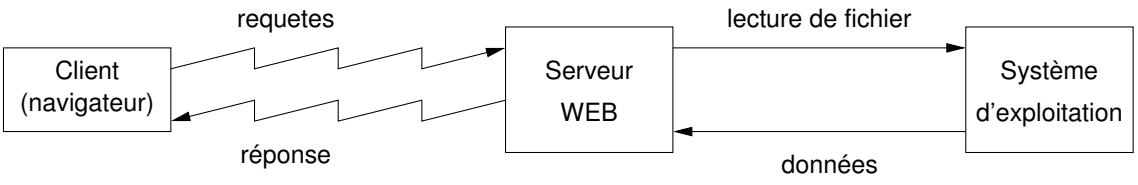
La technologie des Servlets

Table des matières

1 Présentation du protocole HTTP

Le 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 (la première ligne est obligatoire) :

```
<méthode> <URI> <protocole>
<attribut1>: <valeur1>
<attribut2>: <valeur2>
                                    <-- ligne vide
```

Un exemple (méthode `GET`, URI à `/index.html`, protocole à `HTTP/1.0` et trois lignes d'attribus). Notez la ligne vide à la fin :

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

1.2 Les requêtes de lecture

- Méthode `GET` : récupération de données identifiées par l'URI.
- Méthode `POST` : identique à `GET`, mais le client ajoute à la requête un ensemble de paires :

```
nom=DUPOND
prenom=pierre
prenom=paul
...
```

- Méthode `PUT` : dépose d'un fichier.
- Méthode `OPTIONS` : interroge le serveur sur les méthodes disponibles.
- Méthode `HEAD` : demande d'informations sur les données identifiées par l'URI (pas de transmission de données).
- Méthode `DELETE` : ...
- Méthode `TRACE` : ...

1.3 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: 10-12-2020
content-Type: text/html
encoding: UTF-8
                                        <-- ligne vide
<html>
   ....
</html>
```

1.4 Les codes de réponse HTTP

Les principaux codes de réponse :

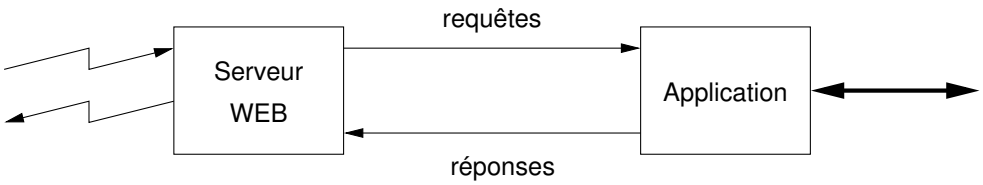
- `200` : requête exécutée avec succès
- `301` : ressource déplacée définitivement
- `302` : ressource déplacée temporairement
- `403` : requête non autorisée
- `404` : ressource non disponible
- `500` : erreur interne du serveur

Retrouvez les principaux codes sur `http://www.codeshttp.com/` ou directement dans la RFC2616 [2].

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 5.0 (JEE 9)
- C'est une spécification
- Les produits qui implantent cette norme :
  - ▷ Tomcat d'Apache,
  - ▷ Glassfish de Sun/Oracle (implantation de référence),

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

- ▷ Jetty,
  ...

- Historique :
  - ▷ 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

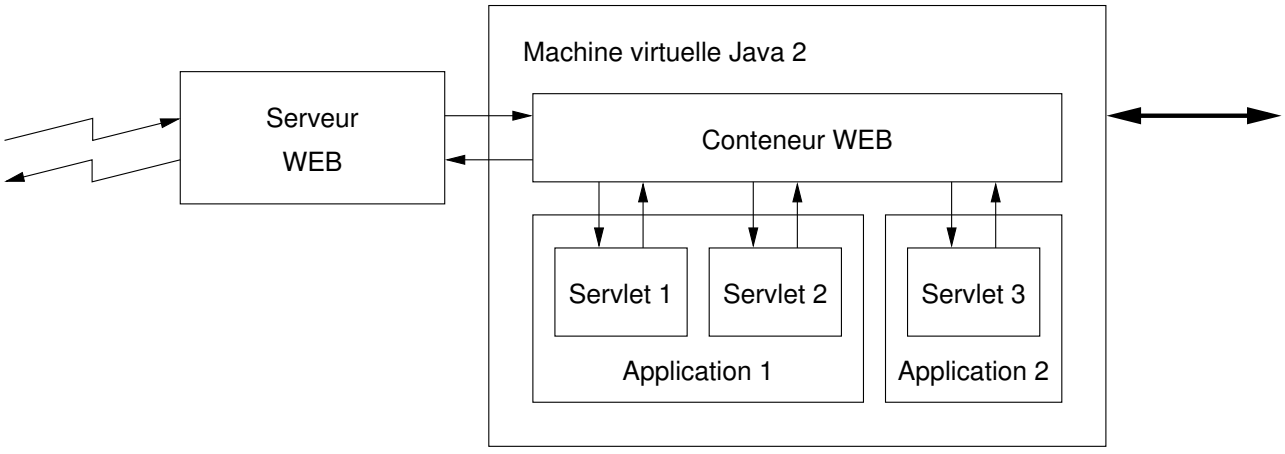
- de classes qui traitent les requêtes (les servlets),
- de ressources statiques (JPG, CSS, (X)HTML, XML, XSL, etc.),
- de librairies Java (fichiers `.jar`),
- d'un fichier `web.xml` de configuration.

Une application WEB a la structure suivante :

```
+ RACINE
    | 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** 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

```java
package myapp.web;

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

```java
protected void doGet(HttpServletRequest request, HttpServletResponse response)
       throws ServletException, IOException {
   response.getWriter()
           .append("ServerName: " + request.getServerName() + "\n")
           .append("contextPath: " + request.getContextPath() + "\n")
           .append("ServletPath: " + request.getServletPath() + "\n")
           .append("PathInfo: " + request.getPathInfo());
}
```

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

```
ServerName: myserver
contextPath: /myapp
ServletPath: /simple/
PathInfo: docs/hello.html
```

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

```
ServerName: myserver
contextPath: /myapp
ServletPath: /product/265/edit.do
PathInfo: null
```

Détail de la méthode `doPost` :

```java
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.toUpper();

    // 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 :

```java
// 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**) :

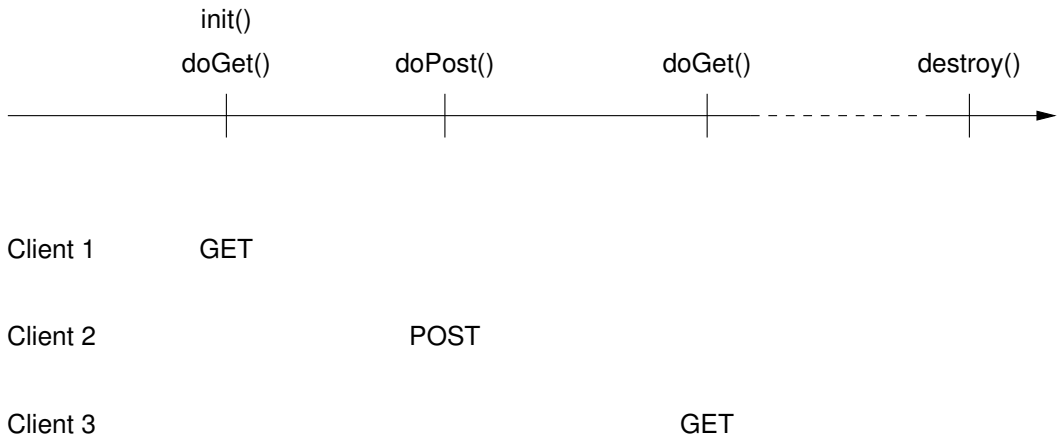
```xml
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
        http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
    id="WebApp_ID" version="3.0">

    <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

- `javax.servlet.http.HttpServletRequest`

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

- `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 :

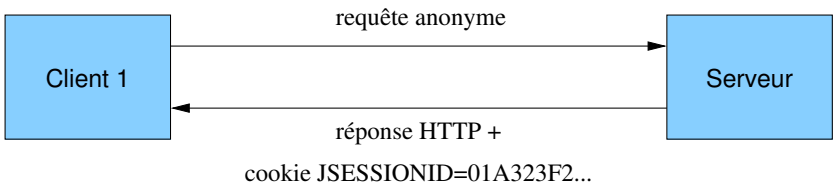
```
<html><body>
    <form action="processForm" method="POST">

      <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` :

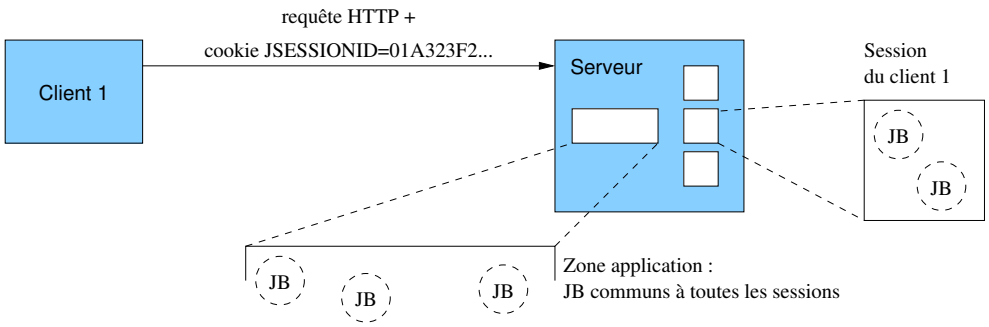
```
public void doPost(HttpServletRequest request, HttpServletResponse response)
   throws IOException, ServletException
{
   String nom = request.getParameter("nom");
   String prenom = request.getParameter("prenom");

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

4 La gestion des sessions

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



- 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 `javax.servlet.http.HttpSession` :

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

Ces méthodes permettent de **récupérer 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 `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,

4.4 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 porté application :

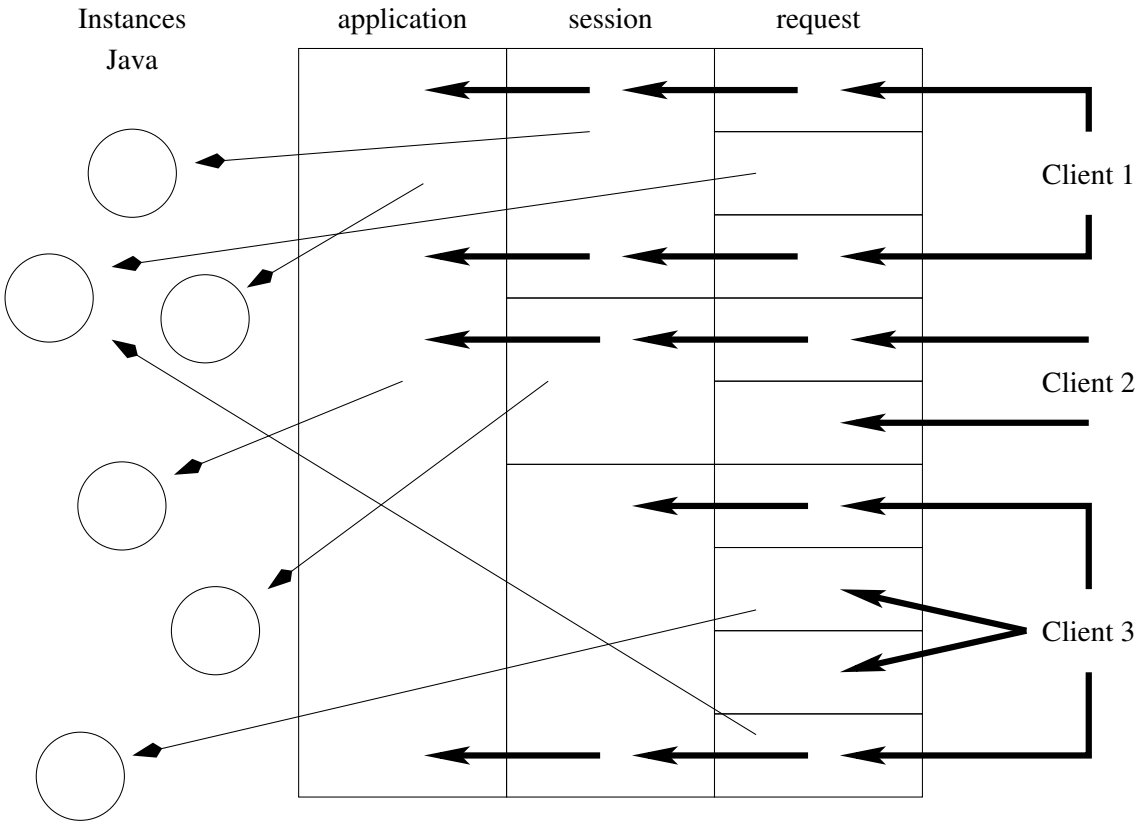
```
// ranger un objet dans la zone application
HttpSession session = request.getSession();
ServletContext context = session.getServletContext();
context.setAttribute("myObject", myObject);
...
// le récupérer
myObject = (MyObject) context.getAttribute("myObject");
```

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

- données métiers globales (liste des paniers),
- services singletons,
- paramêtres de l'application,

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

La porté des instances Java :


Instances
Java
application
session
request
Client 1
Client 2
Client 3