

JEE : Java Server Pages (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

- 1 Introduction
- 2 Créer une page JSP
- 3 Balises JSP
- 4 Directives
- 5 Récupérer les paramètres d'une requête
- 6 Transmission de données entre Servlet/JSP

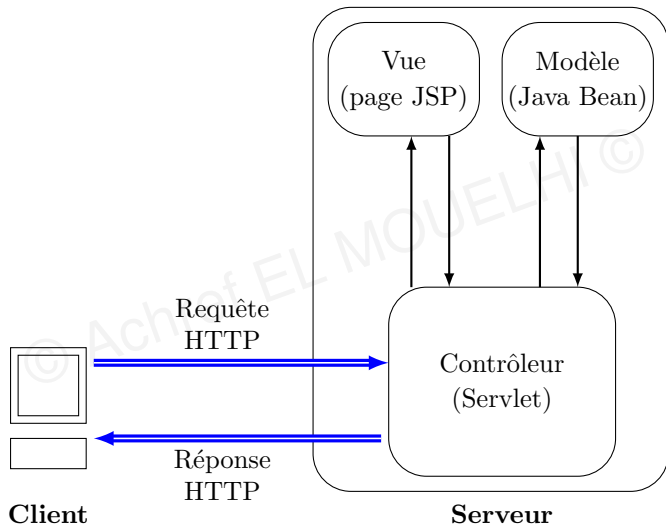
Plan

- 7 Portée d'une variable
- 8 Création d'un objet
- 9 EL : Expression Language
- 10 Objets implicites
- 11 Gérer les exceptions
- 12 Référencement d'une ressource statique
- 13 Extension Eclipse pour JSP

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
- Les **JSP** sont extensibles : on peut créer nos propres balises **JSP** (avec **JSTL**)

Jakarta EE



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

Jakarta EE

Contenu généré

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

</body>
</html>
```

Jakarta EE

Préparons notre Hello World

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


Question

Comment l'appeler ?

© Achref EL MOUADJID

Jakarta EE

Question

Comment l'appeler ?

Réponse

C'est la Servlet qui appelle la vue.

Jakarta EE

Pour construire correctement une page HTML

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

© Achref EL MOUELHI ©

Jakarta EE

Pour construire correctement une page HTML

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

Ou en utilisant l'objet de type `HttpServletRequest`

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

Jakarta EE

Pour construire correctement une page HTML

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

Ou en utilisant l'objet de type `HttpServletRequest`

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

Explication

- `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).

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**.

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); %>`

Attention

Il est déconseillé de mélanger du code **HTML** avec du code **Java**.

Jakarta EE

Directives

- Instructions dans des balises **JSP** spéciales

- Structure :

```
<%@ directive {attribut="valeur"} %>
```

© Achref EL MOU

Jakarta EE

Directives

- Instructions dans des balises **JSP** spéciales
- Structure :

```
<%@ directive {attribut="valeur"} %>
```

Rôle

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

Utiliser la directive `page` pour définir des données relatives à la page (code auto-généré à la création d'une JSP)

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

© Achref EL MOUELHI ©

Utiliser la directive `page` pour définir des données relatives à la page (code auto-généré à la création d'une JSP)

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

Utiliser l'attribut `import` pour importer une classe à utiliser dans la page

```
<%@ page import="java.util.Date" %>
```

© Achref EL MOUL

Utiliser la directive `page` pour définir des données relatives à la page (code auto-généré à la création d'une JSP)

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

Utiliser l'attribut `import` pour importer une classe à utiliser dans la page

```
<%@ page import="java.util.Date" %>
```

Autres attributs

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

Inclure le contenu d'une autre page JSP

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

ou

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

© Achref EL MOUELHI ©

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

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).

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

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

Comme dans les Servlets

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

Jakarta EE

Exemple

```
<%@ 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

Jakarta EE

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 après cette instruction
    request.getRequestDispatcher("/WEB-INF/vue.jsp").forward(request, response);
}
```

Jakarta EE

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 à " + notreVille);
    %>
  </body>
</html>
```

Jakarta EE

Définissons une classe `Personne` **dans** `org.eclipse.model`

```
package org.eclipse.model;

public class Personne {
    private int num;
    private String nom;
    private String prenom;

    // + constructeurs, getters, setters et toString
}
```

Jakarta EE

Envoi d'un objet

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

    Personne personne = new Personne(100, "Wick", "John");
    request.setAttribute("perso", personne);
    request.getRequestDispatcher("/WEB-INF/vue.jsp").forward(request, response);
}
```


Jakarta EE

Récupération de l'objet

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import = "org.eclipse.model.Personne" %>
<!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>
```

Exercice

Modifiez `CalculServlet` pour qu'elle réalise le traitement précédent et affiche le résultat dans une vue `calcul.jsp`

Quatre portées 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

Jakarta EE

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

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

© Achref EL MOUELHI

Jakarta EE

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

```
<jsp:useBean id="per" 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

Jakarta EE

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

```
<jsp:useBean id="per" 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>
```

© Achref EL MOUËL

Jakarta EE

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

```
<jsp:useBean id="per" 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="per" scope="page" class="org.eclipse.model.Personne">  
</jsp:useBean>  
  
<%  
  perso.setNom("wick");  
  perso.setPrenom("wick");  
%>
```

EL : Expression Language

- Proposée par **JSTL** (**J**akarta **S**tandard **T**ag **L**ibrary)
- Disponible depuis la version 2.4 de l'**API Servlet**
- Permettant d'optimiser les pages **JSP** (simplifier le code)
- Forme générale : `${ expression }`

Rôle

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

Jakarta EE

Les EL supportent plusieurs types du langage Java

- Long
- Double
- String : entouré par "... " ou '...'
- Boolean
- ...

© Achref

Jakarta EE

Les EL supportent plusieurs types du langage Java

- Long
- Double
- String : entouré par "... " ou '...'
- Boolean
- ...

Les EL permettent d'évaluer une expression arithmétique

```
${ 5 } <!-- affiche 5 -->  
${ 5.2 } <!-- affiche 5.2 -->  
${ "bonjour" } <!-- affiche bonjour -->  
${ 'bonjour' } <!-- affiche bonjour -->  
${ true } <!-- affiche true -->
```

Jakarta EE

Les EL permettent d'évaluer une expression arithmétique

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

© Achref EL MOUELHI ©

Jakarta EE

Les EL permettent d'évaluer une expression arithmétique

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

Les opérateurs arithmétiques

- + : addition
- - : soustraction
- * : multiplication
- / ou div : division
- % ou mod : reste de la division

Jakarta EE

On peut réaliser des tests en utilisant les opérateurs de comparaison

```
${ 'e' < 'f' } <!-- affiche true -->  
${ 5 + 5 == 25 } <!-- affiche false -->
```

© Achref EL MOUELHI ©

Jakarta EE

On peut réaliser des tests en utilisant les opérateurs de comparaison

```
${ 'e' < 'f' } <!-- affiche true -->  
${ 5 + 5 == 25 } <!-- affiche false -->
```

Opérateurs de comparaison

- `==` ou `eq` : pour tester l'égalité
- `!=` ou `ne` : pour tester l'inégalité
- `>` ou `gt` : supérieur à
- `<` ou `lt` : inférieur à
- `>=` ou `ge` : supérieur ou égal à
- `<=` ou `le` : inférieur ou égal à

Jakarta EE

On peut aussi enchaîner les tests en utilisant les opérateurs logiques

```
${ 2 == 5 || 3 == 4 } <!-- affiche false -->  
${ 2 < 5 && 5 >= 3 } <!-- affiche true -->
```

© Achref EL MOUËL

Jakarta EE

On peut aussi enchaîner les tests en utilisant les opérateurs logiques

```
${ 2 == 5 || 3 == 4 } <!-- affiche false -->  
${ 2 < 5 && 5 >= 3 } <!-- affiche true -->
```

Opérateurs logiques

- `&&` ou `and` : `et`
- `||` ou `or` : `ou`
- `!` ou `not` : `non`

Jakarta EE

Pour les chaînes de caractères, on peut utiliser l'opérateur `empty`

```
${ empty 'chaine' } <!-- affiche false -->
${ !empty 'chaine' } <!-- affiche true -->
${ !empty 'chaine' ? true : false } <!-- test
ternaire affichant true -->
```

© Achref EL M.

Jakarta EE

Pour les chaînes de caractères, on peut utiliser l'opérateur `empty`

```
${ empty 'chaine' } <!-- affiche false -->
${ !empty 'chaine' } <!-- affiche true -->
${ !empty 'chaine' ? true : false } <!-- test
ternaire affichant true -->
```

© Achref EL W.

Jakarta EE

Pour les chaînes de caractères, on peut utiliser l'opérateur `empty`

```
${ empty 'chaine' } <!-- affiche false -->  
${ !empty 'chaine' } <!-- affiche true -->  
${ !empty 'chaine' ? true : false } <!-- test  
ternaire affichant true -->
```

Les résultats sont affichés là où l'EL est appelée

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

Jakarta EE

EL simplifie la récupération des attributs ajoutés depuis la Servlet dans l'objet `request`

```
${ nom } <!-- affiche la valeur de la variable nom définie dans la  
Servlet appelante -->
```

© Achref EL MOUELHI ©

Jakarta EE

EL simplifie la récupération des attributs ajoutés depuis la Servlet dans l'objet `request`

```
${ nom } <!-- affiche la valeur de la variable nom définie dans la  
Servlet appelante -->
```

Avec les scriptlets, pour récupérer un objet

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

Jakarta EE

EL simplifie la récupération des attributs ajoutés depuis la Servlet dans l'objet `request`

```
${ nom } <!-- affiche la valeur de la variable nom définie dans la  
Servlet appelante -->
```

Avec les scriptlets, pour récupérer un objet

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

Avec EL, l'écriture a été simplifiée

```
${ 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() }`

© Achref L.

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() }`

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

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"];
```

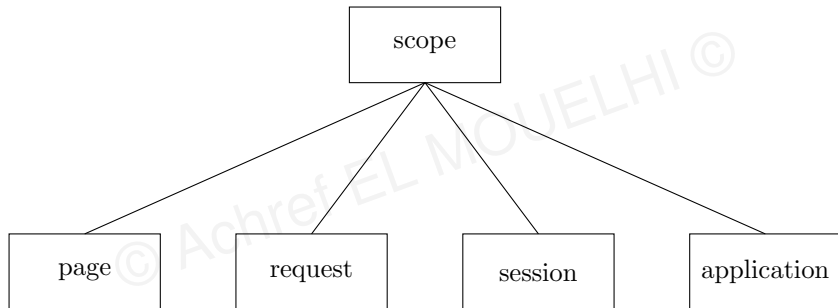
© Achref EL W.

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"] }.
```



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

Autres objets implicites

- `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 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.
- ...

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}
```

Jakarta EE

Considérant le code suivant (contenant une division par zéro)

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

© Achref EL MOUELHI

Jakarta EE

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:      %>
```

Jakarta EE

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

Une deuxième solution consiste à

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

Jakarta EE

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>
```

Jakarta EE

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

```
<%@ 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

Jakarta EE

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.

Ressource statique

- Fichier de style **CSS**
- Fichier de script **JS**
- Image
- ...

Appliquer un style à un paragraphe

- Dans `src/main/webapp`, créer un dossier `css`
- Créer un fichier `style.css` dans le dossier `css`
- Référencer `style.css` dans une vue

© Achref EL

Jakarta EE

Appliquer un style à un paragraphe

- Dans `src/main/webapp`, créer un dossier `css`
- Créer un fichier `style.css` dans le dossier `css`
- Référencer `style.css` dans une vue

Contenu de `style.css`

```
.first {  
    color: blue;  
}
```

Jakarta EE

Pour tester, référençons le fichier `style.css` et utilisons la classe CSS `first` dans n'importe quelle vue

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>hello.jsp</title>
    <link rel="stylesheet" href="css/style.css">
</head>
<body>
    <p class="first">Hello ${ prenom } ${ nom }</p>
</body>
</html>
```

Extension **Eclipse** pour **JSP**

- **Emmet** : extension d'auto-complétion pour les langages Web
- Pour l'utiliser, allez dans `Help > Eclipse Marketplace...` et installez l'extension Emmet (ex-Zen Coding)
- Pour l'activer, allez dans `Window > Preferences`, cherchez Emmet et ajoutez `.jsp` à la liste des extensions supportées.