Chapitre
JavaServer Page (JSP)

Java Server Pages (JSP) est une technologie de programmation côté serveur qui permet la création d'une méthode dynamique et indépendante de la plate-forme pour créer des applications Web. JSP a accès à toute la famille des API Java, y compris l'API JDBC pour accéder aux bases de données d'entreprise.

I. Qu'est-ce que les pages JavaServer ?

JavaServer Pages (JSP) est une technologie de développement de pages Web prenant en charge le contenu dynamique. Cela aide les développeurs à insérer du code Java dans les pages HTML en utilisant des balises JSP spéciales, dont la plupart commencent par <% et se terminent par %>.

Un composant JavaServer Pages est un type de servlet Java conçu pour remplir le rôle d'interface utilisateur pour une application Web Java. Les développeurs Web écrivent des JSP sous forme de fichiers texte combinant du code HTML ou XHTML, des éléments XML et des actions et commandes JSP intégrées.

À l'aide de JSP, vous pouvez collecter les commentaires des utilisateurs via des formulaires de page Web, présenter des enregistrements à partir d'une base de données ou d'une autre source et créer des pages Web de manière dynamique.

Les balises JSP peuvent être utilisées à diverses fins, telles que :

- la récupération d'informations à partir d'une base de données ;
- l'enregistrement des préférences utilisateur ;
- l'accès aux composants JavaBeans ;
- le passage du contrôle entre les pages ;
- le partage d'informations entre les requêtes, les pages, etc.

II. JSP - Configuration de l'environnement

Un environnement de développement est l'endroit où vous développerez vos programmes JSP, les testerez et enfin les exécuterez.

- Installer le sdk et configurer la variable d'environnement PATH.
- Installer et configurer le server Tomcat

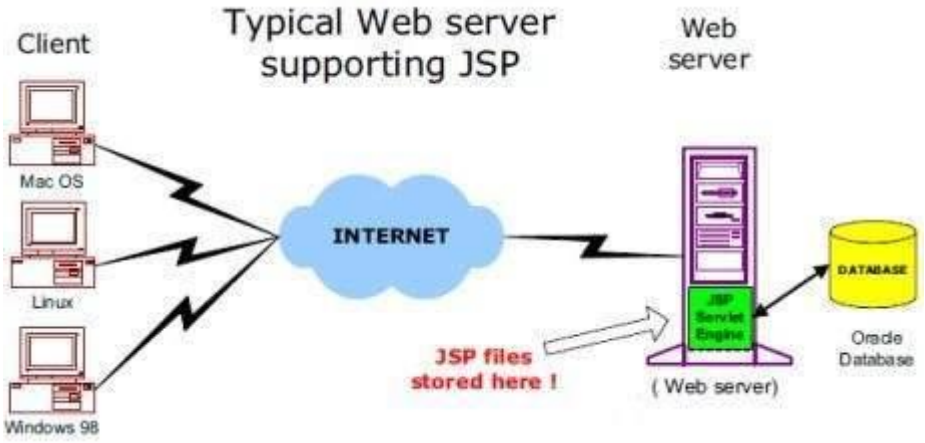
~~Voir TP~~

III. JSP-Architecture

Le serveur Web a besoin d'un moteur JSP, c'est-à-dire d'un conteneur pour traiter les pages JSP. Le conteneur JSP (nous utilisons ici Apache qui possède un conteneur JSP intégré pour prendre en charge le développement de pages JSP) est chargé d'intercepter les demandes de pages JSP.

Un conteneur JSP fonctionne avec le serveur Web pour fournir l'environnement d'exécution et les autres services dont un JSP a besoin. Il sait comprendre les éléments spéciaux qui font partie des JSP.

Le diagramme suivant montre la position du conteneur JSP et des fichiers JSP dans une application Web.

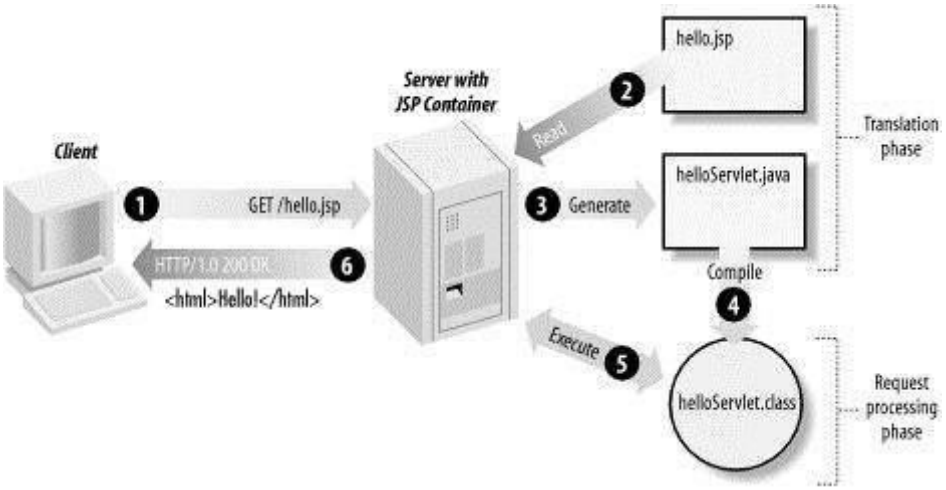


1. Traitement JSP

Les étapes suivantes expliquent comment le serveur Web crée la page Web à l'aide de JSP -

- Comme pour une page normale, votre navigateur envoie une requête HTTP au serveur Web.
- Le serveur Web reconnaît que la requête HTTP concerne une page JSP et la transmet à un moteur JSP. Cela se fait en utilisant l'URL ou la page JSP qui se termine par .jsp au lieu de .html .
- Le moteur JSP charge la page JSP à partir du disque et la convertit en contenu de servlet. Cette conversion est très simple dans laquelle tout le texte du modèle est converti en instructions println() et tous les éléments JSP sont convertis en code Java. Ce code implémente le comportement dynamique correspondant de la page.
- Le moteur JSP compile le servlet dans une classe exécutable et transmet la requête d'origine à un moteur de servlet.
- Une partie du serveur Web appelée moteur de servlet charge la classe Servlet et l'exécute. Lors de l'exécution, la servlet produit une sortie au format HTML. La sortie est ensuite transmise au serveur Web par le moteur de servlet dans une réponse HTTP.
- Le serveur Web transmet la réponse HTTP à votre navigateur sous forme de contenu HTML statique.
- Enfin, le navigateur Web gère la page HTML générée dynamiquement dans la réponse HTTP exactement comme s'il s'agissait d'une page statique.

Toutes les étapes mentionnées ci-dessus peuvent être vues dans le diagramme suivant -



En règle générale, le moteur JSP vérifie si un servlet pour un fichier JSP existe déjà et si la date de modification sur le JSP est antérieure à celle du servlet. Si la JSP est plus ancienne que son servlet généré, le conteneur JSP suppose que la JSP n'a pas changé et

que le servlet généré correspond toujours au contenu de la JSP. Cela rend le processus plus efficace qu'avec les autres langages de script (tels que PHP) et donc plus rapide.
Donc, d'une certaine manière, une page JSP n'est en réalité qu'une autre façon d'écrire un servlet sans avoir à être un expert en programmation Java. À l'exception de la phase de traduction, une page JSP est gérée exactement comme une servlet classique.

IV. JSP-Cycle de vie

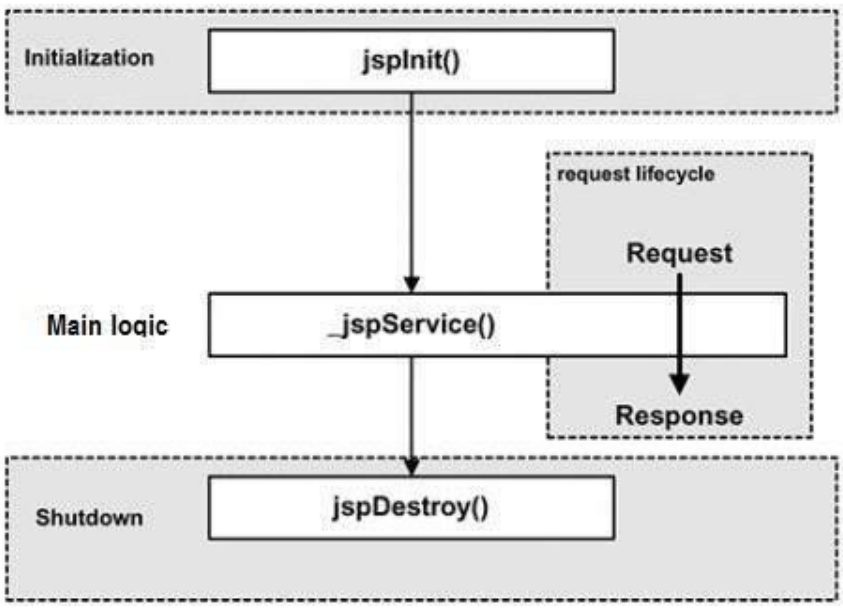
Un cycle de vie JSP est défini comme le processus depuis sa création jusqu'à sa destruction. Ceci est similaire au cycle de vie d'un servlet avec une étape supplémentaire nécessaire pour compiler un JSP en servlet.

1. Chemins suivis par JSP

Voici les chemins suivis par un JSP -

- Compilation
- Initialisation
- Exécution
- Nettoyage

Les quatre phases principales du cycle de vie d'un JSP sont très similaires au cycle de vie d'un servlet. Les quatre phases ont été décrites ci-dessous –



a. Compilation JSP

Lorsqu'un navigateur demande un JSP, le moteur JSP vérifie d'abord s'il doit compiler la page. Si la page n'a jamais été compilée, ou si le JSP a été modifié depuis sa dernière compilation, le moteur JSP compile la page.

Le processus de compilation comporte trois étapes -

- Analyse du JSP.
- Transformer le JSP en servlet.
- Compilation de la servlet.

b. Initialisation JSP

Lorsqu'un conteneur charge un JSP, il appelle la méthode jspInit() avant de répondre à une requête. Si vous devez effectuer une initialisation spécifique à JSP, remplacez la méthode jspInit() -

```
public void jspInit(){
  // Initialisation du code...
}
```

En règle générale, l'initialisation n'est effectuée qu'une seule fois et, comme pour la méthode init du servlet, vous initialisez généralement les connexions à la base de données, ouvrez les fichiers et créez des tables de recherche dans la méthode jspInit.

c. Exécution JSP

Cette phase du cycle de vie du JSP représente toutes les interactions avec les requêtes jusqu'à la destruction du JSP.

Chaque fois qu'un navigateur demande un JSP et que la page a été chargée et initialisée, le moteur JSP appelle la méthode ***_jspService()*** dans le JSP. La méthode ***_jspService()*** prend un HttpServletRequest et un HttpServletResponse comme paramètres comme suit –

```
void _jspService(HttpServletRequest request, HttpServletResponse response) {
        // Execution du code...
}
```

La méthode _jspService() d'un JSP est invoquée sur demande. Ceci est responsable de la génération de la réponse à cette requête et cette méthode est également responsable de la génération des réponses aux sept méthodes HTTP, c'est-à-dire GET, POST, DELETE , etc.

d. Nettoyage JSP

La phase de destruction du cycle de vie d'une JSP représente le moment où une JSP est retirée de son utilisation par un conteneur.

La méthode ***jspDestroy()*** est l'équivalent JSP de la méthode destroy pour les servlets. Remplacez ***jspDestroy*** lorsque vous devez effectuer un nettoyage, tel que la libération des connexions à la base de données ou la fermeture de fichiers ouverts.

La méthode **jspDestroy()** a la forme suivante -

```
public void jspDestroy() {
        // Nettoyage de la servlet.
}
```

V. JSP - Syntaxe

1. Les Éléments de JSP

Les éléments de JSP ont été décrits ci-dessous -

a. Le script

Un scriptlet peut contenir un nombre illimité d'instructions du langage JAVA, de déclarations de variables ou de méthodes, ou d'expressions valides dans le langage de script de page.

Voici la syntaxe de Scriptlet -

<% fragment de code %>

Vous pouvez écrire l'équivalent XML de la syntaxe ci-dessus comme suit -

<jsp:scriptlet>

Fragment de code

</jsp:scriptlet>

Tout texte, balise HTML ou élément JSP que vous écrivez doit se trouver en dehors du scriptlet. Voici le premier exemple simple pour JSP –

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>

      Nous sommes dans le JSP<br />
      <% out.println("Votre adresse ip est : "+request.getRemoteAddr()); %>
      <br />
      <jsp:scriptlet>
            out.println("Votre port Local est : "+request.getLocalPort());
      </jsp:scriptlet>

</body>
</html>
```

REMARQUE -

Parcourez-le en utilisant l'URL http://localhost:8080/index.jsp . Le code ci-dessus générera le résultat suivant –

Nous sommes dans le JSP
Votre adresse ip est : 127.0.0.1
Votre port Local est : 8888

b. Déclarations JSP

Une déclaration déclare une ou plusieurs variables ou méthodes que vous pourrez utiliser dans le code Java ultérieurement dans le fichier JSP. Vous devez déclarer la variable ou la méthode avant de l'utiliser dans le fichier JSP.

Voici la syntaxe des déclarations JSP -

<%! declaration; [declaration;]+ ... %>

Vous pouvez écrire l'équivalent XML de la syntaxe ci-dessus comme suit -

<jsp:declaration>
code fragment
</jsp:declaration>

Voici un exemple de déclarations JSP -

```
<%! int i = 0; %>
<%! int a, b, c; %>
<%! Circle a = new Circle(2.0); %>
```

c. Expression JSP

Un élément d'expression JSP contient une expression de langage de script qui est évaluée, convertie en chaîne et insérée à l'endroit où l'expression apparaît dans le fichier JSP.

Étant donné que la valeur d'une expression est convertie en chaîne, vous pouvez utiliser une expression dans une ligne de texte, qu'elle soit ou non balisée en HTML, dans un fichier JSP.

L'élément expression peut contenir n'importe quelle expression valide selon la spécification du langage Java, mais vous ne pouvez pas utiliser de point-virgule pour terminer une expression.

Voici la syntaxe de l'expression JSP -

<%= expression %>

Vous pouvez écrire l'équivalent XML de la syntaxe ci-dessus comme suit –

<jsp:expression>
code fragment
</jsp:expression>

L'exemple suivant montre une expression JSP –

```
<%@page import="java.util.Date"%>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>JSP - Expression </title>
</head>
<body>
      <h1>Aujourd'hui : <%= (new Date()).toLocaleString() %> </h1>
      <h1>Aujourd'hui :
            <jsp:expression>
                  (new Date()).toLocaleString()
            </jsp:expression>
      </h1>
</body>
```

Le code ci-dessus générera le résultat suivant –

Aujourd'hui : 20 mai 2024, 22:43:00

Aujourd'hui : 20 mai 2024, 22:43:00

d. Commentaires sur JSP

Le commentaire JSP marque le texte ou les instructions que le conteneur JSP doit ignorer. Un commentaire JSP est utile lorsque vous souhaitez masquer ou « commenter » une partie de votre page JSP.

Voici la syntaxe des commentaires JSP -

<%-- This is JSP comment --%>

Il existe un petit nombre de constructions spéciales que vous pouvez utiliser dans divers cas pour insérer des commentaires ou des caractères qui autrement seraient traités spécialement. Voici un résumé –

No.	Syntaxe et objectif
1	<%-- commentaire --%> Un commentaire JSP. Ignoré par le moteur JSP.
2	<!-- commentaire --> Un commentaire HTML. Ignoré par le navigateur.
3	<\% Représente un <% littéral statique.
4	%\> Représente le littéral %> statique.
5	\' Un guillemet simple dans un attribut qui utilise des guillemets simples.
6	\" Un guillemet double dans un attribut qui utilise des guillemets doubles.

e. Directives JSP

Une directive JSP affecte la structure globale de la classe de servlet. Il a généralement la forme suivante -

<%@ directive attribute="value" %>

Exemple :

<%@ page import = "relative url" >

<%@ include file = "relative url" >

<%@ taglib uri = "uri" prefix = "prefixOfTag" >

Vous pouvez écrire l'équivalent XML de la syntaxe ci-dessus comme suit -

<jsp:directive.page attribute = "value" />

<jsp:directive.include file = "relative url" />

<jsp:directive.taglib uri = "uri" prefix = "prefixOfTag" />

Les directives peuvent avoir un certain nombre d'attributs que vous pouvez répertorier sous forme de paires ***clé-valeur*** et séparés par des virgules.

Les espaces entre le symbole @ et le nom de la directive, ainsi qu'entre le dernier attribut et le %> de fermeture, sont facultatifs.

Il existe trois types de balises de directive -

No	Directive et description	Attribut et objectif

1	**<%@ page ... %>** Définit les attributs dépendants de la page, tels que le langage de script, la page d'erreur et les exigences de mise en mémoire tampon.	**buffer** Spécifie un modèle de mise en mémoire tampon pour le flux de sortie
		autoFlush Contrôle le comportement du tampon de sortie du servlet.
		contentType Définit le schéma de codage des caractères.
		errorPage Définit l'URL d'un autre JSP qui signale les exceptions d'exécution Java non vérifiées.
		isErrorPage Indique si cette page JSP est une URL spécifiée par l'attribut errorPage d'une autre page JSP.
		extends Spécifie une superclasse que le servlet généré doit étendre.
		import Spécifie une liste de packages ou de classes à utiliser dans le JSP comme le fait l'instruction d'importation Java pour les classes Java.
		info Définit une chaîne accessible avec la méthode **getServletInfo()** du servlet .
		isThreadSafe Définit le modèle de thread pour le servlet généré.
		language Définit le langage de programmation utilisé dans la page JSP.
		session Spécifie si la page JSP participe ou non aux sessions HTTP
		estELIgnored Spécifie si l'expression EL dans la page JSP sera ignorée ou non.
		isScriptingEnabled Détermine si l'utilisation des éléments de script est autorisée.
2	<%@include ... %> Inclut un fichier pendant la phase de traduction.	
3	<%@taglib ... %> Déclare une bibliothèque de balises, contenant des actions personnalisées, utilisée dans la page	

Exemple avec la directive include

```
<%@page import="java.util.Date"%>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%!
   int pageCount = 0;
   void addCount() {
      pageCount++;
   }
%>
<% addCount(); %>

<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>DIRECTIVE INCLUDE</title>
</head>
<body>
      <p>Ce site a été visité : <%= pageCount %> fois.</p>
      <p>Copyright © 2024</p>
</body>
</html>
```

Exercice :

Utiliser ce code et créer 3 fichiers header.jsp, main.jsp et footer.jsp que vous inclurez dans ce fichier index.jsp

f. Les Actions JSP

Les actions JSP utilisent des constructions en syntaxe XML pour contrôler le comportement du moteur de servlet. Vous pouvez insérer dynamiquement un fichier, réutiliser des composants JavaBeans, rediriger l'utilisateur vers une autre page ou générer du HTML pour le plugin Java.

Il n'y a qu'une seule syntaxe pour l'élément Action, car elle est conforme à la norme XML -

<jsp:action_name attribute="value" />

Les éléments d'action sont essentiellement des fonctions prédéfinies. Le tableau suivant répertorie les actions JSP disponibles -

No.	Syntaxe et objectif
1	**jsp:include** Inclut un fichier au moment où la page est demandée.
2	**jsp:useBean** Recherche ou instancie un JavaBean.
3	**jsp:setProperty** Définit la propriété d'un JavaBean.
4	**jsp:getProperty** Insère la propriété d'un JavaBean dans la sortie.
5	**jsp:forward** Redirige le demandeur vers une nouvelle page.
6	**jsp:plugin** Génère du code spécifique au navigateur qui crée une balise OBJECT ou EMBED pour le plugin Java.
7	**jsp:element** Définit les éléments XML de manière dynamique.
8	**jsp:attribute** Définit l'attribut de l'élément XML défini dynamiquement.
9	**jsp:body** Définit le corps de l'élément XML défini dynamiquement.
10	**jsp:text** Utilisé pour écrire du texte de modèle dans les pages et les documents JSP.

Il existe deux attributs communs à tous les éléments Action :

- L'attribut **id** : L'attribut id identifie de manière unique l'élément Action et permet à l'action d'être référencée dans la page JSP. Si l'action crée une instance d'un objet, la valeur id peut être utilisée pour la référencer via l'objet implicite PageContext.

- L'attribut **scope** : Cet attribut identifie le cycle de vie de l'élément Action. L'attribut id et l'attribut scope sont directement liés, car l'attribut scope détermine la durée de vie de l'objet associé à l'id. L'attribut scope a quatre valeurs possibles : **(a) page, (b)request, (c)session** et **(d) application** .

Exemple :

```java
package cm.dassy.servlet;

public class ActionJSP {
	private String message = "Ceci est un message";

	public String getMessage() {
		return(message);
	}

	public void setMessage(String message) {
		this.message = message;
	}
}
```

```jsp
<%@page import="java.util.Date"%>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>

<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>ACTIONS JSP</title>
</head>
<body>
	<center>
		<h2>Utilisation d'un JavaBeans dans JSP</h2>
		<jsp:useBean id = "test" class = "cm.dassy.servlet.ActionJSP" />
		<jsp:setProperty name = "test" property = "message"
			value = "Bonjour JSP..." />

		<p>Envois du message....</p>
		<jsp:getProperty name = "test" property = "message" />
	</center>
</body>
</html>
```

Resultat :

Utilisation d'un JavaBeans dans JSP

Envois du message....

Bonjour JSP...

g. Déclarations de flux de contrôle

Vous pouvez utiliser toutes les API et éléments de base de Java dans votre programmation JSP, y compris les instructions de prise de décision, les boucles, etc.

- **Déclarations décisionnelles**

Le bloc if...else commence comme un Scriptlet ordinaire, mais le Scriptlet est fermé à chaque ligne avec du texte HTML inclus entre les balises Scriptlet.

```
<%@page import="java.util.Date"%>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>JSP - IF...ELSE</title>
</head>
<body>
	<%! int age = 213; %>
	<% if(age > 0 && age < 18) {%>
		<p>Vous êtes mineur</p>
	<%} else if(age > 18 && age < 100) {%>
		<p>Vous êtes majeur</p>
	<%} else {%>
		<p>Vous m'inquiété</p>
	<%} %>
</body>
</html>
```

Exercice :

L'utilisateur doit saisir un âge que vous devez recuperer, cet âge doit être compris entre 0 et 100 et il n'a droit qu'à 3 essais. Le faire pour le for, while, switch, do..while

h. Opérateurs JSP

JSP prend en charge tous les opérateurs logiques et arithmétiques pris en charge par Java. Le tableau suivant répertorie tous les opérateurs avec la priorité la plus élevée qui apparaissent en haut du tableau, ceux avec la priorité la plus basse apparaissent en bas. Dans une expression, les opérateurs de priorité supérieure seront évalués en premier.

Catégorie	Opérateur	Associativité
Postfix	( ) [ ] .(opérateur point)	De gauche à droite
Unaire	++ - - ! ~	De droite à gauche
Multiplicatif	* / %	De gauche à droite
Additif	+ -	De gauche à droite
Changement	>> >>> <<	De gauche à droite
Relationnel	> >= < <=	De gauche à droite
Égalité	== !=	De gauche à droite
ET au niveau du bit	&	De gauche à droite
XOR au niveau du bit	^	De gauche à droite
OU au niveau du bit	|	De gauche à droite
ET logique	&&	De gauche à droite
OU logique	||	De gauche à droite
Conditionnel	?:	De droite à gauche
Affectation	= += -= *= /= %= >>= <<= &= ^= |=	De droite à gauche
Virgule	,	De gauche à droite

i. Littéraux JSP

Le langage d'expression JSP définit les littéraux suivants -

- Booléen − vrai et faux
- Entier - comme en Java
- Virgule flottante - comme en Java
- String - avec des guillemets simples et doubles ; " est échappé comme \", ' est échappé comme \' et \ est échappé comme \\.
- Nul - nul

VI. JSP - Demande du client

Lorsqu'un navigateur demande une page Web, il envoie de nombreuses informations au serveur Web. Ces informations ne peuvent pas être lues directement car elles font partie de l'en-tête de la requête HTTP.

Le tableau suivant répertorie les informations d'en-tête importantes provenant du navigateur. Ces informations sont fréquemment utilisées dans la programmation Web -

No.	En-tête et description
1	**Accept** Cet en-tête spécifie les types MIME que le navigateur ou d'autres clients peuvent gérer. Les valeurs image/png ou image/jpeg sont les deux possibilités les plus courantes.
2	**Accept-Charset** Cet en-tête spécifie les jeux de caractères que le navigateur peut utiliser pour afficher les informations. Par exemple, ISO-8859-1 .
3	**Accept-Encoding** Cet en-tête précise les types d'encodages que le navigateur sait gérer. Les valeurs de gzip ou compress sont les deux possibilités les plus courantes.
4	**Accept-Language** Cet en-tête spécifie les langues préférées du client au cas où le servlet pourrait produire des résultats dans plusieurs langues. Par exemple en, en-us, ru , etc.
5	**Autorization** Cet en-tête est utilisé par les clients pour s'identifier lorsqu'ils accèdent à des pages Web protégées par mot de passe.
6	**Connection** Cet en-tête indique si le client peut gérer les connexions HTTP persistantes. Les connexions persistantes permettent au client ou à un autre navigateur de récupérer plusieurs fichiers avec une seule requête. La valeur Keep-Alive signifie que des connexions persistantes doivent être utilisées.
7	**Content-Length** Cet en-tête s'applique uniquement aux requêtes POST et donne la taille des données POST en octets.
8	**Cookie** Cet en-tête renvoie les cookies aux serveurs qui les envoyaient précédemment au navigateur.
9	**Host** Cet en-tête spécifie l'hôte et le port tels qu'indiqués dans l'URL d'origine.
10	**If-Modified-Since** Cet en-tête indique que le client souhaite la page uniquement si elle a été modifiée après la date spécifiée. Le serveur envoie un code, 304, qui signifie en-tête Non modifié si aucun résultat plus récent n'est disponible.
11	**If-Unmodified-Since** Cet en-tête est l'inverse de If-Modified-Since ; il précise que l'opération ne doit réussir que si le document est plus ancien que la date spécifiée.
12	**Referer** Cet en-tête indique l'URL des pages Web référentes. Par exemple, si vous êtes sur la page Web 1 et cliquez sur un lien vers la page Web 2, l'URL de la page Web 1 est incluse dans l'en-tête Referer lorsque le navigateur demande la page Web 2.
13	**User-Agent** Cet en-tête identifie le navigateur ou un autre client effectuant la demande et peut être utilisé pour renvoyer différents contenus à différents types de navigateurs.

1. L'objet HttpServletRequest

L'objet de requête est une instance d'un objet javax.servlet.http.HttpServletRequest . Chaque fois qu'un client demande une page, le moteur JSP crée un nouvel objet pour représenter cette demande.

L'objet de requête fournit des méthodes pour obtenir des informations d'en-tête HTTP, notamment les données de formulaire, les cookies, les méthodes HTTP , etc. Le tableau suivant répertorie les méthodes importantes qui peuvent être utilisées pour lire l'en-tête HTTP dans votre programme JSP. Ces méthodes sont disponibles avec l'objet HttpServletRequest qui représente la requête du client au serveur Web.

No.	Méthode et description
1	**Cookie[] getCookies()** Renvoie un tableau contenant tous les objets Cookie que le client a envoyés avec cette requête.
2	**Enumeration getAttributeNames()** Renvoie une énumération contenant les noms des attributs disponibles pour cette requête.
3	**Enumeration getHeaderNames()** Renvoie une énumération de tous les noms d'en-tête que contient cette requête.
4	**Enumeration getParameterNames()** Renvoie une énumération d'objets String contenant les noms des paramètres contenus dans cette requête.
5	**HttpSession getSession()** Renvoie la session en cours associée à cette requête, ou si la requête n'a pas de session, en crée une.
6	**HttpSession getSession (boolean create)** Renvoie la HttpSession actuelle associée à cette requête ou, s'il n'y a pas de session en cours et que create est vrai, renvoie une nouvelle session.
7	**Local getLocale()** Renvoie la langue préférée dans laquelle le client acceptera le contenu, en fonction de l'en-tête Accept-Language.
8	**Objet getAttribute (String name)** Renvoie la valeur de l'attribut nommé en tant qu'objet, ou null si aucun attribut du nom donné n'existe.
9	**ServletInputStream getInputStream()** Récupère le corps de la requête sous forme de données binaires à l'aide d'un ServletInputStream.
10	**String getAuthType()** Renvoie le nom du schéma d'authentification utilisé pour protéger le servlet, par exemple « BASIC » ou « SSL », ou null si le JSP n'était pas protégé.
11	**String getCharacterEncoding()** Renvoie le nom du codage de caractères utilisé dans le corps de cette requête.
12	**String getContentType()** Renvoie le type MIME du corps de la requête, ou null si le type n'est pas connu.

13	**String getContextPath()** Renvoie la partie de l'URI de la demande qui indique le contexte de la demande.
14	**String getHeader (String name)** Renvoie la valeur de l'en-tête de requête spécifié sous forme de chaîne.
15	**String getMethod()** Renvoie le nom de la méthode HTTP avec laquelle cette requête a été effectuée, par exemple GET, POST ou PUT.
16	**String getParameter (String name)** Renvoie la valeur d'un paramètre de requête sous forme de chaîne, ou null si le paramètre n'existe pas.
17	**String getPathInfo()** Renvoie toutes les informations de chemin supplémentaires associées à l'URL envoyée par le client lors de cette demande.
18	**String getProtocol()** Renvoie le nom et la version du protocole utilisé par la requête.
19	**String getQueryString()** Renvoie la chaîne de requête contenue dans l'URL de la demande après le chemin.
20	**String getRemoteAddr()** Renvoie l'adresse IP (Internet Protocol) du client qui a envoyé la demande.
21	**String getRemoteHost()** Renvoie le nom complet du client qui a envoyé la demande.
22	**String getRemoteUser()** Renvoie le login de l'utilisateur faisant cette demande, si l'utilisateur a été authentifié, ou null si l'utilisateur n'a pas été authentifié.
23	**String getRequestURI()** Renvoie la partie de l'URL de cette requête depuis le nom du protocole jusqu'à la chaîne de requête dans la première ligne de la requête HTTP.
24	**String getRequestedSessionId()** Renvoie l'ID de session spécifié par le client.
25	**String getServletPath()** Renvoie la partie de l'URL de cette requête qui appelle le JSP.
26	**String[] getParameterValues (String name)** Renvoie un tableau d'objets String contenant toutes les valeurs du paramètre de requête donné, ou null si le paramètre n'existe pas.
27	**boolean isSecure()** Renvoie un booléen indiquant si cette requête a été effectuée à l'aide d'un canal sécurisé, tel que HTTPS.
28	**int getContentLength()** Renvoie la longueur, en octets, du corps de la requête et mise à disposition par le flux d'entrée, ou -1 si la longueur n'est pas connue.
29	**int getIntHeader (String name)** Renvoie la valeur de l'en-tête de requête spécifié sous forme de entier.
30	**int getServerPort()** Renvoie le numéro de port sur lequel cette demande a été reçue.

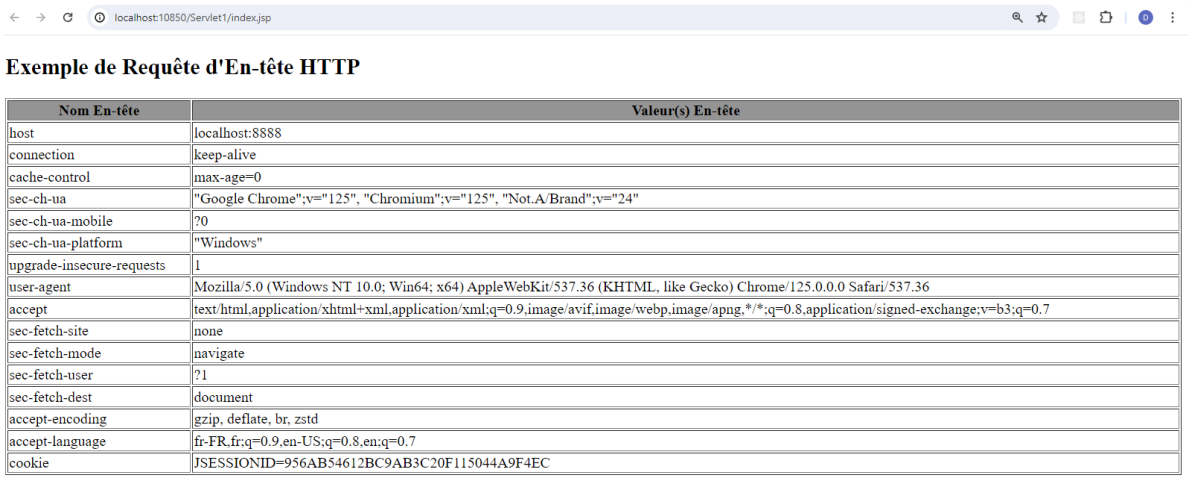
Exemple :

```jsp
<%@page import="java.util.Enumeration"%>
<%@page import="java.util.Date"%>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>

<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>JSP - HTTP Request</title>
</head>
<body>
      <h2>Exemple de Requête d'En-tête HTTP</h2>
      <table width = "100%" border = "1" align = "center">
            <tr bgcolor = "#949494">
                  <th>Nom En-tête</th>
                  <th>Valeur(s) En-tête</th>
            </tr>
         <%
            Enumeration headerNames = request.getHeaderNames();
            while(headerNames.hasMoreElements()) {
               String paramName = (String)headerNames.nextElement();
               out.print("<tr><td>" + paramName + "</td>\n");
               String paramValue = request.getHeader(paramName);
               out.println("<td> " + paramValue + "</td></tr>\n");
            }
         %>
      </table>

</body>
</html>
```

Mettons maintenant le code ci-dessus dans main.jsp et essayons d'y accéder.



VII. JSP – Réponse du serveur

Lorsqu'un serveur Web répond à une requête HTTP, la réponse se compose généralement d'une ligne d'état, de certains en-têtes de réponse, d'une ligne vide et du document. Une réponse typique ressemble à ceci –

```
HTTP/1.1 200 OK
Content-Type: text/html
Header2: ...
...
HeaderN: ...
  (Blank Line)
<!doctype ...>

<html>
  <head>...</head>
  <body>
    ...
  </body>
</html>
```

La ligne d'état est constituée de la version HTTP (HTTP/1.1 dans l'exemple) , d'un code d'état (200 dans l'exemple) et d'un message très court correspondant au code d'état (OK dans l'exemple) .

Voici un résumé des en-têtes de réponse HTTP 1.1 les plus utiles qui retournent au navigateur à partir du serveur Web. Ces en-têtes sont fréquemment utilisés dans la programmation Web -

No.	En-tête et description
1	**Allow** Cet en-tête spécifie les méthodes de requête (GET, POST , etc.) prises en charge par le serveur.
2	**Cache-Control** Cet en-tête spécifie les circonstances dans lesquelles le document de réponse peut être mis en cache en toute sécurité. Il peut avoir des valeurs publiques, privées ou sans cache, etc. Public signifie que le document peut être mis en cache, Private signifie que le document est destiné à un seul utilisateur et ne peut être

	stocké que dans des caches privés (non partagés) et sans cache signifie que le document ne doit jamais être mis en cache.
3	**Connexion** Cet en-tête indique au navigateur s'il doit utiliser ou non des connexions HTTP persistantes. La valeur close indique au navigateur de ne pas utiliser de connexions HTTP persistantes et keep-alive signifie utiliser des connexions persistantes.
4	**Content-Disposition** Cet en-tête vous permet de demander au navigateur de demander à l'utilisateur de sauvegarder la réponse sur le disque dans un fichier du nom donné.
5	**Content-Encoding** Cet en-tête précise la manière dont la page a été codée lors de la transmission.
6	**Contenu-Language** Cet en-tête indique la langue dans laquelle le document est rédigé. Par exemple, en, en-us, ru, etc.
7	**Content-Length** Cet en-tête indique le nombre d'octets dans la réponse. Ces informations ne sont nécessaires que si le navigateur utilise une connexion HTTP persistante (keep-alive).
8	**Content-Type** Cet en-tête donne le type MIME (MultiPurpose Internet Mail Extension) du document de réponse.
9	**Expires** Cet en-tête précise l'heure à laquelle le contenu doit être considéré comme obsolète et ainsi ne plus être mis en cache.
10	**Last-Modified** Cet en-tête indique la date à laquelle le document a été modifié pour la dernière fois. Le client peut ensuite mettre en cache le document et fournir une date par un en-tête de requête If-Modified-Since dans les requêtes ultérieures.
11	**Location** Cet en-tête doit être inclus avec toutes les réponses qui ont un code d'état dans les 300. Cela informe le navigateur de l'adresse du document. Le navigateur se reconnecte automatiquement à cet emplacement et récupère le nouveau document.
12	**Refresh** Cet en-tête spécifie dans quel délai le navigateur doit demander une page mise à jour. Vous pouvez spécifier le temps en nombre de secondes après lequel une page sera actualisée.
13	**Retry-After** Cet en-tête peut être utilisé conjointement avec une réponse 503 (Service non disponible) pour indiquer au client dans combien de temps il peut répéter sa demande.
14	**Set-Cookie** Cet en-tête spécifie un cookie associé à la page.

1. L'objet HttpServletResponse

L'objet de réponse est une instance d'un objet **javax.servlet.http.HttpServletResponse**.

Tout comme le serveur crée l'objet de requête, il crée également un objet pour représenter la réponse au client. L'objet de réponse définit également les interfaces chargées de créer de nouveaux en-têtes HTTP. Grâce à cet objet, le programmeur JSP peut ajouter de nouveaux cookies ou horodatages, codes d'état HTTP, etc.

Les méthodes suivantes peuvent être utilisées pour définir l'en-tête de réponse HTTP dans votre programme de servlet. Ces méthodes sont disponibles avec l'objet **HttpServletResponse** . Cet objet représente la réponse du serveur.

No.	Méthode et description
1	**String encodeRedirectURL (String URL)** Encode l'URL spécifiée à utiliser dans la méthode sendRedirect ou, si le codage n'est pas nécessaire, renvoie l'URL inchangée.
2	**String encodeURL (String URL)** Encode l'URL spécifiée en y incluant l'ID de session ou, si le codage n'est pas nécessaire, renvoie l'URL inchangée.
3	**boolean containsHeader (String name)** Renvoie un booléen indiquant si l'en-tête de réponse nommé a déjà été défini.
4	**boolean isCommitted()** Renvoie un booléen indiquant si la réponse a été validée.
5	**void addCookie (cookie cookie)** Ajoute le cookie spécifié à la réponse.
6	void addDateHeader (String name, long date) Ajoute un en-tête de réponse avec le nom et la valeur de date donnés.
7	void addHeader (String name, String value) Ajoute un en-tête de réponse avec le nom et la valeur donnés.
8	**void addIntHeader (String name, int value)** Ajoute un en-tête de réponse avec le nom donné et une valeur entière.
9	**void flushBuffer()** Force l'écriture de tout contenu du tampon sur le client.
10	**void reset()** Efface toutes les données qui existent dans le tampon ainsi que le code d'état et les en-têtes.
11	**void resetBuffer()** Efface le contenu du tampon sous-jacent dans la réponse sans effacer les en-têtes ou le code d'état.
12	**void sendError(int sc)** Envoie une réponse d'erreur au client en utilisant le code d'état spécifié et en effaçant le tampon.
13	**void sendError (int sc, String msg)** Envoie une réponse d'erreur au client en utilisant l'état spécifié.
14	**void sendRedirect (String location)** Envoie une réponse de redirection temporaire au client à l'aide de l'URL de l'emplacement de redirection spécifié.
15	**void setBufferSize (int size)** Définit la taille de tampon préférée pour le corps de la réponse.
16	**void setCharacterEncoding (String charset)**

	Définit le codage des caractères (jeu de caractères MIME) de la réponse envoyée au client, par exemple sur UTF-8.
17	**void setContentLength(int len)** Définit la longueur du corps du contenu dans la réponse Dans les servlets HTTP ; cette méthode définit également l'en-tête HTTP Content-Length.
18	**void setContentType (String type)** Définit le type de contenu de la réponse envoyée au client, si la réponse n'a pas encore été validée.
19	**void setDateHeader (String name, long date)** Définit un en-tête de réponse avec le nom et la valeur de date donnés.
20	**void setHeader (String name, String value)** Définit un en-tête de réponse avec le nom et la valeur donnés.
21	**void setIntHeader (String name, int value)** Définit un en-tête de réponse avec le nom donné et une valeur entière.
22	**void setLocale (Locale loc)** Définit les paramètres régionaux de la réponse, si la réponse n'a pas encore été validée.
23	**void setStatus(int sc)** Définit le code d'état pour cette réponse.

Exemple de réponse d'en-tête HTTP

Exercice : Que chaque étudiant en écrive une réponse d'en-tête HTTP

VIII. JSP - Traitement des formulaires

Dans ce chapitre, nous aborderons le traitement des formulaires dans JSP. Vous devez avoir rencontré de nombreuses situations dans lesquelles vous devez transmettre certaines informations de votre navigateur au serveur Web et finalement à votre programme backend. Le navigateur utilise deux méthodes pour transmettre ces informations au serveur Web. Ces méthodes sont la méthode GET et la méthode POST.

1. Les méthodes de traitement des formulaires

a. La Méthode GET

La méthode GET envoie les informations utilisateur codées ajoutées à la demande de page. La page et les informations codées sont séparées par le caractère ? comme suit -

http://localhost:10850/Servlet1/index.jsp?key1=value1&key2=value2

La méthode GET est la méthode par défaut pour transmettre les informations du navigateur au serveur Web et elle produit une longue chaîne qui apparaît dans la Location:box de votre navigateur . Il est recommandé de ne pas utiliser la méthode GET si vous avez un mot de passe ou d'autres informations sensibles à transmettre au serveur.

La méthode GET a une limitation de taille : seuls 1 024 caractères peuvent figurer dans une chaîne de requête .

Ces informations sont transmises à l'aide de l'en-tête QUERY_STRING et seront accessibles via la variable d'environnement QUERY_STRING qui peut être gérée à l'aide des méthodes getQueryString() et getParameter() de l'objet de requête.

b. La Méthode POST

Une méthode généralement plus fiable pour transmettre des informations à un programme backend est la méthode POST.

Cette méthode regroupe les informations exactement de la même manière que la méthode GET, mais au lieu de les envoyer sous forme de chaîne de texte après un ? dans l'URL, il l'envoie sous forme de message séparé. Ce message arrive au programme backend sous la forme d'une entrée standard que vous pouvez analyser et utiliser pour votre traitement.

JSP gère ce type de requêtes en utilisant la méthode getParameter() pour lire des paramètres simples et la méthode getInputStream() pour lire le flux de données binaires provenant du client.

2. Lecture des données du formulaire à l'aide de JSP

JSP gère automatiquement l'analyse des données du formulaire en utilisant les méthodes suivantes en fonction de la situation -

- getParameter() - Vous appelez la méthode request.getParameter() pour obtenir la valeur d'un paramètre de formulaire.
- getParameterValues() - Appelez cette méthode si le paramètre apparaît plus d'une fois et renvoie plusieurs valeurs, par exemple une case à cocher.
- getParameterNames() - Appelez cette méthode si vous souhaitez une liste complète de tous les paramètres de la requête en cours.
- getInputStream() - Appelez cette méthode pour lire le flux de données binaires provenant du client.

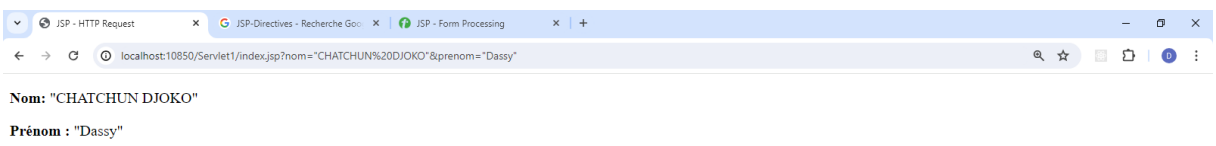
Exemple de méthode GET utilisant une URL

L'URL suivante transmettra deux valeurs au programme HelloForm à l'aide de la méthode GET.

http://localhost:10850/Servlet1/index.jsp?nom="CHATCHUN DJOKO"&prenom="Dassy"

```
<%@page import="java.util.Enumeration"%>
<%@page import="java.util.Date"%>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>

<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>JSP - HTTP Request</title>
</head>
<body>
      <p><b>Nom:</b>
            <%= request.getParameter("nom")%>
      </p>
      <p><b>Prénom :</b>
            <%= request.getParameter("prenom")%>
      </p><
</body>
</html>
```



Exemple de méthode GET utilisant un formulaire

Voici un exemple qui transmet deux valeurs à l'aide du FORMULAIRE HTML et du bouton Soumettre.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
	<form action="index.jsp" method="get">
		Nom :  <input type="text" name="nom"><br />
		Ville :<input type="text" name="ville"><br />
		Sexe :		<input type="radio" name="sexe" value="M.">M.
				<input type="radio" name="sexe" value="Mme.">Mme.
				<input type="radio" name="sexe" value="Mlle.">Mlle.<br
/>
		Votre diplôme :	<select name="diplome" multiple size=2>
					<option value="Probatoire">Probatoire</option>
					<option value="Bacc+2">Bacc+2</option>
					<option value="Licence">Licence</option>
					<option value="Master">Master</option>
				</select><br />
				<input type="submit" value="Envoyer">
	</form>
</body>
</html>
```

TD : Recupérer tous les paramètres et les afficher

IX. JSP - Accès à la base de données

Avant de commencer à accéder à la base de données via un JSP, assurez-vous d'avoir configuré correctement l'environnement JDBC ainsi qu'une base de données.

Pour commencer avec le concept de base, créons :

- Une base de données
- une table
- et créons quelques enregistrements dans cette table

Exercice : Connecter son fichier jsp à votre base de données