Les Servlets

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en Programmation par contrainte (IA)
Ingénieur en Génie logiciel

elmouelhi.achref@gmail.com

Plan

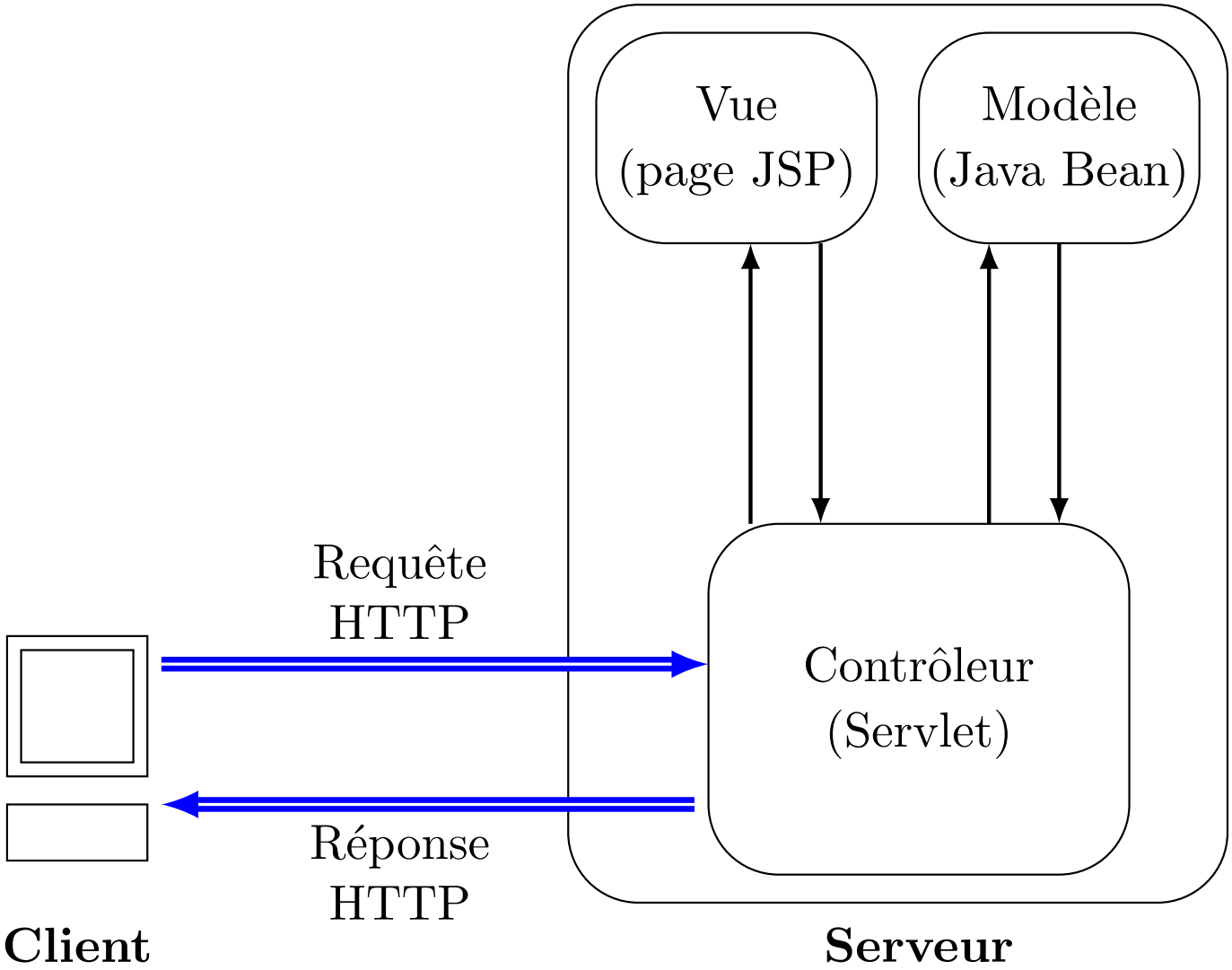
1. Introduction
2. Une première servlet
3. Le format d'une requête utilisateur
4. Tester la Servlet
5. Les paramètres de la requête
6. Rediriger vers une autre servlet

Introduction

Les servlets

- **est une classe Java** qui hérite de la classe `HttpServlet`
- reçoit des requêtes (`get`, `post`...) et retourne des réponses
- correspond au contrôleur du modèle MV**C** dans une application JEE

Servlet : le cœur d'une application JEE



Une première servlet

Trois étapes

- Création
- Déclaration
- Association d'une route à cette servlet
  - avec l'annotation `@WebServlet`
  - dans le fichier `web.xml`

Une première servlet

Création : déroulement

- 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 example) (*ce répertoire servira par la suite à mieux organiser notre application JEE en mettant les servlets ensemble*)
- Remplir le champ `Class name:` par un nom suffixé par le mot `Servlet` : TestServlet (par example)
- Cliquer sur `Next`

Une première servlet

Déclaration et routage avec annotation

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

```
package org.eclipse.controller; // package contenant les
   Servlets
@WebServlet("/mapage")
public class TestServlet extends HttpServlet {
  private static final long serialVersionUID = 1L;
  public TestServlet() { // le constructeur
    super();
  }
  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);
  }
}
```

Une première servlet

Explication

- Une Servlet hérite de la classe `HttpServlet` et doit implémenter au moins une de ses méthodes `do()`
  - `doGet()` : s'exécute quand l'utilisateur demande une page
  - `doPost()` : s'exécute quand l'utilisateur envoie des données via un formulaire par exemple
  - ...
- Chaque méthode prend en paramètre :
  - `HttpServletRequest` : permet de récupérer des informations sur la requête utilisateur
  - `HttpServletResponse` : permet de renvoyer une réponse à l'utilisateur suite à sa requête

Une première servlet

Déclaration et routage avec annotation

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

- déclarer la Servlet
- assurer le routage (ou le mapping) entre (URL/Servlet) (si cela n'a pas été fait avec les annotations)

Une première servlet

Déclaration et routage avec annotation

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

- déclarer la Servlet
- assurer le routage (ou le mapping) entre (URL/Servlet) (si cela n'a pas été fait avec les annotations)

Explication

- Quand l'utilisateur saisit une URL dans le navigateur, il envoie une requête HTTP à notre contrôleur (qui est en vrai une Servlet)
- Mais quelle Servlet ? je peux en avoir plusieurs
- Le serveur va chercher dans le fichier `web.xml` quelle Servlet correspond à cette URL

Une première servlet

Si le fichier n'existe pas

- Faire un clic droit sur `WEB-INF` de `WebContent` 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`

Une première servlet

Contenu du fichier `web.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
   http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd" id
   ="WebApp_ID" version="3.1">
  <display-name>jeeProject</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>
```

Une première servlet

On modifie `web.xml` en rajoutant avant `</web-app>` la déclaration de notre Servlet

```
...
<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

Une première servlet

Autres sous balises sont 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
- ...

Une première servlet

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)

Une première servlet

```
...
<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 la sous-balise précédente

Une première servlet

Le contenu de notre fichier `web.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
   http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd" id
   ="WebApp_ID" version="3.1">
<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>
```

Le format d'une requête utilisateur

Format d'une requête utilisateur

```
http://localhost:8080/nomProjetJEE/URLServlet
```

Comment récupérer ces informations

- `request.getContextPath()` : nom du projet défini par le serveur Apache Tomcat dans la requête
- `request.getServletPath()` : adresse de la servlet demandée par l'utilisateur (définie soit dans `web.xml` ou dans l'annotation `@WebServlet`)
- `request.getServerPort()` : numéro de port utilisé par le serveur

Tester la Servlet

Une seule étape à faire

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

Tester la Servlet

Si on teste une autre URL inexistante

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

Tester la Servlet

Comment afficher le `Hello World`

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

Tester la Servlet

```
package org.eclipse.controller;
public class TestServlet extends HttpServlet {
  private static final long serialVersionUID = 1L;
  public TestServlet() {
    super();
  }
  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);
  }
}
```

Tester la Servlet

Pour exécuter une deuxième fois

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

Tester la Servlet

Faisons les choses d'une façon plus chic

```
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

Tester la Servlet

Pour construire correctement une page HTML

```
  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>");
}
```

Tester la Servlet

Constat

- Beaucoup de code dans la Servlet (trop long) pour faire un simple affichage
- Cela ne respecte pas le modèle MVC : le contrôleur ne doit pas faire le rôle de la vue

Tester la Servlet

Constat

- Beaucoup de code dans la Servlet (trop long) pour faire un simple affichage
- Cela ne respecte pas le modèle MVC : le contrôleur ne doit pas faire le rôle de la vue

Solution

- Utiliser directement des vues pour l'affichage (chapitre suivant)

Récupérer les paramètres d'une requête

Récupérer les paramètres d'une requête

- Pour le moment, notre URL doit forcément être `/mapage`
- Mais, une requête peut avoir de paramètres (par example `/mapage?nom=Wick&prenom=John`)
- Comment, dans ce cas, récupérer les paramètres ?

Récupérer les paramètres d'une requête

Récupérer les paramètres d'une requête

- Pour le moment, notre URL doit forcément être `/mapage`
- Mais, une requête peut avoir de paramètres (par example `/mapage?nom=Wick&prenom=John`)
- Comment, dans ce cas, récupérer les paramètres ?

Solution

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

Récupérer les paramètres d'une requête

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);
}
```

Récupérer les paramètres d'une requête

À ne pas confondre

- Les paramètres de requête : un concept lié à la requête HTTP
- Les attributs de requête : un concept lié à la plateforme JEE (à voir dans le prochain chapitre)

Rediriger vers une autre servlet

Rediriger vers une autre servlet ayant l'url `/MaServlet`

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

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