

## JEE : Servlet

**Mohammed OUANAN**

[m.ouanan@umi.ac.ma](mailto:m.ouanan@umi.ac.ma)



## Plan

- 1 [Introduction](#)
- 2 [Structure d'une Servlet](#)
- 3 [Première Servlet avec Eclipse](#)
  - [Routage par annotation](#)
  - [Routage dans web.xml](#)
- 4 [Tester la Servlet](#)
- 5 [Servlet multi-routes](#)
- 6 [Objet HttpServletRequest](#)
- 7 [Paramètres de requête](#)
  - [getParameter\(\)](#)
  - [getParameterValues\(\)](#)
  - [getParameterMap\(\)](#)
- 8 [Rediriger vers une autre Servlet](#)

## Jakarta EE

### Servlet : le cœur d'une application JEE

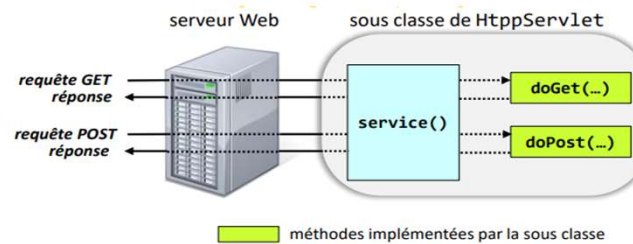
- Une Servlet est un composant web conçu sous la forme d'une classe java qui existe au sein d'une application web et dont la mise en œuvre est gérée par un conteneur web (exemple Tomcat). Une Servlet interagit avec un client web par l'intermédiaire du protocole http, via un mécanisme de requête/réponse.
- Une servlet s'exécute dynamiquement sur le serveur web et permet l'extension des fonctions de ce dernier. Typiquement : accès à des bases de données, transactions d'e-commerce, etc. Une servlet peut être chargée automatiquement lors du démarrage du serveur web ou lors de la première requête du client. Une fois chargées, les servlets restent actives dans l'attente d'autres requêtes du client.
- Lorsqu'une servlet est appelée par un client, la méthode « **service()** » est exécutée. Celle-ci est le principal point d'entrée de toute servlet et accepte deux objets en paramètres : **ServletRequest**, **ServletResponse**

## Jakarta EE

### Servlet : le cœur d'une application JEE

- Classe **Java** héritant de la classe `HttpServlet`
- Recevant une requête **HTTP** (de type GET, POST...) et retournant une réponse **HTTP**
- Contrôleur du modèle MVC dans une application **JEE**

## cycle de vie d'une servlet



### • Cycle de vie géré par le serveur (container)

#### • Initialisation :

- Chargement de la classe (au démarrage ou sur requête).
- Instanciation d'un objet.
- Appel de la méthode `init()` (par exemple : ouverture de lien JDBC)

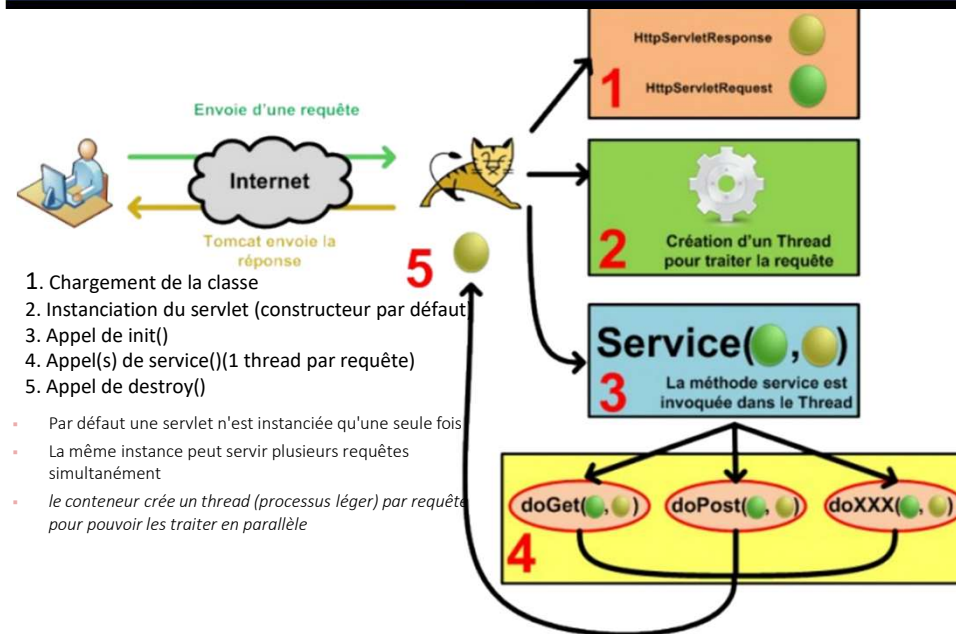
#### • Utilisation :

- Appel de la méthode `service()`
- Dans le cas d'une `HttpServlet` : appel vers `doGet()`, `doPost()`.

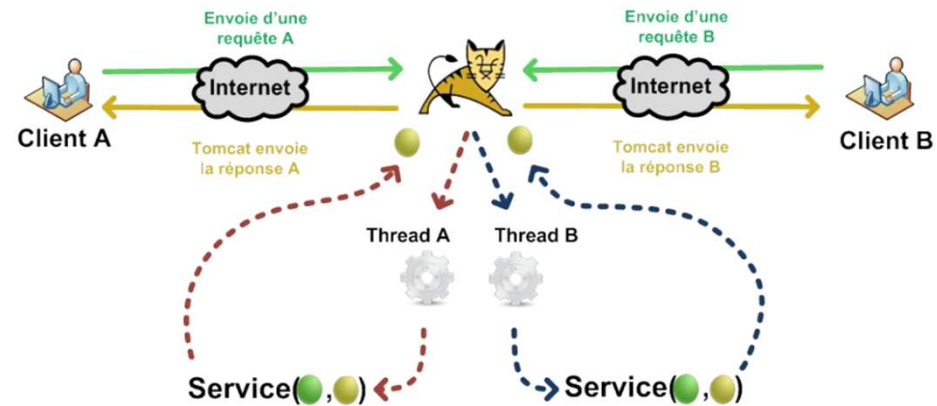
#### • Destruction :

- Peut être provoquée pour optimiser la mémoire
- appel de la méthode `destroy()`

## cycle de vie d'une servlet bien détaillé

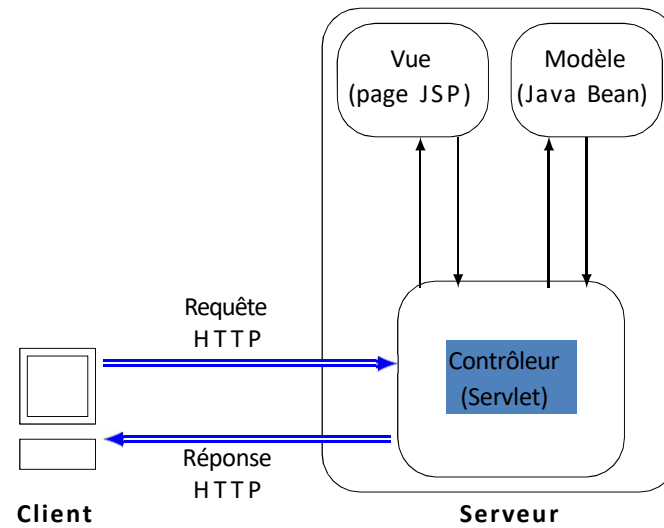


A chaque fois que le conteneur reçoit une requête HTTP, il crée les objets requête et réponse puis, il crée un thread afin d'invoquer la méthode service (HttpServletRequest req, HttpServletResponse rep)... Voici petit schéma résumant ce qu'il se passe lorsque deux personnes envoient une requête vers la même ressource sur le serveur



traitement de deux requêtes par le serveur

## Jakarta EE





## Jakarta EE

**Servlet** : classe Java héritant de `HttpServlet`

```
package org.eclipse.controller;  
  
import javax.servlet.http.HttpServlet;  
  
public class TestServlet extends HttpServlet {  
  
}
```

## Jakarta EE

### Explication

- **HttpServlet** contient des méthodes abstraites, préfixées par `do()`, associées aux différentes méthodes (verbes) **HTTP**.
  - `doGet()` : s'exécute quand l'utilisateur demande une page (via la barre d'adresse, un lien hypertexte...)
  - `doPost()` : s'exécute quand l'utilisateur envoie des données via un formulaire par exemple
  - ...
- Chaque méthode prend en paramètre :
  - `HttpServletRequest` : contenant des informations sur la requête utilisateur
  - `HttpServletResponse` : permettant de personnaliser la réponse à retourner à l'utilisateur

## Jakarta EE

Ajoutons les méthodes `doGet()` et `doPost()` à `TestServlet`

```
package org.eclipse.controller;

import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

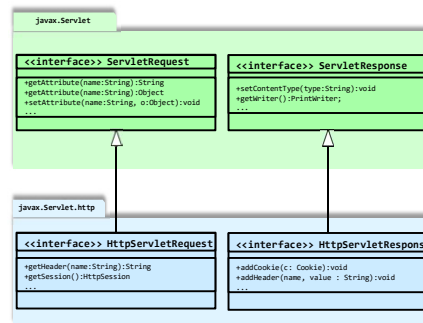
public class TestServlet extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse response) {
        // lecture de la requete
        // traitements
        // envoi de la reponse
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response) {
        // lecture de la requete
        // traitements
        // envoi de la reponse
    }
}
```

## objets *HttpServletRequest* et *HttpServletResponse*

- **HttpServletRequest :**
  - Contexte de l'appel
  - Paramètres de formulaires
  - Cookies
  - Headers
  - ...
- **HttpServletResponse**
  - Contrôle de la réponse
  - Type de données
  - Données
  - Cookies
  - Status
  - ...



## Jakarta EE

Mais quand cette Servlet sera exécuté?

Quand l'utilisateur saisit une **URL** dans le navigateur, il envoie une requête **HTTP** à notre contrôleur (qui est en vrai une Servlet)

## Jakarta EE

Mais quand cette Servlet sera exécuté ?

Quand l'utilisateur saisit une **URL** dans le navigateur, il envoie une requête **HTTP** à notre contrôleur (qui est en vrai une Servlet)

Et si on avait plusieurs Servlets, laquelle sera exécuté ?

- Chaque Servlet aura sa propre route (uniques)
- La Servlet ayant la route demandée sera exécutée

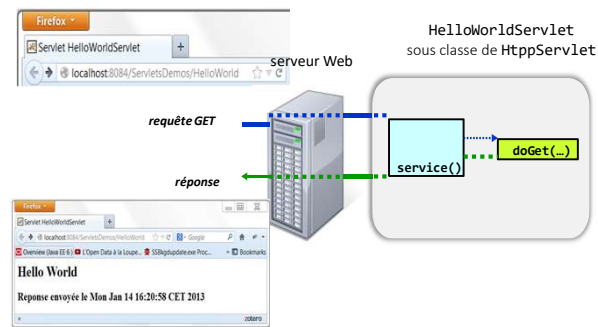
## Invoquer une servlet

`http://localhost:8084/ServletsDemos/HelloWo`

Protocole    serveur    contexte de la servlet (nom de l'application web sur le serveur\*)    url pattern pour la servlet (défini par le servlet mapping\*\*)

\* un serveur peut héberger plusieurs applications

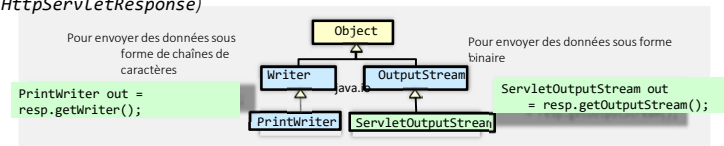
\*\* une application peut être composée de plusieurs servlets



## Utilisation des objets Request et Response

- Un pattern classique pour l'implémentation d'une méthode service (doXXX) d'une servlet est le suivant:

- Extraire de l'information de la requête (paramètre `request` : `HttpServletRequest`)
- Accéder éventuellement à des ressources externes
- Produire une réponse sur la base des informations précédentes
  - Récupérer un flux de sortie pour l'écriture de la réponse (à l'aide du paramètre `response` : `HttpServletResponse`)



- Définir les entêtes (HTTP headers) de la réponse

```
resp.setContentType("text/html");
```

indiquer le type du contenu

Types MIME (Multipurpose Internet Mail Extensions)  
<http://www.iana.org/assignments/media-types/>

```
resp.setContentType("image/png");
```

```
resp.setContentType("application/pdf");
```

- Ecrire le contenu (corps) de la réponse sur le flux de sortie

```
out.println("<p>blabla...</p>"); out.print(...)
```

directement avec l'objet out

```
byte[] tab = ...;
...
out.write(tab);
out.print(...);
```

en passant par API spécialisées ex: iText (<http://itextpdf.com/>) pour pdf  
<http://www.ibm.com/developerworks/java/library/os-javapdf/index.html?ca=dat>

```
Document document = new Document();
PdfWriter.getInstance(document,out);
PdfWriter.getInstance(document,out).write(...);
```



## Jakarta EE

Comment associer une route à une Servlet ?

- soit avec l'annotation `@WebServlet`
- soit dans le fichier `web.xml`

## Jakarta EE

Comment associer une route à une Servlet ?

- soit avec l'annotation `@WebServlet`
- soit dans le fichier `web.xml`

Commençons par créer une Servlet avec **Eclipse**

## Jakarta EE

### Pour créer une **Servlet** sous **Eclipse**

- Faire un clic droit sur `src` situé dans `Java Resources` de notre projet
- Aller dans `New` et choisir `Servlet`
- Remplir le champ `Java package:` par `org.eclipse.controller` (par exemple)
- Remplir le champ `Class name:` par un nom suffixé par le mot `Servlet` : `TestServlet` (par exemple)
- Cliquer sur `Next`

## Jakarta EE

### Routage par annotation (par défaut)

- On peut modifier ou supprimer l'URL Mappings. Remplaçons la chaîne existante (/TestServlet) par /mapage
- Cliquer sur Next
- Décocher la case Constructors from superclass
- Vérifier que les cases correspondantes aux deux méthodes doGet() et doPost sont cochées
- Valider en cliquant sur Finish

## Jakarta EE

Le contenu généré par Eclipse

```
package org.eclipse.controller;

// les imports

@WebServlet("/mapage")
public class TestServlet extends HttpServlet {

    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
        ServletException, IOException {
        response.getWriter().append("Served at: ").append(request.getContextPath());
    }

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

## Jakarta EE

### Routage dans `web.xml`

Le fichier `web.xml` situé dans `WEB-INF` de `webapp` permet de :

- déclarer la Servlet
- associer une URL à une Servlet (Mapping URL/Servlet)

## Jakarta EE

Si le fichier n'existe pas

- Faire un clic droit sur `WEB-INF` de `webapp` de notre projet
- Aller dans `New` et choisir `Other`
- Saisir `xml` dans la zone de recherche
- Choisir `XML File`
- Cliquer sur `Next` et choisir le nom `web.xml`

## Jakarta EE

### Contenu de web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="https://jakarta.ee/xml/ns/jakartaee"
  xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
    https://jakarta.ee/xml/ns/jakartaee/web-app_5_0.xsd" id="
    WebApp_ID" version="5.0">

  <display-name>cours-jee</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```



## Jakarta EE

```
<welcome-file-list>
```

- Contient les différents formats de fichiers qui peuvent être utilisés comme page d'accueil de l'application
- Ces fichiers seront directement dans `webapp`
- Pas besoin d'une Servlet pour les afficher
- Accessibles via la route `/` ou directement via leur nom

## Jakarta EE

Dans `web.xml`, on déclare la Servlet avant `</web-app>`

```
...  
<servlet>  
  <servlet-name>TestServlet</servlet-name>  
  <servlet-class>org.eclipse.controller.TestServlet</servlet-class>  
</servlet>
```

## Jakarta EE

Dans `web.xml`, on déclare la Servlet avant `</web-app>`

```
...
<servlet>
  <servlet-name>TestServlet</servlet-name>
  <servlet-class>org.eclipse.controller.TestServlet</servlet-class>
</servlet>
```

### Explication

- `<servlet>` et `</servlet>` : déclaration de la Servlet
- `<servlet-name>` et `</servlet-name>` : permet d'attribuer un nom à la Servlet qu'on utilisera plus tard
- `<servlet-class>` et `</servlet-class>` : indique le chemin de la classe de la Servlet

## Jakarta EE

### Autres sous balises disponibles pour Servlet

- `<description>` et `</description>` : ajouter une description sur le fonctionnement de la Servlet (comme un commentaire)
- `<load-on-startup>` et `</load-on-startup>` : permet de forcer le chargement de la Servlet lors de démarrage
- ...

## Jakarta EE

N'oublions pas, le rôle du `web.xml` :

- déclarer la Servlet (**c'est fait**)
- faire le mapping (assurer le routage si cela n'a pas été fait avec les annotations)

## Jakarta EE

```
...  
<servlet-mapping>  
  <servlet-name>TestServlet</servlet-name>  
  <url-pattern>/mapage</url-pattern>  
</servlet-mapping>  
</web-app>
```

## Jakarta EE

```
...
<servlet-mapping>
  <servlet-name>TestServlet</servlet-name>
  <url-pattern>/mapage</url-pattern>
</servlet-mapping>
</web-app>
```

### Explication

- `<servlet-mapping>` et `</servlet-mapping>` : pour faire le mapping Servlet/url
- `<servlet-name>` et `</servlet-name>` : permet d'indiquer le nom de la Servlet à appeler
- `<url-pattern>` et `</url-pattern>` : indique l'URL qui provoquera l'appel de la Servlet indiquée dans la sous-balise précédente

## Jakarta EE

Contenu de web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="
https://jakarta.ee/xml/ns/jakartaee" xsi:schemaLocation="https://
jakarta.ee/xml/ns/jakartaee https://jakarta.ee/xml/ns/jakartaee/web-
app_5_0.xsd" id="WebApp_ID" version="5.0">

  <servlet>
    <servlet-name>TestServlet</servlet-name>
    <servlet-class>org.eclipse.controller.TestServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>TestServlet</servlet-name>
    <url-pattern>/mapage</url-pattern>
  </servlet-mapping>
</web-app>
```



## Jakarta EE

### Remarque

Dans la suite de ce cours, on n'utilisera que le routage par annotation.

## Jakarta EE

### Une seule étape à faire

- Cliquer sur Run
- Une page blanche affichée ayant comme
  - adresse : `http://localhost:8080/cours-jee/mapage`
  - contenu : `Served at: /cours-jee`

## Jakarta EE

Si on teste une autre URL inexistante

- Écrire dans la zone d'adresse  
`http://localhost:8080/cours-jee/tapage`
- Une page HTTP 404 sera affichée

[Tester la Servlet](#)

## Jakarta EE

Comment afficher le `Hello World`

Il faut modifier la `Servlet` (l'objet `HttpServletResponse` qui est responsable de la réponse)

## Jakarta EE

### Nouveau contenu de la Servlet

```
public class TestServlet extends HttpServlet {  
  
    private static final long serialVersionUID = 1L;  
  
    protected void doGet(HttpServletRequest request,  
        HttpServletResponse response) throws ServletException,  
        IOException {  
  
        response.getWriter().print("Hello World");  
    }  
  
    protected void doPost(HttpServletRequest request,  
        HttpServletResponse response) throws ServletException,  
        IOException {  
        doGet(request, response);  
    }  
}
```

## Jakarta EE

Pour exécuter une deuxième fois

- Cliquer sur Run
- Choisir Continue without restarting (pas besoin de redémarrer le serveur)

## Jakarta EE

On peut indiquer l'encodage et le type du contenu de la réponse

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

    // pour indiquer le type de réponse
    response.setContentType("text/html");

    // indiquer l'encodage UTF-8 pour éviter les problèmes avec les
    accents
    response.setCharacterEncoding("UTF-8");

    PrintWriter out = response.getWriter();
    out.println("Hello World");
}
```

### L'objet `PrintWriter`

- s'obtient de l'objet `response`
- permet d'envoyer un (ou des) message(s) à l'utilisateur

## Jakarta EE

Pour retourner une page HTML complète

```
protected void doGet(HttpServletRequest request,
    HttpServletResponse response) throws ServletException,
    IOException{
    response.setContentType("text/html");
    response.setCharacterEncoding("UTF-8");
    PrintWriter out = response.getWriter();
    out.println("<!DOCTYPE html>");
    out.println("<html>");
    out.println("<head>");
    out.println("<meta charset=\"utf-8\" >");
    out.println("<title>Projet JEE</title>");
    out.println("</head>");
    out.println("<body>");
    out.println("Hello World");
    out.println("</body>");
    out.println("</html>");
}
```



## Jakarta EE

### Constats

- Beaucoup de code dans la **Servlet** (trop long) pour un affichage simple
- Violation du modèle **MVC** : le contrôleur n'affiche pas de résultat.

## Jakarta EE

### Constats

- Beaucoup de code dans la **Servlet** (trop long) pour un affichage simple
- Violation du modèle **MVC** : le contrôleur n'affiche pas de résultat.

### Solution

Utiliser des vues pour l'affichage (chapitre suivant).

## Jakarta EE

**A une Servlet peuvent être associées plusieurs routes**

```
@WebServlet({"/route1", "/route2", ... /routeN})
```

### Remarque

Pour récupérer la route qui a permis d'exécuter la Servlet, on utilise l'objet de type `HttpServletRequest`.

`http://localhost:8080/nom-projet/route-servlet?param1=value1&param2=value2`

## Jakarta EE

Protocole  
Scheme



<http://localhost:8080/nom-projet/route-servlet?param1=value1&param2=value2>

## Jakarta EE

Protocole  
Scheme

ServerName

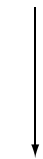
http://localhost:8080/nom-projet/route-servlet?param1=value1&param2=value2

## Jakarta EE

Protocole  
Scheme

ServerPort

ServerName



http://localhost:8080/nom-projet/route-servlet?param1=value1&param2=value2

## Jakarta EE

Protocole  
Scheme

ServerPort

ServerName

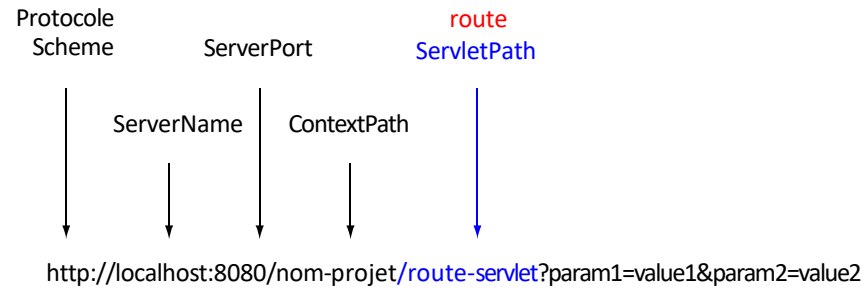
ContextPath

↓      ↓      ↓      ↓

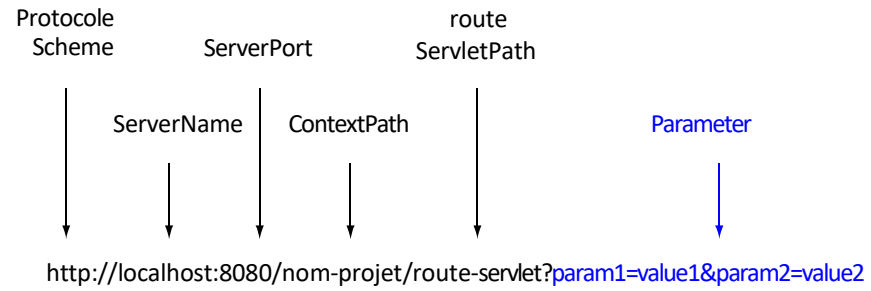
`http://localhost:8080/nom-projet/route-servlet?param1=value1&param2=value2`



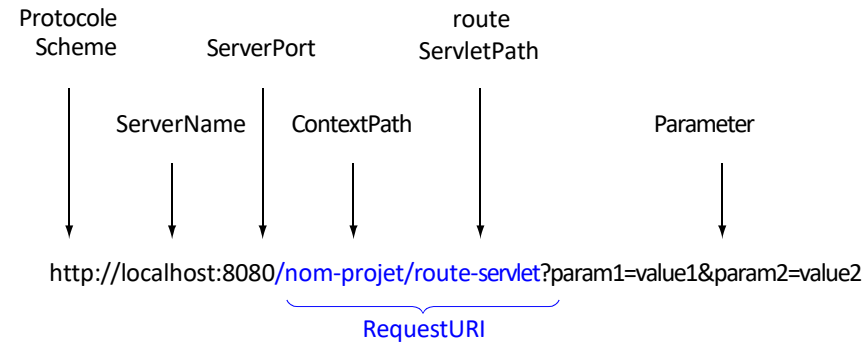
## Jakarta EE



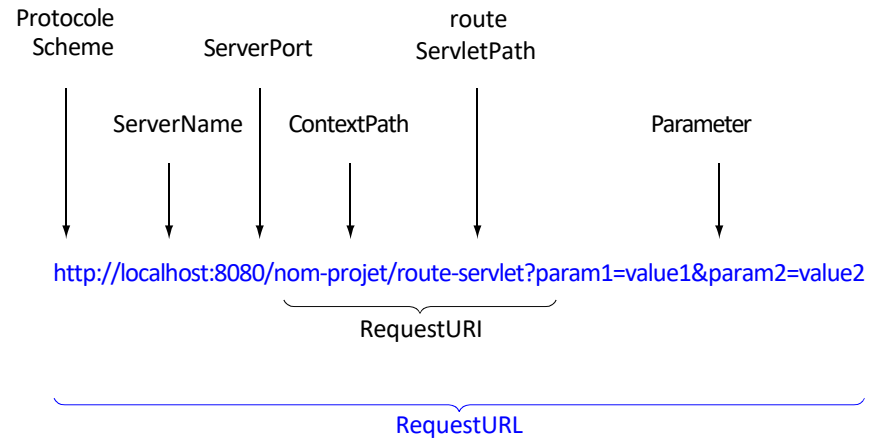
## Jakarta EE



## Jakarta EE



## Jakarta EE



Objet `HttpServletRequest`

## Jakarta EE

Comment récupérer toutes ces informations

Tout est défini dans l'objet de type `HttpServletRequest`.

## Jakarta EE

Comment récupérer toutes ces informations ?

Tout est défini dans l'objet de type `HttpServletRequest`.

### Exemples

Il suffit de préfixer le nom des propriétés précédentes par `get`

- `request.getContextPath()`
- `request.getServletPath()`
- `request.getServerPort()`
- ...

## Methodes de HttpRequest

### **Lire la requete :**

A l'intérieur de la méthode DoXXX() (DoGet() ou DoPost() selon la méthode invoquée), la requête de l'utilisateur est passée en paramètres sous forme d'objet (ou plus exactement l'interface) HttpServletRequest .

Afin de comprendre son fonctionnement, il est essentiel de connaître la façon selon laquelle les requêtes sont transmises du client au serveur par [le protocole HTTP](#).

## Methodes de HttpRequest

Méthode	Description
String getMethod()	Récupère la méthode HTTP utilisée par le client
String getHeader(String Key)	Récupère la valeur de l'attribut Key de l'en-tête
String getRemoteHost()	Récupère le nom de domaine du client
String getRemoteAddr()	Récupère l'adresse IP du client
String getParameter(String Key)	Récupère la valeur du paramètre Key (clé) d'un formulaire. Lorsque plusieurs valeurs sont présentes, la première est retournée
String[] getParameterValues(String Key)	Récupère les valeurs correspondant au paramètre Key (clé) d'un formulaire, c'est-à-dire dans le cas d'une sélection multiple (cases à cocher, listes à choix multiples) les valeurs de toutes les entités sélectionnées
Enumeration getParameterNames()	Retourne un objet <i>Enumeration</i> contenant la liste des noms des paramètres passés à la requête
String getServerName()	Récupère le nom du serveur
String getServerPort()	Récupère le numéro de port du serveur



## Jakarta EE

### Récupérer les paramètres d'une requête

- Mais, une requête peut avoir des paramètres (par exemple `/mapage?nom=Wick&prenom=John`)
- Comment, dans ce cas, récupérer les paramètres ?

## Jakarta EE

### Récupérer les paramètres d'une requête

- Mais, une requête peut avoir de paramètres (par exemple /mapage?nom=Wick&prenom=John)
- Comment, dans ce cas, récupérer les paramètres ?

### Solution

```
request.getParameter("nomParameter");
```

## Jakarta EE

### Exemple de récupération et d'affichage de paramètres de la requête

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

    String nom = request.getParameter("nom");
    String prenom = request.getParameter("prenom");

    PrintWriter out = response.getWriter();
    out.print("Hello " + nom + " " + prenom);

}
```

## Jakarta EE

### Exemple de récupération et d'affichage de paramètres de la requête

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

    String nom = request.getParameter("nom");
    String prenom = request.getParameter("prenom");
    PrintWriter out = response.getWriter();
    out.print("Hello " + nom + " " + prenom);
}
```

#### Remarque

Si le paramètre n'est pas présent, on peut avoir une exception.

## Jakarta EE

Il est recommandé de vérifier si la valeur du paramètre est non nulle avant de l'utiliser

```
protected void doGet(HttpServletRequest request,
    HttpServletResponse response) throws ServletException,
    IOException {
    String nom = request.getParameter("nom");
    String prenom = request.getParameter("prenom");
    PrintWriter out = response.getWriter();
    if (nom != null && prenom != null) {

        out.print("Hello " + nom + " " + prenom);
    } else {
        out.print("Hello world");
    }
}
```

## Récupérer les paramètres d'une requête

### À ne pas confondre

- Les paramètres de requête : concept relatif aux requêtes **HTTP**
- Les attributs de requête : concept introduit dans **JEE** (à voir dans le prochain chapitre)

## Jakarta EE

Si le paramètre est présent plusieurs fois dans l'URL avec le même nom, alors on peut utiliser `getParameterValues()`

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

    PrintWriter out = response.getWriter();

    String[] noms = request.getParameterValues("nom");

    if (noms != null) {
        for (String nom : noms) {
            out.print("Hello " + nom);
        }
    }
}
```

## Jakarta EE

Si le paramètre est présent plusieurs fois dans l'URL avec le même nom, alors on peut utiliser `getParameterValues()`

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

    PrintWriter out = response.getWriter();
    String[] noms = request.getParameterValues("nom");
    if (noms != null) {
        for (String nom : noms) {
            out.print("Hello " + nom);
        }
    }
}
```

### Exemple d'URL pour tester

```
http://localhost:8080/test-jee/home?nom=wick&nom=dalton
```



## Jakarta EE

Pour récupérer tous les paramètres dans un `Map`, on peut utiliser `getParameterMap()`

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

    PrintWriter out = response.getWriter();
    Map<String, String[]> params = request.getParameterMap();
    for (var param : params.entrySet()) {
        out.print(param.getKey());
        for (var value : param.getValue()) {
            out.print(value);
        }
    }
}
```

## Jakarta EE

Pour récupérer tous les paramètres dans un `Map`, on peut utiliser `getParameterMap()`

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

    PrintWriter out = response.getWriter();
    Map<String, String[]> params = request.getParameterMap();
    for (var param : params.entrySet()) {
        out.print(param.getKey());
        for (var value : param.getValue()) {
            out.print(value);
        }
    }
}
```

Exemple d'URL pour tester

```
http://localhost:8080/test-jee/home?nom=wick&nom=dalton&genre=homme&age=45
```

## Objet HttpServletResponse

### Méthodes de HttpServletResponse

#### **Créer une réponse :**

De la même façon, la réponse à fournir à l'utilisateur est représentée sous forme d'objet HttpServletResponse. Voici les différentes méthodes de l'objet HttpServletResponse

## Methodes de HttpServletResponse

Méthode	Description
String setStatus(int StatusCode)	Définit le code de retour de la réponse
void setHeader(String Nom, String Valeur)	Définit une paire clé/valeur dans les en-têtes
void setContentType(String type)	Définit le type MIME de la réponse HTTP, c'est-à-dire le type de données envoyées au navigateur
void setContentLength(int len)	Définit la taille de la réponse
PrintWriter getWriter()	Retourne un objet « <b>PrintWriter</b> » permettant d'envoyer du texte au navigateur client.
ServletOutputStream getOutputStream()	Définit un flot de données à envoyer au client, par l'intermédiaire d'un objet <b>ServletOutputStream</b> , dérivé de la classe <b>java.io.OutputStream</b>
void sendRedirect(String location)	Permet de rediriger le client vers l'URL <i>location</i>

[Rediriger vers une autre Servlet](#)

## Jakarta EE

**Rediriger vers une autre Servlet annotée par** `@WebServlet("/MaServlet")`

```
response.sendRedirect("MaServlet");
```

[Rediriger vers une autre Servlet](#)

## Jakarta EE

**Rediriger vers une autre Servlet annotée par** `@WebServlet("/MaServlet")`

```
response.sendRedirect("MaServlet");
```

**Ne pas mettre "/" avant** `MaServlet`.

Rediriger vers une autre Servlet

## Jakarta EE

**Rediriger vers une autre Servlet annotée par** `@WebServlet("/MaServlet")`

```
response.sendRedirect("MaServlet");
```


Ne pas mettre "/" avant `MaServlet`.

**On peut aussi reconstruire l'URL depuis le** `contextPath`

```
response.sendRedirect(request.getContextPath() + "/MaServlet");
```

```
@WebServlet("/MaPremiereServlet")
public class MaPremiereServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        // Redirection vers la servlet MaServlet
        response.sendRedirect(request.getContextPath() + "/MaServlet");
    }
}

@WebServlet("/MaServlet")
public class MaServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        // Réponse de la servlet MaServlet
        response.getWriter().write("Vous êtes redirigé vers MaServlet !");
    }
}
```


 Copy code



La plupart du temps, la programmation de Servlets consiste de récupérer des données en paramètre, à traiter ces données et à envoyer une réponse adaptée au client. Nous avons utilisé la méthode **sendRedirect()** de l'objet **response** qui permet de rediriger l'utilisateur vers une page précise. Par contre, cette redirection **ne permet pas d'envoyer des données directement dans le corps du message.**

Solution:

**javax.servlet.RequestDispatcher**



- Le *ServletContext* est un conteneur d'informations unique de l'application web. utilisé pour partager des données et des paramètres entre tous les servlets, filtres et autres composants de l'application web

- Une Servlet retrouve le *ServletContext* de son application web par appel à *getServletContext()*.

- **Exemples de méthodes de *ServletContext* :**

- void setAttribute(String name, Object o)* : stocke un objet sous le nom indiqué

- Object getAttribute(String name)* : retrouve l'objet sous le nom indiqué

- Enumeration getAttributeNames()* : retourne l'ensemble des noms de tous les attributs stockés

- void removeAttribute(String name)* : supprime l'objet stocké sous le nom indiqué

- ...

## Différence entre paramètres et attributs

- Les paramètres de requête sont un concept appartenant au protocole HTTP. Ils sont envoyés par le client au serveur directement au sein de l'URL, et donc sous forme de chaînes de caractères.
  - Ce concept n'étant absolument pas spécifique à la plate-forme Java EE mais commun à toutes les technologies web.
  - Les paramètres sont donc des données issues du Client
- 
- Les attributs de requête sont un concept appartenant au conteneur Java, et sont donc créés côté serveur : c'est au sein du code de l'application que l'on procède à leur initialisation.
  - Contrairement aux paramètres, ils ne sont pas présents directement dans la requête HTTP mais uniquement dans l'objet Java qui l'enveloppe (l'objet `HttpServletRequest`), et peuvent contenir n'importe quel type de données.
  - Ils sont utilisés pour permettre à une servlet de communiquer avec d'autres servlets ou pages JSP.
  - Les attributs sont donc des données issues du Serveur

## Principales fonctionnalités de ServletContext

- **Accès aux ressources de l'application** : Vous pouvez accéder aux fichiers de ressources comme des fichiers HTML, des fichiers de configuration ou des fichiers statiques à l'aide de méthodes comme **getResource()** ou **getResourceAsStream()**.
- **Stockage des données au niveau de l'application** : Vous pouvez stocker et partager des objets entre différents servlets et autres composants de l'application en utilisant **setAttribute()** et **getAttribute()**.
- **Informations de configuration de l'application** : Il vous permet d'accéder aux paramètres d'initialisation configurés dans le fichier web.xml ou dans les annotations via la méthode **getInitParameter()**.
- **Journalisation** : Il permet d'enregistrer des informations dans le journal du serveur à l'aide de la méthode **log()**.
- **Accès aux informations de déploiement** : Vous pouvez récupérer des informations sur l'application (comme le nom, le chemin contextuel, la version) via des méthodes comme **getContextPath()** et **getServerInfo()**

## Principales méthodes de ServletContext

### **getInitParameter(String name) :**

Récupère un paramètre d'initialisation de l'application, défini dans le fichier web.xml.

Exemple : `String paramValue = context.getInitParameter("configPath");`

### **getAttribute(String name) / setAttribute(String name, Object value) :**

Permet de stocker et de récupérer des objets partagés au niveau de l'application. Les attributs sont visibles par tous les composants de l'application.

Exemple :// Stocker un objet `context.setAttribute("myAttribute", someObject);`

// Récupérer un objet `Object obj = context.getAttribute("myAttribute");`

### **getResource(String path) / getResourceAsStream(String path) :**

Permet d'accéder aux ressources de l'application, comme les fichiers HTML ou les fichiers de configuration, en utilisant un chemin relatif.

Exemple : `InputStream is = context.getResourceAsStream("/WEB-INF/config.properties");`

### **log(String msg) :**

Permet de consigner des messages dans le journal du serveur. Utile pour la journalisation et le débogage.

Exemple : `context.log("Application started successfully");`

### **getContextPath() :**

Retourne le chemin de contexte de l'application web.

Exemple : `String contextPath = context.getContextPath();`

### **getMimeType(String file) :**

Retourne le type MIME d'un fichier. Exemple : `String mimeType = context.getMimeType("example.pdf");`

### **getRealPath(String path) :**

Retourne le chemin réel sur le système de fichiers du serveur pour une ressource donnée.

Exemple : `String realPath = context.getRealPath("/WEB-INF/config.xml");`

## Utilisation d'autres ressources

- Le traitement d'une requête HTTP par une servlet peut nécessiter l'utilisation d'autres ressources
  - inclusion d'une page HTML statique
  - redirection vers une autre servlet ou page JSP
- Réalisé à l'aide d'un objet `javax.servlet.RequestDispatcher` (fournit par le conteneur web)

### <<interface>> RequestDispatcher

```
+ forward(servletRequest req, ServletResponse rep) : void  
+ include(servletRequest req, ServletResponse rep) : void
```

- Obtention d'un objet `RequestDispatcher` (Récupération de l'objet en fonction de l'url souhaitée)
  - via un objet `javax.servlet.ServletContext`
  - via un objet `javax.servlet.http.HttpServletRequest`
  - `RequestDispatcher` `getRequestDispatcher(java.lang.String path)`
    - path** : chaîne de caractères précisant le chemin de la ressource vers laquelle la requête doit être transférée
      - doit commencer par un '/' et est interprété comme relatif au contexte de l'application

## ***Include***

Le mécanisme **include** permet d'inclure dynamiquement le contenu d'une autre ressource dans la réponse de la servlet courante. Cela signifie que le contrôle reste dans la servlet d'origine, mais une partie de la réponse provient d'une autre ressource.

### **Fonctionnement :**

- La requête est envoyée à la ressource cible.
- La ressource incluse traite la requête et renvoie son contenu au flux de réponse sans interrompre l'exécution de la servlet d'origine.
- La servlet d'origine termine ensuite le traitement et envoie la réponse finale au client.

**Cas d'utilisation :** **include** est utile lorsque vous voulez inclure des composants partagés comme des en-têtes, des pieds de page, ou des menus qui apparaissent sur plusieurs pages, mais où le traitement principal reste dans la servlet d'origine.

## RequestDispatcher : include

- Exemple: insérer dans la réponse des fragments de code HTML

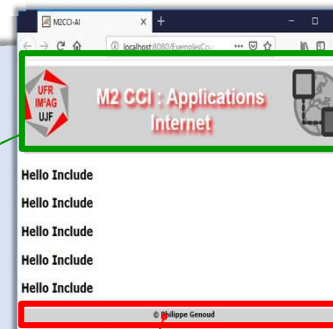
```
package fr.im2ag.m2cci.servletsdemo.servlets;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet(name = "HelloInclude", urlPatterns = {"/demoInclude"})
public class HelloInclude extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            // inclusion de l'en tête
            RequestDispatcher dispatcher =
                getServletContext().getRequestDispatcher("/includes/header.html");
            dispatcher.include(request, response);
            // creation d'un page simple
            for (int i = 0; i < 5; i++) {
                out.println("<h3>Hello Include</h3>");
            }

            // inclusion du pied de page
            getServletContext().getRequestDispatcher("/includes/footer.html").include(request,
                response);
        } finally {
            out.close();
        }
    }
}
```





## RequestDispatcher : include

```
localhost:8080/ExemplesCours/demoInclude *** 🛡️ ☆  
RequestDispatcher dispatcher = getServletContext().getRequestDispatcher("/includes/header.html");  
dispatcher.include(request, response);  
...  
getServletContext().getRequestDispatcher("/includes/footer.html").include(request, response);
```

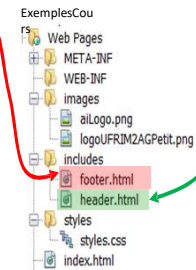
chemins relatifs au contexte de l'application

header.html

```
<!DOCTYPE html>  
<html>  
  <head>  
    <title>M2CCI-AI</title>  
    <meta charset="UTF-8" />  
    <link href="./styles/styles.css" rel="stylesheet" type="text/css" />  
  </head>  
  <body>  
    <header>  
      <a href="http://ufrima.imag.fr/"></a>  
      <h1>M2 CCI : Applications Internet</h1>  
      <a href="http://www.ujf-grenoble.fr"></a>  
    </header>
```

footer.html

```
<footer>  
  <p class="auteur"><small>&copy; Philippe Genoud</small></p>  
</footer>  
</body>  
</html>
```



## *forward*

Le principe de **Forward** avec un **ServletDispatcher** en Java consiste à transférer la requête d'une servlet à une autre ressource, qu'il s'agisse d'une autre servlet, d'une page JSP, ou d'une autre page web. Ce processus de redirection interne est invisible pour l'utilisateur final, car l'URL dans la barre de navigation ne change pas.

Voici les étapes principales :

**Récupération du RequestDispatcher** : Le servlet d'origine obtient un objet **RequestDispatcher** en appelant la méthode ***getRequestDispatcher()*** sur l'objet `HttpServletRequest`. Cette méthode prend en paramètre le chemin vers la ressource de destination (un autre servlet ou une page JSP).

**Transfert de la requête** : Le servlet utilise ensuite la méthode `forward()` de l'objet `RequestDispatcher` pour transférer à la nouvelle ressource la requête et la réponse. La ressource cible peut traiter la requête, générer une réponse, ou encore faire un traitement supplémentaire avant de renvoyer un résultat à l'utilisateur.

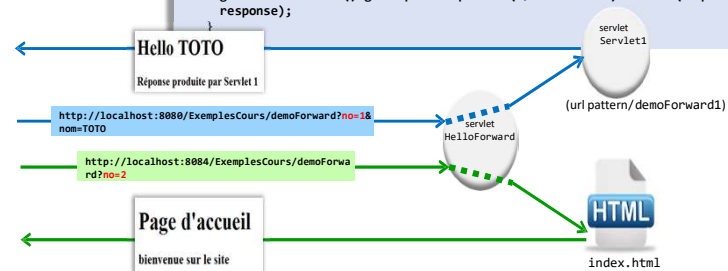
## RequestDispatcher : forward

- redirige vers une autre ressource pour produire la réponse.

```
package
fr.im2ag.m2cci.servletsdemo.servlets;
import
java.io.IOException;
...

@WebServlet(name = "HelloForward", urlPatterns =
{"/demoForward"})
public class HelloForward extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        int noServlet = Integer.parseInt(request.getParameter("no"));
        if (noServlet == 1) {
            getServletContext().getRequestDispatcher("/demoForward1").forward(request,
response);
        } else {
            getServletContext().getRequestDispatcher("/index.html").forward(request,
response);
        }
    }
}
```



## RequestDispatcher : forward

- les objets **request** et **response** de la servlet initiale sont transmis à la ressource vers laquelle la requête est redirigée

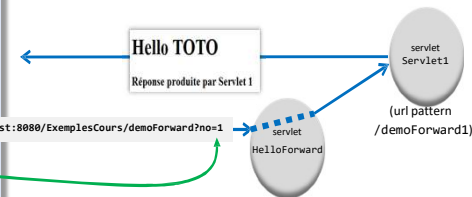
```
...
@WebServlet(name = "Servlet1", urlPatterns = {"/demoForward1"})
public class Servlet1 extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html; charset=UTF-8");
        PrintWriter out = response.getWriter();

        try {
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Exemple Forward</title>"); out.println("</head>");
            out.println("<body>");

            out.println("<h1>Hello " + request.getParameter("nom") + "</h1>");
            out.println("<h3>Réponse produite par Servlet 1</h3>");
            out.println("</body>");
            out.println("</html>");
        } finally {
            out.close();
        }
    }
}
```

```
...
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    int noServlet = Integer.parseInt(request.getParameter("no")); if (noServlet == 1) {
        getServletContext().getRequestDispatcher("/demoForward1").forward(request, response);
    } else {
        getServletContext().getRequestDispatcher("/index.html").forward(request, response);
    }
}
...
```

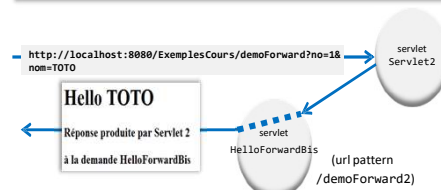


- paramètres de **request** sont ceux de la requête originale

## RequestDispatcher : forward

```
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse
response)
throws ServletException, IOException {
    int noServlet =
    Integer.parseInt(request.getParameter("no"));
    if (noServlet == 1) {
        request.setAttribute("demandeur", "HelloForwardBis");
        getServletContext().getRequestDispatcher("/demoForward2").forward(request,
response);
    } else {
        getServletContext().getRequestDispatcher("/index.html").forward(request, response);
    }
}
```

- la ressource appelante peut transmettre des informations supplémentaires via les attributs de la requête



Les paramètres sont des **String**  
Les attributs peuvent être des objets quelconques

```
public void setAttribute(String name, Object value)
public Object getAttribute(String name)
```

```
...
@WebServlet(name = "Servlet2", urlPatterns = {"/demoForward2"})
public class Servlet2 extends HttpServlet
{
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Exemple Forward</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>Hello " + request.getParameter("nom") + "</h1>");
            out.println("<h3>Réponse produite par Servlet 2</h3>");
            String origine = (String) request.getAttribute("demandeur");
            out.println("<h3>> la demande " + origine + "</h3>");
            out.println("</body>");
            out.println("</html>");
        } finally {
            out.close();
        }
    }
}
```

## RequestDispatcher : forward

- forward comporte plusieurs contraintes :

- la servlet d'origine ne doit pas avoir déjà expédié des informations vers le client sinon une exception `java.lang.IllegalStateException` est lancée
- si des informations sont déjà présentes dans le buffer de la réponse mais n'ont pas encore été envoyées, elles sont effacées lorsque le contrôle est transmis à la deuxième ressource
- lorsque la deuxième ressource a terminé le traitement de la requête, l'objet réponse n'est plus utilisable par la première pour produire du contenu

```
int noServlet = Integer.parseInt(request.getParameter("no"));

PrintWriter out = response.getWriter();
try {
    if (noServlet == 1) {
        response.setContentType("text/html;charset=UTF-8");
        out.println("<h3>Cet en tête n'apparaît pas</h3>");
        out.println("</body>");
        out.println("</html>");
        1 out.flushBuffer();
        getServletContext().getRequestDispatcher("/demoForward1").forward(request, response);
    } else {
        getServletContext().getRequestDispatcher("/demoForward3").forward(request, response);
        out.println("<h3>Ce texte n'apparaît pas</h3>");
        out.println("</body>");
        out.println("</html>");
        3
    }
} finally {
    out.close();
}
```

**Différences entre forward et include :**

- **forward** : Transfère la requête et la réponse à une nouvelle ressource, et la servlet ou page d'origine ne produit plus de sortie après l'appel à forward.
- **include** : Inclut simplement le contenu d'une autre ressource dans la réponse sans interrompre l'exécution de la servlet d'origine.



Méthode forward()	Méthode sendRedirect()
La méthode forward() fonctionne côté serveur.	La méthode sendRedirect() fonctionne côté client.
Il envoie les mêmes objets de requête et de réponse à un autre servlet.	Il envoie toujours une nouvelle demande.
Il ne peut fonctionner qu'au sein du serveur.	Il peut être utilisé à l'intérieur et à l'extérieur du serveur.
Exemple : request.getRequestDispatcher(« servlet2 »).forward(request,response) ;	Exemple : response.sendRedirect(« servlet2 ») ;

