Java web-alkalmazások 2.

A Java Server Pages technológia (JSP, JSP elemkönyvtárak, JSTL)

Simon Károly
simon.karoly@codespring.ro

JSP

- Statikus és dinamikus részekkel egyaránt rendelkező webes tartalmak készítése → megjelenítés.
- JSP: szöveges dokumentum, amely statikus tartalmat (HTML, XML stb.) és JSP elemeket tartalmaz.
- A JSP elemek esetében kétféle szintaxis alkalmazható: standard vagy XML alapú (az utóbbi akkor javasolt, ha a JSP-nek érvényes XML dokumentumnak kell lennie, amely valamilyen XML API-val feldolgozható).
- A JSP háttérben servlet-ként fut, alapviselkedését (pl. kérések kiszolgálása), életciklusát ez a tény határozza meg.
- Egy JSP-hez érkezett kérés esetében a konténer ellenőrzi, hogy a JSP-nek megfelelő servlet régebbi-e az oldalnál, és ha igen servlet forráskódot generál az oldalból és automatikusan lefordítja a servlet osztályt.

JSP elemek

- Szkriptlet:
```
<% code %>
<jsp:scriptlet> code </jsp:scriptlet>
```
- Kifejezés:
```
<%= expression %>
<jsp:expression> expression </jsp:expression>
```
- Deklaráció:
```
<%! code %>
<jsp:declaration> code </jsp: declaration>
```
- Page direktíva:
```
<%@ page attrib="value" %>
<jsp:directive.page> attrib=value />
```
- Include direktíva:
```
<%@ include file="URL" %>
<jsp:directive.include> file="URL" />
```
- Fordítás:
  - Direktívák: szabályozzák, hogy a web-konténer hogyan végezze a fordítást.
  - Szkript elemek: a servlet kódjába lesznek beillesztve.
  - EL kifejezések: a kifejezés kiértékelőnek lesznek átadva paraméterként.
  - jsp:setProperty és jsp:getProperty elemek a JavaBean metódushívásaiba lesznek alakítva.
  - jsp:include és jsp:forward elemek a megfelelő servlet API hívásokba lesznek alakítva.
  - Saját elemek (custom tags): az elemkezelő (tag handler) megfelelő hívásaiba lesznek alakítva.

JSP – hibakezelés

- Az errorPage attribútum határozza meg, hogy a konténer hova továbbítson hiba esetén. Pl. **`<%@page errorPage="errorpage.jsp"%>`**.
- Az isErrorPage paraméter beállítja, hogy az illető JSP oldal legyen a hibakezelő oldal: **`<%@page isErrorPage="true"%>`**.
- Egy javax.servlet.jsp.ErrorData objektum tartalmazza a hibaadatokat, ennek segítségével lehet megmutatni a kliensnek a hiba okára vonatkozó információt. Pl: **`${pageContext.errorData.statusCode}`** a kód lekérdezése, **`${pageContext.errorData.throwable}`** a dobott kivétel lekérdezése.
- A statikus tartalom esetében a contentType attribútum segítségével állíthatjuk be a tartalom típusát és a használt kódolást (a böngésző helyesen értelmezhesse az adatokat):

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

JSP – szkript elemek

- Szkriptlet:
  - <% %> elemek közötti Java kód, amely belekerül a servlet kódjába, a jspService() metódusba, a JSP oldalon elfoglalt helyének megfelelően.
  - Nem definiálhatunk szkriptleten belül metódusokat, mivel az metódusok egymásba ágyazását eredményezné és a Java ezt nem támogatja. Változókat deklarálhatunk, de ezek lokális változók lesznek.
- Kifejezés:
  - <%= %> elemek közötti Java kód, amely egy String-et ad vissza, és ez az out.print() metódusnak lesz átadva.
- Deklaráció:
  - <%! %> elemek közötti rész, változókat, vagy metódusokat deklarál.
  - A metódusok a JSP-nek megfelelő servlet osztályszintű tagjai lesznek.
  - A változók példányváltozók lesznek → ezek használatára figyelni kell a szálkezelési problémák miatt (használatukat kerülni javasolt).
    - Az erőforrás-menedzsment konténer-specifikus, de általában a konténer egy adott servlet osztálynak csak egyetlen példányát hozza létre, vagy osztott rendszer esetében virtuális gépenként egy példányát (az is lehetséges, hogy pooling mechanizmust alkalmaz). A különböző kéréseket viszont különböző véghrehajtási szálakon szolgálja ki (thread pool-t alkalmaz). A szinkronizálással kapcsolatos problémák megoldása a programozó feladata. Megjegyzés: a SingleThreadModel interfész érvénytelennek volt nyilvánítva.
    - Más megoldások: munkamenet (session) objektumban, vagy kérés objektumban tárolhatjuk az adatokat, lokális változókat használhatunk a szkriptleteken belül. Vagy: szinkronizált hozzáférés (de ez a teljesítmény kárára mehet – nem javasolt).

JSP – implicit objektumok

- **request**: a kérés objektum (HttpServletRequest típusú)
- **response**: a JSP által küldött válasz (HttpServletResponse típusú)
- **out**: a válasz objektumhoz tartozó adatfolyam, puffer alkalmazása (közvetlen módon ritkán használjuk, mivel JSP kifejezést használhatunk)
- **session**: a munkamenet objektum (lekérés request.getSession())
- **application**: alkalmazásszintű adatok tárolása (ServletContext típusú, lekérés getServletConfig().getContext())
- **page**: az aktuális JSP oldalra mutat (tulajdonképpen a this szinonímája)
- **config**: ServletConfig típusú, a JSP-nek megfelelő servlet inicializálása
- **pageContext**: a JSP kontextusa, amely get metódusokon keresztül hozzáférhetővé teszi a session, request, response stb. objektumokat
- **exception**: hiba esetén a hibát okozó kivételt tartalmazza

JSP – JavaBean-ek

- jsp:useBean elemmel deklaráljuk, hogy egy JSP egy JavaBean-t fog használni.
- `<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>`
- Pl. `<jsp:useBean id="locales" scope="application" class="mypkg.MyLocales"/>`
- A hatókör (scope) lehet: application, session, request, vagy page
- Ha még nem létezik a bean, és a class meg van határozva, a web-konténer létrehozza és a megfelelő hatókörben tárolja.
- A type lehetőséget ad konverzióra (ha már létezik az objektum). Nem példányosít, ha az objektum még nem létezik, de együtt is használható a class-el (ebben az esetben megtörténik a példányosítás).
- Az id attribútum meghatározza a bean nevét a hatókörön belül, ezen keresztül hivatkozhatunk rá kifejezésekben, vagy más JSP elemekben.

JSP – JavaBean-ek

- A tulajdonságok beállítása a jsp:setProperty elemmel történik. A beanName attribútumnak meg kell egyeznie a useBean elemben megadott id-val. Példák:
- String:
```
<jsp:setProperty name="beanName" property="propName" value="string" />
```
- Kérés paraméter:
```
<jsp:setProperty name="beanName" property="propName"  param="paramName"/>
```
- Kérés paraméter, amely megegyezik 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 tulajdonságok kinyerése a jsp:getProperty elemmel történik. A tulajdonság értékét karaktersorrá (String) alakítja és beszúrja a válaszba
- `<jsp:getProperty name="beanName" property="propName"/>`
- A beanName attribútumnak a useBean id attribútumával kell megegyeznie, a JavaBean-en belül kell léteznie egy getPropName() metódusnak.

JSP – Expression Language

- EL kifejezések segítségével is hozzáférhetünk JavaBean-ekben tárolt adatokhoz. A beanek tulajdonságai a . operátorral érhetőek el, bármilyen mélységig beágyazva.
- `${bookDB.bookDetails.title}`
- Egy ”name” nevű bean elérhető a `${name}` kifejezéssel, egy tulajdonsága elérhető a `${name.valami1.valami2}` szintaxis alkalmazásával.
- Az EL kifejezéseket a JSP kifejezés-kiértékelő dolgozza fel. Ez kikapcsolható az isELIgnored attribútum alkalmazásával:
- `<%@page isELIgnored ="true|false"%>`
- Kiértékelés: `<some:tag value="some${expr}${expr}text${expr}"/>`
- A kifejezések balról jobbra lesznek kiértékelve, majd karaktersorrá alakítva és összefűzve. A keletkezett karaktersor a várt típusba lesz alakítva.
- 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.

JSP – include és forward

- Tartalom újrafelhasználása, include direktíva: a JSP servlet osztályba történő átfordításakor lesz alkalmazva, a statikus vagy dinamikus tartalom hozzá lesz fűzve a JSP tartalmához: `<%@include file="filename"%>`
- A jsp:include a JSP futása közben lesz feldolgozva, statikus vagy dinamikus tartalom beágyazását eredményezi: `<jsp:include page="includedPage"/>`
- A jsp:forward a Servlet API által biztosított továbbítás funkcionalitást valósítja meg: `<jsp:forward page="filename"/>`
- Ha további adatokat akarunk a céloldalnak átadni, 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, nem érvényesek a metódusok visszatérése után.

Java web-alkalmazások: MVC

- MVC:
  - Modell: JavaBean-ek
  - Nézet: JSP
  - Vezérlés: Servlet
    - Fogadja a kérést és paramétereit, a paramétereket a megfelelő típusba alakítja és ellenőrzi helyességüket
    - Meghívja a megfelelő üzleti logikával kapcsolatos metódusokat
    - Az eredmény alapján továbbít a megfelelő nézethez (JSP)
- Működés:
  - A kliens (böngésző) kérést (GET vagy POST) intéz a szerverhez.
  - A konfigurációs állomány alapján a szerver a kérést a megfelelő Servlet-hez továbbítja.
  - A Servlet a kérés URL alapján meghívja a megfelelő parancsobjektumot.
  - A parancsobjektum kommunikál az üzleti logikáért felelős komponensekkel, majd átirányít a megfelelő nézetre (JSP).
  - A nézet megjeleníti a megfelelő információkat a kliens browserében.
- Mivel a JSP-k tulajdonképpen a háttérben Servlet-ként futnak, elvileg felcserélhetőek, de az MVC elv szempontjából mindeniknek megvan a maga előnye: a Servletek alkalmasabbak a vezérlésre és adatfeldolgozásra, a JSP-ket szöveg alapú oldalak (html, xml stb.) létrehozására alkalmasabbak.
- További technológiák és keretrendszerek: JSF, Struts stb.

Servlet, JSP → MVC

- Az adatokat tároló bean-ek létrehozása.
- A kéréseket kezelő servlet(ek) létrehozása. A servlet kiolvassa a kérés paramétereket, ellenőrzi az adatokat stb.
- A vezérlő servlet kommunikál az üzleti logikát megvalósító komponensekkel, amelyek adatokat szolgáltatnak vissza. Ezeket a bean-ekben tároljuk.
- A bean-eket tároljuk valamelyik Web-hatókörben. A servlet meghívja a megfelelő hatókör-objektum setAttribute metódusát. A bean-re mutató referenciához egy azonosító alapján férhetünk hozzá.
- A Servlet továbbít a megfelelő JSP-re. Kiválasztja a megfelelő nézetet majd alkalmazza a RequestDispatcher forward metódusát.
- A JSP-n belül kinyerjük az adatokat a bean-ekből és felépítjük a nézetet. A JSP oldal jsp:useBean-t elem segítségével fér hozzá a bean-ekhez. Attribútumként a meghatározott hatókört kell megadnunk. Ezután jsp:getProperty-t elemeket használhatunk a bean tulajdonságainak kinyeréséhez.
- Általában a JSP nem hozza létre, vagy módosítja a bean-t, csak megmutatja a servlet által előkészített adatokat.

JSP elemkönyvtárak

- A JSP technológia lehetőséget ad arra, hogy elemkönyvtárakba (tag library) rendezett saját elemeket hozzunk létre.
- A fejlesztő által definiált JSP elemek ismétlődő feladatokat oldanak meg. Az elemkönyvtárak több összefüggő elemet tartalmaznak (és azok implementációját) .
- Kiküszöbölik a szkriptlet kódot a JSP-ben, vagy csökkentik annak mennyiségét, egyszerűbbé teszik a szintaxist, növelik a hatékonyságot, újrafelhasználhatóak.
- Szintaxis:
  - Törzs nélküli:

```
<prefix:tag attr1="value1"...attrN="valueN"/>
```

  - Törzs tartalommal:

```
<prefix:tag attr1="value1"...attrN="valueN">
        body
</prefix:tag>
```

  - A prefix az elemkönyvtárat azonosítja, a tag az elemet, attr1 ... attrN az attribútum nevek (az attribútumok módosítják az elem viselkedését).

JSTL - standard elemkönyvtár

- JSP Standard Tag Library: egy egységes elemcsomagot használhatunk, az alkalmazás bármely alkalmazásszerverre telepíthető lesz, az elemek implementációja optimalizált.
- A JSTL különböző feladatkörökre kínál elemeket. Néhány elemkönyvtár, és megfelelő URI:
  - Core: http://java.sun.com/jsp/jstl/core
  - Internationalization: http://java.sun.com/jsp/jstl/fmt
  - XML: http://java.sun.com/jsp/jstl/xml
  - SQL: http://java.sun.com/jsp/jstl/sql
- A JSP-ben a következőképpen hivatkozhatunk ezekre:

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

- Az elemek közötti együttműködés történhet implicit módon, amikor egy beágyazott elem egy meghatározott interfészen keresztül együttműködik az őt tartalmazó elemmel, vagy explicit módon, amikor az elem egy változóban információt ad át a környezetnek. A változó nevét a var attribútummal adjuk meg.

JSTL - core

- Változók használatát támogató elemek: a **set** elem beállítja egy változó értékét egy EL kifejezés alapján, egy adott hatókörben. Ha a változó még nem létezik, akkor létrehozza.
- Egy változó beállítható a value attribútummal, vagy az elem törzsével:

```
<c:set var="valtozonev" scope="session" value="..."/>
<c:set var="valtozonev"> ... </c:set>
```

- A **remove** elem segítségével eltávolítható:

```
<c:remove var="valtozonev" scope="session"/>
```

- Kollekciók bejárása: **forEach**.

```
<c:forEach var="item" items="${sessionScope.cart.items}">
        …
        ${item.quantity}
        …
</c:forEach>
```

- Feltételes elemek:

```
<c:if test="${bean.value}"> ... </c:if>
```

- If-then-else: choose elem alkalmazása

JSTL - core

- **Choose** elem:

```
<c:choose>
      <c:when test="${count == 0}"> Item not found. </c:when>
      <c:otherwise> No. items: ${count}. </c:otherwise>
</c:choose>
```

- Az **import** elem segítségével elérhetünk egy URL által megadott erőforrást, amely befűzhető vagy feldolgozható a JSP-ben:

```
<c:import url="/books.xml" var="xml"/>
<x:parse doc="${xml}" var="booklist" scope="application"/>
```

- Az **url** elem: az encodeURL metódussal megegyező funkcionalitás (munkamenet követése, amikor a sütik ki vannak kapcsolva).
- A **param** elem kérés paramétereket specifikál.
- A **redirect** elem HTTP átirányítást végez el.
- A **catch** elem: kevésbé fontos kivételek esetében nem kell feltétlenül a hibaoldalra továbbítani, a kezelés helyben is történhet, a hiba az oldal hatókörében lesz tárolva.
- Az **out** elem kiértékel egy kifejezést, és az eredményt a JspWriter objektumba teszi.

JSTL – i18n, L10n

- Amikor egy kérés érkezik, a JSTL automatikusan beállítja a locale-t a fejléc alapján és kiválasztja a megfelelő erőforrás állományt, felhasználva a paraméterként megadott alapnevet.
- setLocale elem: a kliens által a böngészőben specifikált locale felülírása.
- requestEncoding elem: a kérés objektum karakterkódolásának beállítása.
- message elem: nyelvfüggő üzenetek megjelenítése:

```
<fmt:message key="Choose"/>
```

- param elemekkel az üzenetnek további paramétereket adhatunk meg.
- Az fmt:setBundle elem az erőforrás állományt egy hatókörhöz rendelt változóba menti le.
- Az fmt:bundle egy adott elem törzsében használt erőforrás állományt állít be.
- Formázó elemek: formatNumber, formatDate, parseDate, parseNumber, setTimeZone, timeZone

```
<fmt:formatNumber value="${item.price}" type="currency"/>
```

JSTL – saját elemek

- Fejlesztés lépései: elemkezelő (tag handler) osztály implementálása, elemkönyvtár leíró (tag library descriptor) létrehozása, elem használata.
- Elemkönyvtár deklarációja:

```
<%@taglib prefix="tt" uri="URI"%>
```

- URI: azonosítja az elemkönyvtár leíróját, amely egy .tld állomány, a WEB-INF könyvtárban, vagy annak valamelyik alkönyvtárában található, vagy JAR állományokban.
- TLD-re hivatkozhatunk direkt módon (útvonal és állomány megadása), vagy indirekt módon egy hozzárendelt név segítségével, mely esetben a telepítés-leíróban meg kell adnunk a megfeleltetést.
- A saját elem egy Java osztály, amely megvalósítja a Tag (törzs nélküli), vagy BodyTag interfészt. Alternatív lehetőség: származtatás a TagSupport vagy BodyTagSupport absztrakt osztályokból.