

Instituto de

computação



Desenvolvimento Web JSP

Prof. Leonardo Cruz.

leonardocruz@id.uff.br

Departamento de Computação, UFF

Contexto

- Uma página de login
 - Verifica se usuário e senha estão corretos
 - Caso afirmativo página de sucesso
 - Caso negativo página de erro
-
- Versão 0 – veja **Ex9**

Exemplo de Login (versão 0)

<form action="**VerificarLogin**" method="post">

Nome

Senha

Login

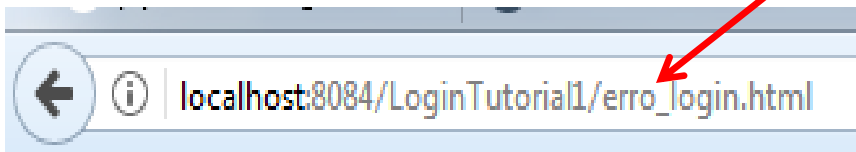


```
@WebServlet(name = "VerificarLogin", urlPatterns = {"/VerificarLogin"})
public class VerificarLogin extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        String nome_user = request.getParameter("nome");
        String senha_user = request.getParameter("senha");

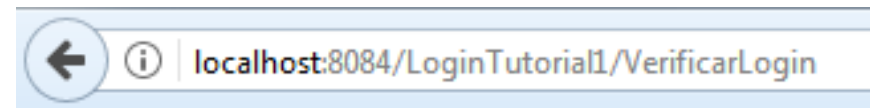
        if (nome_user.equals("admin") && senha_user.equals("123")) {
            request.setAttribute("nome", nome_user);
            RequestDispatcher resposta = request.getRequestDispatcher("sucesso.jsp");
            resposta.forward(request, response);
        } else {
            response.sendRedirect("erro_login.html");
        }
    }
}
```

```
@WebServlet(name = "VerificarLogin", urlPatterns = {"/VerificarLogin"})
public class VerificarLogin extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        String nome_user = request.getParameter("nome");
        String senha_user = request.getParameter("senha");

        if (nome_user.equals("admin") && senha_user.equals("123")) {
            request.setAttribute("nome", nome_user);
            RequestDispatcher resposta = request.getRequestDispatcher("sucesso.jsp");
            resposta.forward(request, response);
        } else {
            response.sendRedirect("erro_login.html");
        }
    }
}
```



Erro de Login

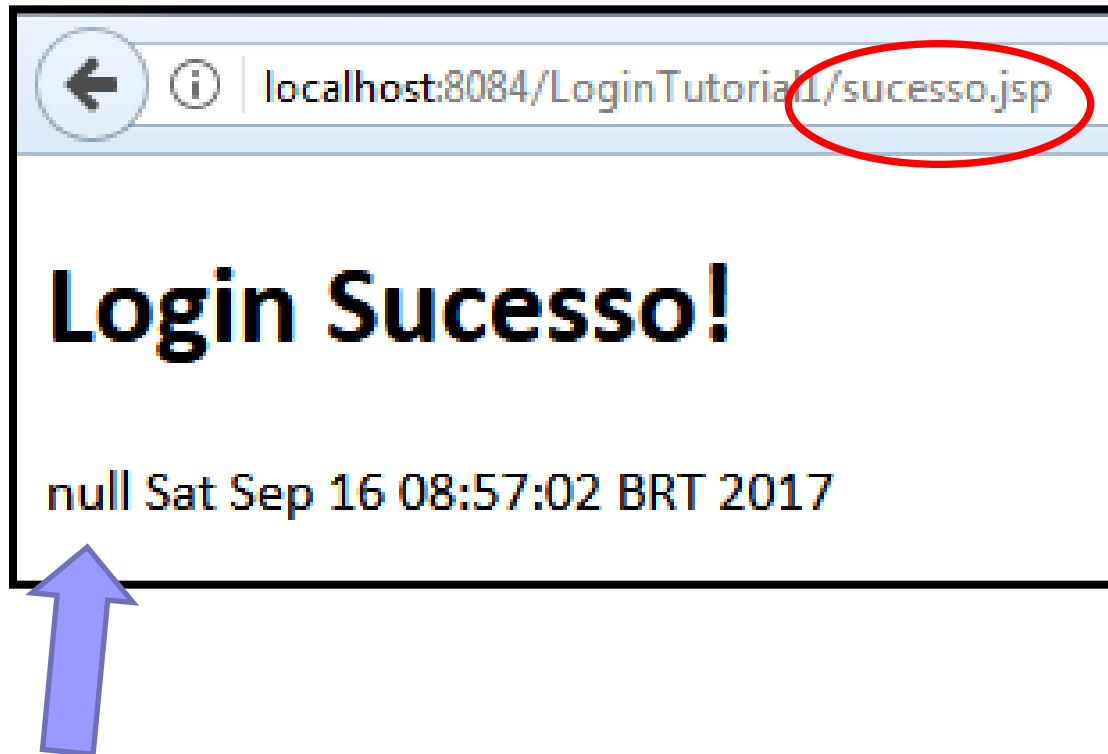


Login Sucesso!

admin Sat Sep 16 08:49:16 BRT 2017

Problema...

- É possível acessar a página sem estar logado



Armazenamento de Dados

- Guardar informação para garantir alguma segurança
- Prevenir acesso indevido

Armazenamento de Dados

- É comum precisar armazenar dados para processamento futuro
- Campo escondido
 - ☐ Dado presente na página de retorno do usuário
- Sessão
 - ☐ Entre diferentes interações do usuário
 - ☐ Mesma execução do browser
- Cookie
 - ☐ Entre diferentes sessões do usuário
 - ☐ Diferentes execuções do browser
- Aplicação
 - ☐ Entre diferentes usuários
 - ☐ Mesma execução do servidor de aplicação
- Banco de dados
 - ☐ Entre diferentes execuções do servidor de aplicação

Armazenamento de Dados

Campos escondidos

- Mecanismo alternativo de gerenciamento de sessão
 - Cada formulário contém campos *hidden* para transferir as informações de sessão em conjunto com seus controles:

```
<input type="hidden" name="total" value="15">
```
- Problemas:
 - O usuário pode alterar o conteúdo dos campos *hidden* alterando o código HTML das páginas
 - Uso não recomendado

Armazenamento de Dados

Sessão

- O objeto HttpSession é usado para gerenciamento de sessão. Uma sessão contém informações específicas para um determinado usuário em toda a aplicação.
- Quando um usuário entra em um site (ou um aplicativo on-line) pela primeira vez, HttpSession é obtida através de `request.getSession ()`, o usuário recebe uma ID exclusiva para identificar sua sessão.

```
@WebServlet(name = "VerificarLogin", urlPatterns = {"/VerificarLogin"})
public class VerificarLogin extends HttpServlet {

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        String nome_user = request.getParameter("nome");
        String senha_user = request.getParameter("senha");

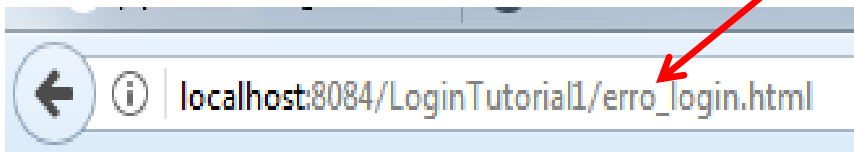
        if (nome_user.equals("admin") && senha_user.equals("123")) {
            HttpSession session = request.getSession();
            session.setAttribute("NomeUsuarioLogado", "admin");
            session.setAttribute("logado", "ok");

            RequestDispatcher resposta = request.getRequestDispatcher("sucesso.jsp");
            resposta.forward(request, response);
        } else {
            response.sendRedirect("erro_login.html");
        }
    }
}
```

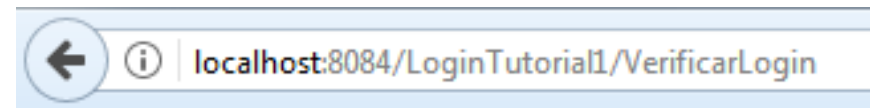
Veja **Ex10**



```
if (nome_user.equals("admin") && senha_user.equals("123")) {  
    HttpSession session = request.getSession();  
    session.setAttribute("NomeUsuarioLogado", "admin");  
    session.setAttribute("logado", "ok");  
  
    RequestDispatcher resposta = request.getRequestDispatcher("sucesso.jsp");  
    resposta.forward(request, response);  
} else {  
    response.sendRedirect("erro_login.html");  
}  
}
```



Erro de Login



Login Sucesso!

admin Sat Sep 16 08:49:16 BRT 2017

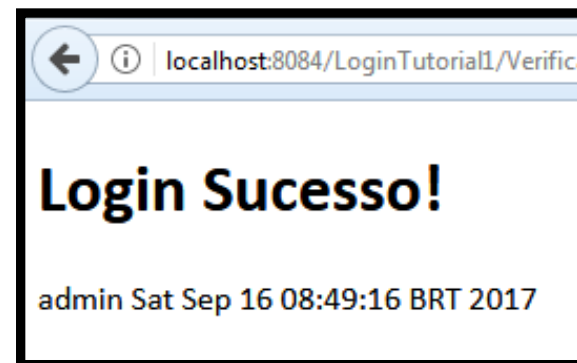
Veja Ex10

```

<%@page contentType="text/html" pageEncoding="UTF-8" import="java.ut
<!DOCTYPE html>
<html>
    <head>
        <title>Sucesso</title>
    </head>
    <body>
        <%
            Object logado = session.getAttribute("logado");
            if (logado != null) {
                String aux = (String) logado;
                if (aux.equals("ok")) { %>
                    <h1>Login Sucesso!</h1>
                    <%= session.getAttribute("NomeUsuarioLogado") %>
                    <%= new Date() %>
                } else { response.sendRedirect("erro_login.html"); }
            } else { response.sendRedirect("erro_login.html"); } %>
        </body>
    </html>

```

**Na página JSP verifico
a existência da sessão**



Veja Ex10

Armazenamento de Dados

Controle de Sessões

- Método **getSession** existente no objeto (recebido como parâmetro) da classe **HttpServletRequest**.
- A chamada ao método getSession deve ser **efetuada antes** de qualquer chamada ao método getWriter da classe HttpServletResponse

Armazenamento de Dados

Controle de Sessões

- O método getSession retorna um objeto da classe HttpSession, onde é possível
 - Ler todos os atributos armazenados com o método **getAttributeNames()**
 - Armazenar valores, através do método **setAttribute(nome, valor)**
 - Recuperar valores, através do método **getAttribute(nome)**

Armazenamento de Dados

Controle de Sessões

- O método **setMaxInactiveInterval** da classe HttpSession permite a configuração do tempo máximo de atividade de uma seção
- O método **invalidate** da classe HttpSession permite a finalização da seção

Armazenamento de Dados

Controle de Sessões **no JSP**

- Objeto session
- `session.setAttribute("hoje", d);`
- `session.getAttribute("hoje");`

Armazenamento de Dados

Cookies

- Cookies servem para armazenar por tempo determinado alguma informação no browser do cliente
- Usos mais comuns são para
 - ☐ Deixar o cliente acessar o sistema sem pedir senha
 - ☐ Memorizar quantas vezes aquele browser já acessou o site
 - ☐ Personalizar propagandas

Armazenamento de Dados

Cookies

- Os cookies existentes são acessados através do método **getCookies** existente no objeto da classe **HttpServletRequest**
- O método **getCookies** retorna um array de objetos da classe **Cookie**
 - Se nenhum cookie tiver sido adicionado até então, retorna null
- Para cada objeto da classe **Cookie**, é possível recuperar seu valor através do método **getValue()**

Armazenamento de Dados

Cookies

- Adição de Cookie no browser do usuário
 - Chamar o método **addCookie** da classe **HttpServletResponse**, passando como parâmetro o novo cookie (mesmo para cookies já existentes)
 - A chamada ao método addCookie deve ser **efetuada antes de qualquer** chamada ao método getWriter da classe HttpServletResponse

Armazenamento de Dados

Cookies

import javax.servlet.http.Cookie;

```
@WebServlet(name = "VerificarLogin", urlPatterns = {"/VerificarLogin"})
public class VerificarLogin extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        String nome_user = request.getParameter("nome");
        String senha_user = request.getParameter("senha");

        if (nome_user.equals("admin") && senha_user.equals("123")) {

            // Cria o objeto Cookie
            Cookie cookieNome = new Cookie("Nome", nome_user);

            //Adiciona os Cookies no reponse
            response.addCookie(cookieNome);

            RequestDispatcher resposta = request.getRequestDispatcher("sucesso.jsp");
            resposta.forward(request, response);
        } else {
            response.sendRedirect("erro_login.html");
        }
    }
}
```

Veja Fonte Ex11

Armazenamento de Dados

Cookies

- Na **página jsp** – pegando o cookie

```
<%  
    Cookie[] cookies = request.getCookies();  
    for(int i = 0; i < cookies.length; i++){  
        if (cookies[i].getName().equals("Nome")){  
            out.println("<b>Nome:</b> " + cookies[i].getValue());  
            out.println("<br>");  
        }  
    }  
%>
```

Veja Fonte Ex11

Armazenamento de Dados

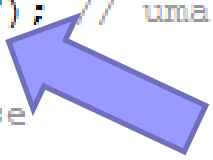
Cookies Persistentes

- ❑ O método **setMaxAge** determina por quanto tempo, em segundos, o cookie é válido
- ❑ O cookie irá persistir após o encerramento da sessão somente se o setMaxAge tiver sido usado

```
if (nome_user.equals("admin") && senha_user.equals("123")) {
    // Cria o objeto Cookie
    Cookie cookieNome = new Cookie("Nome", nome_user);
    cookieNome.setMaxAge(60*60*24*7); // uma semana

    //Adiciona os Cookies no reponse
    response.addCookie(cookieNome);

    RequestDispatcher resposta = request.getRequestDispatcher("sucesso.jsp");
    resposta.forward(request, response);
} else {
    response.sendRedirect("erro_login.html");
}
```



Armazenamento de Dados

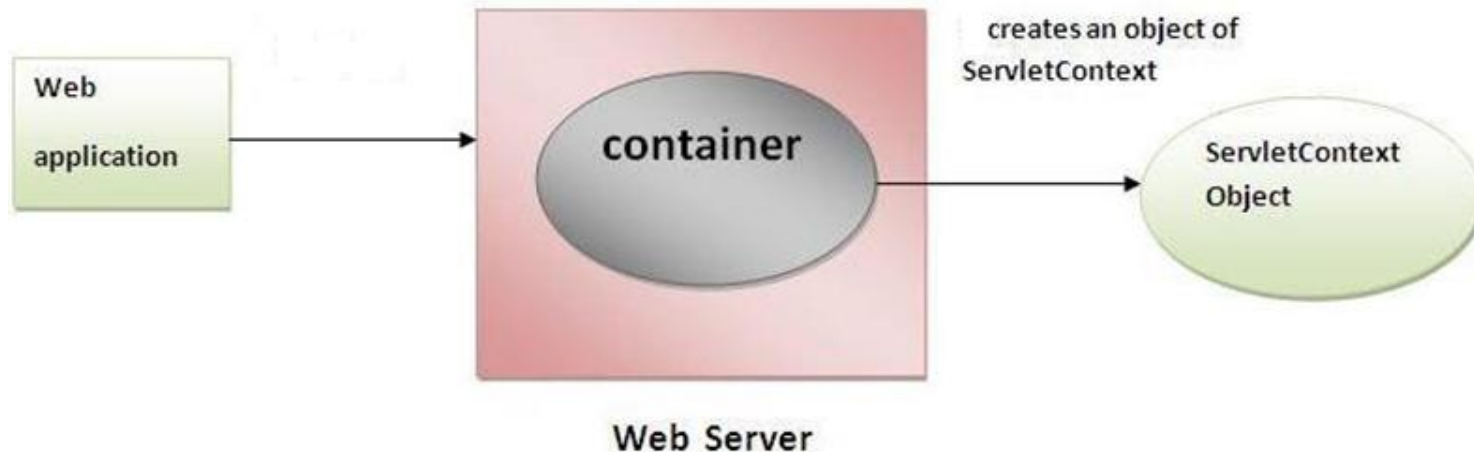
Aplicação

- Permite armazenar informações de forma que qualquer thread de qualquer servlet possa acessar
- Objeto do tipo **ServletContext** Pode ser obtido de **getServletContext()**
 - Representa o container
 - Único para todos os Servlets da aplicação

Armazenamento de Dados

Aplicação

- Este objeto pode ser usado para obter informações de configuração do arquivo web.xml. Existe apenas um objeto ServletContext por aplicativo web.



Armazenamento de Dados

Aplicação

- Permite armazenar informações de forma que qualquer thread de qualquer servlet possa acessar
- Objeto do tipo **ServletContext** Pode ser obtido de **getServletContext()**
 - Único para todos os Servlets da aplicação

Armazenamento de Dados

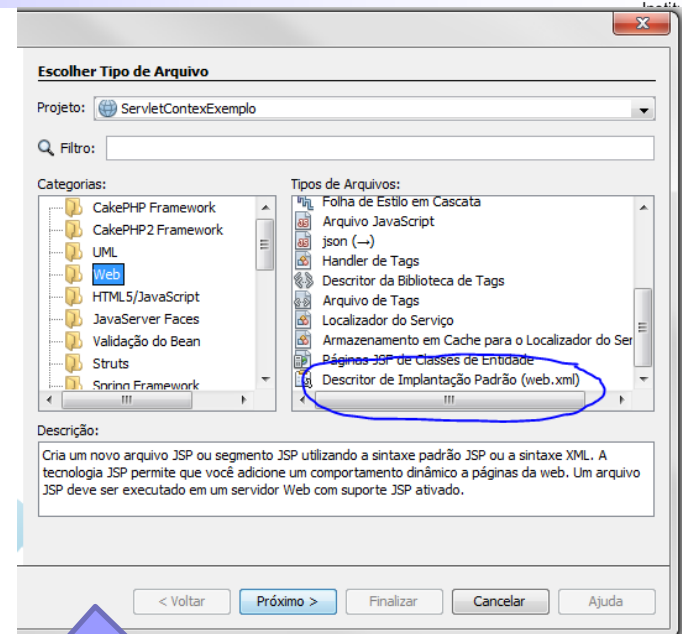
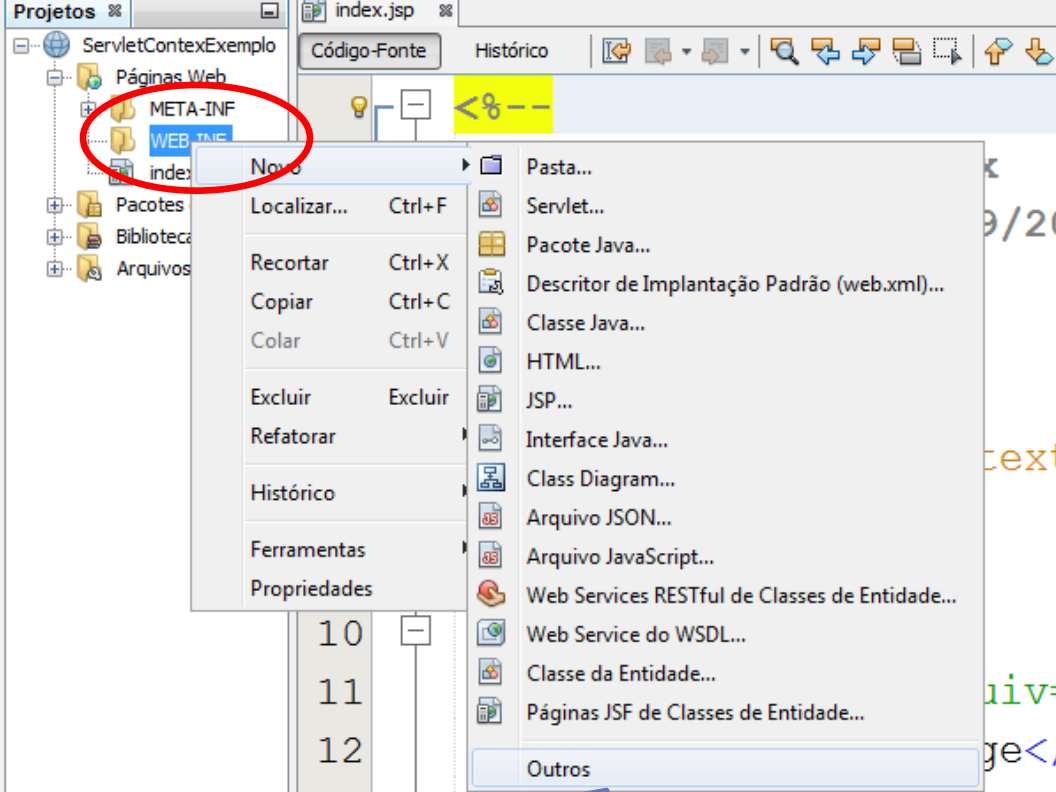
Aplicação

- Guarda um Map de “atributos” onde podem ser escritos/lidos dados temporários dos Servlets
 - Ler todos os atributos armazenados com o método **getAttributeNames()**
 - Armazenar valores, através do método **setAttribute(nome, valor)**
 - Recuperar valores, através do método **getAttribute(nome)**

Armazenamento de Dados

Aplicação

- Não há como incluir os parâmetros com uma anotação (não específico de um servlet e sim de todos os servlets)
- É por isso que precisamos adicionar o parâmetro de contexto do servlet no arquivo web.xml



<?xml version="1.0" encoding="UTF-8"?>

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.
  version="3.0">
  <session-config>
    <session-timeout>
      30
    </session-timeout>
  </session-config>
</web-app>
```

Criação do web.xml

Veja Fonte: Ex12

Armazenamento de Dados

Aplicação

Definição de dois parâmetros globais que poderão ser usados por todos os servlets (email e cel) => **<context-param>**

<context-param>

<param-name>email**</param-name>**

<param-value>webmaster@domain.com**</param-value>**

</context-param>

<context-param>

<param-name>cel**</param-name>**

<param-value>(21)98686755**</param-value>**

</context-param>

Inserir a definição no web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.
    version="3.0">
  <session-config>
    <session-timeout>30</session-timeout>
  </session-config>
  <context-param>
    <param-name>email</param-name>
    <param-value>webmaster@domain.com</param-value>
  </context-param>
  <context-param>
    <param-name>cel</param-name>
    <param-value>(21) 98686755</param-value>
  </context-param>
</web-app>
```

```
@WebServlet(name = "Servlet1", urlPatterns = {"/Servlet1"})
```

```
public class Servlet1 extends HttpServlet {  
    private String email, cel;
```

```
    public void init(ServletConfig config) throws ServletException {  
        ServletContext aux = config.getServletContext();  
        email = aux.getInitParameter("email");  
        cel = aux.getInitParameter("cel");  
    }
```

```
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
        response.setContentType("text/html;charset=UTF-8");  
        PrintWriter out = response.getWriter();  
        try {  
            out.write("<html><body>");  
            out.write("<h2>Servlet Context parametros obtidos pelo servlet1</h2>");  
            out.write("<p><strong>E-mail: </strong>" + email + "</p>");  
            out.write("<p><strong>Phone: </strong>" + cel + "</p>");  
            out.write("</body></html>");  
        } finally {  
            out.close();  
        }  
    }
```

Servlet1

```
@WebServlet(name = "Servlet2", urlPatterns = {"/Servlet2"})
```

```
public class Servlet2 extends HttpServlet {  
    private String email, cel;
```

```
    public void init(ServletConfig config) throws ServletException {  
        ServletContext aux = config.getServletContext();  
        email = aux.getInitParameter("email");  
        cel = aux.getInitParameter("cel");  
    }
```

```
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
        response.setContentType("text/html;charset=UTF-8");  
        PrintWriter out = response.getWriter();  
        try {  
            out.write("<html><body>");  
            out.write("<h2>Servlet Context parametros obtidos pelo servlet2</h2>");  
            out.write("<p><strong>E-mail: </strong>" + email + "</p>");  
            out.write("<p><strong>Phone: </strong>" + cel + "</p>");  
            out.write("</body></html>");  
        } finally {  
            out.close();  
        }  
    }
```

Servlet2

Usando web.xml

- Aproveitando... Web.xml
- Outras definições... Por exemplo: uma página de erro 404 personalizada

Usando web.xml

- Criar uma página de erro da aplicação personalizada

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/
  xsi:schemaLocation="http://java.sun.com/xml/n
  <session-config>
    <session-timeout>30</session-timeout>
  </session-config>
  <context-param>
    <param-name>email</param-name>
    <param-value>webmaster@domain.com</param-value>
  </context-param>
  <context-param>
    <param-name>cel</param-name>
    <param-value>(21) 98686755</param-value>
  </context-param>
  <error-page>
    <error-code>404</error-code>
    <location>/erro.jsp</location>
  </error-page>
</web-app>
```

**Mapeamento do
Erro no Web.xml**

Veja Fonte: Ex12

Usando web.xml

- Uso do atributo isErrorPage na página jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8" isErrorPage="true"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Erro da Aplicação</title>
  </head>
  <body>
    <h1>Erro 404 </h1>
    Tentativa de acessar uma página inexistente
  </body>
</html>
```

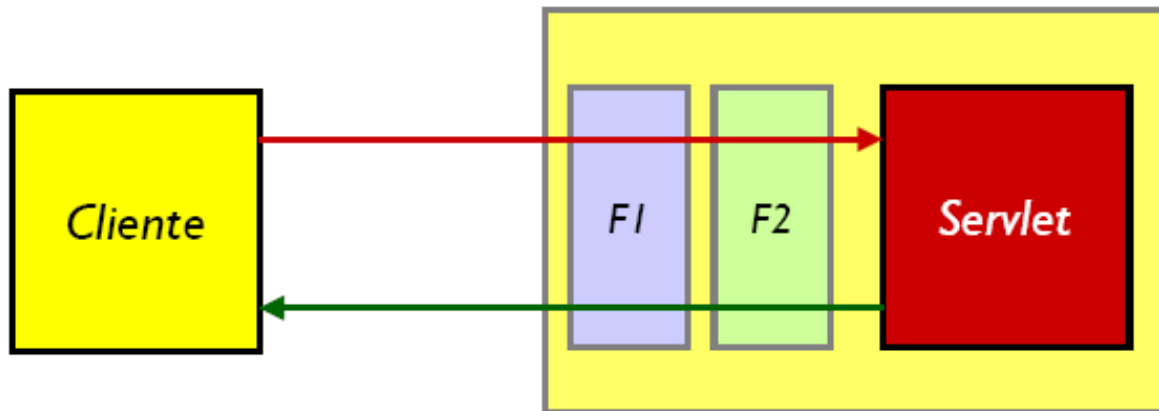
define que a página JSP servirá como a página de erro padrão para um grupo de páginas JSP.



Filtros

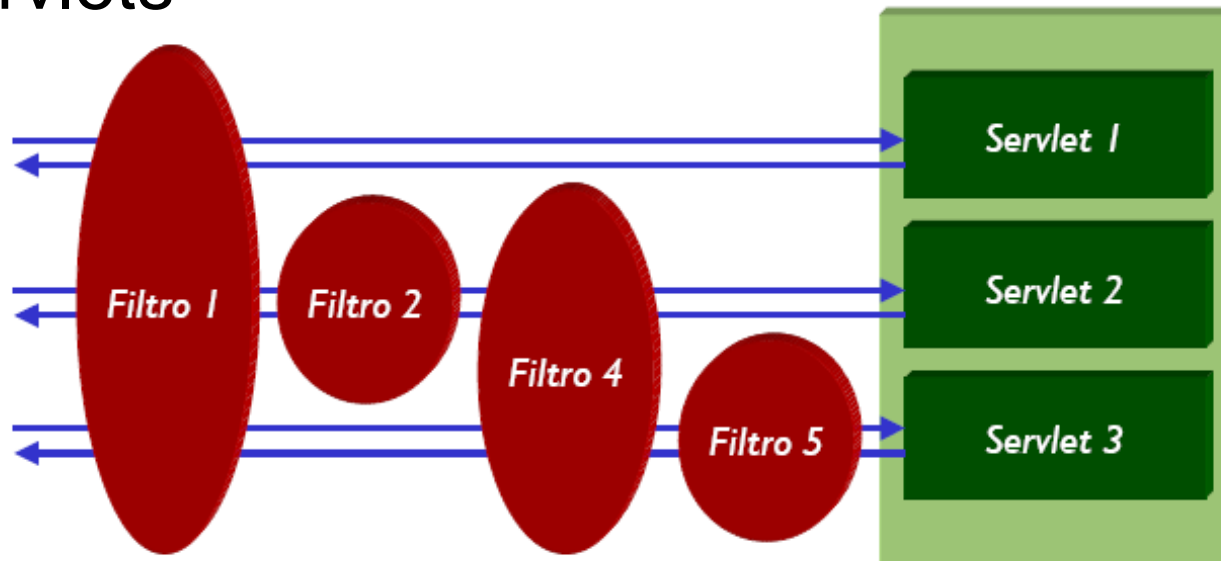
■ Uso de filtros....

- Filtros são objetos que permitem que executemos código **antes** da requisição e também **depois** que a resposta foi gerada.
- Filtros podem ser concatenados em uma corrente
 - Neste cenário, as requisições são interceptadas em uma ordem e as respostas em ordem inversa



Filtros

- Um **filtro** pode realizar diversas transformações, tanto na **resposta** como na **requisição** antes de passar esses objetos adiante (se o fizer)
 - Filtros podem ser **reutilizados** em vários servlets



Para que servem ?

- Filtros permitem:
 - **Tomada de decisões:** podem decidir se repassam uma requisição adiante, se redirecionam ou se enviam uma resposta interrompendo o caminho normal da requisição
 - **Tratamento de requisições e respostas:** podem empacotar uma requisição (ou resposta) em outra, alterando os dados e o conteúdo dos cabeçalhos
- Aplicações típicas
 - Autenticação
 - Conversão de imagens, compressão e descompressão
 - Criptografia

Filtros

■ Vantagens do filtro

- O filtro é plugável.
- Um filtro não tem dependência de outro recurso.
- Menos manutenção.

■ É possível combinação de filtros...

- O objeto de FilterChain é responsável por invocar o próximo filtro ou recurso na cadeia. Este objeto é passado no método doFilter de interface de filtro.

Como funcionam?

- Quando o **container** recebe uma requisição, ele verifica se há um filtro associado ao recurso solicitado. Se houver, a requisição é roteada ao filtro
- O filtro, então, pode:
 1. **Gerar sua própria resposta** para o cliente
 2. **Repassar a requisição**, modificada ou não, **ao próximo filtro da corrente**, se houver, ou ao recurso final, se ele for o último filtro
 - **Rotear a requisição** para outro recurso
- Na volta para o cliente, a resposta passa pelo mesmo conjunto de filtros em ordem inversa

Filtros

■ Métodos

- **init()** O método é invocado apenas uma vez. Ele é usado para inicializar o filtro.
- **doFilter()** O método é invocado sempre que o usuário solicita qualquer recurso, ao qual o filtro é mapeado. Ele é usado para executar tarefas de filtragem.
- **destroy()** Isso é invocado apenas uma vez quando o filtro é retirado do serviço.

Filtros

■ Exemplo

Index.jsp

Exemplo de Login (com filtro)

Nome

Senha

Login

VerificarLogin

Vai verificar o login

Sucesso.jsp

Apresenta link

ServLet1

mensagem

Cria a sessão

Veja fonte Ex13

```
public class Verifica_logado implements Filter {
    Verifica a sessão
```

Filtros

```
@WebFilter(filterName = "Verifica_logado", servletNames = {"Servlet1"}, urlPatterns={"/sucesso.jsp"})
public class Verifica_logado implements Filter {

    public void doFilter(ServletRequest request, ServletResponse response,
        FilterChain chain)
        throws IOException, ServletException {

        HttpSession sessao = ((HttpServletRequest) request).getSession(true);
        Object logado = sessao.getAttribute("logado");
        if (logado != null) {
            String aux = (String) logado;
            if (aux.equals("ok")) {
                chain.doFilter(request, response);
            } else {
                RequestDispatcher rd = request.getRequestDispatcher("index.jsp");
                rd.forward(request, response);
            }
        } else {

            RequestDispatcher rd = request.getRequestDispatcher("index.jsp");
            rd.forward(request, response);

        }
    }
}
```

Armazenamento de Dados

Banco de Dados

- Java Database Connectivity (JDBC)