
INSTITUTO
VIANNA JUNIOR

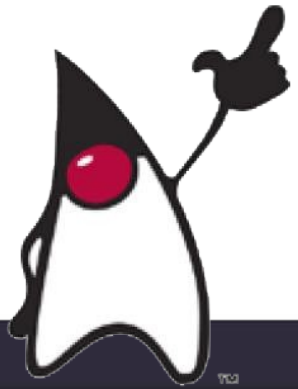
INSTITUTO VIANNA JÚNIOR

Java Web

Daves Martins

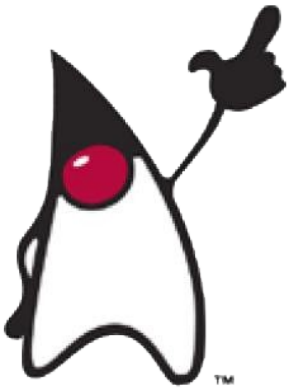
davesmartins@yahoo.com.br

Mestre em Computação de Alto Desempenho pela UFRJ
Especialista em Banco de Dados
Analista Web



INSTITUTO VIANNA JÚNIOR

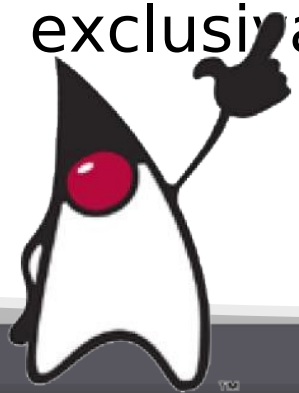
Servlet



SERVLET

- O Servlet foi introduzido pela Sun Microsystems em 1996, aprimorando a funcionalidade de servidores da Web.

- É considerado eficiente para desenvolver soluções baseadas na Web, interagir com o banco de dados em favor de um cliente, gerar dinamicamente documentos HTML personalizados a serem exibidos por navegadores e manter informações de sessão exclusivas de cada cliente.



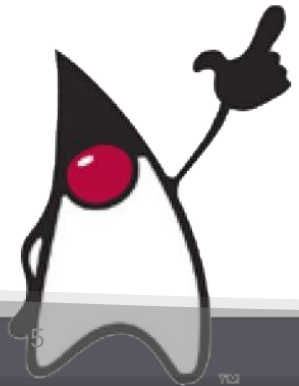
SERVLET

- Toda servlet é basicamente uma classe que processa requisições e respostas dinâmicamente.

- Uma Servlet necessita de um container Web para ser executado.

- Uma servlet funciona como um pequeno servidor que recebe chamadas de diversos clientes.

- Uma primeira idéia da servlet seria que cada uma delas é responsável por uma página, sendo que ela lê dados da requisição do cliente e responde com outros dados (html, gif etc).



5

SERVLET

- Resumindo, cada servlet é um objeto java que recebe tais requisições (request) e retorna algo (response), como por exemplo uma página html ou uma imagem do formato jpg.

- Diversas requisições podem ser feitas à uma mesma servlet ao mesmo tempo em um único servidor, por isso ela é mais rápida que um programa CGI comum. A especificação da servlet cita algumas vantagens que possui sobre o antigo CGI.

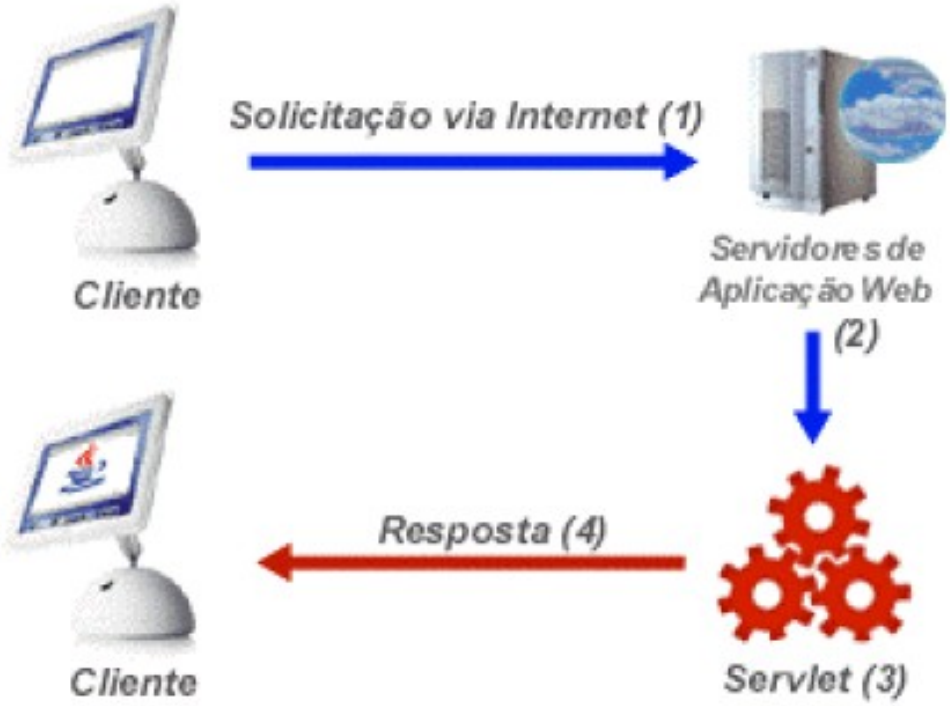
Servlet

Introdução aos Servlets

Dentre as diversas tecnologias J2EE utilizadas no desenvolvimento de páginas para a Internet, a tecnologia *Servlet* foi o ponto de partida para a o estabelecimento de toda uma especificação na qual deu origem às tecnologias mais recentes como a JSP, JSLT, *Struts* e JSF.

Sua estrutura foi criada para permitir que informações solicitadas via HTTP, normalmente realizadas por um cliente utilizando um navegador de Internet, fossem processadas do lado dos servidores.

A figura abaixo demonstra uma solicitação realizada por um cliente via Internet e processada por um *Servlet*.



Servlets

- São o bloco básico do desenvolvimento web em Java
- Servlets são classes hospedadas num servidor que respondem a requisições HTTP
- De maneira informal e resumida, servlets são classes java que geram páginas html
- Um servlet é uma especialização da classe javax.servlet.http.HttpServlet

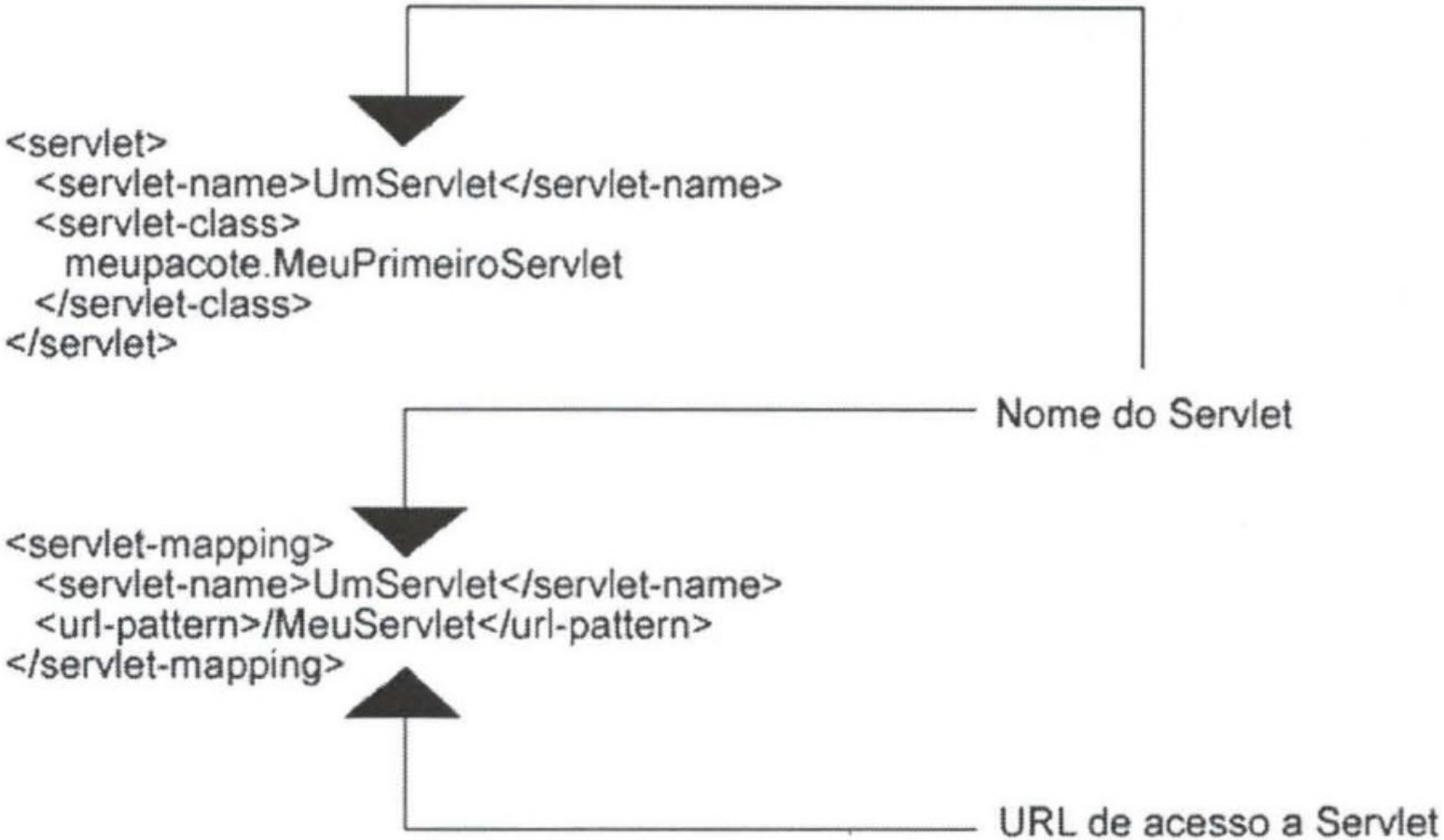
Servlets

```
package br.com.siriusnet.cursojava;

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

public class PrimeiroServlet extends HttpServlet {
   public void doGet (  HttpServletRequest req,
                        HttpServletResponse res  )
   throws IOException {
      ServletOutputStream out = res.getOutputStream();
      res.setContentType("text/html");
      out.println("<html><head><title>Básico</title></
head>");
      out.println("<body>Você está no endereço: " +
req.getRemoteAddr() + "!!</body></html>");
   }
}
```

Mapear um Servlet via web.xml



Descritor de Implantação web.xml

```
<web-app 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
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
version="2.4">
      <servlet>
            <servlet-name>primeiro</servlet-name>
            <servlet-class>
br.com.cursojava.PrimeiroServlet</servlet-class>
      </servlet>
      <servlet-mapping>
            <servlet-name>primeiro</servlet-name>
            <url-pattern>/alou</url-pattern>
      </servlet-mapping>
</web-app>
```

Mapear um Servlet via web.xml

- **Exemplo**

```
<servlet>
  <servlet-name>primeiraServlet</servlet-name>
  <servlet-class>OiMundo</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>primeiraServlet</servlet-name>
  <url-pattern>/oi</url-pattern>
  <url-pattern>/ola</url-pattern>
</servlet-mapping>
```

Mapear um Servlet via annotations

- **Annotations**
  - **Implementado a partir do JDK1.5**
  - **Iniciam com o sinal arroba "@" = "at"**
  - **Utilizadas para definir metadados (informações sobre o próprio código) que podem ser posteriormente interpretadas por um compilador ou pré compilador que irá realizar alguma tarefa pré definida**

Mapear um Servlet via annotations

- @WebServlet **– annotation usada para registrar um Servlet em um container**
  - **Alguns atributos**
    - name – **nome da servlet**
    - description – **descrição da servlet**
    - value – **URL de acesso a servlet (padrão)**
    - urlPatterns – **mesmo propósito de** value**, geralmente utilizado quando mais de uma url é especificada**
    - loadOnStartup – **carrega a servlet na inicialização do container**
  - **Não é possível ter ao mesmo tempo os atributos** value **e** urlPatterns

Mapear um Servlet via annotations

- Exemplos
  - **@WebServlet(“/HelloServlet”)**
  - **@WebServlet(value="/hello", name="hello-servlet")**
  - **@WebServlet({“/HelloServlet”, “/member/*”})**
  - **@WebServlet(name=“Hello”, urlPatterns={“/HelloServlet”, “/*.html”})**

```
@WebServlet(name="OlaServlet", urlPatterns={"/oi",
"/ola"})
public class OlaServlet extends HttpServlet{
  //...
}
```

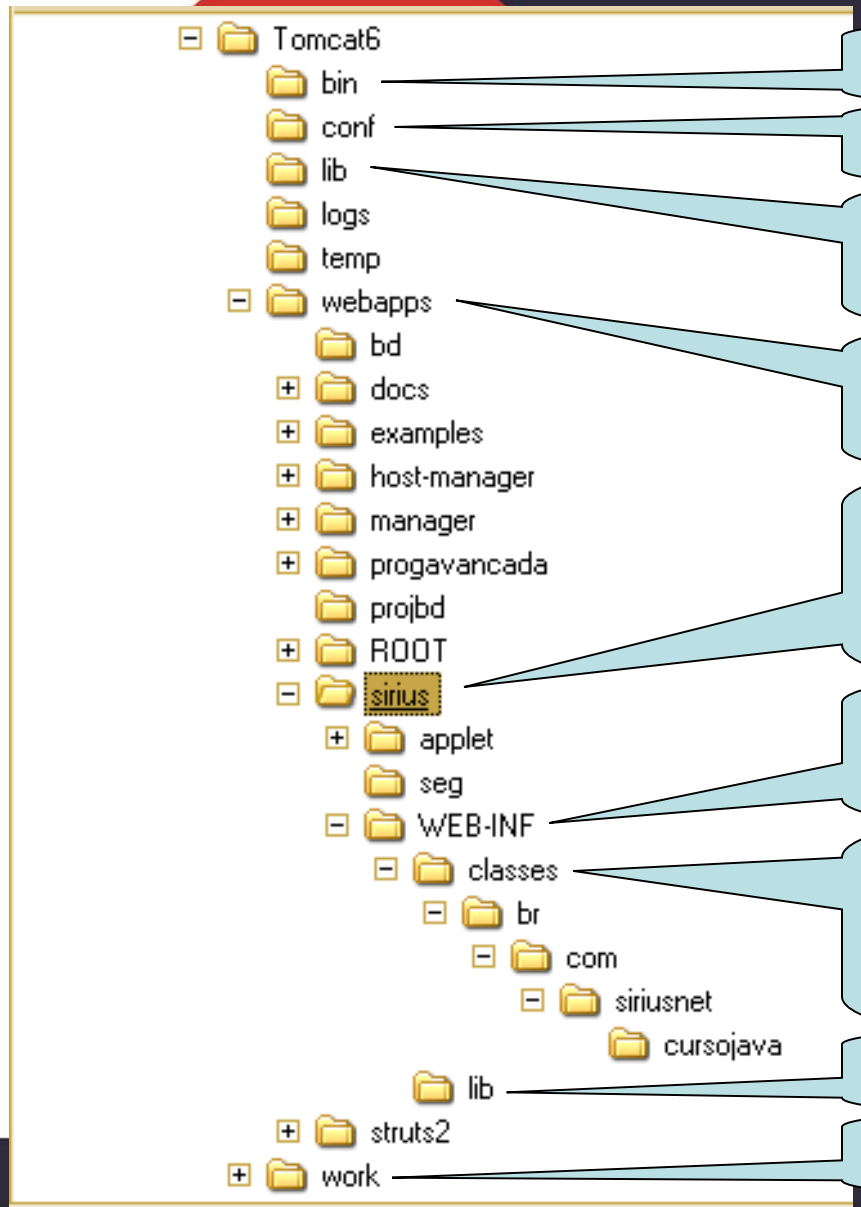
Sintaxe de Mapeamentos

- **Mapeamento exato**
  - **Não aceita /nome/ ou /nome/x na requisição**
    <url-pattern>/nome</url-pattern>
    <url-pattern>/nome/subnome</url-pattern>
- **Mapeamento para servlet default**
  - **Servlet é chamado se nenhum dos outros mapeamentos existentes combinar com a requisição**
    <url-pattern>/</url-pattern>
- **Mapeamento de caminho**
  - **Aceita texto adicional (path info) após nome do servlet na requisição**
    <url-pattern>/nome/*</url-pattern>
    <url-pattern>/nome/subnome/*</url-pattern>
- **Mapeamento de extensão**
  - **Arquivos com a extensão serão redirecionados ao servlet**
    <url-pattern>*.ext</url-pattern>

Acessando um Servlet

- **O acesso a esse Servlet pelo navegador será digitando MeuServlet no caminho da sua aplicação Web:**
  - **http://localhost:8080/<contexto>/<url da servlet>**
  - **Exemplo**
    - **http://localhost:8080/HelloWorld/hello**

Diretórios do Tomcat



Executáveis

Configuração do Tomcat

Bibliotecas compartilhadas entre aplicações

Diretório de implantação de aplicações

Contexto de uma aplicação específica; pode conter arquivos html, jsp, etc

Diretório que contém o descritor de implantação web.xml

Diretório que contém as compilações dos servlets (.class) de uma aplicação

Bibliotecas de uma aplicação

Temporário das aplicações

Servlets – Distribuição de Aplicações

- Podemos colocar toda aplicação Web dentro de um arquivo .war
- WAR é análogo ao JAR para programas Java; deve conter toda a árvore de diretórios da aplicação, exceto a raiz
- No Tomcat, para implantarmos um WAR basta que copiemos o arquivo para o diretório “<tomcat>/webapps”
- Na implantação não é necessário que o Tomcat seja reiniciado

Métodos herdados de HTTPServlet

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.IOException;
public class CicloDeVida extends HttpServlet {
    // Metodo chamado para inicializar o servlet, O metodo da superclasse
       precisa ser chamado quando este metodo eh sobrescrito
    public void init(ServletConfig arg0) throws ServletException { }
    // Este metodo eh invocado quando solicitacoes chegam
    // Este metodo nao precisa ser obrigatoriamente sobrescrito
    public void service(ServletRequest arg0, ServletResponse arg1) throws
       ServletException, IOException { }
    // Chamado por service() quando solicitacoes GET sao recebidas
    public void doGet(HttpServletRequest arg0, HttpServletResponse arg1)
       throws ServletException, IOException { }
    // Chamado por service() quando solicitacoes POST sao recebidas
    public void doPost(HttpServletRequest arg0, HttpServletResponse arg1)
       throws ServletException, IOException { }
    // Chamado quando o servlet eh destruido pelo conteiner
    public void destroy() { }
}
```

Tratamento de Mensagens GET e POST

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.IOException;
public class CicloDeVida extends HttpServlet {
    public void doGet(HttpServletRequest arg0, HttpServletResponse
      arg1) throws ServletException, IOException {
      this.processa(arg0, arg1);
    }

    public void doPost(HttpServletRequest arg0, HttpServletResponse
      arg1) throws ServletException, IOException {
      this.processa(arg0, arg1);
    }

    public void processa(HttpServletRequest arg0,
      HttpServletResponse arg1) throws ServletException, IOException
       {
       // Processamento da requisição
    }
}
```

Servlets – Ciclo de Vida

- O contêiner recebe uma solicitação;
- O objeto servlet existe?
  - Se não existe, o contêiner instancia o servlet e o inicializa com o método *init()*;
- O contêiner invoca o metodo *service()* do servlet;
- A requisição é enviada para o método *service()* do cliente, o qual invoca o método *doXXX()* apropriado;
- A resposta é então retornada para o cliente.

HTTPServletRequest e HTTPServletResponse

- Estas 2 classes são cruciais para o tratamento de requisições através de servlets
- A classe HTTPServletRequest representa uma requisição e contém todas as informações do cliente solicitante
  - IP do cliente, cookies, header do HTTP, etc
- A classe HTTPServletResponse é utilizada para preparar a resposta ao cliente
- Com esta classe podemos:
  - Adicionar novos cookies, adicionar uma nova entrada no cabeçalho HTTP, redirecionar uma requisição para tratamento por outro recurso, etc

Servlets – Alguns Métodos

- HttpServletRequest
  - *String getParameter(String name)* – Retorna parâmetros de solicitação de campos de um formulário;
  - *String getRemoteAddress()* – Retorna o endereço IP da máquina cliente;
  - *Cookie [] getCookies()* – Retorna um array de todos os cookies;
  - *String getHeader(String name)* – Retorna o valor da entrada *name* no cabeçalho HTTP como uma string.

Servlets – Alguns Métodos

- HttpServletRequest
  - *RequestDispatcher getRequestDispatcher (String url)* – Cria um “dispatcher” para o recurso *url*; assim podemos encaminhar a requisição utilizando o método *forward()* da classe RequestDispatcher
  - *HttpSession getSession()* – Retorna uma referência para a sessão associada à requisição, ou cria uma nova caso a sessão não exista

Servlets – Alguns Métodos

- HttpServletResponse
  - *java.io.ServletOutputStream getOutStream()* – Retorna o *outputstream* de forma que possamos gravar dados no cliente;
  - *void setContentType(String mime)* – Configura o tipo de conteúdo de saída; Lista completa dos tipos MIME em http://www.iana.org/assignments/mediatypes/index.html

Servlets – Alguns Métodos

- HttpServletResponse
  - *void addCookie(Cookie cookie)* – Adiciona um *cookie* ao cabeçalho da resposta;
  - *void addHeader(String name, String value)* – Adiciona o cabeçalho especificado à resposta;
  - *void sendRedirect(String url)* – Redireciona uma solicitação para outro recurso (servlet ou página www).

Exercício para casa

- **Criar um servlet que exibe uma saudação dependendo da hora**
  - **Se hora >= 5 e hora < 13, mostre “Bom dia!!!”**
  - **Se hora >=13 e hora < 19, mostre “Boa Tarde!!!”**
  - **Se hora >=19 e hora < 24, mostre “Boa Noite!!!”**
  - **Senão, mostre “Boa Madrugada!!!”**
- **A servlet deve ser acessado pela URL** http://localhost:8080/aula/saudacao
- **Para pegar a hora atual:**
  - **Calendar now = Calendar.*getInstance();***
  - **now.get(Calendar.*HOUR_OF_DAY)***

Processando Formulários

```
<html>
  <head>Um Form Básico</head>
  <body>
      <h1>Entre com seus dados:</h1>
      <form action="processaForm" method="POST">
          Tratamento: <select size="1" name="titulo">
              <option>Sr.</option>
              <option>Sra.</option>
          </select>
          Nome: <input type="text" name="nome" size="20"><br>
          Cidade: <input type="text" name="cidade" size="20"><br>
          <p>Selecione seus interesses:</p>
          <input type="checkbox" name="interesses"
   value="esportes">Esportes<br>
          <input type="checkbox" name="interesses"
   value="musica">Musica<br>
          <input type="checkbox" name="interesses"
   value="leitura">Leitura<br>
          <p><input type="submit" value="Envie"></p>
      </form>
  </body>
</html>
```

Processando Formulários

```
public class ProcessaFormulario extends HttpServlet {
   private void processa(HttpServletRequest req, HttpServletResponse res)
   throws IOException {
   ServletOutputStream out = res.getOutputStream();
   res.setContentType("text/html");
   String tratamento = req.getParameter("titulo");
   String nome = req.getParameter("nome");
   String cidade = req.getParameter("cidade");
   String interesses[] = req.getParameterValues("interesses");
   out.println("<html><head><title>Resp do Servlet</title></head>");
   out.println("<body>");
   out.println("Olá " + tratamento + " " + nome);
   out.println("Você mora em " + cidade + " e seus interesses são: ");
   out.println("<ul>");
   for (String interesse : interesses) {
   out.println("<li>");
   out.println(interesse);
   out.println("</li>");
       }
   out.println("</ul></body></html>");
   } }
```

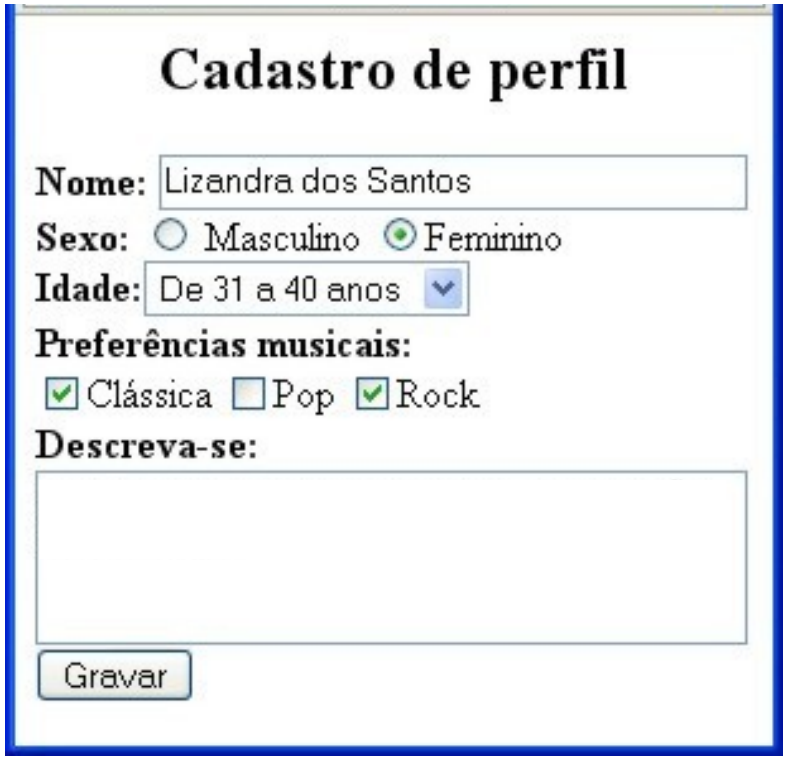
Processando Formulários

```
<web-app 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
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
version="2.4">
      <servlet>
            <servlet-name>segundo</servlet-name>
            <servlet-class>
br.com.cursojava.ProcessaFormulario</servlet-class>
      </servlet>
      <servlet-mapping>
            <servlet-name>segundo</servlet-name>
            <url-pattern>/processaForm</url-pattern>
      </servlet-mapping>
</web-app>
```

Exercício

- **Crie uma nova aplicação para a web através da qual os visitantes possam registrar seu perfil.**
- **Os dados a serem captados são os seguintes:**
  - **Nome: um campo de texto.**
  - **Sexo: dois botões de rádio.**
- **Idade: uma caixa de combinação contendo as opções abaixo.**
  - **Até 18 anos**
  - **De 19 a 30 anos**
  - **De 31 a 40 anos**
  - **De 41 a 50 anos**
  - **Mais de 50 anos**
- **Preferências musicais: caixas de checagem.**
- **Descrição: uma área de texto.**
- **A aplicação deve conter apenas um servlet e uma página HTML.**

Exercício

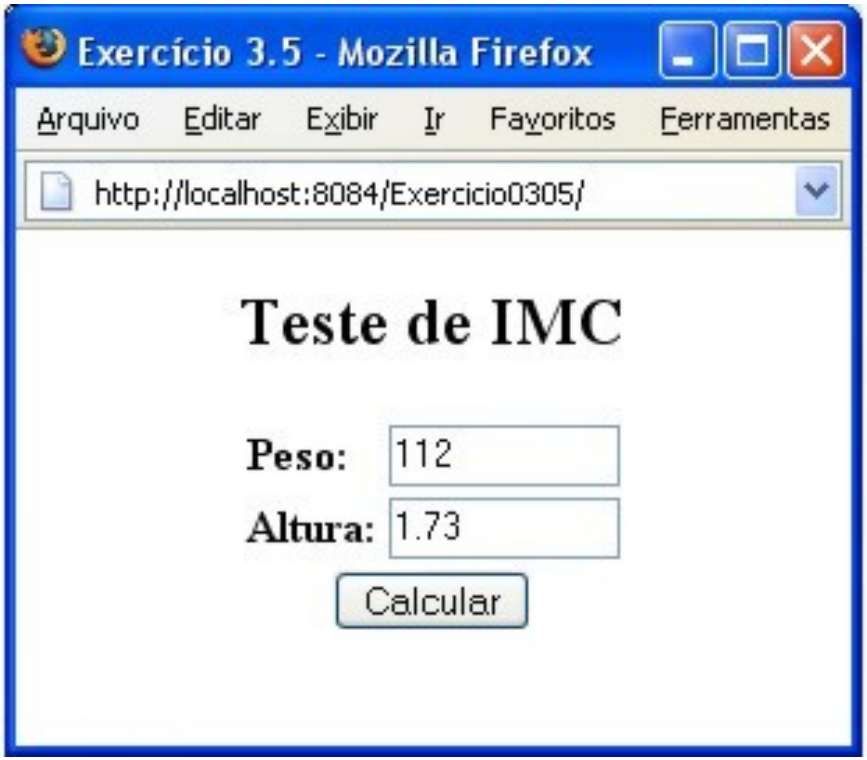

Cadastro de perfil
Nome: Lizandra dos Santos
Sexo: Masculino Feminino
Idade: De 31 a 40 anos
Preferências musicais:
Clássica Pop Rock
Descreva-se:
Gravar

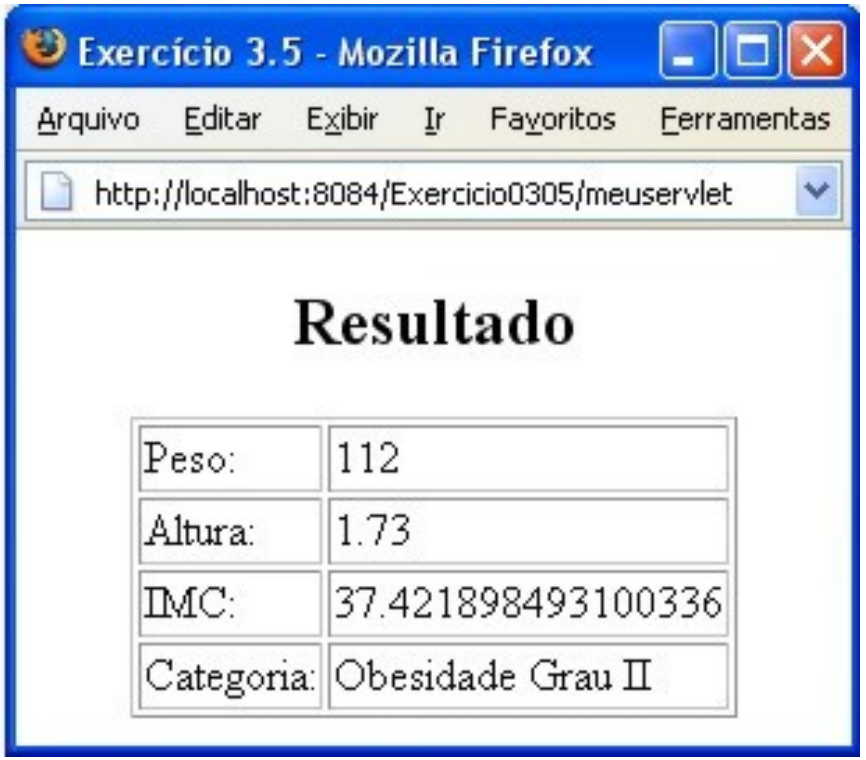

Seu perfil
Nome: Lizandra dos Santos
Sexo: Feminino
Idade: De 31 a 40 anos
Preferências musicais: Clássica, Rock
Descrição:

Exercício

- Crie uma nova aplicação para a web composta por um servlet e uma página HTML.
- A HTML deve conter apenas um formulário através do qual o visitante informará seu peso e sua altura.
- O servlet deve calcular o IMC do visitante e exibir o resultado, que deve conter: o peso e altura informados, o IMC calculado e a categoria na qual se enquadra.
- Fórmula: IMC = Peso / Altura2
- Classificação do IMC em categorias:
- Menor que 18,5: Abaixo do peso
- De 18,5 a 24,9: Peso normal
- De 25,0 a 29,9: Sobrepeso
- De 30,0 a 34,9: Obesidade Grau I
- De 35,0 a 39,9: Obesidade Grau II
- 40,0 ou superior: Obesidade Grau III

Exercício


Exercício 3.5 - Mozilla Firefox
Arquivo Editar Exibir Ir Favoritos Ferramentas
http://localhost:8084/Exercicio0305/
Teste de IMC
Peso: 112
Altura: 1.73
Calcular


Exercício 3.5 - Mozilla Firefox
Arquivo Editar Exibir Ir Favoritos Ferramentas
http://localhost:8084/Exercicio0305/meuservlet
Resultado
Peso: 112
Altura: 1.73
IMC: 37.421898493100336
Categoria: Obesidade Grau II

Redirecionamentos

- Ao receber uma requisição, um servlet pode:
  - Respondê-la diretamente, produzindo algum conteúdo html a ser enviado ao cliente (navegador)
  - Encaminhá-la para algum outro recurso responsável por responder a esse tipo de requisição
- Há, ao menos, 2 formas de encaminhamento:
  - Chamada ao método *HttpServletResponse.sendRedirect(String url)*
  - Criação de um objeto do tipo RequestDispatcher e chamada ao método *forward(HttpServletRequest, HttpServletResponse)*

Redirecionamentos

- A chamada do método *sendRedirect(String url)* faz com que o servidor envie ao cliente uma mensagem para que este envie a solicitação a uma nova url
- Na verdade, o servidor envia ao cliente uma mensagem HTTP com código 302, o qual indica que o recurso foi movido para outra url
- O redirecionamento através da classe RequestDispatcher é interno ao servidor; ou seja, não há comunicação com o cliente

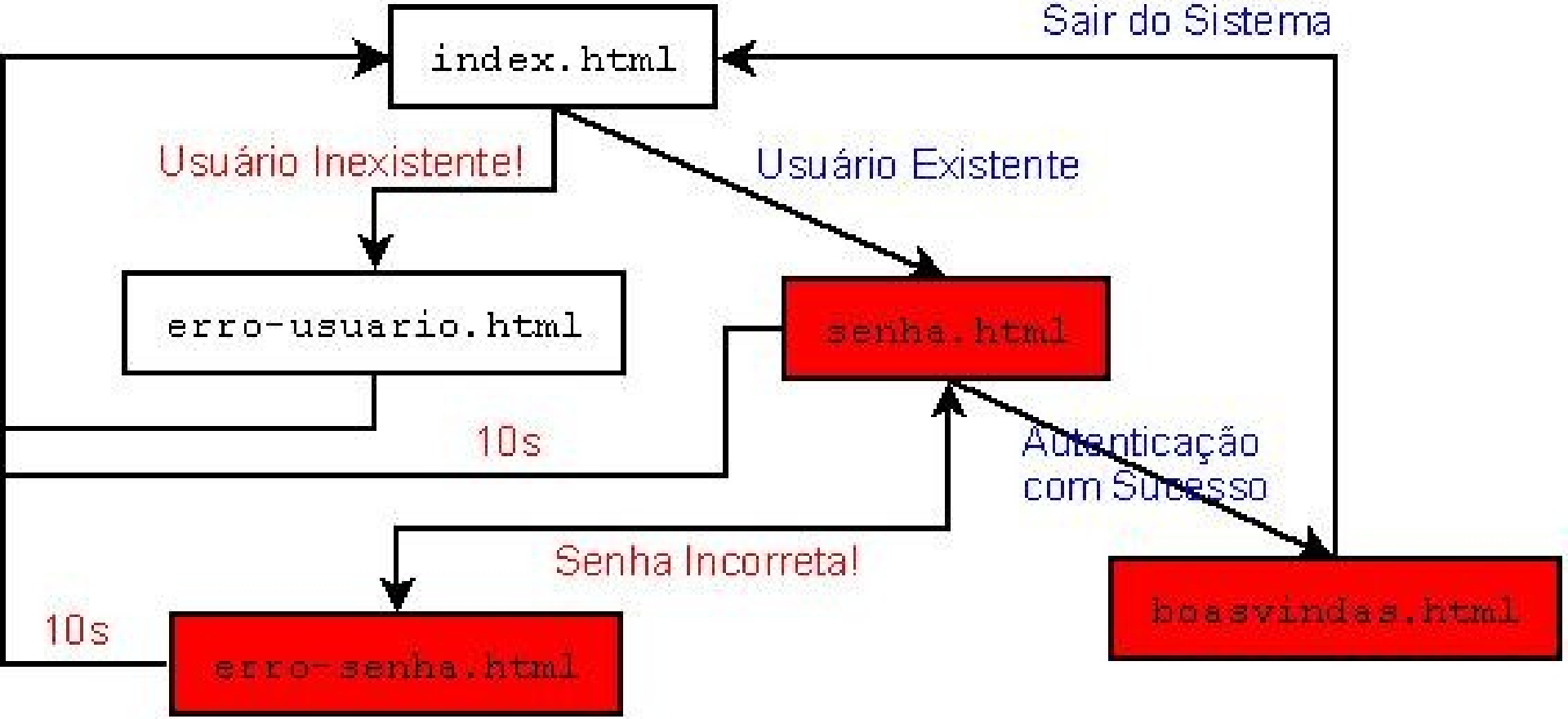
Redirecionamentos

- Em termos de práticos, para o cliente (navegador), a primeira opção modifica o endereço url no navegador, enquanto que a segunda não
- Em temos de projeto de aplicações web, redirecionamentos são utilizados entre Servlets/JSPs como um mecanismo de delegação de tarefas entre recursos

Exercício

- Crie 2 formulários para fazer autenticação de um usuário
- O primeiro deve obter o nome do usuário no sistema
- Caso seja um usuário cadastrado, a solicitação deve ser direcionada para um outro formulário que pedirá a senha
- Este segundo formulário deve ser enviado de forma segura, assim como a senha
- A senha deve ser fornecida por, no máximo, 10 seg; caso seja fornecida após, o sistema retorna para a página inicial
- As informações de usuário/senha podem estar no próprio servlet ou em algum banco de dados
- Após a autenticação correta, deve ser enviada ao cliente uma página de boas-vindas
- Após falha na autenticação, o usuário deve ser direcionado para uma página de erro com um link para nova tentativa

Exercício


index.html
Sair do Sistema
Usuário Inexistente!
Usuário Existente
erro-usuario.html
senha.html
10s
Autenticação
com Sucesso
Senha Incorreta!
10s
erro-senha.html
boasvindas.html

Escopo

- O escopo de um objeto indica quanto tempo o objeto existe depois de ter sido criado;
- Os níveis de escopo variam desde o tempo de vida do contêiner até o tempo de vida de uma página individual;
- Em aplicações Web, podemos definir 4 níveis de escopos diferentes: **Aplicação**, **Sessão**, **Solicitação/Requisição** e **Página**

Escopo

Escopo – Aplicação

- Objetos compartilhados por todos os servlets em uma dada aplicação
- Exemplo: conexão com banco de dados, lista de produtos numa aplicação de controle de estoque, etc.

Obtendo Acesso aos Tipos de Escopos

- Aplicação
  - A classe HTTPServlet possui um método *getServletContext()* que retorna um objeto do tipo *ServletContext*
  - Este objeto nos permite, por exemplo, criar atributos que existirão enquanto o servidor estiver no ar
  - Outra alternativa é através do método *init()* de HTTPServlet, o qual possui um parâmetro que é uma instância da classe *ServletConfig*; esta classe possui um método denominado *getServletContext()* que retorna a mesma referência citada no sub-item anterior

Obtendo Acesso aos Tipos de Escopos

- Aplicação
  - Exemplo

```
ServletContext context = getServletContext();
Integer num = (Integer) context.getAttribute("NumLogados");
context.setAttribute("NumLogados", ++num);
```

Escopo – Sessão

- Objetos compartilhados numa sessão vinculada a um usuário
- Exemplo: carrinho num site de compras

Obtendo Acesso aos Tipos de Escopos

- Sessão
  - A classe HTTPServletRequest (primeiro parâmetro dos métodos doGet()/doPost()) possui o método *getSession(),* que retorna uma referência para a sessão corrente
  - Caso não exista sessão corrente ativa, uma nova é criada

Obtendo Acesso aos Tipos de Escopos

- Sessão
  - Exemplo

```
HttpSession sessao = request.getSession();
sessao.setAttribute("NomeUser", "Daves");
sessao.getAttribute("NomeUser");
```

Escopo – Página

- Objetos compartilhados entre JSPs e servlets na página de execução atual
- Exemplo: variáveis locais declaradas em páginas JSPs

Escopo – Requisição

- Objetos compartilhados disponíveis para JSPs/Servlets numa solicitação
- Difere do escopo de página por permitir o compartilhamento também para JSPs/Servlets incluídas ou redirecionadas
- Exemplo: parâmetros de campo de formulário
- Pode ser acessado através da classe HTTPServletRequest

Listeners e Filters

- Permitem que tenhamos maior controle do uso das nossas aplicações web
- Listeners serão usualmente utilizados para observar o ciclo de vida de atributos (criação, atualização e remoção) para os escopos disponíveis
- Filters, por sua vez, permitem que as requisições possam ser interceptadas antes ou depois de alcançar o recurso desejado (um servlet, por exemplo)
- Ambos funcionam como funções de *callback*, ou seja, após a criação, o contêiner é quem faz chamadas a estes métodos

Listeners de Aplicações Web

- Usando a API de servlets (javax.servlet), podemos criar listeners de eventos para aplicações web
- Estes listeners estão definidos como interfaces nesta API
- Estão disponíveis para os escopos:
  - Escopo de aplicação: ServletContextListener, ServletContextAttributeListener
  - Escopo de sessão: HTTPSessionListener, HTTPSessionAttributeListener
  - Escopo de requisição: ServletRequestListener, ServletRequestAttributeListener

Listener de Aplicação

```
public class EscutadorAplicacao implements ServletContextListener
   {
   public void contextDestroyed(ServletContextEvent arg0) {}
   public void contextInitialized(ServletContextEvent arg0) {}
}


public class EscutadorAtributoAplicacao implements
   ServletContextAttributeListener {
   public void attributeAdded(ServletContextAttributeEvent arg0)
   { }
   public void attributeRemoved(ServletContextAttributeEvent
   arg0) {}
   public void attributeReplaced(ServletContextAttributeEvent
   arg0) {}
}
```

Listener de Aplicação

```
public class EscutadorAplicacao implements ServletContextListener,
   ServletContextAttributeListener {
   public void contextDestroyed(ServletContextEvent arg0) {
   System.out.println("Aplicação encerrada");
   }
   public void contextInitialized(ServletContextEvent arg0) {
   System.out.println("Aplicação inicializada");
   Calendar cal = new GregorianCalendar();
   arg0.getServletContext().setAttribute("iniciou", cal);
   }
   public void attributeAdded(ServletContextAttributeEvent arg0) {
   System.out.println(arg0.getName() + ": ");
   System.out.println(arg0.getValue().toString());
   }
   public void attributeRemoved(ServletContextAttributeEvent arg0) {
   }
   public void attributeReplaced(ServletContextAttributeEvent arg0)
   { }
}
```

Listener de Aplicação
web.xml

```
<web-app 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
  http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
    version="2.4">
...

  <listener>
  <listener-class>
  br.com.cursojava.EscutadorAplicacao
  </listener-class>
  </listener>
...

</web-app>
```

Listener de Aplicação

- A execução da aplicação fornecida no Tomcat fará com que os comandos System.out.println() sejam enviados para um arquivo de log: <Tomcat>/logs
- Testes análogos podem ser feitos para os listeners de sessão e requisição

Exercício

- Criar 2 tipos de contadores:
  - Um contador de acessos, o qual contará a quantidade de acessos à servlets
  - Um outro que contará a quantidade de sessões abertas para a aplicação num dado momento
- Crie um servlet que exiba estes valores

Filtros de Servlets

- Filtros permitem a interceptação de uma solicitação antes/após que esta atinja o recurso solicitado
- Em outras palavras, um filtro dá acesso aos objetos HTTPServletRequest e HTTPServletResponse antes destes serem passados a um servlet
- Um filtro pode ser um ponto ideal para: log de requisições, criptografia, autenticação, compressão de dados, validação do usuário, etc

Filtros de Servlets

- Filtros podem ser organizados em cadeias, de forma que mais de 1 filtro possa ser acionado para uma mesma solicitação
- Para criação de um filtro, basta criar uma classe que implemente a interface javax.servlet.Filter

Filtros de Servlets

```
public class FiltraRequisicao implements Filter {
   @Override
   public void init(FilterConfig arg0)
   throws ServletException {
   }

   @Override
   public void destroy() {
   }

   @Override
   public void doFilter(ServletRequest arg0,
   ServletResponse arg1, FilterChain arg2)
   throws IOException, ServletException {
   }
}
```

Filtros de Servlets

```
public class FiltraRequisicao implements Filter {
   @Override
   public void init(FilterConfig arg0) throws ServletException {
   }
   @Override
   public void destroy() {
   }
   @Override
   public void doFilter(ServletRequest arg0, ServletResponse arg1,
FilterChain arg2)
   throws IOException, ServletException {
   Calendar cal = Calendar.getInstance();
   System.out.println("Endereço: " + arg0.getRemoteAddr());
   System.out.println("Recurso: " +
   ((HttpServletRequest)arg0).getRequestURL().toString());
   System.out.println("Hora: " + cal.getTime().toString());
   arg2.doFilter(arg0, arg1);
   }
}
```

Filtros de Servlets

```
<web-app>
   <servlet>
          <servlet-name>primeiro</servlet-name>
          <servlet-class>br.com.siriusnet.cursojava.PrimeiroServlet
          </servlet-class>
   </servlet>
   <servlet-mapping>
          <servlet-name>primeiro</servlet-name>
          <url-pattern>/alou</url-pattern>
   </servlet-mapping>
   <filter>
          <filter-name>filtro</filter-name>
          <filter-class>br.com.cursojava.FiltraRequisicao
          </filter-class>
   </filter>
   <filter-mapping> // Filtro para um servlet específico !!
          <filter-name>filtro</filter-name>
          <url-pattern>/alou</url-pattern> <!--OU !!!
          <servlet-name>primeiro</servlet-name -->
   </filter-mapping>
</web-app>
```

Filtros de Servlets

```
<web-app>
   <filter>
          <filter-name>filtro</filter-name>
          <filter-class>
                 br.com.siriusnet.cursojava.FiltraRequisicao
          </filter-class>
   </filter>
   <filter-mapping>
          <filter-name>filtro</filter-name>
          <url-pattern>/*</url-pattern> // Qualquer recurso!
   </filter-mapping>
</web-app>
```

Filtros de Servlets

- A chamada ao método *doFilter()* dentro do próprio método encaminha a solicitação para outros filtros na cadeia, ou para o recurso solicitado
- Quando desejamos tratar a resposta de uma solicitação, esta chamada deve ser localizada antes do tratamento desejado
- Pode ser utilizado, por exemplo, para calcular o tempo necessário para o processamento de uma requisição pelo servidor

Filtros de Servlets

- Exemplo

```
public class TimerFilter implements Filter {
    ....
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
                throws IOException, ServletException {

                long inicio = System.currentTimeMillis();
                chain.doFilter(request, response);
                long fim = System.currentTimeMillis();

                String nome = "";
                if (request instanceof HttpServletRequest) {
                        nome = ((HttpServletRequest)request).getRequestURI();
                }
                context.log(nome + ": " + (fim - inicio) + "ms");
    }
}
```