Programação para Web

Java Server Pages (JSP)

Ivo Calado

Instituto Federal de Educação, Ciência e Tecnologia de Alagoas

22 de Fevereiro de 2016

Roteiro

1. Introdução
2. Sintaxe JSP
3. EL e Taglibs
4. Model View Controller

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA ALAGOAS

Observação sobre o conteúdo

Parte deste material é baseada na apostila FJ-21 - Java para Desenvolvimento Web desenvolvido pela Caelum (`www.caelum.com.br`)

Colocando o HTML no seu devido lugar

- Até agora, vimos que podemos escrever conteúdo dinâmico através de Servlets
- Porém, se toda hora criarmos Servlets para fazermos esse trabalho, teremos muitos problemas na manutenção das nossas páginas e também na legibilidade do nosso código

Porque?

porque sempre aparece código Java misturado com código HTML. Imagine todo um sistema criado com Servlets fazendo a geração do HTML

Colocando o HTML no seu devido lugar

- Em situações onde a presença do código HTML é superior a do código Java precisamos de uma solução que possibilite inverter os papéis!
- Ao invés de inserir código HTML dentro do código Java, inserimos código Java dentro do HTML!
- Algo similar ao ASP e PHP
- Essa tecnologia é o JavaServer Pages (JSP).

Nosso primeiro exemplo

Vejamos o nosso primeiro exemplo em JSP

bem-vindo.jsp

```
<html>
  <body>
    Bem vindo
  </body>
</html>
```

Novidade?

- Fica claro que uma página JSP nada mais é que um arquivo baseado em HTML, com a extensão .jsp
- Podemos escrever também código Java, para que possamos adicionar comportamento dinâmico em nossas páginas, como declaração de variáveis, condicionais (if), loops (for, while)

Nosso primeiro exemplo II

```
<body>
        <%-- Exemplo de comentário--%>
        <% String mensagem = "Meu primeiro exemplo
            dinâmico com JSP"; %>

        <% out.println(mensagem); %><br />

        <% String desenvolvido = "Desenvolvido por
            todos!"; %>

        <strong><%=desenvolvido%></strong><br />
        <%  System.out.println("Tudo foi executado!");
            %>
</body>
```

Porque a mensagem “Tudo foi executado!” não foi exibida na

Sobre o código JSP

- O código Java é adicionado entre os elementos <% e %>
- Existe um objeto out.println que já envia a saída direta para o usuário. Lembra algum outro objeto?
- Chamada a System.out.println irá enviar a saída para a saída padrão do servidor!

Objetos Implícitos

- Em JSP alguns objetos são implícitos

request	HttpServletRequest
response	HttpServletResponse
out	JSPWriter
session	HttpSession
application	ServletContext
config	ServletConfig
pageContext	PageContext
page	HttpJspPage

Código HTML entre o código JSP

- A lógica do código JSP pode ser intercalada com código HTML
- No exemplo abaixo, tudo que estiver entre a definição do loop e o símbolo de fechamento (}) será repetido!

```
<body>
<% for(int i = 0; i < 10; i++) { %>

   <!-- Fora do Código Java -->
   Passou aqui =

<%= i %>
   <br/>
<%} %>
</body>
```

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA ALAGOAS

Diretivas

- São mensagens ao container JSP, contendo informações de como o container JSP deve traduzir as páginas
- A sintaxe básica é utiliza uma estrutura chave valor

```
<%@ diretiva (atributo="valor") *%>
```

- Existem três tipos de diretivas:
  - Diretiva de página: **page**
  - Diretiva de inclusão: **include**
  - Diretiva de taglib: **taglib**

A diretiva *page*

- A diretiva page possibilita a configuração de diversas propriedades da página como *contentType*, buffer, import etc

```
<%@ page atributo="valor"%>
```

- São definidos 11 tipos de atributos
  - language: Nome da linguagem do script (padrão "Java")
  - info: String a ser recuperada via **getServletInfo**
  - contentType: Tipo MIME identificando o tipo do conteúdo e a codificação
  - import: análogo ao **import** numa classe Java
  - session: indica se a página participará do controle de sessão
  - errorPage: pagina de erro a ser utilizada caso alguma exceção seja lançada

A diretiva *include*

- Permite a inclusão de conteúdos de outros arquivos na página JSP
- É útil para inclusão de cabeçalhos e rodapés de página, por exemplo

```
<%@ include file="arquivo.jsp"%>
```

- A localização do arquivo é em relação a JSP. Caso seja colocado um / no começo leva-se em consideração o contexto do servidor. Mais a frente veremos uma maneira mais robusta de trabalhar com localizações de recursos!

A diretiva *taglib*

- Possibilita extender a funcionalidade do JSP
- Iremos abordar ele com mais detalhes a frente... :)

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA ALAGOAS

Definição de código

Existem três formas de definir código em uma página JSP

- Declarações: Permitem a definição de variáveis e métodos
- Expressões: Converte o resultado do código em uma string a ser anexada no código JSP
- Scriptlets: Permitem a escrita de código fora de qualquer método a ser executado (análogo a um código JavaScript)

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA ALAGOAS

Exemplo

Scriptlets

```
<%
while(i-- != 0) {
   out.println("Texto");
}%>
```

Expressões

```
Hoje são <%= getSystemTime() %><br/>
```

Declarações

```
<%!
        int i = 10;
        String getSystemTime() {
                return Calendar.getInstance().getTime()
                    .toString();
```

Exercicio I: Criando uma lista de contatos

- Crie uma classe chamada Contato, dentro de algum pacote específicos
- Adicione 3 atributos: Nome, Endereço e Idade
- Crie uma classe ContatosDao com um métodolo que retorne uma lista de Contatos (pelo menos 5 elementos)
- Com base no exemplo anterior, crie uma página JSP que contenha uma tabela e cada coluna da tabela represente um contato
- Faça uso do **Ctrl + espaço** no Eclipse para importar o que for necessário e veja como é feito o **import**

Até agora conseguimos desacoplar o Java do HTML?

- O objetivo inicial do JSP era redudir o acoplamento entre o código Java e o HTML
- Pelo que vimos até agora isso foi obtido? Porque?
- Não! O código Java continua sendo misturado com HTML e um *designer Web* precisa entender lógica de programação
- E existe hoje em dia no mercado muitas aplicações feitas inteiramente utilizando scriptlets e escrevendo código Java, PHP e outras no meio dos HTMLs

EL: Expression Language

- Para remover um pouco do código Java que fica na página JSP, a Sun desenvolveu uma linguagem chamada Expression Language que é interpretada pelo servlet container
- O acesso a expressões é realizada a partir da estrutura ${expr}
- Vejamos como criar uma página que receba um parâmetro e exiba-o na tela

Exemplo EL

```
<body>
        Testando seus parametros:
        <br /> A idade é ${param.idade}.
</body>
```

- **param** é um objeto nativo, assim como **out** que contem um mapa com parâmetros recebidos na requisição

Exemplo: avaliação de expressões

```
<body>
<h2>Operadores de Comparação</h2>
4 > '3'  => ${4 > '3'}<br/>
'4' > 3  => ${'4' > 3}<br/>
'4' > '3'=> ${'4' > '3'} <br/>
4 >= 3  =>  ${4 >= 3}<br/>
4 <= 3  =>   ${4 < 3}<br/>
4 == '4' =>  ${4 == 4}<br/>
</body>
```

Motivação

- Começamos a melhorar nossos problemas com relação à mistura de código Java com HTML através da Expression Language
- No entanto, ela sozinha não pode nos ajudar muito, pois ela não nos permite, por exemplo, instanciar objetos, fazer verificações condicionais (if else), iterações como em um for e assim por diante
- Sendo assim, precisamos de um mecanismo baseado em tags para representar nosso lógica de negócio!
- Para isso, fazemos uso das Taglibs

JavaBeans: acesso aos atributos

- Antes de entrarmos diretamente em Taglibs precisamos aprender sobre Javabeans. Um Bean é um tipo de objeto Java, com construtor *default* e os respectivos *getters* e *setters*
- Instânciá-los na nossa página JSP não é complicado. Basta utilizarmos a tag correspondente para essa função, que no nosso caso é a $< jsp : useBean >$

```
<jsp:useBean
id="contato" class="br.edu.ifal.pweb.classes.Contato" />
```

Agora podemos fazer uso das expressões EL para acessar o valor de **contato**. Exemplo: ${contato.nome}

JavaBeans

- No nosso exemplo, os atributos de **Contato** foram definidos como privados mas estamos conseguindo acessá-los. Por que isso?
- Na verdade não estamos acessando o atributo privado. Quando fazemos **contato.nome** o container implicitamente chama o método **getNome**

JavaBeans: Modificando propriedades

Para modificar uma propriedade em um bean, fazemos uso da propriedade jsp:setProperty

```
<body>

<%contato.setNome("teste"); %>

<jsp:setProperty property="endereco" value="Rua das
    casas" name="contato" />

Nome = ${contato.nome} <br/>
Endereço = ${contato.endereco}
</body>
```

JavaServer Pages Standard Tag Library

- JSTL é a API que encapsulou em tags simples toda a funcionalidade que diversas páginas Web precisam
- Exemplos de funcionalidades incluem: controle de laços (fors), controle de fluxo do tipo if else, manipulação de dados XML e a internacionalização de sua aplicação
- Existem ainda outras partes da JSTL, por exemplo aquela que acessa banco de dados e permite escrever códigos SQL na nossa página. Mas se o designer não compreende Java o que diremos de SQL? O uso de tal parte da JSTL é desencorajado.

Instalação do JSTL

- Para instalar a implementação mais famosa da JSTL basta baixar a mesma no site `https://jstl.dev.java.net/`
- Adicionar as libs no diretórios libs do projeto
- Para removermosas as notificações de erros no Eclipse, teremos de adicionar as libs no *build path*
- Adicione a referência ao conjunto de taglibs do JSTL (equivalente a um **import**) através da seguinte linha

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

forEach

- A JSTL core disponibiliza uma tag chamada c:forEach capaz de iterar por uma coleção
- No c:forEach, precisamos indicar a coleção na qual vamos iterar, através do atributo items e também como chamará o objeto que será atribuído para cada iteração no atributo var.

```
<c:forEach var="ct" items="${contato.
    lista}" varStatus="id">
        <tr bgcolor="#${id.count % 2 ==
            0 ? 'aaee88' : 'ffffff' }"
          >
            <td>${id.count}</td>
            <td>${ct.nome}</td>
            <td>${ct.idade}</td>
        </tr>
```

Atributos

- **var:** variável a ser utilizada na iteração
- **items:** fonte dos dados
- **varStatus:** posição corrente

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA ALAGOAS

Exercício

Altere o exemplo para que seja a coluna e não a linha da tabela apresentada

Criando Ifs

- E se quisermos alterar a tabela para que seja exibida uma obsercação caso o contato tenha menos que 18 anos
- Fazermos uso da Tag If

```
<c:forEach var="ct" items="${contato.
    lista}" varStatus="id">
        <tr bgcolor="#${id.count % 2 ==
            0 ? 'aaee88' : 'ffffff' }"
        >
            <td>${id.count}</td>
            <td>${ct.nome}</td>
            <td>${ct.idade}</td>
            <td><c:if test="${ct.
                idade < 18 }">
            Menor de Idade</c:if>
            </td>
```

Semântica dos parâmetros

- **test:** Recebe como parâmetro uma expressão a ser avaliada como **true** ou **false**

Além de testes diretos, operadores de EL podem ser utilizados

```
<c:if test="${not empty variavel}">
</c:if>
```

Simulando o else

- O JSTL não possui uma tag **else** para complementar o **if**
- Porém podemos utilizar a estrutura choose para este propósito

```
<c:choose>
        <c:when test="${ct.idade < 18}">
        Menor de idade
        </c:when>
        <c:otherwise>Maior de Idade</c:otherwise>
</c:choose>
```

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA ALAGOAS

Exercicio para casa

Pesquisar sobre os demais operadores do EL

Exercicio

- Adicione um atributo **homepage** à classe Contato.
- No métododo **getLista** de ContatoDao adicione para alguns usuários uma página de usuário
- Altere o exemplo de listagem usando JSTL, adicionando uma nova coluna
- Caso o usuário possua a *homepage* definida exiba na coluna correspondente como um link, caso contrário exiba a mensagem "Não disponível"

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA ALAGOAS

Importando páginas

- Um requisito comum que temos nas aplicaçõesWeb hoje em dia é colocar cabeçalhos e rodapé nas páginas do nosso sistema
- Esses cabeçalhos e rodapés podem ter informações da empresa, do sistema e assim por diante
- O problema é que, na grande maioria das vezes, todas as páginas da nossa aplicação precisam ter esse mesmo cabeçalho e rodapés
- Para isso, podemos fazer uso de inclusões de arquivos para possibilitar a inclusão de outras páginas já existentes

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

Outras tags JSTL existentes

- formatDate: possibilita formatar objetos do tipo Data sob os mais diversos aspectos
- curl: permite a localização simplificada de recursos dentro do projeto
- catch: bloco do tipo try/catch
- c:forTokens: for em tokens (ex: "a,b,c" separados por vírgula)
- c:out: saída
- etc

Servlet ou JSP?

Ao se trabalhar com Servlets e JSPs temos um *tradeoff*

- Se utilizarmos apenas Servlets, o que acontece quando precisamos mudar o design da página? O designer não vai saber Java para editar a Servlet, recompilar e colocá-la no servidor
- Imagine usar apenas JSP. Ficaríamos com muito *scriptlet*, que é muito difícil de dar manutenção. O JSP foi feito apenas para apresentar o resultado, e ele não deveria fazer acessos a banco de dados e nem fazer a instanciação de objetos. Isso deveria estar em código Java, na Servlet

Servlet ou JSP?

Ao se trabalhar com Servlets e JSPs temos um *tradeoff*

- Se utilizarmos apenas Servlets, o que acontece quando precisamos mudar o design da página? O designer não vai saber Java para editar a Servlet, recompilar e colocá-la no servidor
- Imagine usar apenas JSP. Ficaríamos com muito *scriptlet*, que é muito difícil de dar manutenção. O JSP foi feito apenas para apresentar o resultado, e ele não deveria fazer acessos a banco de dados e nem fazer a instanciação de objetos. Isso deveria estar em código Java, na Servlet

Uma ideia mais interessante é usar o que é bom de cada um dos dois!!

Objetivo

- O ideal então é que a Servlet faça o trabalho árduo, a tal da **lógica de negócio**
- E o JSP apenas apresente visualmente os resultados gerados pela Servlet.