



Universidade Federal do Rio de Janeiro – IM/DCC

Java para a Web

Prof. Austeclynio Pereira

e-mail: austeclyniop@posgrad.nce.ufrj.br

www.dcc.ufrj.br/~austeclynio



Conteúdo Programático



- ♦ **Modelo cliente-servidor**
- ♦ **Modelos arquiteturais**
- ♦ **Elementos típicos da Internet**
- ♦ **Programando serviços Web em Java**
- ♦ **Componentes de uma aplicação Web**
- ♦ **Páginas estáticas e dinâmicas**
- ♦ **Componentes de uma aplicação Web em Java**
- ♦ **Servlets e JSP**
- ♦ **Arquiteturas para aplicações que usam Servlets e JSP**
- ♦ **MVC**
- ♦ **Tag Form**



Conteúdo Programático



- ♦ **Desenvolvendo aplicativos com Servlets**
- ♦ **Gerenciamento de sessões e cookies**
- ♦ **Codificando Servlets thread-safe**
- ♦ **Como desenvolver uma JSP**
- ♦ **Filtros**
- ♦ **JDBC**
- ♦ **Expression Language**
- ♦ **JSTL**
- ♦ **Struts**
- ♦ **Java Server Faces**
- ♦ **Java Message Service**
- ♦ **EJB3**



Bibliografia



1. Java para Web com Servlets, JSP e EJB – Budi Kurniawan
2. Murach's Java Servlets and JSP – Andrea Steelman
3. **Head First Servlets & JSP – Bryan Basham, Kathy Sierra e Bert Bates**
4. Enterprise Java Developer's Guide – S. Narayanan, Junhe Liu
5. The J2EE Tutorial – Sun Microsystems
6. Core Servlets and JavaServer Pages – Vol I – Marty Hall
7. Como o Tomcat Funciona – Budi Kurniawan e Paul Deck

Avaliação, regras e datas

- ♦ **Trabalho** – Alunos serão organizados em grupos de até dois componentes. Cada dois grupos desenvolverá um site distinto.
- ♦ **Prova** – Será realizada uma prova valendo nove pontos.
- ♦ **Testes-surpresa** – Haverá 2 testes-surpresa, cada um valendo 1(hum) ponto, nos meses de **setembro** e **novembro**. Estes pontos serão adicionados à nota da prova.
 - Para a média ponderada final a prova terá peso 4 e o trabalho peso 6.
- ♦ As avaliações dos trabalhos serão comparativas, trabalhos melhores e mais completos terão notas maiores.

Avaliação, regras e datas

- ◆ Os trabalhos serão apresentados, em público, nas seguintes datas: **28/11** e **05/12**.
- ◆ Cada dupla terá de 15 até 20 minutos para fazer a sua apresentação.
- ◆ A prova será realizada em **17/10**.
- ◆ As evoluções dos trabalhos serão apresentadas em **29/08**, **26/09** e **31/10**.
- ◆ Para cada evolução será atribuída uma nota que irá compor a nota final do trabalho.
- ◆ **Haverá controle formal de presença.**



Ferramentas para o desenvolvimento dos sites



- ◆ IDE NetBeans ou Eclipse ou JCreator.
- ◆ MySql a partir da versão 5.1.
- ◆ TomCat a partir da versão 5.5.12.



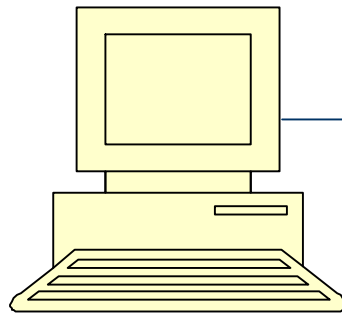
Modelo cliente-servidor



- ◆ Principal padrão utilizado na Internet.
- ◆ Os clientes requisitam os serviços e o servidor realiza os serviços solicitados pelos clientes.
- ◆ Necessidade de uma rede de computadores, de um protocolo de comunicação e de um mecanismo de localização.

Modelos Arquiteturais – uma camada

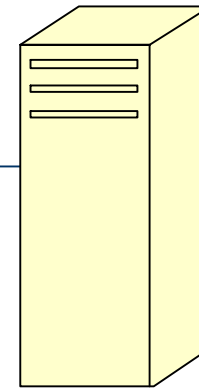
Máquina Cliente



Web Browser

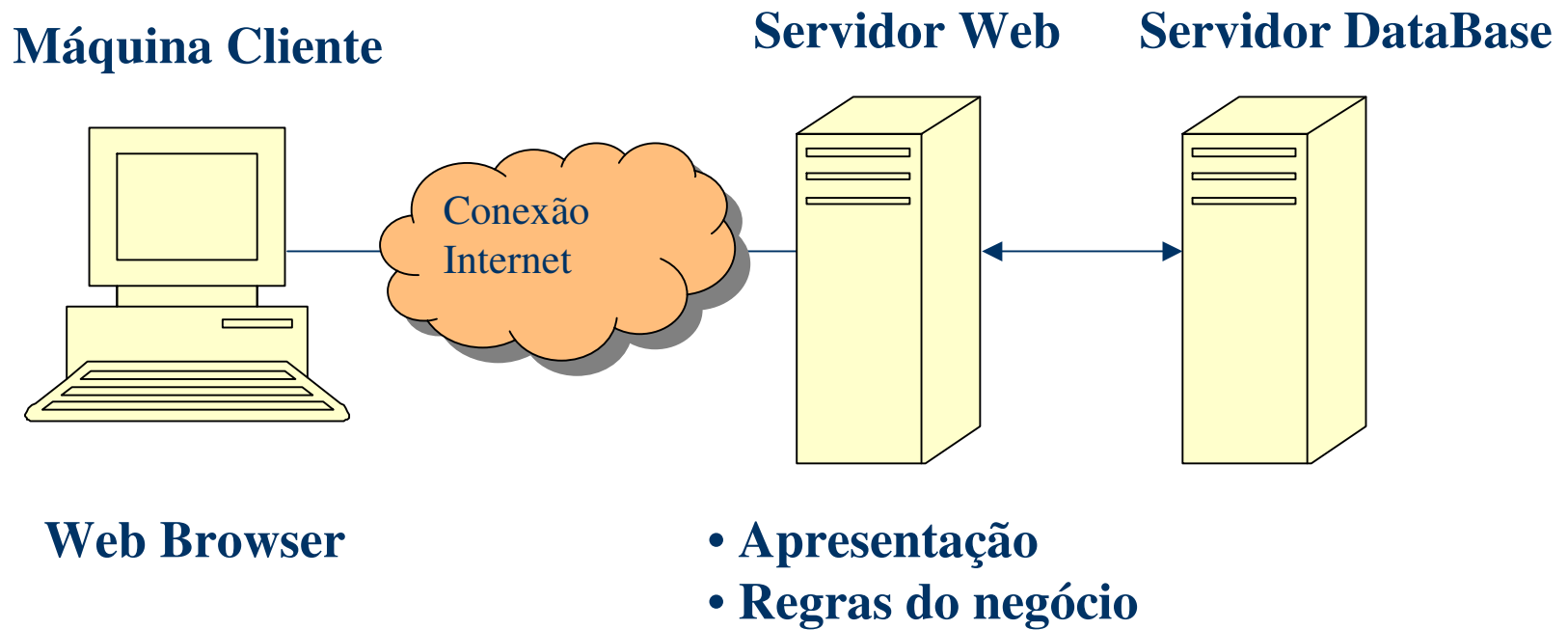


Servidor Web

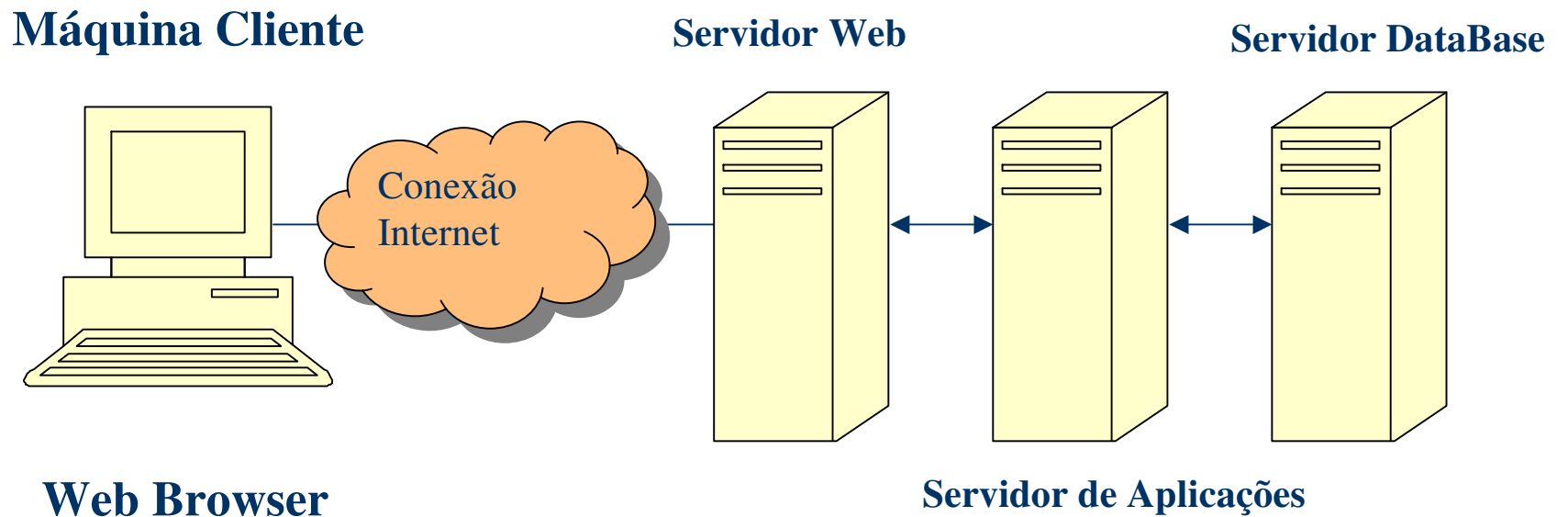


- **Apresentação**
- **Regras do negócio**
- **Persistência**

Modelos Arquiteturais – duas camadas



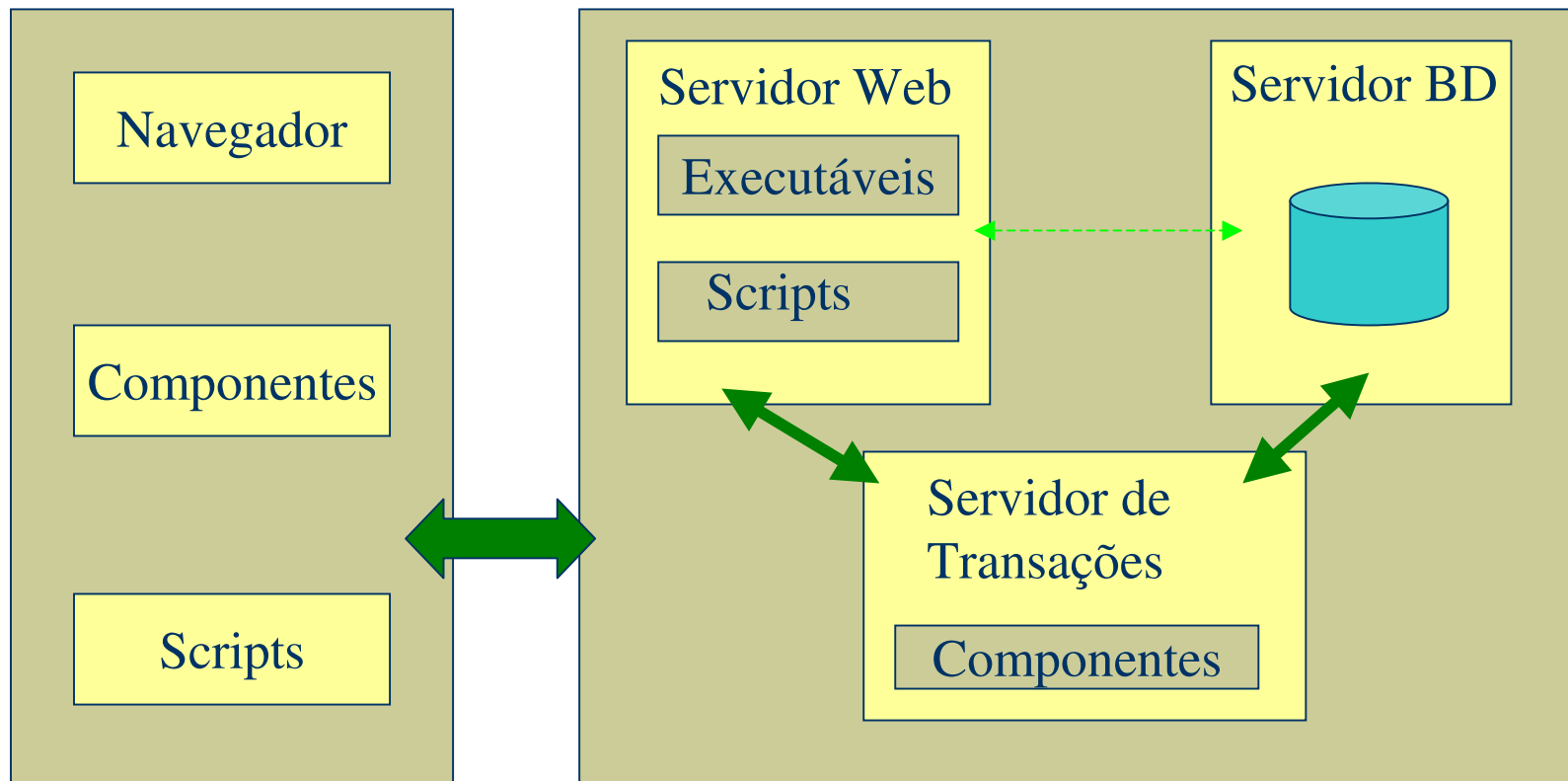
Modelos Arquiteturais – três camadas



Elementos típicos de uma aplicação *web*

Cliente

Servidor





Elementos típicos de uma aplicação *web* - lado cliente



- ◆ **Scripts** – normalmente utilizados para validar dados de entrada. Diminui o número de requisições ao servidor. Ex: JavaScript.
- ◆ **Componentes** – podem conter parte da lógica do negócio, desonerando o servidor. Exs: Applets e Active-X.

Elementos típicos de uma aplicação *web* - lado servidor

- ♦ **Common Gateway Interface** – módulo executável que produz páginas e informações para o cliente. Cada invocação gera um outro processo.
- ♦ **Scripts** – gera uma página HTML para o cliente ou transfere a página para outro servidor. Podem misturar lógica do negócio com apresentação. Exs.: ASP, JSP e PHP.
- ♦ **Componentes** – módulos executáveis invocados por scripts ou por outros módulos executáveis. Exs.: COM+ e EJB.
- ♦ **Executáveis** – executados em um mesmo processo, capacidade de gerenciar sessões, formulários e cookies.Ex.: Servlets.



Estilos Arquiteturais

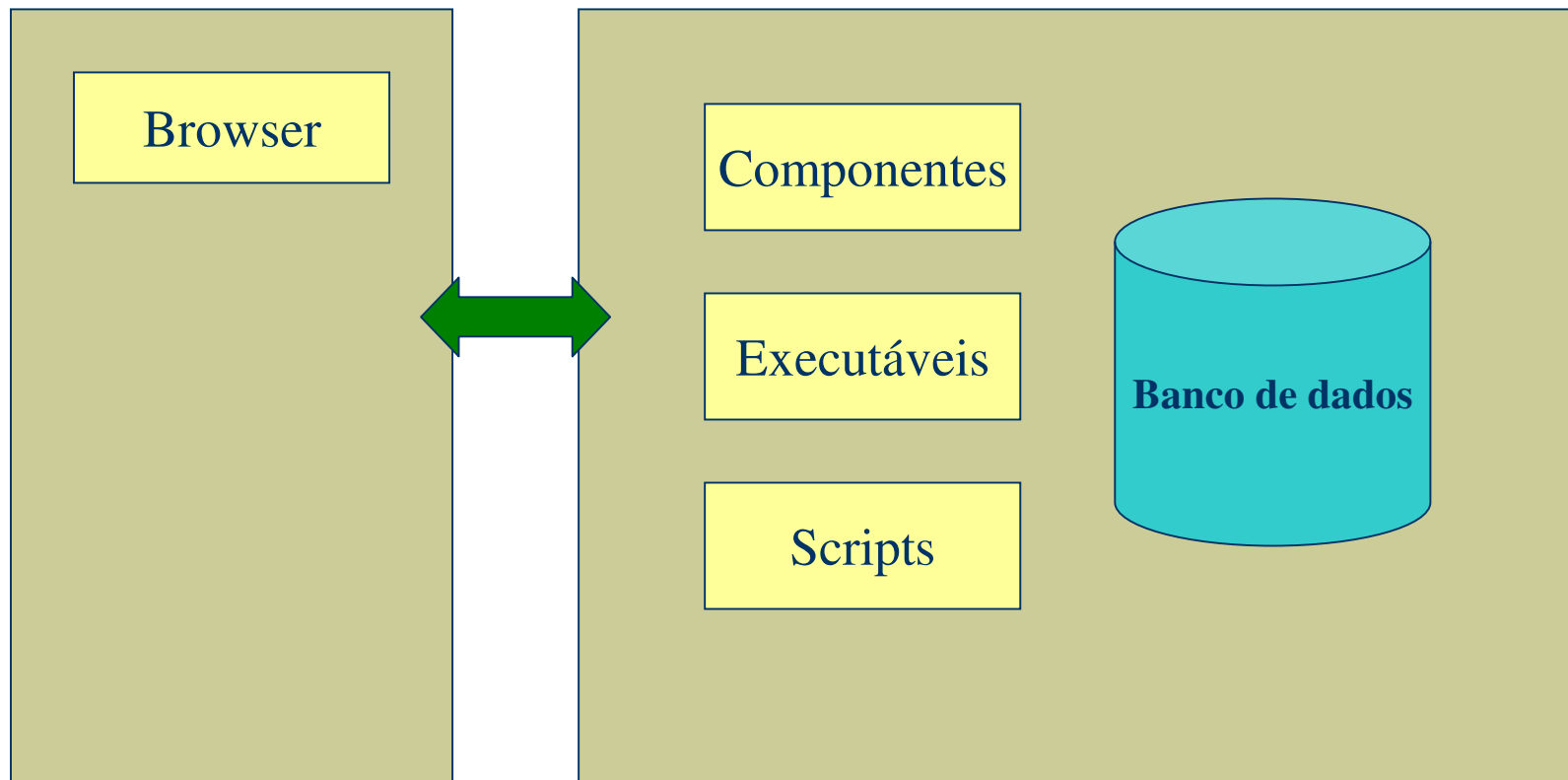


- ♦ **Thin client** – utilização mínima dos recursos da máquina cliente, praticamente tudo é tratado pelo servidor.
- ♦ **Scripted client** – *scripts* na máquina cliente para a verificação de dados.
- ♦ **Thick client** – distribuição da lógica do negócio entre a máquina cliente e a máquina servidora.

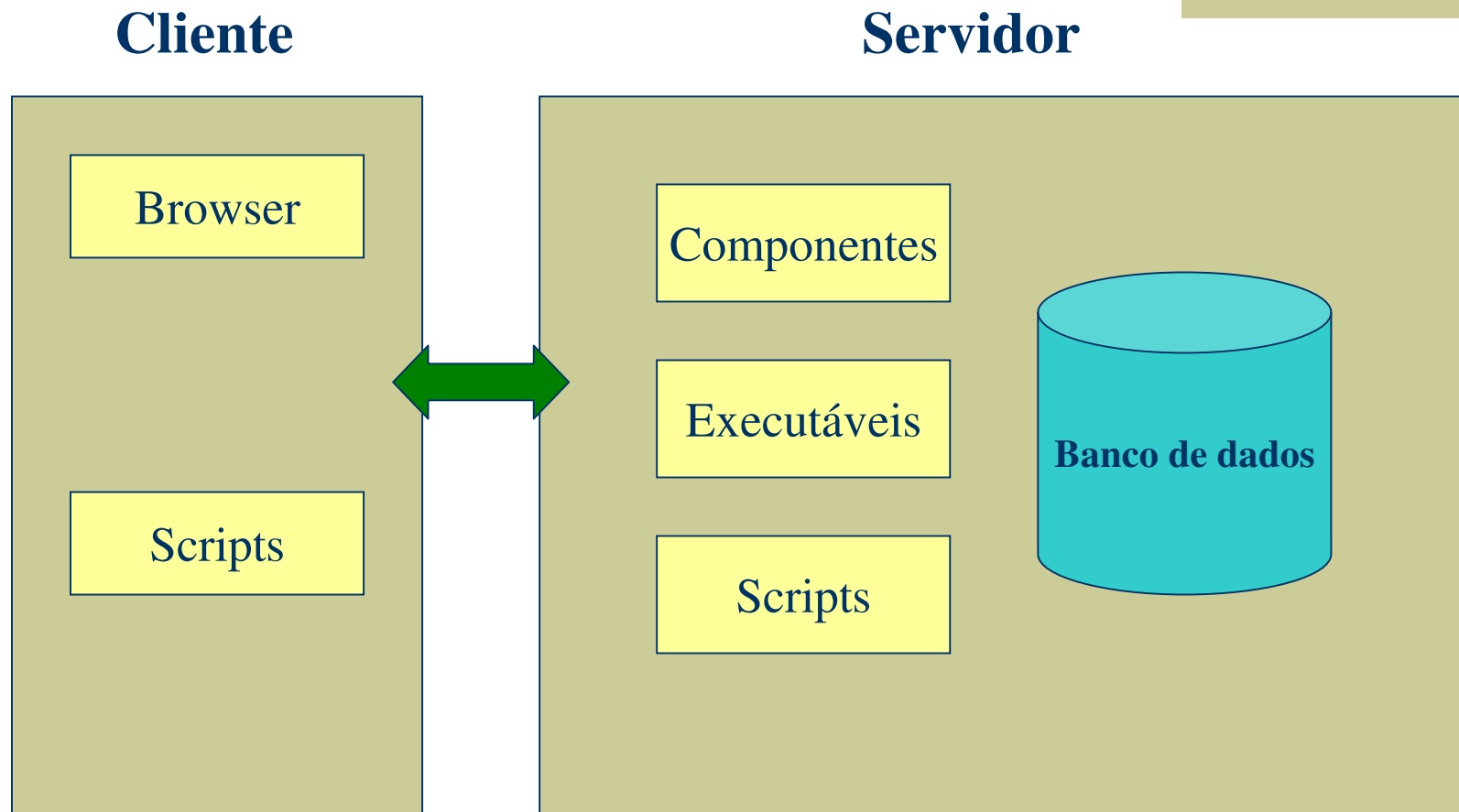
Elementos típicos da Internet thin client

Cliente

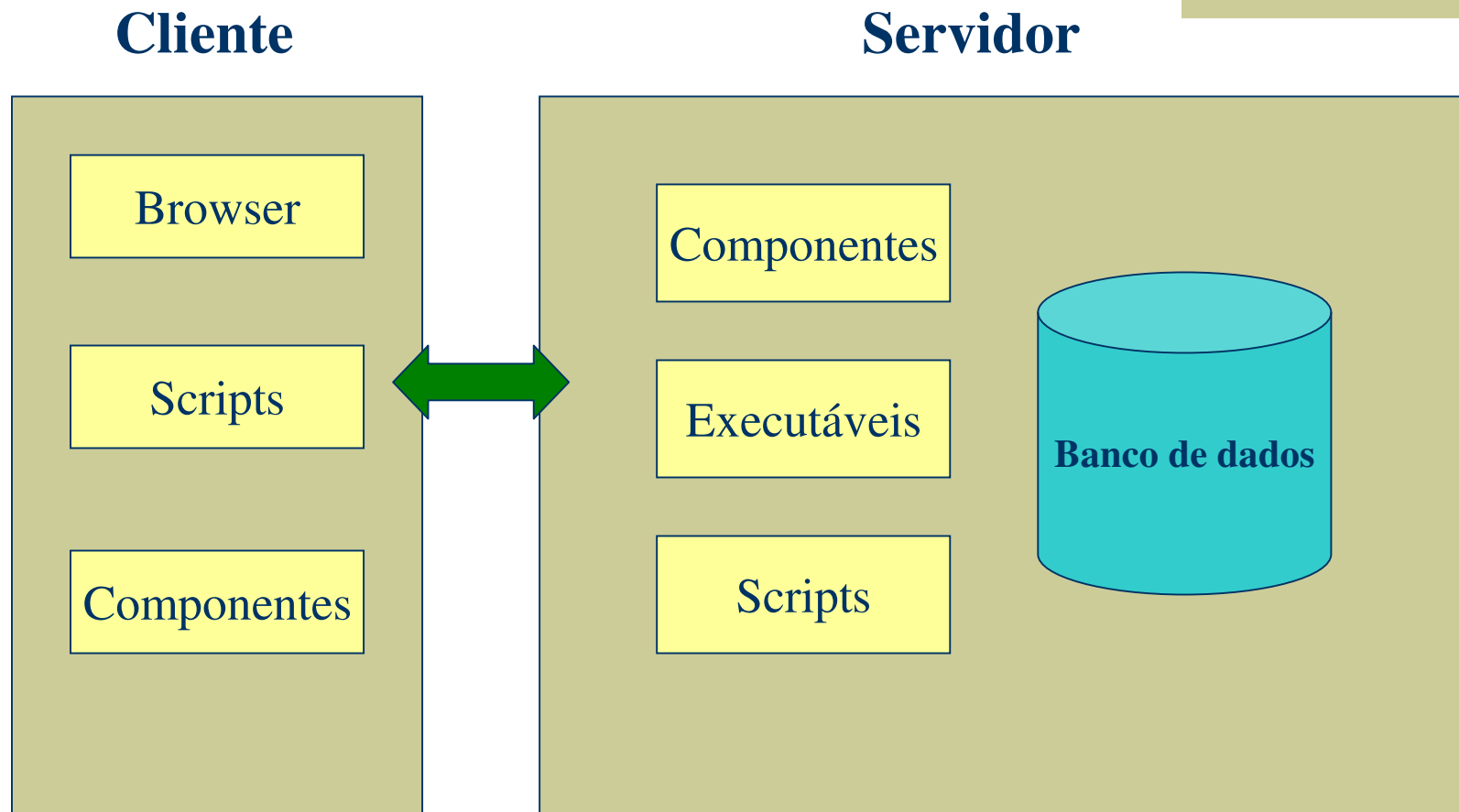
Servidor



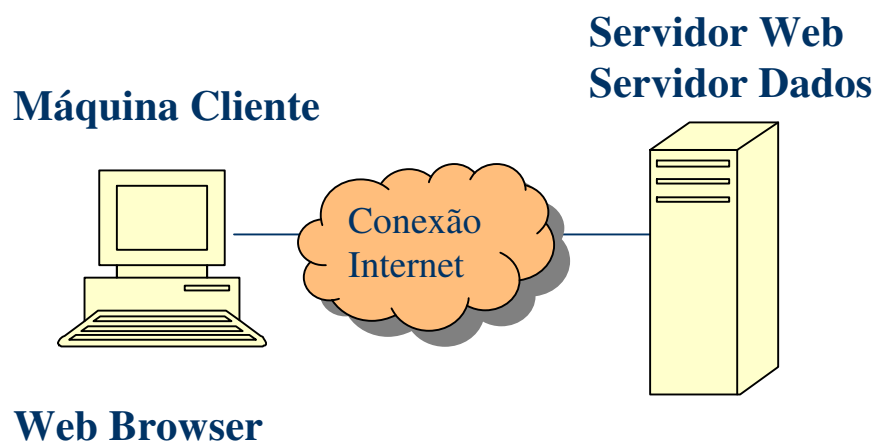
Elementos típicos da Internet scripted client



Elementos típicos da Internet thick client

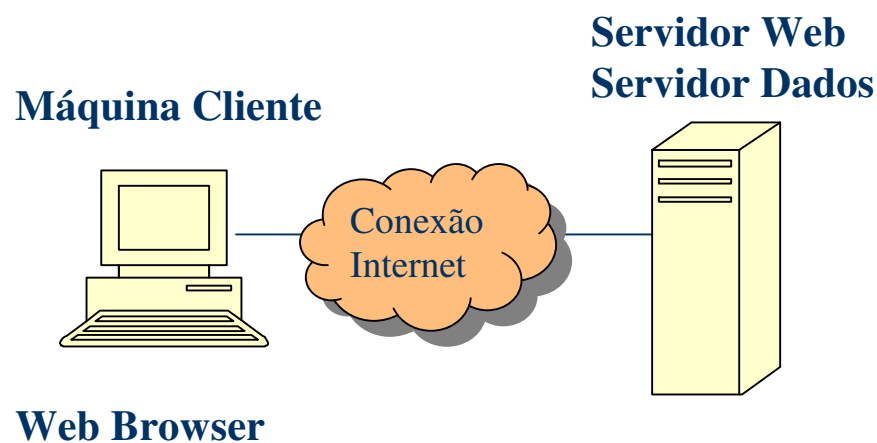


Características de uma aplicação *web*



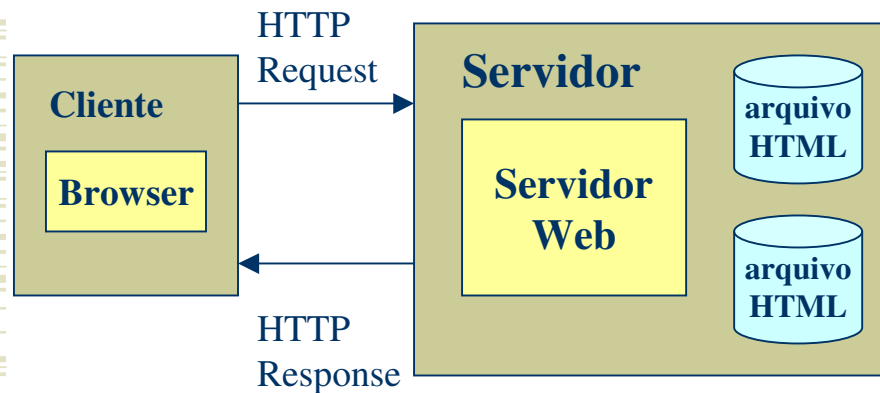
- ♦ Para acessar uma aplicação *web* utiliza-se um *web browser*.
- ♦ O *web browser* renderiza o código HTML para a interface do usuário.
- ♦ O computador servidor executa um *web server* software que enviará as páginas *web* para os *web browsers*.
- ♦ O mais popular *web server* software para executar aplicações Java é o Apache HTTP Server.

Características de uma aplicação *web*



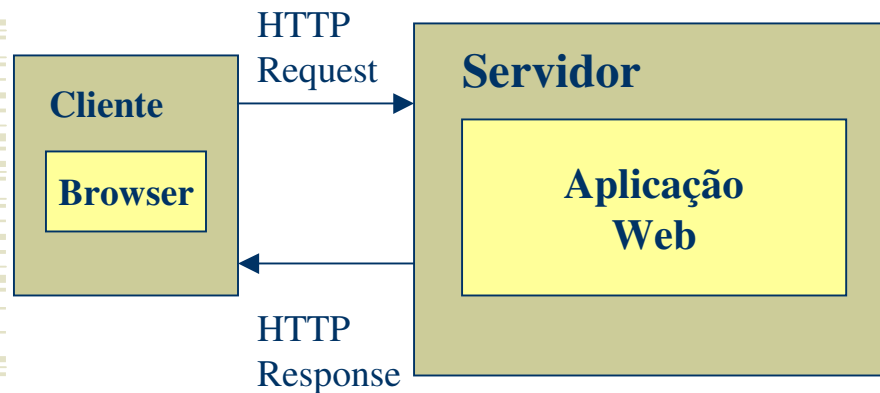
- ♦ Normalmente aplicações *web* necessitam armazenar dados.
- ♦ *Web server* softwares também dão suporte ao DBMS. Os mais populares DBMSs são: Oracle, MySQL, postgresQL e SQLServer.
- ♦ O DBMS deve ser executado, por questões de performance, em um servidor diferente daquele onde reside o *web server* software.
- ♦ Uma vez que a intranet utiliza o mesmo protocolo que a internet uma aplicação *web* pode ser executada em uma LAN.

Como as páginas *web* estáticas trabalham



- ♦ Páginas *web* estáticas são aquelas cujo conteúdo permanece inalterado em resposta a uma ação de um usuário. Possuem o sufixo .htm ou .html.
- ♦ *Web browsers* utilizam o protocolo HTTP (Hypertext Transfer Protocol) para enviar um **request** para o servidor da aplicação *web*.
- ♦ O servidor web recebe o **request**, recupera o arquivo HTML do disco e o envia para o *browser* na forma de um **HTTP response**.
- ♦ O **HTTP response** inclui no documento requisitado todos os recursos descritos no código HTML, tais como gráficos, figuras etc.
- ♦ O *web browser* recebe o **HTTP response**, realiza sua renderização e apresenta ao usuário.

Como as páginas *web* dinâmicas trabalham



- ♦ É um documento que é gerado por uma aplicação *web*, baseado em parâmetros enviados por uma outra página.
- ♦ O servidor *web* recebe o **request** de uma página dinâmica, e passa-o para a aplicação *web*.
- ♦ A aplicação *web* gera um **response** que usualmente é um documento HTML e retorna para o servidor *web*.
- ♦ O servidor embute o documento HTML em um HTTP **response** e envia para o *web browser*.
- ♦ O *web browser* apresenta o documento ao usuário.

O protocolo HTTP

- ◆ Protocolo de rede situado em uma camada acima da TCP/IP.
- ◆ Possui características específicas para aplicações baseadas na Web.
- ◆ A estrutura de um diálogo do tipo HTTP é uma simples seqüência de operações *request/response*.
- ◆ O *web browser* faz o *request* e o *web server* responde.



Elementos chaves de um *request*



- ◆ Ação a ser realizada. Representada por um dos métodos do HTTP.
- ◆ Página que desejamos obter acesso (URL).
- ◆ Parâmetros do formulário invocado.



Elementos chaves de um *response*



- ◆ Código de retorno do *request*.
- ◆ Tipo do conteúdo retornado(texto, figura, HTML etc).
- ◆ Conteúdo(o texto HTML, a figura etc).

Retornando um HTML

Cabeçalho HTTP

```
<html>
<head>
...
</head>
<body>
<img src=...>
</body>
</html>
```

Conteúdo
HTTP

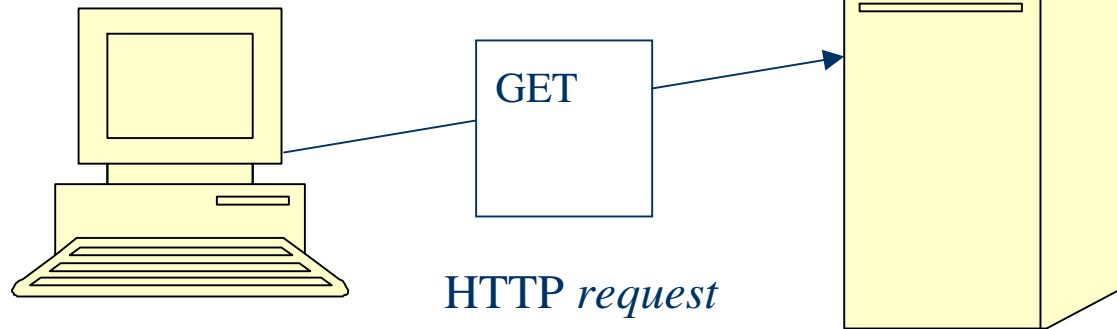
Gera um outro *request*.

Métodos utilizados pelo *request*

- ♦ Um *request* solicita serviços ao *web server* através de métodos do protocolo HTTP.
- ♦ Métodos do HTTP: **GET**, **POST**, HEAD, TRACE, PUT, DELETE, OPTIONS e CONNECT.
- ♦ O *Web browser* envia um HTTP **GET** para o servidor solicitando um recurso. Pode ser: uma página HTML, um JPEG, um PDF etc.
- ♦ O **POST** pode solicitar um recurso e, ao mesmo tempo, enviar um formulário com dados.
- ♦ O **GET** envia dados pela URL!

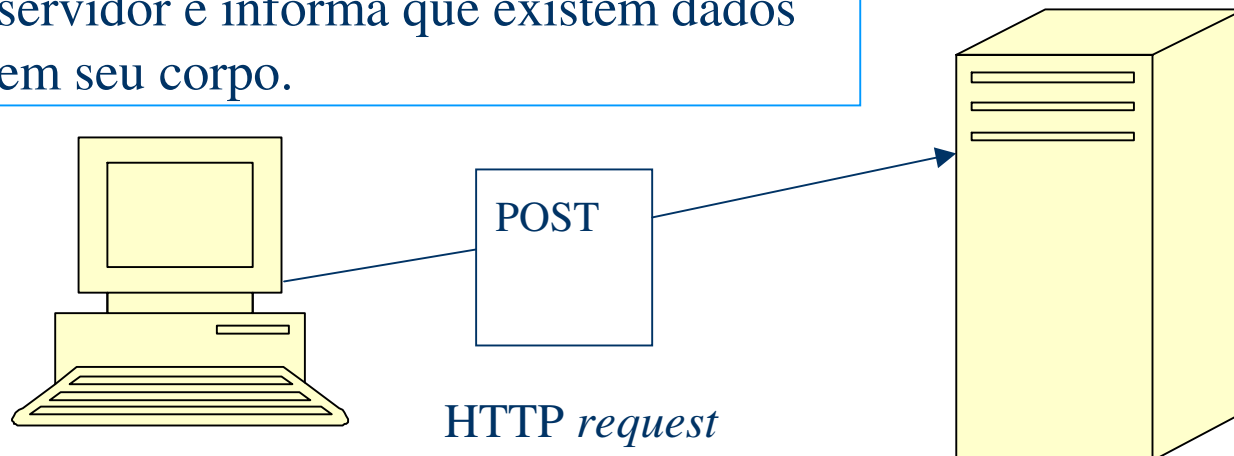
HTTP *request* GET

O GET pede uma página ao servidor passando os parâmetros:
Agência=3080 e Conta=123456



HTTP *request* POST

O POST pede uma página ao servidor e informa que existem dados em seu corpo.



HTTP *response*

- ◆ Composto de um cabeçalho e de um corpo.
- ◆ O cabeçalho serve para informar ao *web browser* :
 - O protocolo utilizado.
 - Se a requisição foi ou não bem sucedida.
 - O tipo do conteúdo que está sendo passado(conhecido como MIME *type*).
- ◆ O corpo contém o conteúdo que será renderizado pelo *web browser*.

Fluxo primário de uma operação *request response*

1. Usuário seleciona uma URL.
2. *Web browser* cria um HTTP GET *request*.
3. O HTTP GET é enviado para o *Web server*.
4. O *Web server* localiza a página solicitada.
5. *Web server* gera um HTTP *response*.
6. O HTTP *response* é enviado para o *Web browser*.
7. O *Web browser* renderiza o HTML.

Uniform Request Locator (URL)

◆ <http://www.nce.ufrj.br:80/concursos/login.html>

Protocolo

Nome do Servidor.
Possui um IP address.

Porta da Aplicação.
Default=80

Nome do recurso
solicitado.
Default=index.html

Caminho onde
o servidor vai
localizar o recurso.

Obs.: Caso seja utilizado o método GET
a URL conterá os parâmetros que serão
passados para o servidor.



Que código Java escrevemos para a *web*?



- ◆ *Servlets.*
- ◆ JavaServer Pages(JSP).
- ◆ Classes de negócio.



Servlets



- ◆ Introduzidos pela Sun em 1996 com o propósito de acrescentar conteúdo dinâmico aos aplicativos *web*.
- ◆ Um *servlet* é uma classe Java executada por um *container*.
- ◆ Tem como benefícios: bom desempenho, portabilidade, rápido ciclo de desenvolvimento e robustez.

Servlet

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

```
public class PrimeiroServlet extends HttpServlet {
```

```
    public void doGet(HttpServletRequest request,  
        HttpServletResponse response)  
        throws IOException, ServletException{
```

```
        PrintWriter out = response.getWriter();
```

```
        out.println("<HTML>");  
        out.println("<HEAD>");  
        out.println("<TITLE>Java para web com servlet e JSP</TITLE>");  
        out.println("</HEAD>");  
        out.println("<BODY>");  
        out.println("BemVindo ao curso de Java para web");  
        out.println("</BODY>");  
        out.println("</HTML>");
```

```
    }
```

```
}
```

JavaServer Page

- ◆ É uma extensão da tecnologia *servlet*. Possui o sufixo *.jsp*.
- ◆ Normalmente, *web designers* escrevem JSPs e programadores *web* escrevem *servlets* e classes Java.
- ◆ Quando uma JSP é requisitada pela primeira vez, ela é convertida para um *servlet* e compilada. Suas requisições futuras invocarão o seu *servlet* correspondente.
- ◆ É executada, também, por um *container*.

JavaServer Page

- ◆ Exemplo 1: HTML puro

```
<HTML>  
<HEAD>  
</HEAD>  
<BODY>  
Java para web  
</BODY>  
</HTML>
```

- ◆ Exemplo 2: HTML + código Java = JSP

```
<HTML>  
<HEAD>  
</HEAD>  
<BODY>  
<%  
    out.println("Java para web");  
%>  
</BODY>  
</HTML>
```

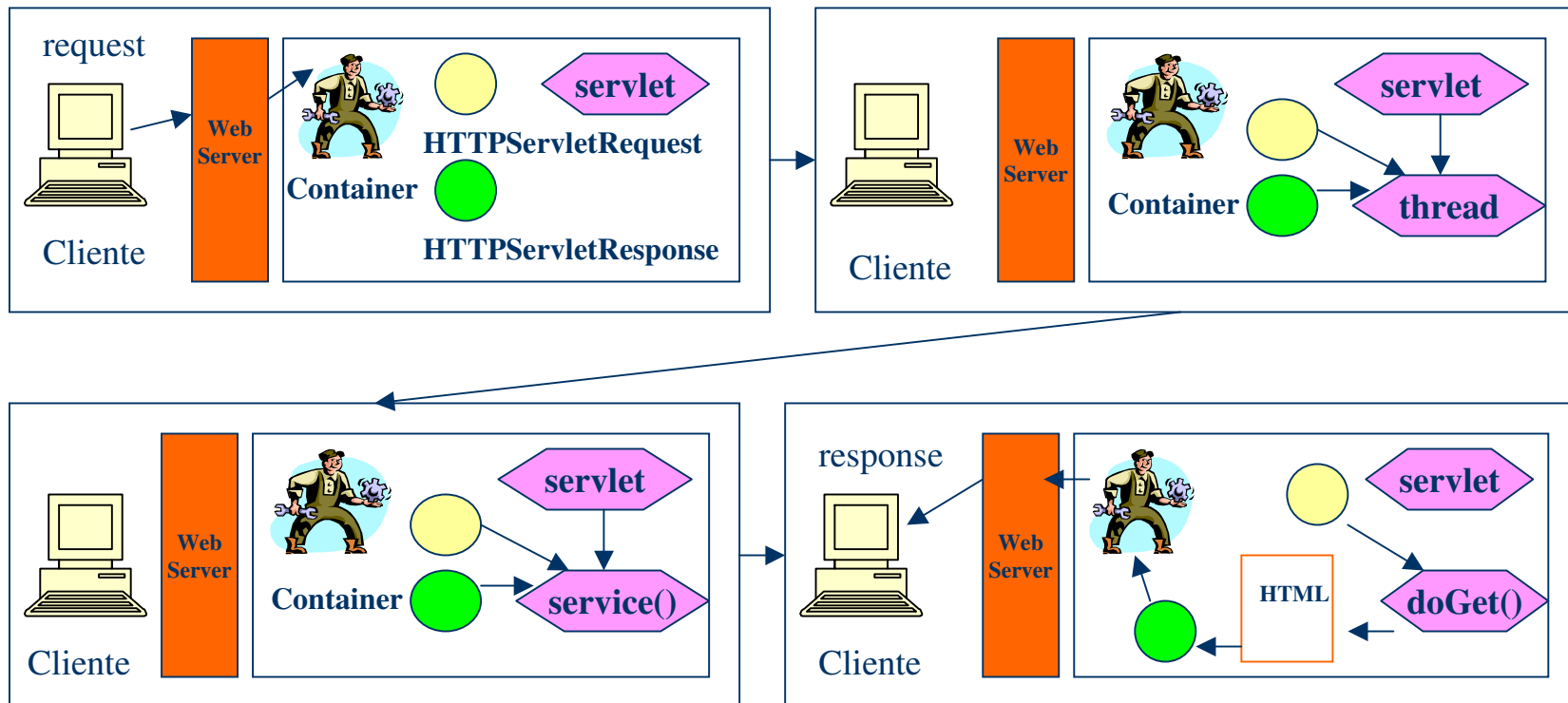
Container

- ♦ O *web server* não sabe tratar páginas dinâmicas.
- ♦ Necessidade de um *container* para abrigar *servlets* e JSPs.
- ♦ O **TomCat** é um dos mais populares *containers* do mercado.
- ♦ O *web server* solicita ao *container* as páginas dinâmicas.
- ♦ *Servlets* e JSPs não possuem um método `main()`. São carregados pelo *container*.

Container - Propósitos

- ◆ Suporte à comunicação de alto nível. Isenta os desenvolvedores de *servlets* de escreverem *sockets*.
- ◆ Administra o ciclo de vida dos *servlets*.
- ◆ Suporte à múltiplas *threads*.
- ◆ Suporte à segurança. Transparente para o desenvolvedor.
- ◆ “Transforma” um JSP em um *servlet*.
- ◆ Pode atuar também como *web server*.

Container – Tratando requests



Uma aplicação *web* em Java





As plataformas Java



- ◆ JME (Micro Edition)
 - Orientada para o desenvolvimento de aplicativos para dispositivos móveis: celulares, *paggers*, *smart cards* e pdas.
- ◆ JSE (Standard Edition)
 - Orientada para o desenvolvimento de aplicativos do tipo *desktop*.
- ◆ JEE (Enterprise Edition)
 - Orientada para o desenvolvimento de aplicativos corporativos de larga escala.

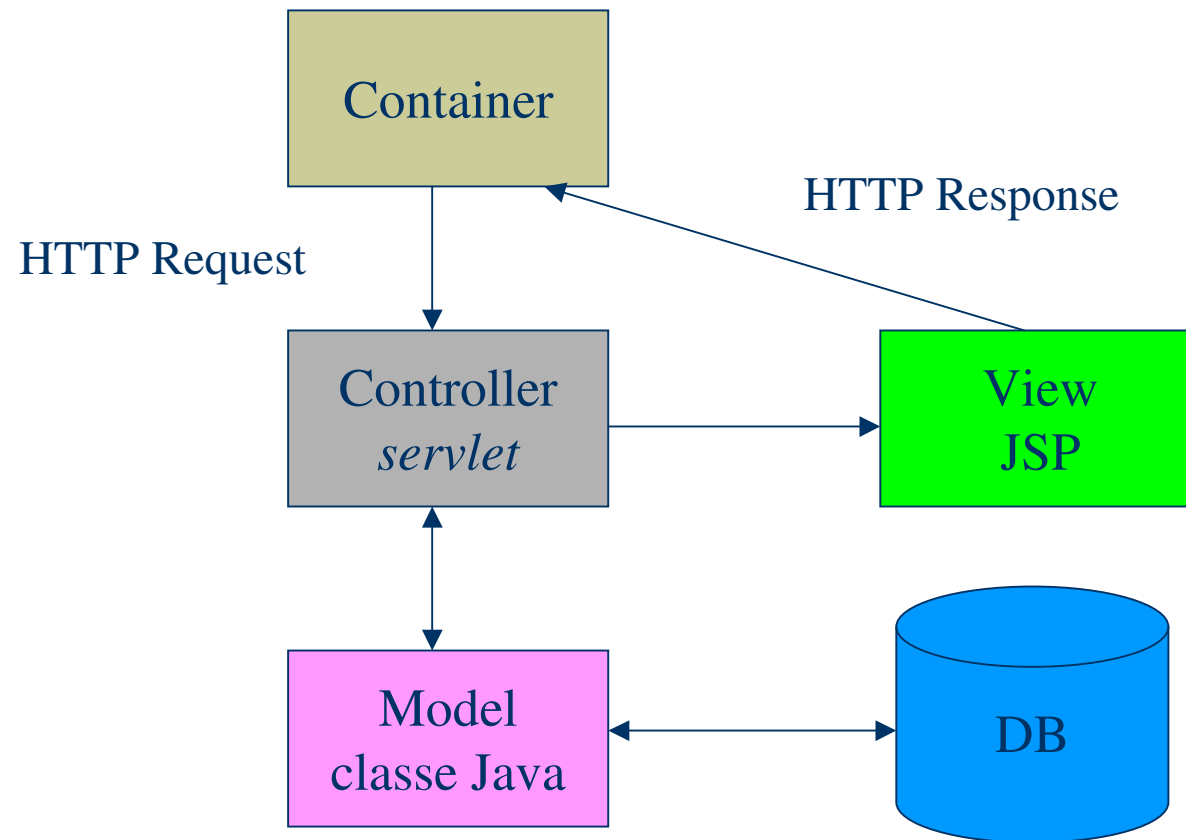
Ferramentas para escrever *servlets* e JSPs

- ♦ Para escrever *servlets* pode ser utilizado qualquer programa que trate aplicações desenvolvidas em Java. Ex: jCreator, Eclipse, NetBeans etc.
- ♦ Para escrever JSPs além do Eclipse e do NetBeans também pode ser usado o Macromedia's HomeSite.
- ♦ IDE para *servlets* e JSPs. Ex: Lomboz(*plugin* para o Eclipse), NetBeans etc.
- ♦ IDE para HTML e JSPs. Ex: Macromedia's Dreamweaver.

Model View Controller (MVC)

- ◆ Padrão de projeto empregado nas aplicações *web*.
- ◆ Separa a lógica do negócio da apresentação.
- ◆ A lógica fica em classes Java específicas.
- ◆ Possibilita o reuso destas classes por outros aplicativos.
- ◆ Divide mais claramente as responsabilidades:
 - A classe é o Model.
 - O *servlet* é o Controller.
 - A JSP é a View.

Model View Controller (MVC)





Model View Controller (MVC)



- ◆ O **Controller** recebe os dados do cliente e os repassa ao Model.
- ◆ O **Model** aplica as regras do negócio e retorna a informação para quem as solicitou.
- ◆ A **View** obtém o estado do Model, repassado pelo Controller, apresentando-o ao cliente.



Criando uma aplicação *web*



- ◆ São necessários 4 passos:
 1. Definir as páginas que serão vistas pelo cliente.
 2. Criar o ambiente de desenvolvimento.
 3. Criar o ambiente de produção/distribuição.
 4. Realizar os testes.

Aplicação *Sugestão Musical*

Visão do cliente

nonononononononononononononononononnn

Selecione o estilo musical preferido:

Rock: ☒
Samba: ☐
Ópera: ☐
MPB: ☐

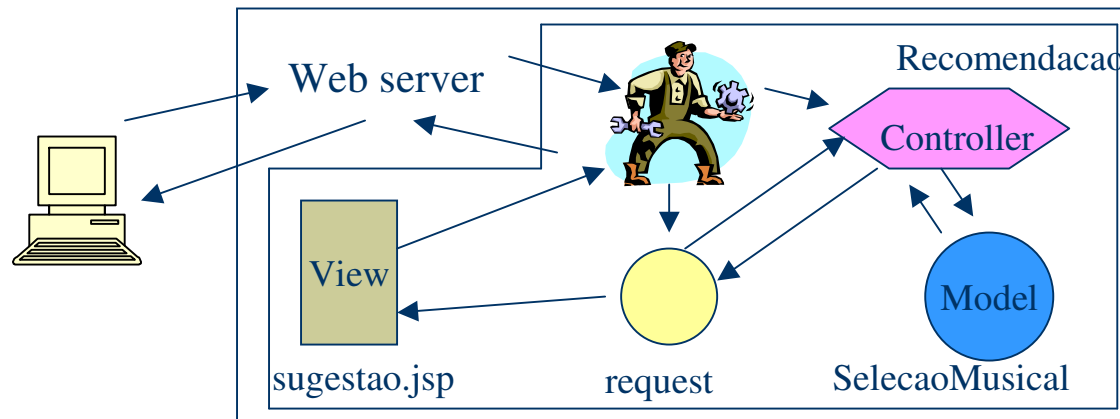
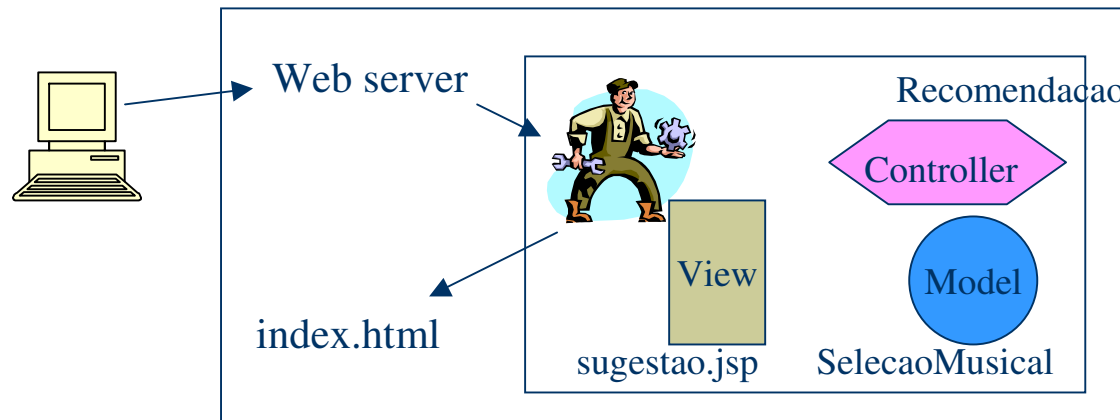
Enviar

nonononononononononononononononononnn

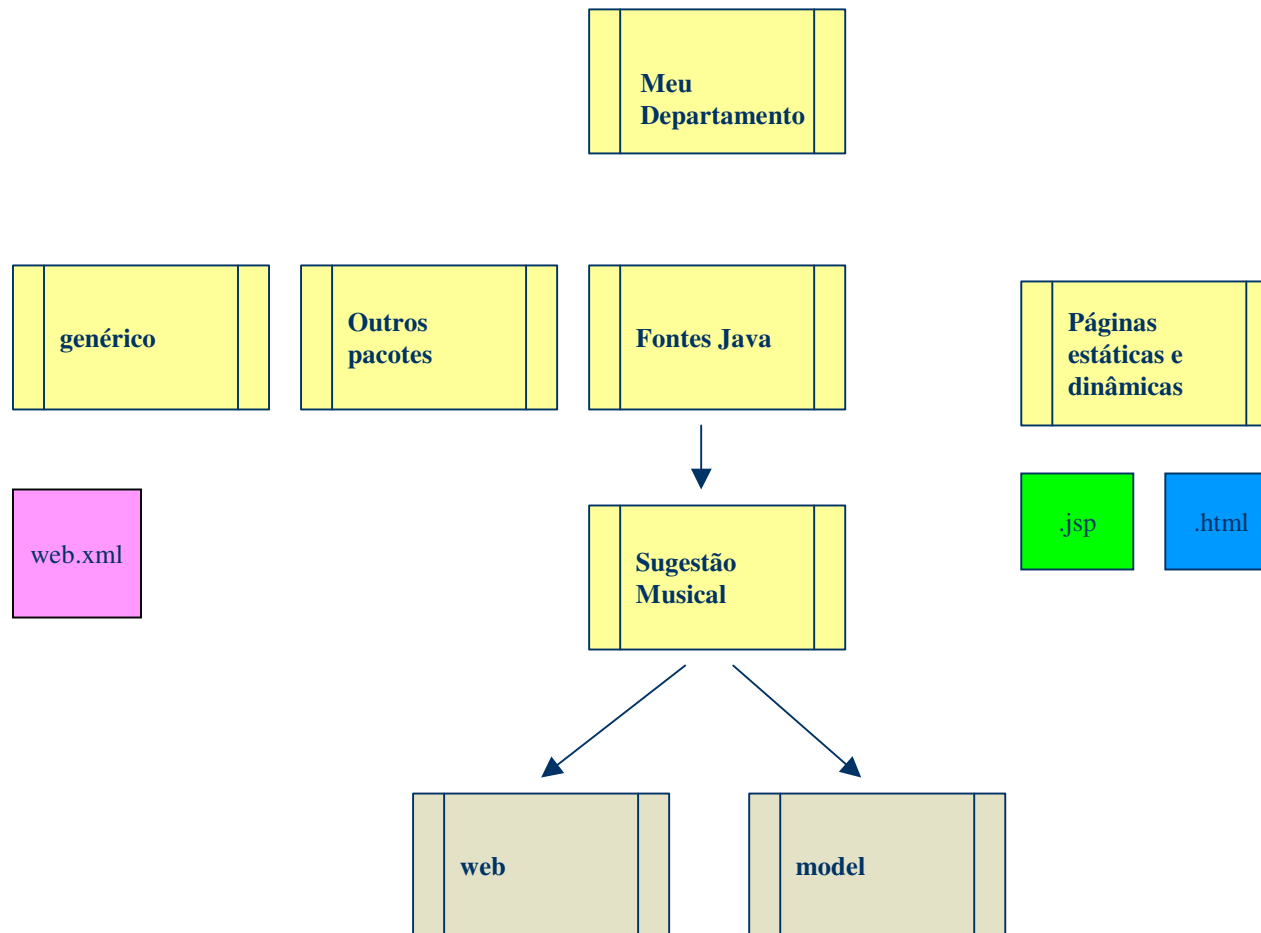
Recomendação musical :

Led Zeppelin
U2
The Who
Yes

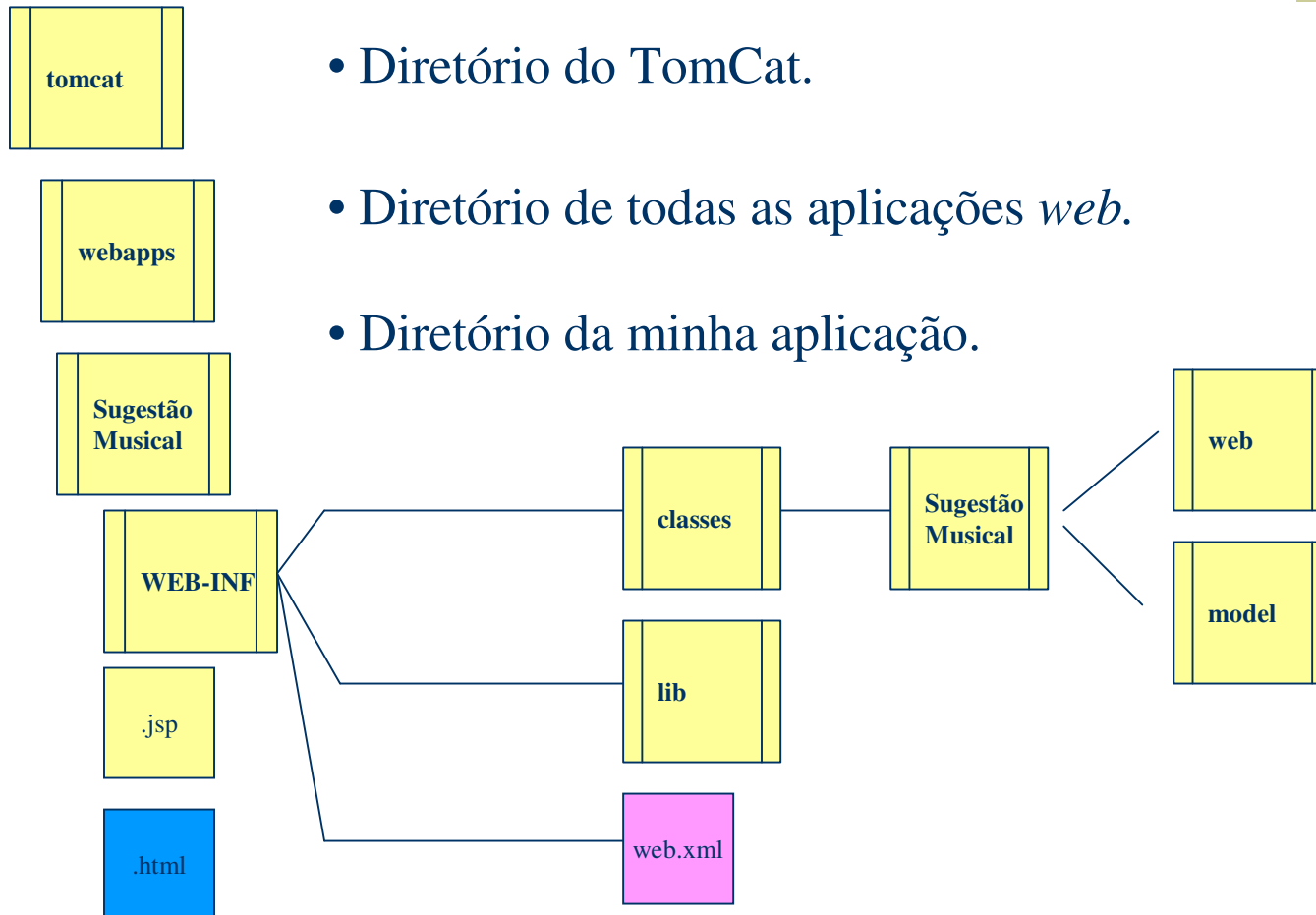
Arquitetura do aplicativo



Ambiente de desenvolvimento



Ambiente de testes



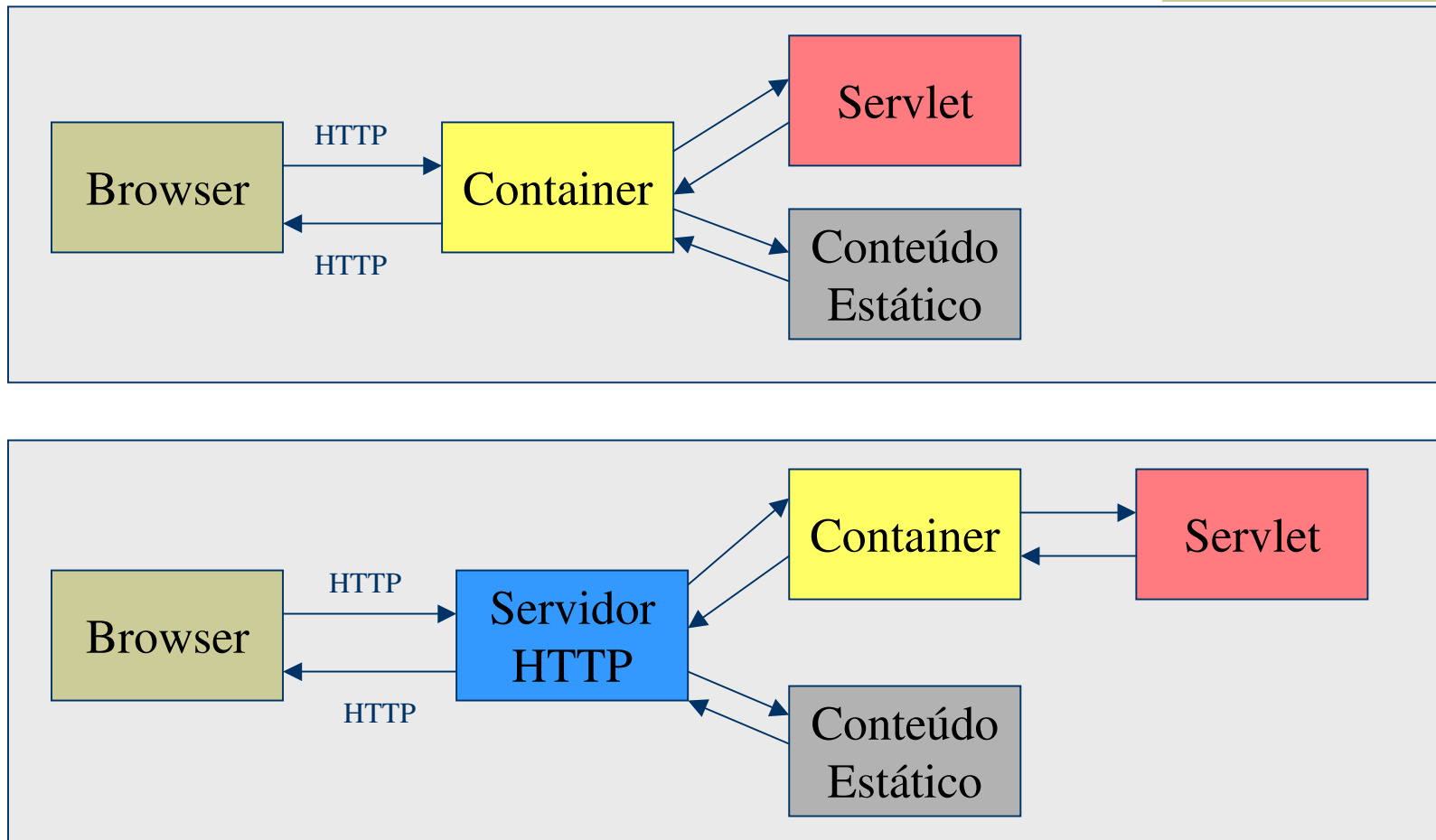


Características do sub-diretório WEB-INF

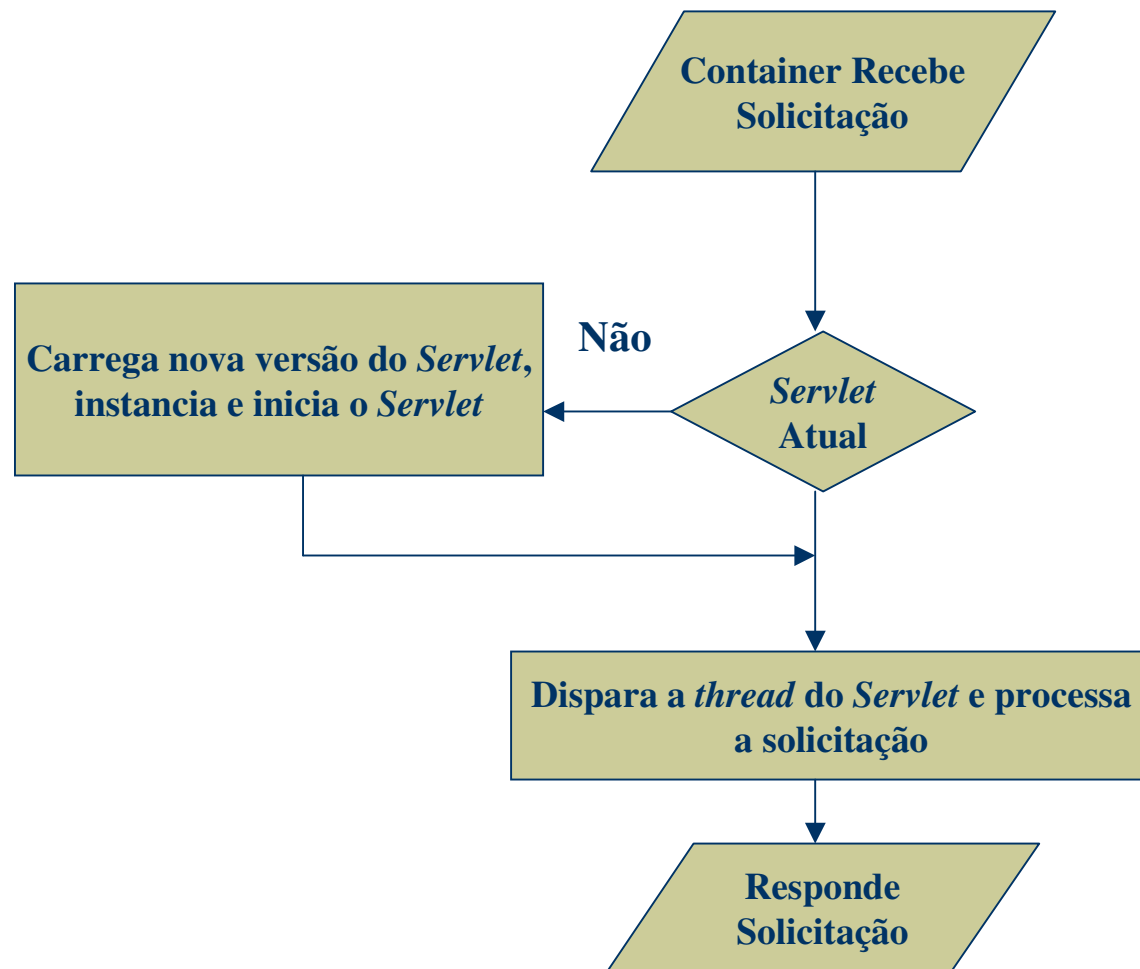


- ◆ Não fica visível para o *web browser* cliente.
- ◆ Residência do arquivo descritor web.xml.
- ◆ Os *servlets* residirão no sub-diretório classes.
- ◆ As classes que refletem as regras do negócio também residirão no sub-diretório classes.

Ativando um *servlet*



A carga de um *servlet*



A distribuição descritiva

- ◆ É um documento XML que contém informações que descrevem os servlets.
- ◆ Denominado web.xml.
- ◆ Possui a *tag* web-app que descreve todos os servlets da aplicação.
- ◆ Associados a cada servlet têm as *tags* <servlet-name> , <servlet-class> e <servlet-mapping>.
- ◆ <servlet-name> é o nome que o Tomcat irá referenciar o servlet.
- ◆ <servlet-class> é o nome efetivo do servlet sem a extensão *.class*.



A distribuição descritiva <servlet-mapping>



- ◆ Associa um URL a cada servlet.
- ◆ Evita que o nome do servlet seja apresentado no *web browser*.
- ◆ Utiliza a *tag* <url-pattern>.
- ◆ <url-pattern> define um nome que estará associado ao servlet desejado.

A distribuição descritiva

```
<web-app>
  <servlet>
    <servlet-name>Login</servlet-name>
    <servlet-class>LoginServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Login</servlet-name>
    <url-pattern>/loginservlet</url-pattern>
  </servlet-mapping>
  <servlet>
    <servlet-name>MinhaCompra</servlet-name>
    <servlet-class>CompraServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>MinhaCompra</servlet-name>
    <url-pattern>/minhacompra</url-pattern>
  </servlet-mapping>
</web-app>
```

Como invocar um *servlet* ?

- ◆ Invocando um servlet :
 - <http://localhost:8080/minhapp/loginservlet>
 - <http://www.dcc.ufrj.br/minhapp/minhacompra>
- ◆ Form *tags* para invocar um servlet:
 - `<form action=“../loginservlet” method=“get”>`
 - `<form action=“../minhacompra” method=“post”>`

A página *index.html*

```
<html><body>
<h1 align="center" >Selecione o estilo musical preferido:</h1>
<form method="POST" action="EscolhaGrupo">
<select name="estilo" size="1">
<option> Rock
<option>Samba
<option> Opera
<option> MPB
</select><br>
<center>
<input type="SUBMIT" value="Enviar" >
</center>
</form>
</body>
</html>
```

O servlet Recomendacao

```
package SugestaoMusical.web;
import SugestaoMusical.model.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;

public class Recomendacao extends HttpServlet{
    public void doPost(HttpServletRequest request,
        HttpServletResponse response) throws IOException, ServletException {

        String estilo = request.getParameter("estilo");
        SelecaoMusical selecao = new SelecaoMusical();
        ArrayList<String> retorno = selecao.getLista(estilo);
        request.setAttribute("listaRecomendada" , retorno);

        RequestDispatcher vista = request.getRequestDispatcher("sugestao.jsp");

        vista.forward(request, response);

    }
}
```

A classe *SelecaoMusical*

```
package SugestaoMusical.model;

import java.util.*;
public class SelecaoMusical{

    public ArrayList getLista(String estilo){

        ArrayList<String> grupos = new ArrayList<String>();
        if (estilo.equals("Rock")){
            grupos.add("Led Zeppelin");
            grupos.add("The Who");
            grupos.add("U2");
            grupos.add("Yes");
        }
        else if (estilo.equals("Samba")){
            grupos.add("Zeca Pagodinho");
            grupos.add("Fundo de Quintal");
            grupos.add("Dona Ivone Lara");
            grupos.add("Martinho da Vila");
        }
    }
}
```

A classe *SelecaoMusical*

```
        else if (estilo.equals("Opera")){
            grupos.add("Placido Domingo");
            grupos.add("Luciano Pavarotti");
            grupos.add("Jose Carreras");
            grupos.add("Enrico Caruso");
        }

        else {
            grupos.add("Chico Buarque");
            grupos.add("Milton Nascimento");
            grupos.add("Ellis Regina");
            grupos.add("Gonzaguinha");
        }

        return grupos;
    }
}
```

A JSP *sugestão*

```
<%@ page import="java.util.*" %>
<html>
<body>
  <h1 align =center="center"> Recomendação Musical: JSP</h1>
    <% ArrayList<String> estilo =(ArrayList) request.getAttribute("listaRecomendada");
      for (String musica:estilo){
        out.print("<br>" + musica);
      }
    %>
  </body>
</html>
```

O descritor *web.xml*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

```
<!DOCTYPE web-app  
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"  
"http://java.sun.com/dtd/web-app_2_3.dtd">
```

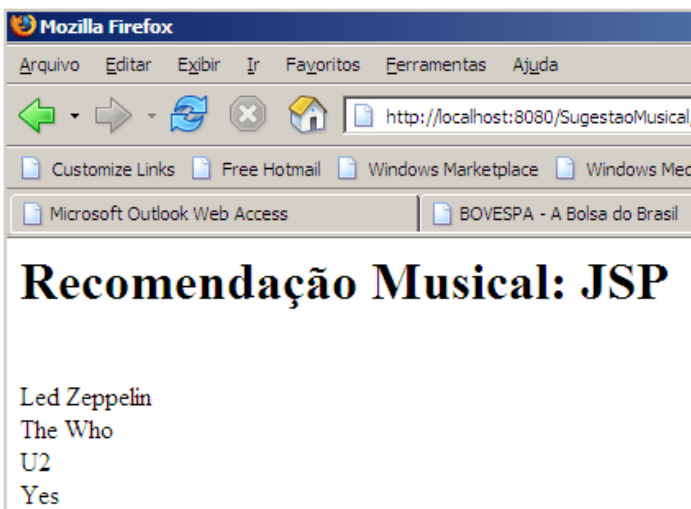
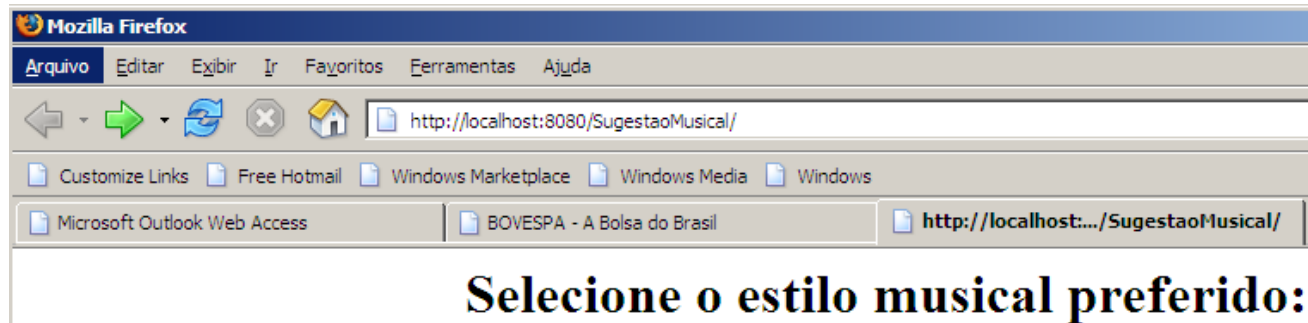
```
<web-app>
```

```
  <servlet>  
    <servlet-name>Musicas</servlet-name>  
    <servlet-class>SugestaoMusical.web.Recomendacao</servlet-class>  
  </servlet>
```

```
  <servlet-mapping>  
    <servlet-name>Musicas</servlet-name>  
    <url-pattern>/EscolhaGrupo</url-pattern>  
  </servlet-mapping>
```

```
</web-app>
```


O resultado



Como codificar um formulário HTML

- ◆ Um formulário contém uma ou mais formas de entradas de dados tais como: *text boxes*, botões, *check boxes*, e *list boxes*.
- ◆ Um formulário **deve conter pelo menos um controle** tal como o botão SUBMIT.
- ◆ Qualquer dado associado ao controle será passado para o *servlet* ou para a JSP que está identificada pelo URL do atributo Action.

Como codificar um formulário HTML

- ♦ Tag `<form>` `</form>` define o início e o fim do formulário.
- ♦ Possui os seguintes atributos:
 - **Action** – especifica o URL do servlet ou da JSP que será chamada quando o usuário clicar o botão SUBMIT.
 - **Method** – especifica que método do protocolo HTTP será usado na operação de *request*. Pode ser GET ou POST.

Uso dos métodos GET e POST

- ◆ Quando usar o método GET ?
 - Se quiser transferir dados mais rapidamente.
 - Se o formulário HTML possui menos de 4 KB de tamanho.
 - Se não há problemas em os parâmetros aparecerem no URL.
- ◆ Quando usar o método POST ?
 - Se estiver transferindo mais do que 4 KB de tamanho.
 - Se não é conveniente os parâmetros aparecerem no URL.

Como codificar um formulário HTML

- ◆ Tag <input> define o tipo da entrada.
- ◆ Atributos comuns:
 - **Name** – é o nome do tipo.
 - **Value** – é o valor *default* do controle.

Como codificar um formulário HTML exemplo

- ◆ Código de um formulário HTML e seu resultado

```
<p>Um form que contém duas text boxes e um botão.</p>  
  
<form action="confirma.jsp" method="post">  
  <p>  
    Nome:<input type="text" name="nome"><br>  
    Email:<input type="text" name="sobrenome">  
    <input type="submit" value="submit">  
  </p>  
</form>
```

Um form que contém duas text boxes e um botão.

Nome:

Email:

Como codificar *text boxes*, *passwords* e campos *hidden*

- ◆ Atributos dos controles de texto:
 - **Type** – especifica o tipo do controle de entrada para os *text boxes*.
 - **Name** – especifica o nome do controle. Este é o nome que será utilizado pela aplicação JSP ou servlet.
 - **Value** – especifica o valor do dado no controle.
 - **Size** – especifica o tamanho do campo de controle em caracteres.
 - **Maxlength** – especifica o número máximo de caracteres que pode estar contido no campo.



Tipos válidos para os *text boxes*



- ◆ Um tipo **Text** cria um *text box* padrão.
- ◆ Um tipo **Password** apresenta um *box* com asteriscos.
- ◆ Um tipo **Hidden** cria um campo *hidden* que armazena textos que não são apresentados pelo *browser*.

Exemplos de *text boxes*, *passwords* e campos *hidden*

```
<p>Login:  <input type="text" name="login" value="jsilva"></p>  
<p>Senha:  <input type="password" name="senha" value="112358"></p>  
<input type="hidden" name="codigoProduto" value="jr01"><br>
```

Login:	<input type="text" value="jsilva"/>
Senha:	<input type="password" value="112358"/>

Como codificar botões

- ◆ Atributos dos botões:
 - **Type** – especifica o tipo do controle de entrada. Os tipos aceitáveis são Submit, Reset ou Button.
 - **OnClick** – especifica o método JavaScript que será executado quando Button for clicado.



Tipos válidos para os botões



- ◆ O tipo **Submit** ativa o atributo Action do formulário.
- ◆ O tipo **Reset** inicia todos os controles do formulário com seus valores originais.
- ◆ O tipo **Button** cria um botão **JavaScript** que quando acionado executa um método pré-estabelecido.

Exemplos do uso de botões

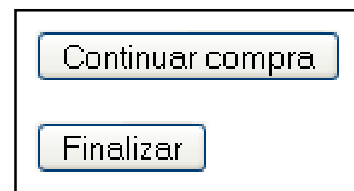
- 3 tipos de botões:

```
<input type="submit" value="Submit">  
<input type="reset" value="Reset">  
<input type="button" value="Confirma" onClick="validate(this.form) ">
```

A rectangular box containing three buttons arranged horizontally. The first button is labeled 'Submit', the second 'Reset', and the third 'Confirma'. Each button has a light gray background and a thin border.

- 2 botões Submit na mesma página:

```
<form action="compra/index.jsp" method="post">  
  <input type="submit" value="Continuar compra">  
</form>  
<form action="servlet/compra.ServletClient" method="post">  
  <input type="submit" value="Finalizar">  
</form>
```

A rectangular box containing two buttons arranged vertically. The top button is labeled 'Continuar compra' and the bottom button is labeled 'Finalizar'. Both buttons have a light gray background and a thin border.



Como codificar *checkboxes* e *radiobuttons*



- ◆ Atributos destes botões:
 - **Type** – especifica o tipo de controle. Os tipos aceitáveis são Checkbox ou Radio.
 - **Checked** – seleciona previamente determinado controle.

Exemplos de *radiobuttons* e *checkboxes*

```
<input type="checkbox" name="addEmail" checked>
```

Sim, me adicione na lista de emails.


```
<br>
```

Entrar em contato por:


```
<input type="radio" name="contatoPor" value="Email">Email
```

```
<input type="radio" name="contatoPor" value="Correios">Correios
```

```
<input type="radio" name="contatoPor" value="Ambos">Ambos<br>
```

```
<br>
```

Me interesse pelos seguintes estilos musicais:


```
<input type="checkbox" name="rock">Rock<br>
```

```
<input type="checkbox" name="classica">Samba<br>
```

```
<input type="checkbox" name="pagode">Pagode<br>
```

☒ Sim, me adicione na lista de emails.

Entrar em contato por:

☐ Email ☐ Correios ☐ Ambos

Me interesse pelos seguintes estilos musicais:

☐ Rock

☐ Samba

☐ Pagode

Como codificar *comboboxes* e *listboxes*

- ◆ Utiliza dois tipos de *tags*: **Select** e **Option**.
- ◆ Deve haver pelo menos uma *tag* Select e duas *tags* Option.
- ◆ Inicia com a *tag* Select que conterà as *tags* Option.
- ◆ A *tag* Option especifica as diferentes opções disponíveis no *box*.
- ◆ A *tag* Select possui o atributo Multiple que converte um *combox* em um *listbox*.
- ◆ A *tag* Option possui o atributo Selected que seleciona previamente uma opção.

Exemplos de *comboboxes* e *listboxes*

- Código de um *combobox*:

```
Selecione um país:<br>
<select name="pais">
    <option value="Brasil" selected>Brasil
    <option value="Canada">Canadá
    <option value="Mexico">México
</select>
```



- Alterando para um *listbox*:

```
<select name="pais" multiple>
```

Selecione um país:



(Para selecionar mais de um país, pressione e segure a tecla Ctrl)

Como codificar uma *textarea*

- ◆ Uma *textarea* difere-se de uma *textbox* pelo fato de suportar múltiplas linhas.
- ◆ Usa a tag `<Textarea> </Textarea>`
- ◆ Atributos da *textarea*:
 - Rows – especifica o número de linhas visíveis na *textarea*. Se exceder é utilizado um *scroll bar*.
 - Cols – especifica a largura da *textarea*.

Exemplo de *textarea*

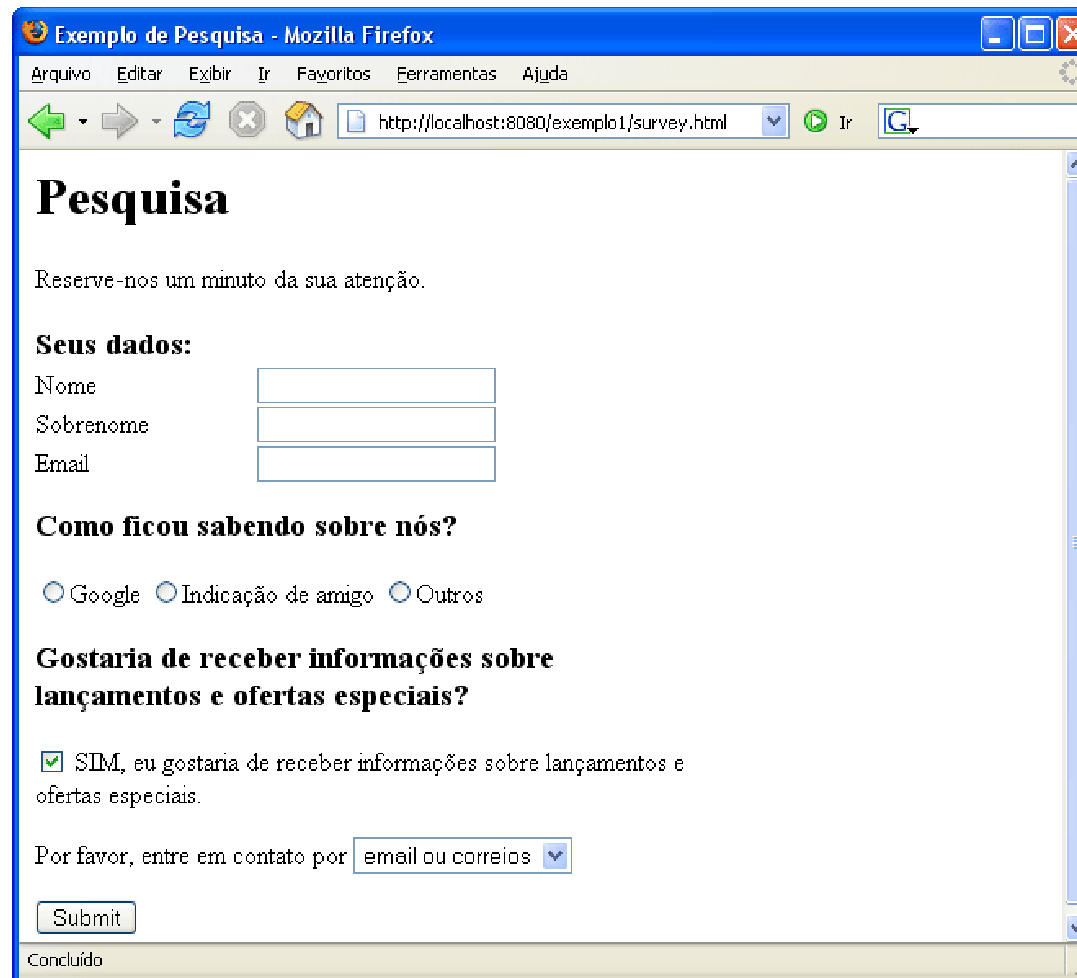
- Código de uma *textarea*:

```
Comentários:<br>  
<textarea name="comentario" rows="8" cols="60"></textarea>
```

Comentários:

Sim, compatriotas, não esperemos mais, a hora é esta. Vamos cometer um haraquiri coletivo, (...). Pronto, aí tudo fica perfeito. Talvez um pouco esquisito, mas objeto inquestionável de admiração internacional e mais uma vez pioneiro: seremos o primeiro país sem povo e todos os problemas desapareceriam. Por que não pensamos nisso antes? Erram, como sempre, os catastrofistas. O Brasil tem futuro, sim, apesar de que não estaremos aqui para testemunhá-lo, mas não se pode querer tudo neste mundo." (João Ubaldo

Combinando *tags* - resultado final



The screenshot shows a Mozilla Firefox browser window with the title "Exemplo de Pesquisa - Mozilla Firefox". The address bar displays "http://localhost:8080/exemplo1/survey.html". The page content includes a menu bar (Arquivo, Editar, Exibir, Ir, Favoritos, Ferramentas, Ajuda), navigation buttons, and a search bar. The main content area is titled "Pesquisa" and contains the following sections:

Pesquisa

Reserve-nos um minuto da sua atenção.

Seus dados:

Nome

Sobrenome

Email

Como ficou sabendo sobre nós?

☐ Google ☐ Indicação de amigo ☐ Outros

Gostaria de receber informações sobre lançamentos e ofertas especiais?

☒ SIM, eu gostaria de receber informações sobre lançamentos e ofertas especiais.

Por favor, entre em contato por

Concluído

Combinando *tags* – código HTML

```
</doctype html public "-//W3C//DTD HTML 4.0 Transitional//EN">

<html>
<head>
  <title>Exemplo de Pesquisa</title>
</head>
<body>

<table border="0" cellpadding="0">
<form action="/exemplo1/servlet/br.ufrj.dcc.poo.SurveyServlet" method="post">
  <tr>
    <td width="410" valign="top" colspan="2">

      <h1>Pesquisa</h1>
      <p>Reserve-nos um minuto da sua atenção.</p>
      <h3>Seus dados:</h3>
    </td>
  </tr>
  <tr>
    <td><p>Nome</p>

    <td><input type="text" name="nome" size="20" tabindex="1"></td>
  </tr>
  <tr>
    <td><p>Sobrenome</p>
    <td><input type="text" name="sobrenome" size="20" tabindex="2"></td>
  </tr>
```

Combinando *tags* – código HTML

```
<tr>
  <td><p>Email</p>

  <td><input type="text" name="email" size="20" tabindex="3"></td>
</tr>
<tr>
  <td colspan="2" height="12"></td>
</tr>
<tr>
  <td width="410" valign="top" colspan="2">
    <h3>Como ficou sabendo sobre nós?</h3>

    <p>
      <input type="radio" name="heardFrom" value="Google" tabindex="4">Google
      <input type="radio" name="heardFrom" value="Amigo">Indicação de amigo
      <input type="radio" name="heardFrom" value="Outros">Outros
    </p>
    <h3>Gostaria de receber informações sobre lançamentos e ofertas especiais?</h3>
    <p><input type="checkbox" name="querAtualiza" checked> SIM, eu gostaria de receber
informações sobre lançamentos e ofertas especiais.<br>
  </p>
  <p>
```

Combinando *tags* – código HTML

```
Por favor, entre em contato por
<select name="contatoPor">
  <option value="Ambos" checked>email e correios
  <option value="Email">email apenas
  <option value="Correios">correios apenas
</select>
</p>
<p><input type=submit value="Submit" tabindex="5"></p>
</td>
</tr>
</form>

</table>
</body>
</html>
```

O ciclo de vida de um *servlet*

- ◆ O método `init()`
 - Inicia o servlet.
 - O *container* chama este método apenas uma vez.
 - Pode ser utilizado para iniciar variáveis, carregar o *driver* de um banco de dados etc.
 - Recebe, através do objeto `ServletConfig`, os valores especificados no arquivo `web.xml`.
 - Assinatura do método:
`public void init(ServletConfig config) throws ServletException`
 - Método de uso opcional.



O ciclo de vida de um *servlet*



- ◆ O método `Service()`
 - É acionado pelo *container* após o término bem sucedido do método `init()`.
 - Executado a cada chamada do *servlet*.
- ◆ `Destroy()`
 - Remove o *servlet*. Ocorre por falta de uso ou *shutdown* do *server*.

Como desenvolver *servlets*

- ◆ Um *servlet* herda da classe `HttpServlet` que herda da classe `GenericServlet` que implementa a interface `Servlet`.
- ◆ Necessário importar os pacotes `javax.servlet`, `javax.servlet.http`.
- ◆ O método `init()` pode ser sobreposto.
- ◆ Pelo menos um método de serviço precisa ser sobreposto.

Como desenvolver *servlets*

- ◆ O método **doGet** processa todos os HTTP *requests* que usam o método Get.
- ◆ O método **doPost** processa todos os HTTP *requests* que usam o método Post.
- ◆ Estes métodos recebem os objetos *request* e *response* repassados pelo *container*.
- ◆ O método `setContentType`, do objeto *response*, indica o tipo de resposta retornada ao *browser*.
- ◆ O método `getWriter`, do objeto *response*, é usado para enviar o arquivo HTML para o *web browser*.

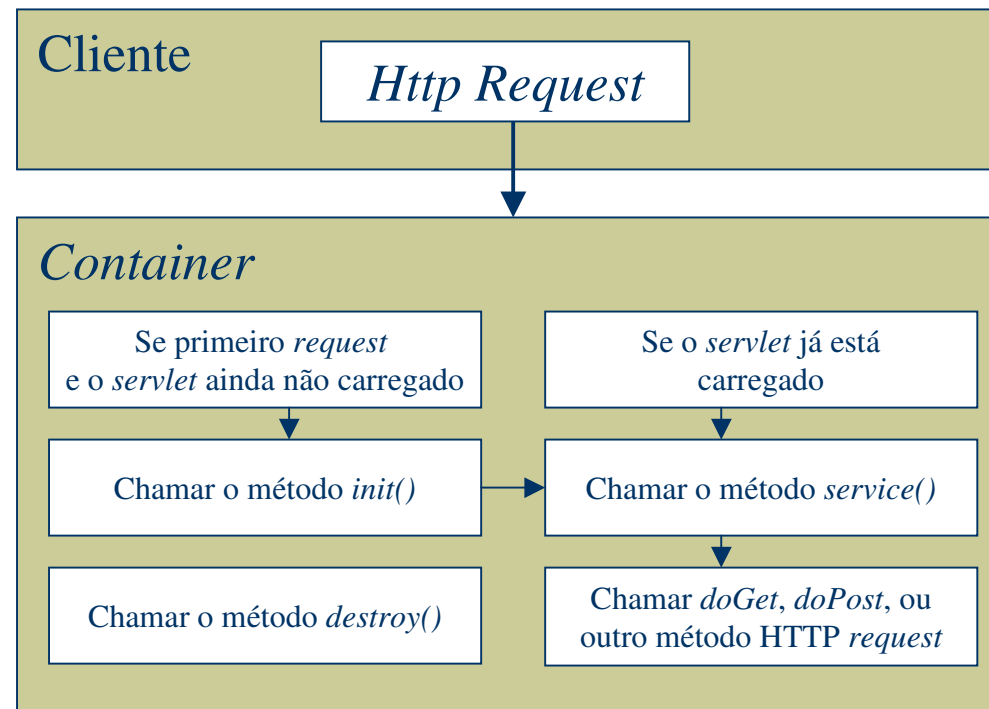


Alguns privilégios dos *servlets*



- ◆ Capacidade de “*logar*” eventos.
- ◆ Obter referências para outros recursos.
- ◆ Passar atributos para outros *servlets*.

Como o *container* trata um *request* para um *servlet* ?



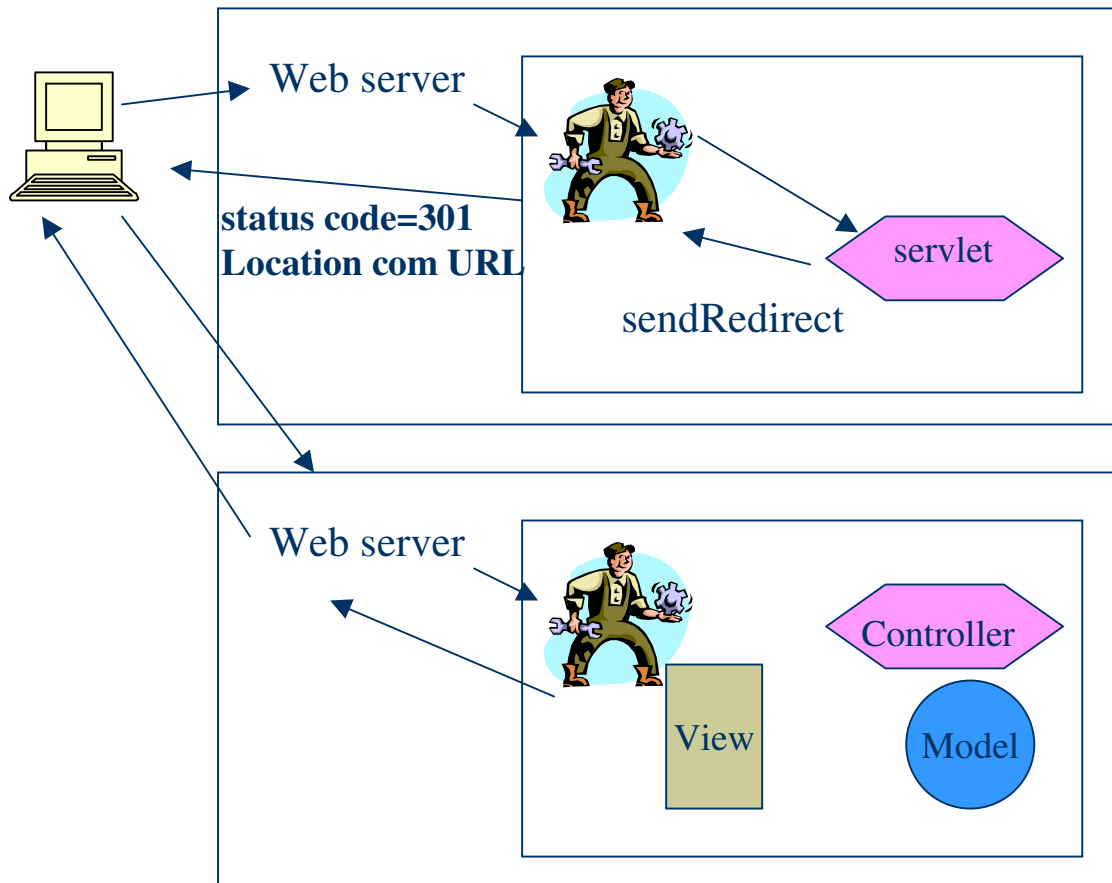


Redirecionando as respostas



- ◆ O *servlet* pode direcionar uma requisição para outro *servlet* ou para uma JSP.
- ◆ O *servlet* ou JSP destino pode residir em uma URL remota ou no mesmo *container*.
- ◆ O recurso remoto não tem acesso aos objetos *request* e *response* do *servlet* original.

Redireccionando para outra URL



Redirecionando para outra URL

♦ O que escrevo para redirecionar?

- `response.sendRedirect("http://www.nce.ufrj.br");`

ou

- `response.sendRedirect("/OutraAplicacao/Sugestao")`
 - Desvia para `http://www.nce.ufrj/ OutraAplicacao/Sugestao`;
 - ♦ A "/" significa conectar-se à raiz (outra webapps).

ou

- `response.sendRedirect("FechaCompra/Sugestao");`
 - Conecta-se à webapps original.

♦ Obs.: A URL do novo destino é apresentada no *web browser*.

Redirecionando para o mesmo local

- ◆ O que escrevo para redirecionar?
 - `RequestDispatcher vista = request.getRequestDispatcher("sugestao.jsp");`
 - `vista.forward(request,response);`
- ◆ O *web browser* desconhece este redirecionamento.



ServletConfig



- ◆ Objeto criado pelo *container* e utilizado para passar parâmetros de iniciação para um *servlet*.
- ◆ Parâmetros são definidos no web-xml.
- ◆ Evita a inserção de valores, passíveis de alterações, nos *servlets*.
- ◆ Para ativar uma nova versão web-xml é só fazer um *redploy* no *container*.
- ◆ Existe apenas um por cada *servlet*.
- ◆ Não pode ser alterado.



ServletConfig



- ◆ Oferece alguns dos seguintes métodos:
 - **getInitParameter(String)**
 - Retorna o conteúdo de um parâmetro específico.
 - **Enumeration getInitParameterNames()**
 - Retorna um conjunto com os nomes dos parâmetros especificados.

ServletConfig

◆ Especificando no web-xml:

```
<servlet>
  <servlet-name>Musicas</servlet-name>
  <servlet-class>com.exemplo.web.Recomendacao</servlet-class>
  <init-param>
    <param-name>faleConosco</param-name>
    <param-value>centralatendimento@nce.ufrj.br</param-value>
  </init-param>
  <init-param>
    <param-name>areaVendas</param-name>
    <param-value>vendasatendimento@nce.ufrj.br</param-value>
  </init-param>
</servlet>
```

• Obtendo no *servlet* ou JSP:

```
getServletConfig().getInitParameter("faleConosco");
getServletConfig().getInitParameter("areaVendas");
```



ServletContext



- ◆ Reflete o ambiente onde o *servlet* é executado.
- ◆ Criado pelo *container* para cada aplicativo *web* existente.
- ◆ Utilizado para os *servlets* compartilharem informações.
- ◆ Independe de sessão.
- ◆ Suporta atributos que podem ser modificados ou recuperados pelos *servlets* ou JSPs.



ServletContext



- ◆ Permite a declaração de parâmetros no web-xml.
- ◆ Estes parâmetros podem ser recuperados, em qualquer instante, pelos *servlets* ou JSPs.
- ◆ Lembrete: Atributos retornam um *Object* e parâmetros retornam um *String*.

ServletContext

- ◆ Oferece alguns dos seguintes métodos:
 - **getAttributeNames()**
 - Retorna um conjunto com os nomes dos atributos armazenados.
 - **getAttribute(String)**
 - Retorna um atributo específico do contexto.
 - **setAttribute(String, Object)**
 - Armazena um atributo no contexto
 - **removeAttribute(String)**
 - Remove um atributo do contexto.



ServletContext



- **getInitParameter(String)**
 - Retorna o conteúdo de um parâmetro específico.
- **Enumeration getInitParameterNames()**
 - Retorna um conjunto com os nomes dos parâmetros especificados.
- **getRequestDispatcher(String)**
 - Desvia para um recurso local.

ServletContext

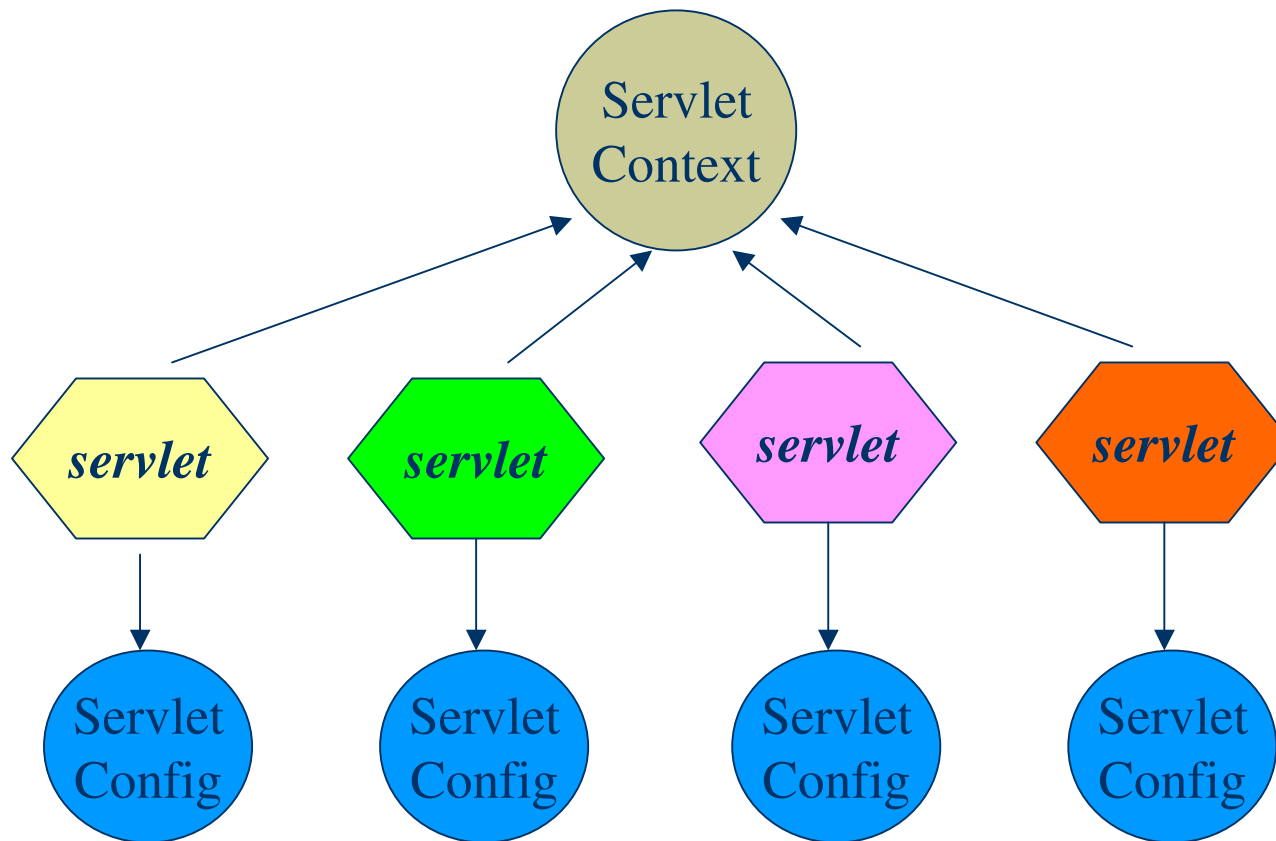
◆ Especificando no web-xml:

```
<servlet>
  <servlet-name>Musicas</servlet-name>
  <servlet-class>com.exemplo.web.Recomendacao</servlet-class>
  <init-param>
    <param-name>faleConosco</param-name>
    <param-value>centralatendimento@nce.ufrj.br</param-value>
  </init-param>
  <init-param>
    <param-name>areaVendas</param-name>
    <param-value>vendasatendimento@nce.ufrj.br</param-value>
  </init-param>
</servlet>
<context-param>
  <param-name>enderecoReal</param-name>
  <param-value>Avenida Rio Branco 156</param-value>
</context-param>
```


ServletContext

- Obtendo um parâmetro pelo *servlet* ou JSP:
`getServletContext().getInitParameter("endereçoReal");`
- Criando um atributo pelo *servlet* ou JSP:
`getServletContext().setAttribute("endereco", "Avenida Rio Branco 156");`
- Obtendo o atributo pelo *servlet* ou JSP:
`getServletContext().getAttribute("endereco");`

ServletConfig e ServletContext



O ServletContextListener

- ◆ Permite que a aplicação seja notificada quando um objeto **ServletContext** é **criado** ou **destruído**.
- ◆ Controla os eventos do ciclo de vida do contexto.
- ◆ Aplicação pode recuperar os parâmetros de iniciação do ServletContext.
- ◆ Acesso ao ServletContext é feito através do objeto **ServletContextEvent**.
- ◆ Inserido no diretório acima do *web* e do *model*.



O ServletContextListener



- ◆ Possui dois métodos:
 - **contextInitialized(ServletContextEvent evento)**
 - Acionado quando o contexto é criado.
 - **contextDestroyed(ServletContextEvent evento)**
 - Acionado quando o contexto é destruído.

O ServletContextListener

- Especificando no web-xml:

```
<web-app>
  <servlet>
    <servlet-name>Musicas</servlet-name>
    <servlet-class>com.exemplo.web.Recomendacao</servlet-class>
    <init-param>
      <param-name>faleConosco</param-name>
      <param-value>centralatendimento@nce.ufrj.br</param-value>
    </init-param>
  </servlet>
  <context-param>
    <param-name>enderecoReal</param-name>
    <param-value>Avenida Rio Branco 156</param-value>
  </context-param>
  <listener>
    <listener-class>
      com.exemplo.IniciaMinhaAplicacao
    </listener-class>
  </listener>
</web-app>
```

O ServletContextListener

```
import javax.servlet.ServletContext;
import javax.servlet.ServletContextListener;
import javax.servlet.ServletContextEvent;

public class AppLifecycleEvent implements ServletContextListener {

    public void contextInitialized(ServletContextEvent sce) {
        System.out.println("Initializing Application !");
        // Load the JDBC driver
        try {
            Class.forName("org.gjt.mm.mysql.Driver ");
        }
        catch (ClassNotFoundException e) {
            System.out.println(e.toString());
        }

        // Get the ServletContext object
        ServletContext servletContext = sce.getServletContext();

        // Set a ServletContext attribute
        servletContext.setAttribute("dbUrl", "jdbc:mysql:///Fred");
        System.out.println("Application initialized");
    }

    public void contextDestroyed(ServletContextEvent cse) {
        System.out.println("Application shut down");
    }
}
```

Código extraído do livro: Java para Web com Servlets, JSP e EJB, de Budi Kurniawan.

Servlet obtendo acesso ao contexto

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

public class ApplicationEventDemoServlet 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>");
        out.println("<TITLE>Application Event Demo Servlet</TITLE>");
        out.println("</HEAD>");
        out.println("<BODY>");
        out.println("Your database connection is ");

        // get the ServletContext object
        ServletContext servletContext = getServletContext();
        // display the "dbUrl" attribute
        out.println(servletContext.getAttribute("dbUrl"));

        out.println("</BODY>");
        out.println("</HTML>");
    }
}
```

Código extraído do livro: Java para Web com Servlets, JSP e EJB, de Budi Kurniawan.

Fluxo de execução

1. *Container* é iniciado e lê o web-xml.
2. *Container* cria o `ServletContext` da aplicação.
3. *Container* cria pares nome-valor dos parâmetros de iniciação do Contexto(se houver).
4. *Container* dá ao `ServletContext` referências para estes pares.
5. *Container* cria uma instância do *listener* da aplicação *web*.
6. Método `contextInitialized()` é acionado e obtém a referência para o `ServletContext`.
7. Código do método `contextInitialized()` cria os atributos de interesse.



Outros *listeners*



- ◆ ServletContextAttributeListener
- ◆ ServletRequestListener
- ◆ ServletRequestAttributeListener
- ◆ HttpSessionListener
- ◆ HttpSessionBindingListener
- ◆ HttpSessionAttributeListener
- ◆ HttpSessionActivationListener



Exercício



- ♦ Criar uma aplicação *web* que valide um *login* e uma senha.
- ♦ O *login* deverá ser: cursoJavaWeb
- ♦ E a senha : 123456
- ♦ Se entrada válida: apresenta o *login* e a senha para o usuário
- ♦ Se não OK: diz qual foi o erro.



O ServletContextAttributeListener



- ◆ Permite que a aplicação seja notificada quando um atributo do ServletContext é criado, modificado ou removido.

O ServletContextAttributeListener

◆ Possui os métodos:

- **attributeAdded(ServletContextAttributeEvent evento)**
 - Acionado quando um atributo é criado.
- **attributeRemoved(ServletContextAttributeEvent evento)**
 - Acionado quando um atributo é removido.
- **attributeReplaced(ServletContextAttributeEvent evento)**
 - Acionado quando um atributo é substituído.

ServletContextAttributeListener - exemplo

- ◆ Contador de visitas a um *site*.
 - Quando o contexto é iniciado a classe *listener* lê o conteúdo de um arquivo txt, que contém o número de visitas ao *site*, e cria um atributo com este valor no ServletContext.
 - Quando o *site* é visitado um *servlet* incrementa o contador modificando este atributo.
 - A classe *listener* “ouve” a modificação do atributo e atualiza o valor no arquivo txt.

O ServletContextAttributeListener

```
import javax.servlet.ServletContext;
import javax.servlet.ServletContextListener;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextAttributeListener;
import javax.servlet.ServletContextAttributeEvent;
import java.io.*;

public class AppAttributeEventDemo
    implements ServletContextListener, ServletContextAttributeListener {

    int counter;
    String counterFilePath = "C:\\counter.txt";

    public void contextInitialized(ServletContextEvent cse) {
        try {
            BufferedReader reader = new
                BufferedReader(new FileReader(counterFilePath));
            counter = Integer.parseInt( reader.readLine() );
            reader.close();
            System.out.println("Reading" + counter);

        }
        catch (Exception e) {
            System.out.println(e.toString());
        }

        ServletContext servletContext = cse.getServletContext();

        servletContext.setAttribute("pageCounter", Integer.toString(counter));
        System.out.println("Application initialized");
    }

    public void contextDestroyed(ServletContextEvent cse) {
        System.out.println("Application shut down");
    }

    public void attributeAdded(ServletContextAttributeEvent scae) {
        System.out.println("ServletContext attribute added");
    }

    public void attributeRemoved(ServletContextAttributeEvent scae) {
        System.out.println("ServletContext attribute removed");
    }
}
```

O ServletContextAttributeListener

```
public void attributeReplaced(ServletContextAttributeEvent scae) {
    System.out.println("ServletContext attribute replaced");
    writeCounter(scae);
}

synchronized void writeCounter(ServletContextAttributeEvent scae) {
    ServletContext servletContext = scae.getServletContext();

    counter = Integer.parseInt((String)
        servletContext.getAttribute("pageCounter"));

    try {
        BufferedWriter writer = new
            BufferedWriter(new FileWriter(counterFilePath));
        writer.write(Integer.toString(counter));
        writer.close();
        System.out.println("Writing");
    }
    catch (Exception e) {
        System.out.println(e.toString());
    }
}
```

Código extraído do livro: Java para Web com Servlets, JSP e EJB, de Budi Kurniawan.

O ServletContextAttributeListener

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

public class PageCounterServlet 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>");
        out.println("<TITLE>Page Counter</TITLE>");
        out.println("</HEAD>");
        out.println("<BODY>");
        ServletContext servletContext = getServletContext();
        int pageCounter = Integer.parseInt((String)
            servletContext.getAttribute("pageCounter"));
        pageCounter++;
        out.println("You are visitor number " + pageCounter);
        servletContext.setAttribute("pageCounter",
            Integer.toString(pageCounter));

        out.println("</BODY>");
        out.println("</HTML>");
    }
}
```

Código extraído do livro: Java para Web com Servlets, JSP e EJB, de Budi Kurniawan.

Distribuição descritiva

```
<web-app>
  <listener>
    <listener-class> AppAttributeEventDemo </listener-class>
  </listener>
  <servlet>
    <servlet-name> PageCounter </servlet-name>
    <servlet-class> PageCounterServlet </servlet-class>
  </servlet>
</web-app>
```

A interface ServletRequest

- ◆ Oferece alguns dos seguintes métodos:

- **getServerPort()**

- Obtém a porta onde o servidor está ouvindo o meio.

- **getServerName()**

- Obtém o nome do servidor.

- **getProtocol()**

- Obtém o protocolo utilizado.

- **getRemoteAddr()**

- Obtém o endereço do cliente remoto.

- **getRemoteHost()**

- Obtém o nome da máquina remota.



A interface ServletRequest



- **getParameter(String)**
 - Obtém o valor de um parâmetro específico.
- **getParameterValues(String)**
 - Obtém um *array* com os valores dos parâmetros passados.
- **getParameterNames()**
 - Obtém os nomes dos parâmetros passados.

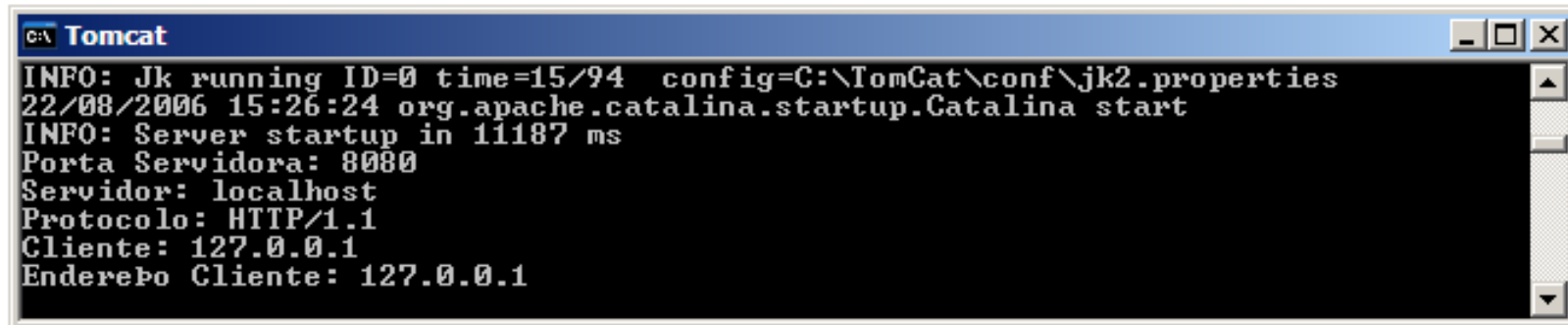
A interface ServletRequest

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

public class ServletRequestTeste extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response) throws IOException, ServletException {
        doPost(request, response);
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response) throws IOException,
    ServletException{
        System.out.println("Porta Servidora: " + request.getServerPort());
        System.out.println("Servidor: " + request.getServerName());
        System.out.println("Protocolo: " + request.getProtocol());
        System.out.println("Cliente: " + request.getRemoteHost());
        System.out.println("Endereço Cliente: " + request.getRemoteAddr());
    }
}
```

A interface ServletRequest

A screenshot of a Windows command window titled "C:\ Tomcat". The window displays the following text:

```
INFO: Jk running ID=0 time=15/94  config=C:\TomCat\conf\jk2.properties
22/08/2006 15:26:24 org.apache.catalina.startup.Catalina start
INFO: Server startup in 11187 ms
Porta Servidora: 8080
Servidor: localhost
Protocolo: HTTP/1.1
Cliente: 127.0.0.1
Endereço Cliente: 127.0.0.1
```

The window has standard Windows window controls (minimize, maximize, close) in the top right corner and a vertical scrollbar on the right side.



O ServletRequestListener



- ◆ Permite que a aplicação seja notificada quando um *request* é criado ou destruído.

O ServletRequestListener

- ◆ Possui os seguintes métodos:
 - **requestInitialized(ServletRequestEvent evento)**
 - Acionado quando o *request* é criado.
 - **requestDestroyed(ServletRequestEvent evento)**
 - Acionado quando o *request* é destruído.

A interface HttpServletRequest

- ◆ Define métodos para tratar cabeçalhos:
 - **getHeaderNames()**
 - Retorna um Enumeration contendo os nomes dos cabeçalhos.
 - **getHeader(String)**
 - Retorna o valor de um cabeçalho específico.
- ◆ Obs.: Suportado apenas pelo **doGet**.

A interface HttpServletRequest

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

public class HeaderRequest extends HttpServlet {

    public void doGet ( HttpServletRequest request,
                      HttpServletResponse response )
                      throws ServletException, IOException {

        response.setContentType("text/html");

        PrintWriter out = response.getWriter();

        out.println("<h2>" + "Obtendo o Cabeçalho" + "</h2>");

        Enumeration enumeration = request.getHeaderNames();

        while (enumeration.hasMoreElements()) {
            String header = (String) enumeration.nextElement();
            out.println("<b>" + header + ":" + request.getHeader(header) + "</b>" + "<BR>");
        }

    }
}
```

Código extraído do livro: Java para Web com Servlets, JSP e EJB, de Budi Kurniawan.

A interface HttpServletRequest

Obtendo o Cabeçalho

host:localhost:8080

user-agent:Mozilla/5.0 (Windows; U; Windows NT 5.0; pt-BR; rv:1.7.2) Gecko/20040803

accept:text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5

accept-language:pt-br,pt;q=0.5

accept-encoding:gzip,deflate

accept-charset:ISO-8859-1,utf-8;q=0.7,*;q=0.7

keep-alive:300

connection:keep-alive

A interface HttpServletRequest

- ◆ Métodos para obter informações de parâmetros:
 - **getQueryString()**
 - informa os parâmetros contidos na URL. Compete ao desenvolvedor promover a separação. **Suportado apenas pelo doGet().**
- **Obs.:** Herda métodos da interface ServletRequest:
 - **getParameterNames(), getParameter(String), getParameterValues(), getAttribute(), setAttribute() etc.**

A interface HttpServletRequest

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

```
public class ValidaUsuarioSenhaMobile extends HttpServlet {
```

```
    public void doPost(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException  
    {
```

```
        String usuario = request.getParameter("usuario");  
        String senha = request.getParameter("senha");
```

```
        .....
```

```
    }  
}
```

A interface HttpServletRequest

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

public class ParameterRequest extends HttpServlet {

    public void doGet ( HttpServletRequest request,
                       HttpServletResponse response )
                       throws ServletException, IOException {

        response.setContentType("text/html");

        PrintWriter out = response.getWriter();

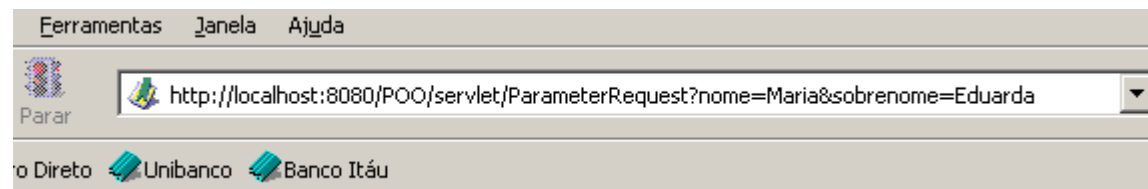
        out.println("<h2>" + "Separando os Parâmetros" + "</h2>");

        out.println("<b>" + "Nome:" + request.getParameter("nome") + "</b>" + "<BR>");
        out.println("<b>" + "SobreNome:" + request.getParameter("sobrenome") + "</b>" + "<BR>");

    }
}
```

Código extraído do livro: Java para Web com Servlets, JSP e EJB, de Budi Kurniawan.

A interface HttpServletRequest



Separando os Parâmetros

Nome:Maria
SobreNome:Eduarda

- Os nomes dos campos são *case sensitive*.

A interface HttpServletRequest

- ◆ Parâmetros com múltiplos valores:
 - Utilizado quando um parâmetro possui diversos valores. Exs: ListBox e CheckBox.
 - O método **getParameter** só fornece o primeiro valor do parâmetro.
 - O método **getParameterValues** retorna um *array* de *strings* contendo todos os valores selecionados.
 - O nome do parâmetro é o argumento para o método **getParameterValues**.

A interface HttpServletRequest

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

public class ParameterRequest extends HttpServlet {

    public void doPost (HttpServletRequest request,
                        HttpServletResponse response )
        throws ServletException, IOException {

        String[] values = request.getParameterValues("musicasFavoritas");
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        if (values != null) {
            int length = values.length;
            out.println("<h2>" + "Você Selecionou: " + "</h2>");
            for (int i=0; i<length; i++) {
                out.println("<BR>" + values[i]);
            }
        }
    }
}
```

Código extraído e modificado do livro: Java para Web com Servlets, JSP e EJB, de Budi Kurniawan.

A interface HttpServletRequest

Selezione sua música favorita:

- ☐ Rock
- ☐ Jazz
- ☐ Pagode
- ☐ MPB

Submit

Você selecionou:

Rock

MPB

A interface HttpServletRequest

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

public class HttpRequestDemoServlet 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>");
        out.println("<TITLE>Obtendo Parâmetros com Múltiplos Valores</TITLE>");
        out.println("</HEAD>");
        out.println("<BODY>");

        out.println("<BR>");
        out.println("<BR>Selecione sua música favorita:");
        out.println("<BR><FORM METHOD=POST>");
        out.println("<BR><INPUT TYPE=CHECKBOX " +
            "NAME=musicaFavorita VALUE=Rock>Rock");
        out.println("<BR><INPUT TYPE=CHECKBOX " +
            "NAME=musicaFavorita VALUE=Jazz>Jazz");
        out.println("<BR><INPUT TYPE=CHECKBOX " +
            "NAME=musicaFavorita VALUE=Pagode>Pagode");
        out.println("<BR><INPUT TYPE=CHECKBOX " +
            "NAME=musicaFavorita VALUE=MPB>MPB");
        out.println("<BR><INPUT TYPE=SUBMIT VALUE=Submit>");
        out.println("</FORM>");
        out.println("</BODY>");
        out.println("</HTML>");
    }
}
```

A interface HttpServletRequest

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    String[] values = request.getParameterValues("musicaFavorita");
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    if (values != null ) {
        int length = values.length;
        out.println("Você selecionou: ");
        for (int i=0; i<length; i++) {
            out.println("<BR>" + values[i]);
        }
    }
}
```

Gerenciando sessões

- ♦ Gerenciamento de sessão:
 - É o acompanhamento dos movimentos realizados pelos usuários em um *web site*.
- ♦ O HTTP é um protocolo que não guarda estados:
 - 1o. HTTP *request* – o *browser* requisita a página.
 - 1o HTTP *response* – o servidor retorna a página requisitada e quebra a conexão (*stateless protocol*).
 - HTTP *requests* seguintes – o *browser* requisita a página. O servidor não tem como associar o *browser* com um *request* prévio.



Técnicas para o gerenciamento de sessões



- ◆ Existem 4 técnicas para o gerenciamento de sessões:
 - Objetos de sessão.
 - Cookies.
 - Reescrita de URL.
 - Campos ocultos.



Como trabalhar com sessões



- ◆ Quando surge um novo cliente, o *container* cria um objeto sessão que fica associado somente a este cliente.
- ◆ Este objeto fica disponível enquanto o cliente estiver ativo.
- ◆ Funciona como uma *Hashtable* onde pode ser armazenada qualquer quantidade de pares chave/objeto.
- ◆ As sessões terminam por tempo de inatividade ou quando o usuário sai do *browser*.
- ◆ Pode ser acessado por qualquer *servlet* do mesmo aplicativo.
- ◆ Para recuperar um objeto previamente armazenado basta informar a sua chave.

Como o Java mantém sessões

- ◆ No primeiro *request* do cliente o *container* gera uma identificação única para a sessão.
- ◆ No *response*, esta identificação é retornada para o cliente.
- ◆ Em cada *request* subsequente o *browser* envia a identificação.
- ◆ O *container* recebe e associa a identificação à sessão do cliente.
- ◆ É utilizado um *cookie* para armazenar a identificação.

Como o Java mantém sessões

- ◆ O *cookie* é enviado no cabeçalho do protocolo HTTP.
- ◆ Cabe ao *servlet* somente informar ao *container* que quer criar, ou usar, o recurso sessão.
- ◆ Cabe ao *container* criar o objeto *cookie*.
- ◆ Se os *cookies* são inibidos no *web browser*, o gerenciamento de sessão não funcionará.

Como trabalhar com sessões

- ◆ Obtendo um objeto sessão:
 - `HttpSession minhasessao=request.getSession();`
- ◆ Nos *servlets*, sempre deve ser especificado
- ◆ Pode ser obtido no método `init()`.
- ◆ Pode ser utilizado nos métodos **doGet** ou **doPost**.
- ◆ Método **getSession()** definido na interface `HttpServletRequest`.

Como trabalhar com sessões

- ◆ Verificando se a sessão é nova:
 - **HttpSession session = request.getSession();**
 - **session.isNew()**
 - Retorna *true* se o cliente ainda não retornou com a sua identificação.
- ◆ Outra forma:
 - **HttpSession session = request.getSession(false);**
 - Retorna uma sessão pré-existente ou *null*.



Métodos da interface HttpSession



- **getAttribute(String)**
 - Retorna um atributo específico da sessão.
- **setAttribute(String, Object)**
 - Armazena um atributo na sessão.
- **removeAttribute(String)**
 - Remove um atributo da sessão.
- **getCreationTime()**
 - Retorna a hora que a sessão foi criada.
- **getId()**
 - Retorna uma string contendo o identificador da sessão.

Métodos da interface HttpSession

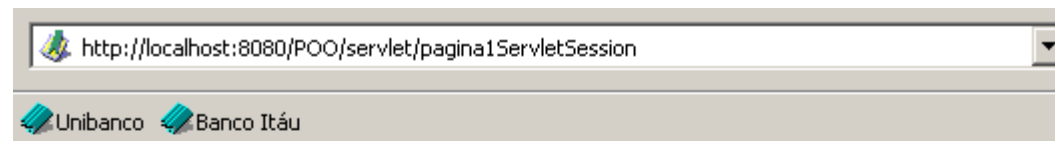
- ◆ **getLastAccessedTime()**
 - Retorna a hora em que o *container* atendeu o último *request* para uma sessão.
- ◆ **setMaxInactiveInterval()**
 - Especifica o tempo máximo , em segundos, de espera entre *requests* de uma sessão.
 - *Default*=30 minutos; Se especificar -1 não sofre *timed out*.
- ◆ **getMaxInactiveInterval()**
 - Retorna o tempo máximo, em segundos, permitido entre *requests*.
- ◆ **invalidate()**
 - Encerra a sessão. Todos os atributos são removidos.

Especificando o *timeout* no web-xml

```
<session-config>  
    <session-timeout>20</session-timeout>  
</session-config>
```

Obs.: Este tempo é em minutos.

HttpSession - exemplo

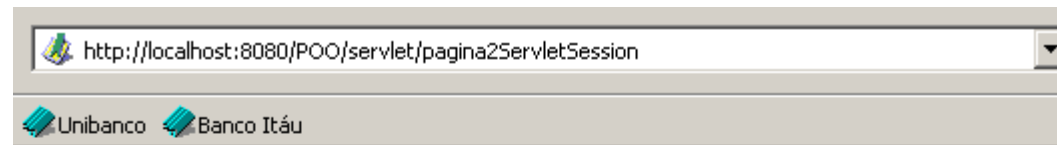


Página 1

Por favor forneça seu nome e sobrenome.

Primeiro Nome	<input type="text" value="joão"/>
Sobrenome	<input type="text" value="silva"/>
<input type="button" value="Restaurar valores"/>	<input type="button" value="OK"/>

HttpSession - exemplo

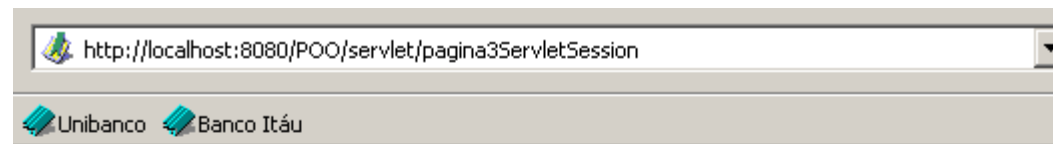


Página 2 - Sessão

Por favor entre com o login desejado e senha.

Login	<input type="text" value="jsilva"/>
Senha	<input type="password" value="*****"/>
<input type="button" value="Restaurar valores"/>	<input type="button" value="OK"/>

HttpSession - exemplo



Página 3 - Sessão

Você forneceu os seguintes valores.

Nome: joão
SobreNome: silva
Login: jsilva
Senha: 654321

HttpSession - exemplo

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

public class pagina2ServletSession extends HttpServlet {
    String page1Url = "pagina1ServletSession";
    String firstName;
    String lastName;

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.sendRedirect(page1Url);
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        firstName = request.getParameter("firstName");
        lastName = request.getParameter("lastName");
        if (firstName==null || lastName==null)
            response.sendRedirect(page1Url);
        sendPage2(request, response);
    }

    void sendPage2(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        HttpSession minhasessao = request.getSession();
        minhasessao.setAttribute("nome", firstName);
        minhasessao.setAttribute("sobrenome", lastName);
    }
}
```

HttpSession - exemplo

```
response.setContentType("text/html");
PrintWriter out = response.getWriter();
out.println("<HTML>");
out.println("<HEAD>");
out.println("<TITLE>Página 2 - Sessão</TITLE>");
out.println("</HEAD>");
out.println("<BODY>");
out.println("<CENTER>");
out.println("<H2>Página 2 - Sessão</H2>");
out.println("<BR>");
out.println("<BR>");
out.println("Por favor entre com o login desejado e senha.");
out.println("<BR>");
out.println("<BR>");
out.println("<FORM METHOD=POST ACTION=pagina3ServletSession>");
out.println("<TABLE>");
out.println("<TR>");
out.println("<TD>Login&nbsp;</TD>");
out.println("<TD><INPUT TYPE=TEXT NAME=username></TD>");
out.println("</TR>");
out.println("<TR>");
out.println("<TD>Senha&nbsp;</TD>");
out.println("<TD><INPUT TYPE=PASSWORD NAME=password></TD>");
out.println("</TR>");
out.println("<TR>");
out.println("<TD><INPUT TYPE=RESET></TD>");
out.println("<TD><INPUT TYPE=SUBMIT VALUE=OK></TD>");
out.println("</TR>");
out.println("</TABLE>");
out.println("</FORM>");
out.println("</CENTER>");
out.println("</BODY>");
out.println("</HTML>");
}
```

. Código extraído e modificado do livro: Java para Web com Servlets, JSP e EJB, de Budi Kurniawan.

HttpSession - exemplo

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

public class pagina3ServletSession extends HttpServlet {
    String page1Url = "pagina1ServletSession";
    String firstName;
    String lastName;
    String userName;
    String password;

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.sendRedirect(page1Url);
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        userName = request.getParameter("userName");
        password = request.getParameter("password");
        if (userName==null || password==null)
            response.sendRedirect(page1Url);
        displayValues(request, response);
    }

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

        HttpSession minhasessao = request.getSession();
        firstName=(String) minhasessao.getAttribute("nome");
        lastName=(String) minhasessao.getAttribute("sobrenome");
    }
}
```

HttpSession - exemplo

```
response.setContentType("text/html");
PrintWriter out = response.getWriter();
out.println("<HTML>");
out.println("<HEAD>");
out.println("<TITLE>Página 3 - Sessão</TITLE>");
out.println("</HEAD>");
out.println("<BODY>");
out.println("<CENTER>");
out.println("<H2>Página 3 - Sessão</H2>");
out.println("<BR>");
out.println("<BR>");
out.println("Você forneceu os seguintes valores.");
out.println("<BR>");
out.println("<BR>");
out.println("<TABLE>");
out.println("<TR>");
out.println("<TD>Nome: &nbsp;</TD>");
out.println("<TD>" + firstName + "</TD>");
out.println("</TR>");
out.println("<TR>");
out.println("<TD>SobreNome: &nbsp;</TD>");
out.println("<TD>" + lastName + "</TD>");
out.println("</TR>");
out.println("<TR>");
out.println("<TD>Login: &nbsp;</TD>");
out.println("<TD>" + userName + "</TD>");
out.println("</TR>");
out.println("<TR>");
out.println("<TD>Senha: &nbsp;</TD>");
out.println("<TD>" + password + "</TD>");
out.println("</TR>");
out.println("</TABLE>");
out.println("</CENTER>");
out.println("</BODY>");
out.println("</HTML>");
}
```

Código extraído e modificado do livro: Java para Web com Servlets, JSP e EJB, de Budi Kurniawan.



O HttpSessionListener



- ◆ A aplicação é notificada quando um objeto HttpSession é criado ou invalidado.
- ◆ Permite verificar quantos usuários estão correntemente ativos.



O HttpSessionListener



- ◆ Possui dois métodos:
 - **sessionCreated(HttpSessionEvent evento)**
 - Acionado quando o objeto HttpSession é criado.
 - **sessionDestroyed(HttpSessionEvent evento)**
 - Acionado quando o objeto HttpSession é invalidado.



O HttpSessionListener – exemplo



- ◆ Contador de usuários em sessão:
 - Conta o número de usuários que estão correntemente em sessão.
 - Incrementado quando uma sessão é estabelecida.
 - Decrementado quando uma sessão é invalidada/encerrada.

O HttpSessionListener – exemplo

```
import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpSessionListener;
import javax.servlet.http.HttpSessionEvent;
import javax.servlet.ServletContextListener;
import javax.servlet.ServletContext;
import javax.servlet.ServletContextEvent;

public class SessionLifecycleEventDemo
    implements ServletContextListener, HttpSessionListener {
    ServletContext servletContext;
    int counter;
    public void contextInitialized(ServletContextEvent sce) {
        servletContext = sce.getServletContext();
        servletContext.setAttribute(("userCounter"), Integer.toString(counter));
    }

    public void contextDestroyed(ServletContextEvent sce) {
    }

    public void sessionCreated(HttpSessionEvent hse) {
        System.out.println("Session created.");
        incrementUserCounter();
    }
    public void sessionDestroyed(HttpSessionEvent hse) {
        System.out.println("Session destroyed.");
        decrementUserCounter();
    }

    synchronized void incrementUserCounter() {
        counter = Integer.parseInt(
            (String)servletContext.getAttribute("userCounter"));
        counter++;
        servletContext.setAttribute(("userCounter"), Integer.toString(counter));
        System.out.println("User Count: " + counter);
    }
}
```


O HttpSessionListener – exemplo

```
synchronized void decrementUserCounter() {  
    int counter = Integer.parseInt(  
        (String)servletContext.getAttribute("userCounter"));  
    counter--;  
    servletContext.setAttribute(("userCounter"), Integer.toString(counter));  
    System.out.println("User Count: " + counter);  
}
```

O HttpSessionListener – exemplo

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

public class UserCounterServlet extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        ServletContext servletContext = getServletContext();
        HttpSession session = request.getSession(true);
        int userCounter = 0;
        userCounter =
            Integer.parseInt((String)servletContext.getAttribute("userCounter"));

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<HTML>");
        out.println("<HEAD>");
        out.println("<TITLE>User Counter</TITLE>");
        out.println("</HEAD>");
        out.println("<BODY>");

        out.println("There are " + userCounter + " users.");
        out.println("</BODY>");
        out.println("</HTML>");
    }
}
```



O HttpSessionBindListener



- ◆ Permite que o objeto colocado como atributo seja acionado em sua criação ou em sua remoção.
- ◆ Não precisa ser informado no web-xml.



O HttpSessionBindListener



- ◆ Possui os métodos:
 - **valueBound(HttpSessionBindingEvent evento)**
 - Acionado quando o objeto torna-se um atributo.
 - **valueUnbound(HttpSessionBindingEvent evento)**
 - Acionado quando o objeto é removido.



O HttpSessionBindListener



- ◆ O objeto em questão, quando acionado, pode obter acesso a uma base de dados e popular os seus atributos.
- ◆ Em tempo de remoção o objeto pode atualizar a base de dados com os seus atributos atualizados.

HttpSessionBindListener

```
package com.exemplo;
import javax.servlet.*;
import javax.servlet.http.*;

public class Aluno implements HttpSessionBindingListener{

    private int DRE;
    private String nome;
    private String curso;

    public Aluno(int DRE){
        this.DRE = DRE;
    }
    public String getNome(){
        return nome;
    }
    public String getCurso(){
        return curso;
    }
    public void setNome(String nome){
        this.nome = nome;
    }
    public void setCurso(String curso){
        this.curso = curso;
    }
}
```



HttpSessionBindingListener



```
public void valueBound(HttpSessionBindingEvent evento){  
    // instruções para obter os dados do banco  
}  
public void valueUnbound(HttpSessionBindingEvent evento){  
    // instruções para atualizar os dados do banco  
}  
}
```



O HttpSessionAttributeListener



- ◆ A aplicação é notificada quando um atributo do objeto HttpSession é criado, alterado ou removido.

O HttpSessionAttributeListener

- ◆ Possui os métodos:
 - **attributeAdded(HttpSessionBindingEvent evento)**
 - Acionado quando o atributo é adicionado.
 - **attributeRemoved(HttpSessionBindingEvent evento)**
 - Acionado quando o atributo é removido.
 - **attributeReplaced(HttpSessionBindingEvent evento)**
 - Acionado quando o atributo é modificado.
- ◆ Obs.: Os valores podem ser obtidos com `evento.getName()` e `evento.getValue()`.



Resumo dos *listeners*



- ◆ *Listeners* de atributos:
 - ServletRequestAttributeListener
 - ServletContextAttributeListener
 - HttpSessionAttributeListener
- ◆ *Listeners* de ciclos-de-vida
 - ServletRequestListener
 - ServletContextListener
 - HttpSessionListener
 - HttpSessionBindingListener
 - HttpSessionActivationListener

Resumo dos *listeners*

- ◆ Métodos comuns a todos os *listeners*(exceção para `BindingListener`):
 - `attributeAdded()`, `attributeRemoved()` e `attributeReplaced()`.
- ◆ Métodos do ciclo-de-vida de uma sessão:
 - `sessionCreated()` e `sessionDestroyed()`.
- ◆ Métodos do ciclo-de-vida de um *request*:
 - `requestInitialized()` e `requestDestroyed()`.
- ◆ Métodos do ciclo-de-vida de um *servlet context*:
 - `contextInitialized()` e `contextDestroyed()`.

Propósitos dos atributos

- ◆ Atributos do *servlet context(não thread-safe)*
 - Utilizados para serem compartilhados por toda a aplicação.
- ◆ Atributos do *HttpSession(não thread-safe)*
 - Utilizados para armazenar informações ligadas às sessões dos clientes.
- ◆ Atributos do *request(thread-safe)*
 - Utilizados para passar informações do servlet para a JSP.



Cookies



- ◆ São criados pelo *container* ou pelo *servlet*.
- ◆ É um par nome-valor utilizado para o servidor e o cliente estabelecerem uma forma de persistência de dados.
- ◆ No lado cliente o *browser* salva os *cookies* e os envia de volta para o servidor cada vez que solicitar uma página.
- ◆ São transferidos no cabeçalho HTTP.



Cookies



- ◆ Foram especificados pelo NetScape e fazem parte do padrão Internet.
- ◆ *Cookies* podem permanecer em um *browser* por até 3 anos.
- ◆ Normalmente um *browser* aceita até 20 *cookies* por *site* e 300 *cookies* no total, podendo possuir, cada *cookie*, até 4KB de tamanho.



Cookies



- ◆ Criado pela classe **Cookie** que faz parte do pacote `javax.servlet.http`.
- ◆ Construtor:
 - `Cookie(String cookieName, String cookieValue)`
- ◆ Enviado para o *browser* através do objeto *response*.
- ◆ Recuperado pelo objeto *request*.

Cookies

- ◆ Alguns dos métodos
 - **addCookie(nome-do-objeto)**
 - Adiciona um *cookie* ao *response*.
 - **Cookie[] getCookies()**
 - Obtém um string de *cookies* enviados pelo browser.
 - **String getName()**
 - Obtém o nome do *cookie*.
 - **String getValue()**
 - Obtém o valor do *cookie*.



Cookies

■ **setMaxAge(int expira)**

- Indica o tempo máximo, em segundos, de vida do *cookie*.
- Para criar um *cookie* persistente basta especificar um número >0.
- Para criar um *cookie* válido para uma única sessão deve ser especificado o valor -1 (valor *default*).

■ **setSecure(boolean flag)**

- Indica ao *browser* que o *cookie* deve somente ser enviado usando um protocolo seguro tal como HTTPS ou SSL.

Cookies - exemplos



Click para ver os cookies !!!

OK

Cookies - exemplos

Aqui estão todos os cabeçalhos.

host: localhost:8080
user-agent: Mozilla/5.0 (Windows; U; Windows NT 5.0; pt-BR; rv:1.7.2) Gecko/20040803
accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
accept-language: pt-br,pt;q=0.5
accept-encoding: gzip,deflate
accept-charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
keep-alive: 300
connection: keep-alive
referer: http://localhost:8080/POO/servlet/CookieServlet
cookie: nome_do_usuario=Maria; senha=Eduarda
content-type: application/x-www-form-urlencoded
content-length: 0

Aqui estão todos os cookies.

Nome do Cookie : nome_do_usuario
Valor do Cookie: Maria
Nome do Cookie : senha
Valor do Cookie: Eduarda

Cookies - exemplos

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

public class CookieServlet extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        Cookie c1 = new Cookie("nome_do_usuario", "Maria");
        Cookie c2 = new Cookie("senha", "Eduarda");
        response.addCookie(c1);
        response.addCookie(c2);

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<HTML>");
        out.println("<HEAD>");
        out.println("<TITLE>Teste de Cookies/TITLE>");
        out.println("</HEAD>");
        out.println("<BODY>");
        out.println("Click para ver os cookies !!!");
        out.println("<BR>");
        out.println("<FORM METHOD=POST>");
        out.println("<INPUT TYPE=SUBMIT VALUE=OK>");
        out.println("</FORM>");
        out.println("</BODY>");
        out.println("</HTML>");
    }
}
```

Cookies - exemplos

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<HTML>");
    out.println("<HEAD>");
    out.println("<TITLE>Teste de Cookies</TITLE>");
    out.println("</HEAD>");
    out.println("<BODY>");
    out.println("<H2>Aqui estão todos os cabeçalhos.</H2>");

    Enumeration enuma = request.getHeaderNames();
    while (enuma.hasMoreElements()) {
        String header = (String) enuma.nextElement();
        out.print("<B>" + header + "</B>: ");
        out.print(request.getHeader(header) + "<BR>");
    }

    out.println("<BR><BR><H2>Aqui estão todos os cookies.</H2>");
    Cookie[] cookies = request.getCookies();
    int length = cookies.length;
    for (int i=0; i<length; i++) {
        Cookie cookie = cookies[i];
        out.println("<B>Nome do Cookie :</B> " + cookie.getName() + "<BR>");
        out.println("<B>Valor do Cookie:</B> " + cookie.getValue() + "<BR>");
    }

    out.println("</BODY>");
    out.println("</HTML>");
}
```

Como apagar *cookies*

- Para apagar *cookies* basta atribuir o valor 0 para a idade do *cookie*.

```
Cookie[ ] cookies = request.getCookies();  
  
for (int=0; i<cookies.length; i++) {  
    Cookie cookie = cookies[i];  
    cookie.SetMaxAge(0);  
    reponse.addCookie(cookie);  
}
```

Reescrita de URL

- ◆ Consiste em anexar a identificação do cliente na URL.
- ◆ Utilizada quando o cliente não permite *cookies*.
- ◆ Feita automaticamente pelo *container*.
- ◆ Sintaxes:
 - `response.encodeURL("/Recurso")`
 - `response.encodeRedirectURL("/Recurso")`

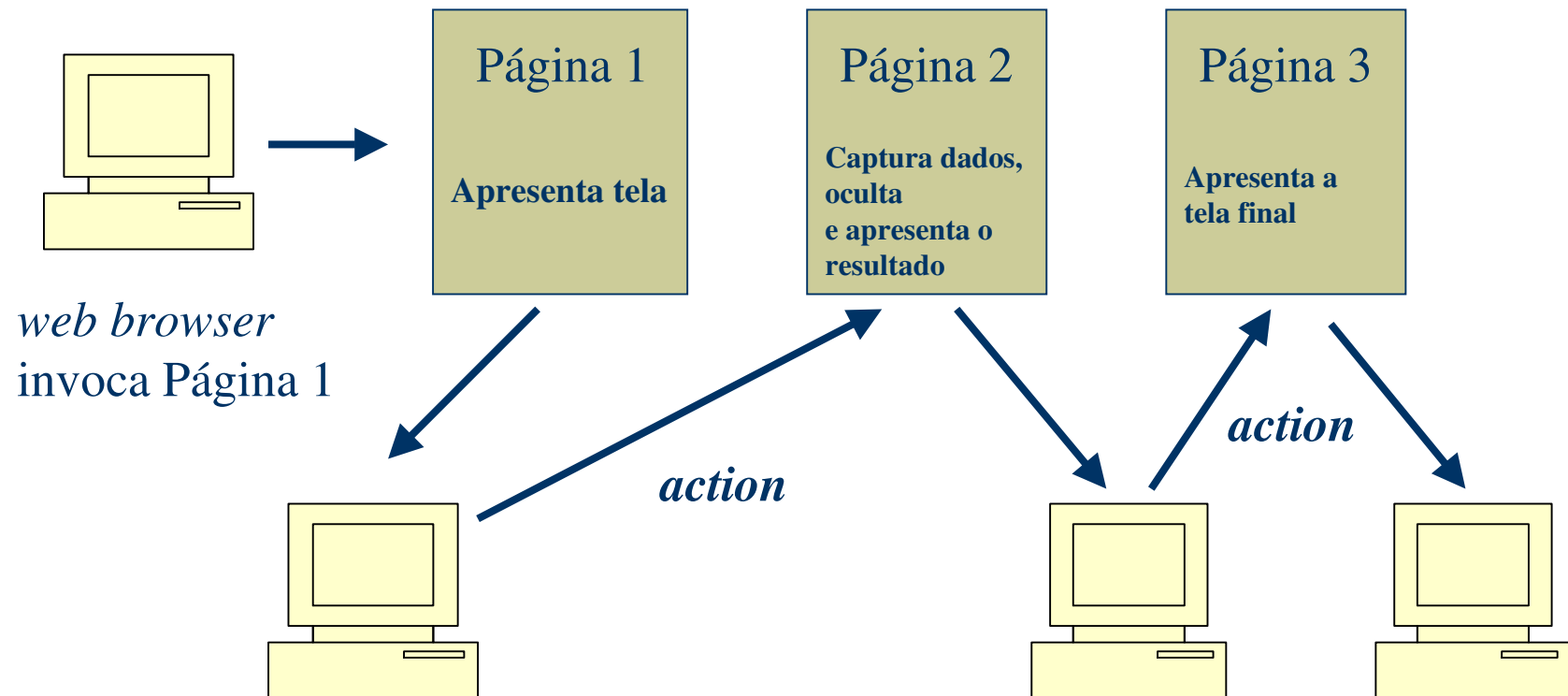


Campos ocultos

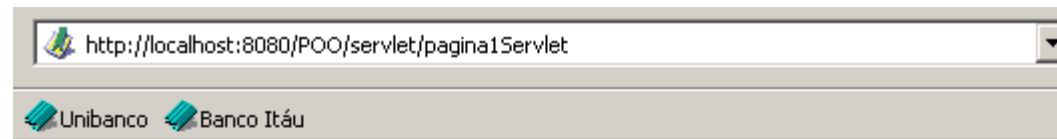


- ◆ Consiste em passar um identificador como um valor de um campo oculto.
- ◆ Valor não aparece na URL mas pode ser visto no código HTML.
- ◆ Utilizados quando o cliente não permite *cookies*.

Campos ocultos - exemplos



Campos ocultos - exemplos

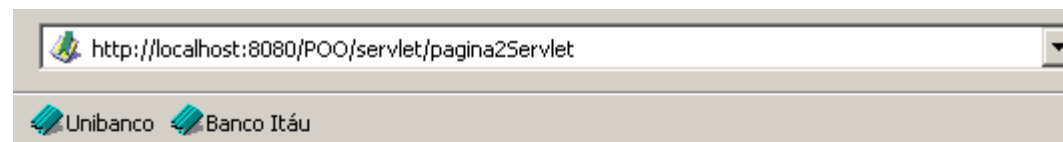


Página 1

Por favor forneça seu nome e sobrenome.

Primeiro Nome	<input type="text" value="Maria"/>
Sobrenome	<input type="text" value="Eduarda"/>
<input type="button" value="Restaurar valores"/> <input type="button" value="OK"/>	

Campos ocultos - exemplos

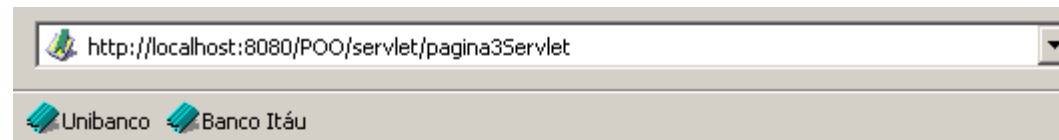


Página 2

Por favor entre com o login desejado e senha.

Login	<input type="text" value="meduarda"/>
Senha	<input type="password" value="XXXXXXXX"/>
<input type="button" value="Restaurar valores"/> <input type="button" value="OK"/>	

Campos ocultos - exemplos



Página 3

Você forneceu os seguintes valores.

Nome:	Maria
SobreNome:	Eduarda
Login:	meduarda
Senha:	123456

Campos ocultos - exemplos

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

public class pagina1Servlet extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        sendPagel(response);
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        sendPagel(response);
    }

    void sendPagel(HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<HTML>");
        out.println("<HEAD>");
        out.println("<TITLE>Página 1</TITLE>");
        out.println("</HEAD>");
        out.println("<BODY>");
        out.println("<CENTER>");
        out.println("<H2>Página 1</H2>");
        out.println("<BR>");
        out.println("<BR>");
        out.println("Por favor forneça seu nome e sobrenome.");
        out.println("<BR>");
        out.println("<BR>");
        out.println("<FORM METHOD=POST ACTION=pagina2Servlet>");
        out.println("<TABLE>");
        out.println("<TR>");
        out.println("<TD>Primeiro Nome&nbsp;</TD>");
        out.println("<TD><INPUT TYPE=TEXT NAME=firstName></TD>");
        out.println("</TR>");
```

Campos ocultos - exemplos

```
out.println("<TR>");
out.println("<TD>Sobrenome </TD>");
out.println("<TD><INPUT TYPE=TEXT NAME=lastName></TD>");
out.println("</TR>");
out.println("<TR>");
out.println("<TD><INPUT TYPE=RESET></TD>");
out.println("<TD><INPUT TYPE=SUBMIT VALUE=OK></TD>");
out.println("</TR>");
out.println("</TABLE>");
out.println("</FORM>");
out.println("</CENTER>");
out.println("</BODY>");
out.println("</HTML>");
}
}
```

Campos ocultos - exemplos

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

public class pagina2Servlet extends HttpServlet {
    String page1Url = "pagina1Servlet";
    String firstName;
    String lastName;

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.sendRedirect(page1Url);
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        firstName = request.getParameter("firstName");
        lastName = request.getParameter("lastName");
        if (firstName==null || lastName==null)
            response.sendRedirect(page1Url);
        sendPage2(response);
    }

    void sendPage2(HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<HTML>");
        out.println("<HEAD>");
        out.println("<TITLE>Página 2</TITLE>");
        out.println("</HEAD>");
        out.println("<BODY>");
        out.println("<CENTER>");
        out.println("<H2>Página 2</H2>");
        out.println("<BR>");
        out.println("<BR>");
        out.println("Por favor entre com o login desejado e senha.");
    }
}
```

Campos ocultos - exemplos

```
out.println("<BR>");
out.println("<BR>");
out.println("<FORM METHOD=POST ACTION=pagina3Servlet>");
out.println("<INPUT TYPE=HIDDEN NAME=firstName VALUE=\"\" + firstName + \"\">");
out.println("<INPUT TYPE=HIDDEN NAME=lastName VALUE=\"\" + lastName + \"\">");
out.println("<TABLE>");
out.println("<TR>");
out.println("<TD>Login&nbsp;</TD>");
out.println("<TD><INPUT TYPE=TEXT NAME=username></TD>");
out.println("</TR>");
out.println("<TR>");
out.println("<TD>Senha&nbsp;</TD>");
out.println("<TD><INPUT TYPE=PASSWORD NAME=password></TD>");
out.println("</TR>");
out.println("<TR>");
out.println("<TD><INPUT TYPE=RESET></TD>");
out.println("<TD><INPUT TYPE=SUBMIT VALUE=OK></TD>");
out.println("</TR>");
out.println("</TABLE>");
out.println("</FORM>");
out.println("</CENTER>");
out.println("</BODY>");
out.println("</HTML>");
}
```


Campos ocultos - ejemplos

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

public class pagina3Servlet extends HttpServlet {
    String pageUrl = "pagina1Servlet";
    String firstName;
    String lastName;
    String userName;
    String password;

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.sendRedirect(pageUrl);
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        firstName = request.getParameter("firstName");
        lastName = request.getParameter("lastName");
        userName = request.getParameter("userName");
        password = request.getParameter("password");
        if (firstName==null || lastName==null ||
            userName==null || password==null)
            response.sendRedirect(pageUrl);
        displayValues(response);
    }

    void displayValues(HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<HTML>");
        out.println("<HEAD>");
        out.println("<TITLE>Página 3</TITLE>");
        out.println("</HEAD>");
        out.println("<BODY>");
```

Campos ocultos - exemplos

```
out.println("</HEAD>");
out.println("<BODY>");
out.println("<CENTER>");
out.println("<H2>Página 3</H2>");
out.println("<BR>");
out.println("<BR>");
out.println("Você forneceu os seguintes valores.");
out.println("<BR>");
out.println("<BR>");
out.println("<TABLE>");
out.println("<TR>");
out.println("<TD>Nome: &nbsp;</TD>");
out.println("<TD>" + firstName + "</TD>");
out.println("</TR>");
out.println("<TR>");
out.println("<TD>SobreNome: &nbsp;</TD>");
out.println("<TD>" + lastName + "</TD>");
out.println("</TR>");
out.println("<TR>");
out.println("<TD>Login: &nbsp;</TD>");
out.println("<TD>" + userName + "</TD>");
out.println("</TR>");
out.println("<TR>");
out.println("<TD>Senha: &nbsp;</TD>");
out.println("<TD>" + password + "</TD>");
out.println("</TR>");
out.println("</TABLE>");
out.println("</CENTER>");
out.println("</BODY>");
out.println("</HTML>");
}
```

Código extraído do livro: Java para Web com Servlets, JSP e EJB, de Budi Kurniawan.

Como depurar problemas em *servlets*

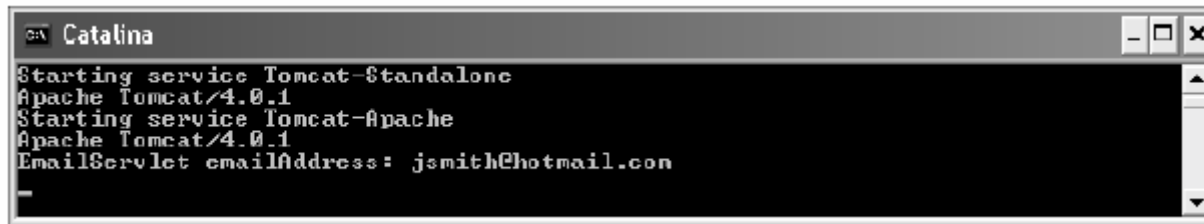
- ♦ O *servlet* não compila:
 - Certifique-se que o compilador tem acesso aos arquivos JAR para todas as APIs necessárias.
 - Certifique-se que o *classpath* está apontando para o diretório que contém o pacote correto.
- ♦ O *servlet* não executa:
 - Certifique-se que o *web server* está executando (Tomcat).
- ♦ As mudanças feitas não estão sendo apresentadas:
 - Certifique-se que a opção de *reloading*(no Tomcat) está ativada.

Como depurar problemas em *servlets*

- ◆ Impressão é feita na console do *container* (Tomcat).
- ◆ Podem ser utilizados os métodos *println*, *print* ou *printf* dos objetos *System.out* ou *System.err*.
- ◆ Recomenda-se incluir, na mensagem, o nome da classe e do método que estão sendo depurados.

Como depurar problemas em *servlets*

A console do TomCat:

A screenshot of a Windows-style console window titled "Catalina". The window has a standard title bar with minimize, maximize, and close buttons. The text inside the console is as follows:

```
Starting service Tomcat-Standalone
Apache Tomcat/4.0.1
Starting service Tomcat-Apache
Apache Tomcat/4.0.1
EmailServlet emailAddress: jsmith@hotmail.com
```

Código para realizar a impressão:

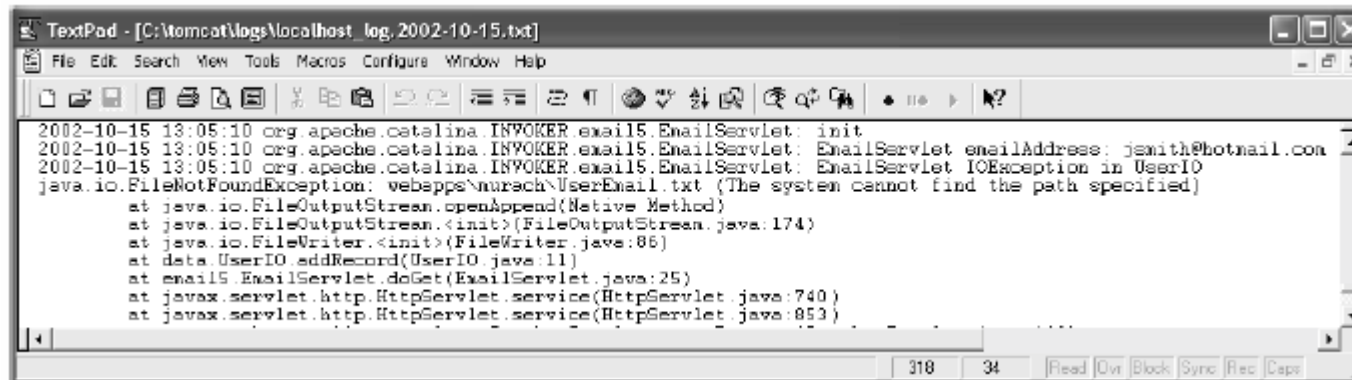
```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
    throws IOException, ServletException{
    // code
    String emailAddress = request.getParameter("emailAddress");
    System.out.println("EmailServlet emailAddress: " + emailAddress);
    // code
}
```

Como depurar problemas em *servlets* arquivo *log*

- ◆ Para tal, existem dois métodos da classe `HttpServlet`:
 - `log(String mensagem)`
 - Escreve uma mensagem no *log* do *container*.
 - `log(String mensagem, Throwable t)`
 - Escreve uma mensagem no *log* do *container* registrando, também, os métodos chamados para executar determinado comando.

Como depurar problemas em *servlets* arquivo *log*

Um arquivo log do TomCat:



```
TextPad - [C:\tomcat\logs\localhost_log, 2002-10-15.txt]
File Edit Search View Tools Macros Configure Window Help

2002-10-15 13:05:10 org.apache.catalina.INVOKER.email5.EmailServlet: init
2002-10-15 13:05:10 org.apache.catalina.INVOKER.email5.EmailServlet: EmailServlet emailAddress: jsmith@hotmail.com
2002-10-15 13:05:10 org.apache.catalina.INVOKER.email5.EmailServlet: EmailServlet IOException in UserIO
java.io.FileNotFoundException: webapps\nurach\UserEmail.txt (The system cannot find the path specified)
    at java.io.FileOutputStream.openAppend(Native Method)
    at java.io.FileOutputStream.<init>(FileOutputStream.java:174)
    at java.io.FileWriter.<init>(FileWriter.java:86)
    at data.UserIO.addRecord(UserIO.java:11)
    at email5.EmailServlet.doGet(EmailServlet.java:25)
    at javax.servlet.http.HttpServlet.service(HttpServlet.java:740)
    at javax.servlet.http.HttpServlet.service(HttpServlet.java:853)
```

Código para escrever em um arquivo log:

```
String emailAddress = request.getParameter("emailAddress");
log("EmailServlet emailAddress: " + emailAddress);
User user = new User(firstName, lastName, emailAddress);
try{
    UserIO.addRecord(user, file);
}
catch(IOException ioe){
    log("EmailServlet IOException in UserIO", ioe);
}
```

Codificando variáveis de instância

- ◆ Uma variável de instância pertence a uma instância de um *servlet*.
- ◆ É compartilhada por qualquer *thread* do *servlet*.
- ◆ Duas *threads* podem conflitar quando tentarem modificar a mesma variável de instância ao mesmo tempo.
- ◆ Para sincronizar o acesso ao bloco de código deve ser usada a palavra-chave *synchronized*.

Protegendo variáveis de instância

```
package com.exemplo.web;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class ContaAcessos extends HttpServlet{

    private int contadorGeral;

    public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws IOException, ServletException {

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

        synchronized(this){
            contadorGeral++;
        }

        out.println("<br> Esta página foi acessada " + contadorGeral + " vezes </br>");
    }
}
```

Codificando *servlets thread-safe*

- ◆ Um *servlet thread-safe* é aquele que trabalha confiavelmente mesmo quando mais de uma cópia da *thread* está ativa.
- ◆ Para sincronizar o acesso ao método ou ao bloco de código deve ser utilizada a palavra-chave *synchronized*.
- ◆ Implementando a interface `SingleThreadModel` o código inteiro do *servlet* torna-se serializável.

Codificando *servlets thread-safe*

Um bloco de código *synchronized*

```
synchronized(this){
    acessCount++;
    if (acessCount == 1000){
        LogUtil.logToFile("Atingimos 1000 usuários dia "
                        + new java.util.Date());
    }
}
```

Um método *synchronized*

```
public static synchronized int addRecord(Connection connection, User user)
    throws SQLException{

    String query =
        "INSERT INTO User " +
        "(EmailAddress, FirstName, LastName) " +
        "VALUES ('" + SQLUtil.encode(user.getEmailAddress()) + "', " +
        "'" + SQLUtil.encode(user.getFirstName()) + "', " +
        "'" + SQLUtil.encode(user.getLastName()) + "')";

    Statement statement = connection.createStatement();
    int status = statement.executeUpdate(query);
    statement.close();
    return status;
}
```

Um *servlet* que não permite múltiplos acessos de *threads*

```
public class EmailServlet3 extends HttpServlet implements SingleThreadModel {
}
```

Codificando *servlets thread-safe*

- ◆ Protegendo o *servlet context*:
 - `synchronized(getServletContext()){`
 }
 }
- ◆ Protegendo o `HttpSession`:
 - `synchronized(session){`
 }
 }



Invocando Classes regulares Java nos *servlets*



- ◆ Utilizadas para implementar a funcionalidade Model do MVC.
- ◆ Deve haver uma preocupação com o sincronismo devido aos múltiplos acessos dos *servlets*.
- ◆ *Servlets* invocam estas classes de forma convencional.



Invocando Classes regulares Java nos *servlets*



◆ Problema:

- Capturar as informações de nome, sobrenome, login e senha, fornecidos pelo usuário. Devem ser criticados e persistidos em um arquivo txt.



Invocando Classes regulares Java nos *servlets*



- ◆ Serão utilizados os 3 *servlets* do exercício sobre sessões. Modificando-se o último *servlet*.
- ◆ Será criada uma classe denominada GravaDados, que será invocada por este último *servlet*.
- ◆ A classe GravaDados implementa o método adicionaRegistro que gravará a informação no arquivo txt.

Invocando Classes regulares Java nos *servlets*

- Classe GravaDados – gravada no mesmo diretório do *servlet*.

```
import java.io.*;

public class GravaDados{

    public synchronized static void adicionaRegistro(
        String nome, String sobrenome, String login, String senha)
        throws IOException{

        PrintWriter out = new PrintWriter( new FileWriter("../webapps/poo/poo.txt", true));
        out.println(nome + "|" + sobrenome + "|" + login + "|" + senha);
        out.close();
    }

}
```


Invocando Classes regulares Java nos *servlets*

```
void displayValues(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    HttpSession minhasessao = request.getSession();
    firstName=(String) minhasessao.getAttribute("nome");
    lastName=(String) minhasessao.getAttribute("sobrenome");

    GravaDados grava = new GravaDados();
    grava.adicionaRegistro(firstName, lastName, userName, password);

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<HTML>");
    out.println("<HEAD>");
    out.println("<TITLE>Página 3 - Sessão</TITLE>");
    out.println("</HEAD>");
    out.println("<BODY>");
    out.println("<CENTER>");
    out.println("<H2>Página 3 - Sessão</H2>");
    out.println("<BR>");
    out.println("<BR>");
    out.println("Você forneceu os seguintes valores.");
    out.println("<BR>");
    out.println("<BR>");
    out.println("<TABLE>");
    out.println("<TR>");
    out.println("<TD>Nome: &nbsp;</TD>");
    out.println("<TD>" + firstName + "</TD>");
    out.println("</TR>");
    out.println("<TR>");
    out.println("<TD>SobreNome: &nbsp;</TD>");
    out.println("<TD>" + lastName + "</TD>");
```



MySQL



- ◆ Introdução ao MySQL.
- ◆ Interagindo com o MySQL.
- ◆ Iniciando e parando o servidor MySQL.
- ◆ Trabalhando com o programa MySQL.
- ◆ Como criar, selecionar e apagar um *database*.
- ◆ Como criar e apagar uma tabela.
- ◆ Como inserir ou carregar dados em uma tabela.
- ◆ JDBC.



MySQL



- ◆ *Open source database* que pode ser baixado gratuitamente de www.mysql.com.
- ◆ É um dos mais rápidos *databases* relacionais do mercado.
- ◆ Comparado a outros *databases*, é fácil de instalar e utilizar.
- ◆ Roda nos sistemas operacionais Windows, Unix, Solaris e OS/2.
- ◆ Suporte à integridade referencial, *subqueries* e transações.



MySQL



- ◆ Suporta SQL que é a linguagem padrão para trabalhar com *databases* relacionais.
- ◆ Suporta acessos de múltiplos clientes e inúmeras linguagens tais como Java, Perl, PHP, Python e C.
- ◆ Pode fornecer acesso aos seus dados via intranet ou internet.
- ◆ Pode restringir o acesso a seus dados somente para usuários autorizados.



Iniciando e parando o MySQL



- ◆ O MySQL *database server* é iniciado após a entrada do sistema operacional.
- ◆ O MySQL fornece uma GUI que permite ver algumas informações sobre o *database server*.
- ◆ Para parar o *database server* basta ir ao ícone do MySQL e parar o serviço.

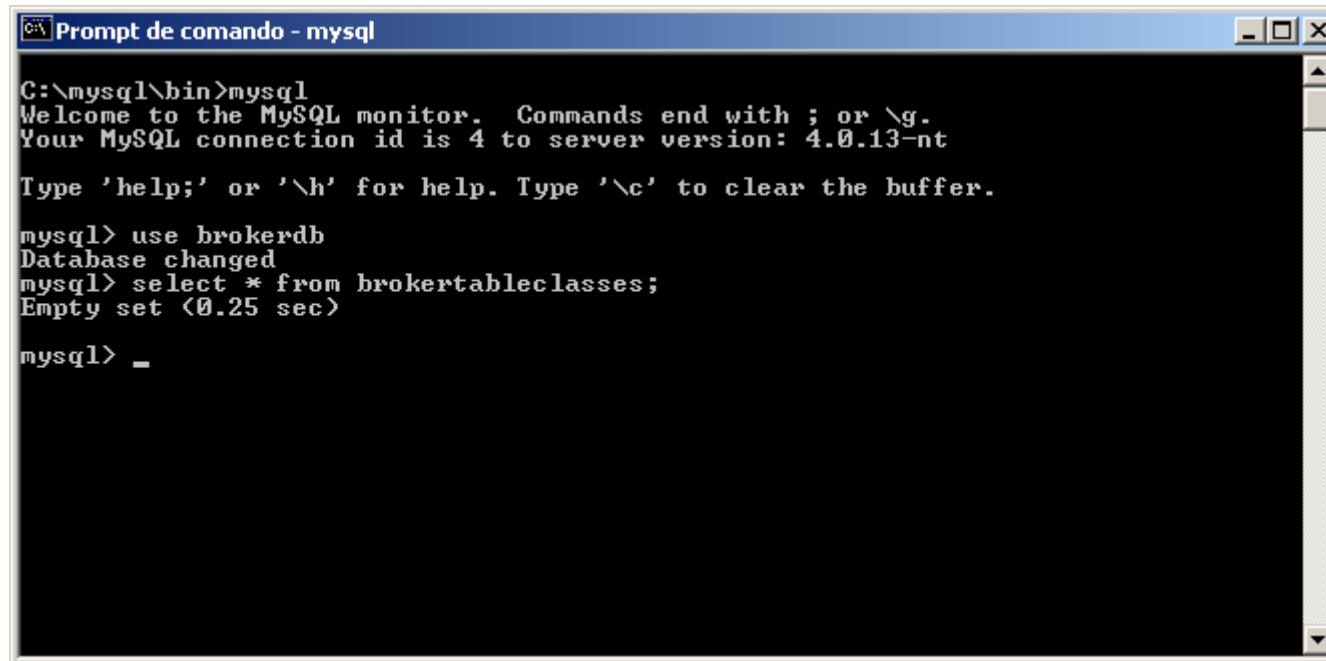


O programa MySQL



- ◆ Utilizado para realizar operações sobre o *database* MySQL.
- ◆ Pode ser conectado, local ou remotamente, ao *database server*.
- ◆ Executa operações MySQL e emite comandos SQL.

O programa MySQL



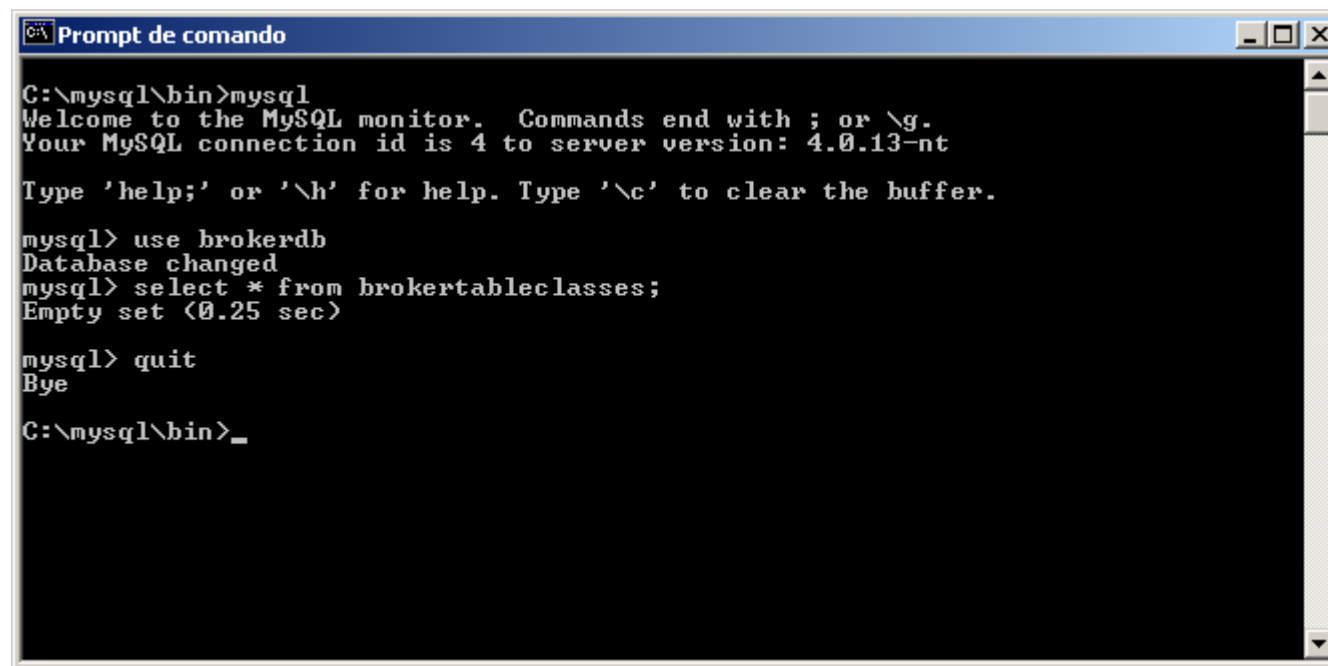
```
C:\mysql\bin>mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4 to server version: 4.0.13-nt

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> use brokerdb
Database changed
mysql> select * from brokertableclasses;
Empty set (0.25 sec)

mysql> _
```

Parando o programa MySQL



```
C:\mysql\bin>mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4 to server version: 4.0.13-nt

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> use brokerdb
Database changed
mysql> select * from brokertableclasses;
Empty set (0.25 sec)

mysql> quit
Bye
C:\mysql\bin>_
```


Como criar, selecionar e apagar um *database*

- ♦ Criando um database:
 - `mysql> create database dbteste;`
- ♦ Como listar os nomes de todos os databases:
 - `mysql> show databases;`
- ♦ Como selecionar um database para uso:
 - `mysql> use dbteste;`
- ♦ Como apagar um database:
 - `mysql> drop database dbteste;`

Como criar e apagar uma tabela

◆ Criando uma tabela:

```
mysql> create table Funcionario(  
-> ident int not null auto_increment,  
-> nome varchar(50),  
-> endereco varchar(50),  
-> sexo char(1),  
-> primary key(ident));
```

Como criar e apagar uma tabela

- ♦ Como listar todas as tabelas em um *database*:
 - `mysql> show tables;`
- ♦ Como apagar uma tabela:
 - `mysql> drop table Funcionario;`
- ♦ Apresentando a descrição dos campos de uma tabela:
 - `mysql> describe Funcionario;`

Como inserir ou carregar dados em uma tabela

◆ Inserindo dados:

```
mysql> insert into funcionario(nome, endereco,sexo)
-> values
-> ("Jose", "Rua abc 210", "M"),
-> ("Maria", "Rua def 220","F");
```

◆ Carregando dados:

```
mysql> load data local infile "c:/cursoweb/Users.txt" into table User;
```

- A tabela Users.txt deve estar formatada.

Alguns comandos SQL

◆ Selecionando todas as colunas de uma tabela:

```
SELECT *  
FROM tabela-1  
[WHERE critério-de-seleção]  
[ORDER BY campo-1 [ASC|DESC] [, campo-2 [ASC|DESC] ...]]
```

◆ Exemplos:

- SELECT * FROM Funcionario;
- SELECT * FROM Funcionario WHERE ident<3;

Alguns comandos SQL

- ◆ Selecionando algumas colunas de uma tabela:

```
SELECT campo-1 [, campo-2] ...  
FROM tabela-1  
[WHERE critério-de-seleção]  
[ORDER BY campo-1 [ASC| DESC] [, campo-2 [ASC| DESC] ...]]
```

- ◆ Exemplo:

- **SELECT nome, sexo FROM Funcionario
WHERE ident>2
ORDER BY nome ASC;**

Selecionando dados de múltiplas tabelas

- ◆ Operação conhecida como *join*.
- ◆ Existem 2 tipos de *join*:
 - INNER– somente são selecionadas as linhas daquelas colunas cujos conteúdos são idênticos.
 - OUTER – todas as linhas de uma tabela são selecionadas mesmo se não existir correspondente na outra tabela.
 - **LEFT OUTER** – são incluídos todos os registros da primeira tabela.
 - **RIGTH OUTER** – são incluídos todos os registros da segunda tabela.

Selecionando dados de múltiplas tabelas

◆ Sintaxe:

```
SELECT campo-1 [, campo-2] ...  
FROM tabela-1  
    {INNER | LEFT OUTER | RIGHT OUTER} JOIN tabela-2  
    ON tabela-1.campo-1 {= | < | > | <= | >= | <> } tabela-2.campo-2  
[WHERE critério-de-seleção]  
[ORDER BY campo-1 [ASC| DESC] [, campo-2 [ASC| DESC] ...]]
```

◆ Exemplo:

Tabela_Empregados

Codigo_Empregado	Nome
01	Silva, João
02	Oliveira, Adriano
03	Oliveira, Marcos
04	Costa, Marina

Tabela_Vendas

Identificação_Produto	Produto	Código_Empregado
234	Impressora	01
657	Mesa	03
865	Cadeira	03

Selecionando dados de múltiplas tabelas

```
SELECT Tabela_Empregados.Nome, Tabela_Vendas.Produto  
FROM Tabela_Empregados  
INNER JOIN Tabela_Vendas  
ON Tabela_Empregados.Codigo_Empregado=Tabela_Vendas.Codigo_Empregado
```

Nome	Produto
Silva, João	Impressora
Oliveira, Marcos	Mesa
Oliveira, Marcos	Cadeira

Selecionando dados de múltiplas tabelas

```
SELECT Tabela_Empregados.Nome, Tabela_Vendas.Produto  
FROM Tabela_Empregados  
LEFT JOIN Tabela_Vendas  
ON Tabela_Empregados.Codigo_Empregado= Tabela_Vendas.Codigo_Empregado
```

Nome	Produto
Silva, João	Impressora
Oliveira, Adriano	
Oliveira, Marcos	Mesa
Oliveira, Marcos	Cadeira
Costa, Marina	

Selecionando dados de múltiplas tabelas

```
SELECT Tabela_Empregados.Nome, Tabela_Vendas.Produto  
FROM Tabela_Empregados  
RIGHT JOIN Tabela_Vendas  
ON Tabela_Empregados.Codigo_Empregado= Tabela_Vendas.Codigo_Empregado
```

Nome	Produto
Silva, João	Impressora
Oliveira, Marcos	Mesa
Oliveira, Marcos	Cadeira

Como inserir, atualizar e apagar dados

◆ Inserindo dados:

- INSERT INTO tabela [(lista-de-campos)]
VALUES (lista-de-valores)

◆ Exemplo:

- INSERT INTO Vendas (UserID,
CodigoProduto) VALUES (1, 'Mic01')

Como inserir, atualizar e apagar dados

◆ Atualizando dados:

- **UPDATE tabela**
SET expressão-1 [, expressão-2]...
WHERE critério-de-seleção

◆ Exemplos:

- **UPDATE Usuario**
SET Nome = 'Rafael',
WHERE EmailAddress='rcardozo@globo.com'
- **UPDATE Produtos**
SET PrecoProduto=40.95
WHERE PrecoProduto=36.50

Como inserir, atualizar e apagar dados

◆ Apagando dados:

- **DELETE FROM tabela**
WHERE critério-de-seleção
- **DELETE FROM tabela (apaga tudo)**
- **DELETE * FROM tabela (apaga tudo)**

◆ Exemplos:

- **DELETE FROM Usuario WHERE**
EmailAddress='rcardozo@globo.com'
- **DELETE FROM Download WHERE DataDownload**
< '2005-08-02'

Java Database Connectivity (*JDBC*)

- ◆ Tecnologia que permite o acesso e a manipulação de um banco de dados em Java.
- ◆ Faz parte do pacote `java.sql`.
- ◆ Existem vários tipos de *drivers*.
- ◆ Para aplicações *web* utiliza-se os *drivers* tipos 3 e 4.
- ◆ O MySQL *driver* é do tipo 4 e é gratuito.

Java Database Connectivity (*JDBC*)

- ◆ Onde obter *database drivers* ?
 - www.java.sun.com/products/jdbc
- ◆ No TomCat o *database driver* é instalado colocando-se o arquivo .jar no diretório common/lib.



O Java e os *databases*



- ◆ Carregando o *database driver*.
- ◆ Conectando-se a um *database*.
- ◆ Escrevendo uma declaração.
- ◆ Trabalhando com um *result set*.
- ◆ Recuperando dados de um *result set*.
- ◆ Inserindo, atualizando e apagando dados.

Carregando o *database driver* do MYSQL

```
try {  
    Class.forName("com.mysql.jdbc.Driver");  
}  
catch (ClassNotFoundException e) {  
    System.out.println(e); /* Driver não encontrado */  
}
```

- Pode ser incluído no método `init()` do *servlet*.

Conectando-se a um *database*

- ◆ Utiliza-se o método `getConnection`, da classe `DriverManager`.

```
private String nomeDoBanco="jdbc:mysql://localhost/brokerdb",  
            username="brokeruser",  
            password="brokerpswd";  
  
try {  
    Connection con = DriverManager.getConnection(nomeDoBanco, username, password);  
} catch (SQLException e) {  
    System.out.println(e);  
}
```

- ◆ Para fechar a conexão utiliza-se o método `close`. Ex: `con.close()`.



Outros métodos do DriverManager



- ◆ `isClosed()` – utilizado para verificar se a conexão está fechada.
- ◆ `createStatement()` – usado para criar um objeto `Statement` que enviará declarações SQL ao banco de dados.
- ◆ `preparedStatement()` – usado para enviar múltiplas declarações para o banco de dados.

Criando uma declaração

- ◆ Utilizada para enviar declarações SQL para o database.
 - Ex.: `Statement stmt = con.createStatement();`
- ◆ Para fazer consultas ao database utiliza-se o método `executeQuery()`.
- ◆ Para fazer atualizações (*update*, *delete* ou *insert*) utiliza-se o método `executeUpdate()`.
- ◆ Esta declaração também deve ser fechada.
 - Ex: `stmt.close()`.

Recuperando informações

- ◆ É gerado um objeto *result set* contendo as colunas e as linhas com o resultado solicitado.
- ◆ Existem métodos específicos para navegar pelo *result set* e para recuperar os campos da tabela produzida.
 - Ex.: `ResultSet produtos = stmt.executeQuery("SELECT * FROM tabela_produtos");`

Navegando pelo *result set* Alguns Métodos

- ◆ `next()` – move o cursor para a próxima linha do *result set*.
- ◆ `last()` – move o cursor para a última linha do *result set*.
- ◆ `first()` – move o cursor para a primeira linha do *result set*.
- ◆ `close()` – fecha o *result set*.
- ◆ `getRow()` – retorna um inteiro que identifica a linha corrente do *result set*.

Recuperando campos do *result set*

- ◆ `getXXX(intColumnIndex)` – retorna dados de uma coluna identificada pelo seu número.
 - Ex.: `produtos.getString(1)`
- ◆ `getXXX(StringColumnName)` – retorna dados de uma coluna identificada pelo seu nome.
 - Ex.: `produtos.getString(“nomedoproduto”)`



Recuperando campos do *result set*



- ◆ O método `getXXX` pode ser usado para retornar todos os 8 tipos primitivos dados.
- ◆ O `XXX` indica o tipo primitivo do dado.
- ◆ Também pode ser usado para retornar *strings* (`getString`), datas (`getDate`) e hora (`getTime`).



Prepared statements



- ◆ Quando a aplicação envia uma declaração, o *database server* realiza as seguintes tarefas:
 - Checa se há erros de sintaxe.
 - Prepara um plano para executar a declaração.
 - Executa a declaração.
- ◆ Se a mesma declaração é enviada novamente o *database server* não checa a sintaxe e nem prepara outro plano. Com isto aumenta-se o desempenho das operações no *database*.

Prepared statements consultando

```
PreparedStatement ps;
```

```
ResultSet rs;
```

```
String consulta = "SELECT * from agregadoanimal where id=? " ;
```

```
    ps = con.prepareStatement(consulta);
```

```
    ps.setInt(1, animal.getId());
```

```
    rs = ps.executeQuery();
```

```
    while(rs.next()){
```

```
        animal.setEspecie(rs.getString("especie"));
```

```
        animal.setRaca(rs.getString("raca"));
```

```
        animal.setNome(rs.getString("nome"));
```

```
        animal.setCaracteristica(rs.getString("caracteristica"));
```

```
        animal.setHabilitado(rs.getBoolean("habilitado"));
```

```
    }
```

Prepared statements atualizando

```
String altera = "UPDATE agregadoanimal SET "
```

```
+ " especie = ?, "  
+ " raca = ?, "  
+ " nome = ?, "  
+ " caracteristica = ?, "  
+ " idcolaborador = ?, "  
+ " habilitado= ? "  
+ " WHERE id = ? ";
```

```
ps = con.prepareStatement(altera);  
ps.setString(1,animal.getEspecie());  
ps.setString(2,animal.getRaca());  
ps.setString(3,animal.getNome());  
ps.setString(4,animal.getCaracteristica());  
ps.setString(5,animal.getIdColaborador());  
ps.setBoolean(6,animal.getHabilitado());  
ps.setInt(7,animal.getId());  
ps.executeUpdate();
```

Prepared statements inserindo

```
String insere = "Insert into agregadoanimal(especie, raca, nome, caracteristica, idcolaborador,habilitado) " +  
                "Values (?, ?, ?, ?, ?, ?)";  
  
ps = con.prepareStatement(insere);  
ps.setString(1,animal.getEspecie());  
ps.setString(2,animal.getRaca());  
ps.setString(3,animal.getNome());  
ps.setString(4,animal.getCaracteristica());  
ps.setString(5,animal.getIdColaborador());  
ps.setBoolean(6,animal.getHabilitado());  
ps.executeUpdate();
```



Prepared statements excluindo



```
String exclui = "DELETE from agregadoanimal WHERE id = ? ";  
ps = con.prepareStatement(exclui);  
ps.setInt(1,animal.getId());  
ps.executeUpdate();
```



Tratando Conexões



- ◆ Abrir e fechar conexões a todo instante é custoso.
- ◆ Existe forte recomendação para sempre fechar uma conexão aberta.
- ◆ Uma conexão aberta é utilizada, exclusivamente, por uma única *thread* do *servlet*.
- ◆ Os modernos bancos de dados suportam conexões simultâneas.
- ◆ Solução conciliatória: criação de um *pool* de conexões abertas.



Homologando uma transação



- ◆ Para evitar perda de informações é necessário que algumas operações no *database* sejam realizadas em conjunto.
- ◆ São disponibilizados três métodos da classe DriverManager: `setAutoCommit()`, `commit()`, `rollback()`.

Homologando uma transação

- ◆ `setAutoCommit()` – indica se a homologação se dará por declaração individual(*true*) ou por um conjunto de declarações(*false*).
- ◆ O valor *default* é *true*.
- ◆ `commit()` – utilizado para homologar a transação, caso o `setAutoCommit` seja *false*.
- ◆ `rollback()` – utilizado para retornar a transação a posição anterior.



Java Server Page



- ◆ É uma extensão da tecnologia *servlet*.
- ◆ Criada para suportar a criação de páginas HTML e XML.
- ◆ Combina conteúdo estático com dinâmico.
- ◆ Desonera o programador de se preocupar com os elementos de arte da página.



Java Server Page



- ◆ Consiste de *tags* HTML e código Java.
- ◆ O código Java fica embutido no código HTML como um *scriptlet* ou uma expressão.
- ◆ Um *scriptlet* é usado para executar um ou mais comandos Java.
- ◆ Uma expressão é usada para apresentar um texto.
- ◆ Para identificar *scriptlets* e expressões são utilizadas *tags* específicas.

Java Server Page

- ♦ Atos do *container* para processar uma JSP:
 - Quando acionada, a JSP é traduzida para um arquivo .java.
 - Este arquivo .java é compilado tornando-se um arquivo .class.
 - O arquivo .class é carregado e transforma-se, finalmente, em um *servlet*.
- ♦ Obs.: Consultar TomCat/work/Catalina/localhost/aplicacao/org/apache/jsp



Onde salvar uma JSP ?



- ◆ Precisa ser salva em um diretório visto pelo *web server*.
- ◆ No Tomcat 5.0 pode ser usado qualquer diretório sob o diretório *webapps*.
- ◆ Página deve ter o nome com o sufixo *jsp*.

Scriptlets e expressões

◆ *Scriptlet*

- `<% java statements %>` **não esquecer o “;”**

◆ *Expressão*

- `<% = expressão %>`
 - Atua como um argumento de `out.print()`.

Scriptlets e expressões

- ♦ `<% String nome=request.getParameter("nome"); %>`
O nome é `<%= nome %>` //ASSIM ou
O nome é `<%= request.getParameter("nome")%>`

```
<%@ page session="false" %>
<% int numeroDeVezes=1;
    while (numeroDeVezes<=5) {
%>
<h3> esta linha é apresentada <%= numeroDeVezes %> vez(es) . </h3>
<%     numeroDeVezes++;
    }
%>
```

Como codificar *scriptlets* e expressões



esta linha é apresentada 1 vez(es) .

esta linha é apresentada 2 vez(es) .

esta linha é apresentada 3 vez(es) .

esta linha é apresentada 4 vez(es) .

esta linha é apresentada 5 vez(es) .

Como codificar *scriptlets* e expressões



The screenshot shows a Mozilla Firefox browser window with the title "Cadastro na Lista - Mozilla Firefox". The address bar displays "http://localhost:8080/exemplo1/cadastro.html". The main content area features a heading "Entrar na lista de emails" and a paragraph: "Para entrar na lista, insira seu nome e email abaixo. Complete a operação clicando no botão Submit." Below this, there are three input fields: "Nome:" with the value "João", "Sobrenome:" with the value "Silva", and "Email:" with the value "jsilva@bol.com.br". A "Submit" button is located below the email field. The status bar at the bottom indicates "Concluído".

Entrar na lista de emails

Para entrar na lista, insira seu nome e email abaixo.
Complete a operação clicando no botão Submit.

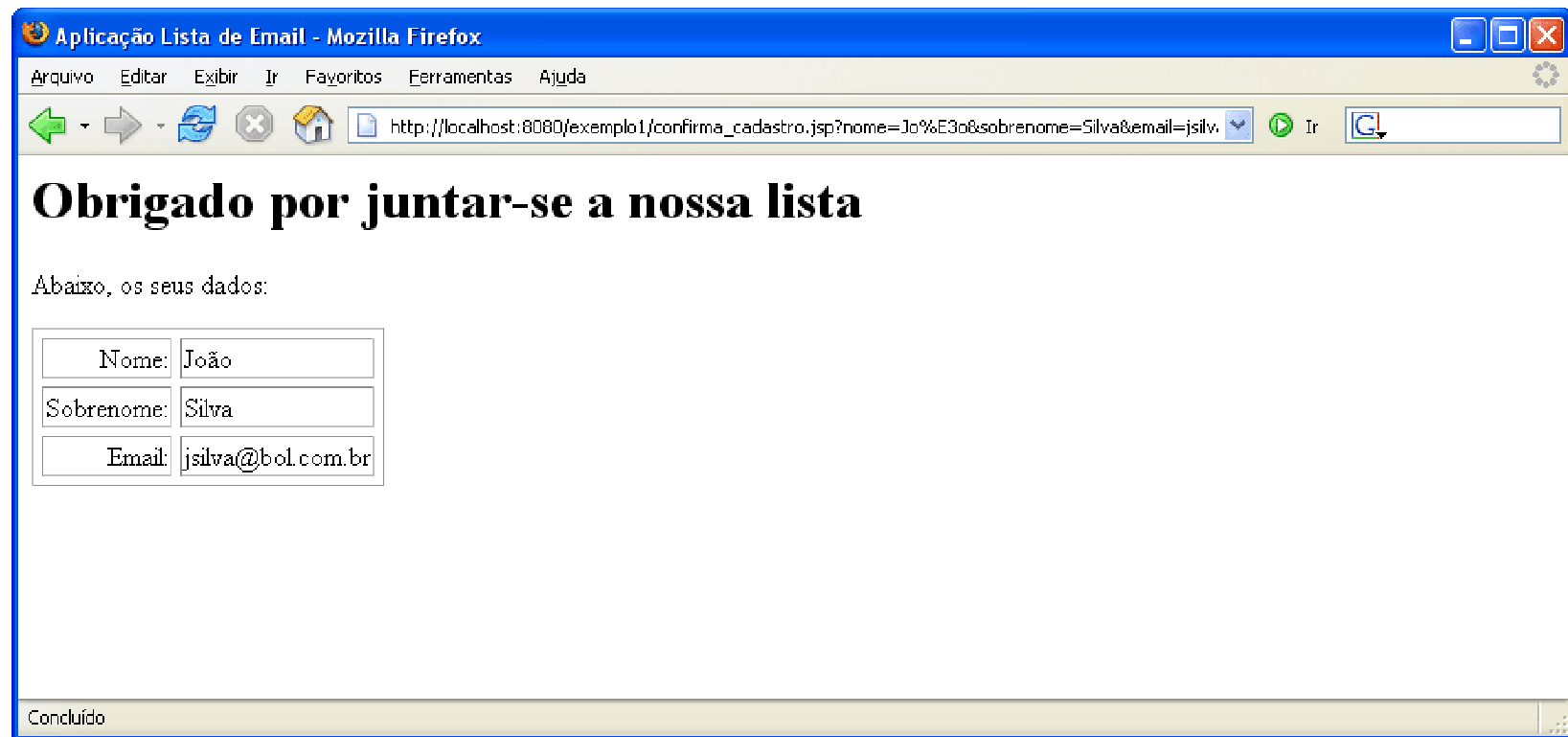
Nome:

Sobrenome:

Email:

Concluído

Como codificar *scriptlets* e expressões



Página HTML

```
<!DOCTYPE HTML PUBLIC "-//w3c//dtd html 4.0 transitional//en">
```

```
<html>
```

```
<head>
```

```
<title>Cadastro na Lista</title>
```

```
</head>
```

```
<h1>Entrar na lista de emails</h1>
```

```
<p>Para entrar na lista, insira seu nome e email abaixo. <br>  
Complete a operação clicando no botão Submit.</p>
```

Página HTML

```
<form action="confirma_cadastro.jsp" method="get">
<table cellpadding="5" border="0">
  <tr>
    <td align="right">Nome:</td>
    <td><input type="text" name="nome"></td>
  </tr>
  <tr>
    <td align="right">Sobrenome:</td>
    <td><input type="text" name="sobrenome"></td>
  </tr>
  <tr>
    <td align="right">Email:</td>
    <td><input type="text" name="email"></td>
  </tr>
  <tr>
    <td></td>
    <td><br><input type="submit" value="Submit"></td>
  </tr>
</table>
</form>
</body>
</html>
```

A página confirma_cadastro.jsp

```
<%@ page language="java" %>
<!DOCTYPE HTML PUBLIC "-//w3c//dtd html 4.0 transitional//en">
<html>
<head>
<title>Aplicação Lista de Email</title>
</head>
<body>
<%
    String nome = request.getParameter("nome");
    String sobrenome = request.getParameter("sobrenome");
    String email = request.getParameter("email");
%>
<h1>Obrigado por juntar-se a nossa lista</h1>
<p>Abaixo, os seus dados:</p>
<table cellpadding="5" cellspacing="5" border="1">
    <tr>
        <td align="right">Nome:</td>
        <td><%= nome %></td>
    </tr>
    <tr>
        <td align="right">Sobrenome:</td>
        <td><%= sobrenome %></td>
    </tr>
    <tr>
        <td align="right">Email:</td>
        <td><%= email %></td>
    </tr>
</table>
</body>
</html>
```



JSP – Ciclo de Vida



- ◆ O cliente aciona uma JSP.
- ◆ O *container* tenta traduzir o JSP para um arquivo .java.
- ◆ Se OK, o .java é compilado gerando um arquivo .class.
- ◆ O *container* carrega o .class passando a tratá-lo como um *servlet*.



JSP – Ciclo de Vida



- ◆ São criados os métodos:
 - `jspInit()`.
 - Pode ser sobreposto.
 - `jspService()`.
 - Não pode ser sobreposto.
 - `jspDestroy()`.
 - Pode ser sobreposto.

JSP – Ciclo de Vida

- ♦ O *container* instancia o *servlet* acionando o método `jspInit()`.
- ♦ O *container* cria uma nova *thread* e aciona o método `jspService()`.
- ♦ A *thread* é disparada.
- ♦ O *container* aciona o método `jspDestroy()`.

Obs.: A tradução e a compilação só acontecem uma única vez.

JSP – Parâmetros de iniciação

```
<web-app...>
```

```
....
```

```
<servlet>
```

```
  <servlet-name>Musicas</servlet-name>
```

```
    <jsp-file>/Recomendacao.jsp</jsp-file>
```

```
    <init-param>
```

```
      <param-name>faleConosco</param-name>
```

```
      <param-value>centralatendimento@nce.ufrj.br</param-value>
```

```
    </init-param>
```

```
    <init-param>
```

```
      <param-name>areaVendas</param-name>
```

```
      <param-value>vendasatendimento@nce.ufrj.br</param-value>
```

```
    </init-param>
```

```
  </servlet>
```

```
...
```

```
</web-app>
```

JSP – Sobrepondo o jspInit()

. Exemplo: Obtendo o parâmetro de iniciação:

<%!

```
public void jspInit() {  
    ServletConfig meuconfig = getServletConfig();  
    String emailFale = meuconfig.getInitParameter("faleConosco");  
    String emailVendas = meuconfig.getInitParameter("areaVendas");  
    ServletContext ctx = getServletContext();  
    ctx.setAttribute("e-mailFale", emailFale);  
    ctx.setAttribute("e-mailVendas", emailVendas);  
}
```

%>



As diretivas de uma JSP



- ◆ São instruções passadas para o *container* em tempo de tradução de uma JSP.
- ◆ Existem 3 tipos de diretivas:
 - *page*.
 - *include*.
 - *taglib*.

Diretiva *page*

◆ Sintaxe:

- `<%@ page atributo1="valor1"... atributon="valorn" %>`

◆ Alguns atributos:

- **import**
 - Define os comandos *imports* que devem ser adicionados à classe gerada.
- **isThreadSafe**
 - Define se a JSP deve ser SingleThreadModel (*false*).
- **contentType**
 - Define o tipo do MIME.
- **isErrorPage**
 - Define se a página corrente representa uma página de erro.
- **errorPage**
 - Define a URL da página que tratará o erro.

Diretiva *page*

♦ Alguns atributos:

■ **language**

- Define a linguagem de *script*. Por enquanto é Java.

■ **extends**

- Define a superclasse de quem a JSP pode herdar.

■ **session**

- Indica se a página terá um objeto *session* implícito (*default =true*).

■ **buffer**

- Define a buferização para o objeto *out*.

■ **isELIgnored**

- Define se expressões EL devem ser ignoradas quando a página for traduzida.

Diretivas de página

- ◆ `<%@ page contentType="text/html;charset=GB2312" %>`
- ◆ `<%@ page language="java" %>`
- ◆ `<%@ page import="java.io.*" %>` único replicável
- ◆ `<%@ page buffer="16kb" %>`
- ◆ `<%@ page session="false" %>`
- ◆ `<%@ page errorPage="PaginadeErro.jsp" %>`
- ◆ `<%@ page session="false" buffer="16kb" %>`

Declaração JSP

- ◆ Utilizada para definir métodos, variáveis estáticas e variáveis de instâncias.
- ◆ Declarada fora do método `jspService()`.
- ◆ Sintaxe:
 - `<%! método %>`
 - `<%! variável; %>`

Declaração JSP

```
<%!  
    String obtemHoraCorrente() {  
        return Calendar.getInstance().getTime().toString();  
    }  
%>
```

```
<%! int numero=0; %>
```

- Pode ser inserida em qualquer lugar da JSP.

Declaração JSP - exemplo

```
<%@ page import="business.*, data.*, java.util.Date, java.io.*" %>

<%! int accessCount = 0; %>
<%!
    public synchronized void addRecord(User user, String filename)
        throws IOException{
        PrintWriter out = new PrintWriter(
            new FileWriter(filename, true));
        out.println(user.getEmailAddress() + "|"
            + user.getFirstName() + "|"
            + user.getLastName());
        out.close();
    }
%>
<%
    String nome = request.getParameter("nome");
    String sobrenome = request.getParameter("sobrenome");
    String email = request.getParameter("email");

    User user = new User(nome, sobrenome, email);
    addRecord(user, "../webapps/exemplo1/WEB-INF/etc/UserEmail.txt");
    int localCount = 0;
    synchronized (this) {
        accessCount++;
        localCount = accessCount;
    }
%>
...
<p><i>Essa página foi acessada <%= localCount %> vezes.</i></p>
```

Como invocar classes em uma JSP

```
package business;

public class User{
    private String firstName;
    private String lastName;
    private String emailAddress;

    public User(){}

    public User(String first, String last, String email){
        firstName = first;
        lastName = last;
        emailAddress = email;
    }

    public void setFirstName(String f){
        firstName = f;
    }
    public String getFirstName(){ return firstName; }

    public void setLastName(String l){
        lastName = l;
    }
    public String getLastName(){ return lastName; }

    public void setEmailAddress(String e){
        emailAddress = e;
    }
    public String getEmailAddress(){ return emailAddress; }
}
```

Como invocar classes em uma JSP

```
package data;

import java.io.*;
import java.util.*;
import business.User;

public class UserIO{
    public synchronized static void addRecord(User user, String fileName)
        throws IOException{
        File file = new File(fileName);
        PrintWriter out = new PrintWriter(
            new FileWriter(file, true));
        out.println(user.getEmailAddress() + "|"
            + user.getFirstName() + "|"
            + user.getLastName());

        out.close();
    }
}
```

- Estas classes devem ser salvas no diretório **WEB-INF/classes**, sob o diretório da aplicação *web*.

Como invocar classes em uma JSP

```
<!doctype html public "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
  <title>Lista de Emails</title>
</head>
<body>
<%@ page import="business.*, data.*, java.util.Date, java.io.*" %>
<%
  String nome = request.getParameter("nome");
  String sobrenome = request.getParameter("sobrenome");
  String email = request.getParameter("email");

  User user = new User(nome, sobrenome, email);
  UserIO.addRecord(user, "../webapps/exemplo1/WEB-INF/etc/UserEmail.txt");
%>
```

JSP – Objetos Implícitos

- ◆ Uma JSP utiliza objetos que não precisam ser explicitamente obtidos ou declarados.

- API

- JspWriter
- HttpServletRequest
- HttpServletResponse
- HttpSession
- ServletContext
- ServletConfig

Objeto Implícito

out

request

response

session

application

config

pageContext

JSP – Objetos Implícitos

No servlet:

```
getServletContext().setAttribute("email",suporte);
```

Na JSP:

```
application. setAttribute("email",suporte);
```

No servlet:

```
request.setAttribute("email",suporte);
```

No JSP:

```
request.setAttribute("email",suporte);
```

No servlet:

```
request.getSession().setAttribute("email",suporte);
```

No JSP:

```
session.setAttribute("email",suporte);
```



JSP – pageContext



- ◆ Utilizado para obter, ou atribuir, valores ou objetos de uma JSP.
- ◆ Escopos:
 - Page (*default*)
 - Request
 - Session
 - Application

JSP – pageContext

- ◆ Atribuindo um valor *page-scoped*:
 - `<% pageContext.setAttribute("email",suporte) %>`
- ◆ Atribuindo um valor *session-scoped*:
 - `<% pageContext.setAttribute("email",suporte,PageContext.SESSION_SCOPE) %>`
- ◆ Obtendo um valor *session-scoped*:
 - `<% =pageContext.getAttribute("email",PageContext.SESSION_SCOPE) %>`
- ◆ Atribuindo um valor *application-scoped*:
 - `<% pageContext.setAttribute("email",suporte,PageContext.APPLICATION_SCOPE) %>`
- ◆ Obtendo um valor *application-scoped*:
 - `<%=pageContext.getAttribute("email",PageContext.APPLICATION_SCOPE) %>`
- ◆ Achando um atributo quando se desconhece o escopo:
 - `<% =pageContext.findAttribute("email") %>`

Diretivas de inclusão

- ◆ Permite incluir o conteúdo de outros arquivos na página JSP atual. Feito em tempo de tradução.
- ◆ A página incluída pode ser uma página estática (HTML) ou dinâmica (JSP).
- ◆ A página inserida tem acesso às variáveis de instância da JSP principal.
- ◆ Sintaxe:
 - `<%@ include file="URLrelativa" %>`
- ◆ Exemplo:
 - `<%@ include file="POO/cabecalho.html" %>`

Diretivas de inclusão - exemplo

- Arquivo Cabecalho.htm

```
<HTML>
```

```
<HEAD>
```

```
<TITLE> Bem-Vindo </TITLE>
```

```
</HEAD>
```

```
<BODY>
```

- Arquivo Rodape.htm

```
</BODY>
```

```
</HTML>
```

Diretivas de inclusão - exemplo

```
<%@ page session="false" %>
<%@ page import="java.util.Calendar" %>
<%@ include file="Cabecalho.htm" %>
<%
    out.println("Hora atual: " + Calendar.getInstance().getTime());
%>
<%@ include file="Rodape.htm" %>
```

Diretivas de inclusão - exemplo



Hora atual: Sat Apr 23 00:16:59 BRT 2005

Actions

◆ Existem dois tipos:

- `<jsp:action />`

- Inclusão é feita em tempo de execução. É o padrão.

- `<x:action />`

- x identifica a *tag*.

Resumo dos elementos em uma JSP

- ◆ `<% %>` *scriptlet* JSP.
 - Para inserir comandos em Java.
- ◆ `<%= %>` expressão JSP.
 - Para apresentar o resultado de uma expressão.
- ◆ `<%@ %>` diretiva JSP.
 - Para atribuir condições aplicáveis a toda JSP.
- ◆ `<%! %>` declaração JSP.
 - Para declarar variáveis de instância e métodos na JSP.
- ◆ `<jsp: />` ou `<x: />`
 - Para realizar uma ação.

JSP – Melhorando a carga inicial

```
<servlet>  
  <servlet-name>RecepcaoJspServlet</servlet-name>  
  <jsp-file>/Recepcao.jsp</jsp-file>  
  <load-on-startup>1</load-on-startup>  
</servlet>
```



JSP – Tornando a JSP a página inicial



```
<welcome-file-list>  
  <welcome-file>index.jsp</welcome-file>  
</welcome-file-list>
```




Filtros



- ◆ Possibilita interceptar um *request* antes de ele chegar ao *servlet*.
- ◆ Possibilita interceptar um *response* antes de ele chegar ao cliente.
- ◆ Tem acesso aos objetos `ServletRequest` e `ServletResponse`.
- ◆ Pode ser usado como dispositivo de criptografia, compressão de dados, validação de entrada de dados etc.

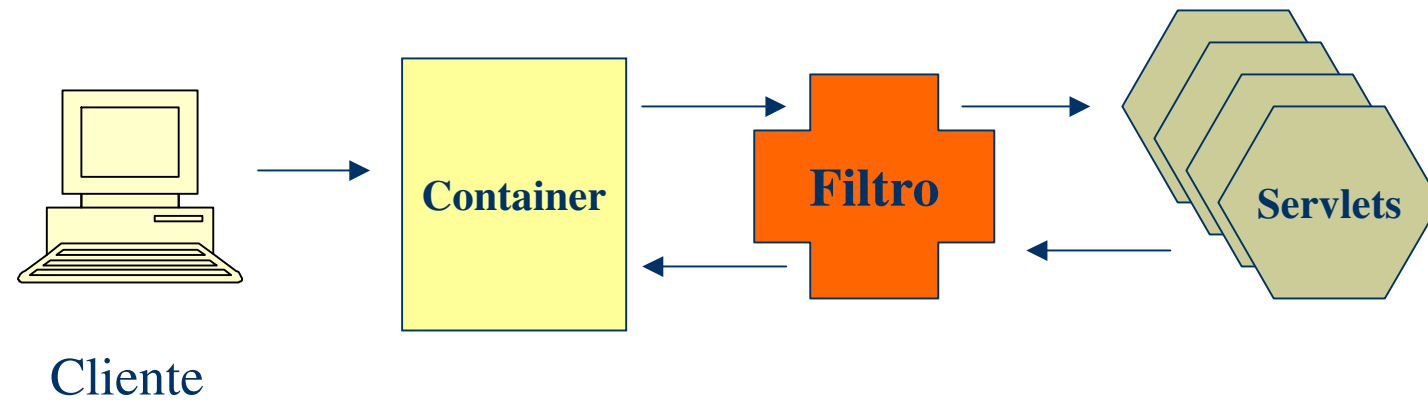


Filtros



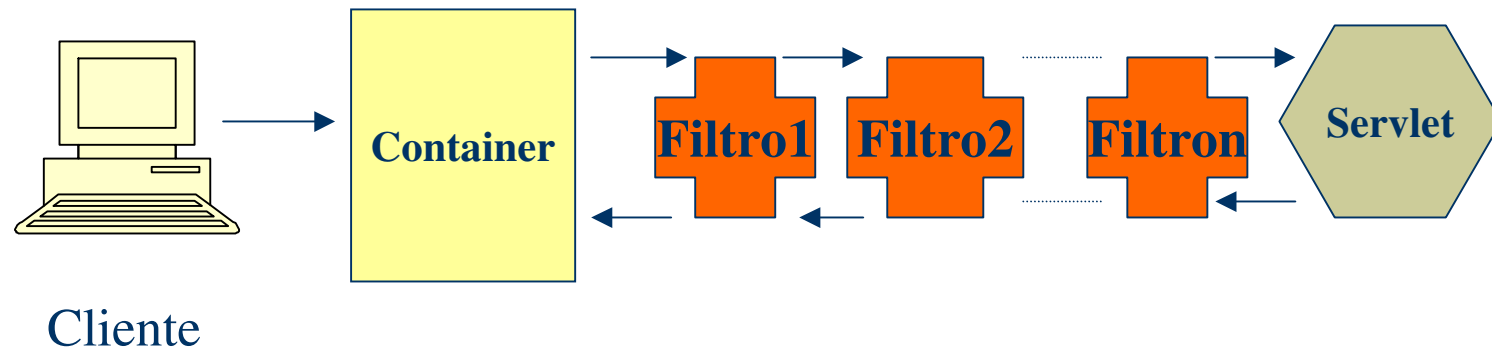
- ◆ Invocados pelo *container* .
- ◆ Declarados no web-xml são desconhecidos pelo desenvolvedor.
- ◆ O web-xml associa o filtro ao *servlet*.
- ◆ Um filtro pode estar associado a mais de um *servlet*.
- ◆ É possível colocar um conjunto de filtros em cadeia. Um filtro apontará para o outro, refinando a filtragem.

Filtros



- Os filtros podem:
 - checar segurança;
 - reformatar *requests* e corpos;
 - comprimir/descomprimir dados;
 - auditar *requests* e *responses*;
 - colocar anexos no *response*;

Filtros - encadeamento



- Os filtros são interindependentes, um desconhece a existência do outro;



Filtros interfaces



- ◆ São utilizadas 3 interfaces do pacote `javax.servlet`:
 - `Filter`.
 - `FilterConfig`.
 - `FilterChain`.

A interface Filter

- ◆ Precisa ser implementada para se escrever um filtro.
- ◆ Cada invocação de um filtro “dispara” uma *thread*.
- ◆ O ciclo de vida de um filtro é representado pelos métodos:
 - `init(FilterConfig filterConfig)`.
 - `doFilter(ServletRequest request, ServletResponse response, FilterChain cadeia)`
 - `destroy()`.

A interface Filter

- ◆ O filtro adquire vida quando o método `init()` é invocado pelo *container*. Este método é executado apenas na primeira chamada do filtro.
- ◆ No método `doFilter()` é onde as operações do filtro são executadas.
- ◆ O *container* chama este método sempre que é solicitado o *servlet* associado ao filtro.
- ◆ Os objetos *request* e *response* podem ser obtidos e modificados pelo método `doFilter()`.



A interface FilterConfig



- ◆ Passa valores de iniciação para o filtro através dos parâmetros obtidos no web-xml.
- ◆ Possui 4 métodos:
 - `getFilterName()`
 - `getInitParameter(String parameterName)`
 - `getInitParameterName()`
 - `getServletContext()`

Filtros – web-xml

```
<filter>
  <filter-name>GravaLog</filter-name>
  <filter-class>web.GravaLogFiltro</filter-class>
  <init-param>
    <param-name>ArquivoLog</param-name>
    <param-value>arquivolog.txt</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>GravaLog</filter-name>
  <servlet-name>MelhoresBandas</servlet-name>
</filter-mapping>
```

Filtros – web-xml

- Os filtros podem capturar *include*, *forward* e *error*:

```
<filter-mapping>
    <filter-name>GravaLog</filter-name>
    <servlet-name>MelhoresBandas</servlet-name>
    <dispatcher>REQUEST</disptacher>           // default
    <dispatcher>INCLUDE</disptacher>
    <dispatcher>FORWARD</disptacher>
    <dispatcher>ERROR</disptacher>
</filter-mapping>
```



Filtros – exemplo



1. Um filtro apresentando os seus ciclos de vida:
 - Nome do filtro – Ciclo Vida
 - Nome da classe do filtro – CicloVida.class
 - Nome do servlet filtrado – FilteredFilter
 - Nome da classe do servlet – FilteredFilter.class

A distribuição descritiva

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <!-- Define os filtros -->
  <filter>
    <filter-name>Ciclo Vida</filter-name>
    <filter-class>CicloVida</filter-class>
  </filter>
  <!-- Define o mapeamento dos filtros -->
  <filter-mapping>
    <filter-name>Ciclo Vida</filter-name>
    <servlet-name>FilteredServlet</servlet-name>
  </filter-mapping>
  <servlet>
    <servlet-name>FilteredServlet</servlet-name>
    <servlet-class>FilteredServlet</servlet-class>
  </servlet>
</web-app>
```

O filtro CicloVida

```
import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class CicloVida implements Filter {
    private FilterConfig filterConfig;

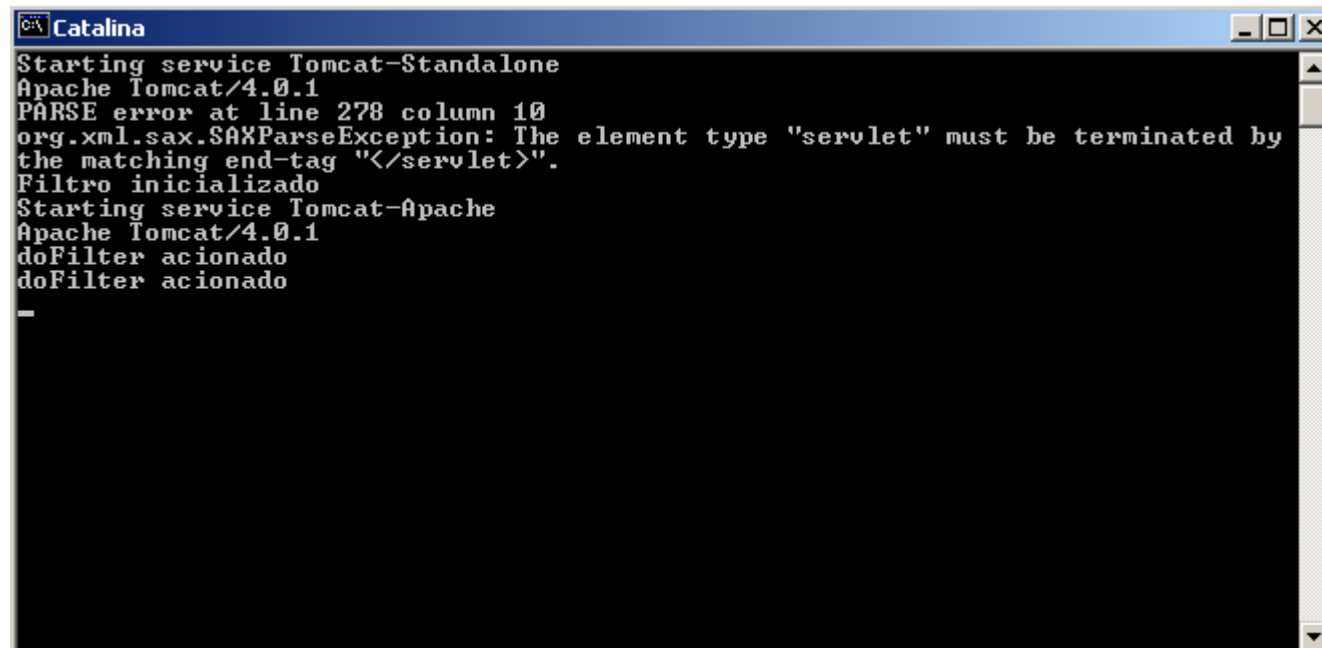
    public void init(FilterConfig filterConfig) throws ServletException {
        System.out.println("Filtro inicializado");
        this.filterConfig = filterConfig;
    }

    public void destroy() {
        System.out.println("Filtro destruído");
        this.filterConfig = null;
    }

    public void doFilter(ServletRequest request, ServletResponse response,
        FilterChain chain)
        throws IOException, ServletException {
        System.out.println("doFilter acionado");
        chain.doFilter(request, response);
    }
}
```

Código extraído, e modificado, do livro: Java para Web com Servlets, JSP e EJB, de Budi Kurniawan.

Filtros - console TomCat



```
Catalina
Starting service Tomcat-Standalone
Apache Tomcat/4.0.1
PARSE error at line 278 column 10
org.xml.sax.SAXParseException: The element type "servlet" must be terminated by
the matching end-tag "</servlet>".
Filtro inicializado
Starting service Tomcat-Apache
Apache Tomcat/4.0.1
doFilter acionado
doFilter acionado
-
```

A distribuição descritiva – um filtro para dois servlets

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <!-- Define filtros -->
  <filter>
    <filter-name>Ciclo Vida </filter-name>
    <filter-class>CicloVida</filter-class>
  </filter>
  <!-- Define o mapeamento do filtro para os 2 servlets -->
  <filter-mapping>
    <filter-name>Ciclo Vida</filter-name>
    <servlet-name>FilteredServlet</servlet-name>
  </filter-mapping>
  <filter-mapping>
    <filter-name>Ciclo Vida</filter-name>
    <servlet-name>FilteredServlet2</servlet-name>
  </filter-mapping>
  <servlet>
    <servlet-name>FilteredServlet</servlet-name>
    <servlet-class>FilteredServlet</servlet-class>
  </servlet>
  <servlet>
    <servlet-name>FilteredServlet2</servlet-name>
    <servlet-class>FilteredServlet2</servlet-class>
  </servlet>
</web-app>
```



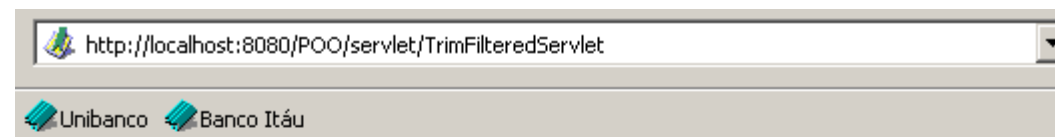
Filtros - exemplo



2. Um filtro suprimindo espaços em branco de campos de uma entrada de dados via web:

- Nome do filtro – Trim Filtro
- Nome da classe – TrimFiltro.class
- Nome do servlet – TrimFilteredServlet
- Nome da classe – TrimFilteredServlet.class

Filtros - exemplo



Please enter your details.

First Name:
Last Name:
User Name:
Password:

Login

Filtros - exemplo



Here are your details.

First Name: rafael

Last Name: cardozo

User Name: rafa

Password: 123456

A distribuição descritiva

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <!-- Define filtros -->
  <filter>
    <filter-name>Trim Filtro</filter-name>
    <filter-class>TrimFiltro </filter-class>
  </filter>
  <!-- Define o mapeamento dos filtros -->
  <filter-mapping>
    <filter-name>Trim Filtro</filter-name>
    <servlet-name>TrimFilteredServlet</servlet-name>
  </filter-mapping>
  <servlet>
    <servlet-name>TrimFilteredServlet</servlet-name>
    <servlet-class>TrimFilteredServlet</servlet-class>
  </servlet>
</web-app>
```

O filtro TrimFiltro

```
import java.io.*;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import java.util.Enumeration;

public class TrimFiltro implements Filter {
    private FilterConfig filterConfig = null;
    public void destroy() {
        System.out.println("Filtro destruído");
        this.filterConfig = null;
    }

    public void doFilter(ServletRequest request, ServletResponse response,
        FilterChain chain)
        throws IOException, ServletException {
        System.out.println("Filtro realizado");
        Enumeration enuma = request.getParameterNames();
        while (enuma.hasMoreElements()) {
            String parameterName = (String) enuma.nextElement();
            String parameterValue = request.getParameter(parameterName);
            request.setAttribute(parameterName, parameterValue.trim());
        }

        chain.doFilter(request, response);
    }

    public void init(FilterConfig filterConfig) throws ServletException {
        System.out.println("Filtro inicializado");
        this.filterConfig = filterConfig;
    }
}
```

Código extraído, e modificado, do livro: Java para Web com Servlets, JSP e EJB, de Budi Kurniawan.

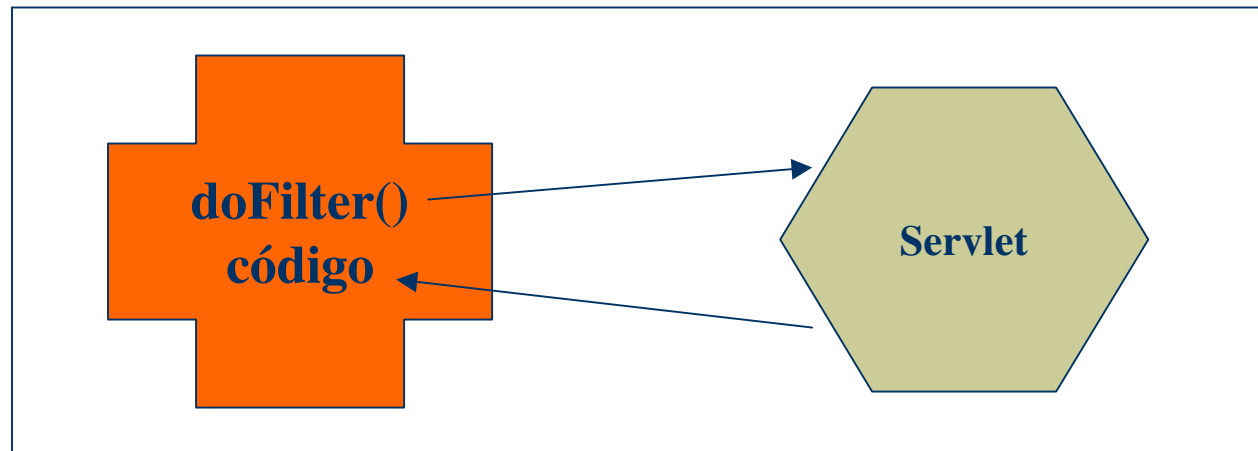
Filtrando a resposta - exemplo

3. Filtro de resposta para anexar um cabeçalho e um rodapé a um *servlet*.

- Nome do filtro – Response Filter
- Nome da classe – ResponseFilter.class
- Nome do servlet – ResponseFilterServlet
- Nome da classe – ResponseFilterServlet.class

Filtrando a resposta

- ◆ Deve ser adicionado um código após a chamada do método `doFilter`.
- ◆ Após a execução do *servlet* o controle retorna para o filtro.



A distribuição descritiva

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <!-- Define os filtros -->
  <filter>
    <filter-name>Response Filter</filter-name>
    <filter-class>ResponseFilter</filter-class>
  </filter>
  <!-- Define o mapeamento dos filtros -->
  <filter-mapping>
    <filter-name>Response Filter</filter-name>
    <servlet-name>ResponseFilteredServlet</servlet-name>
  </filter-mapping>
  <servlet>
    <servlet-name>ResponseFilteredServlet</servlet-name>
    <servlet-class>ResponseFilteredServlet</servlet-class>
  </servlet>
</web-app>
```

O filtro ResponseFilter

```
import java.io.*;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class ResponseFilter implements Filter {
    private FilterConfig filterConfig = null;
    public void destroy() {
        System.out.println("Filtro destruído");
        this.filterConfig = null;
    }
    public void doFilter(ServletRequest request, ServletResponse response,
        FilterChain chain)
        throws IOException, ServletException {
        System.out.println("doFilter");
        PrintWriter out = response.getWriter();
        // Isto é adicionado ao início do PrintWriter
        out.println("<HTML>");
        out.println("<BODY>");
        out.println("<CENTER>");
        out.println("Cabeçalho");
        out.println("<HR>");

        chain.doFilter(request, response);

        // Isto é adicionado ao fim do PrintWriter
        out.println("<HR>");
        out.println("Rodapé");
        out.println("<CENTER>");
        out.println("</BODY>");
        out.println("</HTML>");
    }

    public void init(FilterConfig filterConfig) throws ServletException {
```

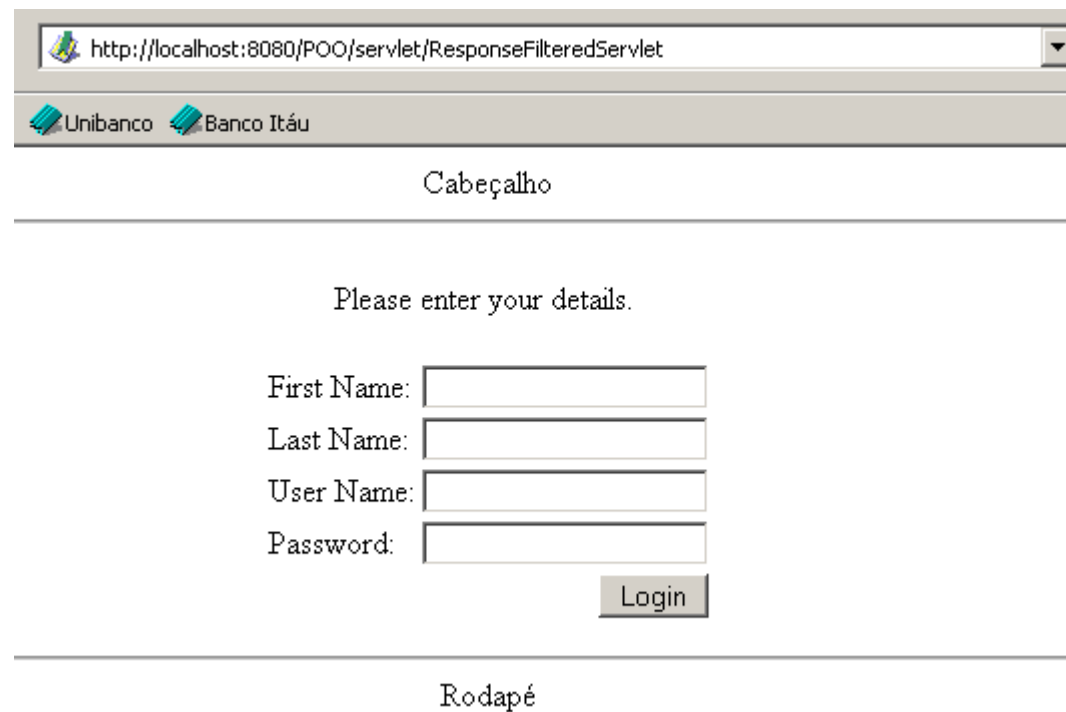

O servlet ResponseFilteredServlet

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

public class ResponseFilteredServlet extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<BR>Please enter your details.");
        out.println("<BR>");
        out.println("<BR><FORM METHOD=POST>");
        out.println("<TABLE>");
        out.println("<TR>");
        out.println("<TD>First Name:</TD>");
        out.println("<TD><INPUT TYPE=TEXT NAME=firstName></TD>");
        out.println("</TR>");
        out.println("<TR>");
        out.println("<TD>Last Name:</TD>");
        out.println("<TD><INPUT TYPE=TEXT NAME=lastName></TD>");
        out.println("</TR>");
        out.println("<TR>");
        out.println("<TD>User Name:</TD>");
        out.println("<TD><INPUT TYPE=TEXT NAME=username></TD>");
        out.println("</TR>");
        out.println("<TR>");
        out.println("<TD>Password:</TD>");
        out.println("<TD><INPUT TYPE=PASSWORD NAME=password></TD>");
        out.println("</TR>");
        out.println("<TR>");
        out.println("<TD ALIGN=RIGHT COLSPAN=2>");
        out.println("<INPUT TYPE=SUBMIT VALUE=Login></TD>");
        out.println("</TR>");
        out.println("</TABLE>");
        out.println("</FORM>");
    }
}
```

Filtrando a resposta - exemplo



The screenshot shows a web browser window with the address bar displaying `http://localhost:8080/POO/servlet/ResponseFilteredServlet`. The browser's toolbar includes icons for Unibanco and Banco Itáu. The page content is divided into three sections by horizontal lines:

- Cabeçalho** (Header): A horizontal line at the top of the content area.
- Body**: Contains the text "Please enter your details." followed by four input fields labeled "First Name:", "Last Name:", "User Name:", and "Password:". A "Login" button is positioned to the right of the "Password" field.
- Rodapé** (Footer): A horizontal line at the bottom of the content area.

Fazendo *download* de um arquivo

◆ Passos necessários:

- Atribuir ao *setContentType*, do objeto *response*, o valor “application/octet-stream”.
- Atribuir ao *setHeader*, do *response*, o valor “content-disposition” especificando o nome do arquivo para *download*.
- Escrever o código Java para ler o arquivo, importando o pacote `java.io.FileInputStream`.

Fazendo *download* de um arquivo – *Servlet*

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

public class ServletDownload extends HttpServlet {

    public void doGet ( HttpServletRequest request,
                       HttpServletResponse response )
                       throws ServletException, IOException {

        doPost(request, response);
    }

    public void doPost (HttpServletRequest request,
                       HttpServletResponse response )
                       throws ServletException, IOException {

        String filename="curriap.doc";
        String filepath="C:/cartas/";
        response.setContentType("application/octet-stream");
        PrintWriter out = response.getWriter();
        response.setHeader("content-disposition","attachment; filename=\""+
            filename + "\"");

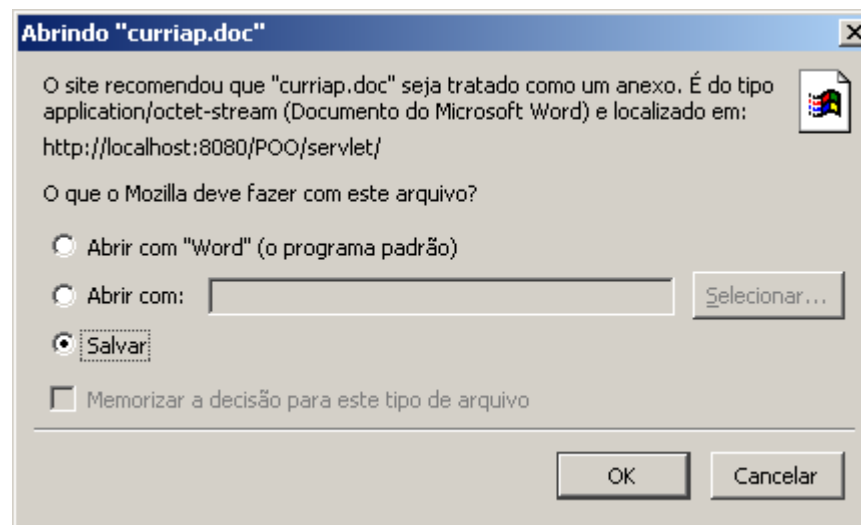
        java.io.FileInputStream fileInputStream =
            new java.io.FileInputStream(filepath + filename);

        int i;

        while ((i=fileInputStream.read()) != -1) {
            out.write(i);
        }

        fileInputStream.close();
        out.close();
    }
}
```

Fazendo *download* de um arquivo – *Servlet*



Fazendo *download* de um arquivo – JSP

```
<%@ page import="java.io.FileInputStream" %>
<%
    String filename="BlackDog.mp3";
    String filepath="C:/cartas/";
    response.setContentType(
        "APPLICATION/OCTET-STREAM");
    response.setHeader("Content-Disposition",
        "attachment; filename=\"\" + filename + "\"");

    java.io.FileInputStream fileInputStream =
        new java.io.FileInputStream(filepath + filename);
    int i;

    while ((i=fileInputStream.read()) != -1) {
        out.write(i);
    }

    fileInputStream.close();
    out.close();

%>
```

Fazendo *refresh* automático de uma página

- ◆ Através do objeto *response*:
 - `response.setIntHeader("refresh", tempo em segundos)`
- ◆ Através de um META comando (na JSP):
 - `<META HTTP-EQUIV="REFRESH" CONTENT="tempo-em-segundos">`

Fazendo *refresh* automático de uma página - exemplo

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<HTML>");
    out.println("<HEAD>");
    out.println("<TITLE>Apresentando o URL encoding</TITLE>");
    out.println("</HEAD>");
    out.println("<BODY>");
    out.println("<BR>");
    out.println("<BR><FORM METHOD=POST>");
    out.println("<P> Nome: <INPUT TYPE=TEXT NAME=nome > </P>");
    out.println("<P> SobreNome: <INPUT TYPE=TEXT NAME=sobrenome > </P>");
    out.println("<INPUT TYPE=SUBMIT VALUE=Submit>");
    out.println("</FORM>");
    out.println("</BODY>");
    out.println("</HTML>");
}

public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    String nome= request.getParameter("nome");
    String sobrenome=request.getParameter("sobrenome");
    i++;
    response.setIntHeader("refresh",60);
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<HTML>");
    out.println("<HEAD>");
    out.println("<TITLE>Apresentando o URL encoding</TITLE>");
    out.println("</HEAD>");
    out.println("<BODY>");
    out.println("<BR>");
    out.println("<TABLE>");
    out.println("<TR>");
    out.println("<TD>" + i + "</TD>");
```


Distribuindo a aplicação

- ◆ Utiliza-se o **Web Archive**.
- ◆ No TomCat o nome da arquivo WAR torna-se o nome da aplicação.
- ◆ Sob o diretório da aplicação emitir o comando:
 - *jar -cvf nome-da-aplicação.war **
- ◆ Enviar o arquivo *war* gerado para o diretório *webapps* do TomCat de destino.
- ◆ O TomCat alvo descomprime o arquivo e cria o diretório da aplicação.



JavaBeans



- ◆ Abordagem centrada em componentes.
- ◆ Quando utilizados em uma aplicação *web* incorpora parte da lógica do negócio.
- ◆ Estimula a reutilização.
- ◆ *Designer* utiliza *action tags* para chamar os *beans* em uma página JSP.

Como codificar um JavaBean

- ◆ Regras para uma classe ser um JavaBean:
 1. Precisa ter um construtor sem argumentos.
 2. Não pode declarar uma variável de instância como sendo pública. Podem ser *privates* ou *protecteds*.
 3. Precisa conter métodos *get* e *set* para obter acesso as suas variáveis de instância.
 4. Métodos que acessam variáveis de instância do tipo *boolean* são prefixados com *is*.

Servlet passando parâmetros

O Servlet:

```
public void doPost(...){  
    String nome = request.getParameter("login");  
    request.setAttribute("login",login);  
    RequestDispatcher vista=request.getRequestDispatcher("apresenta.jsp");  
    vista.forward(request,response);  
}
```

A JSP:

```
<html><body>  
<%= request.getAttribute("login") %> <br>Obrigado pela sua visita</br>  
</body></html>
```

Será apresentado, por exemplo:

Raquel
Obrigado pela sua visita

Servlet passando parâmetros

O Servlet:

```
public void doPost(...){  
    Aluno aluno = new Aluno();  
    aluno.setNome("Raquel");  
    request.setAttribute("aluno",aluno);  
    RequestDispatcher vista=request.getRequestDispatcher("apresenta.jsp");  
    vista.forward(request,response);  
}
```

A JSP:

```
<html><body>  
<%= request.getAttribute("aluno") %> <br>Obrigado pela sua visita</br>  
</body></html>
```

Será apresentado, por exemplo:

Aluno@289d34
Obrigado pela sua visita

Servlet passando parâmetros

O Servlet:

```
public void doPost(...){  
    Aluno aluno = new Aluno();  
    aluno.setNome("Raquel");  
    request.setAttribute("aluno",aluno);  
    RequestDispatcher vista=request.getRequestDispatcher("apresenta.jsp");  
    vista.forward(request,response);  
}
```

A JSP:

```
<html><body>  
<% Aluno aluno = (Aluno) request.getAttribute("aluno"); %>  
<%= aluno.getNome() %>  
<br>Obrigado pela sua visita</br></body></html>
```

Será apresentado:

Raquel
Obrigado pela sua visita



O JavaBean Aluno



```
package estudante;  
public class Aluno{  
  
    private String nome;  
  
    public String getNome(){  
        return this.nome;  
    }  
    public void setNome(String nome){  
        this.nome = nome;  
    }  
}
```

Utilizando *action* com JavaBeans

```
<html><body>
<%@ page import="estudante.*" %>
    <jsp:useBean id="aluno" class="estudante.Aluno" scope="request"/>
    <jsp:getProperty name="aluno" property="nome" />
<br>Obrigado pela sua visita</br></body></html>
```

O que significa?

useBean – identifica o action;

id – identifica o objeto, mesmo efeito que request.setAttribute(**"aluno"**, aluno);

class – identifica a classe JavaBean;

scope – identifica o escopo do atributo;

getProperty – identifica a ação;

name – identifica o objeto alvo;

property – identifica o nome da propriedade que se deseja recuperar;

A *action tag* useBean

♦ Sintaxe:

`<jsp:useBean id="Name" class="package.class" scope="value" />`

♦ Valores possíveis do escopo:

- **Page** – o valor é válido somente nesta página. É o *default*.
- **Request** – o valor deve ser capturado de um objeto *request*.
- **Session** – o valor deve ser capturado de um objeto *session*.
- **Application** – o valor deve ser capturado de um objeto *context*.

A tags getProperty e setProperty

- ♦ getProperty obtém o valor de um atributo de um JavaBean:

```
<jsp:getProperty name="aluno" property="nota" />
```

- ♦ setProperty atribui uma valor para um atributo de um JavaBean:

```
<jsp:setProperty name="aluno" property="nota" value="8.5" />
```

O atributo *param*

- ◆ Utilizado para alterar propriedades de um JavaBean a partir de um objeto *request*.

- ◆ Sintaxe:

```
<jsp:setProperty name="bean" property="propertyName" param="paramName" />
```

- ◆ Exemplo:

```
<jsp:setProperty name="aluno" property="Nome" param="PrimeiroNome" />
```

- ◆ *Scriptlet* correspondente:

```
<% aluno.setNome(request.getParameter("PrimeiroNome")); %>
```

O atributo *param*

- A página inicial:

```
<HTML>
<HEAD>
<TITLE> Testando TAGS </TITLE>
</HEAD>
<BODY>
<CENTER>
  Digite um nome
  <BR>
  <FORM METHOD =POST ACTION=PrimeiraPagina.jsp>
  <INPUT TYPE=TEXT NAME=primeironome>
  <INPUT TYPE=SUBMIT VALUE="OK">
</FORM>
</CENTER>
</BODY>
</HTML>
```

O atributo *param*

- A página que acionará o JavaBean:

```
<%@ page session="false" %>
<jsp:useBean id="meuBean" class="POO.VerNome" />
<jsp:setProperty name="meuBean" property="nome" param="primeironome" />
O nome digitado é <jsp:getProperty name="meuBean" property="nome" />
```

- A classe JavaBean:

```
package POO;

public class VerNome{

    private String nome;

    public String getNome(){
        return this.nome;
    }

    public void setNome(String nome){
        this.nome = nome;
    }

}
```

O atributo *param*



O nome digitado é João Ubaldo Ribeiro

Passando todos os parâmetros obtidos

```
<html>
  <body>
    <%@ page import="estudante.*" %>
    <jsp:useBean id="aluno" class="estudante.Aluno" />
    <jsp:setProperty name="aluno" property="*" />
  </body>
</html>
```

- Válido para todos aqueles valores que possuem nomes iguais aos atributos.
- Funciona como um *param* genérico.

Capturando um atributo de um atributo

```
<%= (Pedido) request.getAttribute("pedido").getPizza().getPreco() %>
```

- O elemento *action* padrão não atende!
- A solução é Expression Language!

```
<%@ page import="model.*" %>
<%@ page isELIgnored="false" %>
<html>
    <body>
        <h1 align =center="center"> Testa BeanEL JSP</h1>
        <br>Pizza muito barata ${pedido.pizza.preco}
        </br>
    </body>
</html>
```




Expression Language - EL



- ◆ Possibilita aos *designers* desconhecerem Java.
- ◆ Oferece uma maneira mais simples para invocar código Java em uma JSP.
- ◆ O código fica em “algum lugar” e é invocado pela EL.
- ◆ “Algum lugar” pode ser um JavaBean, uma classe com métodos estáticos ou um Tag Handler.

Expression Language - EL

- ◆ Sintaxe:

- `${expressão}`

- ◆ Exemplo:

Nosso telefone: `<%= (String)application.getAttribute("fone") %>`

Nosso telefone: `${applicationScope.fone}`

Expression Language - EL

- O primeiro campo variável da expressão é um objeto implícito ou um atributo do escopo.
- Se o primeiro campo é um atributo ele pode ser qualquer um armazenado em um dos quatro escopos.
- Objetos implícitos válidos:
 - **pageScope**
 - **requestScope**
 - **sessionScope**
 - **applicationScope**
 - **param**
 - **paramValues**
 - **header**
 - **headerValues**
 - **cookie**
 - **initParam**
 - **pageContext** (é uma referência para um objeto pageContext)

Obs.: Com exceção do pageContext todos os outros objetos implícitos guardam um mapa com um par chave/valor .

Expression Language - EL

- ♦ O segundo campo pode ser separado do primeiro campo por um ponto.
- ♦ O segundo campo representa um atributo de um *bean* ou um valor de um Map.
- ♦ A alternativa ao ponto é o operador [].
 - `${aluno["nome"] }` ou
 - `${aluno.nome}`
- ♦ Utilizando o [] o primeiro campo pode ser um Map, um *bean*, um List ou um *array*.
- ♦ Um valor colocado entre [] pode ser uma chave de um Map, um atributo de um *bean* ou índice de um List ou de um *array*.

Expression Language - EL

- **O Servlet:**

```
String[] grupoRock = {"Led Zeppelin", "Barão Vermelho", "The Who", "The Police"};  
request.setAttribute("bandas", grupoRock);
```

- **O JSP:**

```
<%@ page import="model.*" %>  
<%@ page isELIgnored="false" %>  
<html>  
<body>  
<h1 align =center="center"> Testa BeanEL JSP</h1>  
<br> A melhor banda: ${bandas[0]} </br>  
</body>  
</html>
```



Expression Language - EL

• O Servlet:

```
Map testaMap = new HashMap();
testaMap.put("Terror", "Dracula" );
testaMap.put("Acao", "Exterminador do Futuro" );
testaMap.put("Humor", "Convidado Bem Trapalhão" );
request.setAttribute("filmes", testaMap);
```

• O JSP:

```
<%@ page import="model.*" %>
<%@ page isELIgnored="false" %>
<html>
<body>
<h1 align =center="center"> Testa BeanEL JSP</h1>
<br>Você selecionou: ${filmes.Humor}</br>
</body>
</html>
```



Expression Language - EL

- ◆ *Scriptlets* podem ser inibidos no web-xml.

```
<jsp-config>
  <jsp-property-group>
    <url-pattern>*.jsp</url-pattern>
    <scripting-invalid>
      true
    </scripting-invalid>
  </jsp-property-group>
</jsp-config>
```

Expression Language - EL

- ◆ Els podem ser inibidos no web-xml.

```
<jsp-config>
  <jsp-property-group>
    <url-pattern>*.jsp</url-pattern>
    <el-ignored>
      true
    </el-ignored>
  </jsp-property-group>
</jsp-config>
```

Ou

```
<%@ page isELIgnored = "true" %>
```


EL – Objetos Implícitos

- ◆ Capturando um *request parameter*:
 - `${param.nome-do-request-parameter}`
- ◆ Capturando um *request parameter* de múltiplos valores:
 - `${paramValues.nome-do-request-parameter[índice]}`
- ◆ Capturando um valor do *header*:
 - `${header.nome-do-header}`
- ◆ Capturando o valor de um *cookie*:
 - `${cookie.nome-do-cookie.value}`



Expression Language - EL



- ◆ EL pode invocar métodos estáticos de uma classe.
- ◆ Passos necessários:
 - Codificar a classe com um método estático público;
 - Codificar um arquivo Tag Library Descriptor(TLD);
 - Colocar a diretiva *taglib* na JSP;
 - Acionar o método através da EL;

Expression Language - EL

- A classe JogaDado com o método estático jogarDado:

```
package model;
```

```
public class JogaDado{
```

```
    public static int jogarDado(){  
        return (int)(Math.random() * 6) +1;  
    }
```

```
}
```

Expression Language - EL

- A TLD (*filename*=nome.tld, pode ficar no mesmo diretório da web-xml):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<taglib xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary_2_0.xsd"
version="2.0">
  <tlib-version>1.2</tlib-version>
  <uri>TestaTagDados</uri>
    <function>
      <name>jogando</name>
      <function-class>model.JogaDado</function-class>
      <function-signature>
        int jogarDado()
      </function-signature>
    </function>
</taglib>
```

Expression Language - EL

- Colocando a *taglib* na JSP acionando o método na EL:

```
<%@ page import="model.*" %>
<%@ page isELIgnored="false" %>
<%@ taglib prefix="joga" uri="TestaTagDados" %>
<html>
    <body>
        <br>A jogada é: ${joga:jogando()}</br>
    </body>
</html>
```



A action include



- ◆ Sintaxe:
 - `<jsp:include page="nome-da-página" />`
- ◆ A página é incluída em tempo de execução.
- ◆ Não tem acesso às variáveis de instância de outras páginas participantes.
- ◆ A página incluída trabalha isoladamente.

A action forward

- ◆ Sintaxe:
 - `<jsp:forward page="nome-da-página" />`
- ◆ Utilizada para direcionar o controle para outra JSP, servlet ou página HTML.
- ◆ Tudo que foi escrito antes do forward não será apresentado.
- ◆ Para apresentar utilizar: **out.flush()**.

JSP Standard Tag Library - JSTL

- ◆ Permite realizar tarefas não suportadas pelas Els e pela *action* padrões.
- ◆ Necessário instalar o JSTL 1.1.
- ◆ Não é padrão na especificação do JSP 2.0.
- ◆ Possui as bibliotecas **jstl.jar** e **standard.jar**.
- ◆ Podem ser copiadas de Tomcat/webapps/jsp-examples/WEB-INF/lib/.
- ◆ Devem ir para Tomcat/webapps/sua-aplicacao/WEB-INF/lib.

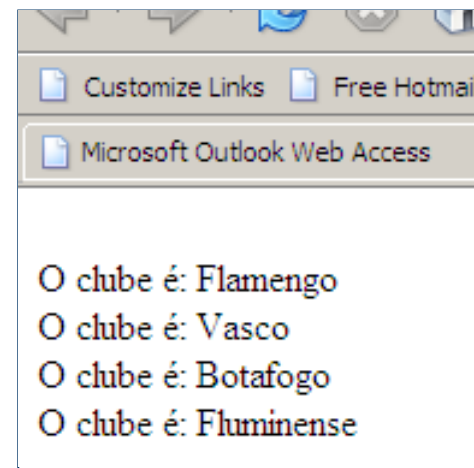
JSTL – <c:forEach>

- ◆ <c:forEach></c:forEach>
 - Permite iteragir sobre *arrays* ou coleções.
- ◆ Sintaxe:
 - <c:forEach var=“variável-de-iteração” items=“\${nome-do-atributo}”
 \${variável-de-iteração}
 </c:forEach>

JSTL – exemplo <c:forEach>

```
Vector<String> vetor = new Vector<String>();
    vetor.add("Flamengo" );
    vetor.add("Vasco" );
    vetor.add("Botafogo" );
    vetor.add("Fluminense" );
    request.setAttribute("times", vetor);
    RequestDispatcher vista =
    request.getRequestDispatcher("TestaBeanEL.jsp");
    vista.forward(request, response);
```

```
<%@ page import="model.*" %>
<%@ page isELIgnored="false" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
    <body>
        <c:forEach var="vetortimes" items="${times}">
            <br>O clube é: ${vetortimes}
        </c:forEach>
    </body>
</html>
```



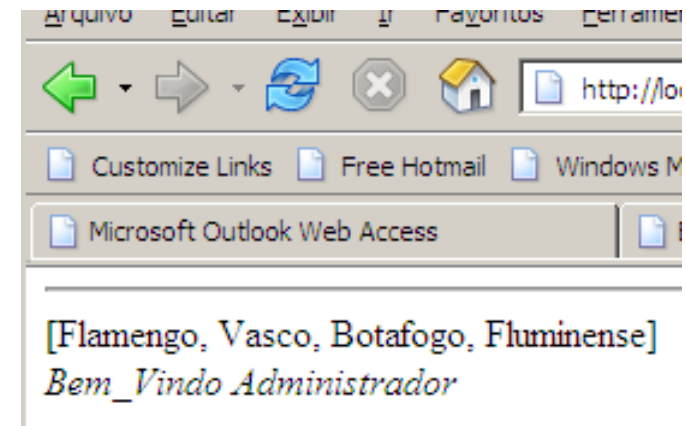
JSTL – <c:if >

- ◆ <c:if></c:if>
 - Permite realizar operações condicionais.
- ◆ Sintaxe:
 - <c:if test="{nome-do-atributo op-relacional 'membro'}">
.....
</c:if>

JSTL – exemplo <c:if>

```
List<String> vetor = new ArrayList<String>();
    vetor.add("Flamengo" );
    vetor.add("Vasco" );
    vetor.add("Botafogo" );
    vetor.add("Fluminense" );
    request.setAttribute("times", vetor);
    String login="admin";
    request.setAttribute("login", login);
```

```
<%@ page import="model.*" %>
<%@ page isELIgnored="false" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
    <body>
        <hr>${times} </hr>
        <c:if test="${login eq 'admin'}">
            <jsp:include page="Administrador.jsp"/>
        </c:if>
    </body>
</html>
```



JSTL - Operadores

◆ Aritméticos:

- +
- -
- *
- / ou div
- % ou mod

◆ Lógicos:

- && ou and
- || ou or
- ! ou not

◆ Relacionais:

- == ou eq
- != ou ne
- < ou lt
- > ou gt
- <= ou le
- >= ou ge

JSTL –

<c:choose> <c:when><c:otherwise>

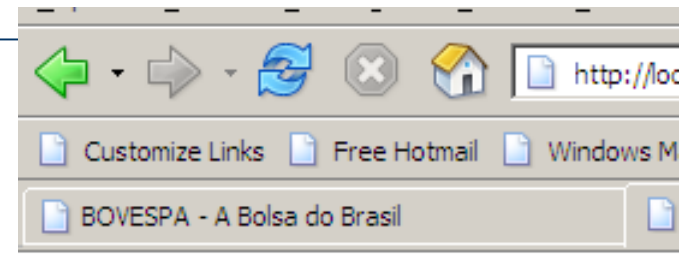
- ♦ **<c:choose>**
 - <c:when ...><c:/when>**
 - ...**
 - <c:when....><c:/when>**
 - < c:otherwise> >< /c:otherwise>****</c:choose>**
- ♦ Utilizado para suprir a ausência do <c:else>.
- ♦ Sintaxe:
 - **<c:when test=“\${nome-do-atributo op-relacional ‘membro’} “>**
 - Operações
 - **</c:when>**

JSTL – exemplo

<c:choose> <c:when> <c:otherwise>

```
List<String> vetor = new ArrayList<String>();
    vetor.add("Flamengo" );
    vetor.add("Vasco" );
    vetor.add("Botafogo" );
    vetor.add("Fluminense" );
    request.setAttribute("times", vetor);
    String login="usuario";
    request.setAttribute("login", login);
```

```
<%@ page import="model.*" %>
<%@ page isELIgnored="false" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
  <body>
    <hr>${times} </hr>
    <c:choose>
      <c:when test="${login eq 'admin'}">
        <jsp:include page="Administrador.jsp"/>
      </c:when>
      <c:otherwise>
        <jsp:include page="UsuarioNormal.jsp"/>
      </c:otherwise>
    </c:choose>
  </body>
</html>
```



[Flamengo, Vasco, Botafogo, Fluminense]
Bem_Vindo Prezado Torcedor



JSTL – `<c:set>`



- ◆ `<c:set></c:set>`
 - Utilizado para atribuir valores para variáveis de escopo ou para *beans*.
 - Pode ou não possuir um corpo.

JSTL –<c:set>

Com variáveis de escopo

◆ Sintaxe (sem corpo)

- <c:set var=“nome-da-variável” scope=escopo value=valor/>

◆ Sintaxe (com corpo)

- <c:set var=“nome-da-variável” scope=escopo>
valor
</c:set>

JSTL –<c:set> Com *bean*

◆ Sintaxe (sem corpo)

- <c:set **target**="\${ Bean}" property="nome-do-atributo" value=valor />

◆ Sintaxe (com corpo)

- <c:set **target**="\${ Bean}" property="nome-do-atributo">
valor
</c:set>



JSTL –<c:remove>



- ◆ <c:remove>
 - Utilizado para remover atributos de escopo.
- ◆ Sintaxe:
 - <c:remove var='nome-do-atributo' scope=escopo />

JSTL –<c:import>

- ◆ <c:import>
 - Utilizado para incluir uma página localizada fora do ambiente da aplicação *web*.
- ◆ Outras formas de inclusão:
 - <%@ include file=“página” %> (estática)
 - <jsp:include page=“página” /> (dinâmica)
- ◆ Sintaxe:
 - <c:import url=“url” /> (dinâmica)

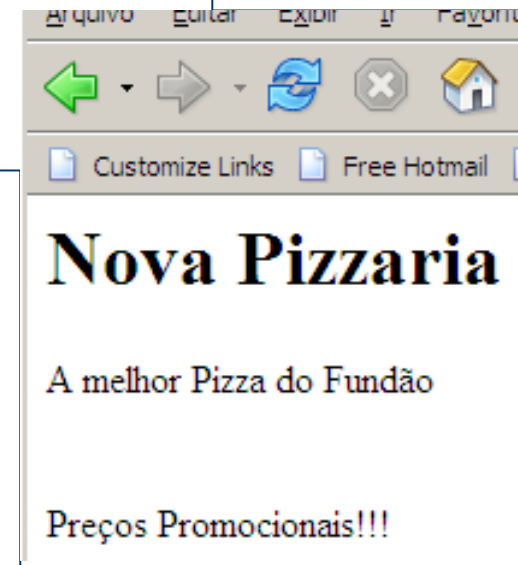
JSTL –<c:param>

- ◆ <c:param>
 - Utilizado para personalizar páginas que estão sendo incluídas.
 - Aplicável com <jsp:include> ou <c:import>.
- ◆ Sintaxe:
 - <c:param name="nome-do-parametro" value="valor-parametro" />
- ◆ Na página a ser incluída:
 - \${param.nome-do-parametro}

JSTL –<c:param>

```
<%@ page isELIgnored="false" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
    <body>
        <c:import url="Cabecalho.jsp">
            <c:param name="mensagem" value="A melhor Pizza do Fundão" />
        </c:import>
        <br></br>
        <br> Preços Promocionais!!! </br>
    </body>
</html>
```

```
<%@ page isELIgnored="false" %>
<html>
    <head>
        <TITLE> Nova Pizzaria </TITLE>
    </head>
    <body>
        <h1 align =center="center"> Nova Pizzaria</h1>
        ${param.mensagem}
    </body>
</html>
```



JSTL – Página de erro

- ◆ Evita que sejam expostos erros para os clientes.
- ◆ `<% @ page isErrorPage="true" %>`
 - Indica que esta é uma página de erro.
- ◆ `<% @ page errorPage="PaginaDeErro.jsp" %>`
 - Indica que caso ocorra um erro o *Container* deve desviar para `PaginaDeErro.jsp`.
- ◆ Inconvenientes:
 - Todas as páginas precisam conter a diretiva.
 - Todos os erros serão apresentados da mesma forma.



JSTL – Página de erro na *web-xml*



- ◆ Define-se na *web-xml* o tipo da exceção e para qual página será feito o desvio.
- ◆ Pode ser sobreposto pela diretiva **errorPage**.
- ◆ Válido para toda a aplicação.

JSTL – Página de erro na *web-xml*

- Tratando uma exceção lançada:

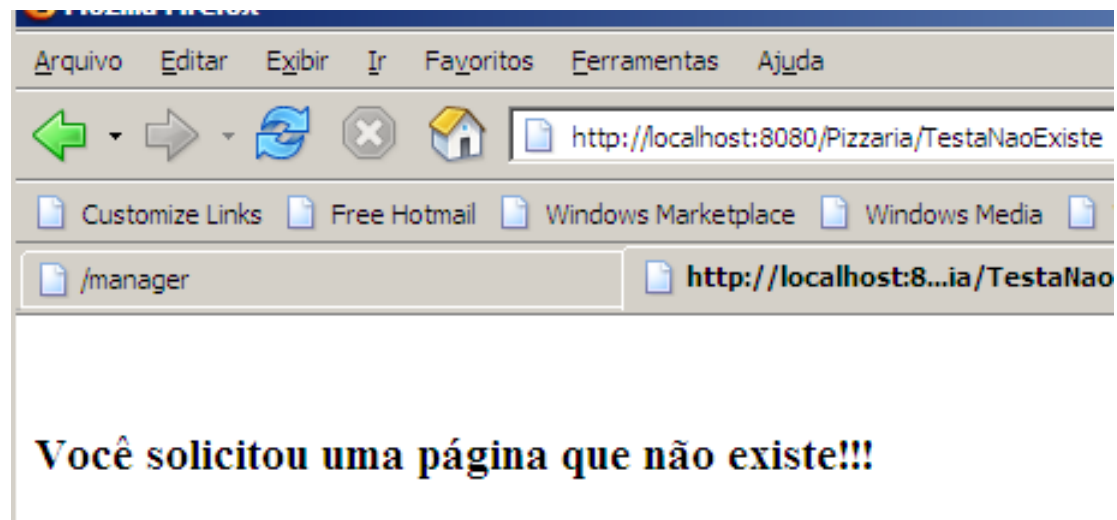
```
<error-page>  
  <exception-type>java.lang.ArithmeticException</exception-type>  
  <location>/PaginaAritmetica.jsp</location>  
</error-page>
```

- Tratando uma página não existente:

```
<error-page>  
  <error-code>404</error-code>  
  <location>/PaginaNaoEncontrada.jsp</location>  
</error-page>
```

JSTL – Página de erro na *web-xml*

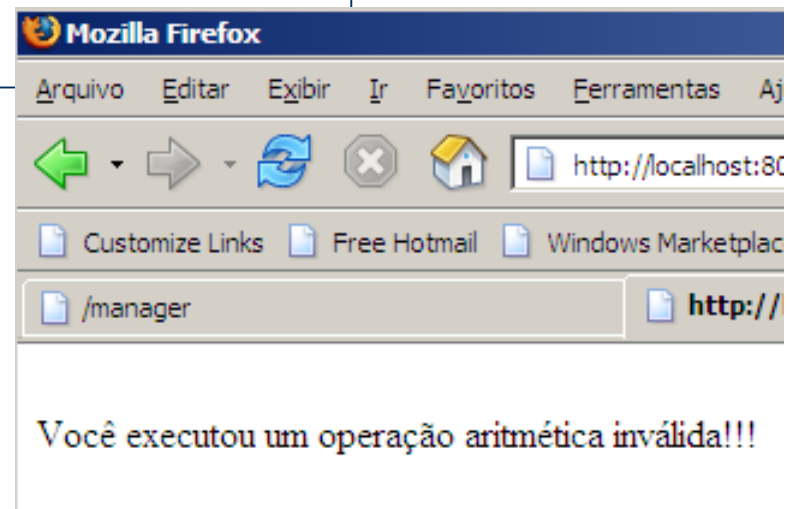
```
<html>
  <body>
    <br><h3> Você solicitou uma página que não existe!!!</h3></br>
  </body>
</html>
```



JSTL – Página de erro na *web-xml*

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws IOException, ServletException {
    int i=10;
    i/=0;
}
```

```
<html>
<body>
  <br> Você executou um operação aritmética inválida!!! </br>
</body>
</html>
```



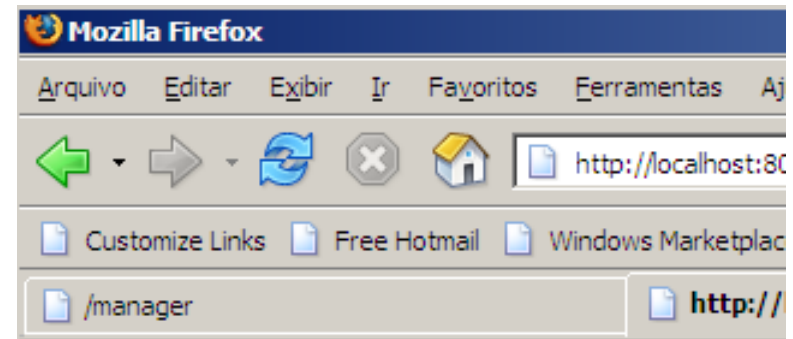
JSTL – <c:catch >

- ◆ <c:catch></c:catch>
 - Permite capturar o erro na própria JSP.
 - Ao ser capturado é desviado para a página de erro.
- ◆ Sintaxe:
 - <c:catch>
 instruções sob risco
 </c:catch>

JSTL – exemplo <c:catch >

```
<%@ page isErrorPage="true" %>
<html>
<body>
  <br> Você executou um operação aritmética inválida!!! </br>
</body>
</html>
```

```
<%@ page errorPage="PaginaAritmetica.jsp" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<body>
  <c:catch>
    <% int i=10/0; %>
    </c:catch> <br></br>
</body>
</html>
```



Você executou um operação aritmética inválida!!!

Personalizando as *tags*

- ◆ Elementos que descrevem uma *tag* estão contidos na TLD.
- ◆ Uma *tag* possui um prefixo e um nome.
- ◆ Sintaxe:
 - `<prefixotag:nomedatag />`
- ◆ Diretiva *taglib*:
 - `<%@ taglib uri="nomedaTLD" prefix="prefixotag" %>`

Personalizando as *tags* – TLD

- A estrutura da TLD:

- Igual para todas as TLDs:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

```
<taglib xmlns="http://java.sun.com/xml/ns/j2ee"
```

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary_2_0.xsd" version="2.0">
```

- Identifica a versão da TLD:

```
•<tlib-version>1.0</tlib-version>
```

- Identifica a TLD:

```
<uri>MinhasTags</uri>
```

Personalizando as *tags* – TLD

- A estrutura da TLD:

- Interior da *tag*:

< tag>

<description> Informa a data corrente </description> // propósito da tag

<name>datadehoje</name> // nome da tag

<tag-class>model.ObtemDataHoje</tag-class> // nome da classe que será ativada pelo *Container*

<body-content>empty</body-content> informa se a *tag* tem ou não corpo

</tag>

< tag>

<description> Informa a hora atual </description> // propósito da tag

<name>horaagora</name> // nome da tag

<tag-class>model.ObtemHoraAgora</tag-class> // nome da classe que será ativada pelo *Container*

<body-content>empty</body-content> informa se a *tag* tem ou não corpo

</tag>

</taglib>

Personalizando as *tags* – <body-content>

- ◆ Valores possíveis para a *tag* <body-content>:
 - empty
 - A *tag* não possui corpo.
 - scriptless
 - A *tag* suporta apenas EL e *actions*.
 - tagdependent
 - A *tag* somente suporta *plain text*.
 - JSP
 - A *tag* suporta qualquer coisa permitida em uma JSP.

Personalizando as *tags* – JSP

◆ Como fica a JSP:

```
<html>
```

```
<body>
```

```
  <%@taglib prefix="estaeminha" uri="MinhasTags" %>
```

```
    <h3> A data é: <estaeminha:datadehoje/> </h3>
```

```
    <h3> A hora é: <estaeminha:horaagora/> </h3>
```

```
</body>
```

```
</html>
```



Personalizando as *tags* – as classes de apoio



- ◆ Devem ser importados os pacotes `javax.servlet.jsp.JspException` e `javax.servlet.jsp.tagext.SimpleTagSupport`.
- ◆ Estão em `Tomcat/common/lib/JSP-API.jar`.
- ◆ Deve herdar da classe `SimpleTagSupport`.
- ◆ Deve ser sobrescrito o método **`doTag()`**.

Personalizando as *tags* – as classes

```
package model;  
import javax.servlet.jsp.JspException;  
import javax.servlet.jsp.tagext.*;  
import java.io.IOException;  
import java.text.DateFormat;  
import java.util.*;
```

```
public class ObtemDataHoje extends SimpleTagSupport{
```

```
    public void doTag() throws JspException, IOException {
```

```
        Calendar data = new GregorianCalendar();  
        Date now = data.getTime();  
        DateFormat shortDate = DateFormat.getDateInstance(DateFormat.SHORT);  
        String hoje = shortDate.format(now);  
        getJspContext().getOut().write(hoje);
```

```
    }
```

```
}
```

Personalizando as *tags* – as classes

```
package model;  
import javax.servlet.jsp.JspException;  
import javax.servlet.jsp.tagext.*;  
import java.io.IOException;  
import java.text.DateFormat;  
import java.util.*;
```

```
public class ObtemHoraAgora extends SimpleTagSupport{  
  
    public void doTag() throws JspException, IOException {  
  
        Calendar data = new GregorianCalendar();  
        Date now = data.getTime();  
        DateFormat shortTime = DateFormat.getInstance(DateFormat.SHORT);  
        String horaatual = shortTime.format(now);  
        getJspContext().getOut().write(horaatual);  
  
    }  
}
```

Personalizando as *tags* – o resultado



A data é: 08/11/06

A hora é: 16:17

Personalizando as *tags* – *tag* com atributo

< tag>

<description> Informa a data corrente </description> // propósito da tag

<name>datadehojecliente</name> // nome da tag

<tag-class>model.ObtemDataHojeCliente</tag-class> // nome da classe que será ativada pelo *Container*

<body-content>empty</body-content> informa se a *tag* tem ou não corpo

<attribute>

 <name> cliente </name>

 <required>true</required>

 <rtexprvalue>true</rtexprvalue>

</attribute>

</tag>

Personalizando as *tags* – *tag* com atributo

◆ Como fica agora a JSP:

```
<html>
```

```
<body>
```

```
<%@ page isELIgnored="false" %>
```

```
<%@ taglib prefix="estaeminha" uri="MinhasTags" %>
```

```
    <h3> <estaeminha:datadehojecliente cliente="${nomeCliente}"/> </h3>
```

```
</body>
```

```
</html>
```


Personalizando as *tags* – o servlet

```
package web;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;

public class DataHojeClienteServlet extends HttpServlet{

    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws IOException, ServletException {

        String nome="João Silva";

        request.setAttribute("nomeCliente",nome);

        RequestDispatcher vista = request.getRequestDispatcher("InformaDataCliente.jsp");
        vista.forward(request, response);

    }
}
```

Personalizando as *tags* – a classe

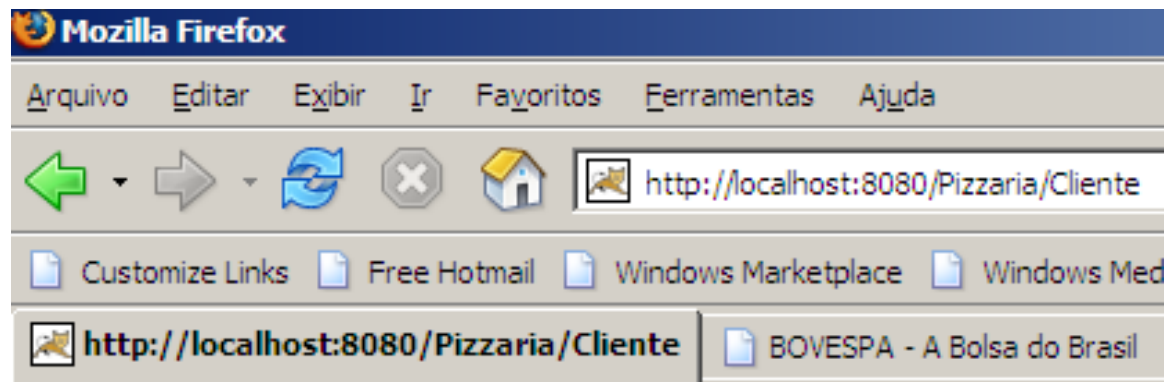
```
package model;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.*;
import java.io.IOException;
import java.text.DateFormat;
import java.util.*;

public class ObtemDataHojeCliente extends SimpleTagSupport{

    private String cliente;
    public void doTag() throws JspException, IOException {
        Calendar data = new GregorianCalendar();
        Date now = data.getTime();
        DateFormat shortDate = DateFormat.getDateInstance(DateFormat.SHORT);
        String hoje = shortDate.format(now);
        getJspContext().getOut().write("Sr.: " + cliente + " hoje é " + hoje);
    }

    public void setCliente(String cliente) {
        this.cliente = cliente;
    }
}
```

Personalizando as *tags* – o resultado



Sr.: João Silva hoje é 08/11/06



Prefixos reservados para JSTLs



- ◆ jsp:
- ◆ jspx:
- ◆ java:
- ◆ javax:
- ◆ servlet:
- ◆ sun:
- ◆ sunw:

Personalizando as *tags* – *tag* com um corpo

< tag>

```
<description> Informa as bandas </description>    // propósito da tag
<name>bandas</name> // nome da tag
<tag-class>model.ObtemBandas</tag-class> // nome da classe que será ativada pelo Container
<body-content>scriptless</body-content>  informa se a tag tem ou não corpo
<attribute>
    <name> cliente </name>
    <required>true</required>
    <rtexprvalue>true</rtexprvalue>
</attribute>
```

</tag>

Personalizando as *tags* – JSP

◆ Como fica a JSP:

```
<html>
```

```
<body>
```

```
<%@ page isELIgnored="false" %>
```

```
<%@ taglib prefix="estaeminha" uri="MinhasTags" %>
```

```
    <h3> <estaeminha:bandas cliente="{nomeCliente}"> </h3>
```

```
        <tr><td> <p>${bandas}</p> </td></tr>
```

```
    </estaeminha:bandas>
```

```
</body>
```

```
</html>
```

Personalizando as *tags* – a classe

```
package model;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.*;
import java.io.IOException;
import java.util.*;

public class ObtemBandas extends SimpleTagSupport{

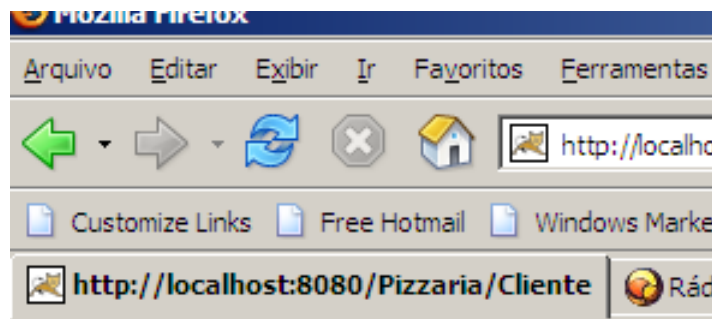
    private String cliente;
    private Vector<String> bandas;
    public void doTag() throws JspException, IOException {
        bandas = new Vector<String>();
        bandas.add("Led Zeppelin");
        bandas.add("Barão Vermelho");
        bandas.add("Deep Purple");
        bandas.add("Black Sabbath");
        bandas.add("Yes");
        getJspContext().getOut().write("Sr. " + cliente + " as bandas são:");
        for (String nome:bandas){
            getJspContext().setAttribute("bandas", nome);
            getJspBody().invoke(null);
        }
    }
    public void setCliente(String cliente) {
        this.cliente = cliente;
    }
}
```

Personalizando as *tags* – o servlet

```
package web;  
import javax.servlet.*;  
import javax.servlet.http.*;  
import java.io.*;  
import java.util.*;
```

```
public class BandasClienteServlet extends HttpServlet{  
  
    public void doGet(HttpServletRequest request,  
                      HttpServletResponse response)  
        throws IOException, ServletException {  
  
        String nome="João Silva";  
  
        request.setAttribute("nomeCliente",nome);  
  
        RequestDispatcher vista = request.getRequestDispatcher("InformaBandas.jsp");  
        vista.forward(request, response);  
  
    }  
}
```


Personalizando as *tags* – o resultado



Sr. João Silva as bandas são:

Led Zeppelin

Barão Vermelho

Deep Purple

Black Sabbath

Yes

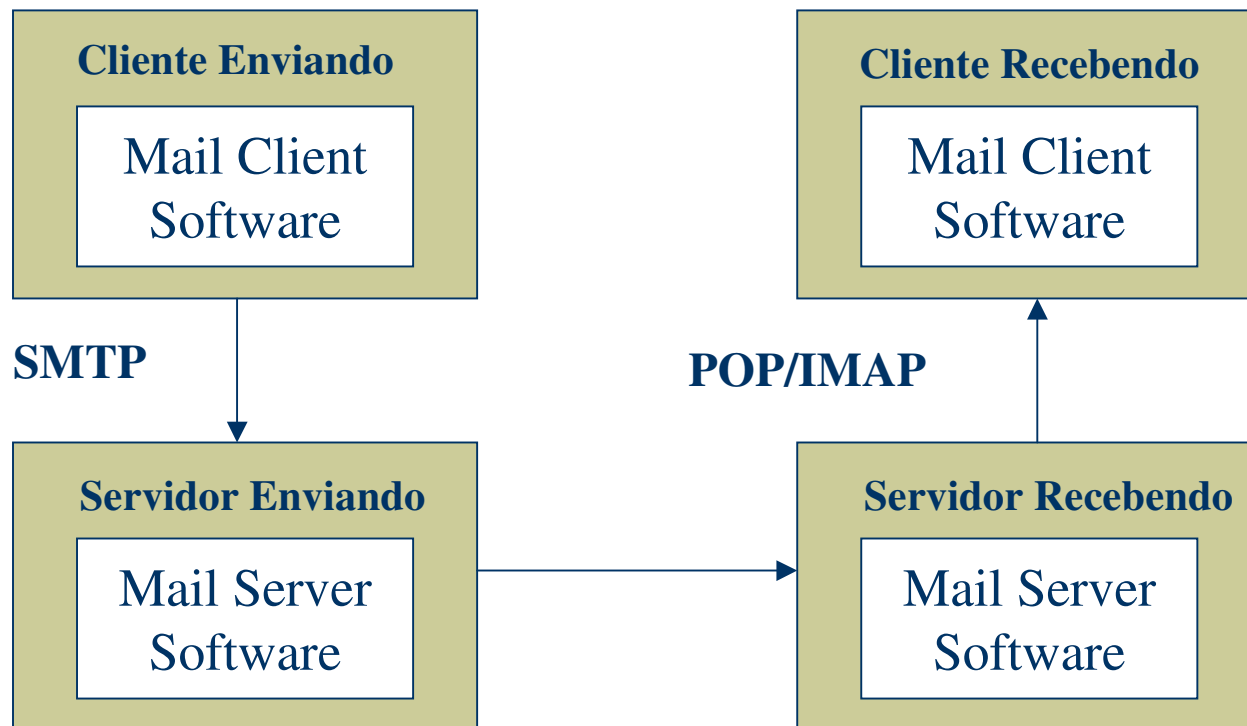


Usando a API JavaMail para enviar *email*



- ◆ A API JavaMail é uma interface que facilita os desenvolvedores escreverem código Java que enviam, automaticamente, e-mails.
- ◆ Interage com a API JavaBeans Activation Framework.

Enviando *email* - funcionamento



Ex.: software de email : OutLook Express, Eudora etc.

Email server: fornecido pelo provedor de acesso à internet.

Usando a API JavaMail para enviar *email* - protocolos

- ◆ SMTP (Simple Mail Transfer Protocol)
 - Utilizado para enviar uma mensagem de um servidor de *email* para outro.
- ◆ POP (Post Office Protocol)
 - Utilizado para recuperar mensagens de um servidor de *email*. Transfere todas as mensagens de um servidor de *email* para o cliente. Está em sua versão 3 e é chamado de POP3.
- ◆ IMAP (Internet Message Access Protocol)
 - Utilizado por serviços de *email* baseados na *web* tais como Yahoo e Hotmail. Permite ao *web browser* ler mensagens armazenadas em um diretório de um servidor de *email*. Está em sua versão 4 e é chamado de IMAP4.
- ◆ MIME (Multipurpose Internet Message Extension)
 - Especifica o tipo de conteúdo que pode ser enviado como mensagem ou como arquivo anexado.

Usando a API JavaMail para enviar *email*

- ◆ Para utilizarmos as APIs são necessários os seguintes arquivos:
 - mail.jar
 - Contém as classes Java da API JavaMail.
 - activation.jar
 - Contém as classes Java da API JavaBean Activation Framework.
- ◆ Devem residir no diretório jre/lib/ext do Java SDK.
- ◆ No J2EE estes arquivos já estão disponíveis.



Os 4 passos para enviar um *email*



1. Criar um sessão de *email*.
2. Criar uma mensagem.
3. Endereçar a mensagem (identificar remetente e destinatário).
4. Enviar a mensagem.

Criando uma sessão de *email*

- ◆ Atividades necessárias:
 - Identificar o *host* onde reside o servidor de *email*. Usar “localhost” se estiver na mesma máquina da aplicação.
 - Criar um objeto Properties que conterà as propriedades necessárias para enviar um *email*.
 - Especificar uma propriedade para a sessão utilizando-se o método *put* de Properties.
 - Criar um objeto *Session*, que define a sessão de *mail*, chamando o método `getDefaultInstance`.



Criando uma sessão de *email*



- ♦ A classe Session integra o pacote javax.mail.
- ♦ A classe Properties integra o pacote javax.util.

Criando uma mensagem

◆ Atividades necessárias:

- Utilizar a classe `MimeMessage` para criar uma mensagem. Fica armazenada no pacote `javax.mail.internet`.
- Criar um objeto `MimeMessage` fornecendo um objeto `Session`.
- Utilizar um `Authenticator` para validar o usuário e a senha.
- Utilizar os métodos `setSubject` e `setText` para especificar o propósito do *email* e seu texto. O texto enviado assume o tipo `text/plain` para o MIME.
- Utilizar o método `setContent` se desejar anexar um documento.



Endereçando a mensagem



- ◆ Atividades necessárias:
 - Definir um endereço de *email* através da classe `InternetAddress`, integrante do pacote `javax.mail.internet`.
 - Utilizar o método `setFrom` do objeto `MimeMessage` para atribuir o endereço “From”.
 - Utilizar o método `setRecipient` para atribuir o endereço “To”.



Enviando uma mensagem



- ◆ Atividades necessárias:
 - Utilizar o método *send* da classe Transport para enviar a mensagem para o *email server*.

Código que envia um e-mail e anexa um arquivo

```
import java.util.*;
import java.io.*;
import javax.mail.*;
import javax.mail.internet.*;
import javax.activation.*;

public class TestaEmail {

    private String usuario;
    private String pswd;
    private String from;
    private String to;
    private String assunto;
    private String texto;

    public TestaEmail(String usuario, String pswd, String from, String to, String assunto, String texto)

        this.usuario=usuario;
        this.pswd=pswd;
        this.from=from;
        this.to=to;
        this.assunto=assunto;
        this.texto=texto;

    }
```

Código que envia um e-mail e anexa um arquivo

```
public void postMail() throws Exception{

    boolean debug=true;

    // Criando a sessão autenticando o usuário e senha
    String host="correio.nce.ufrj.br";
    Properties props=System.getProperties();
    props.put("mail.smtp.host",host);
    props.put("mail.smtp.auth","true");
    Authenticator auth=new SMTPAuthenticator(usuario,pswd);
    Session session=Session.getDefaultInstance(props,auth);
    session.setDebug(debug);

    // Criando a mensagem e anexando um arquivo
    BodyPart messageBodyPart= new MimeBodyPart();
    messageBodyPart.setText("Teste Attach");
    Multipart multipart=new MimeMultipart();
    multipart.addBodyPart(messageBodyPart);
    messageBodyPart=new MimeBodyPart();
    DataSource source=new FileDataSource("c:/tesemestrado/Lookup.java");
    messageBodyPart.setDataHandler(new DataHandler(source));
    messageBodyPart.setFileName("c:/tesemestrado/Looukp.java");
    multipart.addBodyPart(messageBodyPart);

    // Endereçando a mensagem
    MimeMessage message=new MimeMessage(session);
    message.setFrom(new InternetAddress(from));
    message.addRecipient(Message.RecipientType.TO, new InternetAddress(to));
    message.setSubject(assunto);
    message.setText(texto);
    message.setContent(multipart);

    // Enviando a mensagem
    Transport.send(message);

}
```

Código que envia um e-mail e anexa um arquivo

```
private class SMTPAuthenticator extends Authenticator{  
    private String usuario;  
    private String pswd;  
  
    public SMTPAuthenticator(String usuario, String pswd){  
        this.usuario=usuario;  
        this.pswd=pswd;  
    }  
  
    public PasswordAuthentication getPasswordAuthentication(){  
        return new PasswordAuthentication(usuario,pswd);  
    }  
}
```

Enviando e-mail – Outra Solução

```
import java.io.IOException;
import java.io.PrintStream;
import sun.net.smtp.SmtpClient;

public class EnviaEmail {

    public void sendEmail() {
        String from = "mariaeduarda@dominio.com.br";
        String to = "joao@gmail.com";
        try {

            SmtpClient client = new SmtpClient("nome-servidor -smtp");
            client.from(from);
            client.to(to);
            PrintStream msg = client.startMessage();
            msg.println("Subject: Assunto");
            msg.print("\r\n");
            msg.println("frase conteudo");
            msg.println("frase conteudo");
            client.closeServer();

        } catch (IOException e) {
            System.out.println("error" + e);
        }
    }
}
```



Struts



- ◆ Struts é um *framework*, *free* e *open-source*.
- ◆ Projetado para ajudar desenvolvedores a criarem aplicações *web* em Java que utilizam a arquitetura MVC.
- ◆ Desenvolvido pela Apache Software Foundation.
- ◆ Suporta AJAX.

Struts

- ◆ Última versão 1.3.5. Utilizaremos a 1.2.9.
- ◆ Site: <http://struts.apache.org>.
- ◆ Necessário fazer o *download* dos pacotes do struts.
- ◆ Copiar para WEB-INF/lib/ os pacotes struts.jar, commons-beanutils.jar e commons-digester.jar.
- ◆ Copiar também para jre/lib/ext/ , do java, o pacote struts.jar.

Struts

◆ Componentes:

■ **ActionServlet**

- Um único por aplicação, fornecido pelo *struts*. Também chamado de Front Controller.

■ **FormBeans**

- Um para cada formulário HTML da aplicação. São Java *beans*. O *struts* popula o *bean* com os parâmetros do formulário. Aqui é feita a crítica.

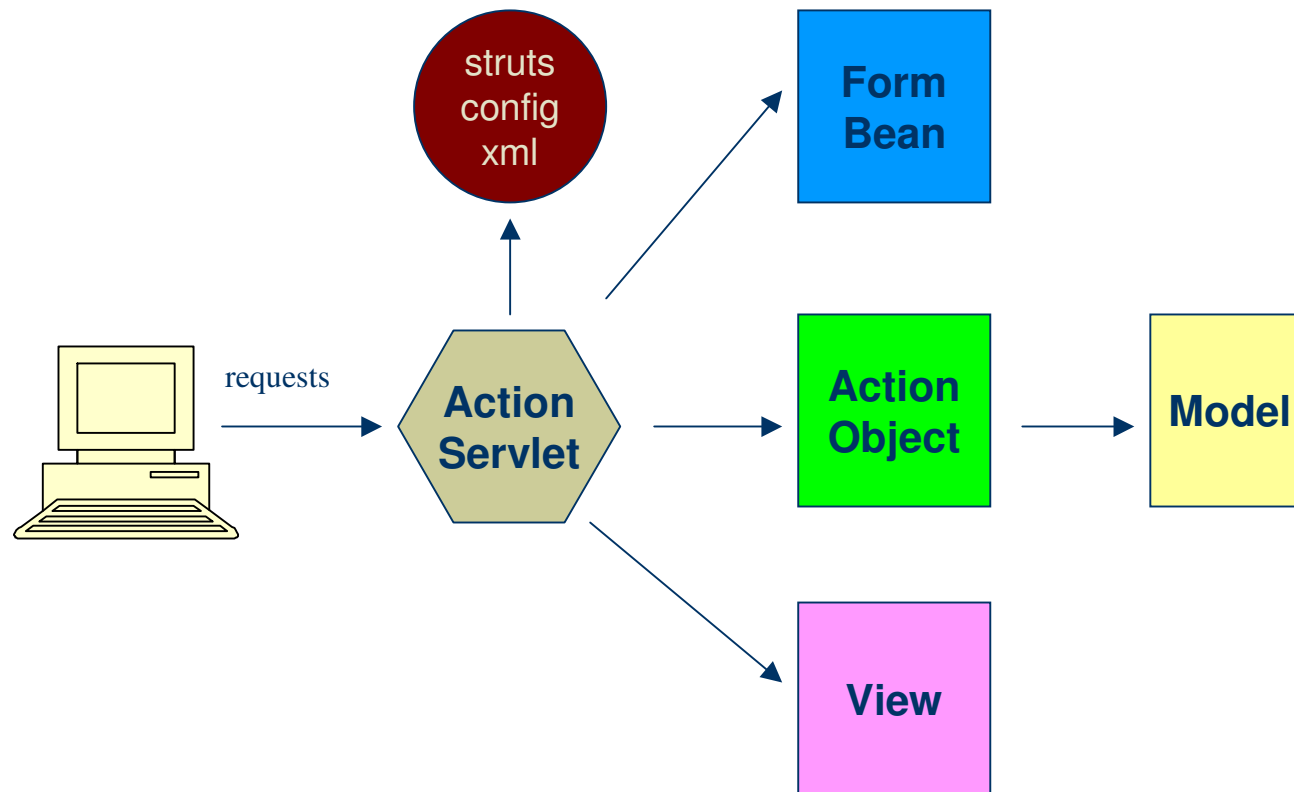
■ **ActionObjects**

- Onde são obtidos os dados que foram validados pelo FormBeans. Semelhante aos *servlets* convencionais.

■ **struts-config.xml**

- É o descritor de distribuição do *struts*. Relaciona *requests* URL com Actions, Actions com FormBeans e Actions com *views*. Fica no mesmo diretório do web-xml.

Struts



Struts

Exemplo - Form Bean

```
package web;
import model.*;
import javax.servlet.http.HttpServletRequest;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionMapping;

public class RecomendacaoMusicalForm extends ActionForm{

    private String estilo;
    public void setEstilo(String estilo){
        this.estilo = estilo;
    }
    public String getEstilo(){
        return this.estilo;
    }
    public ActionErrors validate(HttpServletRequest request, ActionMapping mapping){
        ActionErrors erro = new ActionErrors();
        if (estilo==null){
            erro.add("estilo", new ActionError("error.colorField.notValid"));
        }
        return erro;
    }
}
```

Struts

Exemplo - Action

```
package web;
import model.*;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import java.util.*;

public class Recomendacao extends Action{

    public ActionForward execute(HttpServletRequest request, HttpServletResponse response,
    ActionMapping mapping, ActionForm form) {
        RecomendacaoMusicalForm meuform = (RecomendacaoMusicalForm) form;
        SelecaoMusical selecao = new SelecaoMusical();
        ArrayList<String> retorno = selecao.getLista(meuform.getEstilo());
        request.setAttribute("listaRecomendada" , retorno);
        return mapping.findForward("apresenta_vista01");
    }
}
```

Struts

Exemplo – web-xml

```
<web-app>
  <servlet>
    <servlet-name>EntradaPrincipal</servlet-name>
    <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>

    <init-param>
      <param-name>config</param-name>
      <param-value>/WEB-INF/struts-config.xml</param-value>
    </init-param>

    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>EntradaPrincipal</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>
</web-app>
```

Struts

Exemplo – struts-config.xml

```
<struts-config>
```

```
  <form-beans>
```

```
    <form-bean name="RecomendacaoMusical" type="web.RecomendacaoMusicalForm" />
```

```
  </form-beans>
```

```
  <action-mappings>
```

```
    <action path="/Principal" type="web.Recomendacao" name="RecomendacaoMusical" scope="request"
      validate="true" input="/erro.jsp" >
```

```
      <forward name="apresenta_vista01" path="/sugestao.jsp" />
```

```
    </action>
```

```
  </action-mappings>
```

```
  <message-resources parameter="ApplicationResources" null="false" />
```

```
</struts-config>
```



FIM

Na luta pela vida somente os mais fortes e os mais aptos conseguem sobreviver, e a própria natureza se incumbe de proceder a essa seleção natural.

Charles Darwin