
Cookies

- Un cookie est une information envoyée au navigateur (client) par un serveur WEB qui peut ensuite être relue par le client
 - Lorsqu'un client reçoit un cookie, il le sauve et le renvoie ensuite au serveur chaque fois qu'il accède à une page sur ce serveur
 - La valeur d'un cookie pouvant identifier de façon unique un client, ils sont souvent utilisés pour le suivi de session
 - Les cookies ont été introduits par la première fois dans Netscape Navigator
-

Cookies

- L'API Servlet fournit la classe `javax.servlet.http.Cookie` pour travailler avec les Cookies
 - `Cookie(String name, String value)` : construit un cookie
 - `String getName()` : retourne le nom du cookie
 - `String getValue()` : retourne la valeur du cookie
 - `setValue(String new_value)` : donne une nouvelle valeur au cookie
 - `setMaxAge(int expiry)` : spécifie l'âge maximum du cookie
 - Pour la création d'un nouveau cookie, il faut l'ajouter à la réponse (`HttpServletResponse`)
 - `addCookie(Cookie mon_cook)` : ajoute à la réponse un cookie
 - La Servlet récupère les cookies du client en exploitant la réponse (`HttpServletRequest`)
 - `Cookie[] getCookies()` : récupère l'ensemble des cookies du site
-

Cookies

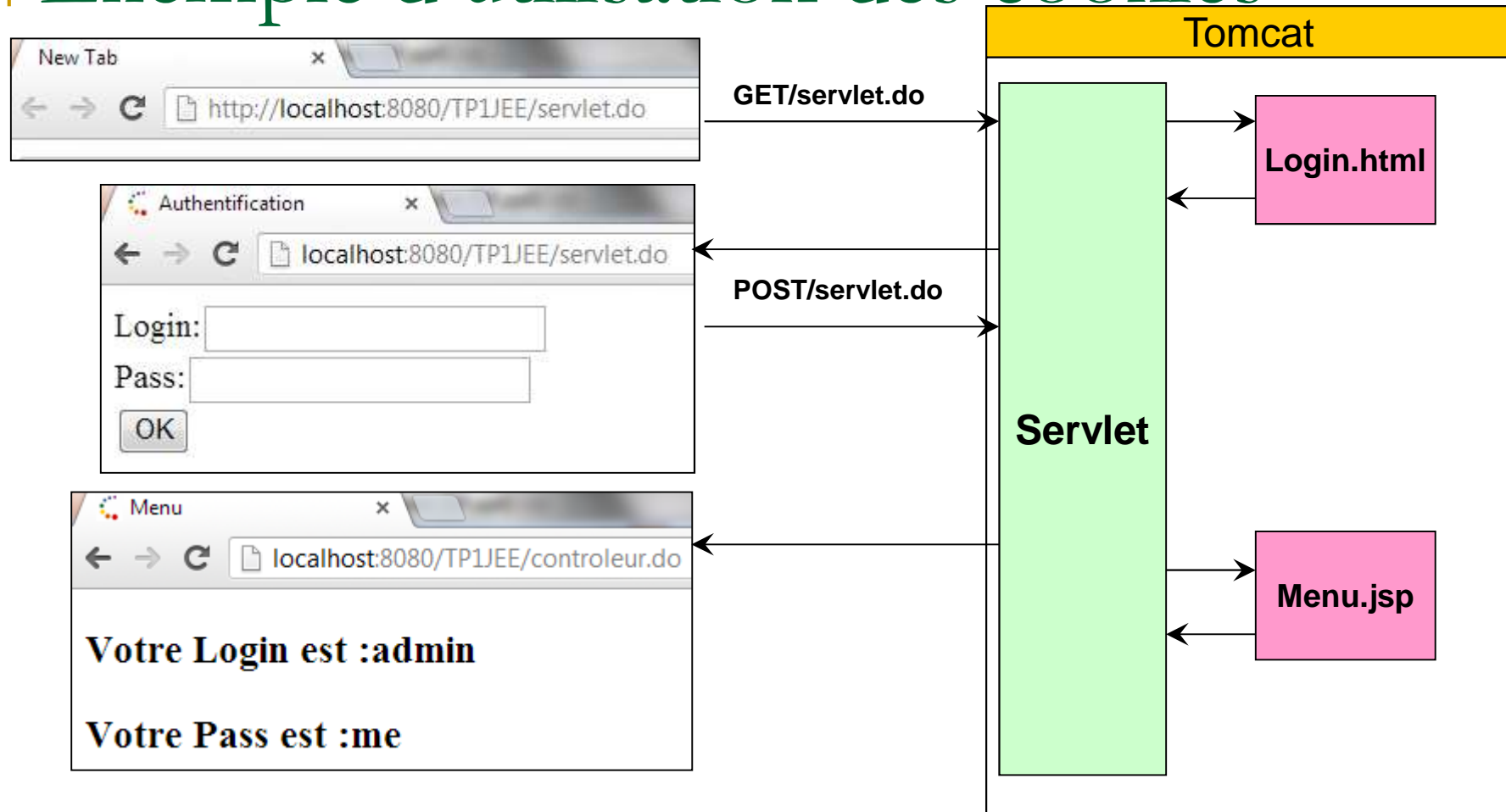
- Code pour créer un cookie et l'ajouter au client

```
Cookie cookie = new Cookie("Id", "123");  
cookie.setMaxAge(2*24*60*60); // Durée de vie=2 Jours  
response.addCookie(cookie);
```

- Code pour récupérer les cookies

```
Cookie[] cookies = req.getCookies();  
if (cookies != null) {  
    for (int i = 0; i < cookies.length; i++) {  
        String name = cookies[i].getName();  
        String value = cookies[i].getValue();  
    }  
}
```

Exemple d'utilisation des cookies



Exemple d'utilisation des cookies

```
package web; import java.io.*;
import javax.servlet.*;import javax.servlet.http.*;
public class FirstServlet extends HttpServlet {
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    Cookie[] cookies=request.getCookies();
    String login=null,pass=null;
    for(Cookie c:cookies){
        if(c.getName().equals("login")) login=c.getValue();
        if(c.getName().equals("pass")) pass=c.getValue();
    }
    if((login!=null)&&(pass!=null)){
        request.setAttribute("login", login); request.setAttribute("pass", pass);
        request.getRequestDispatcher("Menu.jsp").forward(request, response);
    } else{
        request.getRequestDispatcher("Login.html").forward(request, response);
    }
}}
```

Exemple d'utilisation des cookies

```
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse
response)
throws ServletException, IOException {
    String log=request.getParameter("login");
    String pass=request.getParameter("pass");
    Cookie cLog=new Cookie("login", log); cLog.setMaxAge(2*24*60*60);
    Cookie cPass=new Cookie("pass", pass);cPass.setMaxAge(2*24*60*60);
    response.addCookie(cLog);response.addCookie(cPass);
    request.setAttribute("login",log);
    request.setAttribute("pass", pass);
    request.getRequestDispatcher("Menu.jsp").forward(request, response);
}
}
```

Login.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-
    8859-1">
<title>Authentication</title>
</head>
<body>
    <form action="controleur.do" method="post">
        Login:<input type="text" name="Login"><br/>
        Pass:<input type="password" name="pass"><br/>
        <input type="submit" value="OK">
    </form>
</body>
</html>
```

Menu.jsp

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
    charset=ISO-8859-1">
<title>Menu</title>
</head>
<body>
    <h3>Votre Login est :<%=request.getAttribute("login")
    %></h3>
    <h3>Votre Pass est :<%=request.getAttribute("pass")
    %></h3>
</body>
</html>
```

HttpSession

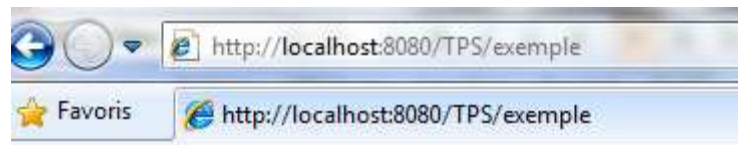
- Le plus gros problème des cookies est que les navigateurs ne les acceptent pas toujours
 - L'utilisateur peut configurer son navigateur pour qu'il refuse ou pas les cookies
 - Les navigateurs n'acceptent que 20 cookies par site, 300 par utilisateur et la taille d'un cookie peut être limitée à 4096 octets
-

HttpSession

- Solutions : utilisation de l'API de suivi de session HttpSession
 - Méthodes de création liées à la requête (HttpServletRequest)
 - HttpSession getSession() : retourne la session associée à l'utilisateur
 - HttpSession getSession(boolean p) : création selon la valeur de p
 - Gestion d'association (HttpSession)
 - Enumeration getAttributeNames() : retourne les noms de tous les attributs
 - Object getAttribute(String name) : retourne l'objet associé au nom
 - setAttribute(String na, Object va) : modifie na par la valeur va
 - removeAttribute(String na) : supprime l'attribut associé à na
 - Destruction (HttpSession)
 - invalidate() : expire la session
-

HttpSession

```
public class ExempleServlet extends HttpServlet {
    protected void doGet(HttpServletRequest req,
        HttpServletResponse res) throws ServletException,
        IOException {
        res.setContentType("text/plain");
        PrintWriter out = res.getWriter();
        HttpSession session = req.getSession();
        Integer count = (Integer)session.getAttribute("count");
        if (count == null)
            count = new Integer(1);
        else
            count = new Integer(count.intValue() + 1);
        session.setAttribute("count", count);
        out.println("Vous avez visité cette page " +count+ " fois." );
    }
}
```









Vous avez visité cette page 4 fois.




Exercice

- Créer une application web J2EE qui respecte le modèle MVC qui permet de simuler un jeu entre les clients http et le serveur web. Le principe du jeu est le suivant :
 - Le serveur choisit un nombre aléatoire entre 0 et 1000
 - Un client http connecté, doit saisir un nombre pour deviner le nombre secret.
 - Le serveur répond avec les éventualités suivantes :
 - ❑ Votre nombre est plus grand
 - ❑ Votre nombre est plus petit
 - ❑ Bravo, vous avez gagné. Et dans ce cas là le jeu s'arrête et pour chaque tentative de jouer le serveur envoi au client un message qui indique que le jeu est terminé en affichant le nombre secret recherché et le gagnant
 - ❑ L'application devrait également permettre de relancer le jeu si ce dernier est terminé.
-

Aperçu du Jeu

Adresse  http://localhost:8080/TPServJSP/VueJeu.jsp

Google  Envoyer     Mes fav

 powered by  SEARCH  Search We

Le serveur à choisi un nombre secrêt entre 0 et 100

Déviniez ce nombre:

Bravo vous avez gagné

Historique:

Nombre :100 Indication:Votre nombre est plus grand
Nombre :60 Indication:Votre nombre est plus grand
Nombre :30 Indication:Votre nombre est plus petit
Nombre :40 Indication:Votre nombre est plus petit
Nombre :50 Indication:Bravo vous avez gagné

Collaboration de servlets

- Les Servlets qui s'exécutant dans le **même** serveur peuvent dialoguer entre elles
 - Deux principaux styles de collaboration
 - Partage d'information : un état ou une ressource.
 - Exemple : un magasin en ligne pourrait partager les informations sur le stock des produits ou une connexion à une base de données
 - Partage du contrôle :
 - Réception d'une requête par une Servlet et laisser l'autre Servlet une partie ou toute la responsabilité du traitement
-

Collaboration de servlets

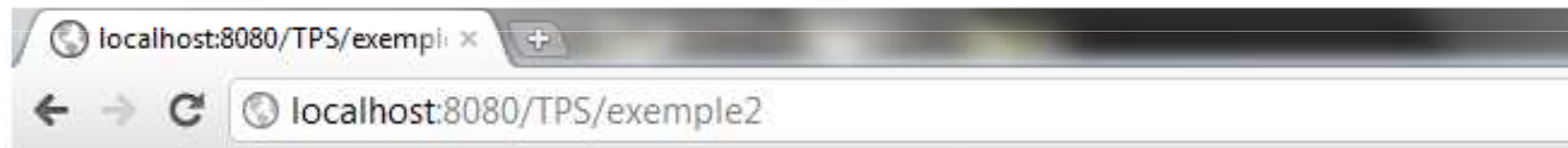
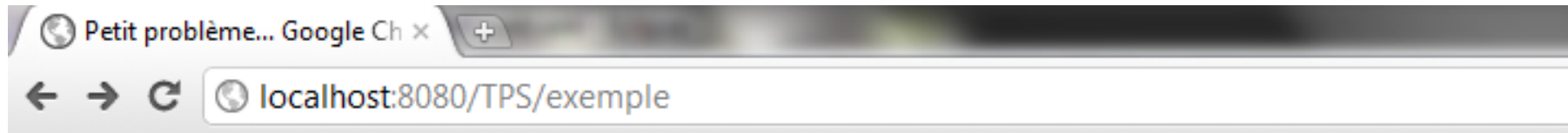
- La collaboration est obtenue par l'interface ServletContext
 - L'utilisation de ServletContext permet aux applications web de disposer de son propre conteneur d'informations unique
 - Une Servlet retrouve le ServletContext de son application web par un appel à `getServletContext()`
 - Exemples de méthodes
 - `void setAttribute(String name, Object o)` : lie un objet sous le nom indiqué
 - `Object getAttribute(String name)` : retrouve l'objet sous le nom indiqué
 - `Enumeration getAttributeNames()` : retourne l'ensemble des noms de tous les attributs liés
 - `void removeAttribute(String name)` : supprime l'objet lié sous le nom indiqué
-

Exemple : Servlets qui vendent des pizzas et partagent une spécialité du jour

```
public class ExempleServlet extends HttpServlet {
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        res.setContentType("text/plain");
        PrintWriter out = res.getWriter();
        ServletContext context = this.getServletContext();
        context.setAttribute("Specialite", "Forestière");
        context.setAttribute("Date", new Date());
        out.println("La pizza du jour a été définie.");
    }
}
```

```
public class ExempleServlet2 extends HttpServlet {
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        PrintWriter out=res.getWriter();
        ServletContext context = this.getServletContext();
        String pizza_spec = (String)context.getAttribute("Specialite");
        Date day = (Date)context.getAttribute("Date");
        DateFormat df = DateFormat.getDateInstance(DateFormat.MEDIUM);
        String today = df.format(day);
        out.println("Aujourd'hui (" + today + "), notre specialite est : " +
            pizza_spec);
    }
}
```


Exemple : Servlets qui vendent des pizzas et partagent une spécialité du jour



Partage d'informations

- Possibilité de partager des informations entre contextes web
 - Première solution : utilisation d'un conteneur d'informations externes (une base de données par exemple)
 - Seconde solution : la Servlet recherche un autre contexte à partir de son propre contexte
 - `ServletContext getContext(String uripath)` : obtient le contexte à partir d'un chemin URI (uripath = chemin absolu)
-

Partage de contrôle

- Les Servlets peuvent partager ou distribuer le contrôle de la requête
 - Deux types de distribution
 - Distribuer un renvoi : une Servlet peut renvoyer une requête entière
 - Distribuer une inclusion : une Servlet peut inclure du contenu généré
 - Les avantages sont
 - La délégation de compétences
 - Une meilleure abstraction et une plus grande souplesse
 - Architecture logicielle MVC (Servlet = contrôle et JSP = présentation)
-

Partage de contrôle

- Le support de la délégation de requête est obtenu par l'interface `RequestDispatcher`
 - Une Servlet obtient une instance sur la **requête**
 - ❑ `RequestDispatcher getRequestDispatcher(String path)` : retourne une instance de type `RequestDispatcher` par rapport à un composant
 - ❑ Un composant peut-être de tout type : Servlet, JSP, fichier statique, ...
 - ❑ `path` est un chemin relatif ou absolu ne pouvant pas sortir du contexte
 - Pour distribuer en dehors du contexte courant il faut :
 - ❑ Identifier le contexte extérieur (utilisation de `getContext()`)
 - ❑ Utiliser la méthode `getRequestDispatcher(String path)`
 - ❑ Le chemin est uniquement en absolu
-

Partage de contrôle

- La méthode `forward(...)` de l'interface `RequestDispatcher` renvoie une requête d'une Servlet à une autre ressource sur le serveur
 - ❑ `void forward(ServletRequest req, ServletResponse res)` : redirection de requête

```
RequestDispatcher dispat =  
req.getRequestDispatcher("/index.html");  
dispat.forward(req,res);
```
 - Possibilité de transmettre des informations lors du renvoi
 - ❑ en attachant une chaîne d'interrogation (au travers de l'URL)
 - ❑ en utilisant les attributs de requête via la méthode `setAttribute(...)`
 - Les choses à ne pas faire ...
 - ❑ ne pas effectuer de modification sur la réponse avant un renvoi
 - ❑ ne rien faire sur la requête et la réponse après une distribution d'un renvoi
-

Partage de contrôle : forward

- Exemple de distribution de renvoi entre deux servlets.

- Envoyeur:

```
public class ExempleServlet extends HttpServlet {  
    protected void doGet(HttpServletRequest req, HttpServletResponse res)  
        throws ServletException, IOException {  
        req.setAttribute("X", 45);  
        RequestDispatcher rd=req.getRequestDispatcher("/exemple2?Y=6");  
        rd.forward(req, res);  
    }  
}
```

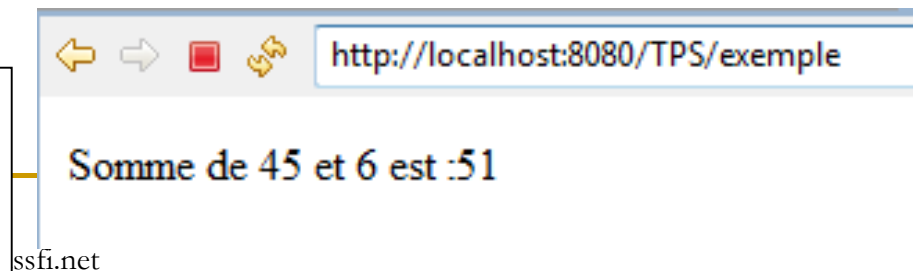
Transmission
d'informations par
attributs

- Récepteur:

```
public class ExempleServlet2 extends HttpServlet {  
    protected void doGet(HttpServletRequest req, HttpServletResponse res)  
        throws ServletException, IOException {  
        int x=(Integer)req.getAttribute("X");  
        int y=Integer.parseInt(req.getParameter("Y"));  
        PrintWriter out=res.getWriter();  
        out.print("Somme de "+x+" et "+y+" est :"+(x+y));  
    }  
}
```

Transmission
d'informations par
chaîne
d'interrogation

L'utilisation des attributs à la place des paramètres donne la possibilité de passer des objets et non des chaînes de caractères



Partage du contrôle : distribuer un renvoi

- Nous avons vu au début de cette partie qu'il existait une méthode de l'objet `response` qui permet de faire une redirection
 - `sendRedirect(...)` est une redirection effectuée par le client
 - `forward(...)` est une redirection effectuée par le serveur
 - Est-il préférable d'utiliser `forward(...)` ou `sendRedirect(...)` ???
 - `forward(...)` est à utiliser pour la partage de résultat avec un autre composant sur le même serveur
 - `sendRedirect(...)` est à utiliser pour des redirections externes car aucune recherche `getContext(...)` n'est nécessaire
 - **Préférez `forward(...)` pour des redirections dans le contexte et `sendRedirect(...)` pour le reste**
-

Partage du contrôle : distribuer un renvoi

- La méthode `include(...)` de l'interface `RequestDispatcher` inclut le contenu d'une ressource dans la réponse courante

```
RequestDispatcher dispat = req.getRequestDispatcher  
    ( "/index.html" );
```

```
dispat.include(req, res);
```

- La différence avec une distribution par renvoi est :
 - la Servlet appelante garde le contrôle de la réponse
 - elle peut inclure du contenu avant et après le contenu inclus
 - Possibilité de transmettre des informations lors de l'inclusion
 - en attachant une chaîne d'interrogation (au travers de l'URL)
 - en utilisant les attributs de requête via la méthode `setAttribute(...)`
 - Les choses à ne pas faire ...
 - ne pas définir le code d'état et en-têtes (pas de `setContentType(...)`)
-

Exemple de partage avec include

```
public class IncludeServlet extends HttpServlet {
protected void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    out.println("<html><body>");
    RequestDispatcher dispat = req.getRequestDispatcher("includedRessource");
    dispat.include(req,res);
    out.println("<br>");
    req.setAttribute("bonjour", "Bonjour");
    dispat.include(req,res);
    out.println("<br>");
    req.setAttribute("bonsoir", "Bonsoir");
    dispat.include(req,res);
    out.println("<br>");
    out.println("</BODY></HTML>");
}
}
```

Exemple de partage avec include.

La servlet incluse dans la précédente:

```
public class IncludedRessourceServlet extends HttpServlet {  
protected void doGet(HttpServletRequest req, HttpServletResponse res)  
throws ServletException, IOException {  
    PrintWriter out = res.getWriter();  
    if(req.getAttribute("bonjour") != null) {  
        out.println(req.getAttribute("bonjour"));  
    }  
    if (req.getAttribute("bonsoir") != null) {  
        out.println(req.getAttribute("bonsoir"));  
    } else {  
        out.println("Pas Bonsoir");  
    }  
    } else {  
        out.println("Rien de rien");  
    }  
}
```

Point de retour à l'appelant

← → ■ 💰 <http://localhost:8080/TPS/includer>

Rien de rien

Bonjour Pas Bonsoir

Bonjour Bonsoir

Java Server Pages : JSP

Éléments syntaxiques d'une JSP

- Une page JSP peut être formée par les éléments suivants :
 - ❑ Les expressions
 - ❑ Les déclarations
 - ❑ Les directives
 - ❑ Les scriptlets
 - ❑ Les actions
 - ❑ Les JSTL
-

Expressions

- Les expressions JSP sont, des expressions Java qui vont être évaluées à l'intérieur d'un appel de méthode *print*.
 - Une expression commence par les caractères `<%=` et se termine par les caractères `%>`.
 - Comme l'expression est placée dans un appel de méthode, il est interdit de terminer l'expression via un point-virgule.
 - Syntaxe: `<%=expression%>`
 - Equivalent à: `out.println(expression) ;`
 - Exemple : `<%=new Date()%>`
 - Equivalent à: `out.println(new Date()) ;`
-

Déclarations

- Dans certains cas, un peu complexe, il est nécessaire d'ajouter des méthodes et des attributs à la servlet qui va être générée (en dehors de la méthode de service).
- Une construction JSP particulière permet de répondre à ces besoins. Elle commence par les caractères `<%!` et se termine, par les caractères `%>`. Voici un petit exemple d'utilisation.
- Exemple:

```
<%@ page language="java" %>
```

```
<HTML>
```

```
  <%! private int userCounter = 0; %>
```

```
<BODY>
```

```
Vous êtes le <%= ++userCounter %><SUP>ième</SUP> client du  
site</P>
```

```
</BODY><HTML>
```

Directives

- Une directive permet de spécifier des informations qui vont servir à configurer et à influencer sur le code de la servlet générée.
 - Ce type de construction se repère facilement étant donné qu'une directive commence par les trois caractères `<% @`.
 - Notons principalement deux directives :
 - `<% @ page ... %>` et
 - `<% @ include ... %>`
 - Voyons de plus près quelques unes des possibilités qui vous sont offertes.
-

Directive `<%@ page .. %>`

- La directive `<%@ page .. %>` permet de pouvoir spécifier des informations utiles pour la génération et la compilation de la servlet.
 - En fait, cette directive accepte de nombreux paramètres dont les principaux sont:
 - ❑ `<%@ page language="java" %>`
 - ❑ `<%@ page import="package|classe" %>`
 - ❑ `<%@ page session="true|false" %>`
 - ❑ `<%@ page extends="classe" %>`
 - ❑ `<%@ page errorPage="url" %>`
 - ❑ `<%@ page isErrorPage="true|false" %>`
-

Directive `<%@ include .. %>`

- La directive `<% @ include ... %>` est très utile si plusieurs pages se doivent de partager une même ensemble d'information.
- C'est souvent le cas avec les entêtes et les pieds de pages. Dans ce cas, codez ces parties dans des fichiers séparés et injectez les, via cette directive, dans tous les autre fichiers qui en ont besoin.

- Voici un petit exemple d'utilisation de cette directive:

```
<%@ page language="java" %>
```

```
<HTML>
```

```
<BODY>
```

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

```
<!-- Contenu de la page à générer -->
```

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

```
</BODY><HTML>
```

Scriptlets

- Les scriptlets correspondent aux blocs de code introduit par le caractère `<%` et se terminant par `%>`.
- Ils servent à ajouter du code dans la méthode de service.
- Le code Java du scriptlet est inséré tel quel dans la servlet générée : la vérification, par le compilateur, du code aura lieu au moment de la compilation totale de la servlet équivalent.
- L'exemple complet de JSP présenté précédemment, comportait quelques scriptlets :

```
<%  
for(int i=1; i<=6; i++) {  
    out.println("<H" + i + " align=\"center\">Heading " +i+  
                "</H" + i + ">");  
}  
%>
```

Les actions

- Les actions constituent une autre façon de générer du code Java à partir d'une page JSP.
 - Les actions se reconnaissent facilement, syntaxiquement parlant : il s'agit de tag XML ressemblant à `<jsp:tagName ... />`.
 - Cela permet d'indiquer que le tag fait partie du namespace (espace de noms) *jsp*. Le nom du tag est préétabli.
 - Enfin, le tag peut, bien entendu comporter plusieurs attributs.
 - Il existe plusieurs actions différentes. Les principales sont les suivantes
-

Les actions

- `<jsp:include>` : Inclusion coté serveur
 - Exemple : `<jsp:include page= "entete.jsp" />`
- `<jsp:forward>` : Redirection vers une page
 - Exemple : `<jsp:forward page="affiche.jsp" />`
- `<jsp:useBean>` : Instanciation d'un objet java (java bean)
 - Exemple :
 - `<jsp:useBean id="jbName" class="TheClass" scope="session" />`
- `<jsp:setProperty>` : Cette action, permet de modifier une propriété sur un objet créé via l'action `<jsp:useBean ...>`
 - Exemple :
 - `<jsp:setProperty name="jbName" property="XXX" value="<%= javaExpression %>" />`
- `<jsp:getProperty>` : cette action est l'inverse de la précédente : elle permet de retourner dans le flux HTML, la valeur de la propriété considérée.
 - Exemple :
 - `<jsp:getProperty name="jbName" property="XXX" />`

JSTL :

- **Sun** a proposé une spécification pour une librairie de tags standard : la **Java Standard Tag Library (JSTL)**.
 - La **JSTL** est une implémentation de Sun qui décrit plusieurs actions basiques pour les applications web **J2EE**. Elle propose ainsi un ensemble de librairies de tags pour le développement de pages **JSP**.
 - Le but de la **JSTL** est de simplifier le travail des auteurs de page JSP, c'est à dire la personne responsable de la couche présentation d'une application web J2EE.
 - En effet, un web designer peut avoir des problèmes pour la conception de pages JSP du fait qu'il est confronté à un langage de script complexe qu'il ne maîtrise pas forcément.
-

Librairies de la JSTL1.1

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

Exemple de déclaration au début d'une JSP :

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

Les fichiers jars à inclure au classpath de votre projets :

- jstl-1.1.2.jar
- standard-1.1.2.jar

<c : / > : Librairie de base

■ Déclaration:

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

■ Gestion des variables de scope :

- Cette section comporte les actions de base pour la gestion des variables de scope d'une application web :
 - L'affichage de variable
 - La création/modification/suppression de variable de scope
 - La gestion des exceptions
-

<c : / > : Librairie de base

■ Afficher une expression <c:out/>

<!-- Afficher l'entête user-agent du navigateur ou "Inconnu" si il est absent : -->

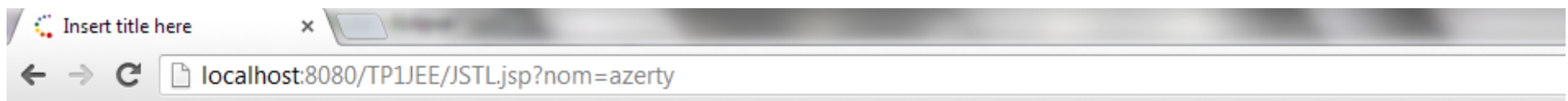
■ <c:out value="\${header['user-agent']}" default="Inconnu"/>

<!-- ou encore : -->

■ <c:out value="\${header['user-agent']}"> Inconnu </c:out>

<!-- Afficher le paramètre nom de la req http : -->

■ <c:out value="\${param['nom']}" default="Inconnu"/>



Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.31 (KHTML, like Gecko) Chrome/26.0.1410.43 Safari/537.31

azerty

<c : / > : Librairie de base

■ <c:if/> : Traitement conditionnel

- ❑ <!-- Afficher un message si le paramètre "page" de la requête HTTP est absent -->
- ❑ `<c:if test="$ {empty param['page']}"> paramètre absent !
</c:if>`

■ <c:chose/> : Traitement conditionnel exclusif

```
<c:choose>  
  <c:when test="$ {param['x'] == 1}">Premier</c:when>  
  <c:when test="$ {param['x'] == 2}">Deuxième</c:when>  
  <c:otherwise>Aucun</c:otherwise>  
</c:choose>
```

<c : / > : Librairie de base

- **<c:forEach/> : Itérer sur une collection :**
 - ❑ Permet d'effectuer simplement des itérations sur plusieurs types de collections de données.
 - ❑ L'attribut **items** accepte les éléments suivant comme collection :
 - Les tableaux d'objets ou de types primaires
 - Une implémentation de **java.util.Collection** en utilisant la méthode **iterator()**.
 - Une implémentation de **java.util.Iterator**.
 - Une implémentation de **java.util.Enumeration**.
 - Une implémentation de **java.util.Map**, en utilisant les méthodes **entrySet().iterator()**.
 - Une **String** dont les différents éléments sont séparés par des virgules (mais il est préférable d'utiliser <c:forTokens/> à sa place).
 - Une valeur **null** sera considérée comme une collection vide (pas d'itération).
 - Si l'attribut **items** est absent, les attributs **begin** et **end** permettent d'effectuer une itération entre deux nombres entiers.



<C : / > : Librairie de base

■ <c:forEach/> : Itérer sur une collection :

□ Exemples :

```
<!-- Afficher tous les éléments d'une collection dans le request-->
<c:forEach var="entry" items="${requestScope['myCollection']}" >
    ${entry}<br/>
</c:forEach>
```

```
<!-- Afficher seulement les 10 premiers éléments -->
<c:forEach var="entry" items="${requestScope['myCollection']}"
begin="0" end="9">
    ${entry}<br/>
</c:forEach>
```

```
<!-- Afficher les nombres de 1 à 10 -->
<c:forEach var="entry" begin="1" end="10">
    ${entry},
</c:forEach>
```

```
<!-- Afficher tous les paramètres de la requête et leurs valeurs -->
```

```
<c:forEach var="p" items="${param}">
    Le paramètre ${p.key} vaut ${p.value}<br/>
</c:forEach>
```

<c : / > : Librairie de base

- **<c:forTokens/> : Itérer sur des éléments d'une String :**
 - ❑ Permet de découper des chaînes de caractères selon un ou plusieurs délimiteurs. Chaque marqueur ainsi obtenu sera traité dans une boucle de l'itération.
 - ❑ Exemple :

```
<!-- Afficher séparément des mots séparés par un point-  
virgule -->  
  
<c:forTokens var="p" items="mot1;mot2;mot3;mot4" delims=";">  
    ${p}<br/>  
</c:forTokens>
```
-

<c : / > : Librairie de base

■ <c:param/> : Ajouter un paramètre à une URL

- ❑ Permet d'ajouter simplement un paramètre à une URL représentée par le tag parent.
- ❑ Cette balise doit avoir comme balise parent une balise <c:url/>, <c:import/> ou <c:redirect/> (mais pas forcément comme parent direct).
- ❑ Exemples:

<!-- La forme suivante : -->

```
<c:url value="/mapage.jsp?paramName=paramValue" />
```

<!-- est equivalente à : -->

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

```
  <c:param name="paramName" value="paramValue" />
```

```
</c:url>
```

<c : / > : Librairie de base

■ <c:url/> : Créer une URL

- Permet de créer des URLs absolues, relatives au contexte, ou relatives à un autre contexte.

- Exemple :

```
<!-- Création d'un lien dont les paramètres viennent d'une  
MAP -->
```

```
<c:url value="/index.jsp" var="variableURL">  
  <c:forEach items="{param}" var="p">  
    <c:param name="{p.key}" value="{p.value}"/>  
  </c:forEach>  
</c:url>  
<a href="{variableURL}">Mon Lien</a>
```

<c : / > : Librairie de base

■ <c:redirect/> : Redirection

- Envoi une commande de redirection HTTP au client.

- Exemple :

```
<!-- Redirection vers le portail de developpez.com : -->
```

```
<c:redirect url="http://www.developpez.com"/>
```

```
<!-- Redirection vers une page d'erreur avec des paramètres: -->
```

```
<c:redirect url="/error.jsp">
```

```
  <c:param name="from"
```

```
    value="${pageContext.request.requestURI}" />
```

```
</c:redirect>
```

<C : / > : Librairie de base

■ <c:import/> : Importer des ressources

- ❑ Permet d'importer une ressource selon son URL.
- ❑ Contrairement à <jsp:include/>, la ressource peut appartenir à un autre contexte ou être hébergée sur un autre serveur...
- ❑ Exemples :

```
<!-- Importer un fichier de l'application (comme <jsp:include/>) -->
```

```
<c:import url="/file.jsp">
```

```
    <c:param name="page" value="1"/>
```

```
</c:import>
```

```
<!-- Importer une ressource distante FTP dans une variable -->
```

```
<c:import url="ftp://server.com/path/file.ext" var="file" scope="page"/>
```

```
<!-- Importe une ressource distante dans un Reader -->
```

```
<c:url value="http://www.server.com/file.jsp" var="fileUrl">
```

```
    <c:param name="file" value="filename"/>
```

```
    <c:param name="page" value="1"/>
```

```
</c:url>
```

```
<!-- Ouverte d'un flux avec un Reader -->
```

```
<c:import url="${fileUrl}" varReader="reader">
```

```
</c:import>
```