DÉVELOPPEMENT WEB

JAVA ENTERPRISE EDITION – JEE

Mohammed Achkari Begdouri

Université Abdelmalek Essaadi

Faculté Polydisciplinaire à Larache - Département Informatique

achkari.prof@gmail.com

Année universitaire 2020/2021

Chapitre 3: Les servlets & JSP

DÉVELOPPEMENT WEB – JEE

SMI – S6

Présentation

- Objectifs

  Développement d'applications Web robustes

  Apprentissage d'une partie de Java EE

    Servlet, JSP, Java beans

- Prérequis

  Maîtrise du langage Java (Java SE)

  Maîtrise du développement Web client

  - XHTML
  - CSS et XML

Java EE?

- Java Enterprise Edition est un framework
  - riche (Java SE + nombreuses API)
  - ouvert
  - dédié au développement, au déploiement et à l'exécution d'applications Web solide

- Favorise la séparation des préoccupations
  - Code métier vs. Propriétés non-fonctionnelles

    persistance (JPA, Hibernate), sécurité, transaction (Spring)

Le développement Web

- Pages statiques
  - Pages HTML préparées à l'avance
  - Le serveur renvoie les pages sans effectuer de traitement Particulier

- Pages dynamiques
  - Pages HTML générées par le serveur
  - Le serveur construit la réponse en fonction de la requête de l'utilisateur

Limites des pages statiques

- Pas de contenu dynamique

```
<html>
  <head>
    <title>Clock</title>
  </head>
  <body>
    Il est toujours 12:12.
  </body>
</html>
```

statique

```
<html>
  <head>
    <title>Clock</title>
  </head>
  <body>
    Il est [getTimeOnServer].
  </body>
</html>
```

dynamique

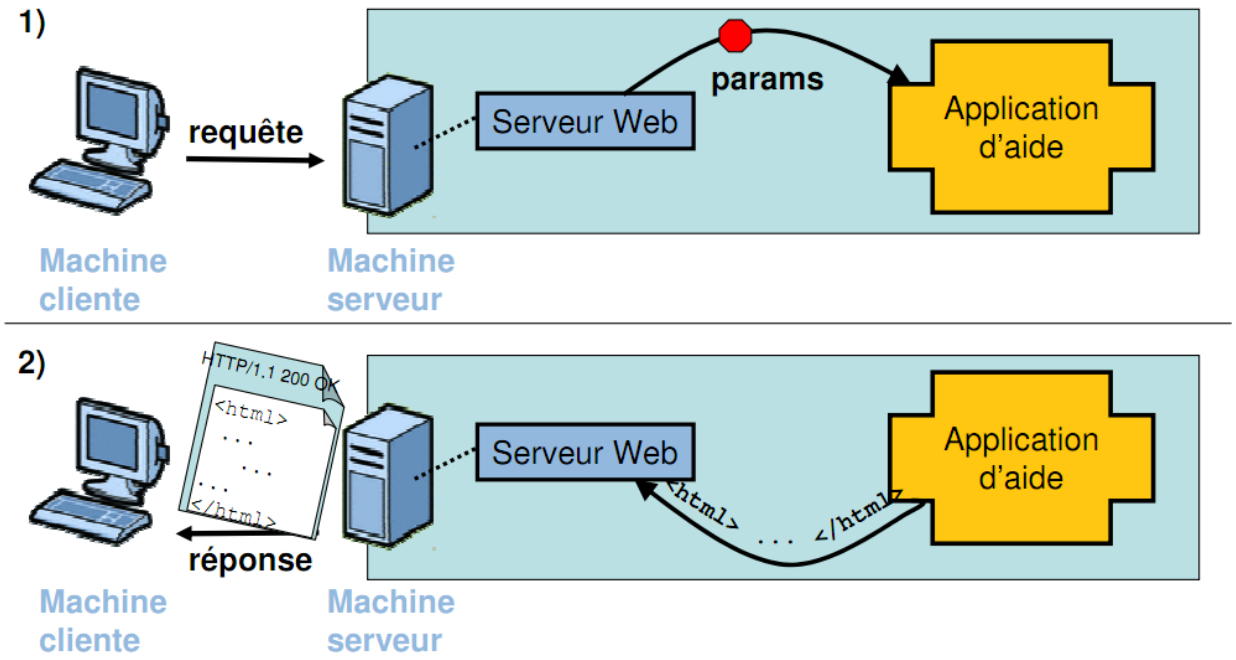
- Pas de sauvegarde de données sur le serveur
  - Traitemenent de formulaires :
    - besoin d'une application d'aide au serveur Web

      pour évaluer les paramètres reçus

      pour générer une réponse appropriée

Pages dynamiques

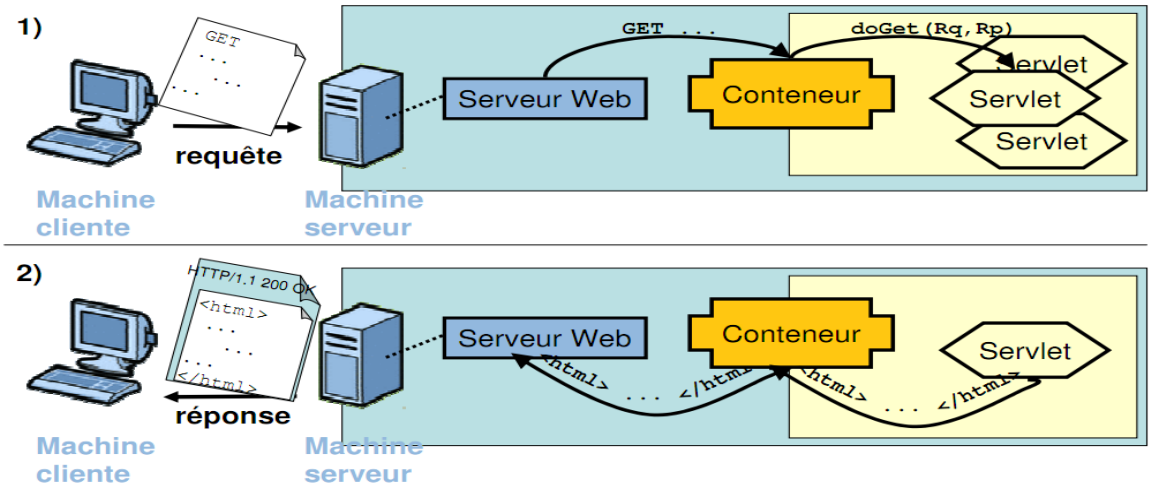
- Le serveur Web a besoin d'aide pour gérer tous ce qui est dynamique
  - C'est le rôle du **CONTENEUR** Java EE avec servlets (ex : Tomcat)



LES SERVLETS

Conteneur

- Un conteneur est un composant logiciel système qui contrôle d'autres composants, dits métier
  - Tomcat est un exemple de conteneur
  - Les servlets n'ont pas de méthode main(), ils sont contrôlés par le conteneur Tomcat
  - Les requêtes ne sont pas adressées aux servlets mais au conteneur dans lequel ils sont déployés

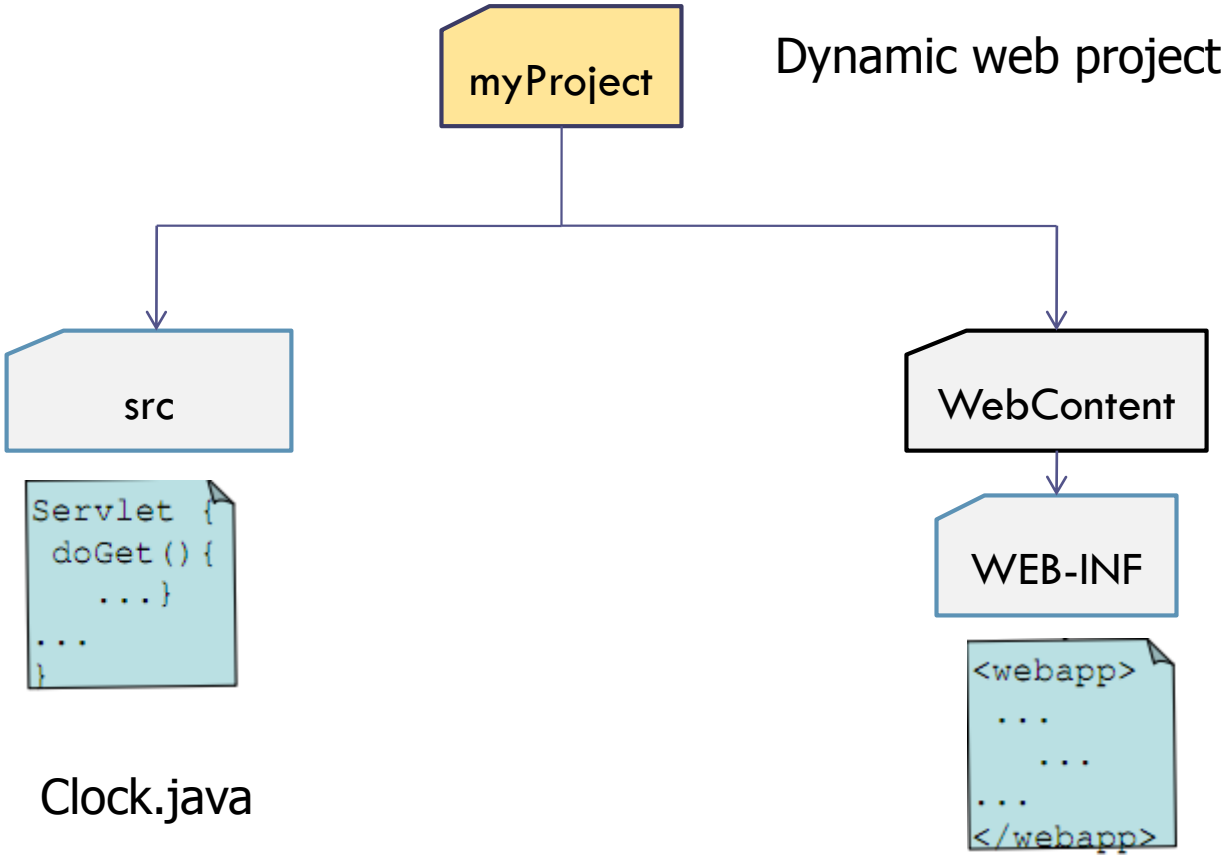


Pourquoi un conteneur ?

- Un conteneur fournit pour les Servlets
  - Un support pour la communication
    - Pas besoin de ServerSocket, Socket, Stream, …
  - La gestion du cycle de vie
  - Un support pour le Multithreading
    - Création automatique des Threads
  - Un support pour les JSP

Servlet en 4 étapes (1/4)

Créer l'arborescence suivante



Imposé par la spécification des servlets

Et la mettre dans
%TOMCAT_HOME%\webapps\myProject

Servlet en 4 étapes (2/4)

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

public class Clock extends HttpServlet {

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

        String title = "Clock";

        reponse.setContentType("text/html");
        PrintWriter out = reponse.getWriter();

        out.println("<html><head><title>"+title+"</title></head>"+
                    "<body><h1>Time On Server</h1>"+
                    new java.util.Date() +
                    "</body></html>");
        out.close();
    }
}
```

Redéfinition de la méthode doGet

Code html incorporé dans du java

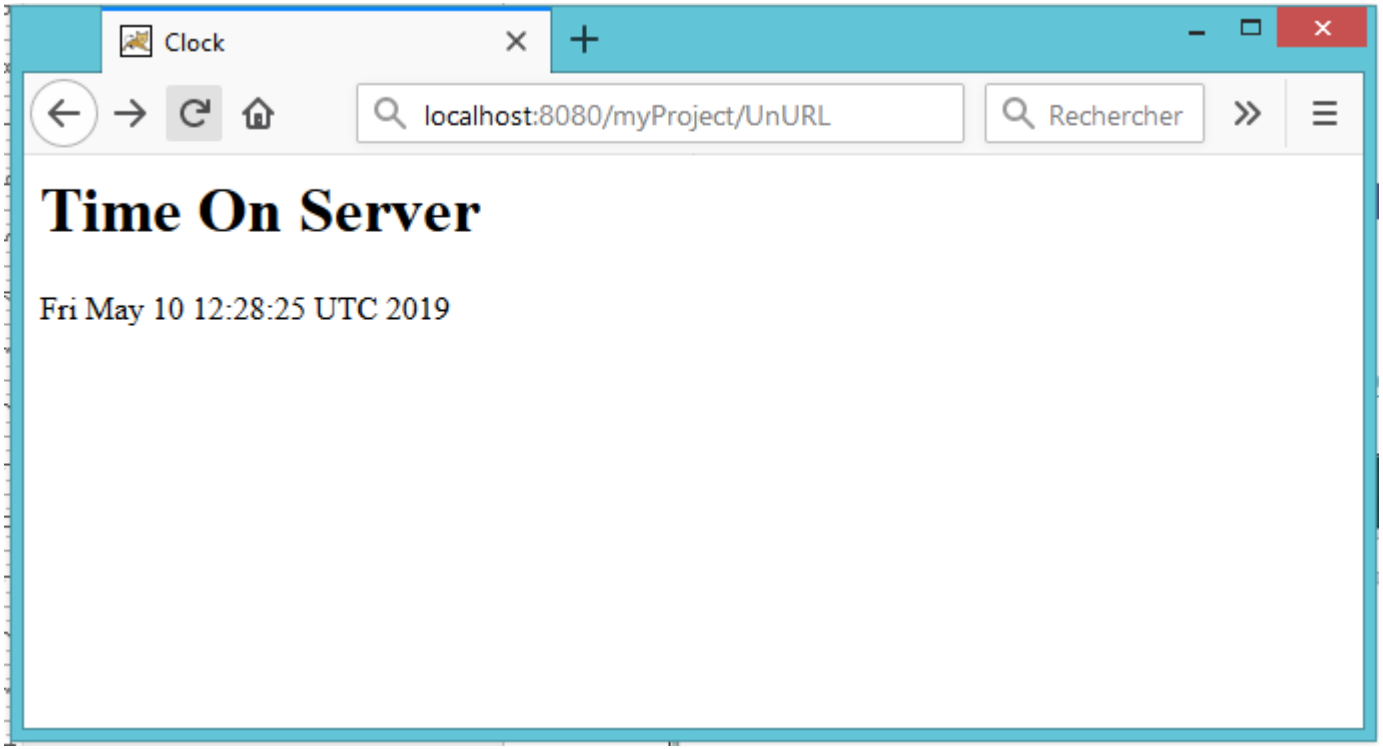
Servlet en 4 étapes (3/4)

Créer un descripteur de déploiement et le mettre dans WEB-INF

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

    <display-name>WebModule1</display-name>

    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
    </welcome-file-list>

    <context-param>
        <param-name>contextParam</param-name>
        <param-value>
            Paramètre global à l'application
        </param-value>
    </context-param>

    <servlet>
        <servlet-name>The Clock</servlet-name>
        <servlet-class>smi.tp.Clock</servlet-class>
        <init-param>
            <param-name>titrePage</param-name>
            <param-value>Nom de la page</param-value>
        </init-param>
    </servlet>

    <servlet-mapping>
        <servlet-name>The Clock</servlet-name>
        <url-pattern>/UnURL</url-pattern>
    </servlet-mapping>

</web-app>
```

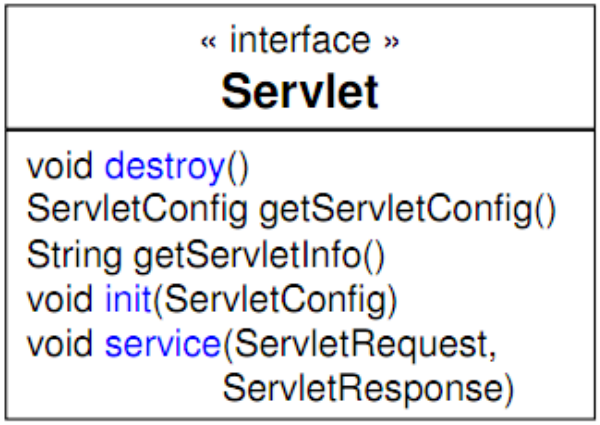
Servlet en 4 étapes (4/4)

- **Démarrer le serveur (Tomcat)**
- **Compiler la servlet**
- **Lancer un navigateur et taper http://localhost:8080/myProject/UnURL**



Servlets

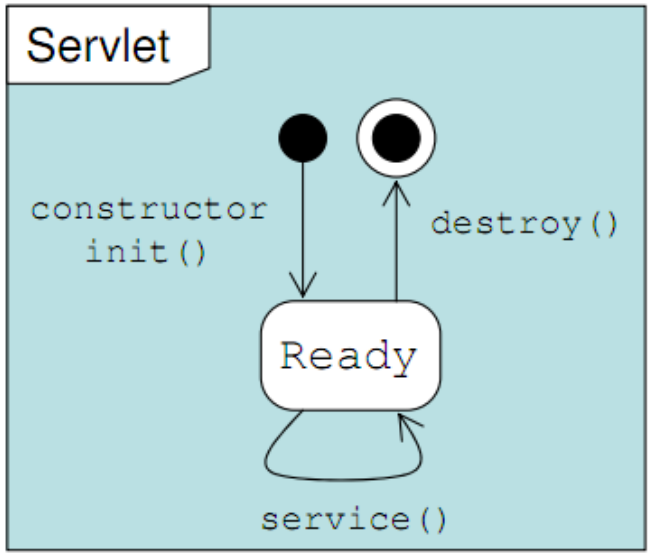
- Une servlet est un objet qui peut être manipulé par le conteneur via l'interface suivante :



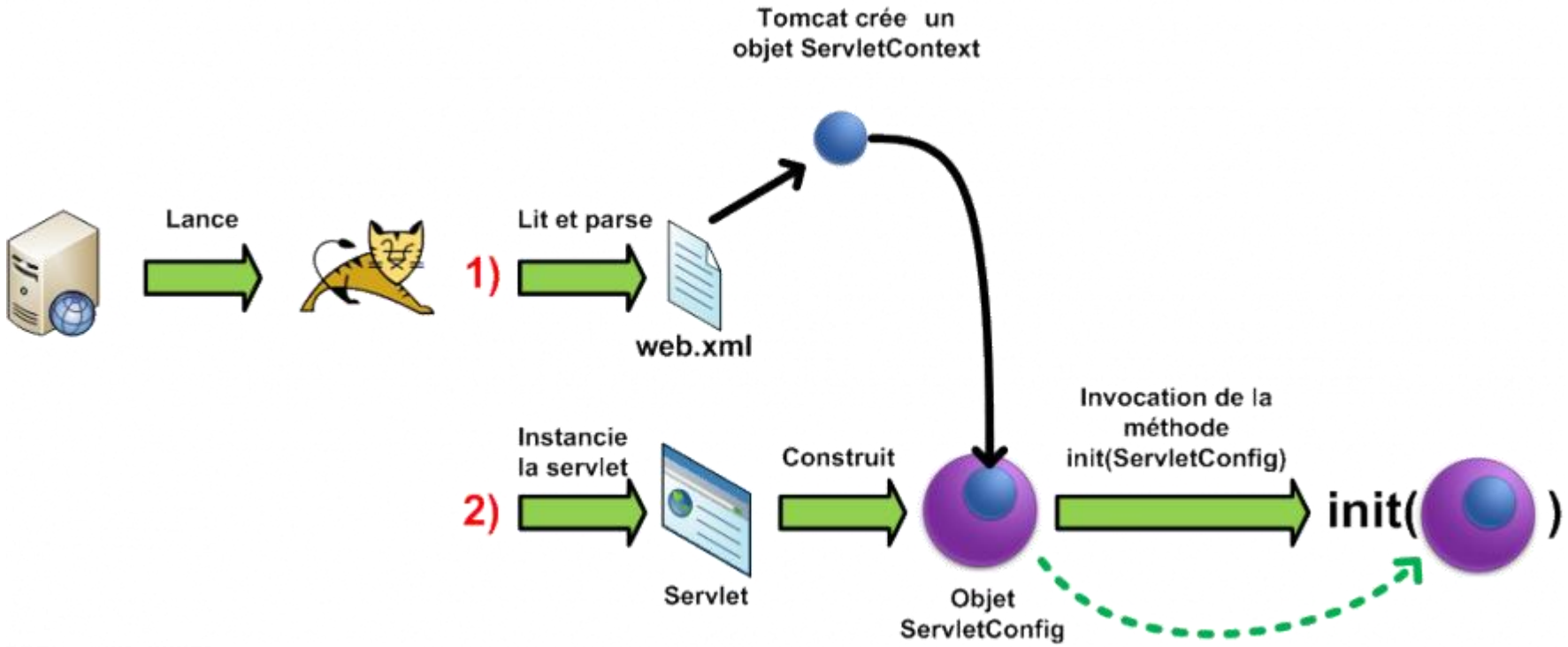
- Lorsque le conteneur reçoit une requête, il la transmet au servlet qui correspond à l'URL pour que la requête soit traitée

Cycle de vie d'une servlet

1. Chargement de la classe
2. Instanciation du servlet
   - constructeur par défaut
3. Appel de init()
4. Appel(s) de service()
   - 1 thread par requête
5. Appel de destroy()

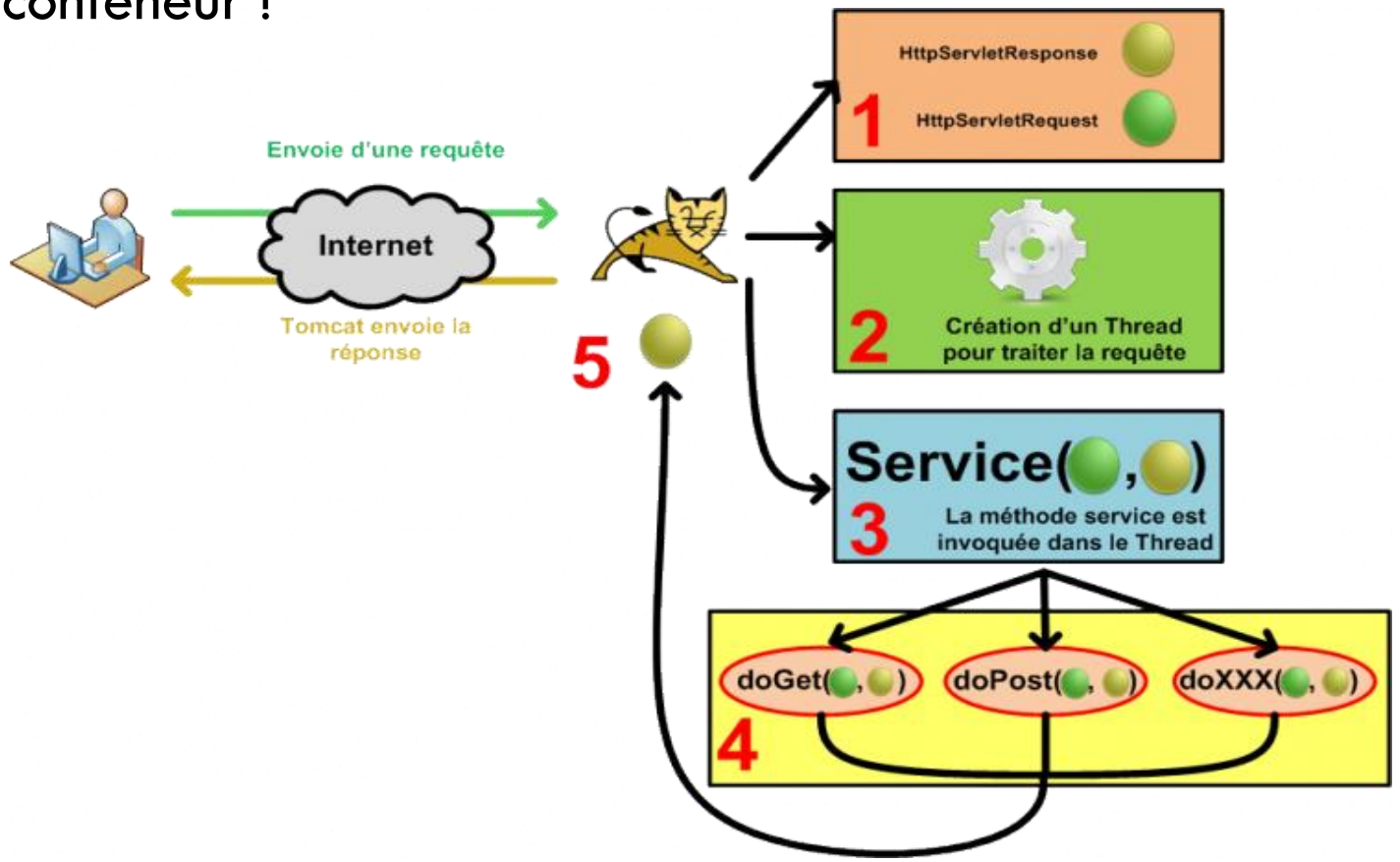


Instantiation d'une servlet



Utilisation des servlets

Il n'existe qu'une et une seule instance d'une servlet dans le conteneur !



Formulaire GET

```
<form method="get" action="Serv2">
        <table border="0">
                <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 colspan="2" align="center">
                                <input type="submit" value=" Envoyer"/>
                        </td>
                </tr>
        </table>
</form>
```

Traitement formulaire GET

```
public class LogServlet extends HttpServlet {

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

        String login = request.getParameter("nom") ;
        String password = request.getParameter("prenom") ;
        //WhatYouWant
        if (checkUserAndPassword(login, password)) {
        grantAccessTo(login);
        } else {
        sendAuthentificationFailure(login);
        }
    }
}
```

Formulaire POST

```
<form method="post" action="Serv2">
        <table border="0">
                <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 colspan="2" align="center">
                                <input type="submit" value=" Envoyer"/>
                        </td>
                </tr>
        </table>
</form>
```

Traitement formulaire Post

```
public class LogServlet extends HttpServlet {
    public void doPost(HttpServletRequest request,HttpServletResponse response) throws
        IOException {
    doGet(request, reponse);
    }
}
```

Paramètres de formulaires

```
public class ParametresFormulaire extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse reponse) throws
    Exception {

     Enumeration<?> paramètres = request.getParameterNames();
          while(paramètres.hasMoreElements()){
               String param = (String) paramètres.nextElement();
               String paramValue = request.getParameter (param);
               out.println(param+" : " + paramValue);
               }
     }
     out.close();
}
```

Redirection via les servlets

- response.sendError(int code, String msg)

- response.sendRedirect(String url)

Exemple sendRedirect

```
public class WrongDestination extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
    {
            response.sendRedirect("https://www.oracle.com");
    }
}
```

Redirection au niveau client : l'URL affiché chez le client est modifié

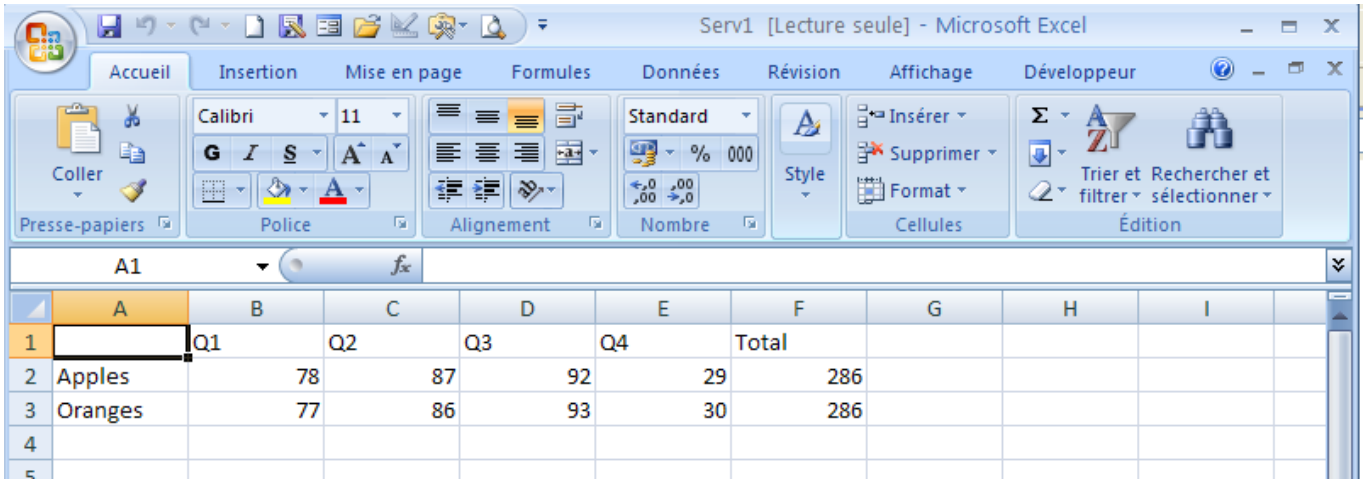
Contrairement à :

request.getRequestDispatcher("Home").forward(request, response);

La redirection se fait au niveau serveur et renvoi la réponse au client sans changer l'url affiché chez le client

Générer un fichier Excel

```
public class FichierExcel extends HttpServlet {
    public void doGet(HttpServletRequest requete, HttpServletResponse reponse) throws Exception
    {
            reponse.setContentType("application/vnd.ms-excel");
            PrintWriter out = reponse.getWriter();
            out.println("\tQ1\tQ2\tQ3\tQ4\tTotal");
            out.println("Apples\t78\t87\t92\t29\t=SOMME(B2:E2)");
            out.println("Oranges\t77\t86\t93\t30\t=SOMME(B3:E3)");
    }
}
```



	A	B	C	D	E	F
1		Q1	Q2	Q3	Q4	Total
2	Apples	78	87	92	29	286
3	Oranges	77	86	93	30	286
4						

LES PAGES JSP

Problème : HTML dans Java

- C'est laid !

```
out.println(docType);
out.println("<html>");
out.println("<head>\n<title>Clock</title>\n</head>");
out.println("<body>\n" +
"<h1>Time on server</h1>\n" +
"<p>" + today + "</p>\n" +
"</body>");
out.println("</html>");
```

- Difficile de séparer les différentes tâches du développement

  (code Web vs. code métier)

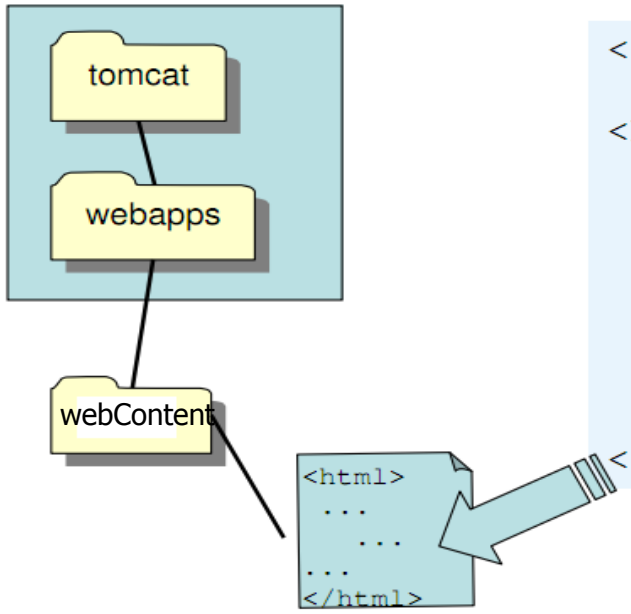
Solution : Java dans HTML

- Une JSP est identique à une page HTML sauf que l'on peut rajouter du Java dedans

```
<!DOCTYPE HTML PUBLIC"-//W3C//DTD HTML 4.0
   Transitional//EN">
<html>
<head><title>Clock</title></head>
<body>
<h1>Time on server</h1>
<p> <%= new java.util.Date() %> </p>
</body>
</html>
```

Une JSP en 1 étape

- Ecrire une JSP Clock.jsp et la mettre à la racine du dossier web au même endroit que les fichier HTML.




```
<!DOCTYPE HTML PUBLIC
 "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
  <head>
    <title>Clock</title>
  </head>
  <body>
    <h1>Time on server</h1>
    <p> <%= new java.util.Date() %> </p>
  </body>
</html>
```

Les JSP c'est quoi?

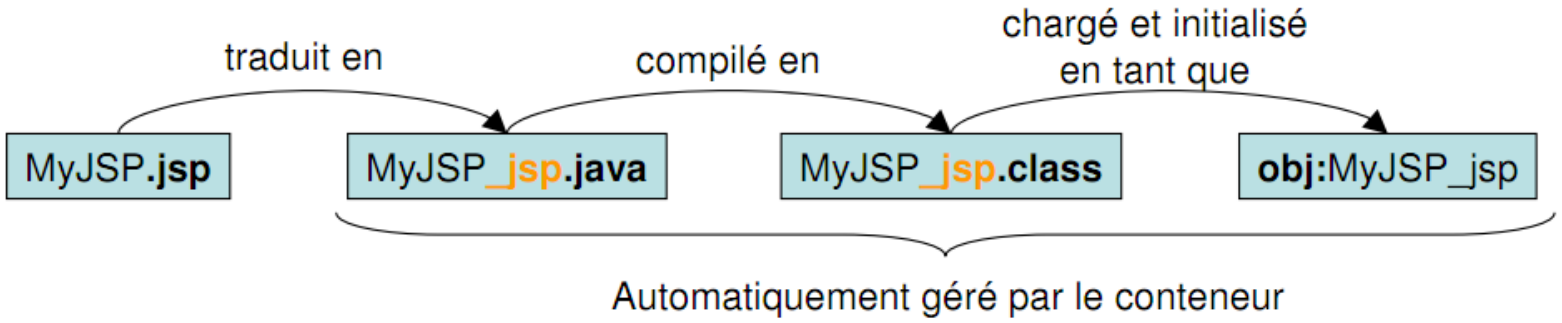
- Les servlets facilitent le traitement des requêtes et réponses HTTP, mais ils ne sont pas appropriés à l'écriture de code HTML

```
out.println("<html><head><title>"+title+"</title>…");
```

- Les JSP permettent d'intégrer du code java dans une page HTML

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

- Mais au final une JSP n'est qu'une servlet !



Correspondance JSP/Servlet

- **JSP d'origine**

```
<h1>Time on server</h1>
<p><%= new java.util.Date() %></p>
```

- **Servlet générée par Tomcat**

```
public final class Clock_jsp
    extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent {
 public void _jspService(HttpServletRequest request,
                         HttpServletResponse response)
       throws java.io.IOException, ServletException {

      response.setContentType("text/html");
      JspWriter out = response.getWriter();
      // ...
      out.write("<h1>Time on server</h1>\r\n");
      out.write("<p> ");
      out.print( new java.util.Date() );
      out.write("</p>\r\n");
      // ... }
```

Stratégie de conception : Limiter le code Java dans les JSP

- Deux options
  - Ecrire 25 lignes de code directement dans une JSP
  - Ecrire ces 25 lignes dans une classe Java à part et 1 ligne dans une JSP pour l'invoquer

- Pourquoi la 2e option est vraiment meilleure ?
  - Développement: Ecriture de la classe dans un environnement Java et pas HTML
  - Débogage: S'il y a des erreurs, elles sont visibles à la compilation
  - Réutilisation: Utilisation de la même classe dans différentes pages JSP

Syntaxe de base

- Texte HTML

  <h1>Blah</h1>

  Passé au client. Réellement traduit en servlet par le code

  ```
  out.print("<h1>Blah</h1>");
  ```

- Commentaires HTML

  <!-- Commentaire -->

  Envoyés au client

- Commentaires JSP

  <%-- Commentaires --%>

  Ne sont pas envoyés au client

Types des éléments de scripts

- Expressions
  - Format : <%= expression %>
  - Evaluée et insérée dans la sortie du servlet se traduit par out.print(expression)
- Scriptlets
  - Format : <% code %>
  - Inséré tel quel dans la méthode _jspService du servlet (appelée par service())
- Déclarations
  - Format : <%! Expression %>
  - Insérée telle quelle dans le corps de la classe servlet, en dehors de toute méthode existante

Expressions JSP

- Format

  <%= Expression Java %>

- Résultat
  - Expression évaluée, convertie en String, et placée dans la page HTML à la place qu'elle occupe dans la JSP
  - L'expression est placée dans _jspService en paramètre du out.print()

- Exemples
  - Heure courante : <%= new java.util.Date() %>

- Syntaxe compatible XML
  - <jsp:expression>Java Expression</jsp:expression>

  On ne peut pas mixer les deux versions dans une même page. Il faut utiliser XML pour la page entière si on utilise jsp:expression

Correspondance JSP/Servlet

- **JSP d'origine**

```
<h1>Un nombre aléatoire</h1>
<%= Math.random() %>
```

- **Code du servlet résultant de la traduction**

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

    response.setContentType("text/html");
    JspWriter out = response.getWriter();
    out.println("<h1>Un nombre aléatoire</h1>");
    out.println(Math.random());
}
```

Variables prédéfinies

- **request**

  Instance de HttpServletRequest (1e argument de service/doGet)

- **response**

  Instance de HttpServletResponse (2e argument de service/doGet)

- **out**

  Intance de JspWriter (une version bufferisée de Writer) utilisée pour envoyer des donnés sur la sortie vers le client

- **session**

  Instance de HttpSession associée à la requête

- **application**

  Instance de ServletContext (pour partager des données) telle que obtenue via getServletContext()

Exemple : Lire des paramètres de la requête

```
<!DOCTYPE …>
<html>
    <head>
        <title>Reading Three Request Parameters</title>
    </head>
    <body>
        <h1>Reading Three Request Parameters</h1>
        <ul>
            <li><b>param1</b>:  <%= request.getParameter("param1") %></li>
        </ul>
    </body>
</html>
```

Scriptlets JSP

- **Format**

  <% Code Java %>

- **Résultat**

  Code inséré tel quel dans _jspService()

- **Exemple**

```
<%
String queryData = request.getQueryString();
out.println("Attached GET data: " + queryData);
%>
<% response.setContentType("text/plain"); %>
```

- **Syntaxe XML**

  <jsp:scriptlet>Code Java</jsp:scriptlet>

Correspondance JSP/Servlet

- JSP d'origine

```
<h2>foo</h2>
<%= bar() %>
<% baz(); %>
```

- Code du servlet résultant de la traduction

```
public void _jspService(HttpServletRequest request,HttpServletResponse  response) throws
ServletException, IOException {
        response.setContentType("text/html");
        HttpSession session = request.getSession();
        JspWriter out = response.getWriter();
        out.println("<h2>foo</h2>");
        out.println(bar());
        baz();
    }
```

Déclarations JSP

- **Format**

  <%! Java Code %>

- **Résultat**

  Insérées telle quelle dans la définition de la classe du servlet, en dehors de toute méthode existante

- **Exemples**

  <%! private int someField = 5; %>

  <%! private void someMethod(...) {...} %>

- **Remarque de conception**

  Les attributs sont clairement utiles. Pour les méthodes, il est la plupart du temps préférable de les définir dans une classe Java séparée

- **Syntaxe XML**

  <jsp:declaration>Code Java</jsp:declaration>

Correspondance JSP/Servlet

- **JSP d'origine**

```
<h1>Some Heading</h1>
<%!
private String randomHeading() {
return("<h2>" + Math.random() + "</h2>");
}
%>
<%= randomHeading() %>
```

(Alternative : créer randomHeading en méthode statique dans une classe Java séparée)

Correspondance JSP/Servlet

- Code du servlet résultant de la traduction

```
public class MyJSP_jsp implements HttpJspPage {

        private String randomHeading() {
            return("<h2>" + Math.random() + "</h2>");
        }

    public void _jspService(HttpServletRequest request, HttpServletResponse response) throws Exception {

        response.setContentType("text/html");
        HttpSession session = request.getSession();
        JspWriter out = response.getWriter();
        out.println("<h1>Some Heading</h1>");
        out.println(randomHeading());
        }
}
```

Déclarations JSP : Exemple

```
<!DOCTYPE …>
<html>
    <head>
    <title>JSP Declarations</title>
    </head>
    <body>
        <h1>JSP Declarations</h1>
        <%! private int accessCount = 0; %>
        <h2>Accesses to page since server reboot:
        <%= ++accessCount %></h2>
   </body>
</html>
```

Déclarations JSP et variables prédéfinies

- Remarque : Problème
  - Les variables prédéfines (request, response, out, session, etc.)

  sont locales à la méthode _jspService.

  Ainsi, elles ne sont pas disponibles pour les méthodes définies par des déclarations JSP et les méthodes des classes externes.
  - Que peut-on faire ?
- Solution : les passer en paramètres. Ex :

```
<%!
private void someMethod(HttpSession s) {
doSomethingWith(s);
}
%>
<% someMethod(session); %>
```

Types des éléments de scripts JSP : les directives de page

- Donnent des informations sur la servlet qui sera généré pour la page JSP

- Principalement utilisées pour :
  - L'importation de classes et paquetages
  - Le type MIME généré par la JSP

L'attribut import

- Format

  <%@ page import="paquetage.classe" %>

  <%@ page import="paquetage.classe1,...,paquetage.classeN" %>

- But

  Générer les instructions d'importation