



**INSTITUTO  
FEDERAL**

Paraíba

Campus  
Cajazeiras

# PROGRAMAÇÃO P/ WEB 1

## 6. JSP

(BASEADO NO MATERIAL DO PROF. FABIO GOMES)

**PROF. DIEGO PESSOA**

✉ [DIEGO.PESSOA@IFPB.EDU.BR](mailto:DIEGO.PESSOA@IFPB.EDU.BR)

🐙 @DIEGOEP



**CST em Análise e  
Desenvolvimento  
de Sistemas**

---

# EXPRESSION LANGUAGE

# Introdução

- As páginas JSP facilitam a geração de páginas com conteúdo dinâmico, mas possuem limitações:
  - Muitas páginas contém uma grande quantidade de código java;



# Introdução

- Isto traz alguns problemas:
  - Ainda existe uma grande dependência entre as funções de web designer e desenvolvedor da aplicação;
    - ✓ Embora tal dependência seja menor do que nos servlets;
  - Muitas páginas ainda são difíceis de serem mantidas;

# Introdução

- Para reduzir estes problemas, precisamos de mecanismos que nos permitam diminuir a quantidade de código java nas páginas;
- Uma solução para este problema é o uso da Expression Language;



# Expression Language

- Linguagem desenvolvida para diminuir a quantidade de scripts em páginas JSP;
- Foi introduzida a partir da versão 2.0 da especificação JSP;
  - E a partir da versão 6.0 do Apache Tomcat;

# Expression Language

- Importante:
  - Expression Language não é java;
  - Ela é uma forma de acessar objetos java sem usar java;



# Expression Language

- Na Expression Language, expressões tem a seguinte forma:
  - $\{parte_1.parte_2.parte_3.,...,parte_N\}$
- O operador . é chamado de operador de conveniência;



# Expression Language

- A primeira parte pode se referir a dois tipos de objeto:
  - Um objeto implícito (da Expression Language);
  - Um atributo em qualquer escopo (página, requisição, sessão ou aplicação);
- ✓ O valor do atributo pode ser um java bean ou um objeto que implementa a interface Map;

# Expression Language

- As demais partes correspondem a uma propriedade do objeto definido na primeira parte;
- Por isso, os nomes das propriedades devem seguir as regras de nomeação de identificadores em java;

# Declarações

- Se o objeto x for um Map, podemos imprimir o valor de uma chave da seguinte forma:
  - `${x.nome_da_chave}`
- Se o objeto x for um java bean, podemos imprimir o valor de uma propriedade da seguinte forma:
  - `${x.nome_da_propriedade}`
- Falaremos sobre objetos implícitos mais tarde;

# Declarações

- Exemplos de declarações:
  - `${person.name}`
  - `${person.dog.name}`
  - `${properties.home}`

# Declarações

- Também podemos especificar uma EL usando o símbolo [] no lugar do ponto;
- Neste caso, o nome da chave ou do atributo deve ser especificado entre os colchetes;
- O nome deve ser especificado entre “”;

# Declarações

- Exemplos:
  - `${person["name"]}`
  - `${properties["home"]}`

# Declarações

- Esta forma de declaração é mais flexível, pois:
  - Dá suporte às mesmas opções oferecidas com o operador de conveniência;
  - No caso de atributos, também permite que o seu valor seja um array ou um objeto que implementa a interface List;



# Declarações

- Neste tipo de declaração, quando a primeira parte é um array ou uma lista, especificamos:
  - O nome do array ou da lista;
  - A posição do elemento cujo valor deve ser acessado;
    - ✓ Iniciando a partir da posição 0;

# Declarações

- Exemplos de declarações nas quais o elemento da esquerda é uma lista ou um array:
  - `${musicList}`
  - `${musicList[0]}`

# Declarações

- Na EL, o valor de um índice também pode ser passado usando aspas:
  - Exemplo: `${musicList["1"]}`
- Nestes casos, o valor do índice é convertido para inteiro em tempo de execução;
  - A expressão acima tem o mesmo efeito da expressão `${musicList[1]}`;

# Declarações

- Suponha um servlet com o código:

```
String[] fruits = {"Maçã", "Banana", "Pera", "Melão"};  
request.setAttribute("frutas", fruits);  
RequestDispatcher dispatcher = request.getRequestDispatcher("show.jsp");  
dispatcher.forward(request, response);
```

- E a seguinte expressão na página JSP:

<p> A primeira fruta é **`${frutas[0]}`** </p>

O resultado final da expressão será o string “Maçã”

# Declarações

- Caso o valor seja passado não seja um string literal, o container o trata como um atributo;
- O valor do atributo, que é avaliado pelo container em tempo de execução, é usado para a resolução da expressão;

# Declarações

- Suponha um servlet com o código:

```
Map<String,String> map = new HashMap();  
map.put("Forró", "Luiz Gonzaga");  
map.put("MPB", "Caetano Veloso");  
map.put("Rock", "Titãs");  
map.put("Brega", "Reginaldo Rossi");  
request.setAttribute("cantores", map);  
request.setAttribute("genero", "Forró");  
RequestDispatcher dispatcher = request.getRequestDispatcher("show.jsp");  
dispatcher.forward(request, response);
```

- E uma página com a expressão:  
<p> O cantor é **`${cantores[genero]}`** </p>

# Declarações

- O container JSP trata o token “genero” como um atributo e o substitui pelo seu respectivo valor;
  - A expressão fica com a forma `${cantores[“Forró”]}`
- A expressão final é avaliada e o seu resultado é o string “Luiz Gonzaga”;



# Expressões Aninhadas

- Em EL, podemos também aninhar declarações;
- Neste caso, as expressões mais internas são resolvidas primeiro;

# Expressões Aninhadas

- Suponha um servlet com o código:

```
Map infos = new HashMap();  
infos.put("nome", "Joao");  
infos.put("idade", "22");  
infos.put("futebol", "Atlético de Cajazeiras");  
String[] preferences = {"futebol", "musica", "programação"};  
session.setAttribute("preferences", preferences);  
session.setAttribute("infos", infos);
```

- E a expressão: `${infos[preferences[0]]}`

# Expressões Aninhadas

- A declaração mais interna `"preferences[0]"` é resolvida primeiro, e a expressão fica com a forma `${infos[futebol]}`;
- Após a resolução da nova declaração, o valor final da expressão é o string `"Atlético de Cajazeiras"`;

# Expressões Aninhadas

- Suponha um servlet com o código:

```
ArrayList nums = new ArrayList();  
nums.add("1");  
nums.add("2");  
nums.add("3");  
session.setAttribute("numbers", nums);  
String[] favoriteMusic = {"Zero 7", "Tahiti 80", "BT", "Frou Frou"};  
session.setAttribute("musicList", favoriteMusic);
```

# Expressões Aninhadas

- Qual será o valor das seguintes expressões?
  - `${musicList[numbers[0]]}`
  - `${musicList[numbers[0] + 1]}`
  - `${musicList[numbers[2]]}`
  - `${musicList[numbers[numbers[1]]]}`



# Expressões Aninhadas

- Qual será o valor das seguintes expressões?
  - $\text{\${musicList[numbers[0]]}}$  = Tahiti 80
  - $\text{\${musicList[numbers[0] + 1]}}$  = BT
  - $\text{\${musicList[numbers[2]]}}$  = Frou Frou
  - $\text{\${musicList[numbers[numbers[1]]]}}$  = Frou Frou

# Objetos Implícitos

- Assim como uma página JSP, a EL tem uma série de objetos implícitos:
  - `pageScope`, `requestScope`, `sessionScope`, `applicationScope`:
    - ✓ Usados para recuperar os valores de atributos definidos nos escopos página, requisição, sessão e aplicação, respectivamente;
    - ✓ Estes objetos são representados internamente como maps;



# Objetos Implícitos

- Assim como uma página JSP, a EL tem uma série de objetos implícitos;
  - **param:**
    - ✓ Map usado para recuperar o valor de um parâmetro da requisição;
  - **paramValues:**
    - ✓ Map usado para recuperar os valores de um parâmetro da requisição;

# Objetos Implícitos

- Assim como uma página JSP, a EL tem uma série de objetos implícitos;
  - **header e headerValues:**
    - ✓ Map usado para recuperar o valor de um cabeçalho da requisição;
  - **cookie:**
    - ✓ Map usado para recuperar o valor de cookies enviados na requisição;

# Objetos Implícitos

- Assim como uma página JSP, a EL tem uma série de objetos implícitos;
  - `initParam`:
    - ✓ Usado para recuperar o valor de parâmetros de inicialização da aplicação;
  - `pageContext`:
    - ✓ Java bean que representa a referência para objeto que encapsula as informações do contexto da página;

# Objetos Implícitos

- Importante:
  - Os objetos implícitos da EL apresentados até agora são diferentes dos objetos implícitos disponibilizados pela tecnologia JSP;
  - Eles são apenas Maps que contém informações relativas a um determinado componente da aplicação;

# Objetos implícitos

```
<table width="70%" border="0" cellspacing="0" cellpadding="0">
  <tr>
    <td width="20%"><strong>Nome Completo:</strong></td>
    <td width="72%"><u>${param.name}</u></td>
  </tr>
  <tr>
    <td><strong>Cidade:</strong></td>
    <td><u>${param.city}</u></td>
  </tr>
  <tr>
    <td><strong>Idade:</strong></td>
    <td><u>${param.age}</u></td>
  </tr>
  <tr>
    <td><strong>Curso:</strong></td>
    <td><u>${param.course}</u></td>
  </tr>
</table>
```

# Objetos Implícitos

- Imprimindo valores de parâmetros da requisição:
  - Na EL, podemos obter valores de parâmetros da requisição através dos objetos `param` e `paramValues`;
  - O objeto `param` é usado para manipular atributos que possuem um único valor;

# Objetos Implícitos

- Imprimindo valores de parâmetros da requisição:
  - O objeto `paramValues` é usado para manipular parâmetros multivalorados;
  - Parâmetros multivalorados são manipulados como uma lista ou array;
    - ✓ Um índice é usado para se obter o valor desejado;



# Objetos Implícitos

- Imprimindo valores de parâmetros da requisição:
  - Caso o objeto `param` seja usado com um parâmetro multivalorado, a EL imprime o primeiro valor encontrado;
  - O objeto `paramValues` pode ser usado com um parâmetro simples, desde que seja usado o índice 0;

# Objetos Implícitos

- Imprimindo valores de parâmetros da requisição:
  - Vamos supor o seguinte código HTML:

```
<form name="form1" method="post" action="mostraParametros.jsp">  
  
  Nome: <input type="text" name="name" id="name">  
  
  Cidade: <input type="text" name="city" id="city">  
  
  Comida preferida 1: <input type="text" name="food" id="food">  
  
  Comida preferida 2: <input type="text" name="food" id="food">  
  
</form>
```

# Objetos Implícitos

- Imprimindo valores de parâmetros da requisição:
  - Neste exemplo, os parâmetros `name` e `city` são simples, mas o atributo `food` é multivalorado;
  - Desta forma, o atributo `food` será tratado como uma lista composta por dois elementos;

# Objetos Implícitos

- Mostrando os valores dos parâmetros do exemplo anterior:
  - `${param.name}`
  - `${param.city}`
  - `${paramValues.food[0]}`
  - `${paramValues.food[1]}`
  - `${paramValues.name[0]}`

# Objetos Implícitos

- Imprimindo cookies:
  - Podemos imprimir o valor de cookies através do objeto implícito `cookie`;
  - O nome do cookie que deve ser recuperado deve ser passado do lado direito da expressão;
  - A propriedade `value` é usada para obter o valor do mesmo;

# Objetos Implícitos

- Imprimindo cookies:
  - Exemplo: `${cookie.JSESSIONID.value}`
    - ✓ Esta expressão recupera o valor do cookie JSESSIONID;

# Objetos Implícitos

- O objeto implícito `pageContext`:
  - É o único objeto implícito que faz referência a um objeto manipulado por uma página JSP;
  - Ele é uma referência para o objeto implícito JSP `pageContext`, que é um java bean;

# Objetos Implícitos

- O objeto implícito `pageContext`:
  - Através deste objeto, podemos recuperar referência para os demais objetos implícitos das páginas JSP;
    - ✓ Requisição, resposta, sessão, etc;
  - Podemos usar este objeto para imprimir informações relacionadas a estes objetos em uma página JSP;



# Objetos Implícitos

- O objeto implícito pageContext:

```
<table width="70%" border="0" cellspacing="0" cellpadding="0">
  <tr>
    <td width="8%">&nbsp;</td>
    <td width="4%"><strong>ID:</strong></td>
    <td width="88%"><u>${pageContext.session.id}</u></td>
  </tr>
</table>
```

# Objetos Implícitos

- No exemplo anterior, `session` se refere ao objeto implícito da tecnologia JSP;
  - Que também é um java bean;

# Usando Funções

- ELs também podem ser usadas para imprimir o resultado de uma função;
  - Por exemplo, o resultado de um método;
- Para isso, definimos os métodos que podem ser invocados através destas expressões;
  - Mas estes métodos devem satisfazer algumas restrições;

# Usando Funções

- A utilização de funções por ELs requer a realização das seguintes etapas:
  - Implementar os métodos que poderão ser chamados;
  - Declarar estes métodos em um descritor de bibliotecas de tags;
  - Definir uma diretiva taglib na página JSP;
  - Definir uma EL para fazer a chamada do método;

# Usando Funções

- Implementando os métodos:
  - Para especificar métodos que podem ser chamados a partir de uma EL, criamos uma nova classe java;
    - ✓ Os métodos que serão chamados pela EL devem ser públicos e estáticos;
    - ✓ A classe deve ser colocada no diretório Web-Inf/classes da aplicação;

# Usando Funções

- Exemplo:

```
package src;

public class DateManager {

    public static String formatDate(int day, int month, int year){
        String date = "";
        if(day<10)
            date+="0";
        date+= day+"/";
        if(month<10)
            date += "0";
        date+=month+"/";
        date+=year;
        return date;
    }
}
```

# Usando Funções

- Declarando os métodos no descritor:
  - Uma vez criados os métodos, precisamos definir para a aplicação como ela deve mapear uma EL para a chamada do método desejado;
  - Esta informação é declarada em um arquivo chamado de **Tag Library Descriptor (TLD)**;
    - ✓ Falaremos mais sobre ele nos próximos capítulos do curso;

# Usando Funções

- Declarando os métodos no descritor:
  - O descritor, que é um arquivo XML, tem uma tag chamada uri, na qual definimos como o descritor será identificado na aplicação;
    - ✓ Esta informação será usada na diretiva taglib;
  - Cada método que pode ser chamado deve ser declarado no descritor através de um elemento **function**;



# Usando Funções

- Declarando os métodos no descritor:
  - Para cada elemento `function`, podemos definir as seguintes informações:
    - ✓ `name`:
      - Corresponde ao nome que a EL deve usar para chamar o método;
      - Não é necessário que o nome tenha o mesmo nome do método na classe, embora seja desejável;

# Usando Funções

- Declarando os métodos no descritor:
  - Para cada elemento `function`, podemos definir as seguintes informações:
    - ✓ `function-class`:
      - ▶ – Especifica o nome totalmente qualificado da classe na qual o método foi definido;
    - ✓ `function-signature`:
      - Especifica a assinatura do método que será utilizado;
      - Ajuda a evitar ambiguidade, já que o método pode ser sobrecarregado;

# Usando Funções

- Declarando os métodos no descritor:
  - O descritor de tags deve ser salvo no diretório WEB-INF da aplicação;
  - Ele deve ter a extensão tld;
  - Abordaremos o descritor de biblioteca de tags em mais detalhes nos próximos assuntos da disciplina;

# Usando funções

```
<taglib xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-jsptaglibrary_2_0.xsd"
  version="2.0">

<tlib-version> 1.2 </tlib-version>
<uri> MyFunctions </uri>
  <function>
    <name> formataData </name>
    <function-class> src.DateManager </function-class>
    <function-signature>
      String formatDate(int, int, int)
    </function-signature>
  </function>
</taglib>
```

# Usando Funções

- Definindo uma diretiva taglib na página JSP:
  - Para invocarmos os métodos definidos no descritor de tags em uma página JSP, precisamos declará-lo na página JSP;
  - Fazemos isso através da diretiva taglib;

# Usando Funções

- Definindo uma diretiva taglib na página JSP:
  - Ao usarmos esta diretiva, definimos o prefixo que identifica estas tags dentro da página;
  - Este prefixo sempre é usado para chamar o método a partir de uma EL;

# Usando Funções

- Definindo uma diretiva taglib na página JSP:
  - Ao usarmos esta diretiva, definimos o prefixo que identifica estas tags dentro da página;
  - Este prefixo sempre é usado para chamar o método a partir de uma EL;
  - O objetivo deste prefixo é evitar ambiguidades, uma vez que diferentes descritores podem ter métodos com a mesma assinatura;

# Usando Funções

- Definindo uma diretiva taglib na página JSP:
  - Nesta diretiva, além do prefixo, definimos a URI do descritor de tags;
  - O seu valor deve ser igual ao que usamos para definir a propriedade uri do descritor;



# Usando Funções

- Definindo uma diretiva taglib na página JSP:
  - Exemplo de declaração de uma diretiva taglib dentro de uma página JSP;

```
<%@ taglib prefix="my" uri="MyFunctions" %>
```

# Usando Funções

- Definindo uma EL pra chamar o método:
  - Feito tudo isso, definimos uma EL para realizar a chamada do método;
  - Esta chamada é realizada através da seguinte sintaxe:  
✓ `${prefixo:método}`
  - Importante: O nome do método deve ser o mesmo usado para definir a propriedade name do mesmo no descritor;

# Usando Funções

- Definindo uma EL pra chamar o método:
  - Exemplo:
    - ✓ `${my:formataData(1, 1, 2007)}`

# Operadores

- Podemos usar EL para imprimir o resultado de expressões;
- Operadores podem ser usados para avaliar estas expressões;
- A EL define operadores aritméticos, lógicos e relacionais;

# Operadores

- Operadores aritméticos:
  - + (soma)
  - - (subtração)
  - \* (multiplicação);
  - / (divisão)
  - div (quociente de divisão);
  - %, mod (ambos são usados para calcular o resto da divisão);

# Operadores

- Operadores Lógicos:
  - && e and (para a operação AND);
  - || e or (para a operação OR);
  - ! e not (para a operação NOT);

# Operadores

- Operadores relacionais:
  - == e eq (igualdade);
  - != e ne (diferença);
  - < e lt (menor);
  - > e gt (maior);
  - <= e le (menor ou igual);
  - >= e ge (maior ou igual);

# Operadores

- Exemplos de ELs com operadores:
  - `${num > 3}`
  - `${num le 12}`
  - `${list[0] and list[1]}`
  - `${myDog.name == "Rex"}`
  - `${cont + 1}`



## Trabalhando com Valores Nulos

- Diferentemente de JSP, a EL não gera exceções;
  - Então o que acontece quando a mesma se depara com valores nulos?
- A EL processa valores nulos de diferentes formas, de acordo com o tipo de situação;

## Trabalhando com Valores Nulos

- Quando ela é usada com um atributo, java bean ou propriedade inexistente, ela não imprime resultado algum;
  - Exemplo: `${foo.bar}`
    - ✓ Caso não exista um atributo ou um java bean chamado foo a EL não será impressa;
    - ✓ O mesmo acontece se ele existir, mas não tiver uma propriedade chamada bar;

# Trabalhando com Valores Nulos

- Caso um valor null aconteça numa expressão aritmética, ele é tratado como zero;
  - Exemplo: `${ 7 + foo }`
    - ✓ O valor da expressão será igual a sete, caso foo tenha um valor nulo;
- Caso ele aconteça em uma expressão lógica, ele é tratado como falso;
  - Exemplo: `${true and foo}`
    - ✓ O valor da expressão será igual a false, caso foo tenha um valor nulo;

---

# JSTL (JSP STANDARD TAG LIBRARY)

# Introdução

- Nós vimos que a Expression Language nos permite diminuir a quantidade de scripts numa página JSP;
- No entanto, ainda temos situações que não são resolvidas pelas ELs;
  - Por exemplo, ainda precisamos de scriptlets para realizar tarefas importantes como repetições, comandos condicionais, entre outras;

# Introdução

A JSP Standard Tag Lib (JSTL) foi proposta para amenizar este tipo de problema;

- Mas ela pode ser usada em conjunto com a Expression Language;
- Ela nos permite especificar tags, que são mapeadas para a execução de comandos java na página JSP;

# Introdução

- A especificação JSTL oferece uma biblioteca central de tags para a realização de várias tarefas;
  - Que pode ser livremente utilizada;
- Entretanto, o programador também pode desenvolver as suas próprias tags;

# Introdução

- A JSTL não faz parte da especificação JSP;
- Para usarmos essa especificação, devemos baixar os arquivos **jstl.jar** e **jstl-api.jar**:
  - <https://jstl.java.net>
- Estes dois arquivos devem ser adicionados à biblioteca da aplicação;



# A Diretiva Taglib

- As bibliotecas de tags usadas em uma página JSP devem ser declaradas em uma diretiva taglib;
- Esta diretiva é composta por dois atributos obrigatórios:
  - **prefix;**
  - **uri;**

# A Diretiva Taglib

- O atributo prefix:
  - Indica o prefixo que vai identificar a biblioteca de tags dentro da página;
  - Usado para solucionar conflitos de nomes;
  - Por convenção, usamos o prefixo **c** para identificar a biblioteca de tags central oferecida pela especificação JSTL;

# A Diretiva Taglib

- O atributo uri:
  - Usado para identificar a localização da biblioteca de tags;
  - Normalmente este valor é identificado através de uma URL;
  - A biblioteca central de tags JSTL está localizada na URI <http://java.sun.com/jsp/jstl/core>;

# A Diretiva Taglib

- Exemplo de declaração de uma diretiva taglib para a utilização da biblioteca de tags padrão JSTL:

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

# A Biblioteca Central de Tags

- Biblioteca central de tags da especificação JSTL;
  - Por isso a convenção de se usar c como prefixo para a mesma;
- Contém tags para a realização de diversas tarefas, como a construção de loops, desvios condicionais, etc;

# A Biblioteca Central de Tags

- Vamos, a partir de agora, dar uma olhada nas principais tags oferecidas nesta biblioteca;



# A Biblioteca Central de Tags

- Importante:
  - Vamos apenas cobrir as principais tags e suas principais características;
  - Você é responsável por estudar para aprofundar os seus conhecimentos;
  - ✓ A especificação da biblioteca pode ser usada livremente para isso;

# A Tag forEach

- Usada para iterar sobre uma coleção de objetos;
- A tag **forEach** tem um funcionamento parecido com o comando for das linguagens de programação;
- A coleção pode ser qualquer objeto que implemente **java.util.Collection** ou **java.util.Map**;



# A Tag forEach

- No corpo de uma tag `forEach` (e das demais tags) podemos executar:
  - **Marcações HTML;**
  - **Expression Languages;**
  - **Código JSP, como scriptlets e expressões;**

# A Tag forEach

- O atributo var:
  - Usado para especificar a variável de controle do loop;
  - A cada iteração, a variável de controle recebe uma referência para o elemento atual da coleção;
  - O seu tipo depende dos objetos contidos na coleção;

# A Tag forEach

- O atributo items:
  - Atributo que identifica a coleção sobre a qual a tag deve iterar;
  - Pode ser um array, ou qualquer objeto que implemente a interface `java.util.Collection` ou `java.util.Map`;

# A Tag forEach

- Os atributos `begin` e `end`:
  - Definem, respectivamente, os índices dos elementos onde a iteração deve começar e terminar;
  - Estes dois atributos são opcionais, e são usados em situações nas quais não queremos processar todos os elementos da coleção;

# A Tag forEach

- Os atributos begin e end:
  - Quando usados, ambos os valores devem receber valores inteiros;
    - ✓ Correspondendo aos índices de seus respectivos elementos;
  - Os valores default destes atributos correspondem aos índices do primeiro e do último elemento da coleção, respectivamente;

# A Tag forEach

- O atributo step:
  - Atributo opcional que define como a variável de controle será atualizada após cada iteração;
  - O incremento é definido com relação ao número de posições;
    - ✓ Por este motivo, o valor deste atributo deve ser um número inteiro;
  - O seu valor default é 1;

# A Tag forEach

- O atributo varStatus:
  - Permite definir uma variável para armazenar o status do loop;
  - O valor desta variável também é uma instância da classe `LoopTagStatus`;

# A Tag forEach

- Vamos supor o seguinte trecho de código definido em um servlet:

```
String[] esportes = {"Futebol", "Volley", "Basquete", "Tênis", "Judô", "Natação"};  
request.setAttribute("games", esportes);  
RequestDispatcher dispatcher = request.getRequestDispatcher("show.jsp");  
dispatcher.forward(request, response);
```



# A Tag forEach

- Exemplo 1:
  - Vamos agora supor que queremos imprimir os componentes deste array em uma tabela numa página JSP;

```
<table>
  <c:forEach var="esporte" items="${games}">
    <tr> <td> ${esporte} </td></tr>
  </c:forEach>
</table>
```

# A Tag forEach

- Note que:

- A tag `forEach` é precedida pelo prefixo que se refere à biblioteca pela qual ela é oferecida;
- Com o atributo `var`, estamos indicando que a variável de controle será chamada de esporte;
- ✓ O escopo desta variável se limita à tag na qual ela foi definida;

# A Tag forEach

- **Note que:**
  - O valor do atributo **items** indica que a coleção a ser iterada corresponde ao valor do atributo **games**;
    - ✓ O **tipode coleção** também é identificado automaticamente pelo container;
    - ✓ Em cada **iteração**, imprimimos o valor atual da **variável esporte**;

# A Tag forEach

- Exemplo 1:
  - O exemplo mostrado tem o mesmo efeito do seguinte scriptlet:

```
<table>
<%
    String[] games = (String[]) pageContext.getAttribute("games");
    for(int k=0;k<games.length;k++){
%>
    <tr><td><%=games[k]%></td></tr>
<%} %>
</table>
```

# A Tag forEach

- Outros exemplos:

```
<table>
  <c:forEach var="esporte" items="${games}" begin="0" end="3">
    <tr> <td> ${esporte} </td></tr>
  </c:forEach>
</table>
```

```
<table>
  <c:forEach var="esporte" items="${games}" step="2">
    <tr> <td> ${esporte} </td></tr>
  </c:forEach>
</table>
```

# A Tag forEach

- Exemplos:

```
<table>
  <c:forEach var="esporte" items="${games}" varStatus="status">
    <tr> <td> Status:</td><td>${status.count}</td></tr>
  </c:forEach>
</table>
```



# A tag forEach

<%

```
String[] esportesAna = {"Volley", "Natação"};
Person pessoaAna = new Person();
pessoaAna.setName("Ana");
pessoaAna.setSports(esportesAna);

String[] esportesCarlos = {"Futebol", "Basquete"};
Person pessoaCarlos= new Person();
pessoaCarlos.setName("Carlos");
pessoaCarlos.setSports(esportesCarlos);

Collection pessoas = new ArrayList();
pessoas.add(pessoaAna);
pessoas.add(pessoaCarlos);
pageContext.setAttribute("people", pessoas);
```

%>

# A Tag forEach

- Para imprimirmos as informações desta coleção, usamos dois comandos forEach aninhados;
  - O primeiro itera sobre as pessoas da coleção;
  - O segundo itera sobre os esportes da pessoa corrente;



# A tag forEach

```
<table>
  <c:forEach var="pessoa" items="{people}">
    <tr>
      <td><b>Pessoa:</b></td>
      <td>${pessoa.name}</td>
    </tr>
    <tr> <td><b>Esportes:</b></td> </tr>
      <c:forEach var="esporte" items="{pessoa.sports}">
        <tr> <td> ${esporte}</td></tr>
      </c:forEach>
    <tr><td>&nbsp;</td></tr>
  </c:forEach>
</table>
```

# A Tag If

- Usada para fazer desvios condicionais;
  - Como o comando if de uma linguagem de programação;
- Uma determinada ação é executada mediante a ocorrência de uma determinada condição;

# A Tag If

- Importante:
  - Diferentemente da linguagem java, esta tag não tem uma cláusula ELSE para determinar o que deve ser feito caso a condição seja falsa;
  - Para isso, existe uma tag mais adequada;

# A Tag If

- A tag if é usada com os seguintes atributos:
  - **test:**
    - ✓ Define a condição que deve ser satisfeita para que as ações associadas à tag sejam executadas;
    - ✓ Este atributo é obrigatório e o seu valor deve ser uma expressão lógica;

# A Tag If

- A tag if é usada com os seguintes atributos:
  - **var:**
    - ✓ Atributo opcional que define uma variável de controle que vai armazenar o resultado do teste;
  - **scope:**
    - ✓ Atributo opcional que define o escopo da variável de controle;

# A Tag If

- Vamos agora ver um exemplo de aplicação da tag if;
- Vamos supor uma página na qual qualquer usuário pode ver os comentários postados, mas apenas membros podem postar novos comentários;

# A Tag If

- Exemplo:

```
<table>
  <c:forEach var="comment" items="${comments}" >
    <tr> <td> ${comment} </td></tr>
  </c:forEach>
</table>

<c:if test="${param.userType eq 'member'}" >
  <jsp:include page="inputComments.htm" />
</c:if>
```

# A Tag If

- **Note que:**
  - Todos os usuários que acessam a página podem ver todos os comentários postados;
  - Entretanto, a página que dá acesso ao formulário para o envio de novos comentários só é visível caso o valor do atributo userType seja igual a member;



# A Tag Choose

- Usada para seleção múltipla;
  - Semelhante ao comando Case da linguagem pascal ou o comando switch da linguagem java;
- Assim como nestes comandos, podemos definir ações para diferentes condições;
  - Edefinir ações para serem executadas quando nenhuma condição for satisfeita;

# A Tag Choose

- Cada desvio condicional da tag choose é definido com uma tag **when**:
  - Esta tag tem um atributo test, no qual definimos a condição que deve ser satisfeita para que o desvio seja executado;
- Uma tag **otherwise** pode ser usada para especificar o que deve ser feito caso nenhum desvio seja executado;

# A Tag Choose

- Exemplo:

```
<c:choose>
  <c:when test="{param.country eq 'Brasil'}" >
    Ah, então você deve gostar de futebol!
  </c:when>
  <c:when test="{param.country eq 'Alemanha'}" >
    Ah, então você deve gostar de cerveja!
  </c:when>
  <c:when test="{param.country eq 'Argentina'}" >
    Ah, então você deve gostar de tango!
  </c:when>
  <c:otherwise>
    Bem, então vamos descobrir do que você gosta!
  </c:otherwise>
</c:choose>
```

# A Tag Set

- Usada para criar e manipular o estado de objetos da aplicação;
- É parecida com a action `setProperty` da tecnologia JSP, mas é mais poderosa;
  - Na verdade, podemos obter as funcionalidades das actions `useBean` e `setProperty` juntas, e ainda podemos fazer mais algumas coisas;

# A Tag Set

- Com esta tag podemos:
  - Criar ou alterar atributos em qualquer escopo;
  - Criar ou alterar java beans;
  - Criar ou alterar uma entrada em um Map;

# A Tag Set

- Os atributos `var` e `target`:
  - A tag set deve ser usada com um atributo `var` ou `target`;  
✓ Mas só um deles pode ser usado;
  - O atributo `var` é usado para a manipulação de atributos;
  - O atributo `target` é usado para a manipulação de beans e valores de Maps;

# A Tag Set

- Outros atributos da tag set:
  - scope:
    - ✓ Indica o escopo do objeto que está sendo manipulado, seja ele atributo, bean ou map;
    - ✓ Caso ele não seja especificado, o container procura o mesmo na página, requisição, sessão e aplicação, respectivamente;

# A Tag Set

- Outros atributos da tag set:
  - **property:**
    - ✓ Atributo usado juntamente com o atributo target;
    - ✓ Caso o elemento definido no target seja um bean, este valor indica o atributo a ser atualizado;
    - ✓ Caso ele seja um Map, este atributo recebe o nome da chave a ser criada ou atualizada;



# A Tag Set

- Outros atributos da tag set:
  - value:
    - ✓ Corresponde ao valor a ser recebido pelo componente que está sendo definido pela tag;
    - ✓ O seu valor pode ser passado como um String ou como uma expressão EL;

# A Tag Set

- Outros atributos da tag set:
  - **value:**
    - ✓ O valor deste atributo também pode ser passado no corpo do tag;
      - Esta forma é apenas mais conveniente, quando o valor é grande para colocar como atributo;
      - Na prática, as duas formas funcionam do mesmo jeito;

# A Tag Set

- **Exemplo 1:** Usando a tag set para definir um novo atributo;
  - `<c:set var="login" scope="session" value="João" />`
- **Efeito:**
  - Caso este objeto não exista neste escopo, o container cria um atributo chamado login e com o valor João, e o adiciona à sessão com o usuário;
  - Caso o atributo já exista neste escopo, o valor dele é alterado para João;

# A Tag Set

- `<c:set var="login" value="{user.name}" />`
- **Exemplo 2:** Usando a tag set para definir um novo atributo:
- **Efeito:**
  - É criado (ou atualizado) um atributo com o nome login,
    - ▶ o valor da expressão e escopo page;
  - Se a expressão avaliar para null o atributo é removido;

# A Tag Set

- **Exemplo 3:** Usando a tag set para definir atributos;
  - `<c:set var="userName" scope="page" >`  
Joaquim José da Silva Xavier  
`</c:set>`
- **Efeito:**
  - Neste caso, o valor do atributo corresponde ao valor da tag;

# A Tag Set

- **Exemplo 4:** Usando a tag set para definir atributos;
  - `<c:set var="user" scope="page" value="${users.user1}" />`
- **Efeito:**
  - É criado um atributo com nome user e escopo página;
  - O valor deste atributo corresponde a um bean;
  - O tipo do bean é avaliado em tempo de execução;

# A Tag Set

- **Exemplo 5:** Usando a tag set para manipular beans;
  - `<c:set target="${user}" property="name" value="fabio" />`
- **Efeito:**
  - Ojava bean é criado (ou recuperado) e o valor da propriedade name é mudado para "fabio";

# A Tag Set

- **Exemplo 6:** Usando a tag set para manipular maps;
  - `<c:set target="{props}" property="so" value="windows" />`
- **Efeito:**
  - O container cria (ou altera) uma entrada no map com a chave "so" e o valor "windows";



# A Tag Set

- Importante: quando a expressão usada para definir o valor do atributo value é avaliada como null, o container realiza alguns tratamentos especiais;
- Este tratamento muda de acordo com o tipo de objeto usado com a tag set;

# A Tag Set

- Caso a tag esteja sendo usada com um atributo, ele é removido do seu respectivo escopo;
- Caso a tag esteja sendo usada com um java bean, o valor null é atribuído à propriedade;

# A Tag Set

- Caso a tag seja usada com um Map, a chave usada com a tag é removida;

# A Tag Remove

- Usada para remover atributos da aplicação;
- Permite remover atributos de qualquer um dos escopos disponíveis;
  - Tem o mesmo efeito que a utilização do método `removeAttribute` dos objetos referentes a estes escopos;

# A Tag Remove

- Atributos da tag remove:
  - `var`:
    - ✓ Atributo obrigatório, que identifica o nome do atributo que deve ser removido;
    - ✓ Importante: o valor deste atributo dever ser sempre uma string literal, não aceitando expressões;

# A Tag Remove

- Atributos da tag remove:
  - scope:
    - ✓ Atributo opcional, que identifica o escopo do objeto que deve ser removido;
    - ✓ Caso o seu valor seja omitido, o container procura no escopo padrão, que é página;

# A Tag Remove

- Exemplo:
  - `<c:remove var="userStatus" scope="session" />`
- Efeito:
  - Ocontainer remove o atributo *userStatus* do escopo *session*;

# A Tag Import

- Usada para incluir o conteúdo de arquivos externos na página;
  - Da mesma forma que a diretiva e a action include de JSP;
- O conteúdo que vai ser adicionado à página pode ser interno ou externo ao container;



# A Tag Import

- O conteúdo a ser incorporado é identificado através do atributo url;
- Nesta tag, o conteúdo é adicionado apenas no momento da solicitação;
- Exemplo:
  - `<c:import url =“imagem.htm” />`

# A Tag Import

- Podemos usar a tag import para passar informações para o conteúdo que vai ser adicionado;
  - Por exemplo, podemos definir valores para os seus parâmetros de entrada;
- Estas informações podem ser passadas por meio da tag **param**;

# A Tag Import

- Vamos considerar a página JSP abaixo, que depende de um parâmetro para a geração do seu conteúdo;

```
  
<br/>  
<em> <strong>${param.subtitle} </strong> </em>
```

# A Tag Import

- Exemplo de como a página pode ser incorporada a outra página JSP usando a tag import:

```
<c:import url ="imagem.jsp" >  
    <c:param  name="subtitle" value="Homer" />  
</c:import>
```

# A Tag Catch

- Usada para a captura de exceções durante a geração da página JSP;
  - Mais ou menos como o bloco try-catch da linguagem java;
- Usamos esta tag para que a página possa se recuperar do lançamento de uma exceção;

# A Tag Catch

- Importante: Caso alguma exceção ocorra dentro do bloco da tag catch:
  - A execução do bloco é finalizada imediatamente;
  - O controle de geração da página avança para o próximo bloco da página;
  - Neste caso, o controle de exceções padrão não é executado;

# A Tag Catch

- Exemplo:

```
<c:catch>
  <%
    int x = 5 / 0;
    int size = 5;
  %>
</c:catch>
<p> Ufa, ocorreu um erro, mas consegui gerar a página! </p>
```

Este trecho não é executado

Este trecho é executado após a exceção

# A Tag Catch

- Com a tag catch, podemos também obter uma referência para a exceção gerada no bloco catch;
- Neste caso, a exceção é armazenada em uma variável cujo nome é definido mediante o atributo **var**;



# A Tag Catch

- Ao usarmos esta opção, o container cria esta variável, que tem o escopo página;
- Importante: O valor desta variável só é acessível fora do bloco da tag catch;

# A Tag Catch

- Exemplo:

```
<c:catch var="myException">
  <%
    String name = null;
    int length = name.length();
  %>
</c:catch>

<c:if test="${myException != null}">
  A página foi gerada com sucesso, mas ocorreu a seguinte exceção: ${myException}
</c:if>
```

# Outras Bibliotecas de Tags

- A JSTL é muito grande, oferecendo uma grande variedade de tags;
- A versão 1.1 é composta por cinco bibliotecas, sendo:
  - Quatro bibliotecas de tags customizadas;
  - Uma série de funções para a manipulação de strings;

# Outras Bibliotecas de Tags

- As bibliotecas de tags customizadas oferecidas pela especificação JSTL são:
  - A biblioteca central;
  - A biblioteca de formatação;
  - A biblioteca XML;
  - A biblioteca SQL;

# A Biblioteca Central

- Exemplos de tags de propósitos gerais:
  - `<c:out>;`
  - `<c:set>;`
  - `<c:remove>;`
  - `<c:catch>;`

# A Biblioteca Central

- Exemplos de tags para iteração:
  - `<c:forEach>;`
  - `<c:forTokens>;`

# A Biblioteca Central

- Exemplos de tags para comandos condicionais:
  - `<c:if>;`
  - `<c:choose>;`
  - `<c:when>;`
    - `<c:otherwise>;`

# A Biblioteca Central

## A biblioteca central

- Exemplos de tags para a manipulação de URLs:
  - `<c:import>;`
  - `<c:url>;`
  - `<c:redirect>;`
  - `<c:param>;`



# A Biblioteca de Formatação

- Exemplos de tags internacionalização:
  - `<fmt:message>;` para questões de
  - `<fmt:setLocale>;`
  - `<fmt:bundle>;`
  - `<fmt:setBundle>;`
  - `<fmt:param>;`
  - `<fmt:requestEncoding>;`

# A Biblioteca de Formatação

- Exemplos de tags para formatação:
  - `<fmt:timeZone>;`
  - `<fmt:setTimeZone>;`
  - `<fmt:formatNumber>;`
  - `<fmt:parseNumber>;`
  - `<fmt:parseDate>;`

# A Biblioteca XML

- Exemplos de tags para ações centrais envolvendo arquivos XML:
  - `<x:parse>;`
  - `<x:out>;`
  - `<x:set>;`

# A Biblioteca XML

- Exemplos de tags para fluxo de controle em arquivos XML:
  - `<x:if>;`
  - `<x:choose>;`
  - `<x:when>;`
  - `<x:otherwise>;`
  - `<x:forEach>;`

# A Biblioteca XML

- Exemplos de tags para transformações em arquivos XML:
  - `<x:transform>;`
  - `<x:param>;`

# A Biblioteca SQL

- Exemplos de tags para o acesso a bancos de dados:
  - `<sql:query>;`
  - `<sql:update>;`
  - `<sql:setDataSource>;`
  - `<sql:param>;`
  - `<sql:dateParam>;`