

Présentation et programme

Guillaume Defrain
2016/2017





Qui suis-je ?



Presentation

Guillaume Defrain – Chef de projet

guillaume.defrain@capgemini.com

Activité

- Gestion de projet
- Pilotage d'équipe
- Suivi des opérations
- Recrutement

Parcours universitaire

- Master MIAGE en alternance en 2009
- Licence MIAGE en 2007
- DUT informatique à Lille 1 en 2006

Formation

- 2014/2016 : module NTSI au M2 MIAGE alternance
- 2013/2014 : module COO à la FLST (catho)
- 2011/2013 : module IFI aux formations IAGL/eservice/TIIR à Lille 1

Expérience et Principales références



ERDF : Pilotage de projets
Projet Enedis-PRAC



Lyonnaises des Eaux : Responsables des développements (équipe de 18 personnes) du projet NSIC:
Progiciels Oracle avec surcouche JAVA

Auchan : Responsable des développements de la partie front du projet Eureka d'Auchan (réseau social d'entreprise)
J2EE : JSF, hibernate



Oxylane : Développement
développements d'applications métiers
Framework Décathlon Tetrix une surcouche de Struts

Projet mutualisés : Développements pour différents « petits » clients.
Technologies J2EE, Hibernate, JSF



Programme ?

Séance 1 : Servlet/JSP/JSTL

Guillaume Defrain
2016/2017





Séance 1 :

- Servlet et serveur d'application
- JSP
- JSTL



Présentation Générale

- But : étendre les fonctionnalités d'un serveur Web.
- Les servlets permettent la communication entre la logique applicative et les clients au sein des applications Web.
- Les servlets s'appuient sur un moteur de servlet (aussi appelé conteneur de servlet) . Ce moteur de servlet permet de masquer au développeur :
 - les aspects de connexion au réseau
 - la récupération des requêtes utilisateur
 - la formulation des requêtes



Présentation Générale

- Une servlet est un programme Java qui s'exécute sur un serveur Web compatible.
- Une fois chargée dans le moteur de servlet, une servlet continue de s'exécuter et attend d'autres requêtes provenants du client.
- Une servlet peut :
 - créer et retourner une page Web HTML en guise de réponse à la requête envoyée par le poste client (navigateur).
 - communiquer avec d'autres ressources du serveur (objets applicatifs, objets métiers, objets d'accès aux données)



Présentation Générale

- La servlet doit être déclarée dans un fichier de description pour le serveur qui contient le moteur de servlet (le fichier « web.xml »)
- Les classes et les interfaces liées aux servlets sont contenues dans deux packages Java :
 - javax.servlet : indépendant du protocole
 - javax.servlet.http : lié à http
- La servlet est une classe qui étend la classe HttpServlet. Elle peut redéfinir 2 méthodes :
 - doPost : méthode qui est appelé quand la servlet est appelé en méthode POST
 - doGet : méthode qui est appelé quand la servlet est appelé en méthode GET



Première Servlet

Voici un exemple simple de servlet dont le seul but est d'afficher du texte sur le navigateur du client :

```
package premiereServlet.servlet;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class PremiereServlet extends HttpServlet {

    public void init() {
    }
    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> Titre </TITLE></HEAD>");
        out.println("<BODY>");
        out.println("Ma première servlet");
        out.println("</BODY>");
        out.println("</HTML>");
        out.close();
    }
}
```



Première Servlet

Fichier web.xml

```
<?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">
  <servlet>
    <servlet-name>nomDeMaServlet</servlet-name>
    <servlet-class> premiereServlet.servlet.PremiereServlet
    </servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>nomDeMaServlet</servlet-name>
    <url-pattern>/maServlet</url-pattern>
  </servlet-mapping>
</web-app>
```



Première Servlet





Gestion des formulaires

```
<FORM Method="POST" Action="http://serveur/servlet/UserInfo">
Nom : <INPUT type=text size=20 name=Nom><BR>
Prénom : <INPUT type=text size=20 name=Prenom><BR>
Age : <INPUT type=text size=2 name=Age><BR>
<INPUT type=submit value=Envoyer>
</FORM>

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

public class UserInfo extends HttpServlet {
    public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<HTML>\n<BODY>\n" +
        "<H1>Recapitulatif des informations</H1>\n" +
        "<UL>\n" +
        "  <LI>Nom: «
        + request.getParameter("Nom") + "\n" +
        "  <LI>Prénom: «
        + request.getParameter("Prenom") + "\n" +
        "  <LI>Age: «
        + request.getParameter("Age") + "\n" +
        "</UL>\n" +
        "</BODY></HTML>");
    }
}
```



Cookies

```
//on crée un cookie
Cookie cookie = new Cookie("nom", "valeur");
cookie.setMaxAge(3600);
cookie.setValue(" une autre valeur");
response.addCookie(cookie);

//on recherche un cookie
Cookie[] cookies = request.getCookies();
for(i=0; i < cookies.length; i++) {
Cookie MonCookie = cookies[i];
if (MonCookie.getName().equals("LeCookieQueJeCherche")) {
String Valeur = cookies[i].getValue();
}
}

//on supprime un cookie
cookie.setMaxAge(0);
response.addCookie(cookie);
```

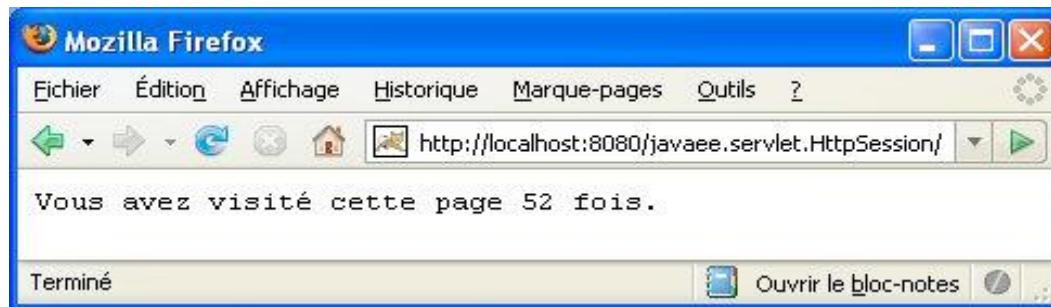


Sessions

- Méthodes de création liées à la requête (HttpServletRequest)
 - HttpSession getSession() : retourne la session associée à l'utilisateur
 - HttpSession getSession(boolean p) : création selon la valeur de p
- Gestion d'association (HttpSession)
 - Enumeration getAttributNames() : retourne les noms de tous les attributs
 - Object getAttribut(String name) : retourne l'objet associé au nom
 - setAttribut(String na, Object va) : modifie na par la valeur va
 - removeAttribut(String na) : supprime l'attribut associé à na
- Destruction (HttpSession)
 - invalidate() : expire la session
 - logout() : termine la session

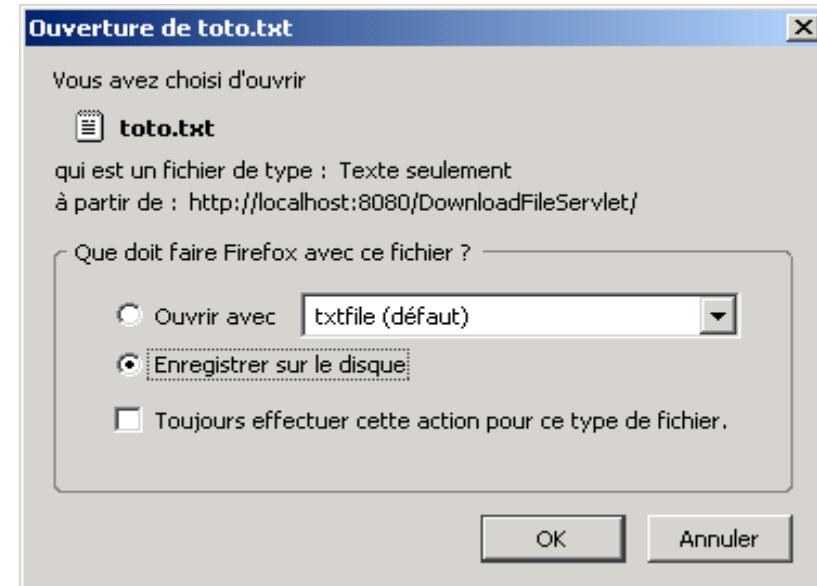
Sessions

```
public class HttpSessionServlet extends HttpServlet {  
    protected void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException,  
        IOException {  
        res.setContentType("text/plain");  
        PrintWriter out = res.getWriter();  
        HttpSession session = req.getSession();  
        Integer count = (Integer)session.getAttribute("count");  
        if (count == null)  
            count = new Integer(1);  
        else  
            count = new Integer(count.intValue() + 1);  
        session.setAttribute("count", count);  
        out.println("Vous avez visité cette page " + count + " fois.");  
    }  
}
```



Téléchargement de fichier

```
public class DownloadFileServlet extends HttpServlet {  
  
protected void doGet(HttpServletRequest pRequest, HttpServletResponse pResponse)  
    throws ServletException, IOException {  
try {  
    InputStream is = new FileInputStream("c:/dd.txt");  
    OutputStream os = pResponse.getOutputStream();  
    pResponse.setContentType("text/plain");  
    pResponse.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.  
}  
}  
}
```





Questions ?



Séance 1 :

- Servlet et serveur d'application
- JSP
- JSTL



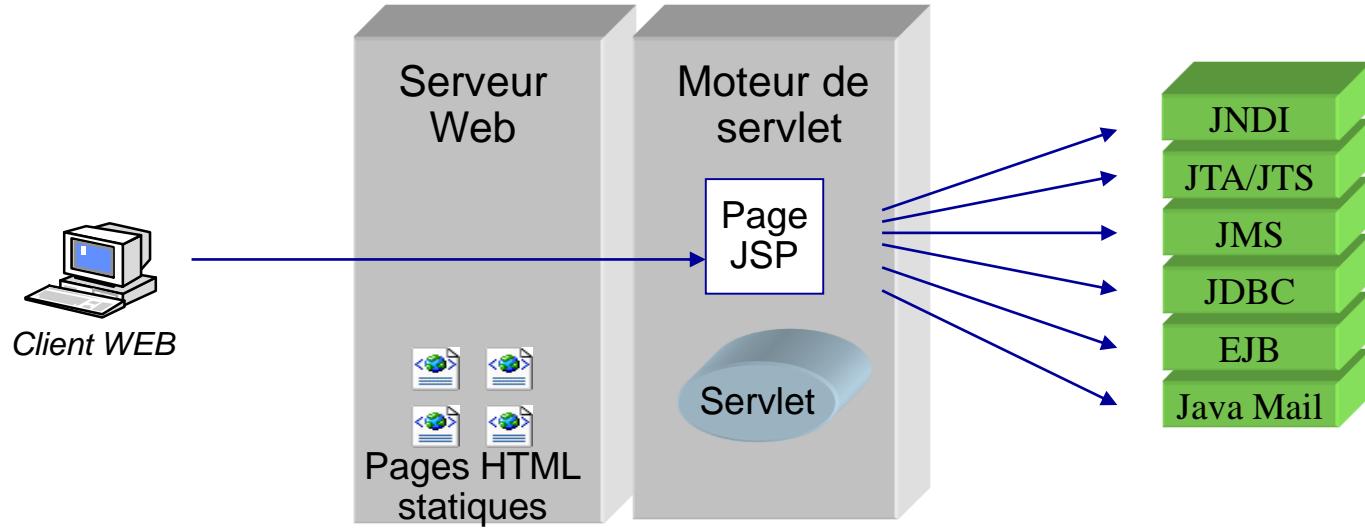
PRÉSENTATION GÉNÉRALE ET MÉCANISME DE COMPILATION

- Les JSP : Java Server Pages
 - objectif : générer des pages HTML dynamiques
 - permet de séparer la présentation des traitements
- JSP, les technologies concurrentes :
 - ASP
 - PHP
- Les fichiers JSP contiennent du code Java inséré dans des balises spécifiques, ainsi que du code HTML
- SUN fournit une spécification pour les JavaServer Pages

PRÉSENTATION GÉNÉRALE ET MÉCANISME DE COMPILEMENT

Mécanisme des JSP :

- On parle de « compilation à la volée » des JSP :
 - Le serveur Web reçoit une requête pour un fichier JSP
 - Ce serveur envoie cette requête au moteur de JSP
 - Ce moteur analyse le fichier JSP demandé
 - Un fichier source (.java) est généré (à partir du JSP) et est compilé (.class)



PRÉSENTATION GÉNÉRALE ET MÉCANISME DE COMPILEATION

- Le moteur de JSP est un moteur de servlet. Une page JSP compilée est une servlet.
- Une page JSP est compilée à la première demande uniquement (puis exécutée) ; aux demandes suivantes, et si le source de la JSP n'a pas changé, le fichier .class existant est exécuté.
- Le code source des fichiers JSP (format servlet) se trouve sous le répertoire du moteur de servlets et porte le nom de « jsp » ou « pageCompile » (utile pour déboguer).



LES TAGS JSP

- Les tags de directives <%@ ... %>
 - la directive page : permet de définir des options de configuration
 - import : permet d'importer des classes JAVA
 - content-type : définit le language de script
 - errorPage : indique la page à afficher si une exception est lancée
 - la directive include : permet d'inclure des fichiers dans la JSP (le code est inséré avant l'exécution).
- Syntaxe : <%@directive attribut="valeur" ... %>
- Exemple :
 - <%@page import="java.util.*" %>
 - <%@include file= "cheminRelatifDuFichier.jsp" %>
 - Remarque: Déclarer en début de page



LES TAGS JSP

- Les tags de scripting permettent d'intégrer du code Java dans les JSP
 - le tag de déclaration `<%! ... %>` : déclaration de variables d'instances, de classes, de méthodes.
 - le tag d'expression `<%= ... %>` : évalue le résultat et l'insère sous forme de String dans la page (équivalent à `out.print(...)`)
 - le tag de scriplet `<% ... %>` : contient du code Java qui sera dans la méthode service de la servlet correspondante
- Exemple :
 - `<% String name = request.getParameter("nom "); %>`
 - `<%= name %>`



LES TAGS JSP

- Les tags de commentaires
 - les commentaires HTML <!-- ... -->
 - les commentaires JSP<%-- ... --%>



LES VARIABLES IMPLICITES

- `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



EXEMPLE

```
<html><head><title>JSP complet</title></head>
<body>
<%! String[] langages = {"Java", "C++", "ADA", "Javascript"};
int random4() {
    return (int) (Math.random() * 4);
}
%>
<p>Parmi tous les langages au programme :</p>
<ul>
<%
for (int i=0; i < langages.length; i++) {
    out.println("<li>" + langages[i] + "</li>");
}
%>
</ul>
<p>celui de ce cours est <b><%= langages[random4()] %> </b></p>
</body>
</html>
```



EXEMPLE



Parmi tous les langages au programme :

- Java
- C++
- ADA
- Javascript

celui de ce cours est **Java**



LES TAGS JSP

■ Les tags d 'action

- le tag <jsp : useBean> : permet de localiser ou d 'instancier un bean
- le tag <jsp : setProperty> : permet de mettre à jour la valeur d'une propriété d'un bean
- le tag <jsp : getProperty> : permet d'accéder à la valeur d'une propriété d 'un bean
- le tag de redirection <jsp : forward> : permet de rediriger la requête vers une autre url (équivalent au sendRedirect de la servlet)
- le tag <jsp : include> : permet d'inclure le contenu d'un fichier dynamiquement



UTILISATION DES BEANS DANS UNE PAGE JSP

■ Présentation des JavaBeans

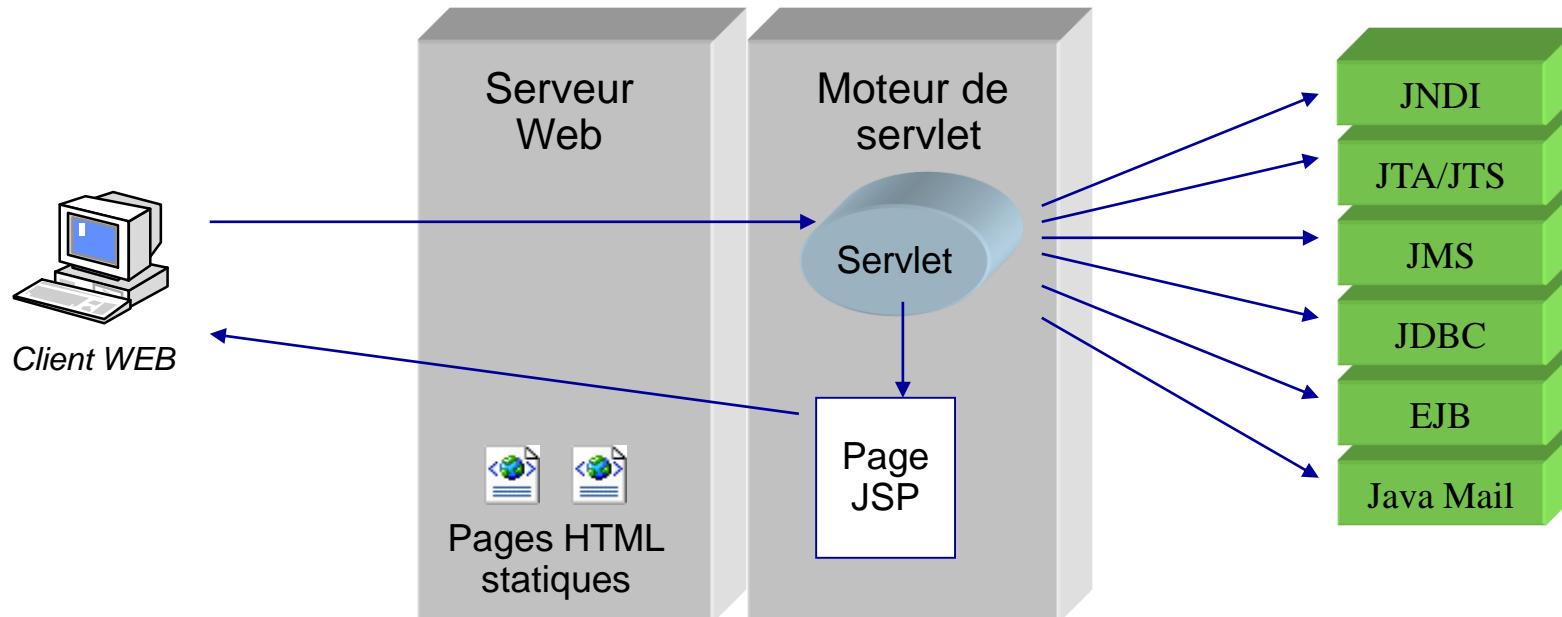
- composants Java pour encapsuler le code Java (allègement des pages JSP)
- Ces composants suivent des règles de développement (qui dépendent de l'utilisation du Bean : composant graphique ou non, ...).

■ Les méthodes

- Un Bean doit posséder un constructeur sans paramètre (qui doit initialiser les attributs du Bean)
- Un Bean peut définir des propriétés : Il s'agit de méthodes (des accesseurs) dont le nom et la signature sont normalisés.

COUPLAGE JSP / SERVLET

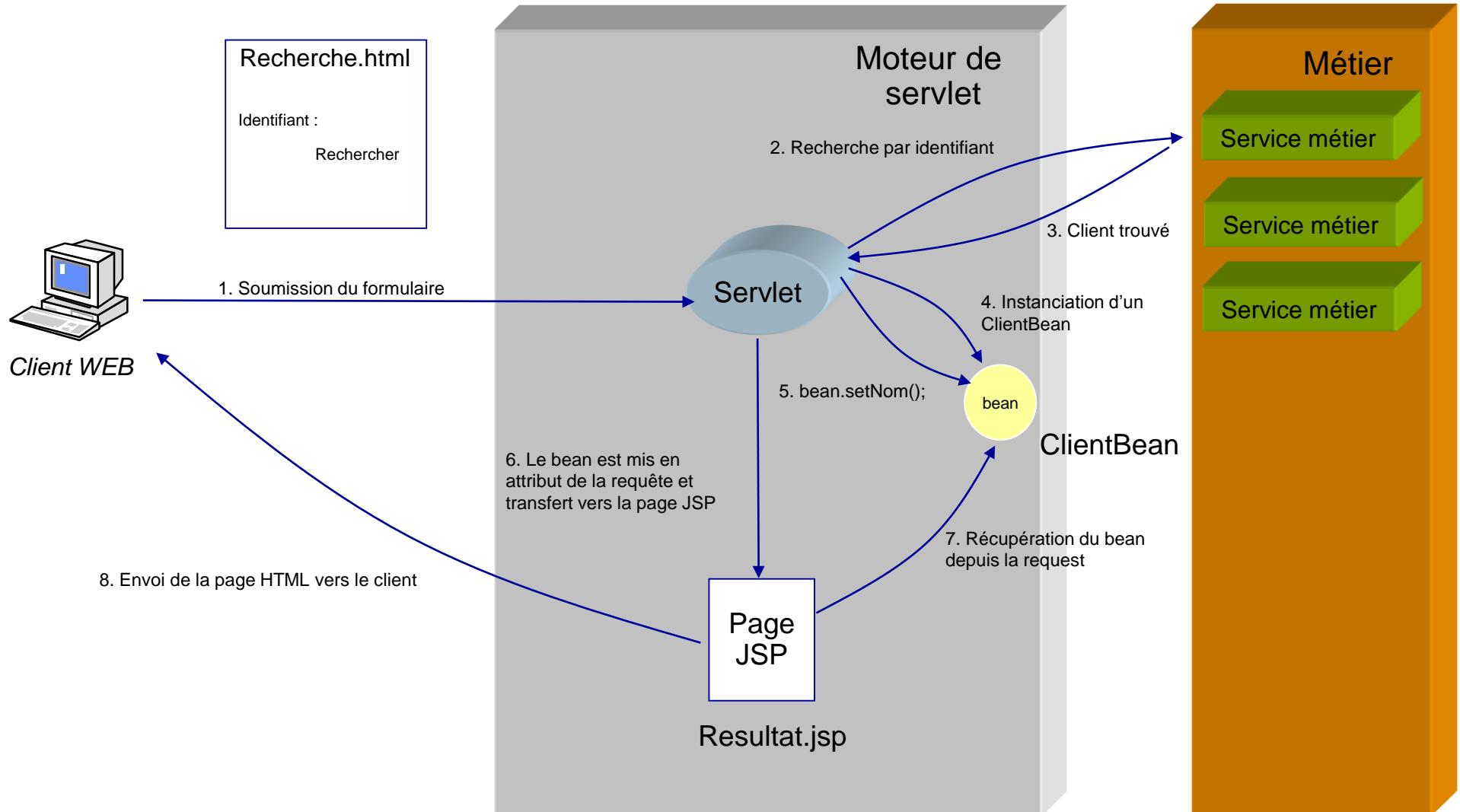
- Le couplage des servlets et des JSP permet de séparer clairement la partie « traitement » de la partie « affichage » du résultat.



- Ainsi, la page JSP va s'appuyer sur des JavaBean et Taglib pour présenter (sous forme de code HTML) les résultats de la requête au client.



COUPLAGE JSP / SERVLET





COUPLAGE JSP / SERVLET

- 1. Le client saisit l'identifiant dans le formulaire et lance la recherche : la Servlet reçoit la requête (HttpServletRequest)
- 2. La Servlet récupère l'identifiant passé en paramètre de la requête (req.getParameter(«identifiant»)) et appelle le métier pour recherche
- 3. Le métier renvoie une personne trouvée
- 4. La Servlet instancie un bean de type ClientBean
- 5. La Servlet positionne le nom du client renvoyé par le métier dans le bean : bean.setNom()
- 6a. La Servlet met le bean en attribut de l'objet HttpServletRequest
- 6b. La Servlet transfert la requête vers la JSP (RequestDispatcher)
- 7. La JSP récupère le bean depuis l'objet request
- 7b. La JSP affiche le nom du client depuis le bean
- 8. La page HTML est renvoyée vers le client



COUPLAGE JSP / SERVLET

■ Appel de la servlet :

- Une requête HTTP sera émise dès que l'utilisateur aura cliqué sur le bouton « Rechercher ». La servlet récupèrera l'identifiant saisi par l'utilisateur dans le formulaire HTML par la méthode « doPost() », effectuera le traitement pour obtenir l'objet métier Client voulu puis créera un objet JspBean (JSP JavaBean) qu'il initialisera par les « input » du formulaire HTML.

```
<html>
  <head><title>PageRecherche</title></head>
  <body>
    <form action="/servlet/RechercheServlet" method="POST">

      <BR><BR>Recherche d'un client :<BR>
      <BR><input type="text" name="identifiant">
      <BR><BR><input type="submit" name="Rechercher">

    </form>
  </body>
</html>
```



COUPLAGE JSP / SERVLET

Déclaration de la classe client.ClientBean

```
package client;  
class ClientBean {  
    private String nom;  
    private int numero;  
  
    public void setNom(String pNom) {  
        nom = pNom;  
    }  
  
    public String getNom() {  
        return nom;  
    }  
...}
```

La Servlet appelle le métier et crée le bean

```
String identifiant = request.getParameter("identifiant");  
Client client = serviceRecherche.getClient(identifiant); // Appel métier  
ClientBean clientBean = new ClientBean(); // Instanciation du bean  
clientBean.setNom(client.getNomClient()); // Remplissage du bean  
...
```



COUPLAGE JSP / SERVLET

- Transfert de la servlet vers la JSP :

- La Servlet positionne le bean en attribut de l'objet request

```
//objet request de type HttpServletRequest  
request.setAttribute("clientBean", clientBean);
```

- L'objet de type RequestDispatcher est créé par le moteur de servlet et permet d'appeler la page JSP.

```
// Code de la servlet pour obtenir l'objet RequestDispatcher  
RequestDispatcher dispatcher = request.getRequestDispatcher("/jsp/Resultat.jsp");
```

- La requête est transmise à la page JSP qui va se charger de retourner la réponse au poste client.

```
//Méthode « forward » pour appeler la page JSP  
dispatcher.forward (request, response);
```



Questions ?



Séance 1 :

- Servlet et serveur d'application
- JSP
- JSTL



PRÉSENTATION

- Un framework = une librairie de balises de présentation
- JSTL = une librairie standard pour la plupart des fonctionnalités de base d'une application J2EE.
- Sun spécifie les bases de la librairie et laisse l'implémentation libre (en TP on va utiliser l'implémentation de Jakarta de la JSTL 1.1)



LES LIBRAIRIES

Librairie	URI	Préfixe
Core	http://java.sun.com/jsp/jstl/core	c
Format	http://java.sun.com/jsp/jstl/fmt	fmt
XML	http://java.sun.com/jsp/jstl/xml	x
SQL	http://java.sun.com/jsp/jstl/sql	sql
Fonctions	http://java.sun.com/jsp/jstl/functions	fn



EL (EXPRESSIONS LANGUAGES)

- accès simple aux beans des différents scopes de l'application web (page, request, session et application)
- Utilisé conjointement avec des librairies de tags, elles permettent de se passer totalement des scriptlets.
- Une expression EL est de la forme suivante : \${ expression }
 \${ object.property }
 \${ object["index property"] }
 \${ object[index] }
 \${ object[0] }



EL – LES OBJETS IMPLICITES

- pageContext : Accès à l'objet PageContext de la page JSP.
- pageScope : Map permettant d'accéder aux différents attributs du scope 'page'.
- requestScope : Map permettant d'accéder aux différents attributs du scope 'request'.
- sessionScope : Map permettant d'accéder aux différents attributs du scope 'session'.
- applicationScope : Map permettant d'accéder aux différents attributs du scope 'application'.
- param : Map permettant d'accéder aux paramètres de la requête HTTP sous forme de String.
- paramValues : Map permettant d'accéder aux paramètres de la requête HTTP sous forme de tableau de String.
- header : Map permettant d'accéder aux valeurs du Header HTTP sous forme de String.
- headerValues : Map permettant d'accéder aux valeurs du Header HTTP sous forme de tableau de String.
- cookie : Map permettant d'accéder aux différents Cookies.
- initParam : Map permettant d'accéder aux init-params du web.xml.



EL – LES OBJETS IMPLICITES

- Exemple :
 \${sessionScope ["name"] } affiche l'attribut "name" de la session
 \${ header["user-agent"] } affiche l'header "user-agent" envoyé par le navigateur.
- Attributs des différents scope
- Lors de l'évaluation d'un terme, si celui ci n'est ni un type primaire, ni un objet implicite, le conteneur JSP recherchera alors un attribut du même nom dans les différents scopes de l'application.
- L'expression suivante : \${ name } entraîne la recherche de l'attribut "name" successivement dans les différents scopes, dans l'ordre page, request, session, application (pageScope["name"], requestScope["name"], sessionScope["name"], et applicationScope["name"])

EL – LES OPÉRATEURS

Opérateurs arithmétiques	Opérateurs relationnels
+ Addition	== ou eq Egalité
- Soustraction	!= ou ne Inégalité
* Multiplication	< ou lt Plus petit que
/ ou div Division	> ou gt Plus grand que
% ou mod Modulo	<= ou le Plus petit ou égal
	>= ou ge Plus grand ou égal

- Les opérateurs arithmétiques ne peuvent être utilisés que sur des données numériques.
- Les opérateurs relationnels d'égalité et d'inégalité utilisent la méthode equals() d'Object. Il est donc conseillé de la redéfinir si on se sert de ces opérateurs.
- Les opérateurs relationnels de comparaison ne s'appliquent que si l'interface Comparable est implémentée La méthode compareTo() est alors utilisée.

EL – LES OPÉRATEURS

Opérateurs logiques :		Autres :	
&& ou and	ET logique (true si les deux opérandes valent true , false sinon)	empty	true si l'opérande est null , une chaîne vide, un tableau vide, une Map vide ou une List vide. false sinon.
 ou or	OU logique (true si au moins une des deux opérandes vaut true , false sinon)	()	Modifie l'ordre des opérateurs.
! ou not	NON logique (true si l'opérande vaut false , et inversement)	?:	Opérateur conditionnel en ligne, format particulier : <i>condition ? valeur_si_true : valeur_si_false</i>

- Les opérateurs logiques ne s'appliquent que sur des booléens.
- Les opérateurs !, not et empty sont des opérateurs unaire



EL – ACCÈS AUX PROPRIÉTÉ DES OBJETS

- Propriété d'un bean standard (opérateur point)
 \${ bean.name}
 Récupère dans le scope l'attribut bean et appelle sur cette objet
 la méthode getName ()

Les appels à des propriétés peuvent être chainés

 \${person.adress.city}



EL – ACCÈS AUX PROPRIÉTÉS DES OBJETS

- Propriétés indexées

Permet d'accéder à un élément d'un tableau ou d'une List

`${list[0]}`

`${list["0"]}`

`${list[config.value]}`



EL – ACCÈS AUX PROPRIÉTÉS DES OBJETS

- Propriétés mappées
Permet d'accéder à un élément d'une Map

```
 ${ map["clef1"] }  
 ${ map['clef2'] }  
 ${ map[config.key] }
```

EL – ACCÈS AUX PROPRIÉTÉ DES OBJETS

- Gestion des exceptions

- NullPointerException (et les valeurs null) :

Les différentes propriétés d'un élément ne sont pas forcément renseignées et peuvent très bien être null.

Exemple, dans l'expression

```
 ${ sessionScope['data'].information.date },
```

si un élément est null l'expression prendra la valeur null mais aucune exception ne sera lancée.

De plus, lors de l'affichage dans la page JSP, toutes les valeurs null sont remplacées par des chaînes vides, afin de ne pas afficher le texte null à l'utilisateur...



EL – ACCÈS AUX PROPRIÉTÉ DES OBJETS

- Gestion des exceptions

- **ArrayOutOfBoundsException** :

De la même manière, l'accès à un index incorrect d'un élément d'un tableau ou d'une List ne provoquera pas d'exception mais renverra une valeur null.



CORE

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

- Gestion des variables de scope (out, set, remove, catch)

- Afficher une expression

```
<!-- Afficher l'user-agent du navigateur ou "Inconnu" si il est absent : -->  
<c:out value="${header['user-agent']}
```

- Définir une variable de scope ou une propriété

```
<!-- Mettre ${expression} dans l'attribut "varName" de la session : -->  
<c:set scope="session" var="varName" value="${expression}" />
```

```
<!-- Changer la propriété "name" de l'attribut "varName" de la session : -->  
<c:set target="${session['varName']}" property="name" value="new value"/>
```

- Supprimer une variable du scope

```
<!-- Supprime l'attribut "varName" de la session -->  
<c:remove var="varName" scope="session"/>
```

- Intercepter des exceptions

```
<!-- Stocker dans le scope page l'exception intercepté : -->  
<c:catch var="varName">  
    <c:set target="beans" property="prop" value="1"/>  
</c:catch>
```



CORE

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

- Actions conditionnels (if, choose, when, otherwise)
 - Test simple

```
<!-- Afficher un message si le paramètre "page" de la requête HTTP est absent-->
<c:if test="${empty param['page']}>
    Le paramètre page est absent !
</c:if>
```

- Test complexe

```
<!-- Afficher un message différent selon la valeur du bean 'value' : -->
<c:choose>
    <c:when test ="${value==1}"> value vaut 1 (Un) </c:when>
    <c:when test ="${value==2}"> value vaut 2 (Deux) </c:when>
    <c:when test ="${value==3}"> value vaut 3 (Trois) </c:when>
    <c:otherwise>
        value vaut ${value} (?)
    </c:otherwise>
</c:choose>
```



CORE

- Itérations (forEach, forTokens)

- Iteration sur une collection

```
<!-- Afficher tous les éléments d'une collection dans le request-->
<c:forEach var="entry" items="${requestScope['myCollection']}" >
    ${entry}<br/>
</c:forEach>
```

- Iteration sur une collection avec index

```
<!-- Afficher seulement les 10 premiers éléments -->
<c:forEach var="entry" items="${requestScope['myCollection']}" begin="0" end="9">
    ${entry}<br/>
</c:forEach>
```

- Boucle sur des id

```
<!-- Afficher les nombres de 1 à 10 -->
<c:forEach var="entry" begin="1" end="10">
    ${entry},
</c:forEach>
```

- Iteration sur une map

```
<!-- Afficher tous les paramètres de la requête HTTP (param est une Map)-->
<c:forEach var="entry" items="${param}" >
    Le paramètre "${entry.key}" vaut "${entry.value}".<br/>
</c:forEach>
```

- Iteration sur une chaîne de caractère

```
<c:forTokens var="p" items="mot1;mot2;mot3;mot4" delims=";" >
    ${p}<br/>
</c:forTokens>
```



CORE

- Les URLs (param, url, redirect, import)

```
<!-- Création d'un lien dont les paramètres viennent d'une MAP -->
<c:url value="/index.jsp" var="variableURL">
    <c:forEach items="${parameterMap}" var="entry">
        <c:param name="${entry.key}" value="${entry.value}" />
    </c:forEach>
</c:url>
<a href="#">Mon Lien
```

- Importer des ressources

```
<!-- Importer un fichier de l'application (similaire à <jsp:include/>) -->
<c:import url="/file.jsp">
    <c:param name="page" value="1"/>
</c:import>

<!-- Importer une ressource distante FTP dans une variable -->
<c:import url="ftp://server.com/path/file.ext" var="file" scope="page"/>
```



FORMAT

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

- Affichage de message en fonction de la locale

- Création d'un fichier properties nommé
 - Message_fr_FR.properties (langue française pays France)
 - Message_fr.properties (uniquement langue française)
 - Message.properties (par défaut)

- Ce fichier contient des couples clé/valeurs :

message.hello = Bienvenue

message.title = Titre de la page

- Utilisation de la balise message pour afficher le contenu

```
<fmt:message key="message.hello"/> → Bienvenu pour les utilisateurs français
```

- Possibilité de paramétriser des messages

```
<!-- Affichage d'un message avec deux paramètres -->
<fmt:message key="message.key">
    <fmt:param value="${mailbox.userName}" />
    <fmt:param value="${mailbox.messageCount}" />
</fmt:message>
```

Message.key = Bienvenue {0}, votre boîte de réception {1}, choice, 0#ne comporte aucun message |
message | 1<comporte {1} messages

1#comporte un

- Gestion de la locale

- Par défaut récupérée en fonction du header HTTP

- En forçant une locale en utilisant `<fmt:setLocale value="fr_FR" scope="session"/>`

FORMAT

- Formatage des données

- Des dates

```
<!-- Créer une date initialisé au premier janvier 2005 : -->
<fmt:parseDate value="01/01/2005" pattern="dd/MM/yyyy" var="date"/>

<!-- Afficher une date selon un format spécifique : -->
<fmt:formatDate value="${dateBeans}" pattern="dd/MM/yyyy"/>

<!-- Afficher une date selon un format standard : -->
<fmt:formatDate value="${dateBeans}" style="full"/>
```

- Des nombres

```
<c:set var="val" value="200.51" />
<fmt:setLocale value="en_US"/>
<fmt:formatNumber value="${val}" /> => affiche 200.51
<fmt:setLocale value="fr_FR"/>
<fmt:formatNumber value="${val}" /> => affiche 200,51

<!-- Afficher une somme en Euros -->
<fmt:formatNumber value="15" type="currency" currencySymbol="&euro; "/>

<!-- Afficher un pourcentage --> <
<fmt:formatNumber value="0.25" type="percent"/>
```



FONCTIONS

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

- \${fn:contains()}
- \${fn:containsIgnoreCase()}
- \${fn:endsWith()}
- \${fn:escapeXml()}
- \${fn:indexOf()}
- \${fn:join()}
- \${fn:length()}
- \${fn:replace()}
- \${fn:split()}
- \${fn:startsWith()}
- \${fn:substring()}
- \${fn:substringAfter()}
- \${fn:substringBefore()}
- \${fn:toLowerCase()}
- \${fn:toUpperCase()}
- \${fn:trim()}

Exemple :

```
 ${ fn:endsWith("Il était une fois", "fois") } → true
```

```
 ${ fn:indexOf("Il était une fois", "une") } → 10
```

Permet d'écrire des test plus complexe comme

```
<c:if test="${fn.length(requestScope["listeLivre"]) > 10}"> ....</c:if>
```

AUTRES LIBRAIRIES

- XML

Permet des manipulation de XML directement dans la JSP (utilisation de Xpath et XSLT notamment)

- SQL

Permet des actions SQL directement dans les JSP.



INTERDIT



SCRIPTLETS - EXEMPLE

```
<%@ page import="packageNames.Book" %>
<%@ page import="java.util.*" %>
<ul>
<% List bookList = (List) request.getAttribute("books-list");
if (bookList!=null) {
    Iterator iterator = bookList.iterator();
    int i = 0;
    while ( iterator.hasNext() ) {
        Book b = (Book) iterator.next();
        out.print ("<li class='<" + (i%2==0?"pair":"impair") + "'>");
        out.print (b.getName() + " (" + b.getPrice() + " &euro;)");
        if ( b.getPrice() < 30.0 )
            out.print (" <img src='hot.png' alt='Moins de 30 &euro;' />");
        out.println ("</li>");
        i++;
    }
}
%>
</ul>
```

- Jakarta Struts - par la pratique (37.05 €)
- Swing : la synthèse (37.91 €)
- Jakarta Struts - précis & concis (O'Reilly) (8.55 €)
- Java, conception et déploiement J2EE (23.75 €)
- Programmation Orienté Aspect pour Java / J2EE (42.75 €)
- Cahiers du Programmeur Java 1 (21.85 €)
- Java & XSLT (40.85 €)
- Java & XML 2nd Edition (47.5 €)
- Enterprise JavaBeans (45.6 €)
- Le Livre de Java premier langage (27.5 €)
- Java en action (55.1 €)
- Java in a Nutshell (51.3 €)
- Au coeur de Java 2 ; tome 1 (38.0 €)
- Total Java (10.0 €)



JSTL - EL - EXEMPLE

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<ul>
    <c:forEach items="${requestScope['books-list']}" var="book"
    varStatus="status">
        <li class="${status.index%2==0?'pair':'impair'}">${book.name}
        (${book.price} &euro;)
            <c:if test="${book.price < 30.0}">
                <img src='hot.png' alt='Moins de 30 &euro;' />
            </c:if>
        </li>
    </c:forEach>
</ul>
```

- Jakarta Struts - par la pratique (37.05 €)
- Swing : la synthèse (37.91 €)
- Jakarta Struts - précis & concis (O'Reilly) (8.55 €)
- Java, conception et déploiement J2EE (23.75 €)
- Programmation Orienté Aspect pour Java / J2EE (42.75 €)
- Cahiers du Programmeur Java 1 (21.85 €)
- Java & XSLT (40.85 €)
- Java & XML 2nd Edition (47.5 €)
- Enterprise JavaBeans (45.6 €)
- Le Livre de Java premier langage (27.5 €)
- Java en action (55.1 €)
- Java in a Nutshell (51.3 €)
- Au coeur de Java 2 ; tome 1 (38.0 €)
- Total Java (10.0 €)



People matter, results count.

www.capgemini.com