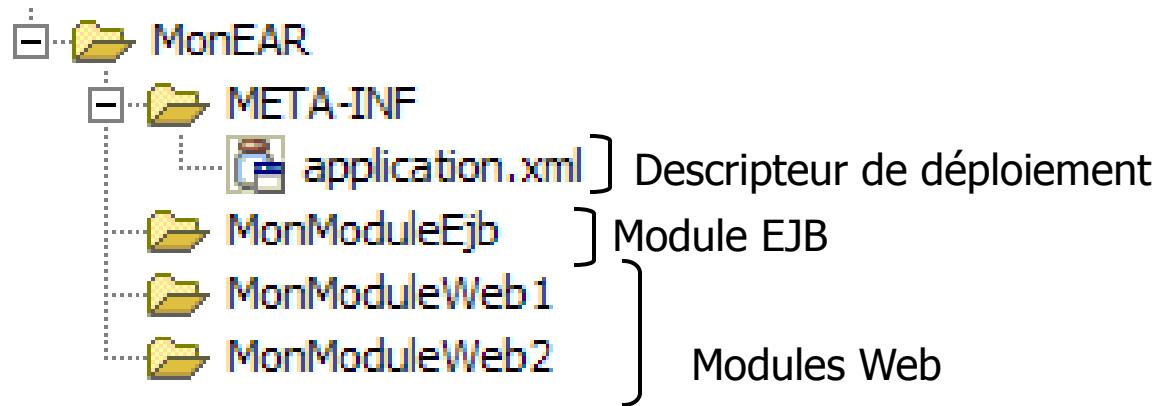
- Distribués : ils peuvent être invoqués à distance (RMI)
- Transactionnels
- Sécurisés
- Persistants
 - La gestion de la persistance peut être prise en charge par
 - le bean lui même (*Bean Managed Persistence* ou BMP)
 - le conteneur d'EJB (*Container Managed Persistence* ou CMP)



- Présentation des composantes JEE**
- Présentation des composantes Web**
 - Servlets et pages JSP
- Présentation des EJB**
- Les formats de déploiement JEE**

❑ Les applications JEE se déploient via un format standard.

- Répertoires contenant une structure très précise.
- Les répertoires sont compressés (zippés) pour la phase de déploiement.

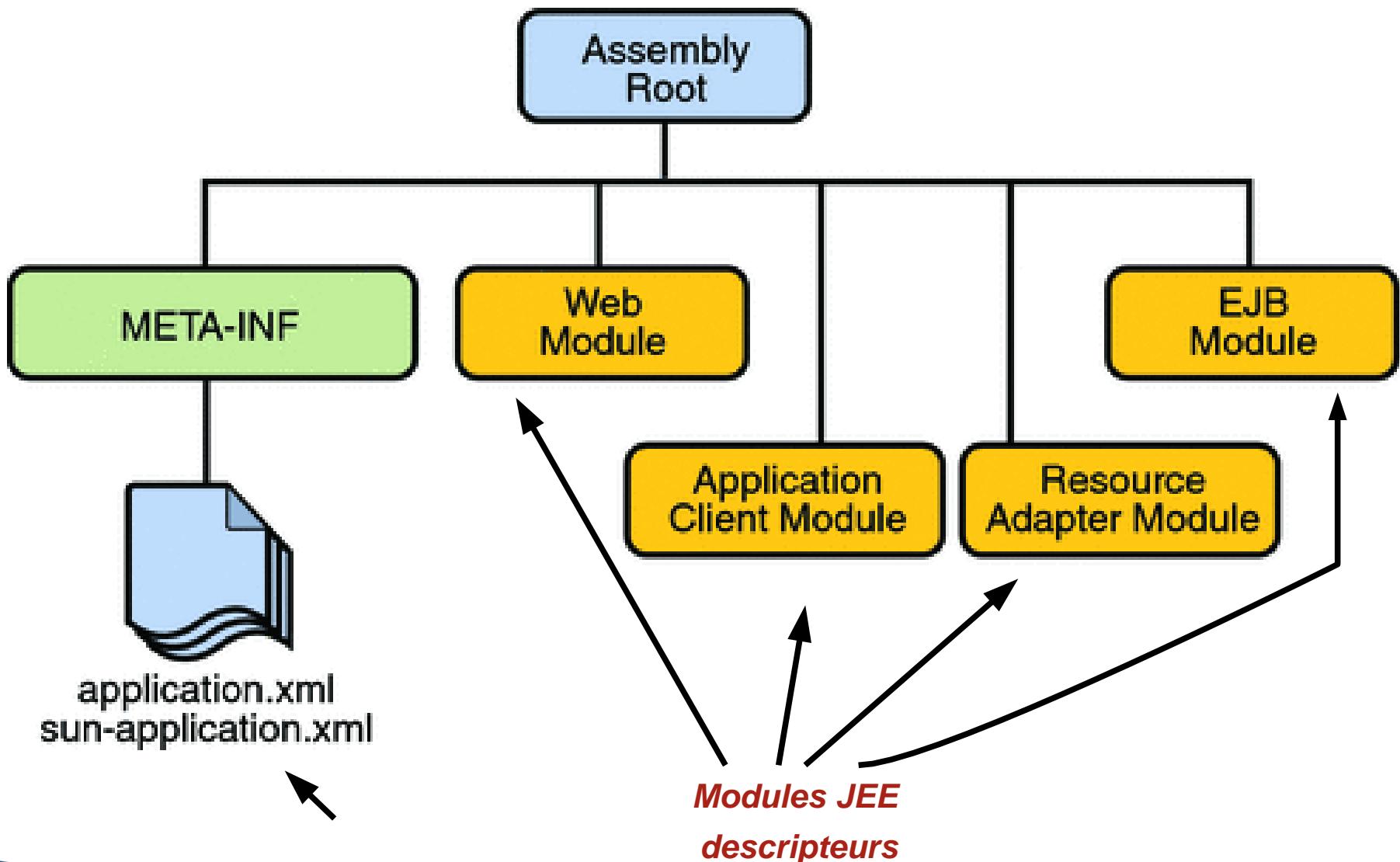


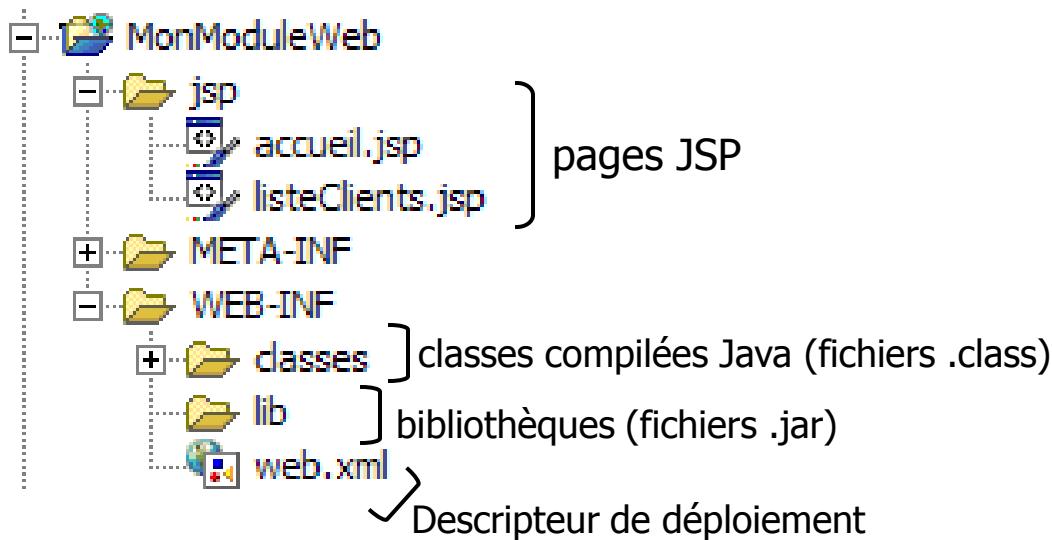
□ Une application d'entreprise contient :

- L'ensemble des modules de l'application
 - Modules Web, modules EJB, modules de connecteurs
- Un descripteur de déploiement 'application.xml'
 - Fournit la liste des modules de l'application
 - contient les URI de chacun des modules Web
- Lors de la livraison, une application d'entreprise est un fichier compressé d'extension .ear

```
<!DOCTYPE application PUBLIC "-//Sun Microsystems, Inc.//DTD JEE  
Application 1.3//EN" "http://java.sun.com/dtd/application_1_3.dtd">  
<application>  
    <display-name>Module1_name</display-name>  
    <module id="WebModule_1126886391515">  
        <web>  
            <web-uri>MonModuleWeb1.war</web-uri>  
            <context-root>web1</context-root>  
        </web>  
    </module>  
    <module id="EjbModule_1126886421689">  
        <ejb>MonModuleEjb.jar</ejb>  
    </module>  
</application>
```

exemple de fichier application.xml



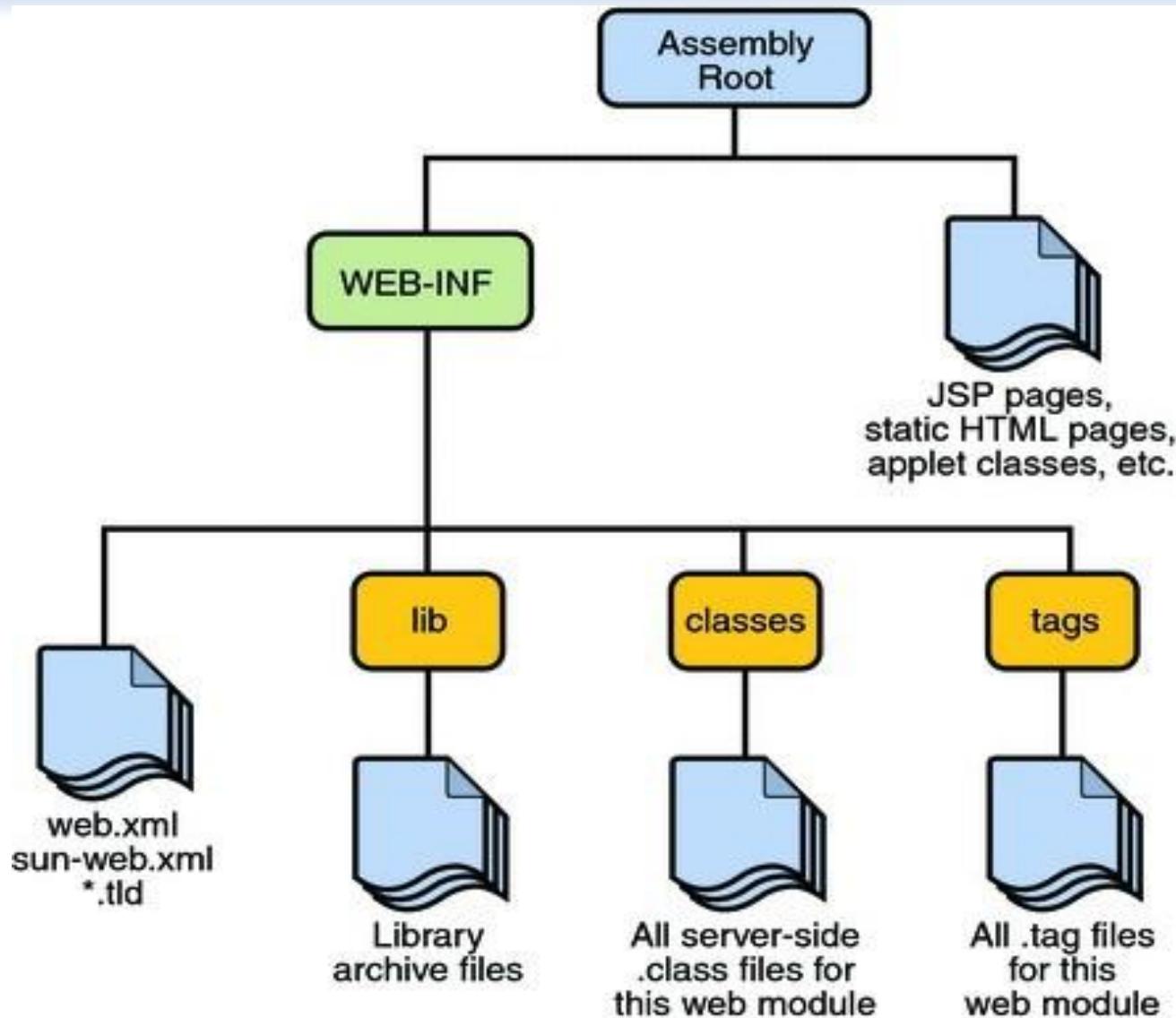


❑ Contient un module web et son contexte

- A chaque module Web correspond un chemin relatif
Ex : http://localhost:80 /bankonet
- Un descripteur de déploiement '**web.xml**'
 - Déclare une page d'accueil et une page d'erreur pour le module
 - Déclare la liste des servlets et leur mapping associé
 - Il n'est pas obligatoire de déclarer les pages JSP
- Lors de la phase de déploiement, un module Web est compressé en une archive d'extension **.war**

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems..." "http://java.sun.com/dtd/web-
app_2_3.dtd">
<web-app id="WebApp">
    <servlet>
        <servlet-name>LoginServlet</servlet-name>
        <servlet-class>com.bankonet.servlet.LoginServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>LoginServlet</servlet-name>
        <url-pattern>/login</url-pattern>
    </servlet-mapping>
    <welcome-file-list>
        <welcome-file>jsp/accueil.jsp</welcome-file>
    </welcome-file-list>
    <error-page>
        <error-code>404</error-code>
        <location>/jsp/erreur404.jsp</location>
    </error-page>
</web-app>
```

exemple de fichier web.xml



❑ Module EJB

- Contient un ensemble d'EJBs
- Le descripteur de déploiement se nomme ejb-jar.xml
- Lors de la phase de déploiement, un module EJB est compressé en une archive d'extension .jar (java archive)
 - L'archive n'est pas décompressée sur le serveur.

□ Module de connecteur

- Permet de se connecter vers un système externe
 - CICS, SAP...
- Le descripteur de déploiement se nomme ra.xml
- Lors de la phase de déploiement, un module de connecteur est compressé en une archive d'extension .rar (resource adapter archive)

□ Compresser chacun des modules de l'application.

- Les extensions des fichiers compressés sont attribuées selon la nature des modules (.war, .jar, .rar).

□ Définir le descripteur de l'application

- Fichier META-INF/application.xml

□ Compresser l'ensemble de l'application

- Lui attribuer une extension .ear



Ces étapes sont généralement automatisées par l'environnement de développement ou par des scripts.

❑ Applications J2SE

- Fichiers classe Java (.class)
- Méta-données sous forme de fichiers de propriétés
- Fichier "Manifest ", peut contenir quelques méta-données

❑ Applets

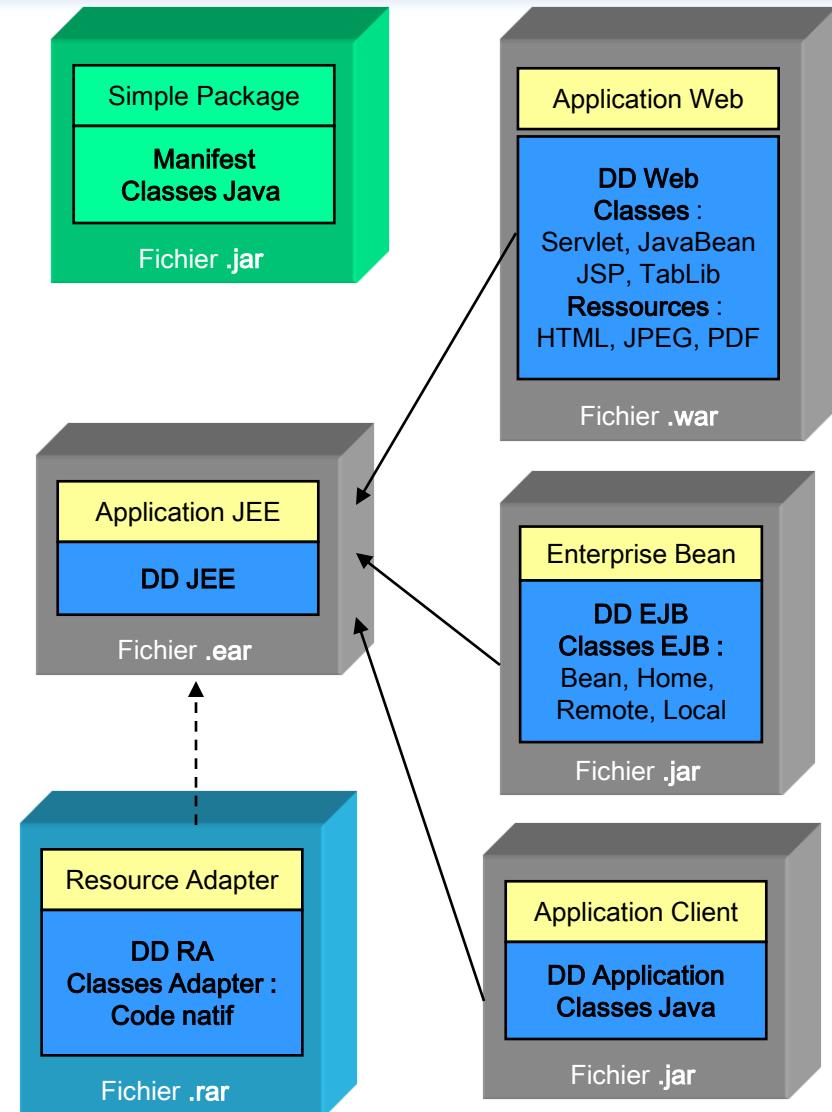
- Similaires aux applications, le "manifest" peut contenir une signature

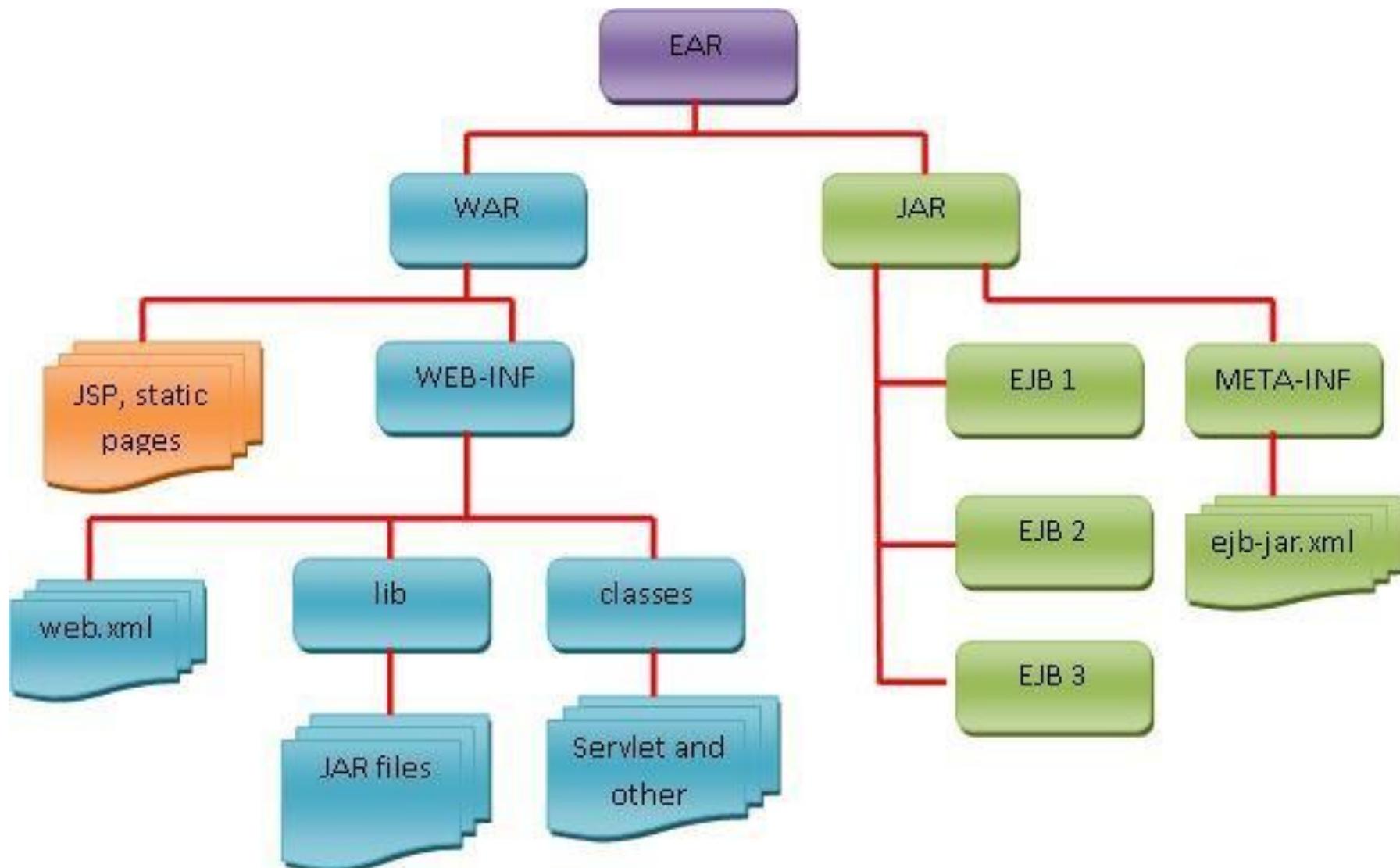
❑ Applications JEE

- S'appuient sur des DD (Descripteur de déploiement)
- Stocke les méta-données concernant les propriétés des composants et le paramétrage des applications

❑ Adaptateurs de Ressource

- S'appuient sur des DD
- Peuvent être indépendants ou contenus dans une archive EAR





TP 0 : Mise en place de l'environnement de développement Web JEE

1 - Présentation de la technologie JEE

2 - Conception et développement des Servlets

3 - Conception et développement de pages JSP

4 - Taglibs, EL et JSTL

5 - Fonctionnalités avancées en JEE

➤ **Principes de base sur les servlets**

➤ **Construction d'une servlet simple**

- Exemple pas à pas.

➤ **Fonctionnement détaillé**

- Méthodes de cycle de vie : init et destroy
- Méthodes d'appel : service, doGet et doPost.
- Renvoi vers une autre ressource : forward ou redirect ?

➤ **Les sessions utilisateur**

➤ **Mise d'informations dans le contexte**

- session, request, servletContext

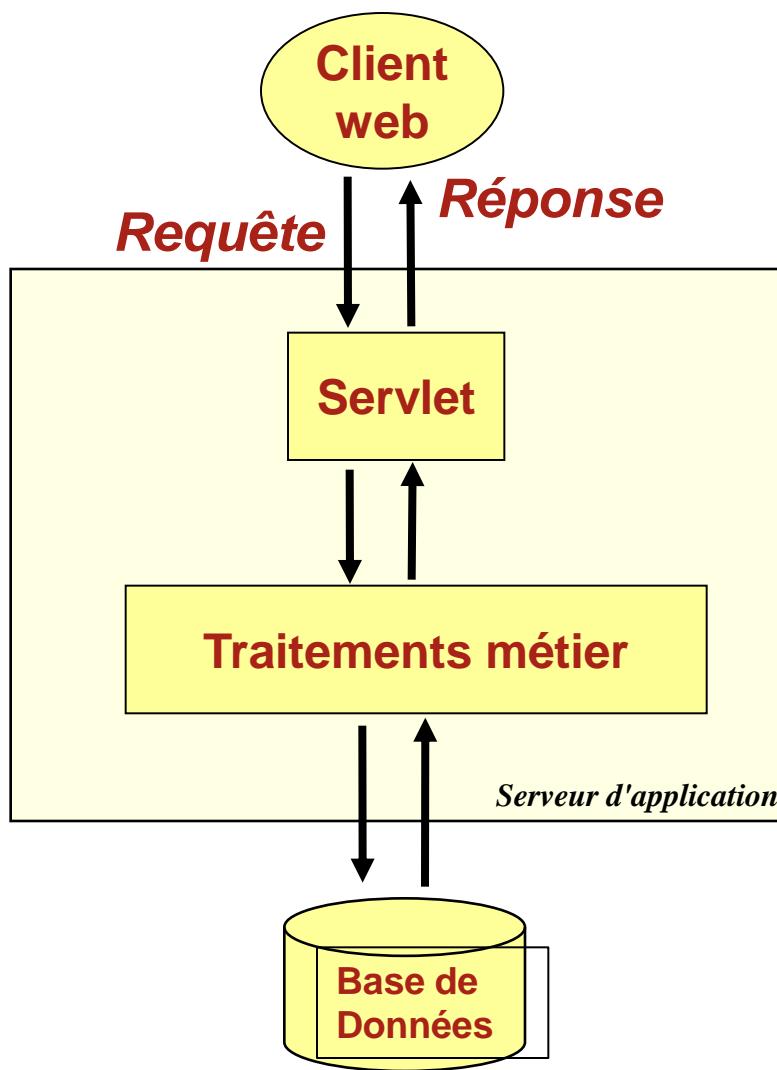
➤ **Gestion des formulaires**

□ Qu'est-ce qu'une Servlet ?

- Code java, exécuté suite à un événement utilisateur
 - Connexion à une url donnée.
 - On parle de 'requête utilisateur'

□ La réponse http générée par une servlet ne contient que les fichiers statiques qui sont renvoyés au client.

- code HTML, image...



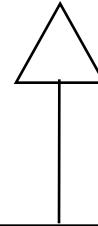
La Servlet réceptionne une **requête** client.
La requête contient un ensemble
d'informations constituant la demande du
client (ex: url demandée, éventuellement
paramètres de formulaire...).

Cette requête est traitée dans la servlet,
et une **réponse** est générée.

► **javax.servlet**

(définition des objets (Interfaces) standard)

- Package générique des servlets
- GenericServlet, ServletRequest, ServletResponse...

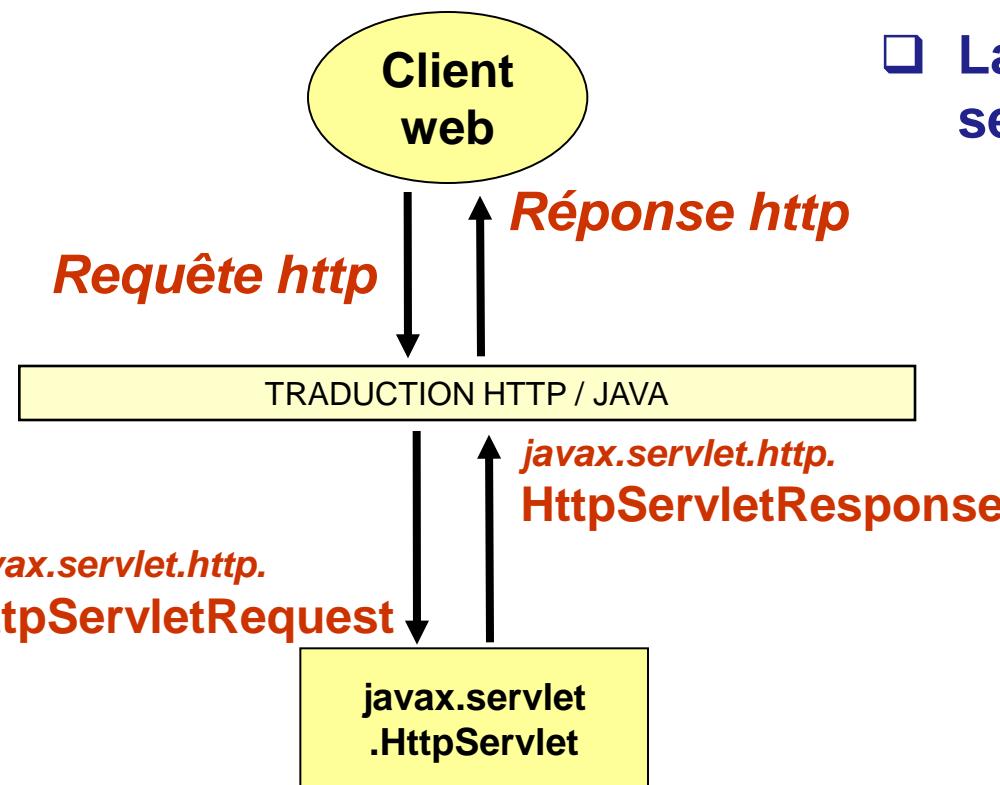


► **javax.servlet.http**

- spécialisation de javax.servlet pour le protocole HTTP
- HttpServlet, HttpServletRequest, HttpServletResponse...



il est très rare d'utiliser les classes du package javax.servlet directement.



□ La traduction est faite par le serveur d'application

- Le client web perçoit les requêtes/réponses comme des objets http.
- La servlet perçoit les requêtes/réponses comme des objets Java.

➤ **Principes de base sur les servlets**

➤ **Construction d'une servlet simple**

- Exemple pas à pas.

➤ **Fonctionnement détaillé**

- Méthodes de cycle de vie : init et destroy
- Méthodes d'appel : service, doGet et doPost.
- Renvoi vers une autre ressource : forward ou redirect ?

➤ **Les sessions utilisateur**

➤ **Mise d'informations dans le contexte**

- session, request, servletContext

➤ **Gestion des formulaires**

- ❑ HelloServlet doit redéfinir la méthode héritée "service".

```
public class HelloServlet extends HttpServlet {  
    public void service(HttpServletRequest req,  
                        HttpServletResponse resp)  
        throws ServletException, IOException {  
        System.out.println("Hello World !");  
    }  
}
```



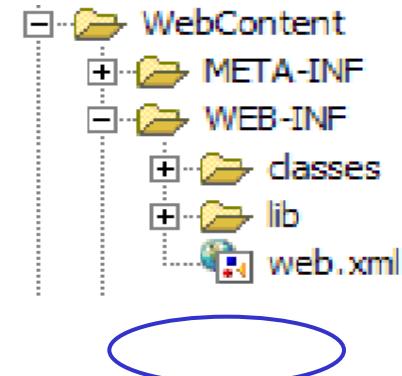
redéfinir la méthode "service" est la plus simple des possibilités. Il en existe d'autres qui seront vues plus tard.

- Une application Web a la structure ci-contre :

- Le fichier web.xml est le descripteur de déploiement Web.

- HelloServlet est déclarée dans le web.xml comme suit :

```
<web-app>
    <servlet>
        <servlet-name>helloServletName</servlet-name>
        <servlet-
            class>com.bankonet.servlet.HelloServlet</servlet-
            class>
    </servlet>
    <servlet-mapping>
        <servlet-name>helloServletName</servlet-name>
        <url-pattern>/hello</url-pattern>
    </servlet-mapping>
</web-app>
```



- A chaque application web correspond une url donnée.
 - ex : <http://www.bankonet.fr>
- Dans le web.xml, les servlets sont définies relativement à l'url de l'application web
 - A "/hello" correspondra <http://www.bankonet.fr/hello>
- Le serveur JEE fait appel à la méthode "service" de la classe définie dans le tag <servlet-class>

1. Un client se connecte à l'url **http://www.bankonet.fr/hello**
2. Le serveur JEE identifie à quelle application web correspond l'url
3. Dans le fichier `web.xml`, il identifie quelle est la classe servlet associée au mapping "/hello"
4. La méthode "service" de `HelloServlet` est exécutée

➤ **Principes de base sur les servlets**

➤ **Construction d'une servlet simple**

- Exemple pas à pas.

➤ **Fonctionnement détaillé**

- Méthodes de cycle de vie : init et destroy
- Méthodes d'appel : service, doGet et doPost.
- Renvoi vers une autre ressource : forward ou redirect ?

➤ **Les sessions utilisateur**

➤ **Mise d'informations dans le contexte**

- session, request, servletContext

➤ **Gestion des formulaires**

- Une seule fois par application Web
- Une instance de la Servlet est créée et servira pour toutes les requêtes des utilisateurs
- Une méthode d'initialisation est exécutée : la méthode init()
 - Exécutée uniquement au moment du chargement de la Servlet

```
public void init(ServletConfig servconfig) throws ServletException
```



Si la méthode init(...) est redéfinie, il faut toujours faire appel à la "super-méthode".

```
public void init(ServletConfig servconfig) throws  
ServletException {  
  
    System.out.println("initialisation");  
  
    super.init(servconfig);  
}
```

□ Il existe 2 modes de chargement pour les Servlets :

➤ Chargement au démarrage du serveur

- La Servlet est chargée dès le démarrage du serveur.
- Spécifié par le paramètre *load-on-startup* du descripteur de déploiement

```
<servlet>
    <servlet-name>helloServletName</servlet-name>
    <servlet-class>com.bankonet.servlet.HelloServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
```



➤ Chargement à la première demande

➤ la Servlet est chargée lors de la première utilisation :

- Un utilisateur fait appel à l'URL correspondante
- Un utilisateur soumet un formulaire vers cette servlet
- Par redirection à partir d'une autre servlet

➤ Comportement par défaut (tag load-on-startup non spécifié)

```
<servlet>
    <servlet-name>helloServletName</servlet-name>
    <servlet-class>com.bankonet.servlet.HelloServlet</servlet-class>
</servlet>
```

! La servlet étant chargée à la première demande, le premier utilisateur sera servi plus lentement

□ Une seule instance de servlet répond à toutes les requêtes client.

- une thread pour chaque appel à **service()**

□ **DANGER** : les variables d'instances sont partagées.

```
public class LoginServlet extends HttpServlet {  
    private String nom;  
    private String prenom;  
  
    public void service(req, resp)  
        throws ServletException, IOException {  
        nom = ...  
        prenom = ...  
    }  
}
```

```
public class LoginServlet extends HttpServlet {  
  
    public void service(req, resp)  
        throws ServletException, IOException {  
  
        String nom = ...  
        String prenom = ...  
    }  
}
```

□ Déchargement de la Servlet

- Arrêt des traitements puis destruction de l'instance de la Servlet
- Cette opération est effectuée lors de l'arrêt 'normal' du serveur d'application.

```
public void destroy()
```

- **Méthode doGet : ne sera appelée que dans le cas d'une requête http de type GET**

Requête Get : Connexion simple d'un utilisateur à une url donnée.

- **Méthode doPost : ne sera appelée que dans le cas d'une requête http de type POST**

Requête Post : validation d'un formulaire html.

□ Méthode service : appelée quel que soit le type de la requête.

- Si la requête est de type 'GET', service appelle doGet
- Si la requête est de type 'POST', service appelle doPost



Si la méthode service est redéfinie, doGet et doPost ne seront jamais appelés.

Note : les méthodes service, doPost, doGet sont toujours appelées suite à un événement utilisateur. On parle de 'méthodes de rappel'.

- ❑ **La Servlet doit pouvoir rediriger le traitement vers une autre ressource**
 - Typiquement une page html, une page jsp et par extension toute ressource qui peut être identifiée par une url.

- ❑ **Il existe deux méthodes : par forward ou par redirect.**
 - Forward : le navigateur client n'est pas informé du passage de la servlet vers la ressource demandée. Il ne verra que le résultat 'final'.
 - Redirect : le navigateur client est informé du passage de la servlet vers la ressource demandée.

□ Un objet RequestDispatcher permet d'effectuer le forward.

```
public void service(HttpServletRequest req, HttpServletResponse resp)  
throws ServletException, IOException {  
    //...  
    RequestDispatcher disp =  
        this.getServletContext().getRequestDispatcher("/login.html");  
    disp.forward(req, resp);  
}
```

- le navigateur client n'est pas informé du passage de la servlet vers la ressource demandée.
- Dans le navigateur, la barre d'url affiche l'url de la servlet (et non celle de login.html).

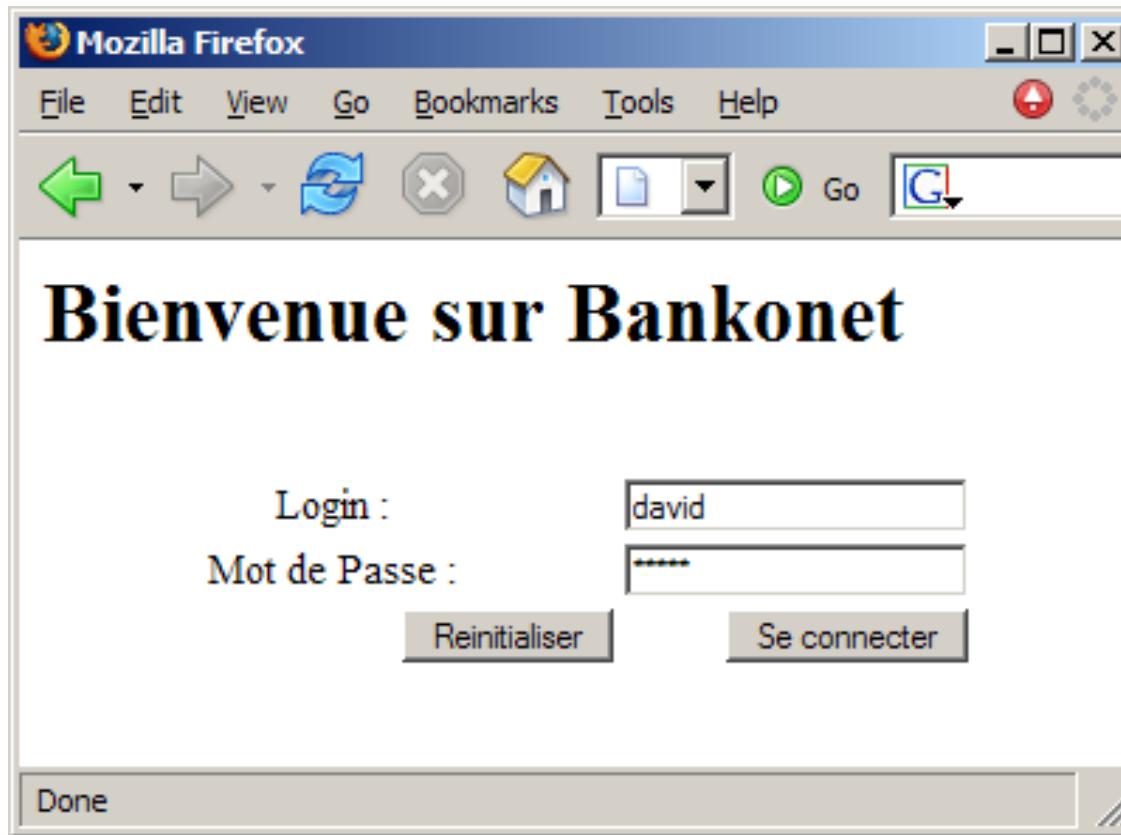
□ La réponse peut demander une redirection automatique

```
public void service(HttpServletRequest req, HttpServletResponse resp)  
throws ServletException, IOException {  
    //...  
    resp.sendRedirect(req.getContextPath() + "/login.html");  
}
```

□ le navigateur client est informé du passage de la servlet vers la ressource demandée.

- Dans le navigateur, la barre d'url affiche l'url de login.html .
- Une nouvelle requête http est créée. Les informations présentes dans la précédente requête sont perdues.

□ TP 1 et TP 2 : DateServlet, cycle de vie





➤ **Principes de base sur les servlets**

➤ **Construction d'une servlet simple**

- Exemple pas à pas.

➤ **Fonctionnement détaillé**

- Méthodes de cycle de vie : init et destroy
- Méthodes d'appel : service, doGet et doPost.
- Renvoi vers une autre ressource : forward ou redirect ?

➤ **Les sessions utilisateur**

➤ **Mise d'informations dans le contexte**

- session, request, servletContext

➤ **Gestion des formulaires**

- **Certaines informations doivent pouvoir être mises à disposition pendant toute la durée de connexion d'un utilisateur.**
- **Exemple classique de l'utilisation des sessions : un panier électronique.**
 - Un objet panier doit être associé à chacun des clients.
 - Le panier doit être consultable jusqu'à déconnexion de l'utilisateur.

- **Lors de la première requête http, le serveur ouvre une session pour le client web.**
 - Le serveur attribue au client un identifiant de session.
 - L'identifiant sera par la suite stocké sur le poste utilisateur (généralement sous forme de cookie).

- **Pour chacune des requêtes suivantes, le client doit présenter son identifiant de session.**
 - Si un cookie est utilisé, il est automatiquement joint à chaque requête client.

- **Le serveur récupère l'identifiant de session du client.**
- **Il existe une zone mémoire sur le serveur où sont stockées toutes les sessions utilisateur.**
 - Les sessions sont indexées par leur identifiant.
- **Le serveur récupère l'objet session en cours.**
 - Il est de type javax.servlet.http.HttpSession

□ A partir de la requête http, le serveur créé un objet **HttpServletRequest**.

- Porte les informations de la requête http
- Mis à disposition dans les servlets.

□ Le serveur place la session dans l'objet request.

- La session doit être récupérée comme suit :

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;
import ...;

public class HelloServlet extends HttpServlet {
    public void service(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        HttpSession session = req.getSession(true);
        ...
    }
}
```

```
public void service(HttpServletRequest req, HttpServletResponse  
resp)  
  
throws ServletException, IOException {  
  
    HttpSession session = req.getSession(true);  
    // ...  
}
```

□ La méthode **request.getSession(boolean)** permet de récupérer une session.

- true : si le client n'a pas de session attribuée, le serveur lui crée une session.
- false : si le client n'a pas de session attribuée, la méthode renvoie 'null'.

□ Utilisation courante :

- Lors de la première connexion de l'utilisateur, la session peut-être créée (paramètre 'true').

```
public void service(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
    HttpSession session = req.getSession(true); // ...
}
```

- Par la suite, dans une autre servlet, la session ne peut plus être créée (paramètre 'false').

```
public void service(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
    HttpSession session = req.getSession(false);
    if (session != null) {
        // traitement usuel
    }
    else {
        // Utilisateur non identifié !!
    }
}
```

□ Il est possible d'invalider une session comme suit :

```
public void service(HttpServletRequest req,  
HttpServletResponse res)  
throws ServletException, IOException {  
  
    HttpSession session = req.getSession(false);  
  
    session.invalidate();  
  
    // ...  
  
}
```

□ La session du client est supprimée après un délai d'inactivité

- Positionné par défaut sur le serveur d'application
- Il est possible de redéfinir cette valeur dans le web.xml :

```
<session-config>  
  
    <session-timeout>2</session-timeout>  
  
    <!-- valeur en minutes -->  
  
</session-config>
```

□ **Les objets session sont stockés en permanence sur le serveur.**

- Dans la mesure du possible, il faut stocker un minimum d'informations dans la session.

□ **Les navigateurs Web peuvent refuser l'utilisation de cookies.**

- Paramétrable dans les options du navigateur.
- Si les utilisateurs d'une application utilisent potentiellement un navigateur sans cookies, il faut encoder les URL comme suit :

`http://www.bankonet.fr/Accueil;JSessionId=12429`

➤ **Principes de base sur les servlets**

➤ **Construction d'une servlet simple**

- Exemple pas à pas.

➤ **Fonctionnement détaillé**

- Méthodes de cycle de vie : init et destroy
- Méthodes d'appel : service, doGet et doPost.
- Renvoi vers une autre ressource : forward ou redirect ?

➤ **Les sessions utilisateur**

➤ **Mise d'informations dans le contexte**

- session, request, servletContext

➤ **Gestion des formulaires**

- Il existe 3 contextes dans lesquels nous pouvons temporairement stocker de l'information lorsque cela est requis :
 - Le contexte commun à toutes les servlets d'une application web (objet *ServletContext*)
 - Le contexte de session (objet *HttpSession*)
 - Le contexte de la requête (objet *HttpServletRequest*)
- Ces 3 contextes proposent les mêmes méthodes *setAttribute* et *getAttribute*.
- Choisir le plus adéquat
 - Pour ne pas conserver l'information plus que nécessaire
 - Pour libérer les ressources (mémoire) dès que possible

□ Un attribut est un objet

- Classiquement utilisé pour le transfert d'informations du contrôleur (servlet) vers la vue (page JSP).

□ Enregistrer et récupérer des informations

- *void setAttribute(String, Object)*
- *Object getAttribute(String)*

□ Supprimer une entrée dans un contexte

- *void removeAttribute(String)*

□ **Les informations présentes dans le contexte de requête sont utilisables pendant un aller-retour client serveur.**

- Après, la requête est automatiquement détruite par le serveur d'applications.
- Très utilisé pour communiquer de la servlet vers la page jsp.
- Chaque client a son propre contexte de requête.

```
public void service(HttpServletRequest req, HttpServletResponse  
resp)  
  
throws ServletException, IOException {  
  
// ...  
  
req.setAttribute("erreurMessage", "L'accès à cette ressource  
n'est pas autorisé");  
  
}
```

□ **Les informations présentes dans le contexte de session sont utilisables jusqu'à la suppression de la session utilisateur.**

- Par déconnexion ou par timeout.
- Pour garder de bonnes performances, une bonne pratique est de stocker un minimum de choses dans la session.
- Chaque client a son propre contexte de session.

```
public void service(HttpServletRequest req, HttpServletResponse resp)  
throws ServletException, IOException {  
    // ...  
  
    HttpSession session = req.getSession(true);  
    session.setAttribute("livre", monLivre);  
}
```

□ **Les informations présentes dans le contexte de servlet sont utilisables tant que l'application est lancée.**

- Permet de stocker globalement des informations.
- Durée de vie :
 - jusqu'à l'arrêt du serveur
 - jusqu'au rechargement du contexte (en cas de modification).
- Le contexte de servlet est partagé par tous les utilisateurs

```
public void service(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
    //...
    this.getServletContext().setAttribute("dateDuJour", new Date());
}
```

Où a-t-on besoin de l'information ?

	Plusieurs pages	Les pages liées par le traitement d'une même requête
Tous les utilisateurs	ServletContext	ServletContext
L'utilisateur courant	HttpSession	HttpServletRequest

Qui utilise l'information ?

➤ **Principes de base sur les servlets**

➤ **Construction d'une servlet simple**

- Exemple pas à pas.

➤ **Fonctionnement détaillé**

- Méthodes de cycle de vie : init et destroy
- Méthodes d'appel : service, doGet et doPost.
- Renvoi vers une autre ressource : forward ou redirect ?

➤ **Les sessions utilisateur**

➤ **Mise d'informations dans le contexte**

- session, request, servletContext

➤ **Gestion des formulaires**

- L'url de retour est positionnée dans la balise d'en-tête du formulaire.
- La requête est envoyée au serveur quand l'utilisateur clique sur le bouton de type "submit".

```
<HTML>
<HEAD><META http-equiv="Content-Style-Type" content="text/css"></HEAD>
<BODY>
    <h1>Connexion à Bankonet</h1>
    <FORM method="POST" action="/processLogin">
        <TABLE border="1">
            <TR>
                <TD>Identifiant</TD>
                <TD><INPUT size="20" type="text" name="identifiant"></TD>
            </TR>
            <TR>
                <TD>Mot de passe</TD>
                <TD><INPUT size="20" type="password" name="motDePasse"></TD>
            </TR>
            <TR>
                <TD colspan="2"><INPUT type="submit" value="Connexion">
                    <INPUT type="reset" value="Reset"></TD>
                </TR>
        </TABLE>
    </FORM>
</BODY>
</HTML>
```

Connexion à Bankonet

Identifiant	
Mot de passe	
Connexion	Reset

- Les paramètres sont des chaînes de caractères portées par la requête
- Ils peuvent être récupérés dans les méthodes service, doGet, doPost... d'une servlet.

```
public void service(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
    String monId = req.getParameter("identifiant");
    String monMotDePasse = req.getParameter("motDePasse");
    // ...
}
```

- Si l'argument method="post" n'est pas positionné dans le tag <form>, la validation de formulaire renverra une requête de type Get.
- Quelle en est la conséquence dans l'exemple ci-dessous ?

```
<HTML>
<HEAD><META http-equiv="Content-Style-Type" content="text/css">
<BODY>
    <h1>Connexion à Bankonet</h1>
    <FORM action="/traiterLogin">
        <TABLE border="1">
            <TR>
                <TD>Identifiant</TD>
                <TD><INPUT size="20" type="text" name="identifiant"></TD>
            </TR>
            <TR>
                <TD>Mot de passe</TD>
                <TD><INPUT size="20" type="password" name="motDePasse"></TD>
            </TR>
            <TR>
                <TD colspan="2">
                    <INPUT type="submit" value="Connexion">
                    <INPUT type="reset" value="Reset">
                </TD>
            </TR>
        </TABLE>
    </FORM>
</BODY>
</HTML>
```

Connexion à Bankonet

Identifiant	
Mot de passe	
<input type="button" value="Connexion"/>	<input type="button" value="Reset"/>

Réponse : le mot de passe apparaît en clair dans l'url...

```
http://localhost:8080/BankonetWeb/traiterLogin?identifiant=Donald&motDePasse=Pluto
```

- Pour éviter cela, une validation de formulaire doit toujours renvoyer une requête "POST"

```
...
<FORM method="POST" action="/traiterLogin">
...
</FORM>
```

- **Il ne faut pas confondre 'attribut de requête' et 'paramètre de requête'**
- **Un paramètre est une chaîne de caractères transmise par la requête**
 - (le plus souvent après soumission d'un formulaire).
- **Un attribut est un objet qui a été placé explicitement dans le contexte de requête**
 - À partir d'une précédente servlet ou page jsp.

```
public void service(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
    String monId = req.getParameter("identifiant");
    Client monClient = (Client) req.getAttribute("client");
    // ...
}
```



- 1 - Présentation de la technologie JEE**
- 2 - Conception et développement des Servlets**
- 3 - Conception et développement de pages JSP**
- 4 - Taglibs, EL et JSTL**
- 5 - Fonctionnalités avancées en JEE**



1. Première approche

- Définition
- Cycle de vie
- Appel d'une page jsp

2. Structure d'une page jsp

3. Page d'accueil, page d'erreur

4. Inclusion de page

5. La bibliothèque de balises JSP

6. JSTL

7. Pour aller plus loin : écrire ses propres balises.

□ JSP : Java Server Page

- Fichier d'extension '.jsp'. Ex : login.jsp, menu.jsp ...

□ Une page jsp est un document texte ayant une syntaxe mixte

HTML/ Java.

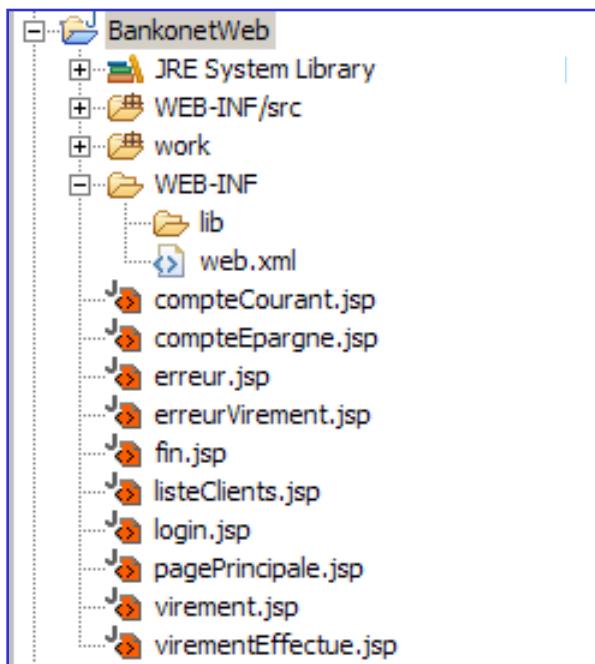
- Code HTML = contenu statique, mise en page
- Code Java = contenu dynamique

□ Une page JSP n'est pas une classe Java.

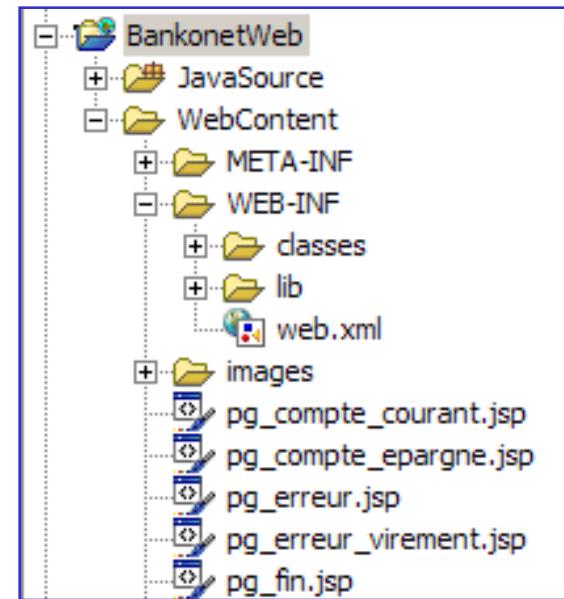
- Le serveur d'application génère une classe java à partir de la page JSP.
- La classe Java générée est compilée, puis exécutée à la demande.

► Où stocker les pages jsp ?

- dans le répertoire courant

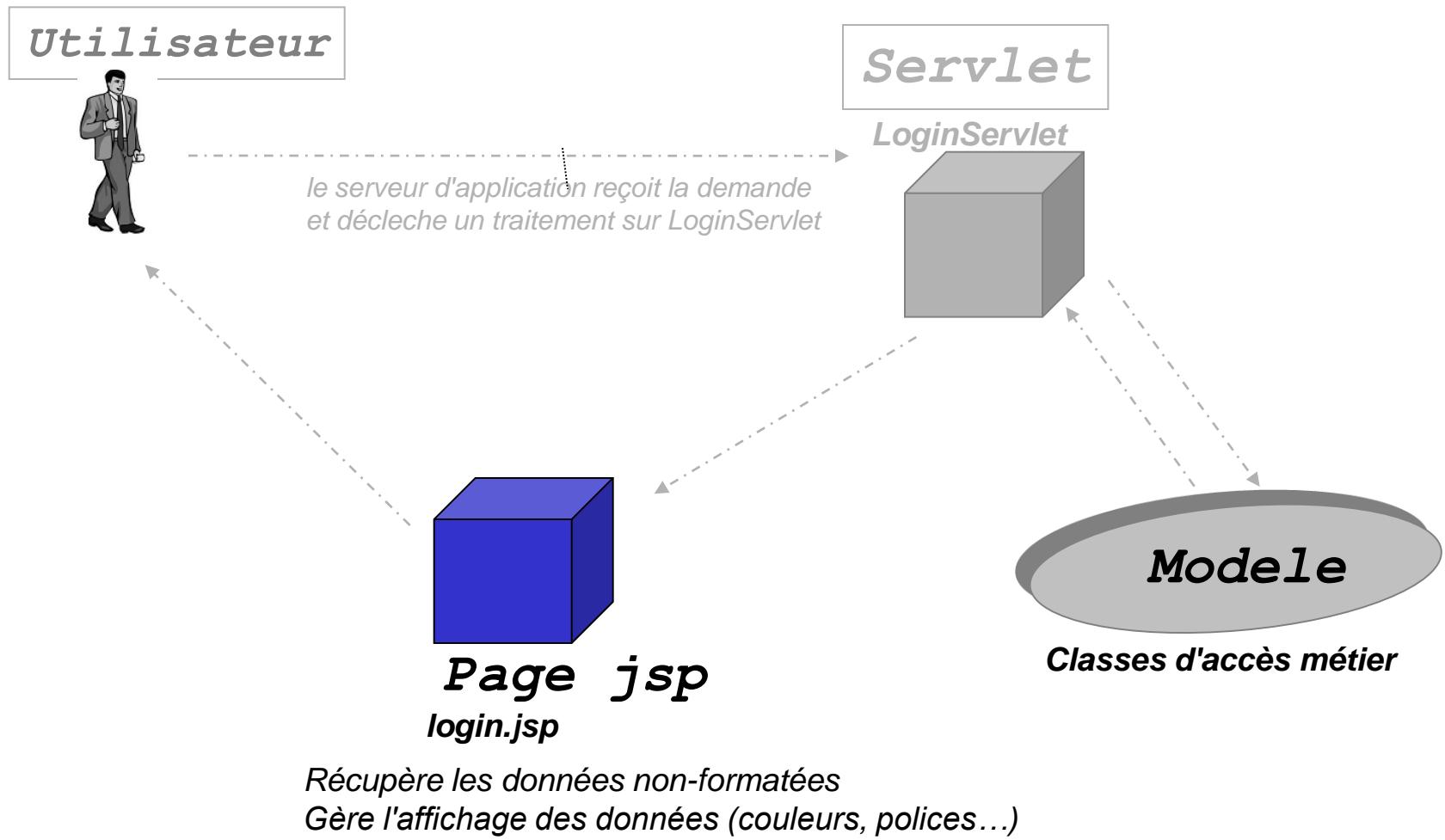


- Ou dans le répertoire WebContent



```
<HTML>
<HEAD>
    <TITLE>Accueil BANKONET</TITLE>
</HEAD>
<%@ page language="java" import="java.util.Date,java.text.*" %>
<BODY>
    <% Date aDate = new Date();
    SimpleDateFormat sdf = new SimpleDateFormat("dd MMMM");
    %>
    <h1>Bienvenue sur Bankonet</h1>
    <P>Nous sommes le : <%=sdf.format(aDate)%></P>
    <P><A href="<%="request.getContextPath()%>/Connexion">Entrer</A></P>
</BODY>
</HTML>
```





□ La page JSP est placée dans l'application J2EE.

- Fichier accueil.jsp

```
<HTML>
<BODY>

    <% Date date = new Date(); %>

    La date du jour : <%= date %>

</BODY>
</HTML>
```

□ Au premier appel, le serveur d'application génère une classe java à partir de accueil.jsp .

- En voici une version simplifiée :

```
out.println("<HTML>");
out.println("<BODY>");
Date date = new Date();

out.println("La date du jour : ");

out.println(date);
out.println("</BODY>");
out.println("</HTML>");
```

□ **La classe Java générée est une servlet.**

- Elle est compilée à la première utilisation
- Génération d'un fichier '.class'.

□ **Le premier appel est plus long**

- Nécessité de générer la servlet et de la compiler.

□ **Les appels suivants sont bien plus rapides**

- Appels de servlet classiques.

□ **A chaque modification de la page jsp, le serveur recompile la page automatiquement.**

□ Par forward

```
public void service(HttpServletRequest req, HttpServletResponse resp)  
throws ServletException, IOException {  
    RequestDispatcher disp =  
        this.getServletContext().getRequestDispatcher("/login.jsp");  
    disp.forward(req, resp);  
}
```

□ Par redirect

AccueilServlet

```
public void service(HttpServletRequest req, HttpServletResponse resp)  
throws ServletException, IOException {  
    resp.sendRedirect(req.getContextPath() + "/login.jsp");  
}
```

AccueilServlet

□ Dans un lien html

```
<A href="<%request.getContextPath() + "/connexion.jsp"%">  
    Se connecter  
</A>
```

- L'instruction request.getContextPath() génère dynamiquement le nom de l'application en cours. Ex : /bankonet
- Après génération par le serveur d'application, le lien ci-dessus permet d'obtenir le résultat html suivant :

```
<A href="/bankonet/connexion.jsp">  
    Se connecter  
</A>
```



1. Première approche

- Définition
- Cycle de vie
- Appel d'une page jsp

2. Structure d'une page jsp

3. Page d'accueil, page d'erreur
4. Inclusion de page
5. La bibliothèque de balises JSP
6. JSTL
7. Pour aller plus loin : écrire ses propres balises.

- Définit des attributs spécifiques à la page
- Chaque attribut ne peut être spécifié qu'une seule fois (sauf l'attribut import)
- Exemple :

```
<%@ page language="java" import="java.util.Date,java.text.*" %>
```

► Attributs (valeur par défaut)

- language (java)
- extends
- import
- session (true)
- buffer (8 ko)
- autoFlush (true)
- contentType (text/html)
- isThreadSafe (true)
- info
- errorPage
- isErrorPage (false)

□ Fragment de code Java

- Le code est inséré en l'état dans la classe java générée.
 <% : début de fragment
 %> : fin de fragment
- Un fragment peut contenir plusieurs lignes.

□ Syntaxe

```
<% Date date = new Date();  
SimpleDateFormat sdf = new SimpleDateFormat("dd MMMM");%>
```

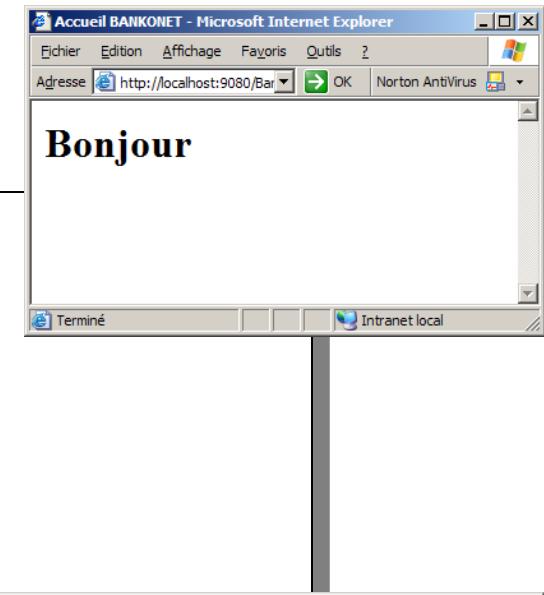


Attention aux ';' . Chaque ligne doit se terminer par un point-virgule ou par une ouverture/fermeture d'accolade.

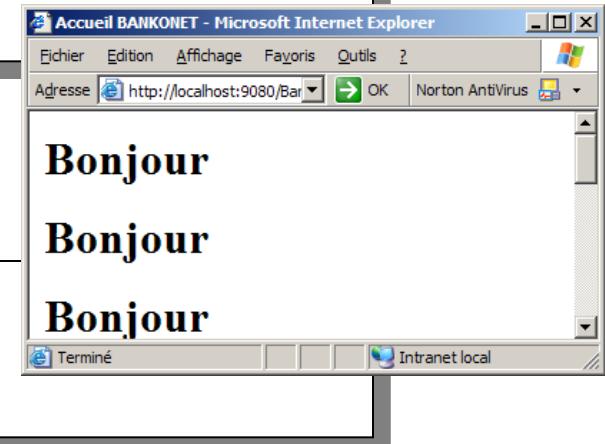
□ Exemples

```
<%@page language="java" import="java.util.Calendar" %>

<% int i = Calendar.getInstance().get(Calendar.AM_PM);
if (i == Calendar.AM) { %>
    <h1> Bonjour </h1>
<% } else { %>
    <h1> Bonsoir </h1>
<% } %>
```



```
<% for (int i=0; i<10; i++) { %>
<h1> Bonjour </h1>
<% } %>
```



□ **Fragment de code Java évalué et dont la valeur est insérée dans la réponse**

<%= : début de fragment

%> : fin de fragment

- Un fragment ne contient qu'une seule ligne

```
<%= new Date() %>
```



Attention : pas de ';' à la fin.

❑ Exemple

```
<%@page language="java" import="java.util.Date" %>
```

Nombre de secondes par heure : <%= 60*60 %>

Date et heure : <%= new Date() %>

❑ Les expressions sont très utilisées pour la récupération de données à afficher.

Cf.slides suivants.

□ Dans la page jsp, ils sont accessibles directement

Vidèle	Type	Résultat	Geste
request	<i>HttpServletRequest</i>	la réponse d'un client	request
application	<i>ServletContext</i>	la contexte de l'application	application
session	<i>HttpSession</i>	la session en cours	session

```
<%@page language="java" %>  
<% String nom = (String) session.getAttribute("nom"); %>
```

➤ Mise à disposition d'un objet dans la servlet

```
public class LivreServlet extends HttpServlet {  
  
    public void service(HttpServletRequest request, HttpServletResponse  
                        response)  
        throws ServletException, IOException {  
  
        //...  
        request.setAttribute("livre", monLivre); }  
}
```

➤ Utilisation des expressions dans la page JSP

```
<%@page language="java" import="com.bibliotheque.model.Livre" %>  
  
<% Livre livre = (Livre) request.getAttribute("livre"); %>  
  
Titre : <%= livre.getTitre() %> </td>  
  
  
Auteur : <td> <%= livre.getAuteur() %> </td>
```



Le nom des objets implicites est fixe : 'session', 'request'... (et non pas 'ses', 'req'...)

➤ Mise à disposition d'un objet dans la servlet

```
public class LivreServlet extends HttpServlet {  
    public void service(HttpServletRequest request, HttpServletResponse  
                        response)  
        throws ServletException, IOException {  
        //..  
        request.setAttribute("livre", monLivre); }  
}
```

➤ Utilisation des expressions dans la page JSP

```
<%@page language="java" import="com.bibliotheque.model.Livre" %>  
<% Livre livre = (Livre) session.getAttribute("livre"); %>  
...
```



Il faut toujours spécifier le bon contexte : un objet placé dans le contexte de requête doit être récupéré à partir de ce même contexte.

- Les commentaires jsp s'écrivent comme suit :

```
<%-- ceci est un commentaire --%>
```

- Lors du passage jsp → java, un commentaire jsp génère un commentaire java

```
// ceci est un commentaire
```

classe java générée

- Le commentaire n'apparaît pas dans le code html renvoyé au client

- Les pages JSP encapsulent des données dynamiques
- Il est généralement recommandé de désactiver le cache de ces pages pour éviter d'afficher des données obsolètes.
- Dans le cas d'un format HTML :

```
<HEAD><!-- Pour eviter le cache de la page -->
<META HTTP-EQUIV="Expires" CONTENT="0">
<META HTTP-EQUIV="Pragma" CONTENT="No-cache">
<META HTTP-EQUIV="Pragma" CONTENT="No-store">
<META HTTP-EQUIV="Cache-control", "No-cache">
</HEAD>
```



- 1. Première approche**
- 2. Structure d'une page jsp**
- 3. Page d'accueil, page d'erreur**
- 4. Inclusion de page**
- 5. La bibliothèque de balises JSP**
- 6. JSTL**
- 7. Pour aller plus loin : écrire ses propres balises.**

- Dans le web.xml, il est possible de spécifier une (ou plusieurs) pages d'accueil.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp">
...
<servlet-mapping>
    <servlet-name>DateServlet</servlet-name>
    <url-pattern>/date</url-pattern>
</servlet-mapping>
<welcome-file-list>
    <welcome-file>/login.jsp</welcome-file>
</welcome-file-list>
</web-app>
```

web.xml

- Dans l'exemple ci-dessus :
 - si l'url de mon application est http://localhost:8080/bankonet, me connecter à cette url invoque la page d'accueil login.jsp .

- Il est également possible de spécifier une (ou plusieurs) pages d'erreur.
 - Selon les codes erreur http.
 - Ou selon un type d'exception renvoyée.
- Exemple pour une erreur 404 (fichier non trouvé) :

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp">
...
    <servlet-mapping> ... </servlet-mapping>
    <welcome-file-list> ... </welcome-file-list>
    <error-page>
        <error-code>404</error-code>
        <location>/erreur.jsp</location>
    </error-page>
</web-app>
```

web.xml

- Dans l'exemple ci-dessus :
 - Si l'url http://localhost:8080/bankonet/aaaabbbddzdd (ressource non-existante) est invoquée, le serveur redirige vers la page /erreur.jsp.



- 1. Première approche**
- 2. Structure d'une page jsp**
- 3. Page d'accueil, page d'erreur**
- 4. Inclusion de page**
 - Inclusion statique
 - Inclusion dynamique
- 5. La bibliothèque de balises JSP**
- 6. JSTL**
- 7. Pour aller plus loin : écrire ses propres balises.**

□ Insère le code source d'une ressource (fichier .html, .jsp, ...) dans la page JSP

- Juste avant la phase de compilation
- la page englobante et la page incluse ne constituent **qu'une seule servlet** générée.
- La page incluse voit les imports java qui ont été déclarés dans la page englobante.

```
<%@ include file="entete.jsp"%>
```

□ Attribut 'file'

- URL relative de la ressource à insérer

□ Insère le résultat de l'exécution d'une ressource (fichier .html, .jsp, ...) dans la page JSP

- lors de la phase de traitement de la requête
- La jsp incluse génère sa propre servlet
- La page incluse ne voit pas les imports déclarés dans la page englobante.

□ Syntaxe

```
<jsp:include page="entete.jsp" flush="true"/>
```

- page : URL relative de la ressource à insérer
- flush (booléen) : s'il est positionné à true, permet un affichage progressif de la page sur le poste client (le buffer est vidé après l'inclusion).

- **Les inclusions statiques sont plus performantes.**
 - Moins de servlets à gérer.
- **Les inclusions dynamiques se rafraîchissent mieux.**
- **Les inclusions dynamiques permettent un affichage progressif de la page sur le poste client**
 - Attribut flush="true"

- TP 5 : page erreur et page de sortie
- TP 6 : Page principale d'accueil
- TP 7 : JSP Compte Courant

The screenshot shows the Mozilla Firefox browser window with the title "Menu BANKONET - Mozilla Firefox". The main content area displays the "Page Principale" of the BANKONET application. It includes a greeting "Bonjour David Dupond", a section for "Opérations disponibles" with links to "Compte Courant" and "Compte Épargne", and a "Done" button at the bottom.

The screenshot shows the Mozilla Firefox browser window with the title "Compte Courant Bankonet - Mozilla Firefox". The main content area displays the "Comptes courants" section of the BANKONET application. It shows a table with two rows of account information:

Intitulé	Solde	Découvert autorisé
courant david 1	1000.0	2000.0
courant david 2	1000.0	1000.0

Below the table, there is a link "Page principale" and a "Done" button at the bottom.

- 1 - Présentation de la technologie JEE**
- 2 - Conception et développement des Servlets**
- 3 - Conception et développement de pages JSP**
- 4 - Taglibs, EL et JSTL**
- 5 - Fonctionnalités avancées en JEE**



- 1. Première approche**
- 2. Structure d'une page jsp**
- 3. Page d'accueil, page d'erreur**
- 4. Inclusion de page**
 - Inclusion statique
 - Inclusion dynamique
- 5. La bibliothèque de balises JSP**
- 6. JSTL**
- 7. Pour aller plus loin : écrire ses propres balises.**

- Exécutées lors de la phase de traitement de la requête
- Peuvent
 - modifier le flot de sortie
 - utiliser, modifier et/ou créer des objets
- Interprétées lors de la phase de traitement de la requête
 - les valeurs de certains attributs peuvent être des expressions JSP
- Exemple de syntaxe :

```
<jsp:useBean id="societe" scope="session"
type="com.neobject.bean.Societe" />
```

Société :

```
<jsp:getProperty name="societe" property="raisonSocial" />
```

- Permet de récupérer un objet porteur de données à afficher (javabean)

- Le composant peut être
 - créé ou récupéré dans un contexte existant
 - **page** (PageContext) : local à la page
 - **request** (ServletRequest) : dans la requête
 - **session** (HttpSession) : dans la session utilisateur
 - **application** (ServletContext) : dans le contexte de l'application

□ Exemple

```
<jsp:useBean id="societe" scope="session"
type="com.neobject.bean.Societe" />
<HTML>
<HEAD>
<TITLE>Titre</TITLE>
</HEAD>
<BODY>
<H1>Présentation</H1>
<UL>
<LI>Société : <%=societe.getRaisonSocial() %>
<LI>Numéro de société : <%=societe.getId() %>
</UL>
</BODY>
</HTML>
```

- Place la valeur de la propriété d'un Bean dans le flot de sortie

- convertie sous forme de chaîne de caractère (*String*)

- Syntaxe

```
<jsp:useBean id="societe" scope="session"  
type="com.neobject.bean.Societe" />
```

Société :

```
<jsp:getProperty name="societe" property="raisonSocialle" />
```

Numéro de société :

```
<jsp:getProperty name="societe" property="id" />
```



- 1. Première approche**
- 2. Structure d'une page jsp**
- 3. Page d'accueil, page d'erreur**
- 4. Inclusion de page**
 - Inclusion statique
 - Inclusion dynamique
- 5. La bibliothèque de balises JSP**
- 6. JSTL**
- 7. Pour aller plus loin : écrire ses propres balises.**

□ **JSTL est un ensemble de bibliothèques de balises correspondant aux besoins courants de développement de pages JSP**

- Parcours d'une collection d'objets
- Accès à un JavaBean
- Formatage de nombres, dates
- ...

□ Une norme pérenne

- Historiquement, simple projet Apache
- Depuis, intégrés à JEE depuis la version 1.4

□ Mise en place

- Simple : pas de code spécifique à déployer

Description	Préfixe	URI
Core	c	http://java.sun.com/jstl/core
I18N & Formatting	fmt	http://java.sun.com/jstl/fmt
XML processing	x	http://java.sun.com/jstl/xml
Database Access	sql	http://java.sun.com/jstl/sql

□ Import dans chaque JSP

```
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
```

□ Si l'import a été omis, il n'y a pas d'erreur de compilation.

- Fonctionnement anormal en phase d'exécution.

□ Utilisation des balises

```
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
```

Bonjour <c:out value="\${client.prenom}" />

c:out écrit dans le flux de la
JSP (comme <%= %>)

Expression dynamique évaluée à l'exécution :
Expression Language (EL)

□ Langage simple et flexible d'accès aux données

- \${...} pour encadrer les expressions dynamiques

\${expression}

- Tout objet présent dans le scope de la JSP peut être utilisé comme variable dans une expression EL
- Pas de typage
- Conversions automatiques (nombres ⇔ String)
- Evaluation des expressions à l'exécution



Pas de vérification syntaxique, attention aux fautes de frappe :
il y aura une erreur à l'exécution

□ Accès à une variable

```
 ${client}
```

- Recherche l'attribut `client` dans les contextes page, request, session, application
- Retourne l'attribut dès qu'il est trouvé, sinon retourne null

□ Accès à une propriété

```
 ${client.nom}
```

- Invocation du getter sur l'objet
- Renvoie une erreur si la propriété n'existe pas

□ Accès à une Map

```
 ${maMap["uneClé"]}
```

- Retourne la valeur associée à la clé, null s'il n'y en a pas

□ Accès à une List ou un tableau

- Retourne la valeur associée à l'index, null s'il n'y en a pas

```
 ${maListe[1]}  
 ${monTableau[i]}
```

□ On peut enchaîner les appels

```
 ${mesClients[i].adresse.ville}  
 ${client.comptes[0].solde}
```

□ Objets implicites EL pour faciliter l'accès aux principaux objets

- Page *pageContext*
- Requête *pageContext.request*
- Réponse *pageContext.response*
- Session *pageContext.session*
- Application *pageContext.servletContext*

- Paramètre de requête (valeur simple) *param*
- Paramètre de requête (valeur multiple) *paramValues*

- Accès à un contexte particulier
pageScope requestScope sessionScope applicationScope

```
 ${sessionScope.client}  
 ${param["login"]}
```

Operateur	Description
.	Accès à une propriété de Bean
[]	Accès à un élément d'une liste
()	Sous-expression
+	Addition
-	Soustraction ou nombre négatif
/ or div	Division
% or mod	Modulo (reste)
== ou eq	Test d'égalité
!= ou ne	Test d'inégalité
< ou lt	Test d'infériorité
> ou gt	Test de supériorité
<= ou le	Inférieur ou égal
>= ou gt	Supérieur ou égal
&& ou and	ET logique
ou or	OU logique
! ou not	Inversion de booléen
empty	Test de valeur vide (null, chaîne vide, collection vide)

Description	Préfixe	URI
Core	c	http://java.sun.com/jstl/core

□ Services génériques

- Écriture JSP
- Boucles
- Conditions
- Inclusions
- URL
- ...

□ <c:out /> renvoie une expression après évaluation

- Exemple:

```
<c:out value="${user.name}" default="Inconnu" />  
Dupont
```

- Évaluation et chaîne de caractère :

```
<c:out value="Bonjour, ${user.name}" />  
Bonjour, Dupont
```

- L'attribut escapeXml

```
<c:out value=<tag>" />  
&lt;tag&gt;  
  
<c:out value=<tag>" escapeXml="false" />  
<tag>
```

- Une erreur fréquente

```
<c:out value="name" />  
name
```

Exemple d'implémentation Scriptlet (Code Java + html)

```
<%@ page language="java" %>
<%@page import="com.shopping.cart.bean.ShoppingCart,
com.shopping.cart.bean.Article, java.util.*" %>
<html>
  <body>
<table border="1">
  <% ShoppingCart cart = (ShoppingCart) session.getAttribute("cart");
List articleList = cart.getArticles();
Iterator articleIte = articleList.iterator();

while (articleIte.hasNext()) {
Article tempArticle = (Article) articleIte.next(); %>
<tr>
  <td> <%= tempArticle.getTitle()%> </td>
  <td> <%= tempArticle.getAuthor()%> </td>
  <td> <%= tempArticle.getPrice()%> </td>
</tr>
<% } %>
</table>
</body>
</html>
```



Exemple d'implémentation JSTL

```
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>

<html>
<body>
<table>
<c:forEach items="${cart.articles}" var="tempArticle">
  <tr>
    <td> <c:out value="${tempArticle.title}" /> </td>
    <td> <c:out value="${tempArticle.author}" /> </td>
    <td> <c:out value="${tempArticle.price}" /> </td>
  </tr>
</c:forEach>
</table>
</body>
</html>
```



(*)

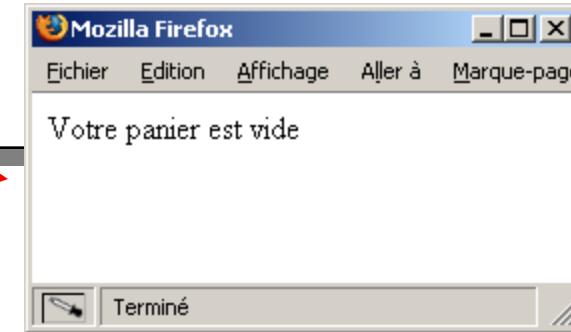
The screenshot shows a Mozilla Firefox window with the title "shoppingCart - Mozilla Firefox". Below the title bar is a menu bar with File, Edit, View, Go, Bookmarks, Tools, and Help. The main content area displays a table with the heading "Articles en stock". The table has three columns: Title, Author, and Price. The data rows are:

Sur la route de la soie	Philippe Valery	15.0
La terre vue du ciel	Yann Arthus Bertrand	50.0
Cul-de-sac	Douglas Kennedy	50.0

At the bottom of the browser window, there is a "Done" button.

□ Gestion du cas d'une collection vide

```
<c:choose>
    <c:when test="${!empty cart.articles}">
        <table>
            <c:forEach items="${cart.articles}" var="tempArticle">
                <tr>
                    <td> <c:out value="${tempArticle.title}" /> </td>
                    <td> <c:out value="${tempArticle.author}" /> </td>
                    <td> <c:out value="${tempArticle.price}" /> </td>
                </tr>
            </c:forEach>
        </table>
    </c:when>
    <c:otherwise>
        Votre panier est vide
    </c:otherwise>
</c:choose>
```



- On peut fixer le début et la fin d'une boucle.
- La variable définie dans varStatus a des propriétés permettant de suivre le parcours de la boucle

```
<table>
<c:forEach items="${cart.articles}" var="tempArticle"
begin="${lineNumberStart}" end="${lineNumberEnd}"
varStatus="loopStatus">
    <tr>
        <td> Ligne no <c:out value="${loopStatus.count}" /> </td>
        <td> Article no <c:out value="${loopStatus.index}" /> dans le panier</td>
        <td> <c:out value="${tempArticle.title}" /> </td>
        <td> <c:out value="${tempArticle.author}" /> </td>
        <td> <c:out value="${tempArticle.price}" /> </td>
    </tr>
</c:forEach>
</table>
```

Commence à 1



□ <c:url /> construit une URL

Exemple depuis une page http://localhost/one/dir/default.jsp

```
<c:url value="page.jsp" />  
page.jsp
```

```
<c:url value="/page.jsp" />  
http://localhost/one/page.jsp
```

```
<c:url value="/page.jsp" context="two" />  
http://localhost/two/page.jsp
```

```
<c:url value="page.jsp">  
<c:param name="id" value="${id}" />  
<c:param name="title" value="modifier page" />  
</c:url>  
http://localhost/one/dir/page.jsp?id=2&title=modifier+page
```

Exemple d'utilisation:

```
<a href=">Cliquez ici!</a>
```

Exemple d'utilisation avec changement de contexte:

```
<a href=">Cliquez ici!</a>
```

L'expression est exécutée sous certaine(s) condition(s)

- Si plusieurs conditions se succèdent, elles ne sont pas mutuellement exclusives.

```
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<c:if test="${article.price==15}" >
    Prix de 15
</c:if>

<c:if test="${article.price>40 || article.price<10 }" >
    Prix trop grand ou trop petit
</c:if>
```

 JSTL ne dispose pas de structure if/else. Mais il dispose de choose/when/otherwise.

□ La balise <c:choose> est utilisée pour des conditions mutuellement exclusives

- Dans l'exemple ci-dessous, si le prix est supérieur à 100, les deux derniers tests ne sont pas effectués.

```
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<c:choose>
    <c:when test="${article.price > 100}">
        Gamme luxe
    </c:when>
    <c:when test="${article.price > 0 }">
        Catégorie normale
    </c:when>
    <c:otherwise >
        Cet article n'est pas en vente
    </c:otherwise>
</c:choose>
```

- **<c:import /> importe le contenu d'une ressource accédée via une url**

```
<h1>Bienvenue sur le site de Google :</h1>
<c:import url="http://www.google.fr" />
```



- ❑ **<c:redirect>** : permet de rediriger vers une autre url

```
<c:choose>
    <c:when test="${empty user}">
        <c:redirect url="/login.jsp" />
    </c:when>
    <c:otherwise >
        une page...
    </c:otherwise>
</c:choose>
```

Description	Préfixe	URI
I18N & Formatting	fmt	http://java.sun.com/jstl/fmt

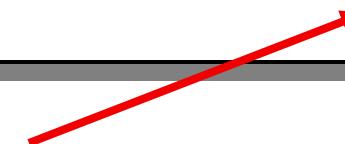
□ Formatage et internationalisation

- Messages internationalisés
- Formatage de dates
- Formatage de nombres
- ...

□ Formatage de dates, nombres, monnaie

```
<%@ taglib prefix= "fmt" uri="http://java.sun.com/jstl/fmt" %>  
  
<fmt:formatNumber value="12.3" pattern=".00"/>  
  
<fmt:formatNumber value="${product.price}" type="currency" />  
  
<fmt:formatNumber value="${stats.result}" type="percent" />  
  
<%-- Création d'une variable contenant la date du 5/3/65 --%>  
  
<fmt:parseDate var="aDay" value="05/03/1965" pattern="dd/MM/yyyy" />  
  
<fmt:formatDate value="${aDay}" type="date" dateStyle="full"/>  
  
<fmt:formatDate value="${aDay}" type="date" pattern="dd MMM yyyy 'à' HH:mm"/>
```

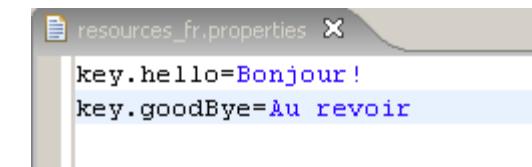
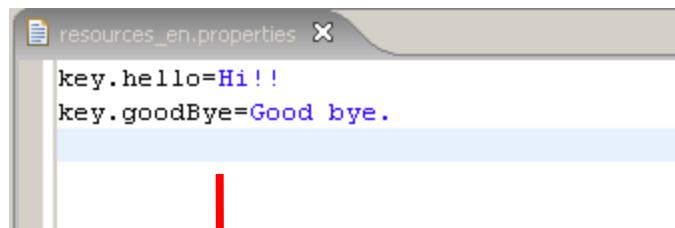
Voir [java.text.SimpleDateFormat](#)



```
<%@ taglib prefix= "fmt" uri="http://java.sun.com/jstl/fmt" %>  
<fmt:setLocale scope="session" value="${myLocal}" />  
<fmt:bundle basename="com.site.resources.resources">  
<fmt:message key="key.hello" />  
</fmt:bundle>
```

myLocal=en

myLocal=fr



- Préparez un **include statique** regroupant toutes les déclarations de bibliothèques de tags.

```
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<%@ taglib prefix= "fmt" uri="http://java.sun.com/jstl/fmt" %>
```

- Et dans chaque page rajouter un **include statique**

```
<%@ include file="includes/includeTagLibs.jsp" %>
```

- TP 8 : JSP CompteEpargne
- TP9, TP10 : Virement, traitement du virement

The screenshot shows a Mozilla Firefox browser window with the title "Compte courant Bankonet - Mozilla Firefox". The main content area displays a table titled "Comptes Epargne" with two rows of data:

Intitulé	Solde	Taux d'intérêt	Plafond
epargne david 1	1000.0	10000.0	0.2
epargne david 2	5000.0	120000.0	0.1

Below the table, there is a link labeled "Page principale". At the bottom of the browser window, there is a status bar with the word "Done".

The screenshot shows a Mozilla Firefox browser window with the title "Virement Bankonet - Mozilla Firefox". The main content area displays a form for a transfer:

Virement

Compte source	Compte destination	Montant à virer
courant david 1	epargne david 1	200

Below the form, there are two buttons: "Virer" and "Réinitialiser". Underneath the form, there is a link labeled "Page Principale". At the bottom of the browser window, there is a status bar with the word "Done".



- 1. Première approche**
- 2. Structure d'une page jsp**
- 3. Page d'accueil, page d'erreur**
- 4. Inclusion de page**
 - Inclusion statique
 - Inclusion dynamique
- 5. La bibliothèque de balises JSP**
- 6. JSTL**
- 7. Pour aller plus loin : écrire ses propres balises.**

□ **Les bibliothèques de balises permettent de définir de nouvelles actions associées à une balise personnalisée.**

- portable (indépendant du conteneur de JSP)
- mécanisme simple
- une large gamme d'actions peut être décrite par ce mécanisme

□ **Pour cela on doit :**

1. Créer un descripteur de bibliothèque de balise (*Tag Library Descriptor*)
2. Créer des classes gestionnaires (*handler*) associées aux balises
3. Déclarer la bibliothèque dans le descripteur de déploiement (*web.xml*)
4. Déclarer les bibliothèques utilisées dans la JSP

□ Tag Library Descriptor

- document XML d'extension *.tld
- informations générales de la bibliothèque
- description de la syntaxe de chaque balise
 - Nom de la balise
 - Nom des attributs (obligatoires ou optionnels)
 - Contenu de la balise (body)
- association d'une balise à sa classe gestionnaire (*TagHandler*)

□ Permet à des outils (éditeurs) de prendre connaissance des nouvelles balises disponibles

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE taglib PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library
1.1//EN" "http://java.sun.com/j2ee/dtds/web-jsptaglibrary_1_1.dtd">
<taglib>
    <tlibversion>1.0</tlibversion>
    <jspversion>1.1</jspversion>
    <shortname>bankonet</shortname>
    <uri>http://www.bankonet.fr/tags</uri>
    <tag>
        <name>link</name>
        <tagclass>com.bankonet.taglib.LinkTag</tagclass>
        <bodycontent>empty</bodycontent>
        <attribute>
            <name>href</name>
            <required>true</required>
        </attribute>
        <attribute>
            <name>label</name>
            <required>true</required>
        </attribute>
    </tag>
</taglib>
</taglib>
```

□ La classe gestionnaire de balise hérite de :

- *javax.servlet.jsp.tagext.TagSupport*
- *javax.servlet.jsp.tagext.BodyTagSupport*

□ Pour chaque attribut de la balise, on a un attribut Java avec le même nom (et ses accesseurs)

□ Les méthodes des interfaces sont implémentées

- *doStartTag()* appelé à l'analyse de la balise de début
- *doEndTag()* appelé à l'analyse de la balise de fin
- *doInitBody()* appelé avant le traitement éventuel du contenu de la balise
- *doAfterBody()* appelé après le traitement éventuel du contenu de la balise

Exemple : génération d'un lien html avec son contexte

```
public class LinkTag extends TagSupport {  
    private String href;  
    private String label;  
  
    public void setHref(String pHref) {href = pHref;}  
    public void setLabel(String pLabel) {label = pLabel;}  
  
    public int doStartTag() throws JspException {  
        try {  
            HttpServletRequest request = (HttpServletRequest) pageContext.getRequest();  
            JspWriter out = pageContext.getOut();  
            out.println("<a href='" + request.getContextPath() + href + "'>"  
                + label + "</a>");  
        }  
        catch (IOException e) { e.printStackTrace(); }  
        return SKIP_BODY;  
    }  
}
```

- L'interpréteur de pages jsp invoque la création d'une instance LinkTag.
- Les méthodes `setHref(...)` et `setLabel(...)` sont invoquées (setters correspondant aux attributs de la balise).
- La méthode `doStartTag(...)` est invoquée. Son contenu sera ajouté à la réponse http.
- La méthode `doEndTag(...)` est invoquée. Son contenu sera ajouté à la réponse http.

□ Déclaration dans la page jsp

```
<%@ taglib prefix="bankonet" uri="/http://www.bankonet.fr/tags"%>
```

- uri : URI déclarée dans le fichier tld.
- prefix : préfixe à utiliser pour marquer une balise personnalisée

□ Appel

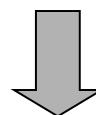
```
<bankonet:link href="/Virement" label="Effectuer un virement"/>
```

* : certains éditeurs ne supportent pas la syntaxe proposée.
Dans ce cas, l'uri doit référencer le fichier directement :

```
<%@ taglib prefix="bankonet" uri="/WEB-INF/bankonetTags.tld"%>
```

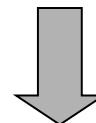
```
<bankonet:link href="/virement" label="Effectuer un virement"/>
```

Déclaration dans la page jsp



```
<a href='/BankonetWeb/Virement'>  
    Effectuer un virement  
</a>
```

Code html généré



- Effectuer un virement

Affichage dans le navigateur

FICHIER DE BALISE (xxx.tag)

□ Comment réutiliser une partie de page dynamique ?

- Copier-coller
- Inclusion de jsp (jsp:include ou c:import)
 - Passage de paramètre limité
- Balises personnalisées
 - Maîtrise de Java obligatoire

□ JSP 2.0 : fichiers de balises (tag files)

- Équivalent des balises personnalisées mais sous forme proche d'un fichier JSP

□ Crédit d'un fichier de balise

- dans un sous-répertoire de WEB-INF/tags
- Nom du fichier = nom de la balise + .tag (ou .tagx)
 - Ex : WEB-INF/tags/texte/majuscules.tag

□ Référencer le répertoire dans la JSP

- Association préfixe / répertoire

□ Utiliser la balise

```
<%@ taglib prefix="txt" tagdir="/WEB-INF/tags/texte" %>
<txt:majuscules>
Ceci est un exemple d'appel de fichier balise.
</txt:majuscules>
```

❑ Syntaxe similaire aux JSP

❑ Quelques différences

➤ Directives

- @page non disponible
- @tag, @attribute et @variable spécifiques

➤ Actions spécifiques (jsp:invoke et jsp:doBody)

❑ Objets implicites

➤ request, response, session, application, out, config (idem JSP)

➤ pageContext n'existe pas, utiliser **jspContext**

□ Insertion du corps de la balise

- <jsp:doBody />

□ Utilisation du corps de la balise

- <jsp:doBody var="nomVariable"/>
- Valeur stockée dans un attribut de JspContext
- Possibilité d'indiquer la portée avec l'attribut scope
 - <jsp:doBody var="nomVariable" scope="request"/>

□ Directive attribute

- <%@ attribute name="nom" %>
- Équivalent de la déclaration d'attribut dans le TLD
 - name (obl.) : nom de l'attribut
 - required (opt.) : attribut obligatoire ? (false par défaut)
 - type : type de l'attribut, java.lang.String par défaut; types primitifs interdits
 - rtxexprvalue (opt.) : valeur dynamique ? (true par défaut)
 - description (opt.) description, utile pour des outils

❑ Fonction = méthode statique d'une classe

```
public class StringUtils {  
    public static String mettreEnMajuscules(String texte) {  
        return texte.toUpperCase();  
    }  
}
```

❑ Configuration dans le TLD

```
<taglib>  
    ...  
    <shortname>myfn</shortname>  
    <uri>http://www.bankonet.fr/utils</uri>  
    <function>  
        <description>Retourne le texte en majuscules</description>  
        <name>majuscules</name>  
        <function-class>com.bankonet.util.StringUtils</function-class>  
        <function-signature>  
            java.lang.String mettreEnMajuscules(java.lang.String)  
        </function-signature>  
    </function>  
</taglib>
```

□ Utilisation dans la JSP

```
<%@ taglib prefix="myfn" uri="http://www.bankonet.fr/utils" %>
texte  ${myfn:majuscules("texte") }
```

□ **Nouvel élément regroupant la configuration des JSP :**

- <jsp-config>

□ **Élément pour regrouper plusieurs JSP :**

- <jsp-property-group>
- Permet d'indiquer des propriétés communes à un groupe de jsp, regroupées par leur URL
 - <el-ignored> indique si on doit ignorer l'EL
 - <scripting-invalid> indique si les scriptlets sont interdits
 - <page-encoding> indique l'encoding des JSP
 - <include-coda> insère un pied de page
 - <include-prelude> insère un en-tête

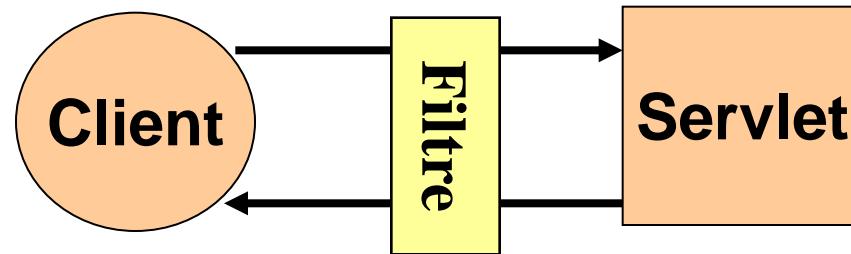
❑ Exemple

```
<jsp-config>
  <jsp-property-group>
    <url-pattern>*.jsp</url-pattern>
    <scripting-invalid>true</scripting-invalid>
    <include-prelude>entete.jspf</inclucle-prelude>
    <include-coda>piedDePage.jspf</include-coda>
  </jsp-property-group>
</jsp-config>
```

- 1 - Présentation de la technologie JEE**
- 2 - Conception et développement des Servlets**
- 3 - Conception et développement de pages JSP**
- 4 - Taglibs, EL et JSTL**
- 5 - Fonctionnalités avancées en JEE**

- 1. Les Filtres de servlet**
- 2. Les Programmes d'écoute**
- 3. Les JSR notables de JEE**

- Un filtre est un intermédiaire de passage entre le client et la servlet.



- Il peut notamment :
 - Intercepter la requête avant traitement
 - Intercepter la réponse après traitement
- Un filtre est une classe Java.
 - Il doit implémenter l'interface javax.servlet.Filter

□ Dans le web.xml, un filtre est positionné sur un type d'URI.

- Filtre portant sur une seule servlet

```
<filter-mapping>
    <filter-name>LogFilter</filter-name>
    <url-pattern>/Accueil</url-pattern>
</filter-mapping>
```

- Filtre portant sur plusieurs servlets de l'application web

```
<filter-mapping>
    <filter-name>LogFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

- Filtre portant sur tous les types d'URI

```
<filter-mapping>
    <filter-name>LogFilter</filter-name>
    <url-pattern>/filtered/*</url-pattern>
</filter-mapping>
```

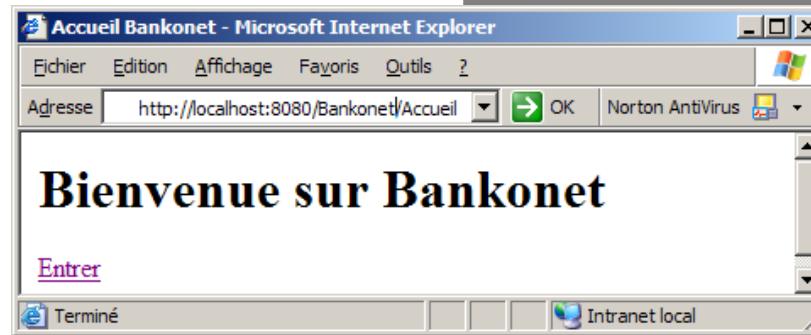
LogFilter.java

```
public class LogFilter implements Filter {
    public void doFilter(ServletRequest req,
    ServletResponse resp, FilterChain chain)
    throws ServletException, IOException {
        System.out.println("passage dans le filtre");
        chain.doFilter(req, resp);
    } ... }
```

web.xml

```
<filter>
    <filter-name>LogFilter</filter-name>
    <display-name>LogFilter</display-name>
    <filter-class>com.bankonet.filter.LogFilter
    </filter-class>
</filter>
<filter-mapping>
    <filter-name>LogFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

Appel à la Servlet "AccueilServlet"



Traces (console)

```
Console [BankonetFilteredServer (WebSphere v5.1)]
[12/07/04 16:45:04:945 CEST] 537b7cb1 WebGroup      I SRVE0180I: [Bankonet_Web] [/B
[12/07/04 16:45:04:945 CEST] 537b7cb1 SystemOut    O passage dans le filtre
[12/07/04 16:45:04:945 CEST] 537b7cb1 SystemOut    O passage dans AccueilServlet
```

- Le filtre englobe l'appel de ressource (servlet, fichier statique...).
- La ressource est appelée lors de l'invocation de la méthode Chain.doFilter(...).
- Le code placé avant cette invocation sera exécuté avant l'appel.
- Le code placé après cette invocation sera exécuté après l'appel.

```
public class LogFilter implements Filter {  
  
    public void doFilter(ServletRequest req, ServletResponse resp,  
        FilterChain chain) throws ServletException, IOException {  
  
        System.out.println("Avant appel de servlet");  
  
        chain.doFilter(req, resp);  
  
        System.out.println("Après appel de servlet");  
  
    } ...  
}
```



Si le Chain.doFilter(...) est omis, la servlet n'est pas traitée et la réponse n'est pas transmise au client

Suppression des blancs html

➤ Pour limiter le trafic du serveur web vers le client.

Authentification/sécurisation

Journalisation (logs) et d'audit

Conversion d'images

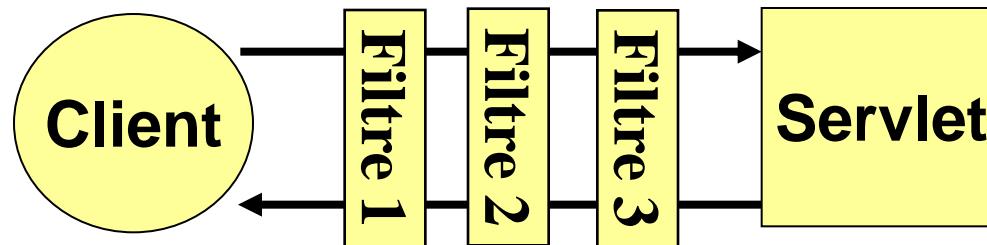
Traduction de format (XML→HTML, XML → WML)

Cache

Ré-écriture d'url (création de répertoires virtuels)

...

- Il est possible d'associer plusieurs filtres à une URI donnée.



- Ils sont exécutés dans l'ordre de déclaration dans le web.xml.

```
<filter-mapping>
    <filter-name>LogFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>

<filter-mapping>
    <filter-name>HtmlFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

- Dans l'exemple de filtre "LogFilter" vu précédemment, la réponse http n'est pas modifiée par le filtre.
- Certains filtres de servlet modifient la réponse http
 - Ex : suppression des blancs html.
- L'écriture de ce type de filtre est beaucoup plus complexe.
 - Pour plus d'informations sur le sujet :
<http://java.sun.com/products/servlet/Filters.html>
<http://www.javaworld.com/javaworld/jw-06-2001/jw-0622-filters-p3.html>

□ Paramétrage dans web.xml

- depuis servlet 2.4
- par défaut, filtre activé seulement par accès direct
- Valeurs possibles :
 - REQUEST (accès direct)
 - FORWARD (dispatcher.forward)
 - INCLUDE (dispatcher.include)
 - ERROR

```
<filter-mapping>
  <filter-name>LogFilter</filter-name>
  <url-pattern>/*</url-pattern>
  <dispatcher>REQUEST</dispatcher>
  <dispatcher>FORWARD</dispatcher>
</filter-mapping>
```

- 1. Les Filtres de servlet**
- 2. Les Programmes d'écoute**
- 3. Les JSR notables de JEE**

□ **Un système d'abonnement permet à certains objets d'être notifiés lorsque des évènements particuliers ont lieu**

- ServletContextListener : cycle de vie du contexte d'application
 - contextInitialized (création)
 - contextDestroyed (destruction)
- ServletContextAttributeListener : modification du contenu du contexte d'application
 - attributeAdded : (ajout, remplacement, suppression)
 - attributeRemoved
 - attributeReplaced

- HttpSessionListener : cycle de vie des sessions (création et destruction)
 - sessionCreated
 - sessionDestroyed
- HttpSessionAttributeListener : modification du contenu des sessions d'application (ajout, remplacement, suppression)
 - attributeAdded
 - attributeRemoved
 - attributeReplaced
- HttpSessionBindingListener : les objets implémentant cette interface seront notifiés lorsqu'ils sont stockés ou supprimés d'une session
 - valueBound
 - valueUnbound

- **ServletRequestListener** : cycle de vie d'une requête (création et destruction)
 - `requestDestroyed`
 - `requestInitialized`
- **ServletRequestAttributeListener** : modification du contenu des attributs de la requête (ajout, remplacement, suppression)
 - `attributeAdded`
 - `attributeRemoved`
 - `attributeReplaced`

□ Déclaration dans le web.xml

```
<!-- déclarations précédentes : filter*, filter-mapping* -->
<listener>
    <listener-class>
        com.bankonet.listener.NeobjectSessionListener
    </listener-class>
</listener>
<!-- déclarations suivantes : servlet*, servlet-mapping* -->
```

- Pour chaque élément **<listener>**, une instance de la classe est créée au démarrage du serveur.
- Si plusieurs listeners du même type (**HttpSessionListener** par ex.) sont déclarés, ils sont notifiés dans l'ordre de déclaration dans le descripteur.

```
public class NeobjectSessionListener
    implements HttpSessionListener {
    public void sessionCreated(HttpSessionEvent event) {
        System.out.println("Session created");
        System.out.println(event.getSource());
    }
    public void sessionDestroyed(HttpSessionEvent event) {
        System.out.println("session destroyed");
    }
}
```

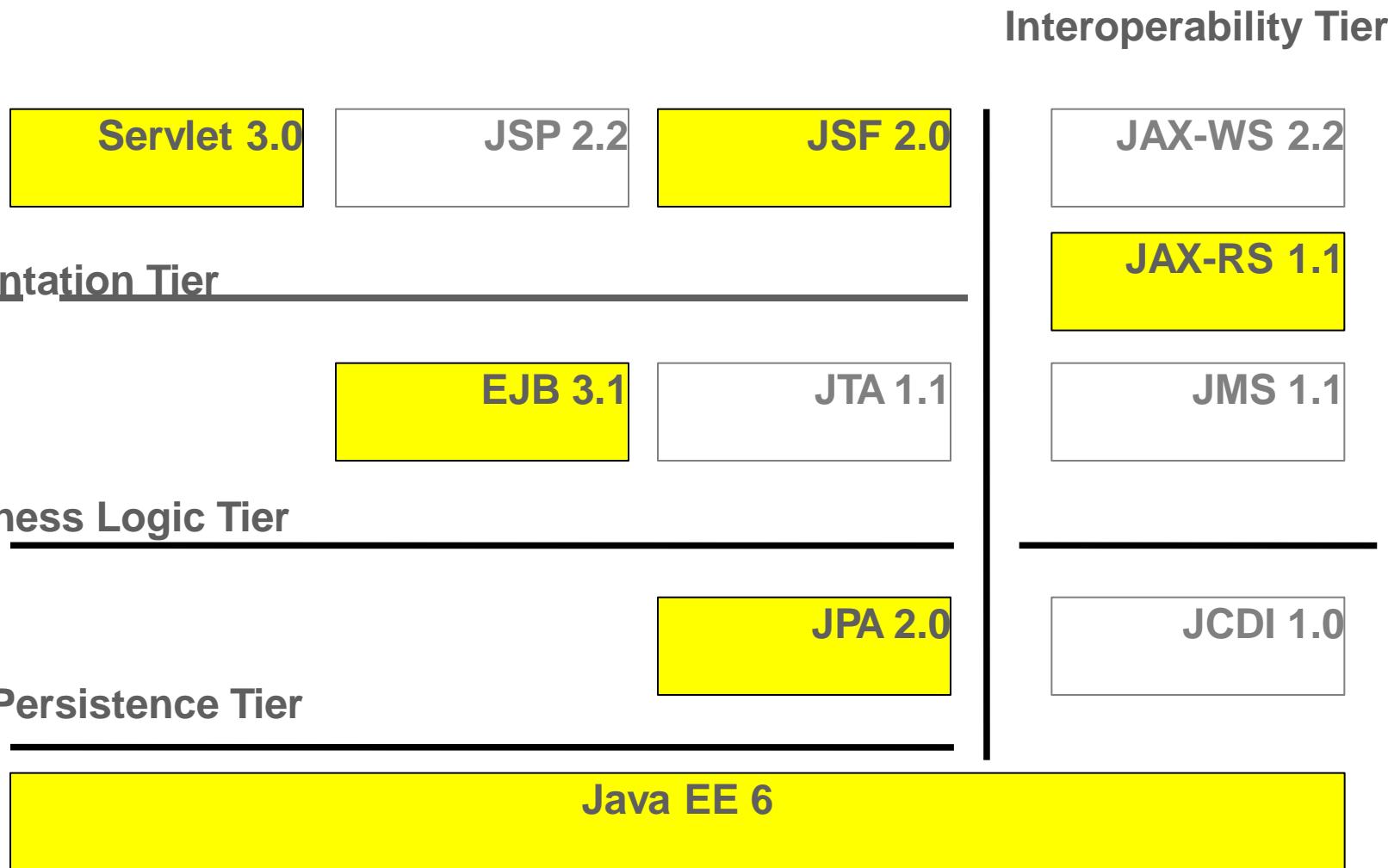
Dans web.xml :

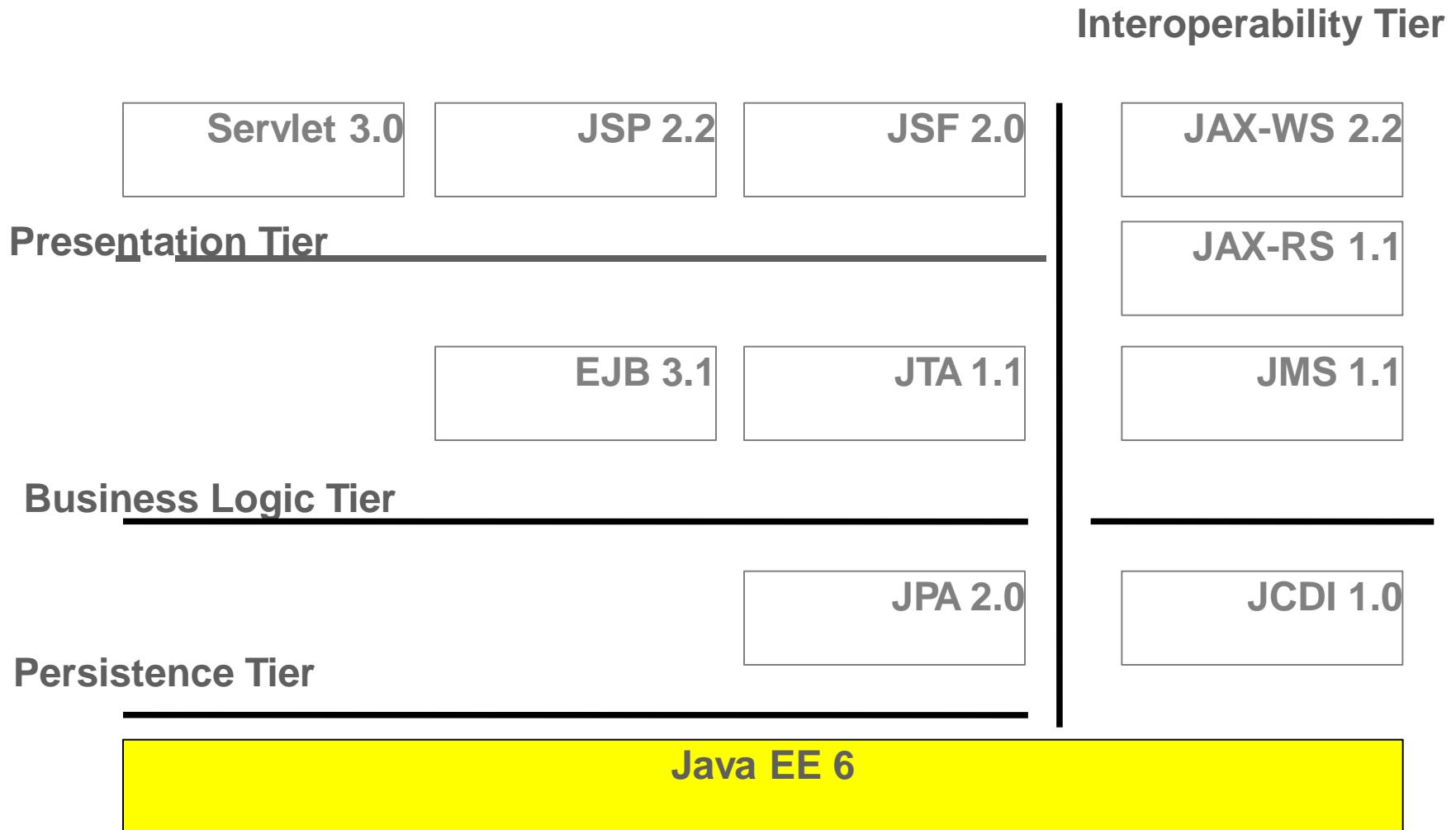
```
<listener>
    <listener-class>
com.bankonet.listener.NeobjectSessionListener
    </listener-class>
</listener>
```

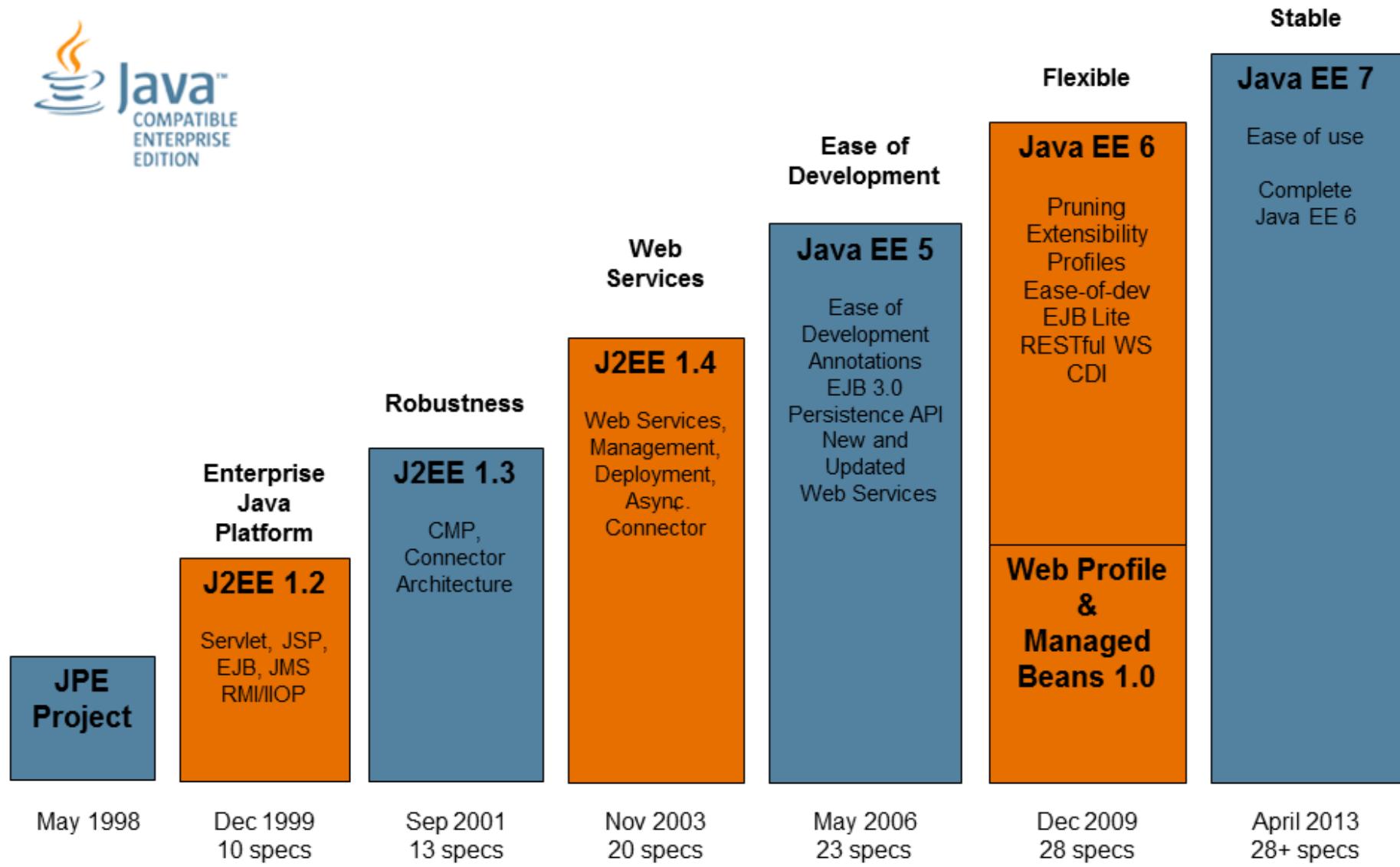
- Le container prend en compte la déclaration du programme d'écoute dans le web.xml
- Il instancie NeobjectSessionListener au démarrage du serveur
- A chaque création / destruction de session, les méthodes correspondantes sont invoquées.

Note : HttpSessionEvent permet de récupérer des informations sur la session en cours et sur l'objet déclencheur de l'événement.

- 1. Les Filtres de servlet**
- 2. Les Programmes d'écoute**
- 3. Les JSR notables de JEE**







Web Services

JAX-RPC	1.1
JAX-WS	2.2
JAXM	1.0
JAX-RS	1.1
JAXR	1.1
StAX	1.0
Web Services	1.2
Web Services Metadata	1.1

Enterprise

EJB	3.1
JAF	1.1
JavaMail	1.4
JCA	1.6
JMS	1.1
JPA	2.0
JTA	1.1

Web

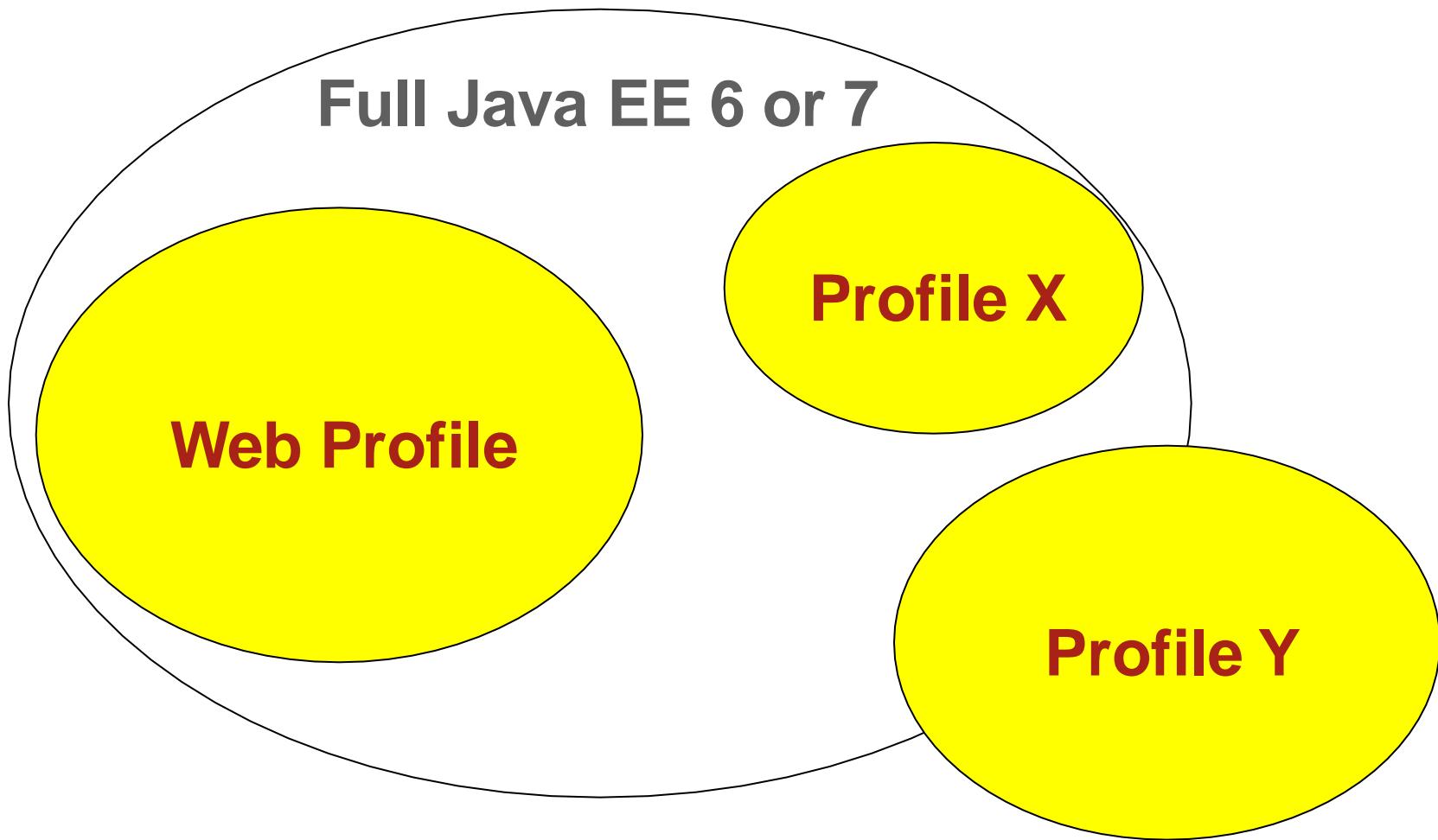
JSF	2.0
JSP	2.2
JSTL	1.2
Servlet	3.0
Expression Language	1.2

Management, Security and other

JCDI	1.0
JACC	1.1
Common Annotations	1.0
Java EE Application Deployment	1.2
Java EE Management	1.1
Java Authentication Service Provider Interface for Containers	
Debugging Support for Other Languages	
Bean Validation	

+ Java SE 6

JAXB	2.2
JDBC	4.0
JNDI	1.5
RMI	
JMX	
JAAS	
JAXP...	



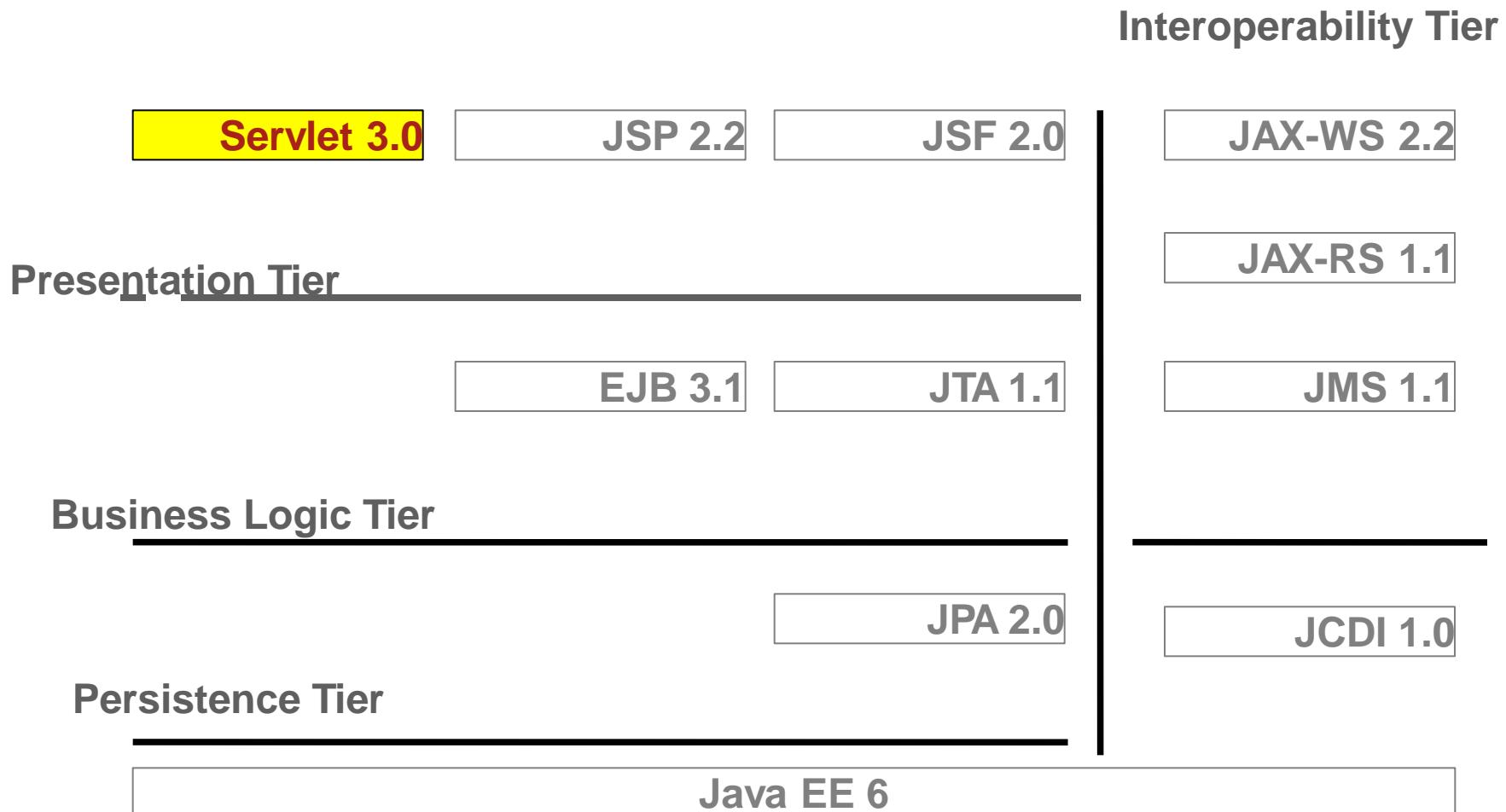
Sous-ensemble de la plateforme complète

- Se concentre sur le développement web
- Spécification séparée
- D'autres profils viendront :
 - Minimal (Servlet/JSP)
 - Portal....

Servlet	3.0
JSP	2.2
EL	1.2
JSTL	1.2
EJB Lite	3.1
JTA	1.1
JPA	2.0
JSF	2.0

« ...you'll see gradual move toward the Web profile »
- Rod Johnson (Spring)

- Certaines spécifications dans JEE 6 deviennent optionnelles pour JEE 7, car elles sont destinées à disparaître (« pruning »)
- Pruned en Java EE 6
 - Entity CMP 2.x
 - JAX-RPC
 - JAX-R
 - JSR 88 (Java EE Application Deployment)
- Plus fort que @Deprecated
- La spécification peut néanmoins continuer à évoluer en dehors de Java EE
- Aide les éditeurs de futurs serveurs d'applications JEE



Modèle de programmation basé sur les annotations:

- @WebServlet
- @ServletFilter
- @WebServletContextListener
- @InitParam

Descripteur de déploiement optionnel (web.xml)

```
package web;
import java.io.*;
import javax.servlet.*;
import javax.servlet.annotation.*;
import javax.servlet.http.*;

@WebServlet(name="cs",urlPatterns={"/fs","*.do"})
public class FirstServlet extends HttpServlet {
```

```
public class MyServlet extends HttpServlet {  
    public void doGet (HttpServletRequest req,  
HttpServletResponse res) {  
    ....  
}  
}
```

Deployment descriptor (web.xml) :

```
<web-app>  
    <servlet>  
        <servlet-name>MyServlet</servlet-name>  
        <servlet-class>samples.MyServlet</servlet-class>  
    </servlet>  
  
    <servlet-mapping>  
        <servlet-name>MyServlet</servlet-name>  
        <url-pattern>/MyApp</url-pattern>  
    </servlet-mapping>  
    ...  
</web-app>
```

```
@WebServlet(urlMappings= { "/MyApp" })  
  
public class MyServlet extends HttpServlet {  
  
    public void doGet (HttpServletRequest req,  
                      HttpServletResponse res) {  
  
        ....  
    }  
}
```

- Le web.xml est optionnel
- Idem pour les filters et listeners

Le web.xml peut-être fragmenté

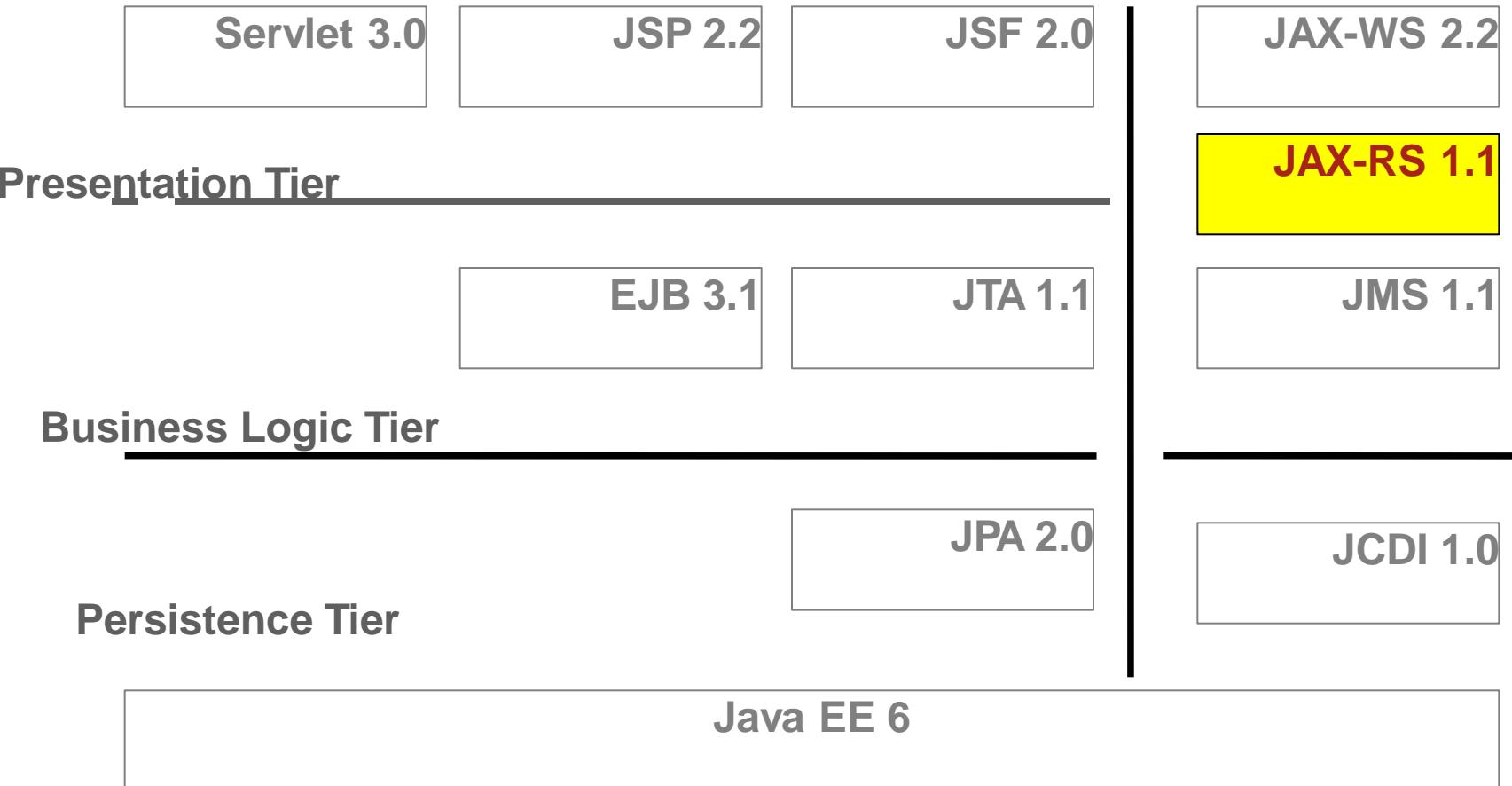
- Partitionnement logique d'une application web

Les annotations et les fragments sont fusionnés

```
<web-fragment>
    <servlet>
        <servlet-name>myservlet</servlet-name>
        <servlet-class>samples.MyServlet</servlet-class>
    </servlet>
    <listener>
        <listener-class>samples.MyListener</listener-class>
    </listener>
</web-fragment>
```

- Les servlets doivent attendre les réponses de :

- Web service
 - Connection JDBC
 - Message JMS....
- `@WebServlet (asyncSupported = true)`
- Nouvelles APIs dispo pour `ServletRequest / ServletResponse`
- Mettre en pause, résumer, suspendre et observer le status d'une requête HTTP



Construction de services RESTful

- Basé sur les POJO et les annotations
- Les fonctions et les données sont considérées comme des ressources
- Se mappe sur HTTP

HTTP	Action	HTTP	Action
GET	Get a resource	PUT	Create or update
POST	Create a resource	Delete	Deletes a resource

```
@Path("/helloworld")
public class HelloWorldResource {

    @GET
    @Produces("text/plain")
    public String sayHello() {
        return "Hello World";
    }
}
```

• → <http://example.com/helloworld>

Requête :

GET /helloworld HTTP/1.1

Host: example.com Accept: text/plain

Réponse :

HTTP/1.1 200 OK

Date: Wed, 12 Nov 2008 16:41:58 GMT

Server: Apache/1.3.6

Content-Type: text/plain; charset=UTF-8

Hello World

```
@Path("/helloworld")
public class HelloWorldResource {

    @GET @Produces("image/jpeg")
    public byte[] paintHello() { ... }

    @POST
    @Consumes("text/xml")
    public void updateHello(String xml) { .. }

    ...
}
```

```
@Path("/users/{userId}")  
public class UserResource {  
  
    @GET  
    @Produces("text/xml")  
    public String getUser(  
        @PathParam("userId") String userName) {  
  
    ...  
}  
  
}
```

http://example.com/users/Smith123

- Et encore de nombreuses autres fonctionnalités...
- Top ten des fonctionnalités JEE 6 :
 1. EJB packagés dans un WAR
 2. Points d'extension Servlet et CDI
 3. web.xml optionnel
 4. Injections de dépendances type-safe
 5. Evènements CDI
 6. Standards Facelets pour JSF
 7. API EJBContainer
 8. Tâches planifiées : @Schedule
 9. EJB No Interface View
 10. Web Profile

Cloud

❑ Nouveautés en JEE 7 :

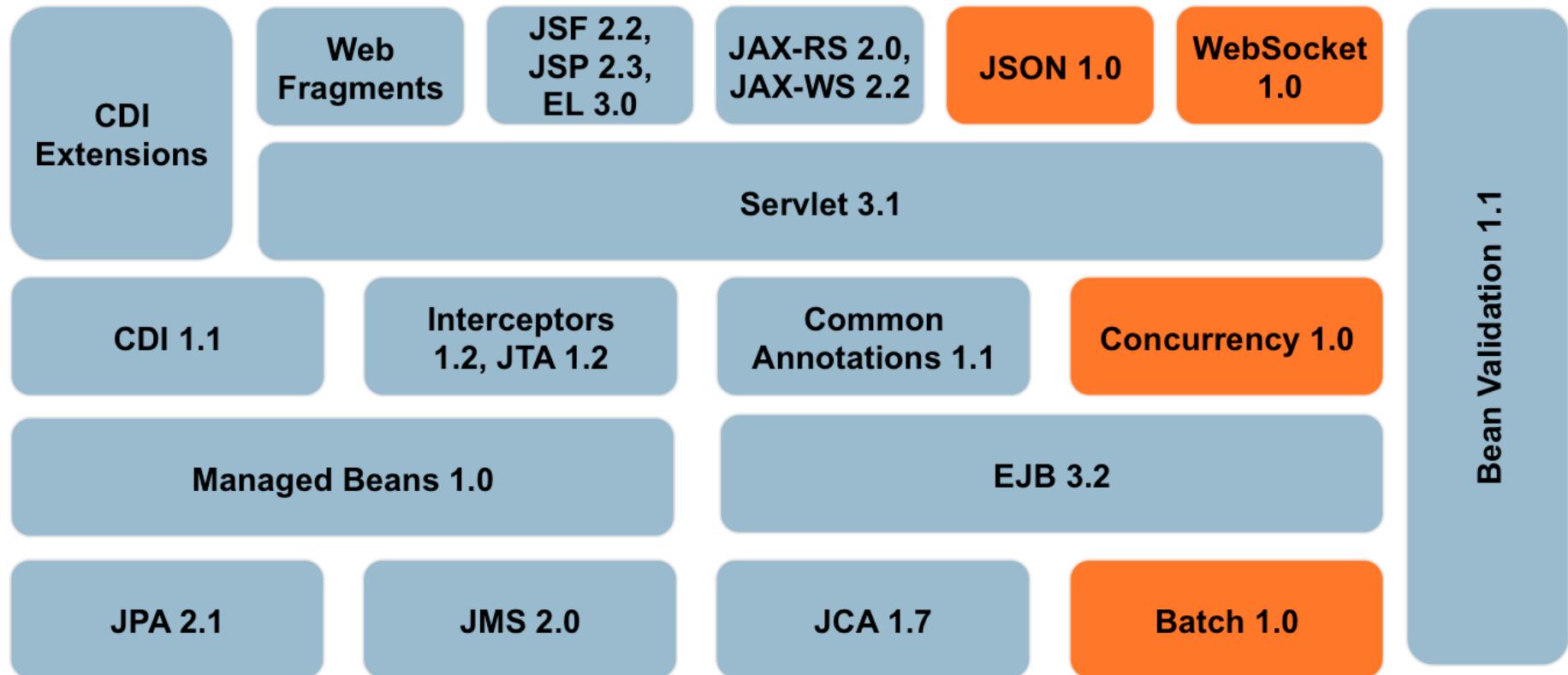


- More annotated POJOs
- Less boilerplate code
- Cohesive integrated platform

- WebSockets
- JSON
- Servlet 3.1 NIO
- REST

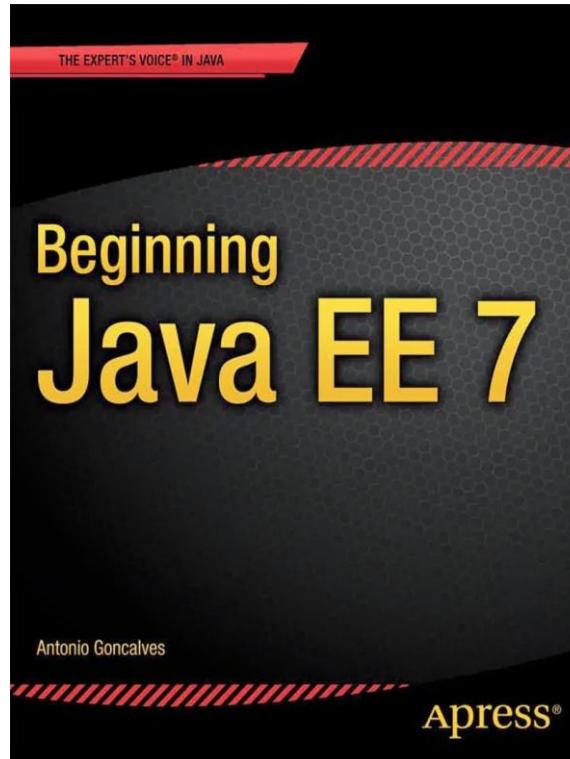
- Batch
- Concurrency
- Simplified JMS

□ De nouvelles API et des API qui évoluent :



- TP 9 : Objectif : Écrire un filtre de servlet qui affiche le temps passé à l'exécution de chaque servlet.

❑ Bibliographie



- ▶ Javadoc Java EE packages :
 - ▶ <http://docs.oracle.com/javaee/7/api/index.html>

□ Deux petites choses avant de partir ☺ :

→ Evaluation du module

- Prenez le temps d'évaluer ce module en remplissant le formulaire :
- <http://goo.gl/forms/Xg6V36uWi6>

→ QCM

- Retournez vos claviers et répondez aux questions du QCM

MERCI !

Fin Cours – JEE