Java Enterprise Edition - Architectures et données

Gaël Guibon

E-mails: prenom.nom@lis-lab.fr

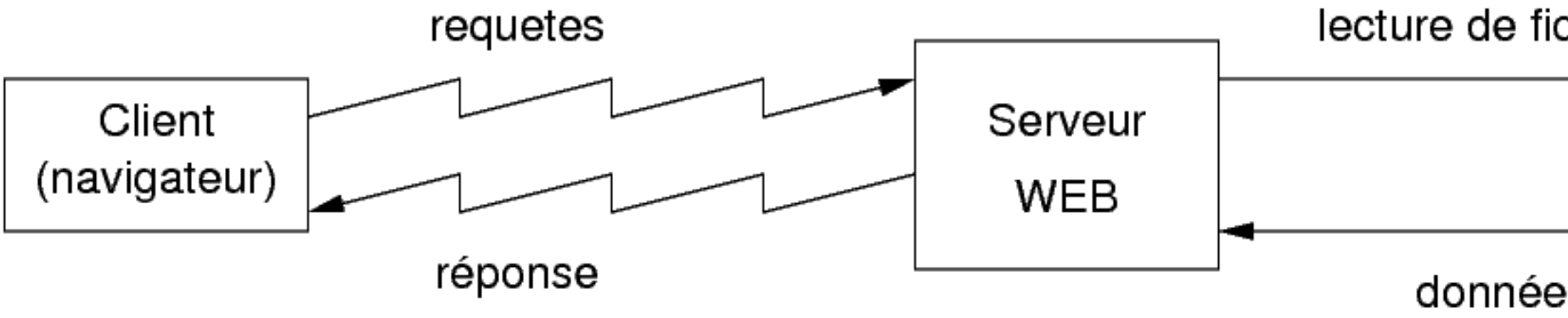
Aix*Marseille université

1. **Les requêtes HTTP**

2. La vue - Les JSP

Les requêtes HTTP

Les requêtes HTTP



Le protocole HTTP

HTTP = Hyper Text Transmission Protocol

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

Les requêtes HTTP

Forme générale

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

Exemple

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

Ligne vide volontaire à la fin.

Les modes 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 :

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

Possible d'associer plusieurs valeurs au même nom

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** : ...

Les réponses HTTP

Forme générale d'une réponse

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

Exemple

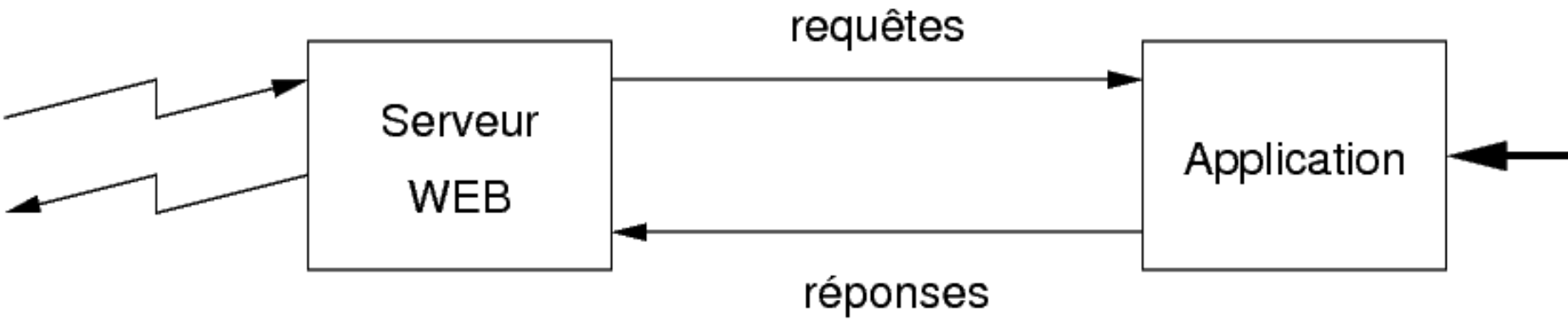
```
HTTP/1.0 200 OK
server: Apache...
date: 10-12-2014
content-Type: text/html
encoding: UTF-8

<html>
   ....
```

Les codes de réponses HTTP

- 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
- ... : ...

Voir : `http://www.codeshttp.com`

Applications WEB

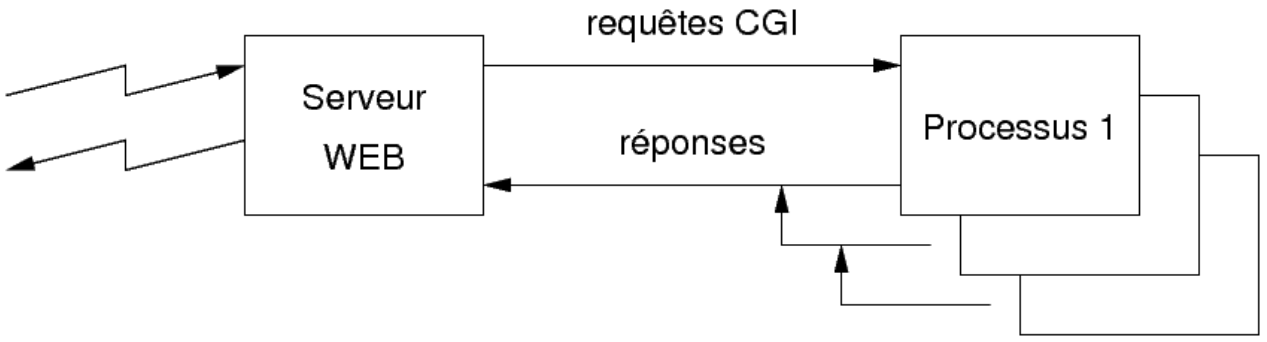


Contexte

- requête
- session
- état de l'application

CGI

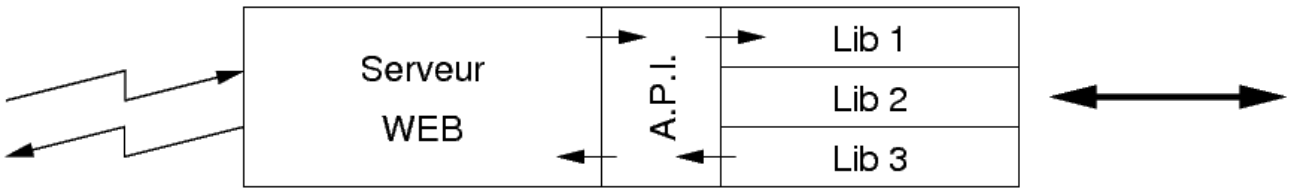
CGI (Common Gateway Interface)



Couteux !

- 1 requête = 1 processus
- Souvent par scripts bash perl python, etc

API propriétaires



- une app devient une librairie ajoutée au serveur WEB
- Java suit le même principe

ASP, PHP et JSP

Comparatif

- **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.

La technologie des Servlets

- C'est une spécification
- Produits qui implantent cette norme :
  - Tomcat d'Apache,
  - Glassfish de Sun/Oracle (implantation de référence),
  - Jetty,
  - ...
- Historique :
  - 3.1 (JEE 7) 2013,
  - 3.0 (JEE 6) fin 2009,
  - 2.5 (JEE 5) en 2005,
  - 2.4 (J2EE 1.4) en 2003,
  - ...
  - 1.0 en 1997.

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.

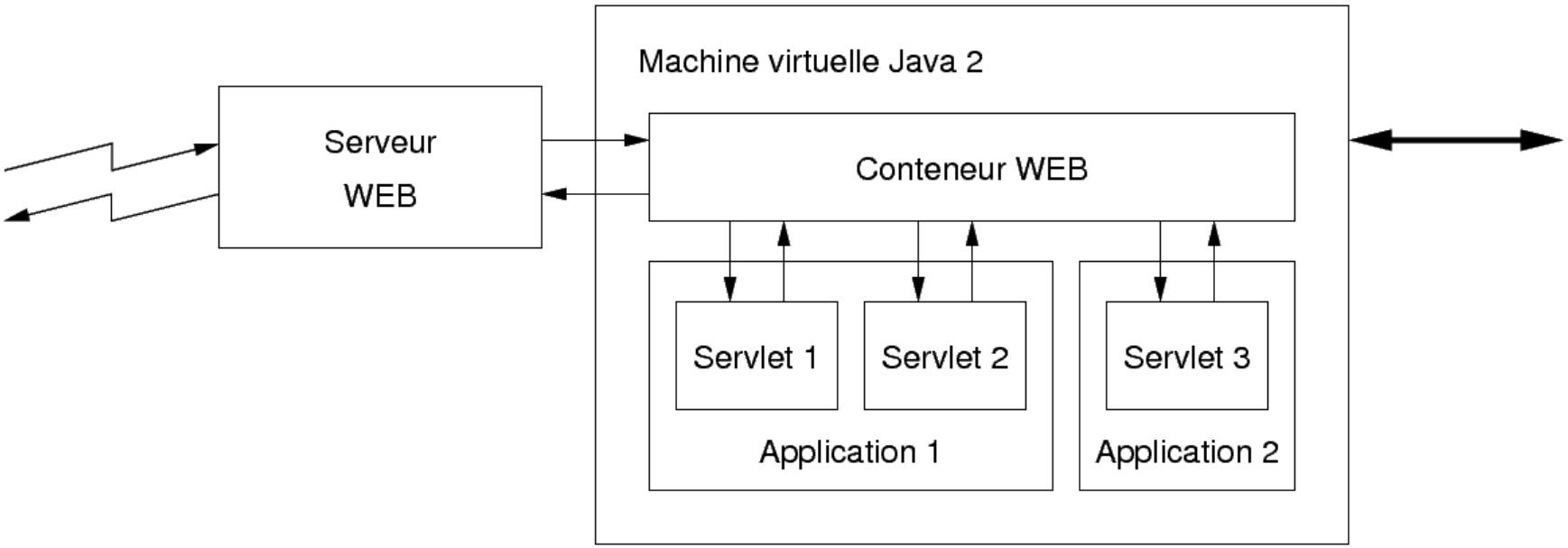
Structure

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

Sortie

Une WAR (Web Application aRchive) en fait une archive jar (qui est un ZIP).

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,

Configuration

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<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>
```

Configuration

Déclaration des servlets

```
<servlet>
    <servlet-name>UneServletSimple</servlet-name>
    <servlet-class>myapp.web.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>
```

Configuration

Mapping servlet-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>
```

Configuration

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
```

Une servlet

```
package myapp.web;

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

import myapp.business.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();
    }
```

Une servlet : 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>");
}
```

Les annotations

Adieu le XML !! Vive les annotations !! @WebServlet @Autowired @Controller @Service etc.

Servlet par annotation

```
package myapp.web;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebInitParam;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet(
        name = "productServlet",
        description = "Product Servlet",
        urlPatterns = { "/product/*", "*.prod" },
        initParams = {
                @WebInitParam(name = "param1", value = "value1",
                    description = "description1"),
        },
        loadOnStartup = 5
        )
```

Ce que fait une requête

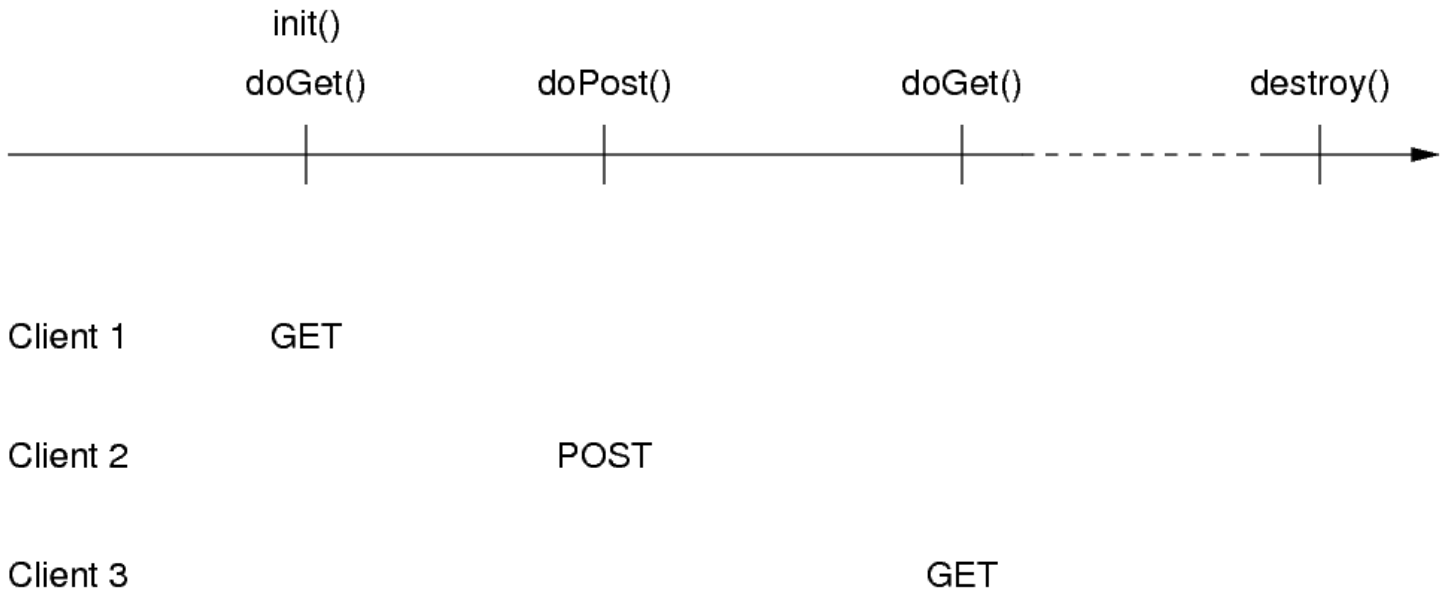
Analyse

```
protected void doGet(HttpServletRequest request, HttpServletResponse
     response)
        throws ServletException, IOException {
    response.getWriter()
            .append("ServerName: " + request.getServerName())
            .append("\ncontextPath: " + request.getContextPath())
            .append("\nServletPath: " + request.getServletPath())
            .append("\nPathInfo: " + request.getPathInfo());
}
```

Résultat

```
http://localhost:8080/myapp/promotion/car.prod
-----------------------------------------------
ServerName: localhost
contextPath: /myapp
ServletPath: /promotion/car.prod
```

Le cycle de vie d'une servlet



- 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.

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

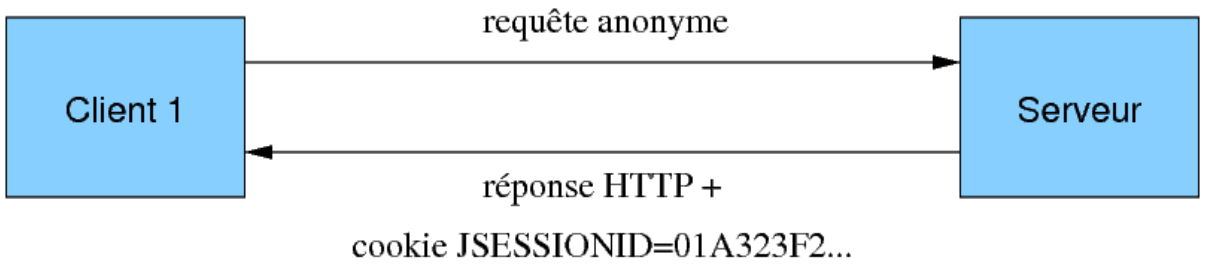
```
public void setContentType(String type)
public java.io.PrintWriter getWriter() throws ...
public ServletOutputStream getOutputStream() throws ...
public void addHeader(String name, String value)
public void addCookie(Cookie cookie)
```

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>
```

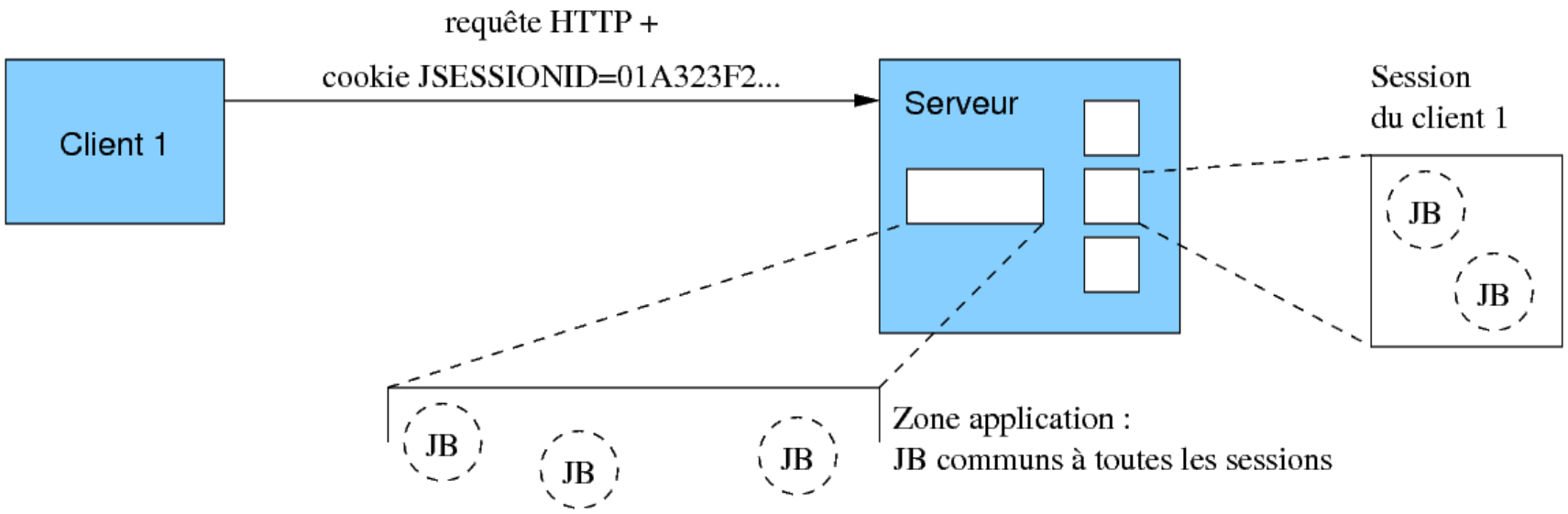
La gestion des sessions



Principe

pour identifier le client, le serveur renvoi un cookie (JESSIONID)

La gestion des sessions



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

La gestion des sessions

Interface HttpServletRequest

```
public HttpSession getSession()
```

interface javax servlet http HttpSession

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

Permettent de récupérer un objet depuis une session, de placer un objet dans une session et finalement, de vider une session.

La gestion des sessions

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>
```

La gestion des sessions

Implémenter HttpSessionBindingListener pour écouter :

- L'attachement
- Le détachement
- création
- changement de contexte
- destruction
- modification

```
// attachement
void valueBound(HttpSessionBindingEvent event) ;
// detachement
void valueUnbound(HttpSessionBindingEvent event) ;
```

La gestion des sessions : durée de vie des objets

```
// ranger un objet dans une requête
request.setAttribute("myObject", myObject);
...
// le récupérer
myObject = (MyObject) request.getAttribute("myObject");
```

Portée requête

- 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.

La gestion des sessions : durée de vie des objets

```
// ranger un objet dans une session
HttpSession session = request.getSession();
session.setAttribute("myObject", myObject);
...
// le récupérer
myObject = (MyObject) session.getAttribute("myObject");
```

Portée session

- Utilité : faire passer des données d'une requête à une autre requête émise par le même client.
- Fin de vie : fin de la session (timeout ou invalidation).

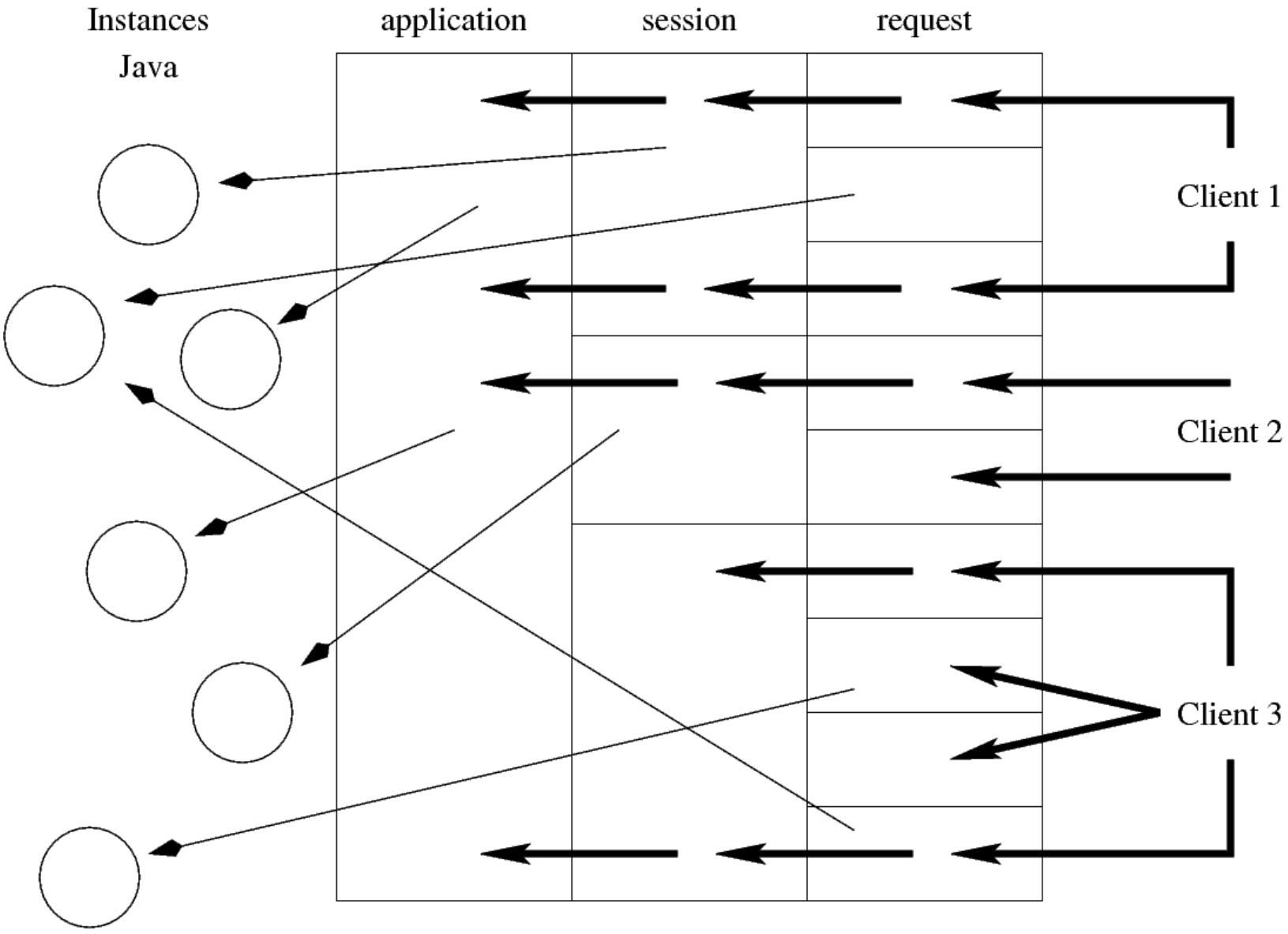
La gestion des sessions : durée de vie des objets

```
// 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");
```

Portée application

- Utilité : rendre des données ou des services accessibles à tous les clients.
- Fin de vie : fin de l'application (durée de vie très longue).

La gestion des sessions : durée de vie des objets



La vue - Les JSP

Pourquoi les JSP

Constat

- La production de pages HTML à l'aide de Servlet est une opération fastidieuse.
- Le respect d'une charte graphique est difficile.
- Les graphistes ne peuvent travailler sur des servlets.

Solution

- Introduire du code Java dans une page HTML (ou XML).
- Exécuter ce code à la volée et le remplacer par le résultat de son exécution.

JSP - principe

```
<html><body>
<p><%
  for(int i=0; i<10; i++) out.print(i + " ");
%></p>
</body></html>
```

- Directives
- Actions
- Code Java (à oublier !)

JSP - directives

```
<%@ directive attribut1=valeur1 ... %>
```

- Les JSP sont compilées en java
- Les directives vont agir sur l'étape de compilation (JSP –¿ Java).

JSP - directive page

```
<%@ page
import="java.io.*;java.sql.*"
session="true"
isThreadSafe="true"
errorPage="bug/erreur.html"
isErrorPage="false"
contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"
language="java"
%>
```

Plusieurs directives pages possibles :

```
  <%@ page import="java.io.*" %>
<%@ page import="java.sql.*" %>
<%@ page isThreadSafe="true" %>
...
```

JSP - directives include et taglib

```
  <%@ include file="banniere.html" %>
```

```
  <%@ taglib uri="monTag" prefix="jlm" %>
...
<jlm:debut> ... </jlm:debut>
```

JSP et code java

```
<%
   if (age > 30) {
      out.println("<p>L'age est supérieur à30</p>");
   } else {
      out.println("<p>L'age est inférieur à30</p>");
   }
%>
```

```
<% for(int i=0; i<10; i++) {
    %>
    <p>i = <%= i %>.</p>
    <%
    }
%>
```

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

JSP - variables implicites

- HttpServletRequest request
- HttpServletResponse response
- javax.servlet.jsp.JspWriter out
- javax.servlet.http.HttpSession session
- javax.servlet.ServletContext application
- javax.servlet.jsp.PageContext pageContext
- javax.servlet.ServletConfig config

JSP - accès aux beans

```
  <jsp:useBean id="product" scope="session" class="myapp.Product" >
    <p>Nouveau produit !</p>
</jsp:useBean>

<p>Nom: <%= product.getName() %></p>
<p>Prix: <%= product.getPrice() %></p>
<p>Desc: <%= product.getDesc() %></p>
```

EL : Expression Language

Motivation

- Les JSP classiques ne respectent pas la séparation MVC (données et logique dans la vue)
- Le code et sa syntaxe sont trop lourdes
- Simplifier l'accès pour garde l'idée de rester proche du HTML de base

Les Expression Langauge

Les EL permettent notamment de :

- Conditionner l'affichage
- Faire des calculs simples
- Accéder aux données / méthodes / paramètres de requête

Les Expression Language

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>Test des expressions EL</title>
</head>
<body>
   <p>Je m'appelle ${ prenom }</p>

   <!-- ou bien -->

   <p>Je m'appelle ${ user.prenom }</p>
</body>
</html>
```

- Permet d'accéder à l'attribut prenom
- On décide de ce qu'il y aura comme attribut dans la vue par sa requête et sa réponse
- La réponse HTTP a été enrichie avec l'attribt "prenom"

Les Expression Language

Comment donner cet attribut ?

```
// donner un attribut àla requête (servlet clasique)
req.setAttribute("test", "youhou"); // pour une servlet classique

// donner un attribut àla requête (contrôleur Spring)
@RequestMapping(value = "/el", method = RequestMethod.GET)
public String add(HttpServletRequest request, Map<String, Object>
    model){
model.put("test", "youhou"); // model est donc une map qui
    représente la requête et ses informations
...
}

// jsp : acceder àl'attribut
${ test } // cela affichera "youhou"
```

Les Expression Language

Comment récupérer les paramètres ? D'où vient-il ?

```
<form action="${pageContext.request.contextPath}/remove"
      method="POST">
       <input type="hidden" value="${product.id}" name="productId" />
       <button name="buttonRemove" type="submit" class="btn btn-danger
            btn-xs" >
          <i class="fa fa-trash fa-lg" aria-hidden="true"></i>
       </button>
</form>
```

- Formulaires
- Requêtes (POST le plus souvent)

Les Expression Language

Listing 1: Quelques tests logiques

```
${ true && false } <br />
<!-- Affiche false -->

${ 'a' > 'b' ? 'oui' : 'non' } <br />
<!-- Le résultat de la comparaison vaut false, non est affiché -->

// acceder au parametre àvaleur multiple
${ paramValues.XXX }

// acceder a une de ces valeurs
${ paramValues.XXX[index] }
```

JSTL - Java Standard Tag Library

JSTL - pourquoi ?

Limite des EL

- C'est juste un langage de simplication
- Volontairement restreint
- ne change pas les balises HTML

Avantage du JSTL

- Fonctions à utiliser dans la vue
- Conditionner l'affichage, itérer sur les éléments à afficher, etc.

Configurer JSTL

```
 <!DOCTYPE html>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<html>
<head></head>
<body></body>
</html>
```

JSTL - exemples

```
<%-- Affiche Hello World -->
<c:out value="Hello World">

<%-- Declare une variable nommee "maVar" avec valeur 250 et une
     portee sur la session -->
<c:set var = "maVar" scope = "session" value = "${50+200}"/>

<%-- Affiche le contenu de la variable "maVar" (i.e. 250) -->
<c:out value="${ maVar }">

<%-- Affiche la balise <p> si "maVar" est superieure à50 -->
<c:if test = "${ maVar > 50 }">
    <p>La variable "maVar" est supérieure à50 ! Wahou!/><p>
</c:if>

<%-- Boucle for sur un indice -->
<c:forEach var = "i" begin = "1" end = "5">
    Item <c:out value = "${i}"/><p>
</c:forEach>
```

JSTL - fait pour être combiné

Principe d'association

- Idée générale : combiner les EL et JSTL dans la Jsp
- Ne plus utiliser les anchor JSP classiques
- Ne pas accéder à la BDD, ni au Java dans la vue : utiliser les attributs et paramètres dans Request Response HTTP.

JSTL - fonctions

- fn:replace()
- fn:length()
- fn:join()
- etc.

Sources

- Documentation Oracle
- Cours JEE année précédente
- Cours de J.L Massat