Java Server Pages

Youssef Saadi

Master Informatique Décisionnelle

Faculté Des Sciences Et Techniques
Université Sultan Moulay Slimane Béni-Mellal

AU: 2019/2020

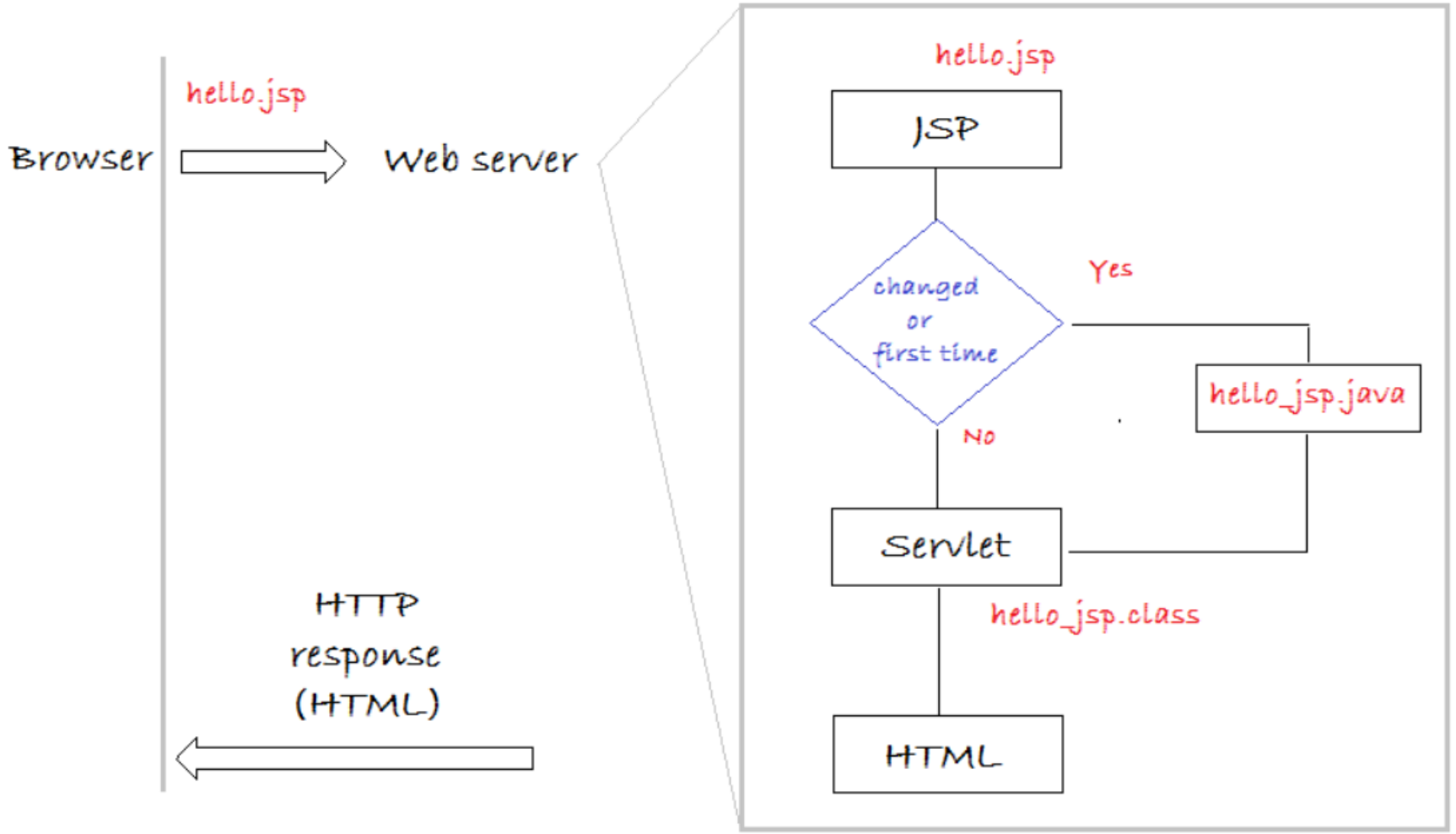
Introduction

- Une JSP : Une page **HTML** qui peut contenir du code **java** exécutable coté serveur afin de la rendre dynamique.
- Les **JSP** sont basées sur la technologie des **servlets**.
- Pages JSP sont converties en Servlet par le moteur de Servlets lors du premier appel à la JSP.
- Elles permettent une certaine séparation entre traitement de la requête et le rendu de la page (génération du flux HTML).
- Les JSP étant des documents, sont référencées par des URL:
  - **http://localhost:8080/AppliWeb_JSP/index.jsp**
- Les JSP sont destinées à la présentation (**mise forme**) et non à la réalisation de traitements lourds.

JSP : Page de présentation de contenu

- Elles permettent à des « développeurs Web » de travailler dans un environnement **Java EE** avec une connaissance superficielle de Java.
- **Séparation** des **rôles** et des tâches :
  - Un développeur Java réalise des composants (servlets, objets métiers ou techniques, etc…),
  - Un développeur de pages JSP utilise ces composants pour les présenter au format HTML, XML, etc…
- Ainsi, la **logique métier** est nettement séparée de la **présentation** des résultats.

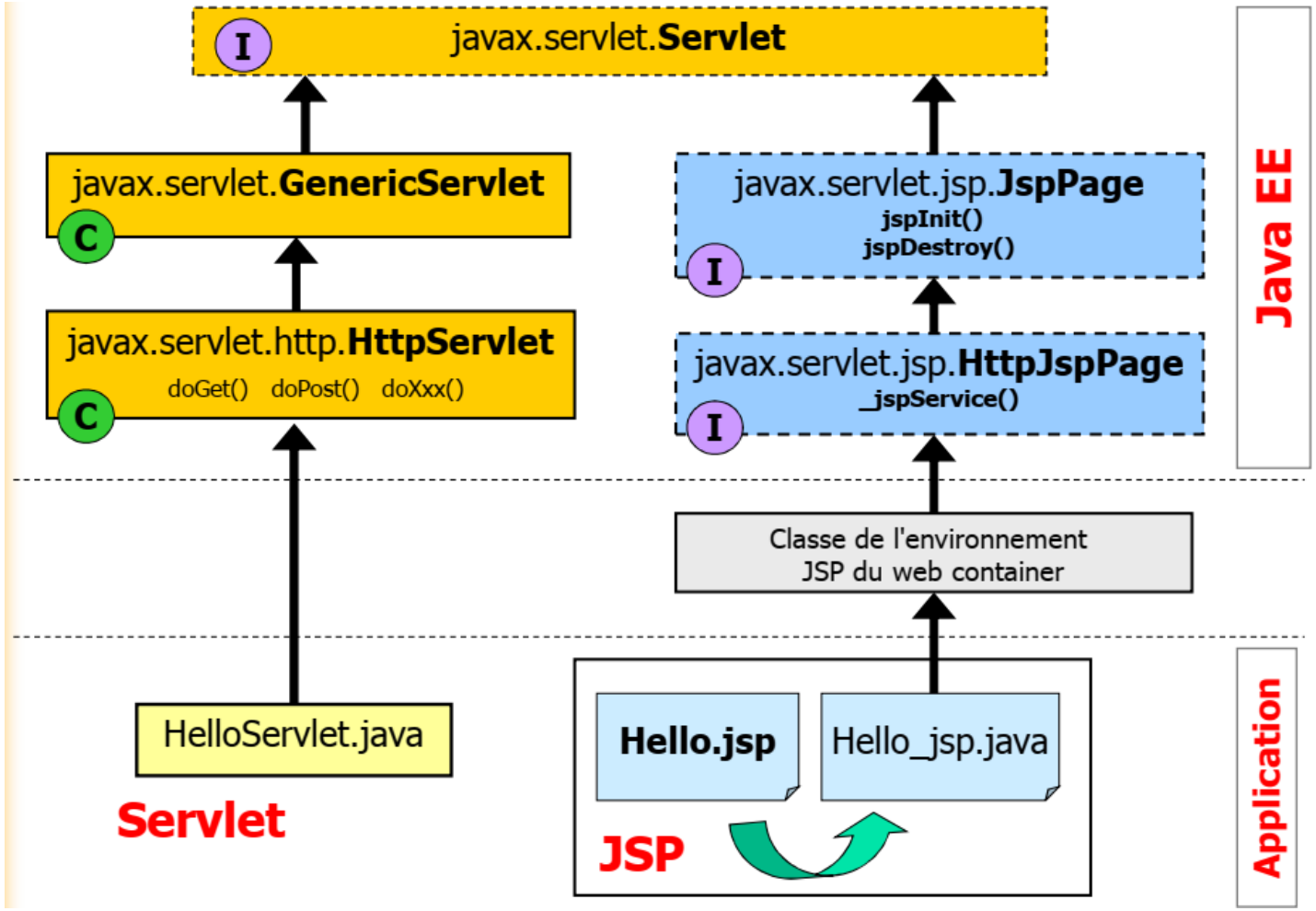
JSP : Mode de fonctionnement



JSP : Cas d'erreurs

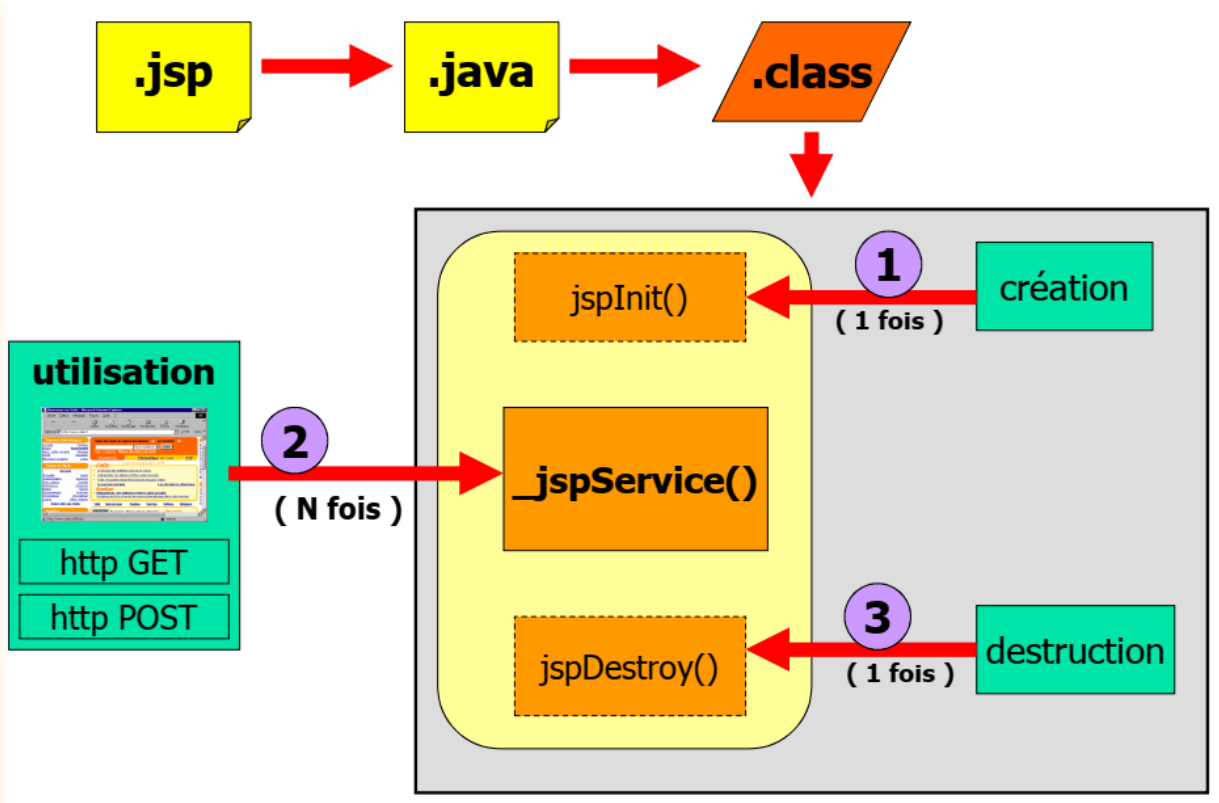
- Le codage des JSP ne permet pas d'anticiper les erreurs, elles ne sont détectées que lors de l'exécution.
- Deux cas d'erreurs avant exécution :
  - **Erreur de syntaxe JSP** :
    la génération de la servlet est impossible.
  - **Erreur dans le code Java** incrusté dans la JSP :
    la génération de la servlet a eu lieu, mais la compilation Java a échoué.
- Dans tous les cas, le serveur envoie un rapport sur le problème rencontré dans la page HTML générée.
  - Exemple : **Error 500 : Unable to compile class for JSP**

JSP : Classes et interfaces



JSP : Cycle de vie

- Appel de la méthode jspInit() après le chargement de la page
- Appel de la méthode _jspService() à chaque requête
- Appel de la méthode jspDestroy() lors du déchargement
- Il est possible de redéfinir dans la JSP les méthodes jspInit() et jspDestroy()



Eléments d'une page JSP

- Page JSP:
  – Html: structure statique de la page
  – Code JSP: éléments dynamiques de la page
- 4 types d'éléments de script:
  – Les directives : indiquent à la pages les informations globales (par exemple les instructions d'importation).
  – Les déclarations : destinées à la déclaration de méthodes et de variables à l'échelle d'une page.
  – Les scriplets : code Java intégré dans la page.
  – Les expressions : sous forme de chaine, en vue de leur insertion dans la sortie de la page.

Hello JSP

index.jsp | My First Jps Page

```jsp
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<html>
<head>
<title>My First Jps Page</title>
</head>
<body>
    <h1>Hello JSP</h1>

    <%
        java.util.Date date = new java.util.Date();
    %>

    <h2>
        Now is
        <%=date.toString()%>
    </h2>
</body>
</html>
```

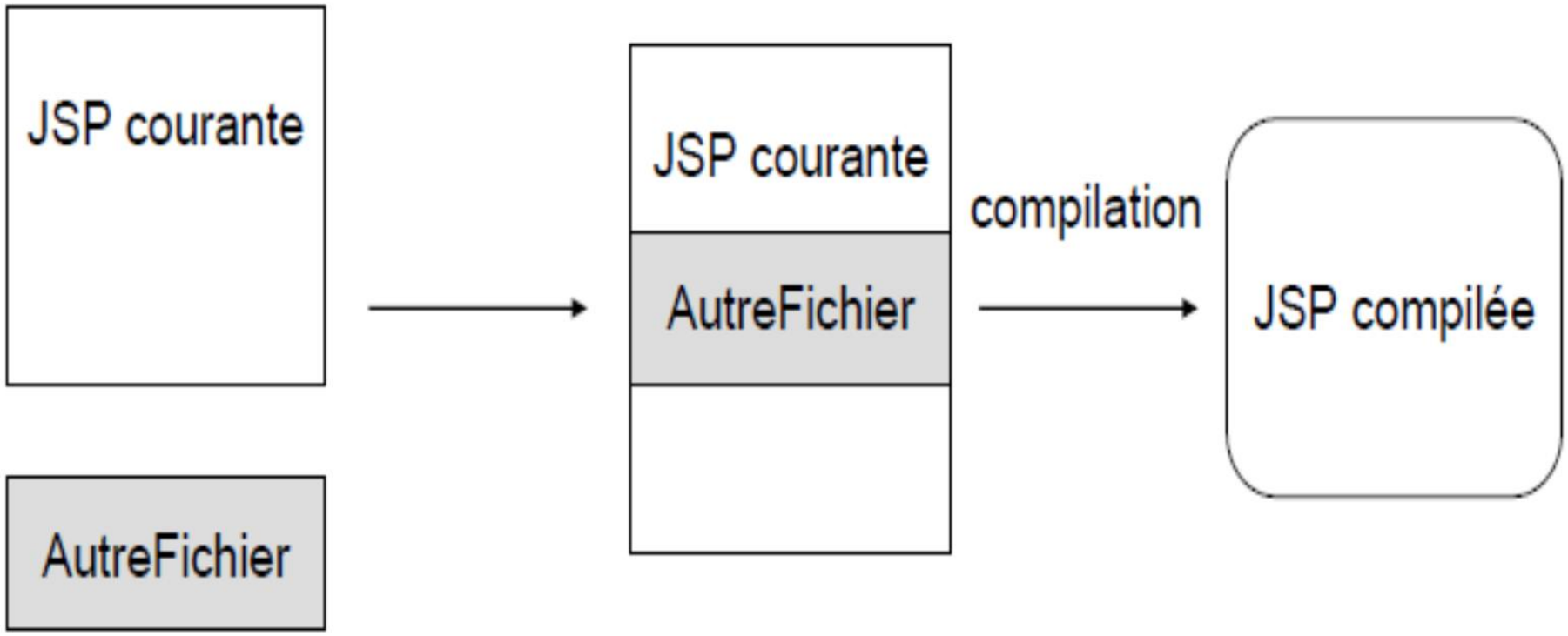
1- Les directives : <%@.....%>

- Page : informations relatives à la page
  - **<%@ page …%>**
- Include : fichiers à inclure littéralement. Inclusion effectuée avant la compilation de la jsp.
  - **<%@ include…%>**
- Taglib : URI d'une bibliothèque de balises utilisée dans la page.
  - **<%@ taglib…%>**

Les directives de page : <%@ page ...%>

- Définir les "import" nécessaires au code Java de la JSP :
  <%@ page **import**="java.io.*"%>
- Définir le type MIME du contenu retourné par la JSP :
  <%@ page **contentType**="text/html"%>
- Fournir l'URL de la JSP à charger en cas d'erreur :
  <%@ page **errorPage**="err.jsp"%>
- Définir si la JSP est une page invoquée en cas d'erreur :
  <%@ page **isErrorPage**="true" %>
- Déclarer si la JSP peut être exécutée par plusieurs clients à la fois :
  <%@ page **isThreadSafe**="false" %>

Les directives d'inclusion
<%@ include file="AutreFichier"%>



Les balises personnalisées :<%@ taglib ……%>

- Permettent d’indiquer une bibliothèque de balises : adresse et préfixe, pouvant être utilisées dans la page.
- **<%@ taglib prefix="pref" uri="taglib.tld" %>**

2- Les déclarations : <%! …..%>

- Permettent de déclarer des méthodes et des variables d'instance connus dans toute la page JSP.

```
3 <%!
4 private int uneVar;
5 private int carre(int var){
6     return var*var;
7 }
8 %>
```

3- Les scriplets <%.....%>

- Permettent d'insérer des blocs de code java (*qui seront placés dans _jspService(…)*)

```
<%
    java.util.Date date = new java.util.Date();
    int c = carre(2);
%>
```

Objets implicites

- Donnent accès à une liste d'objets implicites à partir de l'environnement de la servlet :
  - **request** : requête du client (classe dérivée de **HttpServletRequest**).
  - **response** : réponse de la page JSP (classe dérivée de **HttpServletRsponse**).
  - **session** : session HTTP correspondant à la requête.
  - **out** : objet représentant le flot de sortie.
  - **application** : espace de données partagé entre toutes les JSP (**ServletContext**).
  - **page** : l'instance de servlet associée à la JSP courante (**this**).
  - Etc..

4- Les expressions <%=.....%>

- Permettent d'évaluer une expression et renvoyer sa valeur (String).
- Correspond à out.println(…);
-

```
<%
    java.util.Date date = new java.util.Date();
    int c = carre(2);
%>

        <h2>
            Now is
            <%=date.toString()%>
        </h2>
```

JSP : les commentaires <%--……--%>

- Permettent d'insérer des commentaires (*qui ont l'avantage de ne pas être visibles pour l'utilisateur*)

```
3 <!-- il s agit d un commentaire -->
```

Les éléments d'actions **<jsp: ...../>**

- Permettent de faire des traitements au moment où la page est demandée par le client :
  - Inclure dynamiquement un fichier.
  - Utiliser des Java Bean.
  - Rediriger vers une autre page.
- Constitués de balises pouvant être intégrées dans une page JSP(syntaxe XML).
- Actions JSP standards:
  - jsp:include et jsp:param
  - jsp:forward
  - jsp:useBean
  - jsp:setProperty et jsp:getProperty

JSP : include/param

- **jsp:include** : identique à la directive <%@ include … ……%/>
  – Sauf que l'inclusion est faite au moment de la requête.
  – Donc après compilation…
- **jsp:param** : permet de passer des informations à la ressource à inclure

```
<jsp:include page="file.jsp" flush="true">
    <jsp:param name="nom" value="yoyo" />
    <jsp:param name="prenom" value="saa" />
</jsp:include>
```

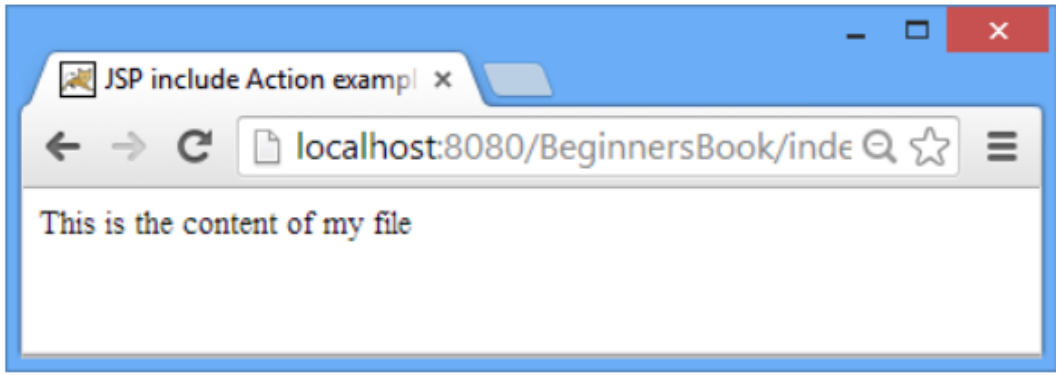
JSP : include/param

```
<html>
<head>
<title>JSP include Directive example</title>
</head>
<body>
<%@ include file="display.jsp" %>
</body>
</html>
```

```
<html>
<head>
<title>JSP include Action example</title>
</head>
<body>
<jsp:include page="display.jsp" />
</body>
</html>
```

display.jsp

```
<p>This is the content of my file</p>
```



JSP : forward

- Permet de passer le contrôle de la requête à une autre ressource.
- **jsp:param** permet ici aussi de passer des informations à la ressource de redirection.

```
<jsp:forward page="file.jsp">
    <jsp:param name="nom" value="yoyo" />
</jsp:forward>
```

JSP : useBean

- Permet de séparer la partie traitement de la partie présentation.
- Permet d'instancier un composant JavaBean (classe java) qui pourra être appelé dans la page JSP.
- Un Java Bean permet de coder la logique métier de l'application web. L'état d'un Bean est décrit par des attributs appelés propriétés.
- Un Java Bean est une classe Java respectant un ensemble de directives :
  - Un constructeur public sans argument.
  - Des propriétés « prop » accessibles au travers de getters et setters: getProp (lecture) et setProp (écriture) portant le nom de la propriété.

Exemple : Java Bean

```
public class Personne {
    private int age;
    private String name;

    public Personne() {
        this.age = 0;
        this.name = "";
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

Propriétés

Accesseurs

Jsp:useBean

```
<jsp:useBean id="monBean" class="com.youssadi.tuto.Personne" scope="session"></jsp:useBean>
```

Nom de l'instance du Bean

Classe du Bean

Portée du Bean : request, session; application, page

JSP : get/set property

- **jsp:setProperty** et **jsp:getProperty**
- Permet de récupérer ou de modifier les valeurs d'une instance de JavaBean.

```
<body>
    <jsp:useBean id="monBean" class="com.youssadi.tuto.Personne"
        scope="request"></jsp:useBean>

    <h3>My name is</h3>
    <jsp:getProperty name="monBean" property="name" />

    <h3>My new age is :</h3>
    <jsp:setProperty name="monBean" property="age" value="30" />
    <jsp:getProperty name="monBean" property="age" />

</body>
```

Les tags personnalisés

- Une directive "**taglib**" sert à indiquer que la JSP courante utilise une librairie de tags spécifiques.
- Cette directive doit être placée avant toute utilisation d'un tag spécifique.
- Syntaxe : **<%@ taglib uri="URI_lib_tag" prefix="nom_prefix" %>**
- Les tags définis dans cette librairie sont des "tags préfixés", le préfixe utilisé est précisé dans l'attribut "**prefix**".
- Il est possible d'utiliser plusieurs directives "**taglib** " (pour utiliser plusieurs librairies de tags dans la même JSP), mais chaque "**prefix**" doit être unique.

Les tags personnalisés

Fichier XML (.tld)

```
<%@ taglib  uri="http://www.soc.com/tag" prefix="ihm" %>

<ihm:infos>
....
</ihm:infos>

<ihm:bouton_standard nom="B_Valid"  lib="Validation"  />

<ihm:bouton_image  nom="B_Annul"  type="2"  />

<ihm:bouton_standard nom="B_Valid"
     lib="<%= rs.getString(x)%>"  />
```

Les tags JSTL et EL

- Grâce aux «taglibs» de nombreuses librairies ont été développées.
- Certaines librairies ont été normalisées par le **JCP**
  - **JSTL** : **J**avaServer Pages **S**tandard **T**ag **L**ibrary.
- Parallèlement un "langage d'expression" à été mis au point pour faciliter l'utilisation des objets Java dans les JSP.
  - **EL** : **E**xpression **L**anguage

EL (Expression Language)

- « **EL** » permet d'évaluer et de récupérer le **résultat d'une expression** insérée à n'importe quel endroit d'une JSP, ou dans un attribut d'un tag de la JSTL.
- Expression encadrée par **${ … }**
  - **<h2> Nom : ${employe.nom} </h2>.**

EL : Accès aux objets

- Accès aux **propriétés** d'un objet :
  - Par l'attribut : **${ employe.nom }**
    - Equivalent à un appel à la méthode **getNom()** de l'objet identifié par le nom symbolique **« employe ».**
  - Par l'indice : **${ liste[2]}** **p**our les tableaux.
  - Par le nom : **${ param['nom'] }** Pour les maps.
  - **NB** : les variables Java définies dans la page ne sont pas accessibles. Ne sont accessibles que :
    - Les **objets implicites,**
    - Les **objets nommés** stockés dans certains "**scopes**".

EL : Objets implicites

- Objets implicites = objets prédéfinis immédiatement utilisables dans les expressions :
  - Contexte de la page JSP en cours : **pageContext** donne accès à tous les autres objets
- Paramètres d'initialisation de la JSP :
  - **initParam** (« init-params » du web.xml)
- Paramètres de la requête http :
  - **param** (paramètres de la requête : Map String),
  - **paramValues** (paramètres de la requête : Map String[]),
  - **header** (headers de la requête : Map String),
  - **headerValues** (headers de la requête : Map String[]),
  - **Cookie.**

EL : Objets implicites - Exemples

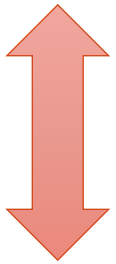
- **pageContext** donne accès aux autres objets classiquement utilisés dans une JSP : request, response, servletContext, etc…
  - **${pageContext.response.contentType}**
  - **${pageContext.servletContext.serverInfo}**
  - **${pageContext.request.contextPath}**
- L'accès aux éléments d'une Map peut se faire par **map["nom"]** ou par **map.nom** :
  - **${param["action"]} ou ${param.action}**
  - **${header["user-agent"]}**
  - **${header.user-agent}** → Ne fonctionne pas car le "-" est un opérateur

EL : Objets implicites & "scope"

- Dans une expression EL, un identifiant fait référence au nom d'un objet pouvant être stocké au niveau ..
  - **page → request → session → application**
- Objets implicites E.L :
  - **pageScope → requestScope →sessionScope → applicationScope**
- « **EL** » remplace donc efficacement les tags **<jsp:useBean ..>, <jsp:getProperty..>** et les expressions **<%=** ..Java.. **%>**.

EL : Exemple

```
<%@ page import="my.package.Societe" %>
...
<h3> Employe
<%= ((Societe)request.getAttribute("soc")).getNom()%>
</h3>
```



```
<h3> Employe ${requestScope.soc.nom} <h3>
<h3> Employe ${soc.nom} <h3>
```

EL : Ordre de scope

- En l'absence de « **scope** », l'ordre de recherche des attributs est : page, request, session et application (utilise le premier trouvé).
  - Si **non trouvé** "" (chaîne vide), pas d'erreur => erreurs masquées !
- **<h3> Employe ${sessionScope.employe.nom} <h3>**
- Référence d'un objet à partir d'un autre :
  - **<h2> Nom : ${employe.societe.ville} </h2>**

EL : Opérateurs

- EL dispose d'un ensemble d'opérateurs classiques :
  - Relationnels : **>, <, ==, !=, <=, >=**
  - Arithmétiques : **+, -, /, *, %**
  - Logiques : **&&, ||, !**
  - Vide ou **null** : **empty**

```
<h2> Prix : ${article.prix * 1.2 } </h2>
<c:if test="${bean1.a < 3}" />
<c:if test="${ !empty employe.nom }" />
<h2> Admin : ${employe.cadre && employe.manager}
</h2>
```

JSTL

- Principe : on remplace les balises et le code Java par du XML spécifique.
- Expression language pour JSP 2.1
- URL de base : http://java.sun.com

Library	TLD	Prefix
« **core** » (fonctions de base) uri = « http://java.sun.com/jsp/jstl/core »	**c.tld**	**c**
« **xml** » (traitements xml) uri = « http://java.sun.com/jsp/jstl/xml »	**x.tld**	**x**
« **fmt** » (formatage, internationalisation) uri = « http://java.sun.com/jsp/jstl/fmt »	**fmt.tld**	**fmt**
« **sql** » (accès aux SGBD, traitements SQL) uri = « http://java.sun.com/jsp/jstl/sql »	**sql.tld**	**sql**
« **functions** » (fonctions) uri = « http://java.sun.com/jsp/jstl/functions »	**fn.tld**	**fn**

Documentation :
http://tomcat.apache.org/taglibs/index.html

JSTL Core → préfixe : c

- Gestion des Variable de scope: remove, set
- Les structures de contrôles: choose, forEach, forTokens, if
- Gestion des URL: import, redirect, url
- Capture et affichage des exceptions : catch, out

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
```

JSTL Core : Les tags basiques

Tag	Description
<c:out>	To write something in JSP page, you can use EL also with this tag
<c:import>	Same as <jsp:include> or include directive
<c:redirect>	redirect request to another resource
<c:set>	To set the variable value in given scope.
<c:remove>	To remove the variable from given scope
<c:catch>	To catch the exception and wrap it into an object.
<c:if>	Simple conditional logic, used with EL and you can use it to process the exception from <c:catch>
<c:choose>	Simple conditional tag that establishes a context for mutually exclusive conditional operations, marked by <c:when> and <c:otherwise>
<c:when>	Subtag of <c:choose> that includes its body if its condition evalutes to 'true'.
<c:otherwise>	Subtag of <c:choose> that includes its body if its condition evalutes to 'false'.
<c:forEach>	for iteration over a collection
<c:forTokens>	for iteration over tokens separated by a delimiter.
<c:param>	used with <c:import> to pass parameters
<c:url>	to create a URL with optional query string parameters

Exemple -- c:forEach

```
<c:forEach items="${departments}" var="dept">
    <h3>${dept.deptName}</h3>
    <ul>

        <c:forEach items="${dept.employees}" var="emp">
            <li>${emp.empName} - (${emp.job})</li>
        </c:forEach>
    </ul>

</c:forEach>
```

Exemple -- c:if

```
<c:forEach items="${departments}" var="dept">

    <!-- Check if collection is not null or not empty -->
    <c:if test="${not empty dept.employees}">
        <h3>${dept.deptName}</h3>
        <ul>

            <c:forEach items="${dept.employees}" var="emp">
                <li>${emp.empName} - (${emp.job})</li>
            </c:forEach>
        </ul>
    </c:if>

</c:forEach>
```

Exemple -- c:choose

```
<c:choose>
    <%-- When color parameter == 'red' --%>
    <c:when test="${param.color=='red'}">
        <p style="color: red;">RED COLOR</p>
    </c:when>

    <%-- When color parameter == 'blue' --%>
    <c:when test="${param.color=='blue'}">
        <p style="color: blue;">BLUE COLOR</p>
    </c:when>

    <%-- Otherwise --%>
    <c:otherwise>
        <b>Other color</b>
    </c:otherwise>
</c:choose>
```

Exemple -- c:set

```
<h2>c:set example</h2>

<c:set scope="request" var="greeting" value="Hello every body" />

Greeting: <c:out value="${greeting}"/>
```

Exemple -- c:remove

```
<h2>c:remove example</h2>

<c:set scope="request" var="greeting" value="Hello every body" />


Greeting:
<c:out value="${greeting}" />
<br />
<br />

<c:remove scope="request" var="greeting" />
After remove:
<br /> Greeting:
<c:out value="${greeting}" />
```

Exemple -- c: catch

```
<h2>c:catch example</h2>

<c:catch var="ex">
    <%
        int a = 100 / 0;
    %>
</c:catch>

<c:if test="${ex != null}">
    Exception : ${ex}
    <br />
    Message: ${ex.message}
</c:if>
```

JSTL –i18n → préfixe : fmt

- Définition de la “Locale”
- Formatage des messages
- Formatage des dates et des nombres

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
```

JSTL : Formattage et Localisation

Tag	Description
<fmt:formatNumber>	To render numerical value with specific precision or format.
<fmt:parseNumber>	Parses the string representation of a number, currency, or percentage.
<fmt:formatDate>	Formats a date and/or time using the supplied styles and pattern
<fmt:parseDate>	Parses the string representation of a date and/or time
<fmt:bundle>	Loads a resource bundle to be used by its tag body.
<fmt:setLocale>	Stores the given locale in the locale configuration variable.
<fmt:setBundle>	Loads a resource bundle and stores it in the named scoped variable or the bundle configuration variable.
<fmt:timeZone>	Specifies the time zone for any time formatting or parsing actions nested in its body.
<fmt:setTimeZone>	Stores the given time zone in the time zone configuration variable
<fmt:message>	To display an internationalized message.
<fmt:requestEncoding>	Sets the request character encoding

Exemple : <fmt:formatDate>

```
<%
    request.setAttribute("now", new java.util.Date());
%>
<h1>JSTL - "fmt:formatDate" :</h1>
<h2>
    type="time" =
    <fmt:formatDate value="${now}" type="time" />
</h2>
<h2>
    type="date" =
    <fmt:formatDate value="${now}" type="date" />
</h2>
<h2>
    type="both" =
    <fmt:formatDate value="${now}" type="both" />
</h2>
<h2>
    pattern="dd/MM/yyyy" =
    <fmt:formatDate value="${now}" pattern="dd/MM/yyyy" />
</h2>
<h2>
    pattern="yyyy-MM-dd" =
    <fmt:formatDate value="${now}" pattern="yyyy-MM-dd" />
</h2>
```

JSTL - "fmt:formatDate" :

type="time" = 11:49:29

type="date" = 17 févr. 2019

type="both" = 17 févr. 2019 11:49:29

pattern="dd/MM/yyyy" = 17/02/2019

pattern="yyyy-MM-dd" = 2019-02-17

JSTL : les fonctions

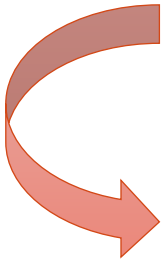
- Fichier « **fn.tld** », Préfix standard « **fn** »
- Il ne s'agit plus de tags, mais de « fonctions » utilisables dans les expressions EL

```
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions"%>
```

boolean	**contains**(java.lang.String, java.lang.String)
boolean	**containsIgnoreCase**(java.lang.String, java.lang.String)
boolean	**endsWith**(java.lang.String, java.lang.String)
java.lang.String	**escapeXml**(java.lang.String)
int	**indexOf**(java.lang.String, java.lang.String)
java.lang.String	**join**(java.lang.String[], java.lang.String)
int	**length**(java.lang.Object)
java.lang.String	**replace**(java.lang.String, java.lang.String, java.lang.String)
java.lang.String[]	**split**(java.lang.String, java.lang.String)
boolean	**startsWith**(java.lang.String, java.lang.String)
java.lang.String	**substring**(java.lang.String, int, int)
java.lang.String	**substringAfter**(java.lang.String, java.lang.String)
java.lang.String	**substringBefore**(java.lang.String, java.lang.String)
java.lang.String	**toLowerCase**(java.lang.String)
java.lang.String	**toUpperCase**(java.lang.String)
java.lang.String	**trim**(java.lang.String)

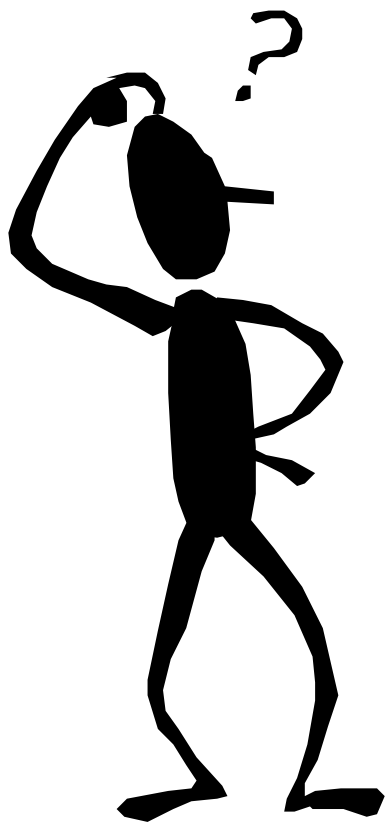
Exemple : Fonctions JSTL

```
<c:set var="tempStr" value="Java is the best language" />
length : ${fn:length(tempStr)}
<br /> Does it contain "test" ? ${fn:contains(tempStr,"test")}
<br /> Does it contain 'Java' ? ${fn:contains(tempStr,'Java')}
<br /> To upper case with fn:toUpperCase() : ${fn:toUpperCase(tempStr)}
<br /> Array length after fn:split() : ${fn:length(fn:split(tempStr," "))}
<br />
```



length : 25
Does it contain "test" ? false
Does it contain 'Java' ? true
To upper case with fn:toUpperCase() : JAVA IS THE BEST LANGUAGE
Array length after fn:split() : 5

Des Questions ??



Démo2

Prise en main de la technologie JSP