JSP (Java Server Pages) technológia

JSP technológia

A JSP technológiával könnyen készíthető olyan web-tartalom, melynek **statikus** és **dinamikus** része van.

A JSP –

- rendelkezésre bocsátja a szervletek dinamikus tulajdonságait
- jóval természetesebb módon áll hozzá a statikus tartalom létrehozásához (mint a szervlet)

Egy JSP egy szöveges dokumentum, amely kétféle szöveget tartalmaz:

- statikus adatok, melyek bármilyen szöveges formátumúak lehetnek (HTML, SVG, WML, XML),
- JSP elemek, amelyek a dinamikus tartalmat hozzák létre

A JSP elemek kétféle szintaxissal használhatók:

- standard
- XML

Egy oldalon belül csak az egyiket használhatjuk.

Ha az XML szintaxist használjuk, akkor a JSP egy érvényes XML dokumentum is lehet, amely XML feldolgozó API-kkal dolgozható fel.

A JSP életciklusa

- A JSP ugyanúgy szolgálja ki a kéréseket, mint egy szervlet.
- A JSP életciklusát és a dinamikus voltát a szervlet technológia határozza meg.
- Amikor egy kérés ékezik egy bizonyos JSP-re, a web-konténer ellenőrzi, hogy a JSP szervletje régebbi-e, mint maga a JSP oldal.
- Ha igen,
  - a konténer átfordítja (translate) a JSP-t egy szervlet osztályra,
  - ezt követően lefordítja (compile) a szervlet osztályt.

Mindez automatikusan történik.

Fordítás

A statikus adat kóddá lesz alakítva, mely a tartalmat közvetlenül a válasz objektumba teszi.

A JSP elemek a következőképpen alakulnak:

- A **direktívák** szabályozzák, hogy a web-konténer hogyan fordítsa (translate) és futtassa a JSP-t
- A **szkript elemek** a szervlet osztály kódjába lesznek beillesztve
- A **kifejezés nyelv (EL) kifejezések** a kifejezés-kiértékelőnek adódnak át paraméterként.
- A `jsp:[set|get]Property` elemek a megfelelő JavaBean komponens metódushívásaivá alakulnak.
- A `jsp:[include|forward]` elemek a megfelelő szervlet API hívásokká alakulnak át.
- **A saját elemek (custom tags)** az elemkezelő (tag handler) osztály megfelelő hívásaivá alakulnak át.

Futtatási paraméterek megadása

A különböző futtatási paramétereket a page direktívában adhatjuk meg.

Pufferelés:

- Amikor a JSP lefut, a válasz objektum automatikusan pufferelve lesz.
- A puffer nagyságát a `buffer` attribútummal állíthatjuk.

```
<%@page buffer="none|xxx kb"%>
```

Hibakezelés

Hibakezelés:

- Az `errorPage` attribútum határozza meg, hogy a konténer hova kell továbbítson hiba esetén:

  `<%@page errorPage="file_name "%>`
  Pl. `<%@page errorPage="errorpage.jsp"%>`

- Az `isErrorPage` paraméter beállítja, hogy az illető JSP oldal épp a hibakezelő oldal

  `<%@page isErrorPage="true"%>`
  Ez a direktíva egy `javax.servlet.jsp.ErrorData` objektumot bocsát rendelkezésre, amely a hibaadatokat tartalmazza. Ennek segítségével meg lehet mutatni a kliensnek a hiba okára vonatkozó információt.
  A következő kifejezéssel kérhető le:
  - `${pageContext.errorData.statusCode}` a státus-kód lekérésére
  - `${pageContext.errorData.throwable}` a dobott hiba lekérésére

Statikus tartalom típusa

Statikus tartalom típusa

- Bármilyen szöveg alapú tartalom lehet: HTML, WML, XML stb.
- Alapértelmezésben HTML.
- Más típusú tartalom esetében a `contentType` attribútumot kell használjuk a tartalom beállítására.
- Ennek a direktívának a célja, hogy a böngésző helyesen értlemezze a kapott tartalmat.

Ha pl. WML-t generálunk akkor:

```
<%@page contentType="text/vnd.wap.wml"%>
```

Válasz és oldal kódolása (encoding)

Válasz és oldal kódolása:

Szintén a `contentType` attribútumot használjuk a válasz kódolásának a maghatározására, a `charset` segítségével:

- Az alábbi példában UTF-8 -t használunk, ami mindenféle lokalizálást (locale) támogat

```
<%@page contentType="text/html;charset=UTF-8"%>
```

Szkript elemek

Szkriptlet:

- A <% %> elemek közötti rész egy az egyben belekerül a szervlet forrásába, a JSP oldalon elfoglalt helyének megfelelően.
- A szkriptletekben változók is deklarálhatók, de mivel az egész <% %> közötti rész a `_jspService()` metódusba kerül, ezek a változók lokálisak lesznek.
- Metódust természetesen nem lehet szkriptletben definiálni, mert a java nem támogatja az egymásba ágyazott metódusokat.
- Metódust a deklarációs részben (lásd később) lehet definiálni.

Kifejezés:

- A <%= %> elemek közötti rész, java kód mely `String`-et ad vissza.
- Ez a `String` az `out.print()` utasításba kerül.

Deklaráció:

- A <%! %> elemek közötti rész.
- Változókat vagy metódusokat lehet így deklarálni.
- Ezek a JSP-nek megfelelő szervlet osztály-szintű tagjai lesznek.
- A deklarált változók példány-változók lesznek, de ezek használatát kerülni kell a szálkezelési problémák miatt.
  (a konténer egy adott szervletosztály egyetlen példányát hozza létre, és minden kéréshez ezt az egy példányt használja)

Dinamikus tartalom létrehozása

- Java objektumokon keresztül valósul meg.
- A JSP néhány objektumot automatikusan rendelkezésre bocsát, de elérhetők alkalmazásspecifikus objektumok is.

Implicit objektumok:

- a web-konténer hozza létre őket
- az oldalhoz (page), kéréshez (request), szesszióhoz (session), alkalmazáshoz (application) kapcsolódó információkat tartalmaznak.
- ezek ugyanazok az objektumok, amelyeket a szervlet technológia definiál.

Alkalmazásspecifikus objektumok:

- Dinamikus adatok bemutatására előkészített `JaveBean` objektumok, melyeket általában standard vagy saját elemek segítségével
  - jelenítünk meg,
  - kapjuk meg vagy állítjuk be a tulajdonságaikat.
- Ugyanezt megtehetjük szkript-elemek (scripting elements) használatával is, azaz közvetlen java kódot írva a JSP-be, de ezt lehetőleg kerüljük el.

Implicit objektumok

- `pageContext`: a JSP kontextusa. Hozzáférhetővé teszi például az alább felsorolt objektumokat
  - `servletContext`: az alkalmazáson belüli bármely komponens közös kontextusa
  - `session`: a szesszió objektum
  - `request`: a kérés, amely kiváltotta a JSP lefutását
  - `response`: a JSP által küldött válasz
  - `param`: a kérés (request) paraméterek nevét hozzárendeli az értékükhöz
  - `paramValues`: a kérés (request) paramétereket hozzárendeli a megfelelő értéktömbökhöz
  - `header`: a kérés fejléc-attribútumait hozzárendeli az értékükhöz
  - `headerValues`: a kérés fejléc-attribútumait hozzárendeli a megfelelő értéktömbökhöz
  - `cookie`: a süti nevét hozzárendeli egy süti értékéhez
  - `initParam`: inicializáló paraméterek nevét hozzárendeli az értékükhöz

- A következő implicit objektumok a megfelelő hatókörben levő attribútum-objektumokat tartalmazzák
  - `pageScope`: oldal (page) attribútumokat rendel hozzá az értékükhöz
  - `requestScope`: kérés (request) attribútumokat rendel hozzá az értékükhöz
  - `sessionScope`: szesszió attribútumokat rendel hozzá az értékükhöz
  - `applicationScope`: alkalmazás attribútumokat rendel hozzá az értékükhöz

JavaBeans komponensek

JavaBean komponens–

bármely olyan java osztály, amelyik betart bizonyos, a szerkezetére vonatkozó konvenciókat.

- ezeket a komponenseket a JSP standard nyelvi elemekkel támogatja.
- könnyen létrehozhatók és incializálhatók
- tulajdonságaik egyszerűen állíthatók, illetve olvashatók

A JavaBeans komponens tulajdonsága lehet:

- írható/olvasható, csak olvasható, csak írható
- egyszerű, azaz egyetlen értéket tartalmazó, vagy indexelt (tömb, lista, map, stb.)

- Egy tulajdonság nem feltétlenül jelent egy példányváltozót is.
- A bean tulajdonságaihoz az alább megadott nyilvános metódusokon keresztül férhetünk hozzá.

Ezek a metódusok a következő konvenciókat kell betartsák:

- Mindegyik olvasható tulajdonságra kell létezzen a következő formájú metódus:
  `PropertyClass getProperty()...`
- Mindegyik írható tulajdonságra kell létezzen a következő formájú metódus:
  `setProperty(PropertyClass pc)...`

Egy JavaBeans komponensnek rendelkeznie kell egy paraméter nélküli konstruktorral is.

JavaBeans létrehozása és használata

`jsp:useBean` elemmel deklaráljuk, hogy egy JSP egy JavaBeans komponenst fog használni.

Két alakja van:

- ```
  <jsp:useBean id="beanName"
    class="fully_qualified_classname" scope="scope"/>
  ```
- ```
  <jsp:useBean id="beanName"
    class="fully_qualified_classname" scope="scope">
      <jsp:setProperty .../>
  </jsp:useBean>
  ```

A másodikkal inicializálhatók a bean tulajdonságai.

A hatókör (scope) lehet:

- `application`, `session`, `request` vagy `page`

Ha még nem létezik a bean, a web-konténer létrehozza és a megfelelő hatókörben tárolja.

Az `id` attribútum meghatározza a bean nevét a hatókörben, amin keresztül hivatkozhatunk rá EL kifejezésekben vagy más JSP elemekben.

PI.

```
<jsp:useBean id="locales" scope="application"
 class="mypkg.MyLocales"/>
```

JavaBeans komponens tulajdonságok beállítása

- A `jsp:setProperty` elemmel töténik.
- A szintaxis a `property` forrásától függ.

`String` konstans:

```
<jsp:setProperty name="beanName"
 property="propName" value="string constant"/>
```

Kérés (request) paraméter (explicit):

```
<jsp:setProperty name="beanName"
 property="propName" param="paramName"/>
```

Kérés (request) paraméterek, melyek megegyeznek a bean tulajdonságával

```
<jsp:setProperty name="beanName"
 property="propName"/>
<jsp:setProperty name="beanName"
 property="*"/>
```

Kifejezés:

```
<jsp:setProperty name="beanName"
 property="propName" value="expression"/>

<jsp:setProperty name="beanName"
 property="propName">
  <jsp:attribute name="value">
    expression
  </jsp:attribute>
</jsp:setProperty>
```

A `beanName` attribútum meg kell egyezzen a `useBean` elem `id` attribútumával.

JavaBeans komponens tulajdonságok kinyerése

`jsp:getProperty` elem:

A tulajdonság értéket karaktersorrá (String) alakítja és beszúrja azt a válasz stream-be

```
<jsp:getProperty name="beanName" property="propName"/>
```

- a `beanName` attribútum a `useBean` `id` attribútumával meg kell egyezzen,
- a `JavaBeans` komponesben kell léteznie egy `getPropName()` metódusnak.

Kifejezés nyelv (Expression language, EL)

EL kifejezések segítségével könnyen hozzáférhetünk JavaBean-ekben tárolt alkalmazásadatokhoz
`${bookDB.bookDetails.title}`

- egy *"name"* nevű bean elérhető a `${name}` kifejezéssel
- egy beágyazott tulajdonsága elérhető a `${name.foo.bar}` szintaxissal

Pl.

```
<c:if test="${sessionScope.cart.numberOfItems >0}">
...
</c:if>
```

- Az EL kifejezéseket a JSP kifejezés-kiértékelő dolgozza fel.
- Hogy kikapcsoljuk az EL kifejezések kiértékelését az `isELIgnored` attribútumot használjuk:
  `<%@page isELIgnored ="true|false"%>`

- EL kifejezések használhatók statikus szövegben vagy bármely standard vagy saját elemben, amely egy kifejezést vár.
- Statikus szöveg esetében a kifejezés kiértékelődik és hozzáadódik az aktuális kimenethez.

Egy elem egy attribútumát több módon lehet beállítni:

- Egy EL kifejezés: `<some:tag value="${expr}"/>`
  A kifejezés ki lesz értékelve és a várt típusra lesz alakítva
- Egy vagy több, szöveggel elválasztott EL kifejezés:
  `<some:tag value="some${expr}${expr}text${expr}"/>`
  A kifejezések balról jobbra lesznek kiértékelve, majd karaktersorrá lesznek alakítva és össze lesznek fűzve.
  A keletkezett karaktersor aztán a várt típusra lesz alakítva.
- Csak sima szöveg: `<some:tag value="sometext"/>`
  Az attribútum karaktersora át lesz alakítva a várt típusra
- JSP kifejezés: `<some:tag value="<%=expression%>"/>`

Változók

A web-konténer a `PageContext.findAttribute(String)`-el keresi meg a változót, amely az EL kifejezésben megjelenik.

Pl. a `${product}` kifejezésre a konténer megkeresi a `product`-ot a `page`, `request`, `session`, illetve `application` hatókörökben és visszaadja annak értékét.
A beanek tulajdonságai a `.` operátorral érhetők el bármilyen mélységig beágyazva.

Tartalom újrafelhasználása

include direktíva:

- akkor kerül feldolgozásra, mikor a JSP át van fordítva szervlet osztálya.
- a (statikus vagy dinamikus) tartalom hozzá lesz fűzve a JSP oldal tartalmához.
- tipikusan bannerek, szerzői jogi információk befűzésére alkalmazzák.

Szintaxis:

```
<%@include file="filename"%>
```

jsp:include:

- a JSP futása közben kerül feldolgozásra.
- statikus vagy dinamikus tartalmat is hozzáfűzhetünk a JSP-hez.
- a statikus tartalom egyszerűen hozzáfűződik a hívó JSP-hez.
- a dinamikus tartalom esetében, a kérés objektum (request) tovább lesz küldve a befűzött erőforráshoz, majd a befűzött oldal lefut és az eredmény hozzá lesz fűzve a hívó JSP válaszához (reponse).

Szintaxis:

```
<jsp:include page="includedPage"/>
```

Vezérlés átadása egy másik web-komponensnek

jsp:forward:

a Java Servlet API funkcionalitását használja fel.

Szintaxis:

```
<jsp:forward page="filename"/>
```

- Amikor egy include vagy forward elemet meghívunk, az eredeti kérés át lesz adva a céloldalnak.
- Ha további adatokat akarunk a céloldalnak átadni, ezt megtehetjük a jsp:param elem segítségével.

```
<jsp:include page="...">
<jsp:param name="param1" value="value1"/>
</jsp:include>
```

Az új paraméterek hatóköre a `jsp:include` vagy `jsp:forward` hívás, azaz az új paraméterek nem érvényesek az include visszatérése után.

JSP map-elés

```
<servlet>
   <servlet-name>readInitParamJSP</servlet-name>
   <jsp-file>/web/readInitParamJSP.jsp</jsp-file>
   <init-param>
      <param-name>sys</param-name>
      <param-value>151.68.167.201</param-value>
   </init-param>
   <init-param>
      <param-name>master</param-name>
      <param-value>OTN Group IDC</param-value>
   </init-param>
</servlet>
<servlet-mapping>
   <servlet-name>readInitParamJSP</servlet-name>
   <url-pattern>/web/readInitParamJSP.jsp</url-pattern>
</servlet-mapping>
```

```
<html>
<head><title>Read Init params from a JSP</title></head>
<body>
<%!
   String emailHost = null;
   String webMaster = null;
   public void jspInit() {
     ServletConfig config = getServletConfig();
     emailHost = config.getInitParameter("sys");
     webMaster = config.getInitParameter("master");  }  %>
<table border="1">
   <tr><td>email server</td><td><%=emailHost%></td></tr>
   <tr><td>WebMaster</td><td><%=webMaster%></td></tr>
</table>
<% // It can also be read directly from the implicit
object - config %>
<%=config.getInitParameter("sys")%><br><br>
<%=config.getInitParameter("master")%>
</body>
</html>
```

The End...