Les page JSP

Achref El Mouelhi

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

elmouelhi.achref@gmail.com

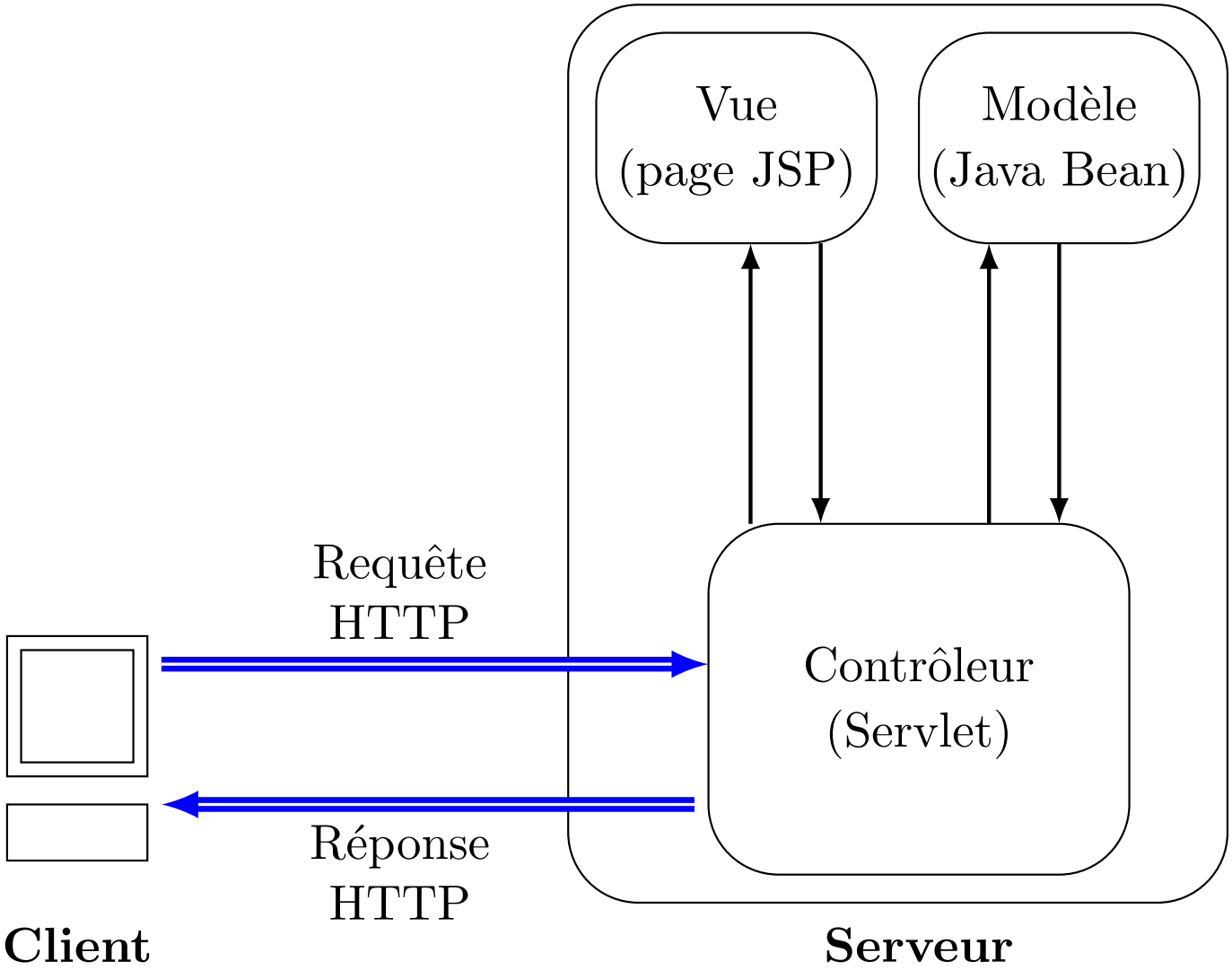
Plan

Introduction

JSP

- `Java Server Pages`
- Une technologie de la plateforme JEE permettant de créer dynamiquement des pages HTML (d'extension `.jsp`)
- Une page JSP sera transformée par le compilateur en Servlet
- Ensuite cette Servlet sera compilée par la JVM (machine virtuelle)
- Les JSP sont extensibles : on peut créer nos propres balises JSP (avec `JSTL`)

Une page JSP : appelée par le contrôleur



Création d'une page JSP

Déroulement

- Faire un clic droit sur `WEB-INF` de notre projet
- Aller dans `New` et choisir `JSP File`
- Remplir le champ `File name:` par vue.jsp (par exemple)
- Valider

Création d'une page JSP

Notre vue générée

```
<%@ page language="java" contentType="text/html; charset=
   UTF-8"
    pageEncoding="UTF-8"%>
<!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=UTF-8">
    <title>Insert title here</title>
  </head>
  <body>

  </body>
</html>
```

Création d'une page JSP

Préparons notre Hello World

```
<%@ page language="java" contentType="text/html; charset=
   UTF-8"
    pageEncoding="UTF-8"%>
<!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=UTF-8">
    <title>Projet JEE</title>
  </head>
  <body>
    Hello World (depuis une JSP)
  </body>
</html>
```

Création d'une page JSP

Comment l'appeler ?

- C'est toujours le contrôleur (Servlet) qui communique avec les vues

Création d'une page JSP

Pour construire correctement une page HTML

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

Explication

- `this.getServletContext()` : permet de communiquer avec d'autres composants (via le conteneur de Servlet)
- `getRequestDispatcher("/WEB-INF/vue.jsp")` : permet d'indiquer l'emplacement de la vue et de la récupérer
- `forward(request, response)` : pour envoyer la requête et la réponse (on les utilisera plus tard)

Initiation aux balises JSP

Observons notre vue

```
<!-- la seule balise JSP -->
<%@ page language="java" contentType="text/html; charset=
   UTF-8"
    pageEncoding="UTF-8"%>
<!-- le reste c'est du HTML -->
<!DOCTYPE HTML>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html;
       charset=UTF-8">
    <title>Projet JEE</title>
  </head>
  <body>
    Hello World (depuis une JSP)
  </body>
</html>
```

Initiation aux balises JSP

Les balises JSP

- sont définies par <% ... %>
- Entre ces deux balises, on peut utiliser les bases algorithmiques du langage Java :
  - des structures conditionnelles
  - des structures itératives
  - ...
- Les balises JSP peuvent être utilisées plusieurs fois dans une page JSP

Initiation aux balises JSP

Balises spéciales

- `<%-- ... --%>` : pour ajouter un commentaire
- `<%! String var; %>` : pour déclarer une variable directement dans la classe de la servlet.
- `<%= var %>` : pour afficher le contenu de la variable var $\equiv$ `<% out.println(var); %>`

Initiation aux balises JSP

Attention

Il est déconseillé de mélanger du code HTML avec du code JAVA

Les directives

Définition

- Les directives sont des instructions dans des balises JSP spéciales
- Elles ont la structure suivante :
  `<%@ directive {attribut="valeur"} %>`

Les directives

Définition

- Les directives sont des instructions dans des balises JSP spéciales
- Elles ont la structure suivante :
  `<%@ directive {attribut="valeur"} %>`

Rôle

- définir des données relatives à la page (directive page)
- inclure une autre page JSP (directive include)
- inclure des bibliothèques de balises (directive taglib)

Les directives

Définir des données relatives à la page

```
<%@ page language="java" contentType="text/html;␣
   charset=UTF-8"  pageEncoding="UTF-8"%>
...
<%@ page import="java.util.Date"  %>
```

Les directives

Définir des données relatives à la page

```
<%@ page language="java" contentType="text/html;␣
   charset=UTF-8"  pageEncoding="UTF-8"%>
...
<%@ page import="java.util.Date"  %>
```

Autres attributs

- `extends`
- `import`
- `session = "true | false"`
- `isELIgnored = "true | false"`
- ...

Les directives

Inclure le contenu d'une autre page JSP

```
<%@ include file="maPage.jsp" %>
```

ou

```
<jsp:directive.include file="maPage.jsp" />
```

Les directives

Inclure le contenu d'une autre page JSP

```
<%@ include file="maPage.jsp" %>
```

ou

```
<jsp:directive.include file="maPage.jsp" />
```

Différence entre les deux solutions

- Avec la première solution, le fichier sera chargé au moment de la compilation (donc le contenu de maPage sera recompilé avec le code de la page appelante)
- Avec la deuxième au moment de l'exécution

Les directives

Inclure le contenu d'une autre page JSP

```
<%@ include file="maPage.jsp" %>
```

ou

```
<jsp:directive.include file="maPage.jsp" />
```

Différence entre les deux solutions

- Avec la première solution, le fichier sera chargé au moment de la compilation (donc le contenu de maPage sera recompilé avec le code de la page appelante)
- Avec la deuxième au moment de l'exécution

Utilisation

Pour inclure (menu, entête...) qui sont généralement définis dans un fichier spécifique et qui sera inclus dans les autres fichiers de l'application (pour éviter le copier/coller et favoriser la réutilisation).

Les directives

Inclure des bibliothèques de balises (à voir dans un prochain chapitre)

```
<%@ taglib uri="maLib.tld" prefix="tag" %>
```

Récupérer les paramètres d'une requête

Comme pour les Servlets

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

Récupérer les paramètres d'une requête

```
<%@ page language="java" contentType="text/html; charset=
   UTF-8"     pageEncoding="UTF-8"%>
<!DOCTYPE HTML>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html;
       charset=UTF-8">
    <title>Projet JEE</title>
  </head>
  <body>
    Hello World (depuis une JSP)
    <%
      String nom = request.getParameter("nom");
      String prenom = request.getParameter("prenom");
      out.println("<br/>Hello " + nom + " " + prenom);
    %>
  </body>
</html>
```

Transmission de données entre Servlet/JSP

- Et si la Servlet veut transmettre des données (variables, objets...) à la vue ?
- On peut utiliser `request.setAttribute()` pour transmettre et `request.getAttribute()` pour récupérer
  - `request.setAttribute("nomAttribut","valeur")`
  - `request.getAttribute("nomAttribut")` : récupère l'objet ayant le nom `nomAttribut` qui doit correspondre au nom utilisé lors de l'envoi

Transmission de données entre Servlet/JSP

Envoi de données par la Servlet

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

     String ville = "Marseille";
     request.setAttribute("maVille",ville);
     // l'envoi de request se fait apres cette
        instruction
     this.getServletContext().getRequestDispatcher("/
        WEB-INF/vue.jsp").forward(request, response);
}
```

Transmission de données entre Servlet/JSP

Récupération de données par la JSP

```
<%@ page language="java" contentType="text/html; charset=
   UTF-8"     pageEncoding="UTF-8"%>
<!DOCTYPE HTML>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html;
       charset=UTF-8">
    <title>Projet JEE</title>
  </head>
  <body>
    <%
      String notreVille = (String) request.getAttribute("
         maVille");
      out.println("Bienvenue a " + notreVille);
    %>
  </body>
</html>
```

Transmission de données entre Servlet/JSP

Envoi d'un objet

```
protected void doGet(HttpServletRequest request,
   HttpServletResponse response) throws
   ServletException, IOException {
     Personne perso = new Personne();
     perso.setNom("Wick");
     perso.setPrenom("John");
     perso.setNum(100);
     request.setAttribute("perso",perso);
     this.getServletContext().getRequestDispatcher("/
        WEB-INF/vue.jsp").forward(request, response);
}
```

Transmission de données entre Servlet/JSP

Récupération de l'objet

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
        pageEncoding="UTF-8"%>
    <%@ page import = "org.eclipse.model.*" %>
<!DOCTYPE HTML>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset
       =UTF-8">
    <title>Projet JEE</title>
  </head>
  <body>
    <%
        Personne p = (Personne) request.getAttribute("perso");
        out.print("Hello " + p.getPrenom() + " " + p.getNom());
    %>
  </body>
</html>
```

Portée d'une variable

Quatre portée pour les variables

- `page` : la variable est accessible seulement dans cette page
- `request` : la variable est accessible seulement entre la servlet et la vue appelée
- `session` : la variable est accessible dans toutes les pages de l'application pour un utilisateur donné
- `application` : la variable est accessible dans toutes les pages de l'application et est partagée par tous les utilisateurs

Transmission de données entre Servlet/JSP

Et si on a besoin de créer un objet dans la page JSP

```
<jsp:useBean id="perso" scope="page" class="org.
   eclipse.model.Personne" >
</jsp:useBean>
```

Transmission de données entre Servlet/JSP

Et si on a besoin de créer un objet dans la page JSP

```
<jsp:useBean id="perso" scope="page" class="org.
   eclipse.model.Personne" >
</jsp:useBean>
```

Explication

- La balise précédente est équivalente en Java à `Personne perso = new Personne();`
- Notre objet est accessible seulement dans cette page JSP (`scope="page"`)
- Il faut que notre classe `Personne` soit un `JavaBean` : obligatoirement un constructeur sans paramètre

Transmission de données entre Servlet/JSP

Et si on a besoin de créer un objet dans la page JSP et affecter des valeurs aux attributs

```
<jsp:useBean id="perso" scope="page" class="org.eclipse.model.
   Personne" >
  <jsp:setProperty name="perso" property="nom" value="wick"/>
  <jsp:setProperty name="perso" property="prenom" value="john"/
     >
</jsp:useBean>
```

Transmission de données entre Servlet/JSP

Et si on a besoin de créer un objet dans la page JSP et affecter des valeurs aux attributs

```
<jsp:useBean id="perso" scope="page" class="org.eclipse.model.
   Personne" >
  <jsp:setProperty name="perso" property="nom" value="wick"/>
  <jsp:setProperty name="perso" property="prenom" value="john"/
     >
</jsp:useBean>
```

Ou aussi

```
<jsp:useBean id="perso" scope="page" class="org.eclipse.model.
   Personne" >
</jsp:useBean>

<%
  perso.setNom("wick");
  perso.setPrenom("wick");
%>
```

EL : Expression Language

Définition

- a été proposée par JSTL (Java server page Standard Tag Library)
- est disponible depuis la version 2.4 de l'API Servlet
- permet d'optimiser les pages JSP (simplifier le code)
- a une forme générale `${expression}`

EL : Expression Language

Définition

- a été proposée par JSTL (Java server page Standard Tag Library)
- est disponible depuis la version 2.4 de l'API Servlet
- permet d'optimiser les pages JSP (simplifier le code)
- a une forme générale `${expression}`

Rôle

- Réaliser des tests, des opérations arithmétiques
- Manipuler des objets, des collections,
- ...

EL : Expression Language

Réaliser des tests

```
${ true && true || false } <!-- affiche true -->
${ 'e' < 'f' } <!-- affiche true -->
${ 5 + 5 == 25 } <!-- affiche false -->
${ empty 'chaine' } <!-- affiche false -->
${ !empty 'chaine' } <!-- affiche true -->
${ !empty 'chaine' ? true : false  } <!-- test
   ternaire affichant true -->
```

Faire des opérations arithmétiques

```
${ 4 * 3 + 5 } <!-- affiche 17 -->
${ 8 % 2 } <!-- affiche 0 -->
```

Les résultats sont affichés là où l'EL est appelée

```
<div> 7 < 5 : ${ 7 < 5 } </div>
<div> 7 < 5 : false </div>
```

EL : Expression Language

Manipuler des objets

```
${ perso.nom } <!-- affiche Wick -->
${ perso.getPrenom() } <!-- affiche John -->
```

Explication

- `perso` est le nom d'objet qui a été ajouté à la requête comme attribut (avec `request.setAttribute()`
- `${ perso.nom }` est équivalent à `${ perso.getNom() }`

Transmission de données entre Servlet/JSP

Revenons au code précédent

```
<%@ page import = "org.eclipse.model.*" %>
<%
  Personne p = (Personne) request.getAttribute("
     perso");
  out.print("Hello␣" + p.getPrenom() + "␣" + p.
     getNom());
%>
```

On peut le remplacer par

```
  Hello ${perso.prenom} ${perso.nom}
```

Transmission de données entre Servlet/JSP

Revenons au code précédent

```
<%@ page import = "org.eclipse.model.*" %>
<%
  Personne p = (Personne) request.getAttribute("
     perso");
  out.print("Hello␣" + p.getPrenom() + "␣" + p.
     getNom());
%>
```

On peut le remplacer par

```
  Hello ${perso.prenom} ${perso.nom}
```

Même si l'objet ou un de ses attributs n'existe pas, `null` ne sera jamais affiché

EL : Expression Language

Considérons la liste suivante définie dans la Servlet

```
ArrayList<String> sport = new ArrayList<String>();
sport.add( "football" );
sport.add( "tennis" );
sport.add( "rugby");
sport.add( "basketball" );
request.setAttribute( "sport" , sport );
```

EL : Expression Language

Considérons la liste suivante définie dans la Servlet

```
ArrayList<String> sport = new ArrayList<String>();
sport.add( "football" );
sport.add( "tennis" );
sport.add( "rugby");
sport.add( "basketball" );
request.setAttribute( "sport" , sport );
```

Pour récupérer l'élément d'indice `i` dans la vue

```
sport.get(i);
sport[i];
sport['i'];
sport["i"];
```

EL : Expression Language

Considérons la liste suivante définie dans la Servlet

```
ArrayList<String> sport = new ArrayList<String>();
sport.add( "football" );
sport.add( "tennis" );
sport.add( "rugby");
sport.add( "basketball" );
request.setAttribute( "sport" , sport );
```

Pour récupérer l'élément d'indice `i` dans la vue

```
sport.get(i);
sport[i];
sport['i'];
sport["i"];
```

Exemple

```
J'aime le ${ sport.get(0) } et le ${ sport[3] }.
Je deteste le ${ sport['1'] } et le ${ sport["3"] }.
```

Objets implicites

Dans les exemples précédents

- on a utilisé des objets (implicites) sans les instancier.
  - `out` : pour afficher un message
  - `request` : pour récupérer des attributs et/ou des paramètres
- ces objets (et certains autres) ont déjà été instanciés dans la Servlet qui correspond à notre page JSP

Les objets implicites

Autres objets implicites pour la JSP

- `session` : permet de récupérer/écrire des données relatives à l'utilisateur courant
- `application` : permet d'obtenir/modifier des informations relatives à l'application dans laquelle elle est exécutée.
- `response` : permet de modifier des données relatives à la réponse (encodage...)
- `exception` : pour récupérer des informations sur l'exception capturée
- `...`

Les objets implicites

Les objets implicites de EL sont des `MAP`

- `sessionScope` : une `MAP` qui permet de récupérer/écrire des données relatives à l'utilisateur courant
- `param` : une `MAP` qui permet de récupérer/écrire les noms et valeurs des paramètres de la requête.
- `cookie` : une `MAP` qui permet d'associer les noms et instances des cookies.
- `...`

Les objets implicites

Le code JSP permettant de récupérer les paramètres d'une requête

```
<%
  String nom = request.getParameter("nom");
  String prenom = request.getParameter("prenom");
  out.println("<br/>Hello␣" + nom + "␣" + prenom);
%>
```

On peut le remplacer par

```
  Hello ${param.prenom} ${param.nom}
```

Gérer les exceptions

Considérant le code suivant (contenant une division par zéro)

```
<%
    int x = 3 / 0;
%>
```

À l'exécution, une exception est affichée

```
org.apache.jasper.JasperException: An exception
  occurred processing JSP page [/WEB-INF/vue.jsp]
  at line [11]

10:     <%
11:             int x = 3 / 0;
12:     %>
```

Gérer les exceptions

Il faut capturer l'exception

```
<%
   try {
      int x = 3 / 0;
   }
   catch(Exception e){
      out.print("Erreur " + e.getMessage());
   }
%>
```

Et le résultat est :

```
Erreur / by zero
```

Gérer les exceptions

Une deuxième solution consiste à

- créer une vue d'erreur
- rediriger vers cette page chaque fois qu'une exception est levée

Gérer les exceptions

La page `erreur.jsp`

```
<%@ page language="java" contentType="text/html;␣
   charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/
       html;␣charset=UTF-8">
    <title> Page d'erreur </title>
  </head>
  <body>
        Erreur
  </body>
</html>
```

Gérer les exceptions

Faisons référence à `erreur.jsp` dans `vue.jsp` (en ajoutant la ligne `errorPage="erreur.jsp"`) et supprimons le bloc `try ... catch`

```
<%@page import="org.apache.jasper.tagplugins.jstl.core.Out"%>
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8" errorPage="erreur.jsp" %>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset
       =UTF-8">
    <title>First Page</title>
  </head>
  <body>
    <%
      int x = 3 / 0;
    %>
  </body>
</html>
```

En exécutant, la redirection a eu lieu mais le message d'erreur a disparu

Gérer les exceptions

Pour afficher le message d'erreur, il faut modifier `erreur.jsp` **et déclarer la page comme page d'erreur** `isErrorPage="true"`

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8" isErrorPage="true" %>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset
       =UTF-8">
<title>Page d'erreur</title>
</head>
<body>
        Erreur
        <%=exception.getMessage() %>
</body>
</html>
```

Ne pas utiliser le navigateur d'**Eclipse** pour tester.