



Chapitre 2 : Les Servlets

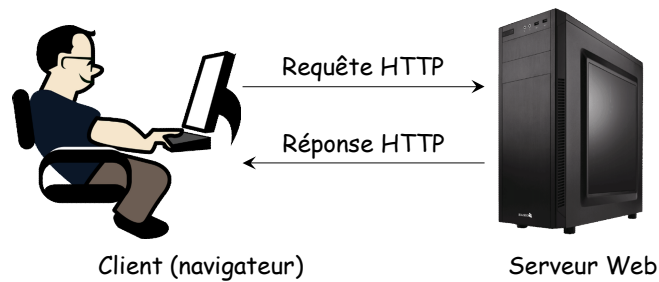


Introduction aux servlets

- Ce sont des classes Java fonctionnant du côté serveur.
- Elles permettent de construire des pages Web dynamiques.
- Une Servlet:
 - Reçoit des requêtes HTTP d'un client Web
 - Effectue traitement
 - Fournit une réponse HTTP dynamique au client Web
- La compréhension du fonctionnement du protocole HTTP est nécessaire

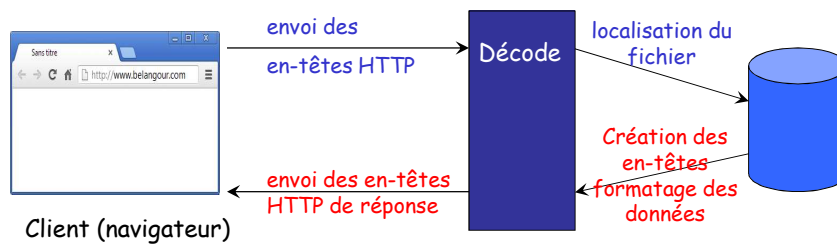
Servlets et protocole HTTP

- Les servlets s'appuient sur le protocole HTTP pour communiquer avec le client.
- HTTP agit comme un protocole de Transport



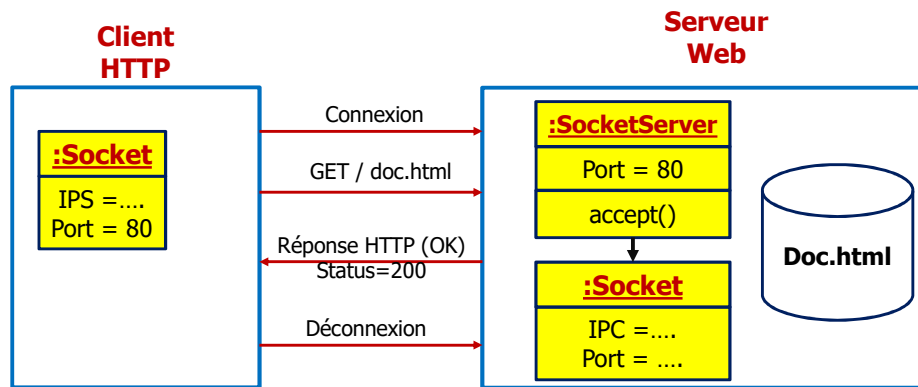
Le protocole HTTP

- HTTP = HyperText Tranfert Protocol
- Protocole qui permet au client de récupérer des documents sur serveur :
 - Documents Statiques (HTML, PDF, Image, etc..) ou
 - Documents Dynamiques (PHP, JSP, ASP...)
- HTTP permet également de soumissionner les formulaires





Le protocole HTTP : Fonctionnement

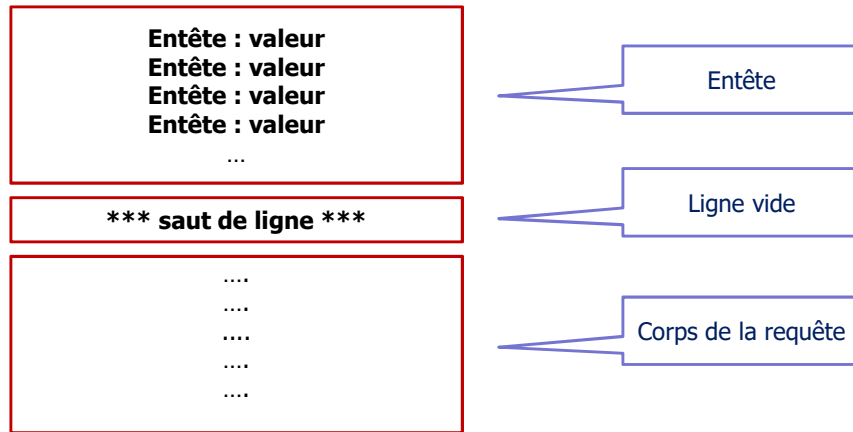


Le protocole HTTP : Méthodes

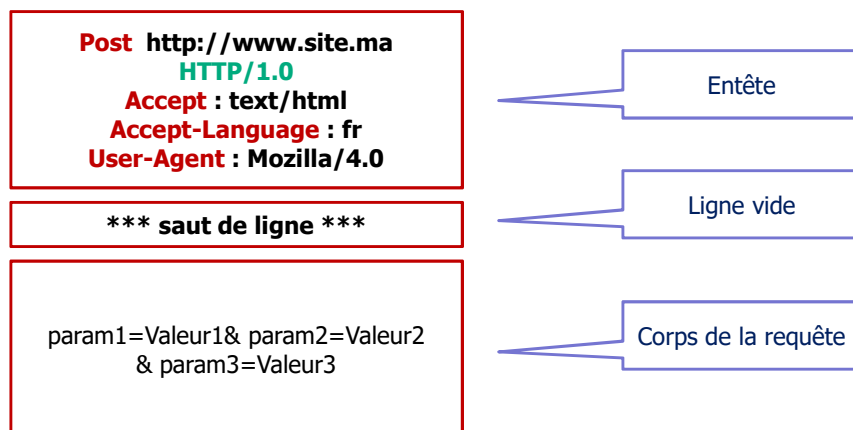
- Une requête HTTP peut être envoyée en utilisant les méthodes suivantes:
 - **GET** : Pour récupérer le contenu d'un document
 - **POST** : Pour soumissionner des formulaires (Envoyer, dans la requête, des données saisies par l'utilisateur)
 - **PUT** pour envoyer un fichier du client vers le serveur
 - **DELETE** permet de demander au serveur de supprimer un document
 - **HEAD** permet de récupérer les informations sur un document (Type, Capacité, Date de dernière modification etc...)



Le protocole HTTP : Forme d'une requête HTTP



Le protocole HTTP : Requête de type POST





Le protocole HTTP : Requête de type GET

GET http://www.site.ma **HTTP/1.0**
Accept : text/html
Accept-Language : fr
User-Agent : Mozilla/4.0

*** saut de ligne ***

Entête

Ligne vide

Corps de la requête
(VIDE)



Le protocole HTTP : Exemple

Requête

GET Nom_Script?login=val1&pass=val2&.... **HTTP/1.0**
Accept : text/html
Accept-Language : fr
User-Agent : Mozilla/4.0

*** saut de ligne ***

Réponse

HTTP/1.0 200 OK
Date : Wed, 15 Sep 2017 15:02:01 GMT
Server : Apache/1.3.24
Last-Modified : Wed 02 Oct 2016 22:05:01 GMT
Content-Type : Text/html
Content-length : 4205

*** saut de ligne ***

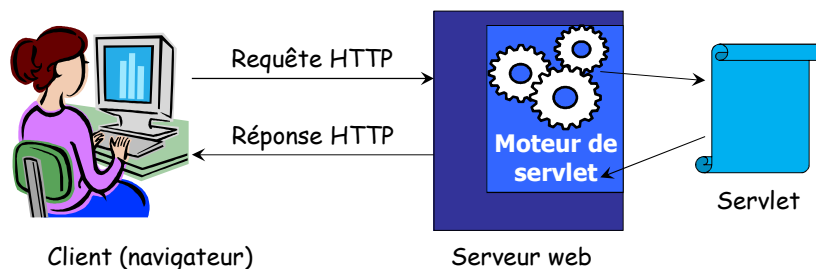
<!DOCTYPE html>
<HTML>
<HEAD>.... </HEAD>
<BODY> </BODY>
</HTML>

Le protocole HTTP et Types MIME

- MIME veut dire Multipurpose Internet Mail Extensions
- Standard utilisé entre autres pour typer les documents transférés par le protocole HTTP
- Un type MIME est constitué comme suit : **Content-type:** TYPE/SOUS-TYPE
 - Exemples :
 - **Content-type:** image/gif (Images gif)
 - **Content-type:** image/jpeg (Images Jpeg)
 - **Content-type:** text/html (Fichiers HTML)
 - **Content-type:** text/plain (Fichiers texte sans mise en forme)

Moteur de servlets

- Une Servlet ne tourne pas directement sur un serveur Web : Elle a besoin de tourner sur un Moteur de Servlet
- Est connu aussi par conteneur de servlets (en anglais Servlet Container)
- Il permet d'établir le lien entre la Servlet et le serveur Web





Moteur de servlets

- Un moteur de Servlets prend en charge et gère les servlets:
 - chargement de la servlet
 - gestion de son cycle de vie
 - passage des requêtes et des réponses
- Nombreux conteneurs de Servlet (Tomcat, JBoss, WebSphere Application Server, WebLogic,...)
- Dans le reste du cours et des TP, nous utiliserons le conteneur Tomcat pour déployer nos servlets.



Jakarta Tomcat

- Tomcat est une Implémentation de référence de la spécification JEE
- Fournit donc une implémentation de l'API JEE (dossier lib)
- Disponible gratuitement sous forme d'une licence Open Source
- Écrit entièrement en Java et nécessite obligatoirement une machine virtuelle (JRE ou JDK).





Jakarta Tomcat

- Disponible pour téléchargement à l'adresse suivante :
<https://tomcat.apache.org/>
- Existe en
 - Version zip (non installable)
 - version installable (.exe)



Jakarta Tomcat : version zip

- Téléchargement du fichier compressé
- Décompression dans un endroit spécifique
- Ajout des variables d'environnement suivantes :
 - **CATALINA_HOME** : dossier de décompression de Tomcat
 - **JAVA_HOME** (ou **JRE_HOME**) : dossier d'installation du JDK (ou dossier du JRE)



Jakarta Tomcat : version zip

- Les script de demarrage de tomcat sont logés dans le dossier « bin » de tomcat :
 - « Startup.bat » pour le demarrage
 - « Shutdown.bat » pour l'arrêt
- Remarque :
 - L'avantage de la version zip est que vous pouvez déplacer Tomcat facilement
 - Il suffit de modifier le chemin dans la variable d'environnement



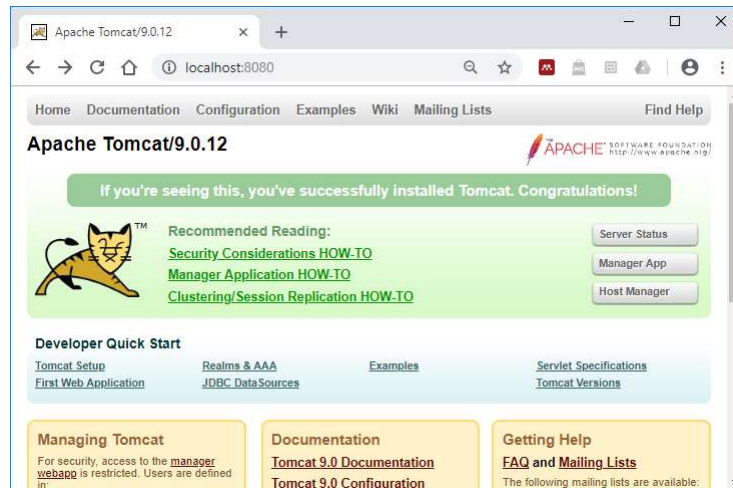
Jakarta Tomcat : version installable

- Après installation de Tomcat :
 - L'icône suivante montre quand il marche
 - L'icône suivante montre quand il est arrêté
 - « Start service » permet de le faire marcher s'il est en arrêt
 - « stop service » permet de l'arrêter s'il est en marche



Jakarta Tomcat

- Page d'accueil de Tomcat : ***http://localhost:8080***

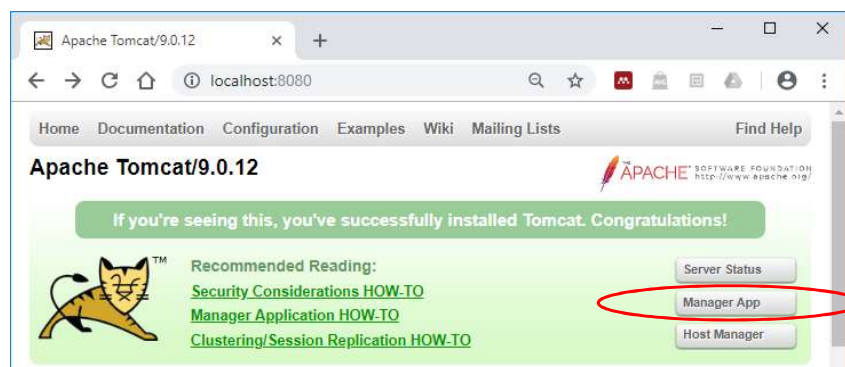


cours JEE - Dr. Abdessamad Belangour

34

Tomcat Manager

- Les applications hébergées sur Tomcat peuvent être gérées (démarrées, arrêtées, rechargées) grâce au **Tomcat Manager**.



22/11/2019

cours JEE - Dr. Abdessamad Belangour

35

35

Tomcat Manager

- Tomcat manager requière un login et un mot de passe.
- Il fournit deux modes de gestion de Tomcat :
 - Une gestion à partir de son interface graphique représentée par le rôle **manager-gui**
 - Une gestion à travers un script (réservées aux IDE comme Eclipse et Netbeans) représentée par le rôle **manager-script**
- Ces deux modes de gestion peuvent être édités à travers le fichier « tomcat-users.xml » dans le dossier « conf » de Tomcat
- Exemple (pour la version 9):


```
<user username="adminGui" password="adminGui" roles="manager-gui"/>
<user username="adminScript" password="adminScript" roles="manager-script"/>
```

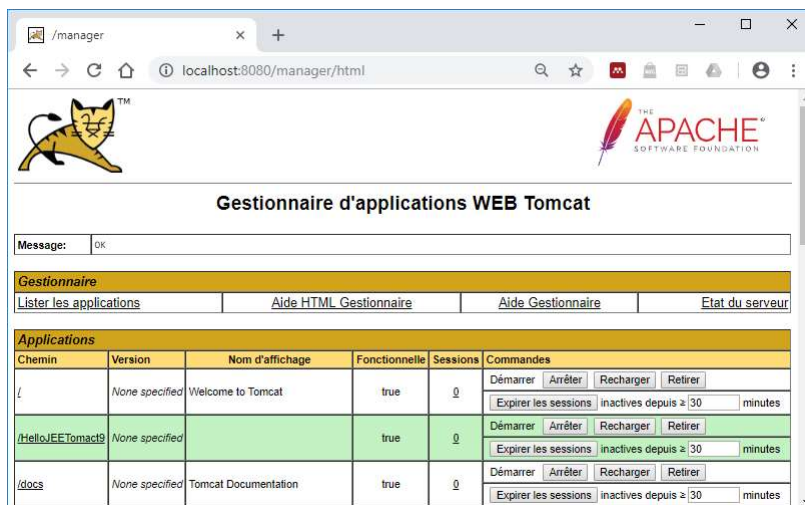
22/11/2019

cours JEE sur IDEs et bases de données Belangour

36

36

Tomcat Manager



The screenshot shows the Tomcat Manager web interface in a browser window. The address bar shows 'localhost:8080/manager/html'. The page title is 'Gestionnaire d'applications WEB Tomcat'. It features the Tomcat logo and the Apache Software Foundation logo. Below the title, there is a 'Message' field showing 'OK'. A navigation bar includes links: 'Lister les applications', 'Aide HTML Gestionnaire', 'Aide Gestionnaire', and 'Etat du serveur'. The main content area is titled 'Applications' and contains a table with columns: 'Chemin', 'Version', 'Nom d'affichage', 'Fonctionnelle', 'Sessions', and 'Commandes'. The table lists three applications: '/', '/HelloJEETomcat9', and '/docs'. Each application row has buttons for 'Démarrer', 'Arrêter', 'Recharger', and 'Retirer', along with a link to 'Expire les sessions' and a timeout setting of 30 minutes.

Chemin	Version	Nom d'affichage	Fonctionnelle	Sessions	Commandes
/	None specified	Welcome to Tomcat	true	0	Démarrer Arrêter Recharger Retirer Expire les sessions inactives depuis ≥ 30 minutes
/HelloJEETomcat9	None specified		true	0	Démarrer Arrêter Recharger Retirer Expire les sessions inactives depuis ≥ 30 minutes
/docs	None specified	Tomcat Documentation	true	0	Démarrer Arrêter Recharger Retirer Expire les sessions inactives depuis ≥ 30 minutes

22/11/2019

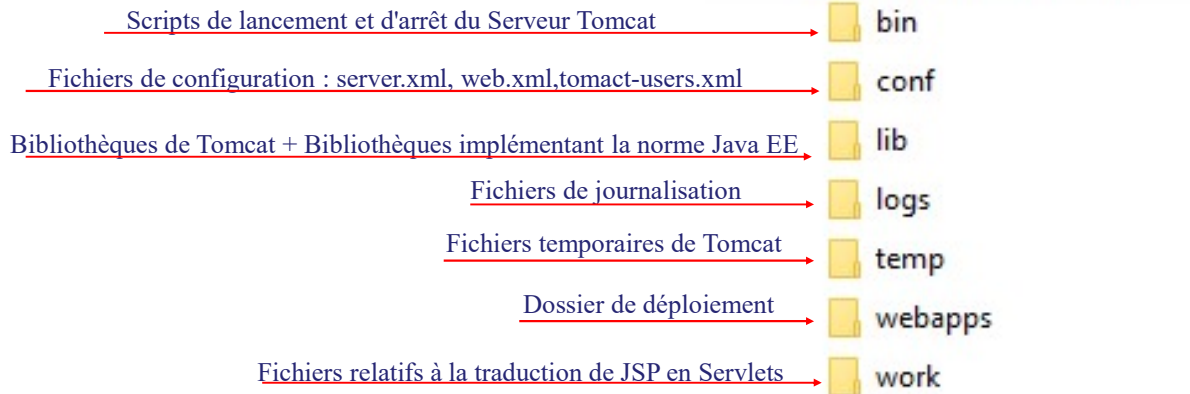
cours JEE sur IDEs et bases de données Belangour

37

37

Hiérarchie des dossiers Tomcat

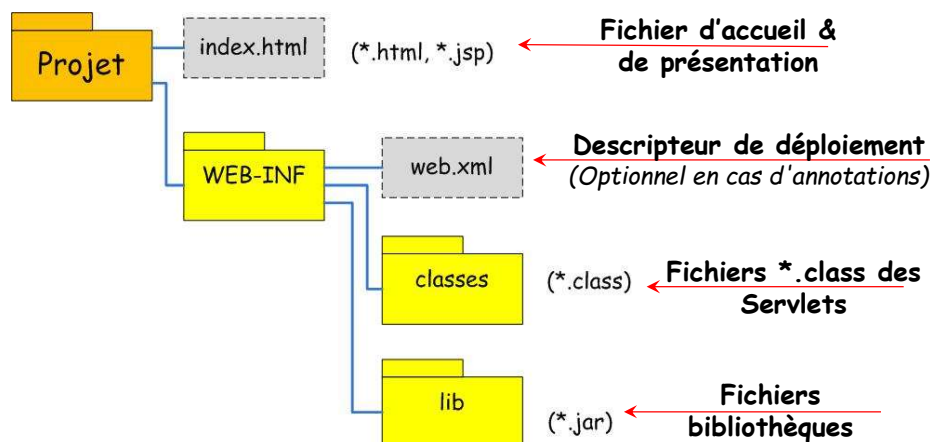
- Organisation partielle des dossiers de Tomcat



Déploiement d'une Servlet dans Tomcat

Optionnel en cas d'annotations

- Une application Web doit être déployée dans le dossier **webapps** et avoir la structure suivante:



Une première Servlet

- Une Servlet est :
 - Classe Java
 - Utilise des bibliothèques JEE
 - Qui écrit du HTML
 - Qui a besoin d'une certaine configuration pour tourner sur le Web côté serveur
- Nous allons écrire une Servlet qui écrit Hello World en HTML

Une première Servlet

```
package com.exemple;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class HelloWorldServlet extends HttpServlet {
    // la méthode doGet traite les requêtes envoyées avec GET
    @Override
    public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {
        res.setContentType("text/html"); // précision du type MIME de contenu à renvoyer au client
        PrintWriter out = res.getWriter(); // objet responsable d'envoyer du texte au client
        // rédaction du code HTML à renvoyer au client
        out.println("<!DOCTYPE html> ");
        out.println("<HTML><HEAD><TITLE> Titre </TITLE></HEAD>");
        out.println("<BODY> Hello World </BODY></HTML>");
        out.close(); //fermeture de l'objet PrintWriter }
    }
}
```

Packages pour la création de servlets

Objet requête

Objet réponse



Une première Servlet

■ Remarques :

- La méthode **doGet** (resp. **doPost**) traite les requêtes envoyées avec **GET** (resp. **POST**)
- La méthode **doGet** (resp. **doPost**) prend deux paramètres :
 - Un paramètre de type **HttpServletRequest** représentant la requête client
 - Un paramètre de type **HttpServletResponse** représentant la réponse à renvoyer au client
- L'objet **HttpServletRequest** permet d'extraire toutes les informations sur le client (adresse IP, navigateur, Domaine, paramètres d'un formulaire, etc..)
- L'objet **HttpServletResponse** doit être complété d'informations par la servlet avant de le renvoyer au client.



Une première Servlet

■ Remarque :

- Pour compiler une Servlet sous DOS il faudra ajouter la bibliothèque « **servlet-api.jar** » à la variable d'environnement classpath.
- Cette bibliothèque se trouve dans le dossier « **lib** » de Tomcat
- Elle se compose des packages : « javax.servlet » et « javax.servlet.http »



Une première Servlet : fichier web.xml

- Le fichier « web.xml » est le fichier de configuration de l'application Web qu'on est en cours de construire.
- Il permet de donner des informations de l'application à Tomcat comme :
 - Les noms des classes des Servlets
 - Le nom d'affichage de l'application
 - Le chemin virtuel pour accéder aux différents servlets
 - Les fichiers d'accueils
 - Etc..



Une première Servlet : fichier web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
    http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">
  <display-name>Ma première application Web</display-name>
  <servlet>
    <servlet-name>Hello</servlet-name>
    <servlet-class>com.exemple.HelloWorldServlet</servlet-
class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Hello</servlet-name>
    <url-pattern>/salut</url-pattern>
  </servlet-mapping>
</web-app>
```



Une première Servlet : fichier index.html

- Le fichier d'accueil par défaut des projets JEE est « index.html » ou « index.jsp »

```
<!DOCTYPE html>
<html>
  <head>
    <title>index</title>
  </head>
  <body>
    <a href="./salut">Cliquez ici pour lancer la Servlet</a>
  </body>
</html>
```



Une première Servlet : chemin d'accès

- Si le dossier contenant notre Servlet se nomme projet , le chemin d'accès à notre projet sera :
 - `http://localhost:8080/projet`
- Equivalent à :
 - `http://localhost:8080/projet/index.html`
- Le chemin d'accès à la servlet est :
 - `http://localhost:8080/projet/salut`



Une première Servlet : avec annotations

- Depuis la norme Servlets 3.0 il est possible de remplacer le fichier « web.xml » par des annotations.
- Ainsi la plupart des informations contenues dans un fichier « web.xml » peut être indiquée par des annotations et leurs attributs.
- Le projet garde exactement la même structure
- Le seul changement est l'absence du fichier web.xml



Une première Servlet : avec annotations

```
package com.exemple;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
@WebServlet("/salut")
public class HelloWorldServlet extends HttpServlet {
    @Override
    public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException,
        IOException {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        out.println("<!DOCTYPE html> ");
        out.println("<HTML><HEAD><TITLE> Titre </TITLE></HEAD>");
        out.println("<BODY> Hello World </BODY></HTML>");
        out.close();
    }
}
```



Une première Servlet : avec annotations

- **Remarque** : Les annotations remplacent tout le paramétrage fait avec le fichier web.xml sauf les fichiers de bienvenue et les pages d'erreurs.

```
<web-app>
  <welcome-file-list>
    <welcome-file>accueil.html</welcome-file>
    <welcome-file>sommaire.html</welcome-file>
  </welcome-file-list>
  <error-page>
    <error-code>404</error-code>
    <location>/erreur404.html</location>
  </error-page>
  <error-page>
    <error-code>500</error-code>
    <location>/erreur500.html</location>
  </error-page>
</web-app>
```



L'API Servlet

- L'API Servlet fournit un ensemble de classes et d'interfaces pour la manipulation des servlets
- Cet API est fourni sous forme d'un kit appelé JSDK (Java Servlet Development Kit)
- L'API servlet regroupe un ensemble de classes dans deux 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 utilisé par les serveurs web.



Le package javax.servlet

- Le package javax.servlet définit plusieurs interfaces, méthodes et exceptions :
 - Les interfaces :
 - **ServletConfig** : Définit d'un objet utilisé par le conteneur de la servlet pour passer de l'information à une servlet pendant son initialisation.
 - **ServletContext** : Définit un ensemble de méthodes qu'une servlet utilise pour communiquer avec le conteneur de servlets. un objet ServletContext est contenu dans un objet ServletConfig.
 - **Servlet** : interface de base d'une servlet
 - **RequestDispatcher** : définit un objet qui reçoit les requêtes du client et les envoie à n'importe quelle ressource (par exemple servlet, fichiers HTML ou JSP) sur le serveur.
 - **ServletRequest** : Définit un objet contenant la requête du client.



Le package javax.servlet

- **ServletResponse** : Définit un objet qui contient la réponse renvoyée par la servlet
- **SingleThreadModel** : Permet de définir une servlet qui ne répondra qu'à une seule requête à la fois
- Les classes :
 - **GenericServlet** : Classe définissant une servlet indépendante de tout protocoles
 - **ServletInputStream** : permet la lecture des données de la requête cliente
 - **ServletOutputStream** : permet l'envoi de la réponse de la servlet
- Les exceptions :
 - **ServletException** : Exception générale en cas de problème durant l'exécution de la servlet
 - **UnavailableException** : Exception levée si la servlet n'est pas disponible



Le package javax.servlet.http

- Le package **javax.servlet.http** définit plusieurs interfaces et méthodes :
 - Les interfaces :
 - **HttpServletRequest** : Hérite de ServletRequest : définit un objet contenant une requête selon le protocole http
 - **HttpServletResponse** : Hérite de ServletResponse : définit un objet contenant la réponse de la servlet selon le protocole http
 - **HttpSession** : Définit un objet qui représente une session



Le package javax.servlet.http

- Les classes :
 - **Cookie** : Classe représentant un cookie (ensemble de données sauvegardées par le browser sur le poste client)
 - **HttpServlet** : Hérite de GenericServlet : classe définissant une servlet utilisant le protocole http
 - **HttpUtils** : Classe proposant des méthodes statiques utiles pour le développement de servlet http (classe devenue obsolète)



Notion de Contexte

- Une application Web peut être composée de plusieurs types de fichiers (Servlets, JSP, pages html, images, sons, etc.),
- L'ensemble des constituants d'une application est appelé **Contexte** de l'application.
- Les servlets et les JSP d'une application partagent le même contexte.
- Un contexte offre à chaque Servlet (ou JSP) d'une même application une vue sur le fonctionnement de cette application.



Notion de Contexte

- La description du contexte peut être faite avec :
 - Pour l'ensemble de l'application avec le fichier « web.xml » ou
 - Directement sur chacune des servlets avec les annotations.
- Le contexte d'une application est représenté par un objet appelé ServletContext
- Grâce à cet objet, il est possible d'accéder à chacune des ressources de l'application Web correspondant au contexte.



Notion de Contexte

- Quelques méthodes de l'objet `ServletContext` :
 - **`String getInitParameter(String name)`** : récupère le paramètre d'initialisation de la servlet, passé en paramètre.
 - **`String getServerInfo()`** : retourne le nom et la version du conteneur de servlet sur lequel la servlet tourne
 - **`int getSessionTimeout()`** : retourne le temps d'expiration de la session en secondes.
 - **`void log(String msg)`** : permet d'écrire dans le fichier journal du conteneur de servlet



L'interface d'une servlet

- Toute Servlet doit implémenter l'interface *Servlet* du package *javax.servlet*
- Le cycle de vie d'une servlet est géré à travers cette interface
- `HttpServlet` qui implémente l'interface `Servlet`
- Nos servlets implémentent indirectement l'interface `Servlet` puisqu'elles héritent de `HttpServlet`



L'interface d'une servlet

- L'interface d'une servlet se compose des méthodes suivantes :
 - la méthode *init()*
 - la méthode *service()*
 - la méthode *getServletConfig()*
 - la méthode *getServletInfo()*
 - la méthode *destroy()*
- Nous pouvons redéfinir l'une des méthodes au besoin

« interface » Servlet
+ init() + service() + getServletConfig() + getServletInfo() + destroy()

22/11/2019

cours JEE - Dr. Abdessamad Belangour

60




La méthode *init()*

- Signature :
 - `public void init(ServletConfig config) throws ServletException`
- Fonctionnement :
 - Est appelée par le conteneur à chaque instantiation de la servlet
 - Lors de l'instanciation, le conteneur de servlet passe en argument à la méthode *init()* un objet *ServletConfig* permettant de charger des paramètres de configuration propres à la servlet.
 - En cas d'anomalie lors de l'appel de la méthode *init()*, celle-ci renvoie une exception de type *ServletException* et la servlet n'est pas initialisée.

cours JEE - Dr. Abdessamad Belangour

61



La méthode init()

- Exemple :

- Écrire une Servlet qui compte le nombre d'utilisation d'une Servlet depuis son chargement.

- Solution :

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

public class ServletCompteur extends HttpServlet {
    private int compteur;
    @Override
    public void init() throws ServletException {
        compteur = 0;
    }
}
```

cours JEE - Dr. Abdessamad Belangour

62



La méthode init()

@Override

```
protected void doGet(HttpServletRequest req, HttpServletResponse res) throws
ServletException, IOException {
    res.setContentType("text/plain");
    PrintWriter out = res.getWriter();
    compteur++;
    out.println("Depuis son chargement, on a accédé à cette Servlet " +compteur+"
fois.");
}
}
```

cours JEE - Dr. Abdessamad Belangour

63



La méthode init()

- Il est possible de définir des paramètres d'initialisations pour la servlet (et de les utiliser dans la méthode init) soit avec l'annotation *@WebInitParam* (comme on verra plus loin) soit au niveau du fichier web.xml.

- Exemple :

```
<servlet>
  <servlet-name>Cmp</servlet-name>
  <servlet-class>Compteur2</servlet-class>
  <init-param>
    <param-name>compteur_initial</param-name>
    <param-value>50</param-value>
    <description>Valeur init du compteur</description>
  </init-param>
</servlet>
```

cours JEE - Dr. Abdessamad Belangour

64



La méthode init()

```
public class ServletCompteur2 extends HttpServlet {
    private int compteur;
    @Override
    public void init() throws ServletException {
        String initial = this.getInitParameter("compteur_initial");
        try { compteur = Integer.parseInt(initial);
        }
        catch (NumberFormatException e) {
            compteur = 0;
        }
    }
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse res) throws
        ServletException, IOException {
        //même traitement que l'exemple dernier...
    }
}
```

cours JEE - Dr. Abdessamad Belangour

65



La méthode service()

- Signature :
 - `public void service(ServletRequest req, ServletResponse res) throws ServletException, IOException`
- Fonctionnement :
 - Cette méthode est exécutée par le conteneur lorsque la Servlet est sollicitée.
 - Elle détermine le type de requête dont il s'agit, puis transmet la requête et la réponse à la méthode adéquate (*doGet()* ou *doPost()*).
 - Chaque requête du client déclenche une seule exécution de cette méthode.



La méthode getServletConfig()

- Signature :
 - `public ServletConfig getServletConfig()`
- Fonctionnement :
 - Renvoie un objet `ServletConfig` qui constitue un intermédiaire permettant d'accéder au contexte d'une application.
 - On peut aussi utiliser `ServletConfig` pour récupérer un paramètre du fichier `web.xml` :
 - Exemple :

```
String param;  
public void init(ServletConfig config) {  
    param = config.getInitParameter("param");  
}
```



La méthode `getServletInfo()`

- Signature:

- `public String getServletInfo()`

- Fonctionnement :

- Lorsqu'elle est surchargée permet de retourner des informations sur la servlet comme l'auteur, la version, et le copyright.
- Ces informations peuvent être exploitées pour affichage par des outils dans les conteneurs Web.
- Exemple :

```
public String getServletInfo() { return " servlet écrite par x (x@y.com)" ; }
```



La méthode `destroy()`

- Signature :

- `void destroy()`

- Fonctionnement :

- La méthode `destroy()` est appelée par le conteneur lors de l'arrêt du serveur Web.
- Elle permet de libérer proprement certaines ressources (fichiers, bases de données ...).
- C'est le serveur qui appelle cette méthode.



Le cycle de vie d'une servlet

- Le cycle de vie d'une Servlet est assuré par le conteneur de servlet (grâce à l'interface Servlet).
- Cette interface permet à la servlet de suivre le cycle de vie suivant :
 1. Le serveur crée un pool de threads auxquels il va pouvoir affecter chaque requête
 2. La Servlet est chargée au démarrage du serveur ou lors de la première requête
 3. La Servlet est instanciée par le serveur
 4. La méthode *init()* est invoquée par le conteneur
 5. Lors de la première requête, le conteneur crée les objets Request et Response spécifiques à la requête



Le cycle de vie d'une servlet

6. La méthode ***service()*** est appelée à chaque requête dans une nouvelle thread. Les objets *Request* et *Response* lui sont passés en paramètre
7. Grâce à l'objet ***Request***, la méthode *service()* va pouvoir analyser les informations en provenance du client
8. Grâce à l'objet ***Response***, la méthode *service()* va fournir une réponse au client
9. La méthode ***destroy()*** est appelée lors du déchargement de la Servlet, c'est-à-dire lorsqu'elle n'est plus requise par le serveur. La Servlet est alors signalée au *garbage collector*



Filtres de Servlets : Définition

- Les filtres de servlet sont des classes Java qui peuvent être utilisées pour :
 - Intercepter les requêtes d'un client avant qu'il n'accède à une ressource en back-end.
 - Manipuler les réponses du serveur avant qu'elles ne soient renvoyées au client.
- La spécification JEE propose plusieurs types de filtres comme :
 - Filtres d'authentification.
 - Filtres de chiffrement.
 - Filtres qui déclenchent des événements d'accès aux ressources.
 - Filtres de journalisation et d'audit...
- Plusieurs filtres peuvent être appliqués de suite : ils sont appelés chaîne de filtres ou Filter Chain.

cours JEE - Dr. Abdessamad Belangour

72



Filtres de Servlets : Interface Filter

- Les filtres de servlet doivent implémenter l'interface **Filter** qui se compose des trois méthodes suivantes:
 - **public void doFilter (ServletRequest, ServletResponse, FilterChain)** : est appelée par le conteneur chaque fois qu'une paire requête / réponse est transmise à travers la chaîne de filtrage avant qu'une requête client n'accède à une ressource située à la fin de la chaîne.
 - **public void init (FilterConfig filterConfig)** : Cette méthode est appelée par le conteneur Web pour indiquer à un filtre qu'il est mis en service.
 - **public void destroy ()** : Cette méthode est appelée par le conteneur Web pour indiquer à un filtre qu'il est mis hors service.

cours JEE - Dr. Abdessamad Belangour

73



Filtres de Servlets : Définition dans le web.xml

- Au niveau du fichier web.xml un filtre est défini comme suit :

```
<filter>
    <filter-name>nomFiltre</filter-name>
    <filter-class>classeFiltre</filter-class>
</filter>
<filter-mapping>
    <filter-name>nomFiltre</filter-name>
    <url-pattern>cheminFiltré</url-pattern>
</filter-mapping>
```



Filtres de Servlets : Exemple

- Exemple : filtre de servlet permettant d'imprimer l'adresse IP du client et l'heure de la date actuelle.

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

public class MonFiltre implements Filter {
    public void init(FilterConfig config) throws ServletException {
        String testParam = config.getInitParameter("testParam");
        System.out.println("Test Param: " + testParam);
    }
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws
        java.io.IOException, ServletException {
        String ipAddress = request.getRemoteAddr();
        System.out.println("IP " + ipAddress + ", Time " + new Date().toString());
        chain.doFilter(request, response); // Repasse la requête à la chaîne de filtrage
    }
    public void destroy() {}
}
```



Filtres de Servlets : Exemple (suite)

- Au niveau du « web.xml »

```
<filter>
  <filter-name>MonFiltre</filter-name>
  <filter-class>MonFiltre</filter-class>
  <init-param>
    <param-name> testParam</param-name>
    <param-value>Paramètre d'initialisation</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>MonFiltre</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```



Filtres de Servlets : plusieurs filtres

- Remarque : Dans le cas de plusieurs filtres, l'ordre est défini par l'ordre dans le fichier web.xml
 - Exemple : ici **AuthenFilter** précède **LogFilter**

```
<filter-mapping>
  <filter-name>AuthenFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
<filter-mapping>
  <filter-name>LogFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```



Servlets 3.0

- Depuis la norme Servlets 3.0 il est possible de remplacer le fichier « web.xml » par des annotations.
- Les annotations les plus utilisés (que nous verrons en détail plus loin) sont :
 - **@WebServlet** : permet de déclarer une servlet
 - **@WebInitParam** : permet de déclarer paramètre d'initialisation
 - **@WebFilter** : permet de déclarer un filtre
- Il existe aussi des annotations pour une utilisation plus avancées et sont comme suit :
 - **@WebListener** : permet de déclarer un WebListener
 - **@HandlesTypes** : Pour déclarer les types de classe qu'un ServletContainerInitializer peut gérer.



Servlets 3.0

- **@HttpConstraint** : est utilisée dans l'annotation ServletSecurity pour représenter les contraintes de sécurité à appliquer à toutes les méthodes du protocole HTTP pour lesquelles un élément HttpMethodConstraint correspondant ne se produit pas dans l'annotation ServletSecurity.
- **@HttpMethodConstraint** : est utilisée dans l'annotation ServletSecurity pour représenter des contraintes de sécurité sur des messages de protocole HTTP spécifiques.
- **@MultipartConfig** : Annotation pouvant être spécifiée sur une classe Servlet, indiquant que ses instances attendent des demandes conformes au type MIME multipart / form-data.
- **@ServletSecurity** : Cette annotation est utilisée dans une classe d'implémentation Servlet pour spécifier les contraintes de sécurité à appliquer par un conteneur Servlet aux messages de protocole HTTP.



L'annotation @WebServlet

- Sert à déclarer une servlet avec les attributs suivants:
 - String **name** : Nom de la servlet
 - String[] **value** : tableau des URL patterns
 - String[] **urlPatterns** : tableau des URL patterns auxquelles ce filtre s'applique
 - Int **loadOnStartup** : valeur entière de l'indicateur de démarrage
 - WebInitParam[] **initParams** : Tableau de paramètres d'initialisation pour la Servlet
 - Boolean **asyncSupported** : Opération asynchrone prise en charge par la Servlet
 - String **smallIcon** : Petite icône pour cette servlet, si présente
 - String **largeIcon** : Grande icône pour cette servlet, si présente
 - String **description** : Description de cette servlet, si présente
 - String **displayName** : Nom d'affichage de cette servlet, s'il est présent

cours JEE - Dr. Abdessamad Belangour

80



L'annotation @WebServlet

- Exemple :

```
package com.exemple;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
@WebServlet("/salut")
public class HelloWorldServlet extends HttpServlet {
    @Override
    public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException,
        IOException {
        .....
    }
}
```

cours JEE - Dr. Abdessamad Belangour

81



L'annotation @WebInitParam

- Sert à définir un paramètre d'initialisation pour une servlet ou un filtre.
- Attributs :
 - String **name** : nom du paramètre d'initialisation
 - String **value** : valeur du paramètre d'initialisation
 - String **description** : description du paramètre d'initialisation



L'annotation @WebInitParam

- Exemple :

```
....
@WebServlet(value = "/salut", initParams = { @WebInitParam(name = "x", value = "Hello "),
    @WebInitParam(name = "y", value = " World!") })
public class HelloWorldServlet extends HttpServlet {
    @Override
    public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException,
        IOException {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        out.println("<!DOCTYPE html> ");
        out.println("<HTML><HEAD><TITLE> Titre </TITLE></HEAD><BODY>");
        out.println(getInitParameter("x"));
        out.println(getInitParameter("y"));
        out.println("<BODY> Hello World </BODY></HTML>");
        out.close();}
}
```



L'annotation @Webfilter

- Sert à déclarer un filtre de servlet qui est appliqué aux URL patterns, servlets et dispatchers.
- Elle admet les attributs suivants :
 - **String filterName** : Nom du filtre
 - **String [] urlPatterns** : Fournit un tableau de valeurs ou urlPatterns auquel le filtre s'applique
 - **DispatcherType [] dispatcherTypes** : Spécifie les types de répartiteur (Request / Response) auxquels le filtre s'applique
 - **String [] servletNames** : Fournit un tableau de noms de servlets
 - **String displayName** : Nom du filtre
 - **String description** : Description du filtre



L'annotation @Webfilter

- **WebInitParam [] initParams** : Tableau de paramètres d'initialisation pour ce filtre
- **Boolean asyncSupported** : Opération asynchrone prise en charge par ce filtre
- **String smallIcon** : Petite icône pour ce filtre, si présent
- **String largeIcon** : Grande icône pour ce filtre, si présent
- **Exemple :**
 - L'exemple suivant est un simple LogFilter qui affiche la valeur de Init-param test-param et l'horodatage de l'heure actuelle sur la console.
 - Cela signifie que le filtre fonctionne comme une couche entre la requête et la réponse.
 - Ici, nous utilisons "/" "*" pour urlPattern (filtre applicable à toutes les servlets)



L'annotation @Webfilter

```
import java.io.IOException;
import javax.servlet.annotation.*;
import javax.servlet.*;
import java.util.*;
// classe implementant l'interface Filter
@WebFilter(urlPatterns = {"/*"}, initParams = { @WebInitParam (name = "test-param", value = "paramètre
d'Initialization")})
public class LogFilter implements Filter {
    public void init(FilterConfig config) throws ServletException {
        String testParam = config.getInitParameter("test-param"); // récupère le paramètre init
        System.out.println("Test Param: " + testParam); //affiche le paramètre init sur la console
    }
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException,
    ServletException { System.out.println("Time " + new Date().toString()); // imprime la date courante.
        chain.doFilter(request,response); // repasse la requête à la chaine de filtres }
    public void destroy() {
        /* Appelée avant que l'instance de filtre ne soit supprimée du service par le conteneur Web */ }
}
```

cours JEE - Dr. Abdessamad Belangour

86



Développer une servlet http

- Les étapes de développement d'une servlet sont les suivantes:
 1. Lecture de la requête (représentée par l'objet *HttpServletRequest*)
 2. Traitement
 3. Création de la réponse (représentée par l'objet *HttpServletResponse*)

cours JEE - Dr. Abdessamad Belangour

87



Développer une servlet http

Lecture d'une requête

- L'objet *HttpServletRequest* fournit un ensemble de méthodes pour avoir toutes les informations concernant une requête.
- Ces méthodes sont comme suit :
 - String **getMethod()** : Récupère la méthode HTTP utilisée par le client
 - String **getHeader(String name)** : Récupère la valeur de l'en-tête demandée
 - String **getRemoteHost()** : Récupère le nom de domaine du client
 - String **getRemoteAddr()** : Récupère l'adresse IP du client



Développer une servlet http

- String **getParameter(String name)** : Récupère la valeur du paramètre name d'un formulaire. Lorsque plusieurs valeurs sont présentes, la première est retournée
- String[] **getParameterValues(String name)** : Récupère les valeurs correspondant au paramètre name d'un formulaire, c'est-à-dire dans le cas d'une sélection multiple (cases à cocher, listes à choix multiples) les valeurs de toutes les entités sélectionnées
- Enumeration **getParameterNames()** : Retourne un objet *Enumeration* contenant la liste des noms des paramètres passés à la requête
- String **getServerName()** : Récupère le nom du serveur
- String **getServerPort()** : Récupère le numéro de port du serveur



Atelier 2

- Écrire une Servlet qui extrait les informations suivantes de la requête:
 - la méthode d'envoi de la requête HTTP utilisée par le client
 - l'adresse IP du client
 - Le nom du serveur
 - le numéro de port du serveur



Solution (1/2)

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

public class TestServlet extends HttpServlet {
    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException{
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html> <head> <title>entêtes HTTP</title> </head> <body>");
        out.println("Method d'envoi du client : "+request.getMethod());
        out.println("Adresse IP du client : "+request.getRemoteAddr());
        out.println("Nom du serveur : " + request.getServerName());
        out.println("Port du serveur : "+ request.getServerPort());
        out.println(" </body></html>");
    }
}
```



Atelier 3

Construire un formulaire HTML comprenant les informations suivantes :

- Nom (zone de texte)
- Prénom (zone de texte)
- Sexe (boutons radio M ou F)
- Fonction (options)
 - Enseignant
 - Étudiant (choix initial)
 - Ingénieur
 - Retraité
 - Autre
- Loisirs (cases à cocher)
 - Lecture
 - Sport
 - Voyage
- Commentaire (textarea)



Atelier 3 (suite)

- On demande d'écrire une servlet qui:
 - récupère ces valeurs lorsque l'utilisateur clique sur envoyer.
 - Affiche le nom de chaque champ et la valeur saisie par l'utilisateur



Solution (1/2)

```
<!DOCTYPE html>
<HTML>
  <HEAD> <title> formulaires et servlets </title></HEAD>
  <BODY>
    <FORM method="post" action="traiterFormulaire">
      Enregistrement d'un utilisateur : <br>
      Nom : <INPUT type="text" name="nom"> <br>
      Prénom :<INPUT type="text" name="prenom"><br>
      Sexe : <br> <INPUT type="radio" name="sexe" value="M" checked>Homme<br>
               <INPUT type="radio" name="sexe" value="F">Femme<br>
      Fonction :<SELECT name="fonction">
                   <OPTION VALUE="enseignant">Enseignant</OPTION>
                   <OPTION VALUE="etudiant">Etudiant</OPTION>
                   <OPTION VALUE="ingenieur" selected>Ingénieur</OPTION>
                   <OPTION VALUE="retraite">Retraité</OPTION> <OPTION VALUE="autre">Autre</OPTION>
               </SELECT> <br>
      Loisirs : <br><INPUT type="checkbox" NAME="loisirs" value="lecture" CHECKED>Lecture <br>
                   <INPUT type="checkbox" NAME="loisirs" value="sport">Sport <br>
                   <INPUT type="checkbox" NAME="loisirs" value="voyage">Voyage<br>
      Commentaire :<br><TEXTAREA rows="3" name="commentaire">Votre Commentaire</TEXTAREA><br>
      <INPUT type="submit" value="Envoyer">
    </FORM>
  </BODY>
</HTML>
```

[Index.html](#)

cours JEE - Dr. Abdessamad Belangour

94



Solution (2/2)

Essentiel de la servlet

```
...
out.println("<br><H3>Récupération des paramètres utilisateur </H3><br>");
out.println("<br>nom:"+request.getParameter("nom"));
out.println("<br>prénom:"+request.getParameter("prenom"));
out.println("<br>sexe:"+request.getParameter("sexe"));
out.println("<br>fonction:"+request.getParameter("fonction"));
out.println("<br>commentaire:"+request.getParameter("commentaire"));
out.println("essai de getParameterValues<br>");
out.println("<br>loisirs:<br>");
String[] valeursDeLoisirs=request.getParameterValues("loisirs");
for (int i=0 ; i < valeursDeLoisirs.length ; i++) {
    out.println(valeursDeLoisirs[i]+" ");
}
...
```

cours JEE - Dr. Abdessamad Belangour

95



Développer une servlet http

Création de la réponse

- Après lecture et traitement d'une requête, la réponse à fournir à l'utilisateur est représentée sous forme d'objet *HttpServletResponse*.
- Les méthodes de l'objet *HttpServletResponse* sont comme suit :
 - **void setHeader(String Nom, String Valeur)** : Définit une paire clé/valeur dans les en-têtes
 - **void setContentType(String type)** : Définit le type MIME de la réponse HTTP, c'est-à-dire le type de données envoyées au navigateur



Développer une servlet http

- **void setContentLength(int len)** : Définit la taille de la réponse
- **PrintWriter getWriter()** : Retourne un objet *PrintWriter* permettant d'envoyer du texte au navigateur client. Il se charge de convertir au format approprié les caractères Unicode utilisés par Java
- **ServletOutputStream getOutputStream()** : Définit un flot de données à envoyer au client, par l'intermédiaire d'un objet *ServletOutputStream*, dérivé de la classe *java.io.OutputStream*
- **void sendRedirect(String location)** : Permet de rediriger le client vers l'URL *location*



Développer une servlet http

- String **setStatus**(int StatusCode) : Définit le code de retour de la réponse
- Rappelons quelques codes de retour:
 - Code 202 (SC_ACCEPTED) : Requête acceptée.
 - Code 204 (SC_NO_CONTENT) : pas d'information à retourner.
 - Code 301 (SC_MOVED_PERMANENTLY) : la ressource demandée a été déplacée.
 - Code 400 (SC_BAD_REQUEST) : La requête est syntaxiquement incorrecte.
 - Code 403 (SC_FORBIDDEN) : le serveur comprend la requête mais refuse de la servir.
 - Code 404 (SC_NOT_FOUND) : la ressource demandée n'est pas disponible.
 - etc...



Développer une servlet http

■ Exemple :

- Écrire une Servlet qui effectue une redirection vers un site web donné.

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class PremiereServlet extends HttpServlet {
    @Override
    public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException,
        IOException {
        res.sendRedirect("http://www.google.co.ma");
    }
}
```



Développer une servlet http

- Remarque :
 - Pour éviter que la page soit rechargée à partir du cache :
 - `response.setHeader("Cache-Control", "no-cache"); //HTTP 1.1`
 - `response.setHeader("Pragma", "no-cache"); //HTTP 1.0`
 - `response.setDateHeader ("Expires", 0); /*prevents caching at the proxy server*/`



Suivi de session

- Le protocole HTTP est un protocole sans état
- Impossibilité alors de garder des informations d'une requête à l'autre (identifier un client d'un autre)
- Obligation d'utiliser différentes solutions pour remédier au problème d'état dont :
 - L'utilisation de cookies
 - L'utilisation de sessions



Suivi de session : cookies

- Les cookies représentent un moyen simple de stocker temporairement des informations chez un client, afin de les récupérer ultérieurement.
- Les cookies ont été introduits par la première fois dans Netscape Navigator
- Les cookies font partie des spécifications du protocole HTTP.
- L'en-tête HTTP réservé à l'utilisation des cookies s'appelle Set-Cookie, il s'agit d'une simple ligne de texte de la forme:
 - **Set-Cookie** : **nom**=VALEUR; **domain**=NOM_DE_DOMAINE; **expires**=DATE
- La valeur d'un cookie pouvant identifier de façon unique un client, ils sont souvent utilisés pour le suivi de session



Suivi de session : cookies

- L'API Servlet fournit la classe *javax.servlet.http.Cookie* pour travailler avec les Cookies
 - *Cookie(String name, String value)* : construit un cookie
 - *String getName()* : retourne le nom du cookie
 - *String getValue()* : retourne la valeur du cookie
 - *setValue(String new_value)* : donne une nouvelle valeur au cookie
 - *setMaxAge(int expiry)* : spécifie l'âge maximum du cookie en secondes
- Pour la création d'un nouveau cookie, il faut l'ajouter à la réponse (*HttpServletResponse*)
 - *addCookie(Cookie mon_cookie)* : ajoute à la réponse un cookie
- La Servlet récupère les cookies du client en exploitant la requête (*HttpServletRequest*)
 - *Cookie[] getCookies()* : récupère l'ensemble des cookies du site



Suivi de session : cookies

- Code pour créer un cookie et l'ajouter au client :

```
Cookie c = new Cookie("Id", "123");
res.addCookie(c);
```
- Code pour récupérer les cookies

```
Cookie[] cs= req.getCookies();
if (cs != null) {
    for (int i = 0; i < cs.length; i++) {
        String name = cs[i].getName();
        String value = cs[i].getValue();
        ...
    }
}
```
- Remarque :
 - Il n'existe pas dans l'API Servlet de méthode permettant de récupérer la valeur d'un cookie par son nom



Atelier 6

- Écrire une Servlet permettant d'identifier un client par l'intermédiaire des cookies.
 - Aide : vous pouvez obtenir un identifiant unique sur le temps par rapport à l'hôte sur lequel il a été généré grâce à la méthode `java.rmi.server.UID().toString()`.



Solution (1/2)

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

public class CookiesServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException,
        IOException {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        String contenu = null;
        Cookie[] tabCookies = req.getCookies();
        if (tabCookies != null)
            for (int i = 0; i < tabCookies.length; i++) {
                if (tabCookies[i].getName().equals("monId"))
                    contenu = tabCookies[i].getValue();
            }
    }
}
```

cours JEE - Dr. Abdessamad Belangour

106



Solution (2/2)

```
if (tabCookies == null || contenu == null) {
    contenu = new java.rmi.server.UID().toString();
    Cookie c = new Cookie("monId", contenu);
    c.setMaxAge(60*60*24*365);
    res.addCookie(c);
    out.println("Bonjour le nouveau");
}
else out.println("Encore vous");
out.close();
}
```

cours JEE - Dr. Abdessamad Belangour

107



Suivi de session : HttpSession

- Le plus gros problème des cookies est que les navigateurs ne les acceptent pas toujours
- L'utilisateur peut configurer son navigateur pour qu'il refuse ou pas les cookies
- Les navigateurs n'acceptent que 20 cookies par site, 300 par utilisateur et la taille d'un cookie peut être limitée à 4096 octets (4 ko)



Suivi de session : HttpSession

- Solutions : utilisation de l'API de suivi de session *javax.servlet.http.HttpSession*
- Méthodes de création liées à la requête (*HttpServletRequest*)
 - *HttpSession getSession()* : retourne la session associée à l'utilisateur si elle existe sinon lui crée une nouvelle
 - *HttpSession getSession(boolean p)* : création selon la valeur de *p*
- Gestion d'association (*HttpSession*)
 - *setAttribute(String an, Object av)* : associe l'objet *av* à la chaîne *an*
 - *Object getAttribute(String name)* : retourne l'attribut *name*
 - *Enumeration getAttributeNames()* : retourne les noms de tous les attributs
 - *removeAttribute(String na)* : supprime l'attribut associé à *na*
- Destruction (*HttpSession*)
 - *invalidate()* : expire la session



Suivi de session : HttpSession

- Exemple : Servlet qui permet d'utiliser le suivi de session pour un compteur

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

cours JEE - Dr. Abdessamad Belangour

110



Collaboration de Servlets

- Les Servlets s'exécutant dans le **même** serveur peuvent dialoguer entre elles

- Deux principaux styles de collaboration :

- **Partage d'information** : un état ou une ressource.

Exemple : un magasin en ligne pourrait partager les informations sur le stock des produits ou une connexion à une base de données

- **Partage du contrôle** : une requête.

Réception d'une requête par une Servlet et laisser l'autre Servlet une partie ou toute la responsabilité du traitement

cours JEE - Dr. Abdessamad Belangour

111



Collaboration de Servlets : partage d'information

- 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 nom, Object o)* : lie un objet sous le nom indiqué
 - *Object getAttribute(String nom)* : retrouve l'objet sous le nom indiqué
 - *Enumeration getAttributeNames()* : retourne l'ensemble des noms de tous les attributs liés
 - *void removeAttribute(String nom)* : supprime l'objet lié sous le nom indiqué

cours JEE - Dr. Abdessamad Belangour

112



Partage d'information

- **Exemple** : Servlets qui vendent des pizzas et partagent une spécialité du jour

```
public class PizzasAdmin extends HttpServlet {  
    protected void doGet(HttpServletRequest req, HttpServletResponse res)  
        throws ServletException, IOException {  
        res.setContentType("text/plain");  
        PrintWriter out = res.getWriter();  
        ServletContext context = this.getServletContext();  
        context.setAttribute("Specialite", "Quatre saisons");  
        out.println("La pizza du jour a été définie.");  
    }  
}
```

Création d'un attribut

```
public class PizzasClient extends HttpServlet {  
    protected void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException,  
        IOException {  
        res.setContentType("text/plain");  
        PrintWriter out = res.getWriter();  
        ServletContext context = this.getServletContext();  
        String pizza_spec = (String)context.getAttribute("Specialite");  
        out.println("Aujourd'hui, notre spécialité est : " + pizza_spec);  
    }  
}
```

Lecture de l'attribut

cours JEE - Dr. Abdessamad Belangour

113



Collaboration de Servlets : partage du contrôle

- Pour une collaboration dynamique entre servlets, deux possibilités existent:
 - Déléguer entièrement la requête à une autre servlet : méthode **forward()**
 - Inclure la réponse d'une autre servlet dans la servlet en cours : méthode **include()**
- Ces deux méthodes appartiennent à l'interface **RequestDispatcher** du package `javax.servlet`
 - *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

cours JEE - Dr. Abdessamad Belangour

114



Partage du contrôle (forward)

- Soit l'exemple suivant :
 - Une servlet (Servlet1) reçoit une requête
 - Elle y place un attribut *X* qu'elle y met la chaîne "salut"
 - Elle renvoie ensuite cette requête à une autre Servlet (Servlet2).
 - Servlet2 récupère cet attribut et se charge de créer la réponse qu'elle renvoie à l'utilisateur.
- Attention:
 - Servlet1 ne doit pas toucher à la réponse car c'est Servlet2 qui s'en charge.
 - Après le renvoi de la requête à Servlet2, Servlet1 ne doit plus toucher à la requête.

cours JEE - Dr. Abdessamad Belangour

115



Partage du contrôle (forward)

Code pour servlet1

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

public class Servlet1 extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {

        req.setAttribute("X", "salut");
        RequestDispatcher dispat = req.getRequestDispatcher("/cheminServlet2");
        dispat.forward(req, res);
    }
}
```



Partage du contrôle (forward)

Code pour servlet2

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

public class Servlet2 extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {

        res.setContentType("text/plain");
        PrintWriter out = res.getWriter();
        out.println("l'attribut que j'ai récupéré de servlet 1 est: "+req.getAttribute("X"));
    }
}
```



Partage du contrôle (include)

■ Soit l'exemple suivant :

- Une servlet (Principale) reçoit une requête
- Elle y place un attribut x qu'elle y met la chaîne "3"
- Elle inclut une autre Servlet dans le traitement (Secondaire)
- Secondaire récupère cet attribut et affiche son carré
- Principale reprend le contrôle, elle modifie x en "5"
- Secondaire récupère encore une fois cet attribut et affiche son carré
- Principale reprend le contrôle

■ Remarque:

- Secondaire ne doit pas fermer la réponse par `</body>` car c'est Principale qui s'en charge.
- C'est Principale qui se charge de préciser le type MiMe de la réponse.



Partage du contrôle (include)

Code pour Principale

```
public class Principale extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        out.println("<HTML><BODY>");
        req.setAttribute("x", "3");
        RequestDispatcher dispat = req.getRequestDispatcher("/cheminServlet2");
        dispat.include(req, res);
        req.setAttribute("x", "5");
        dispat.include(req, res);
        out.println("</BODY></HTML>");
    }
}
```



Partage du contrôle (include)

Code pour Secondaire

```
public class Secondaire extends HttpServlet {  
    @Override  
    protected void doGet(HttpServletRequest req, HttpServletResponse res)  
        throws ServletException, IOException {  
        PrintWriter out = res.getWriter();  
        String ch=(String)req.getAttribute("x");  
        int x=Integer.parseInt(ch);  
        out.println(" Le carré de "+ x +" est :"+x*x);  
    }  
}
```

cours JEE - Dr. Abdessamad Belangour

120