

05) Les Servlets

420-109-GG

LOTFI DERBALI

Le descripteur de déploiement web.xml

La déclaration d'une servlet:

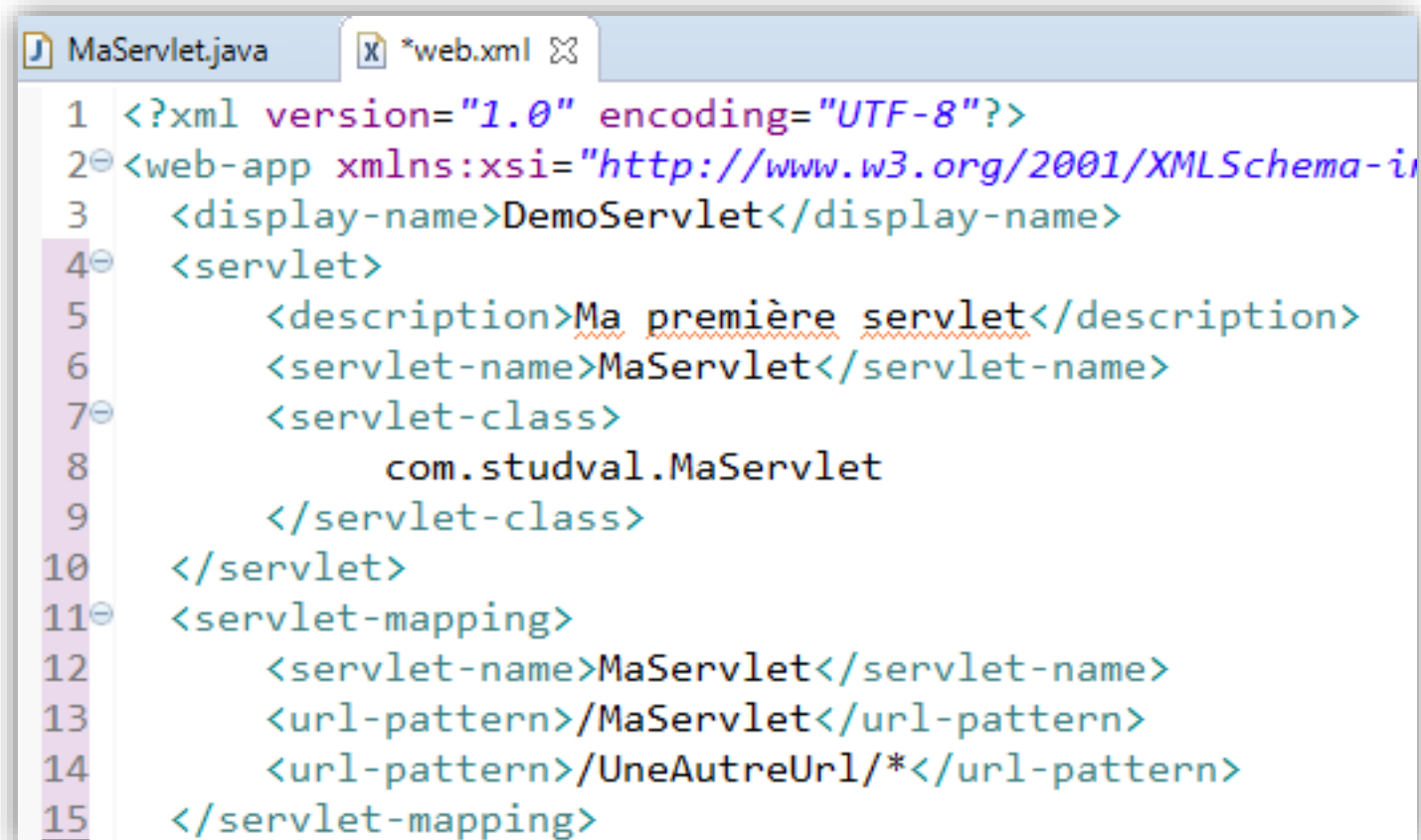
- Pour utiliser une servlet dans une application, il faut réaliser deux opérations :
 - Déclarer l'existence de la servlet.
 - Déclarer les URL pour lesquelles la servlet est utilisée.
- Ces deux opérations peuvent se faire dans **le descripteur de déploiement** ou par **annotations**.
- **Eclipse propose un paramétrage automatique par annotations** à partir de la version 3.0 de la spécification des servlets.

Le descripteur de déploiement web.xml

La déclaration de l'existence de la servlet se fait avec l'aide de la balise **<servlet>**.

- Cette balise accueille plusieurs sous-balises permettant de décrire la servlet.
- Par exemple:
 - **<description>** : cette balise permet de décrire succinctement le rôle de la servlet. C'est l'équivalent d'une Javadoc.
 - **<servlet-name>** : cette balise permet de donner un nom logique à la servlet. Ce nom doit être unique.
 - **<servlet-class>** : cette balise permet de déclarer la classe correspondant à la servlet.
- La déclaration des URL associées se fait à l'aide de la balise **<servlet-mapping>**. Cette balise accueille les sous-balises suivantes :
 - **<servlet-name>** : cette balise permet de faire référence à la balise **<servlet>** contenant une balise **<servlet-name>** avec la même valeur.
 - **<url-pattern>** : cette balise peut être présentée plusieurs fois si la servlet doit être associée à plusieurs URL. Le pattern commence par un slash (/). Attention, l'URL est sensible à la casse. La séquence **/*** est une séquence joker que l'on peut placer à la fin d'un pattern pour indiquer que la servlet est associée à toutes les URL commençant par **/UneAutreUrl/**.

Le descripteur de déploiement web.xml



The image shows a code editor window with two tabs: 'MaServlet.java' and '*web.xml'. The 'web.xml' tab is active, displaying the following XML code:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-in
3   <display-name>DemoServlet</display-name>
4   <servlet>
5     <description>Ma première servlet</description>
6     <servlet-name>MaServlet</servlet-name>
7     <servlet-class>
8       com.studval.MaServlet
9     </servlet-class>
10  </servlet>
11  <servlet-mapping>
12    <servlet-name>MaServlet</servlet-name>
13    <url-pattern>/MaServlet</url-pattern>
14    <url-pattern>/UneAutreUrl/*</url-pattern>
15  </servlet-mapping>
```

Paramétrage par annotations

Le paramétrage par annotations remplace le paramétrage décrit dans le fichier web.xml.

Ce paramétrage est fait directement sur la classe constituant la servlet.

L'annotation nécessaire est **@WebServlet**.

Cette annotation prend plusieurs paramètres. Les paramètres nécessaires pour obtenir un paramétrage équivalent au paramétrage précédent sont les suivants :

- **description** : ce paramètre est l'équivalent de la balise <description>.
- **name** : ce paramètre est l'équivalent de la balise <servlet-name>.
- **value** : ce paramètre est l'équivalent des balises <url-pattern>. Ce paramètre attend un tableau de patterns.

```
14 @WebServlet(description="Ma première servlet",  
15             name="MaServlet",  
16             value={"/MaServlet", "/UneAutreUrl/*"})  
17 public class MaServlet extends HttpServlet {  
18
```

Les paramètres d'initialisation d'une servlet

Paramétrage dans le descripteur de déploiement

La balise `<servlet>` accepte une sous-balise complémentaire `<init-param>`.

Cette balise peut être présente plusieurs fois en cas de paramètres multiples. La balise `<init-param>` dispose de trois sous-balises :

- `<description>` : cette balise permet de décrire le paramètre.
- `<param-name>` : cette balise permet de nommer le paramètre.
- `<param-value>` : cette balise permet de valoriser le paramètre.

```
<servlet>
  <description>Ma première servlet</description>
  <servlet-name>MaServlet</servlet-name>
  <servlet-class>
    com.studval.MaServlet
  </servlet-class>
  <init-param>
    <description>L'auteur de la servlet</description>
    <param-name>auteur</param-name>
    <param-value>Lotfi</param-value>
  </init-param>
</servlet>
```

Paramétrage par annotations

Il est possible de faire ce paramétrage au travers de l'annotation `@WebServlet` avec le paramètre `initParams`.

Ce paramètre attend un tableau d'annotations `@WebInitParam`. Cette annotation possède trois paramètres `description`, `name` et `value` pour définir un paramètre.

```
15 @WebServlet(description="Ma première servlet",
16             name="MaServlet",
17             value={"/MaServlet", "/UneAutreUrl/*"},
18             initParams={@WebInitParam(description="L'auteur de la servlet",
19                                     name="auteur", value="Lotfi")})
20 public class MaServlet extends HttpServlet {
```

Utilisation des paramètre d'initialisation dans la servlet

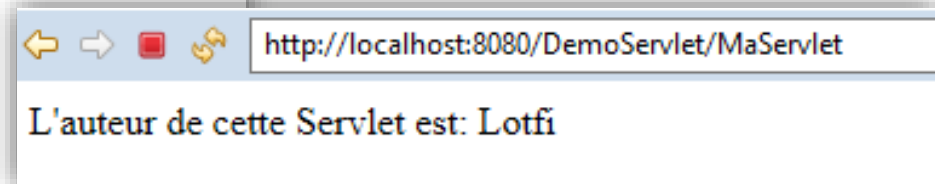
La servlet possède deux méthodes permettant d'exploiter les paramètres d'initialisation d'une servlet :

- **getInitParameter(String name)** : cette méthode prend en paramètre le nom d'un paramètre et retourne la valeur sous la forme d'une chaîne de caractères.
- **getInitParameterNames()** : cette méthode retourne une collection de chaînes de caractères correspondant au nom des paramètres d'initialisation de la servlet.

La lecture des paramètres d'initialisation peut être faite uniquement lors de l'instanciation de la servlet. L'endroit privilégié est la méthode **init()** car elle est automatiquement appelée à la création de la servlet.

Utilisation des paramètre d'initialisation dans la servlet

```
15@WebServlet(description = "Ma première servlet", name = "MaServlet", value = { "/MaServlet",  
16     "/UneAutreUrl/*" }, initParams = {  
17     @WebInitParam(description = "L'auteur de la servlet", name = "auteur", value = "Lotfi")  
18 public class MaServlet extends HttpServlet {  
19  
20     private static final long serialVersionUID = 1L;  
21  
22     // Attention les variables membres sont partagées par tous les threads  
23     private String auteur;  
24  
25     public MaServlet() {  
26  
27     }  
28  
29     @Override  
30     public void init() throws ServletException {  
31         this.auteur = this.getInitParameter("auteur");  
32     }  
33  
34     protected void doGet(HttpServletRequest request, HttpServletResponse response)  
35         throws ServletException, IOException {  
36         // TODO Auto-generated method stub  
37         response.getWriter().println("L'auteur de cette Servlet est: " + this.auteur);  
38     }
```



Les différents contextes

Lorsqu'une requête est prise en charge par une servlet, le traitement peut demander l'usage d'informations se situant dans différents contextes.

- Le premier contexte est l'application. Les informations définies au niveau de l'application sont accessibles pour traiter toutes les requêtes de tous les clients. Les informations présentes au niveau de l'application ne doivent jamais être spécifiques à un client.
- Le second contexte est la session. Les informations contenues dans la session sont accessibles pour traiter toutes les requêtes provenant du même client.
- Le troisième contexte est la requête. Les informations de la requête sont utilisées pour apporter une réponse adaptée au client. Les informations saisies dans un formulaire sont des informations liées à une requête.

Les différents contextes

1. Le contexte d'application

Dans une servlet, le contexte d'application est accessible au travers de la méthode **getServletContext()**

- Cette méthode retourne un objet de type **ServletContext**. Voici les méthodes les plus utilisées de cette interface :
 - **getContextPath()** : cette méthode retourne une chaîne de caractères correspondant à l'URL permettant d'accéder à l'application.
 - **getInitParameter(...)** : cette méthode permet de lire la valeur d'un paramètre lié à l'application. Ce paramètre est défini au niveau du fichier web.xml dans une balise **<context-param>** comme le montre l'exemple suivant :

```
<context-param>
  <param-name>nomSociete</param-name>
  <param-value>Complexe Sportif SA</param-value>
</context-param>
```

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

```
    String name = getServletContext().getInitParameter("nomSociete");
    response.getWriter().println(name + "<br>");
```

- **getAttribute(...)** et **setAttribute(...)** : ces méthodes permettent de lire et d'écrire un attribut au niveau du contexte d'application.
- **getNamedDispatcher(...)** : cette méthode retourne un objet de type **RequestDispatcher** permettant d'accéder à une autre servlet ou une JSP.

Les différents contextes

2. Le contexte de session

Dans une servlet, la session est accessible au travers de la méthode `getSession()` de l'objet de type `HttpServletRequest` passé en paramètre des méthodes `doXXX()`.

3. Le contexte de requête

Dans une servlet, l'objet de type `HttpServletRequest` passé en paramètre des méthodes `doXXX()` permet **d'accéder aux informations de la requête**.

La lecture de la requête

1 . Lecture des informations de l'URL

- **getScheme()** : cette méthode retourne une chaîne de caractères correspondant au protocole réellement utilisé (http, https).
- **getServerName()** : cette méthode retourne le nom d'hôte de la machine vers laquelle la requête a été émise. Le nom d'hôte correspond à l'information située sur l'en-tête de requête Host sans prendre en compte le port si celui-ci est renseigné. Le nom d'hôte peut être un nom de domaine, un nom de machine ou une adresse IP.
- **getServerPort()** : cette méthode retourne le port utilisé par la requête HTTP.
- **getContextPath()** : cette méthode retourne le nom de l'application. Dans un environnement de développement, le nom de l'application correspond au nom du projet.
- **getServletPath()** : cette méthode retourne le chemin d'accès à la servlet. Cette information correspond à un des patterns d'URL définis dans la déclaration de la servlet (dans le fichier web.xml ou dans l'annotation @WebServlet).
- **getQueryString()** : cette méthode retourne une chaîne de caractères correspondant à tous les paramètres passés à la requête HTTP.
- **getRemoteAddr()** : cette méthode retourne sous la forme d'une chaîne de caractères l'adresse IP de l'émetteur de la requête HTTP.
- **getRemoteHost()** : cette méthode retourne sous la forme d'une chaîne de caractères le nom de l'hôte de l'émetteur de la requête HTTP.
- **getLocalAddr()** : cette méthode retourne sous la forme d'une chaîne de caractères l'adresse IP de la carte réseau du serveur par laquelle la requête HTTP a été reçue.
- **getLocalName()** : cette méthode à l'instar de la méthode précédente retourne sous la forme d'une chaîne de caractères le nom d'hôte de la carte réseau du serveur par laquelle la requête HTTP a été reçue.
- **getLocalPort()** : cette méthode retourne sous la forme d'un entier le port sur lequel a été reçue la requête HTTP.

La lecture de la requête

2. Lecture de l'en-tête de la requête

- **getCharacterEncoding()** : cette méthode retourne l'encodage utilisé pour le corps de la requête ou null si la requête ne définit pas d'encodage.
- **getContentLength()** et **getContentLengthLong()** : ces méthodes retournent la taille en octet des informations situées dans le corps de la requête ou -1 si la taille n'est pas connue ou supérieure à Integer.MAX_VALUE. Cela correspond à l'en-tête de requête Content-Length.
- **getContentType()** : cette méthode retourne le type de média (MIME) du corps de la requête ou null s'il n'y a pas de corps ou si le type n'est pas connu. Cela correspond à l'en-tête de requête Content-Type.
- **getLocale()** : cette méthode retourne un objet de type java.util.Locale. Il correspond à la Locale préférée désignée dans l'en-tête de requête Accept-Language. Pour obtenir l'ensemble des locales paramétrées, il faut utiliser la méthode getLocales() qui retourne une collection de Locale sous la forme d'un objet de type Enumeration<Locale>. Ces informations sont modifiables par l'utilisateur en paramétrant les langues à utiliser au niveau du navigateur. Certains frameworks, comme Struts 2, peuvent se baser sur cet attribut pour gérer l'internationalisation de l'application.
- **getMethod()** : cette méthode retourne la méthode utilisée par la requête (GET, POST...)
- **getHeader(String name)** : cette méthode retourne la valeur de l'en-tête passé en paramètre sous la forme d'une chaîne de caractères.
- **getHeaders(String name)** : cette méthode retourne la valeur de l'en-tête passé en paramètre sous la forme d'une collection de chaînes de caractères. Cela permet de lire facilement les attributs qui possèdent plusieurs valeurs séparées par un point-virgule comme Accept-Language.
- **getIntHeader(String name)** : cette méthode retourne la valeur de l'en-tête passé en paramètre sous la forme d'un int ou -1 si l'en-tête n'est pas présent. Attention, l'en-tête lu doit être compatible avec le type int sous peine d'avoir une exception de type NumberFormatException.
- **getDateHeader(String name)** : cette méthode retourne la valeur de l'en-tête passé en paramètre sous la forme d'un long correspondant au nombre de millisecondes depuis le premier janvier 1970 ou -1 si l'en-tête n'est pas présent. Attention, l'attribut lu doit être compatible avec le type Date sous peine d'avoir une exception de type IllegalArgumentException. Cette méthode est utile pour lire un paramètre comme If-Modified-Since.
- **getHeaderNames()** : cette méthode retourne une collection de l'ensemble des noms des en-têtes présents sur la requête. Ceci peut être utile puisque les navigateurs n'envoient pas tous les mêmes en-têtes.

La lecture de la requête

3. Lecture des paramètres

- **getParameter(String name)** : cette méthode retourne la valeur du paramètre passé en paramètre sous la forme d'une chaîne de caractères. Si le paramètre n'existe pas, cette méthode retourne null. S'il existe plusieurs paramètres avec le même nom, c'est la première valeur qui est retournée.
- **getParameterValues(String name)** : cette méthode est utile pour les paramètres apparaissant plusieurs fois dans la requête. Elle retourne un tableau de chaîne de caractères correspondant aux différentes valeurs.
- **getParameterNames()** : cette méthode retourne une collection de chaînes de caractères correspondant aux noms des paramètres disponibles dans la requête. Un nom de paramètre n'est présent qu'une seule fois dans cette collection même s'il est présent plusieurs fois dans la requête. L'usage de la méthode `getParameterValues(...)` est donc adapté ensuite pour connaître la ou les valeurs associées à ce paramètre. Cette méthode retourne une collection vide si la requête ne possède aucun paramètre.
- **getParameterMap()** : cette méthode permet d'obtenir un objet de type `Map<String,String[]>`. La clé correspond au nom du paramètre, la valeur est un tableau de chaîne de caractères correspondant aux valeurs associées à ce paramètre.

La création de la réponse

1. Écriture de l'en-tête de la réponse

- **setStatus(int sc)** : elle prend en paramètre un entier correspondant au statut souhaité. Pour faciliter son usage, l'interface HttpServletResponse propose un ensemble de constantes de type int correspondant aux différents statuts.
 - HttpServletResponse.SC_OK : cette constante correspond à la valeur 200. Cela permet d'indiquer que tout s'est bien passé dans le traitement de la requête.
 - HttpServletResponse.SC_MOVED_PERMANENTLY : cette constante correspond à la valeur 301. Cela permet d'indiquer que la ressource demandée n'est plus accessible à cette URL mais qu'il faut maintenant utiliser une autre URL.
 - HttpServletResponse.SC_FORBIDDEN : cette constante correspond à la valeur 403. Cela permet d'indiquer que le serveur a refusé de traiter la requête.
 - HttpServletResponse.SC_INTERNAL_SERVER_ERROR : cette constante correspond à la valeur 500. Cela permet d'indiquer qu'une erreur de traitement a eu lieu côté serveur.
- **setCharacterEncoding(String charset)** : cette méthode permet de définir l'encodage utilisé pour l'écriture du corps de la réponse. La valeur attendue est une chaîne de caractères correspondant à un encodage (Character Sets) défini par l'IANA. Cette méthode écrase la valeur potentiellement définie à l'aide des méthodes setContentType(...) et setLocale(...).
- **setContentLength(...)** et **setContentLengthLong(...)** : ces méthodes permettent de définir la taille du corps de la réponse. L'information sera écrite au niveau de l'en-tête Content-Length.
- **setContentType(...)** : cette méthode permet de définir le type de média (type MIME) et éventuellement l'encodage.
- **setHeader(String name, String value)** : cette méthode permet d'appliquer une valeur (value) à un en-tête donné (name).
- **setDateHeader(String name, long date)** : cette méthode est spécialisée dans l'écriture des en-têtes de type Date.
- **setIntHeader(String name, int value)** : cette méthode est spécialisée dans l'écriture des en-têtes de type int.

La création de la réponse

2. Écriture du corps de la réponse

- Pour écrire le corps, il existe deux possibilités :
 - L'utilisation d'un objet de type `java.io.PrintWriter` obtenu par l'appel de la méthode `getWriter()` de l'interface `HttpServletResponse`. Cet objet est à utiliser pour écrire un **contenu textuel**.
 - L'utilisation d'un objet de type `javax.servlet.ServletOutputStream` obtenu par l'appel de la méthode `getOutputStream()` de l'interface `HttpServletResponse`. Cet objet est à utiliser pour écrire un **contenu binaire**.
- L'objet le plus couramment utilisé est du type **PrintWriter**. Pour écrire la réponse, cette classe offre les méthodes classiques suivantes :
 - `append(...)`, `print(...)`, `println(...)`, `printf(...)`, `write(...)` : ces méthodes permettent d'ajouter le contenu passé en paramètre.
 - `flush()` : cette méthode permet de valider le contenu ajouté dans l'objet jusqu'au moment de l'appel de cette méthode. Lorsque les méthodes d'écriture sont utilisées, le contenu est placé dans un tampon en attente qu'il soit rempli pour le pousser vers le destinataire.
 - `close()` : cette méthode permet de fermer le flux. Il n'est alors plus possible d'ajouter du contenu.

Déléguer la création de la réponse

Lorsque sa tâche est terminée, la servlet peut **déléguer** la tâche de fournir une réponse **à une autre ressource**.

Cette ressource peut être une autre servlet (mais c'est souvent une JSP).

Pour que cette délégation soit efficace, il faut pouvoir transférer des informations complémentaires calculées ou obtenues lors du traitement métier vers cette seconde ressource.

Pour cela, il y a deux solutions :

1. La première solution est de **placer ces informations en session** pour qu'elles soient accessibles depuis n'importe quelle requête provenant du même client. Cependant, ce n'est pas l'option la plus intéressante pour la gestion d'informations dont la durée de vie est limitée à la requête.
2. La seconde solution est **d'enrichir la requête de nouvelles informations avant de déléguer la suite du traitement à une nouvelle servlet (ou JSP)** si ces informations ont une durée de vie limitée à la requête. **Cette solution consiste à ajouter des attributs.**

Déléguer la création de la réponse

Ajout d'attributs à la requête

- L'ajout d'un attribut se fait grâce à la méthode `setAttribute(String name, Object o)` de l'interface `HttpServletRequest`. Cette méthode attend deux paramètres à savoir le nom de l'attribut (**name**) et la valeur (**o**) dont le type est indéterminé.

```
//...
Terrain terrainAjoute = new Terrain(codeTerrain, "BETON");
//...
//Ajout d'informations complémentaires pour la ressource s'occupant de l'affichage
request.setAttribute("terrainAjoute", terrainAjoute);
```

- La lecture d'un attribut se fait simplement grâce à la méthode `getAttribute(String name)`. Cette méthode retourne la valeur correspondant à l'attribut dont le nom a été passé en paramètre.

```
Object objetAttribut = request.getAttribute("terrainAjoute");
if(objetAttribut instanceof Terrain)
{
    Terrain terrain = (Terrain)objetAttribut;
    //...
}
```

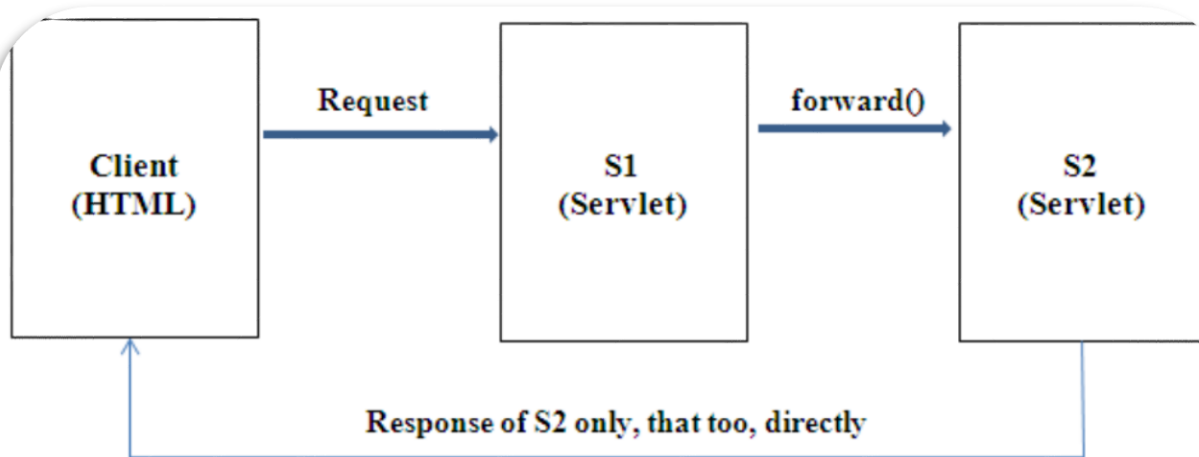
L'interface RequestDispatcher

Cette interface propose les méthodes nécessaires pour manipuler des ressources disponibles sur le serveur (servlets, JSP, fichiers HTML).

- La méthode **forward(...)**
 - prend en paramètre les objets de type `HttpServletRequest` et `HttpServletResponse` de la méthode `doXXX(...)`.
 - Les attributs positionnés sur la requête sont donc bien transmis.
 - La réponse en cours d'élaboration est bien transmise aussi.
 - Cela permet d'écrire la réponse progressivement en transférant la requête vers une autre ressource pour finaliser le traitement.
- Si des traitements complémentaires sont nécessaires par la suite, il est préférable d'utiliser l'inclusion en utilisant la méthode **include(...)**.
 - Cette méthode attend les mêmes paramètres que la méthode `forward(...)`.
 - Cette méthode est adaptée pour inclure des éléments dans le traitement de la servlet à l'origine de l'inclusion.

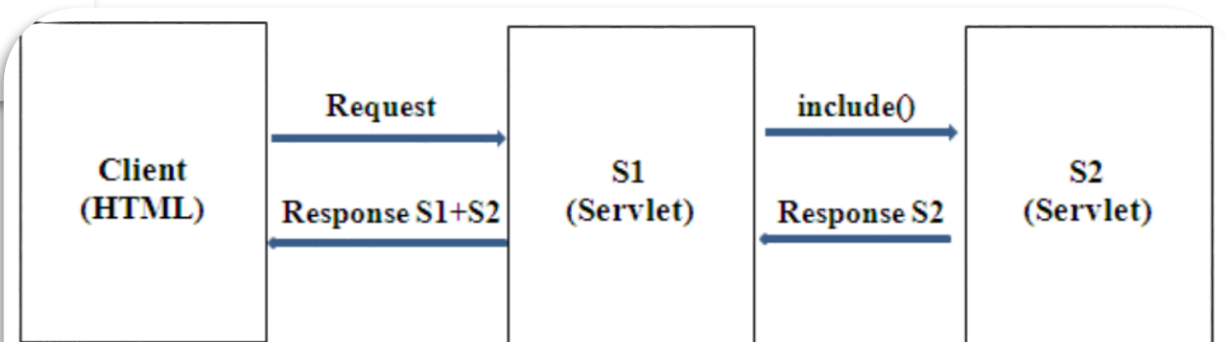
L'interface RequestDispatcher

forward(...)



RequestDispatcher – forward() method

include(...)



RequestDispatcher – include() method

L'interface RequestDispatcher

Avant de pouvoir utiliser les méthodes `forward(...)` et `include(...)`, il faut obtenir un objet de type `RequestDispatcher`. Deux méthodes sont disponibles :

- La méthode `getRequestDispatcher(String path)` de l'interface `HttpServletRequest`. Cette méthode attend un chemin d'accès relatif à la servlet

```
request.setAttribute("test", "Totto");
```

```
RequestDispatcher rd = request.getRequestDispatcher("/Servlet2");  
rd.forward(request, response);
```

```
@WebServlet("/Servlet2")  
public class Servlet2 extends HttpServlet {  
    private static final long serialVersionUID = 1L;  
  
    public Servlet2() {  
    }  
  
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
        // TODO Auto-generated method stub  
        response.getWriter().println((String)request.getAttribute("test"));  
    }  
}
```

- La méthode `getNamedDispatcher(String name)` de l'interface `ServletContext` prend en paramètre le nom de la ressource recherchée telle que définie dans le paramètre **name** de l'annotation `@WebServlet` ou dans la balise `<servlet-name>` du fichier `web.xml`.

L'interface RequestDispatcher

Exemple de la méthode include(...)

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

    RequestDispatcher rd1 = null, rd2 = null;

    // include header content from Header.html
    rd1 = request.getRequestDispatcher("header.html");
    rd1.include(request, response);

    // -----

    // -----//logic to generate main content of web page

    // -----

    // include footer content from Footer.html

    rd2 = request.getRequestDispatcher("footer.html");
    rd2.include(request, response);
}
```

La redirection

La redirection est différente de la délégation décrite précédemment.

Lorsque le mécanisme de redirection est mis en œuvre, le serveur demande au navigateur d'effectuer une nouvelle requête vers un autre URL.

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException {
    // TODO Auto-generated method stub
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    try {
        response.sendRedirect("http://www.google.com");
    } finally {
        out.close();
    }
}
```