JSP
(Java Server Pages)

Instituto Federal de Educação, Ciência e Tecnologia do Triângulo Mineiro
Prof. Edwar Saliba Júnior

Introdução

- Tecnologia para desenvolvimento de aplicações WEB fundamentada na arquitetura SSI (*Server Side Includes*);
- SSI são comandos extensivos a linguagem HTML, os quais podem conter conteúdos estáticos (HTML) e dinâmicos (PHP, ASP, JSP e etc.);
- Os comandos dinâmicos são processados pelo processados pelo servidor WEB antes da página HTML ser enviada ao *browser*.

Características

- Multiplataforma (por ser fundamentada em Java);
- Acesso a banco de dados;
- Manipulação de arquivos;
- Captura de informações por meio de formulários e
- Etc.

J2EE

- Definição:
  - Conjunto de padrões e especificações responsável por receber as requisições do cliente, entendê-las e direcioná-las aos responsáveis pelas ações solicitadas.

Servidor de Aplicação

- Disponibiliza uma API com 9 objetos instanciados que facilitam a programação:
  - request,
  - response,
  - pageContext,
  - session,
  - application,
  - out,
  - config,
  - page e
  - exception.

Objeto *request*

- *request*: solicitação que aciona o processamento da página e busca por valores de parâmetros ou *cookies*;

- Neste exemplo é possível acessar o objeto “gl” em qualquer página que possuir este código:
  - Ex.:

```
GerenciaLogin gl = (GerenciaLogin)
pageContext.getAttribute(“gerenciaLogin”,
PageContext.APPLICATION_SCOPE);
if(gl == null){
   Gl = GerenciaLogin.getInstance();
   pageContext.setAttribute(“gerenciaLogin”,gl,PageCont
   ext.APPLICATION_SCOPE);
}
```

Objeto *response*

- Representa a resposta a ser produzida pela página JSP;
- Objeto muito utilizado, principais métodos no próximo *slide*.

Objeto *response* – Alguns Métodos

Assinatura	Descrição
public void addCookie(Cookie cookie)	Adiciona um cookie à resposta e pode ser utilizado múltiplas vezes para adicionar vários *cookies*.
public void flushBuffer( )	Força qualquer conteúdo do *buffer* a ser escrito para o cliente.
public int getBufferSize( )	Retorna o tamanho atual do *buffer* utilizado para a resposta.
public String getContentType( )	Retorna o tipo de conteúdo da resposta.
public void sendError(int código)	Envia uma resposta de erro para o cliente com o código especificado.
public void sendError(int código, String msg)	Envia uma resposta de erro para o cliente com o código e mensagem especificados.
public void setBufferSize(int tamanho)	Define o tamanho do *buffer* para o corpo da resposta.
public void setContentType(String tipo)	Define o tipo de conteúdo da resposta a ser enviada para o cliente.
public void setHeader(String nome, String valor)	Configura um cabeçalho de resposta com o nome e valor especificados.

Objeto *pageContext*

- Os métodos e atributos deste objeto só são válidos na própria página. Ou seja, só podem ser referenciados nas páginas em que forem declarados.

Objeto *pageContext* – Alguns Métodos

Método	Descrição
public void setAttribute (String nome, Object valor)	Registra um atributo com o nome e o valor especificados no escopo de página.
public void removeAttribute (String nome, int escopo)	Remove o atributo associado ao nome e ao escopo especificados.
public void removeAttribute (String nome)	Remove o atributo associado ao nome especificado de todos os escopos.
public Enumeration getAttributeNamesInScope (int escopo)	Enumera os nomes de todos os atributos contidos no escopo especificado.
public Object getAttribute (String nome, int escopo)	Recupera um atributo associado ao nome e ao escopo especificados.
public Object getAttribute (String nome)	Recupera um atributo associado ao nome especificado no escopo de página.
public void forward (String url)	Encaminha a requisição para outro componente public void forward (String url) da aplicação.
public Object findAttribute (String nome)	Busca por um atributo nos escopos de página, requisição, sessão e aplicação.
public void setAttribute (String nome, Object valor, int escopo)	Registra um atributo com o nome, o valor e no escopo especificados.

Objeto *pageContext* – Algumas Constantes

Constante	Descrição
public static final int PAGE_SCOPE	Representa o escopo de página.
public static final int REQUEST_SCOPE	Representa o escopo de requisição.
public static final int SESSION_SCOPE	Representa o escopo de sessão.
public static final int APPLICATION_SCOPE	Representa o escopo de aplicação.

Objeto *session*

- Possibilita, dentre outras coisas, a identificação de usuários em diversas páginas de um sistema.

Objeto *session* – Alguns Métodos

Método	Descrição
public Object getAttribute(String nome)	Recupera o atributo especificado do escopo de sessão.
public Enumeration getAttibuteNames ()	Retorna os nomes de todos os atributos armazenados no escopo de sessão.
public long getCreationTime ()	Retorna a data e a hora em que a sessão foi criada em milisegundos.
public String getId ()	Retorna o identificador único atribuído à sessão.
public int getMaxInactiveInterval ()	Recupera o tempo pelo qual a sessão é mantida pelo contêiner entre dois acessos.
public void invalidade()	Invalida a sessão, removendo todos os seus atributos.
public boolean isNew()	Retorna *true* se o cliente ainda não estiver utilizando a sessão.
public void removeAttribute (String nome)	Remove o atributo especificado do escopo de sessão.
public void setAttribute(String nome, Object valor)	Grava um novo atributo na sessão com o nome e valor especificados.
public void setMaxInactiveInterval (int intervalo)	Especifica o tempo que a sessão deve ser mantida ativa pelo contêiner entre dois acessos.

Objeto *application*

- Representa a aplicação.

Objeto *application* – Alguns Métodos

Método	Descrição
public Object getAttribute(String nome)	Recupera o atributo especificado do escopo de aplicação.
public Enumeration getAttibuteNames ( )	Retorna os nomes de todos os atributos armazenados no escopo de aplicação.
public String getInitParameter(String nome)	Retorna o parâmetro de inicialização especificado ou *null* se não for encontrado.
public Enumeration getInitParamterNames ( )	Retorna os nomes de todos os parâmetros de inicialização.
public String getRealPath(String caminho)	Retorna o caminho real relativo do caminho virtual especificado.
public RequestDispatcher getRequestDispatcher(String url)	Recupera um objeto que pode ser utilizado para encaminhar a solicitação para a URL local indicada.
public void log(String msg)	Escreve a mensagem no arquivo de log de um servlet.
public void log(String msg, Throwable erro)	Escreve uma mensagem descritiva e a pilha de erro de uma exceção especificada.
public void removeAttribute (String nome)	Remove o atributo especificado do escopo de aplicação.
public void setAttribute(String nome, Object valor)	Grava um novo atributo no escopo de aplicação.

Objeto *out*

- Utilizado dentro de *scriptlets*
- Empregado para produzir saídas dinâmicas:
  - Ex.: `out.println("Um texto qualquer.");`

Objeto *config*

- Utilizado para obter informações de inicialização da página JSP;
- Objeto pouco utilizado.

Objeto *page*

- Representa a própria página JSP:
  - *page import* – importa pacotes de classe:
    - Ex: `<%@ page import="java.util.List" %>`
  - *page language* – informa a linguagem utilizada:
    - Ex: `<%@ page language="java" %>`

Objeto *exception*

- Utilizado no tratamento de exceções que poderão ocorrer nas páginas;
- Só está disponível em páginas de erro, ou seja, páginas JSP que definam o atributo isErrorPage da diretiva *page* como *true*.

Scriptlets

- Trechos de códigos em JSP embutidos no HTML que são interpretados pelo servidor;
- Não confunda com JavaScript, que interpretado na máquina cliente.

Tags JSP

- Expressão:

  `<%= %>` o resultado retornado é uma *string*;

  – Exemplo:

  - Retorna o endereço IP do cliente:

    ```
    <%= request.getRemoteAddr() %>
    ```

  - Soma simples:

    `<%= 2+2 %>` (retorna 4).

- Sintaxe:

  `<% %>` declaração livre ou scriptlets. Basicamente serve para mesclarmos código Java dentro do código HTML;

  – Exemplo:

```
<%
for(int i = 0; i < 10; i++){
   out.println("Número: " + i);
}
%>
```

Tags JSP

- Comentário:

  `<%--  --%>` utilizada pelo desenvolvedor para documentar determinados trechos de código;

  – Exemplo:

```
<%-- Isto é um comentário. --%>
```

- *Standard Actions*:

  `<jsp:  />` São *tags* associadas as *tags* HTML, que modificam o comportamento das páginas JSP;

  – Exemplo:

```
<jsp:useBean id='idDoBean' class='pacote.nomeDaClasse'>
   ...
</jsp:useBean>
```

Diretivas

- `page import` = Importa pacotes de classes:

```
<%@page import=”fib.*” %>
<%@page import=”java.util.List” %>
```

- `page language` = Informa a linguagem utilizada:

```
<%@page language=”java” %>
```

- `taglib` = habilita uma biblioteca de *tags* personalizada:

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

Sessão

- Mecanismo utilizado para prover o controle de usuários dentro de uma aplicação *web*;

- O protocolo HTTP não armazena informações de estado, sendo necessário o tratamento destes dados como “sessão de usuário”.

Sessão

- Exemplo - uma aplicação *web* que necessite de autenticação:
  - é necessário que a autenticação seja solicitada numa página de *login*;
  - alguns recursos (páginas) do *site* só deverão estar disponíveis após a autenticação do usuário e
  - somente o usuário autenticado com o mesmo *browser* é que poderá ter uma referência para o objeto da sessão criada.

Criando uma Sessão

- Para gravar um atributo de usuário:

```
void session.setAttribute(String name,
Object value);
```

- Exemplo:

```
sesseion.setAttribute("login","ADS");
```

- Para recuperar o atributo de um usuário:

```
Object session.getAttribute(String
name);
```

- Exemplo:

```
valor = session.getAttribute("login");
```

Objeto *request*

- Solicitação que aciona o processamento da página e busca por valores de parâmetros ou *cookies*;

- Sintaxe:

```
request.getParameter(String name);
```

Passando Parâmetros

- Página index.jsp

```
<% …
  <form name="f1" method="post" action="verifica.jsp">
   <p>
     <label for="usuario">Usuário:</label>
     <input name="usuario" type= "text" /><br />
     <label for="senha">Senha:</label>
     <input name="senha" type= "password" />
     <br />
     <input name="submit" type="submit" value=
"Verificar" />
   </p>
  </form>
%>
```

Recebendo Parâmetros

- Página verifica.jsp

```
<%
  ...
  String usuario, senha;
  usuario = request.getParameter("usuario");
  senha = request.getParameter("senha");
  ...
  if(...){
    out.println("Usuário: " + usuario + " está ok!");
  } else {
    out.println("Usuário e/ou senha incorretos.");
  }
%>
```

Ambiente de Desenvolvimento

- Para desenvolvermos uma aplicação JSP é necessário que tenhamos instalado no computador:
  - a JDK (*Java Development Kit - http://www.oracle.com/technetwork/java/javase/downloads/index-jsp-138363.html#javasejdk*) do Java e
  - o Tomcat (http://tomcat.apache.org) que será nosso servidor de aplicação e tratará tudo o que for referente a Java (com exceção do JavaScript) no lado servidor.

Ambiente de Desenvolvimento

- Para facilitar sua vida, vá ao *site* https://netbeans.org/downloads/ e faça o *download* do NetBeans, última versão completa;

- Ao instalar o NetBeans no computador lembre-se de marcar a opção de instalação do “Tomcat”, pois, a Oracle possui o servidor de aplicação conhecido por “Glass Fish” e é ele que vem na instalação padrão.

Após a Instalação

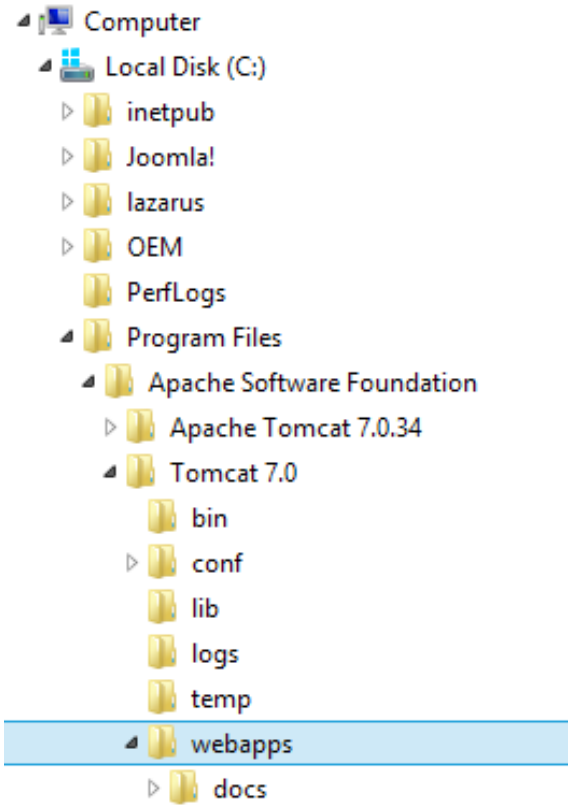
- Como verificar se o Tomcat está funcionando?
  - Abra um *browser* e digite a seguinte url:

    `localhost:8080`
  - deverá aparecer a página de administração do Tomcat.
- Para acessar o Tomcat é necessário usuário (padrão: admin) e senha (cadastrada durante a instalação).

Estrutura de Pastas

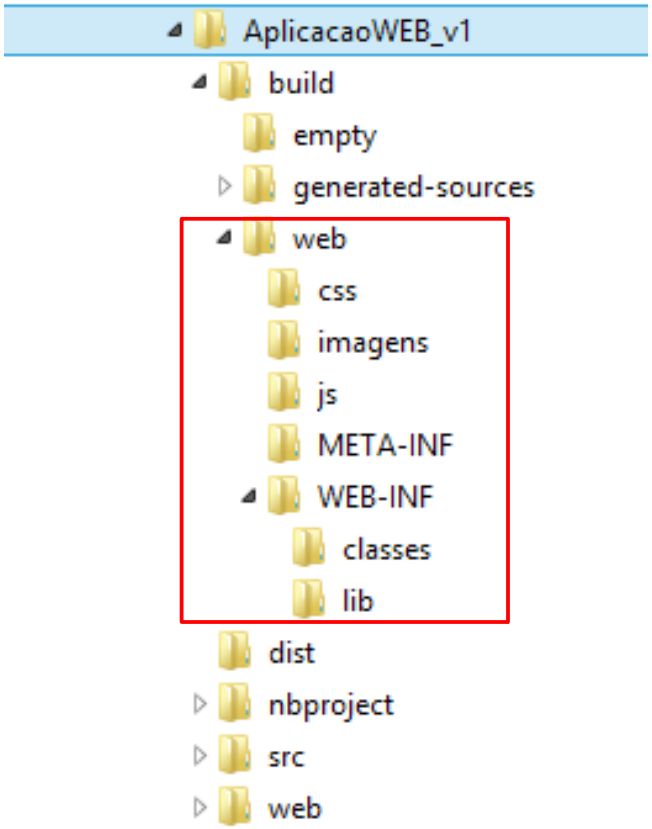
- No NetBeans você poderá colocar seus arquivos referentes a uma determinada aplicação no caminho (sequência de pastas) que você quiser;

- Já para o Tomcat, para que ele reconheça sua aplicação ela deverá ser colocada na pasta *webapps* do diretório de instalação do próprio Tomcat.



Estrutura de Pastas

- Sua aplicação deverá possuir uma estrutura de pastas padrão (figura ao lado);
- Onde, após você construir (dar um *Build*) o projeto:
  - Dentro da pasta *build* haverão as seguintes pastas:
    - **web** – onde estarão os arquivos html, jsp e imagens;
    - **META-INF** – onde estará o arquivo "context.xml"
    - **WEB-INF** – onde estarão as classes, pacotes e bibliotecas utilizadas na aplicação.



JSP

- Vamos colocar a mão na massa?

Primeiro Arquivo JSP

- Vá ao NetBeans e crie um novo projeto web. Para isto escolha:
  - Arquivo | Novo Projeto...
  - Categoria: “Java Web”,
  - Projeto: “Web Application”,
  - Servidor: “Tomcat”,
  - Versão do Java EE: “a mais recente” e
  - Dê um nome para seu projeto (aqui será “Projeto01”) e escolha um local para salvá-lo. Não adicione nenhuma biblioteca e finalize a criação do projeto.

Projeto01

- Feitos os passos do *slide* anterior já teremos um projeto de aplicação web prontinho e funcionando;
- index.jsp
- Aperte F6 para testar.

Comentário

- No projeto anterior não escrevemos nenhum código JSP;
- Mas a extensão do arquivo “index” gerado pelo NetBeans era JSP;
- Saiba que todo conteúdo que você escrever utilizando JSP será convertido para HTML para que o *browser* possa interpretá-lo e imprimi-lo.

Projeto02

- Crie um novo projeto com os mesmos passos do projeto anterior;

- No corpo `<body>` da página criada, apague a *tag* `<h1>` e todo seu conteúdo. No seu lugar coloque as seguintes linhas de comando:

```
<%
  String saudacao = "Hello world!";
  out.println(saudacao);
%>
```

- Execute o projeto e constate se funciona ou não;

- index.jsp

Projeto03

- Podemos imprimir o valor de uma variável JSP de outra maneira;
- Podemos também fazer comentários em páginas JSP como é feito em qualquer linguagem de programação;
- Podemos também utilizar a linguagem Java;
- Vejamos em: index.jsp
- Crie um projeto no NetBeans com o código do Projeto03 e execute-o;
- Tudo funcionou?

Projeto04

- Agora um pequeno projeto utilizando algumas estruturas de “programação estruturada”;

- Vejamos em: index.jsp

Projeto05

- Agora um pequeno projeto verificando se o acesso a um banco de dados está funcionando;

- Vejamos em: index.jsp

Projeto06

- Acessando banco de dados;

- Utilizando DAO e outras classes;

- index.jsp
- Access.java
- Database.java
- AlunoDAO.java
- Aluno.java

Projeto07

- Um sistema de “Lista de Contatos” quase completo em JSP:
  - Cadastro,
  - Alteração,
  - Exclusão,
  - Listagem de Dados,
  - SGDB,
  - DAO,
  - Arquivo JSPF (*Java Server Page File*),
  - CSS,
  - JavaScript e
  - Autenticação de Usuário.
- Arquivos (próximo *slide*).

Projeto07

- Arquivos:
  - verificaSessao.jspf
  - index.jsp
  - principal.jsp
  - contatoLista.jsp
  - contatoCadastro.jsp
  - contatoAlteracao.jsp
  - Access.java
  - Database.java
  - Contato.java
  - ContatoDAO.java
  - Projeto07.css

Bibliografia

- ANSELMO, Fernando. **Tudo sobre a JSP com o NetBeans em Aplicações Distribuídas**. Florianópolis: Visual Books, 2005.

- Caelum. Apostila Fj21. Disponível em: <http://www.cin.ufpe.br/~bnm/courses/caelumf21.pdf> Acesso em: 15 fev. 2013.

- TutorialsPoint. JSP. Disponível em: <http://www.tutorialspoint.com/jsp/index.htm> Acesso em: 25 mar. 2013.