

Applications Web

Servlets, JSP

Cours IHM

Frédéric MOAL
Université d'Orléans

Master 1ère année

année 2011/2012

Plan



- Introduction et rappels
- Applications Web
 - Scripts cgi
 - Servlets
 - JSP
 - Tag lib

Ressources



Introduction

- **Documentation Apache** : <http://tomcat.apache.org/> dans la version des servlets/de tomcat utilisé
- **Le site officiel de Oracle/Sun** <http://java.sun.com>
- **Livres**
 - « Java - la synthèse », 4ème édition, Clavel...
 - « Penser en Java vX », <http://penserenjava.free.fr/>
- **« Cours » en ligne:**
 - Richard GRIN, Université de Nice
<http://deptinfo.unice.fr/~grin/messupports/index.html>
 - Jean Michel DOUDOUX - "Développons en Java" (1000 pages)
<http://www.jmdoudoux.fr/>
 - !! <http://www.developpez.com>, l'onglet java !!
- **Blogs :**
 - Xebia, SQLI, Le toulleur express, Java Posse, Bistro!, ...

3

Rappels



- Java (2, 5, 6 et 7)
- Html
- Eclipse
- Gestion de version (Subversion)
- ...

=> Cf ressources et web

4

Plan



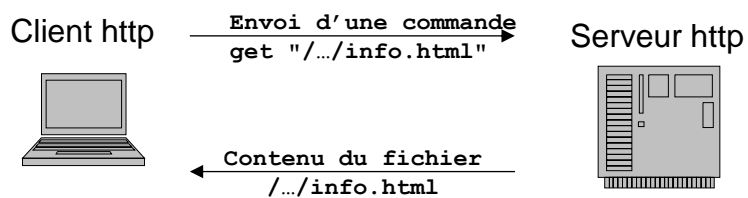
- Introduction
- Applications Web
 - Scripts cgi
 - Servlets
 - JSP
 - Tag lib

5

Applications Web



- Protocole http :
pages web statiques .html stockées sur le disque
du serveur



6

Pages dynamiques par script CGI



- Formulaire html : submit
- Les paramètres saisis dans le formulaire sont transmis au script
- Le script récupère les paramètres, effectue son traitement, et génère (imprime) la page html résultat

7

Pages dynamiques par script CGI

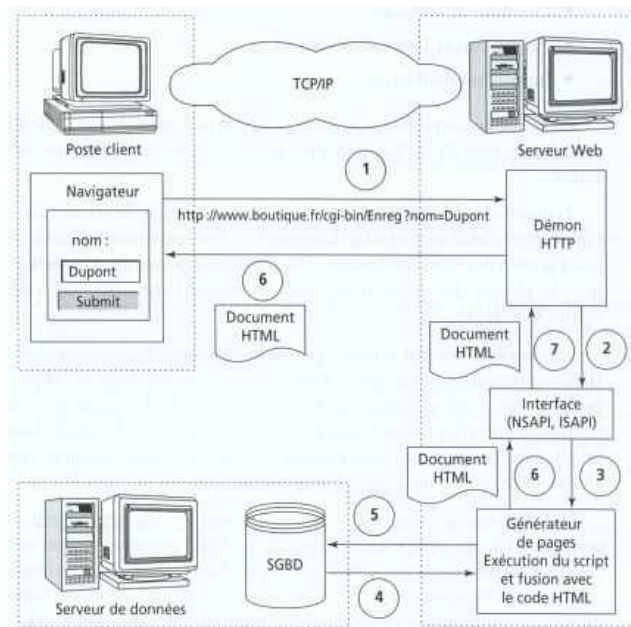


- Exemple simple avec un formulaire

```
<html>
<head>
  <title>Mon formulaire</title>
</head>
<body>
  <h1>Titre de page</h1>
  <form method="get" action="cgi-bin/Enreg">
    Nom : <input type="text" name="nom">
    <input type="submit" value="OK">
  </form>
</body>
</html>
```

8

Pages dynamiques par script CGI



9

Servlets



- alternative de la technologie Java à la programmation avec CGI
- Il s'agit de programmes exécutés sur un **serveur Web**
- servent de couche intermédiaire entre une requête provenant d'un navigateur Web et un autre service HTTP, comme des bases de données ou des applications du serveur HTTP

10



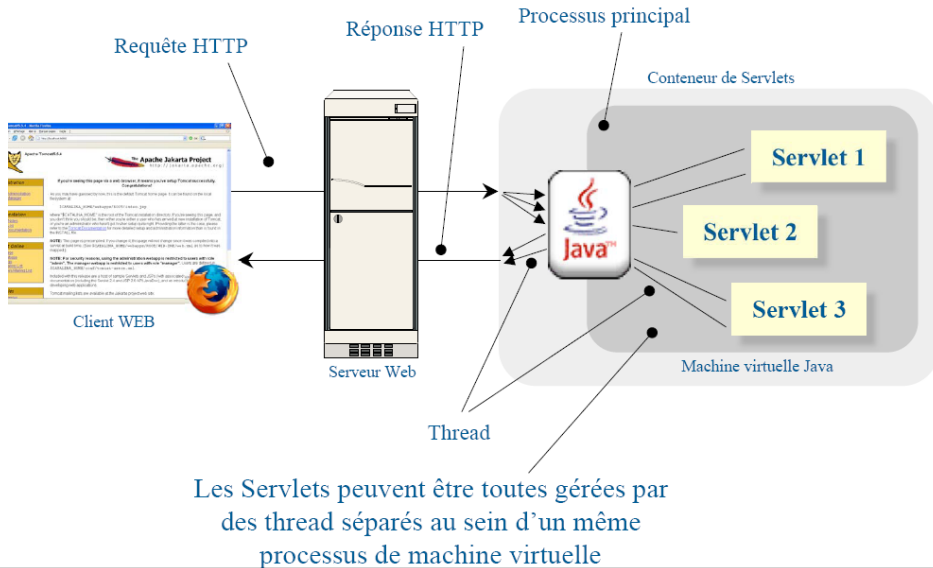
- Technologie Servlets
 - indépendant / OS et serveurs Web
 - peut produire du HTML coté serveur sur une base HTTP
 - mieux que CGI car prise en charge connexion des utilisateurs en multithread
 - peut dialoguer avec applets coté client via RMI



- Différences Servlets / scripts CGI
 - Performances
 - une seule machine virtuelle Java sur le serveur
 - servlet reste placée en mémoire une fois appelée
 - servlet modifiée peut être réactivée sans redémarrage serveur ou application
 - Modularité
 - possibilité d'avoir plusieurs servlets, chacune pouvant accomplir une tâche spécifique

Servlets

■ Architecture :



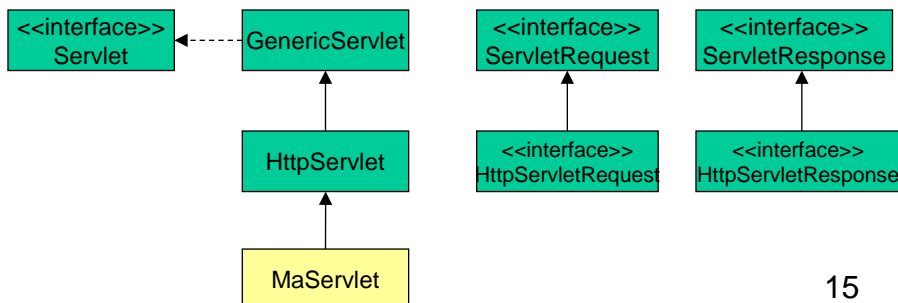
Servlets

■ Tâches de la servlet :

1. Lire toutes les données envoyées par l'utilisateur (d'un formulaire, d'une applet Java ou d'un prog. client HTTP)
2. Chercher d'autres informations sur la requête, à l'intérieur de la requête HTTP (cookies, nom de l'hôte, ..., etc.)
3. Générer des résultats (calculs, communication avec la base de données ...)
4. Formater le résultat dans un document (incorporation des informations dans une page HTML)
5. Définir les paramètres de la réponse HTTP appropriés (type de document, définir les cookies, mémoriser les paramètres...)
6. Renvoyer le document au client (au format texte (HTML), au format binaire (images GIF), ou dans un format compressé comme gzip)



- L'API pour les servlets constituée de deux packages :
 - `javax.servlet`
 - `javax.servlet.http`
 - JSDK (Java Servlet Development Kit) téléchargeable à <http://java.sun.com/products/servlet>



15

- Plusieurs méthodes sont fournies pour traiter les différents types de requêtes (GET, POST, ...).
- Elles sont appelées **méthodes de traitement de requêtes**
- Elles ont un en-tête identique `doXXX(...)` où XXX correspond au type de requête
- `doPost(...)` est la méthode pour traiter les requêtes de type POST
- `doGet(...)` est la méthode pour traiter les requêtes de type GET
- `doHead(...)`, `doTrace(...)`, ...
- Selon le type de requête (GET ou POST) le concepteur redéfinit la méthode concernée

16

Exemple de Servlet

Ne pas oublier d'importer la bibliothèque Java des Servlets

HelloWorld est un objet de type HttpServlet

Redéfinition de la méthode doGet (traitement d'une requête GET)

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet {

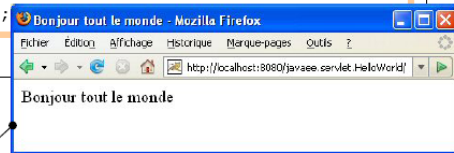
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();

        out.println("<HTML>");
        out.println("<HEAD><TITLE>Bonjour tout le monde</TITLE></HEAD>");
        out.println("<BODY>");
        out.println("<BIG>Bonjour tout le monde</BIG>");
        out.println("</BODY></HTML>");
    }
}
```

Réponse sous format HTML

HelloWorld.java du projet HelloWorldServlet

Le résultat sur le client



17

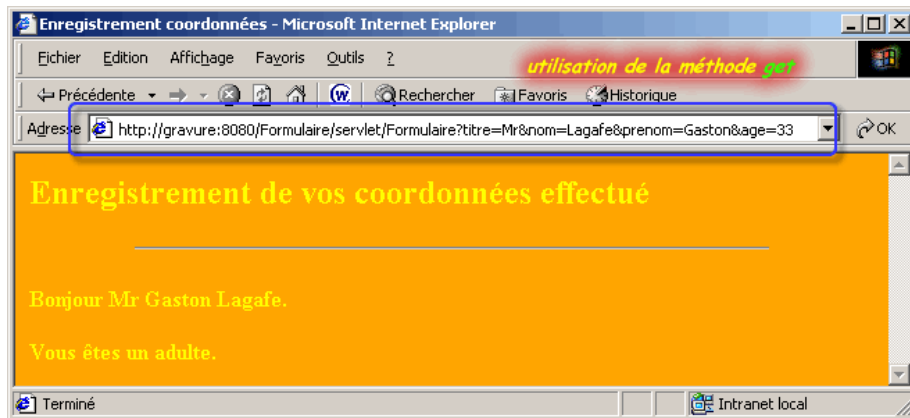
Servlets

■ Deuxième exemple :

18

Servlets

- Deuxième exemple :



19

Servlets

- Deuxième exemple :

```
<html>
<head><title>Formulaire</title></head>
<body bgcolor="orange" text="yellow">
<h2>Enregistrement de vos coordonnées</h2>
<hr>
<form method="get" action="servlet/Formulaire">
  <h3>M/Mme/Mlle :
  <select name="titre">
    <option>Mr</option>
    <option>Mme</option>
    <option>Mlle</option>
  </select></h3>
  <h3>Nom : <input type="text" name="nom" size="24"></h3>
  <h3>Prénom : <input type="text" name="prenom"></h3>
  <h3>Age : <input type="text" name="age" size="5"></h3>
  <p><input type="submit"><input type="reset">
</form>
</body>
</html>
```



```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;

public class Formulaire extends HttpServlet {
    private static final String CONTENT_TYPE = "text/html";
    /**Traiter la requête HTTP Get*/
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType(CONTENT_TYPE);
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head><title>Enregistrement coordonnées</title></head>");
        out.println("<body bgcolor=orange text=yellow>");
        out.println("<h2>Enregistrement de vos coordonnées effectué</h2>");
        out.println("<br width=75%>");
        out.print("<p><b>Bonjour " + request.getParameter("titre") + " ";
        out.print(request.getParameter("prenom") + " ");
        out.println(request.getParameter("nom") + ".>");
        int âge = Integer.parseInt(request.getParameter("age"));
        String message = "Vous êtes un";
        if (âge>0 && âge<12) message += " enfant.";
        if (âge>=12 && âge<18) message += " adolescent.";
        if (âge>=18 && âge<60) message += " adulte.";
        if (âge>=60) message += "e personne du troisième âge.";
        out.println("<p>" + message + "</b></body></html>");
    }
}
```

21



- HttpServletRequest hérite de ServletRequest
- Cet objet encapsule la requête HTTP et fournit des méthodes pour accéder aux informations
 - du client
 - de l'environnement du serveur
- Exemples de méthodes
 - String getMethod() : retourne le type de requête
 - String getServerName() : retourne le nom du serveur
 - String getParameter(String name) : retourne la valeur d'un paramètre
 - String[] getParameterNames() : retourne le nom des les paramètres
 - String getRemoteHost() : retourne l'IP du client
 - String getServerPort() : retourne le port sur lequel le serveur écoute
 - String getQueryString() : retourne la chaîne d'interrogation
 - ... (voir l'API Servlets pour le reste)

22

- HttpServletResponse hérite de ServletResponse
- Cet objet est utilisé pour construire un message de réponse HTTP renvoyé au client, il contient
 - les méthodes nécessaires pour définir le type de contenu, en-tête et code de retour
 - un flot de sortie pour envoyer des données (par exemple HTML) au client
- Exemples de méthodes
 - void setStatus(int) : définit le code de retour de la réponse
 - void setContentType(String) : définit le type de contenu MIME
 - ServletOutputStream getOutputStream() : flot pour envoyer des données binaires au client
 - void sendRedirect(String) : redirige le navigateur vers l'URL

23

Exemple Servlet

```
public class DownloadFileServlet extends HttpServlet {
    protected void doGet(HttpServletRequest arg0, HttpServletResponse arg1)
        throws ServletException, IOException {

        try {
            InputStream is = new FileInputStream("c:/dd.txt");
            OutputStream os = arg1.getOutputStream();

            arg1.setContentType("text/plain");
            arg1.setHeader("Content-Disposition", "attachment; filename=toto.txt");

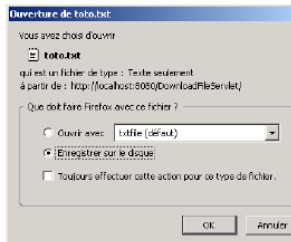
            int count;
            byte buf[] = new byte[4096];
            while ((count = is.read(buf)) > -1)
                os.write(buf, 0, count);
            is.close();
            os.close();
        } catch (Exception e) {
            // Y a un problème.
        }
    }
}
```

DownloadFileServlet.java du projet DownloadFileServlet

Le fichier à télécharger se trouve sur le serveur

Flux de sortie = client

En-tête de la réponse adaptée pour retourner un fichier

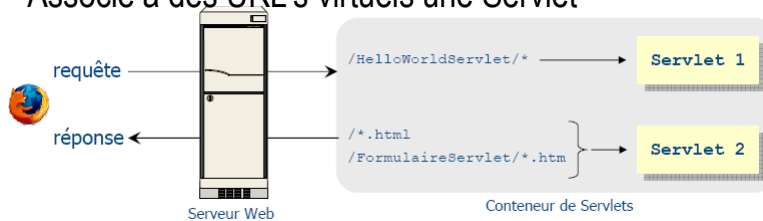


24

Conteneur de Servlets



- Une application WEB peut contenir plusieurs Servlets
- Pour tester et déployer une Servlet, il faut un système d'exécution appelé **conteneur de Servlets**
- Le conteneur réalise le lien entre la Servlet et le serveur WEB
 - Transforme code Java (bytecode) en HTML
 - Associe à des URL's virtuels une Servlet

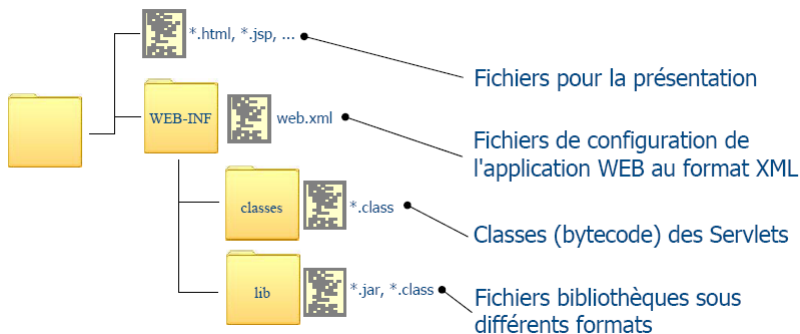


25

Conteneur de Servlets



- Une application WEB est contenue dans un répertoire physique sur le serveur, ou dans un WAR
- Une application WEB possède une hiérarchie de répertoires et de fichiers



26

Fichier de déploiement



- Le fichier web.xml (WebContent/WEB-INF) contient la description de toutes les servlets et JSP à déployer
- Il définit les URLs d'accès à ces éléments (mapping)
- Généré automatiquement (Eclipse...) ou défini « à la main »

27

Web.xml : exemple



En-tête du fichier **web.xml**

Balise principale

Balise de description de l'application WEB

Balise de description d'une Servlet

Nom de la Servlet "Identification"

Classe de la Servlet

Définition d'un chemin virtuel

Nom de la Servlet considéré "Identification"

Définition du chemin virtuel

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
  version="2.4">
  <display-name>Application WEB affichant HelloWorld</display-name>

  <servlet>
    <servlet-name>HelloWorldServlet</servlet-name>
    <servlet-class>HelloWorld</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>HelloWorldServlet</servlet-name>
    <url-pattern>/msg.hello</url-pattern>
  </servlet-mapping>
</web-app>
```

28

Web.xml : exemple



web.xml du projet **HelloWorld**

L'application WEB est ici composée de deux Servlets

Deux façon différentes d'appeler la Servlet *HelloWorldServlet*

Une seule façon d'appeler la Servlet *HelloWorldPrintWriter*

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app ...>

  <display-name>Servlets affichant différemment le message
    HelloWorld
  </display-name>

  <servlet>
    <servlet-name>HelloWorldServlet</servlet-name>
    <servlet-class>HelloWorld</servlet-class>
  </servlet>
  <servlet>
    <servlet-name>HelloWorldPrintWriter</servlet-name>
    <servlet-class>HelloWorldPrintWriter</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>HelloWorldServlet</servlet-name>
    <url-pattern>*.toutpourservlet</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>HelloWorldServlet</servlet-name>
    <url-pattern>/msg.hello</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>HelloWorldPrintWriter</servlet-name>
    <url-pattern>/printwriter.html</url-pattern>
  </servlet-mapping>
</web-app>
```

29

Chemins virtuels



web.xml du projet **HelloWorld**

Trois chemins virtuels ont été définis pour exécuter la Servlet *HelloWorldServlet*

Adresse du Serveur

Port

Contexte de l'application WEB

```
...
<servlet-mapping>
  <servlet-name>HelloWorldServlet</servlet-name>
  <url-pattern>/HelloWorldServlet/msg.hello</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>HelloWorldServlet</servlet-name>
  <url-pattern>*.toutpourservlet</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>HelloWorldServlet</servlet-name>
  <url-pattern>/index.html</url-pattern>
</servlet-mapping>
...
```

http://localhost:8080/HelloWorldServlet

http://localhost:8080/HelloWorldServlet/bonjour.toutpourservlet

http://localhost:8080/HelloWorldServlet/hello.toutpourservlet

http://localhost:8080/HelloWorldServlet/HelloWorldServlet/msg.hello

30

Servlet : cycle de vie



- Entre chaque requête du client les Servlets **persistent** sous forme d'**instances d'objet**
- Au moment où le code d'une Servlet est chargé, le serveur ne crée qu'**une seule instance** de classe
- L'instance (unique) traite chaque requête effectuée sur la Servlet
- Les avantages (rappels)
 - L'empreinte mémoire reste petite
 - Le surcoût en temps lié à la création d'un nouvel objet pour la Servlet est éliminé
 - La persistance est possible c'est-à-dire la possibilité de conserver l'état de l'objet à chaque requête (un exemple le compteur)

31

Servlet : cycle de vie



- A chaque rechargement d'une Servlet par le conteneur de Servlet, il y a **création d'une nouvelle instance** et donc destruction de l'ancienne
- Le rechargement d'une Servlet a lieu quand il y a :
 - Modification d'au moins une **classe** de l'application WEB
 - Demande explicite de l'administrateur du serveur WEB
 - Redémarrage du conteneur de Servlets
- CF problèmes en TP !!!

32

Servlet : cycle de vie



- Un constat : il n'y a **pas de constructeur** dans une Servlet
- L'initialisation des attributs se fait par l'intermédiaire de la méthode `init()`
 - Elle ne possède pas de paramètre
 - Définie et implémentée dans la classe abstraite `GenericServlet`
- La méthode `init()` peut être appelée à différents moments
 - Lorsque le conteneur de Servlets démarre
 - Lors de la première requête à la Servlet
 - Sur demande explicite de l'administrateur du serveur WEB

33

Servlet : cycle de vie



- Possibilité d'utiliser des paramètres d'initialisation exploités exclusivement par la méthode `init()`
- Les paramètres d'initialisation sont définis dans le fichier **web.xml** de l'application WEB

Balise qui détermine le nom du paramètre

Balise qui détermine la valeur du paramètre

Balise qui explique le rôle du paramètre (optionnelle)

```
...
<web-app ...>
  <display-name>Servlet simulant un compteur</display-name>

  <servlet>
    <servlet-name>InitConfigFileCounterServlet</servlet-name>
    <servlet-class>InitConfigFileCounterServlet</servlet-class>
    <init-param>
      • <param-name>initial_counter_value</param-name>
      • <param-value>50</param-value>
      • <description>Valeur initiale du compteur</description>
    </init-param>
  </servlet>
  ...
</web-app>
```

web.xml/du projet
Counter

Servlet : cycle de vie



- Une Servlet doit libérer toutes les ressources qu'elle a acquises et qui ne pourront être passées au ramasse-miettes
- Exemples de ressources
 - Connexion à une base de données
 - Ouverture d'un fichier sur le disque serveur
- La méthode `destroy()` donne une dernière chance d'écrire les informations qui ne sont pas encore sauvegardées
- La méthode `destroy()` est également utilisée pour écrire les informations persistantes qui seront lues lors du prochain appel à `init()`

35

Servlets



- Limitations importantes des servlets

MAINTENANCE DES PAGES !

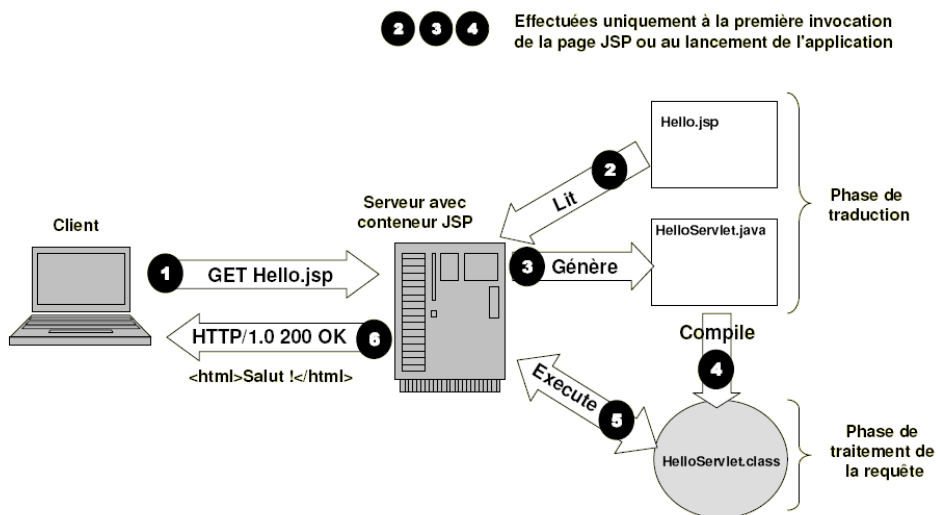
Par exemple pour un graphiste...

=> Introduction des JSP

36

- Séparation contenu statique/dynamique
- Manipulation des composants JavaBeans
- Extension standard aux Servlets
- Portabilité

Equivalent JAVA de
ASP (Application Server Pages de Microsoft)
et de PHP



Tags JSP



Trois types de tags

- `<%@ ... %>`
 - Tags de **directives**
 - contrôlent la structure de la servlet générée
- `<% ... %>`
 - Tags de **scripting**
 - insertion de code java dans la servlet
- `<jsp:... >`
 - Tags d'**actions**
 - facilitent l'utilisation de composants

39

Les directives



- Syntaxe :
`<%@directive attribut="valeur" ...>`
- Permettent de spécifier des informations globales sur la page
- 3 types de directives
 - **page** *options de configuration de la page*
 - **include** *inclusions de fichiers statiques*
 - **taglib** *pour définir des tags d'actions personnalisées*

40

La directive Page



Principaux attributs de la directive page

- `<%@page import="java.util.*;java.sql.Connection" %>`
- `<%@page contentType="text/html; charset=ISO-8859-1" %>`
- `<%@page session="true|false" %>`
*Indique si la page est incluse ou non dans une session. Par défaut **true**, ce qui permet d'utiliser un objet de type **HttpSession** pour gérer des données de session*
- `<%@page errorPage="relativeURL" %>`
Précise la JSP appelée au cas où une exception est levée, URL relative par rapport au répertoire qui contient la page JSP ou relative par rapport au contexte de l'application Web si elle débute par /
- `<%@page isErrorPage="true|false" %>`
Précise si la page JSP est une page de gestion d'erreur (dans ce cas l'objet exception peut être utilisée dans la page), false par défaut.
- `<%@page isThreadSafe="true|false" %>`
Précise si la servlet générée est multithreadée ou non.
- ...

41

La directive include



- Syntaxe :
`<%@include file="chemin relatif du fichier" %>`
 - chemin relatif par rapport au répertoire qui contient la page JSP ou relatif par rapport au contexte de l'application Web si il débute par /
- Inclus le fichier dans le source JSP avant que celui-ci ne soit interprété (traduit en servlet) par le moteur JSP
- Le fichier peut être un fragment de code JSP, HTML ou Java
- Tag utile pour insérer un élément commun à plusieurs pages (en-tête, pied de page)

Insertion à la compilation et non pas à l'exécution
un changement du fichier inclus ne provoque pas une
régénération de la servlet



42

Tags de scripting



- Permettent d'insérer du code Java qui sera inclus dans la servlet générée
- Trois formes de tags
 - `<%! ...%>` tag de déclaration
Code inclus dans le corps de la servlet (déclaration de membres, variables ou méthodes)
 - `<%=expression%>` tag d'expression
*L'évaluation de l'expression est insérée dans le flot de sortie dans la méthode **service()** de la servlet `<==> out.println(expression)`*
 - `<%...%>` tag de scriptlet
*Le code Java est inclus dans la méthode **service()** de la servlet*

43

Variables implicites



Les spécifications des JSP définissent plusieurs objets implicite utilisables directement dans le code Java

- **out : javax.servlet.jsp.JspWriter**
Flux en sortie de la page HTML générée
- **request : javax.servlet.http.HttpServletRequest** Contient les informations de la requête
- **response : javax.servlet.http.HttpServletResponse** Contient les informations de la réponse
- **session : javax.servlet.http.HttpSession**
Gère la session
- **Exception : java.lang.Throwable**
L'objet exception pour une page d'erreur

44

Tags de scripting

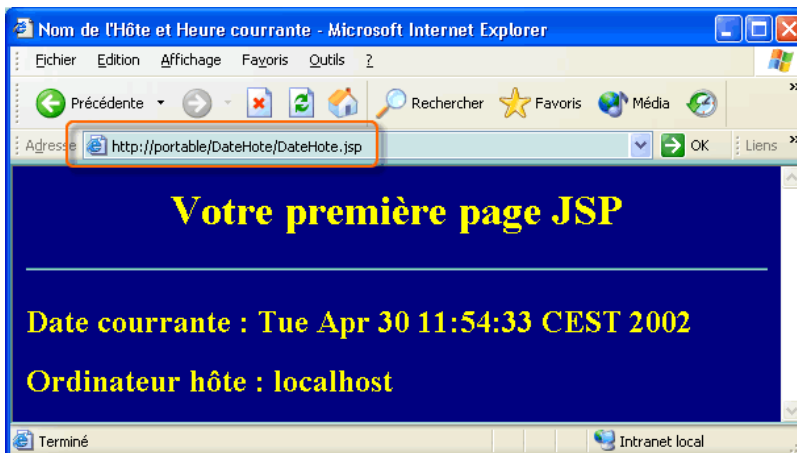
- Exemple de page JSP

```
<html>
<head>
<title>Nom de l'Hôte et Heure courante</title>
</head>
<body bgcolor="#000080" text="yellow">
<h1 align="center">Votre première page JSP</h1><hr>
<h2>Date courante : <%= new java.util.Date() %></h2>
<h2>Ordinateur hôte : <%= request.getRemoteHost() %></h2>
</body>
</html>
```

45

Tags de scripting

- Exemple de page JSP : résultat



46

Tags de scripting

- Exemple de page JSP avec formulaire

```
<html>
<head><title>Formulaire</title></head>
<body bgcolor="orange" text="yellow">
<h2>Enregistrement de vos coordonnées</h2>
<hr>
<form method="post" action="Formulaire.jsp">
  <h3>M/Mme/Mlle :
  <select name="titre">
    <option>Mr</option>
    <option>Mme</option>
    <option>Mlle</option>
  </select></h3>
  <h3>Nom : <input type="text" name="nom" size="24"></h3>
  <h3>Prénom : <input type="text" name="prenom"></h3>
  <h3>Âge : <input type="text" name="age" size="5"></h3>
  <p><input type="submit"><input type="reset">
</form>
</body>
</html>
```

47

Tags de scripting

- Exemple de page JSP avec formulaire

```
<html>
<head>
<title>Enregistrement des coordonnées</title>
</head>
<body bgcolor="orange" text="green">
<h2>Enregistrement des coordonnées effectué</h2>
<hr width="75%">
<p><b>Bonjour
  <%= request.getParameter("titre") %>
  <%= request.getParameter("prenom") %>
  <%= request.getParameter("nom") %>
  <%
    int âge = Integer.parseInt(request.getParameter("age"));
    String message = "Vous êtes un";
    if (âge>0 && âge<12) message += " enfant.";
    if (âge>=12 && âge<18) message += " adolescent.";
    if (âge>=18 && âge<60) message += " adulte.";
    if (âge>=60) message += "e personne du troisième âge.";
  <%
  <p><%= message %></p>
</body>
</html>
```

Expressions

Scriptlets

48

Commentaires



- `<!-- ... -->`
 - Commentaires HTML
 - Intégralement reconduits dans le fichier HTML généré
- `<%-- ... --%>`
 - Commentaires cachés
 - Contenu ignoré par le moteur JSP

49

Gestion des erreurs



- Si une exception est levée dans une page JSP et n'est pas capturée
 - Si pas de page d'erreur associée à la JSP : affichage de la pile d'exécution

```
Etat HTTP 500 -
JSP: Rapport d'exception
Message
Exception Le serveur a rencontré une erreur interne (il ne l'a empêché de satisfaire la requête).
Exception
java.lang.Exception: Mon Exception
    org.apache.jasper.runtime.PageContextImpl.doHandlePageException(PageContextImpl.java:825)
    org.apache.jasper.runtime.PageContextImpl.handlePageException(PageContextImpl.java:750)
    org.apache.jsp.index_jsp._jspService(index_jsp.java:64)
    org.apache.jasper.runtime.HttpServletBase.service(HttpServletBase.java:94)
    javax.servlet.http.HttpServlet.service(HttpServlet.java:602)
    org.apache.jasper.runtime.JspServletWrapper.service(JspServletWrapper.java:204)
    org.apache.jasper.runtime.JspServlet.serviceJspFile(JspServlet.java:292)
    org.apache.jasper.runtime.JspServlet.service(JspServlet.java:236)
    javax.servlet.http.HttpServlet.service(HttpServlet.java:602)
    org.netbeans.modules.web.monitor.server.MonitorFilter.doFilter(MonitorFilter.java:362)
Stack trace
java.lang.Exception: Mon Exception
    org.apache.jsp.index_jsp._jspService(index_jsp.java:55)
    org.apache.jasper.runtime.HttpServletBase.service(HttpServletBase.java:94)
    javax.servlet.http.HttpServlet.service(HttpServlet.java:602)
    org.apache.jasper.runtime.JspServletWrapper.service(JspServletWrapper.java:204)
    org.apache.jasper.runtime.JspServlet.serviceJspFile(JspServlet.java:292)
    org.apache.jasper.runtime.JspServlet.service(JspServlet.java:236)
    javax.servlet.http.HttpServlet.service(HttpServlet.java:602)
    org.netbeans.modules.web.monitor.server.MonitorFilter.doFilter(MonitorFilter.java:362)
```

50

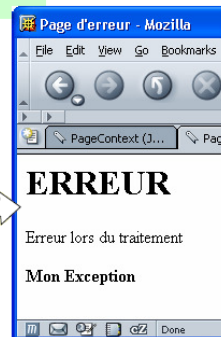
Gestion des erreurs

- Si une exception est levée dans une page JSP et n'est pas capturée
 - Si une page d'erreur est associée : redirection vers cette page

```
<%@page contentType="text/html"%>
<%@page import="java.util.Date"%>
<%@page errorPage="/MaPageErreur.jsp" %>
<html>
  <body>
    <% if (true) throw new Exception("Mon Exception"); %>
  </body>
</html>
```

MaPageErreur.jsp

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<%@page isErrorPage="true"%>
<html>
  <head><title>Page d'erreur</title></head>
  <body>
    <H1>ERREUR</H1>
    <p>Erreur lors du traitement</p>
    <p><b><%=exception.getMessage() %></b></p>
  </body>
</html>
```



JSP et Java Beans

- Un des objectifs des JSP / Servlets : *ne pas mélanger du code HTML au code Java des Servlets*
- D'un autre côté si c'est pour mettre du code Java dans le code JSP qu'est-ce qu'on y gagne ?
- Un point important dans la conception de pages JSP est de minimiser le code Java embarqué dans les pages

JSP et Java Beans



- Idée : déporter la logique métier dans des composants objets qui seront accédés depuis les pages JSP
 - *Simplification des traitements inclus dans la JSP*
 - *Possibilité de réutilisation des composants depuis d'autres JSP ou d'autres composants*
- Les spécifications JSP définissent une manière standard d'interagir avec des composants Java Beans

53

Java Bean



Un composant Java Bean c'est AU MOINS :

- *Classe publique*
- *Possède un constructeur public sans arguments*
- *Regroupe un ensemble de propriétés*
 - *accessibles par des méthode de la forme **getXXX()** où **XXX** est le nom de la propriété*
 - *éventuellement modifiables par une méthode **setXXX()** où **XXX** est le nom de la propriété*
- *Implémente (en option) l'interface **java.io.Serializable***

54

Java Bean : exemple



```
public class Personne {  
    private String nom;  
    private String prenom;  
    private int age;  
    public Personne() {          // instantiation par défaut  
        this.nom = "X";  
        this.prenom = "x";  
        this.age = 0;  
    }  
    public String getNom() {  
        return (this.nom);  
    }  
    public void setNom(String nom) {  
        this.nom = nom;  
    }  
    ...  
    public int getAge () {  
        return (this.ge);  
    }  
    public void setAge(int age) {  
        this.age = age;  
    }  
}
```

55

Utiliser un Bean dans les JSP



- Le tag `<jsp:useBean>` permet de localiser une instance ou bien d'instancier un bean pour l'utiliser dans la JSP

- Syntaxe

```
<jsp:useBean  
    id="beanInstanceName"  
    class="package.class"  
    type="package.class"  
    scope="page|request|session|application"  
>
```

56

Utiliser un Bean dans les JSP



L'attribut scope :

- **page [valeur par défaut]** : bean utilisable dans toute la page JSP ainsi que dans les fichiers statiques inclus.
- **request** : bean accessible durant la durée de vie de la requête. La méthode **getAttribute()** de l'objet **request** permet d'obtenir une référence sur le bean.
- **Session** : bean utilisable par toutes les JSP de la même session que la JSP qui a instanciée le bean. Le bean est utilisable tout au long de la session par toutes les pages qui y participent. La JSP qui crée le bean doit avoir l'attribut session = "true" dans sa directive page
- **Application** : bean utilisable par toutes les JSP qui appartiennent à la même application que la JSP qui a instanciée le bean. Le bean n'est instancié que lors du rechargement de l'application

57

Utiliser un Bean dans les JSP



Exemple :

```
<jsp:useBean
    id="utilisateur"
    class="test.Personne"
    scope="session"
/>
```

Avec TOMCAT, les beans doivent être nécessairement définis dans des packages



```
<HTML>
  <BODY>
    <UL>
      <LI>NOM : <%=utilisateur.getNom() %></LI>
      <LI>PRENOM : <%=utilisateur.getPrenom() %></LI>
      <LI>AGE : <%=utilisateur.getAge() %></LI>
    </UL>
  </BODY>
</HTML>
```

58

Fixer des propriétés



- Le tag `<jsp:setProperty>` permet de mettre à jour la valeur de un ou plusieurs attributs d'un bean
- Syntaxe

```
<jsp:setProperty
  name="beanInstanceName"
  property="propertyName"
  value="string|<%=expression%>"
/>
```

59

Fixer des propriétés



- Exemple :

```
<jsp:useBean id="utilisateur"
  class="test.Personne" scope="session"/>
...
<jsp:setProperty name="utilisateur"
  property="nom" value="Toto"/>
<jsp:setProperty name="utilisateur"
  property="age" value="34"/>
```

Quand le type de la propriété du Bean n'est pas String une conversion automatique est effectuée en utilisant la méthode `valueOf` de la classe enveloppe :

```
<%utilisateur.setAge(Integer.valueOf("34")); %>
```

60

Accès aux propriétés



- Le tag `<jsp:getProperty>` permet d'obtenir la valeur d'un attribut d'un bean

- Syntaxe

```
<jsp:getProperty name="beanInstanceName"
  property="propertyName" />
```

```
<jsp:useBean id="utilisateur" class="test.Personne" scope="session"/>
...
<body>
<UL>
  <LI>Nom : <jsp:getProperty name="utilisateur" property="nom" /></LI>
  <LI>Age : <jsp:setProperty name="utilisateur" property="age" /></LI>
</UL>
```

61

Tag de redirection



- Le tag `<jsp:forward>` permet de rediriger la requête vers un fichier HTML, une autre page JSP ou une Servlet

- Syntaxe

```
<jsp:forward page="relativeURL|<%=expression%"/>
```

```
<jsp:forward page="uneAutrePage.jsp" />
```

- Si URL commence par un / elle est absolue (contexte de l'application) sinon elle est relative à la JSP
- Ce qui suit l'action forward est ignoré, et tout ce qui a été généré dans cette page JSP est perdu

62

Tag de redirection



- Possibilité de passer un ou plusieurs paramètres vers la ressource appelée

```
<jsp:forward  
  page="relativeURL|<%=expression%>">  
<jsp:param name="parametre"  
  value="string|<%=expression%>">  
...  
</jsp:forward>
```

63

Tag d'inclusion



- Le tag `<jsp:include>` permet d'intégrer **dynamiquement** un contenu généré par une autre page JSP ou une autre servlet.
- Comme pour `<jsp:forward>` possibilité de passer un ou plusieurs paramètres vers la ressource incluse en utilisant le tag `<jsp:param>`
- Syntaxe

```
<jsp:include page="relativeURL"  
  flush="true|false"/>
```

(flush spécifie si le tampon doit être envoyé au client et vidé)

64

Tags personnalisés



- Possibilité de définir ses propres tags basés sur XML :
 - *tags personnalisés (custom tags)*
 - *regroupés dans des bibliothèques de tags (taglibs)*
- Objectifs
 - *Déporter dans des classes dédiées le code java contenu dans les scriptlets de la JSP et appeler ce code en utilisant des tags particuliers*
 - *Améliorer la séparation des rôles :*
 - *page JSP : présentation – concepteur de pages Web*
 - *scriptlets / code Java – développeur Java*
- Tags personnalisés / Java Beans
 - *"philosophie" similaire*
 - *Java Beans : objets métier pour stocker et échanger des données*
 - *Tag personnalisé : interagit directement avec environnement JSP dans lequel il s'exécute*

65

Tags personnalisés



Les Tags personnalisés sont regroupés en bibliothèques de Tag (Tag Lib). Défini par :

- Une classe Java (Gestionnaire de balise : Tag Handler)
 - *code exécuté par le conteneur de JSP lorsque ce Tag est invoqué dans une page JSP*
 - *implémente interface **javax.servlet.jsp.tagext.JSPTag***
 - *Accède à un objet **javax.servlet.jsp.JSPWriter** pour générer une réponse*
- Une entrée dans le fichier de description de la bibliothèque à laquelle il est associé (document XML TLD Tag Library Descriptor)
 - *la syntaxe des tags – Nom, attributs*
 - *La classe du Tag Handler associé*

66

Tags personnalisés



- Utilisation :

Pour pouvoir être utilisée dans une page JSP, une bibliothèque de Tags doit être déclarée avec la directive **<%@ taglib>**

```
<%@taglib uri="/WEB-INF/tlds/MaTagLib.tld" prefix="maTagLib" %>
```

- Exemple :

```
<%@taglib uri="/laTagLib" prefix="maTagLib" %>
```

...

```
<h1><maTagLib:HelloTag/></h1>
```

67

Tags personnalisés



Deux manières de déployer des bibliothèques de Tags :

- *Sans les packager*

- Le fichier *.tld* de description doit se trouver dans **/WEB-INF** où un de ses sous répertoire (**/WEB-INF/tlds**)
- Les classes (bytecode) des tag handlers doivent se trouver dans **/WEB-INF/classes**

- *En les "packageant" dans un fichier jar*

- Le fichier jar doit être placé dans **/WEB-INF/lib**

68

Tags personnalisés



- Nombreuses bibliothèques de tags existantes (*Libres, Commerciales*)
- JSTL : Java Standard Tag Library for JavaServer Pages
 - *Bibliothèque standard développée par JSR 052*
 - *Tags de structure (itération, conditions)*
 - *Internationalisation*
 - *Requêtes SQL*
 - ...
- *Nécessite conteneur Web implémentant au moins API 2.3 des servlets et l'API JSP 1.2*

69

JSTL



- Fonctionnalités de JSTL regroupées dans 5 bibliothèques de tags

Rôle	Tag Lib Descriptor	uri
– Fonctions de base	c.tld	http://java.sun.com/jstl/core
– Internationalisation	fmt.tld	http://java.sun.com/jstl/fmt
– Traitements SQL	sql.tld	http://java.sun.com/jstl/sql
– Traitements XML	x.tld	http://java.sun.com/jstl/xml
– Fonctions fn	fn.tld	http://java.sun.com/jsp/jstl/functions
- En plus JSTL propose un langage d'expression (EL) permettant de référencer facilement les objets java accessibles dans le contexte de la JSP

70

JSTL : Core



- Exemple : appel avec paramètres :
core.jsp?sayHello=true&name=Fred

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
...
<html>
<body>
<h1>
<c:if test="${param.sayHello}">
    <!-- Let's welcome the user ${param.name} -->
    Hello ${param.name}!
</c:if>
</h1>
</body>
</html>
```

<https://jstl.dev.java.net/>

71

Autre exemple : Struts Tag Lib



- Exemple : Les tags HTML Struts prennent en charge l'initialisation des éléments du formulaire

```
<%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
<html>
<head>
<title>Formulaire de saisie</title>
</head>
<body>
    <H1>Histogramme des notes</H1>
    <HR>
    <html:form action="notesAnneeMatiere.do">
        Année : <html:text property="annee" size="6"/>
    ...
```

<http://struts.apache.org>

72

EL (Expression Language)



- langage particulier constitué d'expressions qui permet en particulier d'utiliser et de faire référence à des objets java accessibles dans les différents contextes (page, requête, session ou application) d'une JSP
- Initialement introduit avec la JSTL
- Supporté de manière standard à partir de la version 1.4 J2EE (Servlets 2.4, JSP 2.0)
- une expression EL peut être utilisée directement dans n'importe quelle page JSP (à la place d'une expression `<%=expressionJava%>`)

73

EL (Expression Language)



- Exemple : accès à un attribut

```
...  
model.Utilisateur u = new model.Utilisateur();  
u.setNom("DUPONT");  
u.setPrenom("Jean");  
request.setAttribute("utilisateur", u);  
getServletContext().getRequestDispatcher("/testE  
L.jsp").forward(request, response);  
...
```

```
public class Utilisateur {  
    private String nom;  
    private String prenom;  
  
    public User() {  
    }  
  
    public String getNom() {  
        return nom;  
    }  
    public void setNom(String name) {  
        this.nom = name;  
    }  
    public String getPrenom() {  
        return prenom;  
    }  
}
```

```
...  
<jsp:useBean id="utilisateur" type="model.Utilisateur" scope="session"/>  
...  
<H1 align="center">Au revoir<BR>  
${user.prenom} ${user.nom} </H1>  
...
```

EL recherche l'attribut suivant l'ordre :

- portée de la page
- portée de la requête
- portée de la session
- portée de l'application

74

EL (Expression Language)



- Accéder à une propriété d'un JavaBean
`${user.prenom}`
- Accéder à une propriété imbriquée
`${user.adresse.ville}` `${user.adresse}`
- Accéder à une map
`${map.a}` `${map.u.prenom}`
- Accéder à un tableau ou une liste
`${array[3]}` `${liste[5]}`
- Les `[]` utilisables à la place du point
`${user["prenom"]}` `${map["a"]}`

75

EL (Expression Language)



▪ Variables implicites EL

pageContext	objet PageContext de la page JSP
pageScope	Map pour accéder aux attributs définis dans la portée de la page (PageContext)
requestScope	Map pour accéder aux attributs définis dans la portée de la requête (HttpServletRequest)
sessionScope	Map pour accéder aux attributs définis dans la portée de la session (HttpSession)
applicationScope	Map pour accéder aux attributs définis dans la portée de l'application (ServletContext)
param	Map pour accéder aux paramètres de la requête http sous forme de String
paramValues	Map pour accéder aux paramètres de la requête http sous la forme de tableau de String
header	Map pour accéder aux valeurs de l'en-tête de la requête
headerValues	Map pour accéder aux valeurs de l'en-tête de la requête sous la forme de tableau de String
initParam	Map pour accéder aux paramètres d'initialisation (init-params du web.xml)
cookie	Map pour accéder aux cookies

76



■ Opérateurs EL

Opérateur	Rôle	Exemple
. []	Obtenir une propriété d'un objet Obtenir une propriété par son nom ou son indice	<code>\${param.nom}</code> <code>\${param["nom"]}</code> <code>\${tab[0]}</code>
<code>== eq != ne</code> <code>< lt > gt</code> <code><= le >= ge</code>	Opérateurs relationnels	<code><c: if test="\${user.age ge 18}"></code>
<code>+ - * / div</code> <code>% mod</code>	Opérateurs arithmétiques	<code>\${article.prixHT * 0.055}</code>
<code>&& and </code> <code>or ! not</code>	Opérateurs logiques	<code>\${(user.age ge 7) && (user.age le 77)}</code>
<code>empty</code>	Teste si un objet est <code>null</code> ou vide si c'est une chaîne de caractère.	<code>\${empty param.nom}</code>