

La technologie des Servlets

Table des matières

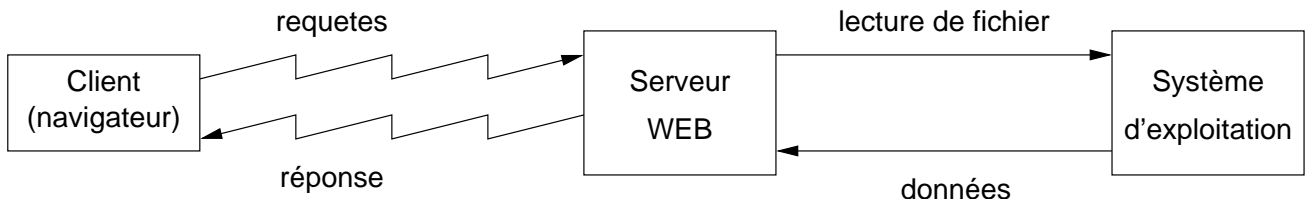
1	Présentation du protocole HTTP	1
1.1	Les requêtes HTTP	2
1.1.1	Les requêtes de lecture	2
1.1.2	Les requêtes de modification	2
1.2	Les réponses HTTP	2
1.3	Les codes de réponse HTTP	3
2	Applications WEB	3
2.1	Solutions propriétaires et CGI	3
2.2	ASP, PHP et JSP	4
3	La technologie des Servlets	4
3.1	Application WEB Java	4
3.2	Conteneur WEB	5
3.3	Configuration (<code>web.xml</code>)	5
3.4	Ma première servlet	7
3.5	Le cycle de vie d'une servlet	7
3.6	Les interfaces de requête et de réponse	8
3.7	Servlet et formulaires HTML	8
3.8	La gestion des sessions	9
3.8.1	Codage des sessions	9
3.8.2	La durée de vie des sessions	10
3.8.3	Suivre les modifications de session	10
3.8.4	Durée de vie des objets	10

1 Présentation du protocole HTTP

Caractéristiques du protocole HTTP (*Hyper Text Transmission Protocol*¹) :

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



1.1 Les requêtes HTTP

Forme générale d'une requête :

```

methode URI protocole
attribut1 : valeur1
attribut2 : valeur2
....
<ligne vide>
  
```

Un exemple (notez la ligne vide à la fin) :

```

GET /index.html HTTP/1.0
accept: */*
connection: keep-alive
cache-control: no-cache
  
```

1.1.1 Les requêtes de lecture

- Méthode GET : récupération de données identifiées par l'URI.
- Méthode HEAD : demande d'informations sur les données identifiées par l'URI (pas de transmission de données).
- Méthode POST : identique à GET, mais le client ajoute à la requête un ensemble de paires :

```

nom1=valeur1
nom1=valeur2
nom2=valeur3
...
  
```

Vous pouvez noter qu'il est possible d'associer plusieurs valeurs au même nom.

1.1.2 Les requêtes de modification

- Méthode PUT : dépose d'un fichier.
- Méthode OPTIONS : interroge le serveur sur les méthodes disponibles sur une URI donnée.
- Méthode DELETE : ...
- Méthode TRACE : ...

1.2 Les réponses HTTP

Forme générale d'une réponse :

```
protocole CODE description
attribut1 : valeur1
attribut2 : valeur2
....
<ligne vide>
<les données>
```

Un exemple (notez la ligne vide) :

```
HTTP/1.0 200 OK
server: Apache...
date: ...
content-Type: text/html

<html>
    ....
</html>
```

1.3 Les codes de réponse HTTP

Les principaux codes de réponse :

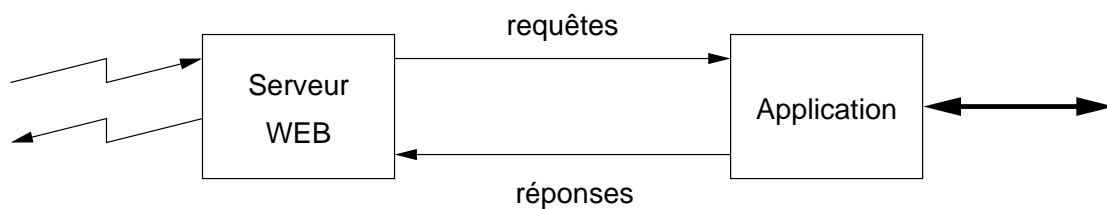
200	requête exécutée avec succès
301	ressource déplacée définitivement
302	ressource déplacée temporairement
403	requête non autorisée
404	ressource non disponible
500	erreur interne du serveur
...	...

Retrouvez les principaux codes sur <http://www.codeshttp.com/> ou directement dans la RFC2616².

2 Applications WEB

Principes des applications WEB :

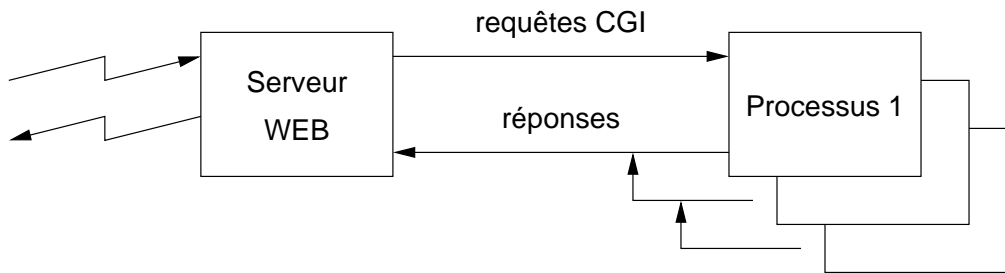
- les requêtes sont interprétées par des applications,
- les réponses sont calculées en fonction du traitement des **requêtes** et d'un **contexte courant** maintenu par l'application.



2.1 Solutions propriétaires et CGI

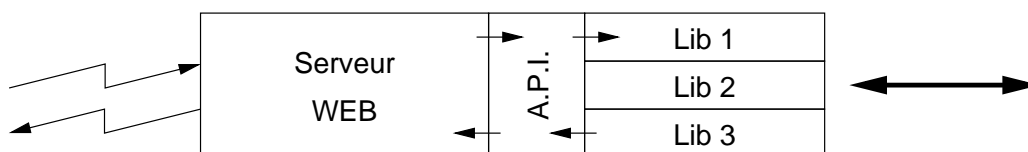
- CGI (*Common Gateway Interface*) une solution simple mais coûteuse :

²<http://www.w3.org/Protocols/rfc2616/rfc2616-sec6.html#sec6.1.1>



Chaque requête donne lieu à la création d'un processus qui traite la requête et produit la réponse. La charge du système est donc très importante. Des langages de scripts (bash, perl, python, etc.) sont souvent choisis pour leur facilité de mise en oeuvre.

- API propriétaires non portables (ISAPI de Microsoft ou NSAPI de NetApp) :



Dans cette approche, chaque application WEB est une librairie ajoutée au serveur WEB. Nous suivrons la même idée dans le monde Java.

2.2 ASP, PHP et JSP

- ASP (*Active Server Page*) de Microsoft. Les pages ASP sont un mélange de VBscript et de HTML.
- PHP (*PHP : Hypertext Preprocessor*) même solution avec un langage de script conçu à cet effet.
- Servlet, JSP (*Java Server Page*) même solution avec un mélange de Java et HTML.

3 La technologie des Servlets

- Version 3.0 (JEE 6)
- C'est une spécification
- Produits qui implantent cette norme :
 - Tomcat d'Apache,
 - Glassfish de Sun/Oracle (implantation de référence),
 - Jetty,
 - ...
- Historique :
 - 3.0 (JEE 6) fin 2009,
 - 2.5 (JEE 5) en 2005,
 - 2.4 (J2EE 1.4) en 2003,
 - ...
 - 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 bibliothèques Java (fichiers .jar),
- d'un fichier de configuration (web.xml).

Une application WEB a la structure suivante :

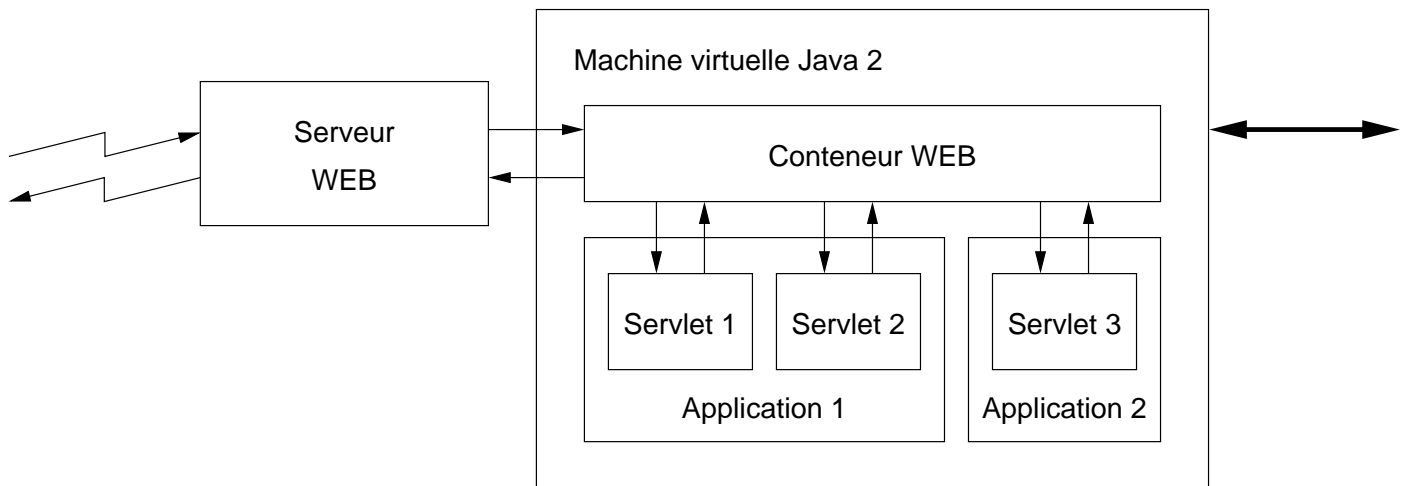
```

mon-application.war
| ressources statiques (html, jpg, css, ...)
+ WEB-INF/
  | web.xml
  + classes/      contient les .class
  + lib/          contient les .jar
  
```

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 :



- Le *conteneur WEB* 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 Configuration (web.xml)

- le fichier web.xml :

```

<web-app 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"
  version="2.4">

  <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>mypackage.SimpleServlet</servlet-class>
  <init-param>
    <param-name>jdbc.url</param-name>
    <param-value>jdbc:mysql://machine/nomdebase</param-value>
  </init-param>
  ...
  <load-on-startup>2</load-on-startup>
</servlet>

```

La clause `load-on-startup` permet d'indiquer un ordre dans le chargement et l'initialisation des servlets.

- Mise en correspondance servlet versus URL :

```

<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>

```

URL traitées par la servlet :

```

http://serveur-name/application-name/simple
http://serveur-name/application-name/simple/hello.html
http://serveur-name/application-name/simple/documents/7419.html

http://serveur-name/application-name/simple/hello.do
http://serveur-name/application-name/logout.do

```

Des méthodes s'appliquant sur la requête permettent de découper l'URL.

3.4 Ma première servlet

```
package mypackage;

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

import mybusiness.Business;

public final class SimpleServlet extends HttpServlet {
    Business bs = null;

    // initialisation de la servlet
    public void init(ServletConfig c) throws ServletException {
        String jdbc = c.getInitParameter("jdbc.url");
        bs = new Business(jdbc);
    }

    // destruction la servlet
    public void destroy() {
        bs.close();
    }

    public void doGet( ... )    { ... }
    public void doPost( ... )   { ... }
    public void service( ... )  { ... }
}
```

- Détail de la méthode doGet :

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

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

    // appel de la couche métier
    String result = bs.action(data);

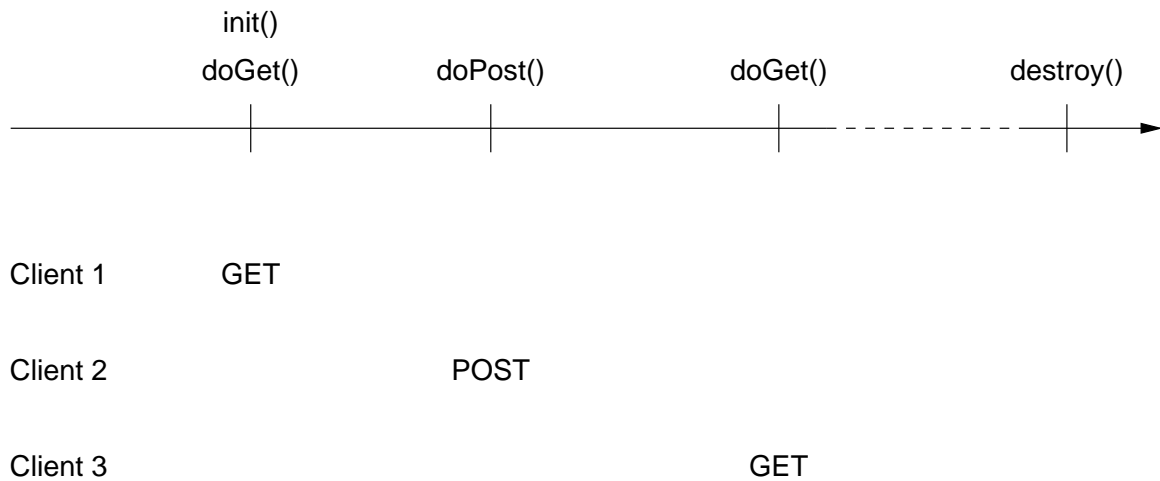
    // calcul du résultat
    response.setContentType("text/html");
    PrintWriter writer = response.getWriter();

    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.

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

- `javax.servlet.http.HttpServletRequest`

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

- `javax.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="test" 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 test :

```

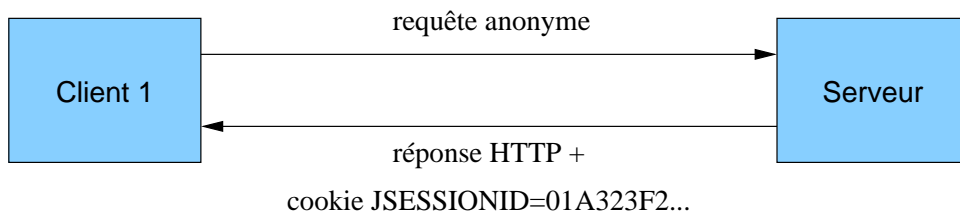
public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws IOException, ServletException
{
    String nom = request.getParameter("nom");
    String prenom = request.getParameter("prenom");

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

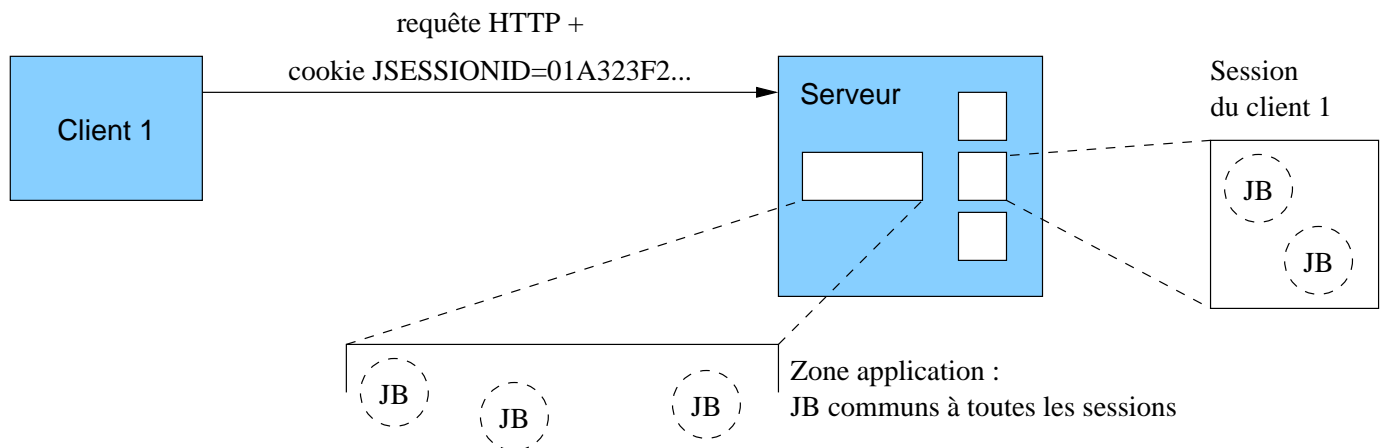
```

3.8 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 :



3.8.1 Codage des sessions

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

```
public HttpSession getSession()
```

- L'interface `javax.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.

3.8.2 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>

...
```

3.8.3 Suivre les modifications de session

- Si un objet référencé en session implante `HttpSessionBindingListener` (interface du package `javax.servlet.http`), alors les évènements

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

lui indiquent sont 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,

3.8.4 Durée de vie des objets

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

Instances de porté requête :

```
// ranger un objet dans une requête
request.setAttribute("myObject", myObject);
...
// 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 :

```
// ranger un objet dans une session
HttpSession session = request.getSession();
session.setAttribute("myObject", myObject);
...
// 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 portée application :

```

// ranger un objet dans la zone application
HttpSession session = request.getSession();
ServletContext context = session.getServletContext();
context.setAttribute("myObject", myObject);
...
// le récupérer
myObject = (MyObject) context.getAttribute("myObject");

```

- **Utilité** : rendre des données ou des services accessibles à tous les clients. A titre d'exemples :
 - 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 :

