VOLTAR

Servlets, JSP e Servidores de conteúdo

Apresentar as tecnologias que serão utilizadas na disciplina, nas aplicações desenvolvidas no lado servidor: discutir a tecnologia Java para sites dinâmicos (servlet e JSP), a arquitetura, o ciclo de vida de servlets e os principais softwares servidores de conteúdo Java disponíveis.

NESTE TÓPICO

Soluções Java para sites dinâmicos

Falar em soluções Java para sites dinâmicos nos remete à primeira aula, qual foram citadas algumas tecnologias e linguagens de programaçã dentre elas, as específicas da plataforma Java.

Como plataforma para o ambiente de desenvolvimento, entre outras, ter as seguintes tecnologias:

- **Java EE** (ou JEE, ou J2EE), que fornece subsídios para o desenvolvimento sites dinâmicos.
- **Java ME** (ou J2ME), para aplicativos móveis, PDAs (*Personal Digital Assistant* – assistente digital pessoal), etc.
- **JavaFX**, que traz uma interface mais amigável e dinâmica.
- **JavaTV**, para desenvolvimento para TVs.

Destas plataformas, destacamos o Java EE (JEE ou, ainda, J2EE em versões anteriores), que será nossa plataforma base para o desenvolvimento de aplicações e sites dinâmicos.

Dentro dessa plataforma, utilizaremos as tecnologias servlet e JSP com conexão e manutenção a banco de dados, além da utilização essencial de HTML, CSS e Javascript a ser executada no lado cliente.

Ainda discutiremos alguns conceitos importantes, entre eles, o conceito de desenvolvimento de sistema em camadas, técnica muito utilizada no desenvolvimento de sistemas robustos para as empresas, que permite flexibilidade para integração de sites com sistemas empresariais.

Servlets e JSP

É hora de falarmos especificamente sobre servlet e JSP. Com essas duas tecnologias, é possível flexibilidade total no desenvolvimento de sites dinâmicos.

Servlets

Grande parte da programação em Java para web pode ser feita tanto com servlets quanto com JSP. Porém, se as duas tecnologias existem, é porque existe também uma razão para isso.

Segundo Santos (2007, p.76), com o JSP desenvolvemos páginas em HTML, com trechos de códigos Java sempre que se fizer necessário, enquanto os servlets fazem o contrário: são programas em Java com trechos de códigos em HTML.

Vamos entender o seguinte quanto aos servlets:

> Osservlets são pequenos programas escritos em Java que rodam dentro de um servidor web. Eles são executados em um contêiner, que oferece a infra-estrutura [sic] de software necessária, e atendem a requisições de aplicações cliente para prestar-lhes algum tipo de serviço.
>
> ((SANTOS, 2007, p.34))

O servlet é uma classe Java a ser compilada em sua primeira execução no servidor. No site do Tomcat (2012), está definido que um servlet é um pequeno programa em Java executado dentro de um servidor web. Os servlets recebem e respondem aos pedidos dos clientes da web, mais comumente através do protocolo HTTP, que é o protocolo de transferência de dados hipertexto.

JSP

O JavaServer Page (JSP) é um arquivo HTML com instruções Java para executar comandos no lado servidor e possibilitar o desenvolvimento de sites dinâmicos. Outras linguagens também fazem este papel, no entanto, uma das vantagens do JSP é que por se tratar da linguagem Java, é compilado no servidor em sua primeira execução. Para que isso seja possível, o servidor web recebe o arquivo JSP e o transforma em um servlet.

> tecnologia JavaServer Pages (JSP) permite a criação de páginas web para exibir conteúdo estático ou dinâmico na tela do navegador. São baseados em texto e suportam elementos da linguagem HTML, XHTML, CSS, JavaScript, WML, etc. para exibir conteúdo estático e instruções da plataforma Java para construir conteúdo dinâmico.
>
> ((TERUEL, 2009, p.97))

Lembrando que *Wireless Markup Language* (WML) é uma linguagem com base em XML para programação em dispositivos móveis.

Em breve, discutiremos detalhadamente como desenvolver páginas estáticas e dinâmicas utilizando as tecnologias JSP e servlet, mas nunca ignorando a tecnologia de conteúdo estático, que não necessita de nenhum processamento lógico do lado servidor da aplicação.

Arquitetura e ciclo de vida de servlets

Os servlets, como já citado, são pequenos programas executados no servidor web, são denominados servidores de conteúdo. São exemplos: JRun, WebLogic, JBoss, WebSphere, Tomcat e GlassFish (entre outros).

Em nossos exemplos, utilizaremos o GlassFish por ser, talvez, o mais utilizado e indicado pela Oracle.

O servidor de conteúdo web será responsável pelo controle de compilação e execução dos servlets. Ao receber a chamada a um servlet pela primeira vez, ele o compilará e, a partir daí, passará a executar o arquivo compilado no servidor.

O ciclo de vida de um servlet tem início quando é inicializado. Enquanto a aplicação ou o servidor não for finalizado, continuará ativo e compartilhando variáveis, inclusive com a possibilidade de manter conexão com o banco de dados durante toda a aplicação. Sendo assim, o servlet só deixa de existir quando a aplicação é finalizada.

Santos (2007, p.36) afirma que mesmo "depois de ter sido criada e carregada no contêiner, a nova instância do servlet permanecerá disponível mesmo depois de ter sido respondido à requisição que motivou sua criação", isso faz com que os recursos do servidor sejam economizados, não necessitando de uma nova instância a cada solicitação do mesmo serviço (servlet).

Ao ser acionado, o servlet executa o método **init**(), que inicia o seu ciclo de vida e que ocorre somente se o objeto ainda não estiver na memória. Durante as requisições **Get** e **Post**, que acionam os métodos doGet() e doPost() do servlet, ocorre a execução do método **service**() e no fechamento da aplicação ou do servidor web ocorre a execução do método **destroy**().

Exemplo de servlet

Agora, para melhor entender o conceito, vejamos o código criado automaticamente pelo NetBeans ao criarmos um servlet:

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class MeuServlet extends HttpServlet {

protected void processRequest(HttpServletRequest request,
                  HttpServletResponse response)
            throws ServletException, IOException {

response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet MeuServlet</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>Servlet MeuServlet at "
                              + request.getContextPath() + "</h1>");
            out.println("</body>");
            out.println("</html>");
        } finally {
            out.close();
        }
    }

@Override
    protected void doGet(HttpServletRequest  request,
                         HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

@Override
    protected void doPost(HttpServletRequest  request,
                          HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

@Override
    public String getServletInfo() {
        return "Short description";
    }
}
```

Todos servlet criado herda a classe HttpServlet e tem, obrigatoriamente, os métodos **doGet**() e **doPost**(). O método doGet() será executado quando as informações vierem do HTML com uma chamada Get. E o método doPost() será executado sempre que as informações vierem do HTML através de uma chamada Post.

A chamada Get ou Post no HTML é feita através do atributo **method** da TAG <**form**>. Exemplo: <**form** action="search.jsp" **method**="**post**" />

O NetBeans cria automaticamente o método **processRequest**(), que é acionado pelos métodos doGet() e doPost(), pois para o servlet não importa se os dados foram enviados via Get ou Post do HTML.

Vamos a um exemplo prático. Teremos um arquivo HTML enviando uma informação de um formulário que será recebido pelo servlet e devolvido ao cliente como HTML.

Trecho HTML para enviar os dados de um formulário ao servidor:

```
<form action="MeuServlet" method="post">
            <pre>
            Primeiro Numero: <input type ="text" name="numero1" size="10" />
            Segundo  Numero: <input type ="text" name="numero2" size="10" />
            <input type="submit" value="Enviar" />
            </pre>
</form>
```

Agora o trecho em Java escrito no servlet. Será transcrito somente o trecho alterado dentro do servlet. O restante continua como no exemplo de servlet já visto.

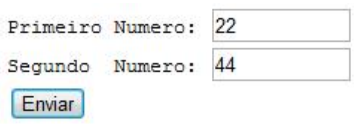
```
try {
    int n1 = Integer.parseInt(request.getParameter("numero1"));
    int n2 = Integer.parseInt(request.getParameter("numero2"));
    int s = n1 + n2;
    out.println("<html>");
    out.println("<head>");
    out.println("<title>Servlet MeuServlet</title>");
    out.println("</head>");
    out.println("<body>");
    out.println("<h1>Servlet MeuServlet at "
                        + request.getContextPath() + "</h1>");
    out.println("<p>Num 1: " + n1 + "</p>");
    out.println("<p>Num 2: " + n2 + "</p>");
    out.println("<p>Soma.: " + s + "</p>");
    out.println("</body>");
    out.println("</html>");
} finally {
    out.close();
}
```

Entendendo o código

O trecho de programa armazena nas variáveis **n1** e **n2**, respectivamente, os valores recebidos do formulário que o chamou, convertendo de texto recebido do HTML para inteiro. Depois, armazena a soma na variável **s**. Em seguida, notamos que o método **println()** da classe **out** dará o retorno ao HTML de uma estrutura válida de documentos hipertexto, concatenando o conteúdo das variáveis criadas.

A sequência de execução, então, será:

- O formulário recebe os dados do usuário e envia ao servidor web (através do método Post neste exemplo).



- O servidor recebe a solicitação, processa os dados via servlet e devolve um HTML via protocolo HTTP.

Se exibirmos o código-fonte da página recebida no navegador, teremos:

```
<html>
        <head>
        <title>Servlet MeuServlet</title>
        </head>
        <body>
        <h1>Servlet MeuServlet at /TesteServlet</h1>
        <p>Num 1: 22</p>
        <p>Num 2: 44</p>
        <p>Soma.: 66</p>
        </body>
    </html>
```

Uma pergunta: mas como o servidor sabe que o nome **MeuServlet** colocado no método action do HTML é o arquivo MeuServlet.Java armazenado em algum lugar no servidor?

Há um arquivo de nome web.xml que serve para mapear os recursos criados no servidor, determinando um vínculo entre as solicitações dos clientes a estes recursos.

Servidores de conteúdo Java

Para finalizar esta aula, apresentamos alguns dos principais servidores de conteúdo Java disponíveis.

- **JRun:** é um servidor de aplicações Java, comprado da Macromedia pela Adobe, que anuncia não mais dar continuidade ao produto. Irá apenas atender os clientes atuais.
- **WebLogic:** é um servidor de aplicações Java mantido pela Oracle.
- **JBoss:** é um servidor de aplicações que está, inclusive, trabalhando com o conceito em nuvem.
- **WebSphere:** é um servidor de aplicações Java robusto da IBM. Este não é "free".
- **Tomcat:** Apache Tomcat é uma implementação open source das tecnologias Java Servlet e JavaServer Pages.
- **GlassFish:** é um servidor de aplicações livre, indicado pela Oracle.

Muito bem! Estamos a caminho do desenvolvimento de sites dinâmicos. São diversas as tecnologias disponíveis utilizadas na indústria de software. Cada um pode adotar uma ou outra tecnologia; a nós, desenvolvedores, cabe o dever de entendermos o necessário da maioria daquelas que estão relacionadas ao nosso meio, tanto para as utilizarmos quanto para auxiliar as empresas em sua escolha.

VALE A PENA CONFERIR

1. Acesse o link e veja um pouco mais sobre o desenvolvimento com Java. O site traz a junção do Java com tecnologias como a Amazon Kindle, BlueJ, Blu-ray Disc, EA Mobile, Google Maps, Sony Ericsson, Sony Pictures, etc. Disponível em: <java.com/pt_BR/ (http://java.com/pt_BR/)>. Acesso em: 14 mar. 2013.

2. Veja mais sobre o JRun. <www.adobe.com/products/jrun (http://www.adobe.com/products/jrun/)>. Acesso em: 14 mar. 2013.

3. Veja mais sobre o WebLogic. Disponível em: Acesso em: 14 mar. 2013.

<www.oracle.com/technetwork/middleware/weblogic (http://www.oracle.com/technetwork/middleware/weblogic)>. Acesso em: 14 mar. 2013.

4. Veja mais sobre o JBoss. Disponível em: <www.jboss.org (http://www.jboss.org/)>. Acesso em: 14 mar. 2013.

5. Veja mais sobre o WebSphere. Disponível em: <www-01.ibm.com/software/websphere (http://www-01.ibm.com/software/websphere/)>.

Acesso em: 14 mar. 2013.

6. Veja mais sobre o Tomcat. Disponível em: <tomcat.apache.org (http://tomcat.apache.org/)>. Acesso em: 14 mar. 2013.

7. Veja mais sobre o GlassFish. Disponível em: <glassfish.java.net (http://glassfish.java.net/)>. Acesso em: 14 mar. 2013.

Referências

SANTOS, Rui Rossi dos. *Java na web*: programando sites dinâmicos. Rio de Janeiro: Axcel Books, 2007.

TERUEL, Evandro Carlos. *Web total*: desenvolva sites com tecnologias de uso livre: prático e avançado. São Paulo: Érica, 2009.

TOMCAT. *Servlet interface*. Disponível em: <tomcat.apache.org/> (http://tomcat.apache.org/tomcat-5.5-doc/servletapi/javax/servlet/Servlet.html). Acesso em 07 set. 2012.

Avalie este tópico

ANTERIOR
A estrutura e funcionamento das redes TCP/IP

Índice

PRÓXIMO
Interface com usuário (HTML, CSS e JavaScript)

Ajuda?
(https://ava.uninove.br/seu/AVA/topico/topico.php?idCurso=)

Biblioteca (https://www.uninove.br/conheca-a-uninove/biblioteca/sobre-a-biblioteca/apresentacao/)
Portal Uninove (http://www.uninove.br)
Mapa do Site