

# La technologie des Servlets

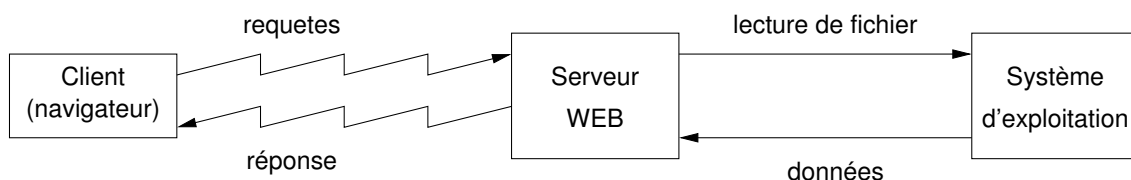
## Table des matières

<b>1</b>	<b>Présentation du protocole HTTP</b>	<b>1</b>
1.1	Les requêtes et réponses HTTP	1
<b>2</b>	<b>Applications WEB</b>	<b>2</b>
<b>3</b>	<b>La technologie des Servlets</b>	<b>2</b>
3.1	Application WEB Java	3
3.2	Conteneur WEB	3
3.3	Ma première servlet	4
3.4	Configuration (web.xml)	5
3.5	Le cycle de vie d'une servlet	6
3.6	Les interfaces de requête et de réponse	6
3.7	Servlet et formulaires HTML	7
<b>4</b>	<b>La gestion des sessions</b>	<b>7</b>
4.1	Codage des sessions	8
4.2	La durée de vie des sessions	8
4.3	Suivre les modifications de session	8
4.4	Durée de vie des objets	9

## 1 Présentation du protocole HTTP

Le protocole HTTP (**Hyper Text Transmission Protocol**<sup>1</sup>) :

- Basé sur TCP/IP (port 80).
- Une structure client/serveur (voir schéma).
- Protocole sans état : pas de notion de session (les requêtes sont indépendantes).



### 1.1 Les requêtes et réponses HTTP

#### Forme d'une requête

```
<méthode> <URI> <protocole>  
<attribut1>: <valeur1>  
<attribut2>: <valeur2>
```

1. <http://www.w3.org/Protocols/rfc2616/rfc2616.html>

### Exemple de requête

```
GET /index.html HTTP/1.0
accept: /*/*
connection: keep-alive
<--- ligne vide
```

### Forme d'une réponse

```
<protocole> <code> <description>
<attribut1>: <valeur1>
<attribut2>: <valeur2>
<--- ligne vide
<les données>
```

### Exemple de réponse

```
HTTP/1.0 200 OK
content-Type: text/html
encoding: UTF-8
<--- ligne vide
<html> ... </html>
```

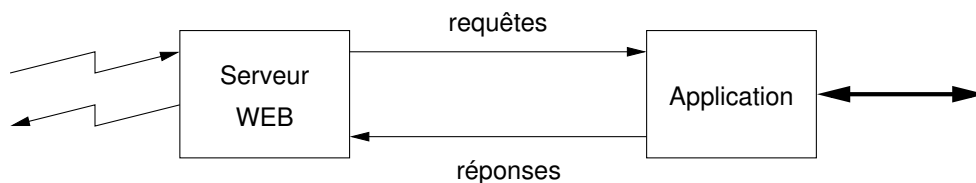
- Les méthodes :
- GET : récupération de données
  - POST : GET avec des données nom=DUPOND , prenom=Léo , prenom=paul
  - PUT : dépose d'un fichier.
  - ...

- Les codes des réponses :
- 200 : succès
  - 301 / 302 : ressource déplacée définitivement/temporairement
  - 403 : requête non autorisée
  - 404 : ressource non disponible

## 2 Applications WEB

Principes des applications WEB :

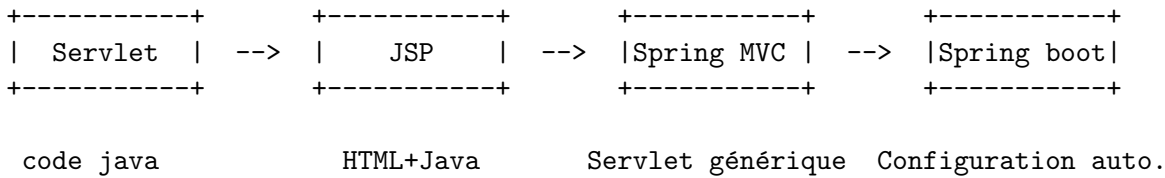
- les requêtes sont interprétées par des applications,
- les réponses sont calculées en fonction du traitement des requêtes et d'un contexte courant maintenu par l'application.



- Contexte :
  - ▷ la requête
  - ▷ la session courante
  - ▷ l'état de l'application

## 3 La technologie des Servlets

- Version 6.0 (JEE 10)
- C'est une spécification
- Les produits qui implantent cette norme :
  - ▷ Tomcat d'Apache,
  - ▷ Glassfish (implantation de référence),
  - ▷ Jetty,
  - ...
- Historique :
  - ▷ 5.0 (JEE 9) 2019,
  - ▷ 4.0 (JEE 8) 2017,
  - ▷ 3.1 (JEE 7) 2013,
  - ▷ 3.0 (JEE 6) fin 2009,
  - ▷ 2.5 (JEE 5) en 2005,
  - ...
  - ▷ 1.0 en 1997.

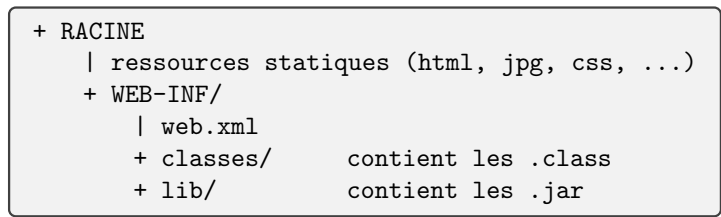


### 3.1 Application WEB Java

Une **application WEB Java** est constituée

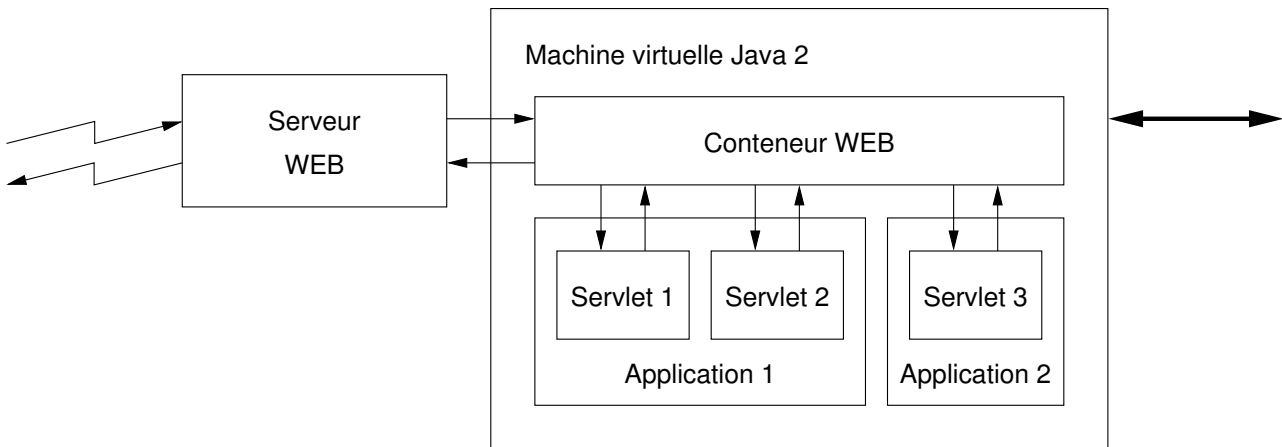
- de classes qui traitent les requêtes (les servlets),
- de ressources statiques (JPG, CSS, (X)HTML, XML, XSL, etc.),
- de librairies Java (fichiers `.jar`),
- d'un fichier `web.xml` de configuration.

Une application WEB a la structure suivante :



Ces fichiers peuvent être rangés dans une WAR (**Web Application aRchive**) en fait une archive jar (qui est un ZIP).

### 3.2 Conteneur WEB



Les applications sont déployées dans un **conteneur WEB** qui assure

- la connexion avec le serveur WEB,
- le décodage des requêtes et le codage des réponses,
- l'aiguillage sur la bonne servlet (et la bonne application),
- la gestion des sessions,
- le cycle de vie des servlets,
- la gestion est l'allocation des **threads**.

### 3.3 Ma première servlet

```
package myapp.web;

import java.io.*;
import jakarta.servlet.*;
import jakarta.servlet.annotation.*;
import jakarta.servlet.http.*;

@WebServlet( name = "UneServletSimple", description = "Une_servlet_simple",
            urlPatterns = { "/simple/*", "*.do" },
            loadOnStartup = 5 )
public final class SimpleServlet extends HttpServlet { // Ancienne méthode

    // initialisation et terminaison de la servlet
    public void init( ... ) throws ServletException { ... }
    public void destroy() { ... }

    // traitement des requêtes GET et POST
    public void doGet( ... ) { ... }
    public void doPost( ... ) { ... }
}
```

#### URL traitées par la servlet

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    PrintWriter writer = response.getWriter();
    writer.printf("ServerName:␣%s\n", request.getServerName());
    writer.printf("method:␣%s\n", request.getMethod());
    writer.printf("contextPath:␣%s\n", request.getContextPath());
    writer.printf("ServletPath:␣%s\n", request.getServletPath());
    writer.printf("PathInfo:␣%s\n", request.getPathInfo());
}
```

GET http://myserver/myapp/simple/docs/hello.html

```
ServerName: myserver
Method: GET
ContextPath: /myapp
ServletPath: /simple/
PathInfo: docs/hello.html
```

GET http://myserver/myapp/product/265/edit.do

```
ServerName: myserver
Method: GET
ContextPath: /myapp
ServletPath: /product/265/edit.do
PathInfo: null
```

Détail de la méthode `doPost` :

```

public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws IOException, ServletException {

    // récupération d'un paramètre de la requête
    String data = request.getParameter("data");

    // traitement métier
    String result = data.toUpperCase();

    // construire du résultat
    response.setContentType("text/html");
    PrintWriter writer = response.getWriter();

    writer.println("<html><body>");
    writer.println("<h1>Hello</h1>");
    writer.printf("<p>_%s_</p>", result);
    writer.println("</body></html>");
}

```

Il existe autant de méthodes à surcharger dans la classe `HttpServlet` que de méthodes HTTP.

Détails des méthodes d'initialisation / terminaison :

```

// initialisation de la servlet
public void init(ServletConfig c) throws ServletException {
    String value1 = c.getInitParameter("param1");
    ...
}

// terminaison de la servlet
public void destroy() {
    ...
}

```

### 3.4 Configuration (web.xml)

Le fichier `web.xml` (qui est **optionnel**) :

```

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
        http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
    id="WebApp_ID" version="3.0">

    <display-name>Application de test</display-name>
    <description>Ma première application</description>

    <!-- déclarations des servlets -->
    <servlet> ... </servlet>

    <!-- correspondance servlets / URL -->
    <servlet-mapping> ... </servlet-mapping>

</web-app>

```

Déclaration des servlets :

```

<servlet>
  <servlet-name>UneServletSimple</servlet-name>
  <servlet-class>myapp.web.SimpleServlet</servlet-class>
  <init-param>
    <param-name>param1</param-name>
    <param-value>value1</param-value>
  </init-param>
  ...
  <load-on-startup>2</load-on-startup>
</servlet>

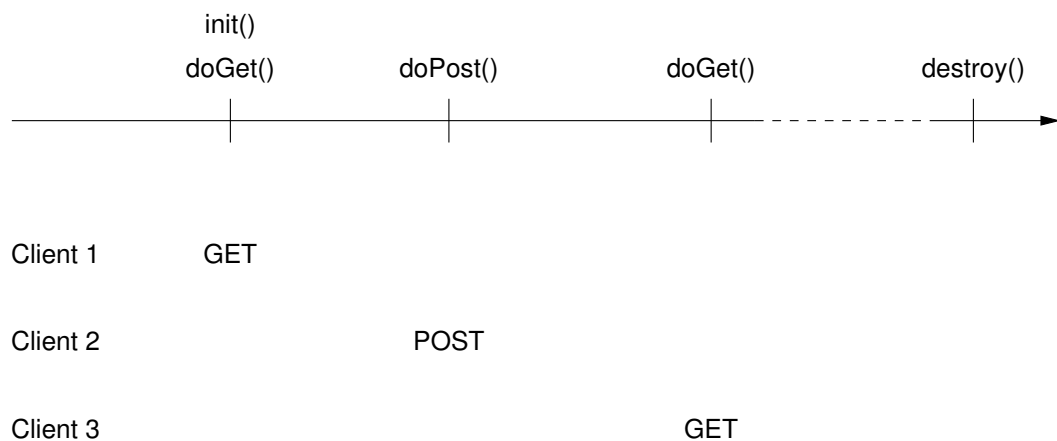
<servlet-mapping>
  <servlet-name>UneServletSimple</servlet-name>
  <url-pattern>/simple/*</url-pattern>
</servlet-mapping>

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

```

### 3.5 Le cycle de vie d'une servlet

- Un exemple :



- C'est la **même instance** (éventuellement exécutée en parallèle dans plusieurs **threads**) qui traite les requêtes de tous les clients.
- Les Servlets peuvent être préchargées au lancement du serveur ou lancées à la demande.

### 3.6 Les interfaces de requête et de réponse

- `jakarta.servlet.http.HttpServletRequest`

```

public String getParameter(String name)
public String[] getParameterValues(String name)
public HttpSession getSession()
...

```

- `jakarta.servlet.http.HttpServletResponse`

```

public void setContentType(String type)
public java.io.PrintWriter getWriter() throws ...
public ServletOutputStream getOutputStream() throws ...
public void addHeader(String name, String value)
public void addCookie(Cookie cookie)
...

```

### 3.7 Servlet et formulaires HTML

Un formulaire HTML :

```

<html><body>
  <form action="processForm" method="POST">

    <label>Nom : </label>
    <input type="text" name="nom" size="15"/><br/>

    <label>Prénom : </label>
    <input type="text" name="prenom" size="15"/><br/>

    <label>Statut : </label>
    <select name="statut" size="1">
      <option value="Etudiant">Etudiant</option>
      <option value="Prof">Enseignant</option>
    </select><br/>

    <input type="submit" name="boutonOK" value="Valider"/>

  </form>
</body></html>

```

La servlet `processForm` :

```

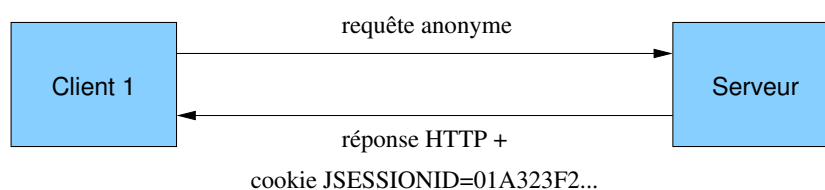
public void doPost(HttpServletRequest request, HttpServletResponse response)
  throws IOException, ServletException
{
  String nom = request.getParameter("nom");
  String prenom = request.getParameter("prenom");

  response.setContentType("text/html");
  response.getWriter().printf("<p>Bonjour %s %s</p>", prenom, nom);
}

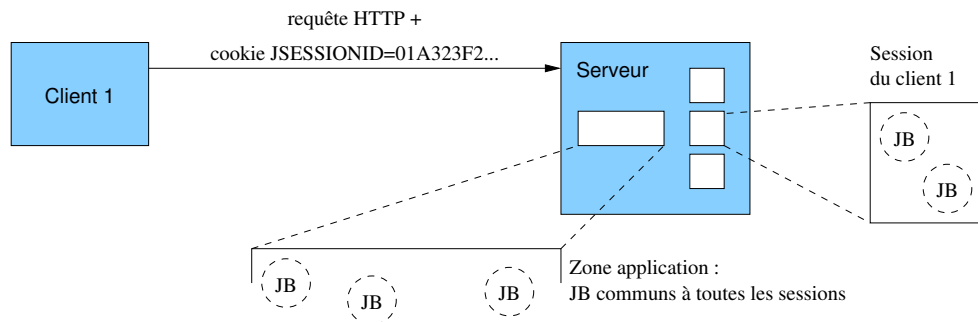
```

## 4 La gestion des sessions

- **Principe** : pour identifier le client, le serveur renvoi, dans la réponse à la première requête, un **cookie** ( `JSESSIONID` ) :



- Les cookies sont tirés au hasard.
- Lors des requêtes suivantes, le client est repéré et le serveur peut lui associer une session :



## 4.1 Codage des sessions

- Rappel : dans l'interface `HttpServletRequest` nous trouvons la méthode

```
public HttpSession getSession()
```

- L'interface `jakarta.servlet.http.HttpSession` :

```
public Object getAttribute(String name)
public void setAttribute(String name, Object value)
public void invalidate()
...
```

Ces méthodes permettent de **recupérer un objet** depuis une session, de **placer un objet** dans une session et finalement, de **vider** une session.

## 4.2 La durée de vie des sessions

Réglage de la durée de vie des sessions :

```
<web-app ... >
    ... premières déclarations ...
    ... déclaration des servlets ...
    <!-- durée de vie des sessions en minutes -->
    <session-config>
        <session-timeout>30</session-timeout>
    </session-config>
    ...
</web-app>
```

## 4.3 Suivre les modifications de session

- Si un objet en session implémente l'interface `HttpSessionBindingListener` (du package `jakarta.servlet.http`), alors les évènements

```
void valueBound(HttpSessionBindingEvent event) ;
void valueUnbound(HttpSessionBindingEvent event) ;
```

lui indiquent son attachement ou son détachement d'une session.

- On peut également écouter les évènements :



- ▷ création, destruction, modification d'une session,
- ▷ changement dans le contexte d'une servlet,

## 4.4 Durée de vie des objets

Il existe plusieurs visibilité et durée de vie pour les objets Java :

**Instances de porté requête :**

```

Requête n° 1 / Servlet 1
// ranger un objet dans une requête
request.setAttribute("myObject", myObject);
...

Requête n° 1 / Servlet 2
...
// le récupérer
myObject = (MyObject) request.getAttribute("myObject");

```

**Utilité :** faire passer des données d'une servlet à une autre servlet (chaînage) ou d'une servlet à une page JSP.

**fin de vie :** fin du traitement de la requête.

**Instances de porté session :**

```

Requête n° 1
// ranger un objet dans une session
HttpSession session = request.getSession();
session.setAttribute("myObject", myObject);
...

Requête n° 2
...
// le récupérer
myObject = (MyObject) session.getAttribute("myObject");

```

**Utilité :** faire passer des données d'une requête à une autre requête émise par le même client. A titre d'exemples :

- panier d'une application de commerce électronique,
- utilisateur authentifié d'une application sécurisée

**fin de vie :** fin de la session (*timeout* ou invalidation).

**Instances de porté application :**

```

Requête n° 100 du client 1
// ranger un objet dans la zone application
HttpSession session = request.getSession();
ServletContext context = session.getServletContext();
context.setAttribute("myObject", myObject);
...

Requête n° 200 du client 2
...
// le récupérer
myObject = (MyObject) context.getAttribute("myObject");

```

**Utilité :** rendre des données ou des services accessibles à tous les clients.

- données métiers globales (liste des paniers),

- services singletons,
- paramètres de l'application,

**fin de vie** : fin de l'application (durée de vie très longue).

La portée des instances Java :

