

# SERVLET

Servlet

# PREMIÈRE SERVLET - ETUDE PRÉLIMINAIRE

- Les servlets représentent une solution technologie Java de la programmation avec CGI.
- Il s'agit de programmes exécutés sur un serveur Web, qui servent de couche intermédiaire entre une requête provenant d'un navigateur Web et un autre service HTTP.

# PREMIÈRE SERVLET - ETUDE PRÉLIMINAIRE

- Leur tâche est de :
  1. **Lire toutes les données envoyées par l'utilisateur** : Ces données sont typiquement saisies dans un formulaire sur une page Web, mais elles peuvent également provenir d'une applet Java ou d'un programme client HTTP particulier.
  2. **Chercher d'autres informations sur la requête, à l'intérieur de cette requête HTTP** : Ces informations contiennent des détails sur les capacités du navigateur, sur les cookies, sur le nom de l'hôte du programme envoyant la requête

# PREMIÈRE SERVLET - ETUDE PRÉLIMINAIRE

- Leur tâche est de :
- 3. **Générer des résultats** : Ce processus peut nécessiter une communication avec la base de données, ou en invoquant une ancienne application, ou encore en calculant directement la réponse.
- 4. **Formater le résultat dans un document** : Dans la plupart des cas, cela impliquera l'incorporation des informations dans une page HTML.

# PREMIÈRE SERVLET - ETUDE PRÉLIMINAIRE

- Leur tâche est de :

- 5. Définir les paramètres de la réponse HTTP appropriés :** Cela signifie qu'il faut indiquer au navigateur le type de document renvoyé (c'est à dire HTML), définir les cookies, mémoriser les paramètres, ainsi que d'autres tâches.
- 6. Renvoyer le document au client :** Ce document peut être envoyé au format texte (HTML), au format binaire (comme pour des images GIF), ou même dans un format compressé comme gzip

# PREMIÈRE SERVLET - ETUDE PRÉLIMINAIRE

- Il existe plusieurs raisons expliquant la nécessité de construire des pages en temps réel
1. **La page Web est fondée sur des données envoyées par l'utilisateur** : Par exemple, les pages de résultats des moteurs de recherche et des pages de confirmation de commande dans les magasins en ligne sont spécifiques à des requêtes particulières d'utilisateurs.

# PREMIÈRE SERVLET - ETUDE PRÉLIMINAIRE

2. **La page Web est calculée à partir d'informations qui sont fréquemment modifiées:** Par exemple, un bulletin météo ou une page résumant les dernières nouvelles quotidiennes peut construire la page de manière dynamique.
3. **La page Web se sert d'informations provenant de bases de données appartenant à des entreprises ou à d'autres sources situées au niveau d'un serveur :** Par exemple, un site de commerce électronique peut utiliser un servlet pour construire une page Web, qui établit la liste des prix et la disponibilité de chaque article en vente.

# DÉCOUVERTE DE NOTRE PREMIÈRE SERVLET

- Notre première servlet sera relativement simple.
- A partir d'un petit formulaire rassemblant les coordonnées d'une personne, cette servlet devra récupérer ces informations et avertir le client qu'elle ont été bien pris en compte en renvoyant une nouvelle page Web.



# DÉCOUVERTE DE NOTRE PREMIÈRE SERVLET

Formulaire x +

← → ↻ ☆ http://localhost:8080/nouv/

---

## Enregistrement de vos coordonnées

---

**Civilite; :**

**Nom :**

**Prenom :**

**Age :**

---

Servlet

- Index.html

# DÉCOUVERTE DE NOTRE PREMIÈRE SERVLET



Servlet

- Page Web dynamique délivrée par la servlet relative à la saisie du client .

# CODAGE DE LA PAGE WEB STATIQUE "INDEX.HTML" REPRÉSENTANT LE FORMULAIRE

```
web.xml index.html X Maservlet.java
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head><title>Formulaire</title></head>
<body>
<h2>Enregistrement de vos coordonnées</h2>
<hr>
<form method="post" action="hello">
  <h3>Civilite; :
  <select name="civilite">
    <option>Monsieur</option>
    <option>Madame</option>
    <option>Mademoiselle</option>
  </select></h3>
  <h3>Nom : <input type="text" name="nom" size="24"></h3>
  <h3>Prenom : <input type="text" name="prenom"></h3>
  <h3>Age : <input type="text" name="age" size="5"></h3>
  <hr /><input type="submit" value="Envoyer le formulaire">
    <input type="reset" value="Tout effacer">
</form>
</body>
</html>
```

Servlet

# CODAGE DE LA SERVLET "MASERVLET.JAVA" QUI PRODUIT LA PAGE WEB DYNAMIQUE CORRESPONDANTE

```
web.xml  index.html  Maservlet.java X

import javax.servlet.*;
public class Maservlet extends HttpServlet {

    //Traiter la requête HTTP Get

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html"); // type MIME pour l'en-tête http --> Page HTML
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head><title>Enregistrement coordonnées</title></head>");
        out.println("<body>");
        out.println("<h2>Enregistrement de vos coordonnées effectué</h2>");
        out.println("<hr width=75%>");
        out.print("<p><b>Bonjour " + request.getParameter("civilite") + " ">");
        out.print(request.getParameter("prenom") + " ">");
        out.println(request.getParameter("nom") + ".");
        int âge = Integer.parseInt(request.getParameter("age"));
        out.println("</p></b></body></html>");
    }

    //Traiter la requête HTTP post
    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        doGet(request, response);
    }
}
```

Servlet

# EXPLICATION ET DESCRIPTION DE MASERVLET

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

- Le package **java.io** est nécessaire pour que notre servlet puisse renvoyer des informations en réponse à la requête qu'elle reçoit.
- Le package **javax.servlet** est nécessaire puisque notre servlet lance une exception de type **ServletException**, défini dans ce package.
- Tous les autres éléments spécifiques aux servlets dont nous avons besoin proviennent du package **javax.servlet.http**.

# EXPLICATION ET DESCRIPTION DE MASERVLET

```
public class Formulaire extends HttpServlet {
```

- Notre servlet est une classe qui hérite de la classe **HttpServlet**.
- Cette classe permet d'être en relation directe avec le serveur **Tomcat**.
- hérite elle-même de la classe **javax.servlet.GenericServlet** et implémente l'interface **Serializable**.
- Elle propose donc des méthodes spécialisées et spécifiques au **protocole HTTP** dont voici les plus courantes :

# EXPLICATION ET DESCRIPTION DE MASERVLET

- **doDelete(HttpServletRequest req, HttpServletResponse resp)**, qui traite les requêtes **HTTP DELETE**.
- **doGet(HttpServletRequest req, HttpServletResponse resp)**, qui traite les requêtes **HTTP GET**.
- **doPost(HttpServletRequest req, HttpServletResponse resp)**, qui traite les requêtes **HTTP POST**.
- **doOptions(HttpServletRequest req, HttpServletResponse resp)**, qui traite les requêtes **HTTP OPTIONS**.

# EXPLICATION ET DESCRIPTION DE MASERVLET

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

- La méthode **doGet** est appelée automatiquement par la méthode service chaque fois que la servlet reçoit une requête de type GET.
- Cette méthode prend deux arguments : **HttpServletRequest** et **HttpServletResponse**.
- **HttpServletRequest** renferme des méthodes grâce auxquelles vous pouvez déterminer des informations sur les données entrantes.
- **HttpServletResponse** permet de spécifier les informations renvoyées, comme les codes d'état HTTP (200, 404, etc.), les en-têtes de réponse (ContentType, Set-Cookie, etc.).



# EXPLICATION ET DESCRIPTION DE MASERVLET

```
PrintWriter out = response.getWriter();
```

- lorsque le navigateur fait appel à cette servlet, il donne, en même temps, tous les renseignements le concernant grâce aux deux classes `HttpServletRequest` et `HttpServletResponse`.
- Dans la classe `HttpServletResponse` se trouve notamment les informations concernant le contexte d'affichage de la zone du navigateur qu'on pourra les récupérer à l'aide de **`getWriter`**.
- l'objet **`out`** de type **`PrintWriter`** représente maintenant la zone d'affichage du navigateur.

# EXPLICATION ET DESCRIPTION DE MASERVLET

```
request.getParameter("Civillite")  
request.getParameter("prenom")  
request.getParameter("nom")  
request.getParameter("age")
```

- L'objet **request** possède une méthode extrêmement importante pour récupérer chacun des champs composant le formulaire.
- Cette méthode s'appelle **getParameter** issue de la classe **HttpServletRequest**.
- Lorsque nous utilisons cette méthode **getParameter**, il suffit de préciser en paramètre de la méthode, sous forme d'une chaîne de caractère de type **String**, le nom du champ désiré (nom).

# STRUCTURATION DE L'APPLICATION WEB INTÉGRANT NOTRE PREMIÈRE SERVLET

- Une servlet représente un CGI, et c'est un programme qui ne fonctionne qu'au sein d'un serveur Web.
- Ce programme est spécialisé et adaptée au protocole HTTP.
- C'est le serveur Web qui se charge de le démarrer au moment opportun.
- Remarque : servlet ne dispose pas de méthode main. Elle dispose d'une structure permettant d'être en relation directe avec le serveur Web.
- Le serveur Tomcat est un serveur Web totalement adapté à toutes les technologies Java comme les servlets et les pages JSP.

# LES APPLICATIONS WEB <WEBAPPS>

- Une application Web doit répondre à un certain nombre d'attente de la part du client.
- Elle est généralement composée d'un certain nombres d'éléments en interaction entre eux.
- dans notre exemple, nous disposons pour la même application Web :
  1. d'une page Web statique qui est composée du formulaire à remplir
  2. d'une servlet dont le but est la construction d'une page Web dynamique en correspondance avec la saisie de l'utilisateur.

# LES APPLICATIONS WEB <WEBAPPS>

- Une application Web peut être relativement conséquente et être composée de beaucoup plus d'éléments que cela.
- Elle peut posséder :
  1. plusieurs pages statiques <\*.html>,
  2. plusieurs servlets,
  3. des pages JSP,
  4. des applets,
  5. Etc.

# LES APPLICATIONS WEB <WEBAPPS>

- Cette complexité ne doit pas du tout apparaître pour le client.
- L'utilisation de l'application Web doit au contraire être la plus conviviale possible et sa structure totalement transparente.
- l'utilisateur doit juste proposer le nom de l'application Web, dans l'URL, à la suite du nom du site.
- Pour que cela soit aussi simple pour l'utilisateur, il est nécessaire d'avoir une architecture particulière, prédéfinie, qui sera la même pour toutes les applications Web de type Java.

# STRUCTURE D'UNE APPLICATION WEB <WEBAPPS>

- Une application web est donc visible par son nom d'appel dans l'URL lors de l'exécution d'un script.
- Une application web est donc visible par son nom d'appel dans l'URL lors de l'exécution d'un script.
- Cette arborescence possède une structure établie par les développeurs du serveur Tomcat
- Lorsque l'utilisateur se connecte sur le site, et grâce à cette organisation particulière, le serveur Web Tomcat va systématiquement procéder à la même démarche.

# STRUCTURE D'UNE APPLICATION WEB <WEBAPPS>

- Cette démarche est la suivante :
  1. Recherche de la page d'accueil correspondant à l'application Web. C'est généralement une page Web statique et pour plus de souplesse elle porte le nom de **<index.html>**.

Elle peut comporter un formulaire à remplir.  
(On peut avoir aussi des pages Web dynamiques de type JSP ou servlets).
  2. Après validation du formulaire, le serveur Web, au travers de la **servlet**, fabrique une nouvelle page Web en correspondance de la saisie réalisée par le client. L'utilisation et la situation de la **bonne servlet** est indiquée par le descripteur de déploiement **<web.xml>**.



# STRUCTURE D'UNE APPLICATION WEB <WEBAPPS>

- Pour que tout se passe convenablement, il faut donc respecter l'architecture des dossiers, telle qu'elle vous est présentée dans le tableau ci-dessous :

Dossiers		Type de fichier
webapp		Pages statiques : <b>&lt;*.html&gt;</b> Pages dynamiques : <b>&lt;*.jsp&gt;</b> Applets : <b>&lt;*.class&gt;</b>
	WEB-INF	Fichier de configuration de l'application WEB : <b>&lt;web.xml&gt;</b>
	classes	Emplacement des servlets : <b>&lt;*.class&gt;</b> . Les servlets sont nécessairement compilés.
	lib	Bibliothèques <b>&lt;*.jar&gt;</b> non standards comme les drivers JDBC.
	paquetages	Dossiers fabriqués par les applets lorsque nous décidons d'intégrer les paquetages.

# LE FICHIER <WEB.XML>

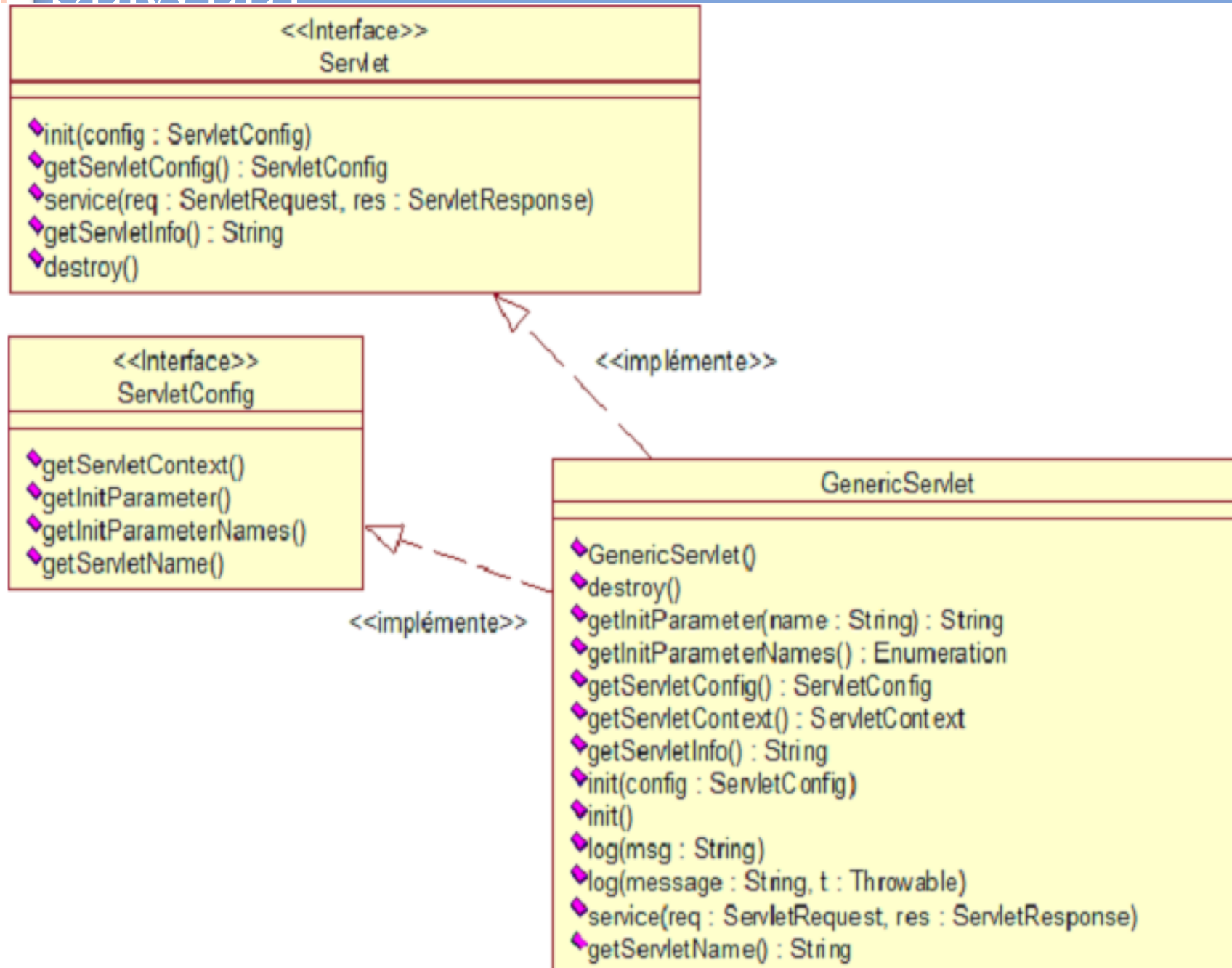
- Le fichier <web.xml> est un fichier XML décrivant notamment comment le client du serveur fait appel aux servlets d'une application Web.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc./DTD Web Application 2.3
3  <web-app>                                Application Web
4  <display-name>FormulairePersonne</display-name>
5  <servlet>
6  <servlet-name>formulaire</servlet-name>
7  <servlet-class>formulairepersonne.Formulaire</servlet-class>
8  </servlet>
9  <servlet-mapping>
10 <servlet-name>formulaire</servlet-name>
11 <url-pattern>/formulaire</url-pattern>
12 </servlet-mapping>
13 </web-app>
```

*Correspondance entre la servlet de nom : formulaire et sa classe : formulairepersonne.Formulaire*

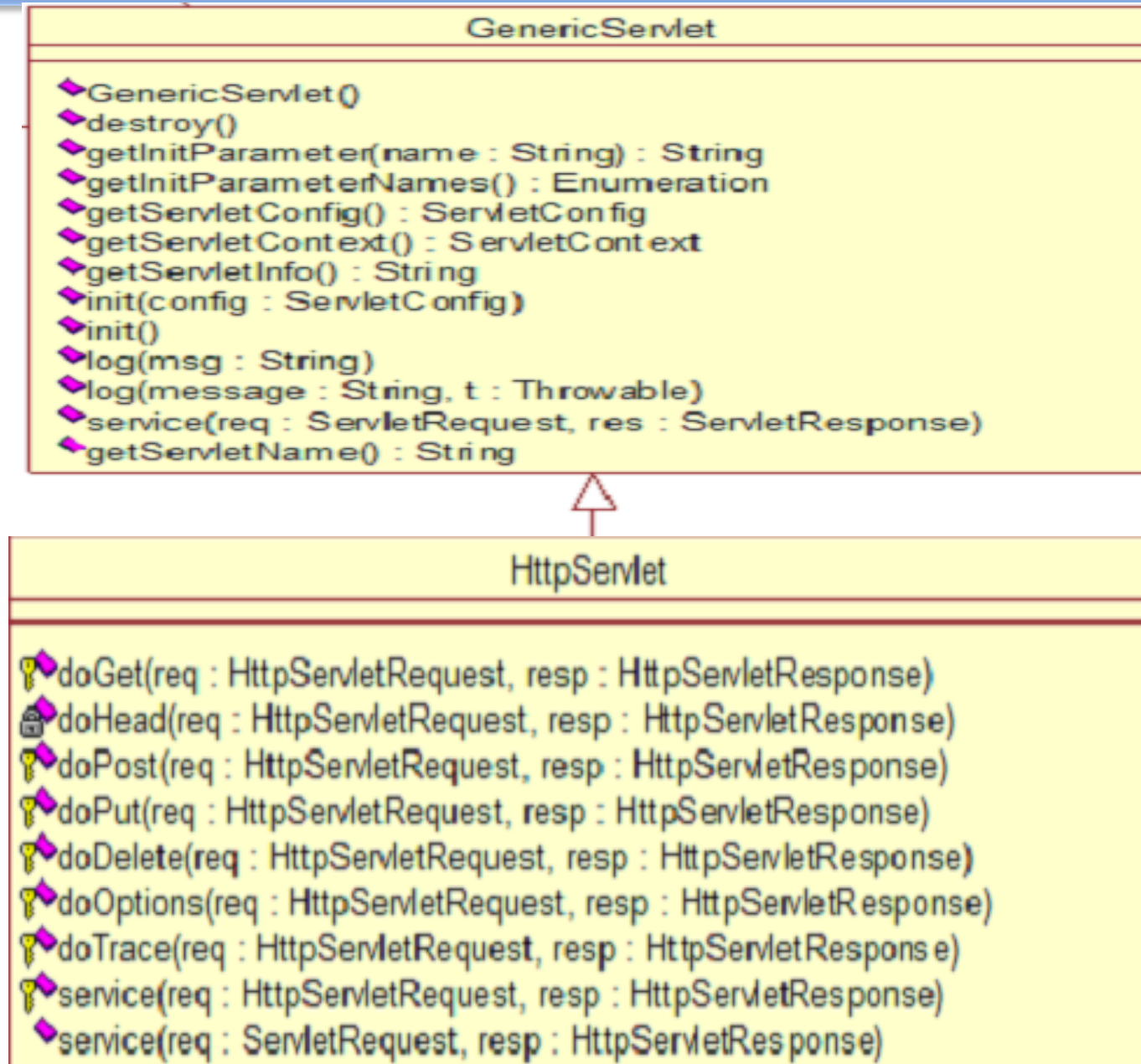
*Correspondance entre la servlet de nom : formulaire et son chemin d'accès dans l'URL : /formulaire*

# ARCHITECTURE FONDAMENTALE D'UNE SERVLET



Servlet

# ARCHITECTURE FONDAMENTALE D'UNE SERVLET



Servlet

# UTILISATION DE L'OBJET REQUEST

L'interface **ServletRequest** définit plusieurs méthodes permettant de lire les données présentes dans la requête :

- **public String[] getParameterValues(String nom)**

retourne les valeurs du paramètre dont le nom est passé en argument (ex. les sélections multiples).

- **public Enumeration getParameterNames()**  
retourne une énumération des noms des paramètres de la requête.

- **public Map getParameterMap()** retourne tous les paramètres stockés dans un objet de type Map.

Chaque nom de paramètre est une clé.



# UTILISATION DE L'OBJET REQUEST

Informations sur le client et le serveur:

- **public String getProtocol ()** - Le protocole employé par la requête. Normalement, il s'agit de "HTTP".
- **public String getServerName ()** - Le nom du serveur ayant reçu la requête. Cette méthode utile en cas d'utilisation de serveurs virtuels.
- **public String getRemoteAddr ()** - L'adresse IP du client ayant envoyé la requête.
- **public String getRemoteHost ()** - Le nom du client ayant envoyé la requête.

# UTILISATION DE L'OBJET REQUEST

Requête HTTP de type GET

- **public String getQueryString()** ;- à l'aide de cette méthode, la servlet lit les paramètres ajoutés à la fin de l'URL
- Exemple :

**http://localhost:8080/UneApplicationWeb/**  
**Identification?utilisateur=ENSA**

Dans ce cas, un appel à la méthode `getQueryString()` retournera : "utilisateur=ENSA".

Servlet

# UTILISATION DE L'OBJET REQUEST

Récupération du chemin spécifique à une ressource stockée dans une application Web

- Il est possible d'ajouter des informations sous forme d'une prolongation apparente du chemin d'accès à la ressource désignée par L'URL.
- Exemple :

<http://localhost:8080/UneApplicationWeb/Identification/extra/path/info>

Dans ce cas, un appel à la méthode getPathInfo() retournera : ["/extra/path/info"](#).

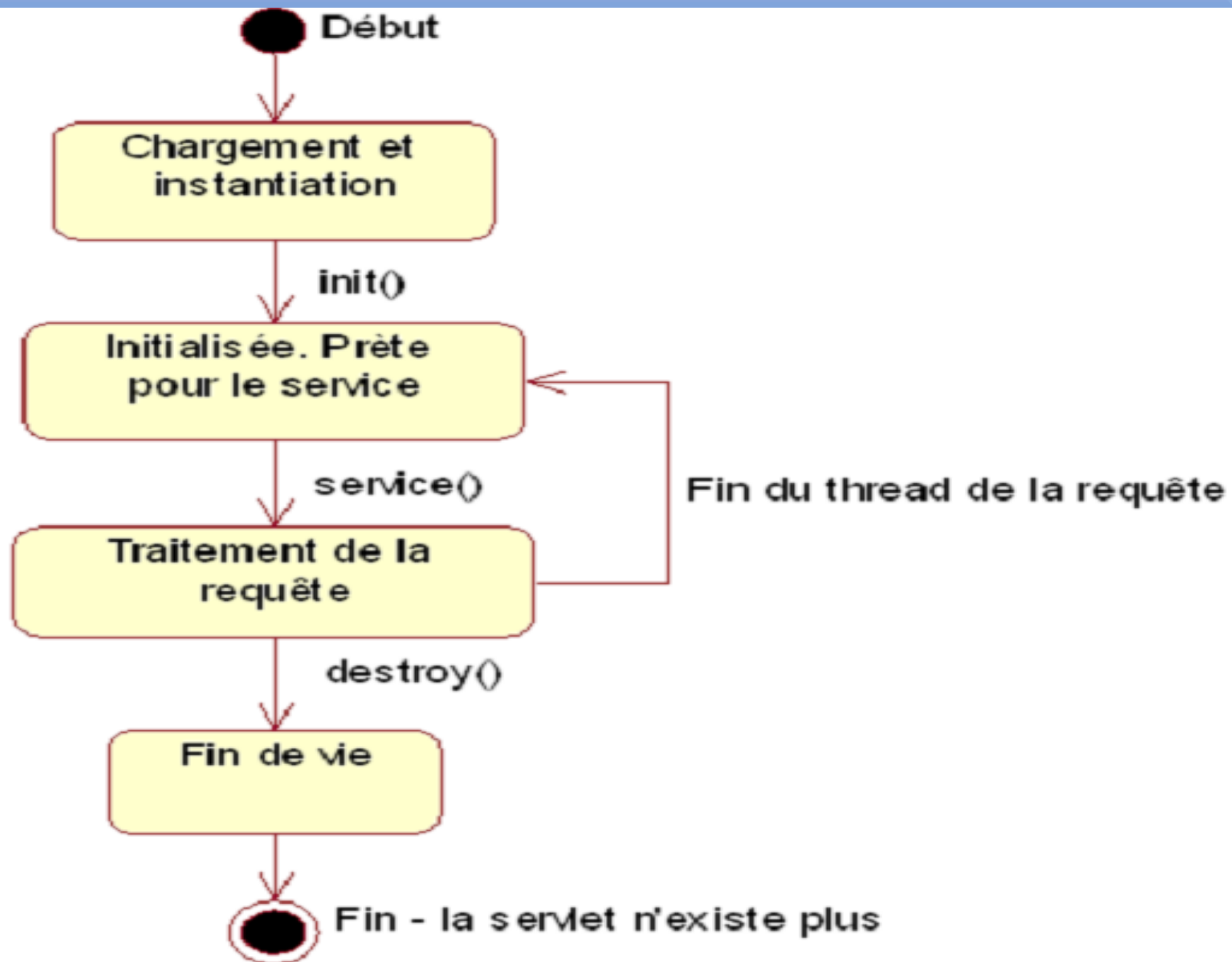


# CYCLE DE VIE DE LA SERVLET

quatre étapes dans le cycle de vie d'une servlet :

- Chargement et instanciation (création de l'objet relatif à la classe mise en oeuvre) ;
- Initialisation ;
- Traitement des requêtes ;
- Fin de vie

# CYCLE DE VIE DE LA SERVLET



# CYCLE DE VIE DE LA SERVLET

Début :

- Pour la phase de création, c'est au développeur de décider.
- L'élément **<load-on-start-up>** lorsqu'il est présent, contient un entier positif qui indique qu'il faut charger la servlet au démarrage du serveur.
- L'ordre de chargement des servlets est déterminé par cette valeur.
- Les servlets ayant la plus petite valeur sont chargées les premières.

# CYCLE DE VIE DE LA SERVLET

```
web.xml  Formulaire.html  Formulaire.java

<?xml version="1.0" encoding="UTF-8" ?>
<web-app version="2.4"
  xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
  <display-name>Gestion du Personnel</display-name>
  <welcome-file-list>
    <welcome-file>Formulaire.html</welcome-file>
  </welcome-file-list>
  <servlet>
    <display-name>Enregistrement du Personnel</display-name>
    <servlet-name>FormulairePersonne</servlet-name>
    <servlet-class>Formulaire</servlet-class>
    <init-param>
      <param-name>jdbc.Driver</param-name>
      <param-value>com.mysql.jdbc.Driver</param-value>
    </init-param>
    <init-param>
      <param-name>localisation</param-name>
      <param-value>jdbc:mysql://localhost/gestion</param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>FormulairePersonne</servlet-name>
    <url-pattern>/formulaire</url-pattern>
  </servlet-mapping>
</web-app>
```

Servlet

# CYCLE DE VIE DE LA SERVLET

```
public void init() throws ServletException {  
    String pilote = getInitParameter("jdbc.Driver");  
    String BaseDonnées = getInitParameter("localisation");  
    try {  
        Class.forName(pilote);  
        Connection connexion = DriverManager.getConnection(BaseDonnées, "root", "manu");  
        instruction = connexion.prepareStatement(requête);  
    }  
    catch (ClassNotFoundException e) {  
        log("Driver BD non trouvé"); throw new ServletException();  
    }  
    catch (SQLException e) {  
        log("Base de données non trouvée"); throw new ServletException();  
    }  
}
```

Servlet

# CYCLE DE VIE DE LA SERVLET

Fin de service :

- Lorsque le conteneur doit supprimer une servlet, soit parce qu'il doit être arrêté ou lorsqu'une **ServletException** est lancée, il appelle sa méthode **destroy()**.
- cette méthode ne détruit pas la servlet. Il s'agit simplement de donner à celle-ci l'opportunité de libérer les ressources qu'elle utilise ou qu'elle a ouvertes.
- Une fois cette méthode appelée, le conteneur ne doit plus transmettre de requêtes à la servlet.

# CYCLE DE VIE DE LA SERVLET

Fin de service :

- La méthode `destroy()` permet donc à la servlet de **libérer les ressources** qu'elle utilise. Il peut s'agir de **fermer une connexion** à une base de données, ou de **fermer des fichiers**, de **vider un flux** ou de **fermer les connexions réseaux**.

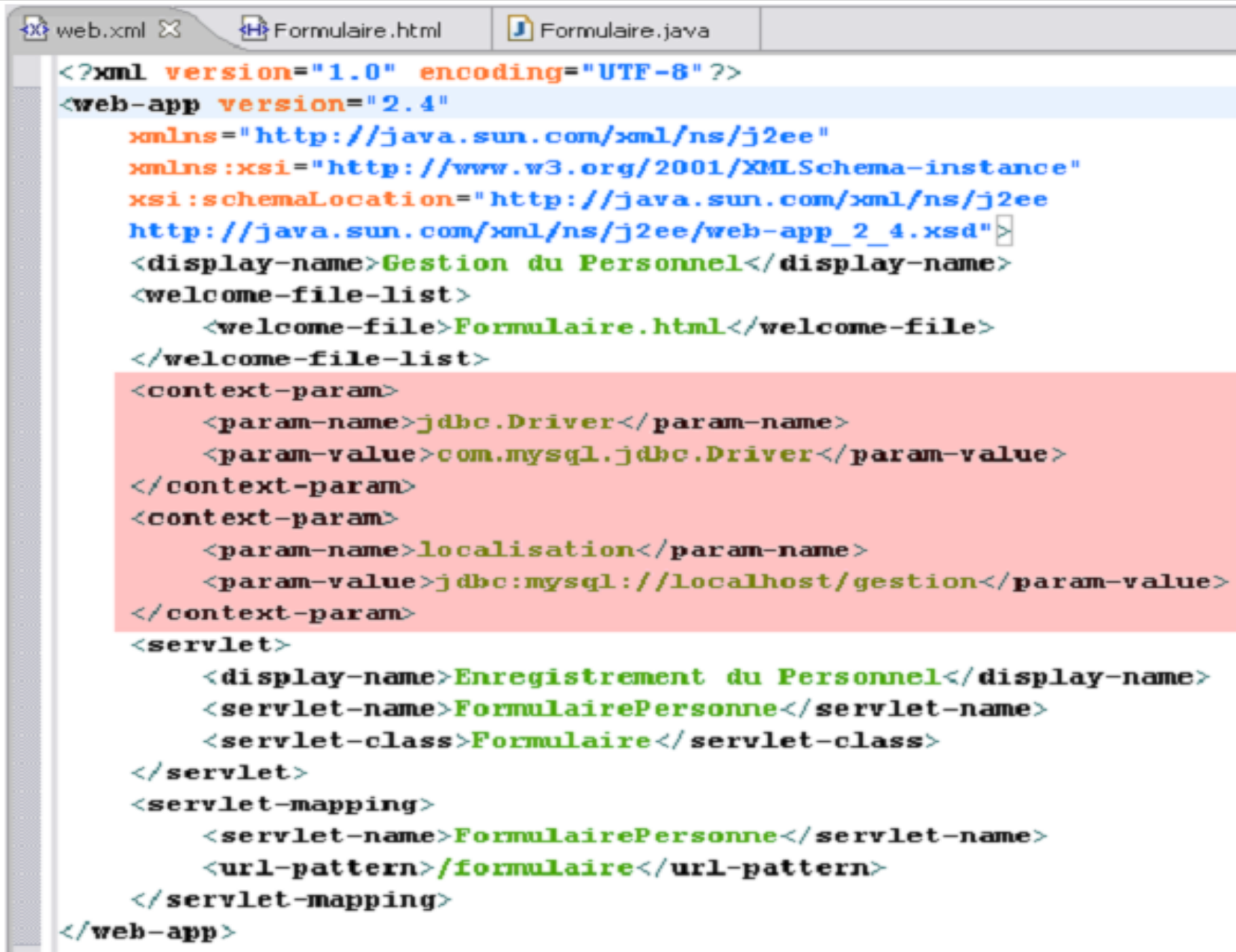
```
public void destroy() {  
    if (connexion != null)  
        try {  
            connexion.close();  
        }  
        catch (SQLException e) { log("Erreur fermeture BD"); }  
}
```

# PARAMÈTRES DE L'APPLICATION WEB

- En imaginant que votre application Web dispose de plusieurs servlets qui font appel à la base de données, il peut être gênant de placer les mêmes paramètres d'initialisation `<init-param>` du pilote JDBC pour chacune de ces servlets.
- Il serait préférable d'utiliser les paramètres d'initialisation de l'application Web, appelé paramètres de contexte `<context-param>`
- Ils sont accessibles pour tous les composants Web constituant l'application Web.



# PARAMÈTRES DE L'APPLICATION WEB



The screenshot shows a code editor with three tabs: 'web.xml', 'Formulaire.html', and 'Formulaire.java'. The 'web.xml' tab is active, displaying the following XML code:

```
<?xml version="1.0" encoding="UTF-8" ?>
<web-app version="2.4"
  xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
  >
  <display-name>Gestion du Personnel</display-name>
  <welcome-file-list>
    <welcome-file>Formulaire.html</welcome-file>
  </welcome-file-list>
  <context-param>
    <param-name>jdbc.Driver</param-name>
    <param-value>com.mysql.jdbc.Driver</param-value>
  </context-param>
  <context-param>
    <param-name>localisation</param-name>
    <param-value>jdbc:mysql://localhost/gestion</param-value>
  </context-param>
  <servlet>
    <display-name>Enregistrement du Personnel</display-name>
    <servlet-name>FormulairePersonne</servlet-name>
    <servlet-class>Formulaire</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>FormulairePersonne</servlet-name>
    <url-pattern>/formulaire</url-pattern>
  </servlet-mapping>
</web-app>
```

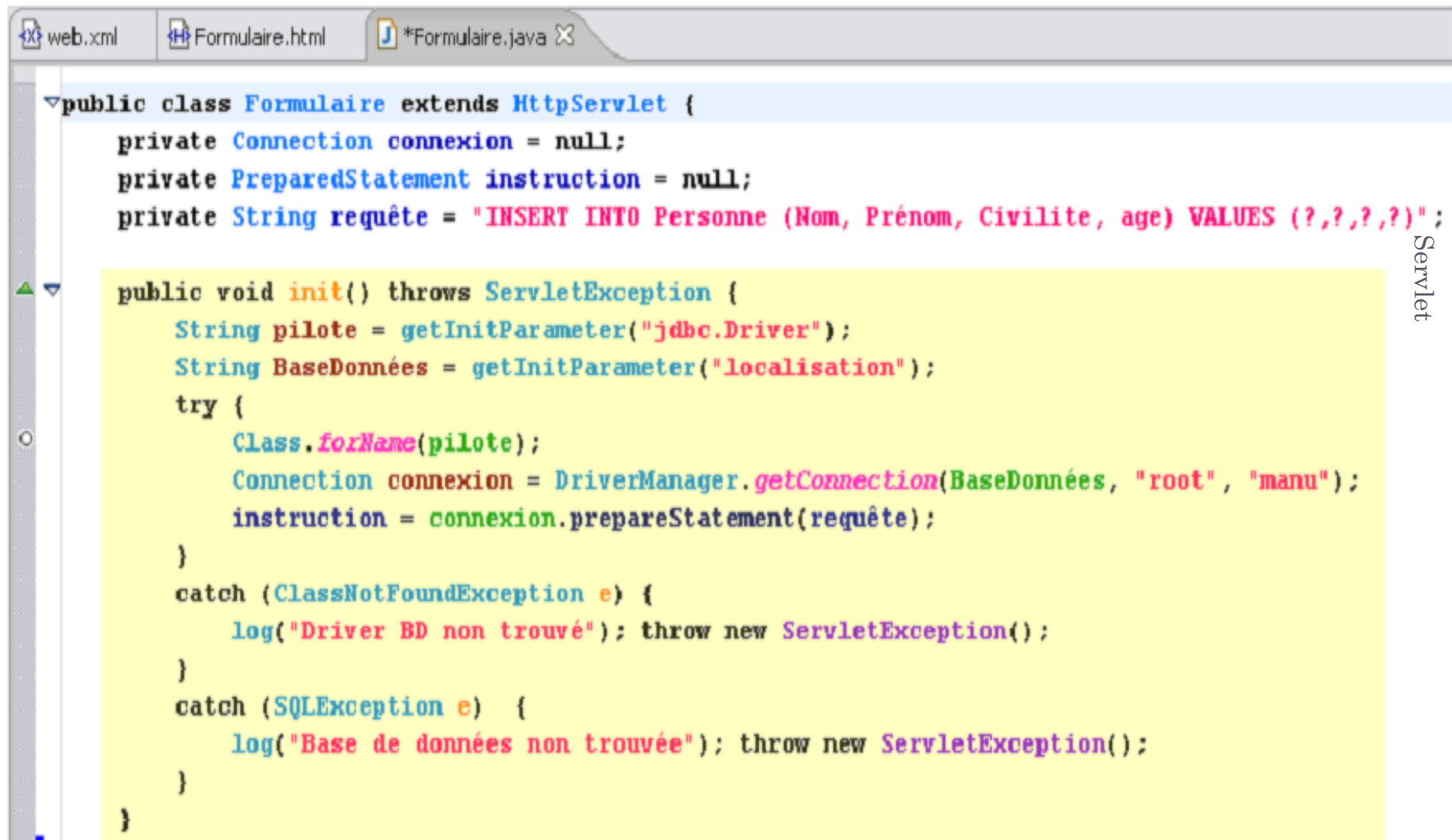
Servlet

# PARAMÈTRES DE L'APPLICATION WEB

```
public class Formulaire extends HttpServlet {  
    private Connection connexion = null;  
    private PreparedStatement instruction = null;  
    private String requête = "INSERT INTO Personne (Nom, Prénom, Civilite, age) VALUES (?, ?, ?, ?)";  
  
    public void init() throws ServletException {  
        String pilote = getServletContext().getInitParameter("jdbc.Driver");  
        String BaseDonnées = getServletContext().getInitParameter("localisation");  
        try {  
            Class.forName(pilote);  
            Connection connexion = DriverManager.getConnection(BaseDonnées, "root", "manu");  
            instruction = connexion.prepareStatement(requête);  
        }  
    }  
}
```

Servlet

# JOURNALISATION DES ÉVÉNEMENTS DANS LES SERVLETS



The screenshot shows an IDE window with three tabs: 'web.xml', 'Formulaire.html', and '\*Formulaire.java'. The code in the active tab is a Java Servlet named 'Formulaire' that extends 'HttpServlet'. It has three private attributes: 'connexion' of type 'Connection', 'instruction' of type 'PreparedStatement', and 'requête' of type 'String' containing an SQL insert statement. The 'init()' method is highlighted in yellow and contains logic to initialize the database connection using 'getInitParameter()' to retrieve 'jdbc.Driver' and 'localisation'. It uses 'Class.forName()' to load the driver, 'DriverManager.getConnection()' to get the connection, and 'prepareStatement()' to create the statement. Two catch blocks handle 'ClassNotFoundException' and 'SQLException', logging errors and throwing 'ServletException'.

```
public class Formulaire extends HttpServlet {  
    private Connection connexion = null;  
    private PreparedStatement instruction = null;  
    private String requête = "INSERT INTO Personne (Nom, Prénom, Civilite, age) VALUES (?, ?, ?, ?)";  
  
    public void init() throws ServletException {  
        String pilote = getInitParameter("jdbc.Driver");  
        String BaseDonnées = getInitParameter("localisation");  
        try {  
            Class.forName(pilote);  
            Connection connexion = DriverManager.getConnection(BaseDonnées, "root", "manu");  
            instruction = connexion.prepareStatement(requête);  
        }  
        catch (ClassNotFoundException e) {  
            log("Driver BD non trouvé"); throw new ServletException();  
        }  
        catch (SQLException e) {  
            log("Base de données non trouvée"); throw new ServletException();  
        }  
    }  
}
```

# JOURNALISATION DES ÉVÉNEMENTS DANS LES SERVLETS



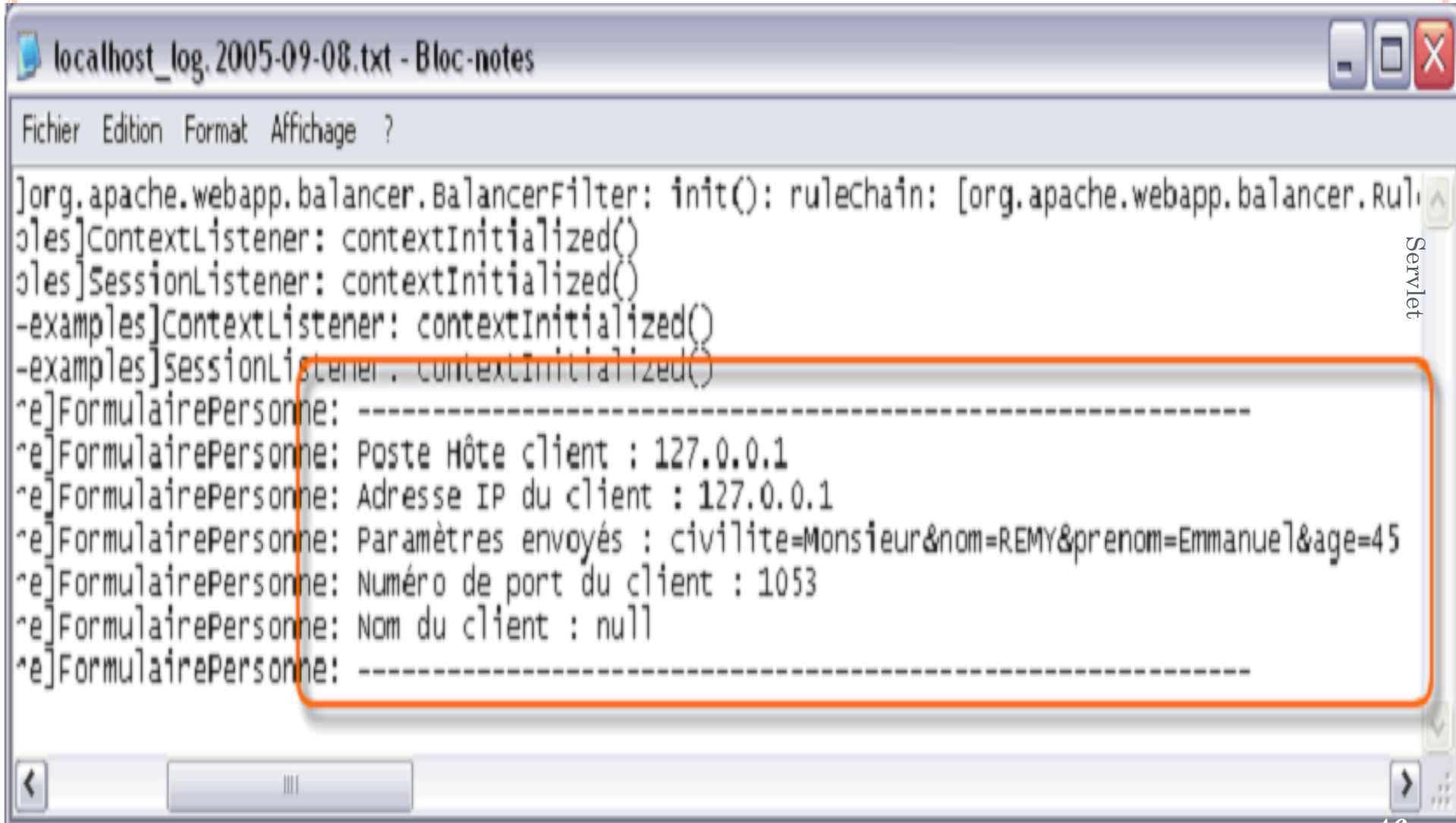
Servlet

# JOURNALISATION DES ÉVÉNEMENTS DANS LES SERVLETS

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    log("-----");
    log("Poste Hôte client : " + request.getRemoteHost());
    log("Adresse IP du client : " + request.getRemoteAddr());
    log("Paramètres envoyés : " + request.getQueryString());
    log("Numéro de port du client : " + request.getRemotePort());
    log("Nom du client : " + request.getRemoteUser());
    log("-----");
    try {
        instruction.setString(1, request.getParameter("nom"));
        instruction.setString(2, request.getParameter("prenom"));
        instruction.setString(3, request.getParameter("civilite"));
        instruction.setInt(4, Integer.parseInt(request.getParameter("age")));
        instruction.executeUpdate();

        response.setContentType("text/html");
        PrintWriter sortie = response.getWriter();
        sortie.println("<HTML><HEAD><TITLE>Réponse Formulaire</TITLE></HEAD><BODY>");
        sortie.println("<H2>Enregistrement de vos coordonnées effectué</H2>");
        sortie.println("</BODY></HTML>");
    }
    catch (SQLException e) { log("Personne non enregistrée"); }
}
```

# JOURNALISATION DES ÉVÉNEMENTS DANS LES SERVLETS

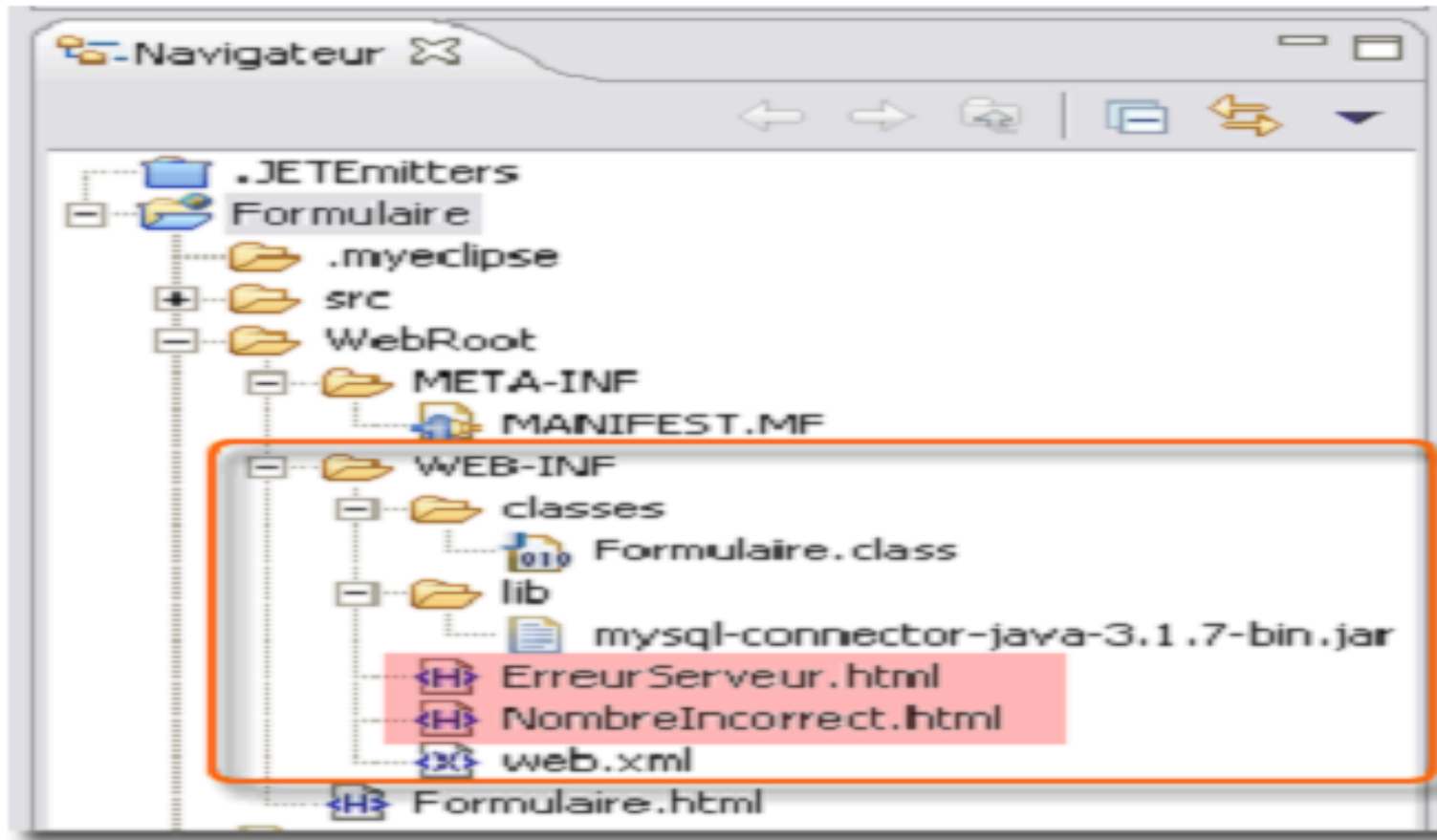


```
localhost_log.2005-09-08.txt - Bloc-notes
Fichier Edition Format Affichage ?

]org.apache.webapp.balancer.BalancerFilter: init(): rulechain: [org.apache.webapp.balancer.Rul
les]ContextListener: contextInitialized()
les]SessionListener: contextInitialized()
-examples]ContextListener: contextInitialized()
-examples]SessionListener: contextInitialized()
^e]FormulairePersonne: -----
^e]FormulairePersonne: Poste Hôte client : 127.0.0.1
^e]FormulairePersonne: Adresse IP du client : 127.0.0.1
^e]FormulairePersonne: Paramètres envoyés : civilite=Monsieur&nom=REMY&prenom=Emmanuel&age=45
^e]FormulairePersonne: Numéro de port du client : 1053
^e]FormulairePersonne: Nom du client : null
^e]FormulairePersonne: -----
```



# GESTION DES EXCEPTIONS - PAGES D'ERREUR



Servlet

# GESTION DES EXCEPTIONS - PAGES D'ERREUR

```
<welcome-file-list>
  <welcome-file>Formulaire.html</welcome-file>
</welcome-file-list>
<context-param>
  <param-name>jdbc.Driver</param-name>
  <param-value>com.mysql.jdbc.Driver</param-value>
</context-param>
<context-param>
  <param-name>localisation</param-name>
  <param-value>jdbc:mysql://localhost/gestion</param-value>
</context-param>
<servlet>
  <display-name>Enregistrement du Personnel</display-name>
  <servlet-name>FormulairePersonne</servlet-name>
  <servlet-class>Formulaire</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>FormulairePersonne</servlet-name>
  <url-pattern>/formulaire</url-pattern>
</servlet-mapping>
<error-page>
  <exception-type>java.lang.NumberFormatException</exception-type>
  <location>/WEB-INF/NombreIncorrect.html</location>
</error-page>
<error-page>
  <error-code>500</error-code>
  <location>/WEB-INF/ErreurServeur.html</location>
</error-page>
</web-app>
```

Servlet



# GESTION DES EXCEPTIONS - PAGES D'ERREUR



Servlet



# LA GESTION DES SESSIONS

- Application Web peut maintenir l'information pour l'ensemble des servlets et des pages JSP au travers de plusieurs transactions au cours de la navigation de l'utilisateur, de sa première requête jusqu'à la fin de la conversation ainsi établie.
- Assurer la continuité à travers une série de page Web est importante dans nombre d'application:
  - la gestion de connexion
  - le suivi des achats dans un caddie

# MÉCANISMES DE GESTION DE SESSION

- Toutes les données constituant une session sont conservées sur le serveur dans l'application Web correspondante.
- Comme plusieurs sessions peuvent exister, le serveur a donc besoin d'un moyen permettant d'associer une session particulière et la requête du client.

# MÉCANISMES DE GESTION DE SESSION

- Il existe deux techniques pour mettre en oeuvre ce principe :
- 1. Les cookies côté client : A la base, un cookie est juste un attribut nom/valeur généré par le serveur et stocké sur le client.
- 2. Réécriture d'URL : Certains utilisateurs n'aiment pas les cookies. Dans ce cas, le serveur doit utiliser une autre technique pour gérer les sessions : la réécriture d'URL.

**public void encodeURL(String urlProchaineServlet)**

# CRÉATION ET UTILISATION DES SESSIONS

```
HttpSession getSession();  
HttpSession getSession(boolean);
```

- Si une session existe déjà, les méthodes **getSession()**, **getSession(true)** et **getSession(false)** renvoient la session existante.
- Dans le cas contraire, **getSession()** et **getSession(true)** créent une nouvelle session.
- Pour supprimer immédiatement une session, on peut utiliser la méthode **invalidate()**.

```
public void invalidate();
```

# EXEMPLE DE GESTION DE SESSION

web.xml Formulaire.html

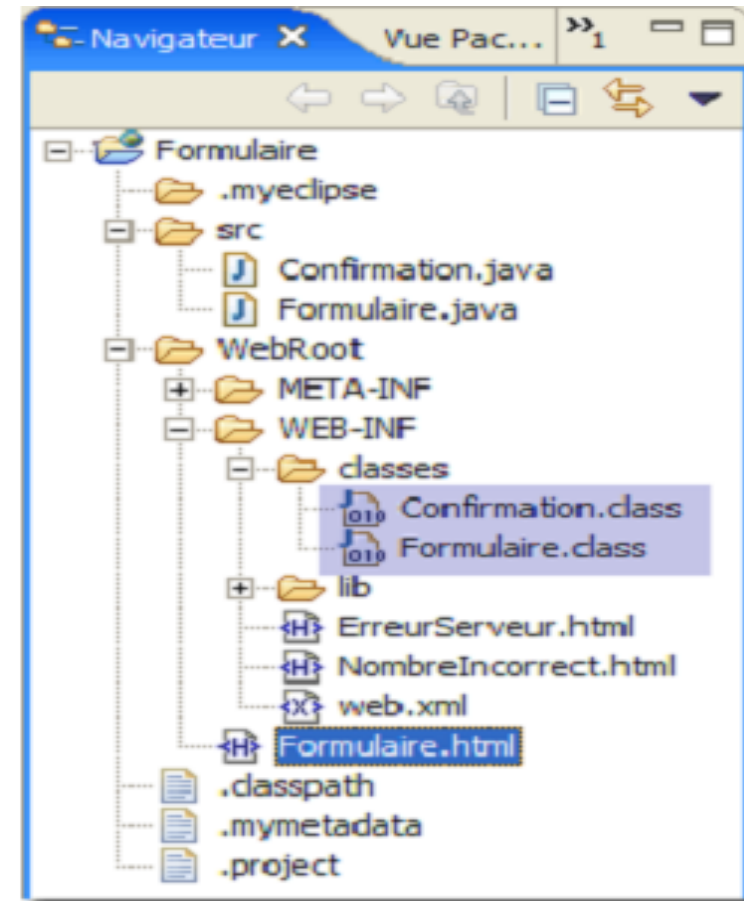
## Enregistrement de vos coordonnées

**Civilité :**

**Nom :**

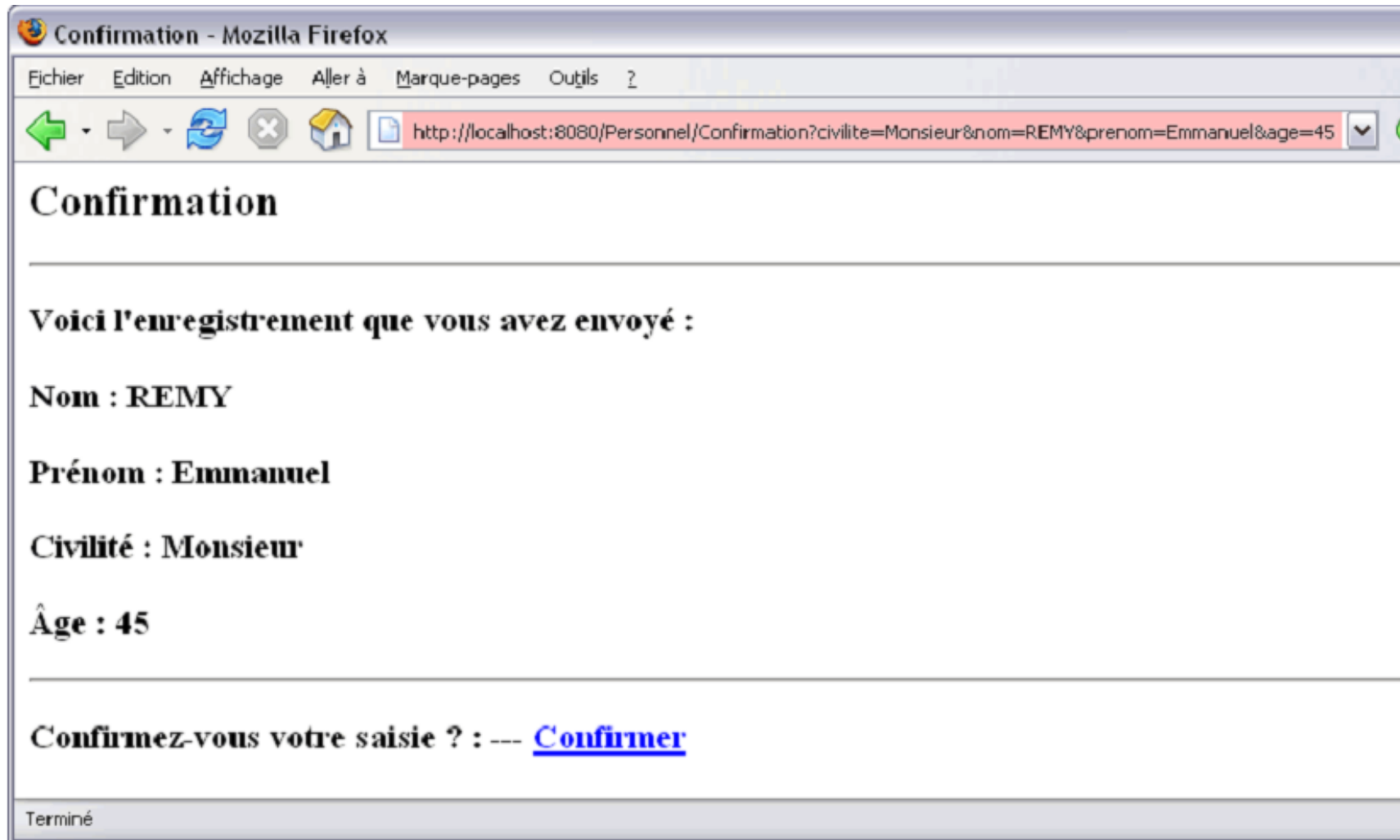
**Prénom :**

**Age :**



Servlet

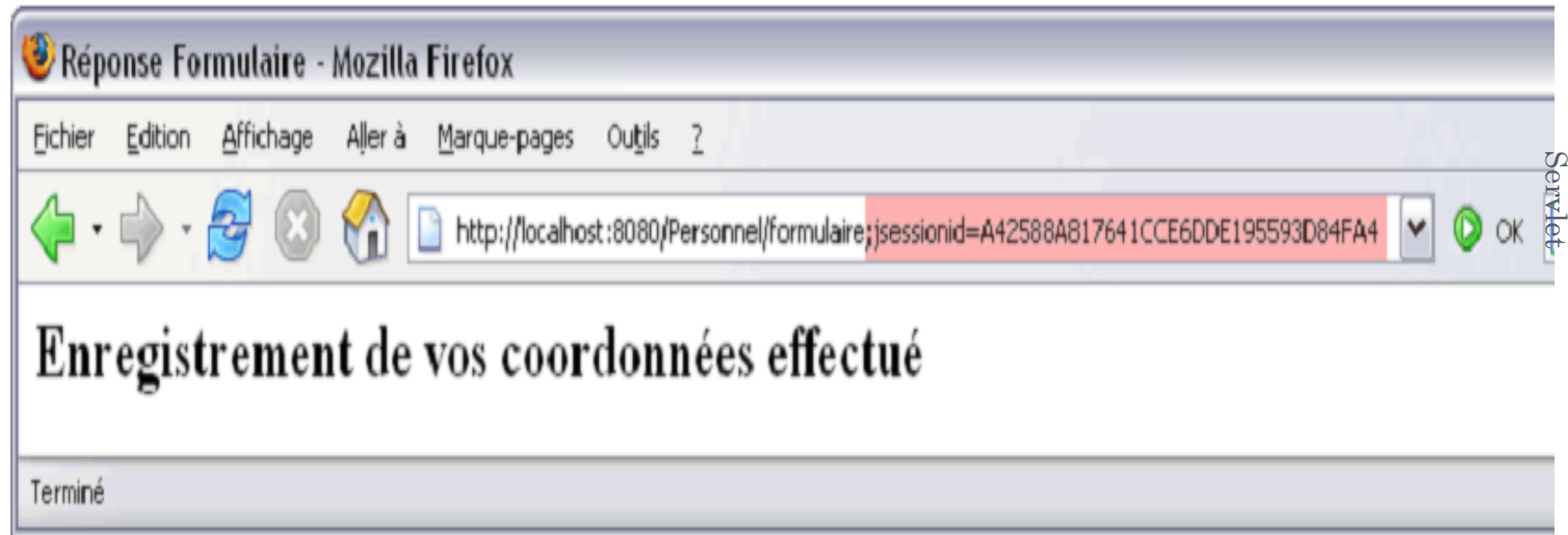
# EXEMPLE DE GESTION DE SESSION



Servlet

Page Web issue de **Confirmation.class**

# EXEMPLE DE GESTION DE SESSION



Page Web issue de **Formulaire.class**



# EXEMPLE DE GESTION DE SESSION

```
*Formulaire.html Confirmation.java Formulaire.java
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>
  <head><title>Formulaire</title>
  </head>

  <body bgcolor="orange" text="yellow">
    <h2>Enregistrement de vos coordonnées</h2><hr>

    <form method="get" action="Confirmation">
      <h3>Civilité : <select name="civilite">
        <option selected>Monsieur</option>
        <option>Madame</option>
        <option>Mademoiselle</option>
      </select></h3>
      <h3>Nom : <input name="nom" size="24"></h3>
      <h3>Prénom : <input name="prenom"></h3>
      <h3>Age : <input name="age" size="5"></h3>
      <hr>
      <input type="submit" value="Envoyer le formulaire">
      <input type="reset" value="Tout effacer">
    </form>

  </body>
</html>
```

Servlet

# EXEMPLE DE GESTION DE SESSION

```
public class Confirmation extends HttpServlet {  
    public void doGet(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
        HttpSession session = request.getSession(true);  
        String nom = request.getParameter("nom");  
        String prénom = request.getParameter("prenom");  
        String civilite = request.getParameter("civilite");  
        String âge = request.getParameter("age");  
        session.setAttribute("nom", nom);  
        session.setAttribute("prenom", prénom);  
        session.setAttribute("civilite", civilite);  
        session.setAttribute("age", âge);  
        String url = response.encodeURL(request.getContextPath()+"/formulaire");  
        response.setContentType("text/html");  
        PrintWriter sortie = response.getWriter();  
        sortie.println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.01 Transitional//EN\">");  
        sortie.println("<HTML><HEAD><TITLE>Confirmation</TITLE></HEAD>");  
        sortie.println("<BODY>");  
        sortie.println("<h2>Confirmaion</h2><hr>");  
        sortie.println("<h3>Voici l'enregistrement que vous avez envoyé :</h3>");  
        sortie.println("<h3>Nom : " + nom + "</h3>");  
        sortie.println("<h3>Prénom : " + prénom + "</h3>");  
        sortie.println("<h3>Civilité : " + civilite + "</h3>");  
        sortie.println("<h3>Âge : " + âge + "</h3><hr>");  
        sortie.println("<h3>Confirmez-vous votre saisie ? : --- ");  
        sortie.println("<a href=\"" + url + "\">Confirmer</a></h3>");  
        sortie.println("</BODY></HTML>");  
        sortie.flush();  
        sortie.close();  
    }  
}
```

# EXEMPLE DE GESTION DE SESSION

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

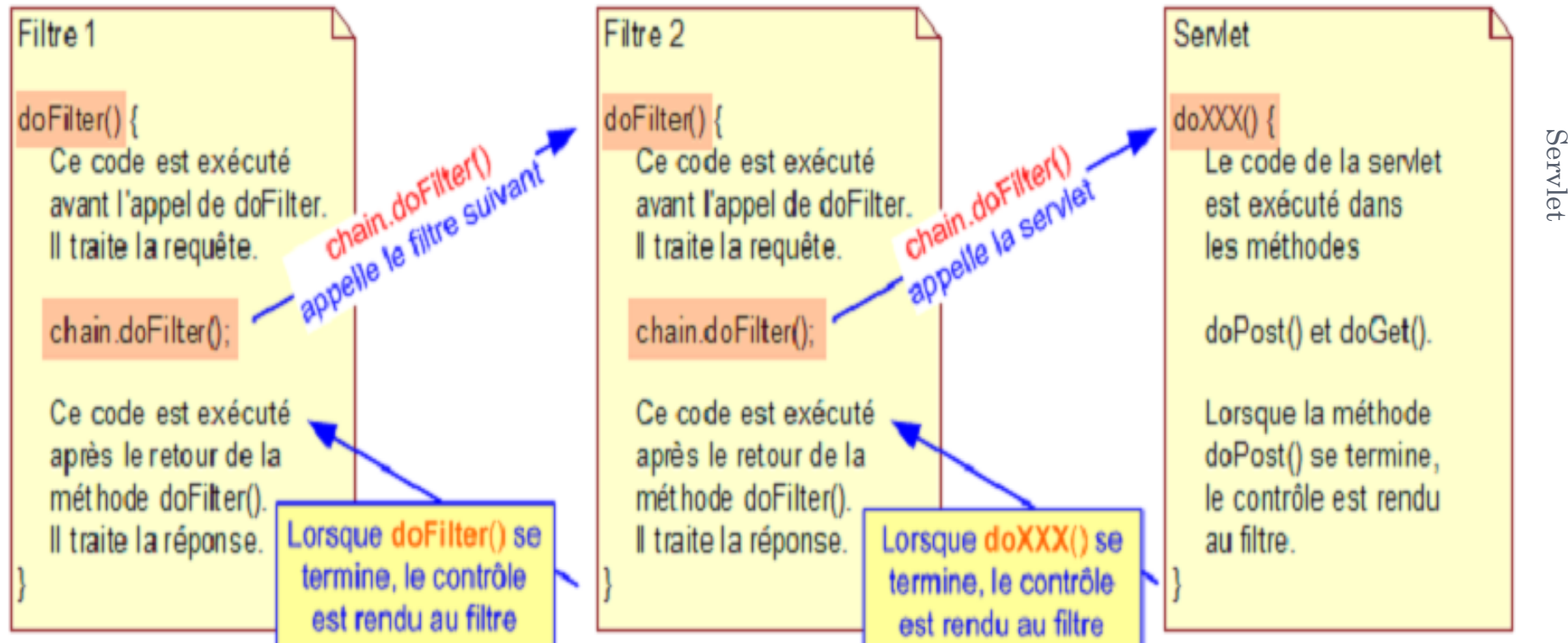
    log("-----");
    log("Poste Hôte client : " + request.getRemoteHost());
    log("Adresse IP du client : " + request.getRemoteAddr());
    log("Paramètres envoyés : " + request.getQueryString());
    log("Numéro de port du client : " + request.getRemotePort());
    log("Nom du client : " + request.getRemoteUser());
    log("-----");
    try {
        response.setContentType("text/html");
        PrintWriter sortie = response.getWriter();
        HttpSession session = request.getSession(false);
        sortie.println("<HTML><HEAD><TITLE>Réponse Formulaire</TITLE></HEAD><BODY>");
        if (session != null) {
            instruction.setString(1, (String)session.getAttribute("nom"));
            instruction.setString(2, (String)session.getAttribute("prenom"));
            instruction.setString(3, (String)session.getAttribute("civilite"));
            instruction.setInt(4, Integer.parseInt((String)session.getAttribute("age")));
            instruction.executeUpdate();
            sortie.println("<H2>Enregistrement de vos coordonnées effectué</H2>");
            sortie.println("</BODY></HTML>");
            session.invalidate();
        } else {
            sortie.println("<H3>Il faut d'abord saisir vos coordonnées : ");
            sortie.println("<a href=\"Formulaire\">Retour</a></h3>");
            sortie.println("</BODY></HTML>");
        }
    }
    catch (SQLException e) { log("Personne non enregistrée"); }
}
```

Servlet

# LES FILTRES DE SERVLET

- Les filtres sont un moyen permettant de donner à une application une structure modulaire
- Ils permettent d'encapsuler différentes tâches annexes qui peuvent être indispensables pour traiter les requêtes.
- par exemple, nous pouvons créer un filtre prévu uniquement pour la journalisation des événements, alors que la servlet principale s'occupe du traitement de la requête.

# STRUCTURATION DES FILTRES



# CRÉATION D'UN FILTRE

- Pour créer un filtre pour notre application Web, nous devons accomplir deux tâches.
- La première consiste à écrire une classe implémentant l'interface Filter.
- la seconde à modifier le descripteur de déploiement de l'application pour indiquer au conteneur qu'il doit utiliser le filtre.



# CRÉATION D'UN FILTRE

- L'API Filter comporte trois interfaces : Filter, FilterChain, et FilterConfig.
- `javax.servlet.Filter` est l'interface implémentée par les filtres. Elle déclare trois méthodes :
  1. `void init (FilterConfig filterConfig)` : appelée par le conteneur pour indiquer au filtre qu'il est mis en service.
  2. `void doFilter (ServletRequest request, ServletResponse response, FilterChain chain)` : appelée par le conteneur chaque fois qu'une paire requête/réponse est traitée pour un client
  3. `void destroy ( )` : est appelée par le conteneur pour indiquer au filtre qu'il est mis hors service.

# DESCRIPTEUR DE DÉPLOIEMENT

```
<filter>  
  <icon>Chemin d'accès à une icône</icon>  
  <filter-name>Le nom du filtre pour l'application Web</filter-name>  
  <display-name>Le nom utilisé par le gestionnaire d'application Web</display-name>  
  <description>Une description de l'application</description>  
  <filter-class>Le nom qualifié de la classe filtre</filter-class>  
  <init-param>  
    <param-name>nom du paramètre</param-name>  
    <param-value>valeur du paramètre</param-value>  
  </init-param>  
</filter>
```

```
<filter-mapping>  
  <filter-name>Même nom que pour l'élément filter</filter-name>  
  <url-pattern>Schéma d'URL auquel le filtre doit être appliqué</url-pattern>  
</filter-mapping>
```

ou

```
<filter-mapping>  
  <filter-name>Même nom que pour l'élément filter</filter-name>  
  <servlet-name>Nom de la servlet auquel le filtre doit être appliqué</servlet-name>  
</filter-mapping>
```



# EXEMPLE ???

```
<filter-mapping>  
  <filter-name>FiltreB</filter-name>  
  <servlet-name>Formulaire</servlet-name>  
</filter-mapping>  
  
<filter-mapping>  
  <filter-name>FiltreA</filter-name>  
  <servlet-name>Formulaire</servlet-name>  
</filter-mapping>
```

Servlet

- Si j'appelle la servlet Formulaire ????

# SETCONTENTTYPE

Type	Meaning
application/msword	Microsoft Word document
application/octet-stream	Unrecognized or binary data
application/pdf	Acrobat (.pdf) file
application/postscript	PostScript file
application/vnd.ms-excel	Excel spreadsheet
application/vnd.ms-powerpoint	Powerpoint presentation
application/x-gzip	Gzip archive
application/x-java-archive	JAR file
application/x-java-vm	Java bytecode (.class) file
application/zip	Zip archive
audio/basic	Sound file in .au or .snd format
audio/x-aiff	AIFF sound file
audio/x-wav	Microsoft Windows sound file
audio/midi	MIDI sound file
text/css	HTML cascading style sheet
text/html	HTML document
text/plain	Plain text
text/xml	XML document
image/gif	GIF image
image/jpeg	JPEG image
image/png	PNG image
image/tiff	TIFF image

## EXAMPLE → Excel

```
public class ApplesAndOranges extends HttpServlet{  
    public void doGet(HttpServletRequest request,  
                      HttpServletResponse response)  
        throws ServletException, IOException{  
        response.setContentType  
            ("application/vnd.ms-excel");  
        PrintWriter out= response.getWriter();  
        out.println("\tQ1\tQ2\tQ3\tQ4\tTotal");  
        out.println("Apples\t78\t87\t92\t29\t=SUM(B2:E2)");  
        out.println("Oranges\t77\t86\t93\t30\t=SUM(B3:E3)");  
    }  
}
```

	A	B	C	D	E	F	G	H
1		Q1	Q2	Q3	Q4	Total		
2	Apples	78	87	92	29	286		
3	Oranges	77	86	93	30	286		
4								
5								
6								