

*Université Saad DAHLAB BLIDA*

*Faculté des Sciences*

*Département d'informatique*



**Licence 3- ISIL-**

## **Cours 4:**

# **Développement des Application Web**

## **JAVA EE: Servlet/JSP**

**Enseignant responsable: Dr. Nesrine LAHIANI**

# Application Web

## Les composants:

- Page Web
- Site Web
- Navigateur
- Serveur

C'est quoi?



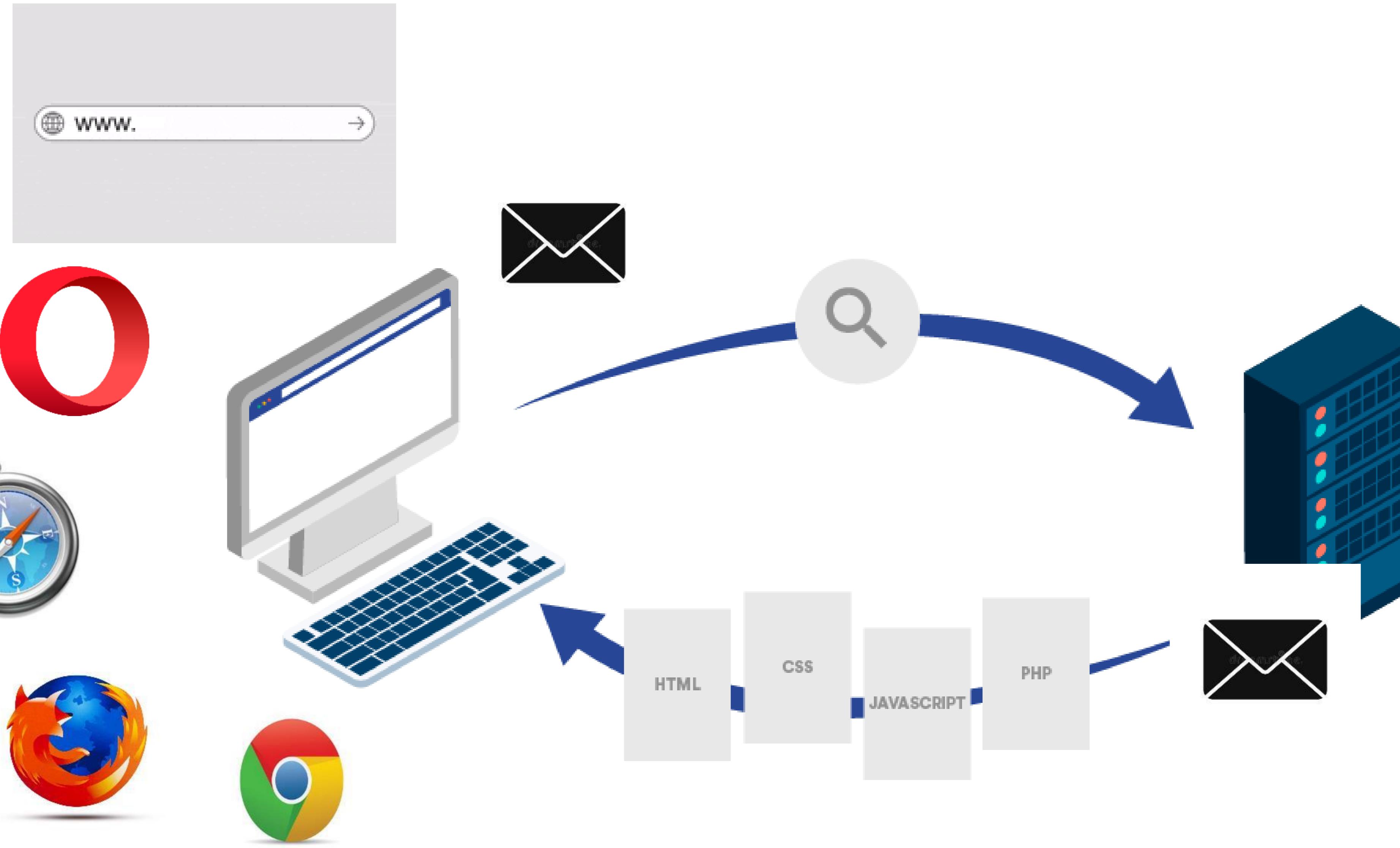
## Le contenu:

- Statique
- Dynamique

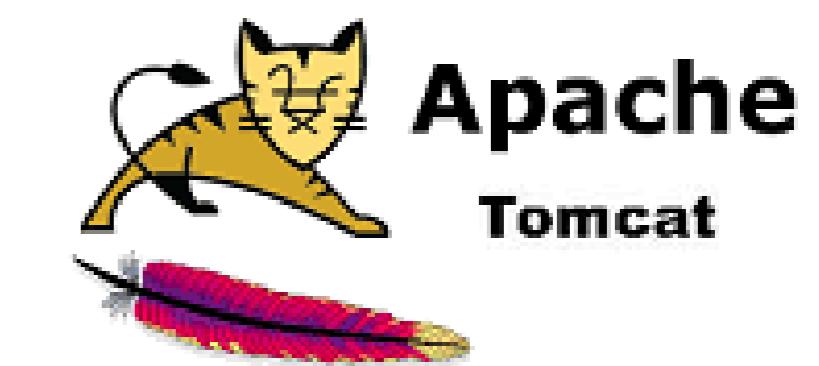
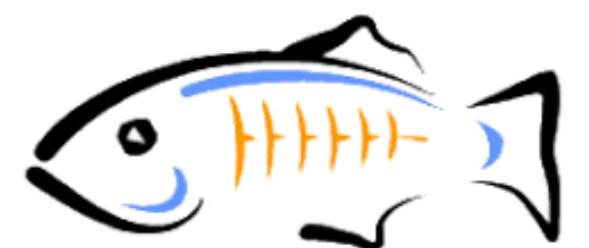
Comment ça marche?

# Comment Fonctionne une application web.?

3



GlassFish



# Backend vs frontend



## Front End

- Markup and web languages such as HTML, CSS and Javascript
- Asynchronous requests and Ajax
- Specialized web editing software
- Image editing
- Accessibility
- Cross-browser issues
- Search engine optimisation

## Back End

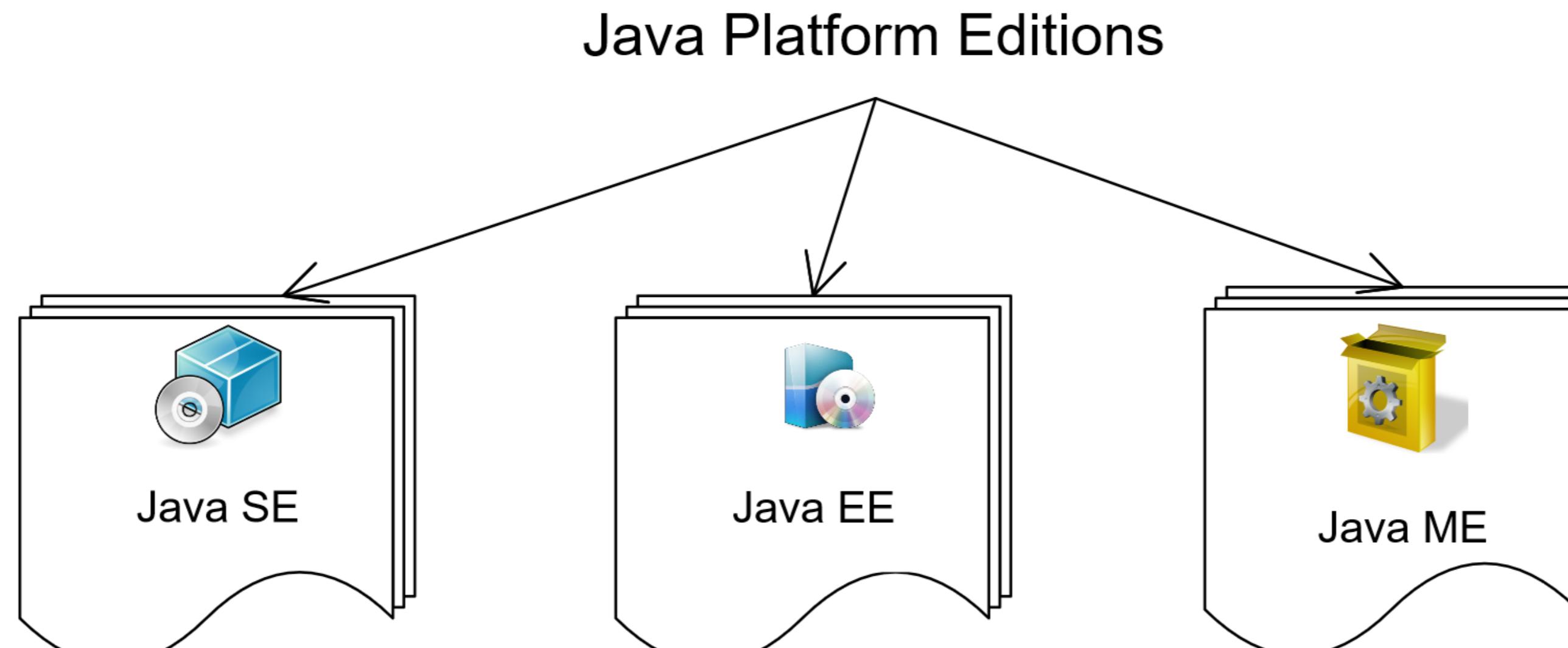
- Programming and scripting such as Python, Ruby and/or Perl
- Server architecture
- Database administration
- Scalability
- Security
- Data transformation
- Backup

# Java EE

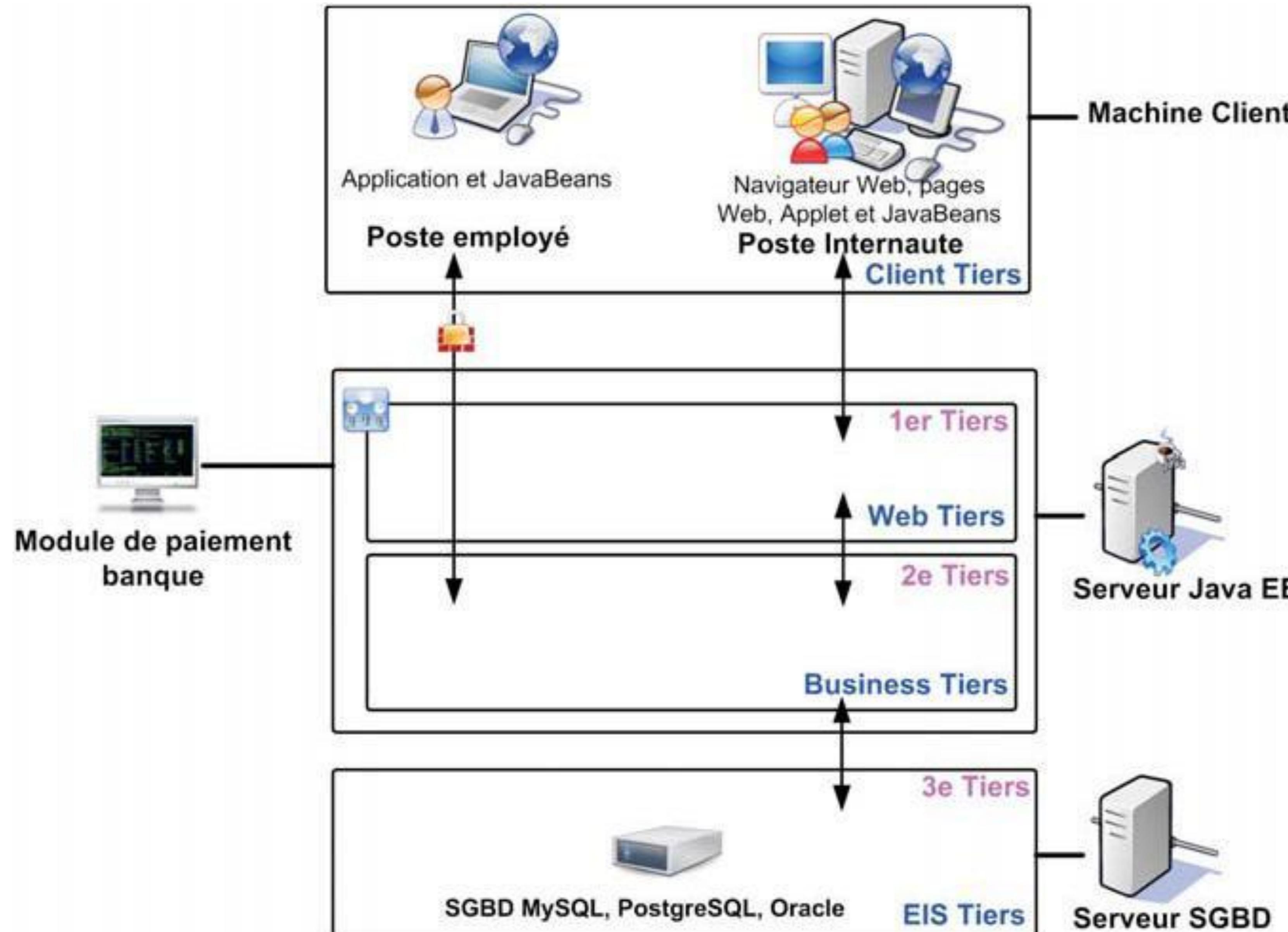
# Java Platform, Enterprise Edition (JEE)

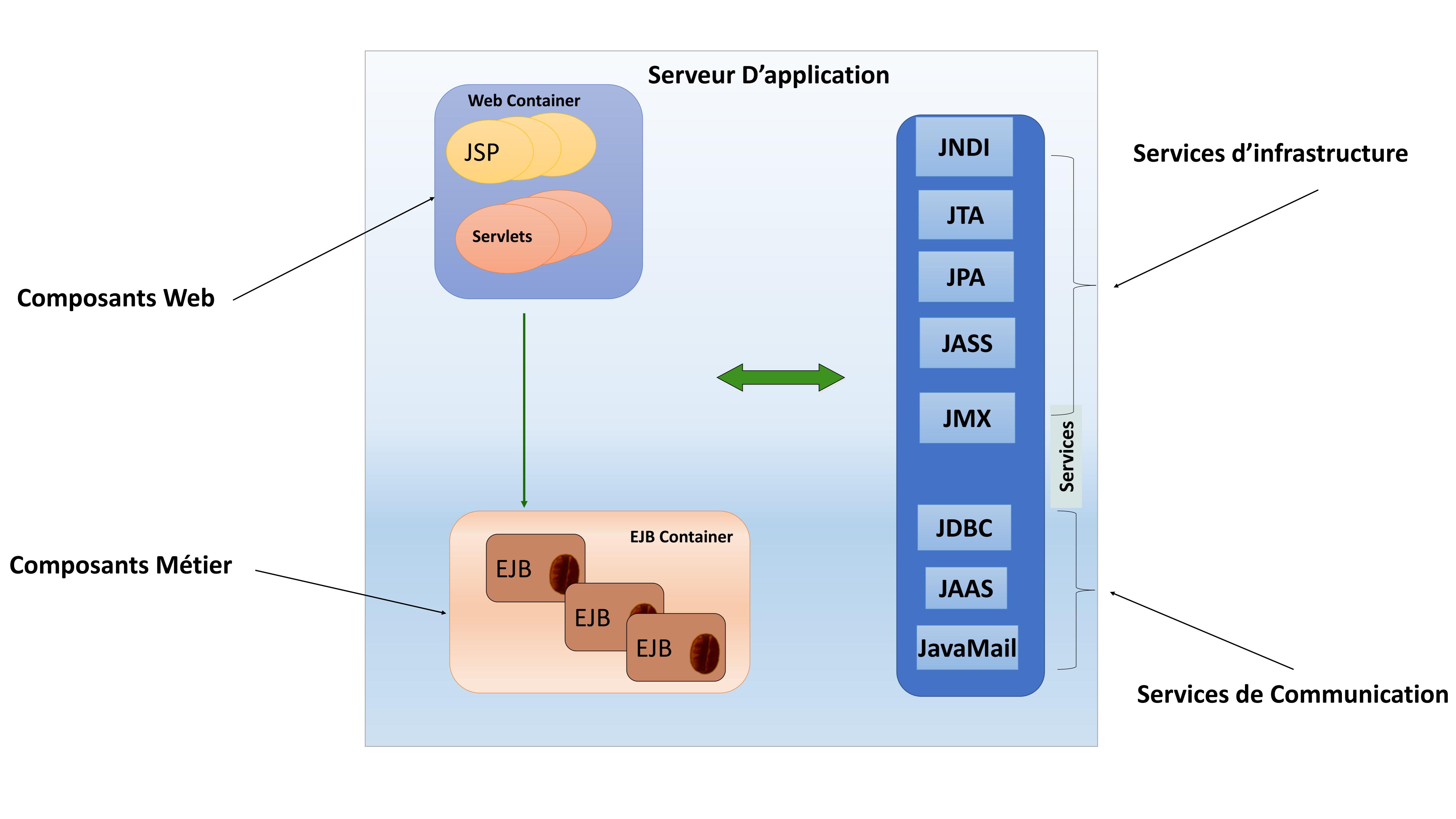
**Jakarta EE** (anciennement **JEE/ J2EE**) est une plate-forme fortement orientée serveur pour le développement et l'exécution d'applications distribuées. Elle est composée de deux parties essentielles :

- ❑ un ensemble de spécifications pour une infrastructure dans laquelle s'exécutent les composants écrits en Java : un tel environnement se nomme serveur d'applications.
- ❑ un ensemble d'API qui peuvent être obtenues et utilisées séparément. Pour être utilisées, certaines nécessitent une implémentation de la part d'un fournisseur tiers.



# Architecture JEE



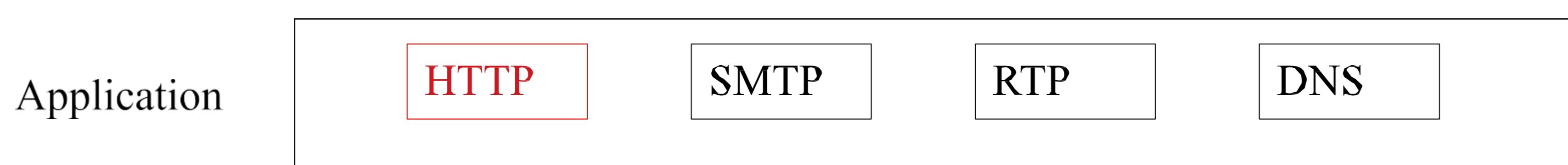


# HTTP

## le protocole de transfert hypertexte

# Qu'est-ce que c'est le protocole HTTP?

- D'une manière générale, un protocole est une convention acceptée par les parties communicantes sur la façon dont leur dialogue doit avoir lieu.
- Le protocole HTTP est **la langue native du Web**, celle parlée par les serveurs web.
- 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
- C'est un protocole de **requête-réponse** simple qui définit le **format** des messages échangés entre les clients et les serveurs web.
- Les requêtes et les réponses HTTP sont transportées grâce au protocole **TCP/IP**.
- Port standard du HTTP est **80** et du HTTPS est **443**.

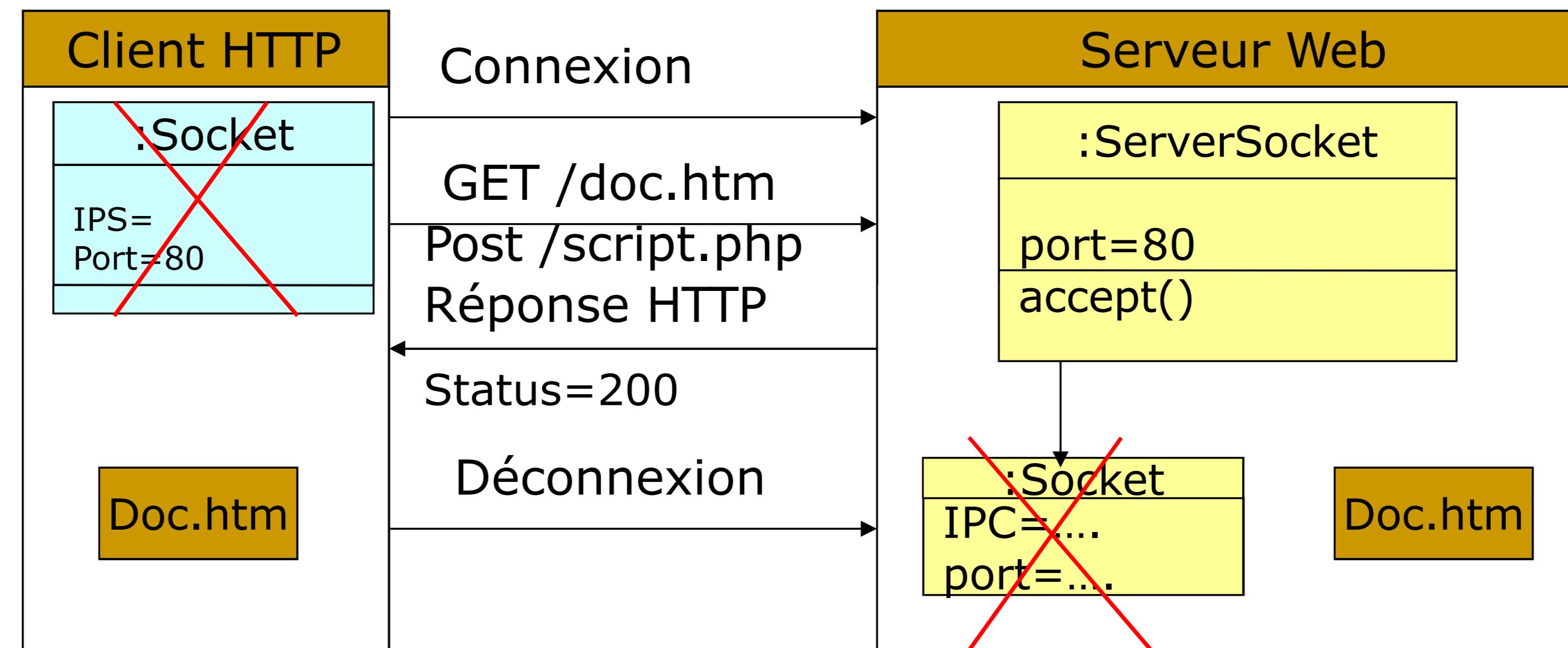


# Evolution

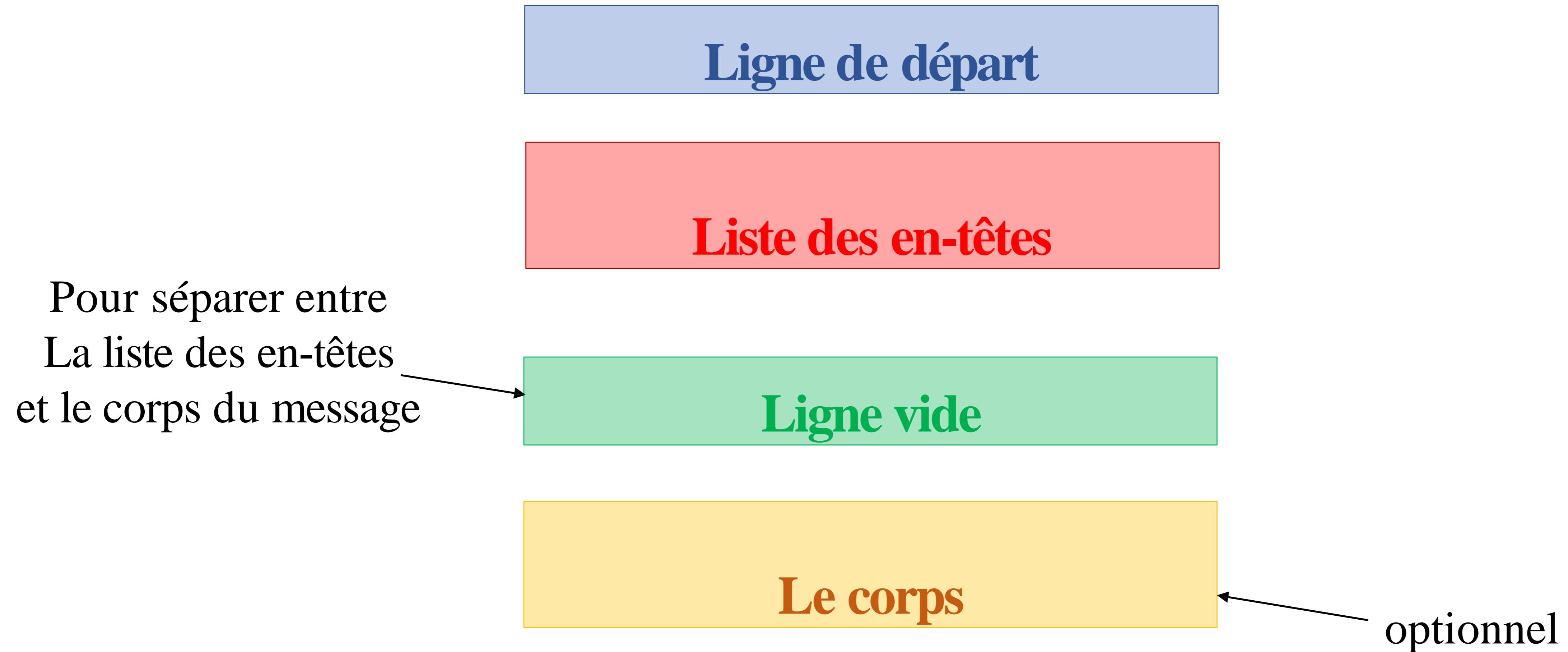
- HTTP/0.9, en 1991.
- HTTP/1.0, en 1996, (RFC 1945).
- **HTTP/1.1** (version présentée dans ce cours, celle la plus communément utilisée)
  - Version 1, en 1997, (RFC 2068).
  - Version 2, en 1999, (RFC 2616).
  - Version 3, en 2014, (RFC 7230 à 7237).
- HTTP/2.0, en 2015, (RFC 7540).
- HTTP/3.0 (Utilise, au lieu du protocole TCP, le protocole de transport QUIC qui est basé sur UDP).

# Fonctionnement

- 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



# Format d'un message HTTP



# 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...)

# Exemple de transaction: requête

- Le client contacte le serveur pour demander le document « index.php » :

```
GET /index.php HTTP/1.1
```

- Le client envoie des informations d'en-tête pour informer le serveur de sa configuration et des documents qu'il accepte:

```
User-Agent : Mozilla/5.0 (X11 ; Ubuntu ; Linux x86_64 ; rv :86.0)
Gecko/20100101 Firefox/86.0
Host : www.univ-blida.dz
Accept : text/html,application/xhtml+xml,application/xml ;
... etc
```

- Le client envoie une ligne vide (fin de l'en-tête) et un contenu vide dans cet exemple (Méthode GET).

# Exemple (1)

Chemin d'accès au document cible

Version HTTP

Nom de la méthode

Un seul espace

GET /page.html HTTP/1.1 ↵

Accept: text/html,application/xhtml+xml,image/jxr/\*/\* ↵

Accept-Encoding: gzip, deflate, br ↵

Accept-Language: \* ↵

Connection: keep-alive ↵

User-Agent: MonNavigateur 1.1 ↵

Host: www.example.com ↵ ↵

Nom en-tête

Deux points de séparation

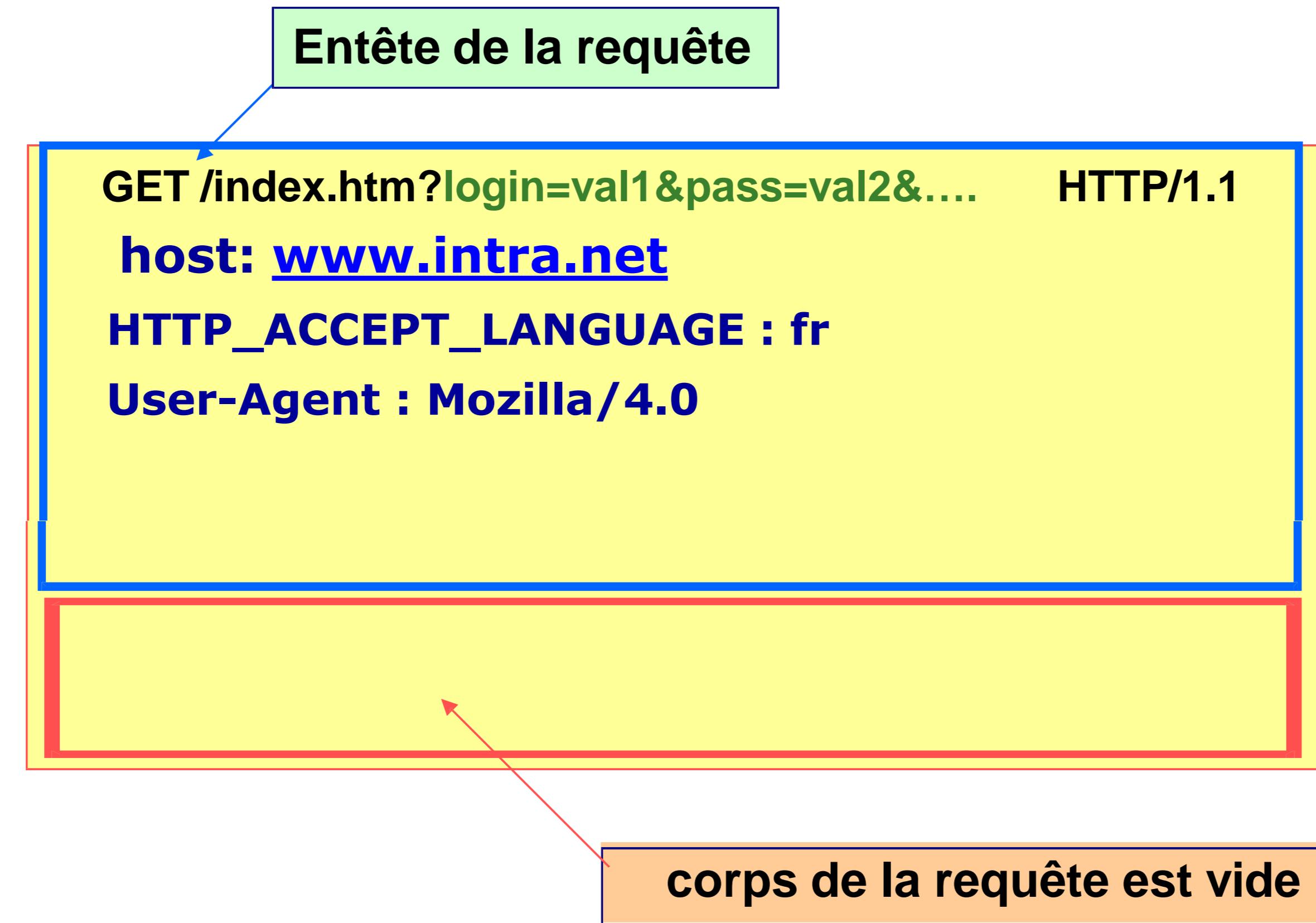
Valeur en-tête

Un saut de ligne

The diagram illustrates the structure of an HTTP request message. At the top, three main components are labeled: 'Chemin d'accès au document cible' (Path to target document), 'Version HTTP' (HTTP version), and 'Nom de la méthode' (Method name). Below these, the request line 'GET /page.html HTTP/1.1 ↵' is shown. An arrow from 'Un seul espace' (a single space) points to the space between 'GET' and '/page.html'. Another arrow from 'Nom de la méthode' points to 'GET'. To the right of the path, an arrow from 'Version HTTP' points to 'HTTP/1.1'. Below the request line, several header fields are listed: 'Accept', 'Accept-Encoding', 'Accept-Language', 'Connection', 'User-Agent', and 'Host'. Arrows from 'Nom en-tête' point to the first four fields, and an arrow from 'Valeur en-tête' points to 'Host'. Arrows from 'Deux points de séparation' point to the colon in 'Accept:' and the colon in 'Host:'. An arrow from 'Un saut de ligne' points to the blank line after 'Host:'.

# Exemple (2)

Le client envoie la requête : Méthode GET



# La méthode de base du HTTP **GET** permet de :

1. Demander une ressource, par exemple une page html, du serveur Web. Par exemple :

```
GET /index.php HTTP/1.1  
Host: www.exemple.com
```

Demander au serveur  
d'envoyer la page d'accueil  
index.php

2. d'envoyer des informations supplémentaires (saisies dans un formulaire par exemple) que le serveur doit traiter. Ces paramètres sont envoyés via l'URL de la façon suivante :

- Ces paramètres sont introduits après un « ? ».
- Chaque paramètre prend la forme : «nom=valeur».
- Si plusieurs paramètres sont introduits alors ils sont séparés par « & ».
- Exemple:

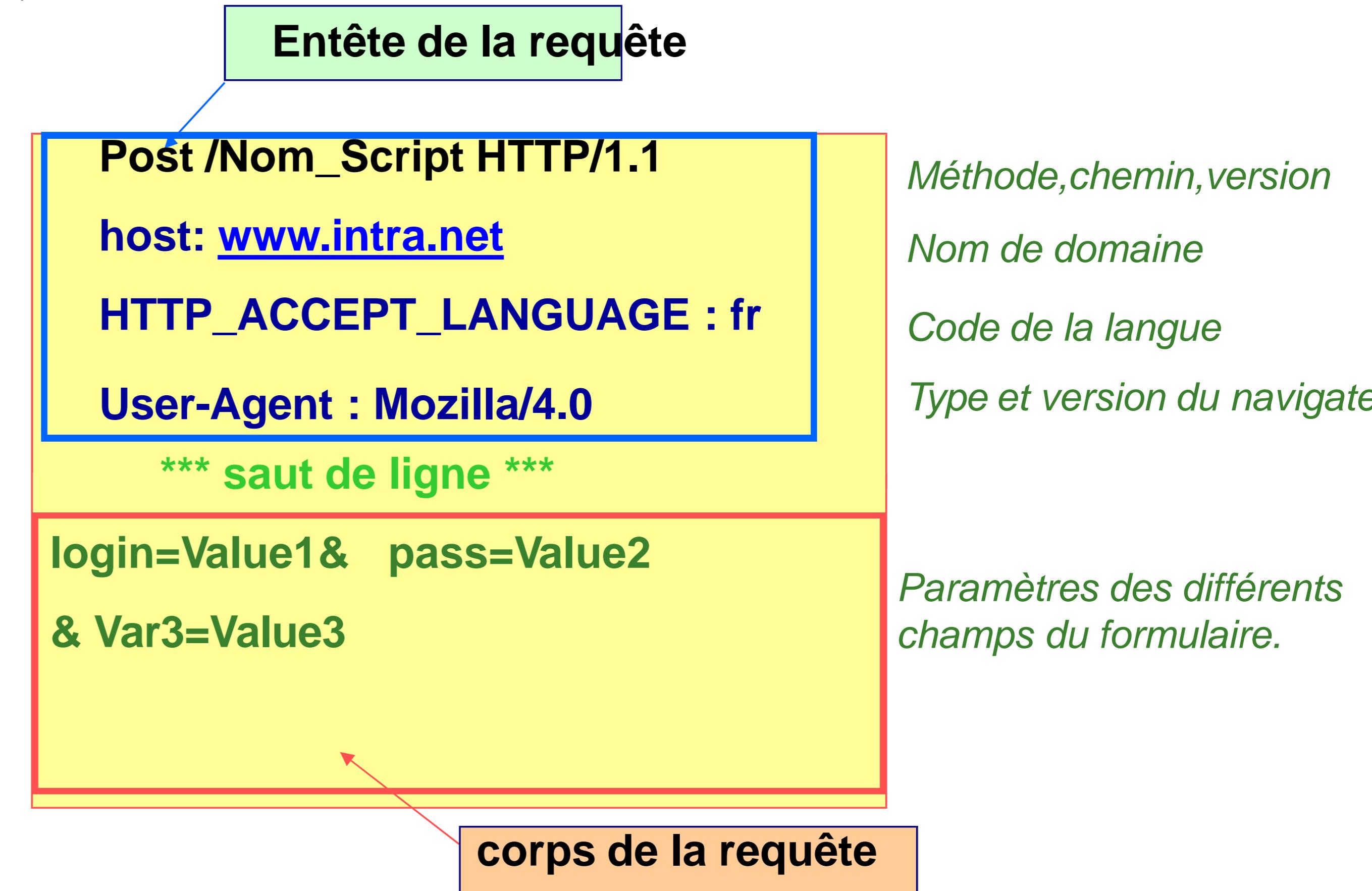
```
GET /CalculerMoy.php?note1=10&note2=15.5&note3=17 HTTP/1.1  
Host: www.exemple.com
```

**Remarque :**Avec GET, le contenu de la requête est toujours **vide**.

# Exemple (3)

La méthode GET n'est pas idéale si vous voulez envoyer, par exemple, des informations confidentielles au serveur, car, ces informations sont écrites ouvertement dans la barre d'adresse du navigateur.

- Dans ce cas, il est préférable d'utiliser la méthode «POST ».
- Cette méthode n'ajoute aucun paramètre à l'URL,
- mais elle les envoie dans le corps/contenu de la requête.



# Exemple de transaction: réponse

- Le serveur répond en commençant par indiquer par un code, l'état de la requête:

HTTP/1.1 200 OK

- Le serveur envoie un ensemble d'en-têtes qui donnent des informations sur lui-même et le document demandé:

Date : Sat, 15 Feb 2021 23:26:48 GMT

Server : Apache

Last-Modified : Sat, 15 Feb 2021 23:26:48 GMT

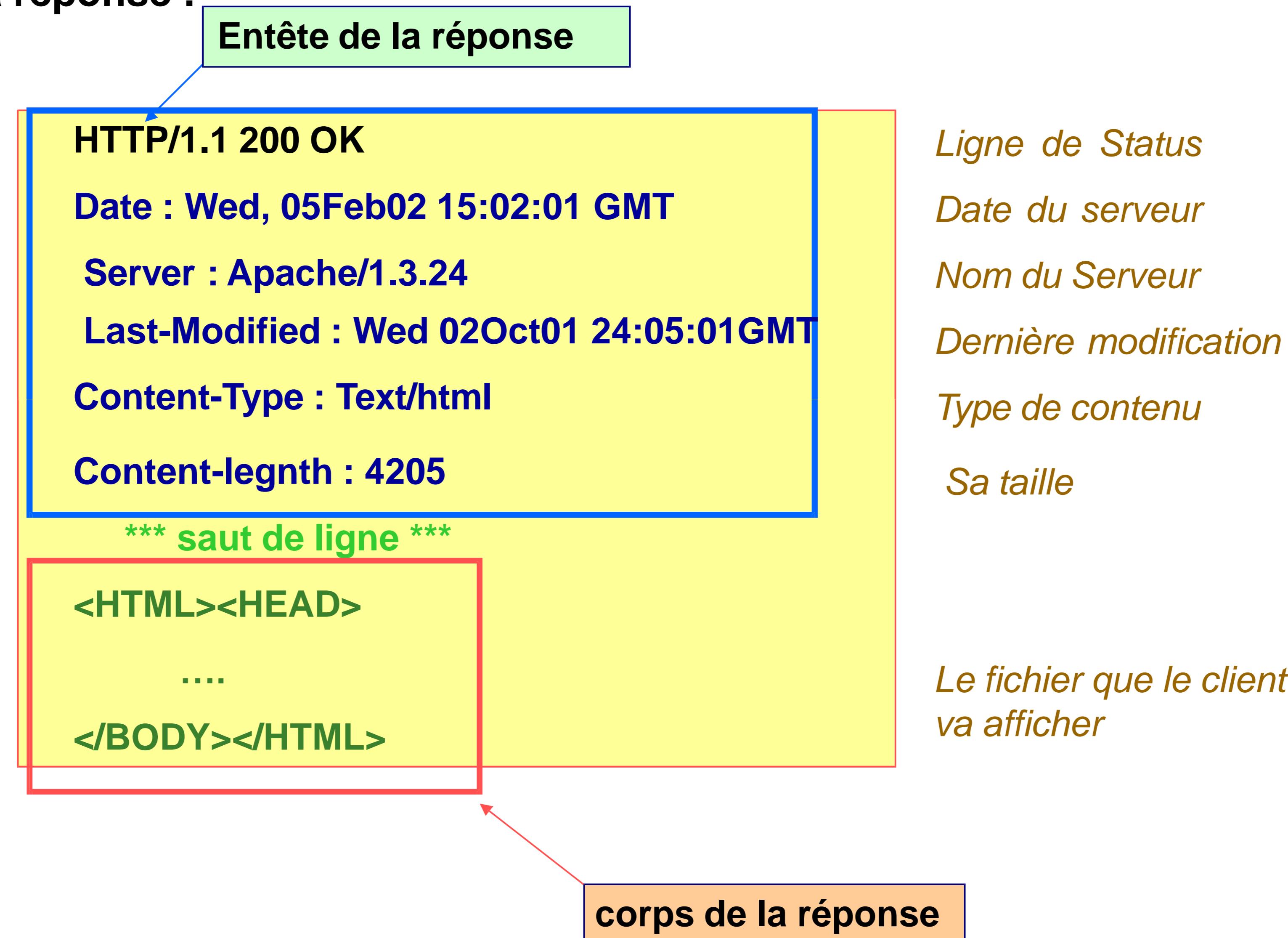
Content-Type : text/html ;

...etc

- Puis une ligne vide (fin de l'en-tête) et le contenu du document si la requête a réussi.

# Exemple

Le Serveur retourne la réponse :



# Les en-têtes d'une requête (1)

En-tête	Description
Connection	Permet d'indiquer si la connexion TCP/IP doit être maintenue (Keep-Alive) ou pas (Close).
Content-Length	Définit la taille en nombre d'octets du corps de la requête ou de la réponse.
Content-Type	Définit le type de média correspondant au corps de la requête ou de la réponse.
Date	Définit la date à laquelle la requête ou la réponse a été générée.

# Les en-têtes d'une requête (2)

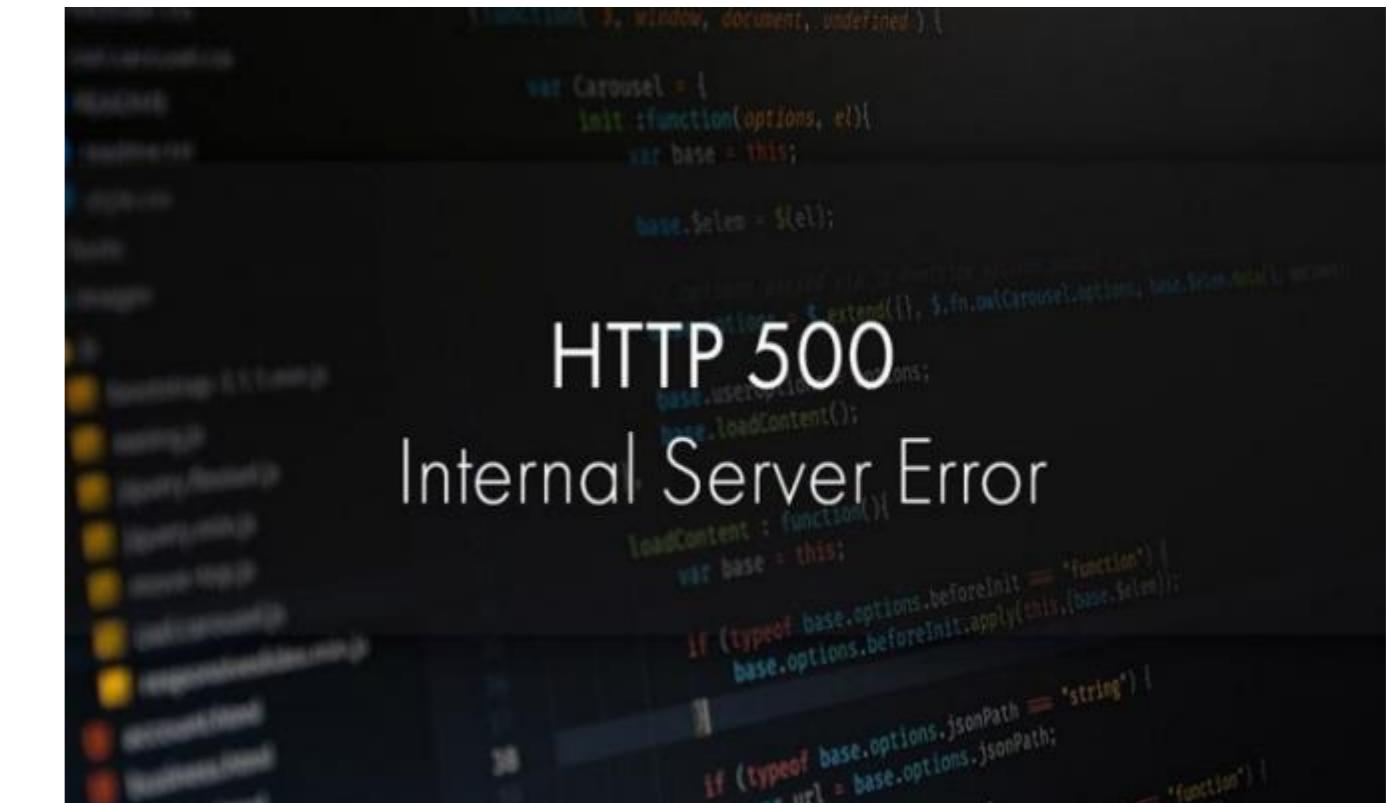
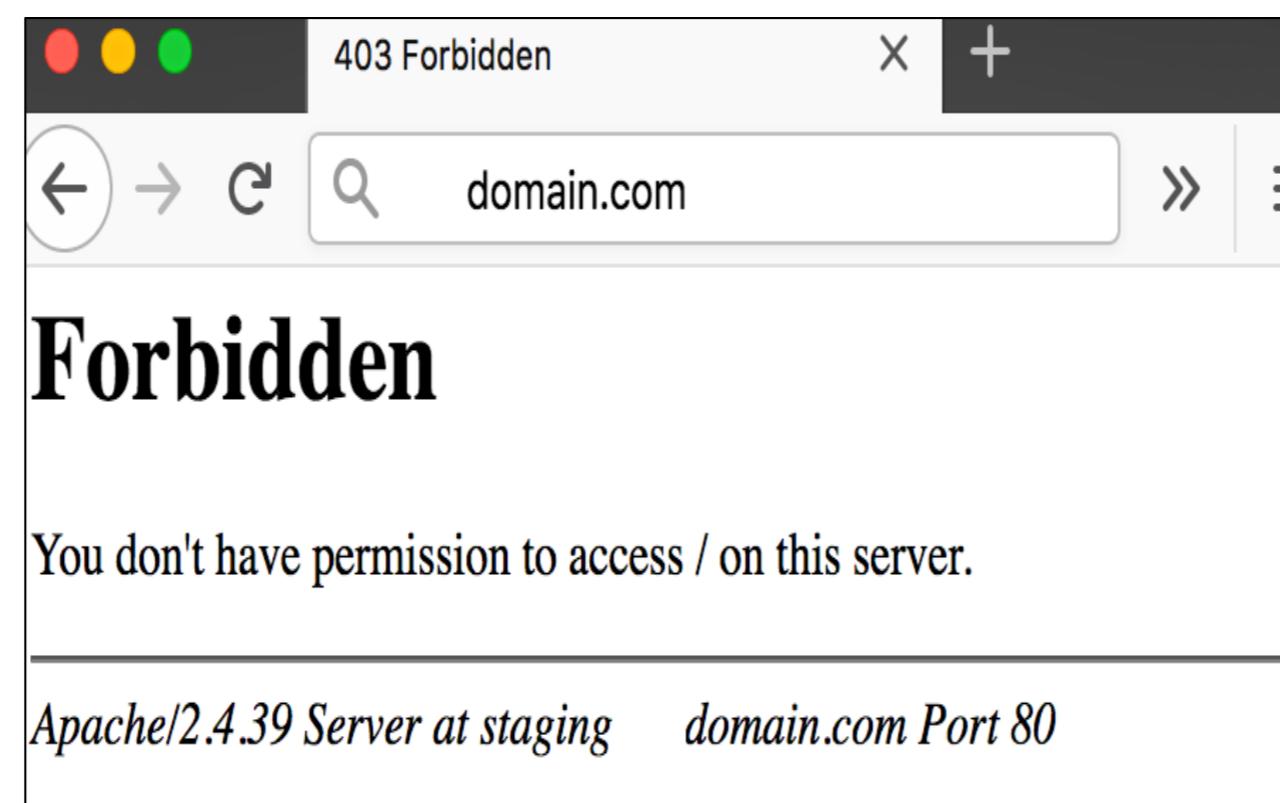
En-tête	Description
Accept	Définit le ou les types de médias que le client accepte.
Accept-Encoding	Définit les méthodes de compression acceptées par le client.
Accept-Language	Définit les langues préférées par le client selon un ordre de priorité.
User-Agent	Définit les informations du logiciel client à partir duquel la requête est émise.
Cookie	Permet d'envoyer au serveur tous les cookies associés au nom de domaine de la requête.

# Les en-têtes d'une réponse

En-tête	Description
Content-Encoding	Définit la méthode de compression du corps de la réponse.
Content-Language	Définit la langue utilisée dans le corps de la réponse.
Server	Définit les caractéristiques du serveur ayant généré la réponse.
Set-Cookie	Permet au serveur d'envoyer des cookies au client.

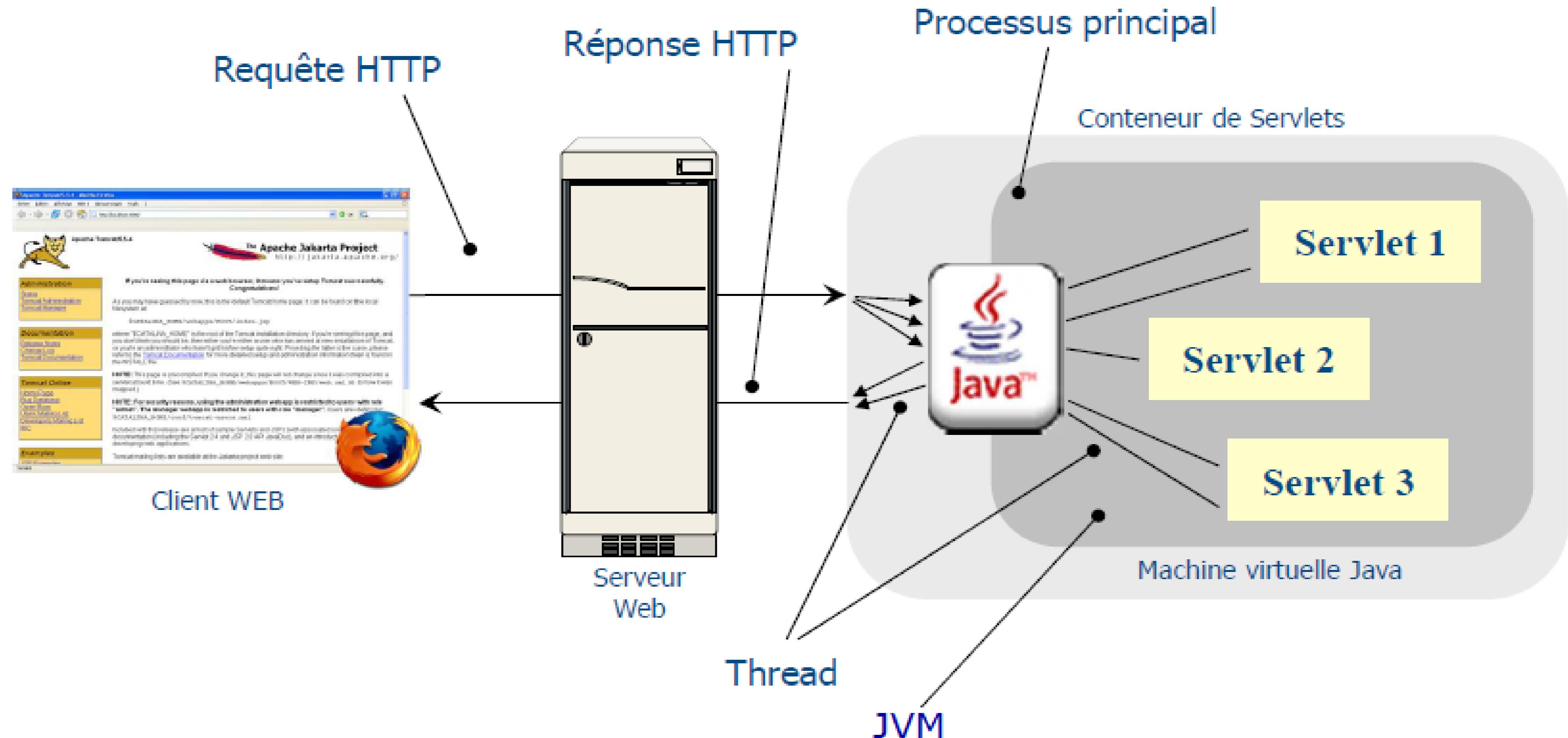
# Groupes de code de réponse

Code	Signification	Exemple
1xx	Information	100=le serveur accepte de traiter la requête du client
2xx	Succès	200=la requête a réussi; 204=indique qu'il n'y a pas de contenu à envoyer.
3xx	Redirection	301=indique que l'URL de la ressource demandée a été modifiée;
4xx	Erreur client	403=page interdite; 404=page inexistante.
5xx	Erreur serveur	500=erreur interne; 503=requête à tenter plus tard



# Les servlets

# Qu'est ce qu'une Servlet



- Un Composant Java
- Permet de Gérer une requête HTTP et Fournier une réponse HTTP au client
- Une servlet s'exécute dans un conteneur de Servlet

# Développement d'une servlet

1. Une servlet est une classe qui hérite de la classe HttpServlet

```
public class Servlet extends HttpServlet {
```

- 2- Si la méthode utilisée est GET, il suffit de redéfinir la méthode :

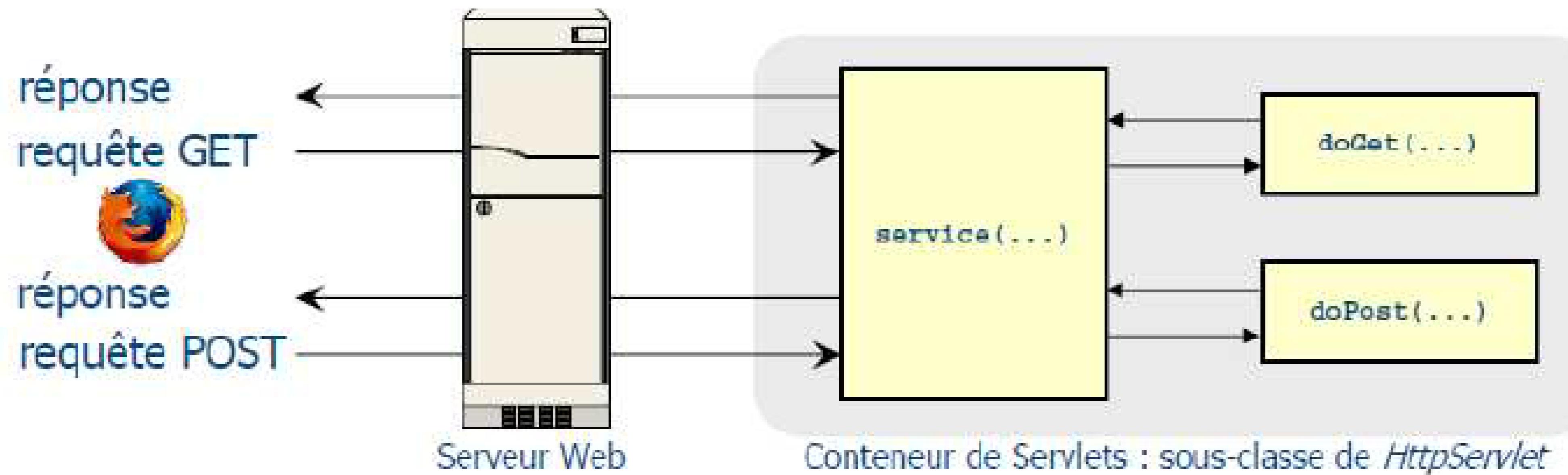
```
public void doGet( HttpServletRequest request, HttpServletResponse response)
```

3. Si la méthode utilisée est POST, il suffit de redéfinir la méthode :

```
public void doPost( HttpServletRequest request, HttpServletResponse response)
```

# Gestion des servlets

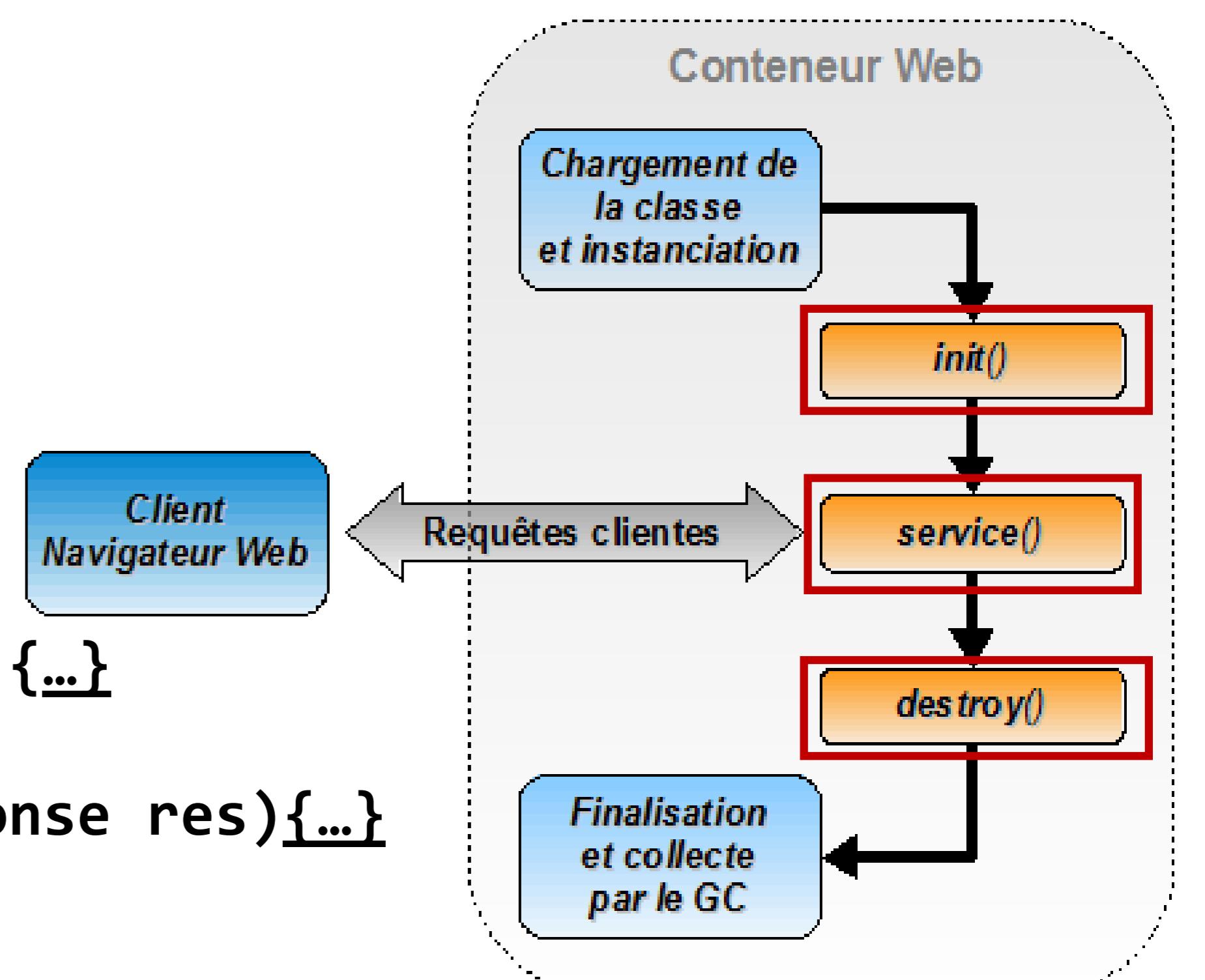
1. La servlet redéfinit la méthode `service(...)`
2. `service(...)` lit la méthode (GET, POST, ...) à partir de la requête
3. Elle transmet la requête à une méthode appropriée de `HttpServlet` destinée à traiter le type de requête (GET, POST, ...)



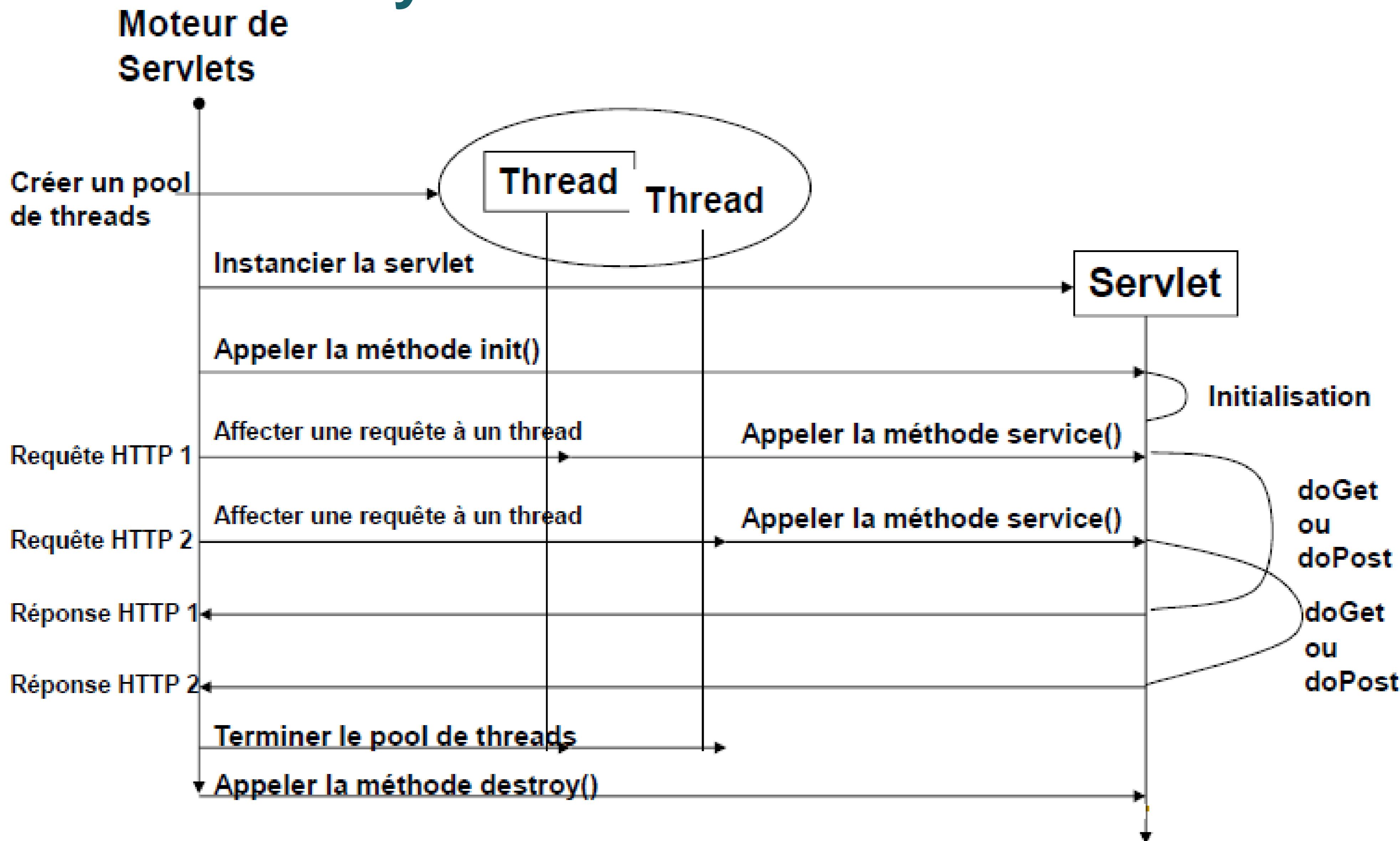
# Cycle de vie d'une Servlet

1. **Création et initialisation** de la Servlet Utilisation de paramètres initiaux **init ()**
2. **Traitement** des services demandés par les clients (au travers de la méthode **service(...)**) notamment), le cas échéant Persistance d'instance
3. **Destruction** de la Servlet **destroy()**

```
import javax.servlet.*;
import javax.servlet.http.*;
public class SimpleServlet extends HttpServlet {
    public void init(ServletConfig c) throws ServletException {...}
    public void doXXX(HttpServletRequest req, HttpServletResponse res){...}
    public void destroy() {...}
}
```



# Cycle de vie d'une Servlet



# Ma Première Servlet

Ne pas oublier d'importer la bibliothèque Java des Servlets

HelloWorld est un objet de type HttpServlet

Redéfinition de la méthode doGet (traitement d'une requête GET)

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

    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();

        out.println("<HTML>");
        out.println("<HEAD><TITLE>Bonjour tout le monde</TITLE></HEAD>");
        out.println("<BODY>");
        out.println("<BIG>Bonjour tout le monde</BIG>");
        out.println("</BODY></HTML>");
    }
}
```

Réponse sous format HTML

*HelloWorld.java du projet  
HelloWorldServlet*

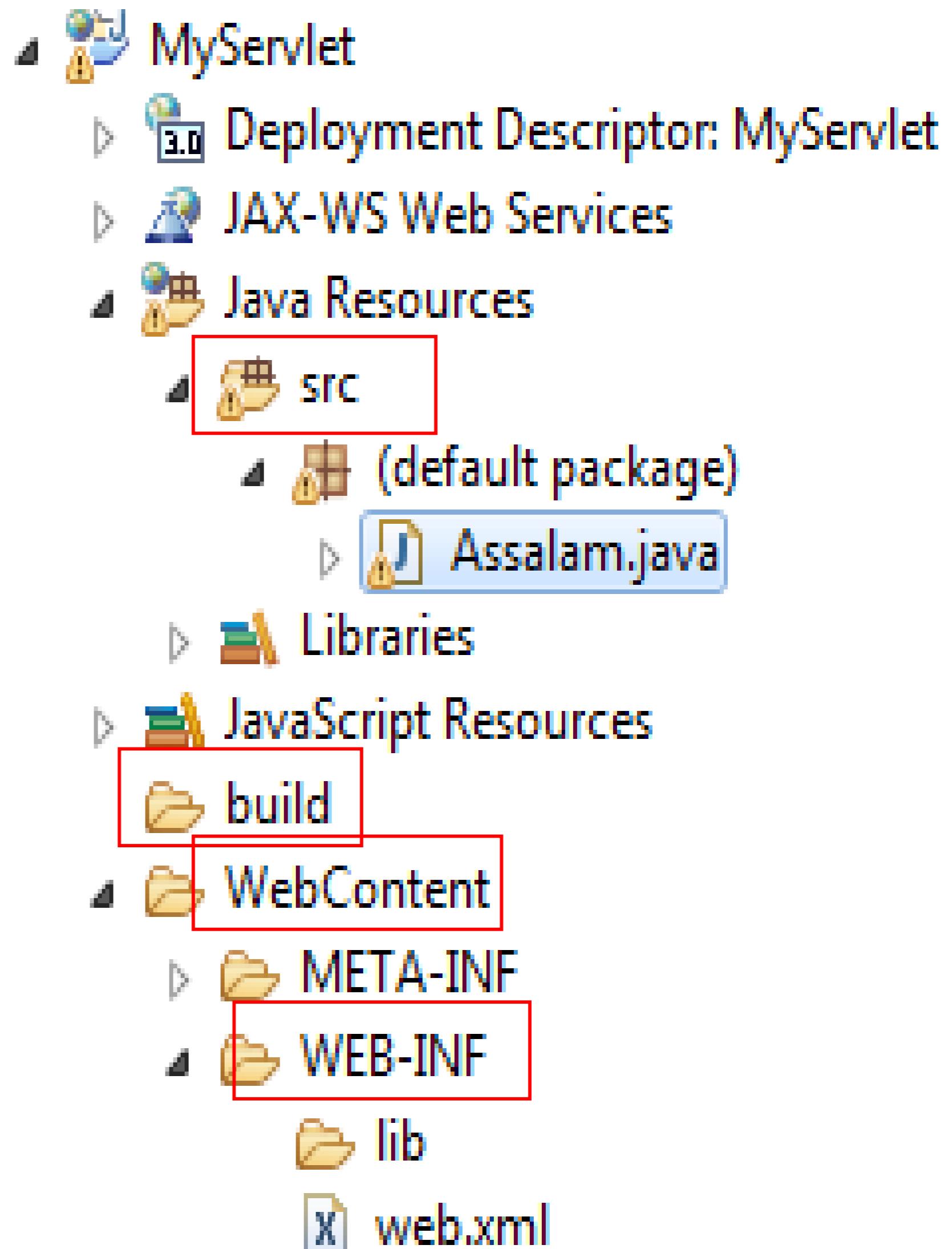
Le résultat sur le client



# Structure d'un Projet JEE

33

- 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



# 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.

```
<?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>
```

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

URL associée à la servlet

# HttpServletRequest

## Exemple : Quelques méthodes de l'objet request

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

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        PrintWriter out=response.getWriter();
        response.setContentType("text/html");
        out.println("<html>");
        out.println( "<body>");
        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("</body>");
        out.println( "</html>");
    }
}
```

# Les Servlets

## (La lecture d'une requête)

Points qui seront évoqués:

- La lecture des informations relatives à l'URL et aux en-têtes.
- Lecture des paramètres.

# Lecture des paramètres

- Savoir récupérer les paramètres envoyés dans une requête donnée est très important.
- Ces paramètres peuvent correspondre aux informations saisies par un utilisateur dans un formulaire.
- Comme nous avons déjà vu, Dans le cas d'une requête GET, ces paramètres sont écrits au niveau de l'URL sous la forme suivante :

URL valide ?param1=valeur 1&param2=valeur 2 . . .

- Dans le cas d'une requête de type POST, ces paramètres sont passés dans le corps de la requête au lieu de l'URL sous la même forme précédente.

# Lecture des paramètres (1)

- Pour illustrer ce principe, prenons l'exemple d'un formulaire HTML simple qui permet à des développeurs de postuler leur candidature pour des postes de développement. Ce formulaire est implémenté par le fichier « **formulaireAjoutCandidat.html** » dont le contenu (partie relative au formulaire) est présenté dans le slide suivant.
- L'accès à ce formulaire se fait via l'URL suivante :

**<http://localhost:8080/MyFirstWebProject/formulaireAjoutCandidat.html>**

# Lecture des paramètres(2)

```
<table>
  <form method="get" action="/MyFirstWebProject/AjoutCandidat">
    <tr>
      <td>Nom</td>
      <td><input type="text" name="nom"></td>
    </tr>
    <tr>
      <td>Prénom</td>
      <td><input type="text" name="prenom"></td>
    </tr>
    <tr>
      <td>Expertise</td>
      <td><select name="expertise" multiple>
          <option value="Web">Web</option>
          <option value="Logiciel">Logiciel</option>
          <option value="Mobile">Mobile</option>
          <option value="Front-end">Front-end</option>
          <option value="Back-end">Back-end</option>
          <option value="Full-stack">Full-stack</option>
        </select>
      </td>
    </tr>
    <tr>
      <td></td>
      <td style="text-align: right"><input type="submit" value="Ok"></td>
    </tr>
  </form>
</table>
```

# Lecture des paramètres (4)

Le résultat d'exécution est le suivant :

The screenshot shows a web browser window titled "Exemple d'un formulaire". The address bar displays "localhost:8080/MyFirstWeb". The form contains three fields:

- Nom:** Benmohamed
- Prénom:** Mohamed
- Expertise:** A dropdown menu with the following options:
  - Web (selected)
  - Logiciel
  - Mobile
  - Front-end

An "Ok" button is visible at the bottom right of the form area.

# Lecture des paramètres (5)

L'objet « **HttpServletRequest** » permet la lecture de ces paramètres dans une servlet. Les méthodes suivantes peuvent être utilisées :

- **getParameter(String name)** : cette méthode retourne, sous la forme d'une chaîne de caractères, la valeur du paramètre qui est passé en paramètre. Si aucun paramètre qui correspond au nom passé n'existe, la méthode retourne null. Dans le cas de l'existence de plusieurs paramètres avec le même nom, la première valeur est retournée.
- **getParameterValues(String name)** : cette méthode retourne un tableau de chaînes de caractères regroupant toutes les valeurs qui correspondent au nom passé en paramètre.

# Lecture des paramètres

- Revenons au formulaire précédent. Ce dernier pointe vers l'URL suivante qui est associée avec une nouvelle servlet nommée «AjoutCandidat» (`/MyFirstWebProject/AjoutCandidat`).
- Nous voulons afficher les informations saisies par l'utilisateur sur la console. Pour cela, voici le code de sa méthode «`doGet(...)`» :

```
protected void doGet ( HttpServletRequest request , HttpServletResponse response )
throws ServletException , IOException {
PrintWriter out=response.getWriter();
response.setContentType("text/html");
out.println("<html>");
out.println( "<body>");

String nom = request . getParameter ( "nom" ) ;
String prenom = request . getParameter " prenom ";
String expertise = request . getParameterValues " expertise " ;
out.println("<li>Nom: " + nom+"</li>");
out.println("<li>Prénom: " +prenom +"</li>");
out.println("<li>Prénom: " + +Arrays . toString ( expertise ) +"</li>");

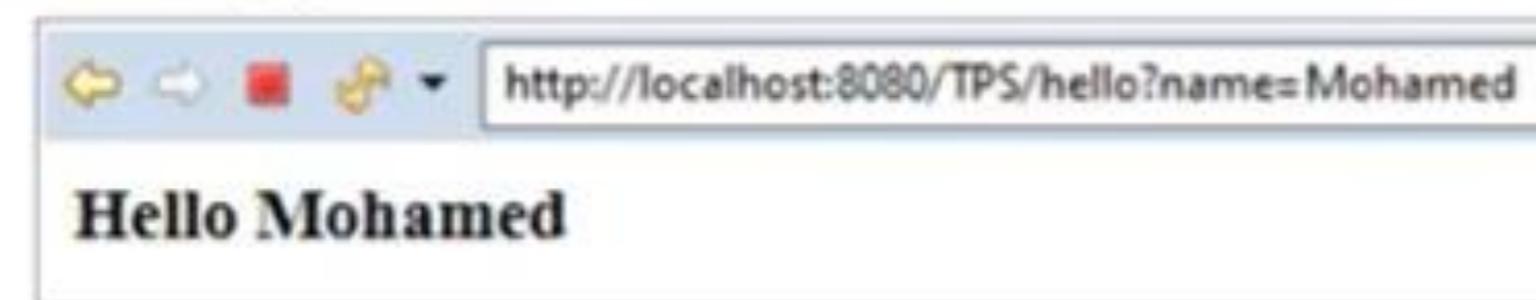
out.println( "</body>");
out.println("</html>");}
```

JSP  
Java Server Pages

# Servlet Vs JSP (Java Server Pages)

## Servlet

```
package web; import java.io.*; import javax.servlet.*; import  
javax.servlet.http.*;  
  
public class ControleurServlet extends HttpServlet {  
    @Override  
    protected void doGet(HttpServletRequest request,  
                         HttpServletResponse response) throws ServletException, IOException  
    {  
        response.setContentType("text/html");  
        PrintWriter out=response.getWriter();  
        String name=request.getParameter("name");  
        out.println("<html><head><title>Hello  
Servlet</title></head>");  
        out.println("<body>");  
        out.println("<h3>Hello "+name+"</h3>");  
        out.println("</body></html>");  
    }  
}
```

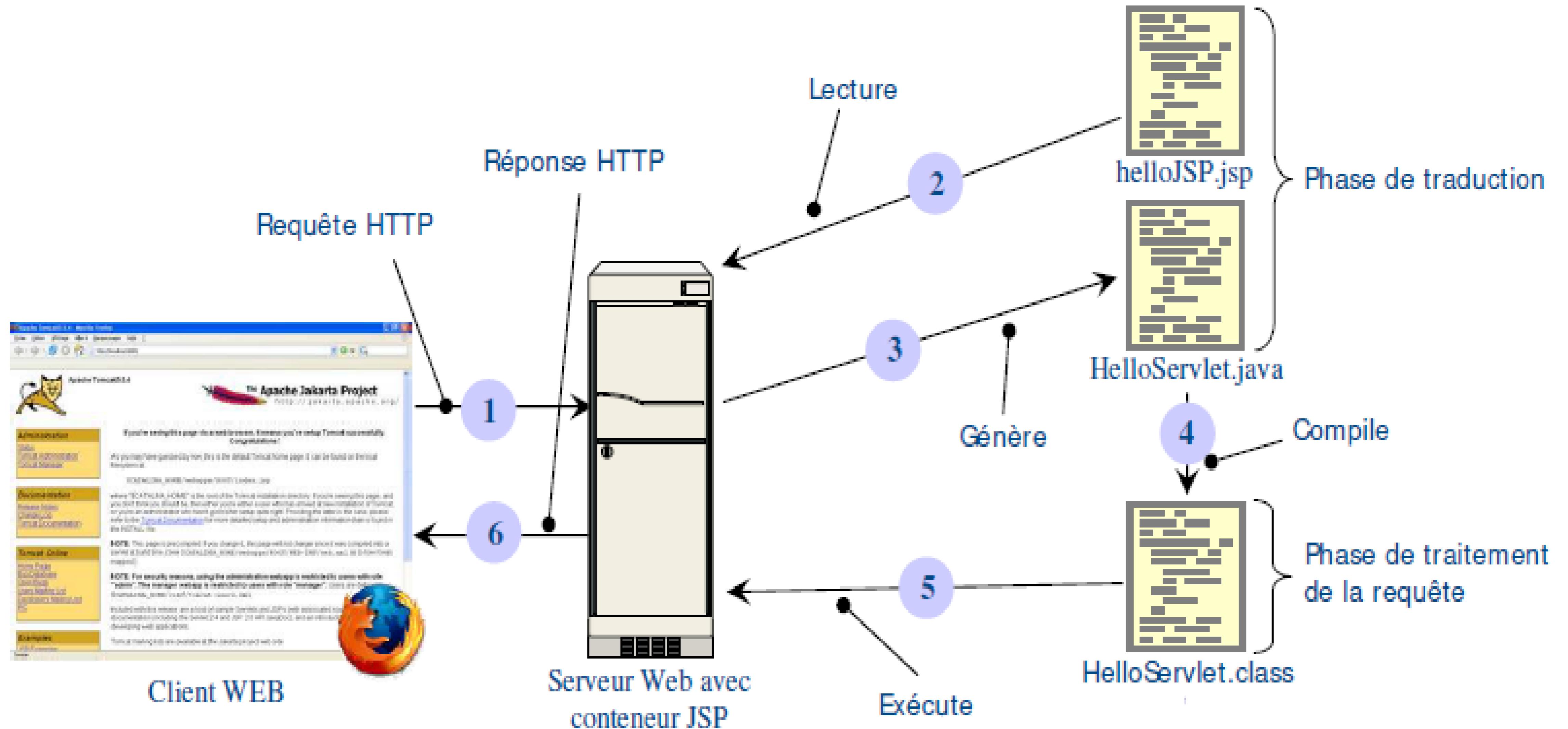


## View.jsp

```
<%  
String name=request.getParameter("name");  
%>  
<!DOCTYPE html>  
<html>  
<head>  
    <title>Hello JSP</title>  
</head>  
<body>  
    <h3>Hello <%=name%></h3>  
</body>  
</html>
```



# Traduction de la JSP en servlet

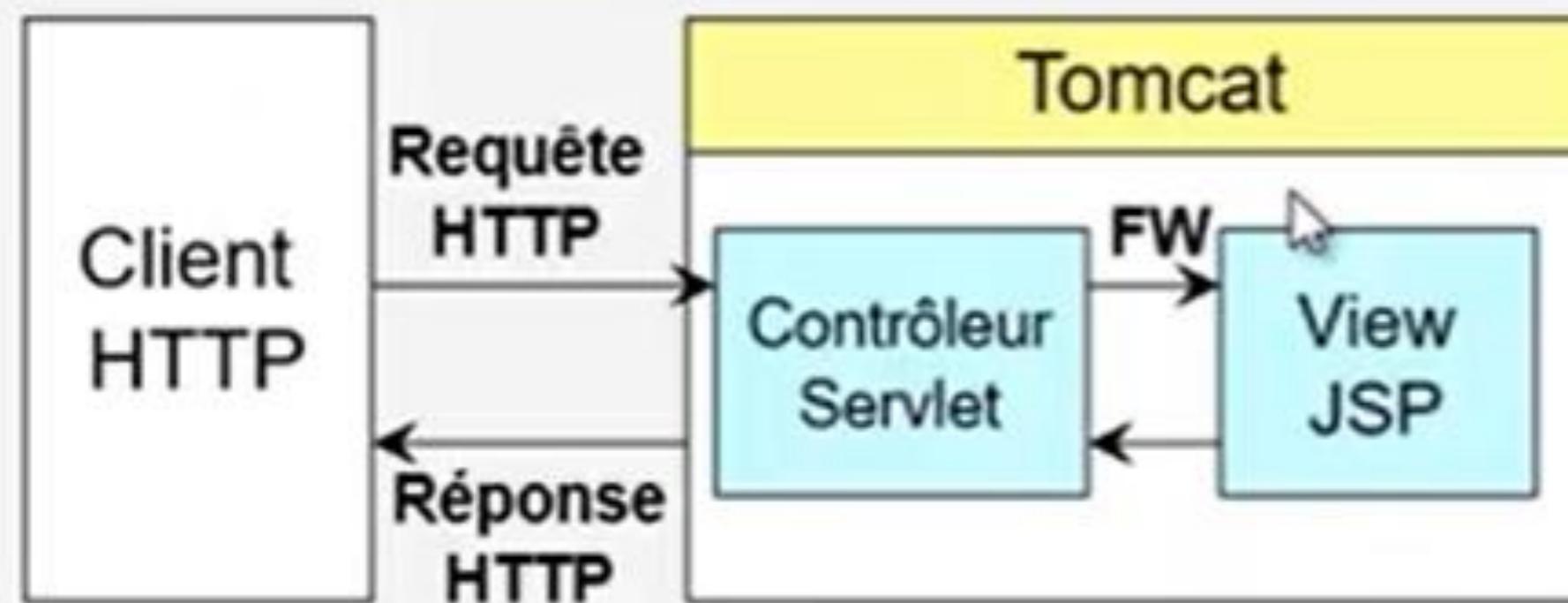


## Communication entre Servlet et JSP : Forward



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

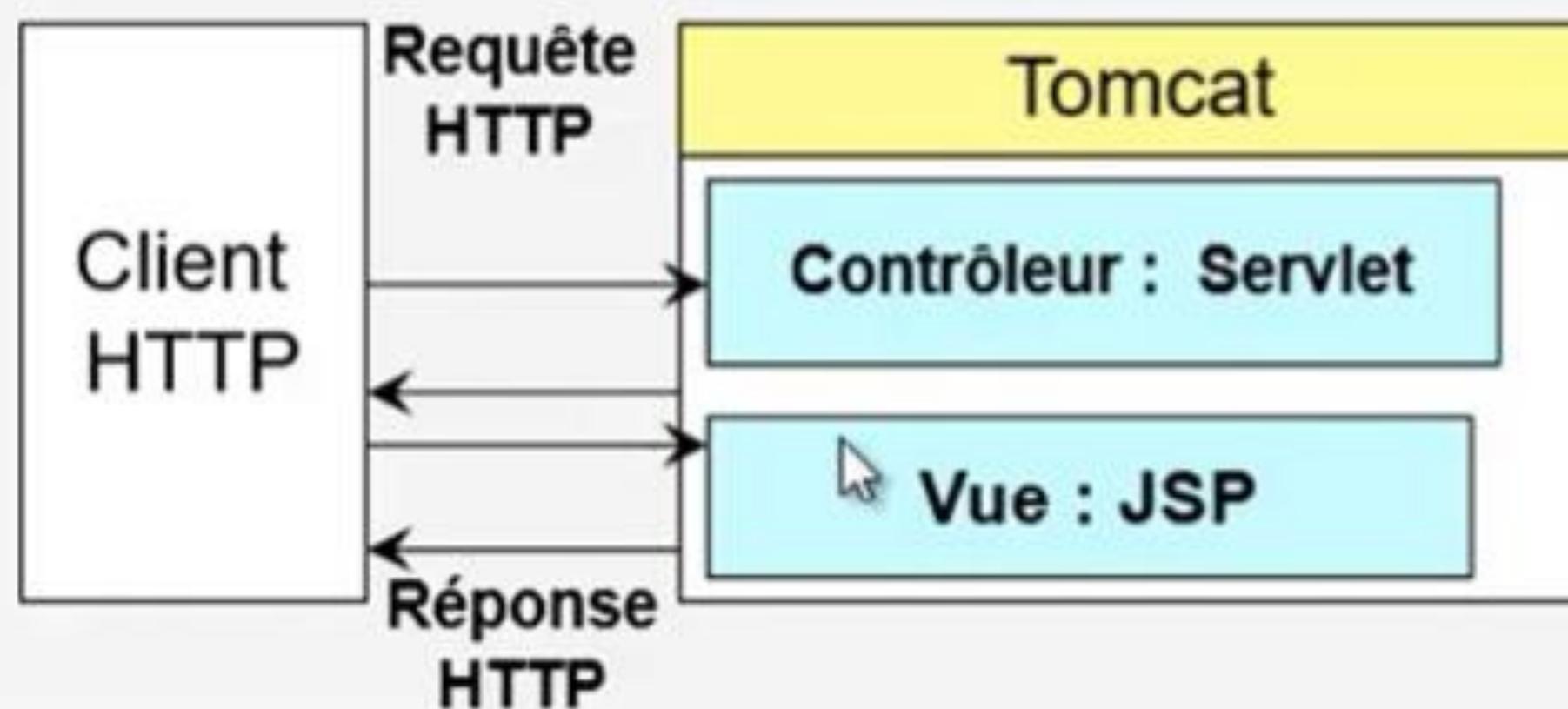
```
public class ControleurServlet extends HttpServlet {  
    @Override  
    protected void doGet(HttpServletRequest request, HttpServletResponse  
    response) throws ServletException, IOException {  
  
        String name=request.getParameter("name");  
        request.setAttribute("inputData", name);  
        request.getRequestDispatcher("View.jsp").forward(request, response);  
    } }
```



```
<%  
String name=(String)request.getAttribute("inputData");  
%>  
<!DOCTYPE html>  
<html>  
<head>  
<title>Hello JSP</title>  
</head>  
<body>  
<h3>Hello <%=name%> From JSP</h3>  
</body>  
</html>
```

# Communication entre Servlet et JSP : Redirection

```
@WebServlet(name="cs",urlPatterns = {"/hello","*.do"})
public class ControleurServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
        String name=request.getParameter("name");
        response.sendRedirect("View.jsp?name="+name);
    }
}
```



http://localhost:8080/TPS/View.jsp?name=Mohamed

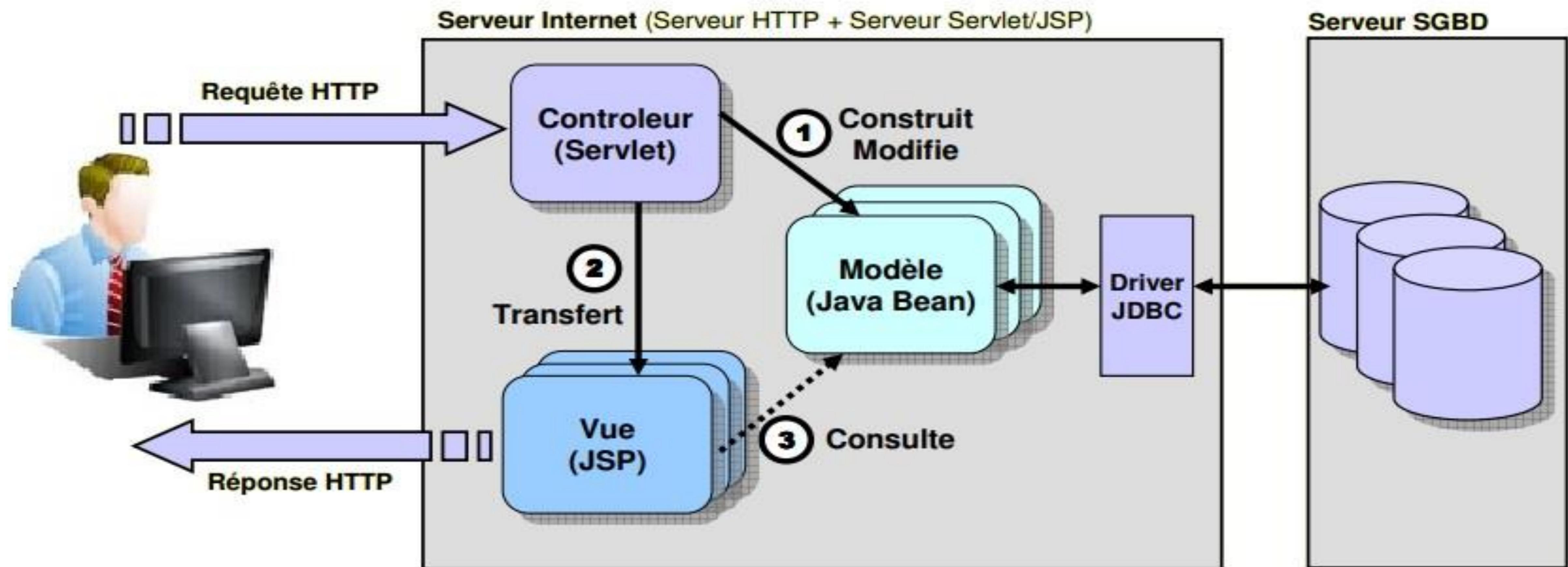
Hello Mohamed From JSP

```
<%
String name=request.getParameter("name");
%>
<!DOCTYPE html>
<html>
<head>
<title>Hello JSP</title>
</head>
<body>
<h3>Hello <%=name%> From JSP</h3>
</body>
</html>
```

# Exemple d'architecture MVC avec servlets, JSP et MySQL

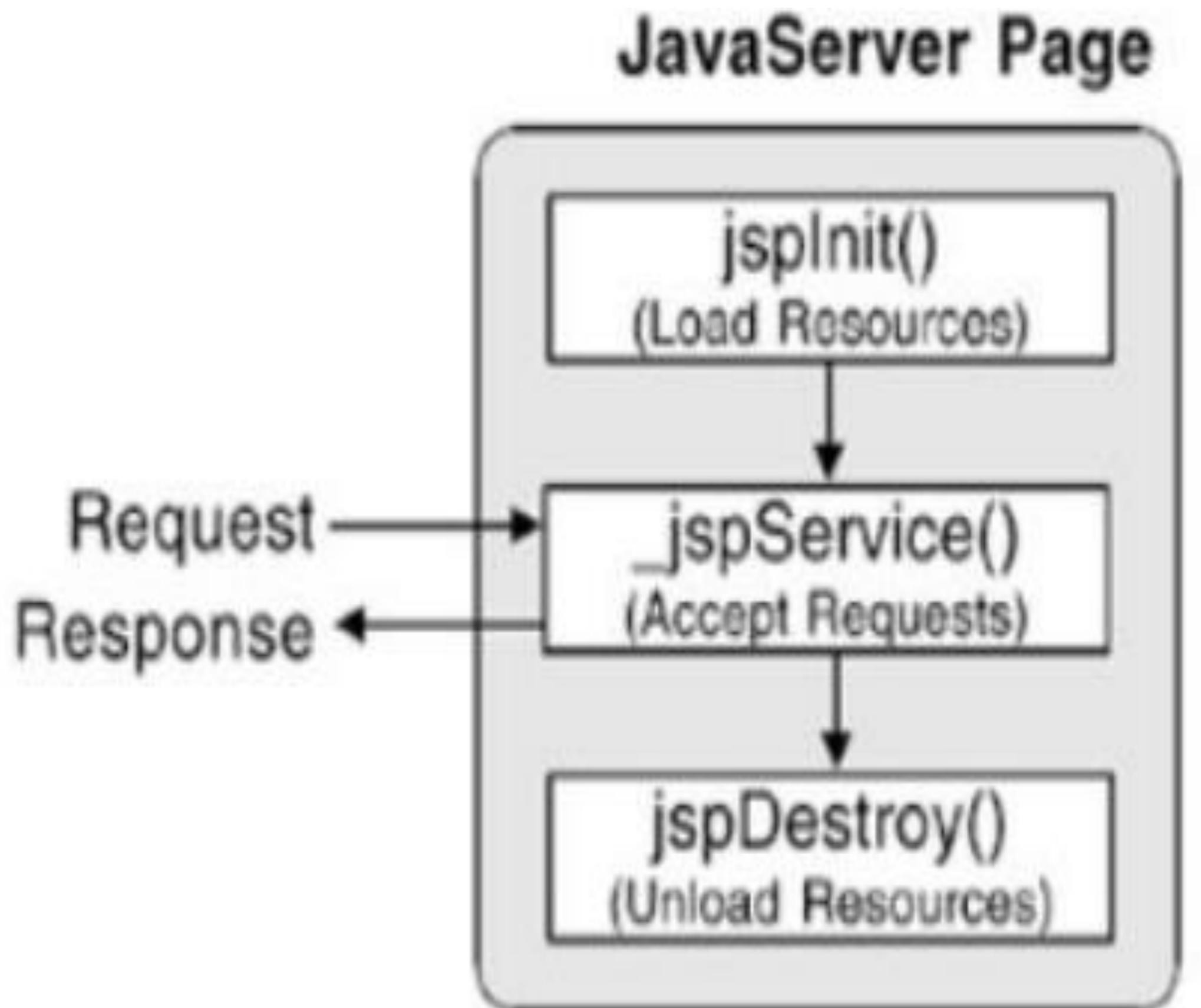
Dans une application Java EE:

- la couche **Modèle** est constituée d'objets Java ;
- la couche **Vue** est constituée de pages JSP ;
- la couche **Contrôle** est constituée de servlets.



# Cycle de vie d'une JSP

- 1) La méthode **jsplInit()** est appelée après le chargement de la page
- 2) La méthode **\_jspService()** est appelée à chaque requête
- 3) La méthode **jspDestroy()** est appelé lors du déchargement,



# Eléments syntaxiques d'une JSP

Une page JSP peut être formée par les éléments suivants :

- Les expressions
- Les déclarations
- Les directives
- Les scriptlets

**Les principaux tag JSP  
Commencent par <%  
et se terminent par %>**

## Une page JSP très simple

```
<html>
<body>
<%@ page language = "java" %>
<%! int counter = 1; %>
<%
    counter=counter+1;
%>
</body>
</html>
```

# Les commentaires

Commentaires HTML :

```
<!-- commentaire HTML --> envoyés au client
```

Commentaires JSP :

```
<% //commentaire java %> Ne sont pas envoyés au client
```

## Balise d'expression: <%= %>

```
<%= new java.util.Date() %>
```

Traduit en

```
out.print(new java.util.Date());
```

## Balise de déclarations : <%! code %>

```
<%! int count=10; %>
```

## Balise de Scriptlets : <% code %>

```
<%
for (i = 0; i < 10; i++) {
    out.println(i);}
%>
```

## Balise de directive: <%@= %>

include <%@include file="entete.jsp" %>

Import <%@ page import="java.util.List, java.util.Date" %>

Langage/Contenu <%@ page language="java" contentType="text/html

Error <%@ page iserrorpage=<< exemple.jsp >> %>

# Vue Générale

56

Déclaration

Scriptlet

```
<html>
<head>
<title>Bonjour tout </title>
</head>
<body>
<%! String contenu[] = {"raoul","john","nicolas"};%>
<%
for (int i = 0; i <contenu.length; i++) {
%>
    Le <%= i+1 %> ème nom est <%= contenu[i] %> <p>
<% } %>
</body>
</html>
```

```
public final class example_jsp extends HttpJspBase {
    String contenu[] = {"raoul","john","nicolas"};
    public void _jspService(HttpServletRequest req,
        HttpServletResponse res) throws IOException, ... {

        out.write("\t\t<title>Bonjour tout le monde</title>\r\n");
        out.write("\t</head>\r\n"); out.write("\t<body>\r\n");

        for (int i = 0; i <contenu.length; i++) {
            out.write("\t\t\tLe ");
            out.print( i+1 );
            out.write(" ème nom est ");
            out.print( contenu[i] );
            out.write(" <p>\r\n");
            out.write("\t\t");
        }
    }
}
```

Expression