JSP : Partie I

Introduction

Java**S**erver **P**ages (JSP) est une technologie de programmation côté serveur qui permet de créer des applications Web dynamiques et multiplate-formes. Elle permet d'accéder à toute la gamme d'API Java, y compris **JDBC**, pour se connecter aux bases de données d'entreprise. Utilisons alors JSP dans le développement de vos applications Web.

Étant basées sur l'API Java Servlets, les pages JSP bénéficient de toutes les fonctionnalités avancées des API Java Enterprise, telles que JDBC, JNDI, EJB, JAXP, etc.

De plus, les JSP peuvent être utilisées en complément des servlets pour gérer la logique métier, suivant le modèle soutenu par les moteurs de templates pour servlets Java.

JSP et Java EE

JSP fait partie intégrante de Java EE, une plateforme complète pour le développement d'applications d'entreprise. Cela permet à JSP d'être utilisé aussi bien pour des applications simples que pour des projets complexes et exigeants.

Applications de JSP

Comme mentionné, JSP est largement utilisé dans le développement Web. Voici quelques exemples de son utilisation :

- **JSP et ASP (Active Server Pages)** : JSP présente deux grands avantages. Premièrement, la partie dynamique est écrite en Java, au lieu de Visual Basic ou d'un autre langage spécifique à Microsoft, ce qui la rend plus puissante et plus intuitive. De plus, JSP est portable et peut fonctionner sur différents systèmes d'exploitation et serveurs Web, y compris ceux non Microsoft.
- **JSP vs Servlets Pures** : Il est souvent plus facile d'écrire et de maintenir du HTML classique avec JSP plutôt que de devoir écrire de nombreuses instructions println pour générer le HTML dans des servlets.
- **JSP et Inclusions Côté Serveur (SSI)** : Les inclusions côté serveur sont principalement conçues pour des tâches simples, comme l'intégration de fragments de code HTML. Elles ne permettent pas de créer de véritables programmes qui nécessitent, par exemple, de traiter des données de formulaires ou de se connecter à des bases de données.
- **JSP et JavaScript** : JavaScript peut générer du HTML dynamique côté client, mais il ne peut pas interagir facilement avec le serveur pour des tâches complexes comme la gestion de bases de données ou le traitement d'images. JSP, en revanche, est conçu pour gérer ces interactions côté serveur.
- **JSP vs HTML Statique** : Le HTML classique ne permet pas d'afficher de manière dynamique les informations. JSP, en revanche, peut générer des pages dynamiques basées sur les données traitées sur le serveur.

Qu'est-ce que JavaServer Pages ?

JavaServer Pages (JSP) est une technologie utilisée pour développer des pages Web dynamiques. Elle permet aux développeurs d'intégrer du code Java directement dans des pages HTML à l'aide de balises spéciales, généralement encadrées par **<%** et **%>.**

Un composant **JSP** est un type de **servlet** Java qui sert de couche d'interface utilisateur pour une application Web Java. Les développeurs créent des JSP sous forme de fichiers texte, combinant du code HTML ou XHTML, des éléments XML, ainsi que des actions et commandes JSP intégrées.

Avec JSP, il est possible de :

- Collecter les données des utilisateurs via des formulaires Web.
- Afficher des enregistrements provenant d'une base de données ou d'autres sources.
- Générer des pages Web de manière dynamique.

Les balises JSP sont polyvalentes et peuvent être utilisées pour diverses tâches telles que :

- Récupérer des informations depuis une base de données ou enregistrer les préférences d'un utilisateur.
- Accéder à des composants JavaBeans.
- Transférer le contrôle entre différentes pages.
- Partager des informations entre les requêtes et les pages.

Éléments de JSP

Les éléments de JSP incluent plusieurs composants clés, décrits ci-dessous :

Le Scriptlet

Un scriptlet est un bloc de code qui peut contenir des instructions Java, des déclarations de variables ou de méthodes, ainsi que des expressions valides dans le langage Java.

La syntaxe d'un scriptlet est la suivante :

<% Portion de code **%>**

L'équivalent XML de cette syntaxe est :

```
<jsp:scriptlet>
  Portion de code
</jsp:scriptlet>
```

Tout contenu de texte, balises HTML ou éléments JSP doit être placé en dehors du **scriptlet**.

Exemple simple de page JSP :

```
<html>
  <head><title>Hello World</title></head>
  <body>
    Hello World!<br/>
    <%
      out.println("Votre Addresse IP est " + request.getRemoteAddr());
    %>
  </body>
</html>
```

Dans cet exemple, le message "Hello World!" s'affiche, suivi de l'adresse IP de l'utilisateur, récupérée par le code Java inséré dans le scriptlet.

Déclarations JSP

Une déclaration en JSP permet de définir une ou plusieurs variables ou méthodes qui peuvent être utilisées plus tard dans le code Java de la page JSP. Il est nécessaire de déclarer ces variables ou méthodes avant de les utiliser.

Syntaxe des Déclarations JSP :

```
<%! declaration;  %>
```

L'équivalent XML de cette syntaxe est le suivant :

```
<jsp:declaration>
  Portion de code
</jsp:declaration>
```

Exemple de Déclarations JSP :

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

Expression JSP

Un élément d'expression en JSP contient une expression Java qui est évaluée, convertie en chaîne de caractères, puis insérée à l'endroit où elle apparaît dans le fichier JSP.

Puisque la valeur de l'expression est transformée en chaîne de caractères, elle peut être utilisée dans le texte d'une page, qu'elle soit ou non encadrée par des balises HTML.

Syntaxe de l'Expression JSP :

<%= expression **%>**

L'équivalent XML est :

```
<jsp:expression>
  expression
```

</jsp:expression>

Exemple d'Expression JSP :

```
<html>
  <head><title>A Comment Test</title></head>
  <body>
    <p>Date d’aujourd’hui est : <%= (new java.util.Date()).toLocaleString() %></p>
  </body>
</html>
```

Ce code affichera quelque chose comme :

Date d’aujourd’hui suivi la date d’aujourd’hui

Commentaires JSP

Les commentaires en JSP permettent de masquer des instructions ou du texte que le conteneur JSP doit ignorer. Cela peut être utile pour désactiver temporairement une partie du code dans une page JSP.

Syntaxe des Commentaires JSP :

```
<%-- Ceci est un commentaire --%>
```

Exemple de Commentaires JSP :

```
<html>
  <head><title>A Comment Test</title></head>
  <body>
    <h2>A Test of Comments</h2>
    <%-- Ceci est un commentaire donc ne sera visible dans le resultat--%>
  </body>
</html>
```

Le commentaire inclus ne sera pas visible dans le code source de la page, offrant ainsi un moyen de documenter ou de masquer des portions de code dans votre fichier JSP.

Objets implicites

En JSP, les objets implicites sont des objets prédéfinis qui sont directement disponibles dans les pages JSP sans nécessiter de déclaration explicite. Ils facilitent le développement en permettant d'accéder à divers objets couramment utilisés, comme les objets de requête HTTP, de session, de réponse, et bien plus encore.

Voici une liste des principaux objets implicites de JSP avec des explications et des exemples :

1. request

- L'objet request est une instance de HttpServletRequest qui contient les informations de la requête HTTP faite par le client, telles que les paramètres de la requête, les en-têtes, et les attributs.

Exemple :

```
<html>
  <body>
    <p>Your IP address is: <%= request.getRemoteAddr() %></p>
    <p>Request Method: <%= request.getMethod() %></p>
  </body>
</html>
```

Dans cet exemple, request.getRemoteAddr() récupère l'adresse IP du client, et request.getMethod() affiche la méthode de la requête (comme GET ou POST).

2. response

- L'objet response est une instance de HttpServletResponse et est utilisé pour envoyer des réponses au client, comme définir les en-têtes HTTP ou rediriger vers une autre page.

Exemple :

```
<%
  response.setContentType("text/html");
  response.setHeader("Refresh", "5; URL=http://example.com");
%>
<html>
  <body>
    <p>This page will redirect to example.com in 5 seconds.</p>
  </body>
</html>
```

Ici, l'en-tête Refresh est défini pour rediriger le client vers une autre URL après 5 secondes.

3. out

- L'objet out est une instance de JspWriter et est utilisé pour envoyer des données de sortie au client. Il est souvent utilisé pour afficher des messages et générer du contenu HTML.

Exemple :

```
<%
  out.println("Hello, this is a message using the out object.");
%>
```

Cela affiche le message directement sur la page Web.

4. session

- L'objet session est une instance de HttpSession qui permet de stocker et de récupérer des données utilisateur entre les différentes pages visitées par un même utilisateur pendant sa session.

Exemple :

```
<%
  session.setAttribute("user", "John Doe");
%>
<html>
  <body>
    <p>Welcome, <%= session.getAttribute("user") %></p>
  </body>
</html>
```

Cet exemple stocke le nom de l'utilisateur dans la session, puis l'affiche sur la page.

5. application

- L'objet application est une instance de **ServletContext** et permet de partager des données entre différentes sessions et utilisateurs. Les données sont accessibles par toutes les pages JSP de l'application.

Exemple :

```
<%
  application.setAttribute("siteName", "My Awesome Site");
%>
<html>
  <body>
    <p>Welcome to <%= application.getAttribute("siteName") %>!</p>
  </body>
</html>
```

Ici, le nom du site est stocké dans l'objet application et peut être récupéré sur n'importe quelle page JSP de l'application.

6. config

- L'objet config est une instance de **ServletConfig** et fournit des informations d'initialisation spécifiques à une servlet.

Exemple :

```
<%
  String servletName = config.getServletName();
%>
<p>Servlet Name: <%= servletName %></p>
```

Ce code affiche le nom de la servlet associée à la page JSP.

7. pageContext

- L'objet pageContext est utilisé pour accéder à d'autres objets comme request, response, session, application, etc., et pour gérer les attributs au niveau de la page.

Exemple :

```
<%
  pageContext.setAttribute("message", "Hello, JSP!");
%>
<p><%= pageContext.getAttribute("message") %></p>
```

Cet exemple définit un attribut message au niveau de la page et le récupère pour l'afficher.

8. page

- L'objet page fait référence à la page JSP elle-même, similaire à **this** en Java. Il est rarement utilisé directement dans les JSP.

Exemple :

```
<%
  out.println("This is the page object: " + page);
%>
```

Ici, page fait référence à l'instance de la page courante.

9. exception

- L'objet exception est disponible uniquement dans les pages JSP qui ont la directive **isErrorPage**="true". Il contient les informations sur l'exception qui a été levée.

Exemple :

```
<%@ page isErrorPage="true" %>
<html>
  <body>
    <p>Error: <%= exception.getMessage() %></p>
  </body>
</html>
```

Ce code affiche le message de l'exception sur une page de gestion des erreurs.

Les directives

En JSP (JavaServer Pages), les directives sont des instructions spéciales qui fournissent des informations au conteneur JSP sur la manière de traiter la page. Les trois directives les plus couramment utilisées sont page, include, et taglib. Chacune a un rôle spécifique dans la configuration et la gestion des pages JSP.

1. Directive page

La directive page est utilisée pour définir des propriétés pour la page JSP, telles que le type de contenu, les imports de classes Java, la gestion des erreurs, etc. Elle doit être placée en haut de la page JSP.

Syntaxe :

```
<%@ page attribute="value" %>
```

Exemples d'attributs :

- **contentType** : Définit le type de contenu de la réponse (par exemple, text/html).
- **import** : Permet d'importer des classes Java nécessaires pour la page.
- **errorPage** : Spécifie une page JSP qui doit gérer les erreurs pour cette page.

Exemple :

```
<%@ page contentType="text/html;charset=UTF-8" language="java" import="java.util.*"
errorPage="error.jsp" %>
<html>
  <head>
    <title>My Page</title>
  </head>
  <body>
    <h1>Welcome to My Page!</h1>
  </body>
</html>
```

Dans cet exemple, le type de contenu est défini, la classe java.util.* est importée, et error.jsp sera utilisé pour gérer les erreurs.

2. Directive include

La directive include permet d'inclure le contenu d'un fichier dans une autre page JSP au moment de la compilation. Cela est utile pour réutiliser des composants communs, comme des en-têtes ou des pieds de page.

Syntaxe :

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

Exemple :

```
<%@ include file="header.jsp" %>
<html>
  <body>
    <h1>Page Title</h1>
    <p>This is the content of the page.</p>
  </body>
</html>
```

Dans cet exemple, header.jsp est inclus au début de la page, ce qui permet de centraliser le code d'en-tête.

3. Directive taglib

La directive taglib est utilisée pour déclarer une bibliothèque de balises personnalisées dans une page JSP. Cela permet d'utiliser des balises personnalisées définies dans des fichiers .tld (Tag Library Descriptor) pour simplifier le développement.

Syntaxe :

```
<%@ taglib uri="uri-of-taglib" prefix="prefix" %>
```

Exemple :

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
  <body>
    <c:if test="${not empty user}">
      <p>Welcome, ${user}!</p>
    </c:if>
    <c:choose>
```

```
      <c:when test="${user == 'admin'}">
        <p>You have admin access.</p>
      </c:when>
      <c:otherwise>
        <p>You do not have admin access.</p>
      </c:otherwise>
    </c:choose>
  </body>
</html>
```

Dans cet exemple, la bibliothèque JSTL (JavaServer Pages Standard Tag Library) est incluse, permettant l'utilisation de balises pour des opérations conditionnelles et des expressions.

Gestion des Exception

En JSP (JavaServer Pages), la gestion des exceptions est cruciale pour assurer la robustesse et la fiabilité des applications web. Les exceptions peuvent survenir pour diverses raisons, telles que des erreurs de syntaxe, des problèmes de connexion à une base de données ou des erreurs de traitement des données. Voici un aperçu de la gestion des exceptions dans JSP, y compris les types d'exceptions, comment les gérer et les bonnes pratiques.

Types d'Exceptions en JSP

1. **Exceptions Non Vérifiées** : Ce sont des exceptions qui ne sont pas contrôlées par le compilateur, comme NullPointerException, ArrayIndexOutOfBoundsException, etc. Elles se produisent généralement en raison d'erreurs logiques dans le code.
2. **Exceptions Vérifiées** : Ce sont des exceptions qui doivent être déclarées ou gérées, comme IOException et SQLException. Elles nécessitent un bloc try-catch pour être gérées.

Gestion des Exceptions dans JSP

1. Directive errorPage

La directive errorPage est utilisée pour spécifier une page JSP qui doit être affichée en cas d'exception. Cela permet de centraliser le traitement des erreurs et d'améliorer l'expérience utilisateur.

Syntaxe :

```
<%@ page errorPage="error.jsp" %>
```

Exemple :

```
<%@ page errorPage="error.jsp" %>
<html>
  <body>
    <%
      // Un code qui pourrait provoquer une exception
      String value = null;
      out.println(value.length()); // Ceci provoquera une NullPointerException
    %>
```

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

Dans cet exemple, si une exception se produit, l'utilisateur sera redirigé vers error.jsp.

2. Page de Gestion des Erreurs

La page spécifiée dans la directive errorPage peut récupérer l'objet exception qui contient des informations sur l'erreur.

Exemple de error.jsp :

```
<%@ page isErrorPage="true" %>
<html>
  <body>
    <h1>Une erreur s'est produite</h1>
    <p>Message d'erreur : <%= exception.getMessage() %></p>
    <p><a href="index.jsp">Retour à l'accueil</a></p>
  </body>
</html>
```

Dans cet exemple, error.jsp affiche le message d'erreur et offre un lien pour retourner à la page d'accueil.

3. Utilisation des Blocs try-catch

Pour gérer les exceptions vérifiées, vous pouvez utiliser des blocs try-catch dans le code Java intégré.

Exemple :

```
<%
  try {
    // Code susceptible de provoquer une exception
    Class.forName("com.mysql.jdbc.Driver");
    Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/mydb", "user",
"password");
  } catch (SQLException e) {
    out.println("Erreur de connexion à la base de données : " + e.getMessage());
  } catch (ClassNotFoundException e) {
    out.println("Pilote JDBC introuvable : " + e.getMessage());
  }
%>
```

Ici, les exceptions de connexion à la base de données sont gérées explicitement.

Bonnes Pratiques pour la Gestion des Exceptions en JSP

1. **Centraliser la Gestion des Erreurs** : Utilisez une page d'erreur centralisée pour gérer les exceptions, ce qui permet de fournir une expérience utilisateur cohérente.
2. **Évitez de Gérer les Exceptions dans les JSP** : La logique de traitement des exceptions devrait être déléguée à des servlets ou à d'autres classes Java. Cela garde les pages JSP simples et centrées sur la présentation.
3. **Utilisez des Messages d'Erreur Utilisateur Amicaux** : Évitez de montrer des messages d'erreur techniques aux utilisateurs. Fournissez des messages clairs et utiles.

4. **Journalisation des Erreurs** : Consignez les erreurs dans un fichier journal pour aider à la détection et à la résolution des problèmes sans exposer les détails aux utilisateurs.
5. **Tester les Scénarios d'Erreur** : Assurez-vous de tester les différentes situations qui peuvent provoquer des exceptions pour vérifier que votre gestion des erreurs fonctionne comme prévu.