

# Desenvolvimento Web I

## Aula 07 - Introdução ao JSTL

# Apresentação

---

Olá, caro aluno, você está gostando do assunto?! Espero que sim! Lembre-se: qualquer dúvida, não hesite em reler as aulas anteriores e utilizar outros recursos, como os meios de comunicação com os tutores. Até agora você já aprendeu a usar scriptlets e diretivas dentro dos arquivos JSP. Esses recursos são muito úteis, mas podem “poluir” o código fonte dos arquivos JSP.

Nesta aula, veremos como o uso de tag libraries padrão pode nos ajudar a escrever um código mais limpo e de fácil entendimento, além de permitir que um mesmo código seja reusado por diversas páginas, mantendo um ponto único de manutenção e extensão.

## Objetivos

- Entender uma nova abordagem de construção de páginas sem o uso de scriptlets e diretivas JSP.
- Produzir páginas JSP mais fáceis de ler e manter usando JSTL (JSP Standard Tag Library).
- Descrever os recursos da taglib core para gerenciamento de URLs.
- Aplicar os recursos da taglib core para controle de fluxo.

## O que é uma *tag*

---

Uma tag é uma etiqueta, um marcador, como você já viu ao estudar XML e HTML. Uma tag JSP é uma tag específica para o ambiente JSP. Um desenvolvedor pode, por exemplo, usar código Java para criar um pequeno componente reutilizável para exibir o nome do usuário que está “logado” no sistema. Para implementar esse componente, o desenvolvedor Java pode criar uma tag que identifica o nome do usuário na sessão da aplicação e o exibe. Essa tag pode ser utilizada por diversos arquivos JSPs, sem precisar ser alterada. E se, além do nome do usuário “logado”, nós quisermos exibir alguma outra informação, não será necessário alterar código em todos os JSPs para atingir esse objetivo, basta alterar o código da tag para que essa nova informação, automaticamente, esteja presente em todas as páginas JSPs que a utilizam. Reutilizar código é possível de várias maneiras diferentes, mas o benefício de utilizar tags é que a sintaxe de uso é semelhante ao uso de tags HTML. Uma vez que a sintaxe é natural para os desenvolvedores HTML, as tarefas podem ser divididas de uma forma que permite aos desenvolvedores concentrar seus objetivos em regras de negócio e deixar a apresentação para os designers.

## O que é uma *tag library*

Nesta aula, veremos como o uso de tag libraries padrão pode nos ajudar a escrever um código mais limpo e de fácil entendimento. Além disso, o uso de tag libraries permite que um mesmo código seja reusado por diversas páginas, mantendo um ponto único de manutenção e extensão. Nesta aula e nas próximas, veremos as principais tag libraries padrão do JSP, chamadas de JSTL (JSP Standard Tag Library). Estas bibliotecas JSTL são: core, XML, internacionalização, SQL e funções. Nesta disciplina, focaremos nas bibliotecas core, internacionalização e funções, bem como teremos uma aula de EL (Expression Language), que nos ajudará a entender como obter os objetos dentro de uma páginas JSP em diferentes escopos (requisição, sessão, aplicação etc.), de forma elegante e padronizada.

Uma *tag library*, ou biblioteca de tags, é um conjunto de tags úteis empacotadas e distribuídas em conjunto, as quais ajudam os desenvolvedores de aplicativos web a realizar tarefas comuns a vários sistemas. O *JSP Standard Tag Library*, ou JSTL, é

uma tag library que muitos desenvolvedores utilizam para implementar a funcionalidade sem recorrer a scriptlets dentro do arquivo JSP.

## Instalando o JSTL

Nas versões mais recentes do Java EE, por exemplo a que recomendamos nessa disciplina) é necessário que você instale o JSTL para fazer uso de suas funcionalidades. Não se preocupe pois o procedimento é muito simples: Baixe o arquivo **jstl-1.2.jar** que está disponível em <http://central.maven.org/maven2/javax/servlet/jstl/1.2/jstl-1.2.jar> o copie para a pasta WebContent/WEB-INF/lib do seu projeto que deseja utilizar essa tecnologia. Pronto, é só isso. Em algumas versões do JSTL não é incluída a biblioteca padrão de componentes (standard) e se faz necessário você baixar também o JAR standard-1.1.2.jar que pode ser encontrado no <http://central.maven.org>.

JSTL inclui tarefas comuns, como a execução condicional, loops e internacionalização, assim como a exibição de valores de variáveis armazenadas em diferentes escopos da aplicação. Embora essas tarefas possam ser feitas usando scriptlets (ver Listagem 1), escrever o arquivo com tags JSTL é considerado mais fácil de se manter (ver Listagem 2).

```
1 <html>
2   <body>
3     <table>
4       <% String[] filmes = (String[]) request.getAttribute("filmes");
5       String filme=null;
6       for (int i = 0; i < filmes.length; i++) {
7         filme = filmes[i];
8         %>
9         <tr><td><%= filme %></td></tr>
10        <% } %>
11      </table>
12    </body>
13  </html>
```

**Listagem 1** - Código JSP que usa *scriptlets*

```

1 <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
2 <html>
3   <body>
4     <strong> Lista de filmes:</strong>
5     <br><br>
6     <table>
7       <c:forEach var="filme" items="${filmes}"><tr><td>${filme}</td></tr></c:forEach>
8     </table>
9   </body>
10 </html>

```

**Listagem 2** - Código JSP equivalente ao da Listagem 1, agora usando JSTL.

## A biblioteca JSTL

JSTL encapsula funcionalidades comuns a muitas aplicações JSP. Ao invés de misturar tags de vários fornecedores em suas aplicações JSP, JSTL nos apresenta um conjunto padrão de tags. Essa padronização permite que você use seus aplicativos em qualquer contêiner JSP que suporte JSTL. As funcionalidades da biblioteca padrão JSTL são:

Área	Subfunções	Prefixo
<b>core</b>	Suporte a variáveis	<b>c</b>
	Fluxo de controle	
	Gerenciamento de URLs	
	Miscelâneas	
<b>XML</b>	Core	<b>x</b>
	Fluxo de controle	
	Transformação	

Área	Subfunções	Prefixo
I18N	Localidade	fmt
	Formatação de mensagens	
	Formatação de números e datas	
Database	SQL	sql
Funções	Tamanho de coleções	fn
	Manipulação de strings	

**Quadro 1** - Funcionalidades da biblioteca padrão JSTL

Para este curso, abordaremos apenas as áreas core, I18N e funções. Deixando a teoria um pouco de lado, vamos agora mostrar quais são essas bibliotecas padrão do JSTL e quais funcionalidades elas fornecem.

## A biblioteca core

---

A biblioteca core implementa funcionalidades de suporte a variáveis, controle de fluxo, gerenciamento de URLs, além de outras (miscelâneas). Você verá agora com mais detalhes cada uma dessas funcionalidades.

Área	Função	Tags	Prefixo
Core	Suporte a variáveis	remove set	c
	Controle de fluxo	choose when otherwise forEach if	
	Gerenciamento de URLs	import param redirect param url param	
	Miscelâneas	catch out	

**Quadro 2** - Funcionalidades detalhadas da biblioteca core

Começaremos mostrando as duas funcionalidades gerais da biblioteca core, que são `<c:out>` e `<c:catch>`. Cada uma delas possui atributos que descrevem como será o seu comportamento.

A tag `c:out` exibe os valores armazenados nas variáveis de escopo na página. O escopo pode ser a própria página, o request, a sessão e a aplicação. Assim, o código `<c:out value="${var}" />` irá procurar por um objeto chamado `var` no escopo da página. Caso não encontre, a procura será feita no request, e, caso não encontre, será a vez da sessão até chegar ao escopo da aplicação. Caso a variável não esteja em nenhum desses escopos, nada será exibido. Um detalhe sobre essa tag é que se o valor da variável for null, uma string vazia será exibida. Dessa forma, não precisamos nos preocupar com exceções do tipo *NullPointerException*.

## A biblioteca core II

Uma vez que explicamos a funcionalidade dessa tag, vamos agora mostrar e explicar cada um dos seus atributos.

Atributo	Descrição	Obrigatório	Valor default
value	Valor a ser avaliado e exibido	Sim	
default	Valor que será exibido, caso o value seja vazio ou nulo	Não	Body
escapeXml	Escapa alguns caracteres que são marcadores HTML	Não	true

**Quadro 3** - Atributos da tag c:out

O atributo **value** é uma expressão que é avaliada no servidor na hora que a página JSP está sendo processada (veremos esse atributo com mais detalhes em uma próxima aula sobre expressões e quais variáveis podem ser usadas nessas expressões). Como vimos anteriormente, há uma ordem de avaliação nessa expressão com relação a onde essas variáveis serão procuradas primeiramente (página,request, sessão e aplicação). Além disso, caso o resultado dessa avaliação seja o valor null, uma string vazia será retornada.

Uma vez que podemos ter como resultado da avaliação do atributo value uma string vazia, essa tag define um atributo de nome **default**, cujo valor é usado no lugar da string vazia. Existem duas formas de se usar um valor padrão (*default*). A primeira e mais óbvia é usar o atributo default da tag (ver Listagem 3). A outra forma é não especificar o atributo default, deixando o valor a ser utilizado no corpo da tag (ver Listagem 4).

```
1 <b>Hello <c:out value='${user}' default='guest' />.</b>
```

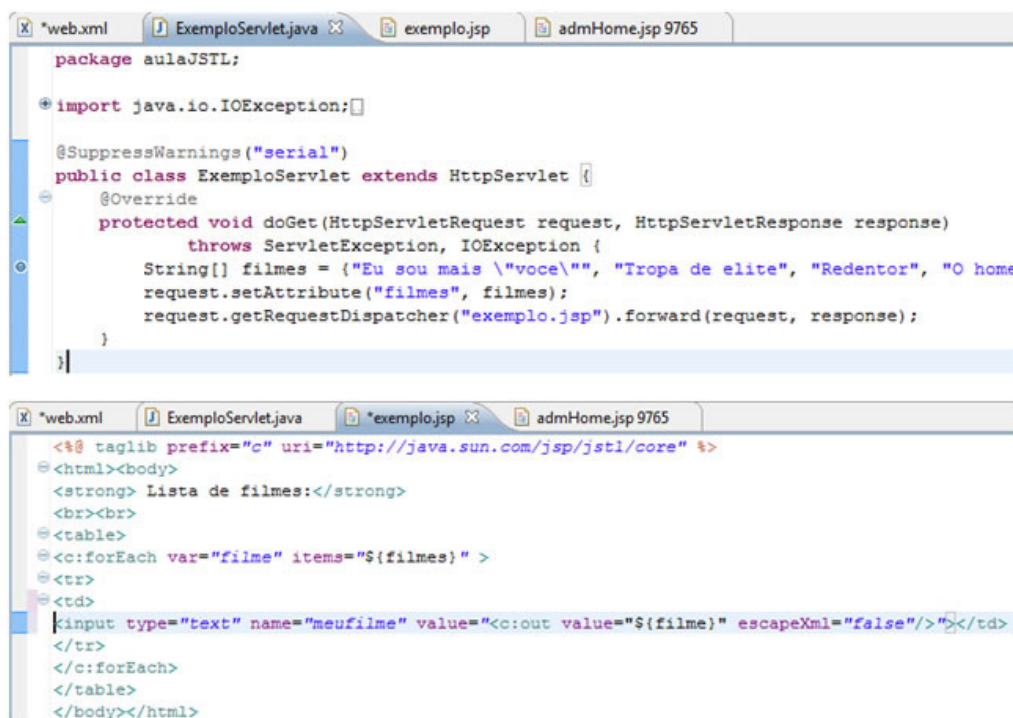
**Listagem 3** - Código JSP que usa o atributo default para exibir “guest” se \${user} for vazio



```
1 <b>Hello <c:out value='${user}'> Guest</c:out>.</b>
```

**Listagem 4** - Código JSP que usa o *body* da tag `c:out` para exibir "guest" se `${user}` for vazio

O terceiro atributo da tag `<c:out>` é o `escapeXml`, um atributo booleano cujo valor padrão é `true`. O `escapeXml` é usado para verificar se é necessário trocar caracteres especiais por seus respectivos códigos em HTML, como `"` para as aspas duplas. Isso evita que, se a tag for usada dentro de uma marcação HTML, o resultado da avaliação da expressão quebre o HTML gerado. Veja o exemplo a seguir:



```
package aulaJSTL;

import java.io.IOException;

@SuppressWarnings("serial")
public class ExemploServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        String[] filmes = {"Eu sou mais \"voce\"", "Tropa de elite", "Redentor", "O home
        request.setAttribute("filmes", filmes);
        request.getRequestDispatcher("exemplo.jsp").forward(request, response);
    }
}

<? taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" ?>
<html><body>
<strong> Lista de filmes:</strong>
<br><br>
<table>
<c:forEach var="filme" items="${filmes}" >
<tr>
<td>
<input type="text" name="meufilme" value="<c:out value='${filme}' escapeXml=false"/></td>
</tr>
</c:forEach>
</table>
</body></html>
```

**Figura 01** - Exemplo de código (e tela) usando `escapeXml`



← → ↺ 🏠 🌐 localhost:8080/aulaJSTL/exemplo

**Lista de filmes:**

<input type="text" value="Eu sou mais"/>
<input type="text" value="Tropa de elite"/>
<input type="text" value="Redentor"/>
<input type="text" value="O homem que copiava"/>

Perceba que a palavra **você** não apareceu na primeira caixa de texto porque essa palavra está entre aspas duplas, gerando um código HTML como:

```
<input type="text" name="meu filme" value="Eu sou mais "você"">
```

Dessa forma, a primeira aspas duplas do “você” está fechando, antecipadamente, o valor do atributo value, deixando a palavra você de fora. Note o destaque no valor associado ao atributo value. Perceba também que no código JSP do exemplo a tag `c:out` não está “ignorando” o caractere que representa as aspas duplas (`escapeXml=false`). Caso isso não tivesse sido indicado, ou seja, fosse utilizado o valor padrão (`escapeXml=true`), o texto gerado seria:

```
<input type="text" name="meu filme" value="Eu sou mais " você"">
```

Já a `tag c:catch` funciona como um complemento ao mecanismo de controles de erro do JSP. Essa tag permite ao desenvolvedor capturar exceções que não são de partes cruciais da página e tratar elas sem que haja a necessidade de invocar a página de erro, que vimos em aulas anteriores. Qualquer exceção que seja lançada em código encapsulado por essa tag é armazenada em uma variável definida pelo atributo `var` e pode ser utilizada para o que for necessário dentro do escopo da página, como podemos ver no exemplo abaixo:

```
1 <c:catch var="excecao">
2   <% int x = 28/0 ;%>
3 </c:catch>
4 <c:if test="{excecao!=null}">
5   <p>Ocorreu um erro pois excecao!=null </p>
6   <p>Mensagem do erro: {excecao.message}</p>
7 </c:if>
```

## Atividade 01

---

1. Um dos caracteres que deve ser escapado quando renderizado pela tag `c:out` já foi visto. Quais outros caracteres devem ser escapados? Dica: são mais quatro. Tente buscar essa informação na internet.
2. Altere o exemplo da Figura 1 para que a tag `c:out` escape as aspas duplas.

3. Usando um dos caracteres identificados na questão 1, monte um cenário de erro similar ao da Figura 1, em que o nome do filme possuía aspas duplas.

## Suporte a variáveis

---

O próximo conjunto de funcionalidades da taglib core que veremos é o suporte a variáveis. Com esse conjunto de tags, podemos definir e remover variáveis dentro do escopo da requisição JSP que está sendo processada. Existem duas tags que são responsáveis por esse suporte: `<c:remove>` e `<c:set>`. Começamos pela tag `c:set`!

A tag `c:set` permite que sejam definidas variáveis que não existiam no escopo da requisição JSP, à medida que essa página é processada. Essa tag tem o comportamento similar à diretiva `<jsp:setProperty ... />`, entretanto, ela pode definir variáveis tanto de JavaBeans quanto em variáveis de qualquer escopo da página JSP (página, requisição, sessão e aplicação). Existem duas formas de informar à tag qual o valor que ela deve armazenar. A primeira é usando o atributo `value`. A segunda é definindo o corpo dessa tag. Ambas as formas são mostradas nas Listagens 5 e 6, respectivamente.

```
1 <c:set var="userLevel" value="admin" />
```

**Listagem 5** - Código JSP que usa o atributo `value` para definir o valor da variável `userLevel`

```
1 <c:set var="userLevel">admin</c:set>
```

**Listagem 6** - Código JSP que usa o body da tag como valor para a variável `userLevel`

Adicionalmente, o atributo `value` pode ser uma expressão, assim como na tag `c:out`. Entretanto, se a avaliação dessa expressão resultar num valor nulo, a variável será removida do escopo.

Uma vez que explicamos a funcionalidade dessa tag, vamos agora mostrar e explicar cada um dos seus atributos.

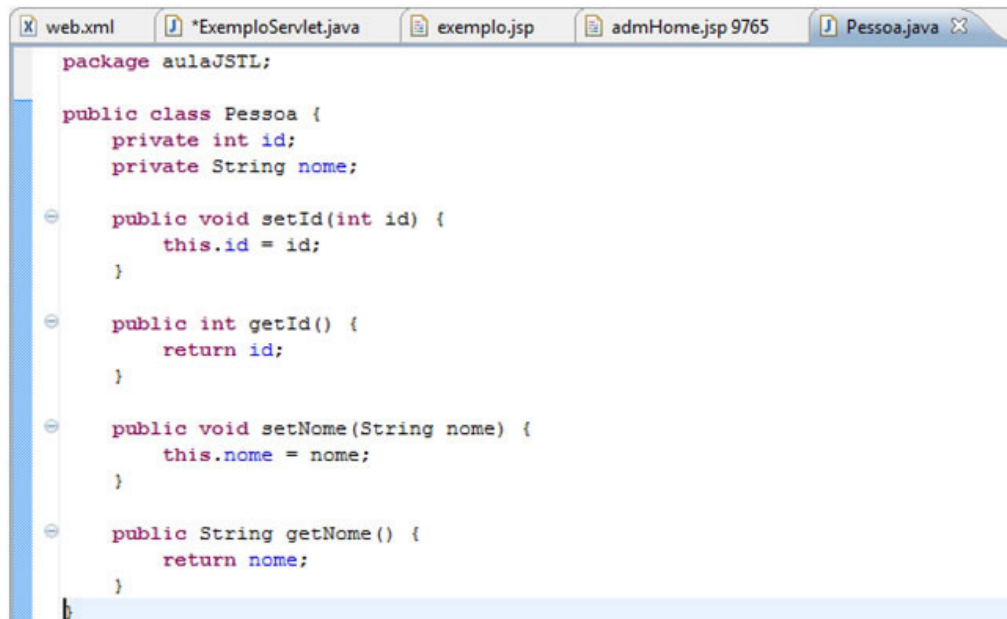
Atributo	Descrição	Obrigatório	Valor default
value	Valor ou expressão a ser armazenado na variável definida pelo atributo var	Não	Texto contido no Body da tag
target	Nome da variável que será modificada	Não	
property	Nome da propriedade do objeto definido pelo atributo target que será modificado	Não	
var	Nome da variável onde será armazenado o valor definido pelo atributo value	Não	
scope	Escopo da variável (Página, Requisição, Sessão ou Aplicação)	Não	

#### Quadro 5 - Atributos da tag c:set

Aqui precisamos diferenciar dois atributos, que são var e target. O atributo var é para definir variáveis no escopo da página, requisição, sessão e aplicação, enquanto que o atributo target define valores para propriedades em beans que estejam definidos em algum escopo. Quando usarmos var, não podemos usar target e vice-versa. Para ficar mais claro, veja os exemplos a seguir.

Definimos uma classe Pessoa que possui os atributos id e nome.

**Figura 02** - Código da classe Pessoa



```
package aulaJSTL;

public class Pessoa {
    private int id;
    private String nome;

    public void setId(int id) {
        this.id = id;
    }

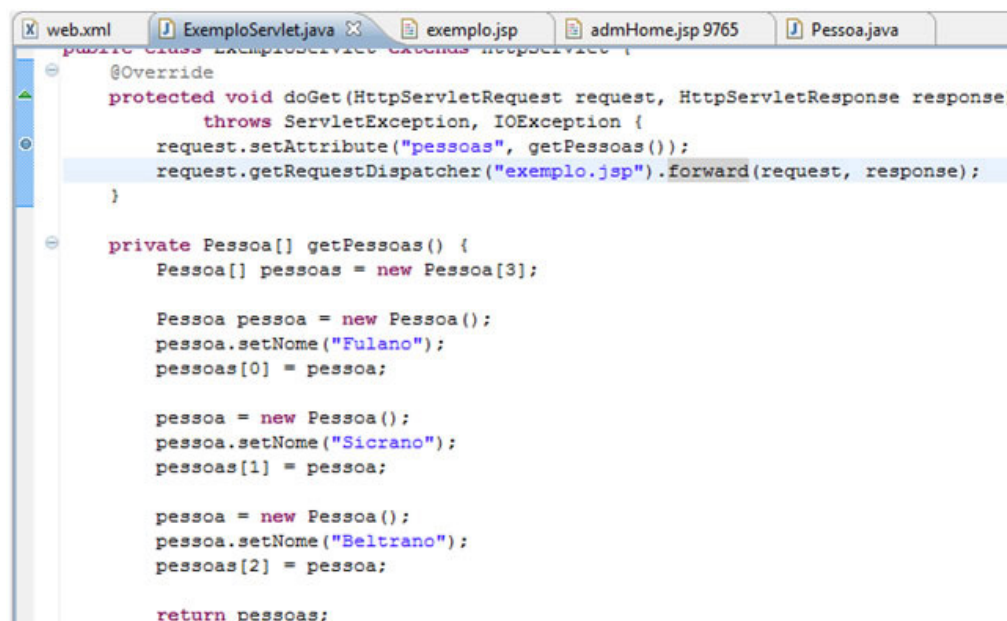
    public int getId() {
        return id;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public String getNome() {
        return nome;
    }
}
```

Alteramos o servlet de exemplo para listar pessoas ao invés de filmes.

**Figura 03** - Método doGet() do Servlet listando pessoas



```
public class ExemploServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        request.setAttribute("pessoas", getPessoas());
        request.getRequestDispatcher("exemplo.jsp").forward(request, response);
    }

    private Pessoa[] getPessoas() {
        Pessoa[] pessoas = new Pessoa[3];

        Pessoa pessoa = new Pessoa();
        pessoa.setNome("Fulano");
        pessoas[0] = pessoa;

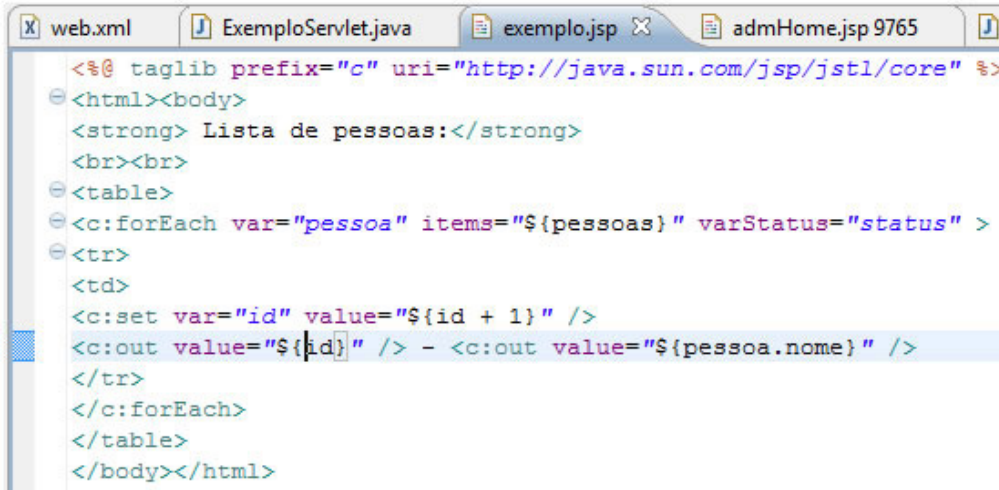
        pessoa = new Pessoa();
        pessoa.setNome("Sicrano");
        pessoas[1] = pessoa;

        pessoa = new Pessoa();
        pessoa.setNome("Beltrano");
        pessoas[2] = pessoa;

        return pessoas;
    }
}
```

Nesse JSP definimos uma variável com o nome de id cujo valor será o id + 1 e, logo em seguida, esse id será exibido na página juntamente com o nome da pessoa. Como estamos dentro de um loop, a cada iteração esse id será incrementado e o novo valor atribuído à variável id.

**Figura 04** - Código JSP responsável por gerar o HTML com a lista de pessoas

The screenshot shows an IDE with several tabs: 'web.xml', 'ExemploServlet.java', 'exemplo.jsp', and 'admHome.jsp 9765'. The 'exemplo.jsp' tab is active, displaying the following JSP code:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html><body>
<strong> Lista de pessoas:</strong>
<br><br>
<table>
<c:forEach var="pessoa" items="${pessoas}" varStatus="status" >
<tr>
<td>
<c:set var="id" value="${id + 1}" />
<c:out value="${id}" /> - <c:out value="${pessoa.nome}" />
</tr>
</c:forEach>
</table>
</body></html>
```

Por fim, vamos alterar diretamente o atributo id da classe pessoa dentro de cada iteração. Para isso, precisamos definir qual o objeto terá a sua propriedade modificada. Nesse exemplo, usaremos um recurso da taglib de loop chamado de `forEach`, que veremos mais adiante, que guarda em uma variável o contador da iteração. Perceba que para cada instância de pessoa que está sendo iterada, o valor do seu atributo id está sendo modificado com o contador da iteração + 1.

Aqui vale ressaltar que o atributo `target` recebe uma expressão que deve ser avaliada para um objeto; caso utilizássemos apenas "pessoa" ao invés de "\${pessoa}" a tag entenderia "pessoa" como uma string e tentaria definir o valor de `status + 1` numa propriedade id que não existe na classe String, o que causaria um erro em tempo de execução. Perceba, então, que para o atributo `target` devemos usar uma expressão que resulte em um objeto, e não apenas o identificador do objeto no escopo, como mostrado a seguir.

**Figura 05** - Código JSP alterado, responsável por gerar o HTML com a lista de pessoas.

The screenshot shows the same IDE as Figure 04, but with an additional tab 'Pessoa.java' and the 'exemplo.jsp' tab containing modified code:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html><body>
<strong> Lista de pessoas:</strong>
<br><br>
<table>
<c:forEach var="pessoa" items="${pessoas}" varStatus="status" >
<tr>
<td>
<c:set target="${pessoa}" property="id" value="${status.index + 1}" />
<c:out value="${pessoa.id}" /> - <c:out value="${pessoa.nome}" />
</tr>
</c:forEach>
</table>
</body></html>
```

Por último, o atributo `scope` define em qual escopo a variável será armazenada, ou seja, no escopo da página, requisição, sessão ou aplicação. Quando esse atributo não é definido, o escopo utilizado pela tag é o da página.

## Atividade 02

---

1. Experimente agora alterar, dentro do JSP, o atributo `nome` da classe `Pessoa` usando `var` e `target`.
2. Implemente duas páginas JSP, utilizando JSTL, para exibir a tabuada de um número. A primeira página irá exibir um formulário solicitando um número e irá chamar outra página JSP. A segunda página JSP irá exibir a tabuada do número informado na primeira página.

## Suporte a variáveis II

---

Agora, veremos a tag `c:remove`. Essa tag permite remover do escopo as variáveis que não mais serão utilizadas. Para isso, basta informar o nome da variável e o escopo onde ela se encontra. Aqui, ao contrário da tag `c:set`, que se o escopo não for definido utiliza como default o escopo da página, se o escopo não for definido a tag `c:remove` procura pela variável a ser removida em todos os escopos.

Uma vez que explicamos a funcionalidade dessa tag, vamos agora mostrar e explicar cada um dos seus atributos.

Atributo	Descrição	Obrigatório	Valor <i>default</i>
<code>var</code>	Nome da variável a ser removida	Sim	
<code>scope</code>	Nome da variável a ser removida	Não	Todos os escopos

**Quadro 6** - Atributos da tag `c:remove`

# Gerenciamento de URLs e controle de fluxo

---

Você viu, nas páginas anteriores, as tags de suporte a variáveis e miscelâneas. Continuaremos os estudos mostrando as tags da taglib core que tratam de gerenciamento de URLs e controle de fluxo. Começaremos, então, pelo gerenciamento de URLs.

Existem três tags básicas para o gerenciamento de URLs. Essas tags são `c:url`, `c:import` e `c:redirect`. Essas três tags podem fazer uso de outra tag que permite a passagem de parâmetros para as funcionalidades de gerenciamento. Essa tag é `c:param`. Então, vamos aos detalhes desse assunto!

A tag `c:url` constrói uma URL aplicando as regras apropriadas de reescrita de URLs. E o que seria isso? Reescrever uma URL significa escrever a URL de uma forma mais completa, por exemplo. A Listagem 7 apresenta o uso da tag `c:url` da taglib core.

```
1 <c:url value="/editProfile.do" var="profileLnk">
2   <c:param name="id" value="${user.id}"/>
3 </c:url>
4 <a href='<c:out value="${profileLnk}"/>'>Edit Profile</a>
```

Listagem 7 - Código JSP que utiliza a funcionalidade de reescrita de URL

Perceba que na linha 01 estamos indicando que o valor definido no atributo `value` será transformado em uma URL a partir da URL base. Como assim? Se você está acessando o JSP **`http://localhost:8080/aula07/profile.jsp`**, nesse caso a URL base seria **`http://localhost:8080/aula07/`**, então, a URL gerada será a URL base concatenada ao atributo `value` da tag, o que resultaria em **`http://localhost:8080/aula07/editProfile.do`**. Perceba também que na linha 02, dentro da tag `c:url`, utilizamos a tag `c:param` para passar o parâmetro chamado “id” que, no caso de nosso exemplo, tem o valor “`${user.id}`”. Então a URL final será gerada com o parâmetro “id=ID\_DO\_USUARIO”. Caso o atributo “user.id” do exemplo tenha o valor 22 a URL final ficaria: **`http://localhost:8080/aula07/editProfile.do?id=22`**



A informação mais importante com relação aos parâmetros é que eles são passados via a própria URL. Cuidado, porém, pois alguns caracteres não são permitidos na URL. Dessa forma, é necessário que esses caracteres sejam codificados, o que NÃO é feito pela tag `c:url`. Espaços em branco, `&`, `?` etc. devem ser codificados antes de serem utilizados como parâmetros nas URLs. E como isso é feito? A tag `c:param` já cuida disso para você automaticamente, ou seja, é bastante aconselhável que se use a tag `c:param` quando se deseja passar parâmetros tanto para a tag `c:url` quanto para as demais tags de gerenciamento de URLs.

Uma vez que explicamos a funcionalidade dessa tag, vamos agora mostrar e explicar cada um dos seus atributos

Atributo	Descrição	Obrigatório	Valor default
value	URL que será processada. Pode ser uma expressão ou a própria URL.	Sim	
var	Nome da variável que guardará o valor processado da URL.	Não	
scope	Escopo onde a variável definida no atributo var será armazenada.	Não	Página

**Quadro 7** - Atributos da tag `c:url`

O exemplo anterior já mostrou uma forma de uso para funcionalidade `c:url`, porém, vamos explicar cada um dos seus atributos. O atributo `value` define qual a URL que será processada. O valor definido nesse atributo pode ser uma string, que representa uma URL diretamente, ou uma expressão, que será avaliada antes de ser processada pela tag.

O atributo `var` define qual o nome da variável onde será armazenado o resultado do processamento da URL, assim, poderemos usar esse resultado em vários lugares dentro da página usando a funcionalidade `c:out`, por exemplo.

O atributo `scope` define em qual escopo a variável definida pelo atributo `var` será armazenada. Se ele não for especificado, o default é a página.

## Gerenciamento de URLs e controle de fluxo II

Continuando com as tags de gerenciamento de URLs, temos a tag `c:import`. Essa tag permite que você inclua dentro da sua páginas JSP outras páginas, que podem estar no mesmo servidor ou em outros. Além disso, essas páginas devem conter trechos de código e não páginas inteiras com marcações `<html><body>...</body></html>`.

Uma vez que explicamos a funcionalidade dessa tag, vamos, agora, mostrar e explicar cada um dos seus atributos.

Atributo	Descrição	Obrigatório	scope
url	URL do recurso (HTML, JSP, REST etc.) a ser importado.	Sim	
var	Nome da variável que guardará o conteúdo importado.	Não	
scope	Escopo onde a variável definida no atributo var será armazenada.	Não	Página

**Quadro 8** - Atributos da tag `c:import`

O recurso a ser importado é definido pelo atributo `url`. Esse recurso pode ser um arquivo HTML, um JSP, um serviço etc. Assim como a tag `c:url`, também podemos passar parâmetros para a URL, utilizando a tag `c:param`. Além disso, os demais atributos têm o mesmo comportamento visto com a tag `c:url`. Veja a seguir um exemplo da importação do código HTML de uma página e em seguida a sua impressão:

```
1 <c:import var="data" url="https://portal.imd.ufrn.br/portal/" />
2 <c:out value="${data}" />
3
```

A última tag de gerenciamento de URLs é a tag `c:redirect`. Essa tag permite que o fluxo da requisição seja desviado para outra URL. Dessa forma, ao processar uma página JSP que usa a tag `c:redirect`, ao encontrá-la, o fluxo é desviado para outra página dentro do servidor. O exemplo abaixo mostra a páginas `index.jsp` que redireciona o fluxo para o servlet de exemplo.

**Figura 06** - Exemplo de código utilizando `c:redirect`



Explicada a tag, vejamos o seu atributo.

Atributo	Descrição	Obrigatório	Valor default
url	URL do recurso para o qual o fluxo será redirecionado.	Sim	

**Quadro 9** - Atributos da tag `c:redirect`

Além de possuir este atributo único, responsável por indicar a URL para a qual ocorrerá o redirecionamento, a tag `c:redirect` também pode passar parâmetros utilizando a tag `c:param`, como visto nas tags anteriores.

Concluimos aqui as tags de gerenciamento de URLs da taglib `core`. Continuaremos depois das atividades com as tags de controle de fluxo.

## Atividade 03

1. Use a tag `c:import` para criar uma estrutura de páginas que possuam um header, um footer e um menu lateral e use essa estrutura em duas páginas JSPs.
2. Modifique o fragmento de página que representa o header para receber um parâmetro que é o título da página. Dica: Para usar o parâmetro na página do header, use a seguinte expressão `${param.nomeDoParametro}`.

## tag de loop

---

As funcionalidades de controle de fluxo da taglib core compreendem as tags de loop e de testes condicionais. Começaremos falando da tag de loop e concluiremos a taglib core mostrando as tags condicionais.

Para controlar loops dentro do JSP, a taglib core usa a tag `c:forEach`. Ela é responsável por fornecer uma maneira simples de iterar em coleções (arrays, listas, mapas etc.) ou de iterar um número definido de vezes. Tudo que estiver dentro do corpo dessa tag será repetido enquanto houver elementos na coleção ou o limite de vezes ainda não foi atingido.

Já vimos nos exemplos anteriores o uso dessa tag quando iteramos por uma coleção de filmes e pessoas exibindo seus nomes. Agora, vamos entender com mais detalhes o funcionamento da tag usando como exemplo a Listagem 8.

```
1 <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
2 <html>
3   <body>
4     <strong> Lista de filmes:</strong>
5     <br><br>
6     <table>
7       <c:forEach var="filme" items="{filmes}"><tr><td>${filme}</td></tr></c:forEach>
8     </table>
9   </body>
10 </html>
```

**Listagem 8** - Uso da tag `c:forEach`

No exemplo acima, a lista de filmes é um array de strings. Na linha 06, estamos dizendo que todo código dentro da tag `c:forEach` (linhas 07 a 09) será repetido enquanto houver elementos no array de filmes. O atributo `items` pode receber tanto uma expressão, como no caso acima, como uma lista de textos separados por vírgulas:

`<c:forEach var="filme" items="A volta dos que não foram, A mordida do banguelo, Um cavalo morto é um animal sem vida" >`.

No exemplo visto anteriormente (Figura 1) vimos que `#{filmes}` é um array que foi definido como um atributo da requisição dentro do nosso servlet de exemplo. O atributo `var` define o nome da variável onde cada elemento da iteração será armazenado.

Uma vez que explicamos a funcionalidade dessa tag, vamos, agora, mostrar e explicar cada um dos seus atributos.

Atributo	Descrição	Obrigatório	Valor default
begin	Índice do primeiro item a ser processado. Inicia em zero.	Não	0
end	Índice do último item a ser processado.	Não	Último item
step	Processa cada enésimo elemento. Por exemplo, se <code>step</code> for dois, processa o segundo, quarto, sexto e assim por diante.	Não	1
var	Nome da variável onde o elemento corrente da iteração será armazenado. Só existe durante a execução do loop.	Não	
items	Coleção, Iterador, Mapa ou Array para ser iterador.	Não	

Atributo	Descrição	Obrigatório	Valor default
	Nome da propriedade que armazenará o status do loop. Esse objeto que representa o status possui quatro atributos:		
varStatus	<ul style="list-style-type: none"> <li>• Index – posição do item corrente.</li> <li>• Count – número de vezes que o loop foi executado (começa em 1).</li> <li>• First – booleano que indica se é a primeira iteração.</li> <li>• Last – booleano que indica se é a última iteração</li> </ul>	Não	

#### Quadro 10 - Atributos da tag c:forEach

O funcionamento básico da tag c:forEach já foi explicado. Agora, mostraremos o que cada atributo representa para a tag.

O atributo begin informa à tag qual deve ser o índice que o loop iniciará a iteração. Esse atributo tem uma semântica bastante similar à variável de controle de um loop for da linguagem Java. Em um loop for Java, temos a seguinte construção for (int i = 0; i < 10; i++). Se definirmos o atributo begin como 5, por exemplo, é como se tivéssemos definindo a variável i do loop Java como 5. De forma análoga funciona o atributo end. Nesse caso, a sua analogia é com a constante 10 do loop Java. O atributo step, por analogia, seria a condição de incremento do loop Java, ou seja, o i++. Para finalizarmos, da mesma forma que a variável i só existe dentro do contexto do loop, a variável que representa o valor corrente da iteração (indicada pelo atributo var) só pode ser usada dentro da tag c:forEach. Na Listagem 9, temos um exemplo correto na linha 01 da variável filme e um exemplo incorreto de uso na linha 02, que tenta usar a variável fora de seu escopo.

```

1 <c:forEach var="filme" items="${filmes}">${filme}</c:forEach>
2 ${filme}

```

### Listagem 9 - Uso indevido da variável **filme** fora da tag `c:forEach`

O atributo `items` pode ser uma expressão que avalia para uma coleção, mapa, iterador, array ou uma string separada por vírgulas, como já dito anteriormente.

Por último, e não menos importante, temos o atributo **varStatus**. Esse atributo, quando definido, coloca, no escopo da página durante a execução do loop, um objeto que possui atributos que auxiliam no controle do loop. No exemplo visto anteriormente, usamos esse atributo para definir o atributo `id` do objeto `pessoa`, como mostrado no exemplo abaixo.

**Figura 07** - Exemplo de código utilizando o atributo `varStatus`



```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html><body>
<strong> Lista de pessoas:</strong>
<br><br>
<table>
<c:forEach var="pessoa" items="${pessoas}" varStatus="status" >
<tr>
<td>
<td>
<c:set target="${pessoa}" property="id" value="${status.index + 1}" />
<c:out value="${pessoa.id}" /> - <c:out value="${pessoa.nome}" />
</tr>
</c:forEach>
</table>
</body></html>
```

## Atividade 04

1. Altere a tag `c:forEach` para imprimir apenas o primeiro e o terceiro elemento do array usando um dos atributos da tag.
2. Altere o uso da tag para imprimir a listagem invertida.
3. Utilize uma variável de status no seu `c:forEach` para colocar duas cores diferentes (azul e amarelo) em cada impressão do filme.

## tag `c:if`

Chegamos, agora, ao último conjunto de funcionalidades da taglib core que é o controle de fluxo condicional. Explicaremos a tag **`c:if`**.

Um exemplo clássico de controle de fluxo condicional é quando nós temos uma única página JSP que serve para todos os usuários de uma determinada aplicação e que, de acordo com o perfil do usuário logado, alguns itens são exibidos e outros não. Um exemplo de uso é mostrado na listagem 10:

```
1 <c:if test="${usuarioLogado.perfil eq 'ADMIN '}" >
2   <c:import url="/menuAdmin.jsp" />
3 </c:if>
```

**Listagem 10** - Uso da tag c:if para importar JSP para os usuários que possuem o perfil ADMIN

Perceba que na linha 01 nós testamos a condição em que o perfil do usuário logado é igual a ADMIN. Se essa condição for verdade, o menuAdmin.jsp será importado.

Uma vez que explicamos a funcionalidade dessa tag, vamos, agora, mostrar e explicar cada um dos seus atributos.

Atribuição	Descrição	Obrigatório	Valor default
test	Expressão condicional que será avaliada.	Sim	
var	Nome da variável onde será armazenado o resultado do teste da expressão condicional.	Não	
scope	Escopo da variável	Não	Página

**Quadro 11** - Atributos da tag c:if

Como acabamos de ver, o atributo test é a expressão condicional que será avaliada para saber se o corpo da tag será processado ou não. O comportamento dos demais atributos é similar ao que já vimos diversas vezes ao longo da aula.



## Atividade 05

---

1. Modifique o exemplo da listagem de filmes para só listar os filmes, quando a variável `varStatus.index` for maior que 1.
2. Agora, modifique o exemplo para só listar os filmes que começam com a letra "A".
3. Busque em fontes alternativas o funcionamento da tag `<c:choose>` e a compare com uma estrutura If-Then-Else.

## Leitura Complementar

---

Para complementar seu aprendizado, veja o assunto abordado nas seguintes referências. A primeira apresenta um guia de referência rápida do JSTL, poderá ser bastante útil para você ao longo do curso. A segunda referência é o tutorial oficial de Java para JSTL, sendo um ótimo recurso para reforço dos seus conhecimentos. Analise essas referências de acordo com o conteúdo já visto nas aulas.

- <<http://cs.roosevelt.edu/eric/books/JSP/jstl-quick-reference.pdf>>
- <<http://download.oracle.com/javaee/5/tutorial/doc/bnakc.html>>
- <<https://docs.oracle.com/javaee/1.4/tutorial/doc/JSPIntro7.html>>

## Resumo

---

Hoje vimos as tags envolvidas na taglib core do JSTL. Aprendemos como lidar com as tags de miscelânea, de suporte a variável, de controle de fluxo e de manuseio de URL. Conhecendo essas tags é possível desenvolver bastante coisa de forma mais organizada e de fácil manutenção. Na próxima aula veremos um pouco mais sobre JSTL, com algumas novas tags, principalmente ligadas a Internacionalização. Até lá!

## Autoavaliação

---

Descreva o funcionamento das tags (incluindo seus atributos) que você aprendeu nesta aula para:

1. Gerenciamento de URLs
2. Controle de fluxo.

# Referências

---

AHMED, K. Z.; UMRYSH, C. E. **Desenvolvendo aplicações comerciais em Java com J2EE J2EE e UML**. Rio de Janeiro: Ciência Moderna, 2003. 324 p.

BASHAM, Bryan; SIERRA, Kathy; BATES, Bert. **Head First Servlets and JSP: passing the Sun Certified Web Component Developer Exam (SCWCD)**. 2nd ed. O'Reilly Media, 2008.

CATTELL, Rick; INSCORE, Jim. **J2EE: criando aplicações comerciais**. Rio de Janeiro: Editora Campus, 2001.

HALL, Marty. **More Servlets and JavaServer Pages (JSP)**. New Jersey: Prentice Hall PTR (InformIT), 2001. 752 p. Disponível em: <<http://pdf.moreservlets.com/>>. Acesso em: 11 maio 2012.

HALL, Marty; BROWN, Larry. **Core Servlets and JavaServer Pages (JSP)**. 2nd ed. New Jersey: Prentice Hall PTR (InformIT), 2003. 736p. (Core Technologies, 1). Disponível em: <<http://pdf.coreservlets.com/>>. Acesso em: 11 maio 2012.

HYPERTEXT Transfer Protocol: HTTP/1.1 – Methods Definition. Disponível em: <<http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html>>. Acesso em: 11 maio 2012.

J2EE Tutorial. Disponível em: <<http://java.sun.com/javaee/reference/tutorials/>>. Acesso em: 11 maio 2012.