Laboratorul 08

Realizarea de aplicații web folosind JavaServer Pages

Aplicaţii Integrate pentru Întreprinderi (AIPI)
Semestrul de Toamnă 2014

Departamentul de Calculatoare

Conținut

- Tehnologia JavaServer Pages – aspecte generale
- Integrarea JavaServer Pages cu serverul web Apache Tomcat 8.x
- Ciclul de viaţă al unei pagini JSP
- Arhitectura unei pagini JSP
  - Directive
  - Acțiuni
  - Scripleți
  - Obiecte Implicite
- Implementarea unor funcționalități complexe
- Utilizarea EL (Expression Language) în pagini JSP
- Utilizarea JSTL (JavaServer Pages Standard Tag Library) pentru implementarea unor aspecte ce țin de logica aplicației

Tehnologia JavaServer Pages – aspecte generale

- dezvoltarea de aplicații Internet
  - generate dinamic
  - bazate pe documente (HTML, XML)
- 1999 – lansată de Sun Microsystems, parte a Java Enterprise Edition
- beneficii în comparație cu alte tehnologii similare
  - PHP – comparabil odată cu introducerea orientării pe obiecte
  - ASP .NET
    - limbajul Java este mai puternic și mai ușor de utilizat decât limbajele Microsoft
    - portabilitate (sisteme de operare / servere de aplicații)
  - Java Script – nu poate genera cod dinamic la nivelul serverului
  - SSI (Server Side Includes) – nu poate fi folosit pentru funcționalități complexe
  - CGI (Common Gateway Interface)
    - performanță îmbunătățită (integrare de elemente dinamice fără a se folosi fișiere separate)
    - compilare înainte de plasare în contextul serverului
    - accesarea funcționalităților oferite de interfețele de programare Java (JDBC, JNDI, JAXP, EJB)
    - delegarea logicii aplicației către Java Servlets

Tehnologia JavaServer Pages – aspecte generale (cont'd)

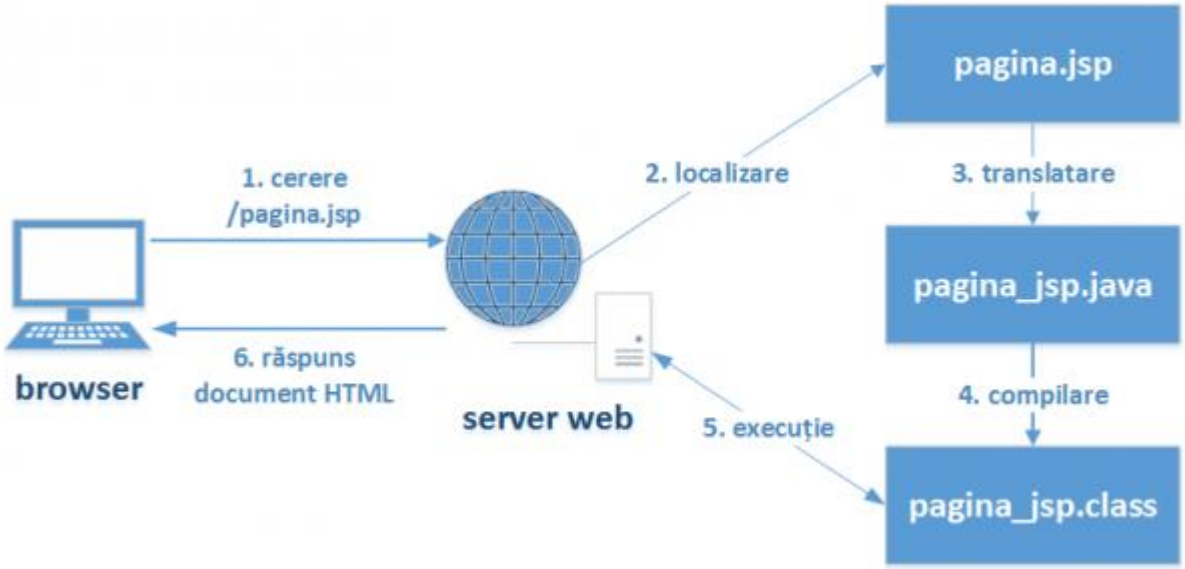
- folosită împreună cu tehnologia Java Servlets
  - întreținere mai facilă a aplicației Internet
  - separare între logica aplicației / nivelul de prezentare
- pagina JSP = document text
  - elemente **statice** (etichete de adnotare): HTML, XML, SVG, WML
  - elemente **dinamice**
    - directive, acțiuni, scripleți
    - cuprinse între <% și %>
    - implementarea unor funcționalități complexe (comunicarea cu baza de date, reținerea preferințelor utilizatorilor, accesarea componentelor Java Beans, transferul controlului între pagini, partajarea informației între cereri și răspunsuri)

Tehnologia JavaServer Pages – aspecte generale (cont'd)

- beneficii
  1. separarea logicii aplicației de nivelul de prezentare – întreținere mai facilă
     - logica aplicației – Java Servlets / componente Java Beans (conținut dinamic)
     - Interfața cu utilizatorul – etichete JSP
  2. portabilitate dată de limbajul de programare Java
  3. generarea de conținut dinamic
  4. abstractizare de nivel înalt a tehnologiei Java Servlets ale cărei avantaje sunt preluate (inclusiv reutilizarea codului sursă)

- Java Servlets = JavaServer Pages
  - Java Servlets – clasă Java (etichetele de adnotare generate prin apelarea unor metode)
  - JavaServer Pages – document (separarea conținutului static de cel dinamic)
  - procesul pagina JSP – transformare – Java Servlets – compilare este realizat în mod automat de fiecare dată când sunt detectate modificări

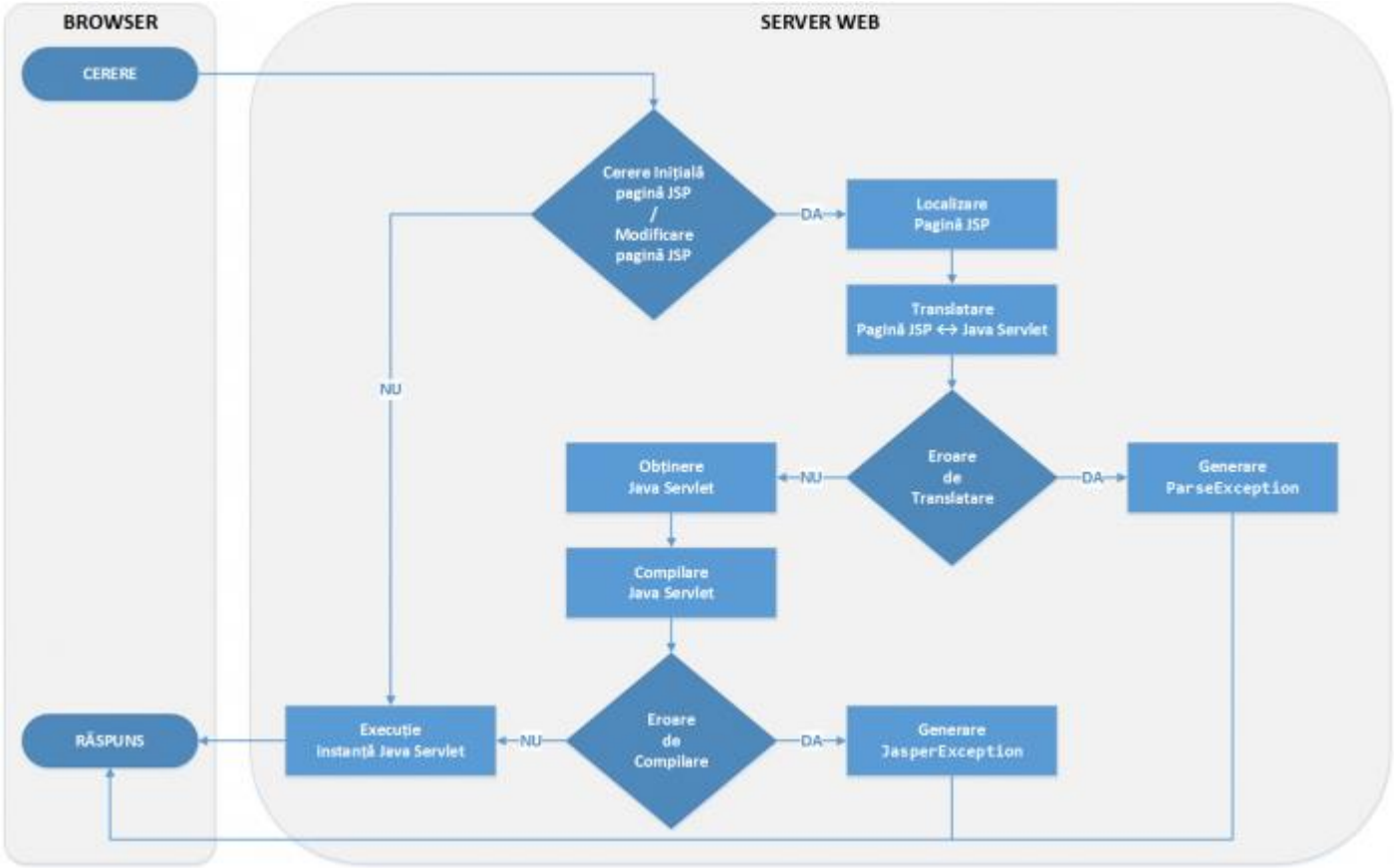
Tehnologia JavaServer Pages – aspecte generale (cont'd)



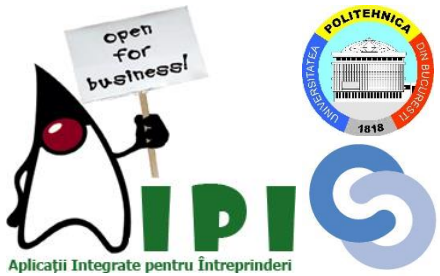
Tehnologia JavaServer Pages – aspecte generale (cont'd)

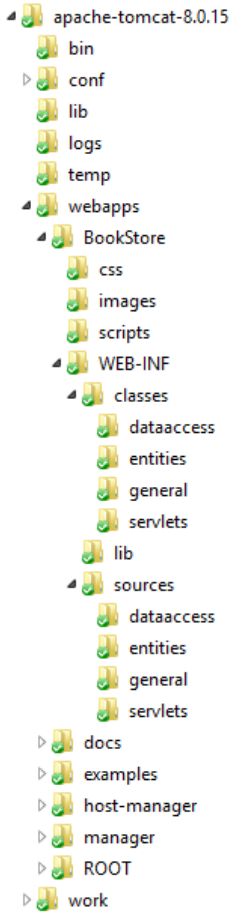
- interacțiunea client/server
  1. clientul transmite din browser o cerere pentru o pagină .jsp
  2. resursa solicitată este încărcată de motorul JSP
  3. dacă nu există un Java Servlet corespunzător celei mai recente versiuni a paginii .jsp, acesta este generat
     - elemente statice – instrucțiuni `println`
     - elemente dinamice – cod Java corespunzător
  4. compilarea Java Servlet într-o clasă executabilă căreia i se transmite cererea originală
  5. rularea clasei corespunzătoare = rezultat în format HTML
  6. documentul generat este afișat în browser ca și cum ar fi conținut static

Tehnologia JavaServer Pages – aspecte generale (cont'd)



Interacțiunea JavaServer Pages cu serverul Apache Tomcat 8.x





- aplicația Internet
  - plasată în `webapps/<application_name>`
  - accesibilă la http://<server_IP>:<server_port>/application_name
  - se afișează
    - `index.jsp` / `index.html`
    - fișierul specificat de proprietatea `<welcome-file>` a secțiunii `<welcome-file-list>` din fișierul de configurare `web.xml`
- directorul `WEB-INF`
  - `classes` – clase Java compilate ce implementează logica aplicației
  - `lib` – driver de conectare la baza de date, accesarea funcționalităților oferite de JSTL
  - `src/sources` – cod sursă (nu există o denumire standard)
  - `web.xml` – parametri ai aplicației Internet (clasele Java Servlet care implementează logica aplicației)
- alte resurse – imagini, foi de stil

Interacțiunea JavaServer Pages cu serverul Apache Tomcat 8.x (cont'd)

- încărcarea claselor se face la pornirea serverului Apache Tomcat 8.x

```
Dec 4, 2014 12:00:00 AM org.apache.catalina.startup.HostConfig deployDirectory
INFO: Deploying web application directory ApacheTomcat\apache-tomcat-8.0.15\webapps\BookStore
```

  - modificări la nivelul claselor Java Servlets – este necesară repornirea serverului pentru încărcarea acestora
  - modificări la nivelul paginilor JSP – NU este necesară repornirea serverului (generarea claselor Java Servlets și încărcarea lor este realizată automat)

- generarea claselor Java Servlets corespunzătoare paginilor JSP
  - localizate la `%CATALINA_HOME%\work\Catalina\<server_IP>\<application_directory>\org\apache\jsp`
  - pagina.jsp → pagina_jsp.java (.class)
  - operația poate dura o perioadă de timp mai mare (ulterior, încărcarea paginilor este realizată într-o perioadă de timp mai mică)
  - erorile (transformare / compilare) sunt afișate la nivelul paginii generate

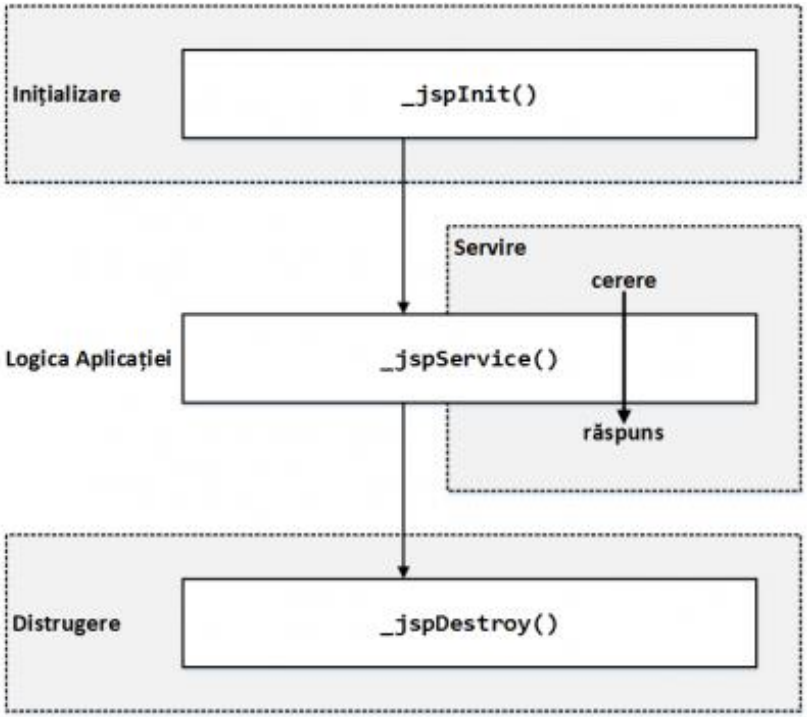
- Apache Tomcat 8.0.15 – JSP 2.3 / EL 3.0

Ciclul de viață al unei pagini JSP

- ciclul de viață al unei pagini JSP = ciclul de viață al Java Servlets
  - etapa suplimentară de translatare / compilare
  - servlet special – verifică dacă servlet-ul corespunzător corespunde celei mai recente versiuni a paginii JSP
    - în caz contrar, declanșează **în mod automat** procesele de translatare / compilare
  1. încărcare clase
  2. instanțiere clase
  3. inițializare – metoda `init()`
     - resurse partajate
     - fișiere de configurare
  4. execuție (realizarea procesărilor) – metoda `service()`
  5. distrugere – metoda `destroy()`

- metode generate – pachetul `javax.servlet.HttpJspPage`
  - `_jspInit()`, `_jspDestroy()` – pot fi suprascrise sub formă de declarații JSP (cuprinse între <%! și %>)
    - metodele `jspInit()`, `jspDestroy()`
    - apelate o singură dată în ciclul de viață al unei pagini JSP
  - `_jspService()` – nu se recomandă suprascrierea sa (este generată în mod automat prin transformarea elementelor JSP în codul Java corespunzător)

Ciclul de viață al unei pagini JSP (cont'd)



1) etapa de **inițializare** – realizarea conexiunii cu baza de date, încărcarea de resurse, configurarea aplicației Internet (inițializarea obiectelor pentru accesarea funcționalității oferite de alte biblioteci)
2) etapa de **execuție** – generarea unui răspuns pentru fiecare cerere, tratând metodele prin care clientul comunică cu serverul (GET, POST, PUT, DELETE, OPTIONS, TRACE)
3) etapa de **distrugere** – eliberarea resurselor utilizate de aplicație

```
public void _jspInit() {
    // ...
}

public void _jspDestroy() {
    // ...
}
```

```
public void _jspService (final javax.servlet.http.HttpServletRequest request,
                         final javax.servlet.http.HttpServletResponse response)
                         throws java.io.IOException, javax.servlet.ServletException {
    // ...
}
```

Ciclul de viață al unei pagini JSP (cont'd)

- generarea claselor Java Servlet din paginile JSP
  1. etapa de translatare (parsare) – excepție `ParseException`
     - clasa Java Servlet este incompletă
     - ultima linie din clasa Java Servlet indică elementul JSP incorect
  2. etapa de compilare – excepție `JasperException` (eroare de sintaxă scriplet)
     - mesaj care include numele clasei Java Servlet asociată paginii JSP
     - linia la care s-a constatat eroarea
- excepțiile trebuie tratate în cadrul unei pagini dedicate

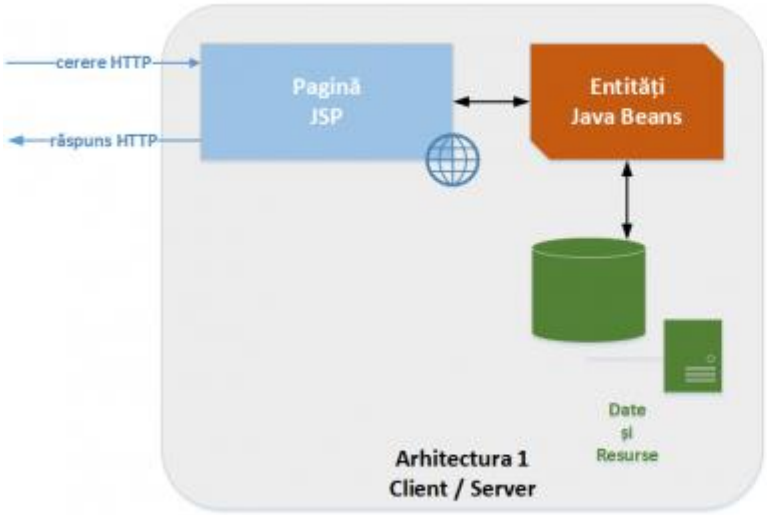
```
<%@ page errorPage="errorpage.jsp" %>
```

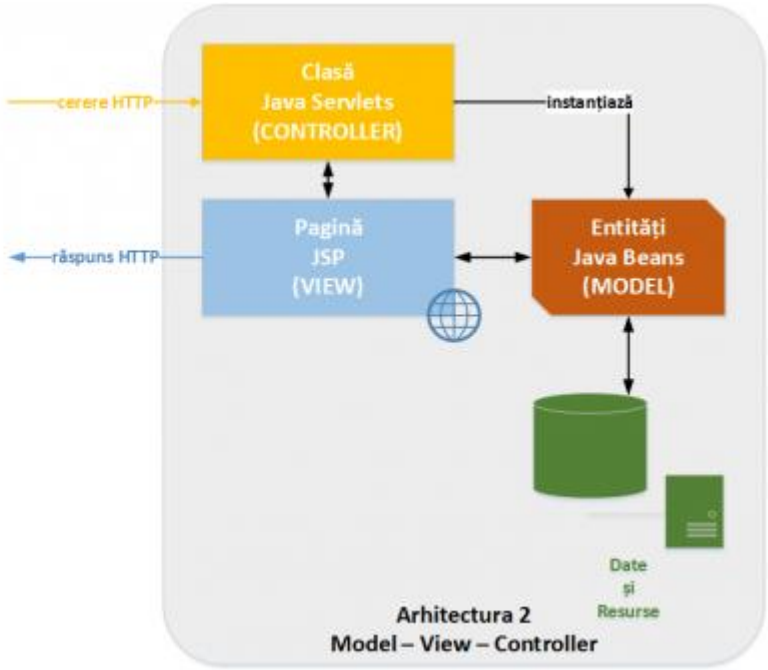
  - excepția are tipul `javax.servlet.jsp.JspException`
  - disponibilitatea excepției este condiționată de precizarea tipului de pagină

```
<%@ page isErrorPage="true|false" %>
```

Arhitectura unei aplicații Internet folosind JavaServer Pages

- logica aplicației
  - întreținută de programatori
  - algoritmi / interacțiune cu baza de date
- nivelul de prezentare
  - întreținută de dezvoltatori web
  - elemente de grafică

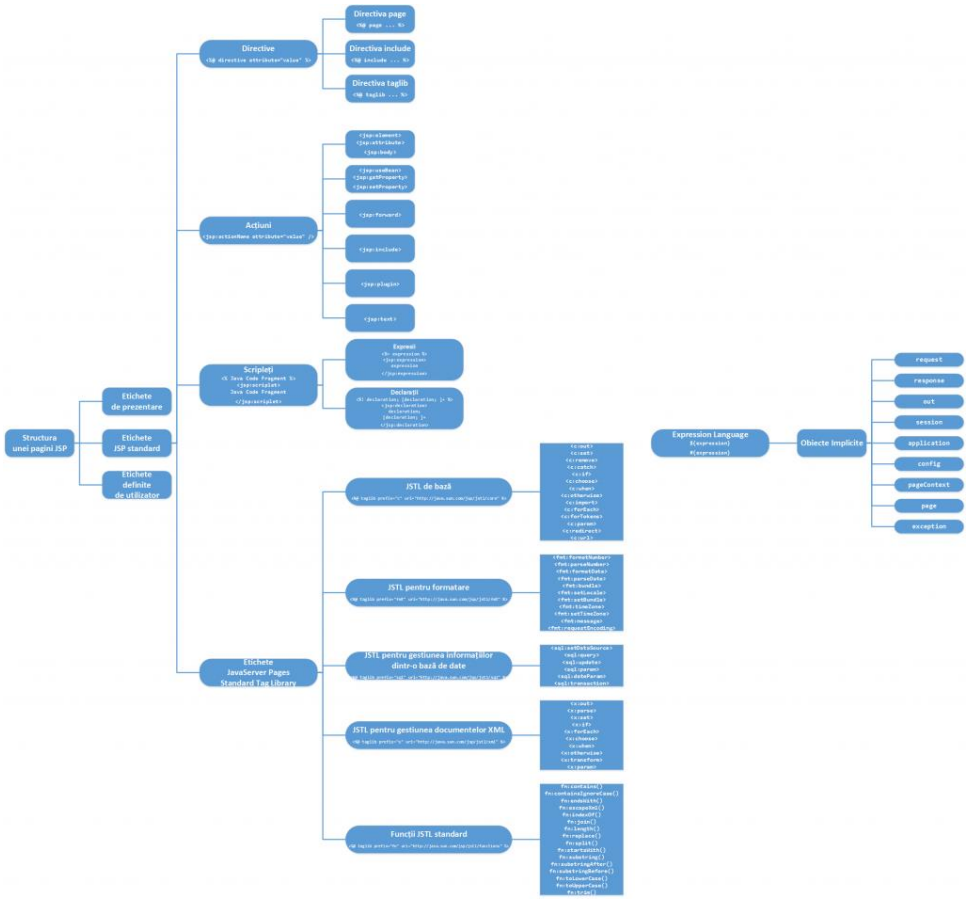




Arhitectura unei aplicații Internet folosind JavaServer Pages (cont'd)

- Arhitectura 1 – Client / Server
  - logica aplicației – componente Java Beans
  - nivelul de prezentare – pagini JSP (realizează întreaga comunicație cu clientul!!!)
  - aplicații Internet nu foarte complexe
  - caracteristici
    - avantaje: simplitate
    - dezavantaj: lipsa de scalabilitate
- Arhitectura 2 – Model – View – Controller = abordare pe N-niveluri
  - componente Java Beans (model)
  - pagina JSP (view) – procesează răspunsul către client
  - Java Servlets (controller) – procesează cererea de la client, implementează logica aplicației
  - aplicații Internet cu nivel ridicat de complexitate
  - caracteristici
    - avantaje: ușurință de întreținere, eficiență
    - dezavantaj: procesul de proiectare este mai dificil

Arhitectura unei aplicații Internet folosind JavaServer Pages (cont'd)



- etichete de prezentare (HTML, XML)
- etichete JSP standard
  - directive
  - acțiuni
  - scripleți
    - expresii
    - declarații
  - comentarii
    - JSP – ignorate de motorul JSP

      `<%-- JSP comment -->`
    - HTML – ignorate de browser

      `<!-- HTML comment -->`
  - etichete JSTL
- etichete definite de utilizator

Arhitectura unei aplicații Internet folosind JavaServer Pages (cont'd) Directive

- instrucțiuni care indică informații generale despre pagina respectivă, afectând întreaga structură a clasei servlet
  - modul în care sunt gestionate aspecte ale procesării paginii JSP

```
<% directiveName attribute="value" %>
```

- număr variabil de perechi (atribut, valoare) separate prin virgulă
- trei tipuri
  - `<%@ page … %>` - atribute specifice paginii JSP (limbajul de programare pentru scripturi, pagina corespunzătoare erorilor, cerințe legate de stocarea temporară);
  - `<%@ include …%>` - include un fișier în cadrul etapei de transformare a paginii JSP în clasa Java Servlet corespunzătoare;
  - `<%@ taglib … %>` - definește o bibliotecă de etichete (conținând acțiuni definite de utilizator) utilizate în cadrul paginii.

Arhitectura unei aplicații Internet folosind JavaServer Pages (cont'd) Directiva page (1)

- informații cu privire la pagina JSP curentă

```
<%@ page attribute="value" %>
<jsp:directive.page attribute="value" />
```

- atribute suportate
  - `autoFlush` – comportamentul buffer-ului (zonei de memorie tampon) asociate servlet-ului în cazul în care se umple
    - `true` (implicit) – conținutul său este golit automat; atributul `buffer` specifică dimensiunea sa
    - `false` – se generează o eroare
  - `buffer` – model pentru fluxul de ieșire
    - none – răspunsul va fi transmis imediat
    - valoare ≠ 0 – răspunsul este construit într-o zonă de memorie tampon
  - `contentType` – schema de codificare a caracterelor pentru pagina JSP / răspunsul generat (exemple: `text/html:charset=ISO-8859-1`, `text/xml`, `application/msword`)
  - `errorPage` – URL către pagina care raportează excepțiile Java netratate, generate la rulare
  - `isErrorPage` – specifică dacă pagina JSP curentă este indicată de proprietatea `errorPage` a altei pagini JSP

Arhitectura unei aplicații Internet folosind JavaServer Pages (cont'd) Directiva page (2)

- atribute suportate
  - `extends` – specifică o superclasă pe care servlet-ul generat trebuie să o extindă
  - `import` – indică o listă de pachete / clase care vor fi utilizate în pagina JSP
    - automat, sunt importate `java.lang.*`, `javax.servlet.*`, `javax.servlet.jsp.*`, `javax.servlet.http.*`
    - mai multe pachete / clase trebuie separate prin virgulp
  - `info` – definește un șir de caractere care poate fi accesat prin metoda `getServletInfo()` a servlet-ului asociat
  - `isELIgnored` – specifică dacă expresiile EL (Expression Language) din pagina JSP vor fi evaluate (valoarea `true`, implicită) sau tratate ca text static (valoarea `false`)
  - `isScriptingEnabled` – determină dacă script-urile (scripleți, expresii non-EL, declarații) sunt permise pentru a fi evaluate (valoarea `true` – implicită) sau nu (valoarea `false`)
  - `isThreadSafe` – definește modelul firului de execuție pentru servlet-ul generat
    - valoarea `true` – paginile JSP sunt considerate a fi sigure pentru a fi folosite într-un context concurent
    - valoarea `false` – container-ul se asigură ca pagina să fie accesată de un singur client la un moment dat
  - `language` – definește limbajul de programare utilizat în pagina JSP
  - `session` – specifică dacă pagina JSP participă sau nu (valorile `true` / `false`) la sesiuni HTTP (poate accesa sau nu obiectul implicit `session` și metodele asociate acestuia)

Arhitectura unei aplicații Internet folosind JavaServer Pages (cont'd) Directivele `include` / `taglib`

- directiva `include` – include resursa specificată în fișierul curent, în cadrul **etapei de translatare** a paginii JSP în clasa Java Servlet corespunzătoare
  - se concatenează conținutul resurselor (indicate prin calea relativă) în locația unde a fost specificată (oriunde în pagina JSP)
  - utilă pentru resurse comune pentru toate paginile aplicației Internet: header, footer, meniuri

```
<%@ include file="myJSPPage.jsp" %>
<jsp:directive.include file="myJSPPage.jsp" />
```

- directiva `taglib` – permite definirea de către utilizator a unor etichete JSP care implementează o anumită funcționalitate (grupate în biblioteci) / utilizarea unor biblioteci standard (frecvent, JSTL)
  - `uri` – locația la care se află biblioteca respectivă
  - `preffix` – mecanism prin care vor fi identificate în cadrul paginii JSP

```
<%@ taglib uri="uri" prefix="prefixOfTag" %>
<jsp:directive.taglib uri="uri" prefix="prefixOfTag" />
```

Utilizare: `<prefixName:tagName ... />`

Arhitectura unei aplicații Internet folosind JavaServer Pages (cont'd) Acțiuni (1)

- marcaj care specifică comportamentul motorului Java Servlets de la nivelul serverului web, în **momentul compilării** marcajele fiind înlocuite cu codul sursă Java corespunzător
  - este de fapt o funcție predefinită

```
<jsp:actionName attribute="value" />
```

- atribute suportate
  - `id` – denumire care identifică în mod unic acțiunea, prin care aceasta poate fi referită în cadrul întregii pagini JSP
    - dacă este folosită pentru crearea unei instanțe a unui obiect, identificatorul poate fi folosit pe toată durata de viață a obiectului implicit `pageContext`;
  - `scope` – vizibilitatea elementului = posibilitatea de utilizare a identificatorului acestuia
    - `page`, `request`, `session`, `application`

Arhitectura unei aplicații Internet folosind JavaServer Pages (cont'd) Acțiuni (2)

- tipuri de acțiuni
  - `<jsp:element>` - definește un element XML în mod dinamic, acesta fiind format din
    - `<jsp:attribute>` - atribut
    - `<jsp:body>` - corp
  - `<jsp:useBean>` - dacă există un obiect vizibil cu identificatorul specificat îl localizează, altfel instanțiază componenta JavaBeans specificată
    - atribute
      - `class` – denumirea componentei JavaBeans (prefixată de denumirea pachetului)
      - `type` – tipul obiectului care va reda obiectul în cauză
      - `beanName` – denumirea componentei JavaBeans, așa cum este indicată de metoda `instantiate()` a clasei `java.beans.Beans`

Arhitectura unei aplicații Internet folosind JavaServer Pages (cont'd) Acțiuni (3)

- tipuri de acțiuni
  - `<jsp:useBean>`
    - `<jsp:getProperty>` - obținerea valorii proprietății unei componente Java Beans pe care o convertește la tipul șir de caractere și o integrează în fluxul de ieșire
      - `name` – componenta JavaBean din care face parte proprietatea (aceeași cu identificatorul din proprietatea `useBean`)
      - `property` – denumirea proprietății
    - `<jsp:setProperty>` - stabilirea valorii proprietății unei componente Java Bean care a fost anterior definită
      - `name` – componenta JavaBean din care face parte proprietatea (aceeași cu identificatorul din proprietatea `useBean`)
      - `property` – denumirea proprietății
      - `value` – valoarea care se dorește asignată proprietății în cauză (ignorată dacă valoarea este `null` sau parametrul nu există)
      - `param` – denumirea parametrului (din cadrul cererii) a cărui valoare va fi asignată proprietății în cauză

Arhitectura unei aplicații Internet folosind JavaServer Pages (cont'd) Acțiuni (4)

- tipuri de acțiuni
  - `<jsp:forward>` - cererea este delegată către o altă resursă (acesta putând fi pagină statică, pagină JSP, Java Servlet)
    - atributul `page` – URL relativ al resursei spre care se dorește a se realiza redirectarea
  - `<jsp:include>` - includerea resursei specificate în momentul în care **pagina JSP este solicitată**
    - atributul `page` – URL relativ al resursei care se dorește a fi inclusă
    - atributul `flush` – determină dacă zona de memorie tampon a resursei va fi golită înainte de operația de includere
  - `<jsp:plugin>` - generarea codului specific browser-ului (eticheta `<OBJECT>` sau `<EMBED>`) pentru o componentă Java (applet sau componentă Java Bean)
    - dacă plugin-ul necesar nu este prezent, acesta este descărcat
    - `<jsp:param>` - transmiterea de parametri
    - `<jsp:fallback>` - text afișat în cazul în care se produce o eroare
  - `<jsp:text>` - transcrierea unui text folosind un anumit format în pagini și documente JSP
    - corpul acțiunii poate conține doar text și expresii EL

Arhitectura unei aplicații Internet folosind JavaServer Pages (cont'd) Scripleți (1)


```
<% Java code fragment %>
<jsp:scriptlet>
   Java code fragment
</jsp:scriptlet>
```

- cod sursă Java (declarații de obiecte și metode, expresii)
  - plasat în metoda `service()` a servlet-ului corespunzător paginii respective
  - executat pe server atunci când se realizează o cere pentru pagina JSP care îl conține
- !!! de evitat
  - încalcă principiul de separare a logicii aplicației de nivelul de prezentare
  - aplicația Internet este mai dificil de întreținut

Arhitectura unei aplicații Internet folosind JavaServer Pages (cont'd) Scripleți (2)

- **expresie** = scriplet cuprins între <%= și %>, evaluat în momentul în care este executat servlet-ul și convertit la tipul șir de caractere
  - inserată în pagina JSP în locația în care este apelată (nu există restricții)
  - poate include orice expresie Java validă, fără a fi încheiată prin ;

```
<%= expression %>
<jsp:expression>
    expression
</jsp:expression>
```

- **declarație** = scriplet cuprins între <%! și %>, permițând definirea de obiecte sau de metode
  - domeniul de vizibilitate este întreaga pagină JSP
  - poate fi accesată din toate metodele clasei Java Servlet corespunzătoare

```
<%! declaration; [declaration; ]+ %>
<jsp:declaration>
    declaration;
    [declaration; ]+
</jsp:declaration>
```

Arhitectura unei aplicații Internet folosind JavaServer Pages (cont'd) Obiecte Implicite (1)

OBIECT IMPLICIT	TIP	DESCRIERE
`request`	subclasă a `javax.servlet.ServletRequest`	cererea care a invocat pagina JSP
`response`	subclasă a `javax.servlet.ServletResponse`	răspunsul pe care îl generează pagina JSP
`out`	`javax.servlet.jsp.JspWriter`	obiect care scrie în fluxul de ieşire
`session`	`javax.servlet.http.HttpSession`	obiect sesiune asociat cererii care a invocat pagina JSP
`application`	`javax.servlet.ServletContext`	context pagină JSP
`config`	`javax.servlet.ServletConfig`	informaţii de configurare referitoare la servlet
`pageContext`	`javax.servlet.jsp.PageContext`	contextul paginii JSP
`page`	`java.lang.Object`	referinţă către pagina JSP prin care poate fi accesat servlet-ul asociat
`exception`	`java.lang.Throwable`	acces la detalii cu privire la eroare, în paginile de tratare a excepţiei

Arhitectura unei aplicații Internet folosind JavaServer Pages (cont'd) Obiecte Implicite (2)

- pot fi folosite doar în cadrul metodei `_jspService()`
  - !!! nu pot fi utilizate într-o declarație
- funcționalitatea lor este identică ca în cazul clasei Java Servlet corespunzătoare
  - transmise ca parametrii ai metodei `service()`
  - obținute prin intermediul obiectelor `request` și `response`

Arhitectura unei aplicații Internet folosind JavaServer Pages (cont'd) Obiectul Implicit `request` (1)

- creat de fiecare dată când un client solicită o pagină JSP
- oferă metode pentru a obține informații cu privire la antetele HTTP
- derivat din `javax.servlet.http.HttpServletRequest`
- metode implementate
  - `Object getAttribute(String)` – valoarea atributului identificat prin denumirea sa
  - `Enumeration getAttributeNames()` – lista atributelor disponibile în cadrul cererii
  - `String getAuthType()` – denumirea mecanismului de autentificare utilizat pentru securizarea servletului (`null` dacă nu se folosește un astfel de mecanism)
  - `String getCharacterEncoding()` – denumirea schemei de codificare folosită pentru corpul cererii
  - `int getContentLength()` – dimensiunea (în octeți) a corpului cererii disponibilă în fluxul de intrare (-1 dacă nu este cunoscută)
  - `String getContentType()` – tipul MIME al corpului cererii (`null` dacă nu este cunoscut)
  - `String getContextPath()` – porțiunea din URI-ul cererii care indică contextul său
  - `Cookie[] getCookies()` – lista obiectelor de tip `Cookie` care au fost transmise de client împreună cu cererea sa

Arhitectura unei aplicații Internet folosind JavaServer Pages (cont'd) Obiectul Implicit `request` (2)

- metode implementate
  - `String getHeader(String)` – valoarea antetului identificat prin denumirea sa sub formă de șir de caractere
  - `Enumeration getHeaderNames()` – lista antetelor disponibile în cadrul cererii
  - `int getIntHeader(String)` – valoarea antetului identificat prin denumirea sa sub formă de întreg
  - `ServletInputStream getInputStream()` – întoarce corpul cererii (ca date binare)
  - `Locale getLocale()` – întoarce localizarea preferată a conținutului în funcție de valoarea conținută de antetul `Accept-Language`;
  - `String getMethod()` – întoarce numele metodei HTTP prin care a fost transmisă cererea (`GET`, `POST`, `PUT`, `OPTIONS`, `TRACE`, `DELETE`)
  - `String getParameter(String)` – valoarea parametrului identificat prin denumirea sa (`null` dacă nu există)
  - `Enumeration getParameterNames()` – lista parametrilor disponibili în cadrul cererii
  - `String[] getParameterValues(String)` – lista valorilor pe care le deține parametrul indicat prin denumirea sa (`null` dacă nu există)
  - `String getPathInfo()` – informații suplimentare refertore la cale asociate cu URL-ul atașat cererii
  - `String getProtocol()` – denumirea și versiunea protocolului prin care a fost transmisă cererea
  - `String getQueryString()` - șir de caractere reprezentând interogarea cuprinsă în URL după cale

Arhitectura unei aplicații Internet folosind JavaServer Pages (cont'd) Obiectul Implicit `request` (3)

- metode implementate
  - `String getRemoteAddr()` – adresa IP (Internet Protocol) a clientului care a transmis cererea
  - `String getRemoteHost()` – denumirea completă a mașinii de pe care clientul a transmis cererea
  - `String getRemoteUser()` – denumirea utilizatorului autentificat care a transmis cererea (`null` dacă utilizatorul nu este autentificat)
  - `String getRequestURI()` – porțiunea din URL-ul asociat cererii începând de la denumirea protocolului până la șirul ce reprezintă interogarea din cadrul cererii
  - `String getRequestSessionId()` – identificatorul sesiunii specificat de client în cerere
  - `int getServletPort()` – portul pe care a fost primită cererea
  - `String getServletPath()` – porțiunea din URL-ul asociat cererii prin care este invocată pagina JSP
  - `HttpSession getSession([boolean])` – sesiunea asociată cererii
    - dacă nu există o sesiune asociată cererii și parametrul are valoarea `true`, asociază cererii o sesiune nouă
  - `boolean isSecure()` – indică dacă cererea a fost transmisă folosind un canal sigur (de exemplu, HTTPS)

Arhitectura unei aplicații Internet folosind JavaServer Pages (cont'd) Obiectul Implicit request (4)


```
...
<table>
    <tr><th>Header Name</th><th>Header Value</th></tr>
    <%
        Enumeration headers = request.getHeaderNames();
        while (headers.hasMoreElements()) {
            String headerName = (String)headers.nextElement();
            String headerValue = request.getHeader(headerName);
    %>
        <tr><td><%= headerName %></td><td><%= headerValue %></td></tr>
    <%
        }
    %>
</table>
...
```

Arhitectura unei aplicații Internet folosind JavaServer Pages (cont'd) Obiectul Implicit `response` (1)

- creat de fiecare dată când un client primește o pagină JSP
- conține
  - starea (denumirea și versiunea protocolului, codul reprezentând starea și un mesaj de dimensiuni mici care o descrie)
  - antetele HTTP
  - documentul reprezentând pagina Internet care va fi afișată în browser
- derivat din `javax.servlet.http.HttpServletResponse`
- metode implementate
  - `void addCookie(Cookie)` – specifică un obiect `Cookie` pentru răspuns;
  - `void addDateHeader(String, long)` – specifică un antent pentru răspuns reprezentând o dată calendaristică furnizată ca parametru;
  - `void addHeader(String, String)` – specifică un antent pentru răspuns, transmițându-se denumirea și valoarea ca parametri;
  - `void addIntHeader(String, int)` – specifică un antent pentru răspuns reprezentând un întreg furnizat ca parametru;
  - `boolean containsHeader(String)` – verifică dacă un antent a fost specificat în cadrul răspunsului;

Arhitectura unei aplicații Internet folosind JavaServer Pages (cont'd) Obiectul Implicit `response` (2)

- metode implementate
  - `String encodeRedirectURL(String)` – codifică URL-ul furnizat ca parametru spre a fi utilizat de metoda `sendRedirect()` (lasându-l neschimbat dacă nu este necesară codificarea);
  - `String encodeURL(String)` - codifică URL-ul furnizat ca parametru incluzând identificatorul sesiunii (lasându-l neschimbat dacă nu este necesară codificarea);
  - `void flushBuffer()` – forțează afișarea conținutului zonei de memorie tampon în browser;
  - `boolean isCommitted()` – verifică dacă răspunsul a fost transmis către client;
  - `void reset()` – șterge toate informațiile din zona de memorie tampon (inclusiv coduri de stare și antete);
  - `void resetBuffer()` – șterge conținutul zonei de memorie tampon (lăsând intacte codurile de stare și antetele);
  - `void sendError(int[, String])` – transmite către client un răspund indicând o eroare (identificată printr-un cod și eventual un mesaj), ștergând conținutul zonei de memorie tampon;
  - `void sendRedirect(String)` – transmite către client un răspuns temporar de redirectare către locația indicată prin URL-ul furnizat ca parametru;
  - `void setBufferSize(int)` – specifică dimensiunea prefertaă pentru zona de memorie tampon care conține răspunsul (exprimată în octeți);
  - `void setCharacterEncoding(String)` – stabilește mecanismul de codificare al caracterelor (setul de caractere MIME) corespunzător răspunsului transmis către client;

Arhitectura unei aplicații Internet folosind JavaServer Pages (cont'd) Obiectul Implicit `response` (3)

- metode implementate
  - `void setContentLength()` – indică dimensiunea conținutului transmis către client ca răspuns (pentru protocolul HTTP aceasta corespunde valorii pe care o are antetul `Content-Length`);
  - `void setContentType(String)` – stabilește tipul de conținut corespunzptor răspunsul către client (dacă aceasta operație nu a fost realizată anterior);
  - `void setDateHeader(String, long)` – stabilește valoarea pe care o deține un antet reprezentând o dată calendaristică;
  - `void setHeader(String, String)` – stabilește valoarea pe care o deține un antet specificându-se denumirea și valoarea ca parametri;
  - `void setIntHeader(String, int)` – stabilește valoarea pe care o deține un antet reprezentând un întreg;
  - `void setLocale(Locale)` – stabilește localizarea răspunsului (dacă acesta nu a fost transmis către client);
  - `void setStatus(int)` – stabilește codul de stare pentru răspuns.

Arhitectura unei aplicații Internet folosind JavaServer Pages (cont'd) Obiectul Implicit response (4)


```
...
<%
    response.setDateHeader("Last-Modified",System.currentTimeMillis());
    response.setHeader("Content-Type","text/html");
    response.setIntHeader("Refresh", 60);
%>
...
```

- antetul `Last-Modified` primește ca valoare momentul curent
- antetul `Content-Type` indică faptul că este vorba despre un document HTML
- antetul `Refresh` face ca pagina să fie reîncărcată în mod automat la fiecare 60 de secunde

Arhitectura unei aplicații Internet folosind JavaServer Pages (cont'd) Obiectul Implicit out

- echivalentul obiectului `PrintWriter`
  - are tipul `JspWriter`
  - poate genera excepții de tipul `java.io.IOException`
  - oferă metode pentru gestiunea zonei de memorie tampon
- utilizat la afișarea conținutului dinamic în cadrul unei pagini JSP, când acest lucru nu este posibil prin intermediul unei etichete HTML
- instanțiat cu o valoare care depinde de valoarea atributului `buffered` a directivei `page`
- metode implementate
  - `print()` / `println()` – afișează conținutul unor obiecte de tip `boolean`, `char`, `String`, `int`, `long`, `float`, `double`, `Object`
  - `flush()` – golește fluxul de ieșire, transmițând conținutul zonei de memorie tampon către client

Arhitectura unei aplicații Internet folosind JavaServer Pages (cont'd) Obiectul Implicit `session` (1)

- creat în mod automat pentru fiecare client care accesează pagina (nu este necesar să fie obținut din obiectul `request` prin metoda `getSession()`)
- dezactivarea acestui mecanism se face prin valoarea atributului `session` din cadrul directivei `page`
- metode implementate (interfața `javax.servlet.http.HttpSession`)
  - `Object getAttribute(String)` – furnizează obiectul asociat denumirii transmise ca parametru (`null` dacă nu există nici un obiect asociat cu denumirea respectivă);
  - `Enumeration getAttributeNames()` – furnizează o listă conținând denumirile tututor obiectelor asociate sesiunii;
  - `long getCreationTime()` – întoarce momentul la care a fost creată sesiunea (exprimată în milisecunde trecute de la 1 ianuarie 1970 - GMT);
  - `String getId()` – întoarce un șir de caractere care identifică în mod unoc sesiunea;
  - `long getLastAccessedTime()` – furnizează momentul la care clientul a transmis ultima dată o cerere (exprimată în milisecunde trecute de la 1 ianuarie 1970 - GMT);
  - `int getMaxInactiveInterval()` – întoarce perioada de timp maximă în care container-ul va menține sesiunea activă, între accesări succesive venite de la clienți (exprimată în secunde);

Arhitectura unei aplicații Internet folosind JavaServer Pages (cont'd) Obiectul Implicit `session` (2)

- metode implementate (interfața `javax.servlet.http.HttpSession`)
  - `void invalidate()` – invalidează sesiunea (șterge orice asociere a unui obiect din cadrul acesteia);
  - `boolean isNew()` – indică dacă clientul are cunoștință de sesiune sau alege să nu participe la aceasta;
  - `void removeAttribute(String)` – șterge valoarea obiectului asociat în cadrul sesiunii prin intermediul denumirii furnizate ca parametru;
  - `void setAttribute(String, Object)` – asociază la sesiune un obiect ale cărui denumire și valoare sunt specificate ca parametrii ai metodei;
  - `void setMaxInactiveInterval(int)` – stabilește perioada de timp maximă (exprimată în secunde) între accesări succesive provenite de la clienți în care sesiunea este menținută de container-ul servlet-ului
    - implicit 30 de minute pentru Apache Tomcat 8.x
    - poate fi specificată în fișierul de configurare `web.xml` (exprimată însă în minute)

```
<session-config>
    <session-timeout>60</session-timeout>
</session-config>
```

Arhitectura unei aplicații Internet folosind JavaServer Pages (cont'd) Cookie-uri (1)

- fișiere text reținute pe mașina client și incluse de browser în cererea transmisă către server (în cadrul antetelor HTTP)
  - câmpul `Set-Cookie` conține perechi de tipul (cheie, valoare) pentru atributele `name`, `expires`, `path`, domain
  - în cazul în care clientul accesează o pagină care corespunde cu valorile reținute de atributele `path` și `domain`, dacă nu a fost depășită perioada specificată de atributul `expires`, cookie-ul va fi inclus în antetele HTTP asociate cererii
- metode implementate
  - `String getComment()` – furnizează comentariul care descrie scopul obiectului cookie (`null` dacă nu există nici un comentariu asociat);
  - `String getDomain()` – returnează domeniul pentru care se aplică obiectul cookie;
  - `int getMaxAge()` – întoarce durata de viață pentru obiectul cookie (exprimată în secunde);
    - valoarea -1 indică faptul că obiectul cookie va persista până când se va părăsi browserul;
  - `String getName()` – obține denumirea obiectului cookie (care nu mai poate fi modificată după crearea sa);

Arhitectura unei aplicații Internet folosind JavaServer Pages (cont'd) Cookie-uri (2)

- metode implementate
  - `String getPath()` – întoarce calea pentru care se aplică obiectul cookie;
  - `void setComment(String)` – stabilește un comentariu care descrie scopul obiectului cookie
    - util în cazul în care browser-ul îl afișează utilizatorului pentru a lua o decizie în privința sa
  - `void setDomain(String)` – indică domeniul pentru care se aplică obiectul cookie;
  - `void setMaxAge(int)` – precizează timpul (exprimat în secunde) care ar trebui să treacă înainte de expirarea obiectului cookie;
    - dacă nu este precizat, obiectul cookie va exista doar pe durata sesiunii curente
  - `void setPath(String)` – setează calea pentru care se aplică obiectul cookie;
    - dacă nu este precizată, obiectul cookie este transmis pentru toate URL-urile care se găsesc în același director (subdirector) cu pagina curentă
  - `void setSecure(boolean)` – determină dacă obiectul cookie ar trebui transmis sau nu numai prin legături criptate;
  - `void setValue(String)` – precizează valoarea asociată unui obiect cookie caracterizat prin denumirea sa furnizată ca parametru

Arhitectura unei aplicații Internet folosind JavaServer Pages (cont'd) Cookie-uri (3)

- utilizarea unui obiect cookie
  1. creare
     - denumire, valoare asociată
     - este interzisă folosirea caracterelor [,], (,), =, ,, ", /, ?, @, :, ;.
  2. precizarea duratei (maxime) de viață
  3. asocierea într-un obiect `response` prin metoda `addCookie()` pentru transmiterea sa către client
  4. ștergere = durată de viață nulă
- !!! nu toate mașinile client permit stocarea unor obiecte cookie (în funcție de configurările din browser)

Arhitectura unei aplicații Internet folosind JavaServer Pages (cont'd) Obiectul implicit `application`

- wrapper pentru obiectul `ServletContext` asociat paginii JSP
  - există pe parcursul întregului ciclu de viață
  - inițializat în metoda `_jspInit()` / distrus în metoda `_jspDestroy()`
- metode implementate
  - `Object getAttribute(String)`
  - `void setAttribute(String, Object)`
  - proprietățile conținute în denumire au un domeniu de viață corespunzător întregii aplicații (formată din mai multe pagini)
- poate fi folosită pentru determinarea **numărului de accesări** al aplicației
- persistența în cazul repornirii serverului este asigurată prin stocarea informațiilor în baza de date

Arhitectura unei aplicații Internet folosind JavaServer Pages (cont'd) Obiectul implicit `config`

- wrapper pentru obiectul `ServletConfig` asociat paginii JSP
- metode implementate
  - `String getInitParameter(String)` – permite accesarea valorii unui parametru de inițializare ai motorului Java Servlet / JavaServer Pages
  - `Enumeration getInitParameterNames()` – permite accesarea denumirii tuturor parametrilor de inițializare ai motorului Java Servlet / JavaServer Pages
  - `String getServletName()` – întoarce denumirea servlet-ului asociat, așa cum este precizat de proprietatea `<servlet-name>` din fișierul de configurare `web.xml`

Arhitectura unei aplicații Internet folosind JavaServer Pages (cont'd) Obiectul implicit pageContext

- reprezentare a întregii pagini JSP
- accesează informații cu privire la aceasta, evitând cele mai multe dintre detaliile de implementare
- conține
  - referințe către obiectele `request` / `response`
  - atribute reprezentând obiectele `config`, `session`, `out`
  - directive transmise în cadrul paginii JSP
- definește domeniile de vizibilitate
- implementează metodele moștenite din clasa `javax.servlet.jsp.JspContext`
- operații pe atribute (getter/ setter) + specificarea domeniului de vizibilitate

Arhitectura unei aplicații Internet folosind JavaServer Pages (cont'd) Obiectul implicit page

- referință către instanța paginii JSP curente
- sinonim pentru `this`

Arhitectura unei aplicații Internet folosind JavaServer Pages (cont'd) Obiectul implicit `exception` (1)

- wrapper care conține excepția generată de pagina anterioară, folosit pentru construirea unui răspuns adecvat la eroarea în cauză
- !!! **disponibil doar în cadrul paginilor JSP de eroare** (referite prin atributul `errorPage` al directivei `page`)
- metode implementate (interfața `java.lang.Throwable`)
  - `String getMessage()` – întoarce un mesaj detaliat cu privire la excepția care s-a produs (transmis în constructorul obiectului `Throwable`);
  - `Throwable getCause()` – indică cauza excepției;
  - `String toString()` – furnizează denumirea clasei concatenată cu mesajul corespunzător excepției care s-a produs;
  - `StackTraceElement[] getStackTrace()` – transmite un tablou conținând fiecare element al stivei de erori;
    - poziția 0 – vârful stivei de erori
    - ultimul element – baza stivei de erori (metoda de la apelul căreia a pornit eroarea)
  - `Throwable fillInStackTrace()` – completează stiva de erori a obiectului `Throwable` cu stiva de erori curentă;

Arhitectura unei aplicații Internet folosind JavaServer Pages (cont'd) Obiectul implicit `exception` (2)

- tipuri de excepții
  - verificate (eroare de utilizator, generată la compilare)
  - generate la rulare (ignorată la compilare, s-ar fi putut evita prin anumite mecanisme de prevenție)
  - eroare (depășește posibilitățile de intervenție)
- obiectul `pageContext` conține atributele
  - `exception` – acces la stiva de erori prin atributul `stackTrace`
  - `errorData`
    - `uri` – calea relativă a paginii JSP care a generat eroarea
    - `statusCode` – codul de stare corespunzător excepției
- alternativă – folosirea unui bloc `try { ...} catch` într-un scriplet cu tratarea erorii în aceași pagină JSP
  - !!! de evitat
  - utilă numai atunci când se dorește o recuperare rapidă din eroare

Implementarea unor funcționalități complexe Filtre (1)

- interceptarea cererilor de la client înainte ca acestea să acceseze anumite resurse de pe server
- gestionarea răspunsurilor generate de server înainte de a fi transmise către client
- implementarea interfeței `javax.servlet.Filter`
  - metoda

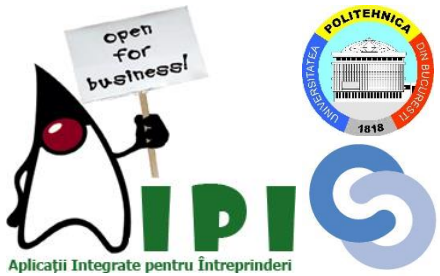
```
public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain) throws java.io.IOException, javax.servlet.ServletException
```

  - metodele `init()` / `destroy()`
- tipuri de filtre
  1. autentificare
  2. compresia datelor
  3. criptare
  4. declanșarea unor evenimente de acces a unor resurse
  5. conversia imaginilor
  6. jurnalizare
  7. auditare
  8. lanțuri de filtre MIME-TYPE
  9. parsare
  10. XSLT (transformarea conținutului XML)

Implementarea unor funcționalități complexe (cont'd) Filtre (2)

- fișierul de configurare `web.xml`
  - elementul `<filter>` - denumirea, clasa asociată, parametrii folosiți la inițializare
  - elementul `<filter-mapping>`
    - asocierea dintre un filtru și un Java Servlet / pagina JSP
    - ordinea de execuție a filtrelor (aceeași cu ordinea în care sunt mapate)
  - un filtru poate fi asociat mai multor resurse - masca `/*` folosită pentru elementul `<url-pattern>`

Implementarea unor funcționalități complexe (cont'd) Încărcarea unui fișier pe server



- formular
  - tipul de criptare (`enctype`) este `multipart/form-data`
  - controlul pentru încărcarea unui fișier este de tipul `<input type="file" … >`

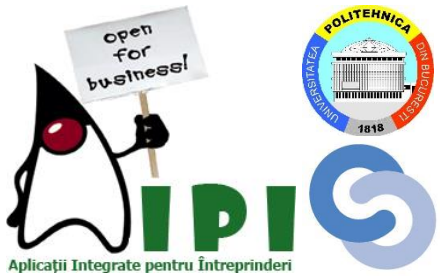
```
<form action="myJSPPage.jsp" method="POST" enctype="multipart/form-data">
    <input type="file" name="file" size="100" />
    <br />
    <input type="submit" value="Upload File" />
</form>
```

- locația la care vor fi încărcate resursele pe server este specificată prin parametrul file-upload în fișierul `web.xml`

```
<context-param>
    <param-name>file-upload</param-name>
    <param-value>
        My Upload Location
    </param-value>
</context-param>
```

- procesarea încărcării fișierului pe server se realizează prin intermediul bibliotecii commons-fileupload (http://commons.apache.org/proper/commons-fileupload/) dezvoltată de Apache: încărcarea mai multor fișiere simultan, obținerea de proprietăți ale resursei (cale absolută, denumire, dimensiune), citire din memorie / scriere pe disc

Implementarea unor funcționalități complexe (cont'd) Invocarea altor resurse



- redirectare
  - documentul a fost mutat la o altă locație
  - echilibrarea încărcării
- implementare
  - metoda `sendRedirect(String)` a obiectului `response` primește ca parametru URL-ul resursei către care se dorește a se realiza redirectarea
  - prin intermediul antetelor HTTP

```
response.setIntHeader("Status", response.SC_MOVED_TEMPORARILY);
response.setHeader("Location", myLocation);
```

Implementarea unor funcționalități complexe (cont'd) Operații cu date calendaristice

- obiecte de tip `java.util.Date`
  - constructor
    - fără parametri – data curentă
    - parametru de tip long (numărul de milisecunde trecute de la 1 ianuarie 1970 GMT)
  - metode implementate
    - `after(Date)`
    - `before(Date)`
    - `clone()`
    - `compareTo(Date[|Object])`
    - `equals(Object)`
    - `getTime()`
    - `hashCode()`
    - `setTime(long)`
    - `toString()`
  - poate fi formatată folosind clasa `SimpleDateFormat` (caractere codifică atribute ale obiectului `Date`)

Implementarea unor funcționalități complexe (cont'd) Poștă electronică (1)

- funcționalitate pusă la dispoziție de
  - JavaMail API (https://java.net/projects/javamail/pages/Home)
  - Java Activation Framework (http://www.oracle.com/technetwork/java/javamail/index-135046.html)
- transmiterea unui mesaj prin poșta electronică
  1. construirea unui obiect de tip Session
     - `mail.smtp.host` (obligatoriu)
     - `mail.user`, `mail.password` (dacă este necasară autentificarea)
  2. construirea unui obiect de tip `MimeMessage` (pe baza obiectului `Session`)
     - `void setFrom(InternetAddress)` – specifică câmpul `From:` pornind de la adresa de poștă electronică a destinatarului
     - `void addRecipients(Message.RecipientType, Address[])` – specifică câmpul `To:` al mesajului
       - tip: `Message.RecipientType.TO`, `Message.RecipientType.CC` (Carbon Copy), `Message.RecipientType.BCC` (Blind Carbon Copy)
       - vector de adrese – constructorul `InternetAddress` căruia i se transmite ca parametru adresa de poștă electronică a expeditorului
     - `void setSubject(String)` – specifică câmpul `Subject:` al mesajului
     - `void setText(String)` – specifică corpul mesajului

Implementarea unor funcționalități complexe (cont'd) Poștă electronică (2)

- transmiterea unui mesaj prin poșta electronică
  2. construirea unui obiect de tip `MimeMessage` (pe baza obiectului `Session`)
     - void setContent(String, String) – precizează corpul mesajului, primind ca parametri conținutul / formatul (documente având alt tip în afară de `plain-text`, de exemplu html/xml)
       - dimensiunea mesajului nu este limitată decât de serverul de poștă electronică
  3. transmiterea mesajului – `Transport.send()`
     - poate genera excepția `MessagingException` dacă mesajul nu a putut fi transmis din diverse motive
- includerea de atașamente – obiect `MimeMultiPart` la care se adaugă fragmente ale atașamentului respectiv (de tip `MimeBodyPart`) prin metoda `addBodyPart()`
  - `setText(String)` – indică corpul mesajului
  - `setDataHandler(DataHandler)` – specifică obiectul care se ocupă cu gestiunea atașamentului
    - se obține dintr-un obiect `FileDataSource()` construit pornind de la
      - denumirea fișierului în cauză
      - calea (relativă sau absolută) – dacă nu este inclusă se caută fișierul în directorul în care este plasată clasa Java Servlet / pagina JSP
  - `setFileName()` – specifică denumirea fișierului care conține atașamentul
  - `setContent()` – stabilirea faptului că tipul de conținut este `MimeMultiPart`

Implementarea unor funcționalități complexe (cont'd) Poștă electronică (3)


```
<%@ page import="javax.mail.*", "javax.mail.internet.*", "javax.activation.*" %>
<%
    Properties properties = System.getProperties();
    properties.setProperty("mail.smtp.host", "localhost");
    properties.setProperty("mail.user", myUser);
    properties.setProperty("mail.password", myPassword);
    Session mailSession = Session.getDefaultInstance(properties);
    try {
        MimeMessage mailMimeMessage = new MimeMessage(mailSession);
        mailMimeMessage.setFrom(new InternetAddress(from));
        mailMimeMessage.addRecipient(Message.RecipientType.TO, new InternetAddress(to));
        mailMimeMessage.setSubject("My Subject");
        Multipart mailMultipart = new MimeMultipart();
        BodyPart mailBodyPart;
        mailBodyPart = new MimeBodyPart();
        mailBodyPart.setText("My Mail Body");
        mailMultipart.addBodyPart(mailBodyPart);
        mailBodyPart = new MimeBodyPart();
        String filename = "myFileName.myExtension";
        mailBodyPart.setDataHandler(new DataHandler(new FileDataSource(filename)));
        mailBodyPart.setFileName(filename);
        mailMultiPart.addBodyPart(mailBodyPart);
        mailMimeMessage.setContent(mailMultiPart);
        Transport.send(mailMimeMessage);
    } catch (MessagingException exception) {
        System.out.println("An exception has occurred: "+exception.getMessage();
        if (Constants.DEBUG)
            exception.printStackTrace();
    }
%>
```

Utilizarea EL (Expression Language) în pagini JSP

- mecanism de comunicare între nivelul de prezentare și nivelul de logică a aplicației (componente Java Beans)
  - operații aritmetice și logice cu tipuri de date întregi, reale, șiruri de caractere, valori adevărat / fals, `null`
- funcționalitate
  - evaluare imediată și întârziată a unor expresii cu diferite operații;
  - apelarea metodelor getter / setter ale obiectelor instanță ale unor componente Java Beans;
  - utilizarea obiectelor implicite;
  - invocarea unor metode publice și statice;
- tipuri de expresii
  1. cu evaluare imediată (a căror valoare este stabilită instant) / cu evaluare întârziată (a căror valoare este determinată ulterior);
  2. expresii valoare (referă date) / expresii metodă (invocă metode);
  3. expresii rvalue (doar citesc date dintr-un obiect extern) / expresii lvalue (pot citi și scrie date din / într-un obiect extern);

Utilizarea EL (Expression Language) în pagini JSP (cont'd)

- **expresii cu evaluare imediată**
  - desemnate prin sintaxa `${expression}`
  - sunt evaluate înainte ca pagina Internet să fie afișată
  - folosite doar pentru a citi informații
  - utilizate doar în cadrul unor etichete care acceptă expresii a căror valoare este cunoscută în momentul rulării

```
<input type="submit" name="${inputName}" value="${inputValue}">
```

- **expresii cu evaluare întârziată**
  - desemnate prin sintaxa #{expression}
  - sunt evaluate mai târziu, prin mecanisme ce țin de tehnologia împreună cu care este folosită (pe parcursul ciclului de viață al paginii Internet) – folosite de regulă împreună cu JavaServer Faces, tehnologie ce are un ciclu de viață împărțit în mai multe faze, determinarea valorii unei expresii putând fi realizată doar la un anumit moment (după tratarea evenimentelor legate de componente / validarea datelor)
  - folosite atât pentru a citi cât și pentru a scrie informații
  - pot fi expresii valoare / expresii metodă

```
<table><tr><td>#{inputName}</td><td>#{inputValue}</td></tr></table>
```

Utilizarea EL (Expression Language) în pagini JSP (cont'd)

- expresiile EL pot referi
  - componente JavaBeans împreună cu atributele lor
    - `${entityBean.someAttribute}` – obține valoarea unui atribut din cadrul unei componente Java Beans, evaluându-l instant și transmițând rezultatul mai departe
    - `#{entityBean.someAttribute}` – eticheta în cadrul căreia este folosită poate decide evaluarea sa la un moment de timp ulterior
      - la solicitarea paginii Internet din browser – expresie rvalue
      - etapa de postback – expresie lvalue
  - colecții
  - structuri de date Java de tip enumerare – elemente referite folosind tipul `String`

`public enum EnumType {attribute`$_1$`, atrribute`$_2$`, …, attribute`$_n$`}`
`${"attribute`$_k$`"}` ⇔ `EnumType.attribute`$_k$

  - obiecte implicite
- evaluarea expresiei – rezultatul furnizat de `PageContext.findAttribute()`
  - domeniile de vizibilitate: `page`, `request`, `session`, `application`
  - `null` – dacă obiectul nu este găsit
  - motoare EL particularizate – modifică modul în care sunt evaluate expresiile

Utilizarea EL (Expression Language) în pagini JSP (cont'd)

- evaluarea atributelor unor componente Java Beans / instanțele unei enumerări
  - notațiile . și [] sunt echivalente

```
${myObject.myAttribute} =
${myObject['myAttribute']} =
${myObject["myAttribute"]}
```

  - șirurile de caractere pot fi incluse între apostroafe / ghilimele
  - !!! pot fi referite doar în situația în care este definită o metodă getter publică prin care poate fi accesat atributul
- evaluarea elementelor unui vector / ale unei liste – notația [] ce folosește valori literale / un întreg

```
// array or list: firstComponent can be narrowed to 0
${myObject.myAttribute[0]}
${myObject.myAttribute[firstComponent]}
```

- evaluarea elementelor de tip asociere – folosirea unei chei de tip șir de caractere

```
// map
${myObject.myAttribute["firstComponent"]}
```

Utilizarea EL (Expression Language) în pagini JSP (cont'd)

- tipuri de date
  - `Boolean`: `true` / `false`;
  - numere întregi / reale – folosite la fel ca în Java
  - `String` (șir de caractere)
    - referite cu apostroafe / ghilimele
    - \ este folosit pe post de caracter escape (" → \", ' → \', \ → \\)
  - `null`
- **expresii de tip valoare** – evaluate în contextul unui tip de date așteptat (se face conversie)
  - pot fi utilizate în text static / etichete (standard / definite de utilizator) care acceptă expresii (rvalue sau lvalue)
  - valoarea unei expresii în text static – evaluată și inclusă în pagina Internet

```
<my:tag>my text1 ${expression} my text2</my:tag>
```

  - atributul valoare al unei etichete poate fi inițializat folosind o expresie rvalue / lvalue

1. folosind o singură expresie – rezultatul este convertit la tipul atributului

```
<my:tag value="${expression}" />  / <my:tag value="#{expression}" />
```

2. folosind expresii compuse (una sau mai multe expresii separate sau înconjurate de text) – evaluate de la stânga la dreapta: expresiile sunt convertite la tipul String, concatenate cu alte texte, iar șirul de caractere este convertit la tipul atribulului

```
<my:tag value="my text1 ${expression1} my text2 ${expression2}" />
<my:tag value="my text1 #{expression1} my text2 #{expression2}" />
```

3. folosind expresii literale (doar text) – convertite la tipul atributului

```
<my:tag value="my text" />
```

Utilizarea EL (Expression Language) în pagini JSP (cont'd)

- **expresii de tip metodă**
  - permit invocarea unei metode implementate în cadrul unei componente Java Bean (care întoarce un rezultat)
  - folosite pentru a realiza procesări asociate cu controlul din contextul căruia sunt apelate (sunt invocate în momentul în care este generat un eveniment din cadrul acelui control)
  - !!! pot fi invocate pe parcursul diferitelor etape ale ciclului de viață = evaluare întârziată
  - !!! trebuie folosite doar în cadrul unei etichete
    - folosind o singură expresie – valoarea sa este transmisă modulului de gestiune a etichetei pentru a fi invocată mai târziu: rezultatul evaluării este de tip String, fiind converit la tipul de date așteptat
    - folosind doar text
  - pot primi 0, 1 sau mai mulți parametrii separați prin virgulă

```
myObject["myMethod"](myparameters)
myObject.myMethod(myParameters)
```

Utilizarea EL (Expression Language) în pagini JSP (cont'd)

- expresii lambda
  - expresii de tip valoare, comportament similar unei funcții
    - poate fi apelată imediat

```
() -> null
(x,y) -> (x+y)/2
((x,y) -> Math.sqrt(x*y))(4,9)
```

    - valoarea sa poate fi atribuită unei variabile care va fi utilizată pentru invocarea metodei respective (primind parametrii disponibili la momentul respectiv)

```
distance = (x1,y1,x2,y2) -> Math.sqrt(Math.sqr(x1-x2)+Math.sqr(y1-y2));
distance(1,1,2,2)
```

  - folosește operatorul `->`
  - stânga expresiei: parametrii expresiei lambda (incluse între (), putând fi omise dacă există un singur parametru)
  - dreapta expresiei: corpul expresiei lambda (o expresie EL)
  - pot fi transmise ca argumente al unei metode, executate în cadrul acestora sau incluse în alte expresii lambda

Utilizarea EL (Expression Language) în pagini JSP (cont'd)

- expresii literale
  - evaluate la textul expresiei, având tipul `String`
  - utilizate atunci când este necesară folosirea sintaxei rezervate `${}` sau `#{}`
  - nu folosește delimitatorii `${}` sau `#{}`
  - `${expression} = ${'${'}expression = \${expression}`
  - pot fi
    - cu evaluare imediată / întârziată – momentul la care este evaluată o expresie depinde de contextul în care este utilzată
    - expresii valoare / metodă

Utilizarea EL (Expression Language) în pagini JSP (cont'd)

- operații pe colecții
  - tipuri de colecții
    - mulțimi: `{1, 2, ..., n}`
    - liste (pot conține elemente având tipuri diferite): `{1, two, "3", ...}`
    - asocieri: `{"one":1, "two":2, ..., "n":n}`
  - utilizare – operațiile nu modifică colecția originală !!!
    - obiecte de tip flux, obținut prin metoda `stream()` a colecției respective
      - `java.util.Collection` / vector Java -> `Stream`
    - obiecte de tip benzi de asamblare: înlănțuirea metodelor care întorc doar fluxuri
      - sursă (obiect de tip `Stream`)
      - un număr de operații intermediare care întorc un flux
      - o operație terminală care nu întoarce un flux
  - metode suportate de fluxul de elemente: `allMatch()`, `anyMatch()`, `average()`, `count()`, `distinct()`, `filter()`, `findFirst()`, `flatMap()`, `forEach()`, `iterator()`, `limit()`, `map()`, `max()`, `min()`, `noneMatch()`, `peek()`, `reduce()`, `sorted()`, `substream()`, `sum()`, `toArray()`, `toList()`

```
collection.stream().filter(a->a.attribute1=='value1')
                   .map(a->a.attribute2)
                   .sorted()
                   .toList()
```

Utilizarea EL (Expression Language) în pagini JSP (cont'd)

Categorie Operator	Operator	Precedenţă	Observaţii
	`. []`	1	
	`()`		folosit pentru a schimba precedenţa operatorilor
aritmetici	`-` (unar)	2	
logici	`not !`	2	
empty	`empty`	2	utilizat pentru a verifica dacă o valoare este null sau vidă
aritmetici	`* / div % mod`	3	
	`+ -` (binar)	4	
concatenare şiruri de caractere	`+=`	5	
relaţionali	`<> < <= > >= lt le gt ge`	6	comparaţia se poate face cu alte valori sau literali
	`== != eq ne`	7	
logici	`&& and`	8	
	`|| or`	9	
condiţionali	`? :`	10	`X ? Y : Z`
lambda	`->`	11	
atribuire	`=`	12	
	`;`	13	

Utilizarea EL (Expression Language) în pagini JSP (cont'd)

- termeni rezervați (nu pot fi utilizați ca identificatori): `and`, `or`, `not`, `eq`, `ne`, `lt`, `gt`, `le`, `ge`, `true`, `false`, `null`, `instanceof`, `empty`, `div`, `mod`
- expresiile EL sunt ignorate dacă atributul `isELIgnored` al directivei `page` are valoarea `true`
  - implicit, valoarea sa este `false` (expresiile EL sunt evaluate într-o pagină JSP)

Utilizarea JSTL (JavaServer Pages Standard Tag Library)

- colecție de etichete JSP care implementează funcționalitatea de bază specifică pentru aplicații JSP
- clasificare
  1. etichete JSTL pentru funcționalitate de bază (controlul fluxului)
  2. etichete JSTL pentru formatare
  3. etichete JSTL pentru gestiunea informațiilor dintr-o bază de date SQL
  4. etichete JSTL pentru gestiunea documentelor XML
  5. funcţii JSTL
  6. etichete definite de utilizator
- biblioteca ce conține etichetele JSTL trebuie plasată în directorul `WEB-INF/lib` corespunzător aplicației web

Etichete JSTL pentru funcționalitate de bază (1)

- implementează funcționalitate legată de controlul fluxului
- cele mai frecvent utilizate
- trebuie precizată locația de unde pot fi încărcate definițiile

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

- tipuri de etichete
  - **`<c:out>`** - evaluează valoarea unei expresii pe care o afișează
    - atribute
      - `value` (obligatoriu) – informația care trebuie evaluată;
      - `default` – informație asociată în cazul în care evaluarea eșuează;
      - `escapeXml` – utilizată pentru a ignora caracterele XML speciale;
    - exemplu: `<c:out value="${myObject.myAttribute}" />`
  - **`<c:set>`** - asociază rezultatul evaluării unei expresii la o variabilă, disponibilă pentru un anumit domeniu de vizibilitate
    - atribute
      - `value` – informația care trebuie evaluată și asociată obiectului;
      - `target` – denumirea obiectului a cărui proprietate trebuie stabilită;
      - `property` – proprietate a obiectului `target` a cărui valoare va fi stabilită la valoarea indicată (trebuie specificat dacă se folosește `target`);
      - `var` – denumirea variabilei care va stoca informația;
      - `scope` – domeniul de vizibilitate al variabilei care va stoca informația;
    - exemplu:

```
<c:set value="${expression}" target="myObject" property="myAttribute" scope="page" />
<c:set value="${expression}" var="myVariabile" scope="application" />
```

Etichete JSTL pentru funcționalitate de bază (2)

- tipuri de etichete
  - **<c:remove>** - disociază o variabilă de contextul unui anumit domeniu de vizibilitateyddswq
    - atribute
      - `var` (obligatoriu) – denumirea variabilei care va fi eliminată;
      - `scope` – domeniul de vizibilitate de variabilei care va fi eliminată;
    - exemplu: `<c:remove var="myVariable" scope="application" />`
  - **<c:catch>** - gestionează orice obiect de tip `Throwable` întânit în corpul său, eventual expunându-l utilizatorului
    - atribute
      - `var` (obligatoriu) – indică denumirea variabilei care va stoca excepția generată, în cazul în care va fi produsă de instrucțiunile din corpul său;
    - exemplu:

```
<c:catch var="${exception}">
    <% some expression(s) that may potentially throw an error %>
</c:catch>
<c:out value="exception: ${exception.message}" />
```

Etichete JSTL pentru funcționalitate de bază (3)

- tipuri de etichete
  - **`<c:if>`** - etichetă de tip condițional, al cărui corp este evaluat în cazul în care condiția asociată este îndeplinită
    - atribute
      - `test` (obligatoriu) – condiția care se dorește a fi evaluată;
      - `var` – denumirea variabilei care va stoca rezultatul condiției;
      - `scope` – domeniul de vizibilitate al variabilei care va stoca rezultatul condiției;
    - exemplu:

```
<c:if test="${expression}" />
    <% do some operations %>
</c:if>
```

  - **`<c:choose>`** - etichetă de tip condițional, indicând condiții care se exclud mutual, specificate prin etichete `<c:when>` și `<c:otherwise>`, ce funcționează ca instrucțiunea `switch` din Java, permițând alegerea dintre mai multe alternative (ramurile `case` sunt speficiate prin `<c:when>`, iar ramura `default` este specificată prin `<c:otherwise>`)
    - atribute: nu are
    - exemplu:

```
<c:choose>
    <c:when test="${expression1}"><% do some operations %></c:when>
    <c:when test="${expression2}"><% do some operations %></c:when>
    ...
    <c:when test="${expressionn}">
        <% do some operations %>
    </c:when>
    <c:otherwise><% do some other operations %></c:otherwise>
</c:choose />
```

Etichete JSTL pentru funcționalitate de bază (4)

- tipuri de etichete
  - **<c:when>** - subetichetă a <c:choose> al cărui corp este evaluat în cazul în care condiția asociată este îndeplinită
    - atribute
      - test – condiția care se dorește a fi evaluată;
  - **<c:otherwise>** - subetichetă a <c:choose> ce urmează uneia sau mai multor etichete <c:when>, determinând evaluarea corpului său, dacă nici una dintre condițiile care o preced nu este îndeplinită
    - atribute: nu are
  - **<c:import>** - primește un URL relativ sau absolut și expune conținutul său întregii pagini unui șir de caractere specificat în atributul var sau unui obiect Reader specificat în atributul varReader
    - atribute
      - url (obligatoriu) – URL-ul ce trebuie obținut și inclus în pagină;
      - context – caracterul / urmat de numele aplicației web;
      - charEncoding – setul de caracter utilizat pentru datele prelucrate;
      - var – denumirea variabilei utilizată pentru a stoca textul preluat;
      - scope – domeniul de vizibilitate al variabilei utilizată pentru a stoca textul preluat;
      - varReader – denumirea unei variabile alternative pentru a expune conținutul sub forma unui obiect java.io.Reader;
    - exemplu: `<c:import var="myVariabile" url="http://www.mysite.com" />`

Etichete JSTL pentru funcționalitate de bază (5)

- tipuri de etichete
  - **<c:forEach>** - etichetă de tip iterație, acceptând mai multe tipuri de colecții, suportând parcurgerea submulțimelor (alternativă pentru instrucțiunile de iterare `for`, `while` și `do-while` din limbajul de programare Java care pot fi utilizate în cadrul unui scriplet)
    - atribute
      - `items` – informația (colecția) pe care se realizează iterația;
      - `begin` – elementul colecției de la care se începe iterația;
      - `end` – elementul colecției la care se încheie iterația;
      - `step` – pasul cu care se parcurg elementele colecției;
      - `var` – denumirea variabilei care expune elementul curent;
      - `varStatus` – denumirea variabilei care expune statutul iterației;
    - exemplu:

```
<c:forEach var="myVariabile" items="${myCollection}">
    <c:out value="${myVariabile}" /><br />
</c:forEach>
```

  - `<c:forTokens>` - iterează asupra unor particule, separate prin delimitatori
    - atribute: aceleași ca `<c:forEach>` + `delim` (indică caracterele utilizate ca delimitatori)
    - exemplu:

```
<c:forTokens var="myVariabile" items="${myString}" delim=",">
    <c:out value="${myVariabile}" /><br />
</c:forTokens>
```

Etichete JSTL pentru funcționalitate de bază (6)

- tipuri de etichete
  - **`<c:param>`** - adaugă un parametru la URL-ul specificat în cadrul unei etichete `<c:import>`, realizând și codificarea acestuia
    - atribute
      - `name` – denumirea parametrului vare va fi inclus în URL;
      - `value` – valoarea parametrului vare va fi inclus în URL;
    - exemplu:

```
<c:import var="myVariabile" url="http://www.mysite.com">
    <c:param name="param1Name" value="param1Value" />
    <c:param name="param2Name" value="param2Value" />
    ...
    <c:param name="paramnName" value="paramnValue" />
</c:import>
```

  - **`<c:redirect>`** - redirectează controlul la un alt URL, realizând suprascrierea sa în browser (suportă URL-uri relative la context și eticheta `<c:param>`)
    - atribute
      - `url` (obligatoriu) – URL-ul la care se realizează redirectarea controlului;
      - `context` – caracterul / urmat de numele aplicației web;
    - exemplu:

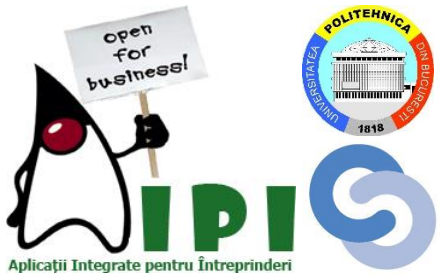
```
<c:redirect url="http://www.mysite.com">
```

Etichete JSTL pentru funcționalitate de bază (7)

- tipuri de etichete
  - **`<c:url>`** - creează un URL cu parametrii de interogare opționali, stocând valoarea acestuia într-o variabilă și realizând rescrierea URL-ului dacă este necesar (alternativă la invocarea metodei `responde.encodeURL()`, suportând și eticheta `<c:param>`)
    - atribute
      - `value` (obligatoriu) – URL-ul de bază;
      - `context` – caracterul / urmat de denumirea aplicației web;
      - `var` – denumirea variabilei ce reține URL-ul procesat;
      - `scope` – domeniul de vizibilitate al variabilei ce reține URL-ul procesat;
    - exemplu:

```
<c:url value="http://www.mysite.com" />
```

Etichete JSTL pentru formatarea conținutului (1)



- folosite pentru a afișa într-un format localizat
  - text
  - informații cu privire la data calendaristică și timp
  - numere
  - site-uri Internet
- trebuie precizată locația de unde pot fi încărcate definițiile

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

- tipuri de etichete
  - **`<fmt:formatNumber>`** - redarea unei valori numerice cu o anumită precizie sau format
    - atribute
      - `value` (obligatoriu) – valoarea numerică ce se dorește afișată;
      - `type` – poate lua valorile `NUMBER`, `CURRENCY`, `PERCENT`;
      - `pattern` – modelul de formatare pentru valoarea numerică ce se dorește a fi afișată;
      - `currencyCode` – codul valutei (pentru tipul `CURRENCY`) preluat din localizarea curentă;
      - `currencySymbol` – simbolul valutei (pentru tipul `CURRENCY`) preluat din localizarea curentă;
      - `groupingUsed` – dacă se folosesc grupuri de numere, se folosește pentru a introduce caracterul , între grupurile care reprezintă miile;
      - `maxIntegerDigits` – numărul maxim de cifre întregi care pot fi afișate (rezultatul se trunchiază dacă valoarea este depășită);
      - `minIntegerDigits` – numprul minim de cifre întregi care pot fi afișate;
      - `maxFractionDigits` – numărul maxim de zecimale care pot fi afișate (rezultatul se rotunjește dacă valoarea este depășită);
      - `minFractionDigits` – numărul minim de zecimale care pot fi afișate;
      - `var` – denumirea variabilei care reține valoarea formatată;
      - `scope` – domeniul de vizibilitate al variabilei care reține valoarea formatată;

Etichete JSTL pentru formatarea conținutului (2)

- tipuri de etichete
  - **<fmt:parseNumber>** - parsează reprezentarea sub formă de șir de caractere pentru o valoare numerică, o valută sau un procentaj
    - atribute
      - `value` (obligatoriu) – valoare numerică ce se dorește a fi parsată;
      - `type` – poate lua valorile `NUMBER`, `CURRENCY`, `PERCENT`;
      - `parseLocale` – localizarea folosită la parsarea numărului;
      - `integerOnly` – indică parsarea de valori întregi sau reale;
      - `pattern` – modelul de parsare;
      - `timeZone` – zona de timp a datei calendaristice parsate;
      - `var` – denumirea variabilei care reține valoarea parsată;
      - `scope` – domeniul de vizibilitate al variabilei care reține valoarea parsată;
  - **<fmt:formatDate>** - utilizat pentru formatarea unei date calendaristice / timp folosind stilurile și modelele oferite
    - atribute
      - `value` (obligatoriu) – valoare numerică ce se dorește a fi parsată;
      - `type` – poate lua valorile `DATE`, `TIME`, `BOTH`;
      - `dateStyle` – poate lua valorile `FULL`, `LONG`, `MEDIUM`, `SHORT`, `DEFAULT`;
      - `timeStyle` – poate lua valorile `FULL`, `LONG`, `MEDIUM`, `SHORT`, `DEFAULT`;
      - `pattern` – modelul de formatare pentru valoarea de timp dată calendaristică / timp ce se dorește a fi afișată;
      - `timeZone` – zona de timp a datei calendaristice afișate;
      - `var` – denumirea variabilei care reține valoarea formatată;
      - `scope` – domeniul de vizibilitate al variabilei care reține valoarea formatată;

Etichete JSTL pentru formatarea conținutului (3)

- tipuri de etichete
  - **<fmt:parseDate>** - parsează reprezentarea sub formă de șir de caractere pentru o valoare de tip dată calendaristică / timp
    - atribute: aceleași ca <fmt:formatDate> + parseLocale, indicând localizarea care trebuie utilizată când se parsează informația de tip dată calendaristică
  - **<fmt:bundle>** - încarcă o resursă ce urmează să fie utilizată în corpul său, prin eticheta <fmt:message>
    - atribute
      - basename (obligatoriu) – specifică numele de bază pentru resursa care se dorește a fi încărcată;
      - prefix – valoarea care va preceda fiecare denumire de etichetă în subeticheta <fmt:message>;
    - exemple

```
<fmt:bundle basename="myBaseName">
    <fmt:message key="myPrefix.myKey" />
</fmt:bundle>
```

echivalent cu

```
<fmt:bundle basename="myBaseName" prefix="myPrefix">
    <fmt:message key="myKey" />
</fmt:bundle>
```

Etichete JSTL pentru formatarea conținutului (4)

- tipuri de etichete
  - **<fmt:setLocale>** - stochează localizarea precizată prin variabila de configurare referitoare la localizare
    - atribute
      - `value` (obligatoriu) – specifică un cod format din două părți
        - ✓ codul de limbă (ISO-639);
        - ✓ codul de țară (ISO-3166);
      - `variant` – variantă specifică browser-ului;
      - `scope` – domeniul de vizibilitate al variabilei de configurare referitoare la localizare
  - **<fmt:setBundle>** - încarcă o resursă pe care o stochează într-o variabilă având un anumit domeniu de vizibilitate, respectiv în variabila de configurare referitoare la resurse
    - atribute
      - `basename` (obligatoriu) – specifică numele de bază pentru familia de resurse pentru a fi expusă ca variabilă având un anumit domeniu de vizibilitate, respectiv ca variabilă de configurare;
      - `var` – danumirea variabilei care reține valoarea resursei;
      - `scope` – domeniul de vizibilitate al variabilei care reține valoarea resursei;
  - **<fmt:timeZone>** - specifică zona de timp pentru orice tip de acțiune vizând formatarea sau parsarea din corpul său
    - atribute
      - `value` (obligatoriu) – indică zona de timp care va fi aplicată corpului său

Etichete JSTL pentru formatarea conținutului (5)

- tipuri de etichete
  - **<fmt:setTimeZone>** - stochează zona de timp în variabila de configurare referitoare la zona de timp
    - atribute
      - `value` (obligatoriu) – zona de timp care va fi expusă ca o variabilă având un anumit domeniu de vizibilitate, respectiv ca o variabilă de configurare;
      - `var` – denumirea variabilei care reține zona de timp;
      - `scope` – domeniul de vizibilitate al variabilei care reține zona de timp;
  - **<fmt:message>** - afișează un mesaj folosind un format localizat
    - atribute
      - `key` – cheia mesajului care se dorește a fi obținut;
      - `bundle` – identificatorul resursei care se dorește a fi folosită;
      - `var` – denumirea variabilei care reține valoarea mesajului localizat;
      - `scope` – domeniul de vizibilitate al variabilei care reține valoarea mesajului localizat;
  - **<fmt:requestEncoding>** - stabilește mecanismul de codificare al caracterelor pentru cerere;
    - atribute
      - `key` (obligatoriu) – indică mecanismul de configurare al caracterelor care va fi aplicat la decodificarea parametrilor din obiectul `request`;

Etichete JSTL pentru gestiunea informațiilor dintr-o bază de date SQL (1)

- permite interacțiunea cu baze de date relaționale cum ar fi Oracle, MySQL, Microsoft SQL Server
- trebuie precizată locația de unde pot fi încărcate definițiile

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

- tipuri de etichete
  - **`<sql:setDataSource>`** - creează un obiect DataSource adecvat numai operațiilor de prototipare
    - atribute
      - `driver` – clasa corespunzătoare "driver"-ului JDBC ce trebuie înregistrat;
      - `url` – URL-ul pentru realizarea conexiunii la baza de date;
      - `user` – numele de utilizator folosit în mecanismul de autentificare pentru baza de date;
      - `password` – parola folosită în mecanismul de autentificare pentru baza de date;
      - `dataSource` – baza de date pregătită anterior;
      - `var` – denumirea variabilei care va reprezenta baza de date (**va fi folosită ca sursă de date pentru interogările realizate în cadrul paginii JSP**);
      - `scope` – domeniul de vizibilitate al variabilei care va reprezenta baza de date
    - exemplu:

```
<sql:setDataSource var="connection" url="${dataBaseConnection}"
user="${dataBaseUser}" password="${dataBasePassword}" />
```

Etichete JSTL pentru gestiunea informațiilor dintr-o bază de date SQL (2)

- tipuri de etichete
  - **<sql:query>** - execută interogarea SQL de tip SELECT definită în cadrul corpului, reținând rezultatul într-o variabilă având un anumit domeniu de vizibilitate
    - atribute
      - `sql`– comanda SQL care va fi executată (ar trebui să întoacă un rezultat de tip `ResultSet`);
      - `dataSource` – conexiunea la baza de date care va fi utilizată (suprascrie valoarea implicită);
      - `maxRows` – numărul maxim de rezultate care vor fi stocate în cadrul variabilei;
      - `startRow` – numărul înregistrării (din cadrul rezultatului) de la care va începe stocarea;
      - `var` – denumirea variabilei care va reține rezultatul;
      - `scope` – domeniul de vizibilitate al variabilei care reține rezultatul;
    - exemplu: `<sql:query sql="${myQuery}" var="result" />`
  - **<sql:update>** - execută interogarea SQL de actualizare a informațiilor (INSERT, UPDATE, DELETE) definită în cadrul corpului, care nu întorce rezultate
    - atribute
      - `sql`– comanda SQL care va fi executată (ar trebui să NU întoacă un rezultat de tip `ResultSet`);
      - `dataSource` – conexiunea la baza de date care va fi utilizată (suprascrie valoarea implicită);
      - `var` – denumirea variabilei care va reține rezultatul (numărul de înregistrări afectate);
      - `scope` – domeniul de vizibilitate al variabilei care reține rezultatul (numărul de înregistrări afectate)
    - exemplu:

```
<sql:update dataSource="${connection}" var="count">
    <%-- some INSERT, UPDATE or DELETE query -->
</sql:update>
```

Etichete JSTL pentru gestiunea informațiilor dintr-o bază de date SQL (3)

- tipuri de etichete
  - **<sql:param>** - stabilește valoarea unui parametru al unei instrucțiuni SQL, fiind folosit de obicei împreună cu etichetele `<sql:query>` sau `<sql:update>`
    - atribute
      - `value` – valoarea care se dorește a fi asociată parametrului
    - exemplu:

```
<sql:query dataSource="${connection}" var="result">
    SELECT FROM table WHERE attribute = ?
    <sql:param value="${missingTableAttributeValue}" />
</sql:query>
<sql:update dataSource="${connection}" var="count">
    <%-- some INSERT, UPDATE or DELETE query with one or more
         missing attribute values -->
    <sql:param value="${missingTableAttributeValue}" />
</sql:update>
```

  - **<sql:dateParam>** - stabilește valoarea unui parametru al unei instrucțiuni SQL având tipul `java.util.Date`
    - atribute
      - `value` – valoarea care se dorește a fi asociată parametrului, având tipul `java.util.Date`
      - `type` – poate avea valorile `DATE` (doar date calendaristice), `TIME` (doar timp), `TIMESTAMP` (dată calendaristică și timp)
    - exemplu: similară cu eticheta `<sql:param>`

Etichete JSTL pentru gestiunea informațiilor dintr-o bază de date SQL (4)

- tipuri de etichete
  - **<sql:transaction>** - execută interogări (desemnate de etichetele <sql:query> și <sql:update>) folosind un obiect Connection partajat, executate în cadrul unei tranzacții, asigurând ca efectul lor să fie unitar asupra bazei de date (menținându-se starea sa inițială în cazul producerii unei erori)
    - atribute
      - dataSource – conexiunea la baza de date care va fi utilizată (suprascrie valoarea implicită);
      - isolation – poate avea valorile READ_COMMITED, READ_UNCOMMITTED, REPETEABLE_READ, SERIALIZABLE;
    - exemplu:

```
<sql:transaction dataSource="${connection}">
    <sql:update var="count1">
        <%-- some INSERT queries -->
    </sql:update>
    <sql:update var="count2">
        <%-- some UPDATE queries -->
    </sql:update>
    <sql:update var="count3">
        <%-- some DELETE queries -->
    </sql:update>
</sql:transaction>
```

Etichete JSTL pentru gestiunea documentelor XML (1)

- permite manipularea documentelor XML
- operații
  - parsare
  - transformare
  - controlul fluxului bazat pe expresii XPath
- trebuie precizată locația de unde pot fi încărcate definițiile

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

- tipuri de etichete
  - **`<x:out>`** - evaluează valoarea unei expresii XPath pe care o afișează
    - atribute
      - `select` (obligatoriu) – expresia XPath care trebuie evaluată, adeseori folosind variabile XPath;
      - `escapeXml` – indică dacă caracterele speciale XML ar trebui ignorate;

Etichete JSTL pentru gestiunea documentelor XML (2)

- tipuri de etichete
  - **<x:parse>** - parsează date XML specificate fie printr-un atribut, fie în corpul etichetei
    - atribute
      - `var` – variabila care conține datele XML parsate;
      - `xml` – textul documentului care se dorește a fi parsat (de tip `String` sau `Reader`);
      - `systemId` – URI-ul identificatorului de sistem pentru a parsa documentul;
      - `filter` – filtrul ce trebuie aplicat documentului sursă;
      - `doc` – documentul XML ce se dorește a fi parsat;
      - `scope` – domeniul de divizibilitate al variabilei specificate de atributul `var`;
      - `varDom` – variabila care conține datele XML parsate;
      - `scopeDom` – domeniul de divizibilitate al variabilei specificate de atributul `varDom`;
    - exemplu:
```
<x:parse xml="${myXmlDocument}" var="variable" />
<x:out select="$variable/root/element[index]/attribute" />
```
  - **<x:set>** - atribuie unei variabile valoarea unei expresii XPath; în funcție de tipul rezultat la evaluarea expresiei XPath, vor fi create obiecte având tipul `java.lang.Boolean`, `java.lang.String`, `java.lang.Number`
    - atribute
      - `var` (obligatoriu) – variabila care va reține valoarea expresiei XPath;
      - `select` – expresia XPath care se dorește evaluată;
      - `scope` – domeniul de vizibilitate al variabilei specificate de atributul `var`;
    - exemplu:
```
<x:parse xml="${myXmlDocument}" var="variable1" />
<x:set var="variable2" select="$variable1/root/element" />
```

Etichete JSTL pentru gestiunea documentelor XML (3)

- tipuri de etichete
  - **<x:if>** - evaluează o condiție care are forma unei expresii XPath și dacă aceasta se verifică se procesează corpul etichetei
    - atribute
      - `select` (obligatoriu) – expresia XPath exprimând condiția care se dorește evaluată;
      - `var` – variabila care va reține rezultatul evaluării condiției;
      - `scope` – domeniul de vizibilitate al variabilei specificate de atributul `var`;
    - exemplu:

```
<x:parse xml="${myXmlDocument}" var="variable" />
<x:if select="$variable/root/element[index]/attribute != 0" />
    The attribute of the element at index is not null!
</x:if>
```

  - **<x:forEach>** - iterează asupra nodurilor unui document XML
    - atribute
      - `select` (obligatoriu) – expresia XPath exprimând condiția care va fi evaluată;
      - `var` – denumirea variabilei care va reține valoarea elementului curent pentru fiecare iterație;
      - `begin` – indexul de început pentru iterație;
      - `end` – indexul de sfârșit pentru iterație;
      - `step` – valoarea cu care va fi incrementat indexul pe parcursul incrementării colecției;
      - `varStatus` – denumirea variabilei în care se va reține statutul iterației;
    - exemplu:

```
<x:parse xml="${myXmlDocument}" var="variable" />
<x:forEach select="$variable/root/element/attribute" var="item" />
    <x:out select="$item" />
</x:forEach>
```

Etichete JSTL pentru gestiunea documentelor XML (4)

- tipuri de etichete
  - **<x:choose>** - etichetă de tip condițional care stabilește un context pentru operații condiționale care se exclud mutual, marcate prin <x:when> și <x:otherwise>
    - atribute: nu are
    - exemplu:

```
<x:parse xml="${myXmlDocument}" var="variable" />
<x:choose>
    <x:when select="$variable//element/attribute eq value1">
        Attribute has value 1!
    </x:when>
    ...
    <x:when select="$variable//element/attribute eq valuen">
        Attribute has value n!
    </x:when>
    <x:otherwise>
        Attribute has unknown value!
    </x:otherwise>
</x:choose>
```

  - **<x:when>** - subetichetă a etichetei x:choose al cărei corp este inclus dacă se verifică condiția asociată; are un singur atribut, `select`, care conține condiția ce se dorește a fi evaluată
  - **<x:otherwise>** - subetichetă a etichetei <x:choose> ce urmează uneia sau mai multor etichetei <x:when> determinând evaluarea corpului său, dacă nici una din condițiile care o preced nu este îndeplinită

Etichete JSTL pentru gestiunea documentelor XML (5)

- tipuri de etichete
  - **<x:transform>** - se aplică la transformarea XSL a unui document XML
    - atribute
      - `doc` – documentul XML sursă pentru transformarea XSLT;
      - `docSystemId` – URI-ul documentului XML original;
      - `xslt` (obligatoriu) – foaia de stil XSLT care conține instrucțiunile pentru transformare;
      - `xsltSystemId` – URI-ul documentului XSLT original;
      - `result` – obiectul rezultat care va accepta rezultatul transformării;
      - `var` – denumirea variabilei în care se va reține rezultatul transformării;
      - `scope` – domeniul de vizibilitate ce va expune rezultatul transformării;
    - exemplu:

```
<x:parse xml="${myXmlDocument}" var="variable" />
<c:import url="http://www.mysite.com/document.xsl" var="xslt" />
<x:transform doc="${variable}" xslt="${xslt}" />
```

  - **<x:param>** - folosit împreună cu eticheta de transformare spre a stabili un parametru în foaia de stil XSLT
    - atribute
      - `name` – denumirea parametrului XSLT ce se dorește a fi stabilit;
      - `value` – valoarea parametrului XSLT ce se dorește a fi stabilit;
    - exemplu:

```
<x:transform doc="${variable}" xslt="${xslt}" />
    <x:param name="attributeName" value="attributeValue" />
</x:transform>
```

Funcții JSTL (1)

- în special pentru manipularea șirurilor de caractere
- trebuie precizată locația de unde pot fi încărcate definițiile

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

- tipuri de metode
  - `fn:contains(stringToBeSearched, substringToSearch)` – verifică dacă un șir de caractere conține un alt subșir (cele două valori de tip `java.lang.String` fiind transmise ca parametri); rezultatul este de tip `boolean`;
  - `fn:containsIgnoreCase(stringToBeSearched, substringToSearch)` - verifică dacă un șir de caractere conține un alt subșir, ignorând capitalizarea acestora (cele două valori de tip `java.lang.String` fiind transmise ca parametri); rezultatul este de tip `boolean`;
  - `fn:endsWith(stringToBeSearched, suffix)` - verifică dacă un șir de caractere conține un anumit sufix (cele două valori de tip `java.lang.String` fiind transmise ca parametri); rezultatul este de tip `boolean`;
  - `fn:escapeXml(stringToBeXmlEscaped)` – transformă caracterele care ar putea fi interpretate ca adnotări XML la valoarea corespunzătpare, pentru un șir de caractere (având tipul `java.lang.String`) transmis ca parametru; rezultatul este de tip `java.lang.String`;
  - `fn:indexOf(stringToBeSearched, substringToSearch)` – întoarce poziția la care este găsit un subșir în cadrul unui șir de caractere (cele două valori de tip `java.lang.String` fiind transmise ca parametri); rezultatul este de tip `int` (se întoarce valoarea -1 dacă șirul de caractere nu este găsit);

Funcții JSTL (2)

- tipuri de metode
  - `fn:join(arrayToBeJoined, separator)` – concatenează toate elementele unui vector (parametru de tip `java.lang.String[]`) folosind un anumit separator (parametru de tip `java.lang.String`); rezultatul este de tip `java.lang.String`;
  - `fn:length(someObject)` – întoarce numărul de elemente al colecției sau dimensiunea șirului de caractere (parametru de tip `java.lang.Object`); întoarce un rezultat de tip `int`;
  - `fn:replace(someString, sequenceToReplace, sequenceToReplaceWith)` – întoarce un șir de caractere obținut prin înlocuirea unei anumie secvențe din cadrul șirului transmis ca parametru cu o altă secvență (parametri de tip `java.lang.String`); întoarce un rezultat de tip `boolean`;
  - `fn:split(someString, delimiter)` – împarte un șir de caractere transmis ca parametru (având tipul `java.lang.String`) într-un vector de subșiruri de caractere pe baza unui delimitator transmis ca parametru (având tipul `java.lang.String`); rezultatul este de tip `java.lang.String[]`
  - `fn:startsWith(stringToBeSearch, prefix)` – verifică dacă un șir de caractere conține un anumit prefix (cele două valori de tip `java.lang.String` fiind transmise ca parametri); rezultatul este de tip `boolean`;
  - `fn:substring(someString, startIndex, endIndex)` – întoarce un subșir al șirului de caractere transmis ca parametru (având tipul `java.lang.String`), cuprins între pozițiile de început și sfârșit precizate (parametrii de tip `int`); rezultatul este de tip `java.lang.String`

Funcții JSTL (3)

- tipuri de metode
  - `fn:substringAfter(someString, subSequence)` – întoarce un subșir al șirului de caractere transmis ca parametru (având tipul `java.lang.String`), ce succede unei anumite subsecvențe a sa (parametru de tip `java.lang.String`); rezultatul este de tip `java.lang.String`
  - `fn:substringBefore(someString, subSequence)` – întoarce un subșir al șirului de caractere transmis ca parametru (având tipul `java.lang.String`), ce precede o anumită subsecvențe a sa (parametru de tip `java.lang.String`); rezultatul este de tip `java.lang.String`
  - `fn:toLowerCase(someString)` – transformă toate caracterele șirului de caractere transmis ca parametru (având tipul `java.lang.String`) în minuscule; rezultatul este de tip `java.lang.String`
  - `fn:toUpperCase(someString)` - transformă toate caracterele șirului de caractere transmis ca parametru (având tipul `java.lang.String`) în majuscule; rezultatul este de tip `java.lang.String`
  - `fn:trim(someString)` – întoarce un șir de caractere obținut prin eliminarea spațiilor care se găsesc la începutul și sfârșitul șirului de caractere transmis ca parametru (având tipul `java.lang.String`); rezultatul este de tip `java.lang.String`

Etichete definite de utilizator

- în momentul transformării servlet-ului corespunzător sunt covertite în operațiile specificate de clasa asociată
- clasa care definește funcționalitatea etichetei
  - derivată din `javax.servlet.jsp.tagtext.SimpleTagSupport`
  - implementează metoda `doTag()`
    - obiectele din cadrul paginii JSP pot fi accesate prin intermediul metodei `getJSPContext()`
    - valorile atributelor din cadrul etichetei pot fi obținute într-un obiect de tip `StringWriter` prin metoda `getJspBody().invoke()`
  - trebuie plasată într-o locație vizibilă prin variabila de mediu `CLASSPATH`
- operațiile sunt executate în momentul în care se accesează pagina respectivă

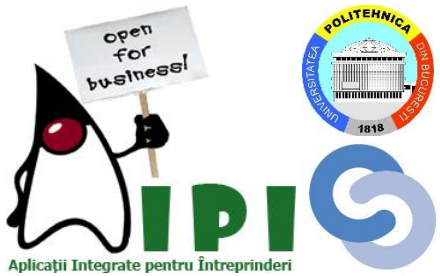
Etichete definite de utilizator (cont'd)


```
import java.io.*;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
public class CustomTag extends SimpleTagSupport {
    private String customAttribute;
    public void setCustomAttribute(String customAttribute) {
        this.customAttribute = customAttribute;
    }
    public String getCustomAttribute() {
        return customAttribute;
    }
    StringWriter stringWriter = new StringWriter();
    public void doTag() throws JspException, IOException {
        if (customAttribute != null) {
            // process tag attribute
            getJspContext().getOut().println(customAttribute);
        } else {
            // get the content from the tag body
            getJspBody().invoke(stringWriter);
            // process tag body's content
            getJspContext().getOut().println(stringWriter.toString());
        }
    }
}
```

Etichete definite de utilizator (cont'd)

- asocierea dintre etichetă și clasa asociată se face în fișierul `custom.tld` plasat în directorul `webapps/root/WEB-INF` al serverului Apache Tomcat 8.x
  - `<name>` - denumirea etichetei, așa cum va fi utilizată în pagina JSP
  - `<tag-class>` - denumirea clasei care implementează funcționalitatea
  - `<body-content>` - tipul de conținut al corpului etichetei
    - valori posibile
      - `empty` – conținut vid
      - `scriptless` – este acceptat numai text static, expresii EL sau alte etichete
      - `tagdependent` – conținutul este scris în alt limbaj, fiind interpretat de implementarea etichetei
  - pentru fiecare atribut trebuie specificate proprietățile
    - `<name>` - identificator unic al atributului în cadrul etichetei
    - `<required>` - distinge între atributele obligatorii și cele opționale
    - `<rtexprvalue>` - stabilește validitatea valorii unei expresii definite în momentul rulării pentru atributul etichetei
    - `<type>` - tipul Java al atributului (implicit, `java.lang.String`)
    - `<description>`
    - `<fragment>` - indică dacă valoarea asociată atributului va fi tratată ca un `JspFragment`

Etichete definite de utilizator (cont'd) Fișierul `custom.tld`




```
<taglib>
  <tlib-version>1.0</tlib-version>
  <jsp-version>2.0</jsp-version>
  <short-name>A short description of my custom tag</short-name>
  <tag>
    <name>CustomTag</name>
    <tag-class>mypackage.CustomTag</tag-class>
    <body-content>scriptless</body-content>
    <attribute>
        <name>customAttribute</name>
        <required>true</true>
        <type>java.lang.Integer</type>
        <fragment>false</fragment>
    </attribute>
  </tag>
</taglib>
```

Etichete definite de utilizator (cont'd)

- utilizare
  - declararea locației din care vor fi descărcate definițiile etichetei

```
<%@ taglib prefix="customPrefix" uri="WEB-INF/custom.tld"%>
```

  - denumirea etichetei (din fișierul `custom.tld`) trebuie prefixată cu identificatorul utilizat la declararea sa
    - trebuie indicate valori pentru toate atributele care au proprietatea `required`

```
...
<customPrefix:CustomTag customAttribute="0">
    Some custom tag body
</customPrefix:CustomTag>
...
```

- se pot defini mai multe biblioteci de etichete, grupate în funcție de tipul de funcționalitate pe care o implementează
  - plasate în fișiere diferite
  - utilizându-se prefixe diferite