Sessão Técnica

Introdução às bibliotecas JSTL (JSP Standard Tag Libraries)

Felipe de Almeida Leme
Michael Nascimento Santos

Autores

- **Felipe de Almeida Leme (AKA felipeal)**
  - Engenheiro de Computação (UNICAMP) com mais de 6 anos de experiência em Java
  - 3 anos de experiência no exterior (California)
  - Sun Certified Programmer for Java 1.1 Releases
  - Consultor Sênior para a Trust Consultores; instrutor Java para a SeedTS
  - Colaborador regular da revista Java Magazine
  - Palestrante no JavaOne 2003
  - Membro individual do JCP

Autores

- **Michael Nascimento Santos (AKA Mister M)**
  - Sun Certified Programmer for the Java 2 Platform 1.2 & 1.4
  - Sun Certified Web Component Developer for J2EE.
  - Membro da organização do SouJava, responsável geral pela moderação da java-list
  - Contribuidor para a JavaMagazine
  - Membro individual do JCP e Expert da JSR-207
  - 4 anos de experiência com Java; 3 com tecnologias J2EE
  - Palestrante no JavaOne 2003

Introdução

- Evolução da forma de trabalho com JSPs. JSP puro:

```
<%@ page import="java.sql.*,java.text" %>
<%try {
  Connection con= DriverManager.getConnection(...);
  ...
  ResultSet rs = st.executeQuery("SELECT a.field1, a.field2...
   FROM Tabela a");
%>
<table>
  <tr>
     <td>Usuario</td><td>Data de cadastro</td>....
  </tr>
<% DateFormat df = new SimpleDateFormat("dd/MM/yyyy");
   while (rs.next()) {%>
  <tr>
     <td><%= rs.getString(1) %></td>
     <td><%= df.format(rs.getTimestamp(1)) %></td>...
```

Introdução

- Evolução da forma de trabalho com JSPs:
  - JSP puro
  - Taglibs (JSP 1.1)
  - JSTL (JSP 1.2/2.0)

JSP Puro

```
<%@ page import="java.sql.*,java.text" %>

<%try {
  Connection con= DriverManager.getConnection(...);
  ...
  ResultSet rs = st.executeQuery("SELECT a.field1, a.field2...
   FROM Tabela a");
%>
<table>
  <tr>
     <td>Usuario</td><td>Data de cadastro</td>....
  </tr>
<% DateFormat df = new SimpleDateFormat("dd/MM/yyyy");
   while (rs.next()) {%>
  <tr>
     <td><%= rs.getString(1) %></td>
     <td><%= df.format(rs.getTimestamp(1)) %></td>...
  </tr>
```

JSP Puro

- Vantagens
  - Facilidade
  - Velocidade
  - Praticidade
  - Centralização
- Desvantagens
  - Baixa legibilidade
  - Dificuldade de manutenção
  - Baixo encapsulamento / reuso
  - Mau tratamento de exceções

Taglibs (JSP 1.1)

- Uma maneira de abstrair funcionalidade usada por uma página JSP por defini-la de uma forma mais natural dentro de um JSP:

```
<%@ taglib uri="http://www.justjava.com.br" prefix="tg" %>
<tg:useConnection var="con" dataSource="<%= request.getAttribute("ds") %>">
  <tg:query var="rs">SELECT a.field1, a.field2... FROM Tabela a</tg:query>
 <table>
     <tr>
        <td>Usuario</td><td>Data de cadastro</td>....
     </tr>
     <tg:loopResults var="<%= rs %>" line="l">
      <tr>
         <td><%= line.getString(1) %></td>
         <td><tg:format value="<%= line.getTimestamp(1) %>"/></td>...
      </tr>
     ....
 </table>
</tg:use>
```

Taglibs (JSP 1.1)

- Vantagens
  - Melhor separação entre código Java e HTML
  - Encapsulamento / reuso
  - Facilidade de manutenção
  - Bom tratamento de exceções
  - Integração com IDEs
- Desvantagens
  - Complexidade na implementação
    - Código da Tag => API complexa
    - Deployment Descriptor via TLD
  - Scriplets necessários com certa freqüência
  - Tempo de desenvolvimento
  - Pouca estrutura pré-construída para tarefas básicas
  - Única implementação

JSTL (JSP 1.2/2.0)

- Conjunto de taglibs prontas, nova linguagem para scripts e classes auxiliares que facilitam o desenvolvimento de JSPs.

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jstl/sql" prefix="sql" %>
<%@ taglib uri="http://java.sun.com/jstl/sql" prefix="fmt" %>
<sql:query var="usuarios" dataSource="${dataSource}">
        SELECT a.field1, a.field2... FROM Tabela a
</sql:query>
<table>
  <tr><td>Usuario</td><td>Data de cadastro</td>...</tr>
<c:forEach var="linha" items="${usuarios.rows}">
  <tr>
    <td><c:out value="${linha.nome}"/></td>
    <td><f:formatDate value="${linha.dataCadastro}" /></td>...
  </tr>
</c:forEach>
</table>
```

JSTL (JSP 1.2/2.0)

- Vantagens
  - Facilidade de implementação
  - Expression Language (EL)
  - Tempo de desenvolvimento
  - Estrutura básica pronta
  - Padrão da plataforma
  - Mútiplas implementações possíveis
- Desvantagens
  - Dependência de aderência ao padrão JavaBeans de propriedades
  - Não permite invocação de métodos
  - EL só pode ser usada em tags

Visão Geral do JSTL

- Expression Language (EL)
- Bibliotecas de tags
  - Core
  - Format
  - SQL
  - XML
- Taglib Validators (TLV)
- API

EL

- Substitui scriplets

```
<%=((Usuario)request.getAttribute("usuario")).getGrupo().getNome()%>
```

```
<c:out value="${usuario.grupo.nome}" />
```

- Simples
- Sem casts
- Sem necessidade de busca explícita em escopos (`page`, `request`, `session`, `application`)

Core

- Propósito geral (`out, set, remove, catch`)
  - Manipulação de variáveis de escopo e tratamento de erros
- Condicionais (`if, choose, when, otherwise`)
  - Facilitam processamento condicional em páginas JSP
- Manipulação de URLs (`import, url, redirect, param`)
  - Criar links, importar e redirecionar para URLs é uma tarefa comum, tediosa e sujeita a erros; tags simplificam e uniformizam trabalho.

Core

- Iteração (`forEach, forTokens`)
  - Iteragir com coleções exige diferentes estruturas, conhecimentos de Java:

```
<% Object[] arr = ..;
     for (int i = 0; i < arr.length; i++) { %>
     ...
    <%= arr[i] %>
<%   } %>
<% Collection c = ..;
     for (Iterator i = c.iterator(); i.hasNext();) { %>
     ...
    <%= i.next() %>
<% } %>
```

Core

- Com JSTL, fica transparente e uniforme:

```
<c:forEach var="item" items="${colecao}">
    ...
    <c:out value="${item}" />
</c:forEach>
```

Format

- I18n (`setLocale, bundle, setBundle, message, param, requestEncoding`)
  - Ajudam a gerar mensagens e saídas de acordo com o idioma e país do usuário
- Formatação (`timeZone, setTimeZone, formatNumber, parseNumber, formatDate, parseDate`)
  - Formatam números, datas e horas de acordo com o idioma e país do usuário e/ou de forma configurável

SQL

- SQL (`query, update, transaction, setDataSource, param, dateParam`)
- Permitem:
  - Trabalhar com DataSources e/ou Connections
  - Realizar consultas
  - Acessar resultados de consultas facilmente
  - Realizar inserções, atualizações e remoções de dados
  - Realizar várias operações de forma transacional

XML

- Core (`parse, out, set`)
  - Manipulam XML de forma simples, exibindo valor de elementos, atributos etc. e/ou definindo variáveis
- Fluxo (`if, choose, when, otherwise, forEach`)
  - Permitem manipular o XML e executar ou não ações com base em expressõs XPath
- Transformação - XSLT (`transform, param`)
  - Facilitam o uso de XSLT

Validators

- Garantem que os JSPs obedecem certas restrições
  - JSPs sem scriplets (ScriptFreeTLV)
    - Lançam exceções de compilação caso o JSP use scripts
  - Limitação de uso das tags (PermittedTaglibsTLV)
    - Lançam exceções de compilação caso o JSP use tags não permitidas

API

- Classes e interfaces necessárias para interagir ou estender a JSTL
  - Loops (LoopTagStatus, LoopTag, LoopTagSupport)
    - Permitem implementar tags de loop de forma simples e obter informações de uma iteração
  - I18n (LocaleSupport, LocalizationContext)
    - Dão suporte para implementação de tags internacionalizáveis
  - SQL (Result, ResultSupport, SQLExecutionTag)
    - Permitem trabalhar com resultados de queries e desenvolver novas tags relacionadas ao SQL

Expression Language

- Nova maneira de se avaliarem expressões em JSTL

(maneira atual: RT – Request Time)

- Disponível no JSTL (em atributos de tags)
- Estará disponível no JSP 2.0 (em qualquer lugar)
- Facilita acesso a objetos e métodos, pois dispensa uso de scriplets JSP
- Uso da convenção JavaBeans para acessar propriedades
- Exemplo:

JSP: `<%= usuario.getNome() %>`

EL: `${usuario.nome}`

Sintaxe

- Expressões delimitadas por **${}**
- Expressões podem ser concatenadas. Exemplo:

```
"${usuario.nome} ${usuario.sobreNome}"
```

- Identificadores resolvidos como:
  - variáveis de escopo (`page`, `request`, `session` ou `aplication`)
  - variáveis de visibilidade de tag
  - objetos implícitos (maior precedência)
- Conversão automáticas entre tipos
- Tolerância a **NullPointerException**

Objetos Implícitos

Nome	Tipo	Descrição	Equivalente JSP	Exemplo
pageContext	PageContext	Objeto PageContext da página JSP	PageContext	${pageContext}
pageScope	Map	Atributos da página (escopo *page*)	pageContext.findAtribute(nome, PAGE_SCOPE)	${pageScope.usuario}
requestScope	Map	Atributos da requisição (escopo *request*)	pageContext.findAtribute(nome, REQUEST_SCOPE)	${requestScope.usuario}
sessionScope	Map	Atributos da sessão (escopo *session*)	pageContext.findAtribute(nome, REQUEST_SCOPE)	${sessionScope.usuario}
applicationScope	Map	Atributos da aplicação (escopo *application*)	pageContext.findAtribute(nome, REQUEST_SCOPE)	${applicationScope.usuario}
param	Map	Valores dos parâmetros do *request*	request.getParameter(nome)	${param['nomeUsuario']}
paramValues	Map	Valores (múltiplos) dos parâmetros do *request*	request.getParameterValues(nome)	${param['fotoId']}
header	Map	Valores dos atributos do *header* HTTP	request.getHeader(nome)	${header['host']}
headerValues	Map	Valores (múltiplos) dos atributos do *header* HTTP	request.getHeaderValues(nome)	${headerValues['host'][0].}
cookies	Map	*Cookies* do *request*	request.getCookies()	${cookie.ultimoAcesso}
initParam	Map	Parâmetros de inicialização da página	pageContext.getServletContext(). getInitParameter(nome)	${initParam.mailSuporte}

Operadores de Acesso (Acessors)

- Dois tipos de operadores
  - . - acesso à propriedades de objetos
  - [ ] - acesso a índices de coleções
- Intercambiáveis
- Cascateáveis
- Exemplos

```
${param[“nomeUsuario”]}          // pagina.jsp?nomeUsuario=fulano
${paramValues[“autor”][0]}       // palestra.jsp?autor=Felipe&autor=Michael
${usuario.endereco.rua}          // <%= usuario.getEndereco().getRua() %>
${usuarios[5].endereco['rua']}   // <%= usuarios[5].getEndereco().getRua() %>
```

Outros Operadores

- Aritméticos

  `+ - * / div % mod`

  Relacionais

  `== eq != ne < lt > gt <= le >= ge`

- Lógicos

  `&& and || or ! not empty`

- **Empty**: usado para determinar se um valor é vazio ou `null`

- Exemplos

```
${ projeto.numeroHoras * tabelas['projeto'].fatorRisco }
${(x gt 0) && (x < 10)}
${ !(empty usuario.endereco) }
```

Precedência de Operadores

1)[ ] .

2)( )

3)-(unário) not ! empty

4)* / div % mod

5)+ - (binário)

6)< > <= >= lt gt le ge

7)== != eq ne

8)&& and

9)|| or

Biblioteca Core

- Manipulação de objetos
- Tratamento de exceções
- Fluxo condicional
- Iterações
- URLs

<c:out>

Descrição: avalia uma expressão e exibe o resultado

Atributos:

- `value` – valor a ser avaliado
- `default` – valor (opcional) a ser retornado caso `value` seja vazio ou `null`
- `escapeXML` – flag (opcional) indicando se caracteres especiais (<, >, “ e ') devem ser convertidos (default = true)

Corpo da tag: atributo `default`

Exemplos de <c:out>

```
<c:out value="literal"/>

<c:out value="${usuario.nome}" default="guest"/>

<c:out value="${usuario.nome}">
guest
</c:out>

<c:out value="" escapeXml="false">
Exemplo do uso de <i>escapeXml</i>.
</c:out>
```

<c:set>

Descrição: define o valor de uma variável (ou propriedade de um objeto)

Atributos:

- `var` – nome da variável
- `value` – valor a ser atribuído à variável
- `scope` – escopo (opcional) da variável (`page` – valor default, `request`, `session` ou `aplication`)
- `target` e `property` - objeto e propriedade (ao invés de `var`)

Corpo da tag: atributo `value`

Exemplos de <c:set>

```
 <c:set var="nome" value="Felipe"/><br>
Variável nome: <c:out value=”${nome}”/><br>


<c:set target="${usuario}" property="nome"
value="${nome}"/>
Propriedade nome do objeto usuário:  
<c:out value=”${usuario.nome}”/><br>
```

<c:remove>

Descrição: remove uma variável de escopo

Atributos:

- `var` – nome da variável a ser removida
- `scope` – escopo (opcional) da variável

Corpo da tag: não utilizado

Exemplo de <c:remove>

```
Antes da remoção: <c:out value="${nome}"/><br>
<c:remove var="nome"/>
<c:out value="${nome}" escapeXml="false">
Variável <i>nome</i> foi removida do escopo!
</c:out>
<br>
```

<c:catch>

Descrição: captura uma exceção ocorrida no corpo da tag

Atributos:

- `var` – variável (opcional), de escopo de página (`page`), que receberá a exceção. Caso não seja definida, a exceção será ignorada.

Corpo da tag: bloco (html/jsp/tags) cujas exceções serão capturadas pela tag

Exemplo de <c:catch>

```
<%-- O seguinte trecho gera exceção, já que a
     propriedade NOME não existe na classe usuario
--%>
<c:catch var=”excecao”>
<c:set target="${usuario}" property="NOME"
value="${nome}"/>
</c:catch>
Exceção: 
<c:out value=”${excecao}”>nenhuma</c:out>
<br>
```

<c:if>

Descrição: executa o corpo da tag apenas se a expressão de teste for verdadeira

Atributos:

- `test` – expressão a ser testada
- `var` – variável (opcional) a receber o valor da expressão
- `scope` – escopo (opcional) da variável
- `target` e `property` - objeto e propriedade (ao invés de `var`)

Corpo da tag: bloco a ser executado se a expressão teste for verdadeira

Exemplos de <c:if>

```
 <c:if test="${usuario.nome == 'Felipe' ||
usuario.nome == 'Michael'}">
 Permissão ok!<br>
 </c:if>


 <c:if test="${usuario.nome == 'Felipe'}"
var="acesso">
 Permissão ok!<br>
 </c:if>
 Acesso: <c:out value="${acesso}"/><br>
```

<c:choose>, <c:when> e <c:otherwise>

Descrição: usados para condições de múltiplas escolha

Atributos:

- `test` – condição a ser testada (**<c:when>**)

Corpo das tags:

<c:choose> - tags **<c:when>** (1+) e **<c:otherwise>** (0-1)

<c:when> - bloco a ser executado quando condição teste for verdadeira

<c:otherwise> - bloco a ser executado caso nenhum **<c:when>** seja aplicado

Exemplo de <c:choose>...

```
 <c:choose>
<c:when ”${usuario.nome == 'Felipe'}”>
Permissão ok!<br>
</c:when>
<c:when ”${usuario.nome == 'Michael'}”>
Permissão ok!<br>
</c:when>
<c:otherwise>
Permissão negada!<br>
</c:otherwise>
</c:choose>
```

<c:forEach>

Descrição: executa o corpo da tag repetidas vezes (iterações)

Atributos:

- `items` – coleção (opcional) a ser iterada
- `var` – variável a receber cada valor da iteração
- `varStatus` – variável a receber status de cada iteração
- `begin` e `end` – índice dos primeiro e último passos da iteração
- `step` – frequência das iterações

Corpo da tag: bloco a ser executado em cada iteração

Interface *LoopTagStatus*

getCurrent() - objeto sendo acessado na iteração atual
getIndex***()*** - índice da iteração atual (começando em 0)
getCount() - contagem do número de passos até a iteração atual (começando em 1)
isFirst() - flag indicando se a iteração atual é a primeira
isLast() - flag indicando se a iteração atual é a última
getBegin() - valor do atributo `begin` (ou `null` se atributo não foi especificado)
getEnd() - valor do atributo `end` (ou `null` se atributo não foi especificado)
getStep() - valor do atributo `step` (ou `null` se atributo não foi especificado)

Exemplos de <c:forEach>

```
<table border="1" cellspacing="0" cellpading="0">
<c:forEach items="${pageContext.request.headerNames}"
   var="parametro" varStatus="status">
<tr>
    <td><c:out value="${status.count}"/></td>
    <td><c:out value="${parametro}"/></td>
    <td><c:out value="${header[parametro]}"/></td>
</tr>
</c:forEach></table>

<ul>
<c:forEach var="i" begin="1" end="10">
<li><c:out value="${i}"/>
</c:forEach>
</ul>
```

<c:forTokens>

Descrição: iteração sobre os *tokens* de uma **String**

Atributos:

- `items` – **String** com os *tokens*
- `delims` – delimitadores dos *tokens*
- `begin`, `end`, `varStatus` e `step` - mesmo significado que em **<c:forEach>**

Corpo da tag: bloco a ser executado em cada iteração

Exemplo de <c:forTokens>

```
 <table>
<c:forTokens items="a,b;c,d,e,f;g;h" delims=",;"
   var="token" varStatus="status">
<tr>
    <td><c:out value="${status.count}"/></td>
    <td><c:out value="${token}"/></td>
</tr>
</c:forTokens>
</table>
```

<c:param>

Descrição: sub-tag (de **<c:url>**, **<c:redirect>** ou **<c:import>** que define um parâmetro de URL

Atributos:

- `name` – nome do parâmetro
- `value` – valor do parâmetro

Corpo da tag: atributo `value`

<c:url>

Descrição: cria uma **String** representando uma URL

Atributos:

- `value` – valor da URL (relativa ou absoluta)
- `var` – variável (opcional - caso omitido, URL será impressa na página) que receberá a URL
- `scope` – escopo (opcional) da variável
- `context` – contexto (opcional) da URL

Corpo da tag: tags **<c:param>** (opcionais)

Exemplos de <c:url>

```
 URL direto na página: 
<c:url value="hello.jsp"/><br>


URL com parâmetros atribuída a uma variável: 
<c:url value="/hello.jsp" var="url">
  <c:param name="nome" value="Felipe"/>
  <c:param name="sobreNome" value="Leme"/>
</c:url>
<c:out value='${url}'/>
<br>
```

<c:redirect>

Descrição: redireciona o cliente (browser) para outra URL (através de um comando HTTP redirect)

Atributos:

- `url` – URL a ser redirecionada
- `context` – contexto (opcional) da URL

Corpo da tag: tags **<c:param>** (opcionais)

Exemplos de <c:redirect>

```
<c:redirect url=”http://www.justjava.com.br"/>


<c:redirect url=”/showUser.jsp" context=”teste”>
   <c:param name=”user” value=”felipeal”/>
</c:redirect>
```

<c:import>

Descrição: importa o conteúdo de uma URL

Atributos:

- `value, var, scope` e `context` – mesmo significado que em **<c:url>**
- `varReader` – variável do tipo **java.io.Reader** que receberá a URL
- `charEncoding` – código de caracters (opcional) do recurso da URL

Corpo da tag: tags **<c:param>** (opcionais)

Exemplos de <c:import>

```
<c:import url="http://www.justjava.com.br/cabecalho"/>
<c:catch var="excecao">
 <c:import url="http://justjava/corpo.jsp"/>
</c:catch>
<c:if test="${not empty excecao}">
 Corpo da página não encontrado: 
<c:out value="${excecao}"/><br>
</c:if>
```

Biblioteca Format

- Formatação de datas
- Formatação de números
- Formatação de horas
- Formatos customizáveis

<fmt:formatNumber>

Descrição: formata um valor numérico de acordo com as preferências de localização do usuário ou de forma customizada como número, moeda ou porcentagem

Atributos:

- `value` – valor a ser formatado
- `type` – (opcional) number, currency ou percentage
- `pattern` – (opcional) padrão de formatação
- `currencyCode` - (opcional) código ISO da moeda
- `currencySymbol` - (opcional) símbolo da moeda
- `groupingUsed` - flag (opcional) indicando se o formato de saída conterá separadores decimais, de milhar, etc.

<fmt:formatNumber>

- `maxIntegerDigits`/`minIntegerDigits` - (opcional) dígitos máximos/mínimos da parte inteira da saída
- `maxFractionDigits`/`minFractionDigits` - (opcional) dígitos máximos/mínimos da parte fracionária da saía
- `var` - variável (opcional) que conterá a saída
- `scope` - (opcional) escopo da variável (`page, request, session, application`)

Corpo da tag: atributo `value`

Exemplos de <fmt:formatNumber>

```
<fmt:formatNumber value="9876543,21"
   type="currency"/>

<fmt:formatNumber value="12,3" pattern=".000"/>

<fmt:formatNumber value="123456,7891"
   pattern="#,#00.0#"/>

<fmt:formatNumber value="123456789" type="currency"
   var="cur"/>
```

Exemplo de <fmt:parseNumber>

```
<c:set var="reais" value="R$ 5,00" />

<fmt:parseNumber value="${reais}" type="currency"
   parseLocale="pt_BR" />
```

<fmt:formatDate>

Descrição: formata datas e/ou horas de acordo com as preferências de localização do usuário ou de forma customizada

Atributos:

- `value` – valor a ser formatado
- `type` - (opcional) date, time ou both
- `dateStyle` - (opcional) padrão de formatação da data
- `timeStyle` - (opcional) padrão de formatação da hora
- `pattern` - (opcional) padrão de formatação customizado
- `timeZone` - (opcional) TimeZone da data/hora a ser formatada

<fmt:formatDate>

- `var` - variável (opcional) que conterá saída
- `scope` - (opcional) escopo da variável (`page, request, session, application`)

Corpo da tag: não utilizado

Exemplos de <fmt:formatDate>

```
<jsp:useBean id="now" class="java.util.Date" />

<fmt:formatDate value="${now}" timeStyle="long"
   dateStyle="long"/>

<fmt:formatDate value="${now}" pattern="dd.MM.yy"/>
```

<fmt:parseDate>

Descrição: converte para datas e/ou horas de acordo com as preferências de localização do usuário ou de forma customizada

Atributos:

- `value` – valor a ser convertido
- `type` – (opcional) date, time ou both
- `dateStyle` - (opcional) padrão de formatação da data
- `timeStyle` - (opcional) padrão de formatação da hora
- `pattern` - (opcional) padrão de formatação customizado
- `timeZone` - (opcional) TimeZone da data/hora a ser convertida

<fmt:parseDate>

- `parseLocale` - (opcional) Locale a ser usado como base da conversão
- `var` - variável (opcional) que conterá saída
- `scope` - (opcional) escopo da variável (`page, request, session, application`)

Corpo da tag: atributo `value`

Exemplo de <fmt:parseDate>

```
<fmt:parseDate value="13:15" pattern="HH:mm" />
```

Biblioteca SQL

- Trabalhar com DataSources e/ou Connections
- Realizar consultas
- Acessar resultados de consultas facilmente
- Realizar inserções, atualizações e remoções de dados
- Realizar várias operações de forma transacional

<sql:update>

Descrição: executa uma instrução SQL do tipo INSERT, UPDATE ou DELETE

Atributos:

- `sql` - instrução a ser executada
- `dataSource` - (opcional) String de JNDI ou DataSource usado
- `var` - variável (opcional) que conterá o número de linhas afetadas
- `scope` - (opcional) escopo da variável (`page, request, session, application`)

Corpo da tag: atributo `sql`

Exemplo de <sql:update>

```
<sql:update>UPDATE usuario
SET total_acessos = total_acessos + 1
WHERE login = ?
<sql:param value="${login}"/>
</sql:update>
```

<sql:query>

Descrição: executa uma instrução SELECT

Atributos:

- `sql` - instrução a ser executada
- `dataSource` - (opcional) String de JNDI ou DataSource usado
- `maxRows` - (opcional) número máximo de linhas retornado pela consulta
- `startRow` - (opcional) número da primeira linha que será retornada para o usuário

<sql:query>

- `var` - variável (opcional) que conterá o número de linhas afetadas
- `scope` - (opcional) escopo da variável (`page, request, session, application`)

Corpo da tag: atributo `sql`

Exemplo de <sql:query>

```
<sql:query var="clientes" dataSource="${dataSource}">
SELECT * FROM clientes WHERE pais = 'China'
</sql:query>
<table>
<c:forEach var="cliente" items="${clientes.rows}">
<tr>
<td><c:out value="${cliente.nome}"/>, cuidado com a
   pneumonia asiática!!!!</td>
</tr>
</c:forEach>
</table>
```

<sql:param>

Descrição: seta o valor de um parâmetro “?” numa query

Atributos:

- `value` – valor do parâmetro

Corpo da tag: atributo `value`

Exemplo de <sql:param>

```
<sql:update>
UPDATE usuario
SET total_acessos = total_acessos + 1
WHERE login = ? <sql:param value="${login}"/>
</sql:update>
```

<sql:dateParam>

Descrição: seta o valor de um parâmetro “?” através de um Date, numa query

Atributos:

- `value` – valor do parâmetro
- `type` – (opcional) date, time ou timestamp

Corpo da tag: atributo `value`

Exemplo de <sql:dateParam>

```
<sql:update>
UPDATE usuario SET data_acesso = ? WHERE login = ?
<sql:dateParam value="${now}"/>
<sql:param value="${login}"/>
</sql:update>
```

Interface *Result*

- Representa o resultado de uma <sql:query>
- ***SortedMap[] getRows()*** - uma array de linhas de um SortedMap. A chave no SortedMap é o nome da coluna.
- ***Object[][] getRowsByIndex()*** - uma array de linhas por colunas do resultado.
- ***String[] getColumnNames()*** - nomes das colunas do resultado
- ***int getRowCount()*** - total de linhas retornadas
- ***boolean isLimitedByMaxRows()*** - indica se o atributo maxRows foi usado para limitar o número de linhas retornado

<sql:transaction>

Descrição: define um contexto transacional dentro do qual **<sql:update>** e **<sql:query>** serão utilizadas

Atributos:

- `dataSource` - (opcional) String de JNDI ou DataSource usado
- `isolationLevel` - (opcional) nível de isolamento da transação

Corpo da tag: instruções **<sql:update>** e **<sql:query>**

Exemplo de <sql:transaction>

```
<sql:transaction>
  <sql:update>UPDATE usuario
    SET data_acesso = ? WHERE login = ?
    <sql:dateParam value="${now}"/>
    <sql:param value="${login}"/>
  </sql:update>
  <sql:update>UPDATE estatisticas
    SET numero_visitas = numero_visitas + 1
  </sql:update>
</sql:transaction>
```

<sql:setDataSource>

Descrição: define um DataSource e o exporta como variável de escopo ou DataSource padrão

Atributos:

- `dataSource` - (opcional) String de JNDI ou DataSource exportado
- `driver/url/user/password` - (opcionais) valores usados pelo DriverManager para obter a conexão
- `var` - variável (opcional) que conterá o DataSource
- `scope` - (opcional) escopo da variável (`page, request, session, application`)

Corpo da tag: não utilizado

Exemplo de <sql:setDataSource>

```
<sql:setDataSource dataSource="jdbc/DB" />

<sql:setDataSource var=”ds”
   driver=”com.mysql.jdbc.Driver”
   url=”jdbc:mysql://localhost/mysql” user=”system”
   password=”manager”/>
```

Biblioteca XML

- Parsing de documentos XML
- Manipulação de elementos XML
- Fluxo condicional
- Iterações
- Transformações
- Uso de **EL** e **XPath**

XPath

- Especificação/recomendação **W3C** (1999)
- Usada para localizar e selecionar elementos dentro de um documento XML
- Espécie de *Expression Language* local para documentos XML
- Possui expressões, *binding* de variáveis, funções padrão, funções extensíveis e definição de *namespaces*

<x:parse>

Descrição: faz o *parsing* de um documento XML

Atributos:

- `xml` – documento a ser "parseado" (**String** ou **Reader**)
- `var` e `scope` – nome e escopo (opcional) da variável que receberá o documento parseado
- `varDom` e `scopeDom` – idem, porém variável será do tipo **org.w3c.dom.Document**
- `systemId` – URI identificando o documento
- `filter` – filtro (**org.xml.sax.XMLFilter**) a ser aplicado ao documento

Corpo da tag: documento XML a ser parseado

Exemplos de <x:parse>

```
<c:import
url="http://www.justjava.com.br/palestra.xml"
varReader=”xmlReader”>

<param name=”id” value=”666”/>

</url>

<x:parse xml=”${xmlReader}” var="documento"/>


<c:import
url="http://www.justjava.com.br/palestras.xml"
var=”xml” filter=”${filtroPalestra}”/>

<x:parse xml=”${xml}” var="documento"/>
```

<x:out>

Descrição: avalia uma expressão **XPath** e imprime o resultado na página

Atributos:

- `select` – expressão **XPath**
- `escapeXML` – flag (opcional) indicando se caracteres especiais devem ser substituídos

Corpo da tag: não utilizado

Exemplo de <x:out>

```
<c:import
url="http://www.justjava.com.br/palestra.xml"
var=”xml”/>

<x:parse xml=”${xml}” var="documento"/>

<x:out select=”$doc/titulo”/><br>

<x:out select=”$doc/autor”/><br>
```

<x:set>

Descrição: avalia uma expressão **XPath** e atribui o resultado à uma variável de escopo

Atributos:

- `select` – expressão **XPath**
- `escapeXML` – flag (opcional) indicando se caracteres especiais devem ser substituídos
- `var` – nome da variável que receberá o valor da expressão
- `scope` – escopo (opcional) da variável

Corpo da tag: não utilizado

Exemplos de <x:set>

```
<c:import
url="http://www.justjava.com.br/palestra.xml"
var=”xml”/>

<x:parse xml=”${xml}” var="documento"/>

<x:set select=”$doc/titulo” var=”titulo”
scope=”session”/><br>

<x:set select=”$doc/autor” var=”autor”
scope=”session”/><br>
```

<x:if>

Descrição: executa o corpo da tag caso uma expressão **XPath** seja avaliada em verdadeiro

Atributos:

- `select` – expressão **XPath** de teste
- `var` – nome da variável (opcional) que receberá o valor da expressão de teste
- `scope` – escopo (opcional) da variável

Corpo da tag: bloco a ser executado se a expressão teste for verdadeira

Exemplo de <x:if>

```
<c:import
url="http://www.justjava.com.br/palestra.xml"
var=”xml”/>

<x:parse xml=”${xml}” var="documento"/>

<x:if select=”$doc/[titulo='JSTL']”>

Introdução às bibliotecas JSTL

</x:if>
```

<x:choose>, <x:when> e <x:otherwise>

Descrição: usados para condições de múltiplas escolha

Atributos:

- `select` – expressão **Xpath** (**<c:when>**) de teste

Corpo das tags:

<x:choose> - tags **<x:when>** (1+) e **<x:otherwise>** (0-1)

<x:when> - bloco a ser executado quando condição teste for verdadeira

<x:otherwise> - bloco a ser executado caso nenhum **<x:when>** seja aplicado

Exemplo de <x:choose>...

```
<c:import
url="http://www.justjava.com.br/palestra.xml"
var=”xml”/>
<x:parse xml=”${xml}” var="documento"/>

<x:choose>
   <x:when select=”$doc/[titulo='JSTL']”>
Introdução às bibliotecas JSTL.
   </x:when>
   <x:otherwise>
Palestra de outros autores.
  </x:otherwise>
</x:choose>
```

<x:forEach>

Descrição: avalia uma expressão **XPath** (tipicamente retornando elementos XML) e iterage sobre o resultado dessa expressão

Atributos:

- `select` – expressão **XPath** a ser avaliada
- `var` – nome da variável (opcional e de escopo apenas no corpo da tag) que receberá o valor de cada iteração

Corpo da tag: bloco a ser executado em cada iteração

Exemplos de <x:forEach>

```
<c:import
url="http://www.justjava.com.br/palestra.xml"
var=”xml”/>
<x:parse xml=”${xml}” var="documento"/>

<x:forEach select=”$doc//autor”>
Autor: <x:out select=”@nome”/><br>
<x:set select=”@nome” var=”autor”/>
</x:forEach>

<x:forEach select=”$doc//autor” var=”autor”>
Elemento autor: <c:out value=”${autor}”/><br>
</x:forEach>
```

<x:param>

Descrição: sub-tag de **<x:transform>** que define um parâmetro da transformação

Atributos:

- `name` – nome do parâmetro
- `value` – valor do parâmetro

Corpo da tag: atributo `value`

<x:transform>

Descrição: transforma um documento XML de acordo com uma *stylesheet* XSLT

Atributos:

- `xml` – documento XML a ser transformado
- `xslt` – *stylesheet* XSLT
- `xmlSystemID` – URI identificando o documento XML
- `xsltSystemID` – URI identificando a *stylesheet* XSLT
- `var` – variável (opcional, do tipo **org.w3c.dom.Document**) contendo o documento transformado
- `scope` – escopo (opcional) da variável
- `result` – variável (opcional, do tipo **javax.xml.transform.Result**) que processará o resultado

Corpo da tag: documento XML e/ou parâmetros(**<x:param>**)

<x:transform>

Descrição: transforma um documento XML de acordo com uma *stylesheet* XSLT

Atributos:

`xml` – documento XML a ser transformado

`xslt` – *stylesheet* XSLT

`xmlSystemID` – URI identificando o documento XML

`xsltSystemID` – URI identificando a *stylesheet* XSLT

`var` – variável (opcional, do tipo **org.w3c.dom.Document**) contendo o documento transformado

`scope` – escopo (opcional) da variável

`result` – variável (opcional, do tipo **javax.xml.transform.Result**) que processará o resultado

Corpo da tag: documento XML e/ou parâmetros(**<x:param>**)

Exemplos de <x:transform>

```
<c:import
url="http://www.justjava.com.br/palestras.xml"
var=”xml”/>
<c:import
url="http://www.justjava.com.br/palestras.xslt"
var=”xslt”/>

<x:transform xml=”${xml}” xslt=${xslt}”/>

<x:transform xml=”${xml}” xslt=${xslt}”>
   <x:param name=”maiusculas” value=”true”/>
</x:transform>
```

Links Úteis

- Página oficial JSTL

  http://java.sun.com/products/jsp/jstl/

- JSR 52

  http://www.jcp.org/jsr/detail/52.jsp

- Especificação final JSTL 1.0

  http://jcp.org/aboutJava/communityprocess/final/jsr052/index.html

- Java Magazine (série de artigos sobre JSTL - jm7-9)

  http://www.javamagazine.com.br

- XPath

  http://www.w3.org/TR/xpath

Contato

- Felipe “felipeal” Leme
  - felipeal @ iname.com
  - felipe @ seedts.com
  - felipe.leme @ trust.com.br
- Michael “Mister M” Nascimento
  - michael @ michaelnascimento.com.br
  - mister__m @ hotmail.com

DEMO

Coming soon in a theater close to you...