
Java Avancé : J2EE

madani.a@ucd.ac.ma

Plan du cours

- Introduction / Rappels
 - Servlets
 - JSP
 - Modèle MVC
 - Framework Hibernate
 - Etudes de cas
-

J2EE

- Introduction-

Introduction

- J2EE (Java Enterprise Edition) est devenu la plateforme de choix des grandes entreprises
 - J2EE est très majoritairement utilisé pour réaliser des applications web solide robuste et bien structuré.
-

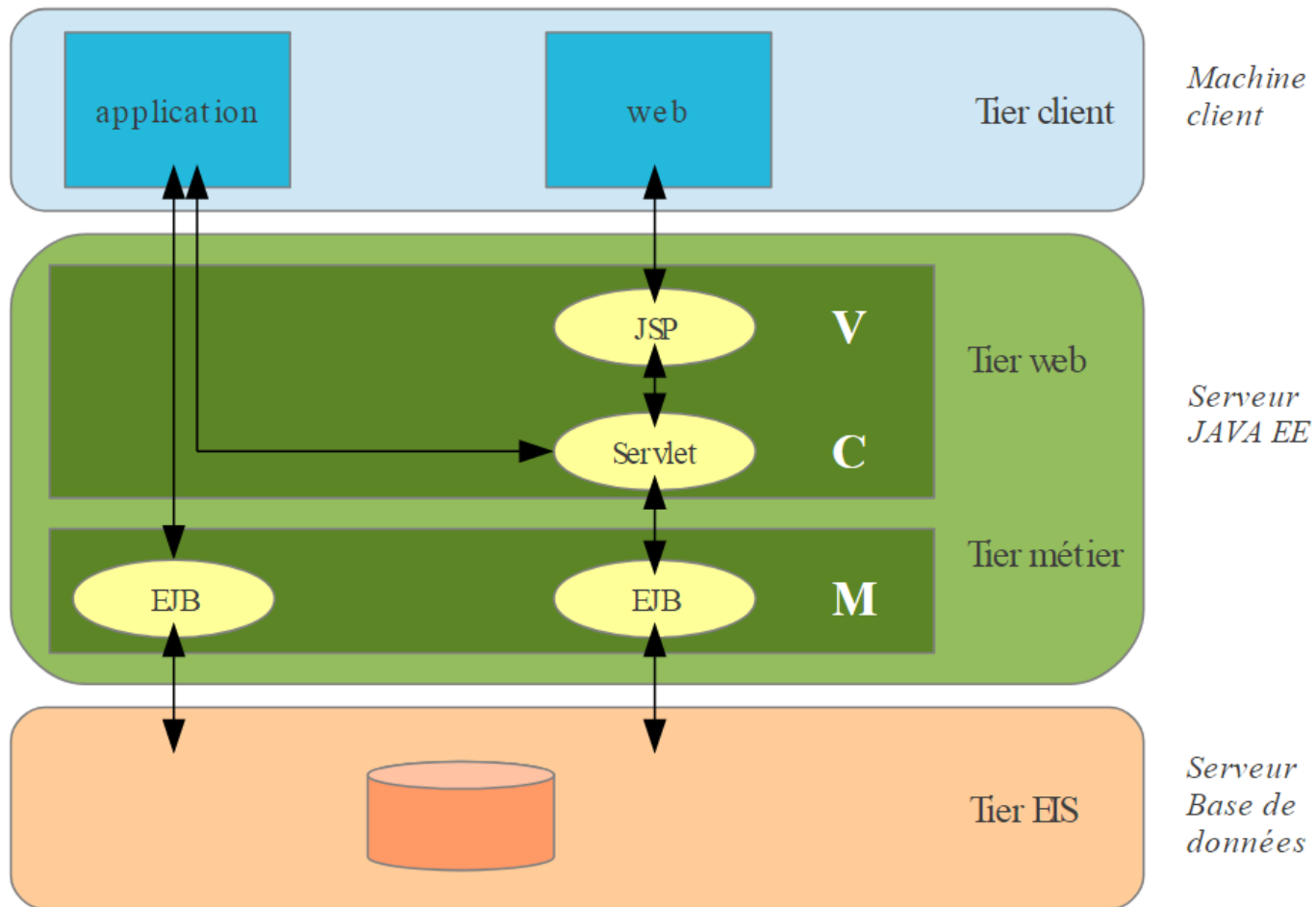
Introduction

- J2EE est conçu comme un langage, une plateforme pour le développement des applications web.
 - On peut le comparer à :
 - PHP
 - Django de python
 - Asp.Net de Microsoft
 - Ruby on Rails
-

Introduction

- J2EE se fonde sur un modèle en plusieurs couches (tiers en anglais)
 - On parle d'architecture 3 tiers ou multi-tiers
 - tier client
 - tier serveur J2EE, composé de :
 - tier web
 - tier métier (business)
 - tier EIS (Enterprise Information Systems)
-

Introduction



J2EE

- Rappels-

Rappel

Pages statiques # pages dynamiques

Lorsque vous visitez une page statique,
<http://www.unserveur.com/index.htm>, par exemple :

- Le client situe le serveur Web spécifié dans la première partie de l'URL (**<http://www.unserveur.com>**, par exemple)
- Le serveur Web demande ensuite la page statique spécifiée dans la deuxième partie de l'URL (/index.htm, par exemple)
- Le serveur Web envoie le contenu de ce fichier au format HTML
- Le client reçoit alors le fichier HTML envoyé par le serveur Web
- Le client traduit les balises HTML du fichier et affiche ensuite le résultat.

Rappel

Pages statiques # pages dynamiques

Lorsque vous visitez une page dynamique,
<http://www.unserveur.com/index.php>, par exemple :

- Le client situe le serveur Web spécifié dans la première partie de l'URL (<http://www.unserveur.com>, par exemple)
- Le serveur Web demande ensuite la page dynamique spécifiée dans la deuxième partie de l'URL (/index.php, par exemple)
- Le serveur lit le fichier et traite le code
- Le résultat du traitement est envoyé au client, au format HTML
- Le client reçoit alors le fichier HTML envoyé par le serveur
- Le client traduit les balises du fichier HTML reçu et l'affiche

Rappel

Pages statiques # pages dynamiques

Page statique (côté client)

- Le navigateur exécute le script après avoir chargé complètement la page
- Le code du script fait partie de la page et peut donc être lu par l'utilisateur (Menu affichage → code source)
- Le code du script doit être pris en charge par le navigateur et les différences entre navigateurs rendent parfois difficile l'écriture des scripts

Page dynamique (côté serveur)

- Le code du script est exécuté par le serveur avant de transmettre au navigateur le résultat
- Le code du script n'est pas transmis au navigateur, il est donc invisible par l'utilisateur
- Le script est indépendant du navigateur, mais impose un équipement adéquat sur le serveur Web

Rappel

Pages statiques # pages dynamiques

The image displays two side-by-side Opera browser windows illustrating static HTML pages.

The left window, titled "Exemple 1", shows the rendered HTML page at `127.0.0.1/test/exemple1.html`. The page content includes:

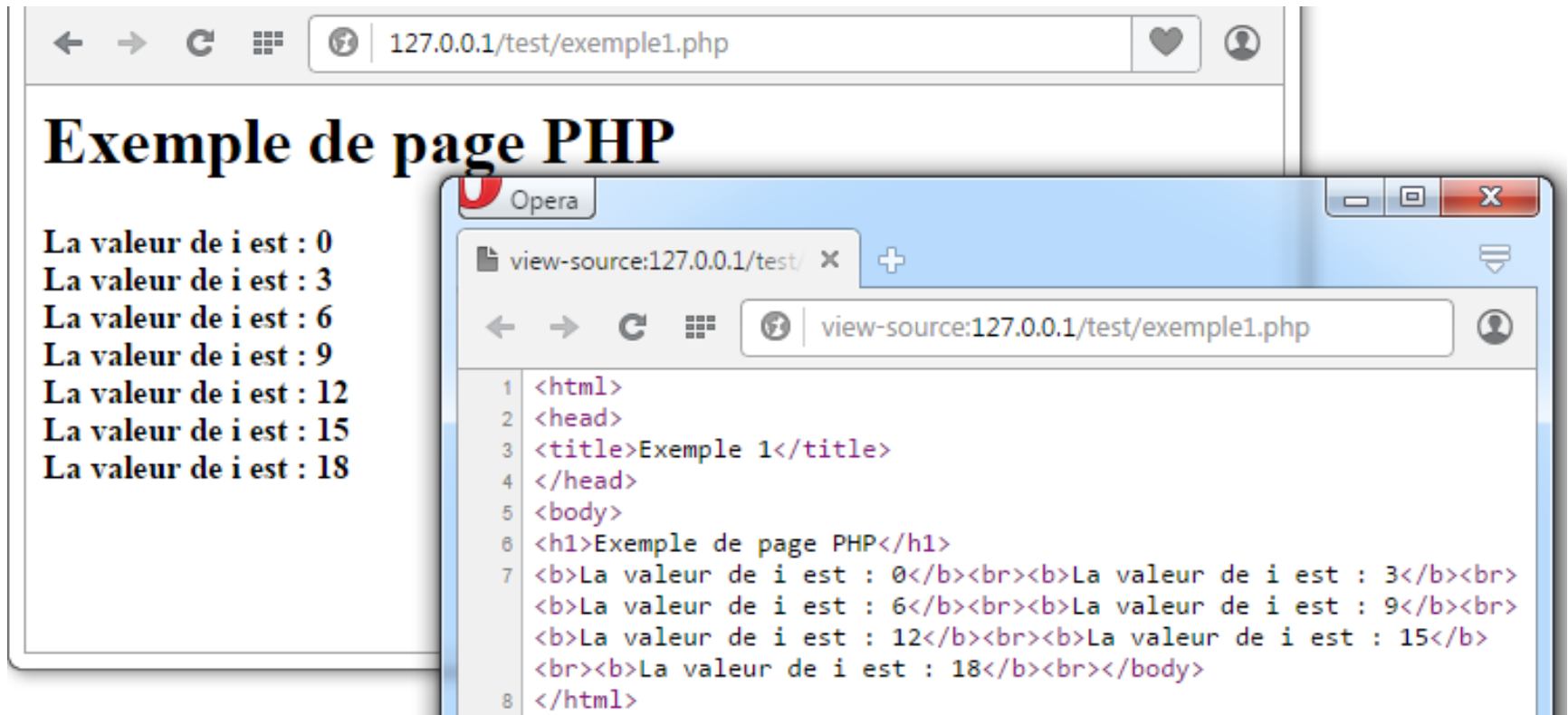
- A heading: **Exemple de page HTML**
- A form section titled "Inscription" containing:
 - Labels "Login" and "Password" next to text input fields.
 - Buttons "OK" and "Annuler" below the input fields.

The right window, titled "view-source:127.0.0.1/test/exemple1.html", shows the source code of the page. The HTML structure is as follows:

```
1 <html>
2 <head>
3 <title>Exemple 1</title>
4 </head>
5 <body>
6 <h1>Exemple de page HTML</h1>
7 <table width="28%" border="0">
8   <tr>
9     <td colspan="2" align="center">Inscription</td>
10  </tr>
11  <tr>
12    <td width="48%">Login</td>
13    <td width="52%"><input type="text"></td>
14  </tr>
15  <tr>
16    <td>Password</td>
17    <td><input type="password"></td>
18  </tr>
19  <tr>
20    <td><input type="submit" value="OK"></td>
21    <td><input type="reset" value="Annuler"></td>
22  </tr>
23 </table>
```

Rappel

Pages statiques # pages dynamiques



The image shows a web browser window displaying a PHP example page. The browser's address bar shows the URL `127.0.0.1/test/exemple1.php`. The page content is titled "Exemple de page PHP" and lists six values of `i`: 0, 3, 6, 9, 12, 15, and 18. An Opera browser window is overlaid on the page, showing the source code of the file. The source code is HTML with PHP output embedded. The code structure is as follows:

```
1 <html>
2 <head>
3 <title>Exemple 1</title>
4 </head>
5 <body>
6 <h1>Exemple de page PHP</h1>
7 <b>La valeur de i est : 0</b><br><b>La valeur de i est : 3</b><br>
  <b>La valeur de i est : 6</b><br><b>La valeur de i est : 9</b><br>
  <b>La valeur de i est : 12</b><br><b>La valeur de i est : 15</b>
  <br><b>La valeur de i est : 18</b><br></body>
8 </html>
```

Rappel

Pages statiques # pages dynamiques

The image shows a side-by-side comparison of a web browser window and its source code. The left window displays the rendered HTML output of a JSP page, while the right window shows the underlying JSP code.

Left Window (Rendered Page):

- Address bar: localhost:8080/Projets_Cours
- Title: Insert title here
- Content:

Exemple de page JSP

La valeur de i est : 0
La valeur de i est : 1
La valeur de i est : 2
La valeur de i est : 3
La valeur de i est : 4
La valeur de i est : 5
La valeur de i est : 6
La valeur de i est : 7
La valeur de i est : 8
La valeur de i est : 9
La valeur de i est : 10
La valeur de i est : 11
La valeur de i est : 12
La valeur de i est : 13
La valeur de i est : 14
La valeur de i est : 15
La valeur de i est : 16
La valeur de i est : 17
La valeur de i est : 18

Right Window (Source Code):

- Address bar: view-source:localhost:8080/Projets_Cours/index.jsp
- Tab: view-source:localhost:8080/Proje...
- Content:

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="ISO-8859-1">
5 <title>Insert title here</title>
6 <script type="text/javascript" src="http://gc.kis.v2.scr.kaspersky-labs
7 <body>
8 <h1>Exemple de page JSP</h1>
9 La valeur de i est : 0<br>
10 La valeur de i est : 1<br>
11 La valeur de i est : 2<br>
12 La valeur de i est : 3<br>
13 La valeur de i est : 4<br>
14 La valeur de i est : 5<br>
15 La valeur de i est : 6<br>
16 La valeur de i est : 7<br>
17 La valeur de i est : 8<br>
18 La valeur de i est : 9<br>
19 La valeur de i est : 10<br>
20 La valeur de i est : 11<br>
21 La valeur de i est : 12<br>
22 La valeur de i est : 13<br>
23 La valeur de i est : 14<br>
24 La valeur de i est : 15<br>
25 La valeur de i est : 16<br>
26 La valeur de i est : 17<br>
27 La valeur de i est : 18<br>
```

Rappel

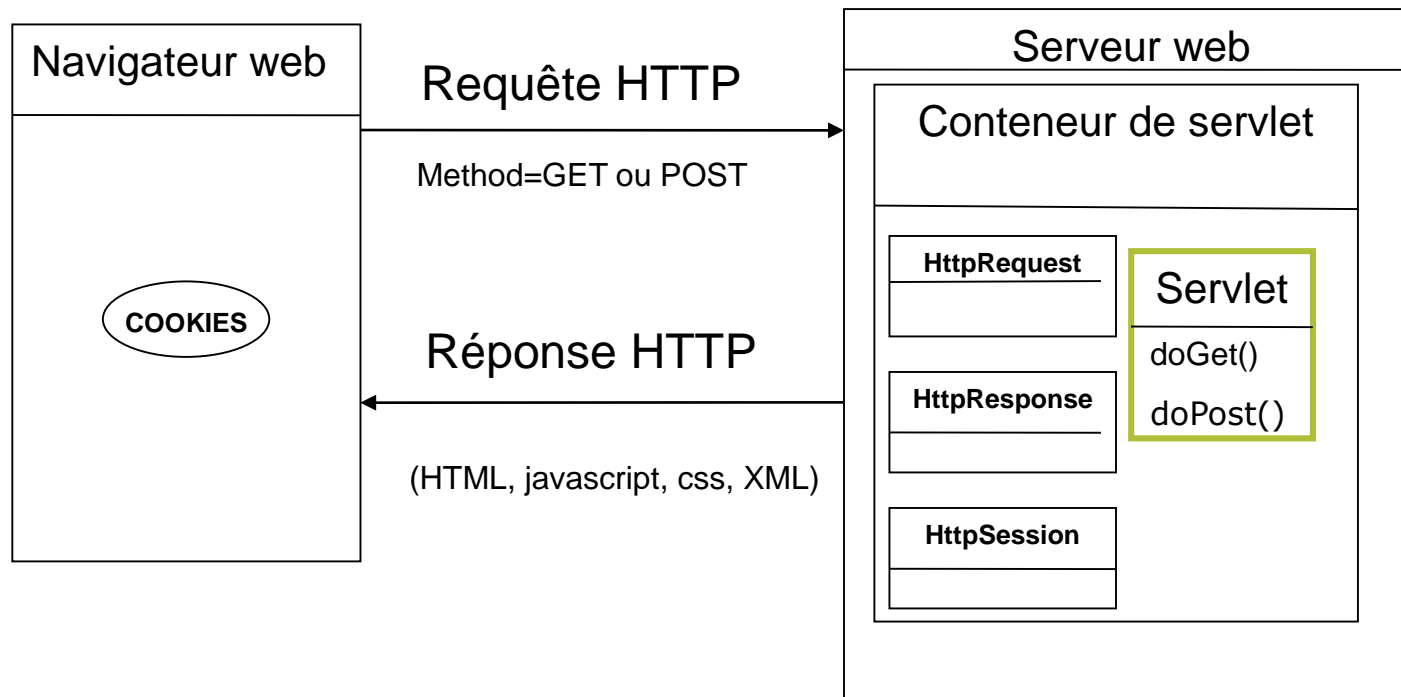
Intérêt du traitement côté serveur

- Diminution du trafic réseau
- Absence de problèmes de compatibilité des navigateurs
- Accès aux bases de données
- Code invisible (confidentialité)

J2EE

- Servlet -

Introduction aux servlets



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, etc.)
- ❑ Ce protocole permet également de soumissionner des données (formulaires, liens hypertexte)
- ❑ Protocole de rapatriement des documents

■ Fonctionnement

- ❑ Connexion
- ❑ demande (GET, POST) d'un document : Requête HTTP
- ❑ renvoi du document (status=200) ou d'une erreur (status=404 quand le document n'existe pas)
- ❑ déconnexion

Méthodes du protocole HTTP

- ❑ Récupération d'un document
 - méthode GET
- ❑ Soumission d'un formulaire
 - méthodes GET ou POST
- ❑ Envoyer un fichier du client au serveur
 - ❑ Méthode PUT
- ❑ Demander au serveur de supprimer un fichier
 - ❑ Méthode DELETE
- ❑ Récupérer des informations sur un document (type, taille, date de dernière modification, ...)
 - ❑ Méthode HEAD

Le client envoie la requête : Méthode POST

Entête de la requête

Post /Nom_Script HTTP/1.0

REMOTE_ADDR: 212.217.122.32

HTTP_ACCEPT_LANGUAGE : fr

Accept : image/jpeg

User-Agent : Mozilla/4.0

Méthode, chemin, version

Documents acceptés

Type et version du navigateur ayant soumis la requête.

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

Var1=Value1&

Var2=Value2

Var3=Value3

Paramètres des différents champs du formulaire.

corps de la requête

Le client envoie la requête : Méthode GET

Entête de la requête

```
GET /Nom_Script?var1=val1&var2=val2&.... HTTP/1.0  
REMOTE_ADDR: 212.217.122.32  
HTTP_ACCEPT_LANGUAGE : fr  
Accept : image/jpeg  
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

Ligne de Status

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

Server : Apache/1.3.24

Nom du Serveur

Mime-Version 1.0

Dernière modification

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

Content-Type : Text/html

Type de contenu

Content-length : 4205

Sa taille

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

<HTML><HEAD>

Le fichier que le client va afficher

....

</BODY></HTML>

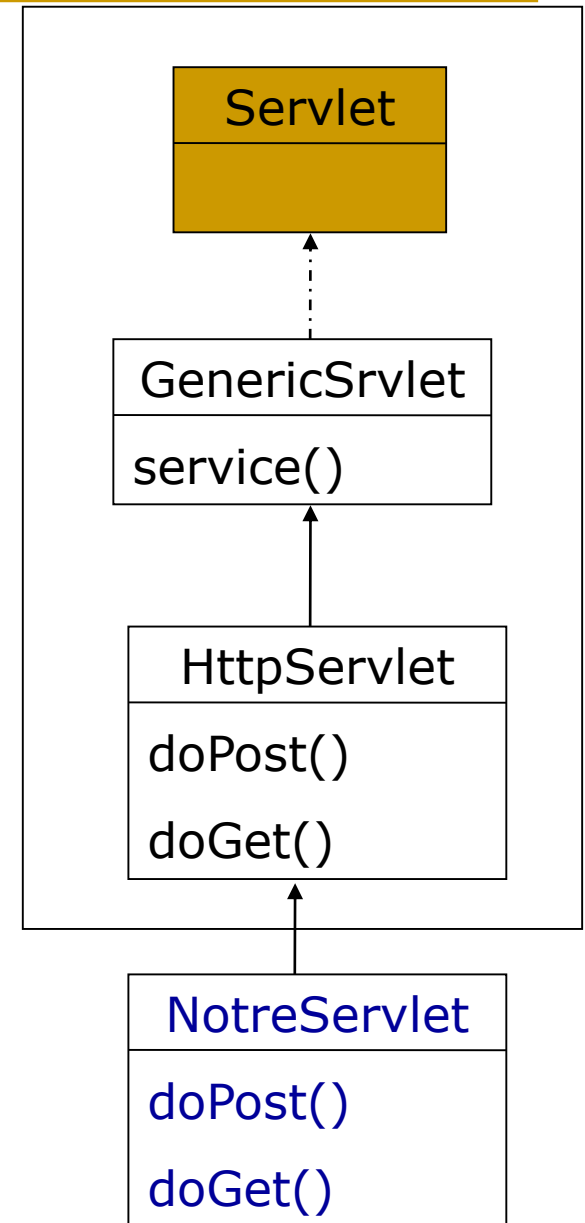
corps de la réponse

Introduction aux servlets

- Une servlet est une classe Java , exécutée par un moteur de servlets, fonctionnant du côté serveur au même titre que les CGI et les langages de script côté serveur tels que ASP ou bien PHP
- Les servlets (PHP) sont au serveur Web ce que les applets (HTML, JS) sont au navigateur pour le client.
- Les servlets permettent donc de gérer des requêtes HTTP et de fournir au client une réponse HTTP dynamique.
- **Le moteur de servlet** (conteneur de servlet) permet d'établir le lien entre la servlet et le serveur Web

Architecture du package `javax.servlet`

- Toutes les servlets implémentent directement ou indirectement l'interface `Servlet`, en dérivant une classe qui l'implémente: c'est-à-dire généralement la classe `HttpServlet`, elle-même issue de `GenericServlet`.
- La classe `GenericServlet` (`javax.servlet.GenericServlet`) définit une classe abstraite (implémentation de base de l'interface `Servlet`).



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, ...)
 - L'objet `HttpServletResponse` permettant de renvoyer une réponse au client (envoyer des informations au navigateur).

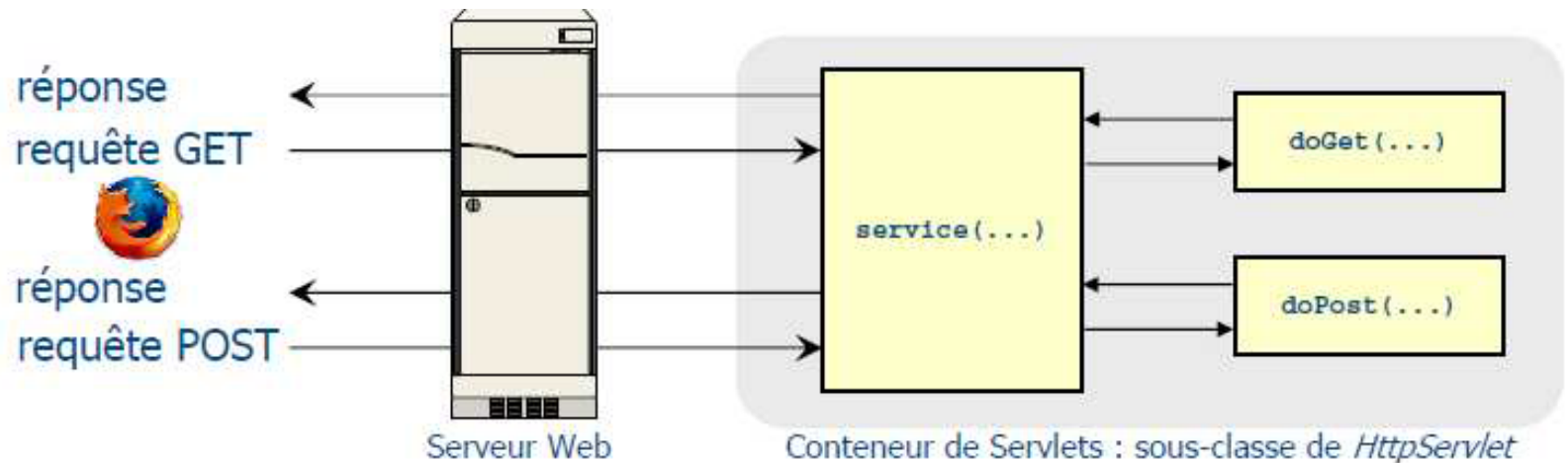
```
protected void service(HttpServletRequest, HttpServletResponse)  
throws ServletException, IOException
```

Développement d'une servlet

- Dans la suite nous allons utiliser uniquement les servlets qui réagissent au protocole HTTP, d'où l'utilisation de la classe `HttpServlet`
 - Afin de développer une servlet fonctionnant avec le protocole HTTP, il suffit de créer une classe étendant *HttpServlet* (qui implémente elle-même l'interface *Servlet*).
 - La classe *HttpServlet* (dérivant de *GenericServlet*) permet de fournir une implémentation de l'interface *Servlet* spécifique à HTTP.
 - La classe *HttpServlet* redéfinit la méthode `service` en lisant la méthode HTTP utilisée par le client, puis en redirigeant la requête vers une méthode appropriée.
-

Développement d'une servlet

- Les deux principales méthodes du protocole HTTP étant GET et POST, il suffit de redéfinir la méthode adéquate afin de traiter la requête :



Développement d'une servlet

- 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
)`

Développement d'une servlet

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class ServletName extends HttpServlet {

    public void doGet(HttpServletRequest req, HttpServletResponse res)    throws
        ServletException {
        // lecture de la requete
        // traitements
        // envoi de la reponse
    }

    public void doPost(HttpServletRequest req, HttpServletResponse res)    throws
        ServletException {
        // lecture de la requete
        // traitements
        // envoi de la reponse
    }
}
```

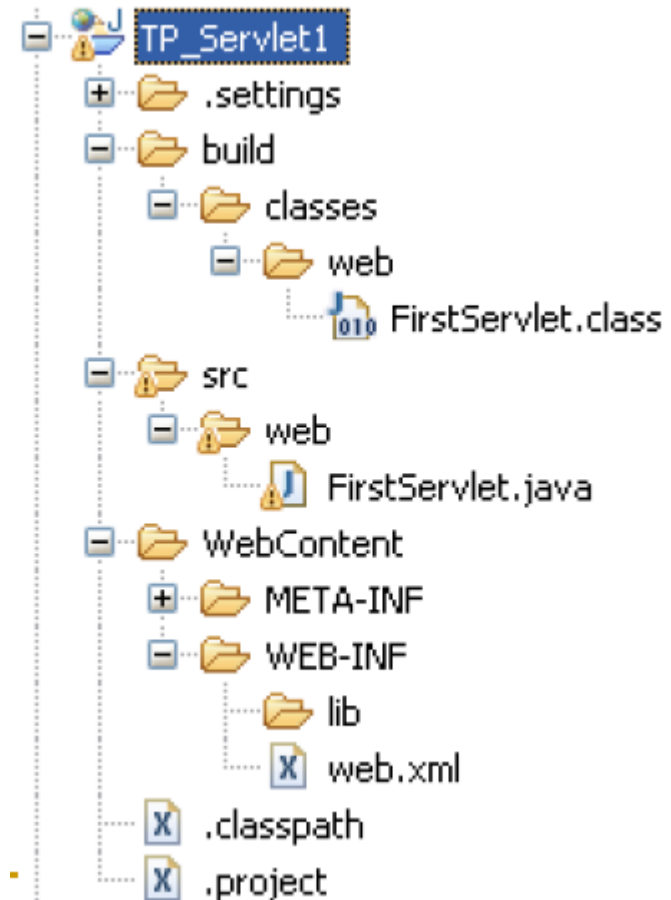
Première Servlet

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

public class PremiereServlet extends HttpServlet {
    public 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 <b>servlet</b>");
        out.println("</BODY>");
        out.println("</HTML>");
        out.close();
    }
}
```

Structure d'un projet 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.
-

Descripteur de déploiement de la servlet: 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>
```

Exemple

- 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 {  
}
```

Charger et invoquer une Servlet

- D'une manière générale, une URL du type :

`http://host:port/nomApp/first?param1=value1&...`

Machine ou
se trouve le
serveur de
servlets

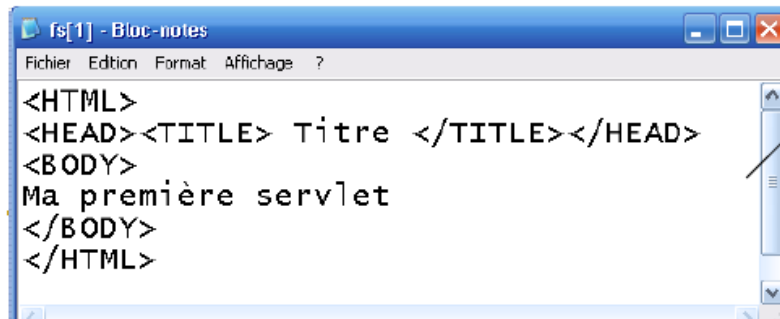
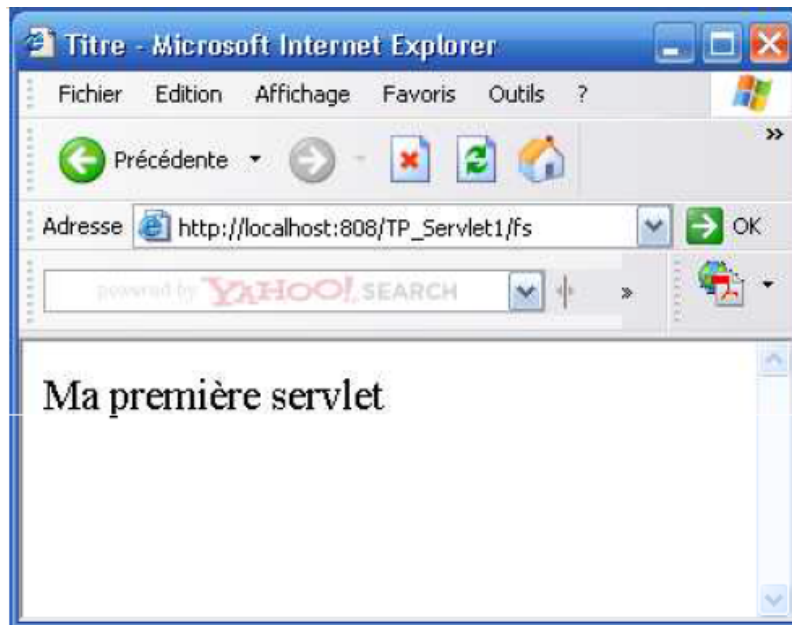
Numéro de port
sur lequel il est à
l'écoute

Web appli.

Alias
correspondant à
<url-pattern>

Liste des
paramètres et
de leurs
valeurs

Charger et invoquer une Servlet: Exemple



A screenshot of a Notepad window titled "fs[1] - Bloc-notes". The menu bar includes "Fichier", "Edition", "Format", "Affichage", and "?". The text area contains the following HTML code:

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

Code source coté client

HttpServletRequest

Méthode	Description
String getMethod()	Récupère la méthode HTTP utilisée par le client
String getHeader(String Key)	Récupère la valeur de l'attribut Key de l'en-tête
String getRemoteHost()	Récupère le nom de domaine du client
String getRemoteAddr()	Récupère l'adresse IP du client
String getParameter(String Key)	Récupère la valeur du paramètre Key (clé) d'un formulaire. Lorsque plusieurs valeurs sont présentes, la première est retournée
String[] getParameterValues(String Key)	Récupère les valeurs correspondant au paramètre Key (clé) d'un formulaire, c'est-à-dire dans le cas d'une sélection multiple (cases à cocher, listes à choix multiples) les valeurs de toutes les entités sélectionnées
Enumeration getParameterNames()	Retourne un objet <i>Enumeration</i> contenant la liste des noms des paramètres passés à la requête
String getServerName()	Récupère le nom du serveur
String getServerPort()	Récupère le numéro de port du serveur

HttpServletResponse

Méthode	Description
<code>void setContentType(String type)</code>	Définit le type de la réponse HTTP, c'est-à-dire le type de données envoyées au navigateur
<code>void setContentLength(int len)</code>	Définit la taille de la réponse
<code>PrintWriter getWriter()</code>	Retourne un objet <i>PrintWriter</i> permettant d'envoyer du texte au navigateur client.
<code>void sendRedirect(String location)</code>	Permet de rediriger le client vers l'URL <i>location</i>

HttpServletRequest : Exemple

```
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>");

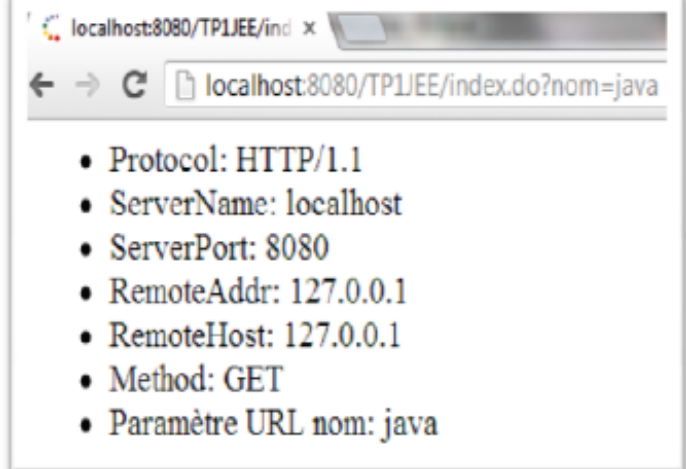
    out.println("<li>Method: " + request.getMethod()+"</li>");

    out.println("<li>Paramètre URL nom: " + request.getParameter("nom")+"</li>");

    out.println("</ul></body></html>");

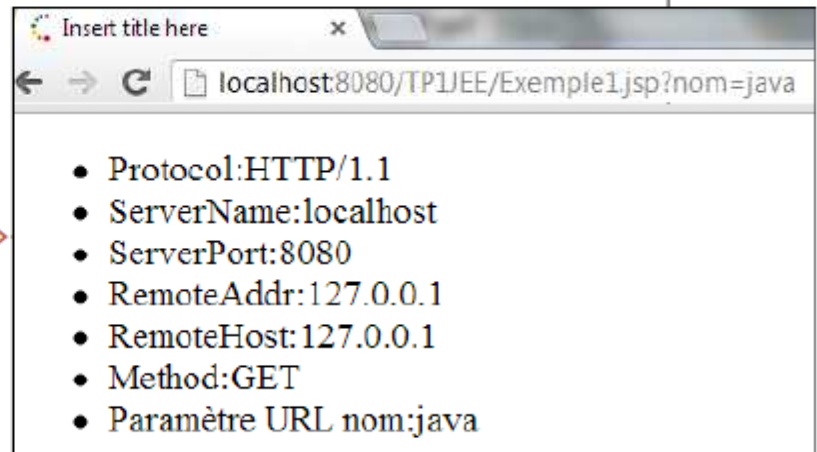
}

}
```



Page JSP équivalente : Exemple

```
<%@ 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>
```



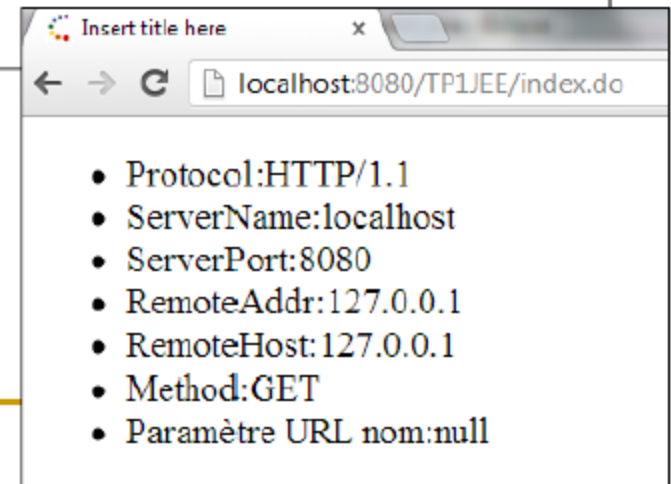
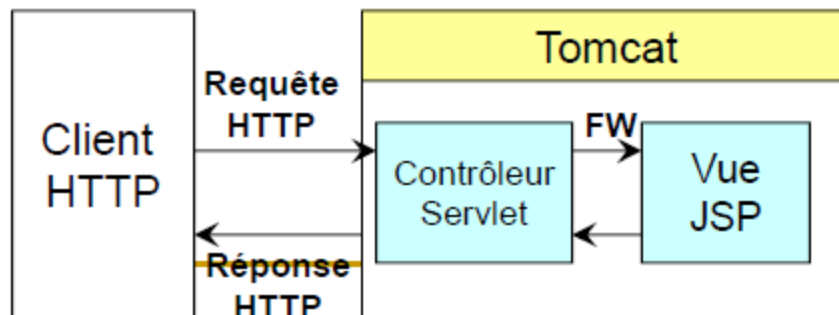
Forwarding : Exemple

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.*;

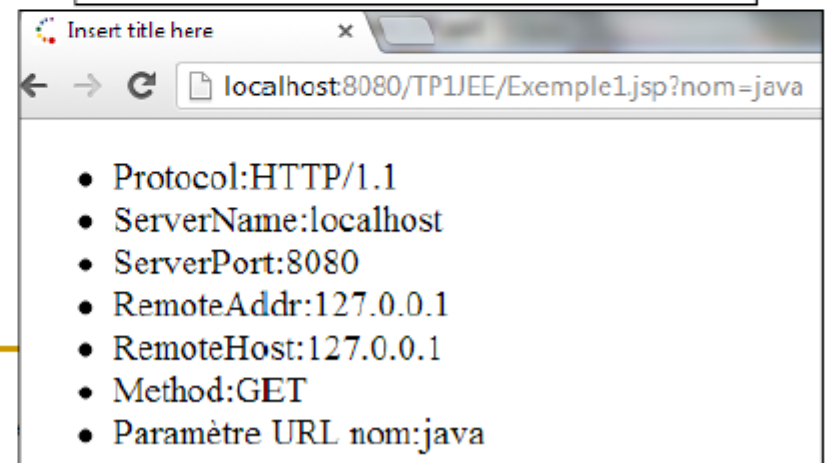
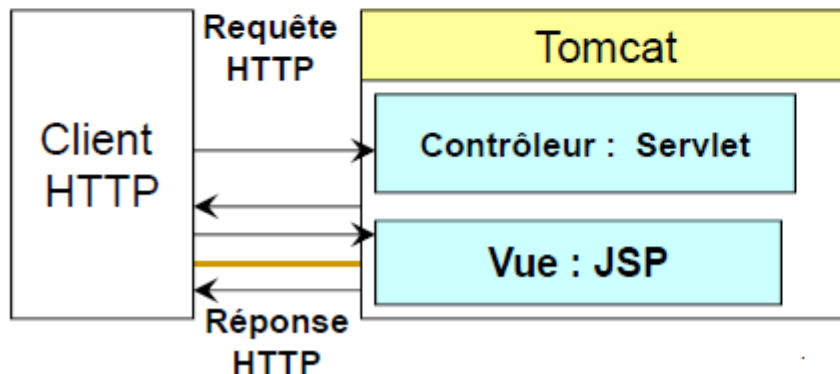
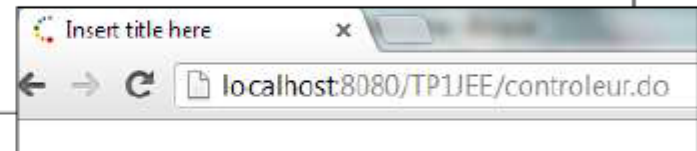
public class FirstServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        request.getRequestDispatcher("Exemple1.jsp").forward(request, response);
    }
}
```



Redirection : Exemple

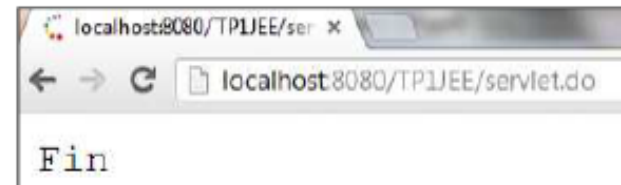
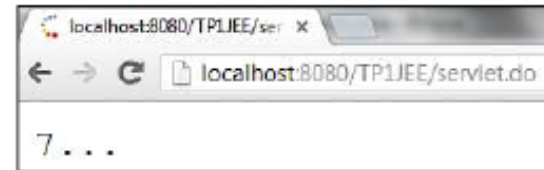
Une servlet peut rediriger vers une autre ressource locale ou distante en utilisant la méthode **sendRedirect()** de l'objet **response**.

```
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");  
    }  
}
```



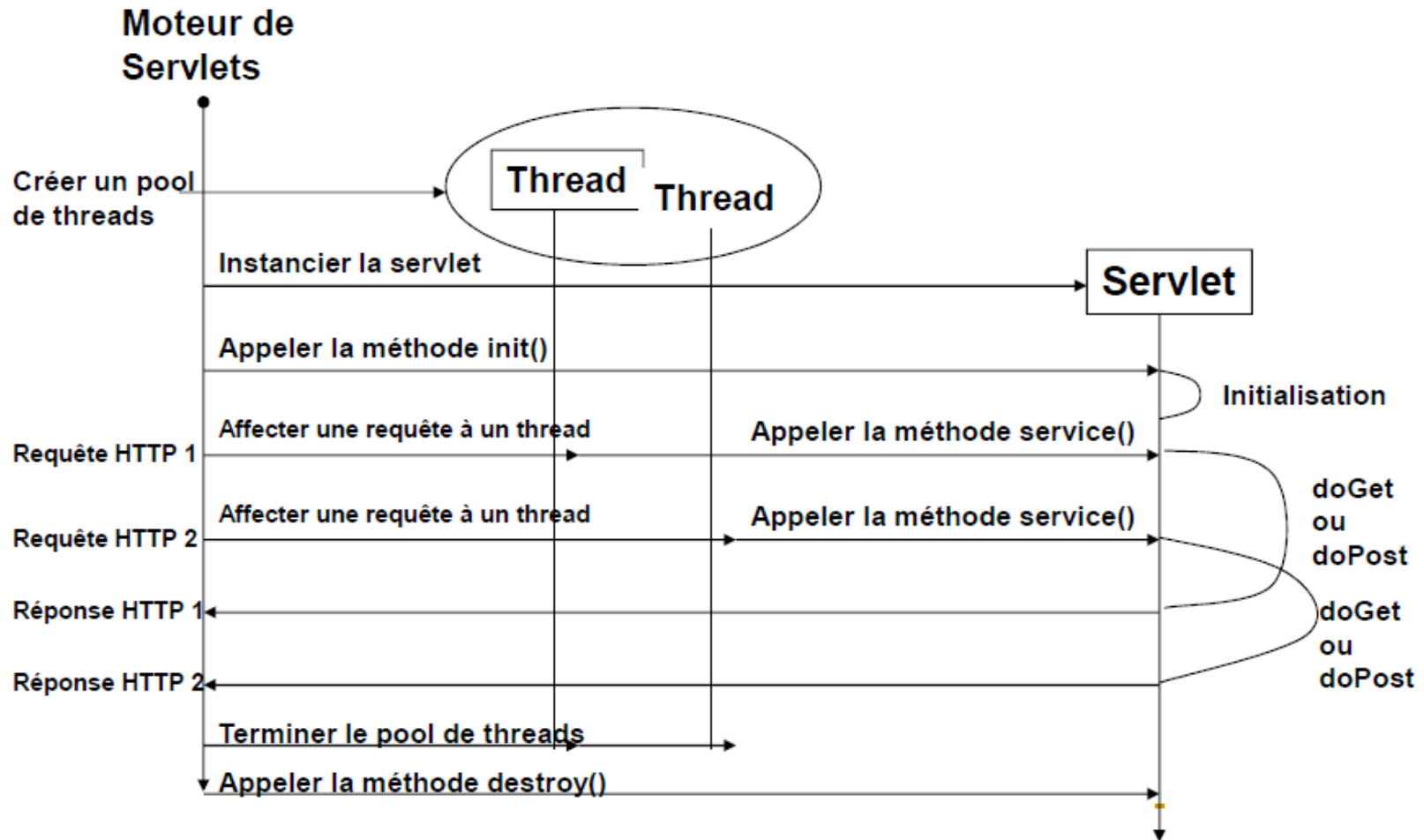
Effectuer un pull client : Exemple

```
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

Cycle de vie d'une Servlet



Cycle de vie d'une Servlet

- 1) Le serveur crée un pool de threads auxquels il va pouvoir affecter chaque requête
- 2) La servlet est chargée au démarrage du serveur ou lors de la première requête
- 3) La servlet est instanciée par le serveur
- 4) La méthode *init()* est invoquée par le conteneur
- 5) Lors de la première requête, le conteneur crée les objets Request et Response spécifiques à la requête.
- 6) La méthode *service()* est appelée à chaque requête dans un nouveau thread. Les objets *Request* et *Response* lui sont passés en paramètre
- 7) Grâce à l'objet *request*, la méthode *service()* va pouvoir analyser les informations en provenance du client
- 8) Grâce à l'objet *response*, la méthode *service()* va fournir une réponse au client
- 9) 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*.

Session

- L'objet **HttpSession** permet de mémoriser les données de l'utilisateur, grâce à une structure similaire à une table de hachage
- Cet Objet permet de relier chaque id de session à l'ensemble des informations relatives à l'utilisateur.
- L'objet *HttpSession* s'obtient grâce à la méthode *getSession()* de l'objet *HttpServletRequest*.

Gérer une session

La gestion des sessions se fait de la manière suivante :

- Obtenir l'ID de session
- Vérifier si une session est associé à l'ID
 - Si la session existe, obtenir les informations
 - Sinon
 - Générer un *ID de Session*
 - Enregistrer la session avec l'ID nouvellement créé

Obtenir une session

- La méthode `getSession()` de l'objet *HttpServletRequest* permet de retourner la session relative à l'utilisateur

HttpSession getSession(boolean create)

- L'argument *create* permet de créer une session lorsqu'il prend la valeur *true*.
-

Obtenir des informations d'une session

- Pour obtenir une valeur précédemment stockée dans l'objet *HttpSession*, il suffit d'utiliser la méthode *getAttribute()* de l'objet *HttpSession*.

Object *getAttribute("cle")*

- La méthode *getAttribute()* retourne un objet. Si l'attribut passé en paramètre n'existe pas, la méthode *getAttribute()* retourne la valeur *null*.

Obtenir des informations d'une session

```
public void doGet (HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
    HttpSession session = request.getSession(true); // Recupere la session
    int Age = (int)session.getAttribute("Age"); // Recupere l'age de l'utilisateur
    if (Age != null) {
        // ... faire quelque chose
        out = response.getWriter(); // Ecrit la reponse
    }
    else {
        Age = new Integer(...);
        // ... faire quelque chose d'autre
        out = response.getWriter(); // Ecrit la reponse
    }
}
```

Stocker des informations dans une session

- Le stockage d'informations dans la session est similaire à la lecture. Il suffit d'utiliser la méthode *setAttribute()* en lui fournissant comme attributs la clé et la valeur associée.

Object `setAttribute("cle","valeur")`
