

Chapitre 2 – Les Servlets



Pr. Ennaji Fatima Zohra

Les servlets java forment la pierre angulaire des composants web java, elles sont utilisées pour gérer la logique métier des applications web, elle peuvent aussi gérer la partie présentation.

- ▶ Le navigateur envoie une requête http au serveur dans les situations suivantes :
 - l'utilisateur clique sur un lien hypertexte dans une page html
 - L'utilisateur remplit une formulaire html dans une page web et l'envoie
 - L'utilisateur saisi une adresse URL dans la barre d'adresse du navigateur et appuie sur entrer.
- ▶ Il y'a d'autres événements qui causent l'envoi d'un requête au serveur comme la fonction javascript reload(). Mais ses événement ne sont en fait que des simulations des actions du client.
- ▶ Par défaut le navigateur utilise la méthode GET pour envoyer des requêtes dans toutes les événements précitées, cependant on peut lui forcer à utiliser POST ou HEAD via l'attribut « method » comme suit :

<FORM name='form1' method='POST' action='/premiereServlet/login.fc' >

Comparaison des méthodes HTTP

► Le tableau suivant résume les points de différence entre GET et POST:

propriétés	GET	POST
Type de ressource	Dynamique ou statique	dynamique
Type de données	Texte	Texte et flux binaire
Taille des données	255 caractères	illimité
Visibilité	Les données sont codées dans l'URL	Les données sont envoyées dans le contenu du message http
Cache du navigateur	Les données peuvent être caché dans l'historique du navigateur	Les données ne sont pas caché par l'historique du navigateur

Traitement des requêtes dans `HttpServlet`

- ▶ HTTP `HttpServlet` définit pour chaque méthode une méthode correspondante avec la signature suivante :
 - `protected void doXXX(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException;`
 - GET → `doGet()`
 - POST → `doPost()`
 - HEAD → `doHead()`
- ▶ La classe `HttpServlet` offre une implémentation vide des méthodes `doXXX()`. Vous devez redéfinir ces méthodes pour implémenter la logique métier de votre application.

Séquence d'événements dans `HttpServlet`

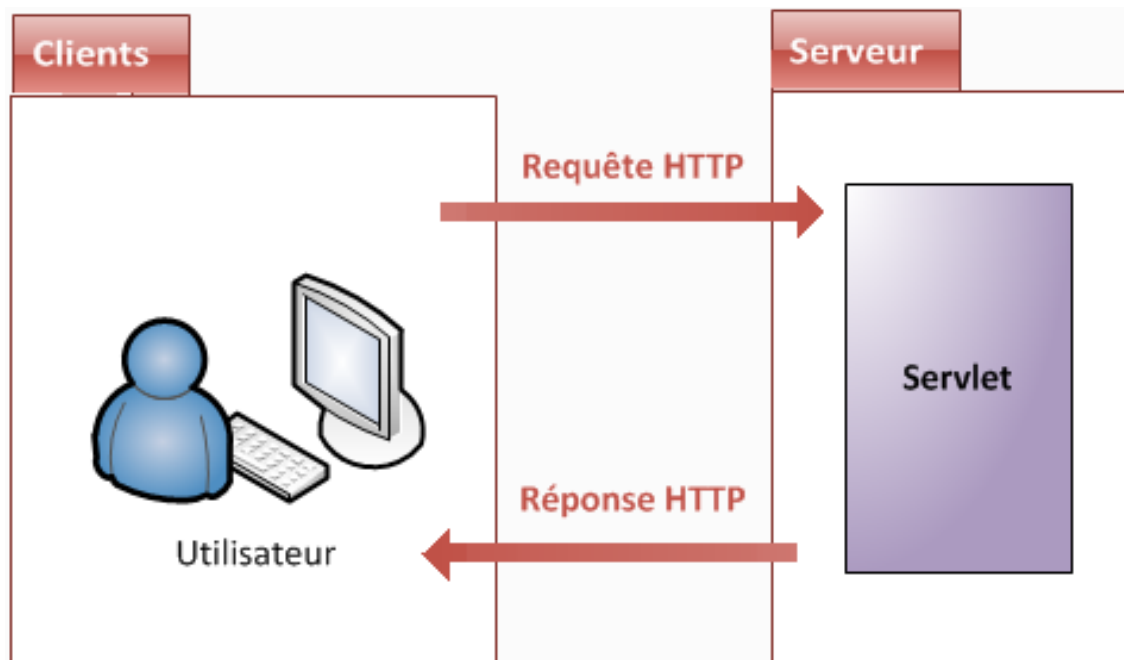
1. Le conteneur de servlets appelle la méthode `service(ServletRequest req, ServletResponse resp)` défini dans `HttpServlet`.
2. La méthode `service(ServletRequest req, ServletResponse)` de `HttpServlet` appelle la méthode `service(HttpServletRequest req, HttpServletResponse resp)` de la même classe (la deuxième méthode est une version surchargée de la première).
3. La méthode `service(HttpServletRequest req, HttpServletResponse resp)` analyse la requête et détecte la méthode HTTP utilisée puis appelle la méthode **doXXX** correspondante.

Remarque :

- Si vous redéfinissez la méthode `service(HttpServletRequest req, HttpServletResponse resp)` vous perdez les fonctionnalités offertes par `HttpServlet`, et par conséquent les méthodes `doXXX` ne seront pas appelés automatiquement
- `HttpServletRequest` et `HttpServletResponse` sont des interfaces, les implémentations concrètes sont offertes par le conteneur des servlets.

Première Servlet

- ▶ Le langage Java n'est pas du tout adapté à la rédaction de pages web ! (voir dernier exemple).
- ▶ Revenir au modèle MVC : l'affichage de contenu HTML n'ayant rien à faire dans le contrôleur (notre servlet), nous allons créer une vue et la mettre en relation avec notre servlet.



- ▶ Le client envoie des requêtes au serveur grâce aux méthodes du protocole HTTP, notamment GET, POST et HEAD.
- ▶ Le conteneur web place chaque requête reçue dans un objet `HttpServletRequest`, et place chaque réponse qu'il initialise dans l'objet `HttpServletResponse`.
- ▶ Le conteneur transmet chaque couple requête/réponse à une servlet (se déclare dans son fichier de configuration `web.xml`) : c'est un objet Java assigné à une requête et capable de générer une réponse en conséquence.
- ▶ Pour pouvoir traiter une requête HTTP de type GET, une servlet doit implémenter la méthode `doGet()` (de même pour une requête de type POST, etc.).
- ▶ Une servlet n'est pas chargée de l'affichage des données, elle ne doit donc pas s'occuper de la présentation (HTML, CSS, etc.).

Séparation de contrôleur et de la vue

► Servlet:

```
package    Servlets;

+ import java.io.IOException;

public class PremiereServlet extends HttpServlet{

-     public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException{
        this.getServletContext().getRequestDispatcher( "/Page.jsp" ).forward( request, response );
    }
}
```

► Page web:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="ISO-8859-1">
        <title>Test</title>
    </head>
    <body>
        <p>Ceci est une page générée depuis une servlet.</p>
    </body>
</html>
```

L'analyse de la requête

ServletRequest et sa sous interface HttpServletRequest offrent des méthodes pour analyser la requête du client et extraire les données envoyées par le navigateur : paramètres, méta-informations contenu texte ou les flux binaires.

Méthode	Description
String getParameter(String paramName)	Retourne la valeur associée au paramètre passé en argument
String[] getParameterValues(String paramName)	Retourne la liste des valeurs associées au paramètre passé en argument
Enumeration getParameterNames()	Retourne la liste des noms de paramètres
String getHeader(String headerName)	Retourne la valeur associée à l'entête passé en argument
Enumeration getHeaders(String headerName)	Retourne la liste des valeurs associées à l'entête passé en argument
Enumeration getHeaderNames()	Retourne la liste des noms des entêtes

Exemple 1 : interception des paramètres

```
<form action="./PremierePage" method="POST">
  <table >
    <tr>
      <td>Nom : </td>
      <td><input type="text" name="nom"></input></td>
    </tr>
    <tr>
      <td>Préférences : </td>
      <td>
        <select name="prefs" size="4" multiple="multiple">
          <option value="art">Art</option>
          <option value="science">Science</option>
          <option value="technologie">Technologie</option>
        </select>
      </td>
    </tr>
  </table>
  <input type="submit" value="Valider">
</form>
```

Redéfinition de doPost()

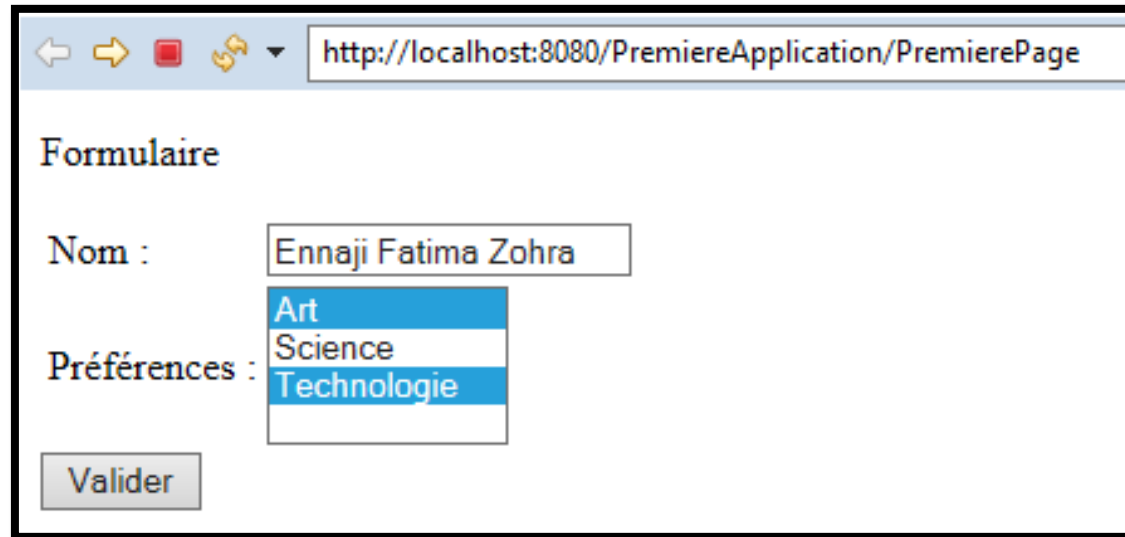
```
public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    String nom = request.getParameter("nom");
    String[] prefs = request.getParameterValues("prefs");
    String prefsString="";

    for (int i = 0; i < prefs.length; i++) {
        prefsString+=prefs[i]+"<br>";
    }

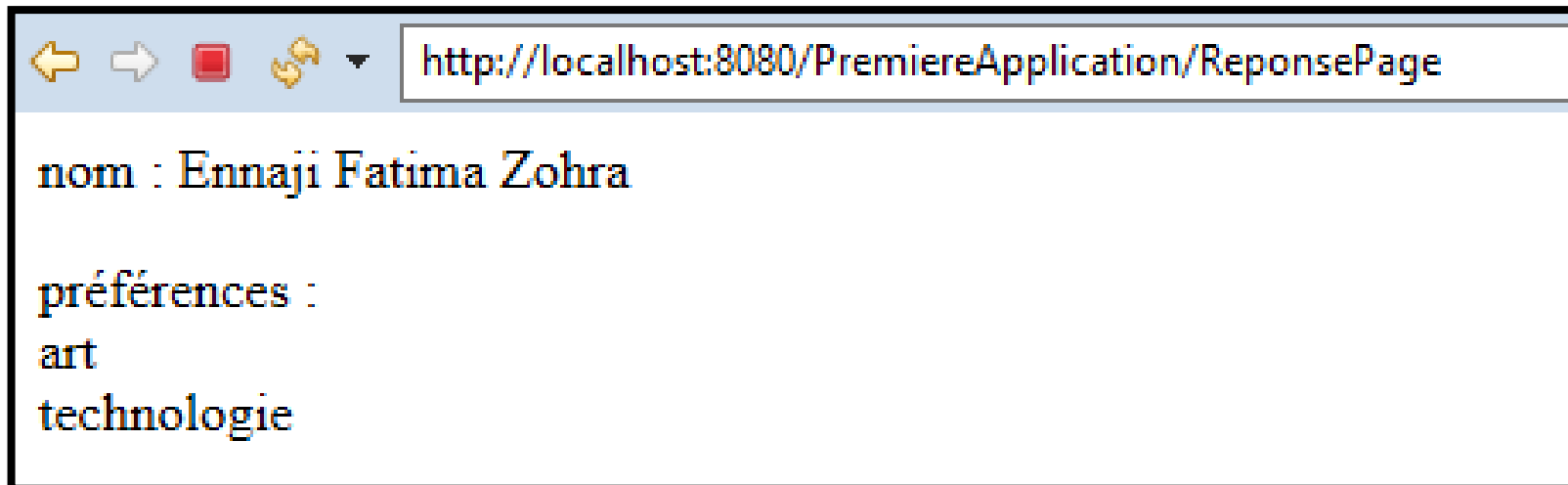
    PrintWriter pw = response.getWriter();

    pw.println("<html><head><title>Reponse POST</title></head><body>"
        + "<p>nom : "+nom+"</p>"
        + "<p>préférences : <br>" +prefsString+"</p>"
        + "</body></html>");
}
```

Résultat



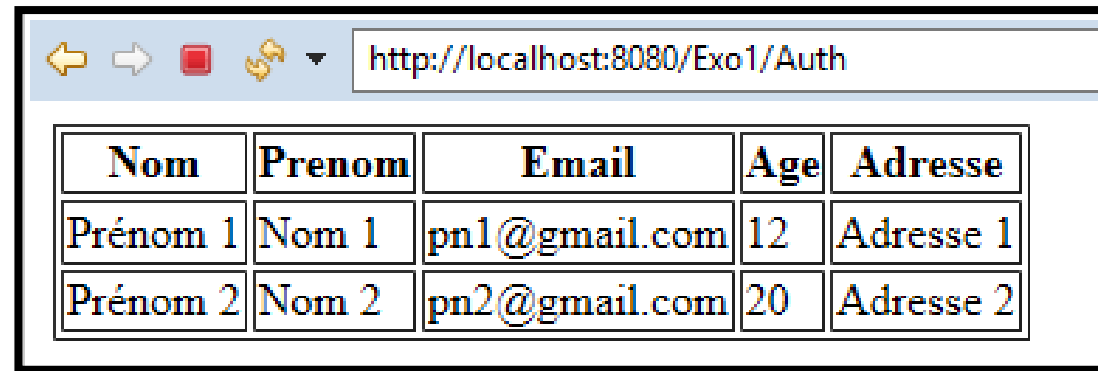
A screenshot of a web browser window. The address bar shows the URL `http://localhost:8080/PremiereApplication/PremierePage`. The page content is titled "Formulaire". It contains a label "Nom :" followed by a text input field containing "Ennaji Fatima Zohra". Below this is a label "Préférences :" followed by a dropdown menu. The dropdown menu is open, showing three options: "Art", "Science", and "Technologie", with "Art" currently selected. At the bottom left of the form is a button labeled "Valider".



A screenshot of a web browser window. The address bar shows the URL `http://localhost:8080/PremiereApplication/ReponsePage`. The page content displays the submitted data in a simple text format: "nom : Ennaji Fatima Zohra", followed by a blank line, then "préférences :", followed by another blank line, and finally the words "art" and "technologie" on separate lines.

Exercice

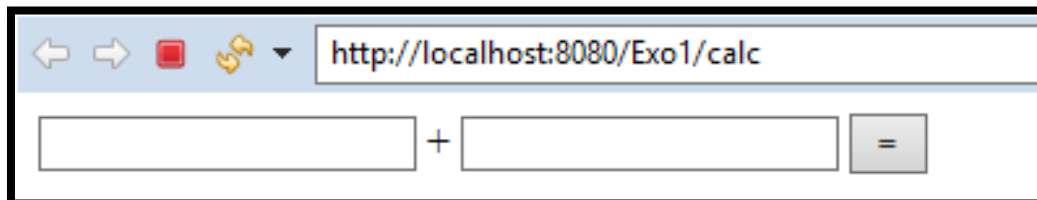
- ▶ Créer un projet Java EE qui permettra d'afficher le tableau suivant contenant les informations des personnes. Ces informations seront générées dans le contrôleur.



A screenshot of a web browser window with the address bar showing `http://localhost:8080/Exo1/Auth`. The browser displays a table with 5 columns: Nom, Prenom, Email, Age, and Adresse. The table contains two data rows.

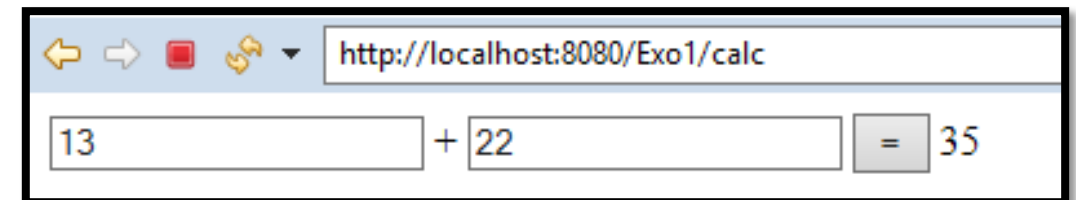
Nom	Prenom	Email	Age	Adresse
Prénom 1	Nom 1	pn1@gmail.com	12	Adresse 1
Prénom 2	Nom 2	pn2@gmail.com	20	Adresse 2

- ▶ Créer un projet Java EE qui permettra de réaliser la somme entre deux chiffres entiers et d'afficher le résultat après avoir cliqué sur « = ».



A screenshot of a web browser window with the address bar showing `http://localhost:8080/Exo1/calc`. The browser displays a simple calculator interface with two input fields, a plus sign, and an equals sign button.

+ =



A screenshot of a web browser window with the address bar showing `http://localhost:8080/Exo1/calc`. The browser displays the result of a calculation: 13 + 22 = 35.

13 + 22 = 35

Puisque une application web est normalement conçue pour interagir avec plusieurs clients en même temps, elle a besoins de mémoriser l'historique des interactions avec chaque utilisateur. Les session offre un mécanisme qui permet de garder les traces des ces interactions.

État et sessions :

- Les protocoles d'interactions sont subdivisées en deux catégories : les protocoles avec état et sans état. Les protocoles de la première catégorie sont capables de mémoriser l'état des requêtes passées, tandis que ceux de la deuxième catégorie considère chaque requête comme une nouvelle requête. HTTP est un protocole sans état, donc un serveur HTTP n'a pas de moyen pour déterminer si une succession de requêtes provient du même client ou pas. Ça veut dire que le serveur HTTP ne peut pas maintenir l'état du client entre deux requêtes.
- Dans certaines applications, on a pas besoins de se souvenir de l'état du client, mais dans d'autres c'est primordial « par exemple pour garder le panier dans une application e-commerce ».

- Une session est une série ininterrompue de requête-réponse échangée entre un serveur et un client.
- Une session commence lorsqu'un client inconnu envoie une première requête à l'application web. Elle se termine lorsque le client explicitement termine la session ou lorsque le serveur ne reçoit aucune requête pendant un certain temps.
- Lorsque le serveur reçoit une première requête de la part d'un client, il initialise une session et lui assigne un « id » unique.
- Le client à son tour doit inclure cet « id » dans les requêtes qui suivent.

- L'API des servlets offre une abstraction de la session représentée par l'interface **javax.servlet.http.HttpSession** qui est implémentée par le conteneur de servlets.
- La methode **getSession(true)** définit dans **HttpRequest** permet d'obtenir le référence de la session en cours s'il y en a une sinon il permet de créer une nouvelle session.
- La manipulation des session passe par trois étapes :
 - 1– la récupération de l'objet **HttpSession**
 - 2- ajout ou suppression des variables de la session
 - 3- destruction de la session si nécessaire

Le temps maximum d'inactivité d'une session est spécifié dans le descripteur du déploiement comme suit :

```
<session-config>
```

```
  <session-timeout>40</session-timeout>
```

```
</session-config>
```

La classe HttpSession définit plusieurs méthodes qui permettent de manipuler la session :

Méthode	Description
void setAttribut(String name, Object value)	Stocke l'objet value dans la session avec la clé name.
long getLastAccessedTime()	Retourne la date de la dernière requête client envoyé avec cette session
Object getAttribute(String name)	Retourne l'objet stocké dans la session avec la clé name.
Enumeration getAttributeNames()	Retourne la liste des noms d'attributs stockés dans la session
long getCreationTime()	Retourne la date de création de la session
void setMaxInactiveInterval(int sec)	Redéfinir le temps max d'inactivité
void invalidate()	Pour détruire la session et effacer toutes ses variable.

Comment le serveur détecte qu'une requête fait partie d'une session ?

1. Un client envoie une requête au serveur. Puisque c'est la première requête il ne contient pas l'id de la session.
2. Le serveur crée une nouvelle session et lui affecte un nouveau id. l'appel de la méthode `isNew()` renvoie `true`. Le serveur envoie l'id au client avec la réponse (stocké dans l'entête avec la clé `cookie`).
3. Le client récupère l'id de la session et il le stocke pour l'utiliser dans les requêtes suivantes.
4. Le client envoie une autre requête, et cette fois si elle envoie l'id de la session avec la requête (toujours codé dans l'entête).
5. Le serveur reçoit la requête et observe l'id de la session et il associe automatiquement cette requête avec la session correspondante. Dans ce cas on dit que le client rejoint la session car c'est pas une nouvelle session. L'appel de la méthode `isNew()` renvoie `false`.

Notez bien que l'id de la session est codé dans l'entête avec la clé : cookie

```
Bonjour ahmed vous êtes maintenant connecté ===== headers ===  
accept : image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-  
xpsdocument, application/xaml+xml, application/x-ms-xbap, application/x-sl  
application/vnd.ms-powerpoint, application/msword, */*  
referer : http://localhost:8080/PremiereServlet/login.html  
accept-language : fr  
ua-cpu : x86  
accept-encoding : gzip, deflate  
user-agent : Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0; SLCC1  
.NET CLR 3.0.04506)  
host : localhost:8080  
connection : Keep-Alive  
cache-control : no-cache  
cookie : JSESSIONID=8887A9A8F877E2C2A8D0076105CD2545
```

Le problème posé par cette technique se manifeste lorsque l'utilisateur désactive les cookies, dans ce cas on perdent les variables de sessions. Le serveur commence une nouvelle session pour toute requête qui arrive.

Pour pallier à ce problème la solution consiste à coder l'id de la session dans l'URL. L'interface `HttpServletResponse` offre des méthodes pour coder l'id de la session dans l'URL.

Méthode	Description
<code>String encodeURL(String url)</code>	Cette méthode retourne l'url avec l'id de la session. Elle est utilisée avec les url normales générées par un servlet
<code>String encodeRedirectURL(String url)</code>	Elle est utilisée avec les url passées en argument à la méthode <code>sendRedirect</code> de l'interface <code>HttpServletResponse</code> .

Envoie de la réponse

L'objet `httpServletResponse` et le moyen par lequel une servlet envoie les informations en réponse à une requête du client. Elle fournit un ensemble de méthodes qui permettent de manipuler la réponse :

Méthode	Description
<code>PrintWriter getWriter()</code>	Elle retourne un objet de type <code>Printwriter</code> utilisé pour envoyer des caractères vers le client.
<code>OutputStream getOutputStream()</code>	Pour envoyer des fichier binaires
<code>void setContentType(String type)</code>	Pour spécifier le type du contenu du message de réponse
<code>void setHeader(String name, String value)</code>	Pour spécifier un entête
<code>void sendRedirect(String location)</code>	Pour rediriger le client vers une adresse donnée.
<code>String encodeURL(String url)</code>	Voir la partie gestion des session
<code>String encodeRedirectURL(String url)</code>	Voir la partie gestion des session

Envoie de la réponse

- `getWriter()` : Cette méthode retourne un objet de type `java.io.PrintWriter` qui est utilisé pour envoyer des caractères au client, comme vous allez le remarquer cet objet est utilisé extensivement pour générer du code html.
- `getOutputStream()` : si vous voulez envoyer un fichier binaire comme un fichier exécutable vous aurez besoin d'un `OutputStream` au lieu d'un `PrintWriter`
- **Manipulation des entêtes** : On utilise les entêtes pour donner plus d'information sur la réponse comme : le type de contenu envoyé, la durée durant laquelle le navigateur peut cacher le contenu envoyé etc.
- **La redirection** : Après un traitement on peut rediriger le client vers une autres ressource : une autre servlet ou un contenu statique local ou sur un serveur distant.

Exemple 1 : `resp.sendRedirect("/PremiereServlet/login.html");`

Exemple 2 : `resp.sendRedirect("http://www.google.com");`

```
@Override
protected void doGet(HttpServletRequest req,
    HttpServletResponse resp)
    throws ServletException, IOException {
    resp.setContentType("application/rar");
    resp.setHeader("Content-Disposition",
        "attachment;filename=fichier.rar");
    File f=new File("c:\\app.rar");
    byte[] tab=new byte[(int)f.length()];
    FileInputStream finput=new FileInputStream(f);
    finput.read(tab);
    OutputStream out=resp.getOutputStream();
    out.write(tab);
    out.flush();
}}
```

À la réception d'une requête destinée à la servlet, le conteneur des servlets appelle la méthode `service(ServletRequest req, ServletResponse resp)` sur l'instance de la servlet chargée en mémoire.

Si la servlet ne reçoit aucune requête durant une longue période le conteneur des servlet invoque la méthode `destroy` pour faire le ménage puis décharge la servlet. On dit que la servlet se trouve dans l'état déchargé.

L'interface ServletContext est considérée comme une fenêtre à travers laquelle une servlet peut explorer son environnement. Une servlet peut utiliser cette interface pour obtenir des informations comme les paramètres d'initialisation de l'application web, la version du conteneur des servlets, elle offre des méthodes utilitaires pour charger les ressources partagés (comme les fichier de propriétés .properties).

Chaque application web possède un et un seul objet de type ServletContext, cet objet est accessible à toute ressource active de l'application, il est généralement utiliser pour partager les données entre servlets.

Dans cette section on va travailler avec les méthodes `getResource` et `getResourceAsStream`, ces méthodes sont utilisées par une servlet pour accéder à une ressource sans spécifier son chemin absolu.

Méthode	Description
<code>java.net.URL getResource (String path)</code>	Cette méthode retourne un objet de type <code>java.net.URL</code> pour la ressource représentée par le chemin passé en argument. Le chemin n'est pas absolue, il est relatif au dossier root de l'application web.
<code>java.io.InputStream getResourceAsStream (String path)</code>	Cette méthode est utilisée si on veut seulement obtenir un flux d'entrée qui pointe sur la ressource passée en argument. Elle est équivalente à <code>getResource(path).openStream()</code> .

ServletContext

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    response.setContentType("application/jar");
    response.setHeader("Content-Disposition", "attachment;filename=file.txt");
    OutputStream os = response.getOutputStream();

    byte[] bytearray = new byte[1024];
    ServletContext context = getServletContext();
    URL url = context.getResource("/files/file.txt");
    InputStream is = url.openStream();
    int bytesread = 0;
    while( (bytesread = is.read(bytearray) ) != -1 )
    {
        os.write(bytearray, 0, bytesread);
    }
    os.flush();
    is.close();
}
```


Initialisation du ServletContext

Une application web possède un seul contexte (instance de `javax.servlet.ServletContext`) , cet objet est initialisé au moment du chargement de l'application web, on peut définir des paramètres d'initialisation du contexte dans le descripteur du déploiement `web.xml` comme suit :

```
<web-app>
...
<context-param>
    <param-name>dburl</param-name>
    <param-value>jdbc:databaseurl</param-value>
</context-param>
...
</web-app>
```

Initialisation du ServletContext

Les paramètres d'initialisation du contexte sont utilisés pour spécifier des informations qui concernent toute l'application web comme des informations sur le développeur de l'application et les ressources partagées par toutes les servlets de l'application, à titre d'exemple : les paramètres de connexion à une base de données.

ServletContext offre les méthodes suivantes pour manipuler les paramètres d'initialisation du context :

Méthode	Description
String getInitParameter(String name)	Cette méthode retourne une chaîne de caractères contenant la valeur du paramètre ou null si le paramètre est inexistant.
java.util.Enumeration getInitParameterNames()	Cette méthode est utilisée pour obtenir une énumération des noms des paramètre d'initialisation du contexte.

Initialisation du ServletContext

Avant d'utiliser les méthodes précitées, vous devez tout d'abord obtenir une référence du Contexte soit à travers un objet de type ServletConfig ou directement en appelant la méthode `getServletContext()` hérité de `GenericServlet`.

```
public void init()  
{  
    ServletContext context =  
        getServletConfig().getServletContext();  
  
    //ServletContext context =    getServletContext();  
  
    String dburl = context.getInitParameter("dburl");  
    //use the dburl to create database connections  
}
```

Partager les données (portée des attributs)

Les servlets partagent les données entre eux suivant le concept du rendez-Vous, une servlet stocke les données dans des objets spécifiques pour cette tâche qui agissent comme des conteneurs, les autres servlets peuvent accéder à ces données via les conteneurs : `ServletRequest`, `HttpSession` et `ServletContext`.

Ces trois objets offrent les méthodes `setAttribute(String name, Object value)` et `Object getAttribute(String name)` respectivement pour stocker ou obtenir un objet.

Portée :

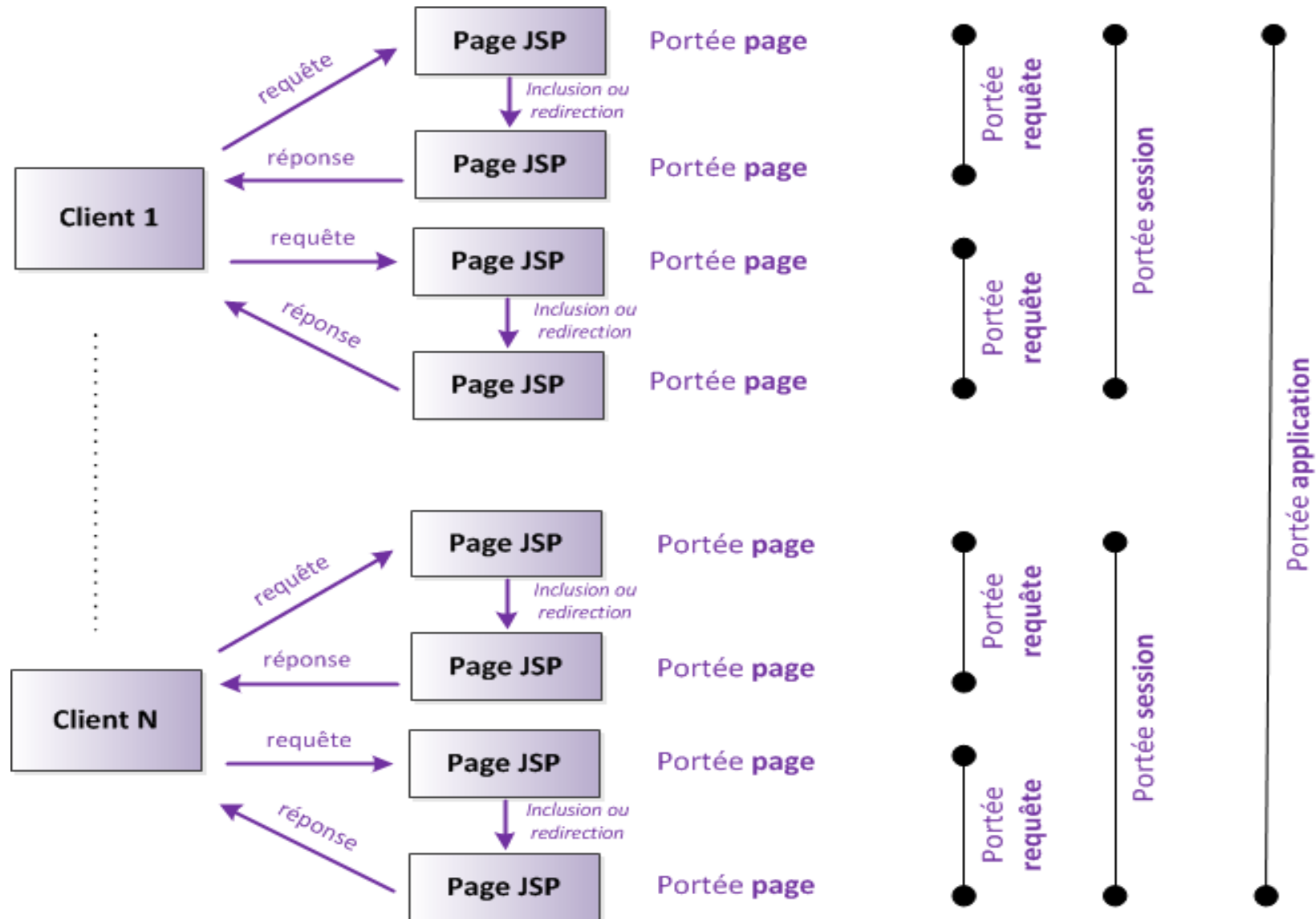


`ServletContext` : accessible durant le cycle de vie de l'application web.

`HttpSession` : accessible durant le cycle de vie de la session

`ServletRequest` : accessible durant le cycle de vie d'une requête http

Partager les données (portée des attributs)



Dans ce chapitre on a exploré en détail le modèle des servlet, depuis la création en passant par l'initialisation, l'exécution qui consiste à analyser la requête client et envoyé la réponse désirée du serveur vers le client jusqu'à la destruction. Dans le chapitre suivant on va voire une technologie complémentaire aux servlets : Java Server Pages.