Java Web

AULA 10 – JSTL

Objetivos e Conceitos

- Objetivos:
  - Apresentar JSTL. Converter código Scriptlet para JSTL/EL. Apresentar as funções, as tags de propósito geral, formatação e SQL.
- Conceitos:
  - JSTL.

Tópicos

- JSTL
- Funções
- Core: Finalidades Gerais
- Formatação
- SQL

JSTL

JSTL

JSTL: *JavaServer Pages Standard Template Library*

Biblioteca de tags para serem usadas nas páginas JSPs

Substituir *Scriptlets* na página, facilitando a escrita e manutenção das páginas

Exemplo:

- Receber o parâmetro “nome” passado de um formulário
- Armazenar em uma variável chamada “meuNome”

```
<c:set var="meuNome" value="${param.nome}" />
```


JSTL

Deve-se adicionar a biblioteca de JSTL no projeto, para poder usá-la

Depois, incluir o cabeçalho apropriado no topo dos JSPs

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

Indica que a biblioteca de *tags*:

`http://java.sun.com/jsp/jstl/core`

Será acessível pelo prefixo

`<c:....`

JSTL

JSTL 1.2 – componente do JavaEE 5, JSP 2.1

Composto pelos seguintes elementos

- EL
- Funções
- Core: propósito geral
  - Finalidades gerais
  - Manipulação de variáveis
  - Manipulação de Beans
  - Tratamento de erros
  - Condicionais
  - Laços
  - URL
- Format: formatação, parsing e recursos
- SQL: execução de *queries* no banco de dados
- XML
- Taglib Validators
- API

Funções

Funções

Conjunto de funções auxiliares do JSTL, que podem ser usados em expressões EL

Adicionar a *taglib*

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

Não esquecer de adicionar a biblioteca do JSTL no projeto

Pode-se usar:

```
<h2>${fn:length("Oi mundo!!!!")}</h2>
```


Funções

`int` **`length`**`(Object)`: Retorna o número de itens em uma coleção ou o tamanho da string.
`String` **`toLowerCase`**`(String)`: Converte a string pra caixa baixa.
`String` **`toUpperCase`**`(String)`: Converte a string pra caixa alta.
`String` **`trim`**`(String)`: Remove espaços antes e depois da string.
`String` **`escapeXml`**`(String)`: Escapa caracteres XML "<", ">" e "&", transforma em suas entidades.
`boolean` **`contains`**`(String, String)`: Testa se uma string contém uma substring, *case sensitive*.
`boolean` **`containsIgnoreCase`**`(String, String)`: Testa se uma string contém uma substring, desconsiderando caixa alta/baixa.
`boolean` **`endsWith`**`(String, String)`: Testa se uma string possui um sufixo (termina com determinada string).
`boolean` **`startsWith`**`(String, String)`: Testa se uma string possui um prefixo (começa com determinada string).
`int` **`indexOf`**`(String, String)`: Retorna o índice da primeira ocorrência de uma substring dentro da string.
`String` **`substring`**`(String, int, int)`: Retorna uma substring.
`String` **`substringAfter`**`(String, String)`: Retorna o resto da string após uma determinada substring.
`String` **`substringBefore`**`(String, String)`: Retorna o resto da string antes de uma determinada substring.
`String` **`join`**`(String[], String)`: Junta todos os elementos do array em uma string.
`String[]` **`split`**`(String, String)`: Separa uma string em um array de substrings conforme um delimitador.
`String` **`replace`**`(String, String, String)`: Retorna uma nova string substituindo uma substring por outra.

Funções

Exemplos:

```
<h2>${fn:length("Oi mundo!!!!")}</h2>
<h2>${fn:toUpperCase("Oi mundo!!!!")}</h2>
<h2>${fn:escapeXml("<<Oi mundo!!!!>>")}</h2>
<h2>${fn:split("Oi mundo!!!!", " ")[1]}</h2>
<h2>${fn:indexOf("Oi mundo!!!!", "!")}</h2>
```


Exercícios..

1. Criar um JSP e executar os exemplos anteriores
   - Não esquecer de adicionar a *taglib* de funções e a biblioteca no projeto

Core: Finalidades Gerais

Core

Deve-se incluir a biblioteca no JSP

Inclusão:

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

Exemplo de uso:

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

<c:out> é uma *tag* que mostra o que estiver em **value**, protegendo o sistema contra XSS

Deve ser sempre usada quando se mostra dados de beans.

Será vista mais adiante

Core.

Tipo da Tag	Tags
Variáveis	`<c:set>` `<c:remove>`

Controle de Fluxo	`<c:if>`
	`<c:choose>`
	`<c:when>`
	`<c:otherwise>`
	`<c:forEach>`
	`<c:forTokens>`

Miscelânea	`<c:out>` `<c:catch>`

URL	`<c:import>`
	`<c:param>`
	`<c:redirect>`
	`<c:param>`
	`<c:url>`
	`<c:param>`

Variáveis : <c:set>

Criar e atribuir valor a uma variável e manipular atributos de Beans

Sintaxe para variáveis:

```
<c:set var="nome" value="valor"
       [scope="{page|request|session|application}"] />
<c:set var="nome" [scope="{page|request|session|application}"] >
   valor
</c:set>
```

Sintaxe para Beans:

```
<c:set target="bean" property="propriedade" value="valor" />
<c:set target="bean" property="propriedade" >
   valor
</c:set>
```

Variáveis : <c:set>

Para variáveis:

- **var** : Nome da variável. Se ela não existe, é criada no escopo apontado (se o valor não for nulo)
- **value** : Indica o valor a ser atribuído. Opcional se o valor estiver no corpo da tag. Se for nulo, remove a variável do escopo.
- **scope** : Escopo da variável (**page** por default). Opcional.

Para Beans:

- **target** : EL que determina o objeto cuja propriedade será atribuída. Objeto já deve existir.
- **value** : Indica o valor a ser atribuído. Opcional se o valor estiver no corpo da tag.
- **property** : Nome da propriedade

Exemplo:

```
<c:set var="idade" value="10" />
<c:set target="${sessionScope.pessoa}" property="idade"
                                       value="10" />
```


Variáveis : <c:set>.

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
   <head><title>Teste</title></head>
   <body>
      <h1>Variável</h1>
      <c:set var="idade" value="10" />
      <h2>${idade}</h2>

      <h1>Bean</h1>
      <jsp:useBean id="aluno" class="beans.Aluno" />
      <c:set target="${aluno}" property="nome" value="Razer" />
      <h2><c:out value="${aluno.nome}" /></h2>
</body>
</html>
```

Variáveis : <c:remove>

Remove uma variável de um determinado escopo

Sintaxe:

```
<c:remove var="variavel"
          [scope="{page|request|session|application}"] />
```

Onde:

- **var**: nome da variável a ser removida
- **scope**: escopo da variável

Variáveis : <c:remove>

Se escopo não for definido, usa **findAttribute()** para encontrá-la

Usa a seguinte ordem de busca:

- **page**
- **request**
- **session**
- **application**

Variáveis : <c:remove>.

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
   <head><title>Teste</title></head>
   <body>
        <c:set var="nome" value="Razer" />
        <h2>Nome: ${nome}</h2>
        <c:remove var="nome" />
        <h2>Nome: ${nome}</h2>
</body>
</html>
```


Fluxo: <c:if>

Condicional simples

Sintaxe:

```
<c:if test="expressao" var="variável"
         [scope="{page|request|session|application}"] />

<c:if test="expressao" [var="variável"]
         [scope="{page|request|session|application}"] >
    Ações
</c:if>
```

Se a expressão dentro de `test` resultar verdadeiro, executa as **Ações** no corpo do **<c:if>**

Armazena o resultado da expressão na variável indicada no escopo indicado (default página)

Caso contrário, ignora as **Ações** no corpo do **<c:if>**

Não possui ELSE

Fluxo: <c:if>

Onde:

- **test** : expressão EL que resulta um valor booleana a ser testado
- **var** : variável que receberá o valor booleano do teste efetuado
- **scope** : escopo da variável **var** (default **page**)

Fluxo: <c:if>

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html><head><title>Teste</title></head>
<body>
    <c:set var="parametro" value="${param.valor}" />

    <c:if test="${parametro > 10}">
       <h2>Condição deu verdadeiro</h2>
    </c:if>

    <c:if test="${parametro > 10}" var="resultado" />

    <h2> Resultado: ${resultado}</h2>
</body>
</html>
```

Fluxo: <c:if>.

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

    <c:if test="${empty sessionScope.logado}" >
       <c:set var="mensagem" value="Precisa fazer o login"
                             scope="request" />
       <jsp:forward page="login.jsp" />
    </c:if>

    <h2> Minha página lindinha</h2>
</body>
</html>
```


Fluxo: <c:choose>

Condicional mutuamente exclusivo

Sintaxe:

```
<c:choose>
     <c:when test="${ <condicao1> }">
              Ações da condição 1
     </c:when>
     <c:when test="${ <condicao2> }">
              Ações da condição 2
     </c:when>
     ...
     <c:otherwise>
              Ações se nenhuma der verdadeiro
     </c:otherwise>
</c:choose>
```

Fluxo: <c:choose>

Onde:

- **test** : expressão booleana a ser testada para decidir se o **<c:when>** respectivo será ou não executado

Executa as ações dentro do primeiro **<c:when>** cujo **test** resultar **true**

Caso nenhum **<c:when>** execute, então será executada a cláusula **<c:otherwhise>** , se estiver presente

Fluxo: <c:choose>

Tag usada para fazer IF...ELSE

```
<c:choose>
    <c:when test="${ <condicao> }">
            Ações caso seja verdadeira
    </c:when>
    <c:otherwise>
            Ações caso seja falsa
    </c:otherwise>
</c:choose>
```

Fluxo: <c:choose>.

```
<c:set var="idade" value="75" />
<c:choose>
   <c:when test="${idade >= 60}">
      <h2>Adulto</h2>
   </c:when>
   <c:when test="${idade >= 18}">
      <h2>Jovem</h2>
   </c:when>
   <c:otherwise>
      <h2>Criança</h2>
   </c:otherwise>
</c:choose>
```


Fluxo: <c:forEach>

Laço ou varrendo uma coleção

Laço repetindo comandos uma determinado número de vezes

Sintaxe:

```
<c:forEach [var="variável"] [items="coleção"]
           [varStatus="variável status"]
           [begin="inicio"] [end="fim"] [step="passo"] >
     Ações
</c:forEach>
```

Fluxo: <c:forEach>

Onde:

- **`var`**: Nome da variável que receberá cada elemento da coleção ou valor de iteração, no escopo da página. Opcional.
- **`items`**: Coleção de itens a ser varrido. Opcional, pois pode iterar sobre números com **`begin`**, **`end`**, **`step`**.
- **`varStatus`**: Nome da variável que recebe o estado da iteração. Opcional.
- **`begin`**: Valor inicial da iteração, ou delimitador quando está varrendo uma coleção com **`items`**. Opcional.
- **`end`**: valor final da iteração, ou delimitador quando está varrendo uma coleção com **`items`**. Opcional.
- **`step`**: passo de iteração, ou delimitador quando está varrendo uma coleção com **`items`**. Opcional.

Coleções suportadas:

- Array (de objetos ou tipos primitivos)
- Implementações de *Collection*
- Implementações de *Iterator*
- Implementações de *Enumeration*
- Implementações de *Map*
- String separado por vírgulas

Fluxo: <c:forEach>

Atributo **`varStatus`**

- Contém o estado da iteração
- Instância de **`javax.servlet.jsp.jstl.core.LoopTagStatus`**

Contém propriedades:

- **`current`** : Item da iteração corrente
- **`index`** : Índice da iteração corrente (iniciado em zero)
- **`count`** : Índice da iteração corrente (iniciado em um)
- **`first`** : Flag indicando se é a primeira iteração
- **`last`** : Flag indicando se é a última iteração
- **`begin`** : Valor do atributo **`begin`**
- **`end`** : Valor do atributo **`end`**
- **`step`** : Valor do atributo **`step`**

Fluxo: <c:forEach>

Mostrar os números de 1 a 10 (laço definido)

```
<c:forEach var="i" begin="1" end="10">
        ${i}<br/>
</c:forEach>
```

Mostrar os números de 1 a 10 (laço definido), de 2 em 2

```
<c:forEach var="i" begin="1" end="10" step="2" >
        ${i}<br/>
</c:forEach>
```


Fluxo: <c:forEach>

Mostrar os dados de um *array*

```
<c:set var="compras"
        value="${fn:split('batata,cebola,carne,alho', ',')}" />
<ul>
<c:forEach var="str" items="${compras}">
    <li> ${str} </li>
</c:forEach>
</ul>
```

O *array* pode ser criado com a sintaxe de *array* no EL 3+

```
<c:set var="compras" value="${['batata','cebola','carne','alho']}" />
```

Fluxo: <c:forEach>

Se o objetivo for varrer só parte do *array*, pode-se delimitar com **begin**, **end** e **step**

O índice do vetor começa em 0 e os atributos são inclusivos (inclui os índices apontados)

```
<c:set var="idades" value="${[10,20,30,40,50,60]}" />

<ul>
<c:forEach var="ida" items="${idades}" begin="1" end="4" >
    <li> ${ida} </li>
</c:forEach>
</ul>
```


Fluxo: <c:forEach>

Mostrar substrings separadas por vírgula

```
<c:set var="meses" value="Janeiro,Fevereiro,Março" />

<c:forEach var="mes" items="${meses}">
    ${mes} <br />
</c:forEach>
```

Fluxo: <c:forEach>

Obter um array a partir de um Bean

```
package pacote;

public class Universidade implements Serializable {
     private String[] cursos;
     public Universidade() {
        cursos = new String[] {"TADS", "TNI", "TGQ"};
     }
     // setter/getter
}
```


Fluxo: <c:forEach>

O fragmento JSP seria

```
<jsp:useBean id="u" class="pacote.Universidade" />

<c:forEach var="nome" items="${u.cursos}">
    ${nome} <br />
</c:forEach>
```

Fluxo: <c:forEach>

Listar os dados de uma lista de Pessoas (propriedades: *nome, email, endereco, cidade*), passados via request (atributo no *request*: **"lista"**), em uma tabela:

```
<table>
<c:forEach var="pessoa" items="${requestScope.lista}">
   <tr>
      <td> <c:out value="${pessoa.nome}" /> </td>
      <td> <c:out value="${pessoa.email}" /> </td>
      <td> <c:out value="${pessoa.endereco}" /> </td>
      <td> <c:out value="${pessoa.cidade}" /> </td>
   </tr>
</c:forEach>
</table>
```


Fluxo: <c:forEach>.

Mostrar a lista anterior como uma tabela zebrada com **varStatus**:

```
<table>
<c:forEach var="pessoa" items="${requestScope.lista}"
           varStatus="status" >
   <tr style="background-color: ${status.index % 2 == 0 ? 'silver' :
'white' }" >
      <td> <c:out value="${pessoa.nome}" /> </td>
      <td> <c:out value="${pessoa.email}" /> </td>
      <td> <c:out value="${pessoa.endereco}" /> </td>
      <td> <c:out value="${pessoa.cidade}" /> </td>
   </tr>
</c:forEach>
</table>
```

Fluxo: <c:forTokens>

Mesmos atributos de **<c:forEach />**

Adiciona o atributo **delims** , para delimitar *strings*

Efetua as repetições sobre uma string, separando-a usando um separador específico

Sintaxe:

```
<c:forTokens items="coleção" delims="delimitadores"
             [var="variável"] [varStatus="variável status"]
             [begin="inicio"] [end="fim"]  [step="passo"] >
     Ações
</c:forTokens>
```


Fluxo: <c:forTokens>

Onde:

- **items**: Obrigatório. *String* separada por delimitadores a ser varrida.
- **delims**: Obrigatório. Delimitador de *strings.*
- **var**: Nome da variável que receberá cada *substring* a cada volta no laço. Opcional.
- **varStatus**: Nome da variável de estado da iteração. Opcional.
- **begin**: Valor inicial da iteração, para limitar o laço. Opcional.
- **end**: Valor final da iteração , para limitar o laço. Opcional.
- **step**: Passo de iteração , para limitar o laço. Opcional.

Fluxo: <c:forTokens>.

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head><title>Teste</title></head>
<body>
     <h2>forTokens com Strings</h2>

     <c:forTokens items="www.tads.ufpr.br" delims="."
                  var="x">
        ${x} <br />
     </c:forTokens>
     <br />
</body>
</html>
```


Miscelânea: <c:out>

Imprime uma *string* em um JSP

Sintaxe:

```
<c:out value="valor" [escapeXml="{true|false}"]
                     [default="valor"] />

<c:out value="valor" [escapeXml="{true|false}"] >
    Valor default
</c:out>
```

Onde:

- **value** : indica o valor a ser impresso
- **escapeXml** : se converterá caracteres especiais em entidades (default **true**)
- **default** : que valor será impresso caso **value** seja avaliado como **null**

Miscelânea: <c:out>

```
<%@taglib prefix="c"
          uri="http://java.sun.com/jsp/jstl/core" %>
<html>
  <head><title>Teste</title></head>
  <body>
     <h2><c:out value="${param.nome}" /></h2>
     <h2><c:out value="${param.idade}" /></h2>
  </body>
</html>
```


Miscelânea: <c:out>

Se **escapeXml** estiver setado, faz as seguintes conversões:

- < → <
- > → >
- & → &
- ' → '
- " → "

Evita ataques XSS

Miscelânea: <c:out>

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html><head><title>JSP Page</title></head><body>
        <c:set var="comprometido"
                value="<script>alert('Oi mundo');</script>" />
        ${comprometido}
        <br/>
        <c:out value="${comprometido}" />
        <c:out value="${comprometido}" escapeXml="false" />
</body></html>
```


Miscelânea: <c:out>.

```
<script>alert('Oi mundo');</script> <br/>
&lt;script&gt;alert(&#039;Oi mundo&#039;);&lt;/script&gt;
<script>alert('Oi mundo');</script>
```

Miscelânea: <c:catch>

Tratamento de Exceções

Sintaxe:

```
<c:catch [var="variável"] >
   Ações
</c:catch>
```

Onde:

- **var** : Opcional. Variável que recebe a exceção gerada (**java.lang.Throwable**). Esta variável sempre terá escopo da página. Se não for definida, a exceção será tratada mas não será salva (como se fosse ignorada) e não é recomendado.
- **Ações** : Comandos que podem gerar um erro (**try**)

Miscelânea: <c:catch>

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head><title>Teste</title></head>
<body>
   <c:catch var="excecao">
     <%
        out.println(5/0);
     %>
   </c:catch>
   ${excecao.message}
</body>
</html>
```

Miscelânea: <c:catch>

Pode-se verificar se foi gerado um erro com a tag **<c:if>**

```
<c:catch var="e" >
   Ações
</c:catch>
<c:if test="${not empty e}">
   Código se erro
</c:if>
```


Fluxo: <c:catch>.

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html><head><title>Teste</title></head>
<body>
   <c:catch var="excecao">
     <%
         out.println(5/0);
     %>
   </c:catch>

   <c:if test="${not empty excecao}">
     Exceção: <c:out value="${excecao.message}" />
   </c:if>
</body>
</html>
```

URL: <c:import>

Faz o include de um recurso do mesmo contexto ou de fora da aplicação (outro contexto)

O mesmo que **<jsp:include />**, mas permite inclusão de recursos fora da aplicação

Importa o conteúdo de três formas diferentes:

- Se não for definida uma variável (nem **var** nem **varReader**), o conteúdo é mostrado no JSP chamador
- Se for definida **var**, o conteúdo é colocado dentro de uma variável com este nome, no escopo escolhido (**scope**)
- Se for definida **varReader**, o conteúdo é disponibilizado como um **java.io.Reader**, cuja variável tem o nome definido no atributo

Sintaxe:

```
<c:import url="url_do_recurso"
          [var="variavel"]
          [scope="{page|request|session|application}"]
          [context="/OutraAplicacao"]
          [charEncoding="ISO-8859-1"]
          [varReader="variavel"] />
```


URL: <c:import>

Onde:

- **url** : URL do recurso a ser incluído
- **var** : variável para onde a saída do recurso será inserida
- **scope** : escopo da variável **var**
- **context**: Usado para importar um recurso de fora do contexto (aplicação) atual. String começando com "/" indicando outro contexto, outra aplicação, de onde deve-se importar o conteúdo
- **charEncoding**: codificação de caracteres usada para disponibilizar o conteúdo via **varReader** (que é do tipo **java.io.Reader**)
- **varReader**: nome de uma variável, que será do tipo **java.io.Reader**, usada para disponibilizar o conteúdo importado.

URL: <c:import>

```
<%@taglib prefix="c"
          uri="http://java.sun.com/jsp/jstl/core" %>
<html>
     <head><title>Teste</title></head>
     <body>
            <h2>Teste IMPORT</h2>

            <c:import url="cabecalho.jsp"/>
     </body>
</html>
```


URL: <c:import>

Exemplos:

```
<c:import url="target_page.html" />
```

Exemplo com variável:

```
<c:import url="http://www.tads.ufpr.br/ementas/ementas/ficha2/648"
          var="saida" />

<p>
   <c:out value="${saida}" />
</p>
```

URL: <c:import>

A URL de importação pode ser Absoluta ou Relativa

- URL **Absoluta**

```
<c:import url="http://www.tads.ufpr.br/ementas/ementas/ficha2/648"
          var="tads" />
```

- URL **Relativa ao Próprio Contexto** (mesma aplicação) – Inicia com "/"

```
<c:import url="/contatos.html" var="contatos" />
```

- URL **Relativa ao Próprio Contexto** (mesma aplicação) e **Relativa à Página** (mesma pasta do JSP chamador) – Não inicia com "/"

```
<c:import url="listagem.html" />
```

- URL **Relativa em Outro Contexto** (outra aplicação) – Atributo **`context`** inicia com "/" e URL inicia com "/"

```
<c:import url="/clientes.html" context="/sistema/" />
```


URL: <c:import>

Pode-se passar parâmetros para a URL:

```
<c:param name="nome" value="valor" />
```

Para passar parâmetros na URL (*request*) para o recurso importado

- **`name`** : nome do parâmetro a ser passado para a URL
- **`value`** : valor do parâmetro a ser passado para a URL

No recurso importado obtém-se o valor com: **`${param.nome}`**

Sintaxe:

```
<c:import url="target_page.html" >
   <c:param name="nome" value="Razer" />
   <c:param name="email" value="${param.email}" />
   <c:param name="descricao">
       ${sessionScope.descricao}
   </c:param>
</c:import>
```

URL: <c:import>.

Exemplo

```
<c:import url="target_page.html">
   <c:param name="nome" value="Razer" />
   <c:param name="idade" value="22" />
</c:import>
```

Fará a importação do seguinte recurso:

```
target_page.html?nome=Razer&idade=22
```


URL: <c:url>

Cria uma URL, usando a codificação de caracteres apropriada e reescrita em URLs relativas

Exemplo:

```
<c:url value="teste.jsp" />
```

URL: <c:url>

Sintaxe:

```
<c:url value="valor" [context="contexto"]
      [var="variavel"]
      [scope="{page|request|session|application}"] />

<c:url value="valor" [context="contexto"]
       [var="variavel"]
       [scope="{page|request|session|application}"]>
   <c:param name="nome" value="valor" />
   <c:param name="nome">
       valor
   </c:param>
</c:url>
```


URL: <c:url>

Onde:

- **value** : Recurso a ser usado na URL a ser gerada. Obrigatório.
- **context** : Usado para gerar uma URL para fora do contexto (aplicação) atual. Nome do contexto da URL: / mais o nome. Se usado, tanto a URL (**value**) como o contexto devem iniciar com / . Opcional.
- **var** : Variável que receberá a URL montada. Opcional.
- **scope** : Escopo da variável em **var.** Default é **page.** Opcional.

<c:param />: parâmetros para a URL gerada

- **name** : nome do parâmetro a ser passado para URL
- **value** : valor do parâmetro a ser passado para a URL

No recurso para onde a URL for gerada, obtém-se o valor com:
${param.nome}

URL: <c:url>

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
   <head><title>Teste</title></head>
   <body>
      <h2>Teste URL</h2>

      <c:url value="mostrar.jsp" var="url" >
         <c:param name="nome" value="Razer" />
         <c:param name="local" value="UFPR" />
      </c:url>
      URL gerada: ${url} <br/>
      <a href="${url}">Mostrar</a>
   </body>
</html>
```

Gera a URL:

mostrar.jsp?nome=Razer&local=UFPR

URL: <c:url>

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
   <head><title>mostrar.jsp</title></head>
   <body>
      <h2>Mostrando dados da URL</h2>

      Nome:  ${param.nome}  <br />
      Local: ${param.local} <br />
   </body>
</html>
```

URL: <c:url>

Motivos para usar a tag **<c:url>**

Tratamento da passagem do ID de sessões via parâmetro na URL

- ID das sessões são, por default, passadas via *cookies*
- Quando *cookies* não estão disponíveis, são passados como parâmetro na URL
- **<c:url>** já trata o caso quando cookies não estejam disponíveis e passa o parâmetro na URL

Escrever uma URL em HTML iniciando com "/" faz com que seja relativa à raiz do servidor (não da aplicação)

- Escrever uma URL usando **<c:url>** faz ela relativa à aplicação

URL: <c:url>.

Por exemplo:

```
<a href="/index.html"> Teste </a>
```

- Aponta para: **http://www.servidor.com.br:8080/index.html**

Para apontar para sua aplicação deve-se fazer:

```
<a href="${pageContext.requet.contextPath}/index.html"> Teste </a>
```

Mas a tag **<c:url>** já faz isso automaticamente, basta iniciar o recurso com "/"

Exemplo:

```
<c:url value="/index.html" />
```

URL: <c:redirect>

Envia ao cliente uma resposta para efetuar o redirecionamento para outra página

Executa um **`response.sendRedirect()`**

Sintaxe:

```
<c:redirect url="url" [context="contexto"] />

<c:redirect url="url" [context="contexto"]>
   <c:param name="nome" value="valor"/>
   <c:param name="nome">
           valor
   </c:param>
</c:redirect>
```


URL: <c:redirect>

Onde:

- **`url`** : URL a ser processada
- **`context`** : Usado para redirecionar para fora do contexto (aplicação) atual. Nome do contexto da URL: / mais o nome. Se usado, tanto a URL (**value**) como o contexto devem iniciar com /

`<c:param>` : para passar parâmetros para o redirecionamento, na URL

- **`name`** : nome do parâmetro a ser passado para url
- **`value`** : valor do parâmetro a ser passado para a url

URL: <c:redirect>.

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
   <head><title>Teste</title></head>
   <body>
      <h2>Teste Redirect</h2>

      <c:redirect url="mostrar.jsp">
         <c:param name="nome" value="Razer" />
         <c:param name="local" value="UFPR" />
      </c:redirect>
      </body>
</html>
```


Diferenças entre Redirecionamentos

Forward

```
<jsp:forward />
```

É feito internamente pela Servlet

O navegador não toma consciência do que está sendo feito, portanto a URL continua a mesma

Se for feito um REFRESH no navegador, ele repete a requisição original

Mantém os atributos e parâmetros da requisição original

Diferenças entre Redirecionamentos

Redirect

```
<c:redirect />
```

A aplicação instrui o navegador a fazer uma nova requisição (segunda URL)

O navegador carrega a segunda URL, sem repetir a requisição original

Um pouco mais lento que o Forward, por necessitar de duas requisições do navegador

Objetos colocados na requisição original não estarão disponíveis na segunda requisição

Em suma, uma nova requisição que o navegador faz ao servidor

Diferenças entre Redirecionamentos

Quando usar:

- **Forward**:
  - Geralmente usado quando a requisição original pode ser repetida sem gerar inconsistência (ex, consulta no BD, atualização de registros, remoção de registros)
  - Quando é necessário passar dados estruturados como parâmetro: beans, lista de beans, etc
- **Redirect**:
  - Quando a requisição original não deve ser repetida pois pode gerar inconsistência (ex, duplicação de dados em uma inserção no BD)
  - Via *redirect* só é possível passar parâmetros via URL (strings)

Em formulários pode-se usar a regra:

- GET – corresponde a um “SELECT-e-Forward”
- POST – corresponde a um “Edição-e-Redirect”

Diferenças entre Redirecionamentos.

`<jsp:forward />`	`<c:redirect />`
Traduzido para um **`forward`** de um **`RequestDispatcher`**	Equivale a um **`response.sendRedirect()`**
Feito internamente	Retorna para o navegador efetuar o redirecionamento
Navegador não toma ciência do redireionamento	Navegador é quem faz o redirecionamento
Não altera URL	Altera a URL
Mantém a mesma requisição	São feitas duas requisições
Refresh no navegador repete a requisição original	*Refresh* no navegador só repete a última requisição
Parâmetros via escopo da requisição Parâmetros via URL (*parameter*)	Somente parâmetros via URL (*parameter*)
Operações idempotentes Operações que, quando repetidas, retornam o mesmo resultado	Operações não-idempotentes Operações que, quando repetidas, podem retornar resultados diferentes
Consulta no BD, Remoção e Alteração de registros	Inserção

Diferenças entre Inclusões..

`<%@include file="" %>`	`<jsp:include />`	`<c:import />`
Diretiva	Tag de ação do JSP	Tag do JSTL
Inclusão estática	Inclusão dinâmica, em tempo de execução	Inclusão dinâmica, em tempo de execução
Inclui o arquivo apontado na posição da diretiva, em tempo de compilação	Efetua uma requisição ao recurso **`include`** do **`RequestDispatcher`**	Efetua uma requisição ao recurso **`include`** do **`RequestDispatcher`**
Faz *merge* dos arquivos	Adiciona o resultado da requisição no chamador Só inclui recursos do próprio Contexto (aplicação)	Adiciona o resultado da requisição no chamador Pode incluir recursos de fora do Contexto (aplicação)
		Outras funcionalidades: Retornar o recurso em uma variável (var) ou em um Reader (varReader)

Formatação

Formatação

Deve-se incluir a biblioteca no JSP

Usa-se para Internacionalização
- `<fmt:setTimeZone>`

Inclusão:

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

Uso:

```
<fmt:formatNumber value="10.54" type="currency" />
```

Formatação.

Formatação

- Converte do tipo básico/primitivo para uma String
- Usado para mostrar os dados em tela

```
10.54 => "10.54"
```

Parsing

- Converte de String para o tipo básico/primitivo
- Usado para receber dados de formulários ou parâmetros

```
"10.54" => 10.54
```


<fmt:formatNumber />

Formata um número conforme linguagem selecionada no navegador

Dado um número, formata para String

Sintaxe:

```
<fmt:formatNumber value="valor" [type="tipo"]
      [pattern="padrão"] [maxIntegerDigits="nr"]
      [minIntegerDigits="nr"] [maxFractionDigits="nr"]
      [minFractionDigits="nr"] [var="variável"]
      [scope="{page|request|session|application}"] />
```

<fmt:formatNumber />

Onde:

- **value**: valor numérico a ser formatado
- **type**: (opcional) tipo para o qual deve ser formatado
  - **number** : número
  - **currency** : moeda
  - **percent** : porcentagem
- **var**: (opcional) nome da variável que armazena o número formatado
- **scope**: (opcional) escopo da variável **var**
- **pattern**: (opcional) especifica um formato para formatação
- **maxIntegerDigits**: (opcional) número máximo de dígitos inteiros para mostrar. Se o número excede `maxIntegerDigits`, ele é truncado
- **minIntegerDigits**: (opcional) número mínimo de dígitos inteiros para mostrar
- **maxFractionDigits**: (opcional) valor a ser formatado. Se o número excede `maxFractionDigits`, ele é arredondado
- **minFractionDigits**: (opcional) valor a ser formatado

<fmt:formatNumber />

Atributo **pattern** usa os seguintes símbolos

- 0 Representa um dígito
- # Representa um dígito; mostra 0 se não existe
- . Separador decimal
- , Separador de grupo
- E Representa forma exponencial em notação científica
- - Sinal negativo
- % Multiplica por 100 e mostra como porcentagem
- entre outros

Exemplo

Exemplo

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<!DOCTYPE html>
<html><head><title>JSP Page</title></head><body>
    <c:set var="numero" value="1234567.9876" /><br/>
    <fmt:formatNumber value="${numero}" type="number" /><br/>
    <fmt:formatNumber value="${numero}" type="currency" /><br/>
    <fmt:formatNumber value="${numero}" type="percent" /><br/>
    <fmt:formatNumber value="${numero}" type="number"
                      maxFractionDigits="2" /><br/>
    <fmt:formatNumber value="${numero}" type="number"
                      minFractionDigits="5" /><br/>
    <fmt:formatNumber value="${numero}" type="number"
                      pattern="###.###E0" /><br/>
</body></html>
```


Exemplo.

Resulta em:

1.234.567,988
R$ 1.234.567,99
123.456.799%
1.234.568,99
1.234.567,98760
1,23457E6

<fmt:formatDate />

Formata uma data conforme linguagem selecionada no navegador

Dado um objeto **Date**, formata para String

Sintaxe:

```
<fmt:formatDate value="data"
                [type="{date|time|both}"]
                [timeStyle="estiloHora"]
                [dateStyle="estiloData"]
                [pattern="padrao"]
                [var="variável"]
                [scope="{page|request|session|application}"] />
```


<fmt:formatDate />

Onde:

- **value** : Data a ser formatada
- **timeStyle** : Estilo pré-definido para horas, usa **java.text.DateFormat** (**full**, **long**, **medium**, **short**)
- **dateStyle** : Estilo pré-definido para datas, usa **java.text.DateFormat** (**full**, **long**, **medium**, **short**)
- **pattern** : Estilo customizado
- **var** : Variável para armazenar a data formatada
- **scope** : Escopo da variável que armazena a data formatada

<fmt:formatDate />.

Atributo pattern usa:

G	Designador de era	AD
y	Ano	2002
M	Mês	April & 04
d	Dia do mês	20
h	Hora (1-12)	12
H	Hora (0-23)	0
m	Minuto	45
s	Segundo	52
S	Milisegundo	970
E	Dia da semana	Tuesday
D	Dia do ano	180
F	Dia da semana no mês	2 (2a Quarta no mês)
w	Semana no ano	27
W	Semana no mês	2
a	Indicador a.m./p.m.	PM
k	Hora no dia (1-24)	24
K	Hora (0-11)	0

- Entre outros

Exemplo

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

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

   Data: <fmt:formatDate value="${data}" dateStyle="long" /> <br />
   Data: <fmt:formatDate value="${data}" dateStyle="short" /> <br />
   Data: <fmt:formatDate value="${data}" pattern="YY/M/d" /> <br />
   Data: <fmt:formatDate value="${data}" pattern="EEE" /> <br />
   Data: <fmt:formatDate value="${data}" pattern="dd/MM/yyyy" /> <br />
</body></html>
```

Exemplo.

Resultado

Data: 8 de Agosto de 2020
Data: 08/08/20
Data: 20/8/8
Data: Sáb
Data: 08/08/2020

<fmt:parseNumber />.

Faz o parse de uma String para Número

```
<fmt:parseNumber value="numero"
                 var="variável"
                 scope="{page|request|session|application}"
                 [type="{number|currency|percentage}"]
                 [integerOnly="{true|false}"]
                 [pattern="padrao"] />
```

<fmt:parseDate />..

Faz o parse de uma String para Data

```
<fmt:parseDate value="valor a ser convertido"
               var="variável"
               scope="{page|request|session|application}"
              [type="{date|time|both}"]
              [timeStyle="{full|long|medium|short|default}"]
              [dateStyle="{full|long|medium|short|default}"]
              [pattern="padrao"] />
```


SQL

SQL.

Deve-se incluir a biblioteca no JSP

Inclusão:

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

Tags disponíveis:

- **<sql:setDataSource>** : Conecta com o banco de dados
- **<sql:transaction>** : Roda os comandos em uma transação
- **<sql:query>** : Efetua uma consulta ao banco (SELECT)
- **<sql:update>** : Efetua uma atualização no banco (INSERT, UPDATE, DELETE)
- **<sql:param>** : Para setar parâmetros em queries (queries com ?)
- **<sql:dateParam>** : Para setar parâmetros do tipo data em queries com ?

<sql:setDataSource />

Cria uma fonte de dados para as *queries*

Cria uma conexão

Sintaxe:

```
<sql:setDataSource
            var="conexao"
            driver="org.apache.derby.jdbc.EmbeddedDriver"
            url="jdbc:derby:C:/apache/webapps/app11/WEB-
INF/lib/bd_teste"
            user="root" password="root" />
```

<sql:setDataSource />.

Nota sobre o Derby

Driver: **derby.jar** no diretório **lib**

Driver:

```
org.apache.derby.jdbc.EmbeddedDriver
```

URL do banco: absoluta, indicando o diretório

```
jdbc:derby:C:/apache/webapps/app11/WEB-INF/lib/bd_teste
```

Onde

- Protocolo: `jdbc:derby`
- Diretório do Banco: `C:/apache/webapps/app11/WEB-INF/lib/`
- Nome do Banco: `bd_teste`

<sql:query />.

Dado uma conexão, executa uma query

Sintaxe:

```
<sql:query dataSource="${conexao}" var="consulta">
      SELECT nm_pessoa, end_pessoa FROM tb_pessoa
</sql:query>
```

Varrer Resultados.

Para mostrar os dados de uma consulta, pode-se usar:

```
<sql:query dataSource="${conexao}" var="consulta">
     SELECT nm_pessoa, end_pessoa FROM tb_pessoa
</sql:query>

<c:forEach var="linha" items="${consulta.rows}">
     Nome:     <c:out value="${linha.nm_pessoa}" />  
     Endereco: <c:out value="${linha.end_pessoa}" /><br />
</c:forEach>
```


<sql:update />.

Dado uma conexão, executa uma query de atualização do banco

Sintaxe:

```
<sql:update dataSource="${conexao}" var="alteracao">
       insert into tb_pessoa values (?, ?)
        <sql:param value="${param.nome}" />
        <sql:param value="${param.endereco}" />
</sql:update>
```

A tag **<sql:param>** é usada para passar parâmetros:

- Os parâmetros são casados em ordem de aparecimento
- O primeiro "?" recebe o valor do primeiro **<sql:param>**
- O segundo "?" recebe o valor do segundo **<sql:param>**
- Etc.

Exemplo Completo

```
<%@page import="java.util.*,java.sql.*" %>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>
<html>
   <head>
      <title>Exemplo SQL</title>
   </head>
   <body>
       <sql:setDataSource var="con"
                    driver="com.mysql.jdbc.Driver"
                    url="jdbc:mysql://localhost:3306/banco"
                    user="root"  password="pass123"/>
```


Exemplo Completo

```
       <sql:query dataSource="${con}" var="result">
           SELECT id_cliente, nm_cliente, email_cliente FROM tb_clientes
       </sql:query>
```

Exemplo Completo.

```
    <table border="1" width="100%">
    <tr>
       <th>ID</th>
       <th>Nome</th>
       <th>E-mail</th>
    </tr>
    <c:forEach var="linha" items="${result.rows}">
        <tr>
           <td><c:out value="${linha.id_cliente}" /></td>
           <td><c:out value="${linha.nm_cliente}" /></td>
           <td><c:out value="${linha.email_cliente}" /></td>
        </tr>
    </c:forEach>
    </table>
</body>
</html>
```


Exercícios..

1. Criar um banco de dados PostgreSQL ou MySQL com a tabela **`tb_clientes`** apresentada anteriormente. Executar o JSP apresentado.