Outils pour applications Web

Olivier Gutierrez

Java JEE

Olivier.Gutierrez

Java EE - Outils pour développer des applications Web

- Différence Java SE / Java EE
- L'architecture client serveur
- Servlets
- JSP
- Les formulaires
- Les filtres

Java SE vs Java EE

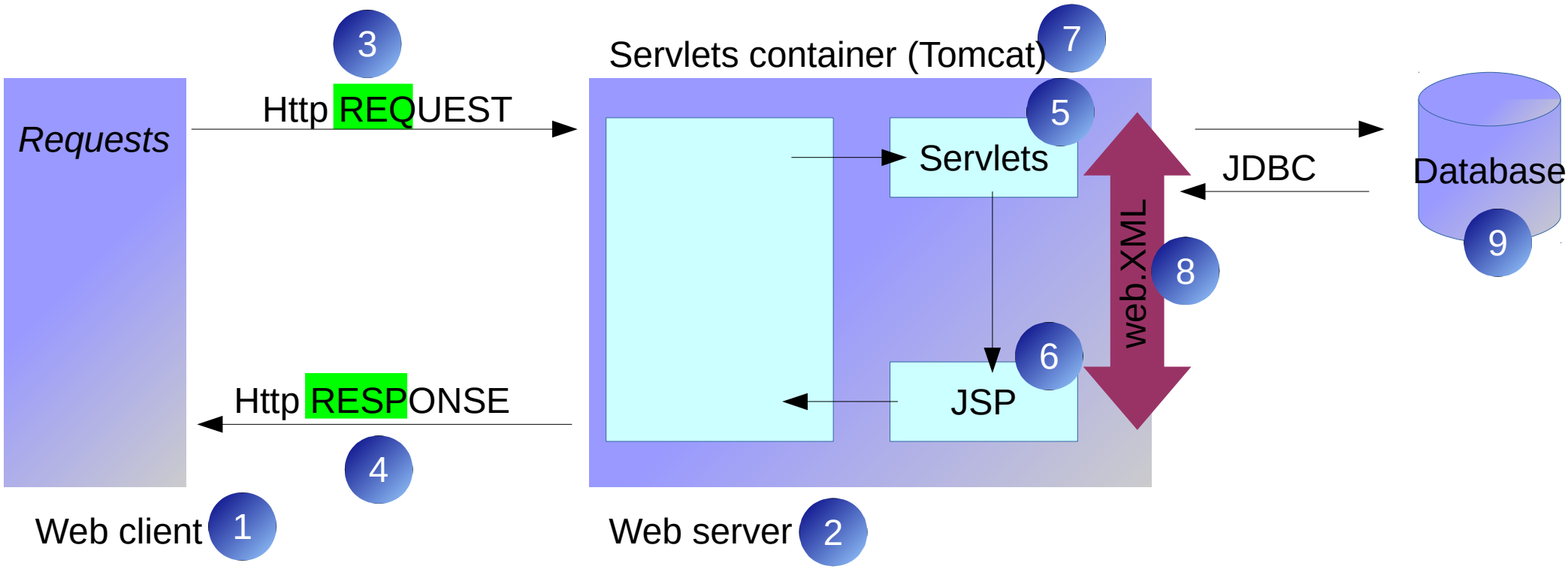
- Java *Standard Edition* pour les applications côté client
- Java *Enterprise Edition* pour les applications côté serveur

Java *Enterprise Edition*

- De nombreuses API avec un rôle spécifique
  - Servlets requêtes HTTP
  - JSP Java Server Pages pour générer du HTML
  - Pour exécuter une servlet il faut un conteneur spécifique
    - Apache Tomcat
    - JBoss
    - JonAS
    - Glassfish

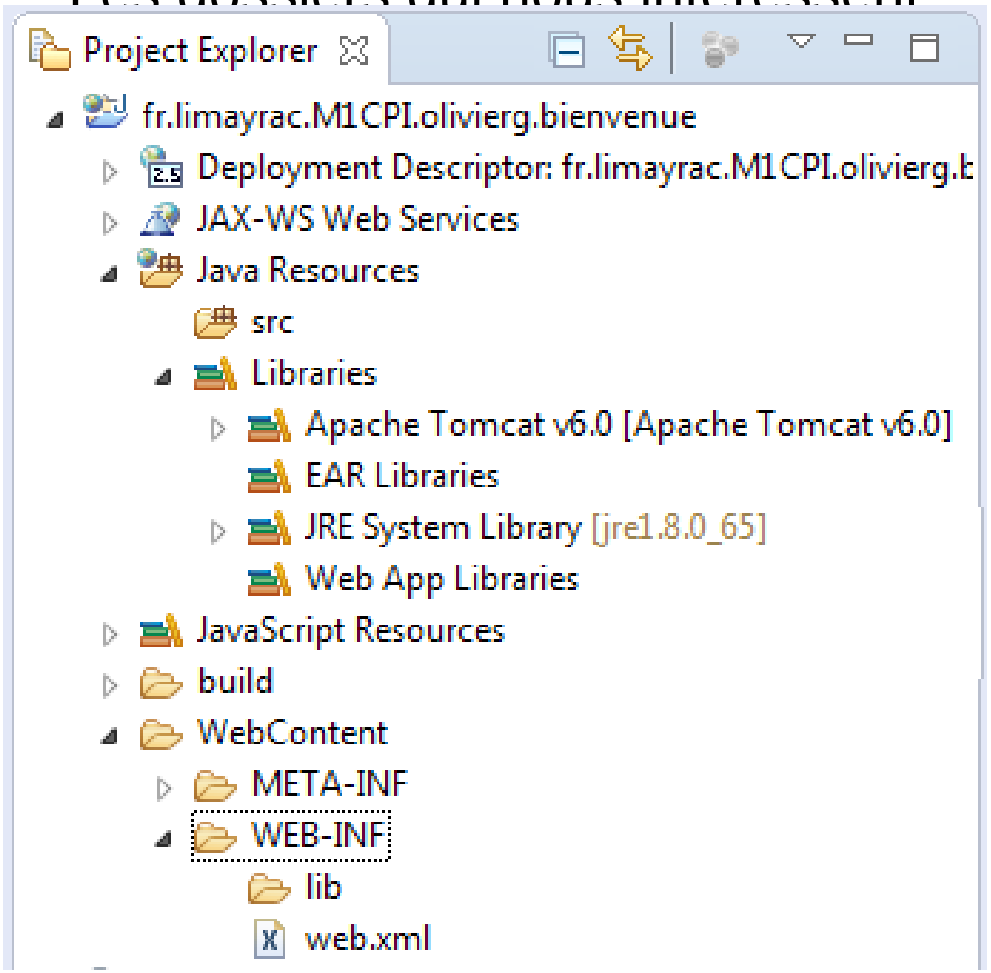
l'architecture client serveur

- Présentation de tous les composants

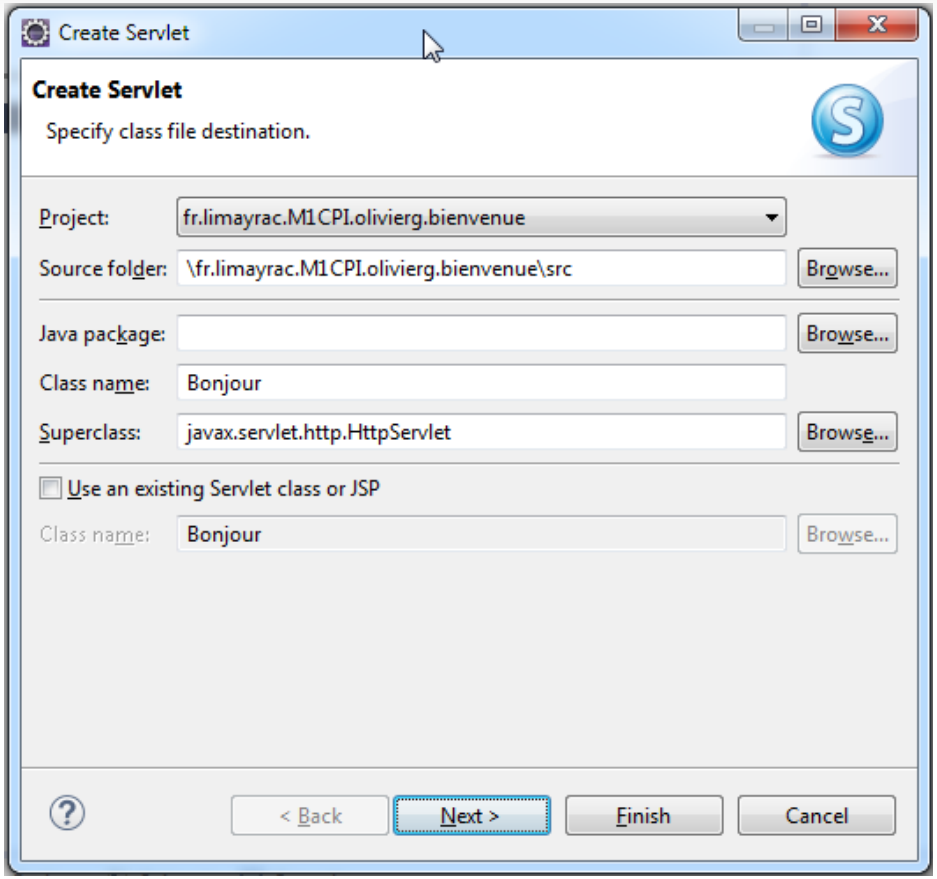


Structure d'un projet Eclipse JEE

- Les dossiers qui nous intéressent

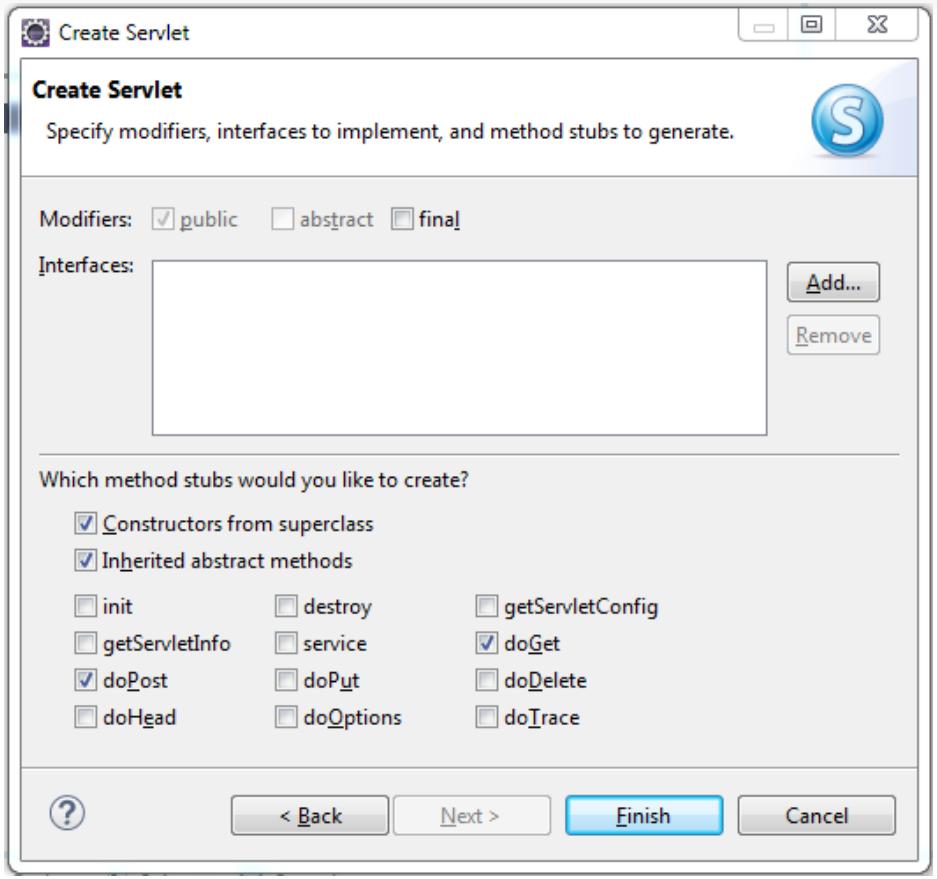


Servlets



- Les servlets doivent étendre la classe HttpServlet
- HttpServlet
  - Reçoit les objets request et response qui permettent de conserver le lien avec l'url (ces objets sont créés par le conteneur de servlets)
  - Fournit les méthodes qui vont permettre d'accéder à ces objets

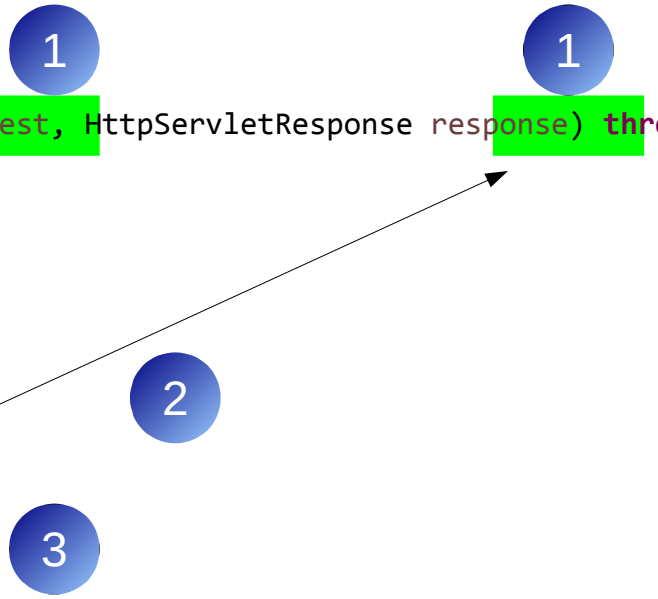
Servlets



Différentes méthodes peuvent être override (au minimum une) :

- doGet, pour gérer les demandes de l'utilisateur
- doPost, pour gérer les envois de données de l'utilisateur

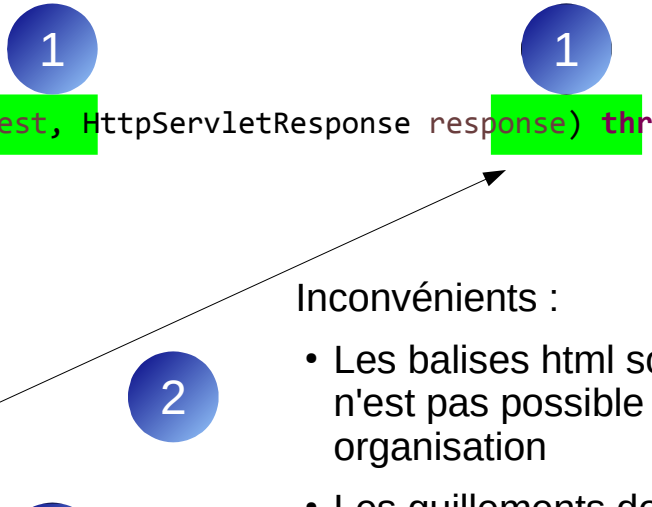
Servlets : Affichage de texte




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

// TODO Auto-generated method stub

// print dans la console

System.out.println("bienvenue (console");

/* print dans la page html */

// acces a la page html via l'objet response

PrintWriter sortie = response.getWriter();

// print simple

sortie.print("bienvenue (texte simple)");

// print utilisant html (commenter le print precedent)

sortie.print("<html><body><h1>"

+ "Bienvenue "
+ "<font color=\"#ff00ff\">"

+ "(texte utilisant html)</h1></body></html>");

}
```

Servlets : Affichage de texte

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

// TODO Auto-generated method stub

// print dans la console

System.out.println("bienvenue (console");

/* print dans la page html */

// acces a la page html via l'objet response

PrintWriter sortie = response.getWriter();

// print simple

sortie.print("bienvenue (texte simple)");

// print utilisant html (commenter le print precedent)

sortie.print("<html><body><h1>"

+ "Bienvenue "
+ "<font color=\"#ff00ff\">"

+ "(texte utilisant html)</h1></body></html>");

}
```



Inconvénients :

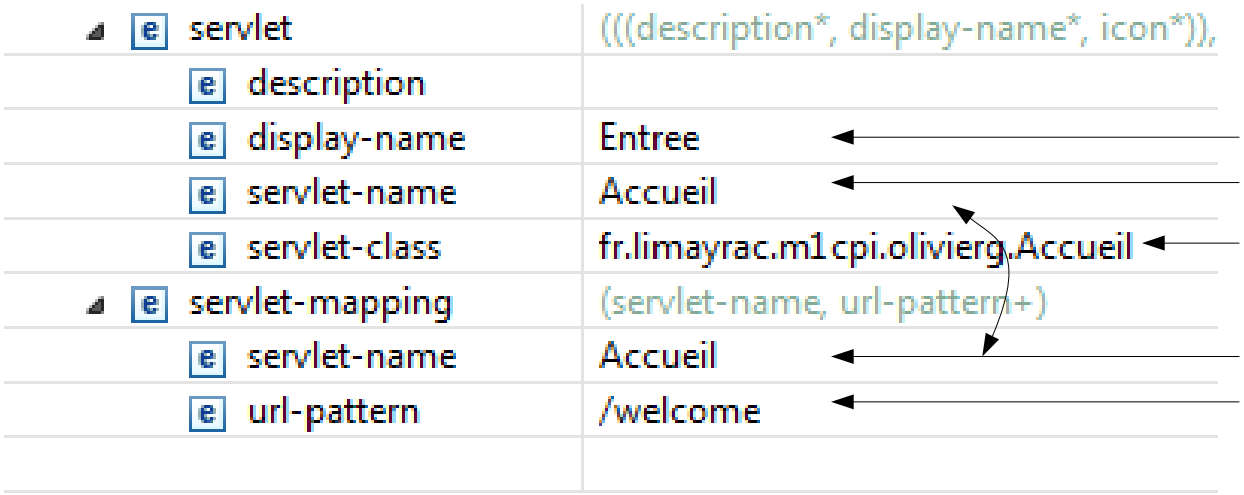
- Les balises html sont très difficiles à gérer et il n'est pas possible de les indenter selon leur organisation
- Les guillements doivent être échappés (source d'erreurs)
- Le modèle et le contrôleur sont dans le même fichier donc :
  - la modification de la vue implique une nouvelle compilation
  - Impossible que le designer travaille sur l'interface si le développeur modifie du code et réciproquement

Le fichier web.xml

servlet	(((description*, display-name*, icon*)),
description	
display-name	Accueil
servlet-name	Accueil
servlet-class	fr.limayrac.m1cpi.olivierg.Accueil
servlet-mapping	(servlet-name, url-pattern+)
servlet-name	Accueil
url-pattern	/Accueil

Le fichier suite à la création d'une classe 'Accueil'

Le fichier web.xml



Un texte qui peut être utilisé par les GUI

La classe elle même

Emplacement de la classe avec le reverse DNS

La classe qui est liée à l'url

l'url est changée

annotations

```java
/**
 * Servlet implementation class StartServlet
 */
@WebServlet("/StartServlet")
public class StartServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public StartServlet() {
        super();
        // TODO Auto-generated constructor stub
    }
```

L’annotation **`@WebServlet`** permet de lier une url à une servlet

Fichier JSP : Affichage de texte

```
<%@ page language="java" contentType="text/html;
charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h1>Bienvenue
    <font color= #ff00ff>
    (texte utilisant JSP)
    </font>
</h1>
</body>
</html>
```

à supprimer

à rajouter

Modification de la Servlet pour qu'elle appelle la JSP

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

        request.getRequestDispatcher("/WEB-INF/BienvenueMsg.jsp").forward(request, response);

    }
```

Permet de récupérer l'objet RequestDispatcher correspondant à l'argument

La méthode forward de l'objet RequestDispatcher transfère le contrôle à cet objet

Création d'un formulaire de login (côté JSP)

Dans la forme il faut préciser la méthode qui doit être appelée

```
<form method = "post">
   <input type="text" placeholder="username" name="nomUser"/>
   <input type="text" placeholder="password" name="pwd"/>
   <input type="submit"  value="login"/>
<p><a href="">mot de passe oublié ?</a></p>
</form>
```

Les champs textes doivent être associés à des variables

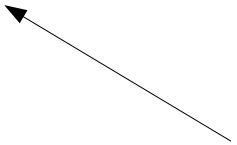
Création d'un formulaire de login (côté servlet)

```
   protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
      String utilisateur = request.getParameter("nomUser");
      String mdp = request.getParameter("pwd");
      System.out.printf("nom=%s mot de passe=%s",utilisateur,mdp);
   }
```

Création de 2 Strings qui utilisent la méthode getParameter de l'objet request (avec les noms définis dans la JSP)

Utilisation de filtres (côté JSP)

```
<filter>
  <display-name>LogFilter</display-name>
  <filter-name>LogFilter</filter-name>
  <filter-class>fr.limayrac.m1cpi.olivierg.filters.LogFilter</filter-class>
  <init-param>
      <param-name>test-param</param-name>
      <param-value>parametre pour test</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>LogFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```



Le pattern d'url permet d'indiquer quand le filtre doit se lancer (ici tout le temps)

Utilisation de filtres (côté servlet)

Méthode appelée à chaque accès

```
public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws
IOException, ServletException {
        String adresseIp = request.getRemoteAddr();
        System.out.println("adresse IP:"+adresseIp + "\t Date : "+new Date().toString());
        System.out.println();
        chain.doFilter(request, response);
    }

    public void init(FilterConfig fConfig) throws ServletException {
        String testParam = fConfig.getInitParameter("test-param");
         //Print the init parameter
         System.out.println("valeur de testParam au démarrage: " + testParam);
    }
```

Méthode appelée au démarrage de l'application