Servlets, JSPs e Java Beans

Prof. Pasteur Ottoni de Miranda Junior – PUC Minas
Disponível em www.pasteurjr.blogspot.com

1-Servlets

1.1 – O que são

Servlets são classes Java executadas em servidores web. Servem para processar regras de negócio em uma camada ligada ao servidor, fora da máquina do cliente. Podem, por exemplo, executar as regras de negócio ou apenas mostrar uma página HTML dinamicamente. São para os servidores o correspondente aos applets para os browsers, mas não têm interface gráfica com o usuário.

1.2- Um exemplo simples

A classe a seguir define completamente um servlet:

```
    public class ExemploServlet extends HttpServlet
    {
     /* Esta classe exemplifica a estrutura básica de um servlet  */


        public void doGet (HttpServletRequest request,
                           HttpServletResponse response)
        throws ServletException, IOException
/* O metodo doGet recebe as requisições do cliente */
        {

   PrintWrite out;// Responsável por escrever conteúdo na        //saída
via método println() visto abaixo
   String parametro= request.getParameter(“parametro”);

          /*
os comandos a seguir definem o conteúdo a ser exibido
*/

            response.setContentType("text/html");
            out = response.getWriter();

            out.println("<HTML><HEAD><TITLE>");
            out.println();
            out.println("</TITLE></HEAD><BODY>");
            out.println("<H1>" + parametro + "</H1>");
            out.println("<P>Esta é a saída do servlet de
ExemploServlet");
            out.println("</BODY></HTML>");
            out.close();
        }
```

```
    }
```

-`ExemploServlet` estende a classe `HttpServlet`, a qual implementa a interface Servlet.

-`ExemploServlet` sobrecarrega o método `doGet` da classe `HttpServlet`. Ele é chamado quando um cliente realiza uma requisição GET (método de requisição default do HTTP) e resulta em uma página simples HTML retornada ao cliente.

-No método `doGet`:

-Requisições do usuário são tratadas pelo objeto `request` da classe `HttpServletRequest`. Através de seus métodos capturam-se, por exemplo, parâmetros passadas pelo usuário(método `getParameter("nome parâmetro")`.

-A resposta ao usuário é tratada pelo um objeto `response` da classe HttpServletResponse.

-Como texto é retornado ao cliente, a resposta é enviada usando o objeto `PrintWrite` obtido a partir do objeto `HttpServletResponse`.

1.3 –Como chamar um servlet em uma página

O servlet acima poderia ser invocado a partir de um link ou de um formulário. Via um link, a forma seria assim, em HTML:

```
<a href="/Zona/ExemploServlet">Texto do link</a>
```

`Zona` corresponde a um endereço ou diretório disponibilizado pelo servidor web (no Apache esta zona está ligada a um diretório onde está o servlet).

Caso haja parâmetros a serem enviados ao servlet, a chamada via link é feita assim:

```
<a href="/Zona/ExemploServlet?parametro=partst&outroparametro=xxx">Texto
do link</a>
```

Repare que o primeiro parâmetro aparece logo após a interrogação. O & é o separador dos diversos parâmetros, caso o servlet possua mais de um.

Quando um servlet for invocado por um formulário, o código HTML deste seria:

```
<form name='form1' action='/Zona/ExemploServlet' method='get' >
```

```
<input name='title' type='text' value='’ size="40" maxlength='60'>

<input type='submit' name='btnsalvar' value='Gravar'>

</form>
```

Quando o botão btnsalvar do código acima for pressionado, o servlet ExemploServlet será automaticamente invocado no servidor e executado. Repare que o método do formulário é get, que corresponde ao método doGet da classe ExemploServlet. Este método é executado e produz uma saída na forma de uma página HTML, através do código abaixo:

```
  PrintWriter        out;
  String title= request.getParameter(“parametro”);
response.setContentType("text/html");
out = response.getWriter();

out.println("<HTML><HEAD><TITLE>");
  out.println(parametro);
  out.println("</TITLE></HEAD><BODY>");
  out.println("<H1>" + parametro + "</H1>");
  out.println("<P>Esta é a saída do ExemploServlet.");
  out.println("</BODY></HTML>");
  out.close();
```

1.4- Como obter valores de parâmetros de formulários via servlets

No exemplo acima, o formulário possui um campo de entrada de dados denominado title, a ser preenchido pelo usuário e submetido ao servidor pressionado-se o botão submit. Este campo é denominado *parâmetro do servlet* e pode ser recuperado via objeto `HttpServletRequest`, que no exemplo chama-se `request`. Este objeto possui um método chamado `getParameter(“nome do parâmetro”)`, capaz de recuperar os valores submetidos no formulário, no exemplo, o título digitado pelo usuário, armazenando-o na variável title:

```
String parametro= request.getParameter(“parametro”);
```

2-Introdução às Java Server Pages (JSP)

2.1-O que são

JSP (Java Server Pages) é uma tecnologia para desenvolvimento de aplicações WEB que permite a inserção de código Java dentro de páginas JSP. Esse código Java, denominado *scriplet,* é executado em um servidor *web* (Tomcat, ou Jboss ,

Glassfish etc), convertido em um *servlet,* processado e o resultado é retornado ao cliente como uma página HTML a ser exibida no *browser.*

2.2-Primeiro programa JSP

JSP permite a inserção de código Java dentro de páginas HTML. Um arquivo HTML pode se transformar em um JSP apenas pela mudança da extensão. O código abaixo, por exemplo, se gravado em arquivo com extensão .jsp vai funcionar como tal, sem alterações.

```
<HTML>
<BODY>
Sou um HTML gravado como JSP!
</BODY>
</HTML>
```

2.3-Adicionando código Java para exibir conteúdo

Seja o código abaixo

```
<HTML>
<BODY>
A hora e data agora são <%= new java.util.Date() %>
</BODY>
</HTML>
```

As sequências de caracteres <%= e %> devem conter expressões Java e são executados em nível de servidor. O servidor transforma esse código em um método de um servlet, que o executa e gera a página HTML , enviando-a ao browser.

2.4-Inserindo expressões em Java - Scriptlets

JSP permite escrever blocos de Java dentro do código JSP, dentro dos chamados *sciplets.* Isto é feito colocando-se código entre **<%** e **%>** O código Java do scriplet é executado no servidor a cada vez que a página JSP é invocada.

O exemplo anterior, adicionando-se um scriplet, fica assim:

```
<HTML>
<BODY>
<%

    System.out.println( "Vou obter a data e a hora" );
    java.util.Date date = new java.util.Date();
%>
A hora agora é <%= date %>
</BODY>
</HTML>
```

O objeto **out** que é do tipo javax.servlet.jsp.JspWriter, o qual vimos no item sobre servlets é pré-declarado quando usamos JSP. Veja abaixo o uso do mesmo.

```
<HTML>
<BODY>
<%
        java.util.Date date = new java.util.Date();
%>
A hora agora é
<%
       out.println( String.valueOf( date ));
%>
</BODY>
</HTML>
```

2.5-Scriptlets e HTML

Suponha que você tenha que gerar uma tabela em HTML contendo números de 1 a N. O código JSP para isso seria:

```
<TABLE BORDER=2>
<%
    for ( int i = 0; i < n; i++ ) {
        %>
        <TR>
        <TD>Number</TD>
        <TD><%= i+1 %></TD>
        </TR>
        <%
    }
%>
</TABLE>
```

Repare nos símbolos <% e %> no meio do loop "for", que tem o objetivo de inserir código Java no meio do código HTML.

No trecho de código abaixo temos um condicional if que determina qual saída será impressa:

```
<%
    if ( opcao1 ) {
        %>
        <P>Opção 1
        <%
    } else {
        %>
        <P>Opção 2
        <%
    }
%>
```

2.6-Diretivas

Seja o código abaixo:

```
<%@ page import="java.util.*" %>
<HTML>
<BODY>
<%
    System.out.println( "Vou obter a data e a hora" );
    Date date = new Date();
%>
A hora agora é <%= date %>
</BODY>
</HTML>
```

Na primeira linha do código temos o uso de uma diretiva JSP. Diretivas começam com os caracteres <%@ e terminam com %>. A do exemplo é uma diretiva "page". Ela pode conter a lista de todas as packages importadas. Para importar mais de um item, os nomes das packages devem ser separados por vírgulas, como em

```
<%@ page import="java.util.*,java.text.*" %>
```

Outra diretiva importante é a include, que inclui o conteúdo de outro arquivo. O arquivo incluído pode ser HTML, JSP ou qualquer outro.

```
<HTML>
<BODY>
Incluindo arquivo teste.jsp...<BR>
<%@ include file="teste.jsp" %>
</BODY>
</HTML>
```

2.7-Declarações de objetos e variáveis

Para adicionar uma declaração, deve-se colocar quaisquer declarações entre **<%!** e **%>** como mostrado abaixo:

```
<%@ page import="java.util.*" %>
<HTML>
<BODY>
<%!
    Date data = new Date();
    Date pegaData()
    {
        System.out.println( "Em pegaData()" );
        return data;
    }
%>
A hora agora é <%= pegaData() %>
</BODY>
</HTML>
```

Nesse exemplos declaramos o objeto data e o método pegaData()

2.8 Tags JSP pré-definidas

Tags pré-definidas iniciam-se com os caracteres **jsp:**. Por exemplo, jsp:include é uma tag pré-definida que é usada para incluir outras páginas. Esta tag é como a include já vista, porém inclui a página no momento da execução:

```
<HTML>
<BODY>
Incluindo teste.jsp...<BR>
<jsp:include page="teste.jsp"/>
</BODY>
</HTML>
```

2.9-Sessions

Sessions (sessões) são objetos associados a cada visitante de uma página JSP toda vez que ela é invocada. Dados podem ser colocados em cada sessão e recuperados. Um conjunto diferente de dados é mantido para cada visitante da página.

No exemplo abaixo temos um conjunto de páginas que coloca o nome do usuário na session e o exibe.

```
<HTML>
<BODY>
<FORM METHOD=POST ACTION="SaveName.jsp">
Qual seu nome?? <INPUT TYPE=TEXT NAME=username SIZE=20>
<P><INPUT TYPE=SUBMIT>
</FORM>
</BODY>
</HTML>
```

Esse arquivo apenas exibe o campo texto username para que o usuário digite seu nome e chama o arquivo SaveName.jsp , exibido abaixo, que salva o nome digitado em um *attributo* da session, que pode conter variáveis ou objetos. O atributo é criado pelo método session.setAttribute (nomeatributo, valoratributo).

```
<%
   String name = request.getParameter( "username" );
 session.setAttribute( "theName", name );
%>
<HTML>
<BODY>
<A HREF="NextPage.jsp">Continue</A>
</BODY>
</HTML>
```

O arquivo NextPage.jsp exibido abaixo mostra como recuperar o nome:

```
<HTML>
<BODY>
Hello, <%= session.getAttribute( "theName" ) %>
</BODY>
</HTML>
```

A session é mantida por um determinado período de tempo, ajustado na configuração do servidor web. Quando este tempo expira, assume-se que o visitante abandonou o site e a sessão é terminada.

3-Introdução aos Java Beans

Um Java Bean é um componente reutilizável de software escrito em Java. São utilizados principalmente por *containers bean* que são uma aplicação ou ambiente de programação. Na essência, são uma classe que tem atributos e métodos de armazenamento e recuperação dos mesmos.

No exemplo abaixo o arquivo GetName.html obtém o endereço de email e a idade de um usuário:

```
<HTML><BODY>
<FORM METHOD=POST ACTION="SaveName.jsp">
Qual é seu nome? <INPUT TYPE=TEXT NAME=username SIZE=20><BR>
Qual é seu email? <INPUT TYPE=TEXT NAME=email SIZE=20><BR>
Qual é sua idade? <INPUT TYPE=TEXT NAME=age SIZE=4>
<P><INPUT TYPE=SUBMIT>
</FORM>
</BODY>
</HTML>
```

Esses dados podem ser facilmente recuperados se definirmos uma classe Java Bean com os campos "username", "email" e "age" e escrevemos os chamados métodos *setter* "setUsername", "setEmail" e "setAge", que armazenam nos campos os valores aos quais se referem. Um método "setter" começa com "set" seguido pelo nome do campo. O primeiro caracter do nome do campo deve ser colocado em maiúsculas. Portanto, se o campo for "email", seu método "setter" será "setEmail". Existem também os métodos denominados *getters*, definidos de maneira similar, com "get" ao invés de "set", capazes de recuperar o valor dos campos. Observe que métodos setters (e getters) devem ser públicos e os campos, privados.

```
public class UserData {

    private String username;
    private  String email;
    private int age;
```

```
    public void setUsername( String value )
    {
        username = value;
    }

    public void setEmail( String value )
    {
        email = value;
    }

    public void setAge( int value )
    {
        age = value;
    }

    public String getUsername() { return username; }

    public String getEmail() { return email; }

    public int getAge() { return age; }
}
```

Os nomes dos métodos devem ser exatamente como mostrado.

Vamos modificar o arquivo "SaveName.jsp" visto anteriormente de forma que utilize um bean para recuperar os dados digitados pelo usuário:

```
<jsp:useBean id="user" class="UserData" scope="session"/>
<jsp:setProperty name="user" property="*"/>
<HTML>
<BODY>
<A HREF="NextPage.jsp">Continue</A>
</BODY>
</HTML>
```

Observe que tudo que precisamos fazer é adicionar as tags jsp:useBean e jsp:setProperty. A tag useBean declara uma instância de uma classe Java Bean, no exemplo, a instância "user" e a classe "UserData" A tag setProperty vai recuperar os dados de entrada automaticamente. O arquivo NextPage.jsp mostrado a seguir exibe os dados recuperados:

```
<jsp:useBean id="user" class="UserData" scope="session"/>
<HTML>
<BODY>
```

```
Você entrou<BR>
Nome: <%= user.getUsername() %><BR>
Email: <%= user.getEmail() %><BR>
Idade: <%= user.getAge() %><BR>
</BODY>
</HTML>
```