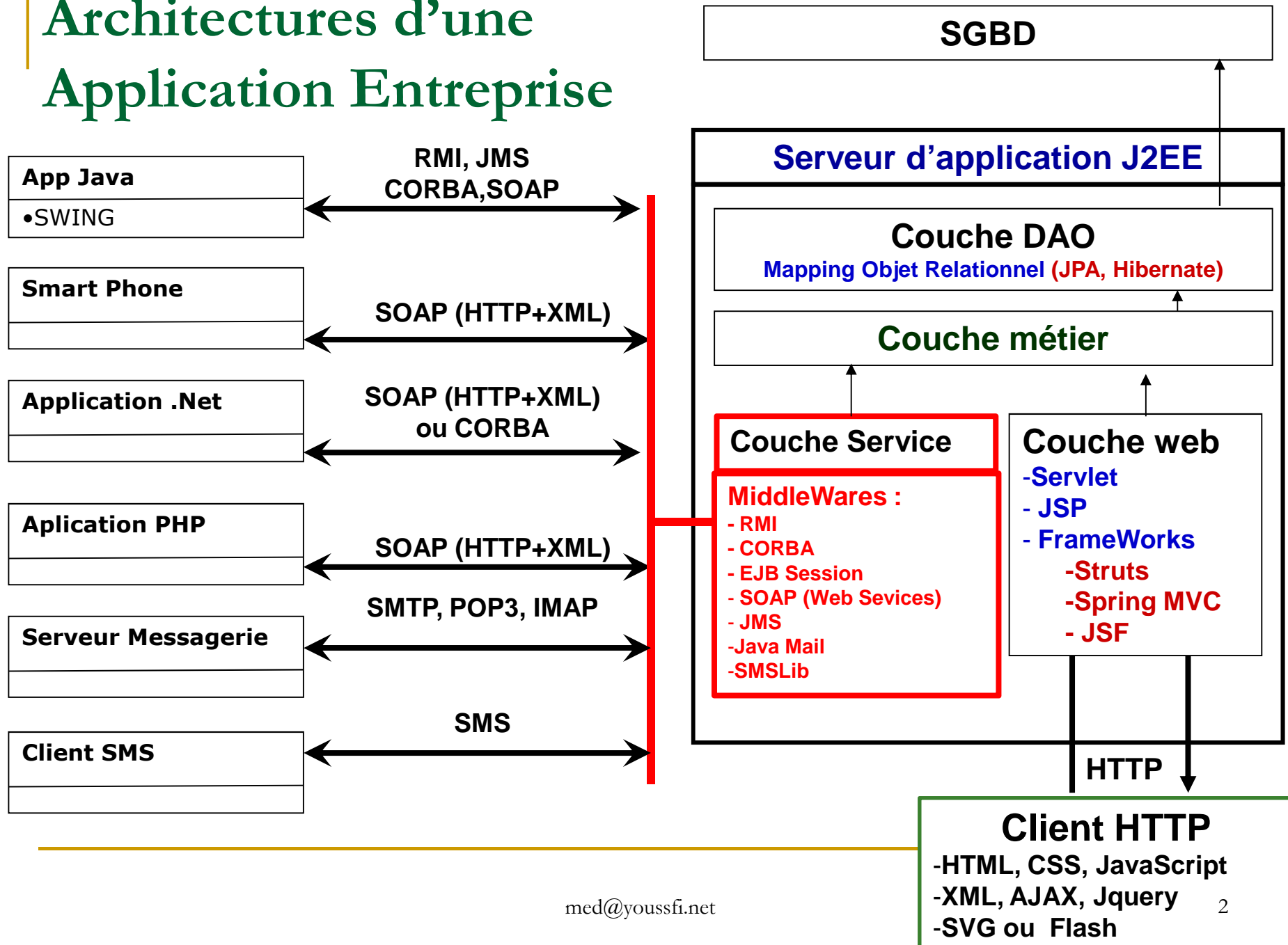


# Développement Web J2EE : Servlet, JSP, MVC,

Mohamed Youssfi

[med@yousfi.net](mailto:med@yousfi.net)

# Architectures d'une Application Entreprise



---

# Rappels :Qualité d'un Logiciel

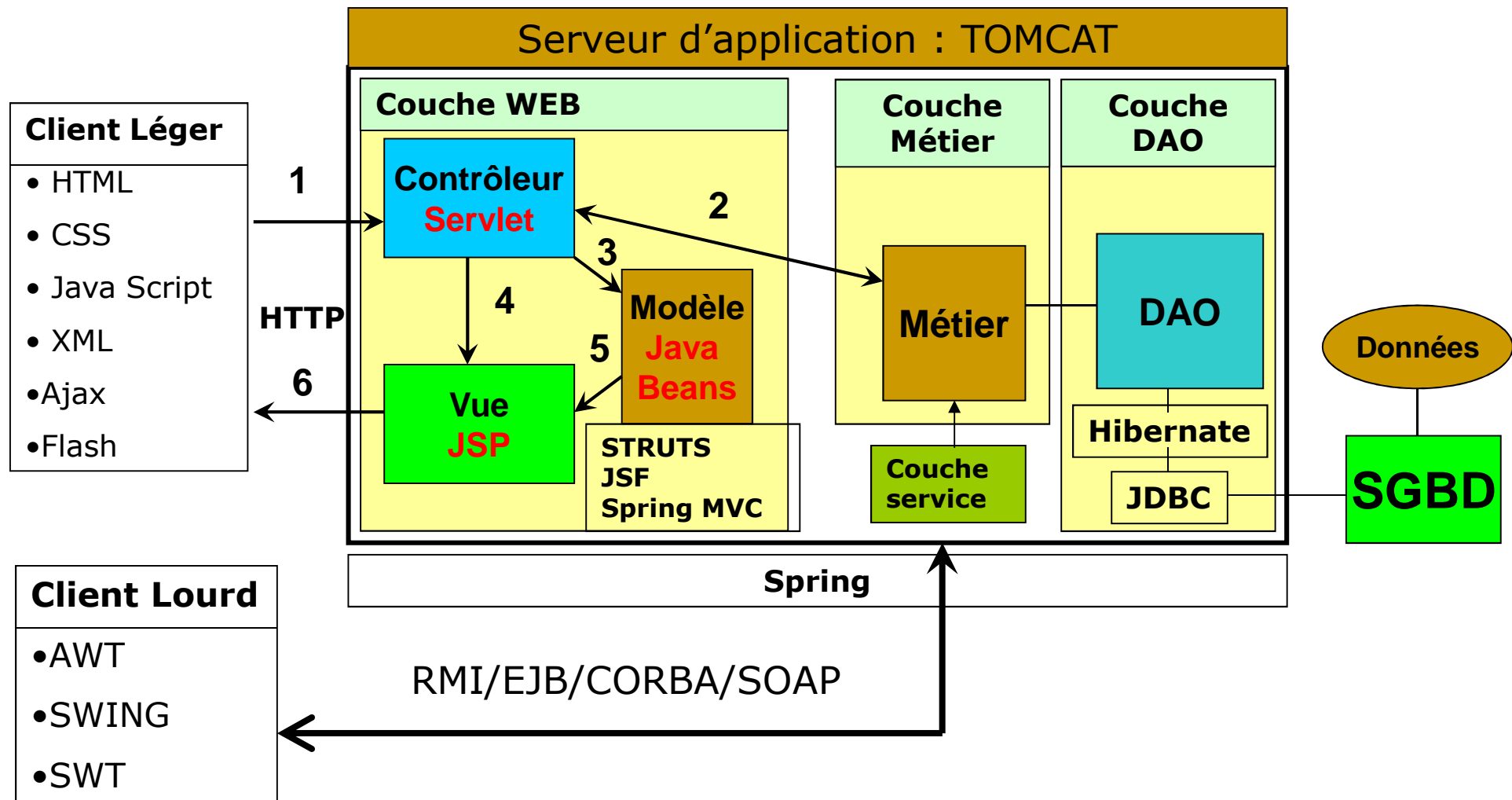
La qualité d'un logiciel se mesure par rapport à plusieurs critères :

- Répondre aux spécifications fonctionnelles :
  - ❑ Une application est créée pour répondre , tout d'abord, aux besoins fonctionnels des entreprises.
- Les performances:
  - ❑ La rapidité d'exécution et Le temps de réponse
  - ❑ Doit être bâtie sur une architecture robuste.
  - ❑ Eviter le problème de montée en charge
- La maintenance:
  - ❑ Une application doit évoluer dans le temps.
  - ❑ Doit être fermée à la modification et ouverte à l'extension
  - ❑ Une application qui n'évolue pas meurt.
  - ❑ Une application mal conçue est difficile à maintenir, par suite elle finit un jour à la poubelle.

# Qualité d'un Logiciel

- Sécurité
  - Garantir l'intégrité et la sécurité des données
- Portabilité
  - Doit être capable de s'exécuter dans différentes plateformes.
- Capacité de communiquer avec d'autres applications distantes.
- Disponibilité et tolérance aux pannes
- Capacité de fournir le service à différents type de clients :
  - Client lourd : Interfaces graphiques SWING
  - Interface Web : protocole http
  - Client SmartPhone
  - Téléphone : SMS
  - ....
- Design des ses interfaces graphiques
  - Charte graphique et charte de navigation
  - Accès via différentes interfaces (Web, Téléphone, PDA, ,)
- Coût du logiciel

# Architecture J2EE



---

# LE PROTOCOLE HTTP

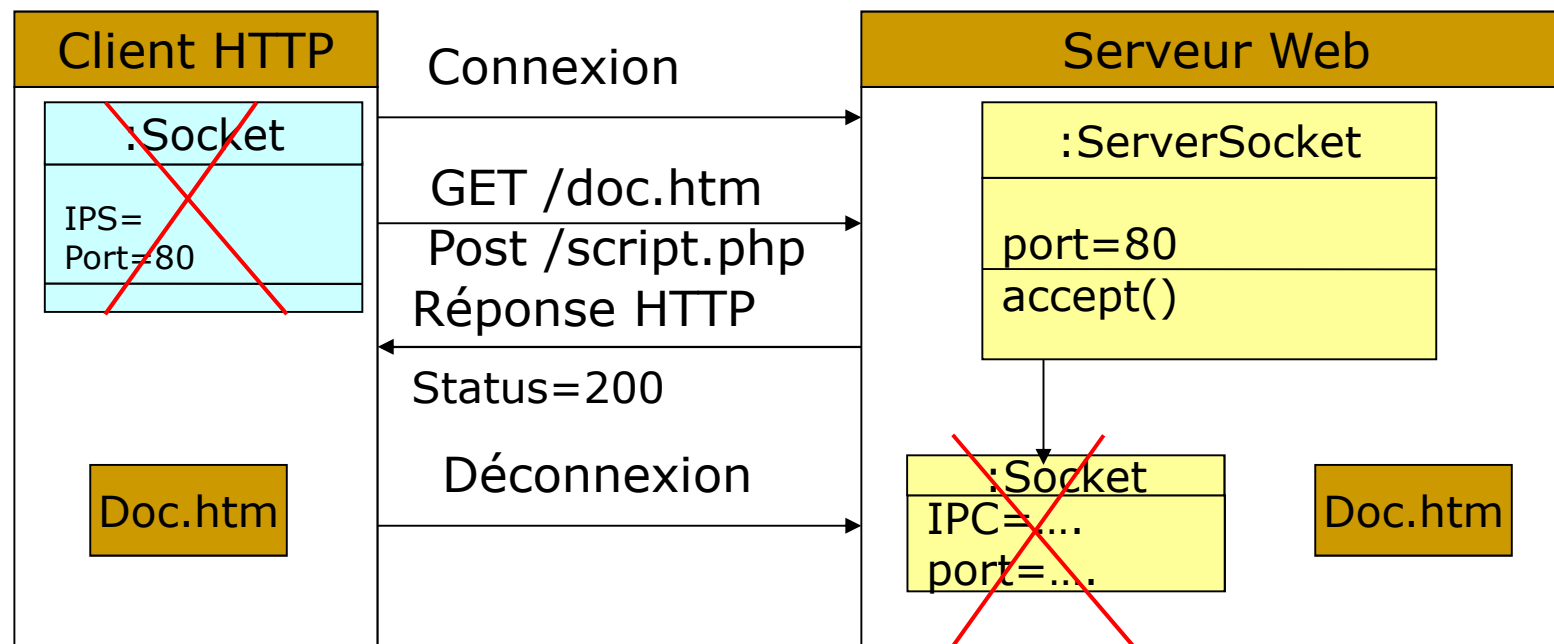
## ■ HTTP :HyperText Tranfert Protocol

- ❑ Protocole qui permet au client de récupérer des documents du serveur
- ❑ Ces documents peuvent être statiques (contenu qui ne change pas : HTML, PDF, Image, etc..) ou dynamiques ( Contenu généré dynamiquement au moment de la requête : PHP, JSP, ASP...)
- ❑ Ce protocole permet également de soumissionner les formulaires

## ■ Fonctionnement (très simple en HTTP/1.0)

- ❑ Le client se connecte au serveur (Créer une socket)
- ❑ Le client demande au serveur un document : Requête HTTP
- ❑ Le serveur renvoi au client le document (status=200) ou d'une erreur (status=404 quand le document n'existe pas)
- ❑ Déconnexion

# Connexion



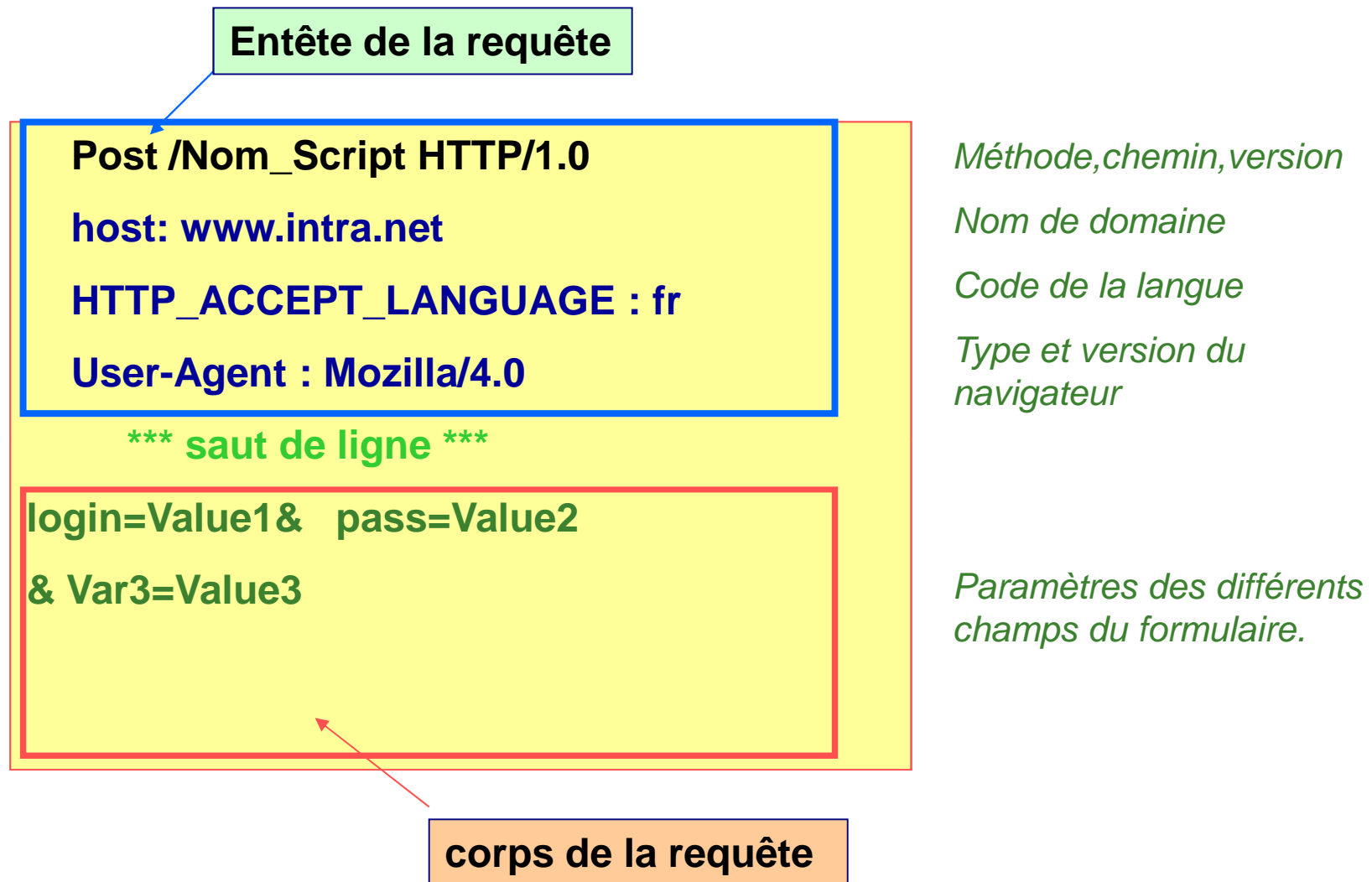
---

# Méthodes du protocole HTTP

- Une requête HTTP peut être envoyée en utilisant les méthodes suivantes:
  - ❑ **GET** : Pour récupérer le contenu d'un document
  - ❑ **POST** : Pour soumissionner des formulaires (Envoyer, dans la requête, des données saisies par l'utilisateur )
  - ❑ **PUT** pour envoyer un fichier du client vers le serveur
  - ❑ **DELETE** permet de demander au serveur de supprimer un document.
  - ❑ **HEAD** permet de récupérer les informations sur un document (Type, Capacité, Date de dernière modification etc...)



## Le client envoie la requête : Méthode POST



## Le client envoie la requête : Méthode GET

Entête de la requête

GET /Nom\_Script?login=val1&pass=val2&.... HTTP/1.0  
host: www.intra.net  
HTTP\_ACCEPT\_LANGUAGE : fr  
User-Agent : Mozilla/4.0

corps de la requête est vide

## Le Serveur retourne la réponse :

### Entête de la réponse

**HTTP/1.0 200 OK**

**Date : Wed, 05Feb02 15:02:01 GMT**

**Server : Apache/1.3.24**

**Last-Modified : Wed 02Oct01 24:05:01GMT**

**Content-Type : Text/html**

**Content-length : 4205**

*Ligne de Status*

*Date du serveur*

*Nom du Serveur*

*Dernière modification*

*Type de contenu*

*Sa taille*

**\*\*\* saut de ligne \*\*\***

**<HTML><HEAD>**

**....**

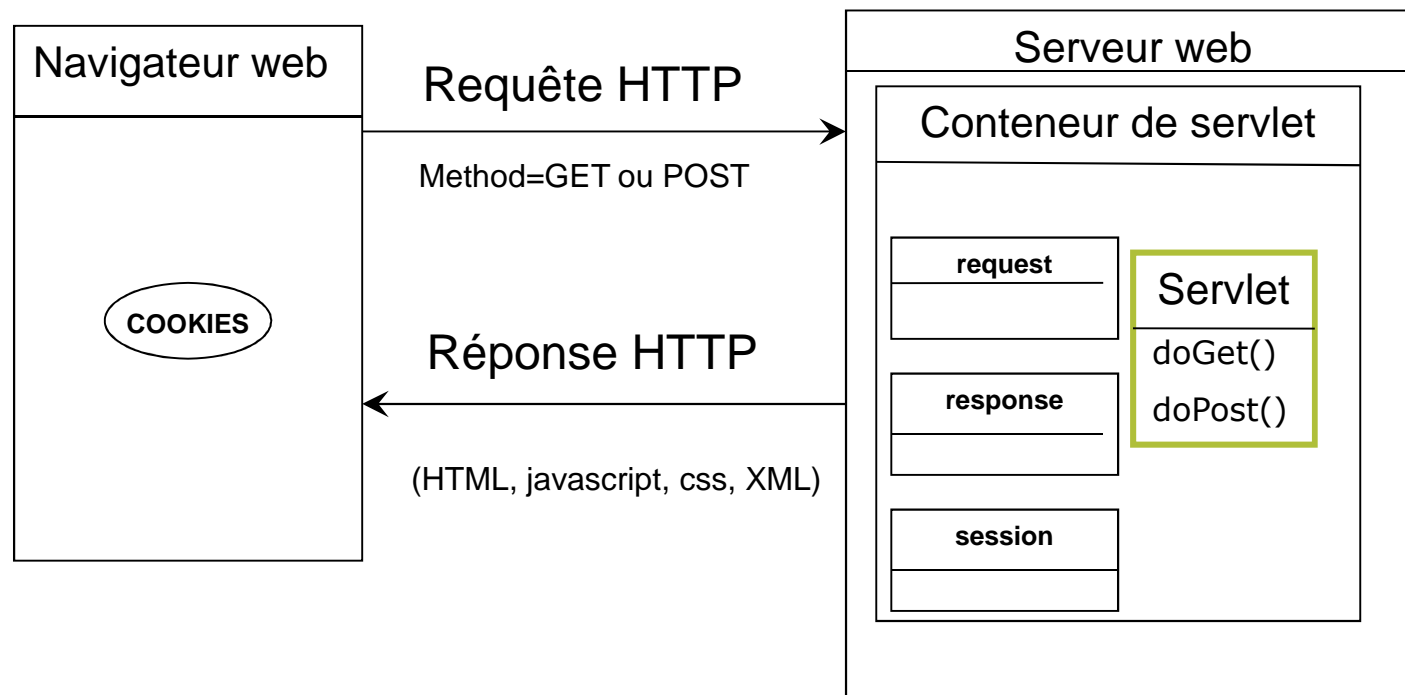
**</BODY></HTML>**

*Le fichier que le client  
va afficher*

med

**corps de la réponse**

# Introduction aux servlets



---

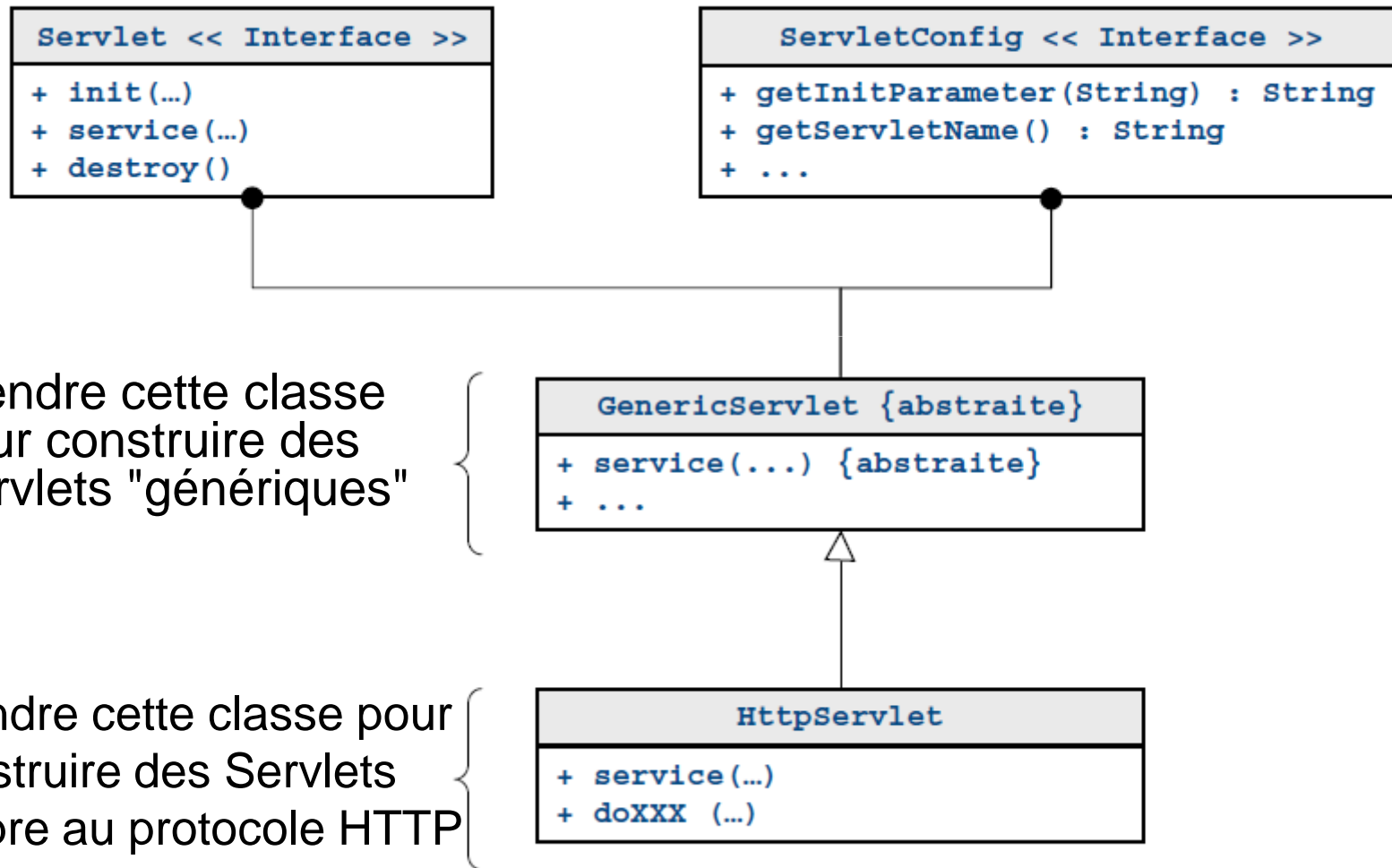
# Démo 1 (voir Vidéo Sevlet JSP MVC)

---

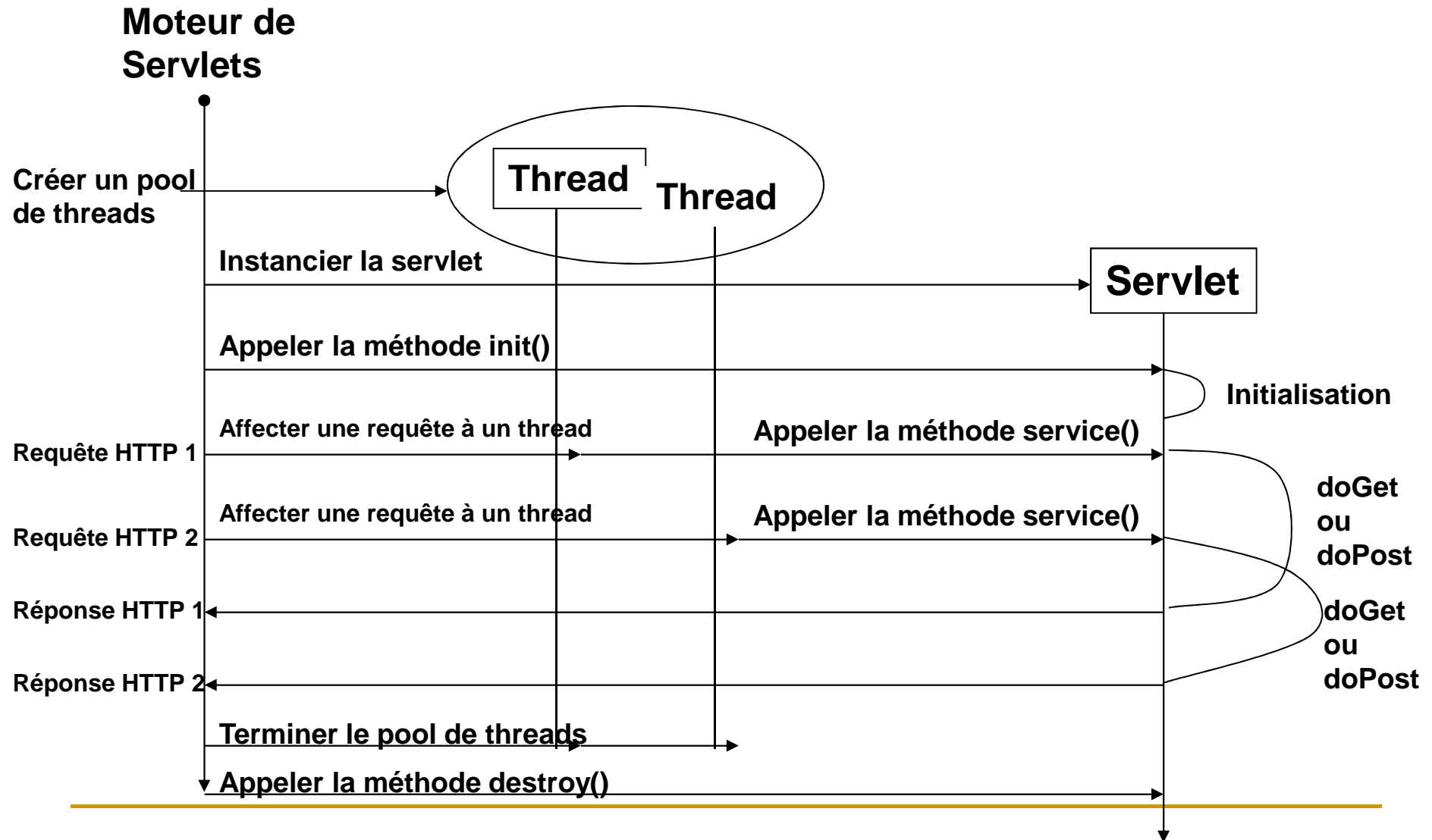
# Introduction aux servlets

- Composant logiciel écrit en Java fonctionnant du côté serveur
- Au même titre nous trouvons :
  - CGI (Common Gateway Interface)
  - Langages de script côté serveur PHP, ASP (Active Server Pages)
- Permet de gérer des requêtes HTTP et de fournir au client une réponse HTTP
- Une Servlet s'exécute dans un **moteur de Servlet** ou **conteneur de Servlet** permettant d'établir le lien entre la Servlet et le serveur Web

# L'API Servlet



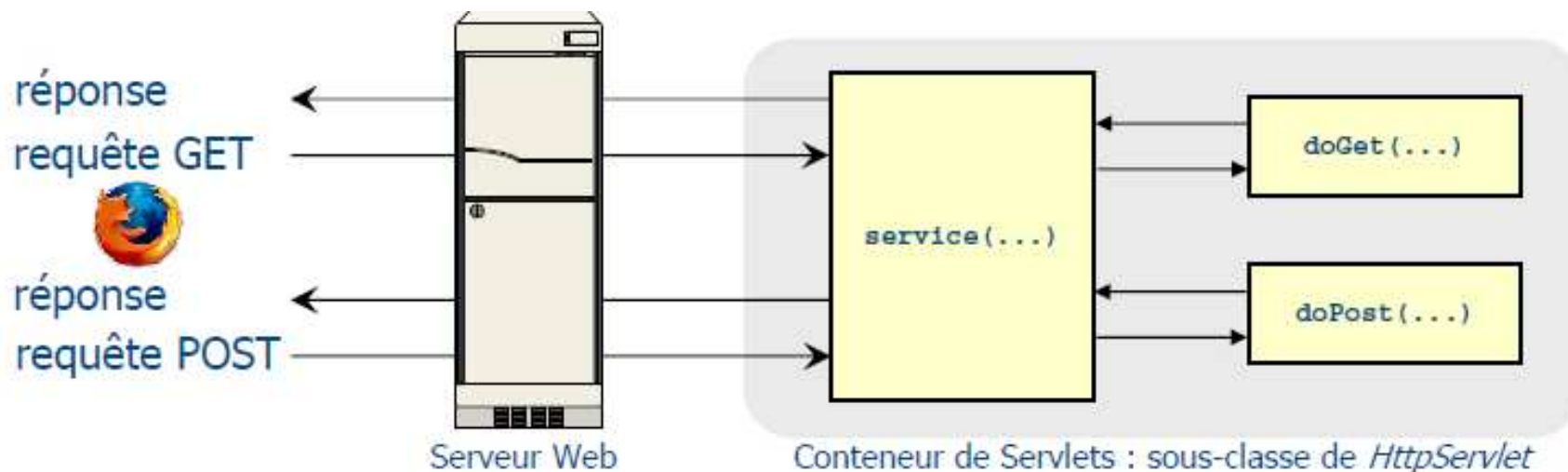
# Gestion des servlets





# HttpServlet

- Dans la suite du cours nous allons utiliser uniquement des Servlets qui réagissent au protocole HTTP d'où l'utilisation de la classe HttpServlet
- HttpServlet redéfinit la méthode `service(...)`
- `service(...)` lit la méthode (GET, POST, ...) à partir de la requête
- Elle transmet la requête à une méthode appropriée de HttpServlet destinée à traiter le type de requête (GET, POST, ...)



---

# Fonctionnement d'une servlet

- Lorsqu'une servlet est appelée par un client, la méthode *service()* est exécutée. Celle-ci est le principal point d'entrée de toute servlet et accepte deux objets en paramètres:
  - L'objet *HttpServletRequest* encapsulant la requête du client, c'est-à-dire qu'il contient l'ensemble des paramètres passés à la servlet (informations sur l'environnement du client, cookies du client, URL demandée, ...)
  - L'objet *HttpServletResponse* permettant de renvoyer une réponse au client (envoyer des informations au navigateur).

# Développement d'une servlet

- Une servlet est une classe qui hérite de la classe `HttpServlet` et qui redéfinit les méthodes du protocole HTTP.
- Si la méthode utilisée est GET, il suffit de redéfinir la méthode :
  - `public void doGet(  
    HttpServletRequest request,  
    HttpServletResponse response  
)`
- Si la méthode utilisée est POST, il suffit de redéfinir la méthode :
  - `public void doPost(  
    HttpServletRequest request,  
    HttpServletResponse response  
)`

# Première Servlet

```
package web;
```

```
import java.io.*;
```

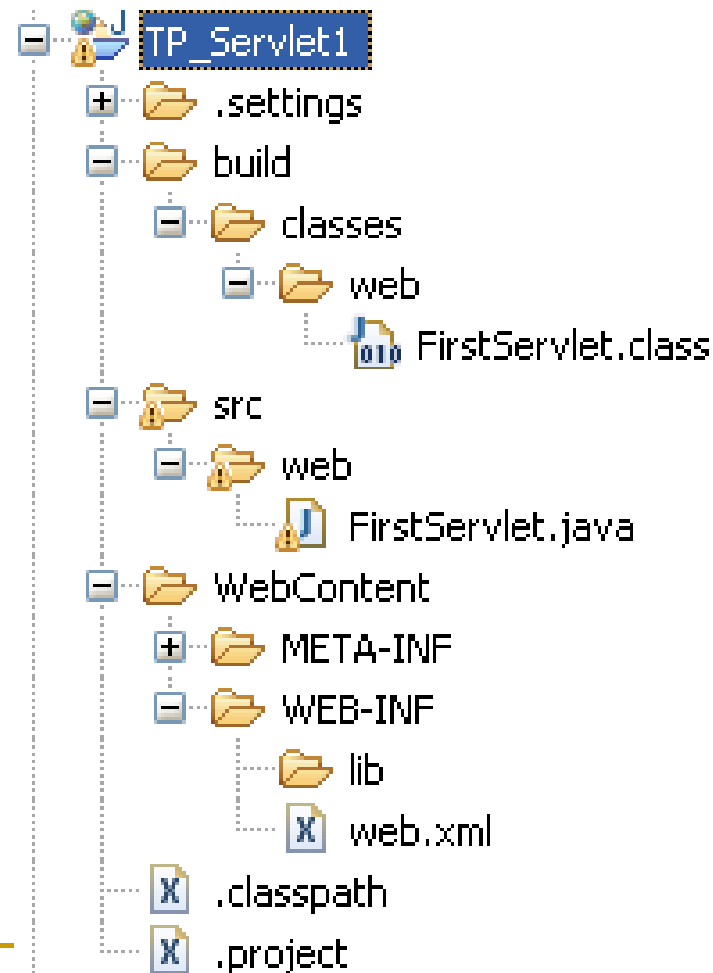
```
import javax.servlet.ServletException;
```

```
import javax.servlet.http.*;
```

```
public class FirstServlet extends HttpServlet {  
    protected void doGet(HttpServletRequest request,  
        HttpServletResponse response) throws  
        ServletException, IOException {  
        response.setContentType("text/html");  
        PrintWriter out = response.getWriter();  
        out.println("<HTML>");  
        out.println("<HEAD><TITLE> Titre </TITLE></HEAD>");  
        out.println("<BODY>");  
        out.println("Ma première servlet");  
        out.println("</BODY>");  
        out.println("</HTML>");  
        out.close();  
    }  
}
```

# Structure d'un projet Web J2EE

- Le dossier src contient les classes java
- Le byte code est placé dans le dossier build/classes
- Les dossier WebContent contient les documents Web comme les pages HTML, JSP, Images, Java Script, CSS ...
- Le dossier WEB-INF contient les descripteurs de déploiement comme web.xml
- Le dossier lib permet de stocker les bibliothèques de classes java (Fichiers.jar)



---

# Déploiement d'une servlet

- Pour que le serveur Tomcat reconnaisse une servlet, celle-ci doit être déclarée dans le fichier web.xml qui se trouve dans le dossier WEB-INF.
- Le fichier web.xml s'appelle le descripteur de déploiement de Servlet.
- Ce descripteur doit déclarer principalement les éléments suivant :
  - ❑ Le nom attribué à cette servlet
  - ❑ La classe de la servlet
  - ❑ Le nom URL à utiliser pour faire appel à cette servlet via le protocole HTTP.

# web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<web-app id="WebApp_ID" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"  
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee  
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
```

```
<display-name>TP_Servlet1</display-name>
```

Balise de description de  
l'application WEB

```
<servlet>
```

```
<servlet-name>FirstServlet</servlet-name>
```

Nom de la Servlet  
"Identification"

```
<servlet-class>web.FirstServlet</servlet-class>
```

Classe de la Servlet

```
</servlet>
```

```
<servlet-mapping>
```

Définition d'un chemin  
virtuel

```
<servlet-name>FirstServlet</servlet-name>
```

```
<url-pattern>/fs</url-pattern>
```

Nom de la Servlet considér  
"Identification"

```
</servlet-mapping>
```

URL associée à la servlet

```
</web-app>
```

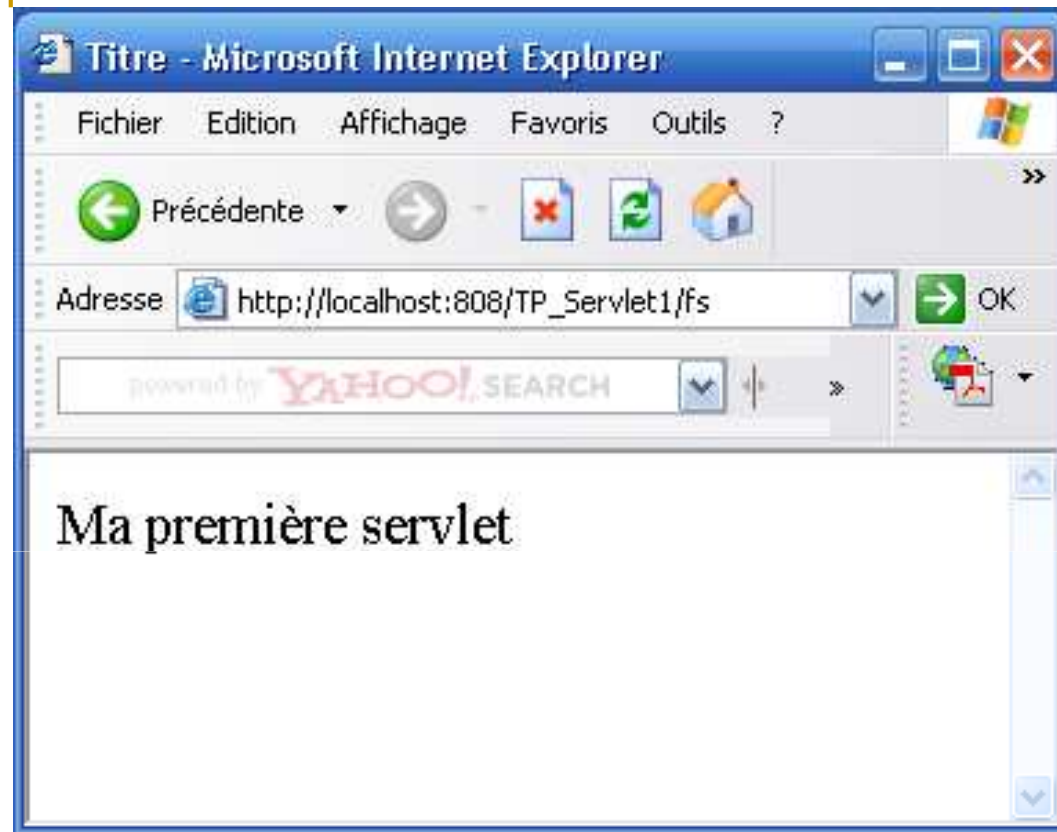
# Servlet 3.0

- Pour un projet web J2EE, utilisant un module web, version 3.0, le fichier web.xml n'est pas nécessaire.
- Dans ce cas, le déploiement d'une servlet peut se faire en utilisant des annotations:

```
package web;  
import java.io.*; import javax.servlet.*;  
import javax.servlet.annotation.*;  
import javax.servlet.http.*;  
@WebServlet(name="cs",urlPatterns={"/fs","*.do"})  
public class FirstServlet extends HttpServlet {  
  
}
```



# Tester la servlet



A screenshot of a Notepad window titled "fs[1] - Bloc-notes". The menu bar includes "Fichier", "Edition", "Format", "Affichage", and "?". The text content is as follows:

```
<HTML>
<HEAD><TITLE> Titre </TITLE></HEAD>
<BODY>
Ma première servlet
</BODY>
</HTML>
```

Code source coté client

---

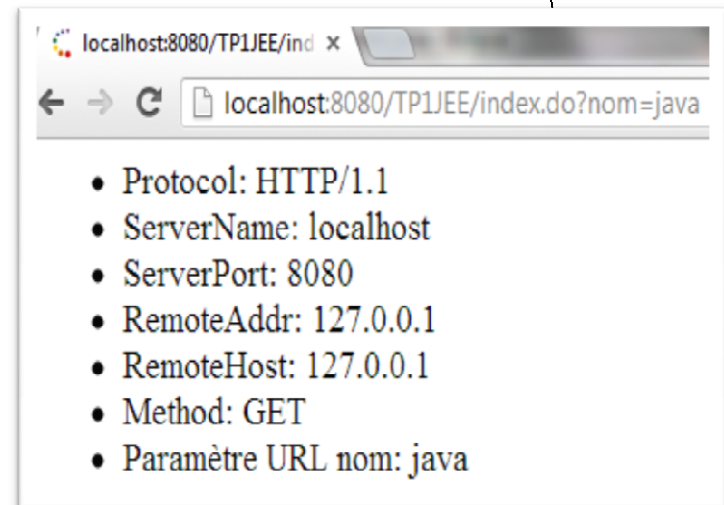
# HttpServletRequest

- HttpServletRequest hérite de ServletRequest
- Cet objet encapsule la requête HTTP et fournit des méthodes pour accéder
  - aux informations du client
  - à l'environnement du serveur
- Exemples de méthodes
  - `String getMethod()` : retourne le type de requête
  - `String getServerName()` : retourne le nom du serveur
  - `String getParameter(String name)` : retourne la valeur d'un paramètre
  - `String[] getParameterNames()` : retourne le nom des les paramètres
  - `String getRemoteHost()` : retourne l'IP du client
  - `String getServerPort()` : retourne le port sur lequel le serveur écoute
  - `String getQueryString()` : retourne la chaîne d'interrogation
  - ... (voir l'API Servlets pour le reste)

# HttpServletRequest

## ■ Exemple : Quelques méthodes de l'objet request

```
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 {
    PrintWriter out=response.getWriter();
    response.setContentType("text/html");
    out.println("<html><body><ul>");
    out.println("<li>Protocol: " + request.getProtocol()+"</li>");
    out.println("<li>ServerName: " + request.getServerName()+"</li>");
    out.println("<li>ServerPort: " + request.getServerPort()+"</li>");
    out.println("<li>RemoteAddr: " + request.getRemoteAddr()+"</li>");
    out.println("<li>RemoteHost: " + request.getRemoteHost()+"</li>");
    out.println("<li>Method: " + request.getMethod()+"</li>");
    out.println("<li>Paramètre URL nom: " + request.getParameter("nom")+"</li>");
    out.println("</ul></body></html>");
}
}
```



# Une page JSP Equivalente : Exemple1.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
```

```
<!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>Insert title here</title>
```

```
</head>
```

```
<body>
```

```
<ul>
```

```
<li>Protocol:<% out.println( request.getProtocol()); %>
```

```
<li>ServerName:<%=request.getServerName()%></li>
```

```
<li>ServerPort:<%=request.getServerPort() %></li>
```

```
<li>RemoteAddr:<%=request.getRemoteAddr() %></li>
```

```
<li>RemoteHost:<%=request.getRemoteHost() %></li>
```

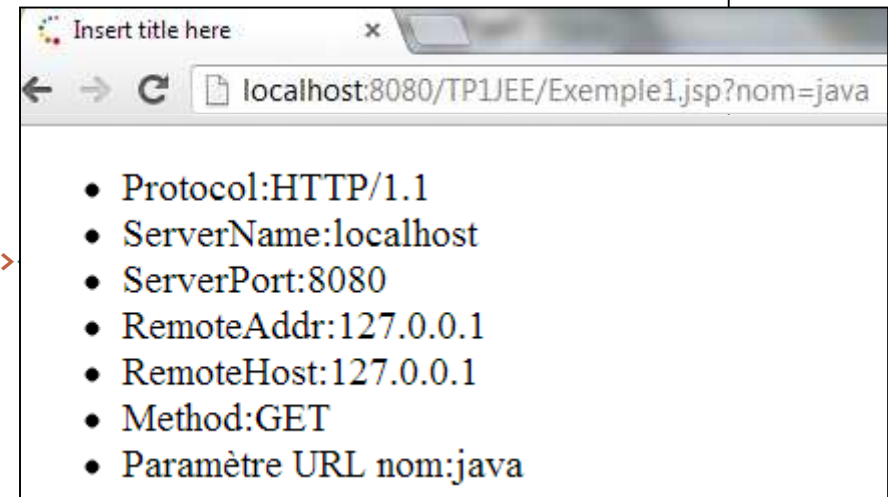
```
<li>Method:<%=request.getMethod() %></li>
```

```
<li>Paramètre URL nom:<%=request.getParameter("nom")%></li>
```

```
</ul>
```

```
</body>
```

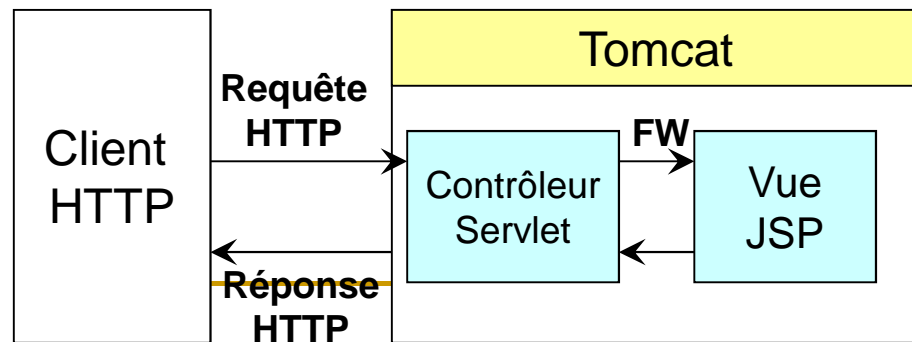
```
</html>
```



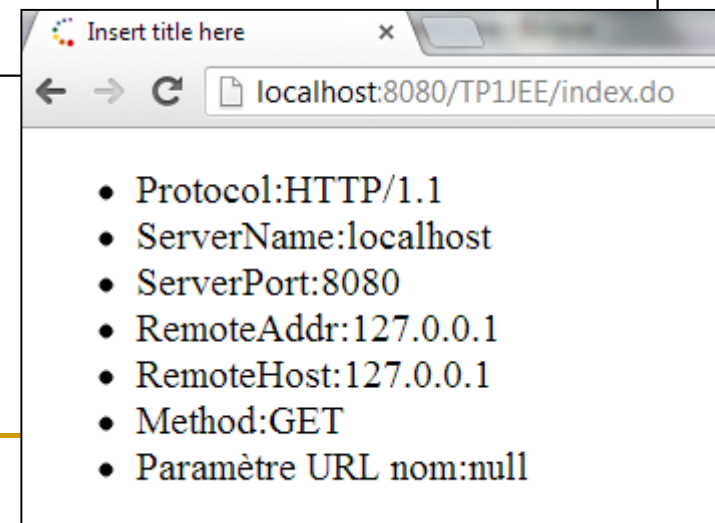
# Forwarding

- Pour séparer les rôles une servlet peut faire un forward vers une JSP de la manière suivante :

```
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 {  
        request.getRequestDispatcher("Exemple1.jsp").forward(request, response);  
    }  
}
```



med@youssefi.net



# Première comparaison entre Servlet et JSP

- Une servlet est une classe java dans laquelle on peut générer du code HTML alors qu'une JSP est une sorte de page HTML, à l'intérieur de laquelle, on peut écrire du code Java.
- Les pages JSP sont très pratiques quand on veut afficher les vues de l'application alors que les servlets sont pratiques pour effectuer les traitements nécessaires au fonctionnement d'une application.
- Si une servlet nécessite d'être déployée (web.xml), une JSP est déployée automatiquement par Tomcat.
- Pendant le premier appel d'une JSP, Tomcat convertit la JSP en servlet et la déploie automatiquement.
- Quand un client HTTP demande une page JSP, c'est la servlet correspondante à cette JSP, qui est générée par Tomcat qui est exécutée.
- Tout ce qu'on peut faire avec une servlet, peut être fait par une JSP. (Une JSP est une servlet)
- La technologie de base des applications web J2EE c'est les servlets.
- Dans une application web J2EE qui respecte le pattern MVC,
  - Les servlets sont utilisées pour jouer le rôle du contrôleur
  - Les JSP sont utilisées pour jouer le rôle des vues

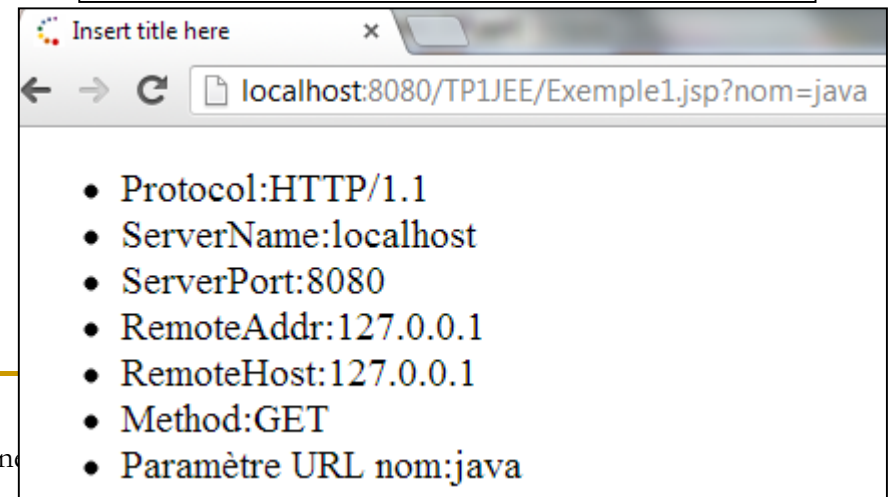
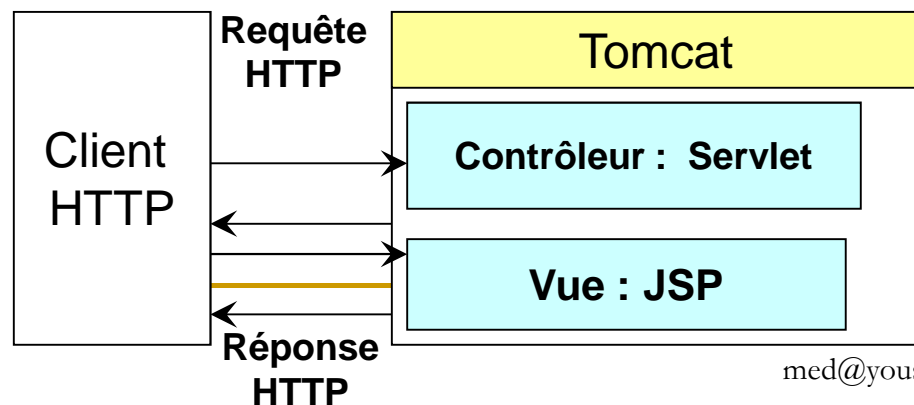
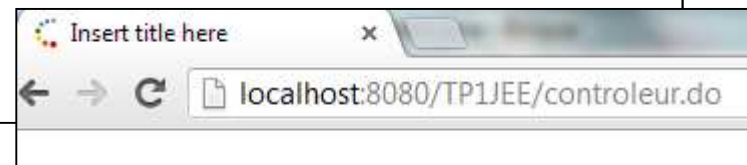
# HttpServletResponse

- HttpServletResponse hérite de ServletResponse
  - Cet objet est utilisé pour construire un message de réponse HTTP renvoyé au client,
  - il contient les méthodes nécessaires pour définir le type de contenu, en-tête et code de retour
  - un flot de sortie pour envoyer des données (par exemple HTML) au client
  - Exemples de méthodes :
    - ❑ `void setStatus(int)` : définit le code de retour de la réponse
    - ❑ `void setContentType(String)` : définit le type de contenu MIME
    - ❑ `PrintWriter getWriter()` : Retourne un objet `PrintWriter` permettant d'envoyer du texte au navigateur client. Il se charge de convertir au format approprié les caractères Unicode utilisés par Java
    - ❑ `ServletOutputStream getOutputStream()` : flot pour envoyer des données binaires au client
- 
- ❑ `void sendRedirect(String)` : redirige le navigateur vers l'URL

# HttpServletResponse : Redirection

- Une servlet peut rediriger vers une autre ressource locale ou distante en utilisant la méthode `sendRedirect()` de l'objet `response`.

```
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 {  
        response.sendRedirect("Vue.jsp");  
    }  
}
```



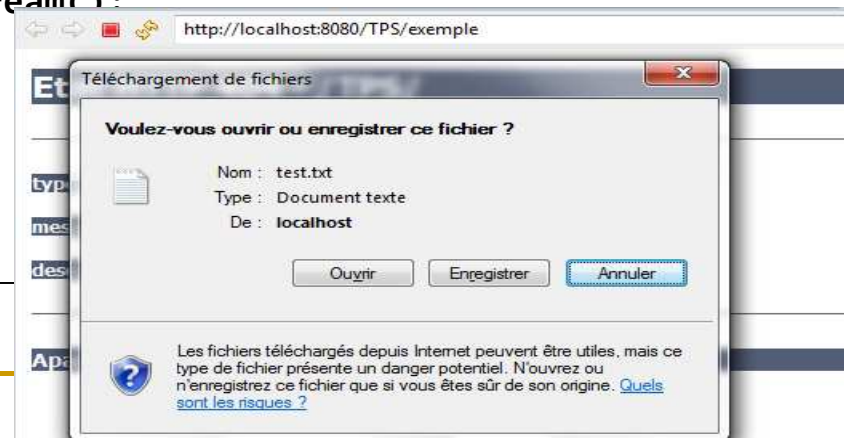


# Exemple : Téléchargement de fichier par le client http

```
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 {
        File f=new File("C:/BP/TP1JEE/Tableau de correspondances.doc");
        FileInputStream fis=new FileInputStream(f);
        response.setHeader("Content-Disposition","attachment;filename=a.doc");
        byte[] data=new byte[(int)f.length()];
        fis.read(data);
        OutputStream os=response.getOutputStream();
        os.write(data);
        os.close();fis.close();
    }
}
```



# Exemple : effectue un pull client

```
package web;

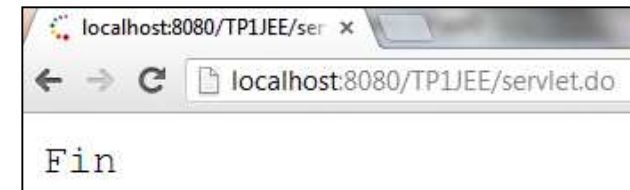
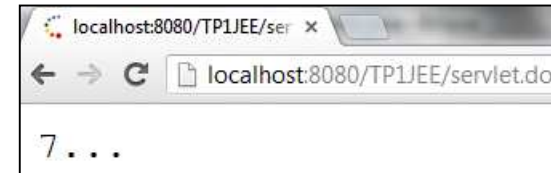
import java.io.*; import javax.servlet.*;import javax.servlet.http.*;

public class FirstServlet extends HttpServlet {

    private int compteur;

    @Override
    public void init() throws ServletException {
        compteur=9;
    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/plain");
        PrintWriter out = response.getWriter();
        if (compteur > 0) {
            response.setHeader("Refresh","1");
            --compteur;
            out.println(compteur + "...");
        } else {
            out.println("Fin");
        }
    }
}
```



Toutes les 1 seconde  
la page est rechargée  
et cela 9 fois de suite

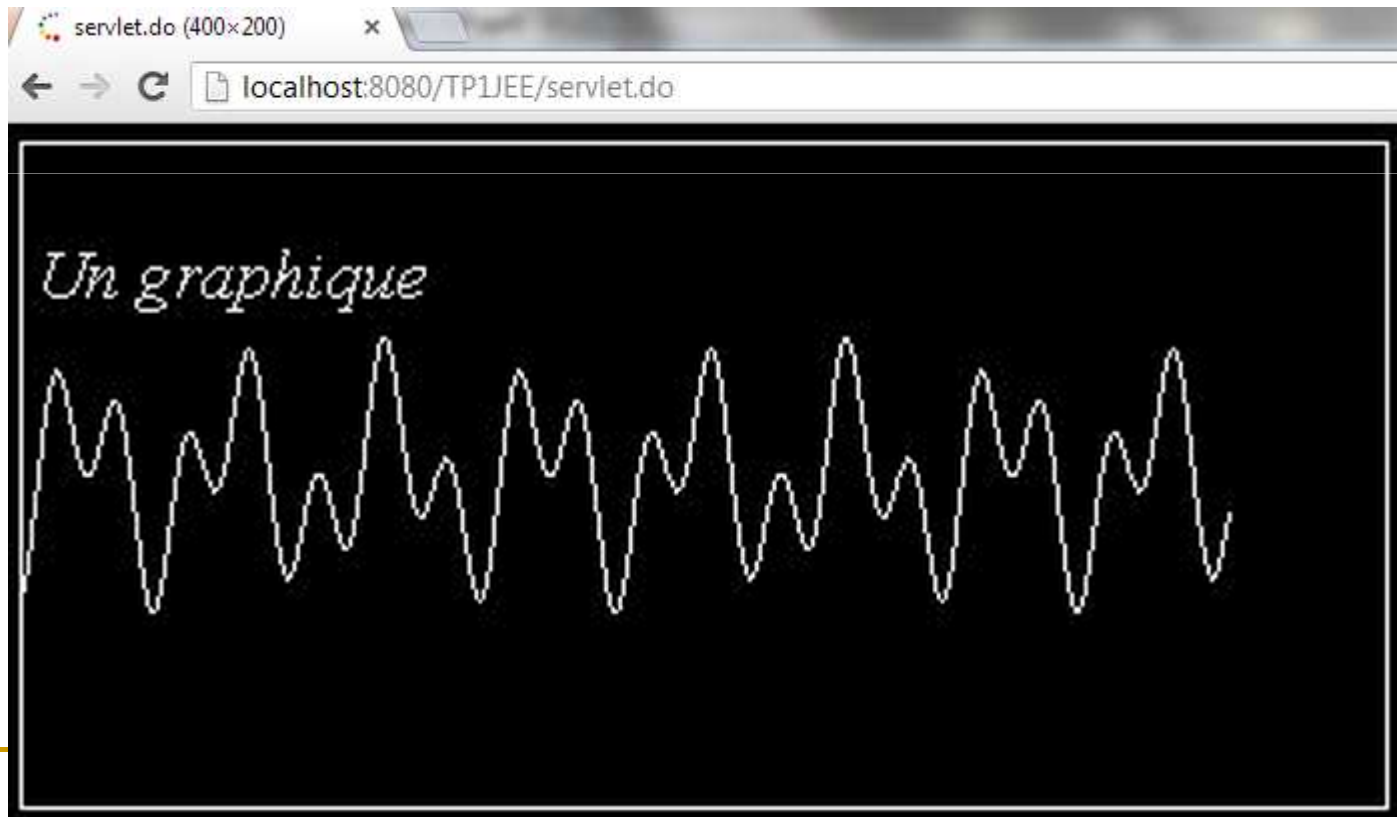
---

# Envoyer du contenu multimédia

- Pour l'instant nous avons écrit des Servlets qui retournaient des contenus HTML
- Parfois, on a besoin de retourner des contenus différents :
  - Contenu XML
  - Génération de contenus multimédias (création de graphes, manipulation d'images)
- L'API Java facilite la gestion des contenus multimédias en proposant des bibliothèques
  - Encodage d'images sous différents formats (GIF, JPEG)
  - Manipulation et traitement d'images

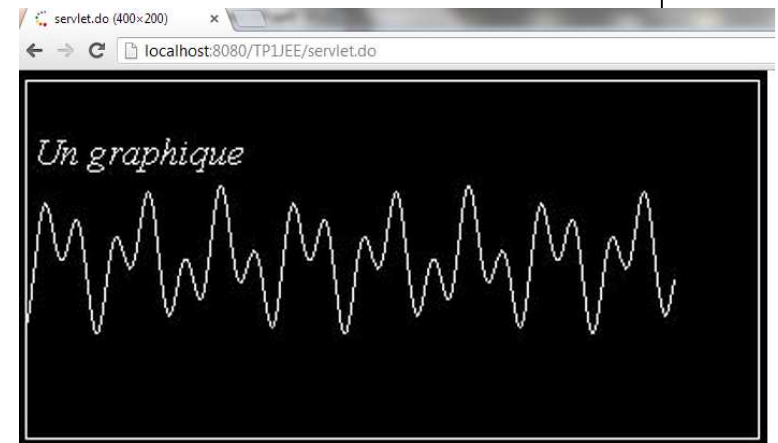
# Envoyer du contenu multimédia

- Exemple : Servlet qui génère et retourne une image JPEG contenant un graphique

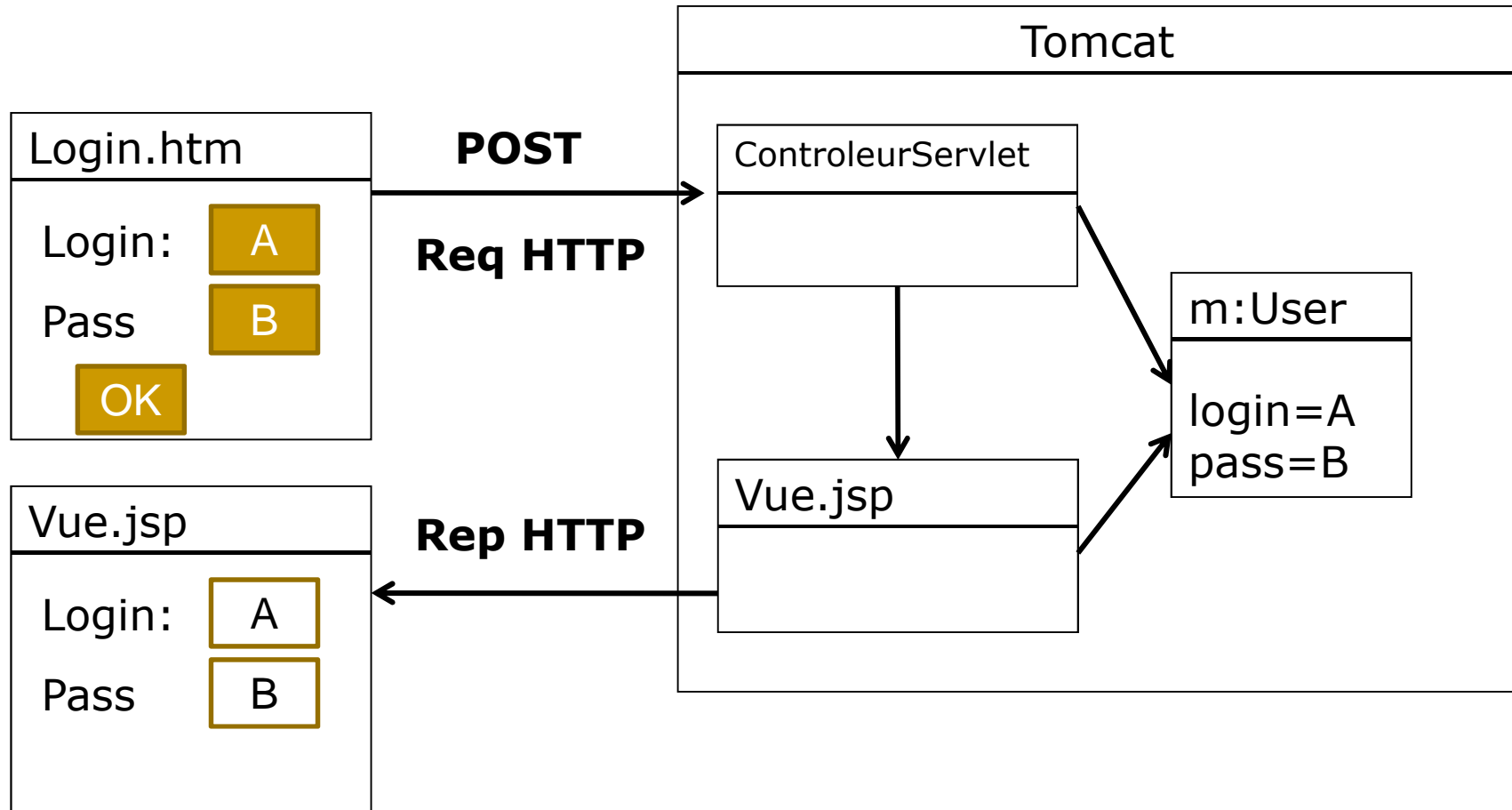


# Envoyer du contenu multimédia

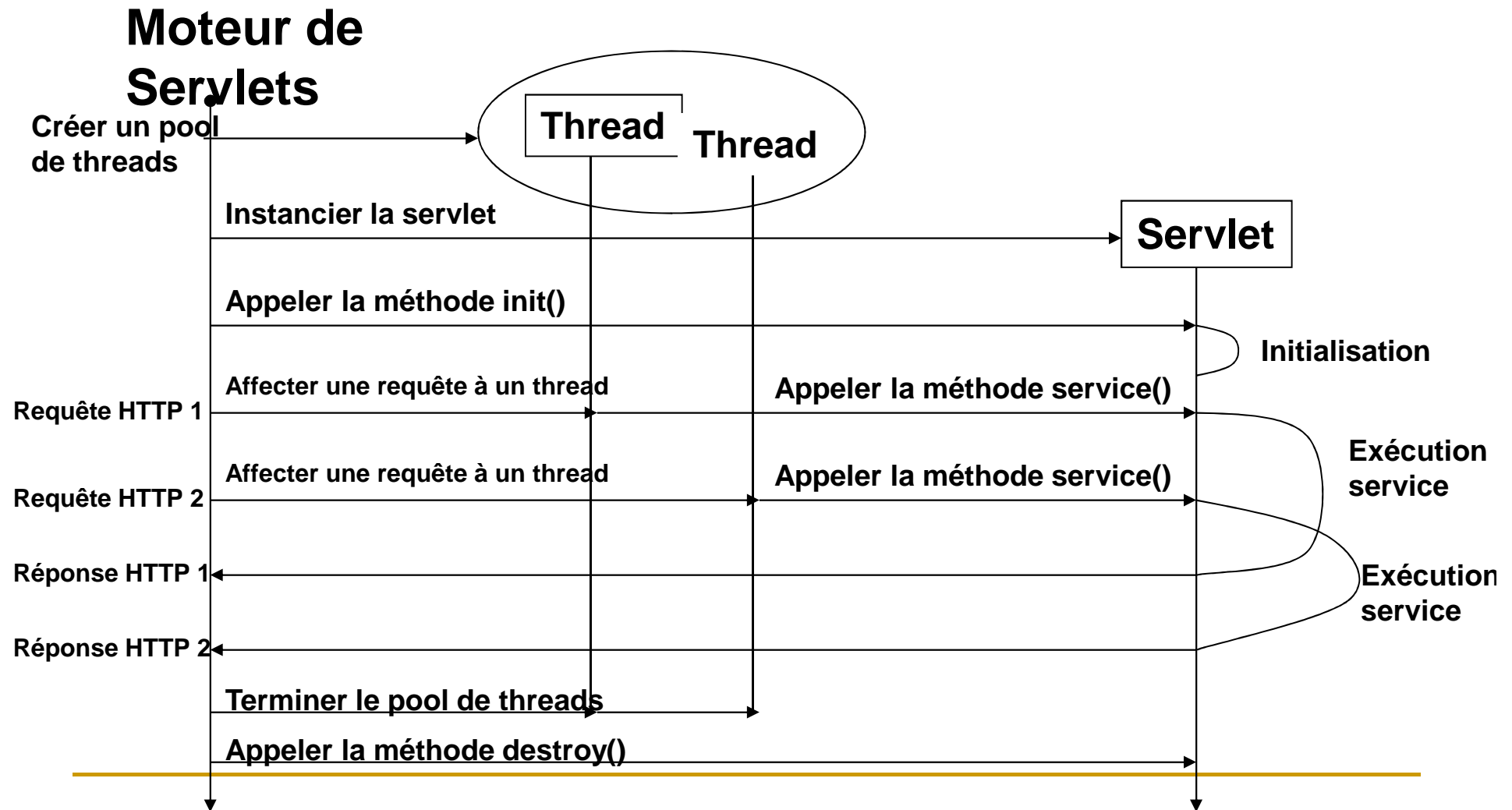
```
package web; import com.sun.image.codec.jpeg.*;
import java.awt.*;import java.awt.image.BufferedImage;
import java.io.*; import javax.servlet.*;import javax.servlet.http.*;
public class FirstServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    ServletOutputStream out = response.getOutputStream();
    BufferedImage bim= new BufferedImage(400, 200, BufferedImage.TYPE_3BYTE_BGR);
    Graphics2D g = bim.createGraphics(); g.setFont(new Font("Serif", Font.ITALIC, 20));
    g.drawString("Un graphique", 10,50); g.drawRect(5, 5, 390, 190);
    int echelle=20;
    for(int i=5;i<350;i++){
        int x1=i;int y1= 100+(int)(f(x1)*echelle); int x2=i+1;int y2=100+ (int)(f(x2)*echelle);
        g.drawLine(x1,y1, x2,y2);
    }
    JPEGImageEncoder encode = JPEGCodec.createJPEGEncoder(out);
    encode.encode(bim); out.close();
}
private double f(double x){
    return Math.cos(x/7)+Math.sin(x/3);
}}
```



# Application : Voir vidéo 1



# Gestion des servlets



---

# Cycle de vie d'une servlet

- Le serveur crée un pool de threads auxquels il va pouvoir affecter chaque requête
- La servlet est chargée au démarrage du serveur ou lors de la première requête
- La servlet est instanciée par le serveur
- La méthode *init()* est invoquée par le conteneur
- Lors de la première requête, le conteneur crée les objets *Request* et *Response* spécifiques à la requête
- La méthode *service()* est appelée à chaque requête dans un nouveau thread. Les objets *Request* et *Response* lui sont passés en paramètre
- Grâce à l'objet *request*, la méthode *service()* va pouvoir analyser les informations en provenance du client
- Grâce à l'objet *response*, la méthode *service()* va fournir une réponse au client
- La méthode *destroy()* est appelée lors du déchargement de la servlet, c'est-à-dire lorsqu'elle n'est plus requise par le serveur. La servlet est alors signalée au *garbage collector*.



---

# Cycle de vie d'une servlet

- A chaque rechargement d'une Servlet par le conteneur de
- Servlet, il y a **création d'une nouvelle instance** et donc destruction de l'ancienne
- Le rechargement d'une Servlet a lieu quand il y a :
  - ❑ Modification d'au moins une **classe** de l'application WEB
  - ❑ Demande explicite de l'administrateur du serveur WEB
  - ❑ Redémarrage du conteneur de Servlets

# Paramétrer une servlet

■ Une servlet peut être paramétrée dans le fichier web.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" xmlns:web="http://java.sun.com/xml/ns/javaee/web-
app_2_5.xsd" 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">

  <servlet>
    <servlet-name>cs</servlet-name>
    <servlet-class>web.FirstServlet</servlet-class>

    <init-param>
      <param-name>echelle</param-name>
      <param-value>20</param-value>
    </init-param>
  </servlet>

  <servlet-mapping>
    <servlet-name>cs</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>

  <welcome-file-list>
    <welcome-file>index.do</welcome-file>
  </welcome-file-list>
</web-app>
```

---

# Lire les paramètre dans une servlet

- La méthode `init()` d'une servlet est l'endroit idéale pour lire les paramètres de configuration d'une servlet.
- Dans la Servlet précédente, pour lire le paramètre `echelle`, on peut réécrire la servlet de la manière suivante:

# Paramétrer une servlet

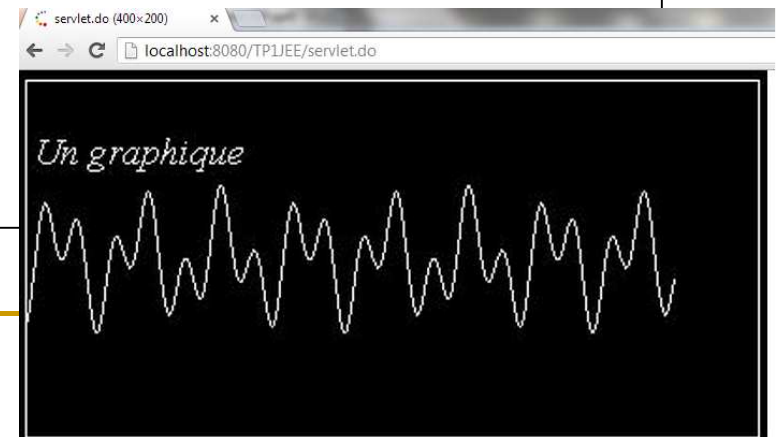
```
package web;

import java.awt.*;import java.awt.image.BufferedImage;
import java.io.*; import javax.servlet.*;import javax.servlet.http.*;
import com.sun.image.codec.jpeg.*;

public class FirstServlet extends HttpServlet {

    private int echelle;

    @Override
    public void init() throws ServletException {
        String param1=getInitParameter("echelle");
        try {
            echelle=Integer.parseInt(param1);
        } catch (Exception e) {
            echelle=100;
        }
    }
}
```

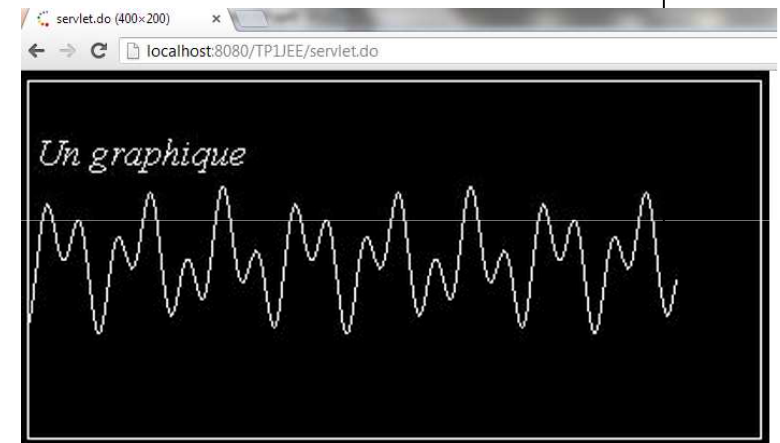


# Paramétrer une servlet (Suite)

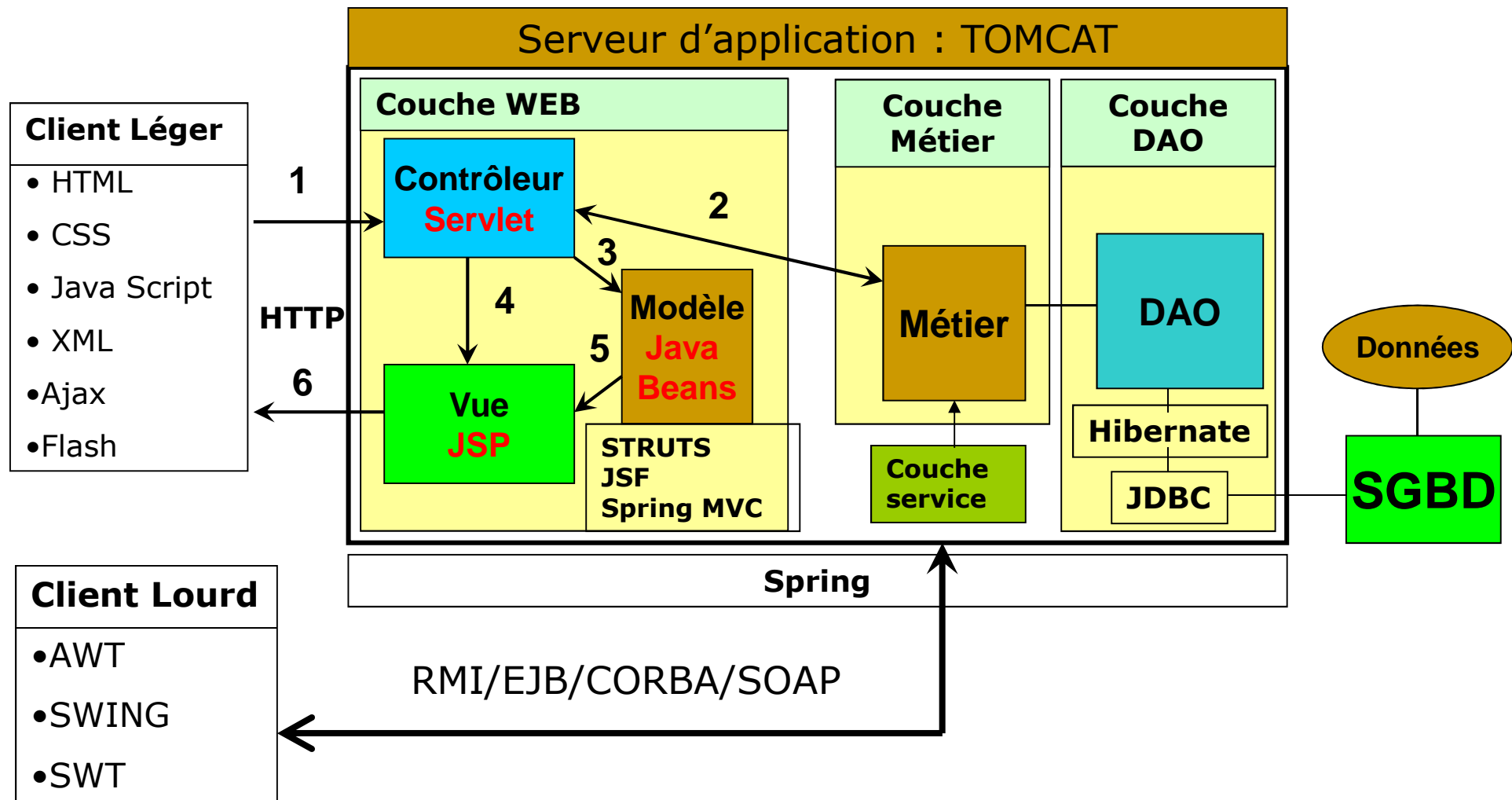
@Override

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    ServletOutputStream out = response.getOutputStream();
    BufferedImage bim= new BufferedImage(400, 200, BufferedImage.TYPE_3BYTE_BGR);
    Graphics2D g = bim.createGraphics();
    g.setFont(new Font("Serif", Font.ITALIC, 20));
    g.drawString("Un graphique", 10,50);
    g.drawRect(5, 5, 390, 190);
    for(int i=5;i<350;i++){
        int x1=i;int y1= 100+(int)(f(x1)*echelle);
        int x2=i+1;int y2=100+ (int)(f(x2)*echelle);
        g.drawLine(x1,y1, x2,y2);
    }
    JPEGImageEncoder encode = JPEGCodec.createJPEGEncoder(out);
    encode.encode(bim);
    out.close();
}

private double f(double x){
    return Math.cos(x/7)+Math.sin(x/3);
}}
```

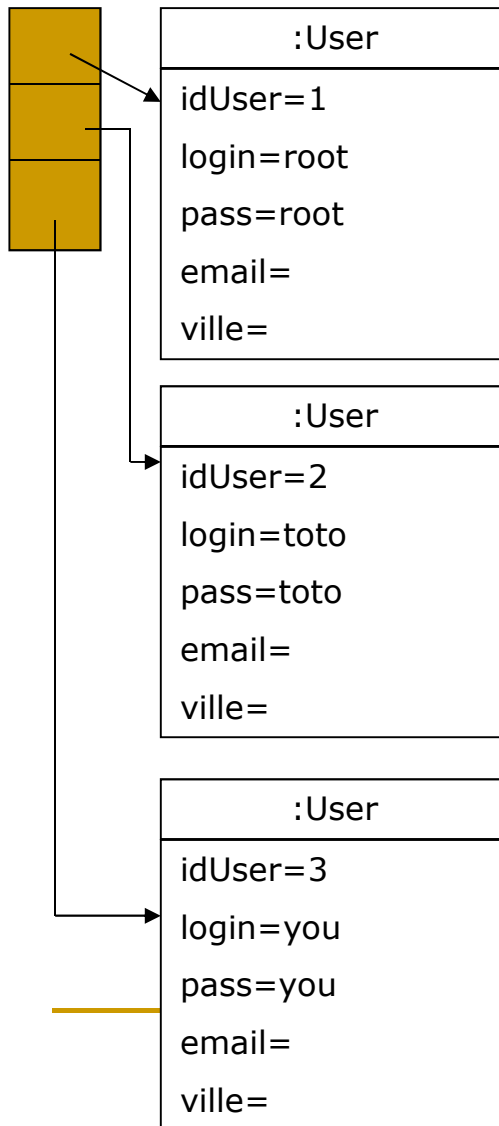


# Architecture J2EE



# Application orientée objet

Users:Collection



## Mapping Objet Relationnel

```
public List<User> getAllUsers() {

    List<User> users=new ArrayList<User>();
    Class.forName("com.mysql.jdbc.Driver");
    Connection conn=DriverManager.getConnection
    ("jdbc:mysql://localhost:3306/DB_USERS","root","");
    PreparedStatement ps=conn.prepareStatement
    ("select * from users");
    ResultSet rs=ps.executeQuery();
    while(rs.next()){
        User u=new User();
        u.setIdUser(rs.getInt("ID_USER"));
        u.setLogin(rs.getString("LOGIN"));
        u.setPass(rs.getString("PASS"));
        users.add(u);
    }
    return(users);
}
```

Users : Table

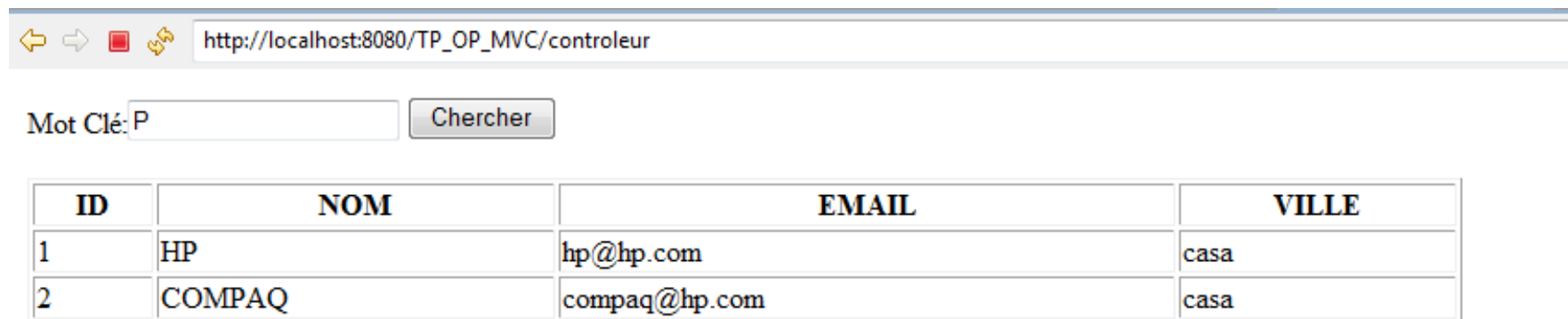
	idUser	login	pass	email	ville
▶	1	root	root	root@yahho.fr	casa
	2	toto	toto	toto@yahoo.fr	rabat
	3	you	you	med@yahoo.fr	casa
*	(NuméroAuto)				

med@vroussfi.net

## Base de données relationnelle

# Application

- Le travail que nous allons réaliser consiste à afficher les clients en saisissant un mot clé qui représente une partie du nom du client
- La vue qu'on souhaite réaliser est la suivante :



The screenshot shows a web browser window with the address bar displaying `http://localhost:8080/TP_OP_MVC/controleur`. Below the address bar, there is a search form with the label "Mot Clé:" followed by a text input field containing the letter "P". To the right of the input field is a button labeled "Chercher". Below the search form is a table with four columns: "ID", "NOM", "EMAIL", and "VILLE". The table contains two rows of data.

ID	NOM	EMAIL	VILLE
1	HP	hp@hp.com	casa
2	COMPAQ	compaq@hp.com	casa

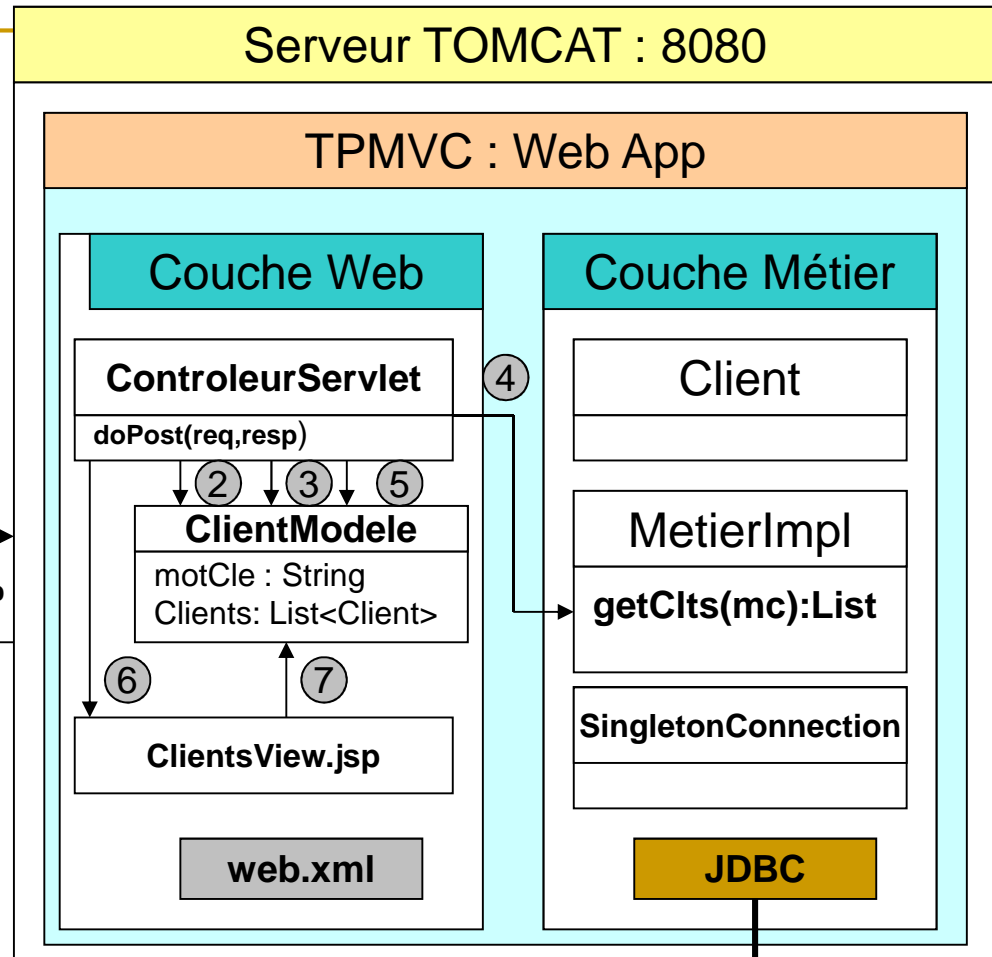


# Architecture

http://localhost:8080/TP\_OP\_MVC/contrôleur

Mot Clé:

ID	NOM	EMAIL	VILLE
1	HP	hp@hp.com	casa
2	COMPAQ	compaq@hp.com	casa



- ① Le client envoie la requête au contrôleur
- ② Le contrôleur instancie le modèle
- ③ Le contrôleur Stocke les données de la requête dans le modèle puis vérifie la validité des données
- ④ Le contrôleur Fait appel à la couche métier pour faire les traitement
- ⑤ Le contrôleur Stocke les résultats de traitement dans le modèle
- ⑥ Le contrôleur fait un forward vers la vue JSP
- ⑦ La vue récupère les résultats du modèle
- ⑧ Le contrôleur affiche au client le résultat de la vue

SGBC MYSQL : 3306

ID_CLI	NOM	EMAIL	VILLE
1	HP	hp@hp.com	casa
2	COMPAQ	compaq@hp.com	casa
3	ACER	admin@acer.com	rabat

# Base de données

## Structure de la table **CLIENTS** :

Champ	Type	Interclassement	Attributs	Null	Défaut	Extra
<u>ID_CLI</u>	int(11)			Non	<i>Aucun</i>	auto_increment
NOM	varchar(25)	latin1_swedish_ci		Non	<i>Aucun</i>	
EMAIL	varchar(23)	latin1_swedish_ci		Non	<i>Aucun</i>	
VILLE	varchar(15)	latin1_swedish_ci		Non	<i>Aucun</i>	

## Données de la table **CLIENTS** :

ID_CLI	NOM	EMAIL	VILLE
1	HP	hp@hp.com	casa
2	COMPAQ	compaq@hp.com	casa
3	ACER	admin@acer.com	rabat

---

# Couche web

- La couche web est définie par :
  - Le modèle qui permet de stocker les données qu'on va saisir (motCle) et le résultat à afficher (List<Client>). Le reste étant les Getters et Setters
  - Le contrôleur qui est une servlet qui va se charger de:
    - Créer un objet du modèle
    - Stocker les données de la requête (motCle) dans ce modèle
    - Récupérer les résultats de la couche métier
    - Stocker ces résultats dans le modèle
    - Faire appel à la vue JSP.
  - La vue (ClientsView.jsp) qui va se charger de récupérer le modèle et d'afficher les résultats.

# Structure du projet

## TP\_OP\_MVC

▷ .settings

▷ build

▷ classes

▷ src

▷ metier

Client.java

MetierImpl.java

SingletonConnection.java

TestMetier.java

▷ web

ControleurServlet.java

ModeleClient.java

▷ WebContent

▷ META-INF

▷ WEB-INF

▷ lib

mysql-connector-java-3.0.17-ga-bin.jar

web.xml

Clients1.jsp

ClientView.jsp

.classpath

.project

**Couche métier**

**Couche Web**

---

# Couche Métier

- La couche métier se compose de :
  - ❑ La classe Client
  - ❑ Un singleton Connection qui contient une méthode getConnection qui retourne un objet Connection unique vers la base de données, quelque soit le nombre de machines clientes connecté au serveur
  - ❑ Une classe MetierImpl qui contient une méthode qui permet de retourner une Liste de client sachant un mot clé.
  - ❑ Une application pour tester MetierImpl

# Classe Client.java

```
package metier;
public class Client {
    private Long idClient;
    private String nom,email,ville;
    public Long getIdClient() {
        return idClient;
    }
    // Constructeurs
    public Client() {
        super();
    }
    public Client(String nom, String email, String ville) {
        super();
        this.nom = nom;
        this.email = email;
        this.ville = ville;
    }
    // Getters et Setters
}
```

# Classe SingletonConnection

```
package metier;

import java.sql.Connection; import java.sql.DriverManager;

public class SingletonConnection {
    private static Connection connection;
    static{
        try {
            Class.forName("com.mysql.jdbc.Driver");
            connection=DriverManager.getConnection
                ("jdbc:mysql://localhost:3306/DB_OPERATEUR","root","");

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    public static Connection getConnection() {
        return connection;
    }
}
```

# La classe MetierImpl.java

```
package metier;
import java.sql.*; import java.util.*;
public class MetierImpl {
    public List<Client> getClientsParMC(String mc){
        List<Client> clts=new ArrayList<Client>();
        Connection conn=SingletonConnection.getConnection();
        try {
            PreparedStatement ps=conn.prepareStatement
                ("select * from CLIENTS where NOM like ?");
            ps.setString(1, "%" + mc + "%");
            ResultSet rs=ps.executeQuery();
            while(rs.next()){
                Client c=new Client();
                c.setIdClient(rs.getLong("ID_CLI"));
                c.setNom(rs.getString("NOM"));
                c.setEmail(rs.getString("EMAIL"));
                c.setVille(rs.getString("VILLE"));
                clts.add(c);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return clts;
    }
}
```



# Classe TestMetier

```
package metier;
import java.util.List;
public class TestMetier {

    public static void main(String[] args) {

        MetierImpl metier=new MetierImpl();

        List<Client> clts=metier.getClientsParMC("");

        for(Client c:clts)
            System.out.println(c.getNom());
    }
}
```

---

# Le modèle : ClientModele.java

```
package web;
import java.util.*;
import metier.Client;
public class ModeleClient {
    private String motCle;
    private List<Client> clients=new ArrayList<Client>();
    // Geters et Setters

}
```

# Le contrôleur : ControleurServlet

```
package web;
import java.io.*;import java.util.*;import javax.servlet.*;
import javax.servlet.http.*; import metier.*;
public class ControleurServlet extends HttpServlet {
    private MetierImpl metier;
    @Override
    public void init() throws ServletException {
        metier=new MetierImpl();
    }
    @Override
    protected void doPost(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        ModeleClient mod=new ModeleClient();
        mod.setMotCle(request.getParameter("motCle"));
        List<Client> clts=metier.getClientsParMC(mod.getMotCle());
        mod.setClients(clts);
        request.setAttribute("modele", mod);
        request.getRequestDispatcher("ClientView.jsp").forward(request,
            response);
    }
}
```

---

# Descripteur de déploiement de servlets : web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!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>
  <servlet>
    <servlet-name>cs</servlet-name>
    <servlet-class>web.ControleurServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>cs</servlet-name>
    <url-pattern>/controleur</url-pattern>
  </servlet-mapping>

</web-app>
```

# La vue : ClientsView.jsp

```
<%@page import="metier.Client"%><%@page import="java.util.List"%>
<%@page import="web.ModeleClient"%>
<%
    ModeleClient mod;
    if(request.getAttribute("modele")!=null){
        mod=(ModeleClient)request.getAttribute("modele");
    }
    else{ mod=new ModeleClient(); }
%>
<html>
<body>
    <form action="controleur" method="post">
        Mot Clé:<input type="text" name="motCle" value="<%=mod.getMotCle() %>">
        <input type="submit" value="Chercher">
    </form>
    <table border="1" width="80%">
        <tr>
            <th>ID</th><th>NOM</th><th>EMAIL</th><th>VILLE</th>
        </tr>
        <%
            List<Client> clts=mod.getClients();
            for(Client c:clts){ %>
                <tr>
                    <td><%=c.getIdClient() %></td> <td><%=c.getNom() %></td>
                    <td><%=c.getEmail() %></td> <td><%=c.getVille() %></td>
                </tr>
            <% } %>
        </table></body></html>
```

# La vue : ClientsView.jsp en utilisant JSTL

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
  <form action="controleur" method="post">
    Mot Clé:<input type="text" name="motCle" value="${modele.motCle}">
    <input type="submit" value="Chercher">
  </form>
  <table border="1" width="80%">
    <tr>
      <th>ID</th><th>NOM</th><th>EMAIL</th><th>VILLE</th>
    </tr>
    <c:forEach items="${modele.clients}" var="c">
      <tr>
        <td>${c.idClient}</td>
        <td>${c.nom}</td>
        <td>${c.email}</td>
        <td>${c.ville}</td>
      </tr>
    </c:forEach>
  </table>
</body>
</html>
```

- 
- Voir Vidéo : TP MVC JDBC

---

# 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

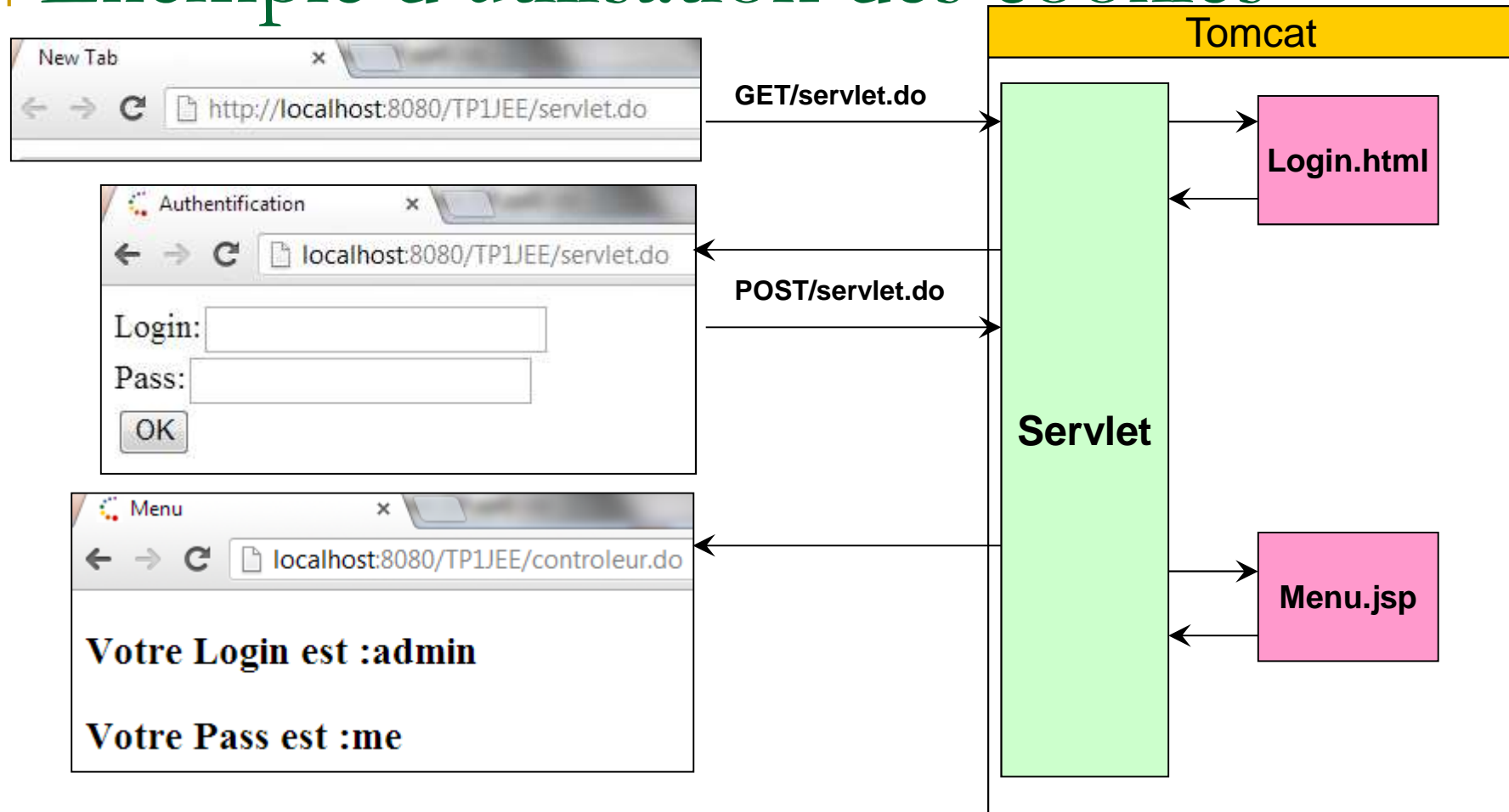
- Code pour créer un cookie et l'ajouter au client

```
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



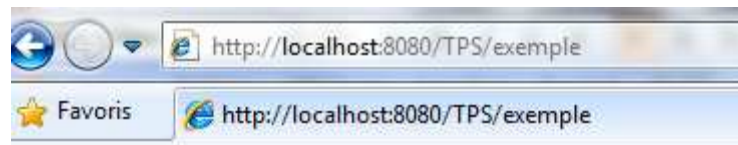
---

# 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 envoie 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

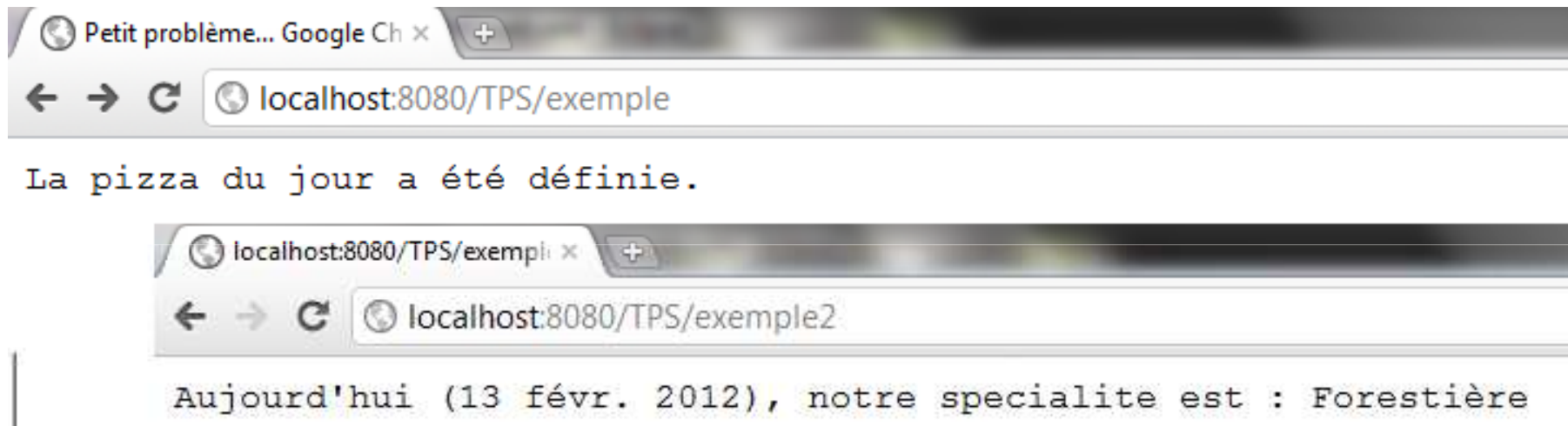
- La collaboration est obtenue par l'interface ServletContext
- L'utilisation de ServletContext permet aux applications web de disposer de son propre conteneur d'informations unique
- Une Servlet retrouve le ServletContext de son application web par un appel à `getServletContext()`
- Exemples de méthodes
  - ❑ `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);  
    }  
}
```

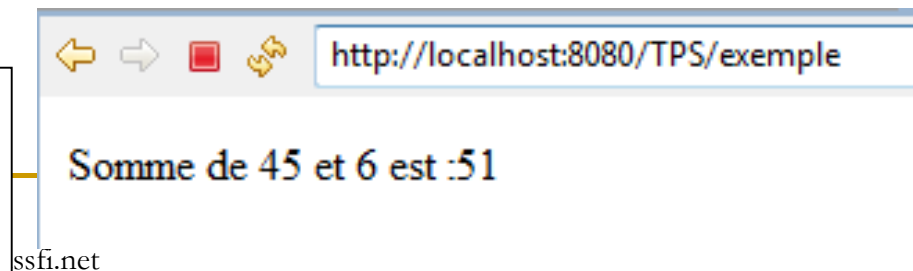
Transmission  
d'informations par  
attributs

- 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

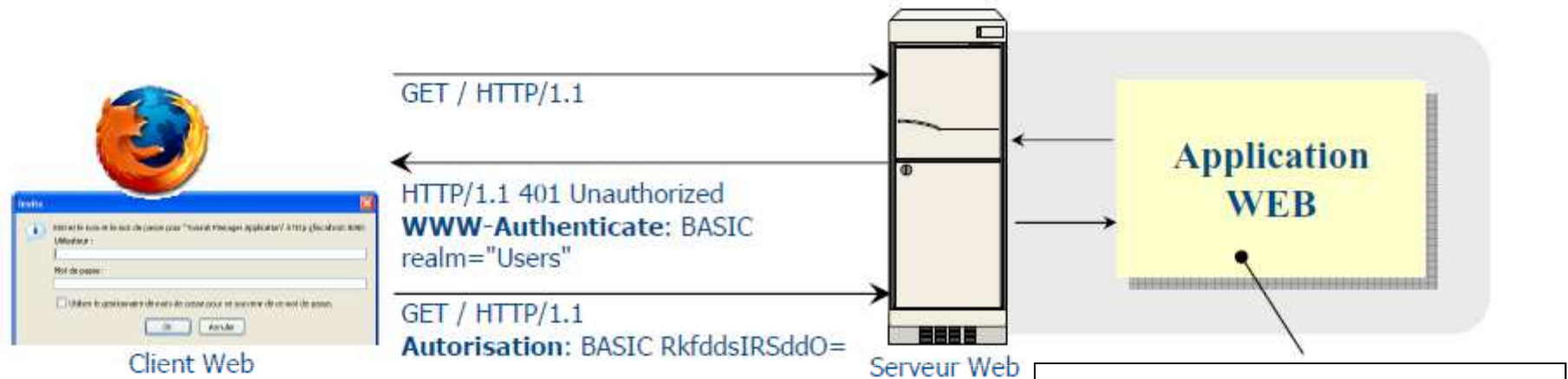
---

# Sécurité et Authentification

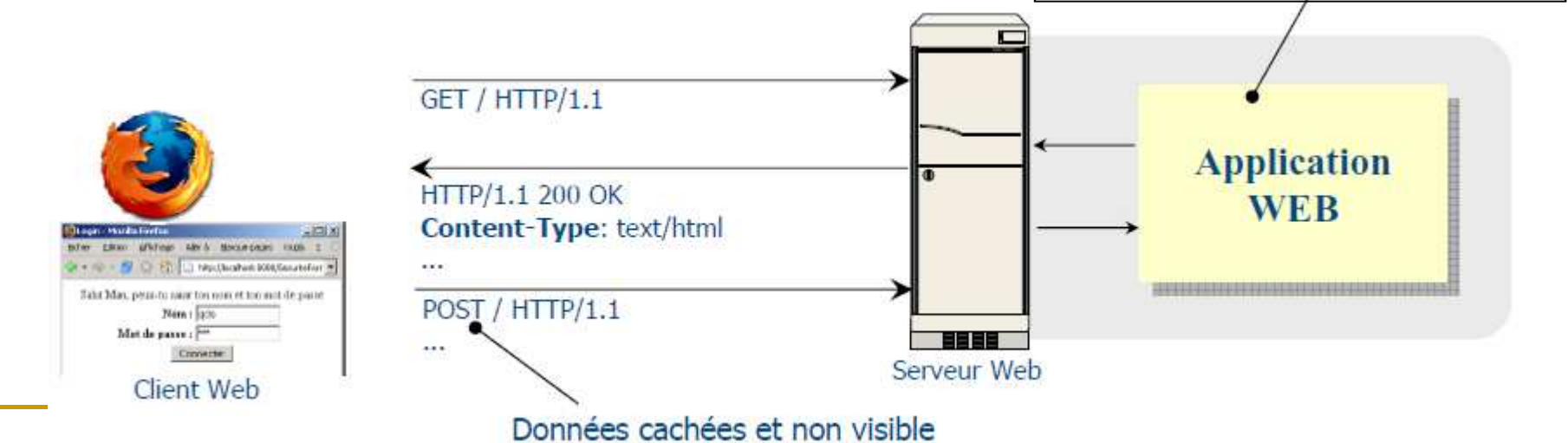
- La sécurité consiste à conserver les informations sensibles dans les mains des utilisateurs
  - Authentification : capable de vérifier l'identité des parties impliquées
  - Habilitation : limiter l'accès aux ressources à un ensemble d'utilisateurs
  - Confidentialité : garantir la communication des parties impliquées
- Nous distinguons plusieurs types d'autorisation :
  - **BASIC** : fournit par le protocole HTTP basé sur un modèle simple de demande/réponse (codage Base64)
  - **FORM** : authentification ne reposant pas celle du protocole HTTP

# Sécurité et Authentification

## ■ Principe des autorisations de type BASIC



## ■ Principe des autorisations de type FORM



---

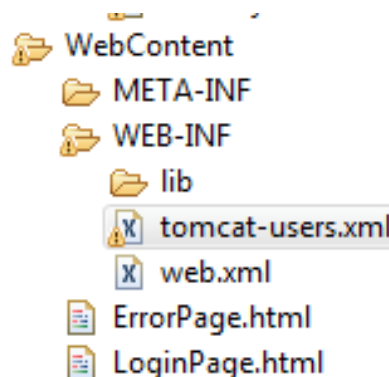
# Sécurité et Authentification

- Gérée par le conteneur de Servlets (**Identification 1**) :
  - Spécification d'un domaine de sécurité dans le fichier de configuration **web.xml**
  - Les utilisateurs sont gérés (l'utilisateur existe-il, le mot de passe est-il correct, ...) **uniquement** par le conteneur de Servlets
    - Basée sur les rôles (BASIC)
    - A base de formulaire (FORM)
- Effectuée à l'intérieur des Servlets (**Identification2**) :
  - Les utilisateurs sont stockés dans une base de données, un fichier, ...
  - La vérification est effectuée dans les Servlets (besoin d'un codage)
    - Personnalisée (BASIC)
    - Personnalisée à base de formulaire (FORM)

## Exemple : Servlet qui après identification affiche un ensemble d'informations

```
public class SecurityRolesServlet extends HttpServlet {  
    protected void doGet(HttpServletRequest req,  
        HttpServletResponse res)  
        throws ServletException, IOException {  
        res.setContentType("text/html");  
        PrintWriter out = res.getWriter();  
        out.println("Bonjour : " + req.getRemoteUser());  
        out.println("Information Top-Secrête");  
        out.println("Type d'authentification : " + req.getAuthType());  
        out.println("Est-un administrateur : "+  
            req.isUserInRole("admin"));  
    }  
}
```

Fichier tomcat-users:



```
<?xml version="1.0" encoding="UTF-8"?>  
<tomcat-users>  
    <role rolename="manager" />  
    <user username="admin" password="admin"  
        roles="manager" />  
</tomcat-users>
```

## Authentification basée sur les rôles : Identification1 (web)

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app ...>
  <servlet>
    <servlet-name>ss</servlet-name>
    <servlet-class>web.SecurityRolesServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>ss</servlet-name>
    <url-pattern>/secure</url-pattern>
  </servlet-mapping>

  <security-constraint>
    <web-resource-collection>
      <web-resource-name>ss</web-resource-name>
      <url-pattern>/secure</url-pattern>
      <http-method>GET</http-method>
    </web-resource-collection>
    <auth-constraint>
      <role-name>manager</role-name>
    </auth-constraint>
  </security-constraint>
  <login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>Authentication de
    SecuriteRoleServlet</realm-name>
  </login-config>
</web-app>
```

Authentication requise

Le serveur localhost:8080 requiert un nom d'utilisateur et un mot de passe. Message du serveur : Authentification de SecuriteRoleServlet.

Nom d'utilisateur :

Mot de passe :

**Définition des Servlets**  
contenues dans  
l'application WEB et  
des  
chemins virtuels

Définit pour quelles  
URL la contrainte  
d'identification  
doit être mise en  
oeuvre

Protection des  
ressources pour la  
méthode GET

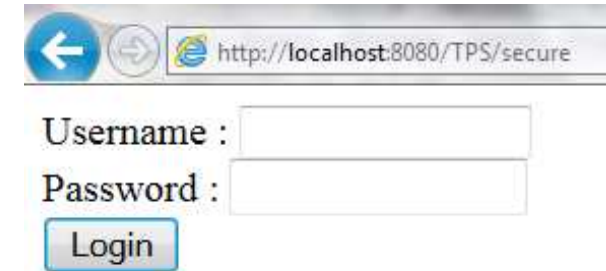
Rôle(s) ayant le droit  
d'accéder aux  
ressources de  
l'application WEB

« Habillage » de la  
boîte d'identification

# Authentification personnalisée basée sur les rôles :Identification 1

## ■ Web.xml

```
<web-app ...>
    ...
    <servlet> ... </servlet>
    <servlet-mapping> ... </servlet-mapping>
    <security-constraint> ... </security-constraint>
<login-config>
    <auth-method>FORM</auth-method>
    <form-login-config>
        <form-login-page>/loginpage.html</form-login-page>
        <form-error-page>/errorpage.html</form-error-page>
    </form-login-config>
</login-config>
</web-app>
```



Username :

Password :

Login

Balise qui stocke la page contenant le formulaire d'authentification

Balise qui stocke la page des erreurs de connexion

# Page d'authentification

## ■ LoginPage.html:

- Le formulaire de la page LoginPage.html doit employer la méthode POST pour la transmission des données des valeurs spéciales pour les noms des composants

Le moteur de Servlet avec les informations contenues dans le fichier web.xml se charge de traiter l'identification

```
<html>
```

```
<body>
```

```
<form method="POST" action="j_security_check">
```

```
Username : <input type="text" size="15" maxlength="25"  
           name="j_username"><br>
```

```
Password : <input type="password" size="15" maxlength="25"  
           name="j_password"><br>
```

```
<input value="Login" type="submit">
```

```
</form>
```

```
</body>
```

```
</html>
```

Valeur pour le nom  
d'utilisateur

Valeur pour le mot  
de passe



# Authentification personnalisée basée sur les rôles :Identification 1

## ■ Web.xml

```
<web-app ...>
    ...
    <servlet> ... </servlet>
    <servlet-mapping> ... </servlet-mapping>
    <security-constraint> ... </security-constraint>
<login-config>
    <auth-method>FORM</auth-method>
    <form-login-config>
        <form-login-page>/loginpage.html</form-login-page>
        <form-error-page>/errorpage.html</form-error-page>
    </form-login-config>
</login-config>
</web-app>
```

Balise qui stocke la page contenant le formulaire d'authentification

Balise qui stocke la page des erreurs de connexion

---

# 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	<a href="http://java.sun.com/jsp/jstl/core">http://java.sun.com/jsp/jstl/core</a>	c
Format	<a href="http://java.sun.com/jsp/jstl/fmt">http://java.sun.com/jsp/jstl/fmt</a>	fmt
XML	<a href="http://java.sun.com/jsp/jstl/xml">http://java.sun.com/jsp/jstl/xml</a>	x
SQL	<a href="http://java.sun.com/jsp/jstl/sql">http://java.sun.com/jsp/jstl/sql</a>	sql
Fonctions	<a href="http://java.sun.com/jsp/jstl/functions">http://java.sun.com/jsp/jstl/functions</a>	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 : -->

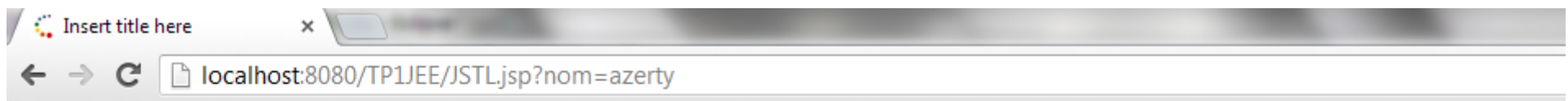
■ <c:out value="\${header['user-agent']}" default="Inconnu"/>

<!-- 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:choose/> : 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>
```