

Einführung in JavaServerPages (JSP) (verfasst von Hannes Restel)

JSP erlaubt das Erstellen dynamischer Websites, indem serverseitig Java-Code in eine HTML-Seite eingebaut und interpretiert wird. Der Client (z.B. Benutzer eines Webshops) erhält dann nur eine statische HTML-Seite.

Zum Verständnis: JSP-Dateien werden beim Kompilieren in eine JavaServlet umgewandelt. Eine JSP ist also nichts anderes als ein Servlet mit ausgelagerten HTML-Tags, was leichter zu lesen ist. Zusätzlich wird das drumherum wie main-Methode, Servlet-Einbindung usw. direkt vom Compiler (hier: Ants) automatisch generiert. Deshalb ist folgende einfache Struktur möglich:

Typisches Beispiel:

„beispiel.jsp“:

```
<html>
  <head> ... </head>
  <body>
    <HTML-Tag> ... </HTML-Tag Ende>
    <%@ ...Direktiven-Code... <%>
    <%! ein paar Deklarationen %>
    zwischendrin normaler HTML-Text
    <%= JavaAusdrücke interpretiert als HTML %>
    <% ganz normaler Java-Code <%>
  </body>
</html>
```

Es finden zwei Interpretier-Schritte bzw. Kompilier-Schritte statt:

- 1) Der Server interpretiert eine JSP mittels einer JSP-Engine und erzeugt daraus statischen HTML-Code
- 2) der Browser des Clients empfängt diese „normale“ HTML-Seite und interpretiert diese wie sonst auch

Es gibt zwei Modelle:

Model One Architecture:

Client schickt Daten an Server -> JSP wertet Daten aus und schickt statische HTML-Seite an Client

Model Two Architecture (Trennung von: Datenauswertung und grafischem Design)

Client schickt Daten an Server -> Auswertung erfolgt von einem Controller (meist ein Java-Servlet), welches die relevanten Daten dann zur graphischen Umsetzung an eine JSP schickt.

-> Client enthält statischen HTML-Code von der JSP

Model2Architecture ist für größere Projekte geeignet, da Datenabstraktion betrieben wird.

JSP-Syntax:

Es gibt verschiedene Arten von Code in einer JSP-Datei:

- 1) Direktiven: (Befehle/Botschaften für die JSP-Engine) (keine Erstellung von „sichtbarem Code“)
- 2) Deklarationen: (globale Variablen, Methoden) (besser als JavaBeans implementieren)
- 3) Ausdrücke (Expressions): (einfache Output-Variablen oder JavaBeans.Get-Methoden)
- 4) Scriptlets (am wichtigsten): normaler Java-Code mit HTML-Code
- 5) Kommentare: (ausgezeichnete JSP-Kommentare; nicht vom Client einsehbar)

Spezifischer JSP-Code wird immer von `<% ... %>` umschlossen.

Die verschiedenen Arten des Codes bekommen nach dem Prozentzeichen noch ein weiteres ausgezeichnetes Zeichen!

(ab hier ist alles aus wikipedia kopiert und ein wenig abgewandelt!)

Direktiven

Eine Direktive dient zum Übermitteln von speziellen Seiteninformationen an den JSP-Compiler; dadurch kann man angeben, ob die JSP eine Taglib einbindet oder wie im Fehlerfall weiter zu verfahren ist.

Die allgemeine Syntax für eine Direktive ist `<%@ ... %>`. Folgende Direktiven sind vorhanden:

- **include**, diese Direktive weist den JSP-Compiler an, den vollständigen Inhalt einer externen Datei in die Originaldatei zu kopieren.

```
<%@ include file="BeispielDatei.ext" %>
```

- **page**
 - *import*, ein Java-Import-Statement wird in der Datei generiert
 - *contentType*, gibt die Art des Datei-Inhaltes an. Sollte dann eingesetzt werden, wenn man kein HTML benutzt oder den Default-Zeichensatz nicht verwendet
 - *errorPage*, gibt die Seite an, die im Fehlerfall angezeigt werden soll
 - *isErrorPage*, gibt an ob diese Seite eine Error-Page ist oder nicht, wenn ja ist das exception Objekt verfügbar
 - *isThreadSafe*, gibt an ob das aus der JSP generierte Servlet threadsicher ist oder nicht

```
<%@ page import="java.util.*" %> //import
<%@ page contentType="text/html" %> //contentType
<%@ page isErrorPage=false %> //die Seite ist keine Error-Page
<%@ page isThreadSafe=true %> //eine threadsichere JSP
```

- **tagLib**, diese Direktive gibt an, dass eine Taglib verwendet werden soll. Es muss ein Prefix und eine [URI](#) für die Taglib vergeben werden.

```
<%@ taglib prefix="MeinPrefix" uri="taglib/MeineTagLib.tld" %>
```

Skriptelemente

Standardvariablen

Die folgenden Variablen können in jeder JSP verwendet werden.

- out, JSPWriter, der die Daten in den HTTP-Response-Stream schreibt
- page, das Servlet selber
- pageContext, eine Instanz des PageContext, welche die Daten der gesamten Seite enthält
- request, das HTTP-Request-Objekt
- response, das HTTP-Response-Objekt
- session, das HTTP-Session-Objekt. Es kann dazu benutzt werden, Information über den Benutzer von einem Request zum nächsten weiterzureichen.

Skriptelemente

Es gibt drei grundlegende Skriptelemente, die es erlauben, Java-Code direkt in das Servlet einzufügen.

- Ein Tag, der es erlaubt, Code in die Klasse einzufügen. Dieser Tag kann dazu verwendet werden, Daten der Klasse festzulegen.

```
<%! int serverInstanceVariable = 1;%>
```

- Ein Tag, der es erlaubt, Code in die _jspService() Methode des resultierenden Servlets einzufügen.

```
<% int localStackBasedVariable = 1; %>
```

- Ein Tag, der es erlaubt, Code zu expandieren und direkt in die HTTP-Antwort zu schreiben. Das Semikolon wird hier nicht benutzt, da der Code als Ausdruck ausgewertet wird.

```
<%= "expanded inline data " + 1 %>
```

Deklarationen

Deklarationen dienen zur Definition von [Variablen](#) und [Methoden](#), die von anderen Elementen in der JSP verwendet werden können. Deklarationen erzeugen keine Ausgabe innerhalb der JSP.

```
<%! int variableMeinerKlasse = 0; %>
```

Dies definiert eine „static“ Variable, die nur einmal im Arbeitsspeicher abgelegt wird! Zeitgleiche Aufrufe aus verschiedenen Instanzen greifen also auf dieselben Daten zu. Möchte man aber eine „VariableMeinerKlasse“ haben, dessen Wert in jeder Instanz meiner Klasse einen eigenen Wert hat, dann muß man eine Variable wie gewohnt in den normalen <% %>-Tags definieren:

```
<% String meinName; meinName=theReult.getString("name"); ... %>
```

Ausdrücke

Ausdrücke (expressions) werden dazu verwendet, Variablen oder Methoden direkt in den [HTML](#)- oder [XML](#)-Ausgabestrom zu integrieren.

```
Die Klassenvaribale ist <%= variableMeinerKlasse %>
```

Skriptlets

JSP-Skriptlets können zur Implementierung der Ablauflogik sowie der Erzeugung der [HTML](#)- oder [XML](#)-Ausgabe eingesetzt werden. Der Skriptlet-Code wird innerhalb der _jspService() Methode des generierten Servlets eingefügt.

```
<% int variable = 0; out.println("Der Wert der Variable ist : " + variable); %>
```

Kommentare

Kommentare sind nur innerhalb der originalen JSP sichtbar, sie werden nicht in den Ausgabestrom geschrieben.

```
<%-- Kommentar innerhalb einer JSP --%>
```

Aktionen

JSP-Aktionen sind [XML](#)-Tags die die eingebaute Funktionalität von Webservern einbinden. Die folgenden Aktionen sind verfügbar:

- **jsp:include**, die angegebene JSP wird vom Java-Servlet aufgerufen, dabei wird der Request und der Response übergeben. Ist die angegebene JSP abgearbeitet, kommt die Steuerung zur gegenwärtigen JSP zurück. Diese JSP-Aktion bewirkt, dass JSP-Code zwischen zwei JSPs geteilt anstatt kopiert wird.

```
<jsp:include page="mycommon.jsp">
  <jsp:param name="extraparam" value="myvalue"/>
</jsp:include>
```

- **jsp:param**, definiert einen Parameter, der zu den Request-Parametern hinzugefügt wird. Diese Aktion kann innerhalb eines jsp: include oder jsp: forward Blocks verwendet werden.
- **jsp:forward**, der Request und der Response wird an eine andere JSP oder ein [Servlet](#) übergeben. Die Steuerung kommt nicht zur gegenwärtigen JSP zurück.

```
<jsp:forward page="subpage.jsp">
  <jsp:param name="forwardedFrom" value="this.jsp"/>
</jsp:forward>
```

- **jsp:plugin**, diese Aktion generiert je nach verwendetem [Browser](#) einen Tag zum Einbinden eines Java-Applets. Dies wird benötigt, da in den älteren Versionen der [Browser](#) von Netscape (Navigator) und Microsoft (Internet Explorer) verschiedene Tags zum Einbinden eines [Applets](#) verwendet werden.

```
<jsp:plugin type="applet" height="100%" width="100%"
archive="myjarfile.jar,myotherjar.jar"
codebase="/applets"
code="com.foo.MyApplet">
  <jsp:params>
    <jsp:param name="enableDebug" value="true"/>
  </jsp:params>
  <jsp:fallback>
    Your browser does not support applets.
  </jsp:fallback>
</jsp:plugin>
```

- **jsp:fallback**, definiert den Inhalt der angezeigt wird, falls der Browser keine Applets unterstützt.
- **jsp:setProperty**, diese Aktion setzt ein Property in der definierten [Java-Bean](#).

```
<jsp:setProperty name="myBean" property="lastChanged" value="<%= new Date() %>" />
```

- **jsp:getProperty**, diese Aktion holt sich ein Property von der definierten Java-Bean.

```
<jsp:getProperty name="myBean" property="lastChanged" />
```

- **jsp:useBean**, diese Aktion erstellt oder verwendet eine Java-Bean wieder. Das Attribut *scope* gibt an wie lange Attribute zur Verfügung stehen. Folgende Werte können definiert werden:

- *request* – Attribute sind nur solange verfügbar, wie der Request existiert.
- *page* – Attribute sind nur für die gegenwärtige JSP verfügbar
- *session* – Attribute sind nur solange verfügbar, wie die Benutzer-Session existiert
- *application* – Attribute sind immer verfügbar

```
<jsp:useBean id="myBean" class="com.foo.MyBean" scope="request"/>
```