Universidade Presbiteriana Mackenzie
Faculdade de Computação e Informática

Introdução a aplicações Java EE

NOTAS DE AULA - Teoria 17

Linguagem de Programação II
2º semestre de 2015
Versão 1.0

Referências

As referências para esta aula são:

o capítulos 1 de:

ORACLE. **The Java EE 5 Tutorial**. Disponível em: <http://docs.oracle.com/javaee/5/tutorial/doc/bnaay.html>. Acesso em: 18.mar.2015.

o capítulos 3 de:

HALL, M.; et al. **Core Servlets and Javaserver Pages: vol. 1: Core Technologies.** New jersey: Prentice-Hall, 2003.

e os capítulos 2 e 3 de:

BATES, B. **Use a Cabeça! Servlets e JSP.** São Paulo: Starlin Alta Consult, 2008.

- As notas de aula são material de apoio para a aula e não têm o objetivo de apresentar o assunto de maneira exaustiva. Não deixe de ler o material de referência da disciplina.

Arquitetura de Software 2-*Tier*

- Também chamada de arquitetura cliente/servidor.
- São sistemas distribuídos que separam o sistema em cliente e servidor, com uma conexão de rede entre eles.
- Em sua forma mais simples envolve uma aplicação servidora que é acessada por múltiplos clientes.

Arquitetura de Software 2-*Tier* (cont.)

- **Cliente/servidor:** de uma forma genérica, é qualquer solução em que múltiplos clientes enviam requisições que são atendidas por um ou mais servidores.
- **Exemplos**: a World Wide Web (onde o browser é o cliente e o servidor HTTP é o servidor), o serviço de FTP (cliente e servidor) e de e-mail (onde aplicações como o Outlook ou o Apple Mail são clientes).
- **Observação**: historicamente este termo é utilizado para se referir a soluções formadas por:
  1. Uma aplicação com interface gráfica instalada em várias estações de trabalho, que acessa os dados em um servidor de banco de dados.
  2. O servidor de banco de dados, que possui a maior parte da lógica de negócios implementada em **stored procedures**.

Arquitetura de Software 3-*Tier*

- Com o surgimento da World Wide Web e o desenvolvimento de **aplicações web** (indo além do simples oferecimento de páginas estáticas em um site), as soluções passaram a ser distribuídas em 3 locais:
  1. As máquinas clientes.
  2. O servidor de aplicações.
  3. O banco de dados (ou outro sistema legado).
- As plataformas Java Enterprise Edition e Microsoft .Net adotaram esta arquitetura, e muitos ainda utilizam o termo (3-*Tier*) apesar de efetivamente as soluções atuais poderem ter um número maior de *tiers* (N-*tier* ou *multitier*).

Java EE

- Java Enterprise Edition
- Modelo de aplicação que pode ser implementado utilizando tecnologias baseadas na linguagem de programação Java e na máquina virtual Java.

Java EE (cont.)

Exemplo 1: arquitetura Java EE 3-*tier*

1. Cliente:
   - Navegador web
2. Servidor de aplicações:
   - Apresentação: servlets e páginas JSP
   - Regras de Negócio: Enterprise Java Beans
3. Banco de dados (ou outro sistema legado)

Java EE (cont.)

Exemplo 2: arquitetura Java EE 3-*tier*

1. Cliente:
   - Aplicação GUI desenvolvida em Java (utilizando *swing* ou *JavaFx*)
2. Servidor de aplicações:
   - Regras de Negócio: Enterprise Java Beans
3. Banco de dados (ou outro sistema legado)

Java EE (cont.)

Exemplo 3: arquitetura Java EE 4-*tier*

1. Cliente:
   - Navegador web
2. Servidor de aplicações web (apresentação):
   - Servlets e páginas JSP
3. Servidor de aplicações (regras de negócio):
   - Enterprise Java Beans
4. Banco de dados (ou outro sistema legado)

Servlet

- Criar um **servlet** é declarar uma classe filha de **HttpServlet** e instalá-lo em um **container de servlets**.
- No **servlet** define-se como tratar requisições HTTP.

Servlet (cont.)

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorldHtmlServlet extends HttpServlet {

    public void doGet(HttpServletRequest request,
            HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<HTML>");
        out.println("<HEAD><TITLE>Hello</TITLE></HEAD>");
        out.println("<BODY>");
        out.println("<H1>Olá a todos!</H1>");
        out.println("</BODY>");
        out.println("</HTML>");
    }
}
```

Este exemplo mostra um servlet que gera o conteúdo HTML da resposta HTTP.

Nas próximas aulas veremos que geralmente não utilizamos os servlets desta forma.

Servlet (cont.)

Principais tarefas de um servlet:

- Ler os dados enviados pelo cliente explicitamente em um formulário.
- Ler os dados enviados de forma implícita (por exemplo, cookies) em uma requisição HTTP.
- Gerar os resultados. Enviar os dados para o cliente.
- Enviar os dados implícitos de uma resposta HTTP.

Servlet (cont.)

Estrutura básica de um servlet

- Estende a classe **HttpServlet**.
- Sobreescreve os métodos **doGet** e/ou **doPost**.

Servlet (cont.)

Métodos doGet e doPost de um servlet

- **doGet**: executado quando chega uma requisição HTTP do tipo GET
- **doPost**: executado quando chega uma requisição HTTP do tipo POST
- Os dois métodos recebem 2 argumentos: um do tipo **HttpServletRequest** e outro do tipo **HttpServletResponse**

Servlet (cont.)

HttpServletRequest

- Permite que o **servlet** obtenha todos os dados enviados na **requisição HTTP.**
- método **`public String getParameter(String nomeParametro)`** Permite obter o parâmetro identificado como **nomeParametro** passado na **requisição HTTP**; funciona para parâmetros tanto de requisições que utilizam o método GET quanto de requisições que utilizam o método POST.

Servlet (cont.)

HttpServletResponse

- Permite que o **servlet** defina todas as informações que serão enviadas na **resposta HTTP.**
- método **`public PrintWriter getWriter()`**
  Permite obter uma instância de **java.io.PrintWriter** para "imprimir" o conteúdo do documento que será enviado na resposta HTTP.

Servlet (cont.)

Ciclo de vida de um servlet

- Em um container de servlets há **uma única instância** de cada **servlet**.
- Quando um servlet é criado pela primeira vez, o seu método **init** é chamado.
- Cada requisição HTTP que chega faz com que o container de servlets crie uma **thread** e chame o método **service** do **servlet** (múltiplas requisições concorrentes resultam em **múltiplas threads**).
- O método **service** chama o método **doGet**, **doPost** ou **do*Xxx*** de acordo com o tipo de requisição HTTP.
- Quando, por algum motivo, o container decide descarregar o **servlet**, o seu método **destroy** é chamado.

Servlet (cont.)

Servlet e threads

- Uma nova **thread** é criada quando uma requisição HTTP vai ser tratada pelo container de servlets.
- Como pode ocorrer o acesso de uma única instância do servlet por **múltiplas threads**, é necessário tomar medidas para evitar problemas de concorrência. Exemplo: evitar armazenar informações em atributos do servlet.
- Caso seja necessário evitar que um servlet seja acessado por mais de uma **thread** simultaneamente, pode-se fazer com que implemente a interface **SingleThreadModel**. No entanto, isto irá limitar a capacidade do servlet atender uma demanda muito alta de solicitações.

JSP

- Páginas JSP permitem inserir conteúdo gerado dinamicamente em páginas HTML.
- Com o uso de JSP, as páginas HTML podem ser desenvolvidas utilizando ferramentas dedicadas para a construção de páginas web, e as partes dinâmicas são inseridas utilizando as tags especiais <% e %>.

Exemplo de página JSP

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Exemplo de página JSP</title>
    </head>
    <% java.util.Date d = new java.util.Date(); %>
    <body>
        <h1>Página JSP que informa a data e hora do servidor</h1>
        <p>Data e hora do servidor no momento da geração desta página:
        <%= new java.util.Date() %></p>
    </body>
</html>
```

Exemplo de página JSP (cont.)

Texto template

Refere-se ao conteúdo estático da página, que serve como template para o conteúdo dinâmico.

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Exemplo de página JSP</title>
    </head>
    <% java.util.Date d = new java.util.Date(); %>
    <body>
        <h1>Página JSP que informa a data e hora do servidor</h1>
        <p>Data e hora do servidor no momento da geração desta página:

        <%= new java.util.Date() %></p>

    </body>
</html>
```

Exemplo de página JSP (cont.)

Expressões:

Sintaxe: <%= ***expressão Java*** %>

O resultado da expressão é apresentado como conteúdo da página.

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Exemplo de página JSP</title>
    </head>
    <% java.util.Date d = new java.util.Date(); %>
    <body>
        <h1>Página JSP que informa a data e hora do servidor</h1>
        <p>Data e hora do servidor no momento da geração desta página:

        <%= new java.util.Date() %></p>

    </body>
</html>
```

Exemplo de página JSP (cont.)

Scriptlet:

Sintaxe: <% ***código Java*** %>

Sequência de instruções em Java.

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Exemplo de página JSP</title>
    </head>
    <% java.util.Date d = new java.util.Date(); %>
    <body>
        <h1>Página JSP que informa a data e hora do servidor</h1>
        <p>Data e hora do servidor no momento da geração desta página:

        <%= new java.util.Date() %></p>

    </body>
</html>
```

Servlet equivalente à página JSP

- Quando uma página JSP é implantada em um servidor de aplicações, o contâiner gera um servlet equivalente à página JSP.
- No NetBeans é possível visualizar o código do servlet equivalente clicando com o botão direito do mouse na página JSP e selecionando o item "**Exibir servlet**".

Bom estudo!