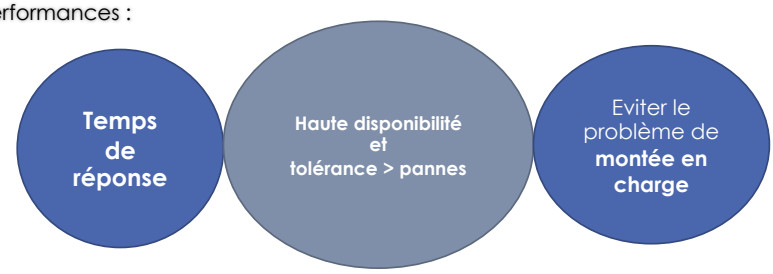
Cours 1 : Architecture Java EE

El MIDAOUI OMAR
EMSI Rabat
P_EL-MIDAOUI.Omar@emsi-edu.ma

Qu'exige un projet informatique ?

- Besoins ou Exigences fonctionnelles :
  - Une application est créée pour répondre , tout d'abord, aux besoins fonctionnels de l'entreprise.
- Exigences Techniques :
  - En Terme de performances :



2

Exigence d'un projet informatique

- Exigences Techniques :
  - En Terme de maintenance :
    - ✓ Une application doit évoluer dans le temps.
    - ✓ Doit être fermée à la modification et ouverte à l'extension
  - En Terme de Sécurité
  - Portabilité
  - Distribution
  - Capacité de communiquer avec d'autres applications distantes.
  - Capacité de fournir le service à différents type de clients (Desk TOP, Mobile, SMS, http…)
  - …
  - Coût du logiciel

3

C'est Quoi La solution ???

- On peut constater facilement qu'il est très difficile de développer un système logiciel qui respecte ces exigences sans utiliser l'expérience des autres

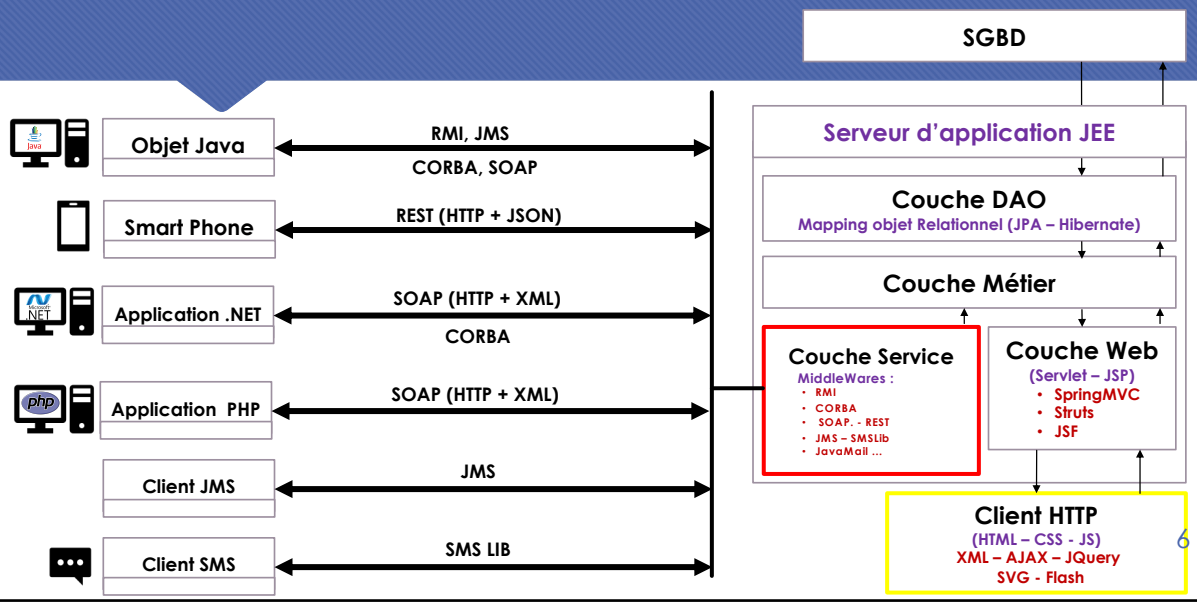
4

Les Architectures existants

- Serveur d'application JEE:
  - JBOSS
  - Web Sphere,
  - GlassFish,
  - Tomcat,
  - …
- Frameworks :
  - Mapping objet relationnel (ORM) : JPA, Hibernate, Toplink, …
  - Applications Web : Struts, JSF, SpringMVC
- Framework pour l'Inversion de contrôle:
  - Spring (Conteneur léger)
  - EJB (Conteneur lourd)
- middleware
  - RMI, CORBA : Applications distribuées
  - JAXWS pour Web services SOAP
  - JAXRS pour les Web services RESTful
  - JMS : Communication asynchrone entre les application
  - …

5

Architecture Distribué se schématise comme suit :



6

Plateforme JAVA-EE

C'est une **Plate-forme de choix pour les Grande Entreprise**

Pour développer des sites web Solides, Robuste et bien structurés

Très utilisés en Finance : Banques, ou encore même dans les sites du gouvernement, …

.

C'est une Plateforme de confiance, qui a acquis une très grande maturité professionnelle

7

Concrètement : qu'est ce que JEE ?

Java

Basé sur le langage JAVA

Bien connu et bien établit dans le monde des entreprises…

Par dessus JAVA

Java EE™

JEE = JAVA + un ensemble de bibliothèques

Ajoutant des fonctionnalités a JAVA …

Aujourd'hui très majoritairement utilisé pour réaliser des sites WEB solides robuste et bien structurés

C'est une Plateforme de développement d'application WEB comme :

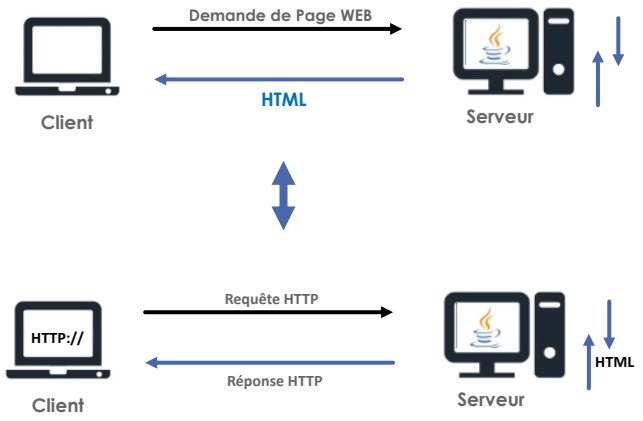
php

django Pour Python

ASP.NET

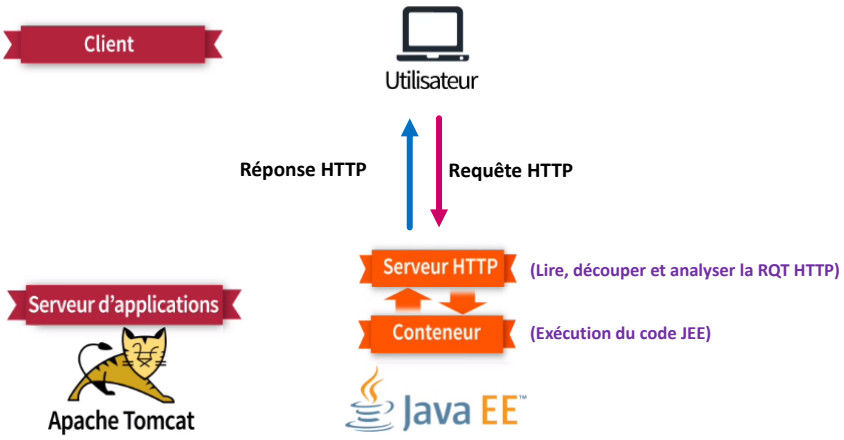
8

JAVA EE : Est un Echange Client Serveur



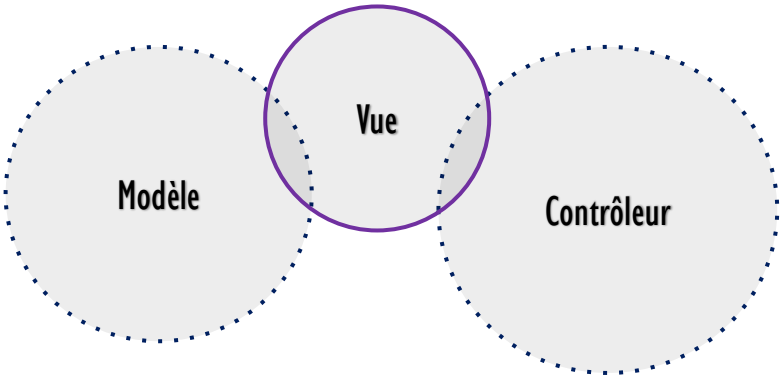
9

JAVA EE : Est un Echange Client Serveur



10

Le Design Pattern (Modèle) MVC



11

Le Design Pattern (Modèle) MVC

JEE n'impose aucun rangement de code

C'est à dire on peut développer n'importe comment en JEE

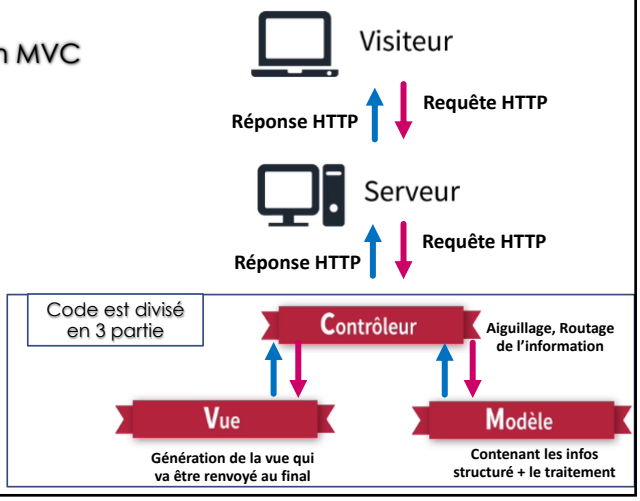
En faisant ça notre code va ressembler vite a ce bureau

trop difficile de retrouver une information

12

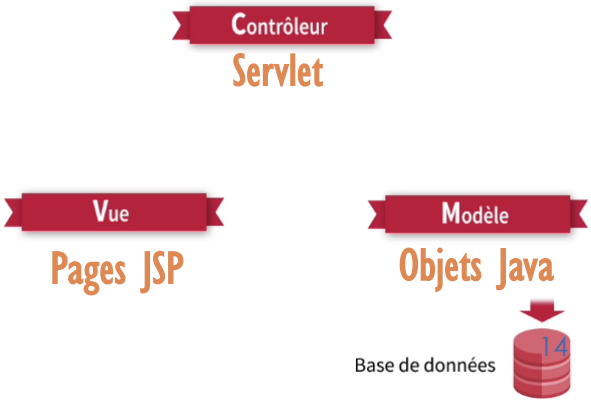
Le Design Pattern (Modèle) MVC

La Solution c'est d'utiliser :
Une Bonne Pratique de développement : Pattern MVC



Le Design Pattern (Modèle) MVC

Dans le pattern MVC chaque partie est représenté par un composant JEE



Le Modèle MVC

Ce cours utilise Java EE pure, **Sans Framework**
ce qui va nous permettre de comprendre les bases de JEE

15

Le Design Pattern (Modèle) MVC

Structurés en MVC

Struts

16

Le Protocole HTTP

17

Le Protocole HTTP

HTTP :HyperText Tranfert Protocol

- Protocole qui permet au client de récupérer des documents du serveur
- Ces documents peuvent être statiques (contenu qui ne change pas : HTML, PDF, Image, etc..) ou dynamiques (Contenu généré dynamiquement au moment de la requête : PHP, JSP, ASP…)
- Ce protocole permet également de soumissionner les formulaires

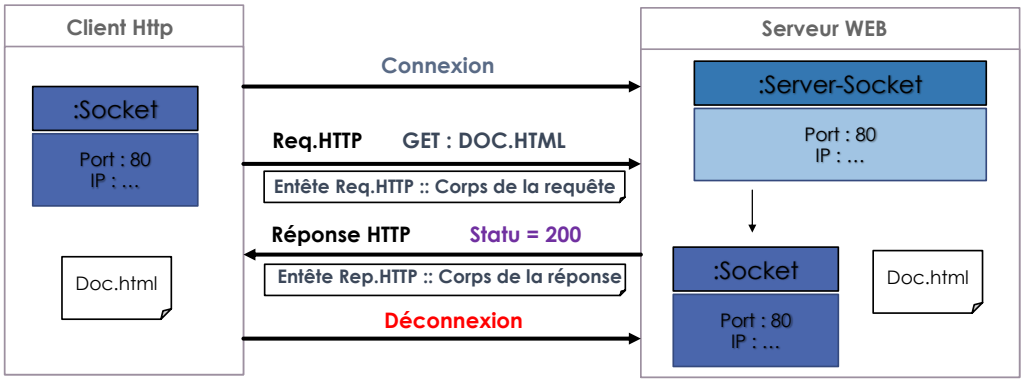
Fonctionnement (très simple en HTTP/1.0)

- Le client se connecte au serveur (Créer une socket)
- Le client demande au serveur un document : Requête HTTP
- Le serveur renvoi au client le document (**status=200**)
- ou Le serveur renvoi une erreur (**status=404)** quand le document n'existe pas
- Déconnexion

18

Le Protocole HTTP

Pour que le client le le serveur communiquent : deux sockets sont créés des deux côtés



Les sockets sont détruite après chaque connexion

19

Méthodes du Protocole HTTP

✓ Une requête HTTP peut être envoyée en utilisant les méthodes suivantes:

- **GET** : Pour récupérer le contenu d'un document
- **POST** : Pour soumissionner des formulaires (Envoyer, dans la requête, des données saisies par l'utilisateur)
- **PUT** pour envoyer un fichier du client vers le serveur
- **DELETE** permet de demander au serveur de supprimer un document.
- **HEAD** permet de récupérer les informations sur un document (Type, Capacité, Date de dernière modification etc…)

✓ Avec HTTP/1.1:

- **OPTIONS** : Permet au client HTTP d'interroger le serveur pour fournir les options de communications à utiliser pour une ressource ciblée ou un ensemble de ressources
  (Méthodes autorisées, Entêtes autorisées, Origines autorisés, …)
  -> utilisé souvent comme requête de pré-vérification CROSS-ORIGINE.
  souvent pour des raisons de sécurité les navigateur refusent l'envoie de requête d'un domaine vers un autre à mois que si le domaine cible autorise cette opération

20

Exemple de Requête HTTP avec POST

POST /login **HTTP/1.1**
HOST : www.domaine.net
ACCEPT : application/json
Content-type : application/x-www-form-urlencoded
Cookie : JSESSIONID : C4512152018CFDDOP345677

// Saut de ligne

username=admin & password=1234 & action=login

Entête de la requête

Corps de la requête

21

Exemple de Requête HTTP avec POST

POST /login **HTTP/1.0**
HOST : www.domaine.net
ACCEPT : application/json
Content-type : application/json
Cookie : JSESSIONID : C4512152018CFDDOP345677

// Saut de ligne

{ "username" : " admin " , "password" : "1234" , "action" : " login" }

Entête de la requête

Corps de la requête

22

Exemple de Requête HTTP avec GET

GET /login?**username=admin & password=1234 & action=ok** **HTTP/1.0**
HOST : www.domaine.net
ACCEPT : application/json
Content-type : application/x-www-form-urlencoded
Cookie : JSESSIONID : C4512152018CFDDOP345677

// Saut de ligne

Entête de la requête

Corps de la requête

23

Exemple de Réponse HTTP

HTTP/1.1 200 OK
Date : Wed, 05Feb02 15:02:01 GMT
Server : Apache/1.3.24
Last-Modified : Wed, 02Oct01 24:02:01 GMT
Content-type : application/json
Content-length : 77
SET-Cookie : JSESSIONID : C4512152018CFDDOP345677

// Saut de ligne

```
[
    { "id" : 1 , "TaskName" : "T1"}, { "id" : 2 , "TaskName" : "T2"},

    { "id" : 3 , "TaskName" : "T3"}
]
```

Entête de la réponse

Corps de la Réponse

24

Code de Statu

- ✓ Lorsque le serveur renvoie un document, il lui associe un code e statu renseignant ainsi le client sur le résultat de la requête (requête invalide, document non trouvé …)
- ✓ Ci dessous Les principaux code de statu HTTP :
  - **Information 1xx** :
    - 100 (Continue): Utilisé dans le cas où la requête possède un corps volumineux.
    - 101 (switching protocole) : Demander au client de changer de protocole.
      i.e. switching de HTTP 1.0 vers HTTP 1.1
  - **Succès 2xx** :
    - 200 (OK): Le document a été trouvé et son contenu suit…
    - 201 (Created) : Le document a été créé en réponse à un **PUT**
    - 202 (Accepted) : Requête acceptée, mais traitement non terminé.
    - 204 (No Response) : Le serveur n'a aucune information à renvoyer.
    - 206 (Partial Content) : une partie du document suit…

25

Code de Statu

- **Redirection 3xx** :
  - 301 (Moved): Le document a changé d'adresse d'une façon permanente
  - 302 (Found) : Le document a changé d'adresse temporairement.
  - 304 (Not modified) : Le document demandé n'a pas été modifié.
- **Erreurs du Client 4xx** :
  - 400 (Bad Request): La syntaxe de la requête est incorrecte…
  - 401 (Unauthorized) : Le client n'a pas les privilèges d'accès au document
  - 403 (Forbidden) : L'accès au document est interdit
  - 404 (Not Found) : Le document demandé n'a pu être trouvé
  - 405 (Method not allowed) : La méthode de la requête n'est pas autorisée.
- **Erreurs du Serveur 5xx** :
  - 500 (Internal Error): Une erreur inattendu est survenue au niveau du serveur…
  - 501 (Not implemented) : La méthode utilisé n'est pas implémentée.
  - 502 (Bad Gateway) : Erreur du serveur distant lors d'une requête proxy.

26

Entêtes HTTP

- **Entêtes HTTP génériques** :
  - Content-length : Longueur en octets des données suivant les en-têtes
  - Content-Type: type des données qui suivent.
  - Connection: indique si la connexion TCP doit rester ouverte (Keep-Alive) ou être fermée (close).
- **Entêtes de la requête** :
  - Accept. : type que le client accepte et attend.
  - Accept-encoding: type d'encodage que j'accepte.
  - Accept-language: le code de la langue utilisé par le navigateur
  - Cookie. : Données de cookie mémorisées par le client.
  - Host : Hôte virtuel demandé.
  - If-modified-since: ne retourne le document que s'il a changé depuis la date indiquée
  - If-none-match. : ne retourne le document que s'il a changé.
  - Referer : URL de la page à partir de laquelle le document est demandé.
  - User-agent: nom et versión du logiciel utilisé

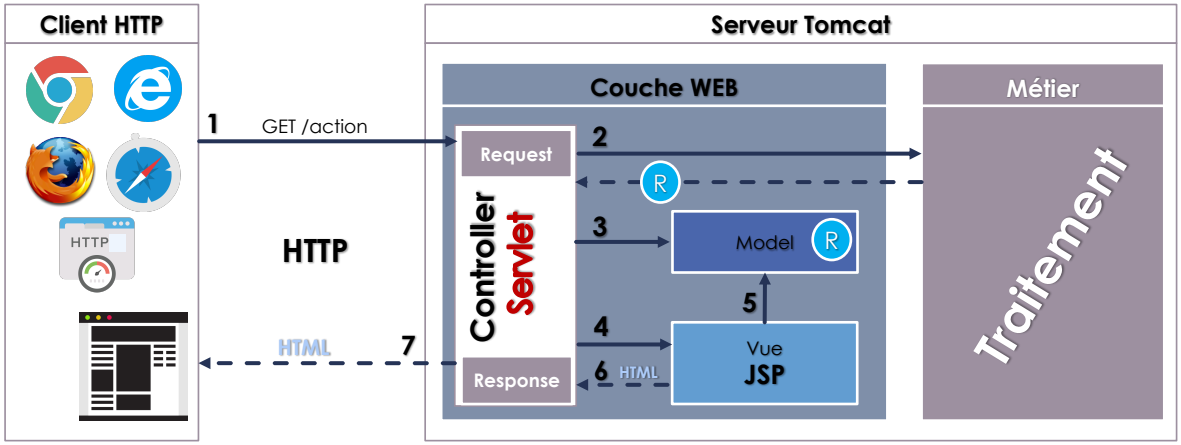
27

Entêtes HTTP

- **Entêtes de la réponse** :
  - Allowed. : Méthodes HTTP autorisées pour cette URI (comme PORT).
  - Date : date et heure courante (d'envoie de la réponse) .
  - Expires: date d'expiration du document (pour le supprimer du cache)
  - Last-modified. : Date du dernière modification.
  - Location : Adresse du document lors d'une redirection.
  - Etag: numéro de version du document
  - Pragma. : données annexes pour le navigateur (i.e. no.cache).
  - Server : nom et version du serveur .
  - Set-cookie: permet au serveur d'écrire un cookie sur le disque du client.

28

Architecture WEB JEE



Architecture WEB JEE

1 Le Client envoie une **requête HTTP** de type **GET** ou **POST** vers le contrôleur représenté par un composant Web JEE appelé **SERVLET** .
- Pour lire les données de la requête, le contrôleur crée un objet de type **HTTPServletRequest**,
- Pour personnaliser la réponse qui va être envoyé après au client, il crée un autre objet de type **HTTPServletResponse**.

2 Le Contrôleur fait appel à la couche métier pour effectuer les traitements et récupère les résultats R

3 Le Contrôleur stocke le résultat R dans le modèle **M**.
Le Modèle est généralement un objet qui permet de représenter toutes les données qui seront affichées dans la vue **V**.
Généralement ce modèle, est stocké par le contrôleur **C** dans l'objet **Request** ou **Session**.

4 Le contrôleur fait appel à la vue JSP (JAVA SERVER PAGE), en lui transmettant les objets **request** et **response**.
Cette opération s'appelle le **Forwarding** ou navigation.

5 La vue JSP récupère le résultat à partir du modèle.
La vue retrouve le modèle à partir de l'objet **Request** ou **Session**

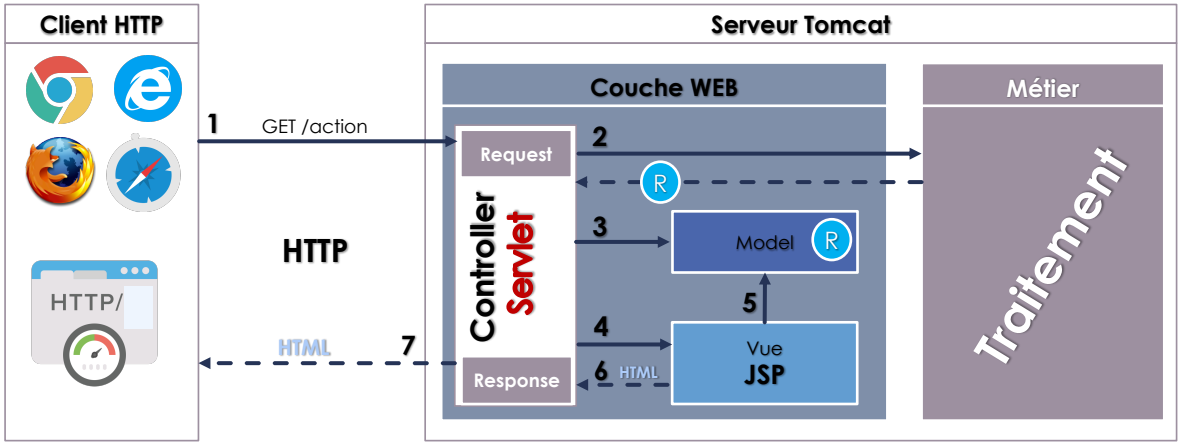
6 La vue JSP génère dynamiquement une page HTML contenant les résultats du modèle.
Qui va être stocké dans l'objet **HTTPServletResponse** (Response)

7 La page HTML générée est envoyée dans le corps de la réponse HTTP du serveur au client HTTP.
Et le browser affiche le rendu de la page HTML reçue.

30

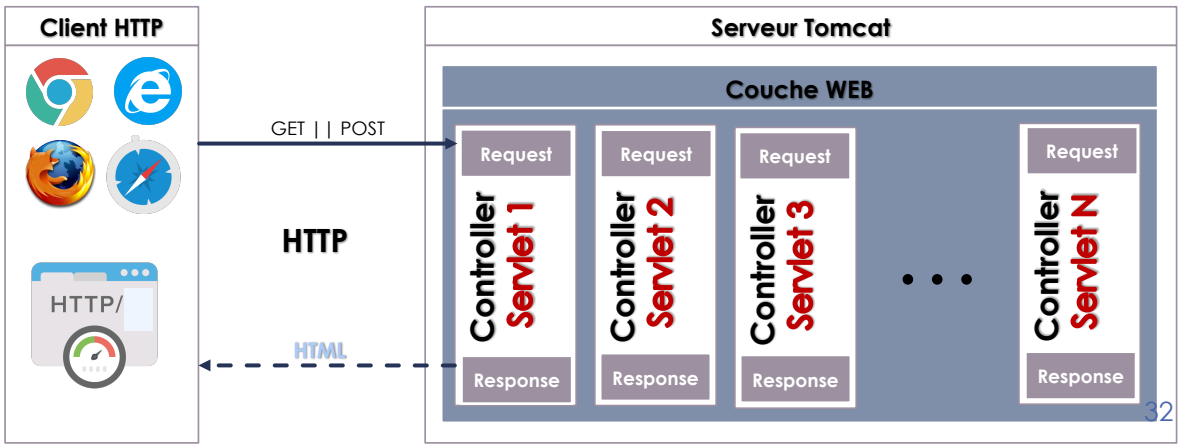
Architecture WEB JEE

En Respectant cette Architecture, on dit qu'on a respecter un patron de conception **MVC**



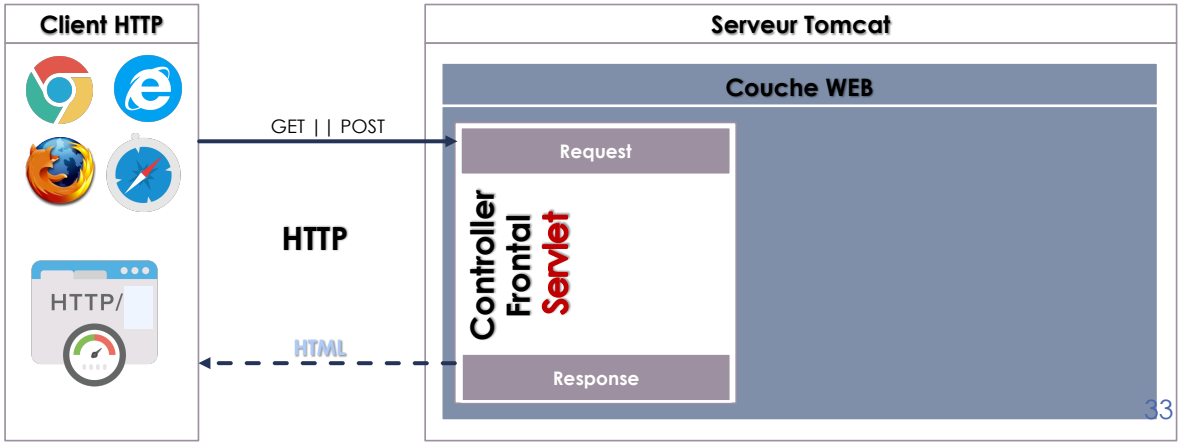
Architecture WEB JEE

On peut avoir plusieurs contrôleur dans notre application JEE, un contrôleur pour chaque besoin ou partie de notre application

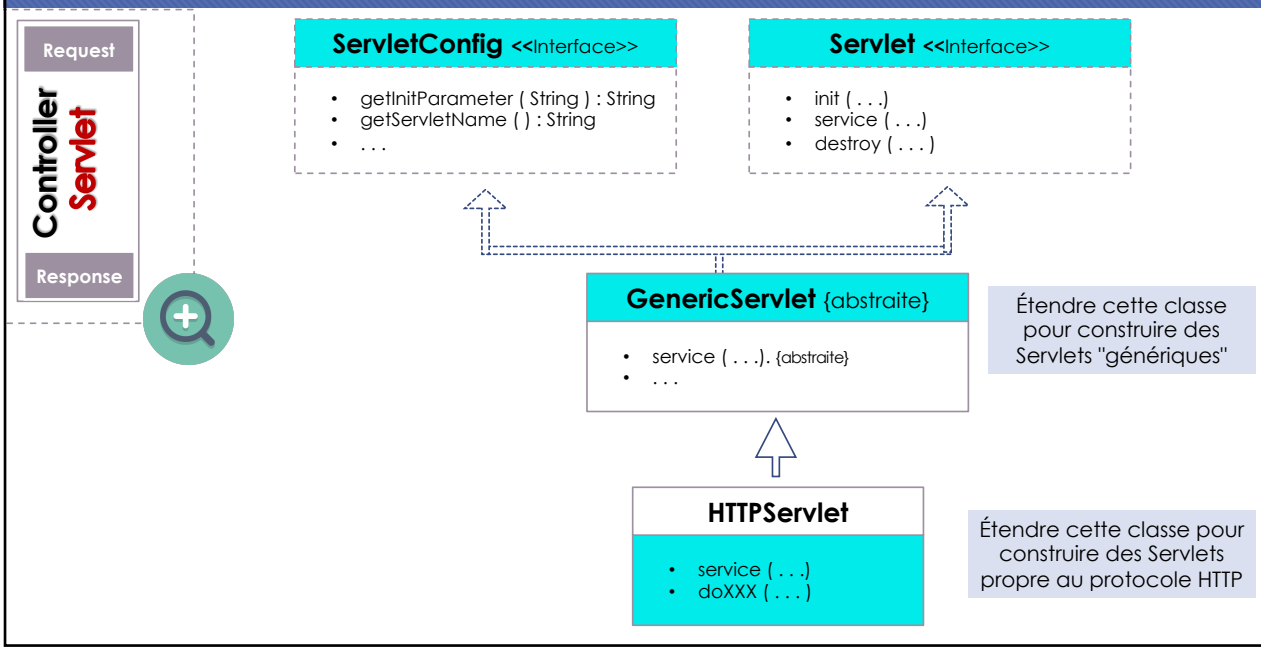


Architecture WEB JEE

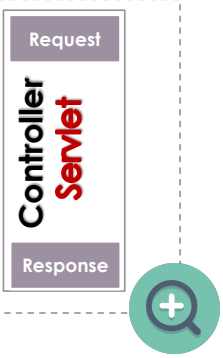
Ou bien un seule contrôleur pour toute l'application JEE.
On dit on utilise **MVC 2**
Ce contrôleur s'appel, le contrôleur **frontale**, et on a besoin de sous contrôleurs qui vont s'occuper de faire des traitement en fonction de chaque type de requête envoyée.



SERVLET



SERVLET



HTTPServlet

- service (. . .)
- doXXX (. . .)

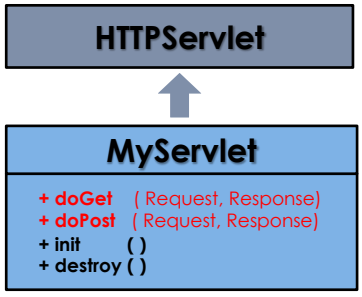
Dans la suite du cours nous allons utiliser uniquement des Servlets qui réagissent au protocole HTTP d'où l'utilisation de la classe **HttpServlet**

Une **Servlet** est concrètement une classe Java qui hérite de la classe **HTTPServlet** qui implémente l'interface **Servlet**, et bien sur qui peut redéfinir les méthodes correspondants aux méthode HTTP : **doGet**(), **doPost**(), **doPut**(), **doDelete**() et **doHead**(). Et d'autres méthodes qui définissent son cycle de vie.

35

Structure d'une SERVLET

```
package controllers;

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

public class MyServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;


        @Override
        public void init() throws ServletException {

        // Initialisation
        // Exécutée juste après instantiation de la servlet par le serveur Tomcat
        }

        @Override
        protected void doGet(HttpServletRequest request,
                             HttpServletResponse response)
                                    throws ServletException, IOException {
        // Traitement effectué si une requête HTTP est envoyée avec la méthode GET
        }

        @Override
        protected void doPost(HttpServletRequest request,
                              HttpServletResponse response)
                                    throws ServletException, IOException {

        // Traitement effectué si une requête HTTP est envoyée avec la méthode POST
        }

        @Override
        public void destroy() {

        // Exécutée juste avant la destruction de la servlet.
        // Au moment de l'arrêt de l'application
        }

   }
```



36

Cycle de vie d'une SERVLET

1. Au démarrage de l'application : C.-à-d. qu'on vous déployer l'application dans le serveur => il cherche et lit le fichier Web.xml « **le descripteur de déploiement de servlet** »

2. Dès que le serveur lit le fichier web.xml, il récupère les information concernant la servlet à charger et instancier

3. Pour qu'une classe java soit exécutable, elle doit être chargée. Lorsqu'une Servlet est chargée, il n'existe qu'une seule instance de la Servlet durant son cycle de vie.

4. Une fois que la Servlet est chargée, le conteneur crée **une seule instance** de la Servlet. Le container exécute cette étape une et une seule fois pour une Servlet donnée.

5. Une fois la servlet est instanciée, une initialisation est réalisée, la méthode **init**() est appelée par le serveur pour exécuter le code des traitements à effectuer lors de l'initialisation.

6. Maintenant que la servlet est initialisée et chargée en mémoire : elle attend des requêtes HTTP à partir du client.

7. Quand un client envoie une requête HTTP avec GET par exemple, cette requête est reçu par le serveur Tomcat qui joue le rôle de conteneur web, il fait appel à la méthode **service()** de la servlet.

37

Cycle de vie d'une SERVLET

8. La méthode **Service**() est la méthode principale qui exécute la logique métier de la servlet et c'est elle qui est aussi responsable du traitement des requêtes des clients. Quand une nouvelle requête arrive, le conteneur génère un nouveau Thread; ce thread exécute la méthode **service**() puis la logique métier de l'application ou la tâche prévue de la Servlet.

9. La méthode **Service**() est une méthode héritée de la classe **HTTPServlet**, qui prend deux arguments de type HTTPServletRequest et HTTPServletResponse.

10. Une requête HTTP peut être de différents type comme **GET**, **POST**, **PUT**, **DELETE**, etc; le rôle de la méthode **service**() de voir la requête HTTP requise et de faire appel à la méthode adéquate qui peut être **doGet**, **doPost**, **doPut**, **doDelete**, etc .

11. Si la requête HTTP envoyé par le client est de type GET, la méthode doGet() de la servlet est exécuté, en exécutant les traitement souhaitées par la requête et avec un objet HTTPServletResponse elle renvoie une réponse HTTP qui va être renvoyée vers le client.

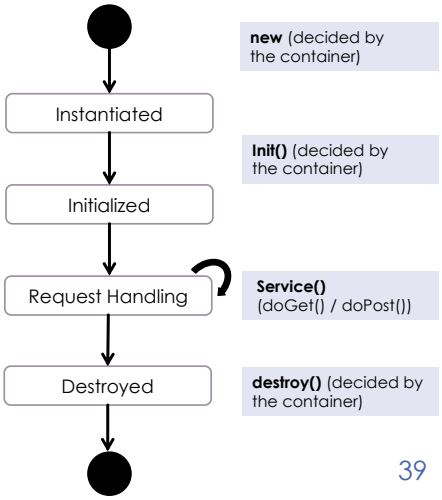
12. La méthode **destroy()** est appelée à la fin du cycle de vie de la Servlet; elle permet aussi, si elle est invoquée, de fermer tous les fichiers ou bases de données ouverts par la Servlet lorsqu'elle était chargée par le conteneur.

38

Cycle de vie d'une SERVLET

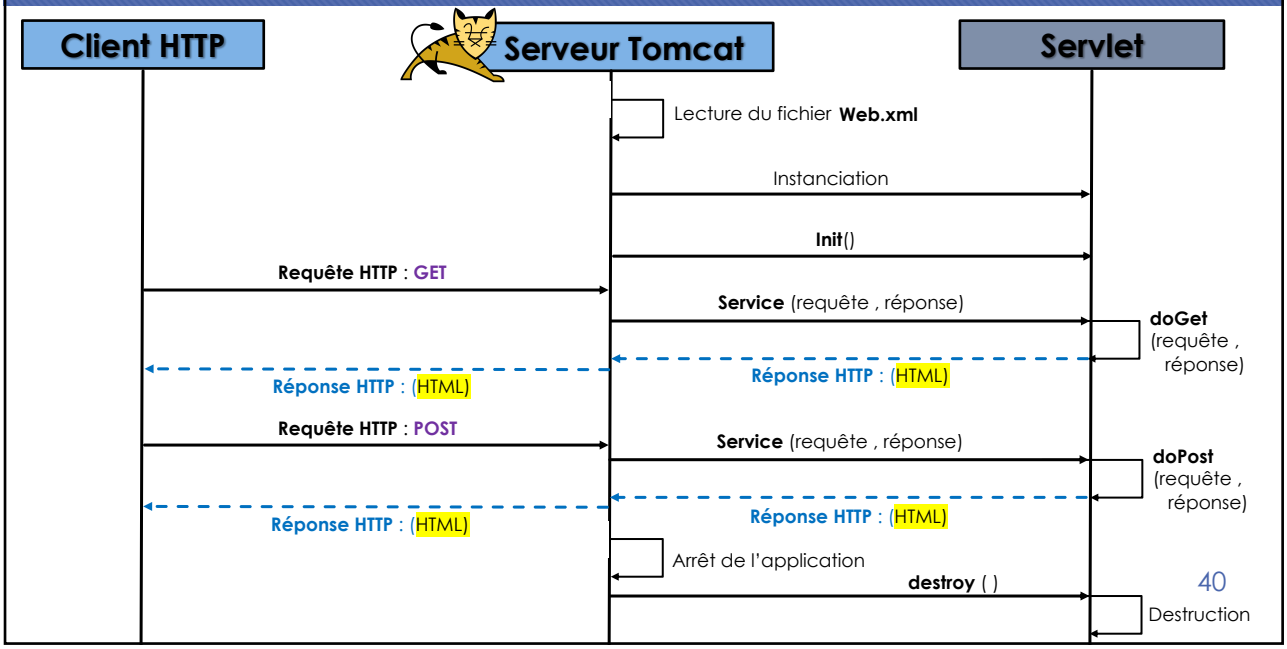
13. Une Servlet est déchargée par le conteneur dans les deux cas suivants :
    - Si le conteneur s'arrête,
    - Si le conteneur recharge l'environnement d'exécution de l'application web

Après l'appel de la méthode **destroy()**, la Servlet es envoyée au ramasse-miette.



39

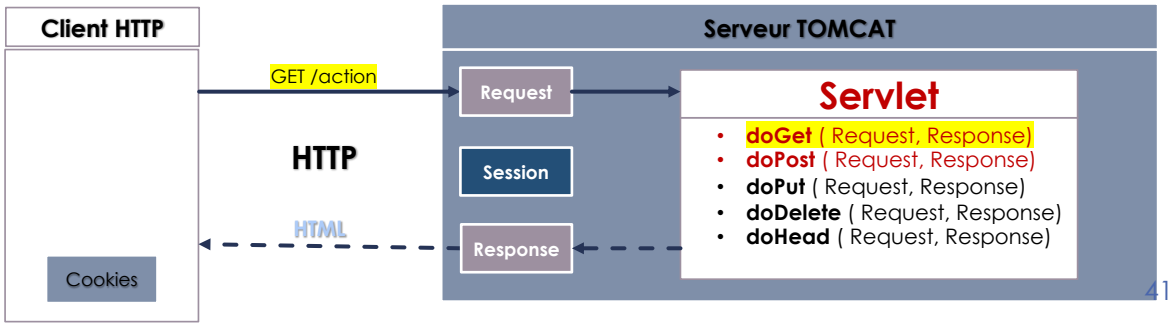
Cycle de vie d'une SERVLET



40

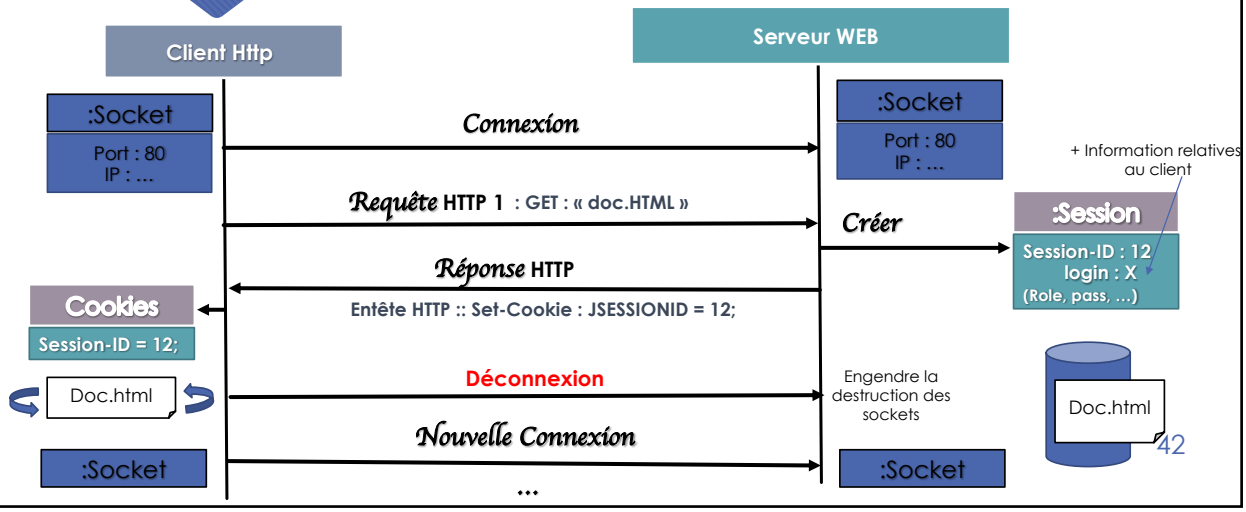
SERVLET (Sessions et cookies)

- Pour lire les données de la requête, la servlet utilise l'objet **REQUEST**.
- Pour envoyer la réponse HTTP au client, la servlet utilise l'objet **RESPONSE**.
- Pour chaque nouveau client HTTP, le serveur crée un objet **Session** qui permet de stocker les données relatives au client dans la mémoire du serveur.
- Une session possède un timeout (20 min) au bout duquel, si le client n'envoie pas de requête HTTP, la session est détruite.
- Dans une réponse HTTP, le serveur peut demander au client HTTP d'enregistrer des données relatives au client, dans des fichiers de la machine du client. Ce qu'on appelle des Cookies.
- Les cookies ont également une durée de vie, quand cette durée expire, ils sont détruit par le navigateur WEB.



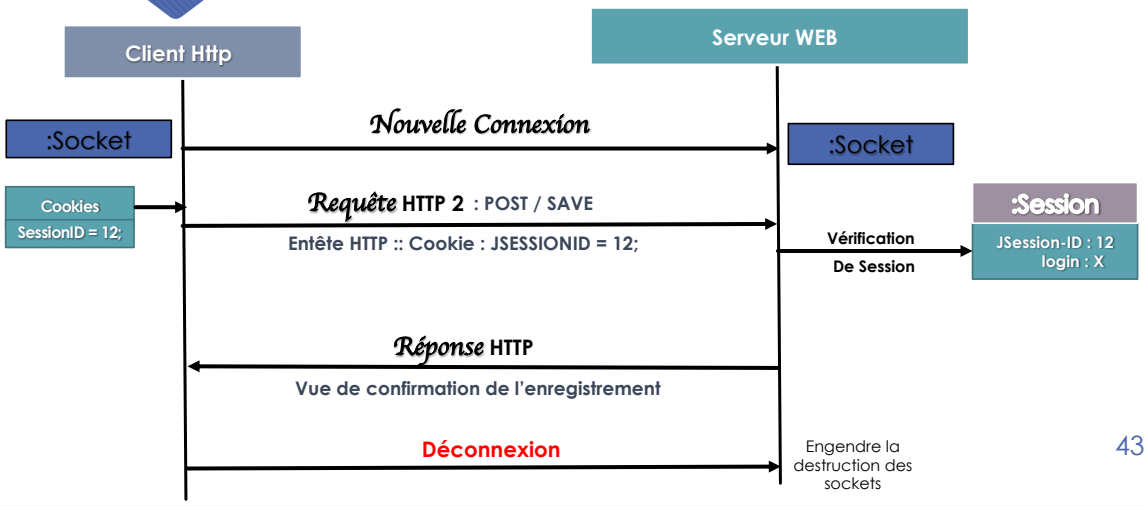
41

Session et Cookies



42

Session et Cookies

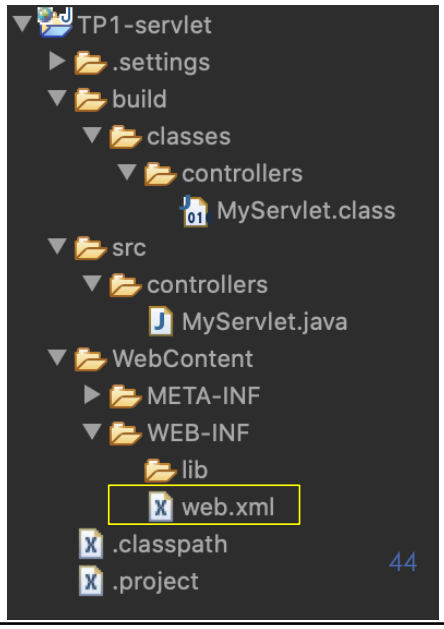


43

Déploiement d'une SERVLET

Structure d'un projet Web JEE

- Le dossier **src** contient les classes java.
- Le byte code est placé dans le dossier **build/classes**
  - ✓ fichiers.class
- Les dossier **WebContent** contient les documents Web :
  - ✓ comme les pages HTML, JSP, Images, JavaScript, CSS …
- Le dossier **WEB-INF** contient les descripteurs de déploiement
  - ✓ web.xml. (utiliser pour l'enregistrement des Servlet)
- Le dossier **lib** permet de stocker les bibliothèques de classes java
  - ✓ Fichiers.jar



44

Déploiement d'une SERVLET

Web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
         http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
         id="WebApp_ID" version="4.0">

  <display-name>Test-Servlet</display-name>

  <servlet>
            <servlet-name>MyS</servlet-name>
            <servlet-class>controllers.MyServlet</servlet-class>

  </servlet>


  <servlet-mapping>

            <servlet-name>MyS</servlet-name>
            <url-pattern>*.do</url-pattern>

  </servlet-mapping>


</web-app>
```

Balise de description de l'application WEB

Enregistrement de la servlet :
- Identification (nom de la servlet)
- Class de la servlet

Mappage de la servlet :
- Identification (nom de la servlet)
- modèle de l'URL associé à la servlet

45

Déploiement d'une SERVLET

Web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
         http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
         id="WebApp_ID" version="4.0">

  <display-name>Test-Servlet</display-name>

 <servlet>
          <servlet-name>MyS</servlet-name>
          <servlet-class>controllers.MyServlet</servlet-class>
          <init-param>
                      <param-name>Author</param-name>
                      <param-value>Omar</param-value>
          </init-param>

          <load-on-startup>0</load-on-startup>
 </servlet>
```

Balise de description de l'application WEB

Enregistrement de la servlet :
- Identification (nom de la servlet)
- Class de la servlet

+ Paramètres de description
+ Paramétrer la servlet à être charger dès le chargement de l'application, avec une priorité de chargement.

46

Déploiement d'une SERVLET

Servlet 3.0

Pour un projet web JEE, utilisant un module web, version 3.0 ou plus, le fichier Web.xml n'est plus nécessaire.

Dans ce cas, le déploiement d'une servlet peut se faire en utilisant des annotations @WebServlet

```
package controllers;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.*;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;


@WebServlet(name="MyS",urlPatterns={"/","*.do"})
public class MyServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;




                                                }
```

Ou bien seulement : @WebServlet("*.do")

47

Exemple d'une SERVLET

http://localhost:8088/TP1-servlet/*.do?nom=MyServlet

http://localhost:8088/TP1-servlet/*.do?nom=MyServlet

- Protocol : HTTP/1.1
- ServerName : localhost
- ServerPort : 8088
- RemoteAddr : 0:0:0:0:0:0:0:1
- RemoteHost : 0:0:0:0:0:0:0:1
- Method : GET
- Paramètre URL nom : MyServlet

```
package controllers;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.*;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;


@WebServlet(name="MyS",urlPatterns={"/","*.do"})
public class MyServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    @Override
    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response)
                                 throws ServletException, IOException {

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

                out.println("<html>");
                out.println("   <body>");
                out.println("       <ul>");
                out.println("           <li> Protocol   : " + request.getProtocol()     + "</li>");
                out.println("           <li> ServerName : " + request.getServerName()   + "</li>");
                out.println("           <li> ServerPort : " + request.getServerPort()   + "</li>");
                out.println("           <li> RemoteAddr : " + request.getRemoteAddr()   + "</li>");
                out.println("           <li> RemoteHost : " + request.getRemoteHost()   + "</li>");
                out.println("           <li> Method     : " + request.getMethod()       + "</li>");
                out.println("           <li> Paramètre URL nom  : " + request.getParameter("nom") + "</li>");
                out.println("       </ul>");
                out.println("   </body>");
                out.println("</html>");
```

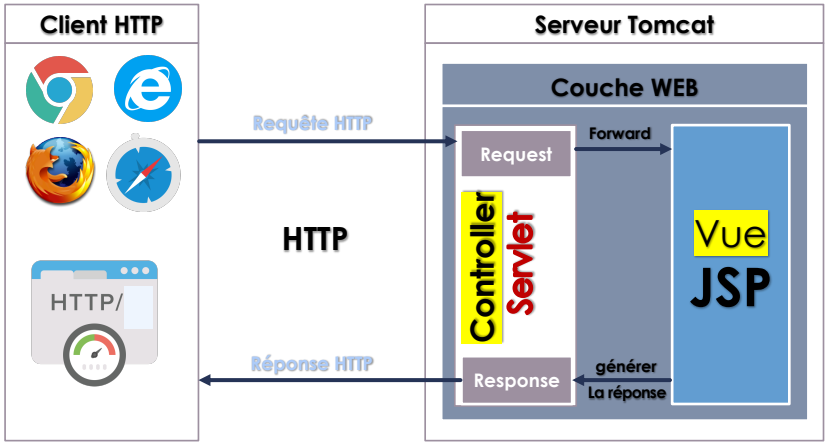
48

```
                                                }
```

Forward SERVLET > JSP

Pour séparer les rôles une servlet peut faire un **forward** vers une JSP de la manière suivante :

```
request.getRequestDispatcher("reponse.jsp").forward(request, response);
```



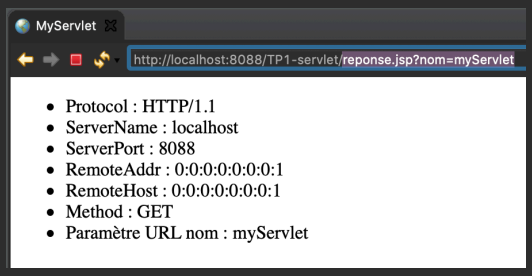
49

Forward SERVLET > JSP

On peut commence par créer la vue **reponse.jsp** comme suit :

```
<%
    String nom          = request.getParameter("nom");
    String protocol     = request.getProtocol();
    String serverName   = request.getServerName();
    int    serverPort   = request.getServerPort();
    String remoteAddr   = request.getRemoteAddr();
    String remoteHost   = request.getRemoteHost();
    String method       = request.getMethod();
%>
<!DOCTYPE html>
<html>
<head> <title>MyServlet</title> </head>
<body>
        <ul>
            <li> Protocol   : <%=protocol    %> </li>
            <li> ServerName : <%=serverName  %> </li>
            <li> ServerPort : <%=serverPort  %> </li>
            <li> RemoteAddr : <%=remoteAddr  %> </li>
            <li> RemoteHost : <%=remoteHost  %> </li>
            <li> Method     : <%=method      %> </li>
            <li> Paramètre URL nom  : <%=nom %></li>
        </ul>
</body>
</html>
```

On peut y accéder via url comme suit :



50

Forward SERVLET > JSP

```
package controllers;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.*;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;


@WebServlet(name="MyS",urlPatterns={"/","*.do"})
public class MyServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    @Override
    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response)
                               throws ServletException, IOException {
```

On peut y accéder via notre **Servlet** en faisant un **forward** vers notre **vue** qui va s'occuper de **l'affichage**.

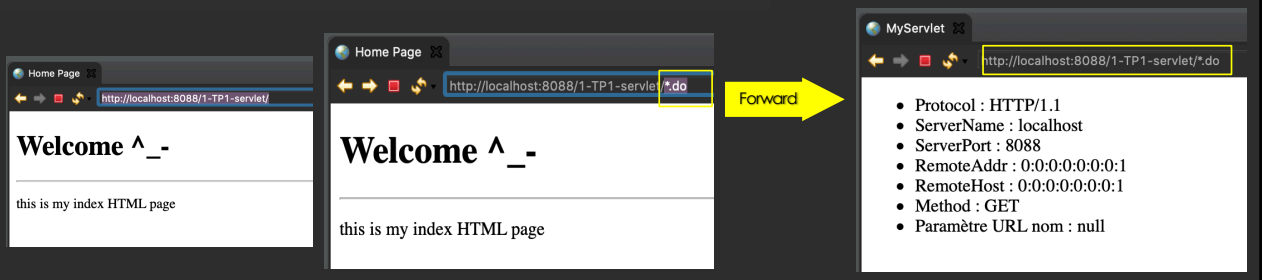
```
request.getRequestDispatcher("reponse.jsp").forward(request, response);
```

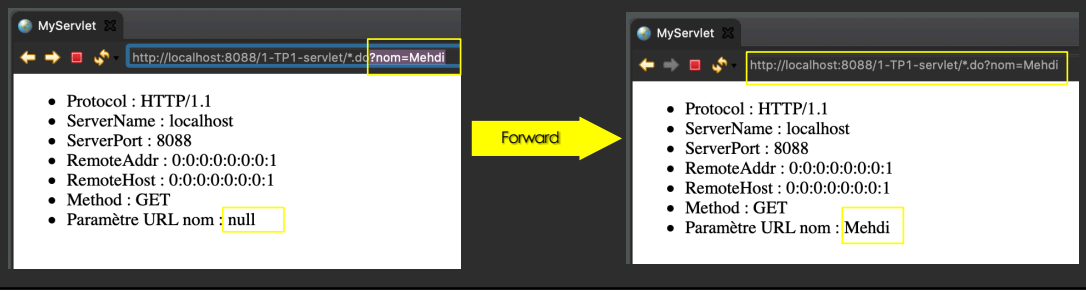
51

```
}
```

Forward SERVLET > JSP

```
request.getRequestDispatcher("reponse.jsp").forward(request, response);
```



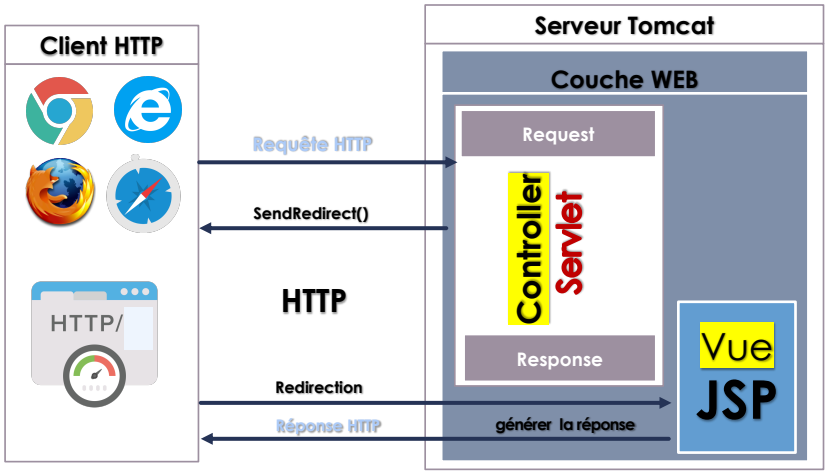


52

Redirection SERVLET > JSP

Une servlet peut rediriger vers une autre ressource locale ou distante en utilisant la méthode **sendRedirect**() de l'objet **response**.

```
response.sendRedirect("reponse.jsp");
```



53

Redirection SERVLET > JSP

Rediriger la réponse vers la page JSP à partir de la servlet via :

MyServlet
http://localhost:8088/1-%20TP1-servlet/controller.do

- Protocol : HTTP/1.1
- ServerName : localhost
- ServerPort : 8088
- RemoteAddr : 0:0:0:0:0:0:0:1
- RemoteHost : 0:0:0:0:0:0:0:1
- Method : GET
- Paramètre URL nom : null

L'url change vers la ressource

MyServlet
http://localhost:8088/1-%20TP1-servlet/reponse.jsp

- Protocol : HTTP/1.1
- ServerName : localhost
- ServerPort : 8088
- RemoteAddr : 0:0:0:0:0:0:0:1
- RemoteHost : 0:0:0:0:0:0:0:1
- Method : GET
- Paramètre URL nom : null

La redirection ne transporte pas les paramètres de la requête:

MyServlet
http://localhost:8088/1-%20TP1-servlet/controller.do?nom=Mehdi

- Protocol : HTTP/1.1
- ServerName : localhost
- ServerPort : 8088
- RemoteAddr : 0:0:0:0:0:0:0:1
- RemoteHost : 0:0:0:0:0:0:0:1
- Method : GET
- Paramètre URL nom : null

Redirection

MyServlet
http://localhost:8088/1-%20TP1-servlet/reponse.jsp

- Protocol : HTTP/1.1
- ServerName : localhost
- ServerPort : 8088
- RemoteAddr : 0:0:0:0:0:0:0:1
- RemoteHost : 0:0:0:0:0:0:0:1
- Method : GET
- Paramètre URL nom : null

54

Redirection SERVLET > JSP

Pour garder les paramètres de la requête lors d'une redirection il faut les ajoutés dans l'url :

```
response.sendRedirect("reponse.jsp?nom=Mehdi");
```

55

SERVLET VS JSP

- Une servlet est une classe java dans laquelle on peut générer du code HTML
- Une JSP est une sorte de page HTML à l'intérieur de laquelle, o peut écrire du code Java.

- Les pages **JSP** sont très pratique quand on veut afficher les vues de l'applications
- Les **servlets** sont pratique pour effectuer les traitements nécessaires au fonctionnement d'une application.

- Une **servlet** nécessite d'être déployée (web.xml),
- Alors qu'une **JSP** est déployée automatiquement par Tomcat.

- Pendant le premier appel d'une **JSP**, Tomcat convertit la **JSP** en **servlet** et la déploie automatiquement.
- Quand un client HTTP demande une **page JSP**, c'est la **servlet** correspondante à cette **JSP**, qui est générée par Tomcat qui est exécutée.

- Tout ce qu'on peut faire avec une **servlet**, peut être fait par une **JSP** (Une JSP est une servlet)
- La technologie de base des applications web JEE c'est les **servlets**.

56

SERVLET VS JSP

- Une **JSP** est *intimement* liée à une **servlet**. En fait, une fois la compilation effectuée, la **JSP** devient une **servlet**.
  Donc les objets **request, session, out, …etc.** reste accessibles dans une page JSP.

- Dans une application web JEE qui respecte le pattern MVC :
  - ✓ Les **servlets** sont utilisées pour jouer le rôle du **contrôleur**
  - ✓ Les **JSP** sont utilisées pour jouer le rôle des **vues**

57