DÉVELOPPEMENT DES APPLICATIONS WEB EN JAVA

Programmation 3 GCR2 | I.AZAIEZ | 2024

1

Vue d'ensemble

2

1. Principe des applications Web
2. Principaux composants JEE pour le Web
3. Les servlets
4. Java Server Page (JSP)
5. JSTL
6. Persistance des données (JPA)
7. Hibernate

2

Chapitre 1

3

PRINCIPE DES APPLICATIONS WEB

3

Architecture des applications Web

4

- Une application Web est un ensemble de composants ou unité logique de traitement.
  - Objectif: modularité et réutilisation
  - Trois couches applicatives composent ces composants
    - Présentation : interaction avec l'utilisateur
    - Traitement : traitement métiers
    - Données : gestion des données ou des ressources.
- Ces 3 niveaux peuvent être imbriques ou repartis de différentes manières entre plusieurs machines physiques ou logiques
  - Suivant les contraintes d'utilisation ou contraintes techniques, on distingue: **Architecture 2 tiers, Architecture 3 tiers,** et **Architecture n tiers.**

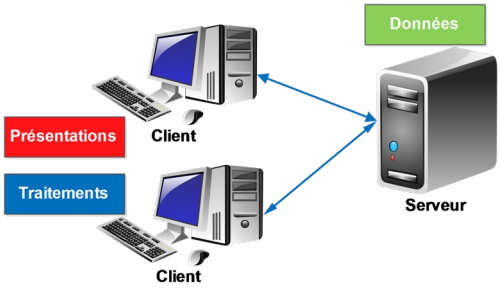
4

Architecture des applications Web

5

Architecture 2-tiers (client-serveur)

- La couche présentation et traitements sont sur le client
- Les données sont sur le serveur
  - Contextes multi-utilisateurs avec accès aux données centralisées.



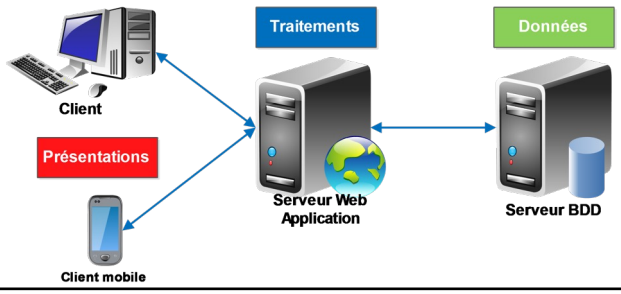
5

Architecture des applications Web

6

Architecture 3-tiers

- La couche présentation (IHM) est sur le client
- La couche métier ou les traitements sont pris par un serveur intermédiaire
- Les données (fichiers, fichiers XML, BDD,…) sont sur un serveur de données
  - Contexte multi-utilisateur: Internet



6

Architecture des applications Web

7

- Ce qu'il faut comprendre et retenir de tout ça :
  - les données sont échangées entre le client et le serveur via le protocole HTTP ;
  - le client ne comprend que les langages de présentation de l'information, en d'autres termes les technologies HTML, CSS et Javascript ;
  - les pages sont générées sur le serveur de manière dynamique, à partir du code source du site.

7

Architecture des applications Web

8

- Quelle est la différence entre un serveur web et un serveur d'application ?
  - Pour répondre à cette question je vous propose d'exposer les différentes couches à empiler pour arriver à monter un serveur d'application.
- Couche 1 : Serveur web http
  - Un serveur http, c'est un serveur qui gère exclusivement des requêtes HTTP. Il a pour rôle d'intercepter les requêtes Http, sur un port qui est par défaut 80, pour les traiter et générer ensuite des réponses Http. Tous les serveurs web embarquent un daemon Http (httpd) ou équivalent qui s'occupe de cette fonctionnalité. Exemple serveur Appache

8

Architecture des applications Web

9

- Couche 2 : Conteneur web
  - Maintenant, on va étendre notre serveur web pour devenir un conteneur Web. Cette extension va permettre d'avoir la possibilité d'exécuter des programmes écrits avec des langages de programmation (java, php, C# ou autres) dans le serveur web.
  - Par exemple : le serveur Tomcat n'est autre qu'un serveur Apache couplé avec un moteur web java et, les serveurs tel-que easyPhp, wamp ou Xamp ne sont que des serveurs Apaches couplés avec un moteur web php.

9

Architecture des applications Web

10

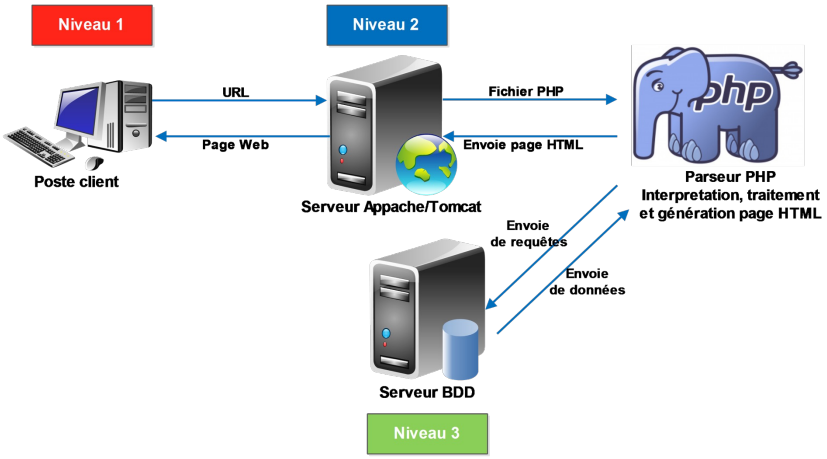
- Couche 3 : Serveur d'application
  - Donc, il faut étendre encore plus le serveur Tomcat pour devenir un vrai serveur d'application java jee. L'extension nécessaire est composée de deux parties essentielles :
    - Un conteneur EJB qui encapsule les traitements des Entreprise JavaBeans.
    - Un ensemble de services répartie en :
      - Des services d'infrastructures
      - Des services de communication.

10

Architecture des applications Web

11

Architecture 3-tiers: Exemple



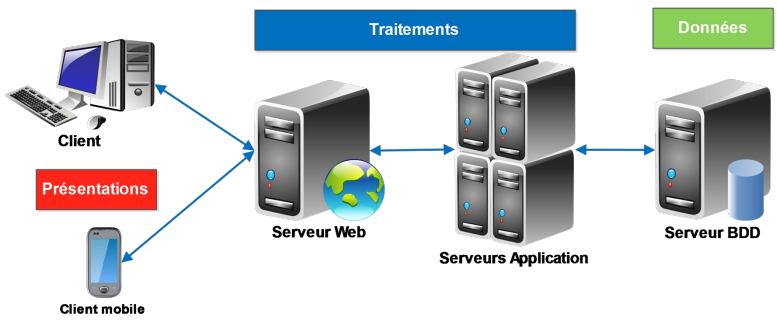
11

Architecture des applications Web

12

Architecture n-tiers

- C'est une architecture N-tiers dans laquelle le traitements des données contient lui-même plusieurs couches multipliant ainsi les tiers.

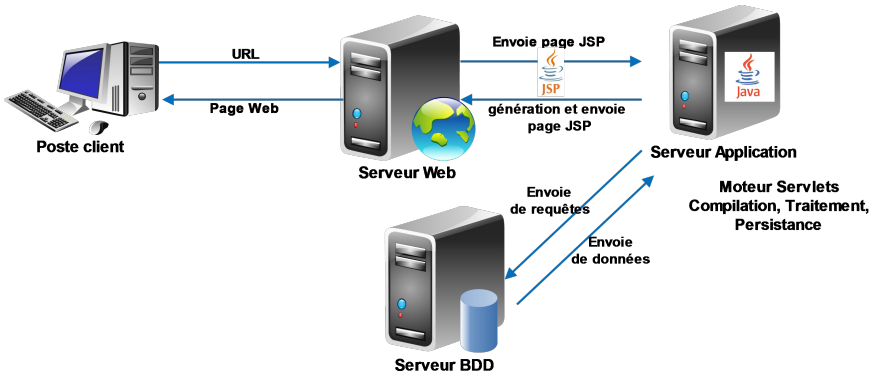


12

Architecture des applications Web

13

Architecture n-tiers - Exemple



13

Architecture JEE (1/5)

14

Fournir un cadre du développement par composants avec services JEE = **J**ava **E**ntreprise **E**dition.

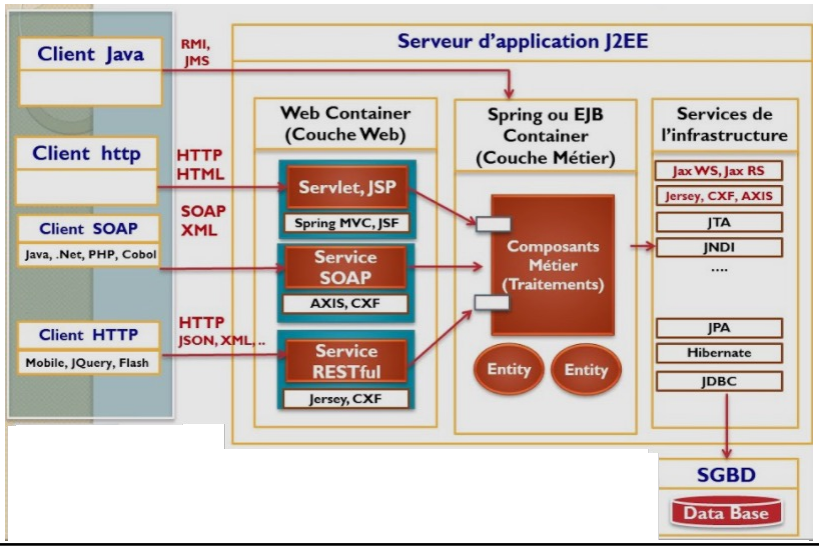
JEE est une plate-forme fortement orientée serveur pour le développement et l'exécution des applications distribuées. Elle propose des APIs :

L'invocation de méthodes distantes: RMI, corba, Web services	L'accès aux bases de données relationnelles: JDBC	L'accès aux annuaires: JNDI	L'utilisation du XML: DOM, SAX	HTM dynamique et traitement de requêtes HTTP: JSP et Servlet	La gestion du mail : JAVA mail

14

Architecture JEE (2/5)

15



15

Architecture JEE (3/5)

16

- Serveur d'application
  - Le rôle du serveur d'application est de faire fonctionner les applications d'entreprises écrites en Java.
  - Le serveur d'application prend en charge l'environnement permettant à l'application d'être disponible sur le web.
  - Il fournit un certain nombre de services :
    - service de nommage
    - service de gestion des transactions
    - service de sécurité
    - service d'administration

16

Architecture JEE (4/5)

17

Enterprise **J**ava **B**eans (EJBs) sont des composants logiciels résidant sur le serveur.

Ils permettent de:

- Représenter des données (Entités).
- offrir des services (Session).
- réaliser des traitements asynchrones (MDB).

17

Architecture JEE (5/5)

18

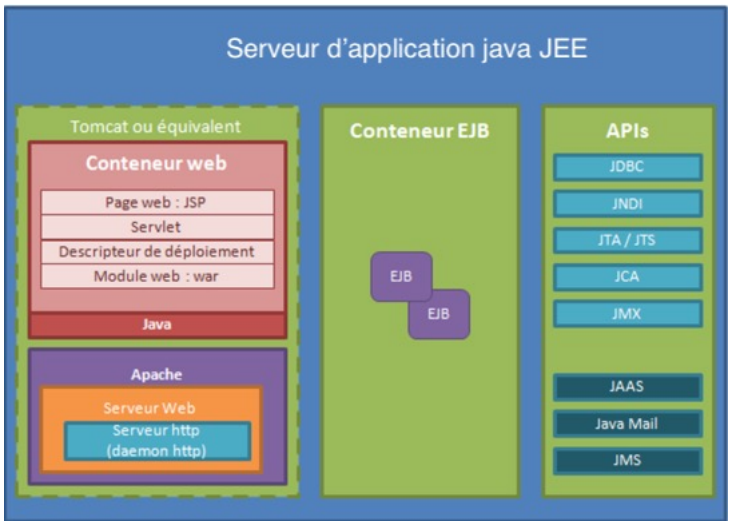
- Serveurs JEE
  - JBoss http://www.jboss.org
  - Glassfish https://glassfish.java.net
- Conteneurs Web
  - Tomcat http://tomcat.apache.org/
  - Jetty http://www.eclipse.org/jetty/
- Conteneur EJB
  - EasyBeans

http://wiki.easybeans.ow2.org/xwiki/bin/view/Main/WebHome

18

Architecture serveur d'application JEE

19



19

20

Chapitre 2

LES SERVLETS

20

Qu'est-ce qu'une Servlet

21

- Programme qui s'exécute coté serveur.
- Permet de gérer des pages web dynamiques.
- Recevoir une requête du client, exécute des traitements et renvoie le résultat.
- Peut être invoquée plusieurs fois en même temps pour répondre à plusieurs requêtes simultanées.
- Elle s'exécute dans un **moteur de servlets** ou **conteneur de servlet** permettant d'établir le lien entre le servlet et le serveur Web.
- Se positionne dans une architecture Client/Serveur trois tiers dans le tiers du milieu entre le client léger charge de l'affichage et la source de données.

21

Avantages

22



Portables et évolutives

- Technologie indépendante de la plate-forme et du serveur
- Quasi déployable quelque soit le serveur
- Utilise le langage (Java)



Performantes et rapides : les servlets sont chargées une seule fois dès le premier appel.



Puissance

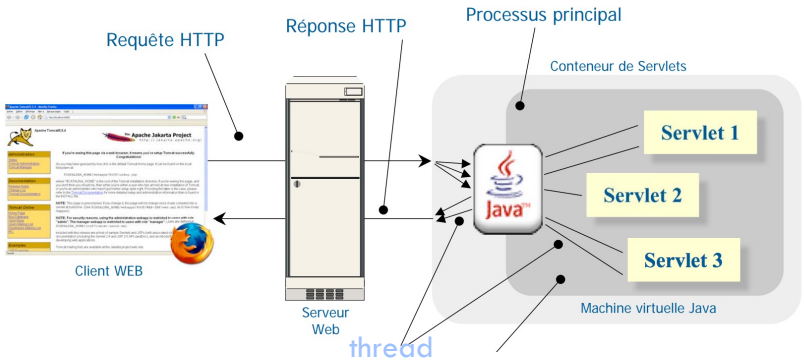
- Disponibilité de l'API de Java
- Manipulation d'images, connectivite aux bases de données (JDBC), …

22

Architecture et fonctionnement (1/4)

23

- Architecture



Les servlets peuvent être toutes gérées par des threads séparés au sein d'un même processus de machine virtuelle

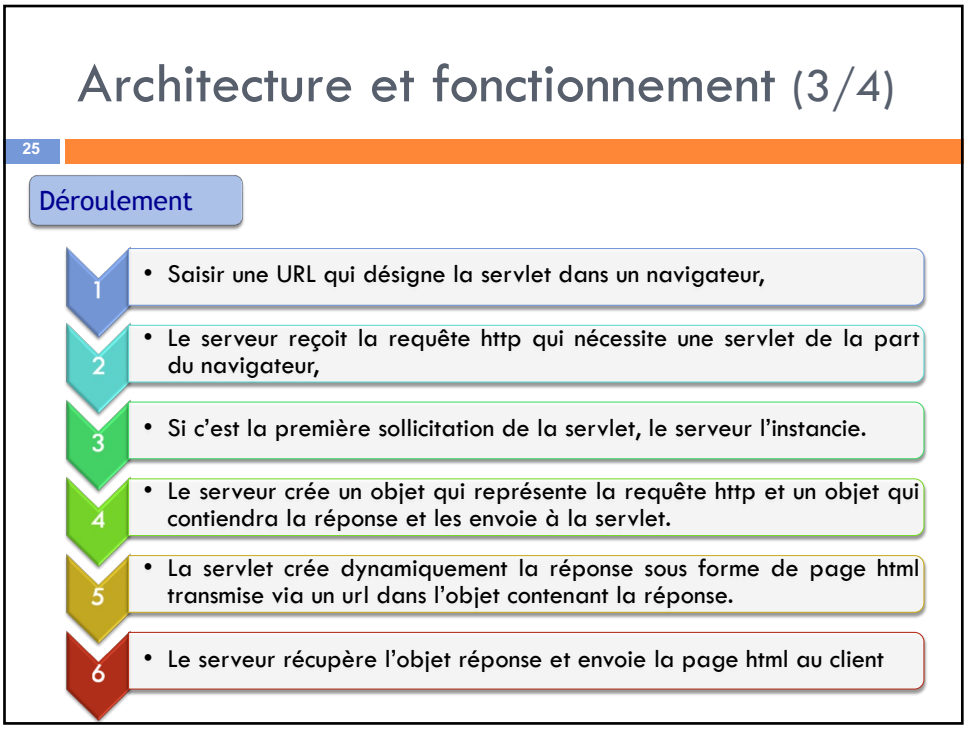
23

Architecture et fonctionnement (2/4)

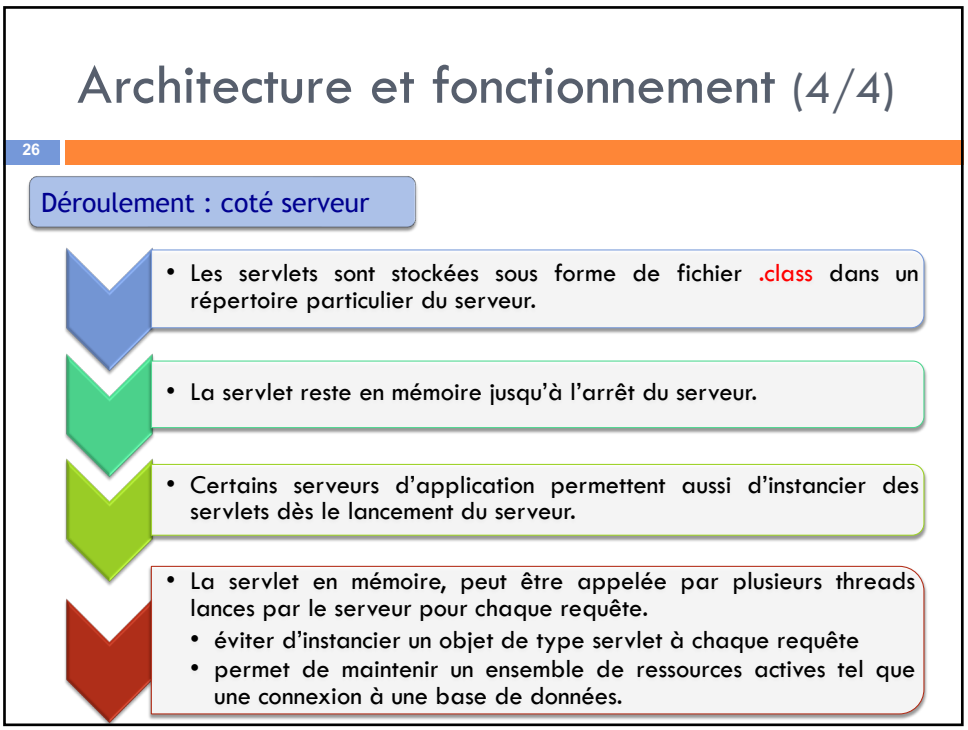
24

- Fonctionnement
  - Les servlets sont exécutées dans une JVM par un serveur d'application.
  - Le serveur d'application contient un moteur de servlets
    - Gère les servlets.
    - C'est un plug-in (JRun, tomcat) pour des serveurs Web existants
  - Les clients Web possèdent un interpréteur Java permettant de lancer des applets (d'où le nom de servlets)

24


Architecture et fonctionnement (3/4)
25
Déroulement
1
Saisir une URL qui désigne la servlet dans un navigateur,
2
Le serveur reçoit la requête http qui nécessite une servlet de la part du navigateur,
3
Si c'est la première sollicitation de la servlet, le serveur l'instancie.
4
Le serveur crée un objet qui représente la requête http et un objet qui contiendra la réponse et les envoie à la servlet.
5
La servlet crée dynamiquement la réponse sous forme de page html transmise via un url dans l'objet contenant la réponse.
6
Le serveur récupère l'objet réponse et envoie la page html au client

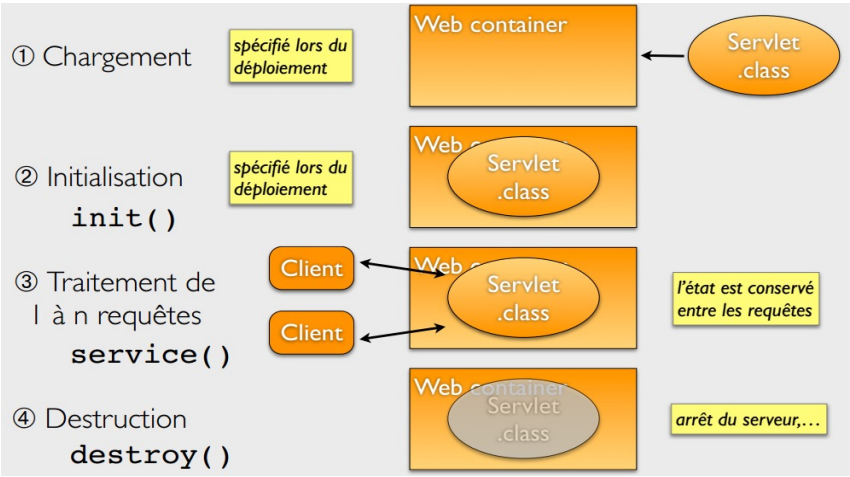
25


Architecture et fonctionnement (4/4)
26
Déroulement : coté serveur
Les servlets sont stockées sous forme de fichier .class dans un répertoire particulier du serveur.
La servlet reste en mémoire jusqu'à l'arrêt du serveur.
Certains serveurs d'application permettent aussi d'instancier des servlets dès le lancement du serveur.
La servlet en mémoire, peut être appelée par plusieurs threads lances par le serveur pour chaque requête.
éviter d'instancier un objet de type servlet à chaque requête
permet de maintenir un ensemble de ressources actives tel que une connexion à une base de données.

26

Cycle de vie d'une Servlet (1/3)

27



27

Cycle de vie d'une Servlet (2/3)

28

Étapes

① **Chargement**
- Création et initialisation de la servlet (utilisation des paramètres initiaux)

② **Initialisation**
- Une fois crée, la servlet est initialisée avec cette méthode **init()**
- Elle n'est appelée qu'une seule fois par le serveur
- Similaire à un constructeur
- On peut y placer les connexions réseaux, les connexions BDD, la récupération des paramètres d'initialisation, la récupération des paramètres de contexte de l'application.

28

Cycle de vie d'une Servlet (3/3)

29

Étapes

③ **Traitements de requêtes**

- Appelée automatiquement par le serveur à chaque requête
- Dispatch en fonction du type de requête (GET/POST)

④ **Destruction**

- Destruction de la servlet via la méthode **destroy**
- Appelée qu'une seule fois quand la servlet est déchargée par le serveur, ou quand le serveur est arrêté
- On y place la fermeture des connexions réseaux, BDD, etc.

29

API Servlet : organisation

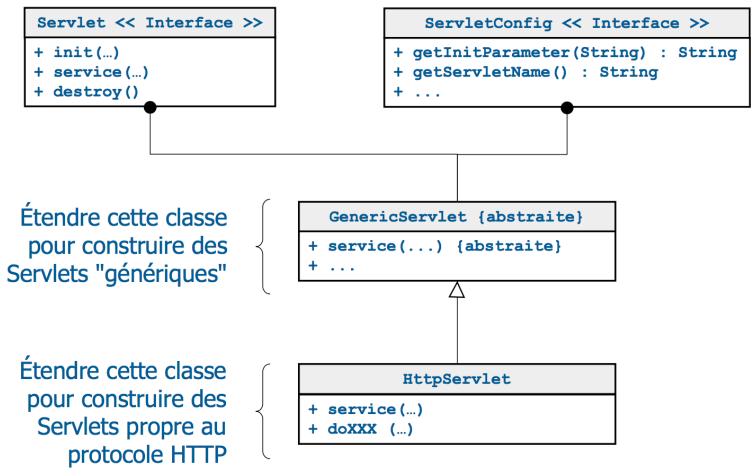
30

- L'API servlets est une extension du JDK de base
- **Packages**
  - **javax.servlet** contient les classes pour développer des servlets génériques indépendantes d'un protocole
  - **javax.servlet.http** : contient les classes pour développer des servlets qui reposent sur le protocole http permettant également la gestion des sessions
- **Classes abstraites** implémentant deux interfaces
  - **javax.servlet.GenericServlet** indépendante du protocole
  - **javax.servlet.http.HttpServlet** spécifique à HTTP
- **Interfaces**
  - **javax.servlet.ServletConfig**
  - **javax.servlet.Servlet**

30

API Servlet du générique à HTTP

31



31

API Servlet: interface Servlet (1/2)

32

- void service (ServletRequest req, ServletResponse res) exécutée par le conteneur chaque requête du client déclenché une seule exécution de cette méthode
- void init(ServletConfig conf) initialiser la servlet, appelée une seule fois après l'instanciation de la servlet. Aucun traitement ne peut être effectuée par la servlet tant que l'exécution de cette méthode n'est pas terminée.
- ServletConfig getServletConfig() Renvoie l'objet ServletConfig passée à la méthode **init()**

32

API Servlet: interface Servlet (2/2)

33

- String getServletInfo() Renvoie des informations sur la servlet
- void destroy() appelée par le serveur lors de la destruction de la servlet permet de libérer proprement certaines ressources (Fichiers, BDD, etc.)

Une servlet doit implémenter l'interface javax.servlet.Servlet, soit directement, soit en dérivant d'une classe qui implémente déjà cette interface (comme GenericServlet ou HttpServlet)

33

API Servlet

34

ServletRequest

- Cette interface définit plusieurs méthodes qui permettent d'obtenir des données sur la requête du client :
  - **ServletInputStream getInputStream()** Permet d'obtenir un flux pour les données de la requête
  - **BufferedReader getReader()** Idem

ServletResponse

- **SetContentType()** Permet de préciser le type MIME de la réponse, i.e., le format de sortie.
- **ServletOutputStream getOutputStream()** Permet d'obtenir un flux pour envoyer la réponse
- **PrintWriter getWriter()** Permet d'obtenir un flux pour envoyer la réponse

34

API Servlet: la classe HttpServlet

35

- Dans la suite du cours nous allons utiliser uniquement des Servlets qui réagissent au protocole HTTP d'où l'utilisation de la classe HttpServlet:
  - La classe HttpServlet permet de fournir une implémentation de l'interface Servlet spécifique à HTTP
  - La classe HttpServlet surcharge la méthode service en lisant la méthode HTTP utilisée par le client
  - Elle redirige la requête vers une méthode appropriée de HttpServlet destinée à traiter le type de requête (GET, POST)

35

API Servlet

36

Exemple

```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class ServletExp extends HttpServlet {
    private static final long serialVersionUID = 1L;

    public ServletExp() {
        super();
        // TODO Auto-generated constructor stub
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        // TODO Auto-generated method stub

    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        // TODO Auto-generated method stub
    }

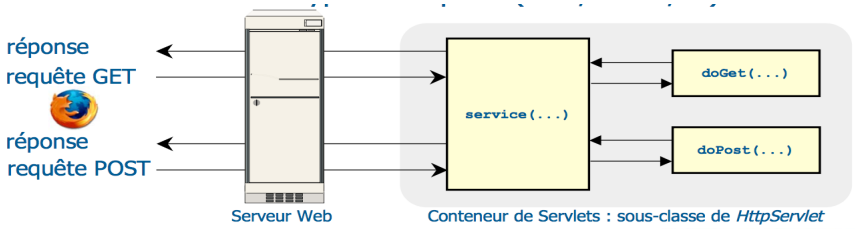
}
```

36

La classe HttpServlet

37

- Une servlet Web étend la classe javax.servlet.http.HttpServlet
  - elle implémente javax.servlet.Servlet
- HttpServlet redéfinit la méthode service(..)
  - service(..) lit la méthode (GET, POST, ...) à partir de la requête Elle transmet la requête à une méthode appropriée de HttpServlet destinée à traiter le type de requête (GET, POST, etc.)



37

HttpServlet: méthodes de traitement de requêtes

38

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(...), doDelete(…)

Selon le type de requête (GET ou POST) le concepteur redéfinit la méthode concernée.

Nous utiliserons essentiellement les méthodes **doPost(...)** et **doGet(...)**

38

HttpServlet : requête et réponse

39

- La méthode service(...) et par conséquent les méthodes de traitement de requêtes (ex : doPost(...)) sont appelées avec :
  - un objet requête
  - un objet réponse

```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class ServletExp extends HttpServlet {
    private static final long serialVersionUID = 1L;

    public ServletExp() {
        super();
        // TODO Auto-generated constructor stub
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        // TODO Auto-generated method stub

    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        // TODO Auto-generated method stub
    }

}
```

39

HttpServlet : objet requête HttpServletRequest

40

- 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 getServeurName() : 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, …

40

HttpServlet : objet requête HttpServletRequest

41

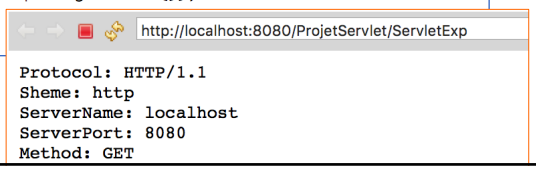
- Exemple: Servlet qui affiche un certains nombre d'informations issues de HttpServletRequest

```
import java.io.IOException;

public class ServletExp extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/plain");
        PrintWriter out = response.getWriter();
        out.println("Protocol: "+request.getProtocol());
        out.println("Sheme: "+request.getScheme());
        out.println("ServerName: "+request.getServerName());
        out.println("ServerPort: "+request.getServerPort());
        out.println("Method: "+request.getMethod());

    }
```



41

HttpServlet : objet réponse HttpServletResponse

42

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

42

HttpServlet : objet réponse HttpServletResponse

43

- Exemple: affichage

```
import java.io.IOException;

public class ServletExp extends HttpServlet {
    private static final long serialVersionUID = 1L;

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

        PrintWriter out = response.getWriter();
        out.print("Bonjour, ma premiere servlet");

    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
}
```



43

HttpServlet : objet réponse HttpServletResponse

44

- Exemple 2: effectue une redirection vers un site web
  - Utilisation de la méthode sendRedirect(String)

```
import java.io.IOException;

public class ServletExp extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/plain");
        response.sendRedirect("http://www.google.tn");

    }
```



44

Servlets et formulaires : du coté HTML

45

- Utilisation de la balise **<FORM> </FORM>**
  - Option METHOD : type de requête GET ou POST
  - Option ACTION : l'URL où envoyer les données
- Utilisation de composants IHM pour saisir des informations
- Contenu à l'intérieur de la balise FORM (bouton radio, case à cocher, zone texte, etc.)
- Chaque composant est défini au moyen d'une balise particulière SELECT, INPUT, TEXTAREA, etc.
- A l'intérieur de chaque balise du composant (SELECT par exemple) plusieurs options et notamment une propriété (NAME) qui permet d'identifier le composant : NAME ="mon_composant"
- Les données sont envoyées quand l'utilisateur clique sur un bouton de type SUBMIT

45

Servlets et formulaires : du coté HTML

46

- Exemple

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Formulaire</title>
</head>
<body>
<p>Formulaire de satisfaction du cours developpement Web</p>
<form name="form1" method="get" action="ServletExp">
<p>
   Nom : <input type="text" name="nom">
   Prénom : <input type="text" name="prenom">
</p>
<p>
    Sexe : <input type="radio" name="radio1" value="mas" checked>
    masculin <input type="radio" name="radio1" value="fem">féminin
</p>
<p>Commentaire :<br>
    <textarea name="textarea" cols="50" rows="10"> </textarea><br>
    <input type="submit" name="Submit" rows="5" value="Envoyer">
</p>
</form>

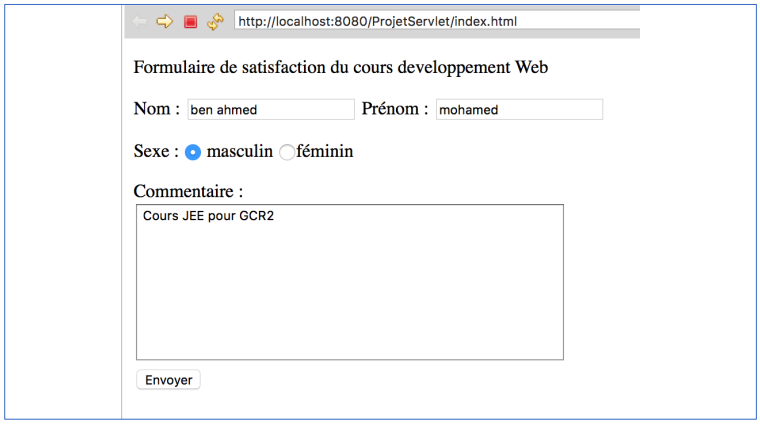
</body>
</html>
```

46

Servlets et formulaires : du coté HTML

47

- Formulaire HTML



47

Servlets et formulaires : du coté Servlet

48

- Pour lire les données du formulaire : traiter la requête
- Accéder par l'intermédiaire de l'objet HttpServletRequest aux paramètres
  - String getParameter(String p) : retourne la valeur du paramètre p
  - String[] getParamterValues(String p) : retourne les valeurs du paramètre p

48

Servlets et formulaires : du coté Servlet

49

- Exemple

```
public class ServletExp extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/plain");
        PrintWriter out = response.getWriter();
        out.println(request.getParameter("nom"));
        out.println(request.getParameter("prenom"));
        String sexe = request.getParameterValues("radio1")[0];

        if(sexe.equals("mas")){
            out.println("c'est un homme");}
        else{
                out.print("c'est une femme");}
        out.println("Qui a comme remarques: ");
        out.println(request.getParameter("textarea"));

    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
```

```
http://localhost:8080/ProjetServlet/S
ben ahmed
mohamed
c'est un homme
Qui a comme remarques:
 Cours JEE pour GCR2
```

49

Architecture de développement d'une application WEB

50

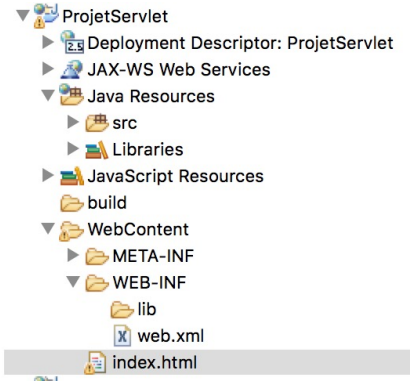
- Une application WEB peut contenir plusieurs Servlets
- Pour tester et déployer une Servlet, il faut un système d'exécution appelé conteneur de Servlets ou moteur de Servlets
- Le conteneur réalise le lien entre la Servlet et le serveur WEB
  - Transforme code Java (bytecode) en HTML
  - Associe à des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

50

Architecture de développement d'une application WEB

51

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



51

Architecture de développement d'une application WEB

52

- Le fichier web.xml regroupe les informations de fonctionnement de l'application WEB (description de déploiement du contexte)
- Utilisation du format XML
  - eXtensible Markup Language
  - Syntaxe universelle pour la structuration des données créée par le World Wide Web Consortium (W3C) en 1996
  - Langage basé sur des balises permettant de donner une signification au document (HTML s'intéresse essentiellement à l'affichage)
  - Extensible par la possibilité de créer de nouvelles balises
  - Balises XML sensibles à la case et à la rigueur (balise ouvrante = obligation d'une balise fermante)
  - La structure du document est défini par un fichier XSD (XML Schema Description)

52

Architecture de développement d'une application WEB

53

- La présence du fichier web.xml est obligatoire pour que vos servlets WEB fonctionnent

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java
id="WebApp_ID" version="2.5">
  <display-name>ProjetServlet</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>
  <servlet>
    <description></description>
    <display-name>ServletExp</display-name>
    <servlet-name>ServletExp</servlet-name>
    <servlet-class>ServletExp</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>ServletExp</servlet-name>
    <url-pattern>/ServletExp</url-pattern>
  </servlet-mapping>
</web-app>
```

53

Architecture de développement d'une application WEB

54

- Structure du fichier web.xml
  - web-app: La description d'une Application Web
  - servlet: La relation entre le nom de la servlet et la classe qui l'implante
  - servlet-mapping: La relation entre le nom de la servlet et l'url lui associé
  - servlet-name: contient une chaîne de caractère identifiant la Servlet au sein de l'application web.
  - url-pattern: définit comment une Servlet est invoquée.

54

Collaboration de servlets

55

- Les applications sont souvent composées de plusieurs servlets. On a besoin de faire des appels entre les servlets
- Deux principaux styles de collaboration:
  - Partage d'information: un état ou une ressource.
    - Exemple : un magasin en ligne pourrait partager les information sur le stock des produits ou une connexion à une base de données
  - Partage du contrôle : une requête.
    - Exemple : Réception d'une requête par une Servlet et laisser l'autre Servlet une partie ou toute la responsabilité du traitement

55

Collaboration de servlets

56

- La collaboration est obtenue par l'interface ServletContext
- L'utilisation de ServletContext permet aux applications web de disposer de son propre conteneur d'informations unique
- Une Servlet retrouve le ServletContext de son application web par un appel à getServletContext()
- Exemples de méthodes
  - void setAttribute(String name, Object o) : lie un objet sous le nom indique
  - Object getAttribute(String name) : retrouve l'objet sous le nom indique
  - Enumeration getAttributeNames() : retourne l'ensemble des noms de tous les attributs lies
  - void removeAttribute(String name) : supprime l'objet lié sous le nom indique

56

Collaboration de servlets

57

- Le support de la délégation de requête est obtenu par l'interface RequestDispatcher
- Une Servlet obtient une instance sur la requête
  - RequestDispatcher getRequestDispatcher(String path) : retourne une instance de type RequestDispatcher par rapport à un composant
  - Un composant peut être de tout type : Servlet, JSP, fichier statique
  - path est un chemin relatif ou absolu ne pouvant pas sortir du contexte
- Pour distribuer en dehors du contexte courant il faut :
  - Identifier le contexte extérieur (utilisation de getContext())
  - Utiliser la méthode getRequestDispatcher(String path)
  - Le chemin est uniquement en absolu

57

Collaboration de servlets

58

- Exemple:

```
Client cl1 = new Client(1,"toto","toto");
Client cl2 = new Client(2,"ali","toto");
request.setAttribute("cl1", cl1);
request.setAttribute("cl2", cl2);
request.getRequestDispatcher("serv2").forward(request,
response);
```

58

Collaboration de servlets

59

- Un objet ServletContext est créé par le conteneur Web au moment du déploiement du projet. Cet objet peut être utilisé pour obtenir des informations de configuration à partir du fichier web.xml. Il n'y a qu'un seul objet ServletContext par application Web.
- Si des informations sont partagées avec plusieurs servlets, il est préférable de les fournir à partir du fichier web.xml en utilisant l'élément :
  - <context-param>.

59

Collaboration de servlets

60

- Exemple de ServletContext pour obtenir le paramètre d'initialisation :

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

    response.setContentType("text/html");
    PrintWriter out =response.getWriter();
    ServletContext context = getServletContext();

    String drname=context.getInitParameter("dname");
    out.println("driver name is="+drname);


}
```

```
<servlet>
  <description></description>
  <display-name>ServPrincipal</display-name>
  <servlet-name>ServPrincipal</servlet-name>
  <servlet-class>controller.ServPrincipal</servlet-class>
</servlet>
<context-param>
  <param-name>dname</param-name>
  <param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
  </context-param>
<servlet-mapping>
  <servlet-name>ServPrincipal</servlet-name>
  <url-pattern>/ServPrincipal</url-pattern>
</servlet-mapping>
<servlet>
```

60

Collaboration de servlets

61

ServletConfig vs ServletContext

- ServletConfig et ServletContext
  - Tous deux sont des objets créés au moment de l'initialisation du servlet et utilisés pour fournir des paramètres initiaux ou des informations de configuration au servlet.
  - La différence réside dans le fait que les informations partagées par ServletConfig sont destinées à un servlet spécifique, tandis que les informations partagées par ServletContext sont disponibles pour tous les servlets de l'application Web.

61

Collaboration de servlets

62

ServletConfig

- Exemple :
  - Supposons que l'on crée un portail d'emploi et que l'on souhaite partager différents identifiants de messagerie avec le recruteur et le candidat.
  - On décide donc d'écrire deux servlets, l'un pour gérer la demande du recruteur et l'autre pour le candidat.

62

Collaboration de servlets

63

ServletContext

- Exemple :
  - Supposons que le nom de ce portail d'emploi soit «tanitjob.tn». En affichant le nom du site Web en haut des pages Web fournies par différentes servlets, il faut stocker le nom du site Web dans chaque servlet invitant à la redondance.
  - Étant donné que les informations partagées par ServletContext sont accessibles par chaque Servlet, il est préférable d'utiliser ServletContext et de récupérer le nom du site Web à l'aide de :

```
getServletContext().getInitParameter("Nom");
```

chaque fois que nécessaire.

63

Collaboration de servlets

64

```
<servlet>
   <description></description>
    <display-name>
        ServrRcruteur
    </display-name>
    <servlet-class>
        controler.ServRecruteur
     </servlet-class>
    <init-param>
      <param-name>
        Email
      </param-name>
      <param-value>
        recruteur@tanitjob.tn
      </param-value>
    </init-param>
  </servlet>
```

```
<servlet>
   <description></description>
    <display-name>
        ServCandidat
    </display-name>
    <servlet-class>
        controler.ServCandidat
     </servlet-class>
    <init-param>
      <param-name>
        Email
      </param-name>
      <param-value>
        candidat@tanitjob.tn
      </param-value>
    </init-param>
  </servlet>
```

web.xml

64

Collaboration de servlets

65

```
<web-app …>
  <display-name>
      MonProjetWeb
  </display-name>

<context-param>
    <param-name>
      NomSiteWeb
    </param-name>
    <param-value>
      tanitjob.tn
     </param-value>
  </context-param>
</web-app>
```

web.xml

65

Collaboration de servlets

66

```
protected void doGet(HttpServletRequest request,
HttpServletResponse response)
      throws ServletException, IOException {
response.setContentType("text/html;charset=UTF-8");

String email=
getServletConfig().getInitParameter("Email");
String siteweb = getServletContext()
              .getInitParameter("NomSiteWeb");
PrintWriter out = response.getWriter();
out.println("<center> <h1>" + siteweb
+ "</h1> </center> <br> <p> Contact us:"
                + email);
}
```

Code : ServletRecruteur & ServletCandidat

66

Collaboration de servlets

67



67

Gestion des sessions

68

- Problématique:
  - Comment maintenir l'état d'un client au cours d'une série de requêtes d'un même utilisateur pendant un temps donné ?
- Définition:
  - Une session est une suite d'interactions entre un client et un serveur Web
  - Elle couvre plusieurs requêtes HTTP sur une période donnée
  - Utilisée pour mémoriser les actions d'un utilisateur unique (Exp. achats en ligne)
- Principe:
  - Un objet session peut être associé à chaque requête
  - Il sert comme « conteneur » pour des informations persistantes
  - Durée de vie limitée = celle de la session

68

Traitement des objets HttpSession

69

Tout d'abord il faut tester s'il y'a un objet session dans la requête:

Si la méthode getSession() retourne null, alors il n'y a pas de session et il faut créer une nouvelle.

Avec l'instruction suivante on retourne la session courante s'il y en a une, sinon on crée une nouvelle

```
HttpSession session =
request.getSession(true);
```

On peut toujours tester par la méthode isNew() si une session est nouvelle

69

Traitement des objets HttpSession

70

- L'information contenue dans un objet HttpSession est de la forme:
  - (attribut : String, valeur : Object)
- La méthode getAttribute (String) :
  - permet d'extraire la valeur correspondante à un attribut donné dans l'objet session.
  - Elle retourne un Objet de type Object
  - Il est nécessaire de faire le casting pour manipuler la valeur retournée.
- Exemple :
  - Panier p=(Panier) session.getAttribute ("Schopping");
  - Cette méthode retourne null si l'attribut n'existe pas.

70

Traitement des objets HttpSession

71

La méthode setAttribute(String, valeur): Permet d'insérer une paire (attribut, valeur) dans un objet de type HttpSession

Si l'attribut existe déjà dans l'objet session sa valeur sera mise à jour.

Exemple :

```
Panier p = new Panier();
session.setAttribute("Schopping", p);
```

71

Traitement des objets HttpSession

72

La méthode removeAttribute(String att): Permet de supprimer un pair (attribut, valeur) dans un objet de type HttpSession

Exemple :

```
session.removeAttribute("Schopping");
```

72

Traitement des objets HttpSession

73

Les méthodes :

- String[] getAttributeNames()
- Enumeration getAttributeNames()
- Permettent de retourner les noms de tous les attributs dans un objet session.

La méthode :

- String getId()
- retourne l'identificateur unique généré pour chaque session

La méthode :

- void invalidate()
- permet d'annule une session et détacher tous les objets associés

73

Traitement des objets HttpSession

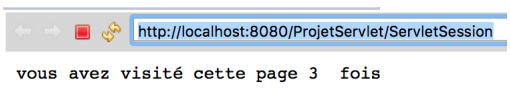
74

Exemple

```
public class ServletSession extends HttpServlet {
    private static final long serialVersionUID = 1L;

   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 = 1;
        else
            count= new Integer(count+1);
        session.setAttribute("count", count);
        out.println("vous avez visité cette page "+count + "  fois");
```



74

Connexions à la base de données

75





Utilisation de l'API JDBC (Java Data Base Connectivity)

JDBC est une API du niveau SQL, elle permet d'exécuter des instructions SQL et de retrouver les résultats (s'il y en a)

L'API est un ensemble d'interfaces et de classes conçues pour exécuter des actions sur toute base de données (MySQL, ORACLE, SQL Server, …)

JDBC en quatre étapes:

Charger le pilote du driver approprie à la BDD

Se connecter à la base

Créer et exécuter une requête SQL

Traiter le résultat si nécessaire

75

Connexions à la base de données

76

Exemple

```
public class DataBaseServlet extends HttpServlet {

protected void doGet(HttpServletRequest req, HttpServletResponse res)
            throws ServletException, IOException { res.setContentType("text/html");
PrintWriter out = res.getWriter();
Try { Class.forName("com.mysql.jdbc.Driver"); // Pont jdbc
    Connection con =
     DriverManager.getConnection("jdbc:mysql://localhost:3306/MaBase","user","pswd");
    Statement stmt = con.createStatement ();
     ResultSet resultats = stmt.executeQuery ( "SELECT * FROM pers");
     while (resultats.next()) {
            String name = resulats.getString(1);
            int Age = resultats.getInt ("age")
            out.println("Nom"+name +","+Age+" ans");
            }
       }
Catch(SQLException e ) { }
    } //  fin de la classe
```

76

77

Chapitre 3

JAVA SERVER PAGE (JSP)

77

JSP: Définition

78

- JSP=**J**ava **S**erver **P**age
- JSP est est un langage de scripts puissant pour construire des applications Web
- Les scripts sont des instructions Java embarquées dans du code HTML, SVG, WML, XML entre les balises <% %>
- Chaque script est interprété par un serveur JSP qui différencie le code JSP du code HTML grâce à des balises spécifiques
  - Exécutée cote serveur et non cote client
  - Portable
  - Séparent la présentation du contenu
  - Mélange le HTML et des directives
  - Permet la mise en ouvre facile des sites dynamiques
  - Basée sur Java

78

JSP: Présentation (1/3)

79

Avec les Servlets, il est facile de :

- lire des formulaires
- lire les entêtes HTTP
- envoyer les entêtes réponses, les codes d 'état, les pages réponses
- partager les données entre Servlets
- mémoriser des informations entre les requêtes

Mais il est pénible de :

- gérer les instructions println de génération HTML

79

JSP: Présentation (2/3)

80

- Principe
  - Utilisation du HTML classique pour la majorité de la page,
  - Ajout du code Java marqué par des balises spéciales,
  - La page JSP est traduite par le serveur en Servlet une seule fois,
  - la Servlet obtenue est invoquée à chaque requête.
- Exemple d'une portion de page JSP
  - <h3>Nous sommes le</h3>
  - <label><%= new java.util.Date().toString() %></label>
- Résultat



80

JSP: Présentation (3/3)

81

- Avantage des JSP
  - Écriture directe du HTML,
  - Possibilité d'utiliser les outils HTML classiques,
  - Possibilité de séparer la présentation réalisée en HTML et le traitement réalisé en Java,
  - Possibilité de modularité et de réutilisation en utilisant des beans dans la partie Java,
  - Appel des pages JSP comme des pages HTML standards, par utilisation d'une URL avec un nom de fichier avec une extension .jsp
    - http:\\localhost:8080\MaWebApp\NomFichier.jsp
    - http:\\localhost:8080\MaWebApp\package\NomFichier.jsp

81

JSP: Principe et fonctionnement (1/2)

82

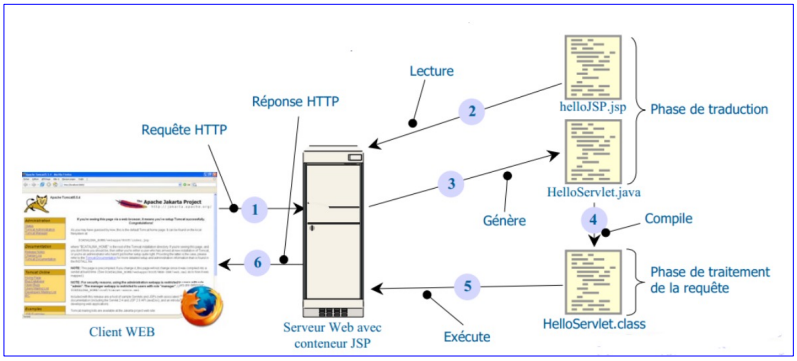
- Une JSP est un fichier contenant du code HTML et des fragments de code Java exécutes par un conteneur Web (moteur de Servlets)
- Les pages JSP sont converties en Servlets par le conteneur Web lors du premier appel de la JSP. Le code associé, figure à l'intérieur de la méthode jspService()
- Démarrage de la Servlet
- C'est la Servlet générée qui répond aux requêtes.

82

JSP: Principe et fonctionnement (2/2)

83

- Le processus de compilation comporte trois étapes:
  - Analyser (Parsing) le JSP
  - Transformer le JSP en servlet
  - Compilation de la servlet.



83

Cycle de vie d'un JSP

84

- Une page JSP a un cycle de vie comparable au servlet (init(), service(), destroy())
  - jspInit(): est appelée après le chargement de la page
  - jspService (accept request, generate response): est appelée à chaque requête
  - _jspDestroy(): est appelé lors du déchargement
- Possibilité de redéfinir dans le code JSP les méthodes jspInit() et jspDestroy() en utilisant un élément de scripts déclaration.

84

Les balises JSP

85

- Un script JSP contient :
  - Des balises HTML :
    - Exemples:

<html>, <head>, <title>, <body>, </title>, </head>,</body>,</html>…

  - Des Balises JSP :
    - Les balises de commentaire,
    - Les balises de déclaration,
    - Les balises de scriptlet,
    - Les balises d'expression,
    - Les balises de directives,
  - Des Variables Prédéfinis

85

Les balises JSP: les objets implicites

86

- Les objets implicites sont les objets présents dans la méthode service(...) qui ont été employés dans la partie Servlet
- Ils sont identifies par des noms des variables uniques
  - request (request scope): HttpServletRequest, requête courante
  - response (page scope) : HttpServletResponse, reponse courante
  - pageContext (page scope): PageContext, utilise pour partager directement des variables entre des pages JSP et supportant les beans et les balises

86

Les balises JSP: les objets implicites

87

- out (page scope) JspWriter: flot de sortie permet l'écriture sur la réponse
- session (session scope) : HttpSession, session courante
- application (application scope) ServletContext, contient des méthodes log() permettant d'écrire des messages dans le journal du contenu (ServletContext)
- config (page scope) : ServletConfig
- Exception (page scope): Throwable, disponible uniquement dans les pages erreurs donnant information sur les erreurs

87

Les balises JSP: classification

88

- Les commentaires
  - faire un commentaire dans le code JSP
  - Encadrées par <% -- -- %>
- Les directives
  - Instruction pour le moteur JSP
  - Informations globales relatives à la page.
  - Encadrées par < %@ % >
- Les déclarations
  - Déclaration des variables ou des méthodes utilisables dans la page
  - Encadrées par < %! % >
- Les expressions
  - Un expression est évaluée, transformée en chaine et incluse dans la page
  - Encadrées par <%= %>
  - Les scriptlets morceau de code java exécuté dans la page
  - Encadrés par < % % >

88

Les balises JSP: déclaration

89

- Déclaration des variables et des fonctions locales à la page (en java)

```
<%@ page language="java" contentType="text/html;
charset=UTF-8" pageEncoding="UTF-8"%>
<html>
<head>
<title>Ma premiere page JSP</title>
</head>
<body>
<%! int i=0; %>
<%! String s=" GCR"; %>
<%! public String affiche(){
        return "Bonjour";}
    %>
</body>
</html>
```

89

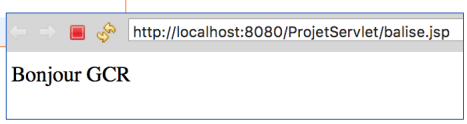
Les balises JSP: les expressions

90

- Une expression est une expression java dont la valeur est un String et insérée dans la page.

```
<body>
<%! int i=0; %>
<%! String s=" GCR"; %>
<%! public String affiche(){
        return "Bonjour";}
    %>
<div> <%= affiche()+s%></div>

</body>
```



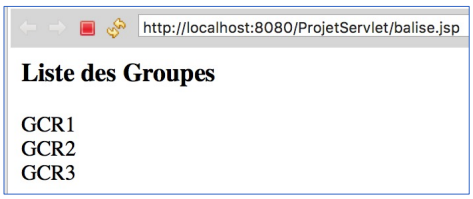
90

Les balises JSP: les scriptlets

91

- Peuvent contenir des fragments de code valide en java
- Ils sont exécutés quand la requête est évaluée
- Il s'agit d'un code Java qui sera traduit en code dans la méthode service() du servlet résultant
- est un programme Java embarqué dans une page HTML

```
<H3> Liste des Groupes</H3>
<div><% for (i=1; i<4; i++) {%>GCR<%=i%><br><%} %> <br> </div>
```



91

Les balises JSP: les directives (1/10)

92

- Définition:
  - des messages passes au moteur du JSP places à l'entête de la JSP
  - Comprises entre < %@ % >
  - On distingue les directives:
    - page informations relatives à la page.
    - include fichiers à inclure littéralement, permet d'inclure un fichier texte ou autre ressource à la création de la servlet.
    - taglib permet de définir une bibliothèque de balises à utiliser
  - Syntaxe
    - < %@ directive attr="value" % >
    - attr={import, extends, contentType, pageEncoding, isThreadSafe, session, buffer, errorPage, etc}

92

Les balises JSP: les directives (2/10)

93

- Valeurs possible
  - import importer des types ou des packages (les packages lang, servlet et jsp sont importes par défaut)
    ```
    <%@ page import="java.util.Date"%>
    ```
  - langage définit le langage de script utilisé dans la page
    ```
    <%@ page language="java"%>
    ```
  - contentType définit le type de contenu de la page générée
    ```
    <%@ page contentType="text/html"%>
    ```
  - session indique si La page participe à une session (true si c'est le cas, false sinon)
    ```
    <%@ page session="true"%>
    ```

93

Les balises JSP: les directives (3/10)

94

- Valeurs possible (suite)
  - errorPage spécifie la page à afficher en cas d'erreur
    ```
    <%@ page errorPage="/erreur.jsp"%>
    ```
  - isErrorPage vaut true si la page est une page d'erreur, false sinon
    ```
    <%@ page isErrorPage="true"%>
    ```
- Dans la JSP provoquant l'erreur :
  - On peut récupérer une exception déclenchée dans une scriptlet JSP.
  - Son traitement pourra être assurée par une page d'erreur
  - La page d'erreur est spécifiée par la balise page et son attribut errorPage
    ```
    <%@ page errorPage="/erreur.jsp" %>
    ```

94

Les balises JSP: les directives (4/10)

95

- Dans la page de traitement de l'erreur:
  - on definit la page JSP comme une page d'erreur par
    - < %@ page isErrorPage=true % >
  - objet accompagnant l'exception est référencé par la variable implicite exception. On peut donc recueillir des informations sur l'erreur par les méthodes classiques de la classe Exception:
    - exception.getMessage(): renvoie le message d'erreur décrivant l'exception
    - exception.printStackTrace(): retourne la liste des appels de méthodes ayant conduit `a l'exception

95

Les balises JSP: les directives (5/10)

96

- Exemple : la page "division.jsp"

```
<%@ page language="java" contentType="text/html;
    pageEncoding="UTF-8"%>
<%@ page errorPage="erreur.jsp" --%>
<html>
<head>
<title>Division avec erreur</title>
</head>
<body>
<% int x=0; x=x/0; %> Division par zero = <%=x %>
</body>
</html>
```

96

Les balises JSP: les directives (6/10)

97

- Exemple : la page "erreur.jsp"

```
<%@ page language="java" contentType="text/html;
    pageEncoding="UTF-8"%>
<%@page isErrorPage="true" %>
<html>
<head>
<title>Page d'erreur</title>
</head>
<body>
<b>ERREUR : </b> <%= exception.getMessage() %>
</body>
</html>
```

97

Les balises JSP: les directives (7/10)

98

- Exemple : la trace d'exécution
  - Sans la gestion des exceptions

Etat HTTP 500 - An exception occurred processing JSP page /division.jsp at line 9

type Rapport d"exception

message An exception occurred processing JSP page /division.jsp at line 9

description Le serveur a rencontré une erreur interne qui l"a empêché de satisfaire la requête.

exception

```
org.apache.jasper.JasperException: An exception occurred processing JSP page /division.jsp at line 9

6: <title>Division avec erreur</title>
7: </head>
8: <body>
9: <% int x=0; x=x/0; %> Division par zero = <%=x %>
10: </body>
11: </html>
```

  - Avec la gestion des exceptions

http://localhost:8080/ProjetServlet/division.jsp

ERREUR : / by zero

98

Les balises JSP: les directives (8/10)

99

- La directive "include"
  - include permet d'inclure dans la page un fichier texte ou autre ressource à la création de la servlet
  - Cette inclusion se fait au moment de la conversion, pendant la création de la servlet correspondante.
  - Les ressources à inclure doivent être contenues dans le contexte de l'application web

```
<%@  include file="/autrePage.jsp" %>
```

  - l'inclusion est statique, elle se fait au moment de la compilation

99

Les balises JSP: les directives (9/10)

100

- Exemple: directive "include"
  - Fichier "entete.html"

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>entete</title>
</head>
<body>
```

  - Fichier "corps.jsp"

```
<%! int i=0; %>
<%! String s=" GCR"; %>
<%! public String affiche(){
        return "Bonjour";}
%>
```

  - Fichier "pied.html"

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

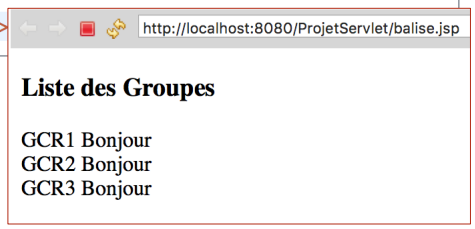
100

Les balises JSP: les directives (10/10)

101

- Exemple: directive "include"
  - Fichier de récupération : "balise.jsp"

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8" %>
<%@  include file="/entete.html" %>
<%@ include file="/corps.jsp" %>
<H3> Liste des Groupes</H3>
<div><% for (i=1; i<4; i++) {%>GCR<%=i %> <%=affiche()%><br><%} %>
</div>
 <%@ include file="/pied.html" %>
```



101

102

Chapitre 4

JAVA BEANS

102

Java Beans

103

- Les JavaBeans sont des classes Java (POJO) qui suivent certaines conventions :
  - Doivent avoir un **constructeur vide** (aucun paramètre)
    - On peut satisfaire cette contrainte soit en définissant explicitement un tel constructeur, soit en ne spécifiant aucun constructeur
- Ne doivent pas avoir d'attributs publics
  - Une bonne pratique réutilisable par ailleurs
- La valeur des attributs doit être manipulée à travers des accesseurs (getters et setters)
  - Si une classe possède une méthode getTitle qui retourne un String, on dit que le bean possède une propriété String nommée title
  - Les propriétés boolean utilisent isXXX à la place de getXXX

103

Java Beans : utilisation

104



Une fois la classe créée, les beans sont des objets Java sur lesquels on peut faire ces actions :

instanciation d'un nouveau Bean
récupération de la valeur d'une propriété du Bean
affectation/modification de la valeur d'une propriété du Bean



Pour faire cela dans une Servlet, ça ne pose pas de problème particulier, le bean est traité comme un objet Java standard.



Cependant, dans une JSP, l'utilisation des balises de scriptlets n'est pas très "propre", on utilisera plutôt l'une des manières suivantes :

balises JSP:

- <jsp:useBean>, <jsp:getProperty>, <jsp:setProperty>

104

Java Beans et JSP

105

Exemple

```
package model;
import java.io.Serializable;
public class Client implements
Serializable {
        private int numCl;
        private String nom;
        private String prenom;
public Client(){}
public Client(int numCl, String nom,
String prenom) {
        this.numCl = numCl;
        this.nom = nom;
        this.prenom = prenom;
        }
```

```
public int getNumCl() {
        return numCl;
        }
public void setNumCl(int numCl) {
        this.numCl = numCl;
        }
public String getNom() {
        return nom;
        }
public void setNom(String nom) {
                this.nom = nom;
        }
        public String getPrenom() {
                return prenom;
        }
        public  void  setPrenom(String
prenom) {
                this.prenom = prenom;
        } }//fin classe
```

105

Java Beans et JSP

106

- Pour déclarer et allouer un Java Bean dans une page JSP il faut employer l'action: <jsp:useBean>

```
<jsp:useBean id="Reference_Bean" class="Pakage.NomClass" scope="attribut"/>
```

  - id = "nom de l'instance" : nom de l'instance pour identification
  - class = "Nom de la classe" : package du Bean.
  - scope = "attribut" : champ d'existence de l'objet
- Un scope peut être vu comme un **conteneur de beans** stocké du coté du **serveur** (le client ne peut avoir aucune connaissance sur ces beans)

106

Java Beans et JSP: scope

107

- Il existe 4 types de Scopes, que l'on distingue par la durée de vie des beans qui y sont stockés:
  - Requête : ce scope est créé à chaque requête du client, les beans qu'il contient sont détruit lorsque le serveur à envoyer la réponse au client.
  - Session : ce scope est créé automatiquement par le conteneur JEE pour chaque client qui se connecte au serveur. Les beans qu'il contient ne sont visible que par le client détenant la session, et sont détruit lorsque la session du client se termine (géré par un timeout du coté du serveur)
  - Application : ce scope est créé automatiquement au lancement du projet JEE sur le serveur. Les beans qu'il contient sont partagés par tous les clients et ne sont détruit que lors de l'arrêt /rechargement du projet JEE sur le serveur.
  - Page : ce scope restreint la durée de vie des beans aux requêtes POST effectuées sur une page donnée, dès qu'on en sort, les beans sont détruit.
    - Ce scope n'est quasiment pas utilisé

107

Java Beans et JSP: Lecture des propriétés

108

- Pour lire une propriété du Bean deux éléments sont utilises:
  - La référence du Bean définie par l'attribut id
  - Le nom de la propriété
- Deux manières possibles pour interroger la valeur d'une propriété et la convertir en String:
  - En utilisant un tag action <jsp:getProperty>

```
<jsp:getProperty property="Nom_propriété" name="Reference_Bean"/>
```

    - *"Reference bean"*= Reference du Bean désignée dans l'attribut id
    - *"Nom_propriété"*=Nom de la propriété
  - En utilisant l'éléments de scripts JSP : expression

```
<%= Nom_instance.getNomProriete() %>
```

    - Nom_instance=Référence du Bean désignée dans l'attribut id
    - getNomPropriete= Méthode liée à la propriété

108

Java Beans et JSP: modification des propriétés

109

- Modification de la valeur d'une propriété en utilisant le tag action <jsp:setProperty>

```
<jsp:setProperty property="nom_propriété"
name="Reference_Bean" param="param" value="valeur"/>
```

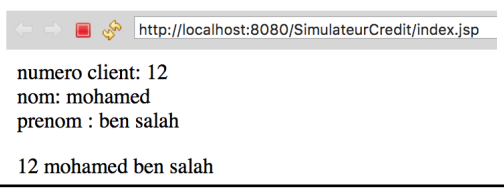
  - Reference_Bean: Référence du Bean désigné dans l'attribut id
  - nom_propriété: Nom de la propriété à modifier écrit en minuscule
  - param: Nom d'un paramètre de la requête contenant la valeur pour la propriété indiquée.
  - valeur: Valeur explicite à donner à la propriété. Ne peut pas être combinée à l'attribut param.

109

Java Beans et JSP: Exemple

110

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<jsp:useBean id="Client" class="model.Client" scope="request"/>
<jsp:setProperty property="numCl" name="Client" value="12"/>
<jsp:setProperty property="nom" name="Client" value="mohamed"/>
<jsp:setProperty property="prenom" name="Client" value="ben salah"/>
<html>
<head>
<title>Exemple Java beans & JSP</title>
</head>
<body>
numero client: <jsp:getProperty property="numCl" name="Client"/><br>
nom: <jsp:getProperty property="nom" name="Client"/><br>
prenom : <jsp:getProperty property="prenom" name="Client"/>
<p>
<%= Client.getNumCl() %>
<%= Client.getNom() %>
<%= Client.getPrenom() %>
</body>
</html>
```



110

111 MVC et JEE

111

Application web suivant le modèle MVC

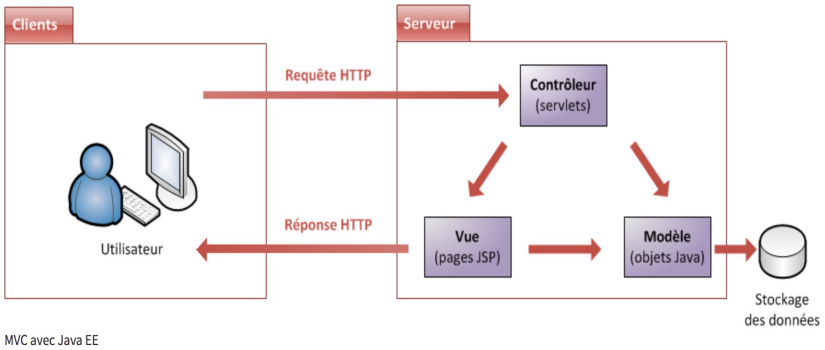
112

- Le modèle de conception MVC impose une répartition stricte des tâches au sein d'une application :
  - la couche **Modèle** se charge des traitements à effectuer sur les données et de leur stockage ;
  - la couche **Vue** se charge de la présentation des données pour l'utilisateur et de l'interaction ;
  - la couche **Contrôle** se charge d'aiguiller les requêtes entrantes vers les traitements et vues correspondants.
- Dans une application JEE :
  - la couche **Modèle** est constituée d'objets Java ;
  - la couche **Vue** est constituée de pages JSP ;
  - la couche **Contrôle** est constituée de servlets.

112

Application web suivant le modèle MVC

113



MVC avec Java EE

113

114

Chapitre 5

JSTL

JSP STANDARD TAG LIBRARY

114

JSTL : JSP Standard Tag Library

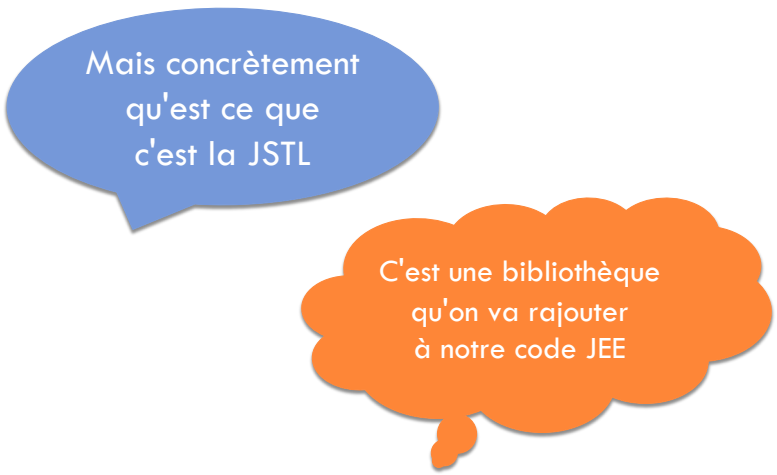
115

- Depuis la version 2.0 des JSP, il est possible de placer à n'importe quel endroit d'une page JSP des expressions qui sont évaluées et remplacées par le résultat de leur évaluation : les Expressions Languages (EL) .
- Elles permettent de manipuler les données au sein d'une page JSP plus simplement qu'avec les scriptlets Java (<% … %>).
- Utilisées conjointement avec des librairies de tags (telles que les JSTL que nous verront plus tard), elles permettent de se passer totalement des scriptlets.
- Une EL permet également d'accéder simplement aux beans des différents scopes de l'application web (page, request, session et application).

115

JSTL: c'est…

116



116

JSTL: c'est…

117

- Constituée de cinq sous bibliothèques :



  - Gère les variables, les conditions, les boucles, …
  - Formate les données et faire l'internationalisation de mon site
  - Manipule des données XML
  - Permet d'écrire des requêtes SQL
  - Traiter des chaines de caractères

117

JSTL : JSP Standard Tag Library

118

- Syntaxe: ${une expression}

EL

- La syntaxe EL comporte plusieurs types de constantes littérales qui peuvent être employées dans les expressions. Il supporte aussi un ensemble des operateurs.
  - Constantes littérales
    - Les valeur booléennes
    - Les chaines de caractères
    - Les valeurs entières
    - Les valeurs réelles

      ${true}, ${2*4} = > 8, ${(2*4)+3} = >11, ${"bonjour"}
  - Les opérateurs
    - Les operateurs arithmétiques +, -, *,
    - Les operateurs logiques &&, | |, !
    - Les operateurs relationnels == et eq, ! = et ne, < et lt, > et gt, <= et le, >= et ge

118

JSTL : JSP Standard Tag Library

119

EL et Java Beans

- En réalité la technologie EL est basée sur les spécifications des JavaBeans.
- Il est possible, via une expression EL, d'accéder directement à une propriété d'un bean.

El et les collections

- Les EL permettent de manipuler les collections java tel que java.util.List, java.util.Set, java.util.Map, etc.

119

JSTL : EL et les collections

120

Exemple

```
<%@ page import="java.util.*" %>
<% List<String> lv = new ArrayList<String>();
lv.add("Java");
lv.add("UML");
lv.add("Anglais");
request.setAttribute("lv", lv);
%>
```

```
<!-- accès avec EL -->

${lv[0]}
${lv['0'] }
${lv["0"] }
```

120

JSTL : EL et la bibliothèque Core

121

- Elle propose les tags suivants répartis comme suit :
  - Actions généralistes: out, set, remove, catch
  - Structures conditionnelles: if, choose, when, otherwise
  - Structures itératives: forEach, forTokens
  - gestion d'url: import, url, redirect
  - Dans chaque JSP qui utilise un ou plusieurs tags de la bibliothèque, il faut la déclarer avec une directive taglib :

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

  - Cette ligne permet d'inclure la librairie "core" de la JSTL qui contient les actions de base d'une application web.

121

JSTL : EL et la bibliothèque Core

122

- La directive taglib permet d'utiliser des balises personnalisées dans le fichier JSP

```
<%@ taglib uri="maTagLib.tld" prefix="tagExemple" %>
```

  - Le préfixe désigne un espace de nommage (namespace) qui permet d'identifier une balise comme appartenant à cette bibliothèque.
  - inclure une bib personnalisée nommée maTagLib. TLD (Tag Librairy Descriptor), il s'agit d'un fichier xml spécifiant l'ensemble des attributs de chaque balise.

122

JSTL : EL et la bibliothèque Core

123

- Actions généralistes
  - c:set Modifier la valeur d'une variable El ou une propriété d'un objet

```
<c:set var="cpt" value="${cpt+1}" scope="request" />
```

  - c:out
    - Afficher une expression

```
<c:out value="${cpt}"></c:out>
<c:out value="Bonjour"></c:out>
```

    - Afficher la propriété nom du bean Client

```
<c:out value="${Client.nom}"></c:out>
```

123

JSTL : EL et la bibliothèque Core

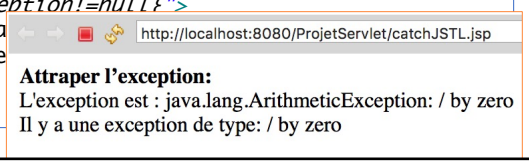
124

- c:remove Sert à supprimer une variable

```
<c:remove var="cpt"/>
```

- c:catch Intercepter les exceptions qui sont levées lors de l'exécution du code inclus dans son corps.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html><head>
<title>Gestion des Exception avec JSTL</title></head>
<body> <strong>Attraper l'exception:</strong> <br>
        <c:catch var="catchException">
                <%int x=5/0; %>
        </c:catch>
        <c:if test="${catchException!=null}">
        L'exception est : ${ ca
        Il y a une exception de
        </c:if>
</body></html>
```



124

JSTL : EL et la bibliothèque Core

125

- Structures conditionnelles
  - c:if

```
<c:if test="${Client.numCl<='12'}">ok</c:if>
```

  - c:choose garantie une seule de ses actions sera exécutée
  - c:when permet de traiter différents cas mutuellement exclusifs. Ne possède pas d'attribut mais possède un ou plusieurs tags when
  - c:otherwise ne peut être que la dernière action dans une action c:choose

125

JSTL : EL et la bibliothèque Core

126

- Exemple c:choose

```
<c:choose>
<c:when test="${Client.nom=='mohamed'}">Bonjour mohamed</c:when>
<c:when test="${Client.nom=='toto'}">Bonjour toto</c:when>
<c:when test="${Client.nom=='sami'}">Bonjour sami</c:when>
<c:otherwise>Bonjour</c:otherwise>
</c:choose>
```

- Structures itératives
  - c:forEach exécute le contenu pour chacune des valeurs d'un ensemble

```
<c:forEach var="i" begin="1" end="10" step ="2"> ${i}
</c:forEach>
```

  - Afficher le contenu d'une List de String nomme lv

```
<c:forEach var="v" items="${lv}"> ${v}<br> </c:forEach>
```

126

JSTL : EL et la bibliothèque Core

127

- c:forTokens semblable à l'action c:forEach utilisée avec une chaine de caractères mais on peut choisir plusieurs délimiteurs au lieu de la simple virgule grâce à l'attribut delims.
  - Ex : Décomposer str et afficher chaque sous-chaîne sur une ligne différente.

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

```
<% String str="mot1;mot2;mot3;mot4";
request.setAttribute("str", str); %>
<c:forTokens var="mot" delims=";" items="${str}">
${mot} <br/> </c:forTokens>
```

127

128

Partie II

JPA

128