



FORMATION
JEE AVANCÉE

Formateur

 **Pr. O. EL MIDAoui**

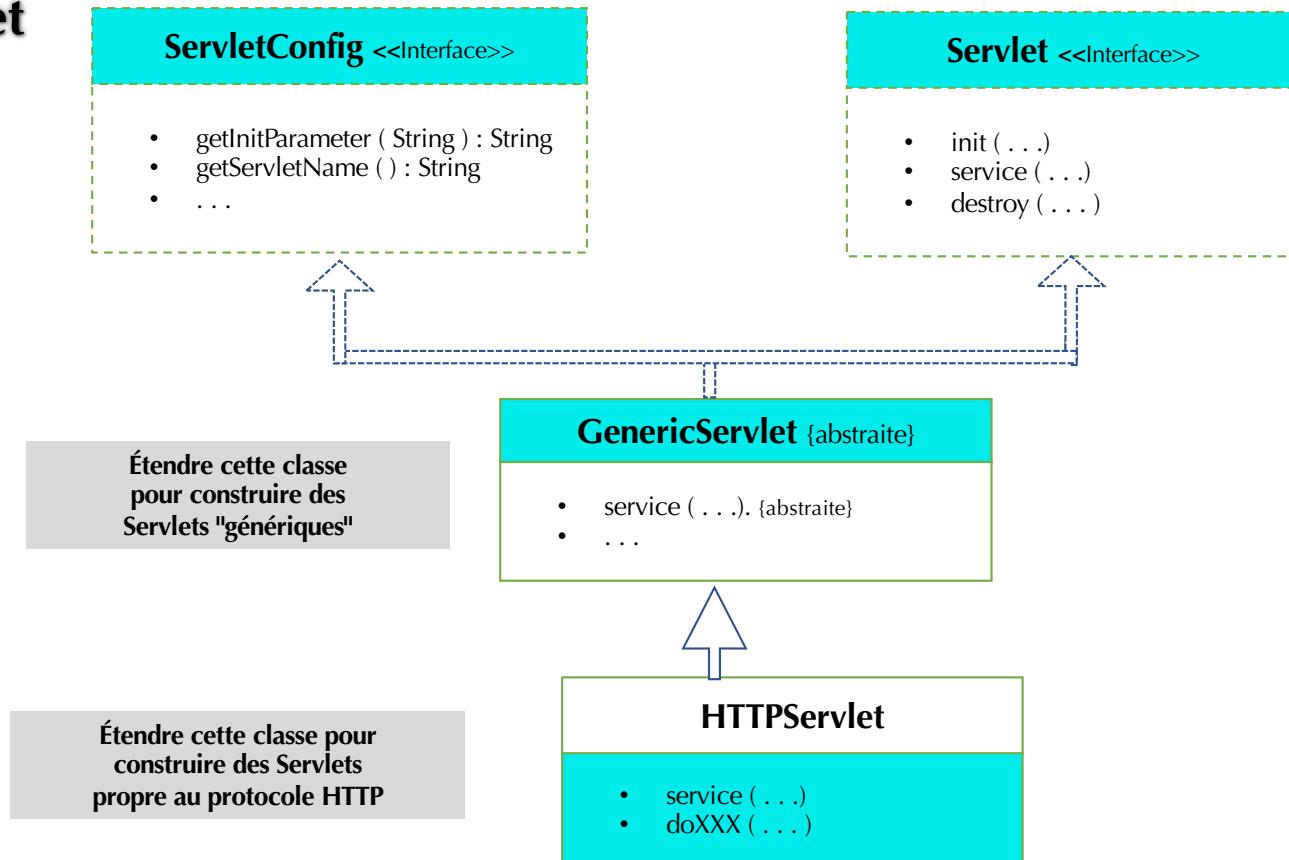
Professor & Senior JAVA Software Engineer

PhD in Data Science - Web Geographic Information Retrieval GIR

Plateforme
Java EE  L'API Servlet

Modèle MVC : Contrôleur [SERVLET]

La classe : HttpServlet



Modèle MVC : Contrôleur [SERVLET]

La classe : **HttpServlet**

HTTPServlet

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

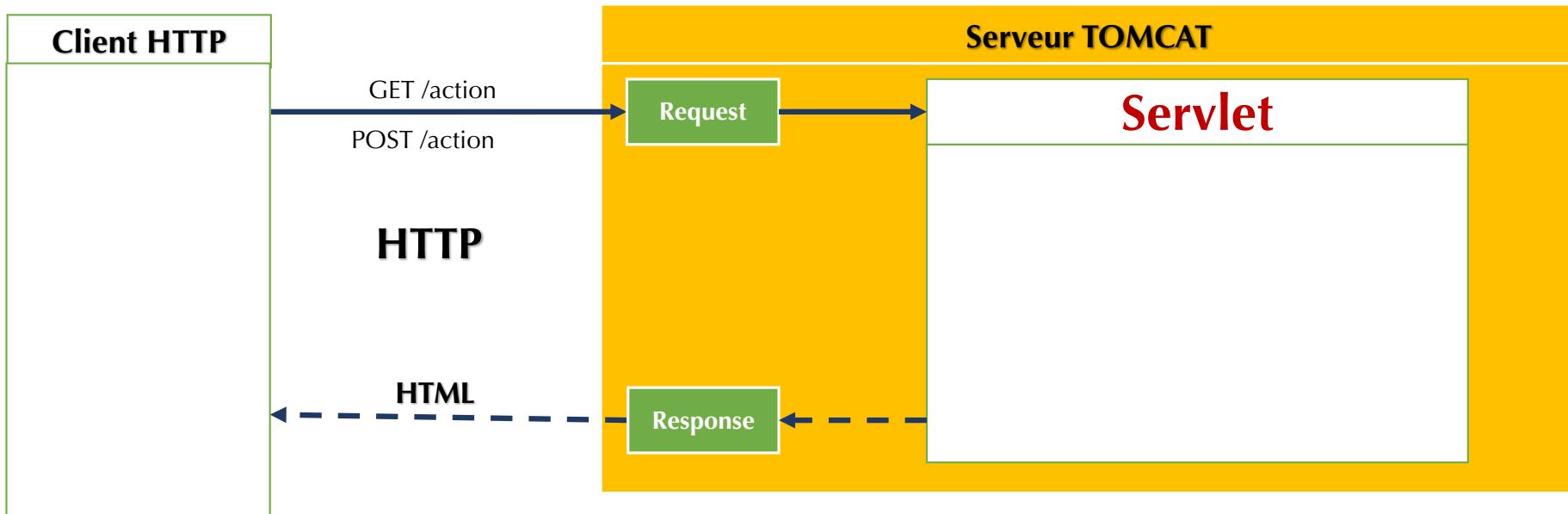
Dans la suite du cours nous allons utiliser uniquement des Servlets qui réagissent au protocole HTTP d'où l'utilisation de la classe **HttpServlet**



Modèle MVC : Contrôleur [SERVLET]

La classe : HttpServlet

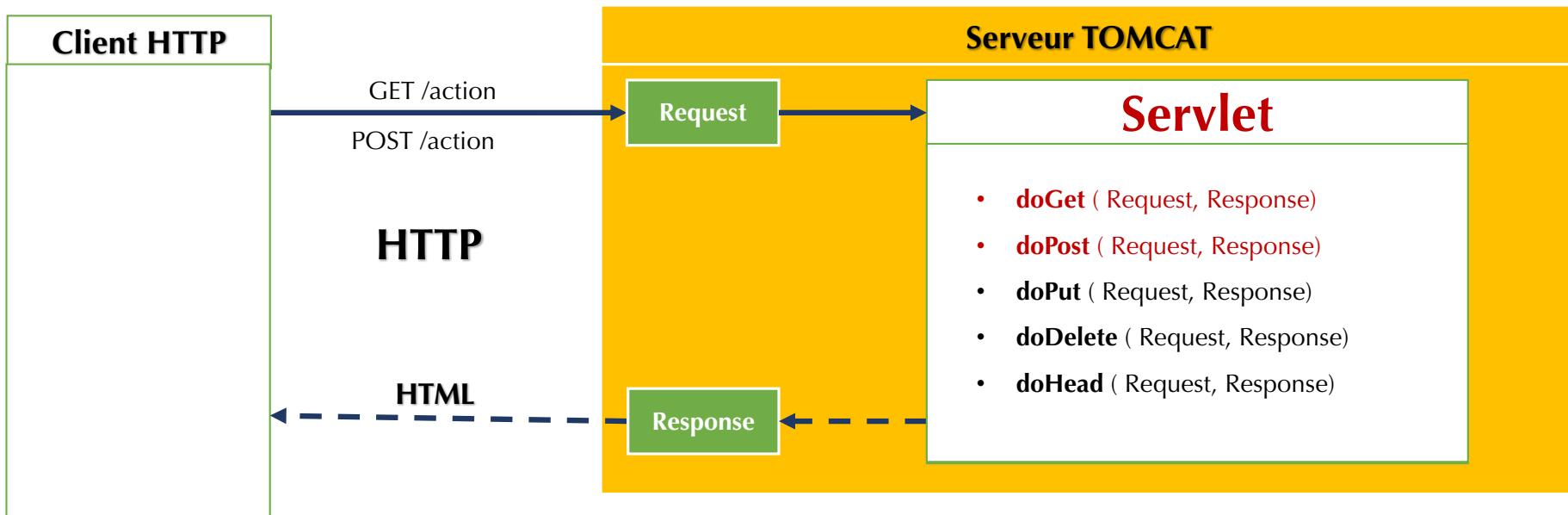
- Une **Servlet** est un composant Web JEE qui permet d'effectuer des traitement de **routage** ou d'aiguillage d'information côté serveur suite à une **requête HTTP**, et envoyer une **réponse HTTP**.



Modèle MVC : Contrôleur [SERVLET]

La classe : HttpServlet

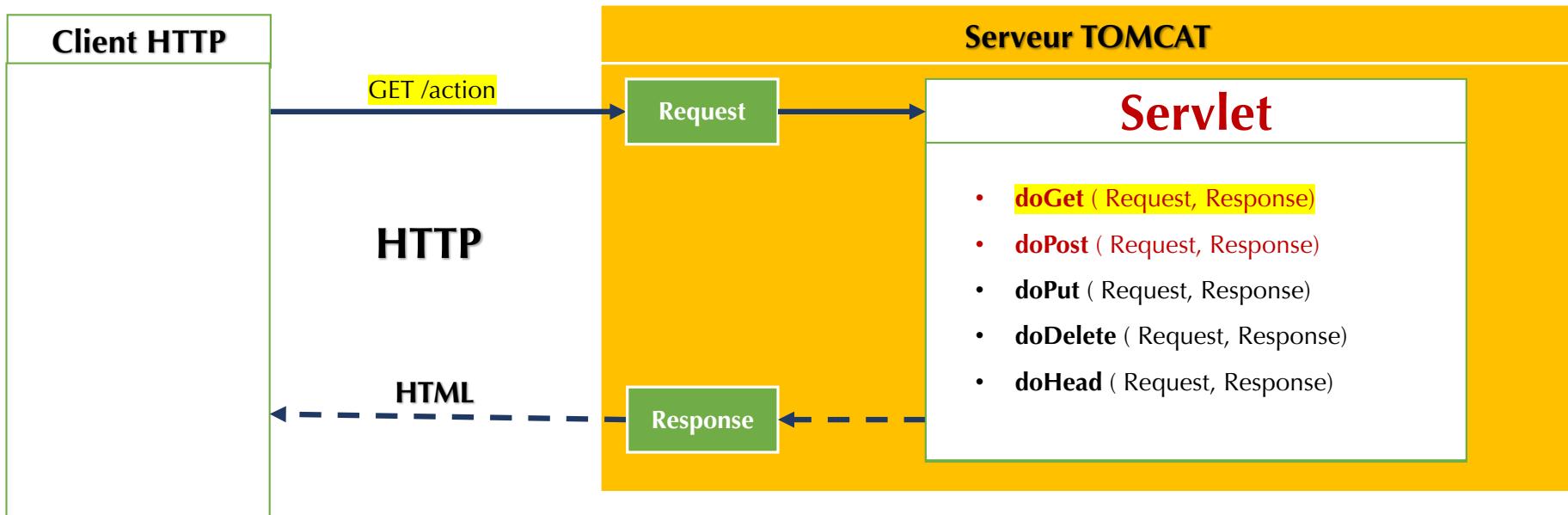
- Une **Servlet** est concrètement une classe Java qui hérite de la classe **HTTPServlet** qui implémente l'interface **Servlet**, et bien sur qui peut redéfinir les méthodes correspondants aux méthode HTTP : **doGet()**, **doPost()**, **doPut()**, **doDelete()** et **doHead()**. Et d'autres méthodes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



Modèle MVC : Contrôleur [SERVLET]

La classe : HttpServlet

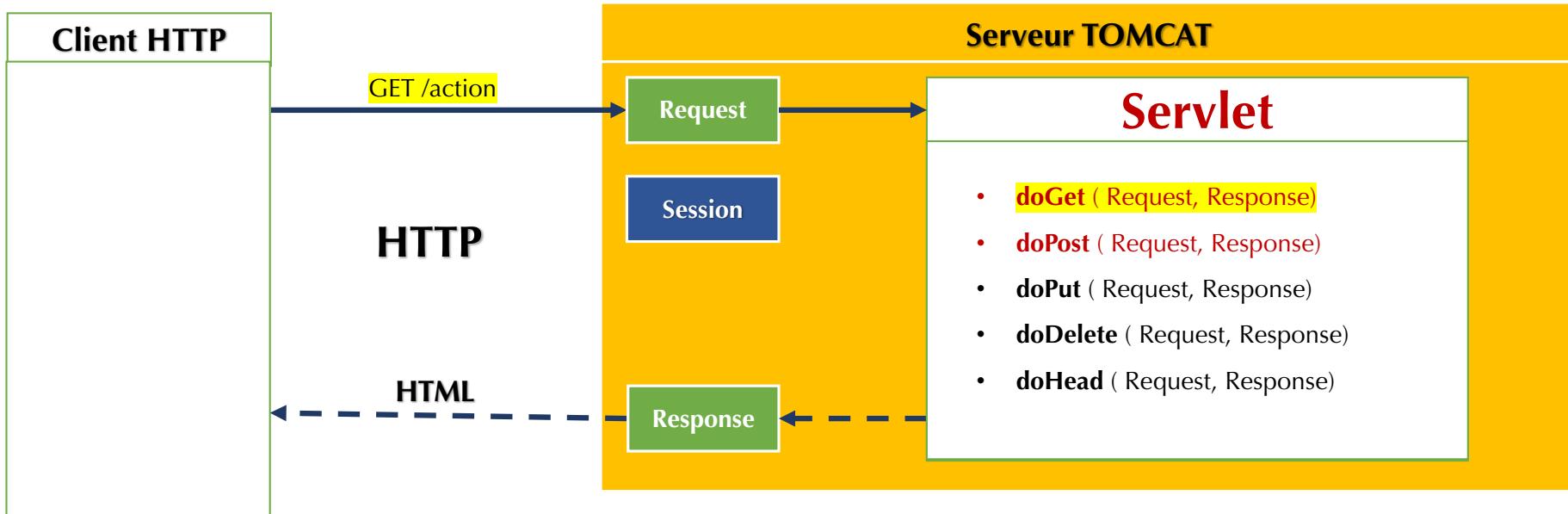
- La méthode **doXXX()** est appelé si une requête HTTP est envoyé par un client HTTP vers le serveur avec la méthode **XXX**.



Modèle MVC : Contrôleur [SERVLET]

La classe : HttpServlet

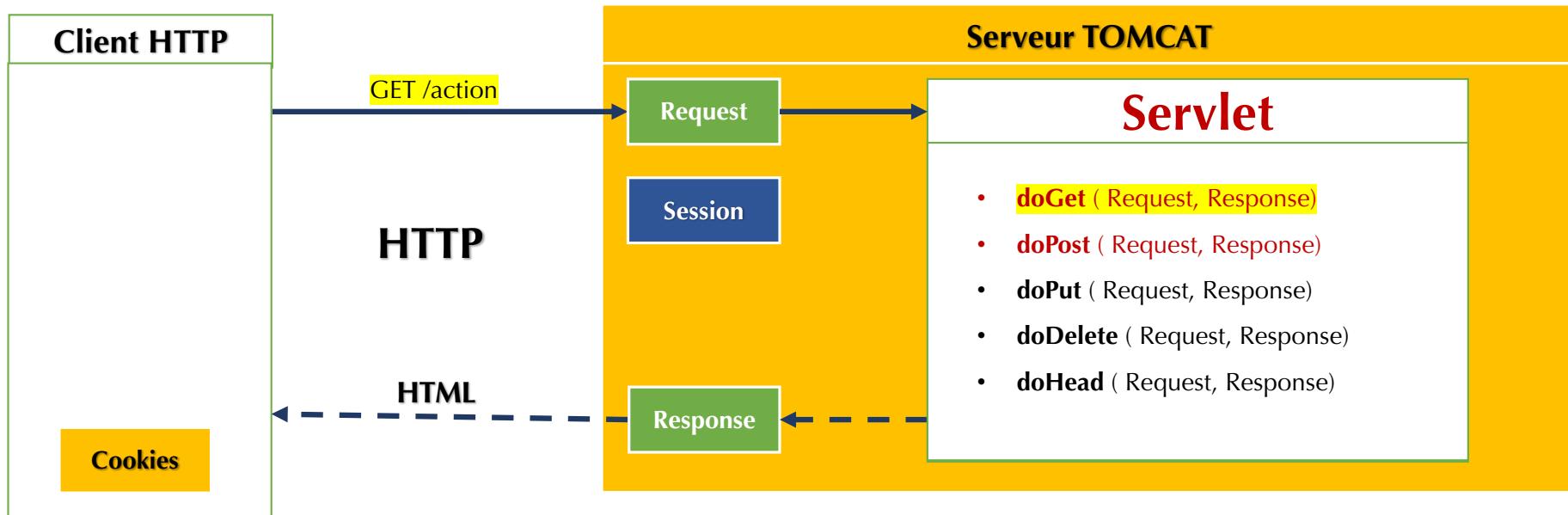
- Pour chaque Client HTTP qui va solliciter le serveur, le serveur va utiliser un objet Session pour enregistrer les données des requêtes envoyées par un Client HTTP donné.
- À chaque fois qu'un client va s'authentifier au serveur, dans la RAM du serveur on va lui créer un espace mémoire nommé session.
- Chaque session va correspondre un Client différent.



Modèle MVC : Contrôleur [SERVLET]

La classe : HttpServlet

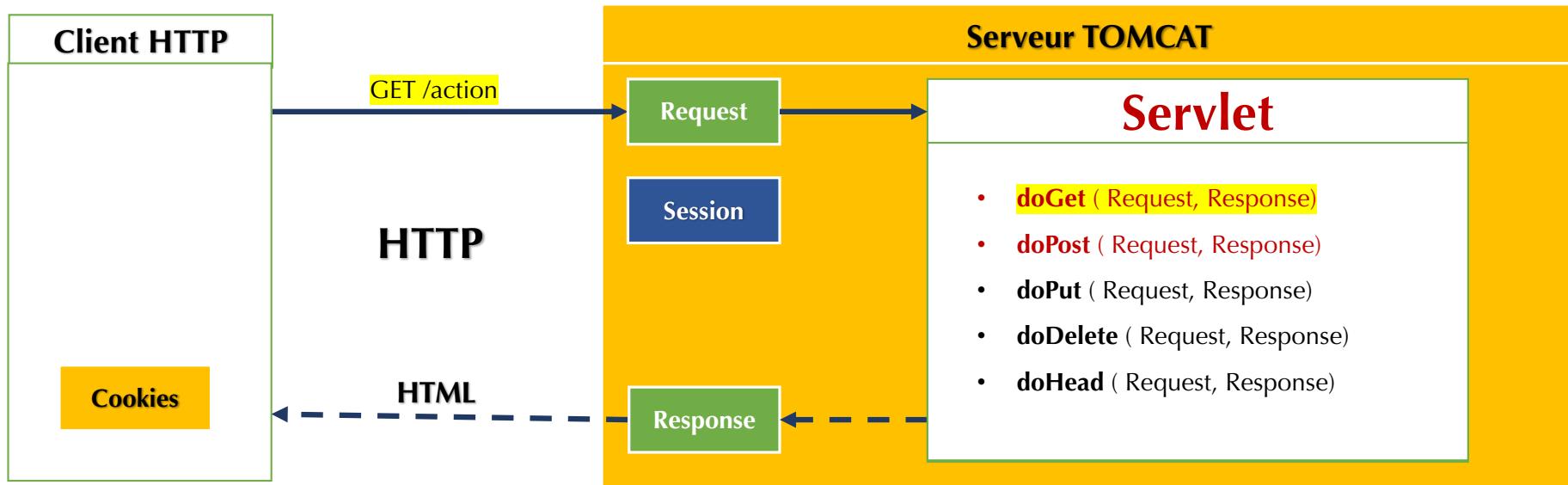
- Et ensuite le serveur aussi à le droit de demander au client de sauvegarder des données dans sa propre machine sous forme de ce qu'on appelle les **cookies**.
- Les cookies ont une durée de vie, quand cette durée expire, ils sont détruits par le navigateur WEB.



Modèle MVC : Contrôleur [SERVLET]

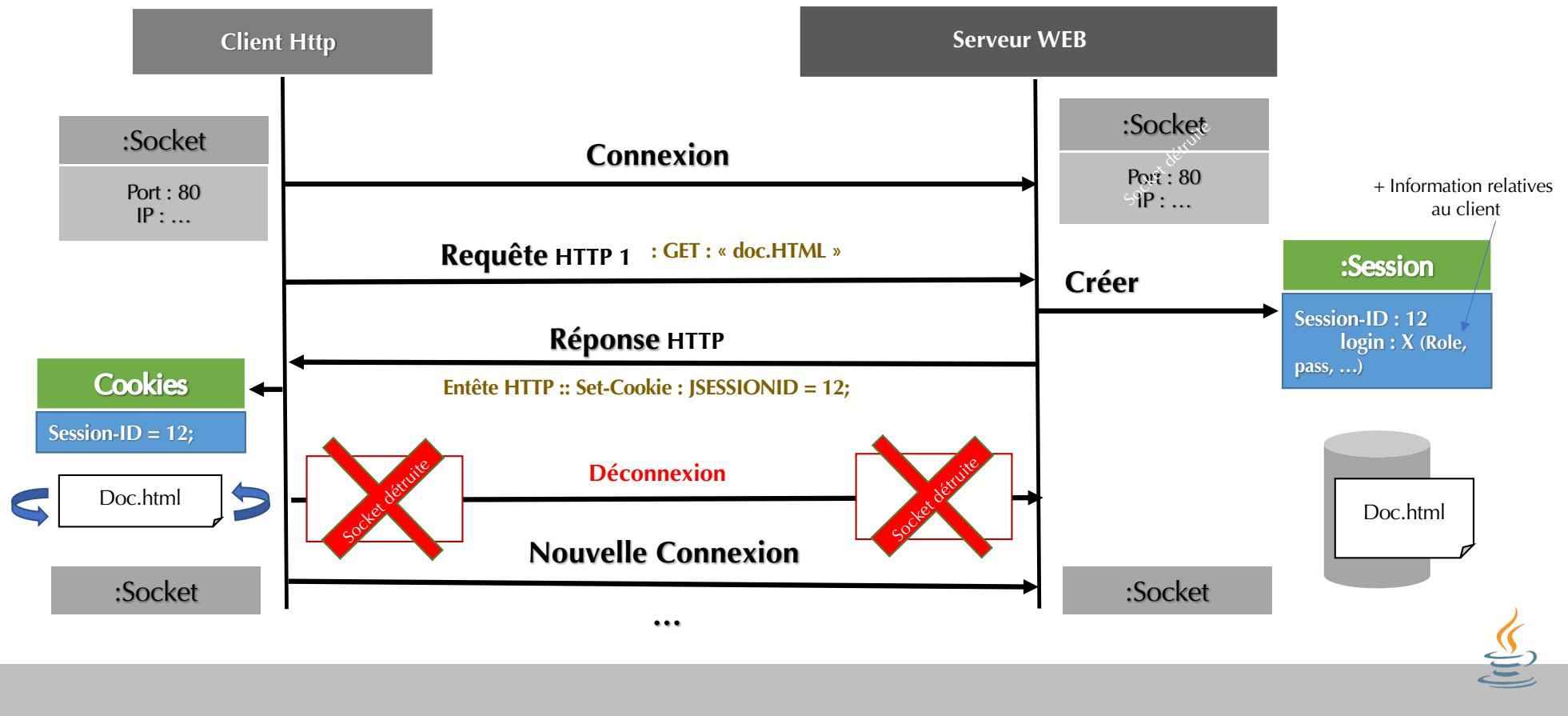
La classe : HttpServlet

- Pour lire les données de la requête, la servlet utilise l'objet **REQUEST** de type **HttpServletRequest**.
- Pour envoyer la réponse HTTP au client, la servlet utilise l'objet **RESPONSE** de type **HttpServletResponse**.
- 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.



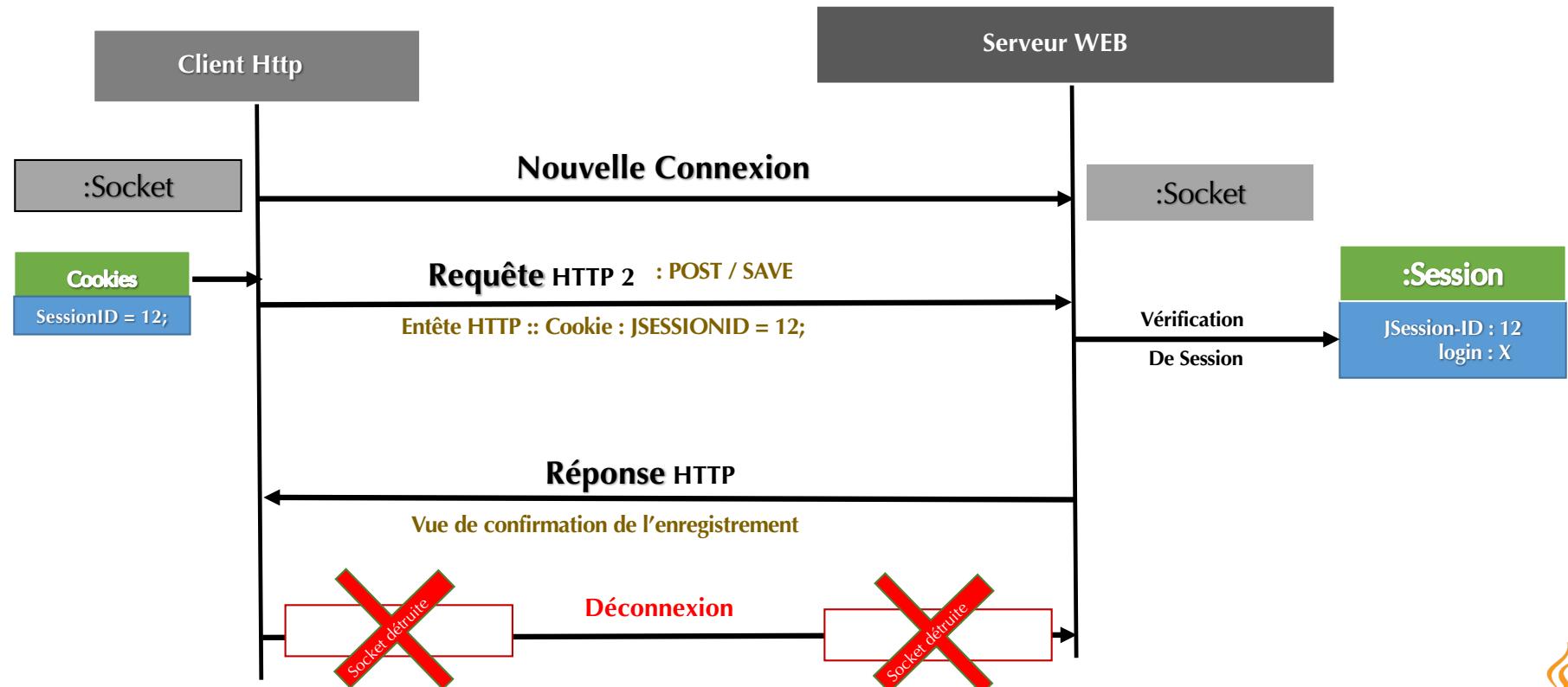
Modèle MVC : Contrôleur [SERVLET]

Session & Cookies



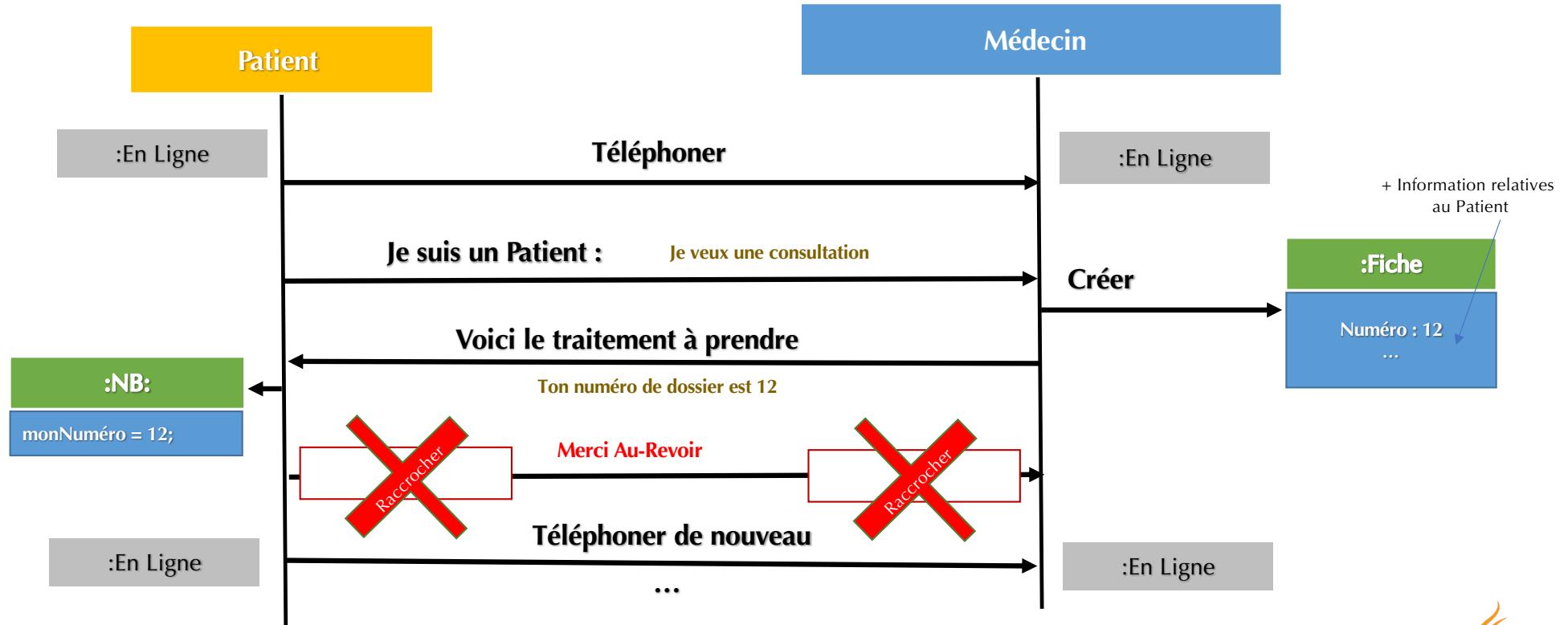
Modèle MVC : Contrôleur [SERVLET]

Session & Cookies



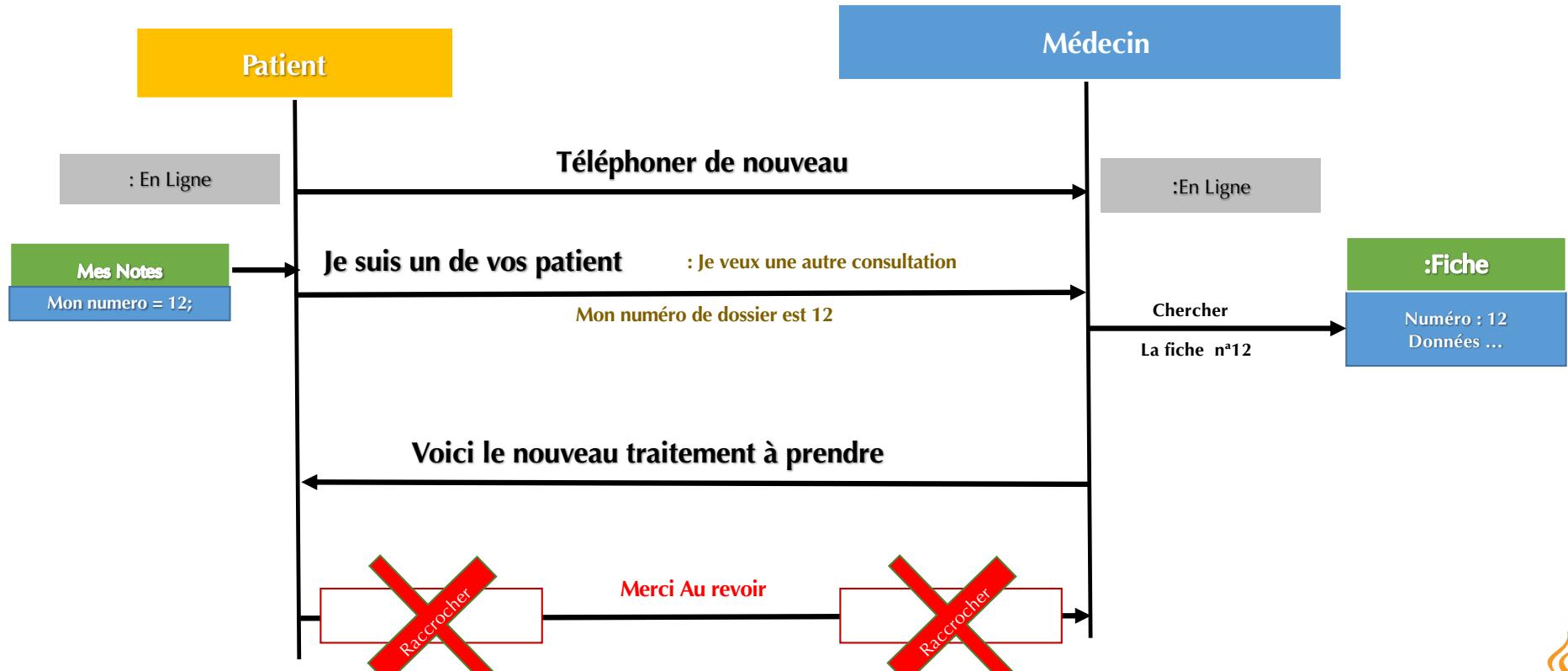
Modèle MVC : Contrôleur [SERVLET]

Session & Cookies



Modèle MVC : Contrôleur [SERVLET]

Session & Cookies



Modèle MVC : Contrôleur [SERVLET]

Structure d'une SERVLET

```
package controllers;

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

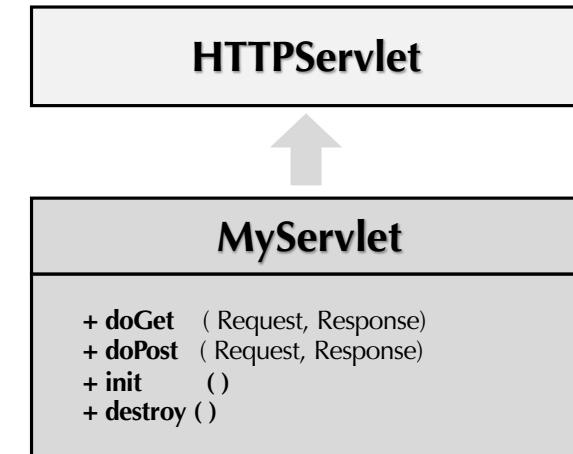
public class MyServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    @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
    }
}
```



Modèle MVC : Contrôleur [SERVLET]

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.



Modèle MVC : Contrôleur [SERVLET]

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.



Modèle MVC : Contrôleur [SERVLET]

Cycle de vie d'une SERVLET

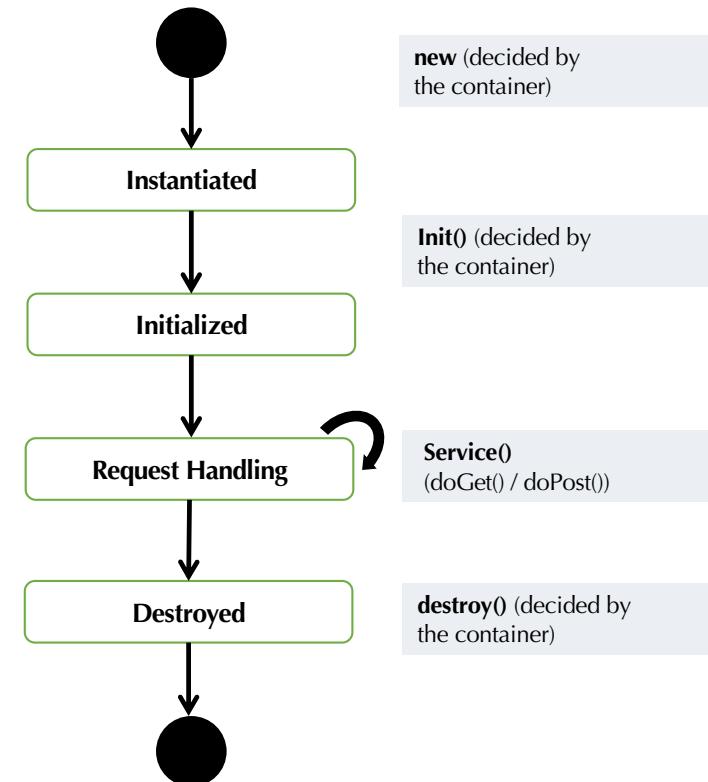
13. Une Servlet est déchargée par le conteneur dans les deux cas suivants :
 - Si le conteneur s'arrête,
 - Si le conteneur recharge l'environnement d'exécution de l'application web

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



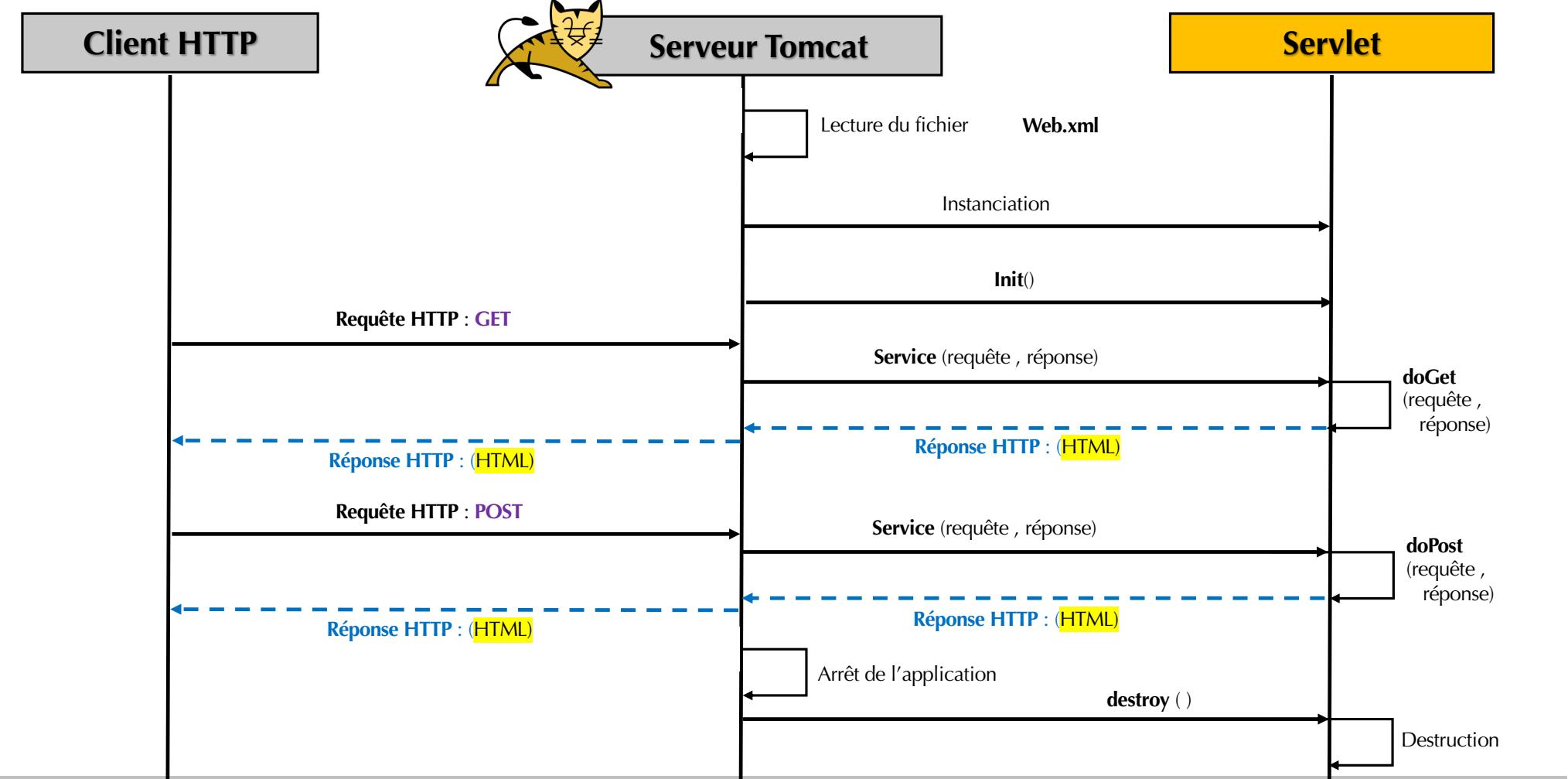
Modèle MVC : Contrôleur [SERVLET]

Cycle de vie d'une SERVLET



Le Contrôleur [SERVLET]

Cycle de vie

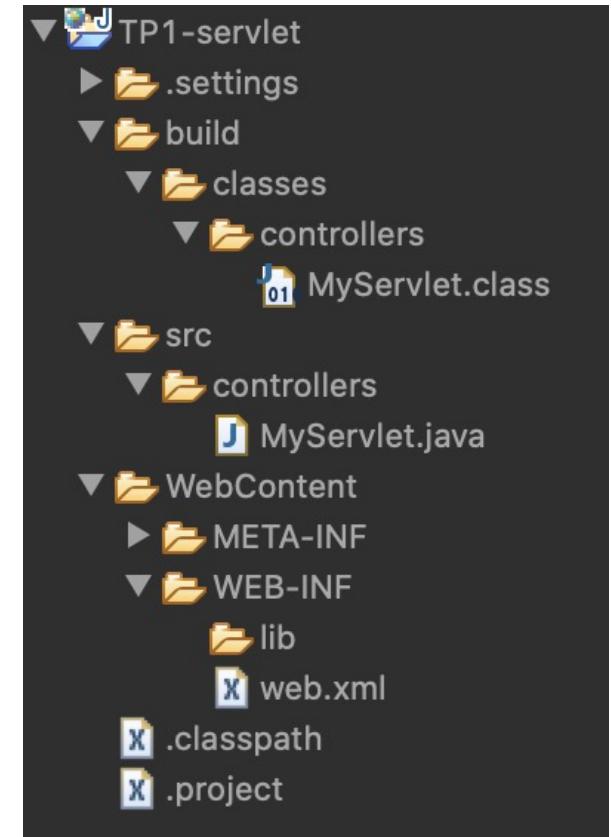


Modèle MVC : Contrôleur [SERVLET]

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 ou webApp** contient les documents Web :
 - ✓ comme les pages HTML, JSP, Images, JavaScript, CSS ...
- Le dossier **WEB-INF** contient les descripteurs de déploiement
 - ✓ web.xml.
- Le dossier **lib** permet de stocker les bibliothèques de classes java
 - ✓ Fichiers.jar



Modèle MVC : Contrôleur [SERVLET]

Déploiement d'une SERVLET

```
<?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> ← Balise de description de l'application WEB

  <servlet> ← Enregistrement de la servlet :
    <servlet-name>MyS</servlet-name>
    <servlet-class>controllers.MyServlet</servlet-class>
    - Identification (nom de la servlet)
    - Class de la servlet

  </servlet>

  <servlet-mapping> ← Mappage de la servlet :
    <servlet-name>MyS</servlet-name>
    <url-pattern>*.do</url-pattern>
    - Identification (nom de la servlet)
    - modèle de l'URL associé à la servlet

  </servlet-mapping>

</web-app>
```

Web.xml

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



Modèle MVC : Contrôleur [SERVLET]

Déploiement d'une SERVLET

```
<?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>

```

Web.xml

Balise de description de l'application WEB

Enregistrement de la servlet :

- Identification (nom de la servlet)
- Class de la servlet
- + Paramètres de description
- + Paramétrier la servlet à être charger dès le chargement de l'application, avec une priorité de chargement.



Modèle MVC : Contrôleur [SERVLET]

Déploiement d'une 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")**



Modèle MVC : Contrôleur [SERVLET]

Exemple d'une SERVLET

```
package controllers;

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

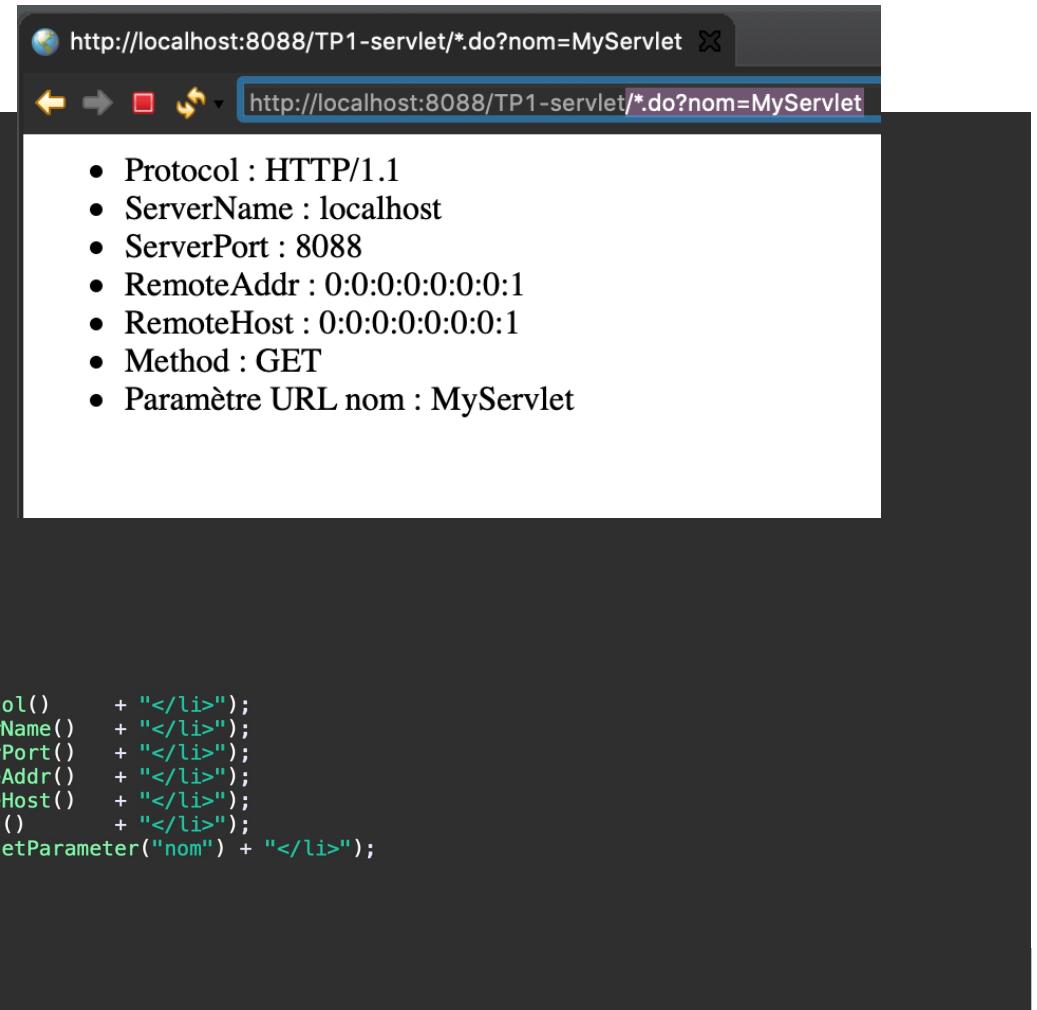
@WebServlet(name="MyS", urlPatterns={"/", "*.do"})
public class MyServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    @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>");

    }
}
```

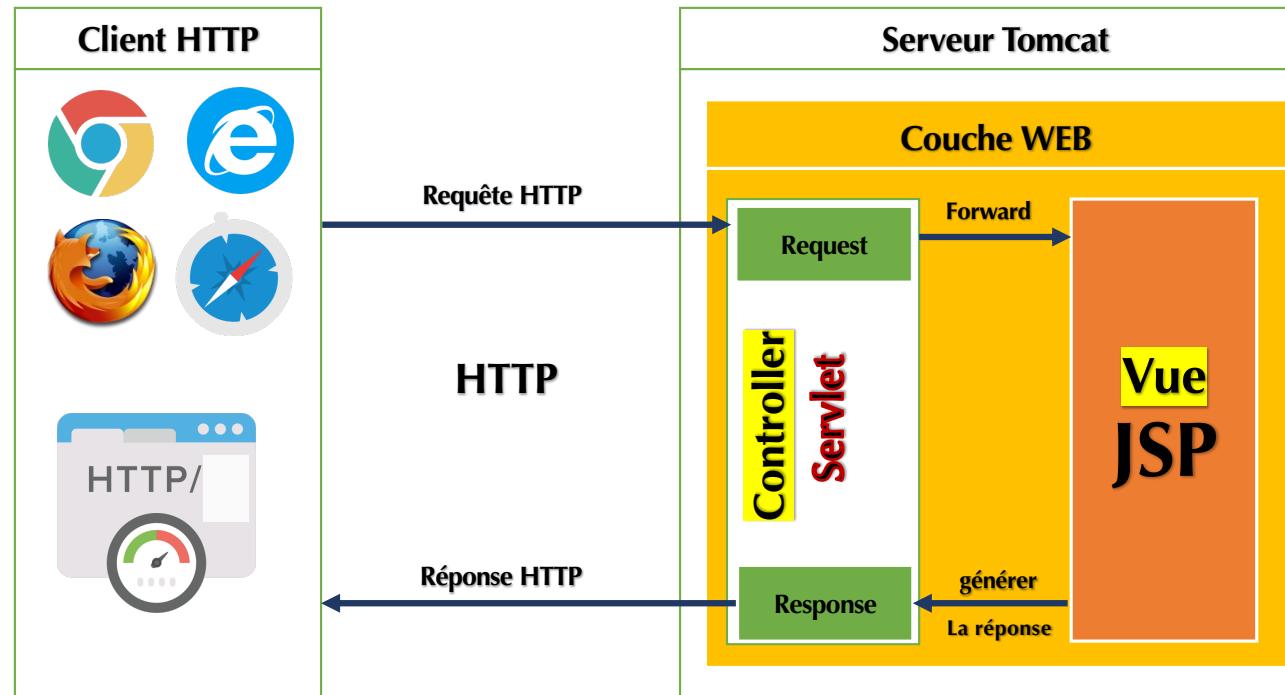


Modèle MVC : Contrôleur [SERVLET]

Forward (Navigation): 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);
```



Modèle MVC : Contrôleur [SERVLET]

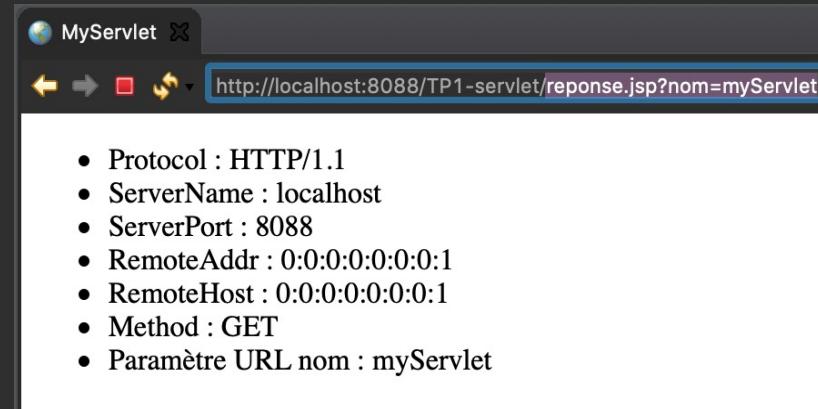
Forward (Navigation): SERVLET > JSP

On peut commencer 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 :



Modèle MVC : Contrôleur [SERVLET]

Forward (Navigation): 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 {

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

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

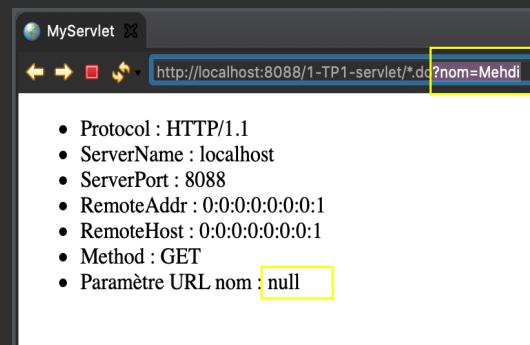
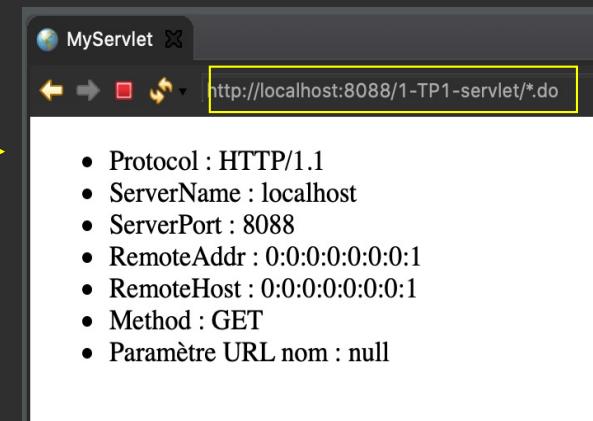
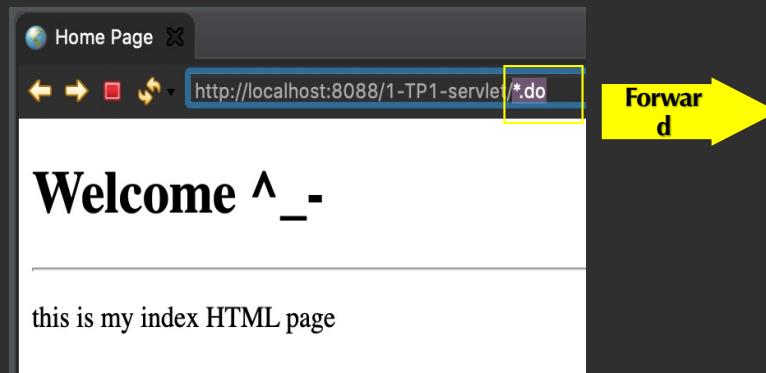
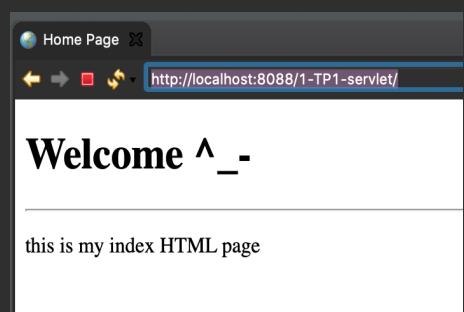
```
request.getRequestDispatcher("reponse.jsp").forward(request, response);
```



Modèle MVC : Contrôleur [SERVLET]

Forward (Navigation): SERVLET > JSP

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

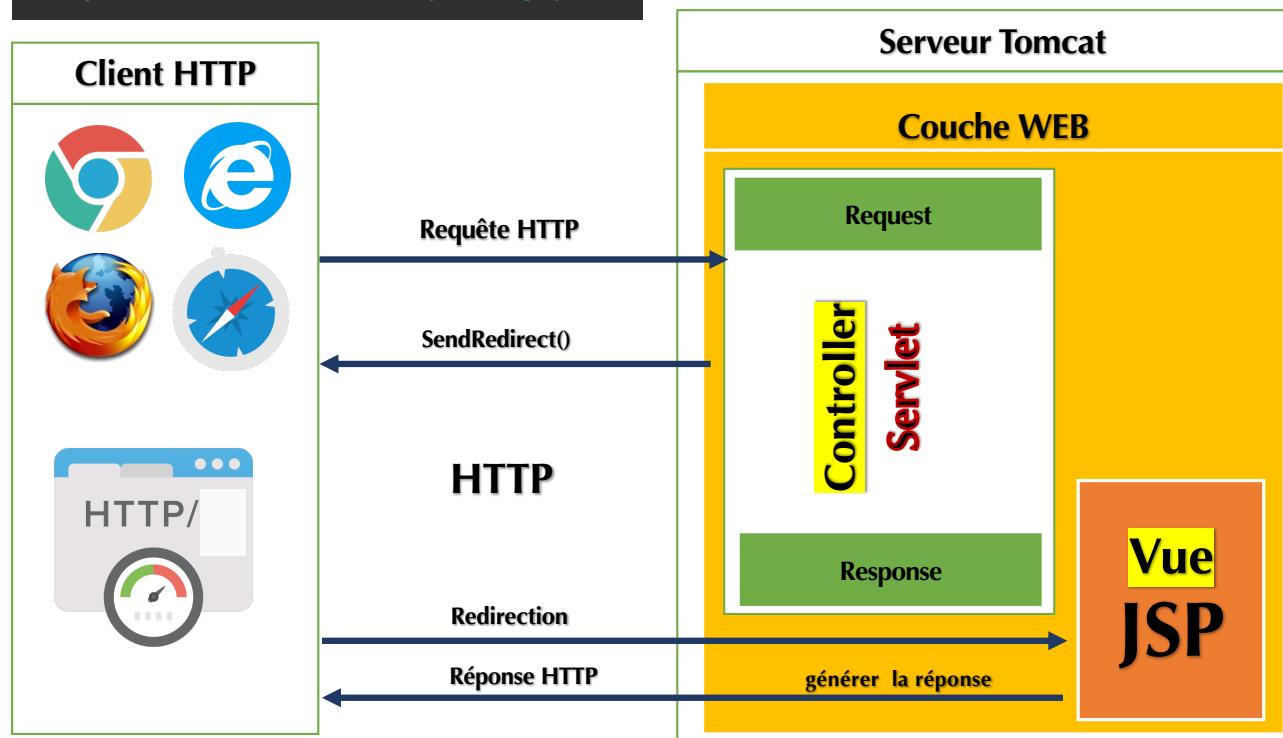


Modèle MVC : Contrôleur [SERVLET]

Redirect (Redirection): SERVLET > JSP

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

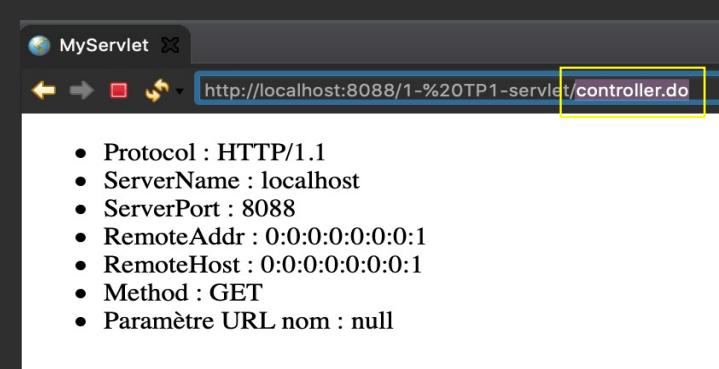
```
response.sendRedirect("reponse.jsp");
```



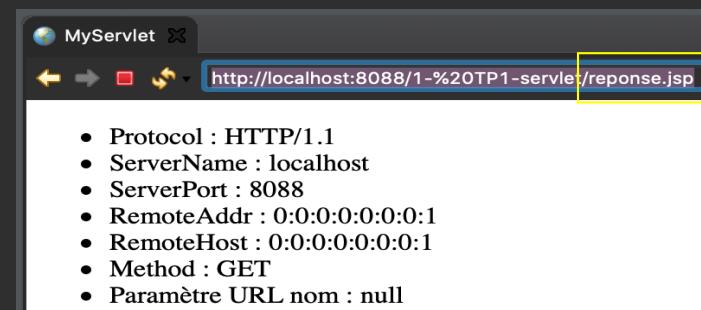
Modèle MVC : Contrôleur [SERVLET]

Redirect (Redirection): SERVLET > JSP

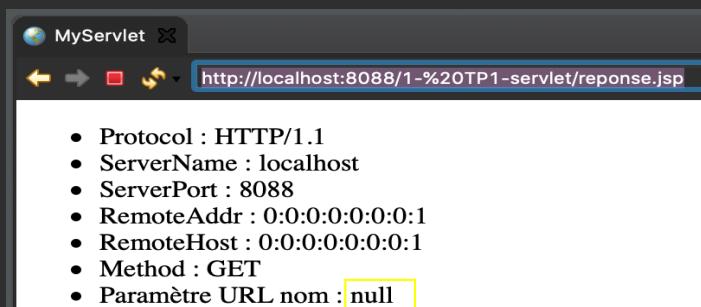
Rediriger la réponse vers la page JSP à partir de la servlet via :



L'url change vers la ressource



La redirection ne transporte pas les paramètres de la requête:



Modèle MVC : Contrôleur [SERVLET]

Redirect (Redirection): SERVLET > JSP

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

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



Modèle MVC : Contrôleur [SERVLET] et vue [JSP]

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, on peut écrire du code Java.
-
- Les pages **JSP** sont très pratiques quand on veut afficher les vues de l'application
 - Les **servlets** sont pratiques 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**.
-
- 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**

