Université Cheikh Anta Diop
Ecole Supérieure Polytechnique
Département Génie Informatique

JAVA ENTERPRISE EDITION

TIERS WEB

Formateur

M. Mouhamed DIOP

mouhamed.diop@esp.sn

DIC3 / ESP

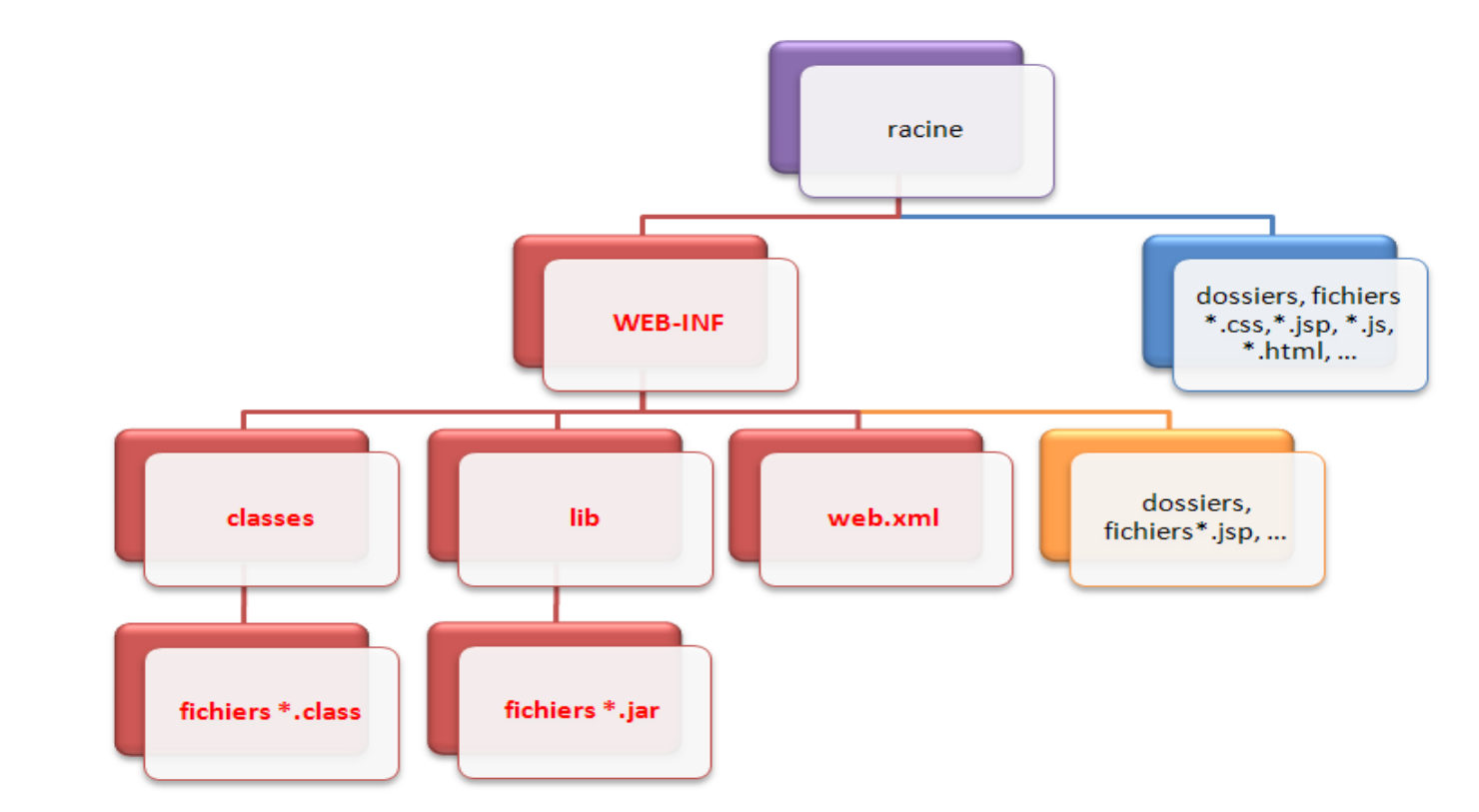
MASTER II / EC2LT

Plan

- Empaquetage des applications Web en Java
- Présentation des servlets
- Présentation de JSP
- Java Standard tag Library (JSTL)
- Accès à une base de données en Java
- Modèle MVC

Empaquetage des applications Web en Java

- La norme JEE décrit comment doivent être organisés les fichiers d'une application pour pouvoir être prise en charge par n'importe quel serveur d'application compatible



Empaquetage des applications Web en Java

- La racine de l'application
  - C’est le dossier qui porte le nom de votre projet et qui contient l'intégralité des dossiers et fichiers de l'application.
- WEB-INF : contient les éléments accessibles uniquement par le serveur
  - le fichier de configuration de l'application (web.xml)
  - le dossier « classes » qui contient les classes compilées (fichiers .class)
  - le dossier « lib » qui contient les bibliothèques nécessaires au projet (archives .jar)
- Les fichiers et dossiers « personnels »
  - Placés sous WEB-INF, ils sont privés et ne sont pas accessible via leurs URL
  - Placés directement sous la racine, ils sont directement accessibles aux clients via leurs URL

Présentation des servlets

Présentation des Servlets

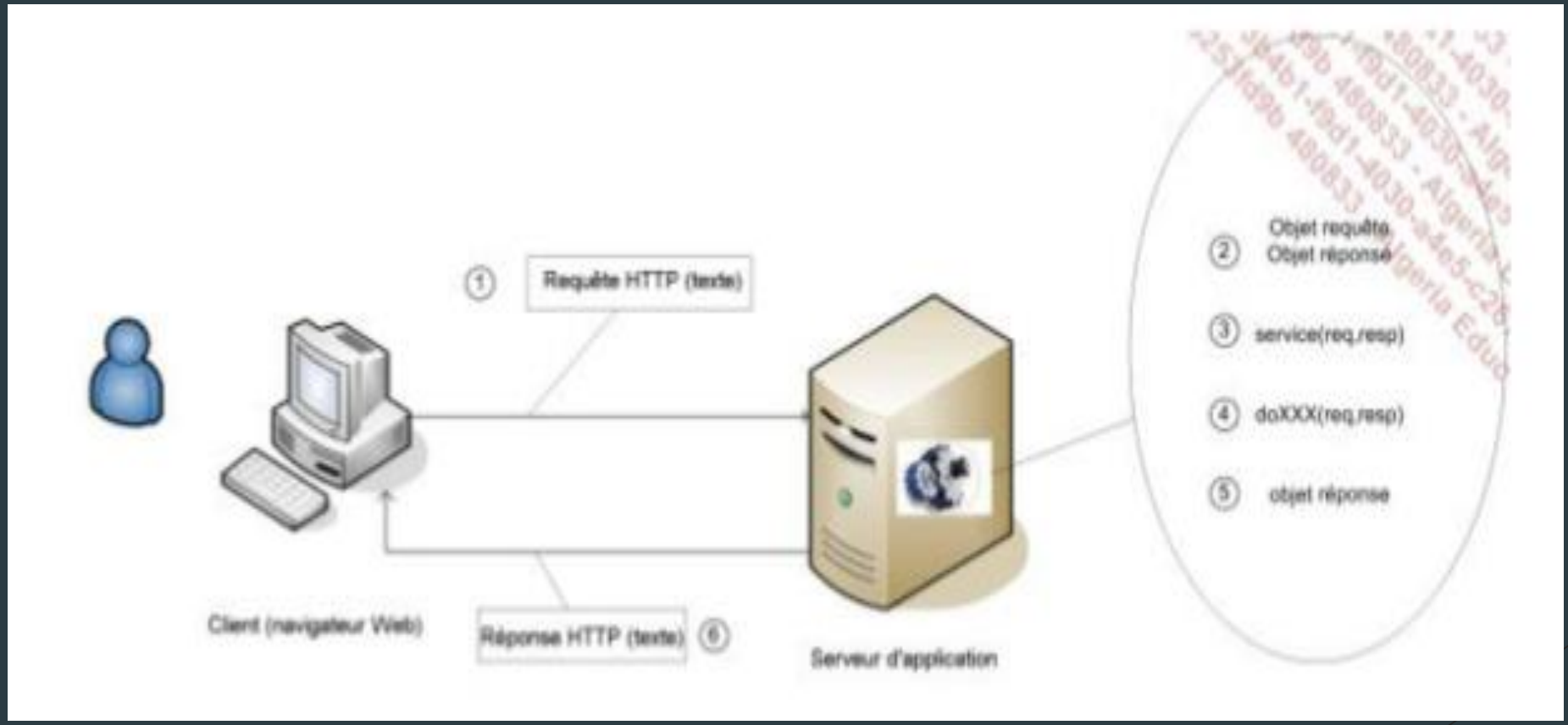
- Une servlet est une simple classe Java capable de générer du contenu Web dynamique
- Pour qu'une classe représente une servlet, elle doit implémenter l'interface « javax.servlet »
- Des servlets de base existent déjà en Java EE
  - Servlet de traitement de flux HTTP (javax.servlet.http.HttpServlet)
- Pour qu'une classe représente une servlet, elle peut donc étendre la classe « HttpServlet »

Présentation des Servlets

- Le protocole HTTP est basé sur des couples question/réponse.
  - Le client génère une requête contenant des informations
  - Le serveur analyse la requête et effectue des traitements nécessaires pour construire la réponse.
- A la réception d'une requête, le serveur :
  - Crée un objet Java représentant la requête (javax.servlet.http.HttpServletRequest)
  - Crée un objet Java représentant la réponse (javax.servlet.http.HttpServletResponse)
  - Transmet le couple d'objets à une Servlet suivant l'URL (appel de sa méthode « service »)
  - La méthode service de la servlet fait appel à la méthode « doGet » ou « doPost » selon le type de requête effectuée
  - La méthode « doXXX » transmet la réponse

Présentation des Servlets

- Traitement d'une requête HTTP



Présentation des Servlets

- La servlet est instanciée par le serveur qui la garde en mémoire
  - Une seule instance de la Servlet instanciée gère les requêtes HTTP la concernant
- Déclaration d'une servlet (fichier web.xml)
  - Déclaration

```
<servlet>
    <servlet-name>PremiereServlet</servlet-name>
    <servlet-class>sn.ec2lt.PremiereServlet</servlet-class>
</servlet>
```

  - Mapping des URLs

```
<servlet-mapping>
    <servlet-name>PremiereServlet</servlet-name>
    <url-pattern>/PremiereServlet</url-pattern>
</servlet-mapping>
```

Présentation des Servlets

- Déclaration d'une servlet (Annotation)

```
@WebServlet(name="PremiereServlet",
    urlPatterns={"/PremiereServlet"})
public class PremiereServlet HttpServlet {}
```

- Paramètres d'initialisation

```
<servlet>
    <servlet-name>PremiereServlet</servlet-name>
    <servlet-class>sn.ec2lt.PremiereServlet</servlet-class>
    <init-param>
        <description>adresse du serveur de base de données </description>
        <param-name>adresseIpBDD</param-name>
        <param-value>127.0.0.1</param-value>
    </init-param>
</servlet>
```

Présentation des Servlets

- Paramètres d'initialisation

```
@WebServlet(name="PremiereServlet",
urlPatterns={"/PremiereServlet", "/FirstServlet" },
initParams={@WebInitParam(description="adresse du serveur de base de données",
name="adresseIpBDD" , value="127.0.0.1")})
public class MyServlet extends HttpServlet {}
```

- Méthode init () : Appelée par le serveur à l'instanciation de la servlet

```
@Override
public void init() throws ServletException{
    super.init();
    String ipServeur = getInitParameter("adresseIpBDD");
    if (ipServeur == null)
        throw new ServletException();
    cnx = ouvrirConnexionBDD(ipServeur);
    if(cnx == null){throw new ServletException();}
}
```

Présentation des Servlets

- Méthode destroy() : appelée lorsque le serveur va détruire la servlet (à l'arrêt du serveur)
- Méthode service() : détermine le type de requête (POST, GET …) afin de décider de quelle méthode appelée.
- Méthodes doXXX() : constituent le cœur de la servlet

```
public class TestServlet extends HttpServlet{
    @Override
    protected void doGet(HttpServletRequestrequest,HttpServletResponse response) throws
         ServletException, IOException {
        //code de la servlet
    }
    @Override
    protected void doPost(HttpServletRequestrequest,HttpServletResponse response)
        throws ServletException, IOException {
        doGet(request, response);
    }
}
```

Présentation des Servlets

- Informations sur l'URL
- getServerName() : extrait de l'URL l'adresse du serveur devant traiter la requête.
- getServerPort() : Le port de la requête. 80 par défaut.
- getContextPath() : Récupère le nom de l'application qui héberge la servlet
- getServletPath() : retourne la chaine permettant d'identifier la servlet
- getRequestURL() : retourne l'URL utilisé pour contacter la servlet
- getLocalAddr() : retourne l'adresse IP de provenance de la requête

```
out.println("<a href=" + request.getContextPath() +"/page2.html>vers la suite</a>");
```

Manipulation de la requête

- Informations sur l'URL : Exemple

```
protected void doGet(HttpServletRequestrequest,
    HttpServletResponse response) throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    try{
        out.println(request.getServerName());
        out.println(request.getServletPath());
        out.println(request.getMethod());
        out.println(request.getQueryString());
        out.println(request.getRequestURL());
        out.println(request.getLocalAddr());
        out.println(request.getLocalName());
        out.println(request.getLocalPort());
        out.println(request.getRemoteAddr());
        out.println(request.getRemoteHost());
    }……
```

Manipulation de la requête

- Lecture des paramètres
- Les URLs sont sous la forme :

  …?param1=valeur_de_param1&param2=valeur_de_param2&...
- getParameter(String name)
  - Récupère le paramètre dont le nom est spécifié. Retourne null s'il n'existe pas.
- getParameterValues(String name)
  - Retourne toutes les valeurs du paramètre dont le nom est name (ex: cases à cocher)
- getParameterNames()
  - Retourne sous forme d'énumération le nom de tous les paramètres de la requête.
- getParameterMap()
  - Retourne les valeurs sous forme de map.

Manipulation de la requête

- Ajout d'autres informations à la requête
- setAttribute(String name,Object o)
  - Stocke un objet dans la requête.
- getAttribute(String name)
  - permet d'extraire un objet ajouté par setAttribute
- getAttributeNames()
  - Fournit la liste des noms des attributs dans la requête
- removeAttribute(String name)
  - Supprime un attribut de la requête.

Construire la réponse

```
protected void doGet(HttpServletRequestrequest,
    HttpServletResponse response) throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    try{
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Accueil</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h2> Bienvenue sur le site de EC2LT </h2> ");
        out.println("</body></hmtl>");
    }finally{
        out.close();
    }
}
```

Construire la réponse

- Contexte de l'application
  - getServletContext().getInitParameter(string name)

```
<context-param>
    <param-name>logoClient</param-name>
    <param-value>eniecole.gif</param-value>
</context-param>
```

```
out.println(" <imgsrc=\"images/" + getServletContext()
    .getInitParameter("logoClient")
    + "\" width=\"50\" height=\"50\" alt=\"logo\"/>");
```

- La session
  - Elle peut également être manipulée depuis la servlet et permet de stocker des informations.
  - Représentée par l'objet HttpSession
  - HttpServletRequest#getSession() : HttpSession
    - Retourne la session de la requête HTTP

Eléments accessibles depuis la servlet

- La session (suite)
  - HttpSession#setAttribute(String name, Object value)
    - Ajouter ou mettre à jour une information dans la requête HTTP
  - HttpSession#getAttribute(String name) : Object
    - Récupérer un objet ajouté dans la session
  - HttpSession#removeAttribute(String name)
    - Supprimer un attribut de la session
  - HttpSession#getAttributeNames() : Enumeration
    - Retourne sous forme d'énumération la liste des attributs présents dans la session
  - Mettre fin à la session : HttpSession#invalidate()
  - Temps d'inactivité avant fermeture de la session

```
<session-config>
    <session-timeout>10</session-timeout>
</session-config>
```

Autres ressources de la servlet

- RequestDispatcher
- Include

```
RequestDispatcher rd=null;
rd=getServletContext().getRequestDispatcher("/entete.html");
rd.include(request, response);
out.println("ce texte est généré par la servlet");
rd=getServletContext().getRequestDispatcher("/pied.html");
rd.include(request, response);
```

- Forward

```
RequestDispatcher rd=null;
rd =getServletContext().getRequestDispatcher("/resultats.jsp");
request.setAttribute("resultat", resultat);
rd.forward(request, response);
```

- Redirection

```
response.sendRedirect("http://www.eni-ecole.fr");
```

Les filtres

- Les filtres sont utilisés pour effectuer des traitements sur la requête avant qu'elle n'arrive au serveur, et sur la réponse avant d'être envoyée au client.
- Pour être considérée comme filtre, une classe doit implémenter l'interface « javax.servlet.Filter »
- Comme avec les servlets, la déclaration des filtres peut se faire de deux manières :
  - Au niveau du fichier de configuration de l'application (web.xml)
  - Au niveau de la classe elle-même en utilisant des annotations
- Plusieurs filtres peuvent être appliqués sur une requête / réponse
  - L'ordre d'application des filtres suit leur ordre de déclaration dans le fichier web.xml
  - En cas d'utilisation des annotations, l'ordre d'application ne peut être définie

Les filtres

- Déclaration au niveau du fichier web.xml

```
<filter>
    <description>…..</description>
    <filter-name>statistiques navigateur</filter-name>
    <filter-class>ec2lt.FiltreNavigateur</filter-class>
    <init-param>
        <param-name>nomFichierLog</param-name>
        <param-value>c:\logs\statNavigateur.log</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>statistiques navigateur</filter-name>
    <url-pattern>/RevenuPlacement</url-pattern>
</filter-mapping>
```

Les filtres

- Déclaration avec les annotations

```
@WebFilter(filterName="Statistique navigateur",
    urlPatterns= {"/revenuPlacement"}, description="...",
    @WebInitParam(name="nomFichierLog", value="c:/logs/statNavigateur.log"))
public class FiltreNavigateur implements Filter
{
    @Override
    public void destroy(){}

    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
            FilterChain filterChain) throws IOException, ServletException
    {
        // Le code du filtre ici
        filterChain.doFilter(); // Passer la main au filtre suivant ou à
        //la servlet de traitement de la requête
    }

    @Override
    public void init(FilterConfig arg0) throws ServletException{}
}
```

Présentation de JavaServer Pages

Présentation de JavaServer Pages

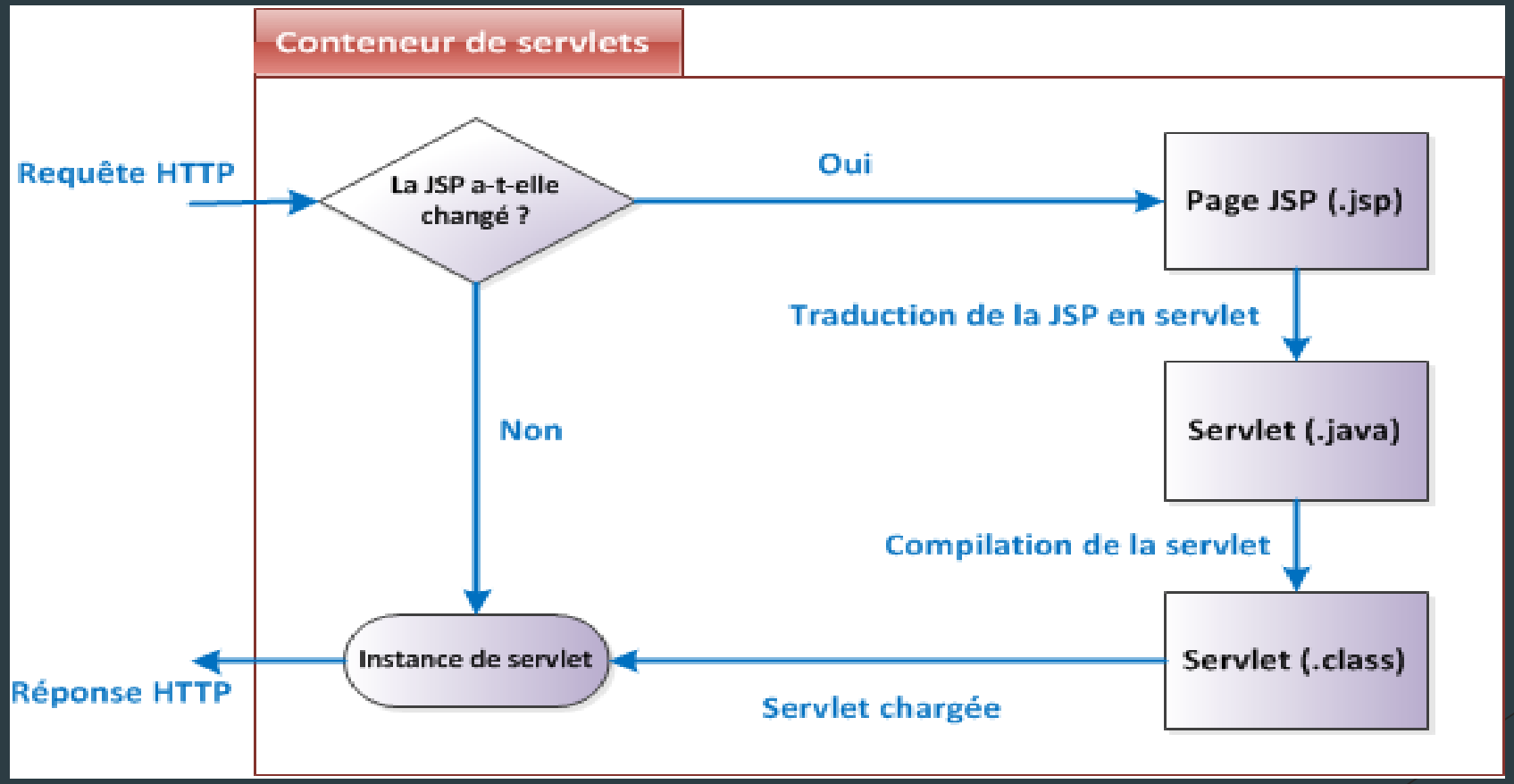
- Le modèle MVC préconise que tout ce qui touche à l'affichage final (texte, mise en forme, etc.) soit dans une couche à part : la vue
- Une page JSP ressemble à une page HTML mais en diffère par plusieurs aspects
  - Son extension est « .jsp » et non pas « .html »
  - Elle peut contenir des balises HTML, mais également des balises JSP qui appellent de manière transparente du code Java
  - Elle n'est pas directement renvoyée au client
    - Elle est exécutée côté serveur pour ainsi générer la page renvoyée au client.
- Les pages JSP sont donc un mélange de code HTML et de code Java destiné à produire du contenu dynamique
  - Nous les utiliserons pour générer du contenu dynamique au format HTML
  - Toutefois, une page JSP peut générer n'importe quel type de format (XML, JSON, etc.)

Présentation de JavaServer Pages

- Traitement d'une page JSP par le serveur
- Les pages JSP sont transformées par le serveur en des servlets
  - Quand une JSP est demandée pour la première fois, ou quand l'application web démarre, le conteneur de servlets va :
    - vérifier, traduire puis compiler la page JSP en une classe héritant de HttpServlet
    - Utiliser l'instance de la classe ainsi générée durant l'existence de l'application.
- La même instance sera utilisée pour une nouvelle requête HTTP portant sur la JSP
  - En cas de modification du code source de la JSP
    - les étapes de la compilation vont se faire à nouveau
    - une nouvelle instance sera créée pour gérer les futures requêtes

Présentation de JavaServer Pages

- Cycle de vie d'une JSP



Présentation de JavaServer Pages

- Exemple de page JSP

```
<%@page import="java.util.Date" pageEncoding="UTF-8"%>
<html>
    <head>
        <title>Ma première page JSP</title>
    </head>
    <body>
        <h1>Bonjour</h1>
        <p>
            <% Date date = new Date(System.currentTimeMillis()); %>
            Nous sommes le <% out.println(date.toLocaleString()); %>
        </p>
    </body>
</html>
```

Directives JSP

- Utilisés pour fournir des informations au serveur
- @page : Définit les caractéristiques du code généré
  - Language= "langage" : Par défaut Java
  - contentType="type Mime": indique le type de documents contenus dans la réponse HTTP généré par la JSP
  - import = "nomPackage1, nomPackage2…." : Définit les packages importés dans le code.
  - errorPage="url relative" : Lorsqu'une exception est générée dans la JSP, l'url relative est appelée par le serveur si l'exception n'est pas gérée.
  - pageEncoding="type d'encodage" : L'encodage de la page JSP
  - session="booléen" : cet attribut détermine si la session est accessible depuis la page JSP. Si c'est le cas la variable session accessible depuis la page JSP permet d'y faire référence
  - Etc. `<%@nomDeLaDirective attribut1="valeur1", attribut2="valeur2", ... %>`

```
<%@page contentType="text/html" pageEncoding="UTF-8" errorPage="erreur.jsp" import="java.util.*" %>
```

Directives JSP

- Directive include
  - Permet d'inclure une autre ressource (document html, texte, JSP ou XML) dans une page JSP

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

- Directive taglib
  - Permet le référencement de bibliothèques de balise externes.

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

Scriptlets

- Ils permettent d'insérer du code java dans une page JSP
- <%! %>
  - Pour déclarer de variables ou de méthodes qui seront utilisée dans les autres scriptlets de la page

```
<%!
    double tva=1.196;
    double calculTarif(double prix) {
        return prix*tva;
    }
%>
```

- <% %>
  - Pour l'insertion d'instruction Java dans la page JSP

```
<%
    double tva = 1.21;
    out.println(Double.parseDouble(request.getParameter("prix"))*tva);
%>
```

Scriptlets

- <%= %>
  - Cette balise permet d'insérer dans la réponse HTTP envoyée vers le client le résultat de l'expression qu'elle contient. L'expression est convertie en chaîne de caractères.
  - L'expression ne doit pas être terminée par un point virgule
  - Peut être vu comme un raccourci de « out.println »

```
<%= etudiant.getNom() %>
```

- <%----%>
  - Pour ajouter des commentaires dans la page JSP
  - Tout ce qui se trouve à l'intérieur de cette balise est ignoré lors de l'analyse de la page JSP par le serveur

```
<%--calcul du prix ttc --%>
<%
    double tva=1.196;
    out.println(Double.parseDouble(request.getParameter("prix")) *tva);
%>
```

Les objets implicites

- Des objets couramment utilisés sont accessibles directement par l'intermédiaire de variables prédéfinies
- request
  - cette variable permet d'obtenir une référence sur l'objet HttpServletRequest utilisé pour contacter la page JSP.
- response
  - cette variable permet d'obtenir une référence sur l'objet HttpServletResponse qui est utilisé pour transférer vers le client le résultat du traitement de la page JSP
  - Cet objet n'est pas très utile puisque beaucoup d'autres solutions sont disponibles pour insérer des informations dans la réponse HTTP faite au client

Les objets implicites

- pageContext
  - cette variable permet un accès à l'objet pageContext associé à la page JSP
- session
  - cette variable contient une référence vers l'objet HttpSession associé au client
  - Pour que cette variable soit accessible il ne faut pas avoir désactivé l'utilisation des sessions dans la directive page
- application
  - cette variable référence l'objet ServletContext associé à l'application web contenant la page JSP

Les objets implicites

- out
  - cette variable représente le flux de sortie sous forme d'un objet JspWriterpermettant d'écrire dans le corps de la réponse HTTP.
  - Cet objet est utilisé entre autre par le scriptlet <%= %>
- exception
  - cette variable est disponible uniquement dans une page JSP dédiée à la gestion des erreurs.
  - Elle permet d'obtenir une référence sur l'objet Exception à l'origine de la redirection vers cette page de gestion d'erreur

Les objets implicites : résumé

Identifiant	Type de l'objet	Description
pageContext	PageContext	Il fournit des informations utiles relatives au contexte d'exécution. Entre autres, il permet d'accéder aux attributs présents dans les différentes portées de l'application. Il contient également une référence vers tous les objets implicites suivants.
application	ServletContext	Il permet depuis une page JSP d'obtenir ou de modifier des informations relatives à l'application dans laquelle elle est exécutée.
session	HttpSession	Il représente une session associée à un client. Il est utilisé pour lire ou placer des objets dans la session de l'utilisateur courant.
request	HttpServletRequest	Il représente la requête faite par le client. Il est généralement utilisé pour accéder aux paramètres et aux attributs de la requête, ainsi qu'à ses en-têtes.
response	HttpServletResponse	Il représente la réponse qui va être envoyée au client. Il est généralement utilisé pour définir le Content-Type de la réponse, lui ajouter des en-têtes ou encore pour rediriger le client.
exception	Throwable	Il est uniquement disponible dans les pages d'erreur JSP. Il représente l'exception qui a conduit à la page d'erreur en question.
out	JspWriter	Il représente le contenu de la réponse qui va être envoyée au client. Il est utilisé pour écrire dans le corps de la réponse.
config	ServletConfig	Il permet depuis une page JSP d'obtenir les éventuels paramètres d'initialisation disponibles.
page	objet this	Il est l'équivalent de la référence this et représente la page JSP courante. Il est déconseillé de l'utiliser, pour des raisons de dégradation des performances notamment.

Les balises JSP

Le but des balises est de limiter au maximum l'utilisation de scriptlets dans les pages JSP

- <jsp:useBean>

```
<jsp:useBean id="..." scope="..." type="..." class="...">
    <!-- code à exécuter si le bean a été créé (n'existait pas) -->
</jsp:useBean>
```

  Le but de cette balise est de récupérer ou de créer une instance d'un objet JavaBean

  - id
    - représente le nom avec lequel le JavaBean est recherché
    - Utilisé ensuite pour nommer la variable accessible depuis la page JSP permettant la manipulation du JavaBean
  - class
    - cet attribut indique le nom complet de la classe (avec le package) utilisée pour instancier le JavaBean dans le cas où celui-ci n'est pas trouvé à l'emplacement indiqué par l'attribut scope.
  - type
    - détermine le type de la variable utilisée pour référencer le JavaBean dans la page JSP. Ce type doit être le même que celui indiqué par l'attribut class ou un super type de celui-ci.

Les balises JSP

- <jsp:useBean> (suite)

```
<jsp:useBean id="..." scope="..." type="..." class="...">
    <!-- code à exécuter si le bean a été créé (n'existait pas) -->
</jsp:useBean>
```

  - scope
    - indique l'emplacement où est recherché le JavaBean et où il sera stocké si la balise doit l'instancier.
    - Ses valeurs possibles sont :
      - page
      - request
      - session
      - application

```
<jsp:useBean id="monClient" type="sn.ec2lt.beans.Personne" class="sn.ec2lt.beans.Client" scope="request">
    <!-- code à exécuter si le bean a été créé (n'existait pas) -->
</jsp:useBean>
```

Les balises JSP

- <jsp:getProperty>

  utilisée conjointement avec la balise useBean pour insérer dans la page JSP la valeur d'une propriété d'un JavaBean

  - name
    - représente le nom du JavaBean présent dans la page JSP à partir duquel la propriété est obtenue
    - doit correspondre à la valeur de l'attribut id de la balise <jsp:useBean>
    - <jsp:useBean> doit apparaître dans la page JSP avant la balise <jsp:getProperty>
  - property
    - spécifie le nom de la propriété recherchée. La méthode getXXXXX est utilisée sur le JavaBean pour obtenir la valeur de cette propriété
- Affichage de la valeur de la propriété « nom » du JavaBean nommé « monClient » dans la page JSP.

```
<jsp:getProperty name="monClient" property="nom"/>
```

↕

```
<%= monClient.getNom() %>
```

Les balises JSP

- <jsp:setProperty>
  - permet l'affectation d'une valeur à une propriété d'un JavaBean
  - le JavaBean doit être disponible dans la page grâce à une balise <jsp:useBean>
  - peut également être utilisée pour transférer les valeurs issues d'un formulaire vers les propriétés d'un JavaBean
  - name
    - permet d'identifier le JavaBean dont la ou les propriétés seront modifiées
  - property
    - précise le nom de la propriété qui doit être modifiée
    - La valeur * peut également être utilisée pour cet attribut
      - la balise recherche dans la requête HTTP les paramètres portant le même nom que les propriétés du JavaBean et transfère la valeur de ces paramètres dans les propriétés correspondantes du JavaBean

Les balises JSP

- <jsp:setProperty>
  - value
    - utilisé lorsque l'on souhaite modifier une propriété d'un JavaBean.
    - C'est lui qui indique la valeur à transférer dans la propriété du JavaBean
  - Récupération d'un JavaBean depuis la session et affectation d'une valeur à sa propriété « heureConnexion »

```
<jsp:useBean id="client" scope="session" class="sn.ec2lt.beans.Client"/>
<jsp:setProperty name="client" property="heureConnexion" value="<%=new Date().toLocaleString()%>"/>
```

Les balises JSP

- <jsp:setProperty>

  Utilisation d'un formulaire HTML pour permettre la saisie du nom et du prénom du client

```
<form action="accueilClient.jsp">
    <table>
        <tr>
            <td>Nom</td>
            <td><input type="text" name="nom" value="" /></td>
        </tr>
        <tr>
            <td>Prénom</td>
            <td><input type="text" name="prenom" value="" /></td>
        </tr>
        <tr>
            <td><input type="submit" value="Valider" name="valider" /></td>
            <td><input type="submit" value="Annuler" name="annuler" /></td>
        </tr>
    </table>
</form>
```

Les balises JSP

- <jsp:setProperty>

  Les données recueillies sur ce formulaire sont ensuite envoyées à la page JSP ci-dessous qui initialise un JavaBean et affiche ensuite ses propriétés

```
<jsp:useBeanid="client" scope="session" class="sn.ec2lt.beans.Client" type="sn.ec2lt.beans.Client"/>
<jsp:setPropertyname="client" property="*"/>
<jsp:setPropertyname="client" property="heureConnexion" value="<%= new Date().toLocaleString() %>"/>
Bienvenue
<jsp:getPropertyname="client" property="nom"/>
<jsp:getPropertyname="client" property="prenom"/>
Heure de connexion <jsp:getPropertyname="client" property="heureConnexion"/>
```

Les balises JSP

- <jsp:include>

  permet d'inclure dans la réponse le contenu d'une autre ressource statique ou dynamique

  - page :

    URL de la ressource à inclure dans la réponse HTTP

  - flush :

    indique si le buffer doit être envoyé vers le client avant que la ressource soit incluse

  - <jsp:param>
    - permet de transmettre des paramètres à la ressource incluse
    - value :

      spécifie la valeur du paramètre

Les balises JSP

- <jsp:include>
  - appel à une servlet qui va chercher dans une base de données le nombre de commandes du client dont le code est passé dans une requête HTTP

```
<jsp:include page="/NbCommandes">
    <jsp:param name="code" value='<%= request.getParameter("codeClient") %>' />
</jsp:include>
```

  - Inclusion de l'entête et du pieds de page dans la page courante

```
<jsp:include page="inc/entete.jsp"/>
    <p>Contenu de ma page</p>
<jsp:include page="inc/pied.html"/>
```

- <jsp:forward>
  - joue le même rôle que la méthode forward d'un objet RequestDispatcher
  - Les attributs de cette balise sont identiques à ceux de la balise <jsp:include>
    - A l'exception de l'attribut « flush » qui n'existe pas pour le forwarding
  - le code présent après cette balise dans la page n'est pas exécuté.

Expression Language (EL)

Expression Language

- Utilisées à l'intérieur d'une page JSP afin de faciliter la manipulation des données accessibles depuis cette page
- Elles permettent de s'affranchir en grande partie de l'écriture de scriptlets (du code Java) dans nos pages JSP.
- Les expressions EL permettent via une syntaxe très épurée
  - d'effectuer des tests basiques sur des expressions
  - de manipuler simplement des objets et attributs dans une page (sans utiliser de code ni de script Java)
  - Facilite la maintenance des pages JSP en améliorant la lisibilité du code

Expression Language

- Syntaxe générale : ${ expression }
- Dans une expression, on peut effectuer diverses sortes de tests
- Pour réaliser ces tests, il est possible d'inclure tout une série d'opérateurs
  - opérateurs arithmétiques, applicables à des nombres : +, -, *, /, %
  - opérateurs logiques, applicables à des booléens : && (and), || (or), ! (not)
  - opérateurs de comparaison, basés sur l'utilisation des méthodes equals() et compareTo() des objets comparés :

    == (eq), != (ne), < (lt), > (gt), <= (le), >= (ge).

```
${ true && false } <br /> <!-- Affiche false -->
${ !true || false } <br /> <!-- Affiche false -->
${ 10 % 4 } <br /> <!-- Affiche 2 -->
${ 6 * 7 } <br /> <!-- Affiche 42 -->
${ 7 / 0 } <br /> <!-- Affiche Infinity -->
${ 'a' < 'b' } <br /> <!-- true -->
${ 'hip' gt 'hit' } <br /> <!-- false -->
${ 'a' < 'b' && 'hip' gt 'hit' } <br />  <!-- false -->
```

Expression Language

- Deux autres types de test sont fréquemment utilisés au sein des expressions EL :
  - les conditions ternaires, de la forme : test ? si oui : sinon
  - les vérifications si vide ou null, grâce à l'opérateur empty

```
<!-- Vérifications si vide ou null -->
${ empty 'test' } <!-- La chaîne testée n'est pas vide, le résultat est false -->
${ empty '' } <!-- La chaîne testée est vide, le résultat est true -->
${ !empty '' } <!-- La chaîne testée est vide, le résultat est false -->
<!-- Conditions ternaires -->
${ true ? 'vrai' : 'faux' } <!-- Le booléen testé vaut true, vrai est affiché -->
${ 'a' > 'b' ? 'oui'  : 'non' } <!-- Le résultat de la comparaison vaut false, non est affiché -->
${ empty 'test' ? 'vide' : 'non  vide'  } <!-- La chaîne testée n'est pas vide, non vide est affiché -->

<!-- La ligne suivante : -->
<p>12 est inférieur à 8 : ${ 12 lt 8 }.</p>
<!-- Sera rendue ainsi après interprétation de l'expression -->
<p>12 est inférieur à 8 : false.</p>
```

Expression Language

- Accès aux objets (javabeans)
  - On peut facilement accéder aux propriétés des objets avec les EL
  - Il en est de même pour leurs méthodes
  - Toutefois l'accès à travers les propriétés est conseillé partout où c'est possible
    - Le conteneur appelle implicitement le getter adéquat
    - L'accès se base sur les conventions d'écritures des méthodes en Java

```
<!-- Syntaxe conseillée pour récupérer la propriété 'prenom' du bean 'etudiant'. -->
${ etudiant.prenom }
<!-- Syntaxe correcte, mais pas recommandée -->
${ etudiant.getPrenom() }
<!-- Syntaxe erronée : la première lettre de la propriété doit être une minuscule. -->
${ etudiant.Prenom }
```

Expression Language

- Lorsqu'une EL est évaluée, il peut se produire une exception
  - Dans ce cas, l'expression prend automatiquement la valeur null
- Les expressions EL sont protégées contre un éventuel retour null

```
<!-- Affiche "null" si le prénom de l'étudiant n'a pas été initialisé  -->
<%= etudiant.getPrenom() %>

<!-- Affiche "null" si le prénom de l'étudiant n'a pas été initialisé  -->
<jsp:getProperty name="etudiant" property="prenom" />

<!-- N'affiche rien si le prénom de l'étudiant n'a pas été initialisé  -->
${ etudiant.prenom }
```

Expression Language

- Les EL n'étaient pas définies dans les premières versions de JSP
- Pour la rétrocompatibilité, Il est possible de désactiver l'évaluation des EL
  - au cas par cas, grâce à la directive page
  - sur tout ou partie des pages, grâce à l'ajout d'une section dans le fichier web.xml
- Le comportement par défaut dépend de la version de l'API servlet utilisée
  - si la version est supérieure ou égale à 2.4, alors les expressions EL seront évaluées par défaut
  - si la version est inférieure à 2.4, alors il est possible que les expressions EL soient ignorées par défaut, pour assurer la rétrocompatibilité
- La version de l'API servlet utilisée est précisée dans le fichier web.xml
  - Elle correspond à la valeur de l'attribut « version » de la balise <web-app>
  - Elle doit être supportée par le serveur d'application

Expression Language

- Désactivation des EL avec la directive page `<%@ page isELIgnored ="true" %>`
  - Les seules valeurs acceptées par l'attribut isELIgnored sont true et false :
    - s'il est initialisé à true, alors les expressions EL seront ignorées et apparaîtront en tant que simples chaînes de caractères
    - s'il est initialisé à false, alors elles seront interprétées par le conteneur.
- Désactivation des EL depuis le fichier web.xml

```
<jsp-config>
    <jsp-property-group>
        <url-pattern>*.jsp</url-pattern>
        <el-ignored>true</el-ignored>
    </jsp-property-group>
</jsp-config>
```

  - L'url pattern permet de définir les pages concernées par la désactivation
  - Le comportement de <el-ignored> est le même qu'avec isELIgnored

Les objets implicites EL

Catégorie	Identifiant	Description
JSP	pageContext	Objet contenant des informations sur l'environnement du serveur.
Portée	pageScope	Une Map qui associe les noms et valeurs des attributs ayant pour portée la page.
	requestScope	Une Map qui associe les noms et valeurs des attributs ayant pour portée la requête.
	sessionScope	Une Map qui associe les noms et valeurs des attributs ayant pour portée la session.
	applicationScope	Une Map qui associe les noms et valeurs des attributs ayant pour portée l'application.
Paramètres de requête	param	Une Map qui associe les noms et valeurs des paramètres de la requête.
	paramValues	Une Map qui associe les noms et multiples valeurs des paramètres de la requête sous forme de tableaux de String.
En-têtes de requête	header	Une Map qui associe les noms et valeurs des paramètres des en-têtes HTTP.
	headerValues	Une Map qui associe les noms et multiples valeurs des paramètres des en-têtes HTTP sous forme de tableaux de String.
Cookies	cookie	Une Map qui associe les noms et instances des cookies.
Paramètres d'initialisation	initParam	Une Map qui associe les données contenues dans les champs <param-name> et <param-value> de la section <init-param> du fichier web.xml.

Java Standard Tag Library (JSTL)

Présentation

- Le développement d'une application est confié à plusieurs personnes en fonction de leurs spécialités
  - Présentation au spécialiste HTML et autres technologies associées (feuilles de style, graphisme …).
- Le but de la bibliothèque JSTL est de donner la possibilité d'insérer des portions dynamiques dans une page JSP sans pour autant être obligé d'apprendre le langage Java
  - Elle propose un ensemble de balises XML permettant d'insérer dans une page JSP des portions dynamiques

Présentation

- Quatre groupes sont définis
  - Bibliothèque de base
    - préfixe = c
    - uri = http://java.sun.com/jsp/jstl/core
  - Bibliothèque XML
    - préfixe = x
    - uri= http://java.sun.com/jsp/jstl/xml
  - Bibliothèque d'internationalisation
    - préfixe = fmt
    - uri = http://java.sun.com/jsp/jstl/fmt
  - Bibliothèque SQL
    - préfixe = sql
    - uri = http://java.sun.com/jsp/jstl/sql

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

Présentation

- <c:out> → Affichage sécurisé d'une expression
  - value
    - Seul attribut obligatoire, peut contenir une chaîne de caractères simple ou une expression EL
  - default
    - permet de définir une valeur affichée par défaut si le contenu de l'expression évaluée est vide
  - escapeXml
    - permet de remplacer les caractères de scripts < , > , " , ' et & par leurs équivalents en code html <, >, ", ', &
    - Cette option est activée par défaut, et on doit mettre sa valeur à « false » pour la désactiver.

```
<c:out value="test" escapeXml="false"/> <!-- Affiche test -->
<c:out value="${ 'a' < 'b' }" /> <!-- Affiche true -->
<c:out value="${client}" default="test" /> <!-- affiche "test" si "client" n'existe pas -->
<c:out value="${client}">test</c:out> <!-- même chose que la précédente -->
```

Présentation

- <c:set> → Affectation d'une valeur à une variable
  - var : indique le nom de la variable à créer ou à modifier
  - scope : indique l'emplacement où sera stockée la variable
  - value : la valeur à enregistrer dans cette variable
  - target : utilisé lorsque l'on souhaite modifier une propriété d'un objet
  - property: utilisé avec target pour identifier la propriété de l'objet que l'on souhaite modifier. Cette propriété doit être accessible via une méthode setXXXX.

```
<c:set var="navigateur" scope="session" value='${header["user-agent"]}' />
```

Présentation

- <c:if>

```
<c:if test="${!empty sessionScope['client'].nom }">
    ${sessionScope["client"].nom } <!-- ou encore ${sessionScope.client.nom } -->
</c:if>
```

- <c:choose>

  Effectue le même traitement que le switch du langage Java

```
<c:choose>
    <c:when test="${param['couleur']==1}">red</c:when>
    <c:when test="${param['couleur']==2}">green</c:when>
    <c:when test="${param['couleur']==3}">blue</c:when>
    <c:otherwise>yellow</c:otherwise>
</c:choose>
```

Présentation

- <c:forEach>

```
<table>
    <caption>Liste des utilisateurs</caption>
    <tr>
        <th>ID</th>
        <th>NOM</th>
        <th>PRENOM</th>
        <th>LOGIN</th>
    </tr>
    <c:forEach items="${ requestScope.utilisateurs }" var="utilisateur">
        <tr>
            <td><c:out value="${ utilisateur.id }"/></td>
            <td><c:out value="${ utilisateur.nom }"/></td>
            <td><c:out value="${ utilisateur.prenom }"/></td>
            <td><c:out value="${ utilisateur.login }"/></td>
        </tr>
    </c:forEach>
</table>
```

Présentation

- <c:import>
  - Pour l'insertion d'une ressource
  - Ressemble à <jsp:include>
- <c:redirect>
  - Pour la génération d'une réponse de redirection
- <c:url>
  - pour la génération d'une URL avec identifiant de session.