Cursus Java

-

Java EE – Développement Web

Octobre 2016

Emmanuel Fernandez

Copyright

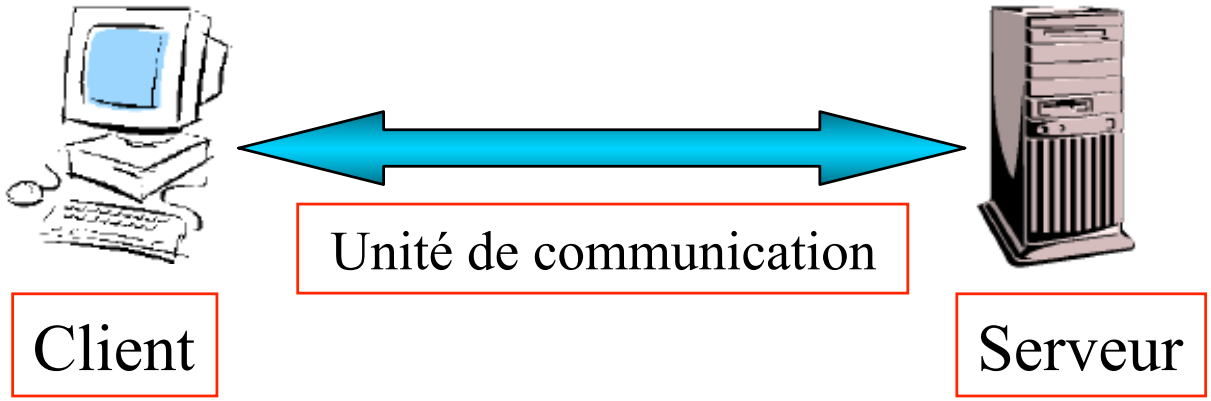
Architectures distribuées et plate-forme Java EE

Les différents modèles d'architecture technique

- Système centralisé
  - Limitation très forte du nombre d'utilisateurs
  - Pas de communication distante
- Architecture multi-niveaux ou multi-tiers
  - 2 tiers
  - 3 tiers

Architecture 2 tiers

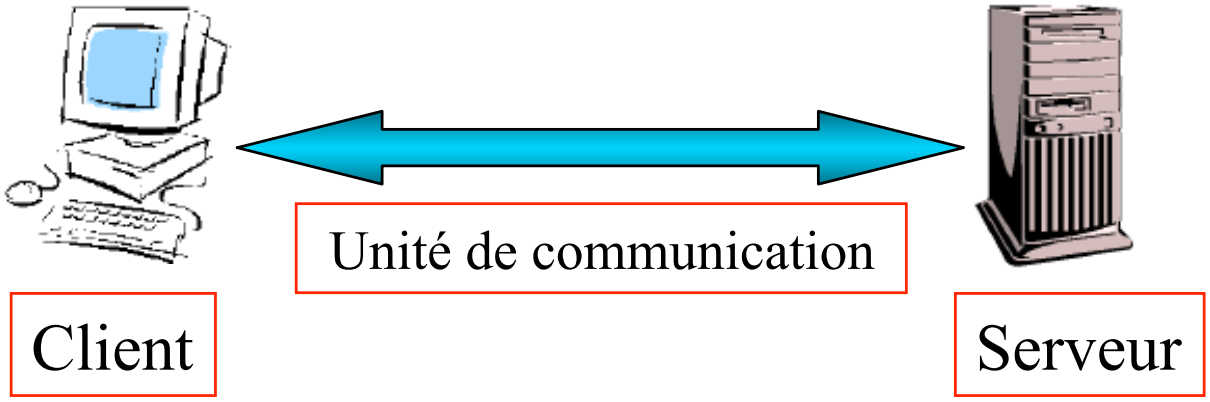
- Client serveur première génération :
  - Le client gère uniquement la couche présentation
  - Le serveur réalise l'ensemble des applications



La couche de présentation est très succincte
Les performances du serveur s'écroulent au delà de quelques utilisateurs

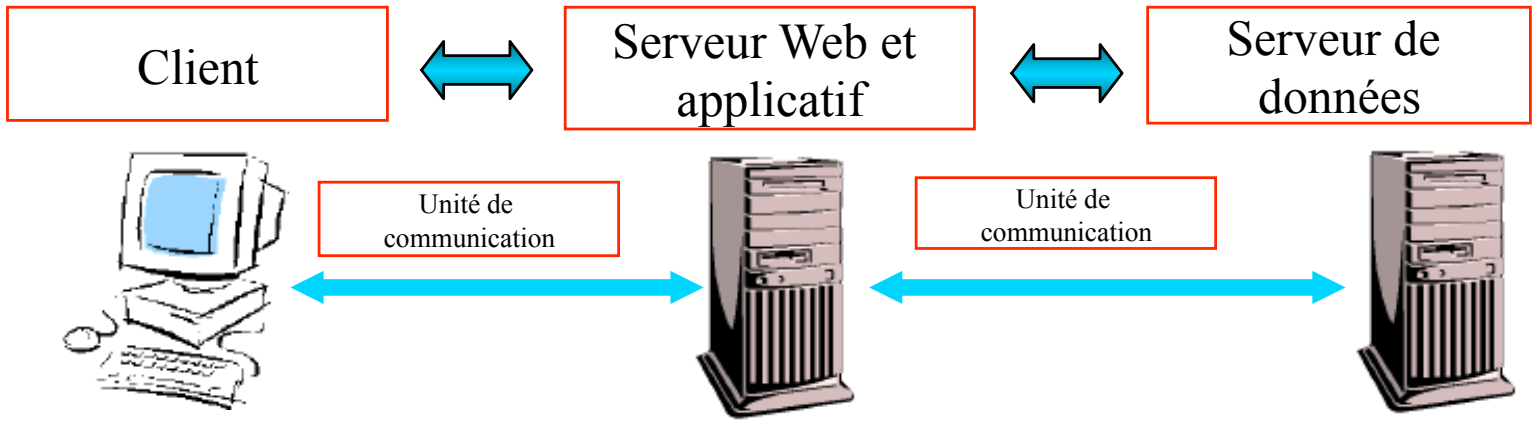
Architecture 2 tiers

- Client serveur deuxième génération :
  - Pour utiliser la puissance des PC, les traitements applicatifs sont déportés sur ces derniers
  - Le serveur gère les accès à la base de données selon les requêtes des clients



Déploiement coûteux et difficile
Trafic réseau important
Difficultés d'administration

Architecture 3 tiers

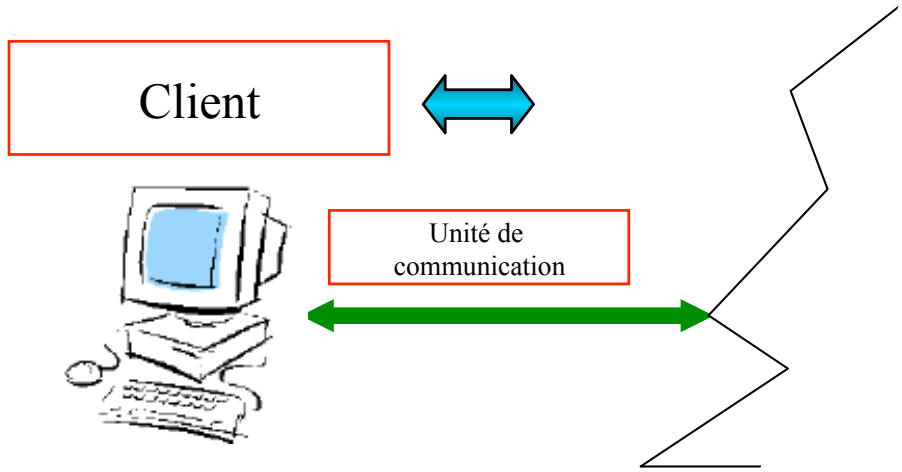


- Réponse aux limitations du 2-tiers
- L'architecture est composée
  - D'un niveau client gérant la présentation
  - D'un niveau agent, ou serveur applicatif, prenant en compte les applications et les objets métiers
  - D'un niveau contenant les serveurs de base de données

Les différents « tiers » d'une architecture 3-tiers

Le client

- Les postes clients sont pourvus
  - D'un navigateur Web (HTML, XHTML)
  - D'un navigateur Web avec des applets ou des contrôles ActiveX, plugs-in...
  - D'une application

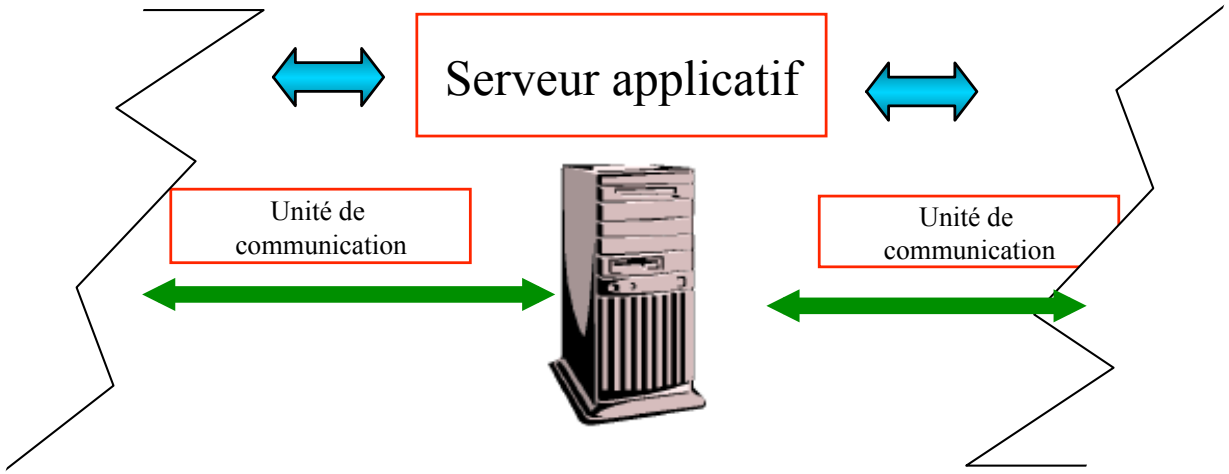


L'agent (1)

- L'agent à pour rôle de :
  - Fournir des services de routage de requêtes
  - Fournir des services de répartition de charge
  - Fournir des services de sécurité
  - Filtrer les requêtes
  - Fournir des services de collecte de données
  - Fournir des services transactionnels (moniteur transactionnel)
  - Tolérance aux pannes

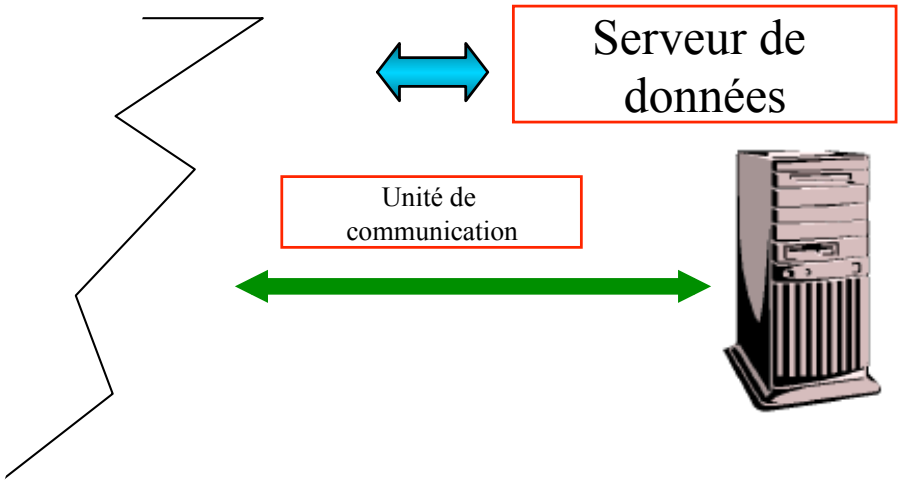
L'agent (2)

- L'agent, ou serveur applicatif
  - Est le point d'entrée du système
  - Gère la complexité du système d'information auquel on veut se connecter



Le serveur de données

- Le serveur de données se concentre sur l'accès aux données:
  - Celles-ci ne sont plus vues directement par le client
  - Le serveur applicatif peut ainsi se connecter sur des bases hétérogènes



Avantages

- Trois niveaux standardisés
- La logique métier se déplace sur le serveur
- Montées en charge mieux gérées
- Administration du système distribué centralisée
- Clients légers, déploiement facile
- Multiplication des types de clients facilitée

Les technologies privilégiées

- Java pour la programmation
- HTML pour les interfaces graphiques
- HTTP pour la diffusion d'informations et de programmes
- XML pour les échanges de données
- IIOP pour la communication inter-objets (associé à CORBA)
- LDAP pour l'authentification

Les technologies de composants côté client

Le client léger

- L'élément essentiel : le navigateur
- Aucune installation sur le poste client
- Ne plus se limiter aux seuls PCs
- Les solutions actuelles
  - HTML
  - ActiveX
  - Plugs-in
  - Applet Java
  - XML

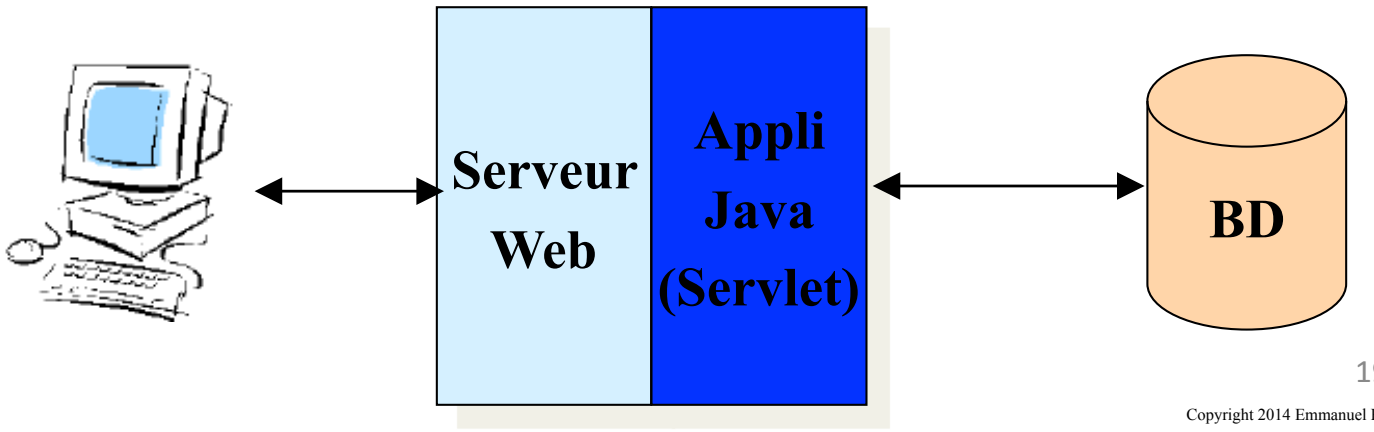
La génération de pages dynamiques

Le principe

- Page HTML diffusée au client suite à une requête HTTP
- Le contenu est calculé en fonction du contexte :
  - Paramètres de la requête HTTP
  - État de la session de l'utilisateur
- Deux types d'approches :
  - Orientée traitement
  - Orientée présentation

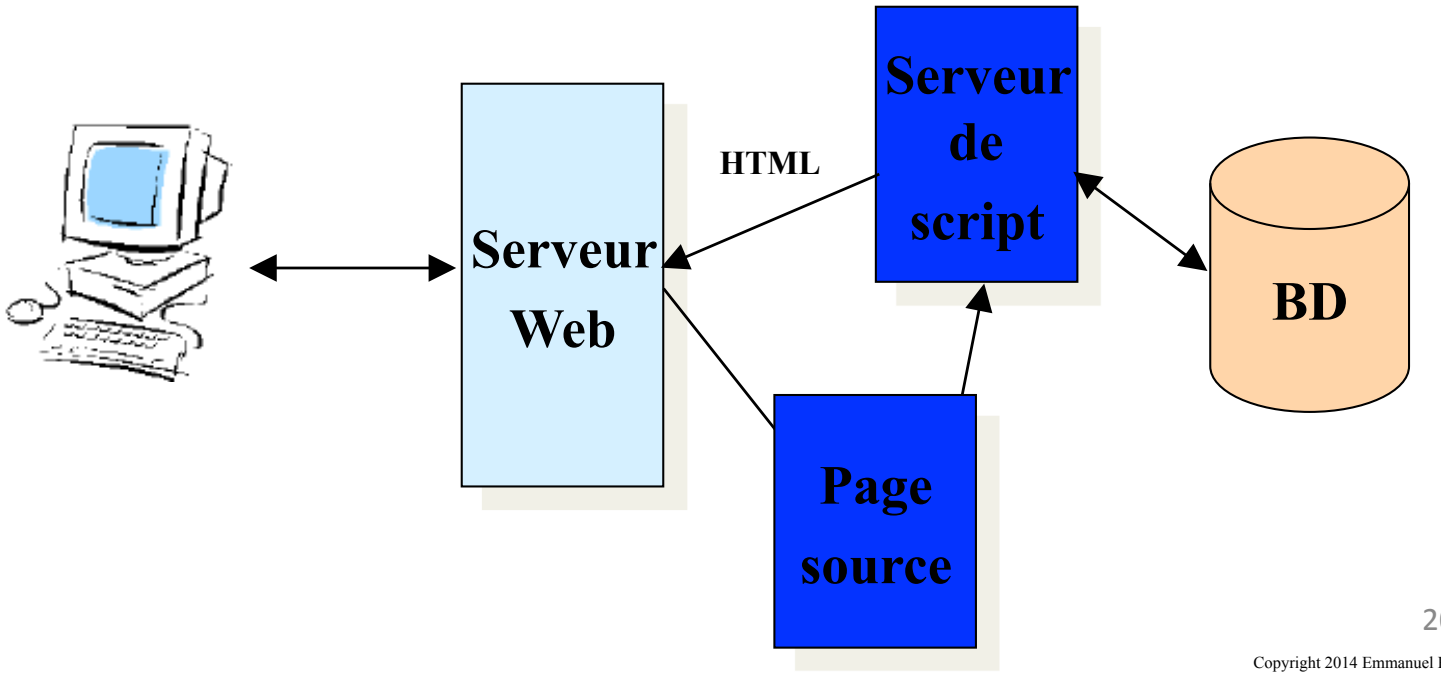
Les technologies orientées traitement

- On définit des traitements côté serveur
- Produisent un résultat (page HTML)
- Plusieurs technologies présentes :
  - CGI
  - NSAPI / ISAPI
- En Java : les servlets



Les technologies orientées présentation

- Pages écrites dans le langage cible
- Contiennent des appels en langage de scripts
- Technologies courantes :
  - ASP, ColdFusion, PHP…
- En Java : JSP



Les technologies orientées présentation

- Avantages
  - Elles sont orientés présentation
  - Elles sont simples à mettre en œuvre
  - Elles possèdent des fonctionnalités de connexion aux bases de données
- Inconvénients
  - Code rapidement incompréhensible
  - Définir l'interface avec les graphistes en général non informaticiens
  - Maintenance difficile

Les serveurs d'applications

Problématique

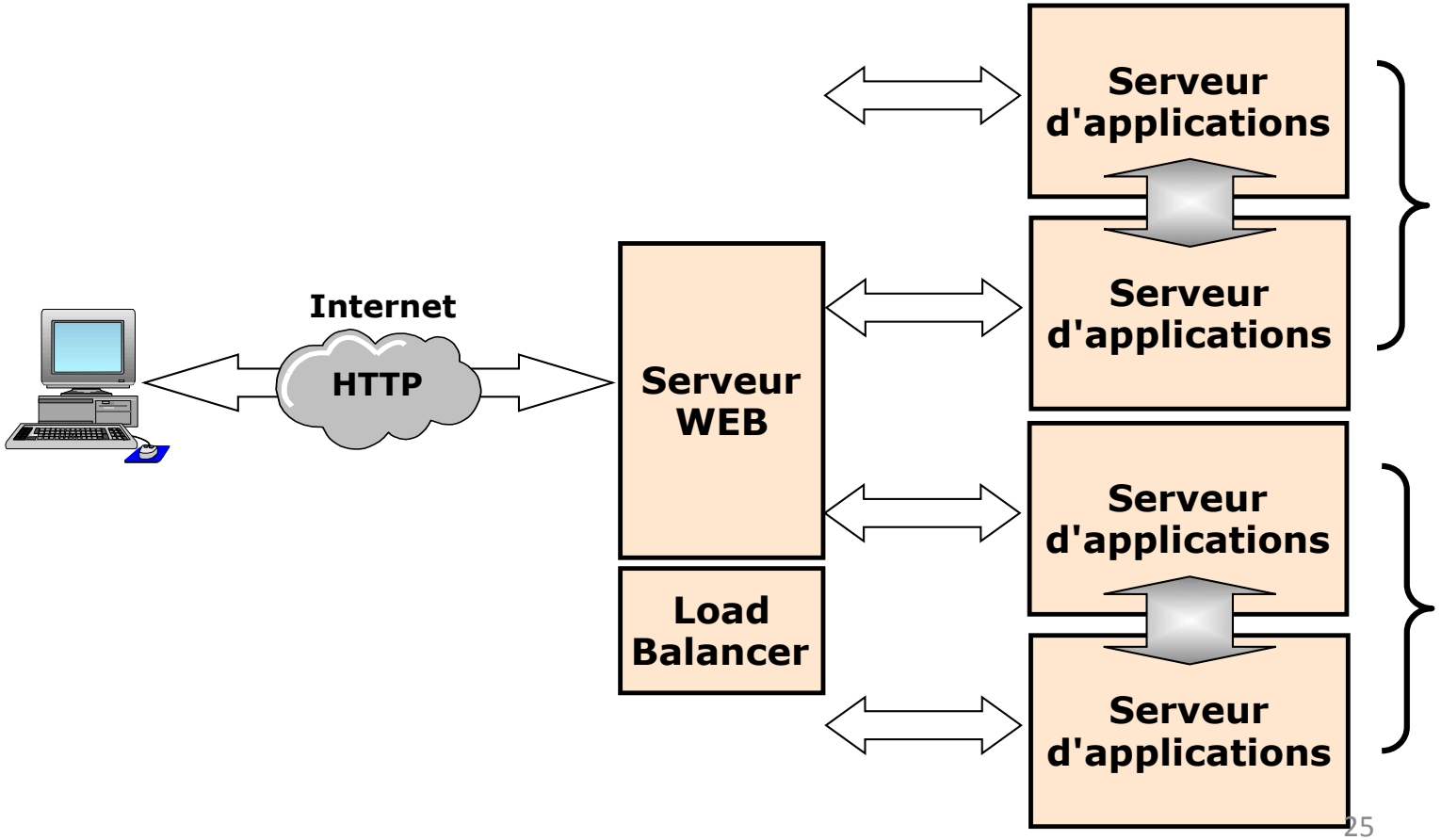
- Les problèmes rencontrés dans la mise en place d'architectures réparties :
  - Gestion du cycle de vie des objets
  - Montée en charge
  - Transactions
- Pour éviter de « réinventer la roue »
  - Les serveurs d'applications proposent des *frameworks* qui prennent en charge ces types de problèmes

Les serveurs d'applications

- Intégration de services :
  - de nommage des composants,
  - d'authentification des utilisateurs,
  - de déploiement,
  - d'équilibrage de charge,
  - de tolérance aux pannes.
- Concentration sur la programmation des traitements applicatifs

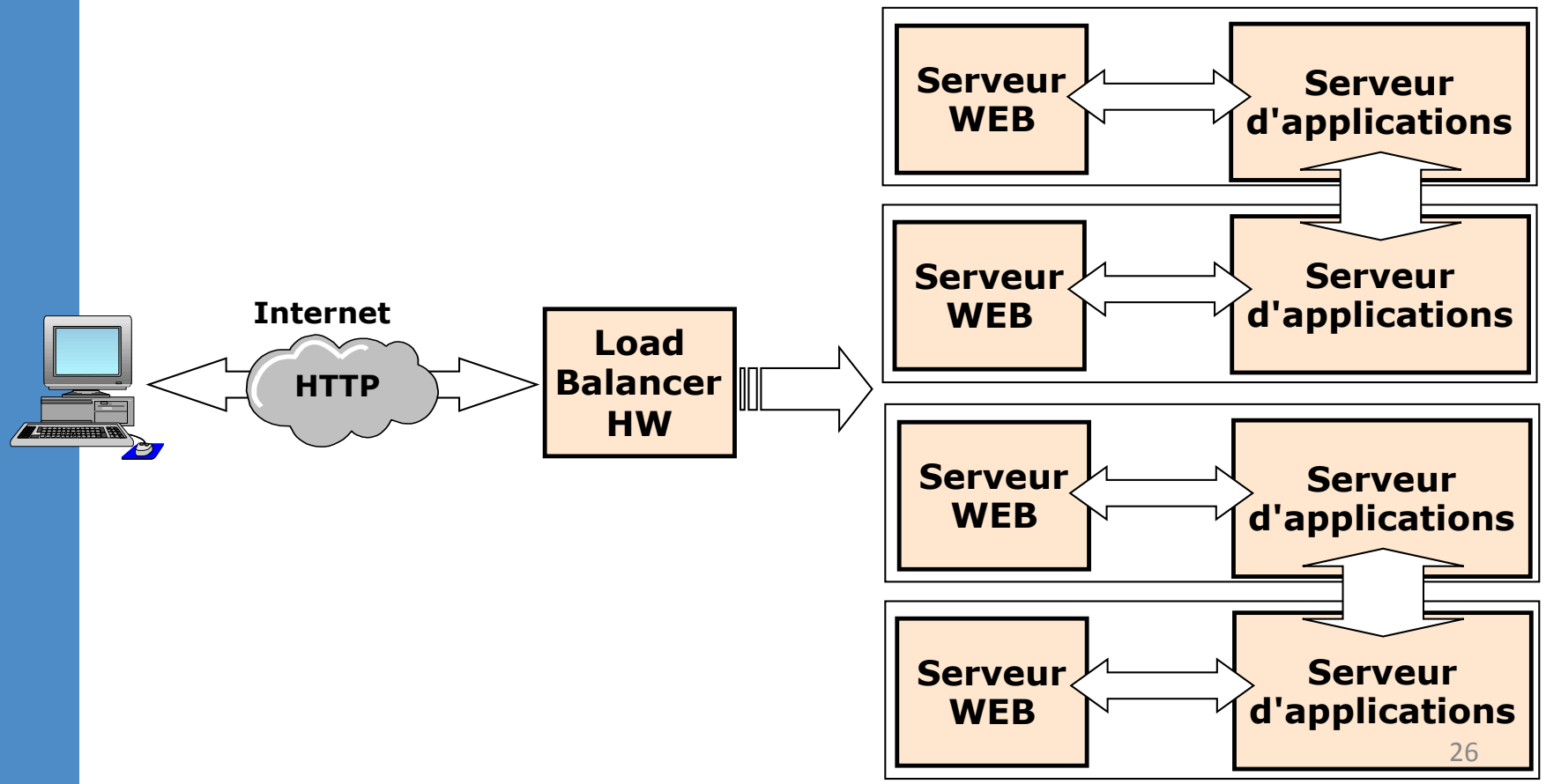
Equilibrage logiciel

- Montée en charge
- Tolérance aux pannes

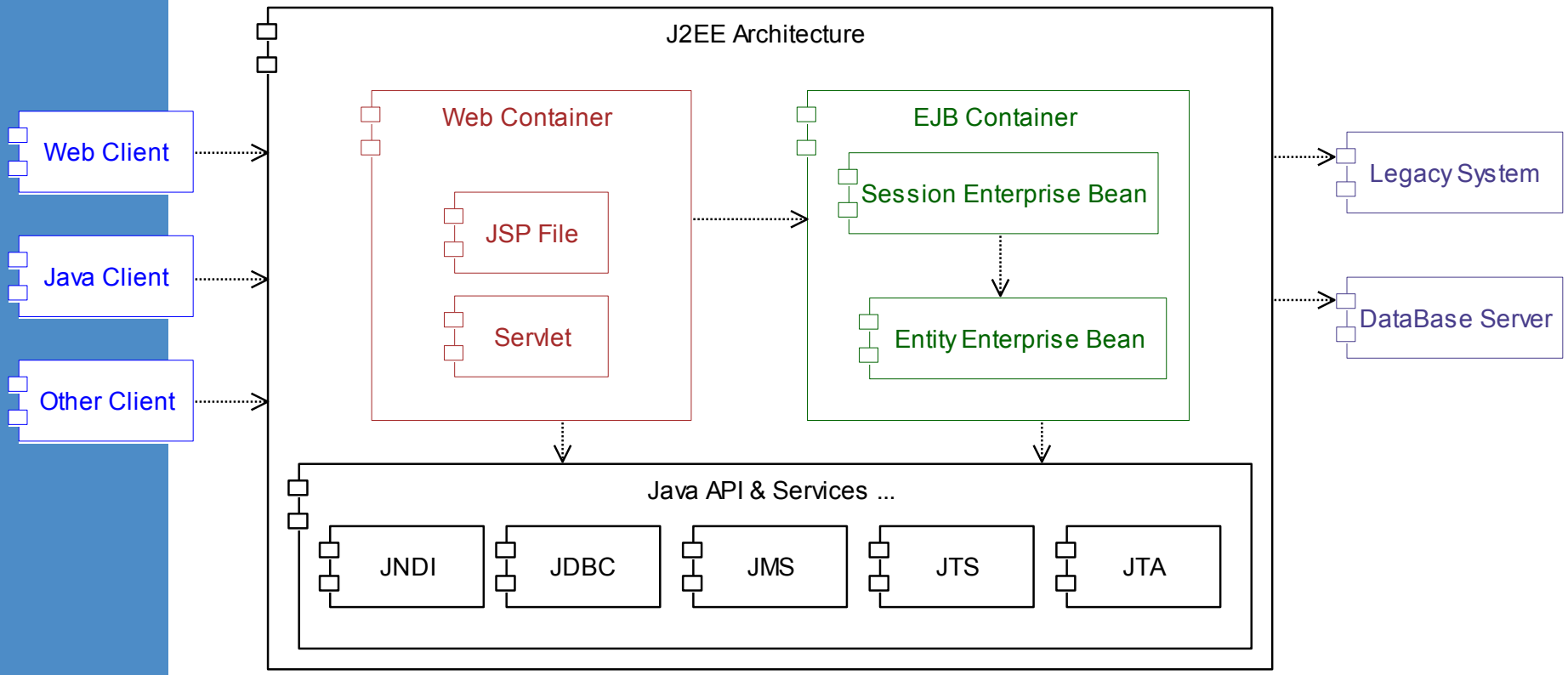


Equilibrage physique

- Montée en charge
- Tolérance aux pannes



L'architecture Java EE



- L'architecture Java EE est :
  - distribuée, transactionnelle,
  - portable, multi-tiers,
  - sécurisée.

Les Servlets

Les Servlets

- Présentation
- Cycle de vie
- Principe de fonctionnement
- Échange d'informations
- Gestion d'une session

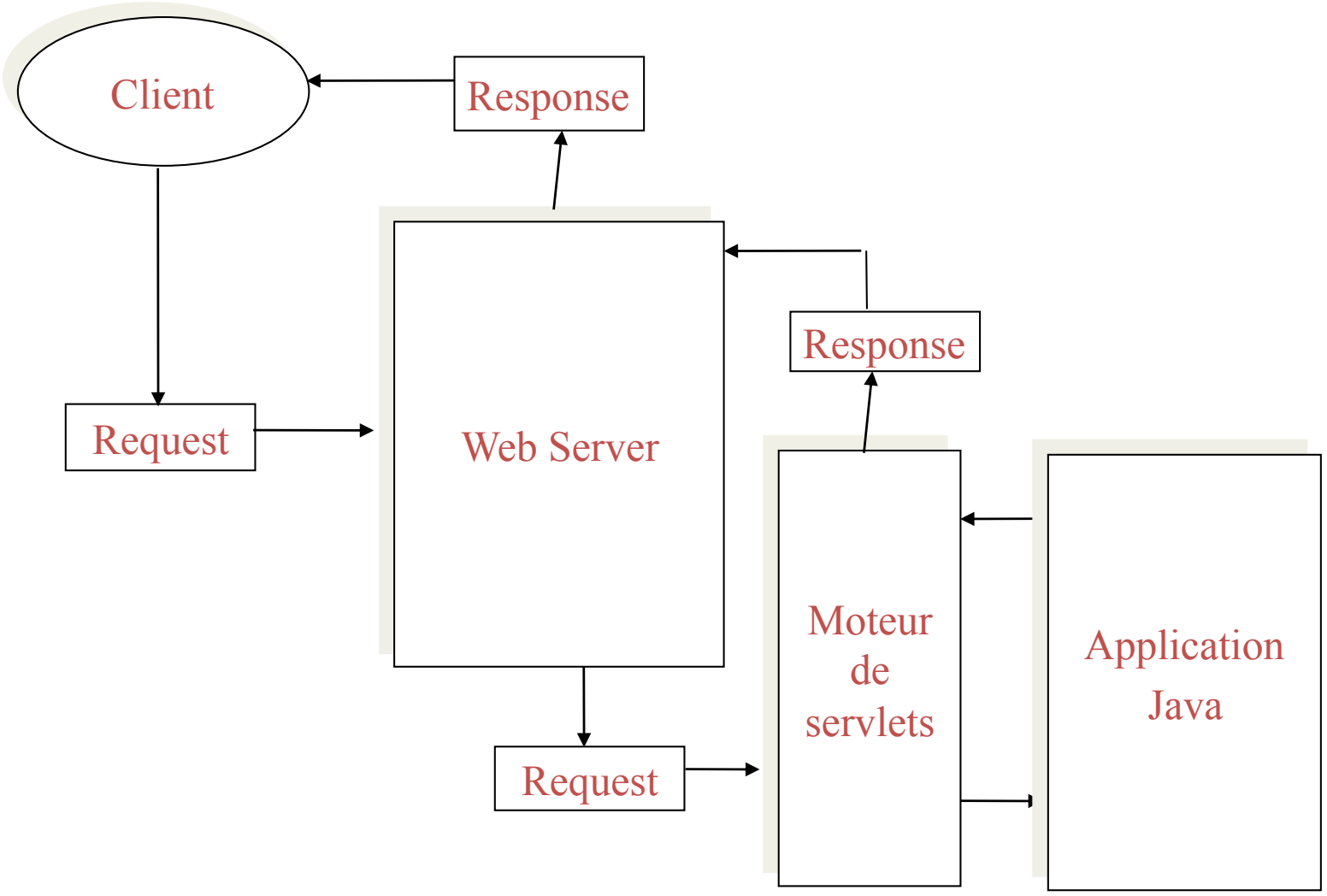
Présentation des Servlets

- Exécution de traitements sur un serveur suite à une requête d'un client
- Prévu pour tout type de protocole avec une implémentation pour HTTP
- Génération de pages dynamiques orientée sur une logique de traitement
- Concurrent aux technologies :
  - CGI
  - NSAPI / ISAPI

Le principe

- Processus Java qui s'exécute côté serveur suite à une requête d'un client
- Exploite les paramètres de la requête
- Repose sur les classes de l'API *servlet* de l'environnement Java EE
- S'exécute au sein d'un moteur de servlets couplé au serveur Web

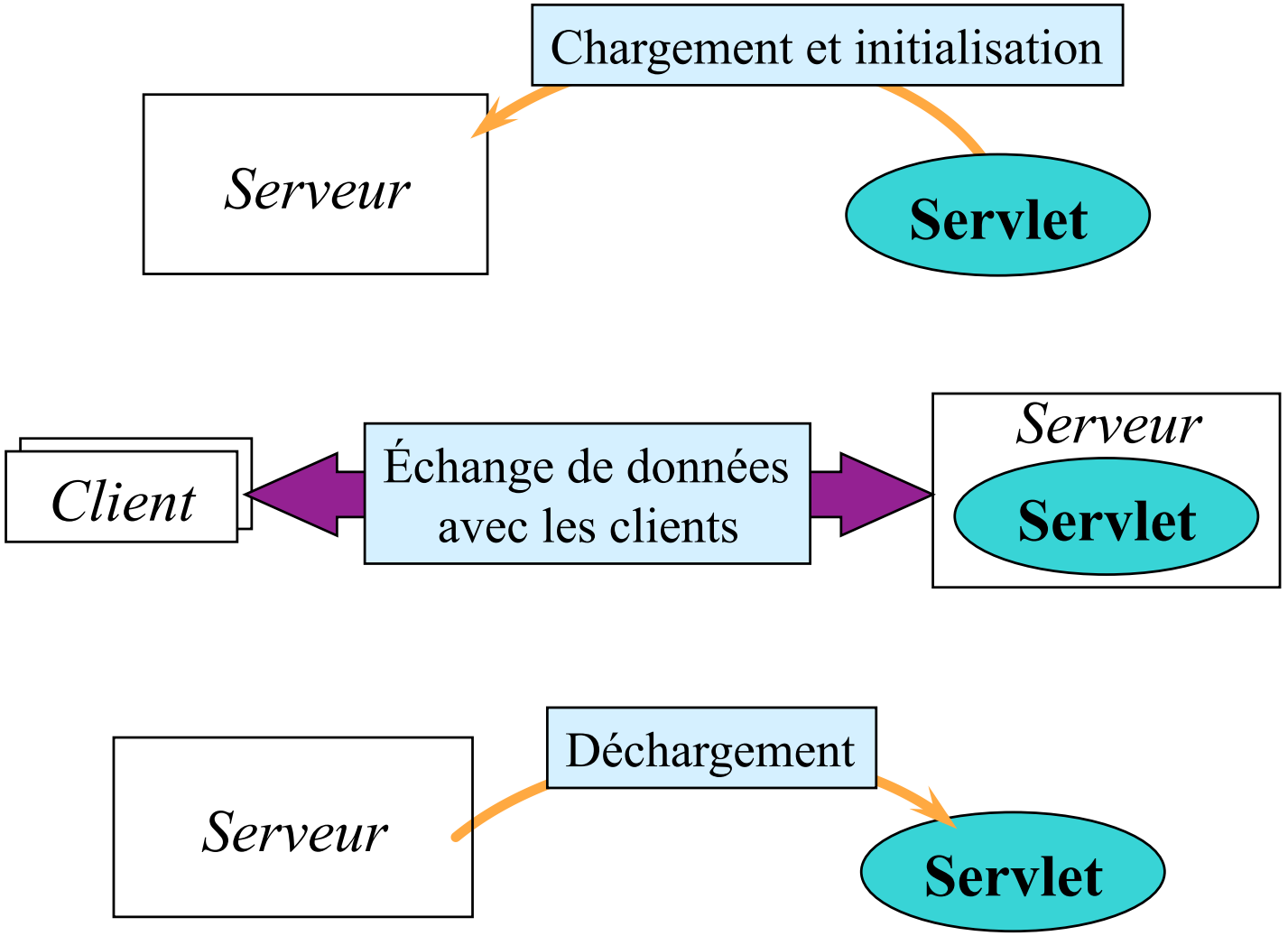
Fonctionnement



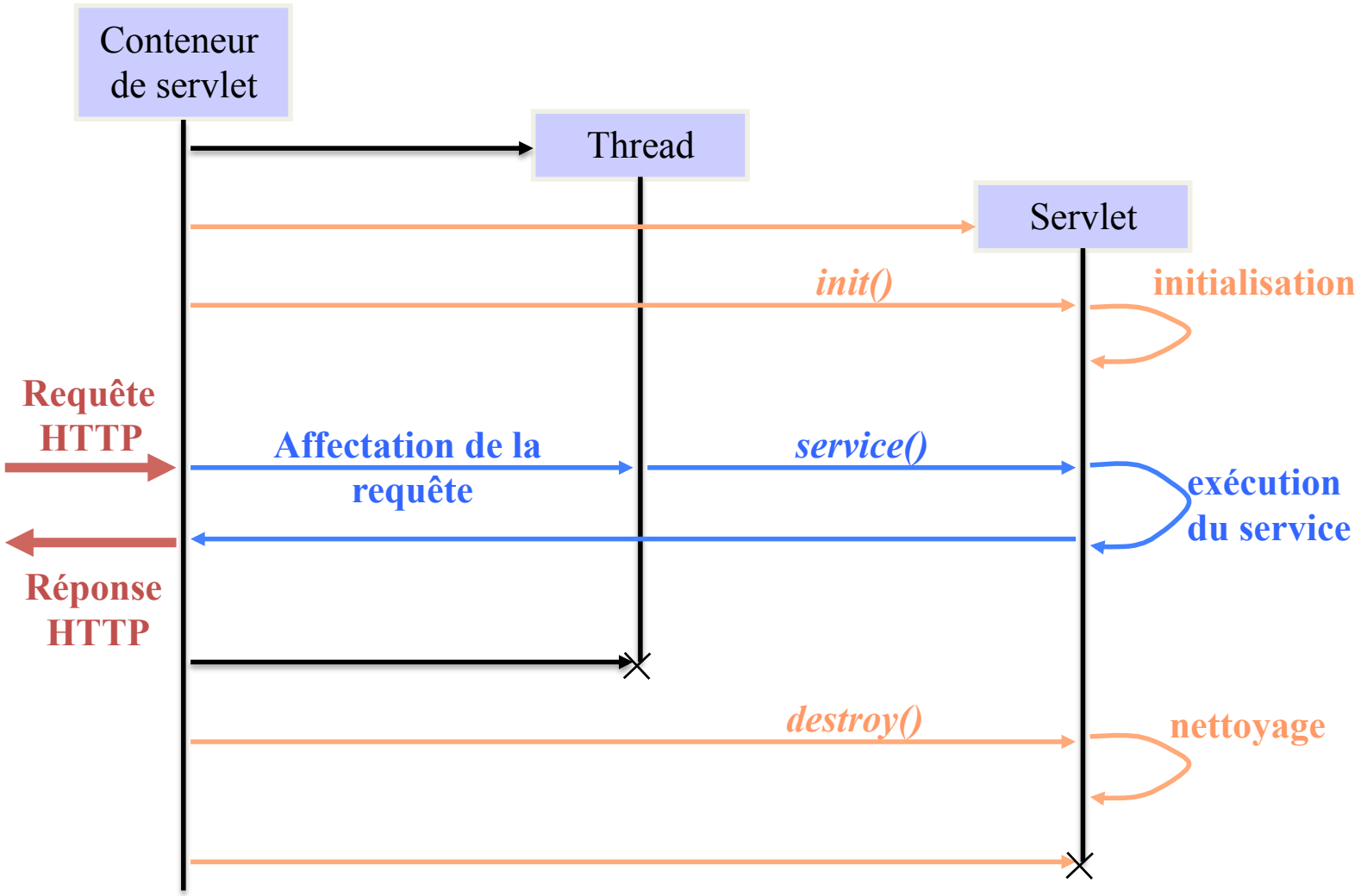
Moteurs de servlets

- Les moteurs peuvent être directement intégrés aux serveurs Web ou ajoutés comme modules par la suite
- Parmi tous les serveurs supportant les servlets, on peut noter :
  - Tomcat, Resin, JRun, Weblogic Server, IBM Websphere, Jonas, JBoss, WildFly, GlassFish, ...

Cycle de vie d'une servlet



Conteneur de servlet



Mise en oeuvre (1)

- Le package *javax.servlet* contient des interfaces et des classes pour l'écriture de *servlets*
- Une *servlet* hérite de la classe *HTTPServlet* qui elle-même implémente l'interface *Servlet*
- La *servlet* peut redéfinir des méthodes de *HTTPServlet* qui correspondent à des requêtes HTTP classiques : *service(), doGet(), doPost()* ...

Mise en oeuvre (2)

- Les méthodes de type *doGet()* et *doPost()* reçoivent en paramètre deux objets :
  - Un objet *HttpServletRequest* qui encapsule la requête du client (renseignée automatiquement par le moteur)
  - Un objet *HttpServletResponse* qui encapsule la réponse qui sera retournée au client (à renseigner dans le code de la servlet)

```
public class ExempleServlet  extends HttpServlet {

        /**
         * Réponse à une requête HTTP GET : retourne une simple page HTML
         */
        public void doGet (  HttpServletRequest request,
                             HttpServletResponse response  )
        throws ServletException, IOException
        {
           PrintWriter   out;
           String          titre =  "Exemple simple de document généré par une servlet";

           // Renseigne la réponse sur le type du document retourné
           response.setContentType( "text/html" );

           // Récupère un flux sur la réponse afin d'y écrire la page
           out = response.getWriter();

           // Ecriture de la chaîne correspondant à la page HTML retournée
           out.println("<HTML><HEAD><TITLE>");
           out.println( titre );
           out.println("</TITLE></HEAD><BODY>");
           out.println("<H1>" + titre + "</H1>");
           out.println("<P>Cette page provient d'une servlet.");
           out.println("</BODY></HTML>");
           out.close();
        }

}
```

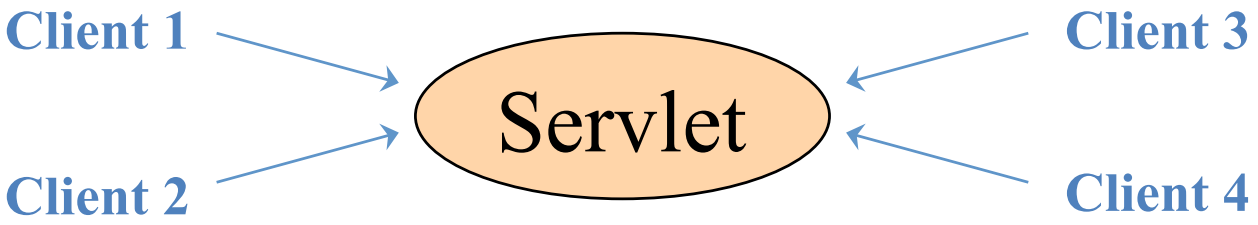
Surcharge des méthodes

void init()	GenericServlet	Appelée au chargement de la servlet : initialisation, lecture des fichiers de conf…
void service()	GenericServlet	Appelée à chaque invocation de la servlet
void destroy()	GenericServlet	Appelée au déchargement de la servlet : destruction de tous les objets créés

Partage des données

- Chaque servlet n'est instanciée qu'une seule fois, il y a donc persistance des données entre deux appels de la servlet
- La valeur d'un attribut de la classe dérivant d'*HttpServlet* dépend des autres invocations (multi-threads)

  => Il ne faut pas utiliser les attributs pour stocker des informations !



Echange d'informations

- Trois objets peuvent servir d'intermédiaires pour échanger des informations entre composants Web :
  - La requête
  - La session
  - L' application
- Ils se comportent comme des hashtables et disposent des méthodes :
  - setAttribute()
  - getAttribute()

La requête

- La requête est passée en paramètre de la méthode *service()* (ou *doGet()*, *doPost()*…)
- On y récupère l'URL d'appel, les paramètres HTTP et la session utilisateur
  - *getRequestURL()*
  - *getParameter(String name)*
  - *getSession()*
- Elle peut être transmise à une autre servlet lors d'une délégation de traitement (*forward*)
- On y place des objets servant à la génération de la réponse

L'application

- L'application est un objet de la classe ServletContext récupéré à partir de la servlet

```
ServletContext application = this.getServletContext();
```

- L'application est partagée entre tous les objets Web et tous les clients d'une même application Web
- On y place les objets servant pour toute l'application (ressources…)

La session

- La session peut être récupérée à partir de la requête

```
HttpSession session = request.getSession();
```

- La session est liée à un client. Elle existe tant qu'elle n'a pas été détruite volontairement ou par un time out
- On y place les objets servant durant toute la connexion du client (typiquement : un caddie)

Les cookies

- Permettent le stockage d'informations au niveau du client
- Utilisation de la classe *javax.servlet.http.Cookie*
  - Création

```
Cookie monCookie = new Cookie("nom", "valeur");
response.addCookie(monCookie);
```

  - Récupération

```
Cookie[] mesCookies = request.getCookies();
```

URL Rewriting

- Gestion des sessions pour des clients n'acceptant pas les cookies
- Méthodes
  - *HttpServletResponse.encodeURL()*
  - *HttpServletResponse.encodeRedirectURL()*

```
out.println(
    "<tr>" + ... + "<a href=\"" +
    response.encodeURL(monLien) +
    "\"> + books[i].getTitle() + "</a> );
```

- Utilise le session ID pour retrouver la session de l'utilisateur

Déléguer des traitements

- Possibilité de rediriger la requête HTTP sur une autre URL

```
response.sendRedirect(<location>);
```

- Possibilité de déléguer une partie de la réponse à une autre ressource :
  - Par inclusion : *include()*
  - Par délégation complète : *forward()*

```
ServletContext ctx = getServletContext();
RequestDispatcher rd;
rd = ctx.getRequestDispatcher(<RelativeURLOfJSP>);
rd.include( <RequestObject>, <ResponseObject> );
rd.forward( <RequestObject>, <ResponseObject> );
```

Les Java Server Pages

Les Java Server Pages

- Dans ce chapitre, nous allons :
  - Présenter l'architecture des JSP
  - Comprendre la syntaxe JSP
  - Développer un exemple de JSP

Présentation des JSP

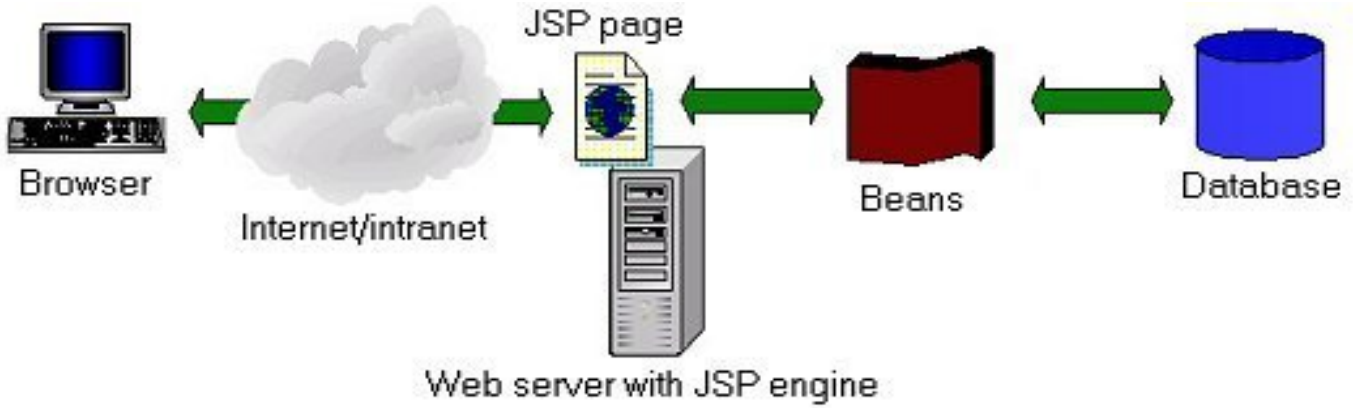
- Génération de pages dynamiques orientée sur une logique de présentation
- Ce sont des scripts interprétés côté serveur contenant du code Java
- Insèrent de la logique de traitement dans des pages de présentation (HTML, WML, XML)
- Concurrent aux technologies :
  - ASP
  - PHP

Exemple de fichier JSP

- Une page JSP est écrite dans le langage cible de publication (HTML, …)
- Elle contient des appels à du code Java

```
<HTML>
    <HEAD>
        <TITLE> Exemple de page JSP </TITLE>
    </HEAD>
    <BODY>
   Un nombre aléatoire est :
   <%= Math.random() %>
    </BODY>
</HTML>
```

Fonctionnement



- Les pages JSP sont exécutées par un moteur de JSP couplé au serveur Web
- Elles accèdent à du code Java qui se charge de la logique et de l'accès aux données

Compilation des JSP

- Modèle d'exécution :
  - Chaque page JSP est convertie en *servlet*
  - Lorsque la page JSP a déjà été compilée et exécutée une fois, la *servlet* reste en mémoire, elle est directement exécutée

Format des Tags JSP

- L'écriture de pages JSP se base sur l'utilisation de balises (tags) prédéfinies
- Il existe deux formats pour les tags JSP

JSP (*Short-hand*)	Directives
	Déclaration
	Expression
	Scriptlet
XML	Actions

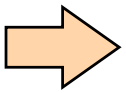
Expressions

- Permettent d'intégrer des valeurs directement dans le code HTML

```
<%= value %>
```

- Utilisées pour afficher une chaîne de caractères, typiquement une valeur de retour
- Exemple :

```
<P>Prix de l'article :
<%= monArticle.getPrice() %></P>
```



Equivalent à l'utilisation du *out.println()* dans la méthode *service()* d'une servlet

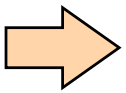
Scriptlets

- Permettent d'insérer du code Java

```
<% code java %>
```

- Elles ont accès aux variables et aux composants déclarés dans la page
- Exemple :

```
<P>
    <% if (true) { %>
        Ce texte est affiché
    <% } else { %>
        Celui-ci n'est jamais pris en compte
    <% } %>
</P>
```



Equivalent à du code simple dans la méthode *service()* d'une servlet

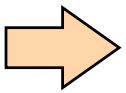
Déclarations

- Utilisées pour déclarer des variables ou des méthodes dans une page

```
<%! declaration %>
```

- Fortement déconseillé !
- Exemple :

```
<%! String var1 =“x”;
    int count = 0;
    private void increment() {
     count++;
    }
%>
```



Equivalent à une déclaration d'attribut ou d'une méthode dans la classe de la servlet

Directives JSP

- Permettent de configurer la JSP

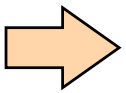
```
<%@ directive { attribute = "value" } %>
```

- Sont souvent placées au début du fichier
- Exemples :
  - Inclusion statique de fichiers :

```
<%@ include file="headers/copyright.html" %>
```

  - Import de classes :

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



Equivalent à l'écriture des imports de packages et de classes au début du code de la servlet

Directive pour les exceptions

- Possibilité d'intercepter toutes les exceptions non catchées
- Redirection de tous les cas d'erreur vers une page unique
- Utilisation de la directive *page* :

```
<%@ page isErrorPage="false" errorPage="errorHandler.jsp" %>
```

- Au début de la page *errorHandler.jsp*, il faut ajouter :

```
<%@ page isErrorPage="true"%>
```

Liste des directives

- <%@ page language="**java**" %>
- <%@ page extends="*package.class*" %>
- <%@ page import= "{ *package.class* | *package.**}, ..." %>
- <%@ page session="**true**|false" %>
- <%@ page buffer="none|**8kb**|*size*kb" %>
- <%@ page autoFlush="**true**|false" %>
- <%@ page isThreadSafe="**true**|false" %>
- <%@ page info="*text*" %>
- <%@ page errorPage="*relativeURL*" %>
- <%@ page isErrorPage="true|**false**" %>
- <%@ page contentType="*mimeType* [;charset=*characterSet*]" | **"text/html ; charset=ISO-8859-1"** %>

Actions

- Elles effectuent une tâche prédéfinie
- Aucun codage en Java n'est nécessaire
- Elles se caractérisent par une balise standard XML spécifique :
  - *useBean*
  - *setProperty/getProperty*
  - *include*
  - *forward*

Utilisation de Beans

- **useBean** : Instancie un Bean sur le serveur

```
<jsp:useBean id="notreBean" scope="application"
             class="com.example.OurBean" />
```

- **getProperty** : Récupère la propriété d'un Bean

```
<jsp:getProperty name="notreBean" property="unAttr"/>
```

- **setProperty** : Met à jour la propriété d'un Bean (utilisé pour les formulaires)

```
<jsp:setProperty name="notreBean"
                 property="unAttr" value="uneValeur"/>
```

```
<jsp:setProperty name="notreBean"
                 property="unAttr" param="unParam"/>
```

```
<jsp:setProperty name="notreBean" property="*"/>
```

Actions - Scope

- L'action **useBean** permet de récupérer ou de créer un bean suivant une portée
- 4 portées différentes :
  - *page* : objet accessible seulement dans la page où il a été créé
  - *request* : objet accessible seulement pour les pages qui sont liées à une même requête
  - *session* : objet accessible par un seul client sur toutes les pages JSP pendant la même session
  - *application* : objet accessible par tous les clients pendant toute la durée de l'application

Utilisation d'autres ressources

- **include** : Permet d'insérer dynamiquement le contenu d'une autre ressource (servlet, JSP, HTML)

```
<jsp:include page="shoppingcart.jsp" />
```

- **forward** : délègue le traitement de la requête à une autre ressource

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

- le *sendRedirect* reste possible, mais ce n'est pas une action

Variables implicites

9 variables implicites peuvent être utilisées dans les pages JSP :

Variable (scope - classe)	**Méthodes courantes**	**Commentaires**
request *(request - Sous-classe de javax.servlet. ServletRequest)*	getParameter getParameterNames getParameterValues	Répresente la requête HTTP
response *(response – Sous-classe de javax.servlet. ServletResponse)*		Représente la réponse HTTP Peu utilisé dans les JSP

Variables implicites

pageContext *(Page - javax.servlet.jsp.Page Context)*	findAttribute getAttribute getAttributesScope getAttributeNamesInScope	
session *(Session - javax.servlet.http. HttpSession)*	getId getAttribute putAttribute getAttributeNames	Représente la session utilisateur
application *(application - javax.servlet. ServletContext)*	getMimeType getRealPath	Représente la Web application

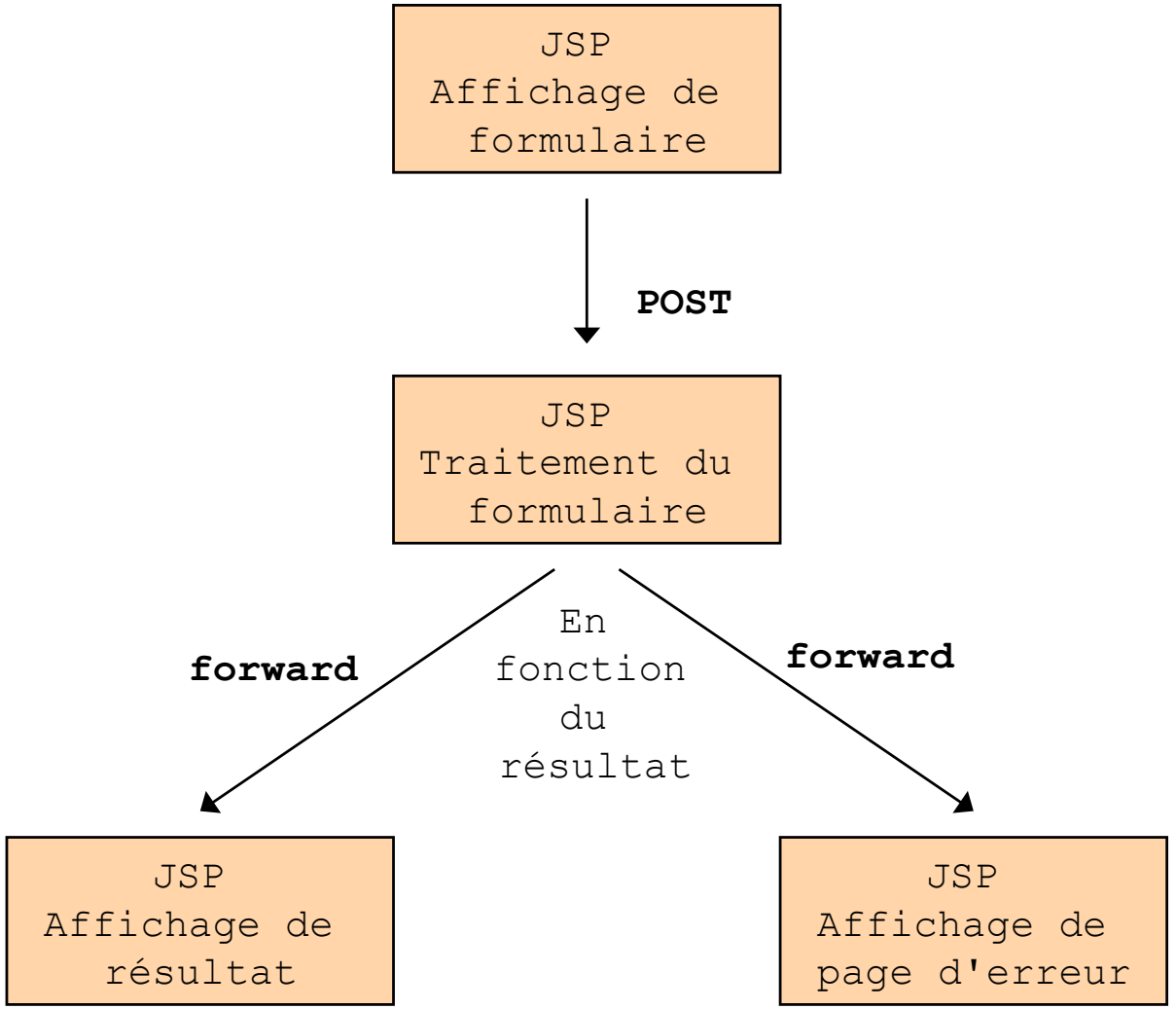
Variables implicites

out *(Page - javax.servlet.jsp.JspWriter)*	clear, clearBuffer, flush, getBufferSize, getRemaining	Représente la réponse http sous la forme d'un flux de caractères
config *(Session - javax.servlet. ServletConfig)*	getInitParameter, getInitParameterNames	
page *(page - java.lang.Object)*	getMimeType, getRealPath	Peu utilisé
exception *(page - java.lang.Throwable)*	getMessage, getLocalizedMessage, printStackTrace	

Ecrire des JSPs (1)

- JSPs de présentation :
  - Générer dans un premier temps le code HTML de la partie présentation
  - Enregistrer le fichier avec l'extension *.jsp*
  - Programmer le code Java sur le serveur qui gère la partie fonctionnelle
  - Insérer les appels JSP pour la communication avec ce code Java
- Certaines JSPs servent uniquement à réaliser des traitements, l'écriture de la réponse est alors déléguée

Ecrire des JSPs (2)



Un exemple complet

JavaBean

```
package com.example ;
public class OurBean {
    private String value;
    public void setValue (String s ) {
        value = s ;
    }
    public String getValue ( ) {
        return value ;
    }
}
```

JSP

```
<HTML>
  <BODY>
    <H2> cette page teste notre bean </H2>
    <jsp:useBean id= "notreBean"
     class= "com.example.OurBean" scope= "session" />
    <% notreBean.setValue("présent"); %>
    <P>
      notre bean est <%= notreBean.getValue(); %>
    </P>
  </BODY>
</HTML>
```

Servlets / JSP
Utilisation avancée

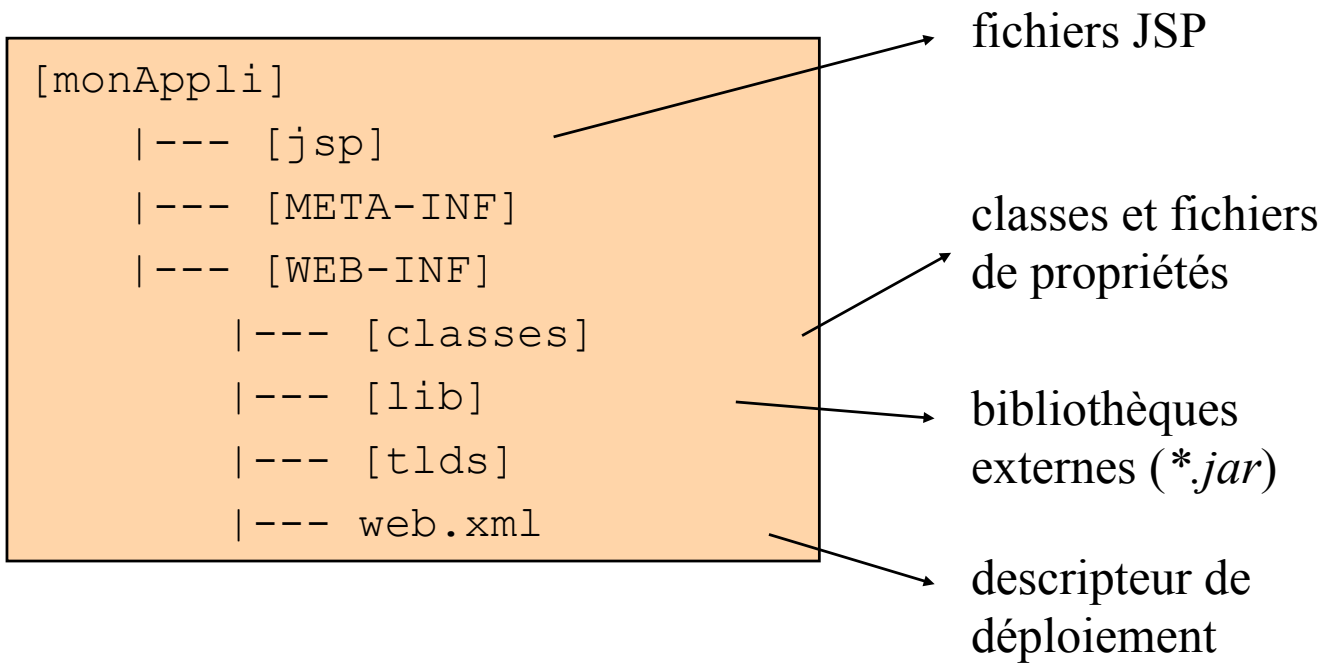
Servlets / JSP : Utilisation avancée

- Dans ce chapitre nous allons :
  - Présenter le déploiement des applications Web
  - Comparer les servlets et les JSPs
  - Comprendre le concept des filtres
  - Comprendre les *tags-library*

Déploiement des Web Applications

Déploiement d'applications

- Une Web Application est une application à base de servlets/JSPs
- Une arborescence précise à respecter



Le descripteur de déploiement

- Fichier de configuration des comportements dans le fichier %WEBAPP %\WEB-INF**web.xml**
- Validé par la DTD

```
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD
  Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-
  app_2_3.dtd">
```

Le descripteur de déploiement

- Les informations données par le fichier de configuration sont :
  - les paramètres d'initialisation du contexte
  - les définitions des servlets et des JSPs
  - les alias des servlets et des JSPs
  - la configuration de la session
  - les mappings des types MIME
  - la liste des fichiers de bienvenue
  - les pages d'erreur
  - les contraintes de sécurité

Le fichier web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <context-param>
    <param-name>…</param-name>
    <param-value>…</param-value>
  <context-param>
  <servlet> … </servlet>
  <servlet-mapping> … </servlet-mapping>
  <session-config>
    <session-timeout>…</session-timeout>
  </session-config>
  <mime-mapping> … </mime-mapping>
  <welcome-file-list> … </welcome-file-list>
  <error-page>…</error-page>
</web-app>
```

Déclaration des servlets/JSPs

- Une définition par composant `<servlet>`
  - Une référence unique `<servlet-name>`
  - La classe ou la page d'implémentation `<servlet-class>` ou `<jsp-file>`
- Une déclaration du mapping `<servlet-mapping>`
  - Une référence unique `<servlet-name>`
  - Un alias (ou URI) unique utilisé pour appeler ce composant `<url-pattern>`

Exemple

```
<servlet>
    <servlet-name>SnoopServlet</servlet-name>
    <servlet-class>snoopy.SnoopServlet</servlet-class>
    <init-param>
        <param-name>foo</param-name>
        <param-value>bar</param-value>
    </init-param>
</servlet>
<servlet>
    <servlet-name>SnoopJSP</servlet-name>
    <jsp-file>/jsp/Snoop.jsp</jsp-file>
</servlet>
<servlet-mapping>
    <servlet-name>SnoopServlet</servlet-name>
    <url-pattern>/snoop</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name> SnoopJSP </servlet-name>
    <url-pattern>/snoop2</url-pattern>
</servlet-mapping>
```

Construction d'un .war

- Le WAR (Web Archive) est un format standardisé de déploiement
- Un unique fichier WAR par Web Application pour :
  - l'archivage des fichiers de l'application
  - la compression de ces fichiers

```
cd monAppli
jar –cvf monArchive.war *
```

- Un WAR peut être distribué sur n'importe quel serveur Java EE

Utilisation d'un .war

- Format connu par tous les moteurs de servlets et de JSP
- Même configuration que pour une application qui ne serait pas archivée

Web application classique	Le contexte pointe sur la racine de l'arborescence
Web application déployée	Le contexte pointe sur le fichier .war

Servlet / JSP : comparaison

Servlets et JSP : les avantages

- Indépendance de la plate forme
- Modèle de sécurité hérité du serveur Web
- Large support de nombreux serveurs Web
- Peuvent utiliser toutes les APIs Java courantes de Java SE et Java EE
- Les servlets peuvent fonctionner sur différents protocoles

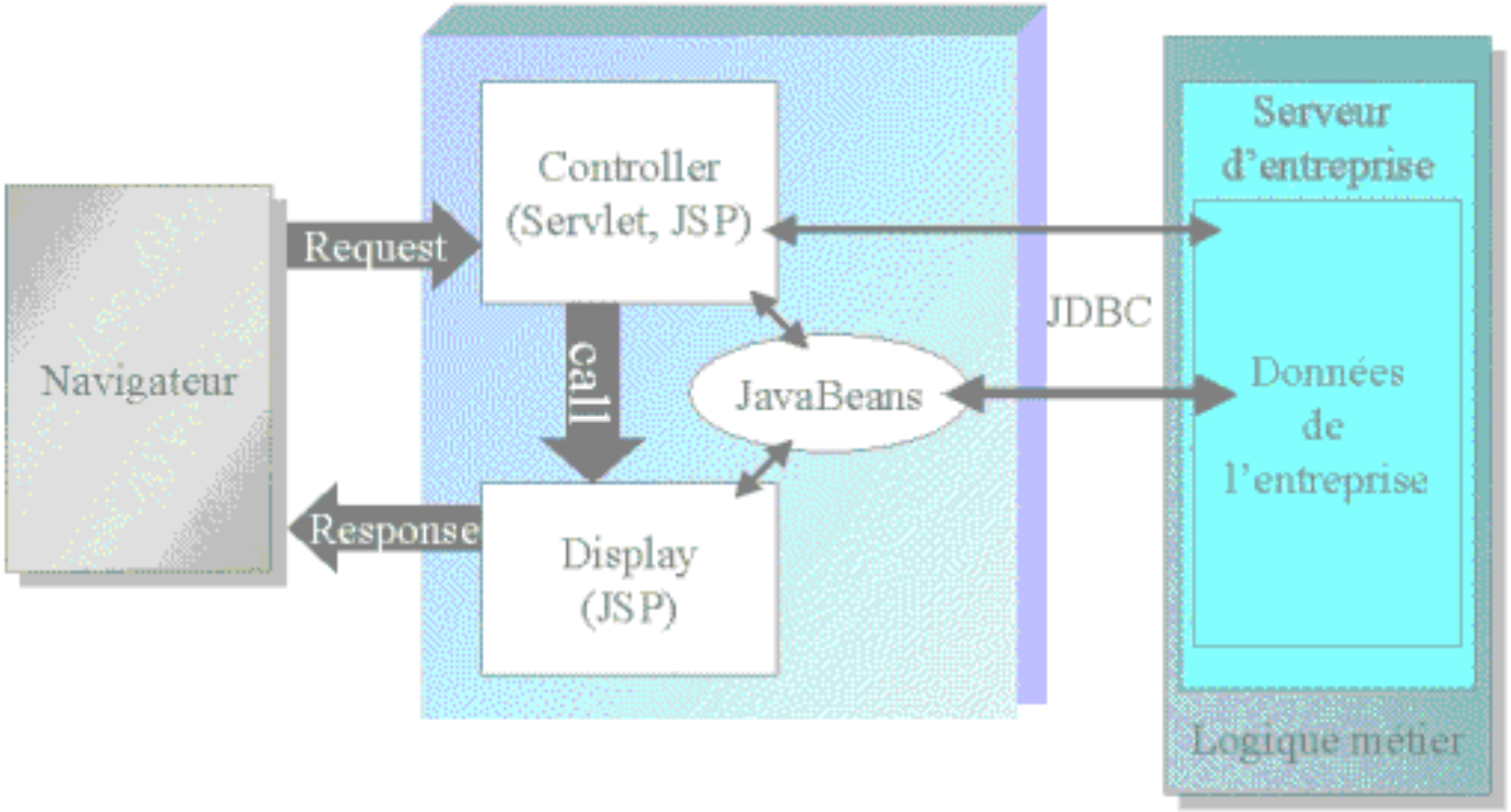
Comparaison

- Servlets
  - Idéal pour la réalisation de traitements de calculs, de règles de gestion
  - Écriture de la présentation des résultats peu adaptée
  - Requiert une bonne expertise pour la conception des pages
- JSP
  - Idéal pour la conception de pages Web
  - L'écriture des scripts Java peut devenir vite "lourde"
  - Requiert une connaissance Java de la part du designer

La solution

- Proposer une architecture séparant au maximum traitement et présentation
- Utiliser la complémentarité des deux technologies
  - Les servlets se chargent d'effectuer la logique de traitement
  - Les pages JSP se chargent de présenter le résultats des calculs

Modèle d'interaction Servlets/JSP



Source SUN

Délégation de traitement

- Utiliser la classe *RequestDispatcher* pour transmettre une demande à une page JSP :
  - Récupérer la classe *RequestDispatcher* côté *servlet* :

```
ServletContext ctx = getServletContext();
RequestDispatcher rd ;
rd = ctx.getRequestDispatcher(<RelativeURLOfJSP>);
```

  - Déléguer tout le traitement :

```
rd.forward( <RequestObject>,<ResponseObject> );
```

  - Inclure une partie de traitement déléguée :

```
rd.include( <RequestObject>,<ResponseObject> );
```

Les filtres

Les filtres

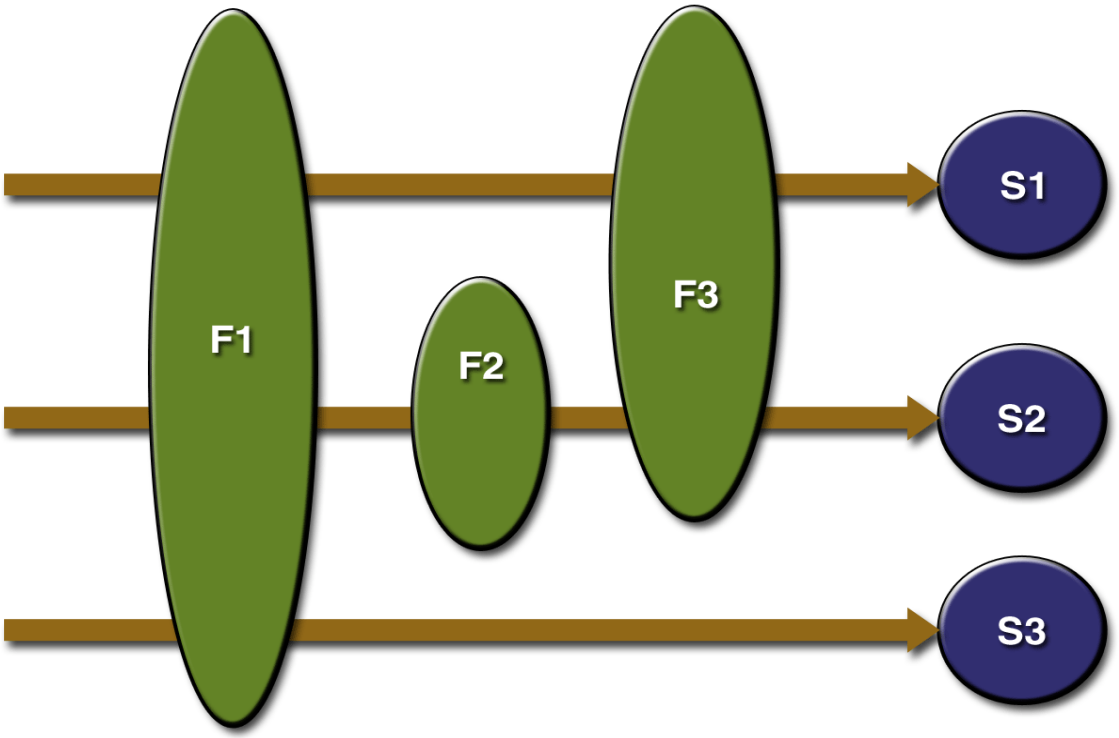
- Composants introduits avec la spécification 2.3 des servlets
- Interceptent les requêtes et les réponses
- Code modulaire et réutilisable :
  - Les filtres sont des classes indépendantes des servlets.
  - « brancher - débrancher » un filtre = modifier la configuration du mapping des servlets

Les filtres : applications

- Exemples d'applications:
  - Authentification : bloquer les requêtes basées sur l'identité de l'utilisateur.
  - Logging and auditing : Tracer le trafic des utilisateurs sur une application web.
  - Image conversion-Scaling maps, etc.
  - Data compression : alléger les downloads.
  - Localization : Cibler les requêtes et les réponses
  - XSL/T transformations of XML content : Envoyer les réponses vers différents types de clients.

Chaînes de filtres

- On peut mapper un filtre sur une ou plusieurs servlets
- On peut mapper plus d'un filtre sur une même servlet



Source : tutorial J2EE

Les filtres : API

- Package *javax.servlet*
- Interface *Filter*
  - cœur du filtre
  - méthodes *doFilter() , init(), destroy()*
- *FilterChain*
  - permet de chaîner les filtres
- *FilterConfig*
  - contient les données d'initialisation

Écrire un filtre

```
public final class HitCounterFilter implements Filter {
  private FilterConfig filterConfig = null;
  public void init(FilterConfig filterConfig) throws ServletException {
    this.filterConfig = filterConfig;
  }
  public void destroy() {
    this.filterConfig = null;
  }
  public void doFilter(ServletRequest request,  ServletResponse response,
      FilterChain chain) throws IOException,ServletException {
    if (filterConfig == null)  return;
    StringWriter sw =  new StringWriter();
    PrintWriter writer = new PrintWriter(sw);
    Counter counter =    (Counter)filterConfig.getServletContext().getAttribute("hitCounter");
    writer.println("The number of hits is:" + counter.incCounter());
    writer.flush();
    filterConfig.getServletContext().log(sw.getBuffer().toString());
    chain.doFilter(request, response);
  }
}
```

Configuration (1)

- Fichier *web.xml*
- Élément <filter>
  - Déclarer le filtre : Nom du filtre ; classe ; paramètres d'initialisation
- Élément <filter-mapping>
  - Mapper le filtre à la (aux) servlet(s) grâce à l'URL pattern

Configuration (2)

```
<filter>
   <filter-name>Compression Filter</filter-name>
   <filter-class>CompressionFilter</filter-class>
   <init-param>
      <param-name>compressionThreshold</param-name>
      <param-value>10</param-value>
 </init-param>
</filter>
```

```
<filter-mapping>
   <filter-name>Compression Filter</filter-name>
   <servlet-name>CompressionTest</servlet-name>
</filter-mapping>
<servlet>
   <servlet-name>CompressionTest</servlet-name>
   <servlet-class>CompressionTest</servlet-class>
</servlet>
<servlet-mapping>
   <servlet-name>CompressionTest</servlet-name>
   <url-pattern>/CompressionTest</url-pattern>
</servlet-mapping>
```

Configuration : exemple

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems,
Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web- app_2_3.dtd">
<web-app>
  <filter>
    <filter-name>XSLTFilter</filter-name>
    <filter-class>XSLTFilter</filter-class>
  </filter>

  <filter>
    <filter-name>HitCounterFilter</filter-name>
    <filter-class>HitCounterFilter</filter-class>
  </filter>

  <filter-mapping>
    <filter-name>HitCounterFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>

  <filter-mapping>
    <filter-name>XSLTFilter</filter-name>
    <servlet-name>
        FilteredFileServlet
    </servlet-name>
  </filter-mapping>

  <servlet>
    <servlet-name>
      FilteredFileServlet
    </servlet-name>
    <servlet-class>FileServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>
      FilteredFileServlet
    </servlet-name>
    <url-pattern>/ffs</url-pattern>
  </servlet-mapping>
</web-app>
```

Configuration par annotations

@WebFilter

- Attributs disponibles:
  - urlPatterns
  - initParams
  - filterName
  - servletNames
  - dispatcherTypes

Configuration par annotations

```java
@WebServlet(name="firstFilter" ,urlPatterns = "/session.do")
public class TestFilter extends HttpServlet {
```

```java
@WebFilter(servletNames = "firstFilter")
public class FirstFilter implements Filter{

    public void destroy() { }

    public void doFilter(ServletRequest request, ServletResponse response,
            FilterChain chain) throws IOException, ServletException {
    }
    public void init(FilterConfig config) throws ServletException {
    }
}
```

remplace

```xml
<servlet>
    <servlet-class>com.test.session.TestFilter</servlet-class>
    <servlet-name>firstFilter</servlet-name>
</servlet>
<servlet-mapping>
    <servlet-name>firstFilter</servlet-name>
    <url-pattern>/session.do</url-pattern>
</servlet-mapping>
<filter>
    <filter-name>firstFilter</filter-name>
    <filter-class>com.test.filter.FirstFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>firstFilter</filter-name>
    <servlet-name>firstFilter</servlet-name>
</filter-mapping>
```

Configuration par annotations

```
@WebServlet(urlPatterns = "/result.do")
public class ResultatFilter extends HttpServlet {
```

```
@WebFilter("/result.do")
public class SecondFilter implements Filter{
```

```
@WebFilter( urlPatterns = "/result.do",
            initParams = @WebInitParam(
                    name="param",
                    value="Valeur du param"))

public class ThirdFilter implements Filter{
    private FilterConfig config;
    public void destroy() { }
    public void doFilter(ServletRequest request, ServletResponse response,
            FilterChain chain) throws IOException, ServletException {
        System.out.println("Parametre du troisieme filtre : " +
                           config.getInitParameter("param"));
        chain.doFilter(request, response);
    }
    public void init(FilterConfig config) throws ServletException {
        this.config = config;
    }
}
```

Configuration par annotations

```
<servlet>
    <servlet-class>com.test.session.ResultatFilter</servlet-class>
    <servlet-name>resultFilter</servlet-name>
</servlet>
<servlet-mapping>
    <servlet-name>resultFilter</servlet-name>
    <url-pattern>/result.do</url-pattern>
</servlet-mapping>

<filter>
    <filter-name>secondFilter</filter-name>
    <filter-class>com.test.filter.SecondFilter</filter-class>
</filter>
<filter>
    <filter-name>thirdFilter</filter-name>
    <filter-class>com.test.filter.ThirdFilter</filter-class>
    <init-param>
        <param-name>param</param-name>
        <param-value>Valeur du param</param-value>
    </init-param>
</filter>

<filter-mapping>
    <filter-name>secondFilter</filter-name>
    <url-pattern>/result.do</url-pattern>
</filter-mapping>
<filter-mapping>
    <filter-name>thirdFilter</filter-name>
    <url-pattern>/result.do</url-pattern>
</filter-mapping>
```

Les listeners

Les listeners

- Les listeners sont des objets dont les méthodes sont invoquées en fonction du cycle de vie d'un servlet

- A la création et à la destruction d'un contexte (une appli)
  - javax.servlet.ServletContextListener
- Quand on modifie les attributs du contexte
  - javax.servlet.ServletContextAttributeListener
- A la création, la suppression, le timeout d'une session
  - javax.servlet.HttpSessionListener
- A la création, modification, suppression d'un attribut de session
  - javax.servlet.HttpSessionAttributeListener

Les listeners

- Classe Servlet

```
public class MaServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;


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

        String attrR = (String) request.getAttribute("attrR");
        if (attrR == null)
            request.setAttribute("attrR", "debut requete ");
        else
            request.setAttribute("attrR", attrR + "x");

        String attrS = (String) request.getSession().getAttribute("attrS");
        if (attrS == null)
            request.getSession().setAttribute("attrS", "debut session ");
        else
            request.getSession().setAttribute("attrS", attrS + "x");

        String attrC = (String) getServletContext().getAttribute("attrC");
        if (attrC == null)
            request.getServletContext().setAttribute("attrC", "debut context ");
        else
            request.getServletContext().setAttribute("attrC", attrC + "x");

    }
}
```

Les listeners

- Classe Listener (1/2)

```
package com.formation;

import javax.servlet.ServletContextAttributeEvent;
import javax.servlet.ServletContextAttributeListener;
import javax.servlet.ServletRequestAttributeEvent;
import javax.servlet.ServletRequestAttributeListener;
import javax.servlet.http.HttpSessionAttributeListener;
import javax.servlet.http.HttpSessionBindingEvent;

public class Listen implements HttpSessionAttributeListener,
        ServletRequestAttributeListener, ServletContextAttributeListener {

    public void attributeRemoved(HttpSessionBindingEvent arg0) {
        System.out.println("attributeRemoved HttpSessionBindingEvent : "
                + arg0.getName());
    }

    public void attributeReplaced(HttpSessionBindingEvent arg0) {
        System.out.println("attributeReplaced HttpSessionBindingEvent : "
                + arg0.getName() + " "
                + arg0.getSession().getAttribute(arg0.getName()));
    }

    public void attributeAdded(HttpSessionBindingEvent arg0) {
        System.out.println("attributeAdded HttpSessionBindingEvent : "
                + arg0.getName() + " "
                + arg0.getSession().getAttribute(arg0.getName()));
    }

    public void attributeAdded(ServletRequestAttributeEvent arg0) {
        System.out.println("attributeAdded ServletRequestAttributeEvent : "
                + arg0.getName() + " "
                + arg0.getServletRequest().getAttribute(arg0.getName()));
    }
```

Les listeners

- Classe Listener (2/2)

```
    public void attributeRemoved(ServletRequestAttributeEvent arg0) {
        System.out.println("attributeRemoved ServletRequestAttributeEvent : "
                + arg0.getName());
    }

    public void attributeReplaced(ServletRequestAttributeEvent arg0) {
        System.out.println("attributeReplaced ServletRequestAttributeEvent : "
                + arg0.getName() + " "
                + arg0.getServletRequest().getAttribute(arg0.getName()));
    }

    public void attributeAdded(ServletContextAttributeEvent arg0) {
        System.out.println("attributeAdded ServletContextAttributeEvent : "
                + arg0.getName() + " "
                + arg0.getServletContext().getAttribute(arg0.getName()));

    }

    public void attributeRemoved(ServletContextAttributeEvent arg0) {
        System.out.println("attributeRemoved ServletContextAttributeEvent : "
                + arg0.getName());

    }

    public void attributeReplaced(ServletContextAttributeEvent arg0) {
        System.out.println("attributeReplaced ServletContextAttributeEvent : "
                + arg0.getName() + " "
                + arg0.getServletContext().getAttribute(arg0.getName()));

    }

}
```

Les listeners

- web.xml

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

    <servlet>
        <description></description>
        <display-name>MaServlet</display-name>
        <servlet-name>MaServlet</servlet-name>
        <servlet-class>com.formation.MaServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>MaServlet</servlet-name>
        <url-pattern>/MaServlet</url-pattern>
    </servlet-mapping>

    <listener>
        <listener-class>com.formation.Listen</listener-class>
    </listener>
    <listener>
        <listener-class>com.formation.SessionCounterListen</listener-class>
    </listener>
</web-app>
```

Les listeners

- Résultats

```
attributeAdded ServletRequestAttributeEvent : org.apache.catalina.ASYNC_SUPPORTED false
attributeAdded ServletRequestAttributeEvent : attrR debut requete
attributeAdded ServletContextAttributeEvent : compteur 1
attributeAdded HttpSessionBindingEvent : attrS debut session
attributeAdded ServletContextAttributeEvent : attrC debut context
attributeAdded ServletRequestAttributeEvent : org.apache.catalina.ASYNC_SUPPORTED false
attributeAdded ServletRequestAttributeEvent : attrR debut requete
attributeReplaced HttpSessionBindingEvent : attrS debut session x
attributeReplaced ServletContextAttributeEvent : attrC debut context x
attributeAdded ServletRequestAttributeEvent : org.apache.catalina.ASYNC_SUPPORTED false
attributeAdded ServletRequestAttributeEvent : attrR debut requete
attributeReplaced HttpSessionBindingEvent : attrS debut session xx
attributeReplaced ServletContextAttributeEvent : attrC debut context xx
```

Les listeners

- Autre exemple : Compteur de sessions :

```
package com.formation;

import javax.servlet.http.HttpSessionEvent;
import javax.servlet.http.HttpSessionListener;

public class SessionCounterListen implements HttpSessionListener {

    public void sessionCreated(HttpSessionEvent arg0)  {
        int cpt = 0;
        try {
            cpt = (Integer) arg0.getSession().getServletContext().getAttribute("compteur");
        } catch (Exception e) {
            cpt = 0;
        }

        arg0.getSession().getServletContext().setAttribute("compteur", (cpt + 1));
        System.out.println("Nb sessions = " + (cpt + 1));
    }

    public void sessionDestroyed(HttpSessionEvent arg0)  {
        int cpt = 0;
        try {
            cpt = (Integer) arg0.getSession().getServletContext().getAttribute("compteur");
        } catch (Exception e) {
        }
        arg0.getSession().getServletContext().setAttribute("compteur", cpt - 1);
        System.out.println("Nb sessions = " + (cpt - 1));
    }
}
```

Configuration par annotations

@WebListener

- Attribut disponible:
  - Value (description)

Permet de définir un listener

```
<web-app>
    <listener>
        <listener-class>com.test.listener.MyListener</listener-class>
    </listener>
    [...]
</web-app>
```

Configuration par annotations

```
@WebListener
public class MyListener implements HttpSessionListener, HttpSessionAttributeListener,
                                   ServletContextListener {
    public void attributeAdded(HttpSessionBindingEvent event) {
        System.out.println("Attribut ajouté : " + event.getName());
        System.out.println("Attribut ajouté : " + event.getSession().getAttribute(event.getName()));
    }
    public void attributeRemoved(HttpSessionBindingEvent event) {
        System.out.println("Attribut supprimé : " + event.getName());
    }
    public void attributeReplaced(HttpSessionBindingEvent event) {
        System.out.println("Attribut ajouté : " + event.getName());
        System.out.println("Attribut ajouté : " + event.getSession().getAttribute(event.getName()));
    }
    public void sessionCreated(HttpSessionEvent se) {
        System.out.println("Session créée : " + se.getSession());
    }
    public void sessionDestroyed(HttpSessionEvent se) {
        System.out.println("Session détruite : " + se.getSession());
    }
    public void contextDestroyed(ServletContextEvent sce) {
        System.out.println(sce.getServletContext().getContextPath() + " détruite");
    }
    public void contextInitialized(ServletContextEvent sce) {
        System.out.println(sce.getServletContext().getContextPath() + " activée");
    }
}
```

Configuration par annotations

```
[#|[AutoDeploy] Sélection du fichier [...]\autodeploy\Listener02.war pour l'autodéploiement.|#]
[#|/Listener02 activée|#]
[#|WEB0671: Loading application [Listener02] at [/Listener02]|#]
[#|Listener02 a été déployé en 563 ms.|#]
[#|[AutoDeploy] Déploiement automatique accompli avec succès : [...]\autodeploy\Listener02.war.|#]
[#|Traitement dans la servlet : firstFilter.|#]
[#|Session créée : org.apache.catalina.session.StandardSessionFacade@41b2bb|#]
[#|Attribut ajouté : login|#]
[#|Attribut ajouté : MonLogin01|#]
[#|Attribut ajouté :  password|#]
[#|Attribut ajouté : MonPwd01|#]
[#|Autoundeploying application :Listener02|#]
[#|Session détruite : org.apache.catalina.session.StandardSessionFacade@41b2bb|#]
[#|Attribut supprimé :  password|#]
[#|Attribut supprimé : login|#]
[#|/Listener02 détruite|#]
[#|[AutoDeploy] Annulation du déploiement automatique accompli avec succès : [...]\autodeploy\Listener02.war.|#]
```

Les tags libraries

Les tags libraries

- Extension des balises standards
- Permettent de définir ses propres tags et d'y associer des traitements spécifiques
- Encapsulent des comportements complexes qui se feront sur le serveur
- Apparues avec la version 1.1 des JSP
- Séparent encore davantage les rôles du graphiste et du développeur

Exemple

```
<%@ taglib uri="/ImInfoSample.tld" prefix="sample" %>

<html>
    <head>
        <title>ImInfo Standard Hello World Demo</title>
    </head>
    <body bgcolor="#ffffff">
        <hr />
        <sample:hello name="Ben"/>
        <hr />
    </body>
</html>
```

Composantes

- Pour écrire des tags libraries, il faut 3 éléments :
  - Un fichier JSP normal qui intègre des appels aux tags libraries développées
  - Une classe Java qui a pour but de définir un traitement et d'insérer des éléments (code HTML, …)
  - Un fichier de description XML qui définit la structure des tags et fait le lien entre les appels JSP et la classe Java associée

Le fichier JSP

- Contient des tags HTML et JSP
- Doit déclarer, au début, l'ensemble des tags libraries utilisées

```
<HTML>
<HEAD>
    <%@ taglib uri="iminfo-taglib.tld" prefix="iminfo" %>
    <TITLE>Exemple de Tag Libs</TITLE>
</HEAD>
<BODY>
    <H1><iminfo:example /></H1>
    <P><iminfo:example /></P>
</BODY>
</HTML>
```

La classe Java (TagHandler)

- Doit hériter de la classe *TagSupport*
- Doit définir les méthodes appelées lorsque le fichier JSP est interprété

```
package coreservlets.tags;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
import java.io.*;

public class ExampleTag extends TagSupport {
   public int doStartTag() {
      try {
         JspWriter out = pageContext.getOut();
         out.print("Custom tag example
            (coreservlets.tags.ExampleTag)");
      } catch(IOException ioe) {
         System.out.println("Error in ExampleTag: " + ioe);
      }
      return(SKIP_BODY);
    }
}
```

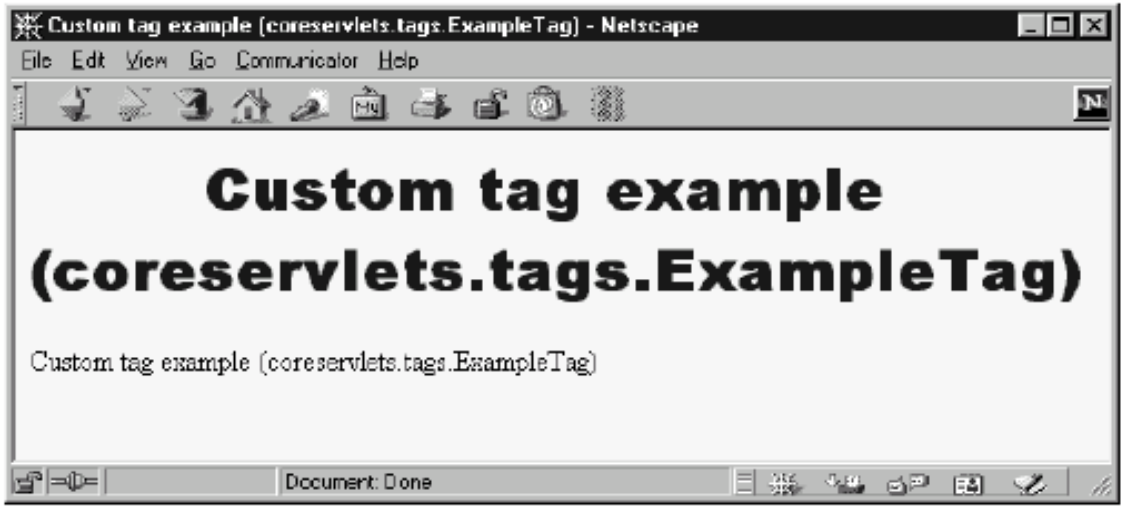
Le fichier de description

- Fait correspondre chaque tag-lib JSP avec la classe Java associée
- A une extension ".tld"

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!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>iminfo</shortname>
    <urn></urn>
    <info> A tag library from Core Servlets and Java Server Pages
    </info>
    <tag>
        <name>example</name>
        <tagclass>coreservlets.tags.ExampleTag</tagclass>
        <bodycontent>EMPTY</bodycontent>
        <info>Simplest example: inserts one line of output</info>
    </tag>
</taglib>
```

Exemple

- Pour le tag lib *iminfo:example*, on obtient le résultat suivant :



Attributs de Tags Libraries (1)

- Il est possible de passer des paramètres aux tags libraries
- On gagne alors en flexibilité
- La syntaxe pour le fichier JSP est alors :

```
<prefix:name attribute1="value1" attribute2="value2"/>
```

- La classe Java doit alors posséder :

```
public void setAttribute1(String value1) {
    doSomethingWith(value1);
}
```

Attributs de Tags Libraries (2)

- Le fichier XML doit décrire ces attributs
- Il permet de préciser s'ils sont obligatoires

```
...
<tag>
    <name>prime</name>
    <tagclass>coreservlets.tags.PrimeTag</tagclass>
    <info>Outputs a random N-digit prime.</info>
    <bodycontent>EMPTY</bodycontent>
    <attribute>
        <name>length</name>
        <required>false</required>
    </attribute>
</tag>
...
```

Utilisation du TagBody (1)

- Le texte situé entre les balises (le corps du tag) peut être exploité

```
<prefix:name attribute1="value1">
    Corps du tag
</prefix:name>
```

- Il peut également être intéressant d'intercepter la fermeture du tag
- On peut alors redéfinir la méthode *doEndTag()*

Utilisation du TagBody (2)

- On peut préciser si le corps du tag doit être inclus dans le fichier HTML
- Une des applications possible est l'utilisation d'un mode « debug »

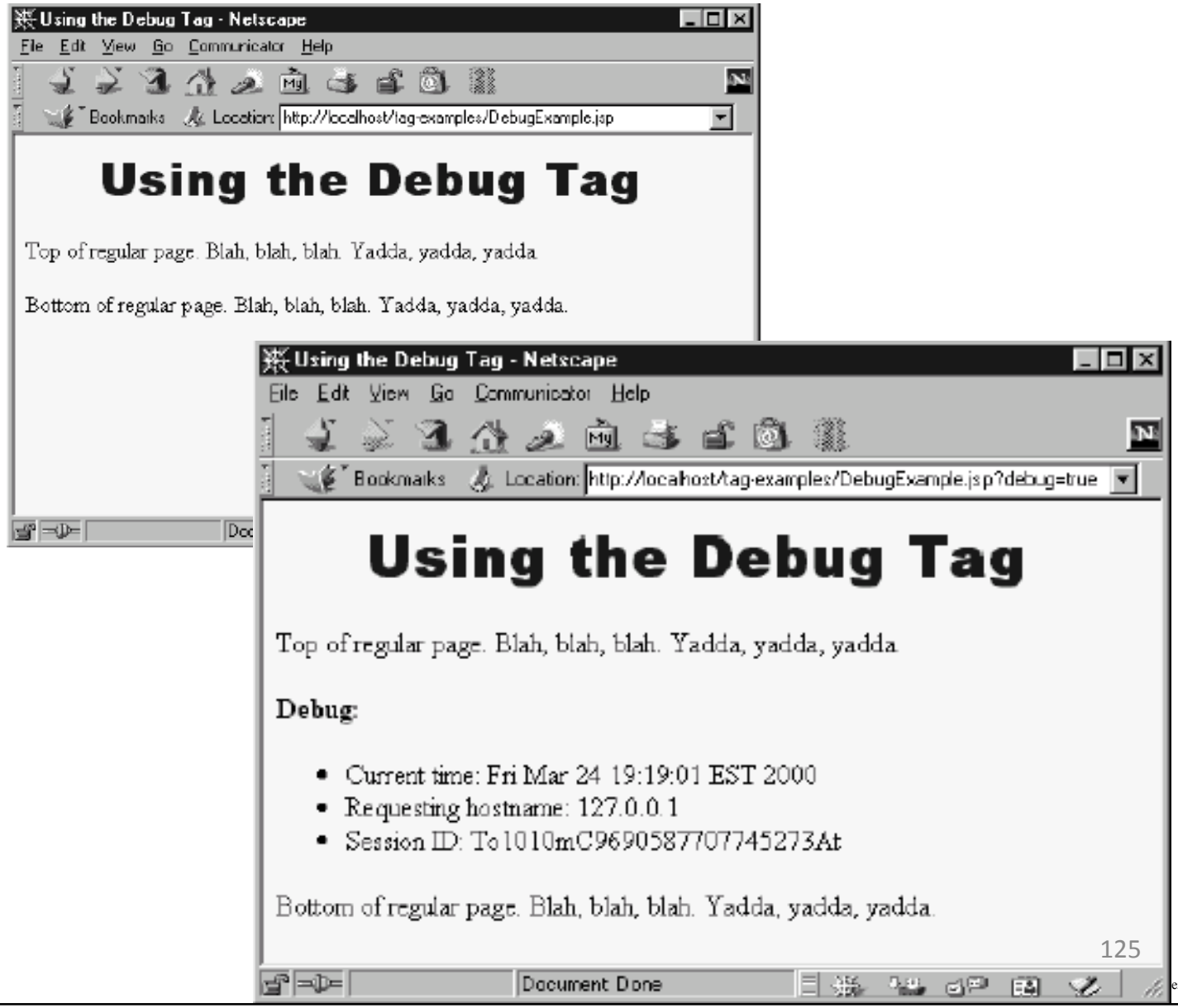
```
public class DebugTag extends TagSupport {
    public int doStartTag() {
        ServletRequest request = pageContext.getRequest();
        String debugFlag = request.getParameter("debug");
        if ((debugFlag != null) &&
                (!debugFlag.equalsIgnoreCase("false"))) {
            return(EVAL_BODY_INCLUDE);
        } else {
        return(SKIP_BODY);
        }
    }
}
```

Utilisation du TagBody (2)

- Le fichier JSP devient alors

```
<BODY>
<H1>Using the Debug Tag</H1>
<%@ taglib uri="iminfo-taglib.tld" prefix="csajsp" %>
Top of regular page. Blah, blah, blah. Yadda, yadda,
<P>
<csajsp:debug>
<B>Debug:</B>
<UL>
    <LI>Current time: <%= new java.util.Date() %>
    <LI>Requesting hostname: <%= request.getRemoteHost() %>
    <LI>Session ID: <%= session.getId() %>
</UL>
</csajsp:debug>
</P>
Bottom of regular page. Blah, blah, blah. Yadda, yadda.
</BODY>
```

Utilisation du TagBody (3)

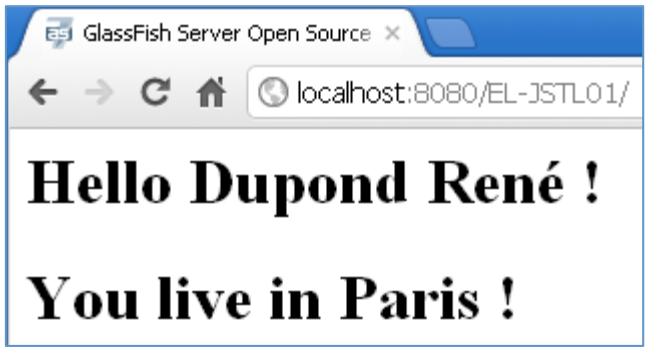


Expression Language 2.2

- Langage de script permettant l'accès à des objets Java à travers des JSP
- Evolution de la spec afin de faire évoluer JSF

```
<html>
  <body>
  <jsp:useBean id="p" class="com.test.Personne" />
       <jsp:setProperty property="nom" value="Dupond" name="p"/>
       <jsp:setProperty property="prenom" value="René" name="p"/>
       <jsp:setProperty property="ville" value="Paris" name="p"/>
    <h1>Hello <%=p.getNom() %> ${p.prenom} !</h1>

    <h1>You live in ${p.getVille()} !</h1>
  </body>
</html>
```



JSTL

- Java server page Standard Tag Library
- Ensemble de tags personnalisés
  - Tag de structure (itération, conditionnement ...)
  - Internationalisation
  - Exécution de requête SQL
  - Utilisation de document XML
- JSTL nécessite un conteneur d'application web qui implémente l'API servlet 2.3 et l'API JSP 1.2

JSTL 1.2 (Java EE5)

Librairie	URI	Préfixe
core	http://java.sun.com/jsp/jstl/core	c
Format	http://java.sun.com/jsp/jstl/fmt	fmt
XML	http://java.sun.com/jsp/jstl/xml	x
SQL	http://java.sun.com/jsp/jstl/sql	sql
Fonctions	http://java.sun.com/jsp/jstl/functions	fn

JSTL 1.2 (Java EE5)

- <c:/> : librairie de base :

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

  - Gestion des variables de scope

```
<c:set scope="session" var="varName" value="${expression}" />
```

  - Actions conditionnelles

```
<c:if test="${expressionBooleenne}">
   Affichage conditionnel
</c:if>
```

  - Itérations

```
<c:forEach var="compteur" begin="10" end="100">
   ${compteur}
</c:forEach>
```

JSTL 1.2 (Java EE5)

- Les URLs

```
<c:url value="/index.jsp?variable=valeur"/>
            <!-- ou -->
<c:url value="/index.jsp">
   <c:param name="variable" value="valeur"/>
</c:url>
```

- <fmt:/> : librairie de formatage

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

- Internationalisation

```
<fmt:setLocale value="fr" scope="page"/>
```

- Formatage

```
<fmt:formatNumber value="0.75" type="percent"/>
```

JSTL 1.2 (Java EE5)

- <sql:/> : Librairie SQL

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

```
<sql:setDataSource dataSource="jdbc/maDataSource" scope="page"/>

<sql:query sql="select * from maTable" var="resultat"/>

<c:forEach var="ligne" items="${resultat.rowsByIndex}">
   <c:forEach var="valeur" items="${ligne}">${valeur}</c:forEach>
</c:forEach>
```

JSTL 1.2 (Java EE5)

- <x:/> : Librairie XML

```
<x:parse var="parsedDoc">
   <a><b><c></c></b>
      <d><e x="1"><f>nombre 1</f></e><e x="2"><f>nombre 2</f></e>
      </d>
   </a>
</x:parse>

<x:out select="$parsedDoc/a/d/e[@x='2']/f"/>
```

```
<c:import url="/fic.xml" var="ficXml"/>
<c:import url="/fic.xsl" var="ficWsl"/>

<x:transform doc="${ficXml}" xslt="${ficWsl}"/>
```

JSTL 1.2 (Java EE5)

- ${fn:} : Librairie de fonctions EL
  - La plupart des fonctions de cette librairie concernent la gestion des chaînes de caractères.

```
<c:if test="${ fn:contains('abcdefghijk', 'bcd') }"> OK </c:if> <br>

<c:out  value="${ fn:length('abcdefghijk') }"/>
```