

JSP technológia

Áttekintés

- ▶ Bevezetés
- ▶ JSP életrajza
- ▶ Szkript elemek, implicit objektumok, bean-ek, EL
- ▶ include, (forward)
- ▶ Visszatekintés – MVC

JSP technológia

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

A JSP –

- ▶ rendelkezésre bocsátja a servlet-ek dinamikus tulajdonságait
- ▶ jóval természetesebb módon áll hozzá a statikus tartalom létrehozásához (mint a servlet)

Egy JSP egy szöveges dokumentum, amely kétféle szöveget tartalmaz:

- ▶ statikus tartalom, amely bármilyen szöveges formátumú lehet (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.

Az XML szintaxist akkor érdemes használni, amikor a JSP egy érvényes XML dokumentum kell legyen, amelyet valamilyen XML API-val szeretnénk feldolgozni.

Néhány JSP elem szintaxisa:

JSP elem	standard szintaxis	XML szintaxis
szkriptlet	<pre>< % kód % ></pre>	<pre><jsp:scriptlet> kód </jsp:scriptlet></pre>
kifejezés	<pre>< % = kifejezés % ></pre>	<pre><jsp:expression> kifejezés </jsp:expression></pre>
deklaráció	<pre>< % ! kód % ></pre>	<pre><jsp:declaration> kód </jsp:declaration></pre>
page direktíva	<pre>< % @ page attrib= "ertek" % ></pre>	<pre><jsp:directive.page> attrib= "ertek" /></pre>
include direktíva	<pre>< % @ include file= "url" % ></pre>	<pre><jsp:directive.include> file= "url" /></pre>

A JSP életciklusa

- ▶ A JSP ugyanúgy szolgálja ki a kéréseket, mint egy servlet.
- ▶ A JSP életciklusát és a dinamikus voltát a servlet technológia határozza meg.
- ▶ Amikor egy kérés érkezik egy bizonyos JSP-re, a web-konténer ellenőrzi, hogy a JSP servlet-je régebbi-e, mint maga a JSP oldal.
- ▶ Ha igen,
 - ▶ a konténer a JSP-ből servlet forráskódot generál,
 - ▶ ezt követően lefordítja (kompilálja) a servlet osztályt.

Mindezt automatikusan végzi a Web-konténer.

Fordítás

A statikus rész olyan 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 servlet forráskóddá és futtassa a JSP-t
- ▶ A **szkript elemek** a servlet 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 servlet 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"%>
```

PI. `<%@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 (lásd:

[jspbasic/errorPage/trigger.jsp](#)):

- ▶ `${pageContext.errorData.statusCode}` a státus-kód lekérésére
- ▶ `${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 használnunk a tartalom beállítására.
- ▶ Ennek a direktívának a célja, hogy a böngésző helyesen értelmezze a kapott tartalmat.

Ha pl. WML-t generálunk akkor:

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

Regisztrált MIME típusok:

<http://www.iana.org/assignments/media-types/>

Oldal kódolása (encoding)

Oldal kódolása:

Szintén a `contentType` attribútumot használjuk az oldal kódolásának a meghatározására, a `charset` segítségével:

- ▶ Az alábbi példában UTF-8 -t használunk, ami mindenféle karakterkészletet támogat

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

Szkript elemek

Szkriptlet:

- ▶ A `<% %>` elemek közötti rész egy az egyben belekerül a servlet 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 servlet osztály-szintű tagjai lesznek.
- ▶ A deklarált változók példány-változók lesznek, de ezek használatát kerülni kell a szálkezelési problémák miatt.
(a konténer egy adott servlet-osztály 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, illetve használhatunk alkalmazásspecifikus objektumokat 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 servlet technológia definiál.

Alkalmazásspecifikus objektumok:

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

Implicit objektumok

Használhatjuk őket anélkül, hogy előbb létrehoztuk volna őket.

implicit objektumok:

- ▶ `request`: a kérés objektum (`HttpServletRequest` típusú)
- ▶ `response`: a JSP által küldött válasz (`HttpServletResponse` típusú objektum)
- ▶ `out`: a válasz objektumba írás rajta keresztül történik, puffertelt módon (ritkán használjuk explicit módon, helyette JSP kifejezést használunk)
- ▶ `session`: a szesszió objektum (ugyanaz, mint amit így kérünk le: `request.getSession()`)
- ▶ `application`: alkalmazás-szintű adatok tárolására alkalmas. `ServletContext` típusú objektum (lekérése `getServletConfig().getServletContext()`)

implicit objektumok:

- ▶ `page`: a `this` szinonímája
- ▶ `config`: `ServletConfig` típusú objektum. A JSP-nek megfelelő servlet inicializására van használva.
- ▶ `pageContext`: a JSP kontextusa. Get metódusok segítségével hozzáférhetővé teszi például a szesszió, request, response objektumokat (és másokat, melyek nagyrésze közvetlenül is elérhető)
- ▶ `exception`: hiba esetén a hibát okozó kivételt tartalmazza

Bean-ek – visszatekintés

Java osztályok, melyek bizonyos szabályoknak tesznek eleget:

- ▶ Rendelkeznek üres (azaz paraméter nélküli) konstruktorral (is), vagy egyáltalán nincs konstruktoruk
- ▶ Ne legyen publikus példány– (azaz nem statikus) mezőjük
- ▶ A mezőkhöz való hozzáférés *setXxx*, illetve *getXxx* metódusok segítségével történik
 - ▶ ha pl. az osztálynak van egy *getName* metódusa, ami `String` típust térít vissza, azt mondjuk, hogy a bean-nek van egy *name* nevű *tulajdonsága*
 - ▶ Boolean típusú tulajdonság esetén használható *isXxx* a *getXxx* helyett

Bean – pl.

```
package bean.pelda;

public class PersonBean {
    private String name;

    public String getName(){
        return name;
    }

    public String setName(String name){
        this.name=name;
    }
}
```

JavaBeans komponensek

JavaBean-nek minősül

bármely olyan java osztály, amelyik betartja az emített, szerkezetére vonatkozó konvenciókat.

- ▶ ezeket a komponenseket a JSP standard nyelvi elemekkel támogatja.
- ▶ könnyen létrehozhatók és inicializálhatók
- ▶ 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.)

JavaBeans létrehozása és használata

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

Több alakja van:

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

A hatókör (scope) lehet:

- ▶ application, session, request vagy page

Ha még nem létezik a bean, a web-konténer létrehozza (amennyiben meg van adva a class attribútum) 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 történik.
- ▶ 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 értékét 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 komponensben kell léteznie egy `getPropName()` metódusnak.

Példák

- ▶ `jspbasic/beans/...`, illetve `jspbasic/scope/...`

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.valami1.valami2}` szintaxissal

- ▶ 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 beállítani:

- ▶ 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 karaktersorra 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%>"/>`

Pl. lásd: [jspbasic/beans/sessionScopeBean.jsp](#)

Változók

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

PI. 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 servlet osztállyá.
- ▶ 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 (response).

Szintaxis:

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

pl:

lásd: [jspbasic/include/index.jsp](#)

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.

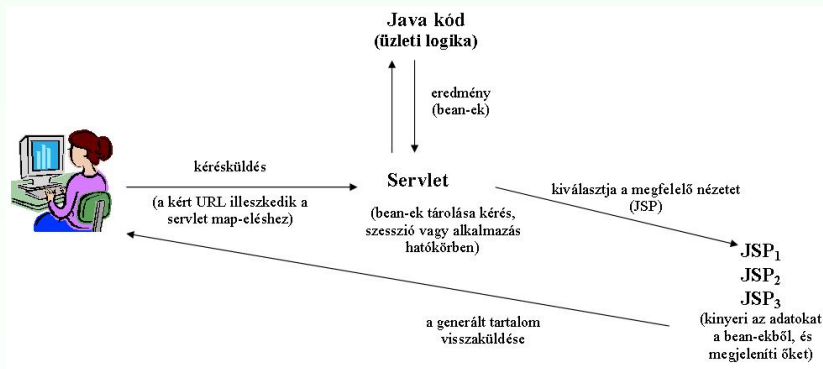
```
<jsp:include page="...">  
<jsp:param name="param1" value="value1"/>  
</jsp:include>
```

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

egyéb példák:

- ▶ formelemek feldolgozása: `jspbasic/formElements/...`
- ▶ összetettebb pl.: `jspbasic/converter/converter.jsp`

MVC elv – visszatekintés



MVC működési elv egyszerű Java Web-alkalmazás esetén

MVC elv – visszatekintés

MVC elv implementálása egyszerű Java Web-alkalmazás esetén

1. Definiáljuk az adatokat tároló bean-eket
2. A kérések lekezelésére használjunk servlet-et
 - ▶ a servlet kiolvassa a kérés paramétereit, ellenőrzi a hiányzó vagy helytelen adatokat, stb.
3. Töltsük fel a bean-eket
 - ▶ A vezérlő servlet meghívja az üzleti logikát, mely adatokat szolgáltat vissza. Ezeket tároljuk az 1. pontnál meghatározott bean-ekben.
4. Tároljuk a bean-eket valamelyik Web-hatókörben (kérés, szesszió vagy alkalmazás hatókör)
 - ▶ a servlet meghívja a megfelelő hatókör-objektum `setAttribute` metódusát, mely egy bizonyos kulcs alapján hozzáférhető referenciát tárol a megfelelő bean-re

MVC elv – visszatekintés

MVC elv implementálása egyszerű Java Web-alkalmazás esetén

5. Továbbítsunk (forward) a megfelelő JSP-re
 - ▶ a servlet kiválasztja a megfelelő nézetet (megfelelő JSP), és a `RequestDispatcher` `forward` metódusa segítségével átirányít az illető JSP-re.
6. Kinyerjük az adatokat a bean-ekből és megmutatjuk őket
 - ▶ a JSP oldal hozzáfér a megfelelő bean-hez `jsp:useBean`-t használva. Attribútumként megadjuk a 4. pontnál meghatározott hatókört. Ezután `jsp:getProperty`-t használunk a bean tulajdonságainak megjelenítésére.
 - ▶ a JSP tipikusan nem hozza létre vagy módosítja a bean-t csupán megmutatja a servlet által előkészített adatokat.

pl. MVC elvre alapozó egyszerű Web-alkalmazás:

- ▶ `mvc/index.jsp`