



Programmation Web avec l'API Servlet

**Support de cours : 2^{ème} Année Génie Informatique
et 2^{ème} Année Ingénierie des données**

Préparé par : Tarik BOUDAA

Table des matières

Chapitre 0 : Rappels sur le Protocol HTTP	1
1. Protocol http :	1
2. Rappelles sur les méthodes http	3
3. URI, URL, URN	4
Chapitre 1 : Introduction à la programmation Web Java	5
1. Introduction	5
2. Descripteur de déploiement : Fichier web.xml	6
3. Configuration d'une servlet dans Web.xml ou par annotation	6
4. Invocation d'une servlet depuis un navigateur	7
5. Cycle de vie d'une servlet.	8
6. Récupération des paramètres de la requête (données issues du client)	12
7. Insérer et récupérer des données depuis l'objet <i>request</i> côté serveur	13
8. Récupération des en-têtes de la requête http	13
9. Les pages JSP	14
Chapitre 2 : Gestion du contexte et de la session	17
1. Notion de contexte d'une application Web :	17
2. Paramètres d'initialisation de l'application Web (paramètres du contexte):	18
3. Gestion des Cookies	21
4. Gestion de la session :	26
5. Portée des objets (<i>Objects scope</i>) dans une application Java Web :	29
Chapitre 3 : Redirection et Filtres.....	29
1. La redirection au niveau du client :	29
2. La redirection au niveau du serveur	30
3. Différences essentielles à connaître:.....	32
4. Encodage des URL envoyés au client	33
5. Implémentation des filtres dans l'API Servlet de Java EE	33

Chapitre 0 : Rappels sur le Protocol HTTP

1. Protocol http :

HTTP (*Hyper Text Transfert Protocol*) est le protocole utilisé pour les transactions Web. Il permet d'établir un dialogue entre un client (généralement, un navigateur) et un serveur Web. C'est un protocole sans état (Le client émet une requête, le serveur l'analyse, émet une réponse et c'est terminé).

2.1. Exemple de demande client et réponse du serveur :

▪ La demande du client :

Soit l'URL: <http://serveur.ensah.ma:80/>

Le navigateur l'interprète de la façon suivante :

- **http://** : Utiliser le protocole http
- **Serveur.ensah.ma** : Contacter le serveur de nom d'hôte serveur.ensah.ma
- **:80** : Se connecter sur le port 80. (C'est le port par défaut pour http)
- **/** : Tout ce qui suit est considéré comme le chemin du document dans le serveur.

Le message envoyé au serveur est:

<i>GET / HTTP1.1</i>	<i>1</i>
<i>Accept: image/gif, image/x-bitmap, image/jpeg, */*</i>	<i>2</i>
<i>Accept-Language: en-us</i>	<i>3</i>
<i>Accept-Encoding: gzip, deflate</i>	<i>4</i>
<i>User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT)</i>	<i>5</i>
<i>Host: serveur.ensah.ma</i>	<i>6</i>
<i>Connection: Keep-Alive</i>	<i>7</i>

Signification des lignes de la requête :

- 1- Le client demande au serveur (GET) le document dont la localisation est /. Le client indique sa version de protocole (HTTP1.1)
- 2- Le client indique au serveur quels sont les types de documents qu'il accepte.
- 3- Le client indique sa langue préférée (ici, l'anglais.)
- 4- Le client indique qu'il sait traiter une réponse compressée par gzip et/ou deflate
- 5- Le client s'identifie comme étant Microsoft Internet Explorer 5.01 tournant sous Window NT compatible Mozilla4.0
- 6- Le client fournit le nom du serveur vis-à-vis de lui-même.
- 7- Le client demande au serveur de conserver la connection ouverte. Il s'agit d'une connectionTCP et non d'une connection applicative.

▪ La réponse du serveur :

Le serveur recherche la ressource associée à « / » et la retourne en la faisant précéder par des en-têtes d'informations. La ressource peut être soit un fichier statique soit être générée dynamiquement en fonction de la requête.

<i>HTTP1.1 200 OK</i>	1	} En-tête
<i>Date: Mon, 15 Jul 2002 11:50:20 GMT</i>	2	
<i>Server: Tomcat-Apache/4.0.3 Win NT</i>	3	
<i>Last-Modified: Fri, 04 oct 2001 14:05:22 GMT</i>	4	
<i>Etag: "2f8ce-732-351e1ad6"</i>	5	
<i>Accept-Ranges: bytes</i>	6	
<i>Content-length: 212</i>	7	
<i>Connection: close</i>	8	
<i>Content-type: text/html</i>	9	
	10	(Ligne vide)

<i><html></i>	} Document
<i><head><title>Hello</title></head></i>	
<i><body></i>	
<i>...</i>	
<i></body></i>	
<i></html></i>	

Signification des lignes de la réponse :

- 1- Le serveur indique la version d'http qu'il utilise. Il fournit un code de retour (200 signifiant que le document demandé a été trouvé).
- 2- Le serveur indique l'heure et la date de la réponse.
- 3- Le serveur indique le logiciel employé.
- 4- Date de modification du document.
- 5- Le serveur fournit un descripteur pour le document. Ceci permet au client de gérer son cache.
- 6- Le serveur indique qu'il peut renvoyer des portions de document.
- 7- Le serveur indique la taille du corps du message qui suit les en-têtes.
- 8- Le serveur indique qu'il fermera la connexion après son envoi.
- 9- Le serveur indique la nature du document inclus dans la réponse (ici du html)
- 10- Le serveur envoie ensuite une ligne blanche puis le document. Le client se charge de l'afficher grâce au type de document (point 9).

2. Rappelles sur les méthodes http

2.1. Méthodes sûres et idempotentes

- Les méthodes sûres (*safe methods*) sont des méthodes HTTP qui ne modifient pas les ressources.
- Une méthode HTTP idempotente (*Idempotent method*) est une méthode HTTP qui peut être appelée plusieurs fois sans aboutir à des résultats différents.

Le tableau ci-dessous récapitule les différentes méthodes http avec leurs caractéristiques *safe* ou *idempotent* :

HTTP Method	Idempotent	Safe
OPTIONS	yes	yes
GET	yes	yes
HEAD	yes	yes
PUT	yes	no
POST	no	no
DELETE	yes	no
PATCH	no	no
TRACE	yes	yes

Source : <https://tools.ietf.org/html/rfc7231#page-75>

▪ Méthodes http (ou verbes http)

- **GET** : Récupération d'une ressource sur le serveur. Ce peut être : Le contenu d'un fichier statique (html) l'invocation d'un programme (un script, une servlet...) qui va générer une réponse. Ci-dessous d'autres caractéristiques d'une requête GET :
 - ✓ *La chaîne de requête (query string) de paires "nom/valeur" est envoyée dans l'URL d'une requête GET.*
 - ✓ *Les requêtes GET peuvent être mises en cache (can be cached)*
 - ✓ *Les requêtes GET restent dans l'historique du navigateur*
 - ✓ *Les requêtes GET peuvent être mises en signet (can be bookmarked)*
 - ✓ *Les requêtes GET ne doivent jamais être utilisées pour traiter des données sensibles*
 - ✓ *Les requêtes GET ont des restrictions de taille de données à envoyer*
 - ✓ *Les requêtes GET ont des restrictions sur le type de données (Seuls les caractères ASCII autorisés)*
 - ✓ *Les requêtes GET sont uniquement utilisées pour demander des données (pas pour les modifier)*
- **POST** : Cette méthode est utilisée pour transmettre des données en vue d'un traitement à une ressource sur le serveur (le plus souvent depuis un formulaire HTML). Les données ne sont pas visibles dans l'URL. On peut envoyer du binaire (par exemple un fichier en pièce jointe). Ci-dessous d'autres caractéristiques d'une requête POST :
 - ✓ *Les requêtes POST n'ont aucune restriction sur la longueur des données à envoyer*
 - ✓ *Les requêtes POST ne sont jamais mises en cache*
 - ✓ *Les requêtes POST ne restent pas dans l'historique du navigateur*

✓ *Les requêtes POST ne peuvent pas être mises en signet (cannot be bookmarked)*

- **HEAD** : est presque identique à GET, mais sans le corps de réponse. Généralement utilisée pour demander d'info sur un document. Par exemple : la date du document afin de savoir s'il est déjà dans le cache). Les requêtes HEAD sont utiles pour vérifier ce qu'une requête GET retournera avant de faire une demande GET - comme avant de télécharger un fichier volumineux ou un corps de réponse.
- **PUT** : est utilisée pour envoyer des données à un serveur afin de créer / mettre à jour une ressource. La différence entre POST et PUT est que les requêtes PUT sont idempotentes. C'est-à-dire qu'appeler plusieurs fois la même demande PUT produira toujours le même résultat. En revanche, l'appel répété d'une requête POST a pour effet secondaire de créer la même ressource plusieurs fois.
- **DELETE** : Permet de supprimer une ressource sur le serveur.

Autres méthodes http :

- **OPTIONS** : La méthode OPTIONS est utilisée pour décrire les options de communications avec la ressource visée.
- **TRACE** : La méthode TRACE réalise un message de test aller/retour en suivant le chemin de la ressource visée.
- **PATCH** : La méthode PATCH est utilisée pour appliquer des modifications partielles à une ressource.

Pour plus de détails voir : <https://tools.ietf.org/html/rfc7231>

3. URI, URL, URN



Un URI Uniform Resource Identifier, soit littéralement identifiant uniforme de ressource, peut être de type « locator » ou « name » ou les deux.

URL, désigne le nommage uniforme d'une ressource localisée. Les URL constituent un sous-ensemble des URI : c'est un identifiant unique d'accès à une ressource.

Un Uniform Resource Name (URN) est un URI qui identifie une ressource par son nom

Chapitre 1 : Introduction à la programmation Web Java

1. Introduction

Les servlets sont des programmes exécutés sur un serveur Web qui analysent les requêtes HTTP en provenance d'un navigateur Web, traitent la demande du client, puis fournissent une réponse adaptée au format HTML.

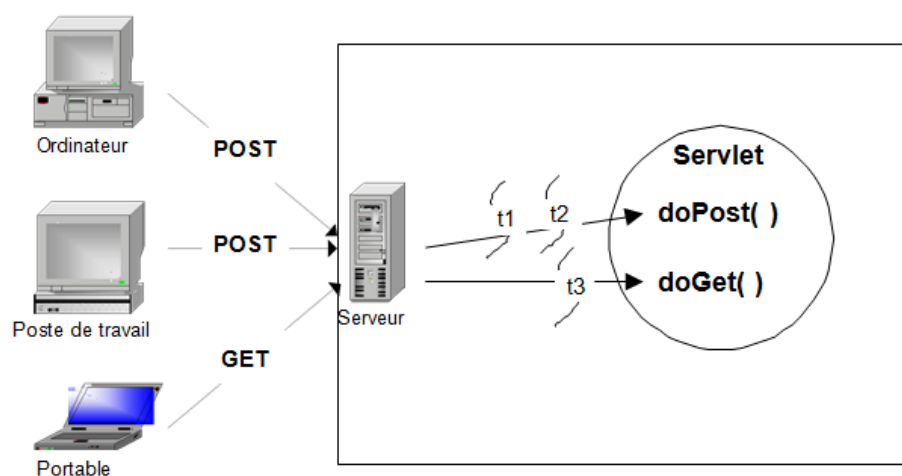
Les servlets sont la base de la programmation Web J2EE. Toute la conception d'une application Web en Java repose sur ces éléments que sont les servlets

Les rôles principaux d'une Servlet sont les suivants :

- Lire les données envoyées par l'utilisateur ces dernières proviennent généralement d'un formulaire d'une page Web, d'une Applet ou de tout application client HTTP.
- Acquérir des informations supplémentaires sur la requête : Identification du navigateur, type de méthode employée, valeurs des cookies, etc.
- Générer les résultats. Cela implique généralement l'accès à une couche logique métier.
- Formater le résultat dans un document. Dans la plupart des cas les résultats sont incorporés dans une page HTML.
- Définir les paramètres de la réponse HTTP appropriés. Il faut indiquer au navigateur le type de document renvoyé, définir des cookies, établir une session etc.
- Renvoyer le document au client. Le format peut être du texte (HTML), un format binaire (image, par exemple), un zip etc.

Parmi les avantages des Servlets :

- **Efficacité** : Pour les servlets, une seule instance traite simultanément les demandes identiques grâce au multithreading. L'instance est créée par le conteneur de servlet.



Servlet multi threadée (Un thread par requête entrante pour la même servlet)

- **Simplicité** : Les servlets disposent de moyens complets (classes et méthodes) pour analyser et traiter automatiquement les données de formulaires HTML, pour lire et

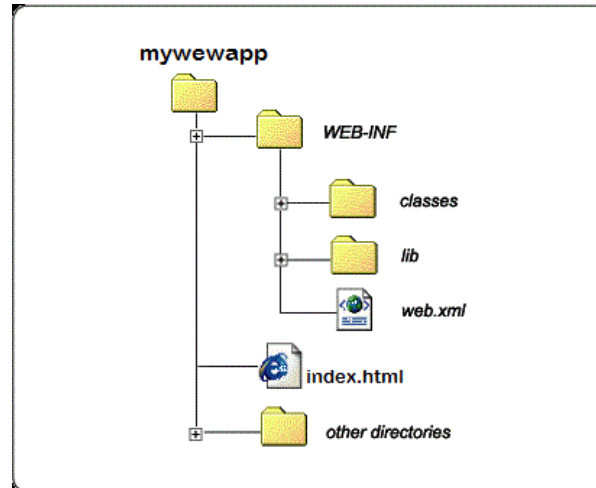
définir des en-têtes HTTP, pour gérer les cookies et le suivi de sessions. Elles accèdent à toute l'API Java, puisque déjà les Servlet sont du Java.

- **Puissance** : Les servlets sont gérées par un conteneur Web qui leur fournit des services indisponibles dans d'autres technologies (paramètres de servlets, suivi de sessions etc.)
- **Portabilité** : C'est une caractéristique du Java, les Servlets sont du java donc elles sont portables. Elles peuvent être exécutées, a priori sans modification, sur n'importe quel serveur de servlets : Tomcat, Websphere (IBM), WebLogic (BEA). Elles font partie de la plate-forme JEE.
- **Sécurité** : C'est une caractéristique du Java, les Servlets sont du java donc elles sont sécurisées.
- **Economique** : il existe des conteneurs de Servlet gratuits (Tomcat) , Java est gratuit.

2. Descripteur de déploiement : Fichier web.xml

Les applications Java EE doivent avoir une structure interne commune définie par la spécification. Toute application Java EE dispose d'un répertoire nommé WEB-INF, ce dossier présente la partie privée de l'application et il contient un fichier nommé web.xml qui présente le descripteur de déploiement. Le sous-dossier *WEB-INF/classes* contient les fichiers .class (compilés) de l'application. Le sous dossier *WEB-INF/lib* peut contenir les librairies externes sous forme de fichiers.jar

Le descripteur de déploiement d'une application web est un fichier nommé web.xml et il est situé dans le répertoire WEB-INF. Il contient les caractéristiques et les paramètres de l'application.



3. Configuration d'une servlet dans Web.xml ou par annotation

Pour que la servlet soit connue par le conteneur, elle doit être décrite dans le fichier web.xml ou annotée par l'annotation `@WebServlet`.

Exemple :

Supposons que nous avons écrit une servlet dans la classe `ensah.ihm.EnsahServlet` et que nous voulons que cette Servlet soit invoquée `/HelloEnsah`. Pour ce faire, nous pouvons, employer l'une des configurations ci-dessous :

- **Configuration par le fichier web.xml :**

```
<!DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_4.dtd">
<web-app>
<servlet>
    <servlet-name>HelloEnsahServlet</servlet-name>
    <description>C'est un identifiant pour la classe qui suit. </description>
    <servlet-class>ensah.ihm.EnsahServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>HelloEnsahServlet</servlet-name>
    <url-pattern>/HelloEnsah</url-pattern>
</servlet-mapping>
</web-app>
```

Identification de la Servlet

Mapping url – identifiant de servlet
(Définir comment invoquer la servlet ayant l'identifiant HelloEnsahServlet)

- **Configuration par l'annotation @WebServlet**

L'API Servlet 3.0 a introduit des annotations permettant de configurer les Servlets. L'annotation `@WebServlet` permet de configurer une Servlet sans avoir besoin d'utiliser le fichier Web.xml. L'exemple précédent peut être écrit d'une façon plus simple avec le code ci-dessous :

```
@WebServlet( name="HelloEnsahServlet", urlPatterns = "/HelloEnsah" )
public class EnsahServlet extends HttpServlet {
    //code de la servlet
}
```

Si vous nous n'avons pas besoin du nom de la Servlet nous pouvons utiliser la syntaxe suivante :

```
@WebServlet("/HelloEnsah" )
public class EnsahServlet extends HttpServlet {
    //code de la servlet
}
```

4. Invocation d'une servlet depuis un navigateur

On prend comme exemple la servlet de la classe `ensah.ihm.EnsahServlet` de l'application nommée `ensah` :

- **Invocation d'une servlet depuis un navigateur :**

On peut appeler la Servlet avec son mapping url : `http://serveur:port/ensah/HelloEnsah`

- **Invocation d'une servlet depuis une page html**

Sur une balise `<a >` :

```
<a href="http://serveur:port/ensah/HelloEnsah">Cliquez ici</a>
```

Sur une balise <form> :

```
<form action="http://serveur:port/ensah/HelloEnsah" method="post">
```

.....

```
</form>
```

Remarque :

Dans les exemples précédents nous avons introduit dans le code des URL (*la partie `http://serveur:port/ensah`*) qui peuvent être changées après le déploiement réel de l'application. Pour faire les choses d'une façon plus propre il faut récupérer le contexte de l'application d'une façon dynamique, comme dans l'exemple suivant :

```
<form action="<%=request.getServletContext().getContextPath()%>/HelloEnsah"
method="post">
```

Ou plus proprement avec une expression EL :

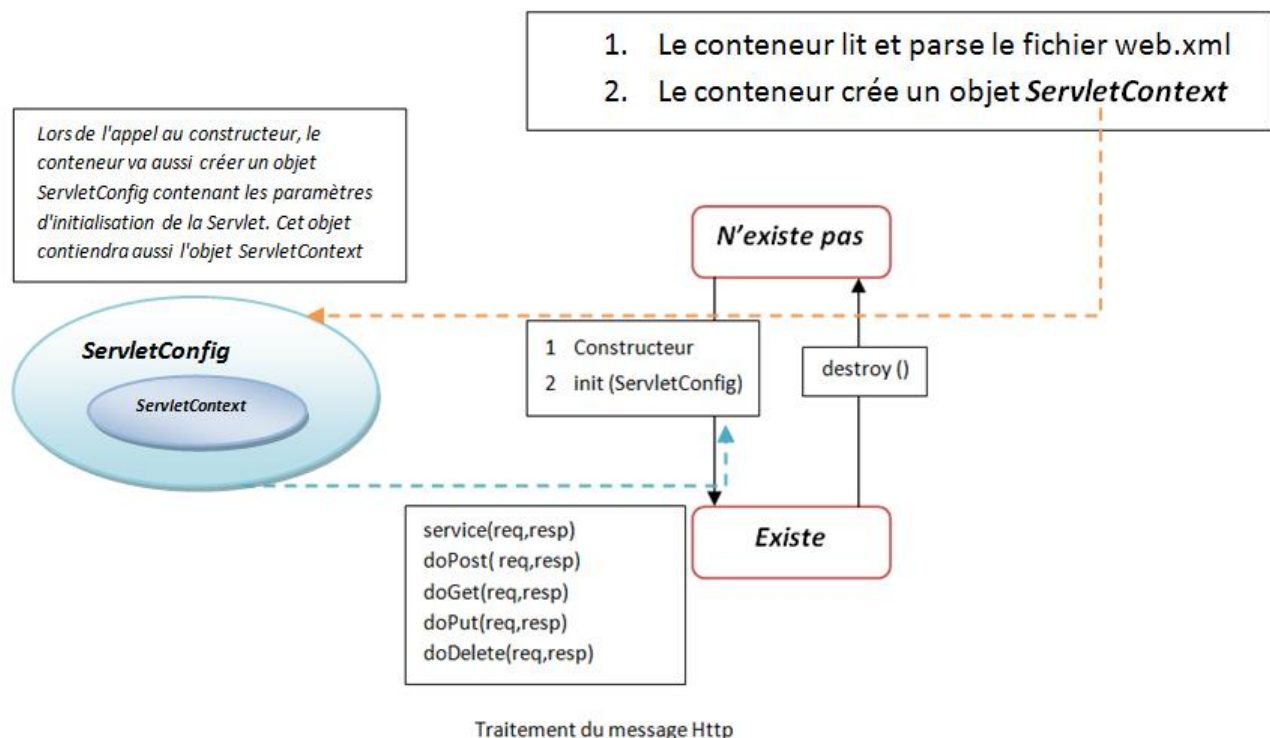
```
<c:set var="context" value="${pageContext.request.contextPath}" />
```

```
<form action="${context}/HelloEnsah" method="post">
```

5. Cycle de vie d'une servlet.

- Le conteneur de servlet gère son cycle de vie. l'instance est créée automatiquement par le conteneur Web.
- Le conteneur peut détruire la servlet à tout moment.
- Les méthodes des servlets sont multiThreadées.

6.1. Initialisation de la Servlet :



- La méthode init

La méthode `init(ServletConfig)` est appelée une seule fois après la création de la servlet. Elle reçoit un paramètre de type `ServletConfig` (récupérable par la suite grâce à `getServletConfig()`) contenant les paramètres de configuration de la servlet.

La méthode `init(ServletConfig config)` qu'utilise le conteneur ne doit pas être redéfinie, il est plus astucieux de redéfinir `init()`, car cette méthode est appelée automatiquement par `init(ServletConfig)` et, en standard, ne fait rien. On peut y récupérer les paramètres de la servlet dans le `ServletConfig`.

- La méthode service

Chaque fois que le serveur reçoit une requête pour une servlet il crée un thread et y appelle la méthode `service` (ou il obtient un thread prêt depuis un pool de thread). Celle-ci vérifie le type de la requête http (GET, POST, PUT, etc ...) et appelle la méthode correspondante (`doGet`, `doPost`, `doPut`, etc.). Cette méthode n'est pas à redéfinir.

- Astuce :

Si vous traitez de la même façon GET et POST il est préférable de créer une méthode de traitement et de l'invoquer dans `doPost` et `doGet`

```
public void performTask(HttpServletRequest request,
                        HttpServletResponse response)
{
    //traitement de la demande
    ...
}

public void doPost(HttpServletRequest request,
                  HttpServletResponse response)
{
    performTask(request, response);
}

public void doGet(HttpServletRequest request, HttpServletResponse
                  response)
{
    performTask(request, response);
}
```

6.2. Paramètres d'initialisation de la Servlet

Lorsque la servlet est instanciée par le conteneur en appelant son constructeur, ce dernier peut lui passer des paramètres d'initialisation, grâce à l'appelle la méthode `init(ServletConfig)`. Ces paramètres d'initialisation sont définis dans le fichier `web.xml`.

Le descripteur de déploiement peut définir des paramètres qui s'appliquent à la servlet au moyen de l'élément `<init-param>`. Le conteneur de servlets lit ces paramètres dans le

fichier<web.xml> et les stocke sous forme de paires clé/valeur dans l'objet ServletConfig. L'interface Servlet ne définissant que init(ServletConfig), c'est cette méthode que le conteneur doit appeler. GenericServlet implémente cette méthode pour stocker la référence à ServletConfig puis appelle la méthode init() sans paramètres.

Ainsi pour effectuer l'initialisation, nous avons seulement besoin de redéfinir init() sans paramètre. La référence à ServletConfig étant déjà mémorisée, la méthode init() a accès à tous les paramètres d'initialisation qui y sont stockées. Pour récupérer ces paramètres, il suffit d'utiliser la méthode getInitParameter(String).

- Exemple :

Voici un exemple de paramètres initiaux définissant le serveur de base de données MySQL définies dans le descripteur de déploiement associés à la servlet TestConnexion:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4"
xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
    <display-name>exemple</display-name>

    <servlet>

        <servlet-name>TestConnexion</servlet-name>
        <servlet-class>com.ensah.tp.ihm.TestConnexion</servlet-class>

        <init-param>
            <param-name>jdbc.Drvier</param-name>
            <param-value>com.mysql.jdbc.Driver</param-value>
        </init-param>

        <init-param>
            <param-name>localisation</param-name>
            <param-value>jdbc:mysql://localhost/gestion</param-value>
        </init-param>

    </servlet>
</web-app>
```

Il est à noter que nous pouvons faire cette configuration également en utilisant l'annotation @WebInitParam.

Exemple :

Ici la servlet a deux paramètres d'initialisation param1 et param2 :

```
@WebServlet(
    value = "/MyServlet",
    initParams = {
        @WebInitParam(name = "param1", value = "val1"),
        @WebInitParam(name = "param2", value = "val2")
    }
)
public class MyServlet extends HttpServlet {
}
```

Récupération des paramètres dans la méthode init :

```

public void init() {

    // La servlet a une méthode de récupération des noms de ces
    // paramètres
    Enumeration e = getInitParameterNames();

    // On récupère un objet de configuration de servlets
    ServletConfig conf = getServletConfig();

    // Et on parcourt le tout
    while (e.hasMoreElements()) {
        String name = e.nextElement().toString();
        // On appelle la méthode getInitParameter(String name)
        // afin de récupérer la valeur
        System.out.println(conf.getInitParameter(name));
    }

}

```

La spécification Servlet exige que la méthode `init` soit terminée sans erreur pour traiter une requête. Si votre code rencontre un problème lors de l'exécution de la méthode `init()`, vous devez lancer une `ServletException`. Le conteneur saura ainsi que l'initialisation ne s'est pas déroulée correctement et qu'il ne doit pas utiliser cette servlet pour traiter des requêtes.

```

public void init() throws ServletException {

    // La servlet a une méthode de récupération des noms de //ces
    // paramètres
    Enumeration e = getInitParameterNames();

    // On récupère un objet de configuration de servlets
    ServletConfig conf = getServletConfig();

    try{
        Class.forName(conf.getInitParameter("Driver"));
        Connection conne =
        DriverManager.getConnection(conf.getInitParameter("location"), "user", "pass");
        PreparedStatement ins = conne.prepareStatement("Select ....");
    }
    catch (ClassNotFoundException e2) {
        Logguer.log("erreur driver");
        throw new ServletException();
    } catch (SQLException e1) {

        Logguer.log("erreur serveur introuvable");
        throw new ServletException();
    }
    ... }

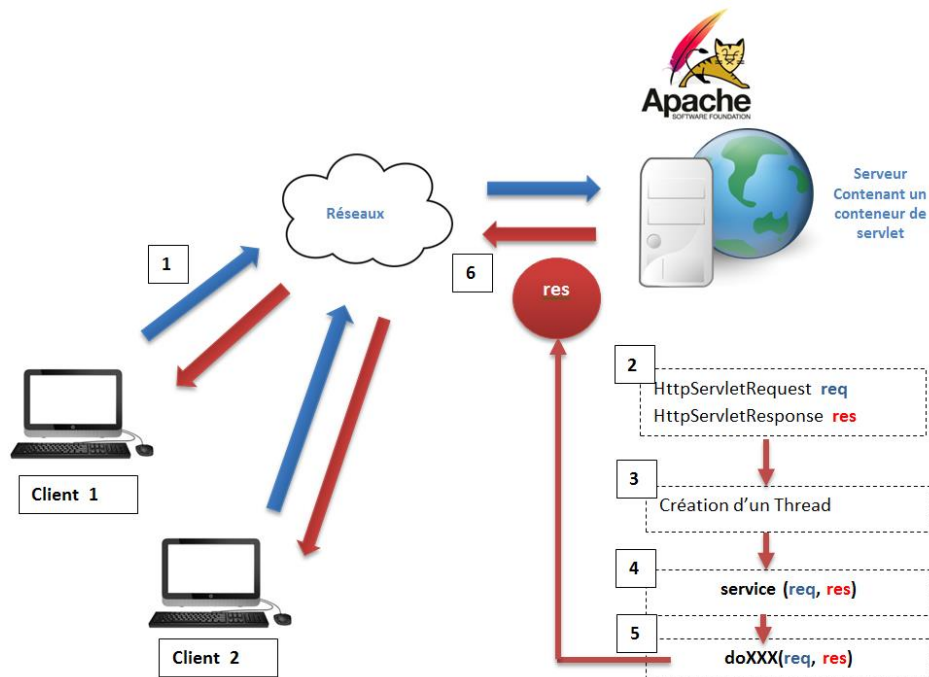
```

6.3. Utilisation de la servlet :

On suppose à présent que la servlet est déjà initialisée. Le schéma ci-dessous résume les étapes déclenchées après l'envoi d'une requête au serveur web.

- **Etape 1 :** Le client envoie une requête
- **Etape 2 :** Le conteneur Web Crée les deux objets `req` et `res` de types `HttpServletRequest` et `HttpServletResponse` respectivement
- **Etape 3 :** Création d'un Thread pour traiter la requête

- **Etape 4** : Appel de la méthode service avec en paramètres les objets req et res
- **Etape 5** : appel de la méthode doGet, doPost, ou doXXX selon le type de la requête
- **Etape 6** : Le conteneur envoie la réponse au client



6. Récupération des paramètres de la requête (données issues du client)

6.1. Traitement d'un GET :

Envoi des paramètres avec la méthode GET dans un lien:

``
 Cliquer ICI ``

Récupération des paramètres dans la Servlet avec la méthode `getParameter`:

```
String valparam1 = request.getParameter("param1");
```

```
String valparam2 = request.getParameter("param2");
```

6.2. Traitement d'un POST

Pour un champ de formulaire portant un nom « *nameParameter* » on utilise la méthodes `request.getParameter(nameParameter)`.

- **Le formulaire HTML :**

```
<form action="http://localhost:8080/TP2/EnsaServlet/RecupParam"  
method="post">
```

```
<table width="100" border="0">  
<tr><td>Nom</td><td> <input type="text" name="Nom"></td></tr>  
<tr><td>Prenom</td><td> <input type="text" name="Prenom"></td></tr>
```

```

<tr><td>AnneeNaissance</td><td><input type="text" name="AnneeNaissance">
</td></tr>
<tr><td>Sexe</td><td><input type="text" name="Sexe"></td></tr>
</table>
<p><input type="submit" name="Submit" value="Soumettre"></p>

</form>

```

- Traitement dans la Servlet :

```

doPost(HttpServletRequest req , HttpServletResponse rep)
{
    nom = req.getParameter ("Nom");
    prenom =req.getParameter ( "Prenom" );
    anneeNaissance =
    Integer.parseInt(req.getParameter("AnneeNaissance");
    sexe = req.getParameter("Sexe").charAt(0);
}

```

7. Insérer et récupérer des données depuis l'objet *request* côté serveur

A part les données issues du client sous forme de paramètres d'une requête de type clé/valeurs où la clé et la valeur sont des chaînes de caractères, nous pouvons également insérer des données dans la requête au niveau serveur sous forme également de clé/valeur mais cette fois-ci la valeur est de type objet quelconque. Pour ce faire, il suffit d'appeler la méthode *setAttribute()* de l'objet *request* pour y enregistrer un attribut de type objet quelconque. Cette méthode prend en paramètre le nom que l'on souhaite donner à l'attribut (clé) suivi de l'objet à mettre dans la requête (la valeur). Exemple : *request.setAttribute("user", new User());* ;

En ce qui concerne la récupération de l'attribut depuis la requête il se fait en appelant la méthode *request.getAttribute("nom_attribut")*

Nous pouvons employer cette technique pour passer des données d'une Servlet à une autre dans le cadre d'une redirection de type *forward* par exemple.

8. Récupération des en-têtes de la requête http

Il peut être intéressant pour la servlet de pouvoir les récupérer les en-têtes de la requête http. Cela se fait directement sur l'objet *request* de type *HttpServletRequest*, fabriqué par le container et reçu dans les méthodes *doGet*, *doPost* , *doXxx*.

- Exemple :

```

public class EnsahServlet extends HttpServlet {

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

    // Renvoie la partie de L'Url située entre le Port et les données
    //de formulaires (queryString)
    String getRequestURI = request.getRequestURI();

    // Renvoie le chemin réel correspondant à la partie de L'Url
    //située entre l'url pattern demandé et la queryString

```

```

        String pathTranslated = request.getPathTranslated();

        // Renvoie le nom de la méthode Get, Post ...
        String method = request.getMethod();

        // Renvoie le username si l'utilisateur a été authentifié
        String remoteUser = request.getRemoteUser();

        // Renvoie la taille en octets du message qui suit les en-têtes
        int contentLength = request.getContentLength();

        // Renvoie le protocole utilisé.
        String protocol = request.getProtocol();

    }

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

        doGet(request, response);

    }

}

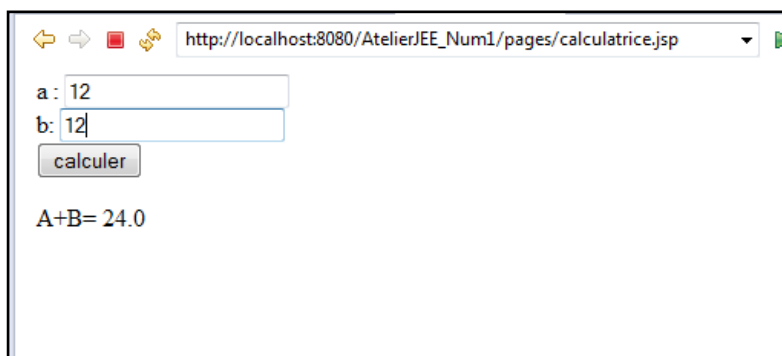
```

9. Les pages JSP

Les pages JSP (Java Server Pages) sont une autre façon d'écrire des applications serveurs web. En fait ces pages JSP sont traduites en servlets avant d'être exécutées et on retrouve alors la technologie des servlets. Les pages JSP permettent de mieux mettre en relief la structure des pages HTML générées.

- Exemple

On considère ici l'exemple d'un programme web « calculatrice » permettant de faire le calcul de l'addition de deux nombres :



Etudions le code de la page JSP associée à la figure précédente :


```

String valeura = request.getParameter("a");
String valeurb = request.getParameter("b");

%>

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Calculatrice</title>
</head>
<body>
    <form method="POST">
        a : <input type="text" size="20" name="a"> <br>
        b : <input type="text" size="20" name="b"> <br>
        <input value="calculer" type="submit">
    </form>

    <p>
        <%
            if (valeura != null || valeurb != null) {

                String result = String.valueOf(Double.parseDouble(valeura)
                    + Double.parseDouble(valeurb));
                out.println("A+B= " + result + "<br>");

            }

        %>
    </p>
</body>
</html>

```

On note les points suivants :

- Le code ressemble fortement à une page HTML. On y trouve cependant des balises de type `<% %>` qui sont propres au langage JSP et qui englobent des instructions Java.
- La balise `<%` introduit du code Java. Ce code se termine à la rencontre de la balise de fermeture de code `%>`
- L'ensemble du code précédent (HTML + JSP) va faire l'objet d'une conversion en servlet Java. Il sera enfermé dans une unique méthode, appelée méthode principale de la page JSP. C'est pourquoi, les variables java déclarées au début de la page JSP sont connues dans les autres portions de code JSP qui parsèment le code HTML : ces variables et portions de code feront partie de la même méthode Java
- Cet exemple utilise les méthodes de l'objet Java request qui est l'objet request déjà rencontré dans l'étude des servlets. C'est donc un objet `HttpServletRequest`.
- On peut arriver au même résultat avec une servlet mais ici la structure de la page web est plus évidente.
- Pour inclure des parties dynamiques dans le code HTML deux méthodes sont possibles : `<%= expression %>` ou `out.println(expression)`. L'objet out est un flux de sortie analogue à celui de même nom rencontré dans les exemples de servlets mais pas du même type : c'est un objet `JspWriter` et non un `PrintWriter`. Il permet d'écrire dans le flux HTML avec les méthodes `print` et `println`.

La page JSP reflète mieux la structure de la page HTML générée que la servlet équivalente.

Voici une liste de balises qu'on peut rencontrer dans une page JSP et leur signification.

<code><!--Commentaire--></code>	Commentaire HTML envoyé au client
---------------------------------------	-----------------------------------

< %-- Commentaire --%>	Commentaire JSP ne sera pas envoyé au client
< %! Déclaration, Méthode %>	déclare des variables globales et méthodes. Les variables seront connues dans toutes les méthodes
<%= expression =>	la valeur de expression sera intégrée dans la page HTML à la place de la balise
<% code java %>	contient du code Java qui fera partie de la méthode principale de la page JSP
<%@ page attribut1= valeur1 attribut2 = valeur2... %>	Fixe des attributs pour la page JSP. Par exemple : import="java.util.*,java.sql.*" pour préciser les bibliothèques nécessaires à la page JSP extends="uneClasseParent" pour faire dériver la page JSP d'une autre classe

Dans les exemples précédents, nous avons rencontré deux objets non déclarés : request et out. Ce sont deux des objets qui sont automatiquement définis dans la servlet dans laquelle est convertie la page JSP. On les appelle des objets implicites ou prédéfinis

- Objets implicites

HttpServletRequest request	l'objet à partir duquel on a accès à la requête du client Web (getParameter, getParameterNames, getParameterValues)
HttpServletResponse response	l'objet avec lequel on peut construire la réponse du serveur Web à son client. Permet de fixer les entêtes http à envoyer au client Web
JspWriter out	le flux de sortie qui nous permet d'envoyer du code HTML au client (print,println)

- Méthodes de collaboration JSP et Servlet

Comment une servlet peut-elle passer la requête qu'elle a reçue d'un client à une autre servlet ou à une page JSP ? Nous utiliserons les méthodes suivantes :

[ServletContext].getRequestDispatcher(String url)	Méthode de la classe <i>ServletContext</i> qui rend un objet <i>RequestDispatcher</i> . Le paramètre url est le nom de l'URL à qui on veut transmettre la requête du client. Ce passage de requête ne peut se faire qu'au sein d'une même application. Le paramètre url est un chemin relatif à l'arborescence web de cette application.
[RequestDispatcher].forward(ServletRequest request, ServletResponse response)	Méthode de l'interface <i>RequestDispatcher</i> qui transmet à l'URL précédente la requête <i>request</i> du client et l'objet <i>response</i> qui doit être utilisé pour élaborer la réponse.
[ServletRequest].setAttribute(String nom, Object obj)	lorsqu'une servlet ou page JSP passe une requête à une autre servlet ou page JSP, elle a en général besoin de passer à celle-ci d'autres informations que la seule requête du client, informations issues de son propre travail sur la requête. La méthode <i>setAttribute</i> de la classe <i>ServletRequest</i> permet d'ajouter des attributs à l'objet <i>request</i> du client sous une forme qui ressemble à un dictionnaire de couples (attribut, valeur) où attribut est le nom de l'attribut et valeur un objet quelconque représentant la valeur de celui-ci.

[ServletRequest].getAttribute(String attribut)	permet de récupérer les valeurs des attributs d'une requête. Cette méthode sera utilisée par la servlet ou la page JSP à qui on a relayé une requête pour obtenir les informations qui y ont été ajoutées.
---	--

Chapitre 2 : Gestion du contexte et de la session

1. Notion de contexte d'une application Web :

Un contexte constitue pour chaque servlet d'une même application une vue sur le fonctionnement de cette application. Une application web peut être composée de :

- servlets
- JSP
- classes utilitaires
- documents statiques (pages html, images, sons, etc.)
- beans, applications clients
- méta informations décrivant la structure de l'application

Dans le code source d'une servlet, un contexte est représenté par un objet de type `ServletContext` qui peut être obtenu par l'intermédiaire d'un objet de type `ServletConfig`.

Grâce à ce contexte, il est possible d'accéder à chacune des ressources de l'application web correspondant au contexte. Il faut noter qu'à une application correspond un et un seul contexte, et qu'à un contexte correspond une et une seule application. On peut donc en déduire que chaque

contexte est propre à une application et qu'il n'est pas possible de partager des ressources entre applications différentes via un contexte.

Les ressources qu'il est possible de partager sont :

- Des documents statiques. Vous pouvez accéder à n'importe quel document statique d'un contexte grâce à la méthode `getResource` ou à `getResourceAsStream`. Le chemin passé en paramètre est interprété de façon relative à la racine de l'application correspondant au contexte en cours.
- Des instances de servlets : en utilisant la méthode `getServlet`
- Des paramètres d'initialisation pour toute l'application. En utilisant la méthode `getInitParameter`, il est possible de connaître la valeur d'un paramètre d'initialisation si on possède son nom. Pour connaître tous les paramètres d'initialisations définis, il faut utiliser la méthode `getInitParameterNames`.
- Des attributs d'instance de servlets. Il est ainsi possible de partager des données au sein d'une même application. Pour cela, utilisez les méthodes `setAttribute` et `getAttribute`, en fournissant à chacune de ces méthodes le nom que devra avoir l'attribut au sein de l'application. Pour obtenir le nom de tous les attributs partagés, vous devez utiliser la méthode `getAttributeNames` et pour retirer un attribut, il suffit d'utiliser la méthode `removeAttribute`

2. Paramètres d'initialisation de l'application Web (paramètres du contexte):

Les paramètres d'initialisation de l'application Web, appelés aussi paramètres de contexte sont des paramètres accessibles pour tous les composants Web constituant l'application Web. Ils sont définis par la balise `<context-param>` dans le descripteur de déploiement `web.xml`.

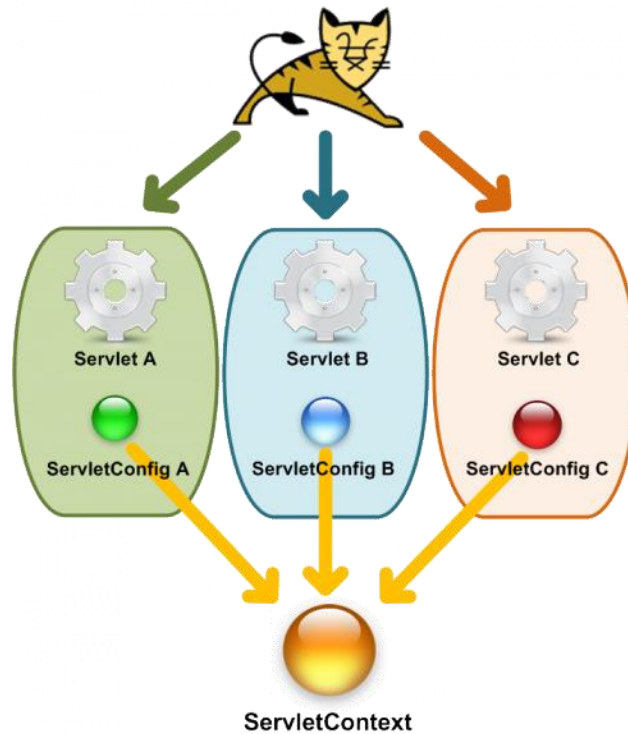
La balise `<context-param>`, comme pour `<init-param>` est composé de deux sous-éléments qui correspondent respectivement au nom du paramètre suivi de sa valeur :

- `<param-name>` : nom du paramètre
- `<param-value>` : valeur du paramètre

Pour que chacune des servlets récupèrent ces paramètres, il est nécessaire d'abord de solliciter l'objet correspondant au contexte de la servlet à l'aide de la méthode `getServletContext()`. Cet objet ensuite, peut délivrer les paramètres initiaux de l'application Web grâce à la méthode `getInitParameter()`.

- Remarque :

- La durée de vie d'un contexte est celle du conteneur Web
- Il y a un objet `ServletContext` par application Web.
On peut l'utiliser pour fabriquer des singletons.
- Il y a un objet `ServletConfig` par Servlet



- Exemple1 :

Définition des paramètres du contexte dans le fichier web.xml

```
<!-- parametres de contexte -->
<context-param>
    <param-name>Serveur</param-name>
    <param-value>ENSA A1 Hoceima</param-value>
</context-param>
```

- Servlet 1 :

```
public class EnsahServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

        ServletContext srvCntx = getServletContext();
        System.out.println(srvCntx.getInitParameter("Serveur"));

        Personne p = new Personne("Boudaa", "Tarik");
        System.out.println("On est dans la Servlet 1");
        srvCntx.setAttribute("personne", p);
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException,
        IOException {

        doGet(request, response);
    }
}
```

- Servlet 2 :

```
public class EnsahServlet2 extends HttpServlet {
```

```

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

    ServletContext srvCntx = getServletContext();

    System.out.println("on est dans la Servlet 2");

    Personne p = (Personne) srvCntx.getAttribute("personne");

    System.out.println(p.getNom());

}

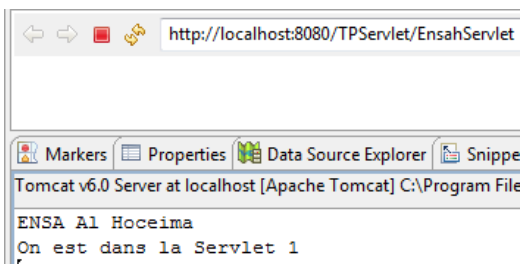
protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException,
    IOException {

    doGet(request, response);

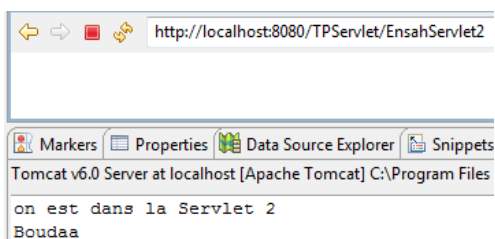
}
}

```

- Invocation de la Servlet 1 :



- Invocation de la Servlet 2:



Dans l'exemple ci-dessous Les servlets partagent l'objet Personne via le contexte.

- Exemple 2 :

On définit des paramètres d'initialisation du contexte dans le descripteur de déploiement :

```

<web-app>

<servlet>
    <servlet-class>com.ensat.TestServlet</servlet-class>
    <servlet-name>test</servlet-name>
    <init-param>
        <param-name>test</param-name>
        <param-value>1</param-value>
    </init-param>
</servlet>

<context-param>
    <param-name>contextParam1</param-name>

```

```

        <param-value>Je suis une application ENSAH !</param-value>
    </context-param>

    <context-param>
        <param-name>contextParam2</param-name>
        <param-value>Je suis une application du groupe NovaTeam</param-value>
    </context-param>

    <servlet-mapping>
        <servlet-name> test </servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
</web-app>

```

Récupération des paramètres du contexte dans la méthode init de la Servlet :

```

public void init() throws ServletException {

    Enumeration e = getInitParameterNames();
    ServletConfig conf = getServletConfig();

    while(e.hasMoreElements()){
        String name = e.nextElement().toString();
        System.out.println(conf.getInitParameter(name));
    }

    //Ici, on récupère le contexte de l'application
    ServletContext context = getServletContext();
    // Ou avec conf.getServletContext();

    //Nous récupérons la liste de paramètres
    e = context.getInitParameterNames();

    while(e.hasMoreElements()){
        String name = e.nextElement().toString();
        //On appelle la méthode getInitParameter(String name)
        //afin de récupérer la valeur
        System.out.println(context.getInitParameter(name));
    }
}

```

3. Gestion des Cookies

C'est une information (*couple nom/valeur*) généralement écrite par un serveur sur le poste du client le consultant.

Initialement, ils sont prévus pour pallier le problème du travail hors connexion via le protocole HTTP. Ils permettent d'authentifier de manière unique un client pour connaître les pages qu'il a déjà visitées. On simule ainsi la notion de session. Ce concept actuellement est fortement utilisé à but marketing.

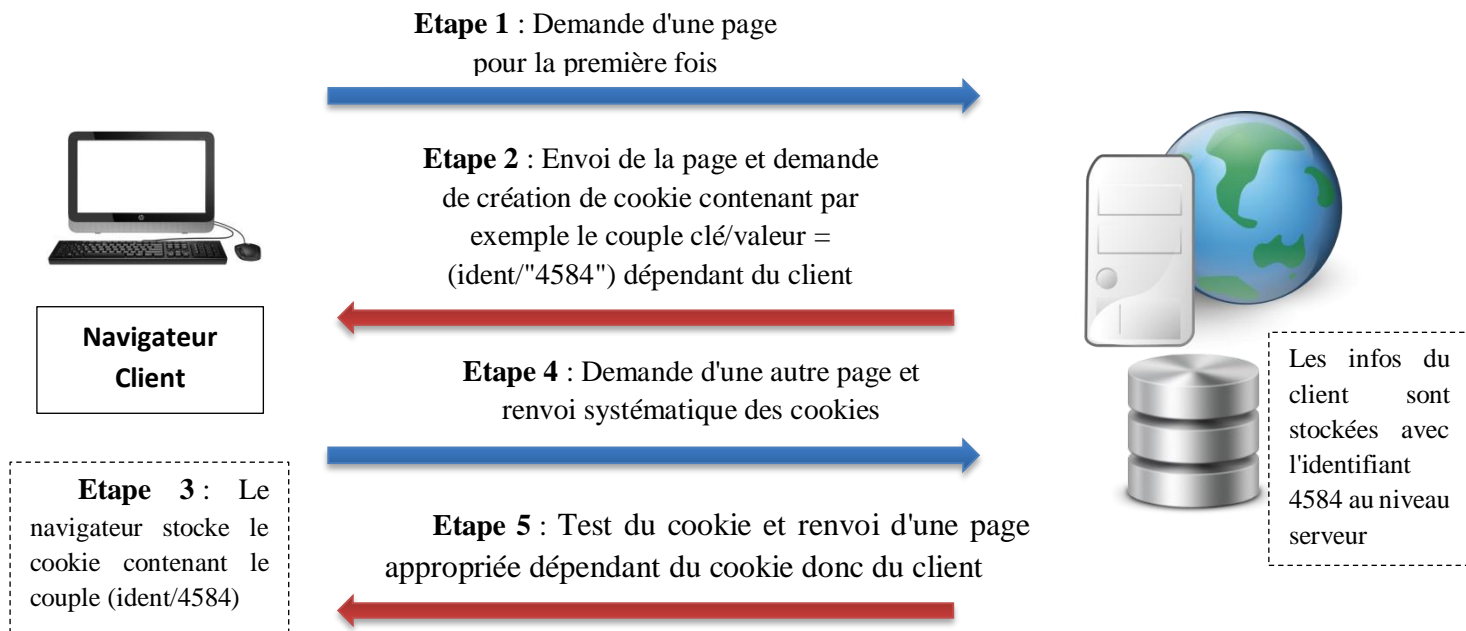
C'est le protocole HTTP qui embarque dans ses en-têtes la demande de création et la valeur d'un cookie. Le système de stockage au niveau client dépend du navigateur utilisé. Le schéma de la figure ci-dessous explique le mécanisme d'utilisation des cookies entre le client et le serveur.

C'est le protocole HTTP qui embarque dans ses en-têtes la demande de création et la valeur d'un cookie. Le système de stockage est laissé au navigateur.

Le cookie est géré en Java sous la forme d'une instance de la classe Cookie.

Deux points sont à retenir:

- L'utilisateur peut refuser au niveau de son navigateur, la création de cookies. Vous ne devez donc pas baser la gestion des sessions sur ce concept.
- Si vous utilisez malgré tout des cookies, prenez soin de ne pas y stocker des informations confidentielles sur l'utilisateur



11.1. Quelques méthodes de manipulation des cookies :

- Classe Cookie

String getName ()	Lire le nom d'un cookie.
String getValue () void SetValue (valeur)	Définir et lire la valeur d'un cookie
int getMaxAge() void setMaxAge(dVie)	Définir et lire la durée de vie en secondes. Si dVie > 0, le cookie sera stocké sur disque. Si dVie < 0, (valeur par défaut) le cookie sera stocké en mémoire jusqu'à ce que le client quitte le navigateur (il s'agit d'un cookie de session) Si délai = 0, le cookie sera supprimé.

- Interface HttpServletResponse

addCookie (cookie) Ajoute le cookie à la réponse.

- Interface HttpServletRequest

Cookie [] get_cookies () Récupère un tableau contenant tous les cookies envoyés par le client

- Modification d'un cookie

Pour modifier un cookie on peut suivre la procédure suivante :

- Récupérer tous les cookies .
- Récupérer le cookie désiré (par son nom)
- Le modifier (par les méthodes de Cookie)
- L'insérer dans la réponse

- Exemples :

1. Implémentons une classe *CookiesUtils* qui permet de :
 - ✓ Récupérer un cookie à partir de son nom
 - ✓ Récupérer la valeur d'un cookie à partir de son nom
2. Ecrire une Servlet EnsahServlet1 qui insère un cookie dans la réponse
3. Ecrire une Servlet EnsahServlet2 qui récupère le cookie
4. Ecrire une Servlet EnsahServlet3 qui récupère et modifie le cookie

- La classe CookiesUtils :

```
public class CookiesUtils {  
  
    /** Récupération d'un Cookie à partir de son nom */  
  
    public static Cookie getCookie(String pName, Cookie[] pCookies) {  
  
        if (pName == null || pCookies == null)  
            return null;  
        for (int i = 0; i < pCookies.length; i++) {  
            if (pCookies[i].getName().equals(pName))  
                return pCookies[i];  
        }  
        return null;  
    }  
  
    /** Récupération de la valeur d'un Cookie à partir de son nom */
```

```

    public static String getCookieValue(String pName, Cookie[] pCookies) {

        if (pName == null || pCookies == null)
            return null;
        for (int i = 0; i < pCookies.length; i++) {
            if (pCookies[i].getName().equals(pName))
                return pCookies[i].getValue();
        }
        return null;
    }
}

```

Définition des Servlets dans le fichier web.xml

```

<servlet>
    <description>
    </description>
    <display-name>EnsahServlet1</display-name>
    <servlet-name>EnsahServlet1</servlet-name>
    <servlet-class>
        com.ensahapp.web.EnsahServlet1</servlet-class>
</servlet>
<servlet>
    <description>
    </description>
    <display-name>EnsahServlet2</display-name>
    <servlet-name>EnsahServlet2</servlet-name>
    <servlet-class>
        com.ensahapp.web.EnsahServlet2</servlet-class>
</servlet>

<servlet>
    <description>
    </description>
    <display-name>EnsahServlet3</display-name>
    <servlet-name>EnsahServlet3</servlet-name>
    <servlet-class>
        com.ensahapp.web.EnsahServlet3</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>EnsahServlet1</servlet-name>
    <url-pattern>/EnsahServlet1</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>EnsahServlet2</servlet-name>
    <url-pattern>/EnsahServlet2</url-pattern>
</servlet-mapping>

    <servlet-mapping>
    <servlet-name>EnsahServlet3</servlet-name>
    <url-pattern>/EnsahServlet3</url-pattern>
</servlet-mapping>

```

- La classe EnsahServlet1 :

```

public class EnsahServlet1 extends HttpServlet {

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

        response.addCookie(new Cookie("user", "Tarik"));

    }

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

    }

}

```

- La classe EnsahServlet2 :

```

public class EnsahServlet2 extends HttpServlet {

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

        Cookie[] cookies = request.getCookies();
        Cookie cookie = CookiesUtils.getCookie("user", cookies);
        if (cookie != null)
            System.out.println(cookie.getValue());

    }

    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException,
        IOException {
        doGet(request, response);
    }

}

```

- La classe EnsahServlet3 :

```

public class EnsahServlet3 extends HttpServlet {

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

        Cookie[] cookies = request.getCookies();
        Cookie cookie = CookiesUtils.getCookie("user", cookies);
        if (cookie != null) {

            cookie.setValue("bousata");
            response.addCookie(cookie);

        }

    }

}

```

```

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

        }

}

```

4. Gestion de la session :

4.1. Techniques classiques de gestion de session :

Le protocole HTTP est un protocole sans état. Chaque fois qu'un client ouvre une page Web, il obtient automatiquement une connexion au serveur Web. Il n'existe pas de solution intégrée permettant de conserver des informations contextuelles sur le client, ce qui crée des difficultés pour gérer certains types de gestion. La gestion du panier dans une application e-commerce fait partie des cas d'école classiques. Lorsque le client ajoute un produit dans son panier, comment le serveur peut-il savoir ce que contient déjà le panier et à quel client il appartient ?

▪ Solution classique :

Une Hashtable par client stockée dans une Hashtable globale et identifiée par un numéro de session associé au client.

La gestion des Hashtables et la génération d'un identifiant unique pour chaque session sont des détails assez techniques et fastidieux.

Lors de la première requête client :

```

//Création d'un Numéro de session unique pour le client
String sessionId = uniqueString( );

//Création d'une Hashtable pour la session
Hashtable sessionInfo= new Hashtable( );

//Récupération de appliTable (hashtable de l'application)
Hashtable appliTable = trouverAppliTable( );

//stockage de sessionInfo identifié par sessionId dans appliTable
appliTable.put(sessionId,sessionInfo );

```

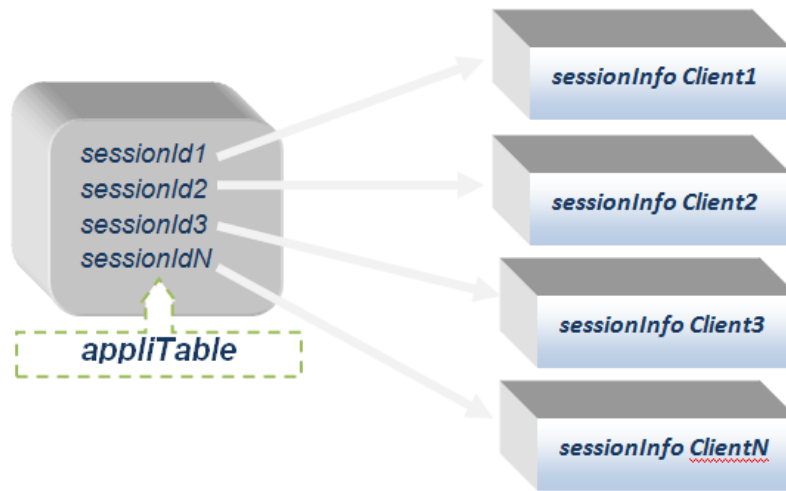
Pour les requêtes suivantes, la servlet se sert du Hashtable appliTable global à toutes les sessions pour retrouver le Hashtable sessionInfo. Pour cela, le serveur doit récupérer le sessionId du client.

```

//Récupération de sessionId du client
String sessionId = récupération...

//Récupération de la Hashtable du client
Hashtable sessionInfo = (Hashtable) appliTable.get(sessionId);

```



Il existe ensuite 3 solutions classiques pour échanger le sessionId entre le server et le client

- **Les cookies :**

```
//Création d'un cookie contenant sessionId
Cookie sessionCookie= new Cookie( "SESSIONID",sessionId) ;

//Stockage du cookie sur le client
response.addCookie(sessionCookie) ;
```

Inconvénient : bien que peu probable, le client peut refuser les cookies.

- **La réécriture d'url :**

Le client ajoute des données supplémentaires à la fin de chaque URL. Ces données identifient la session, et le serveur associe cet identificateur aux données enregistrées pendant la session.

*Http://host:8080/ensah/UneServlet;**jsessionID=1234***

Avantage : fonctionne même si le client refuse les cookies.

Inconvénient : toutes les pages contenant des urls participant à la session doivent être générées dynamiquement.

- **Les champs cachés de formulaires :**

```
<input type="hidden" name ="sessionID" value="1234">
```

Principal inconvénient : Tous les accès serveur doivent se faire via un formulaire.

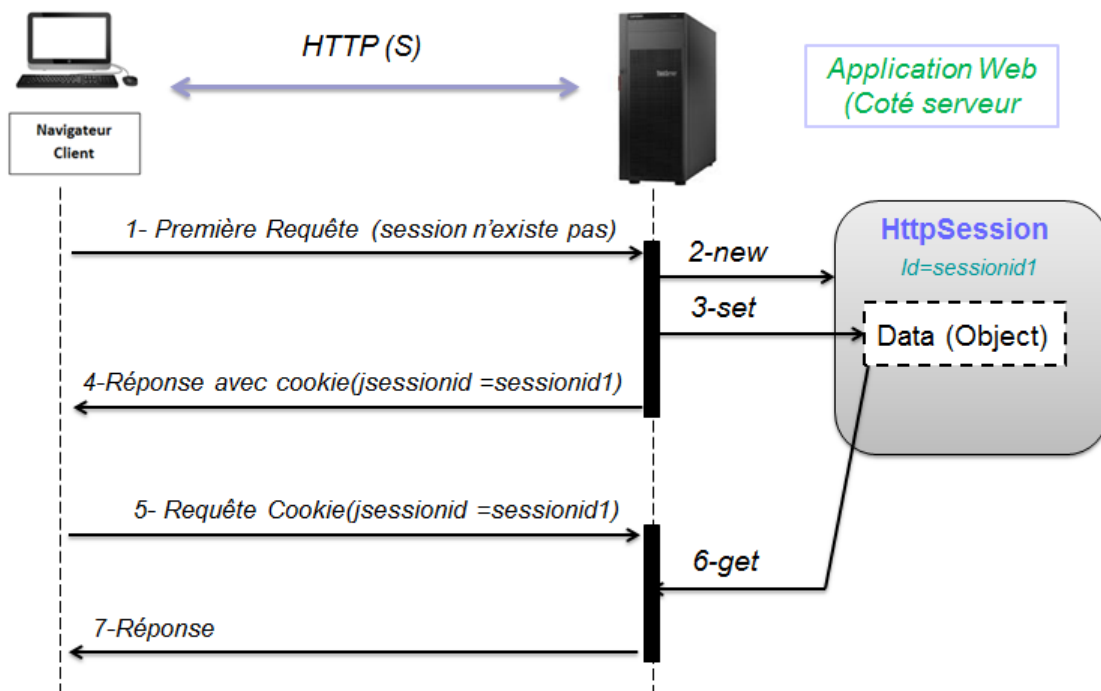
4.2. Gestion de la session avec l'API HttpSession

L'interface HttpSession permet de définir la notion de session, qui n'existe pas en HTTP. La gestion d'une session dans le standard Servlet peut se faire de deux manières. La manière la plus simple est de créer un cookie de session, envoyé au client, qu'il retournera dans l'en-tête HTTP à

chaque requête. La valeur de ce cookie est un code de hachage, qui identifie un internaute de façon unique. Le serveur conserve la trace de ces cookies, et il les associe aux bons utilisateurs.

Si le navigateur client refuse les cookies, alors ce code de hachage est ajouté à l'URL de requête dans le cas de requête par la méthode GET, et aux paramètres internes pour les requêtes de type POST.

La figure suivante montre le processus de gestion d'une session et l'échange de l'identifiant de session entre le serveur et le client.



Les objets qui matérialise une session sont stockés dans le serveur et possèdent une structure de données dans laquelle on peut stocker des objets quelconques identifiés par des clefs (instances de String). Chaque requête entrante est associée à une session active créée par le client (le navigateur). La session est détruite s'il n'y a pas de nouvelles requêtes dans une durée configurable au niveau du serveur.

- Créer et/ou récupérer une session avec l'interface `HttpServletRequest`

<code>HttpSession getSession (boolean create)</code>	Si create = true renvoie la session associée ou la crée. Pour savoir s'il s'agit d'une nouvelle session, utiliser la méthode <code>isNew()</code> sur la session récupérée. Si create = false et qu'il n'y a pas de session en cours cette méthode renvoie null
<code>HttpSession getSession ()</code>	Equivalent à <code>getSession(true)</code> .

- Invalidation de la session avec l'interface `HttpSession`:

`void invalidate ()` Invalide la session et libère tous les objets qui y sont stockés.

- **Récupérer et stocker des informations dans la session avec l'interface HttpSession**

Object <i>getAttribute (String name)</i>	Renvoie l'objet enregistré sous l'identifiant name. Renvoie null si non trouvé. Cette méthode renvoyant une référence d'Object, il faut, en principe, la retyper (cast) vers une classe dérivée.
Enumeration <i>getAttributeNames ()</i>	Renvoie une énumération sur tous les identifiants
void <i>setAttribute (String name , Object value)</i>	Stocke l'objet value avec l'identifiant name
void <i>removeAttribute (String name)</i>	Retire l'objet identifié par name
Enumeration <i>getAttributeNames ()</i>	Renvoie une énumération sur tous les identifiants.

5. Portée des objets (*Objects scope*) dans une application Java Web :

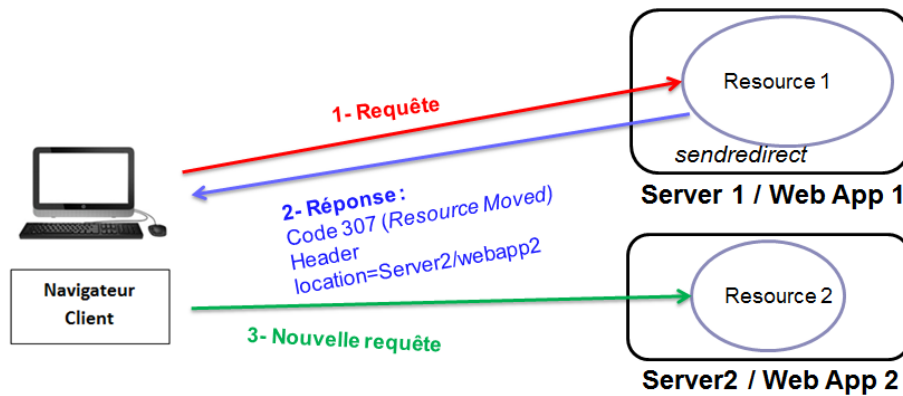
Dans une application Java Web Il existe plusieurs visibilités (durée de vie) pour les objets Java :

- **Page** (pour JSP seulement) : les objets dans cette portée sont uniquement accessibles dans la page JSP en question ;
- **Requête** (Request): les objets dans cette portée sont uniquement accessibles durant l'existence de la requête en cours ;
- **Session** : les objets dans cette portée sont accessibles durant l'existence de la session en cours ;
- **Application** : les objets dans cette portée sont accessibles durant toute l'existence de l'application.

Chapitre 3 : Redirection et Filtres

1. La redirection au niveau du client :

la méthode *sendRedirect(String urlCible)* (méthode de l'objet *HttpServletResponse*): est à utiliser pour rediriger vers une adresse située sur un autre serveur ou une autre web app. On parle de redirection au niveau du client



- Exemple :

```

public class EnsahServlet4 extends HttpServlet {
protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    String cibleURL = "http://perso.menara.ma/tarik/";

    //indiquer la redirection au client
    response.sendRedirect(cibleURL);
}

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

}

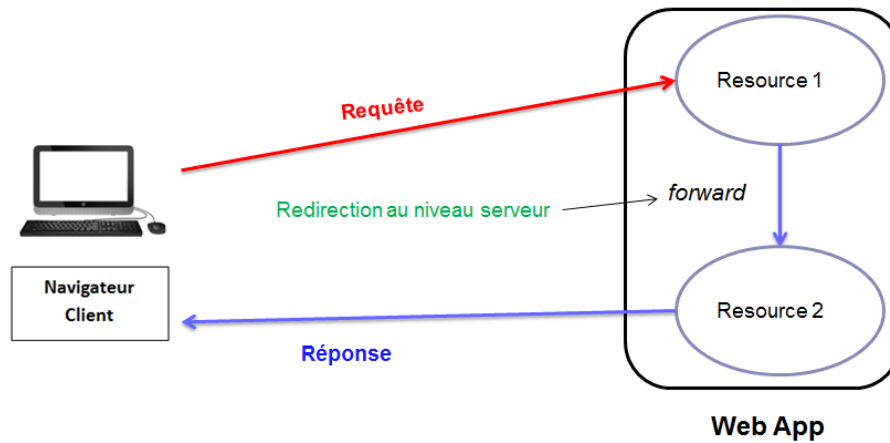
}

```

2. La redirection au niveau du serveur

Dans ce cas la servlet se contente de jouer le rôle de contrôleur et va donc chaîner vers d'autres servlets et/ou JSP en fonction des paramètres récupérés dans sa requête pour leur confier le rôle de construction de la réponse.

Il faut commencer par récupérer un **RequestDispatcher** auprès du contexte de la servlet puis confier à cet objet le soin de chaîner (en appelant **forward**) ou d'inclure (en appelant **include**) le résultat d'une autre servlet ou JSP en fournissant son url. Les objets requête et réponse doivent, bien sûr, être transmis.



- Exemple 1: Redirection vers une autre Servlet

```

public class EnsahServlet4 extends HttpServlet {

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

        String urlCible = "/EnsahServlet2";
        RequestDispatcher dispatcher =
getServletContext().getRequestDispatcher(urlCible);
        dispatcher.forward(request, response);

    }

    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException,
        IOException {
        doGet(request, response);
    }

}
  
```

- Exemple 2: Redirection vers une page JSP

```

public class EnsahServlet4 extends HttpServlet {

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

        String urlCible = "/pages/ensah.jsp";
        RequestDispatcher dispatcher =
getServletContext().getRequestDispatcher(urlCible);
        dispatcher.forward(request, response);

    }

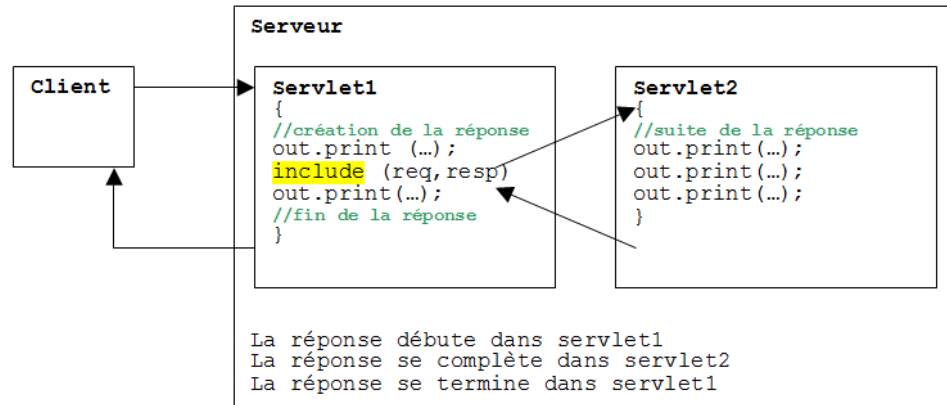
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException,
        IOException {
        doGet(request, response);
    }

}
  
```

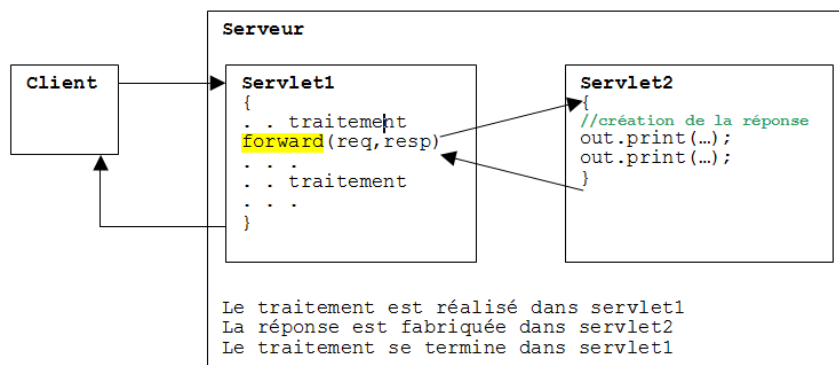
3. Différences essentielles à connaître:

3.1. Forward Vs Include

Forward : Séparer le traitement de la requête du client (effectué sur la servlet appelante) de la génération de la réponse (effectuée sur la servlet appelée). Ou répartir le traitement de la requête sur plusieurs servlets.



Include : Répartir la génération de la réponse sur plusieurs servlets.



3.2. Forward Vs sendRedirect

Forward()	SendRedirect()
La requête est transférée vers une autre ressource du même serveur	La requête est transférée vers une autre ressource vers un domaine ou un serveur différent
Le conteneur de servlet gère le process des redirections en interne, le client ne sera pas au courant des redirections effectuées au niveau serveur. On ne peut pas voir la nouvelle cible au niveau de la barre d'adresse du navigateur par exemple.	Le conteneur redirige le client vers une autre ressource en lui indiquant cette ressource dans la réponse, le client est donc au courant de la redirection et c'est lui qui envoie la nouvelle requête vers la nouvelle cible. On peut voir la nouvelle adresse au niveau de la barre d'adresse
On passe les objets request et response à la méthode forward de telle sorte que la nouvelle cible aura accès à ces mêmes objets. La servlet source peut insérer des données dans l'objet request avec la méthode setAttribute, et la servlet cible peut les récupérer avec la méthode getAttribute	Dans ce cas la nouvelle cible n'a pas accès aux anciens objets request et response

4. Encodage des URL envoyés au client

Puisque le suivi de session pouvant utiliser la réécriture d'URL (si les cookies ne sont pas acceptés) il faut prévoir l'encodage des URL pour ajouter automatiquement l'id de session dans les URL, pratiquement il faut faire l'encodage des URL dans deux cas :

- les URL sont intégrées dans une page générée par une servlet ;
- lors de la redirection vers une URL de son propre site avec `sendRedirect` ;

- **Exemples :**

- Lorsque l'URL est intégrée à la page Web générée par la servlet.

```
String originalURL = "/gestionsession";
String encodeURL= response.encodeURL(originalUrl);
out.println("<form action=\""+encodeURL+ "\" method=\"post\">");
```

- Lors d'une redirection au niveau du client par un `sendRedirect` et que la page destination participe à la session

```
String originalURL = "/identification";
String encodeURL= response.encodeRedirectURL(originalUrl);
response.sendRedirect(encodeURL) ;
```

Si on ne peut pas imposer l'usage des cookies de session aux clients, alors il faut utiliser systématiquement l'encodage d'URL.

5. Implémentation des filtres dans l'API Servlet de Java EE

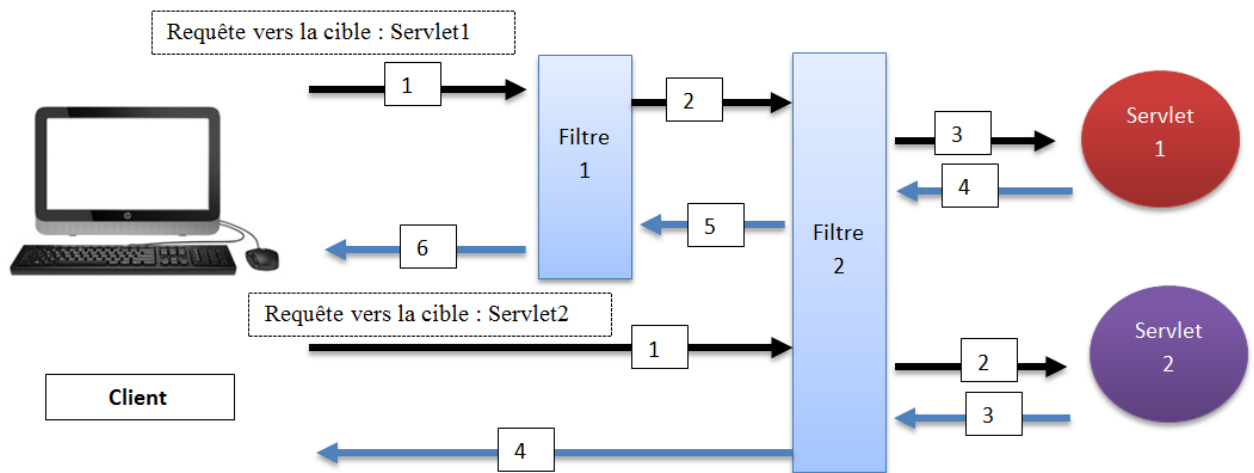
Dans l'API Servlet de Java EE les filtres sont des objets qui peuvent transformer une requête ou modifier une réponse, ils sont basés sur le design pattern décrit dans la section précédente. Les filtres ne créent pas une réponse comme les servlets. Ce sont des préprocesseurs de la requête avant qu'elle n'atteigne une servlet (ou plusieurs servlets) et/ou des postprocesseurs de la réponse après l'exécution une servlet.

Un même filtre peut être associé à une ou plusieurs ressources Web (servlet ou toute autre ressource sur le serveur), autrement on peut associer plusieurs filtres à une même ressource.

En Anglais : *Filters are components that can be used to pre process the request and post process the response. Based on the Interceptor Design Pattern, filters are useful in separating the cross cutting concerns like Security, logging, ...*

You can map a filter to one or more Web resources, and you can map more than one filter to a Web resource.

Dans la figure ci-dessous la servlet 1 est associée à deux filtres Filtre 1 et Filtre 2. La servlet 2 est associée à un seul filtre qu'est le Filtre 2. Le filtre 2 est utilisé avec la servlet 1 et la servlet 2. Les numéros indiqués sur la figure montrent la chronologie d'exécution des filtres et servlets



Un filtre peut faire les choses suivantes :

- Intercepter une invocation de servlet avant qu'elle ne soit appelée ;
- Examiner une requête avant l'appel d'une servlet ;
- Intercepter l'invocation d'une servlet après qu'elle a été appelée
- Modifier les en-têtes et les données de la requête en fournissant une version personnalisée de l'objet de requête (l'objet de type `HttpServletRequest`) qui enveloppe la requête réelle.
- Modifier les en-têtes et les données de réponse en fournissant une version personnalisée de l'objet de réponse (l'objet de type `HttpServletResponse`) qui enveloppe la réponse réelle.

Un filtre peut être configuré (dans le fichier `web.xml`) d'une manière à agir sur une servlet ou un groupe de servlets. Cette servlet ou ce groupe de servlets peuvent être filtrés par zéro ou plusieurs filtres.

Dans la pratique les filtres sont utilisés généralement pour traiter les préoccupations transversales (*the cross cutting concerns*), ainsi parmi leurs utilisations classiques on trouve :

- l'authentification,
- la journalisation (en anglais *logging*),
- La conversion d'image,
- Upload des fichiers,
- La compression et le chiffrement de données,
- La conversion de données,
- Ajout systématique d'en-têtes et de pieds de pages
- etc....

- **Classes et interfaces mises en jeu :**

▪ **Interface Filter**

void destroy()	Méthode appelée à la destruction du filtre.
void doFilter (ServletRequest, ServletResponse, FilterChain)	Méthode appelée lorsqu'un filtre est appelé par le container pour une servlet filtrée. Le filtre enchaînera sur la prochaine entité de la chaîne en appelant <code>doFilter(req,resp)</code> sur le troisième argument.
void init (FilterConfig)	Méthode appelée après l'initialisation du filtre. Permet de récupérer les paramètres de configuration du filtre.

▪ **Interface FilterConfig**

String getFilterName()	Retourne le nom du filtre défini dans le descripteur de déploiement
String getInitParameter (String name)	Retourne la valeur d'un paramètre d'initialisation
ServletContext getServletContext()	Retourne le contexte du filtre
Enumeration getInitParameterNames ()	Retourne une énumération sur les noms de paramètres.

▪ **Interface FilterChain**

void doFilter (ServletRequest, ServletResponse)	Invoke le prochain filtre dans la chaîne.
---	---