

Servlet e JSP



Desenvolvimento Web

Introdução ao Servlet

Conceitos iniciais

ശാസ്ത്രങ്ങളുടെയും സാങ്കേതികവിദ്യകളുടെയും

Comunicação cliente-servidor

Servlet : lado “servidor”

HTML, CSS, JavaScript, etc: lado “cliente”

Servlet utilizado para:

- Tratar a lógica de negócio
- Atuar como controlador (modelo MVC)

Métodos de HttpServlet

```
void init (ServletConfig config)
```

```
ServletConfig getServletConfig()
```

```
void service (ServletRequest request, ServletResponse  
response)
```

```
String getServletInfo()
```

```
void destroy()
```

Outros Métodos

```
protected void doGet(HttpServletRequest request,  
                        HttpServletResponse response)
```

```
protected void doPost(HttpServletRequest request,  
                       HttpServletResponse response)
```

Acionados pelo método "service"

HttpServletRequest

O servidor web cria o objeto HttpServletRequest. HttpServletRequest contém informações sobre a solicitação cliente.

MÉTODOS ASSOCIADOS

String getParameter (String name)

Enumeration getParameterNames()

String[] getParameterValues (String name)

Cookie[] getCookies ()

HttpSession getSession (boolean create)

HttpServletResponse

O servidor web cria o objeto HttpServletResponse
HttpServletResponse contém (conterá)
informações para o cliente.

MÉTODOS ASSOCIADOS

void addCookie (Cookie cookie)

ServletOutputStream getOutputStream()

PrintWriter getWriter ()

void setContentType (String type)

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

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

```
StringBuffer temporario = new StringBuffer();
```

```
temporario.append("<h2>Listagem</h2>");  
temporario.append("<h3>Numeros</h3>");  
temporario.append("<ul>");
```

```
for(int i=0; i < 10; i++){  
temporario.append("<li>"+i);  
}
```

```
temporario.append("</ul>");  
temporario.append("<br><br>");  
temporario.append("<a href='Form1.html'>voltar</a>");
```

```
retorno.println(temporario.toString());  
retorno.close();
```

```
}
```

Exemplo GET

Exemplo POST

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

```
String nome = request.getParameter("nome");
```

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

```
pw.println("<b>O nome informador eh:</b> " + nome);
```

```
pw.println("<br><br>");
```

```
pw.println("<a href='Form1.html'>voltar</a>");
```

```
pw.close();
```

```
}
```

Meu Primeiro Servlet

Usando Formulários

Um Exemplo de Cadastro

Insert title here - Mozilla Firefox

Inbox (7,549) - julion... x SD A commitment-base... x Julio Cesar Na

← → http://localhost:8080/Test_Servlet/Form2.html

Cadastro de Cliente

Nome:

CPF:

Endereco:

Sexo:
 Masculino Feminino

Tipo:
Normal ▾

Receber comunicados:

Obs.:

OK Reset

Enviando Dados(CLIENT SIDE)

```
<body>
```

```
<h2>Cadastro de Cliente</h2>
```

```
<form name="cadastrocliente" method="POST" action="CadastrarCliente">
```

```
Nome:<br>
```

```
<input type="text" name="nome" value=""><br>
```

```
CPF:<br>
```

```
<input type="text" name="cpf" value=""><br>
```

```
Endereco:<br>
```

```
<input type="text" name="endereco" value=""><br>
```

```
Sexo:<br>
```

```
<input type="radio" name="sexo" value="masc">Masculino<input type="radio" name="sexo" value="fem">Feminino<br>
```

```
Tipo:<br>
```

```
<select name="tipo">
```

```
<option value="Normal">Normal</option>
```

```
<option value="VIP">VIP</option>
```

```
</select><br>
```

```
Receber comunicados:<br>
```

```
<input type="checkbox" name="comunicados" value="sim"><br>
```

```
Obs.:<br>
```

```
<textarea name="obs" rows="4" cols="20"></textarea><br>
```

```
<input type="submit" value="OK">
```

```
<input type="reset" value="Reset">
```

```
</form>
```

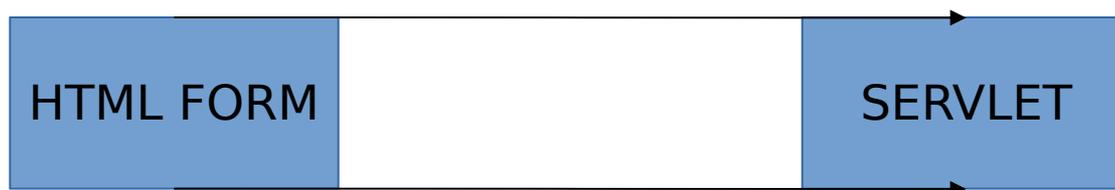
```
</body>
```

Recebendo Dados (SERVER SIDE)

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

```
    String nome = request.getParameter("nome");  
    String cpf = request.getParameter("cpf");  
    String end = request.getParameter("endereco");  
    String sexo = request.getParameter("sexo");  
    String tipo = request.getParameter("tipo");  
    String comunicados = request.getParameter("comunicados");  
    String obs = request.getParameter("obs");  
  
    ...  
    // EXIBIÇÃO DOS DADOS RECEBIDOS EM UMA NOVA PÁGINA...  
}
```

Enviando datos...



Enviando Dados por Meio de Links

Insert title here - Mozilla Firefox

Inbox (7,549) - julion... x SD A commitment-base... x Julio

http://localhost:8080/Test_Servlet/Form3.html

Listagem de Clientes Cadastrados

João da Silva([Excluir](#))([Alterar](#))

Pedro Moreno([Excluir](#))([Alterar](#))

Rosa Souza([Excluir](#))([Alterar](#))

Sandra Alencar([Excluir](#))([Alterar](#))

Rosimere Costa([Excluir](#))([Alterar](#))

Alexandre Machado([Excluir](#))([Alterar](#))

Servlet Excluir

Servlet Alterar

Exercícios

Exercício 1

Crie o formulário apresentado abaixo e estabeleça o envio de dados desse formulário para um Servlet que recebe e exibe os dados recebidos...



The image shows a screenshot of a Mozilla Firefox browser window. The title bar reads "Insert title here - Mozilla Firefox". The address bar shows the URL "http://localhost:8080/Test_Servlet/Form2.html". The page content is a form titled "Cadastro de Cliente". The form fields are: "Nome:" (text input), "CPF:" (text input), "Endereco:" (text input), "Sexo:" (radio buttons for "Masculino" and "Feminino"), "Tipo:" (dropdown menu with "Normal" selected), "Receber comunicados:" (checkbox), and "Obs.:" (text area). At the bottom of the form are "OK" and "Reset" buttons. A vertical toolbar with various application icons is visible on the left side of the browser window.

Insert title here - Mozilla Firefox

Inbox (7,549) - julion... x SD A commitment-base... x Julio Cesar Na

http://localhost:8080/Test_Servlet/Form2.html

Cadastro de Cliente

Nome:

CPF:

Endereco:

Sexo: Masculino Feminino

Tipo:

Receber comunicados:

Obs.:

OK Reset

Exercício 2

Com base no exercício anterior...

Ajuste o servlet criado de modo que ele receba os dados do formulário, crie um objeto da classe Cliente, insira esse objeto em uma lista de clientes (essa lista pode ser implementada como um atributo estático da classe Cliente), e depois exibe todos os clientes já criados.

Usando a Classe Calendar

```
...
String dataEmTexto = request.getParameter('dataNascimento');
    Calendar dataNascimento = null;

    // fazendo a conversão da data
    try {
        Date date = new SimpleDateFormat('dd/MM/yyyy').parse(dataEmTexto);
        dataNascimento = Calendar.getInstance();
        dataNascimento.setTime(date);
    } catch (ParseException e) {
        out.println('Erro de conversão da data');
        return; //para a execução do método
    }
...

```

Exercício 3

Ajuste o exercício anterior para passar a utilizar a data de nascimento do Cliente.

Tratamento de Exceções via Container



Mensagens de exceções não fazem sentido para o usuário comum



Podemos tornar nossa aplicação mais confiável e mais amigável ao tratar exceções



Uma opção é fazer o tratamento de exceções no web.xml (nível mais geral / container)



É possível também tratar erros HTTP

Tratamento de Exceções via Container

Exemplo de tratamento de exceções

```
<error/ page|  
  <exception/ type|java.lang.Exception</exception/ type|  
  <location|/erro.html</location|  
</error/ page|
```

Exemplo de tratamento de erros HTTP. Ex.: Erro 404: página não encontrada.

```
<error/ page|  
  <error/ code|404</error/ code|  
  <location|/404.html</location|  
</error/ page|
```

Não deixe de tratar exceções ao longo de todo o seu código!

Exercício 4

Faça o tratamento de erros e de exceções via container (web.xml).

Init e Destroy



Init: utilizado pelo container para inicializar o servlet; Pode ser utilizado para:

- Inicializar parâmetros/variáveis e recursos comuns a todas as requisições



Destroy: utilizado na finalização do servlet. Utilizado para:

- Liberar parâmetros/variáveis e recursos utilizados nas requisições do servlet.

Obs.: O servlet é instanciado uma única vez. Todas as solicitações são tratadas por threads específicas.

Dica: Se sobrescrever os métodos `init()` e/ou `destroy()` não deixe de chamar os correspondentes da superclasse (ex.: `super.init()` e `super.destroy()`).

Servlet: Instância Única



Variáveis definidas no servlet são acessíveis a todas as threads



Se isso for necessário, uma solução é sincronizar o método service

- No entanto, isso trará grandes problemas de desempenho

Assim: Evite utilizar variáveis compartilhadas entre as várias threads.

6.3. ServletContext e ContextListener

6.4 Welcome pages

6.5. Parâmetros de inicialização

6.6. Utilizando atributos do ServletContext

Introdução ao JSP

Conceitos iniciais

Script-like language

Código Java escrito entre “<% ... %>”

```
<html>
```

```
<%@ page import="java.util.*, br.com.agenda.dao.*" %>
```

```
<%
```

```
    out.println(nome);
```

```
    for(int i = 0; i < 10 ; i++)
```

```
        out.println("teste");
```

```
%>
```

```
</html>
```

Enviando dados

```
<body>
```

```
<h2>Cadastro de Cliente</h2>
```

```
<form name="cadastrocliente" method="post"  
action="cadastrarCliente.jsp">
```

```
Nome:<br>
```

```
<input type="text" name="nome" value=""><br>
```

```
CPF:<br>
```

```
<input type="text" name="cpf" value=""><br>
```

```
Endereco:<br>
```

```
<input type="text" name="endereco" value=""><br>
```

```
...
```

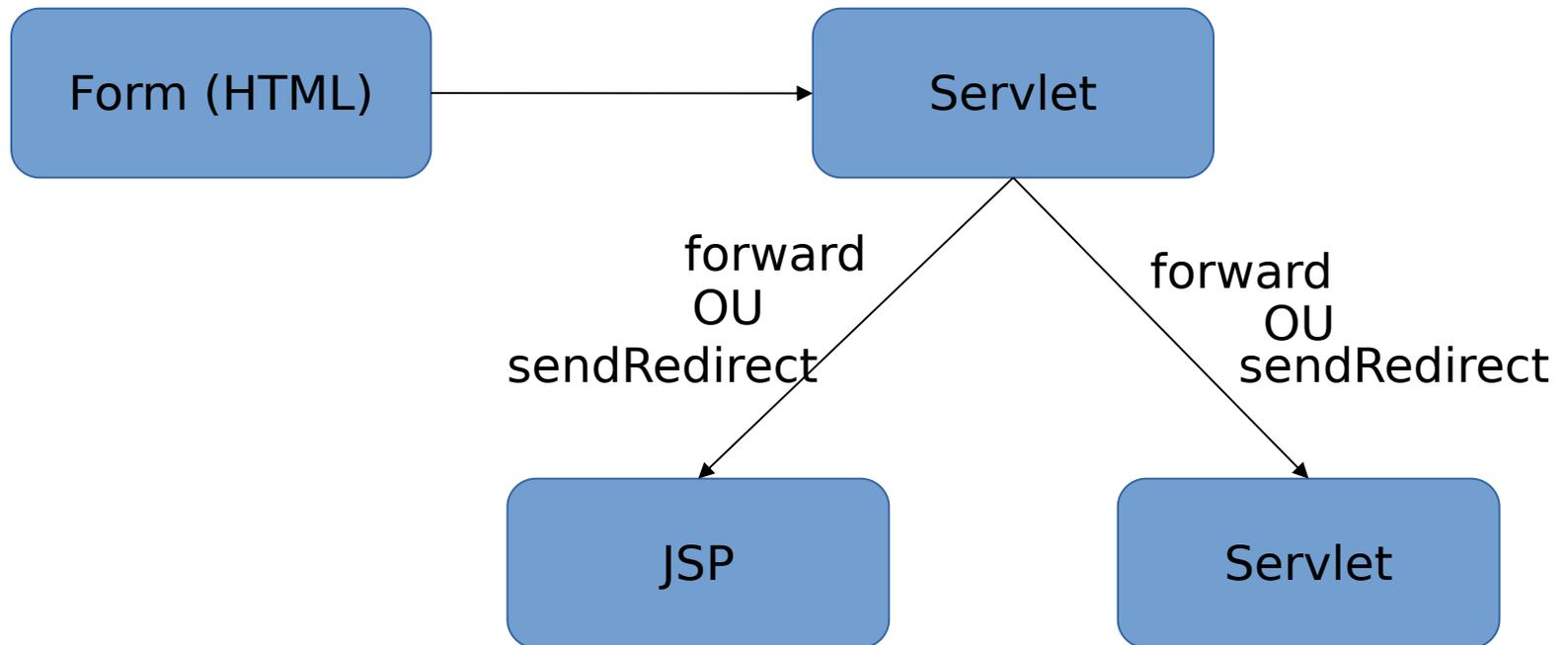
O envio pode ser por GET ou POST

Recebendo e exibindo dados

```
<body>
<%
    String nome = request.getParameter("nome");
    String cpf = request.getParameter("cpf");
    String end = request.getParameter("endereco");
    String sexo = request.getParameter("sexo");
    String tipo = request.getParameter("tipo");
    String comunicados = request.getParameter("comunicados");
    String obs = request.getParameter("obs");
%>

<h4>Cliente Cadastrado com Sucesso!</h4>
Nome: <%out.println(nome);%>
<br>
CPF: <%out.println(cpf);%>
<br>
Endereco: <%out.println(end);%>
...
```

Comunicação Servlet e JSP



Comunicação Servlet e JSP

Forward

Mantem todo o cabeçalho HTTP
(HttpServletRequest e HttpServletResponse)

Redireciona a solicitação sem a ajuda do
browser

sendRedirect

O cabeçalho HTTP é perdido

Diz ao browser qual a URL a ser acionada

Há todo o caminho de comunicação

Servidor → Browser → Servidor

Comunicação Servlet e JSP

Forward

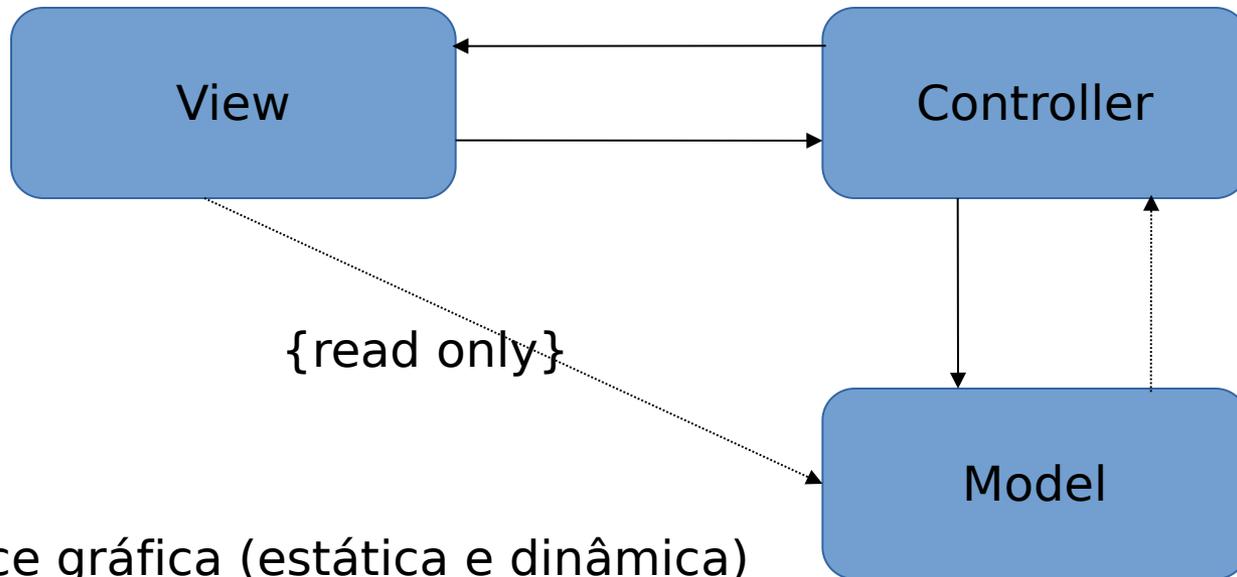
```
...  
// Chama diretamente o arquivo JSP  
RequestDispatcher rd =  
request.getRequestDispatcher("../retornoCadastro.jsp");  
rd.forward(request, response);  
...
```

sendRedirect

```
...  
/** Há uma comunicação com o browser, o qual efetua uma nova  
requisição  
ao recurso em questão... */  
response.sendRedirect("retornoCadastro.jsp?nome="+nome);  
...
```

O Modelo MVC

O Modelo MVC



View

- Interface gráfica (estática e dinâmica)

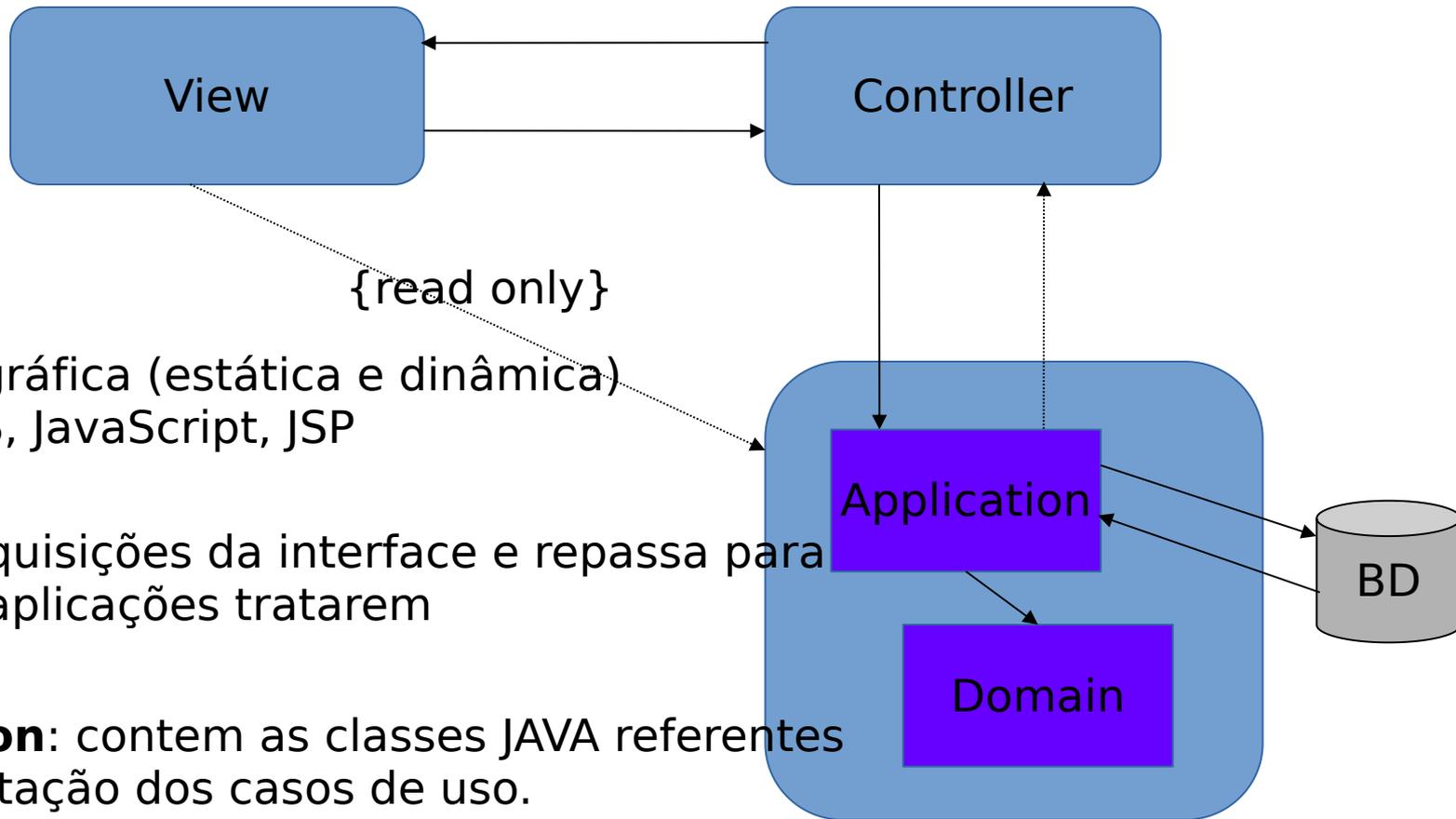
Controller

- Recebe requisições da interface e encaminha para os tratadores adequados

Model

- Componente independente. Recebe e trata requisições relacionada à lógica do domínio.

Nosso Modelo MVC



View

- Interface gráfica (estática e dinâmica)
- HTML, CSS, JavaScript, JSP

Controller

- Recebe requisições da interface e repassa para as devidas aplicações tratarem

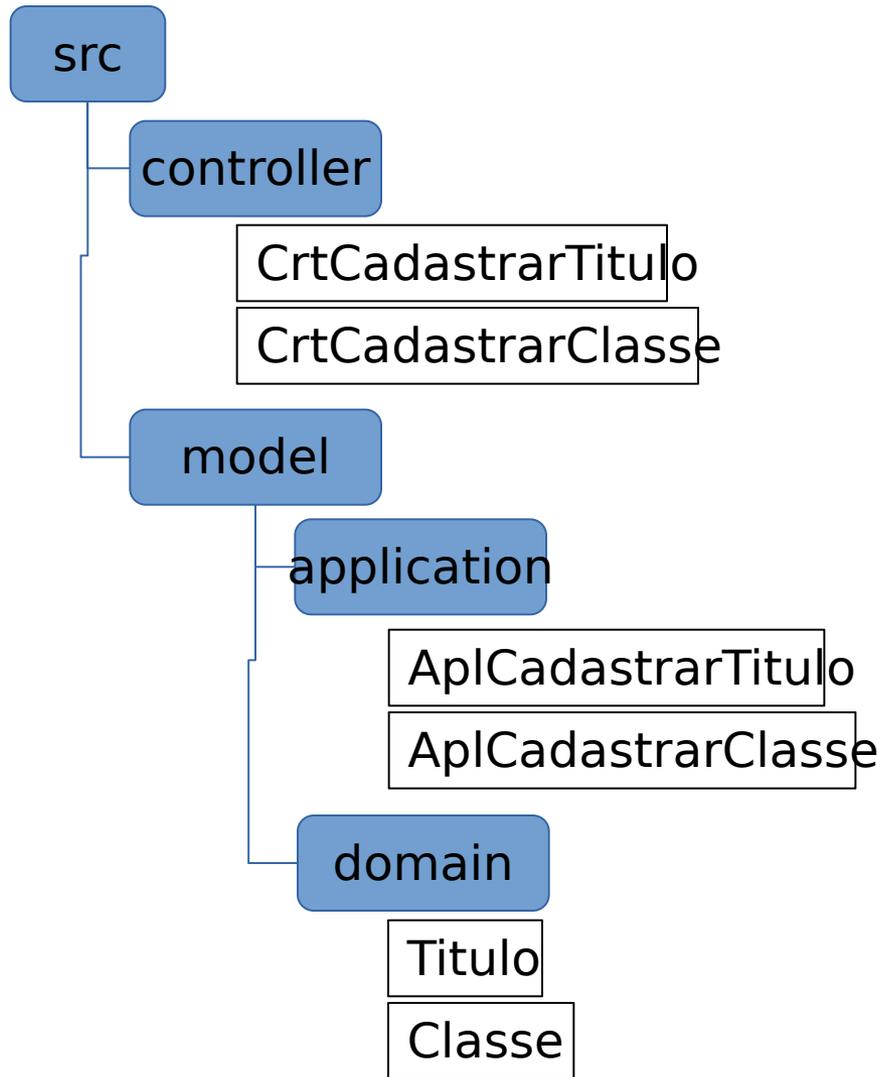
Model

- **Application:** contem as classes JAVA referentes à implementação dos casos de uso.

Estratégia de granularidade: Uma classe de aplicação por Caso de Uso.

- **Domain:** contem as classes JAVA referentes à modelagem de domínio do problema.

Estrutura de Pastas



WEB-INF

FormCadastrarTitulo.html

ListagemTitulos.jsp

DICA: Há a possibilidade de organizar controladores, aplicações, classes de domínio e páginas de GUI em subsistemas (ex., Atendimento a Cliente e Controle de Acervo)

Definição de Servlets

- Granularidade -

Granularidade

- Um Servlet por evento de caso de uso
- Um Servlet por caso de uso
- Um Servlet por pacote
- Um Servlet para todo sistema

Usar “doGET” e “doPOST” como pontos de entrada e saída.

Implementar a lógica de negócio em outros métodos/classes.
(MODULARIZAR)

Em breve discutiremos o modelo MVC...

Gerenciamento de Sessão

Noções Iniciais

O protocolo HTTP entre servidor e cliente

A cada comunicação entre servidor e cliente a conexão é fechada

A cada comunicação, o servidor não sabe quem é o solicitante

Gerenciamento de sessão

Associação entre solicitações HTTP e browsers cliente.

Gerência de informações fornecidas a e solicitadas por clientes durante a comunicação com o servidor

Técnicas Abordadas

Cookies

Objetos de sessão

Cookies

Cookie é uma pequena porção de informação que é trocada entre servidor e cliente nos cabeçalhos HTTP

Um cookie pode ser criado tanto no lado do cliente quanto no lado do servidor

Criando cookies

...

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

...

```
Cookie c1 = new Cookie ("meu_nome", "Pedro");
Cookie c2 = new Cookie ("meu_sobrenome", "Silva");
```

```
// Acrescenta o cookie ao cabeçalho HTTP
response.addCookie(c1);
response.addCookie(c2);
```

...

```
}
```

Recuperando cookies

...

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

...

```
    Cookie[] cookies = request.getCookies();
    for (int i=0; i < cookies.length; i++){
        Cookie c = cookies[i];
        // Exibe no console nome e valor de cada cookie
        System.out.println(c.getName());
        System.out.println(c.getValue());
    }
```

...

```
}
```

Desvantagens dos cookies

O usuário pode bloquear o uso

Pode, entretanto, verificar se o browser está habilitado ou não e avisar ao usuário

O usuário pode “rackear” o cookie

META tag

O método *sendRedirect* não funciona bem com cookies, uma vez que não reencaminha os cookies em uma nova solicitação...

Para tanto, use a tag META...

...

```
if (login (usuario, senha)) {  
    Cookie c1 = new Cookie ("usuario", usuario);  
    Cookie c2 = new Cookie ("senha", senha);  
    response.addCookie(c1);  
    response.addCookie(c2);  
    response.setContentType("text/html");  
    PrintWriter out = response.getWriter();  
    out.println("<META HTTP-EQUIV=Refresh CONTENT=0;  
URL=ContentServlet>");  
}
```

...

Cookies Persistentes

Os cookies, em geral, duram enquanto o browser estiver aberto.

É possível configurar cookies para que tenham um ciclo de vida maior.

...

```
Cookie c1 = new Cookie ("meu_nome", "Pedro");
```

```
/* Para definir o tempo máximo de vida de um cookie usamos o método  
setMaxAge fornecendo um valor inteiro representando a quantidade de  
segundos */
```

```
int ano = 60 * 60 * 24 * 365;  
c1.setMaxAge(ano);
```

```
// Acrescenta o cookie ao cabeçalho HTTP  
response.addCookie(c1);
```

```
...
```

```
}
```

Verificando configuração de cookie

Abordagem 1: enviar mensagem para o usuário como alerta (mesmo sem verificação).

Abordagem 2: verificar automaticamente a configuração de cookie.

Abordagem 2

- 1) Servidor: Envia uma mensagem com *cookie* para o *browser* forçando retorno
- 2) O *browser* retorna uma chamada para o servidor
- 3) Se o retorno do *browser* contiver o *cookie*, o *browser* aceita/suporta o uso de *cookies*

Abordagem 2 (Exemplo)

...

```
response.setContentType("text/html");
PrintWriter out = response.getWriter();
if (request.getParameter("flag") == null) {
    Cookie c = new Cookie ("teste", "ativo");
    response.addCookie(c);
    out.println("<META HTTP-EQUIV=Refresh CONTENT=0;
    URL="+request.getRequestURI()+"?flag=1>");
}
else {
    Cookie[] cookies = request.getCookies();
    boolean achou = false;
    for (int i=0; i < cookies.length; i++){
        Cookie c = cookies[i];
        if (c.getName().equals("teste") && c.getValue().equals("ativo"))
            achou = true;
    }
    if (achou)
        out.println("Configuracao permite cookie");
    else
        out.println("Configuracao NAO permite cookie");
}
}
```

...

Objeto Sessão

Técnica simples e poderosa

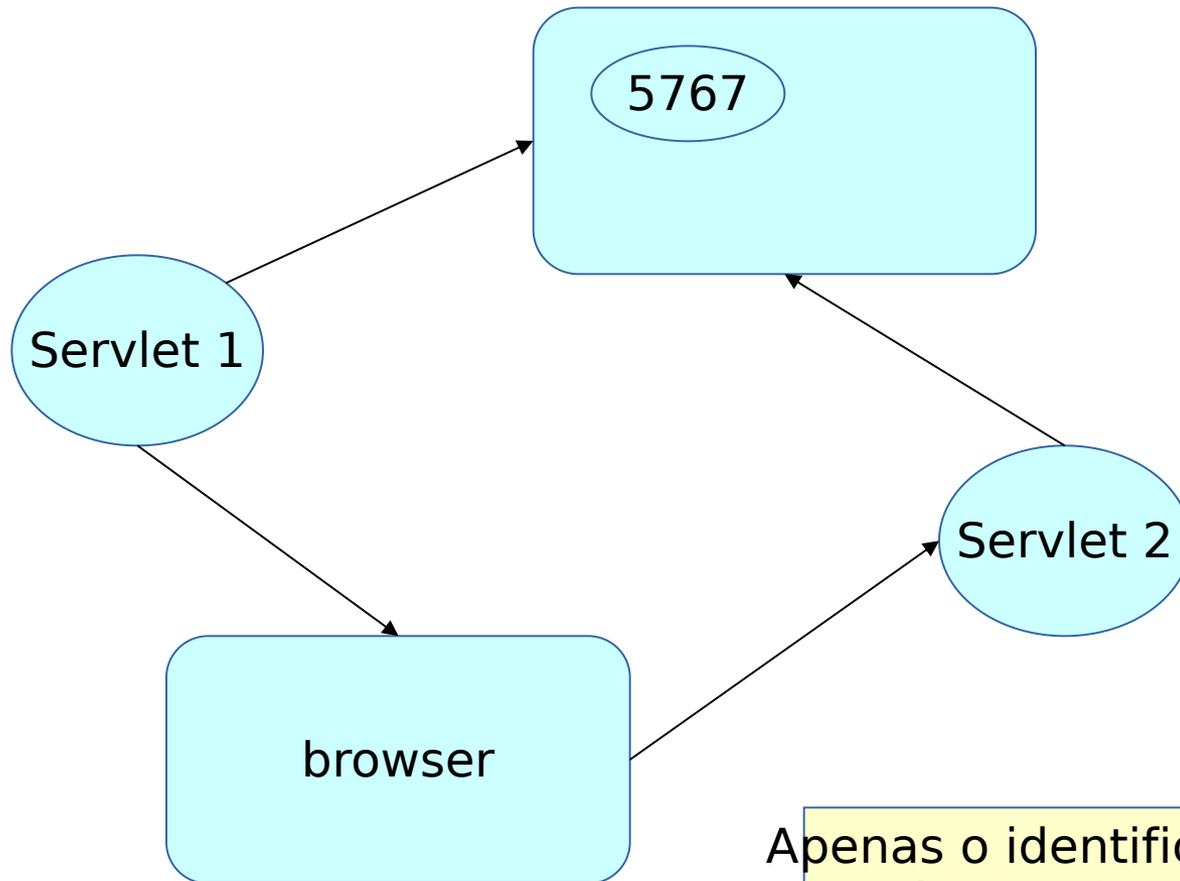
Cada usuário possui um objeto sessão
(*HttpSession*)

O objeto sessão é uma tabela de pares
chave / valor

O objeto sessão é acessível aos servlets/JSPs
da mesma aplicação

Em geral, usa cookies para a comunicação
servidor - cliente (transporte do ID da sessão)

Comunicação servidor - cliente



Apenas o identificador do objeto sessão é enviado nas comunicações servidor - cliente

Etapas da Comunicação

1. O servlet 1 cria o objeto sessão. Um identificador único é atribuído pelo servidor ao objeto.
2. O objeto sessão é enviado ao browser.
3. O browser chama um recurso qualquer no servidor por meio do servlet 2. Nessa requisição, o ID do objeto sessão é enviado.
4. O servlet 2 tem acesso ao objeto sessão por meio do método “getSession”, que busca o objeto sessão associado ao ID enviado no cabeçalho HTTP da requisição

Métodos de HttpSession

getSession()

getSession(boolean create)

getAttribute(String name)

removeAttribute(String name)

setAttribute(String name, Object attribute)

getId()

getLastAccessedTime()

getValue

...

Objeto Sessão (Exemplo)

CRIANDO A SESSÃO

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

// valida no banco de dados
if (login (usuario, senha)){
    HttpSession sessao = request.getSession(true);
    sessao.setAttribute("logado", new String("OK"));
    response.sendRedirect("home.jsp");
}
else {
    response.sendRedirect("erro_login.jsp");
}
...
```

Observe que diferentemente do exemplo com cookies, foi apenas utilizado UM atributo "logado" para indicar se o usuário está autorizado a acessar os recursos

Objeto Sessão (Exemplo)

TESTADO A SESSÃO

```
...
HttpSession sessao = request.getSession();

if (sessao == null)
    response.sendRedirect("pagina_login.jsp");
else{
    String status = (String) sessao.getAttribute("logado");
    If (!status.equals("OK"))
        response.sendRedirect("pagina_login.jsp");
}

// Login OK.
// Executa normalmente o código
...
}
...
```

Considerações Finais

Caso o espaço em memória ocupado pelos objetos de sessão possa comprometer o servidor, tais objetos podem ser persistidos (temporariamente) em memória secundária.

O uso de cookies podem ser conveniente para evitar que certas informações de usuário ocupem espaço (memória) no servidor.

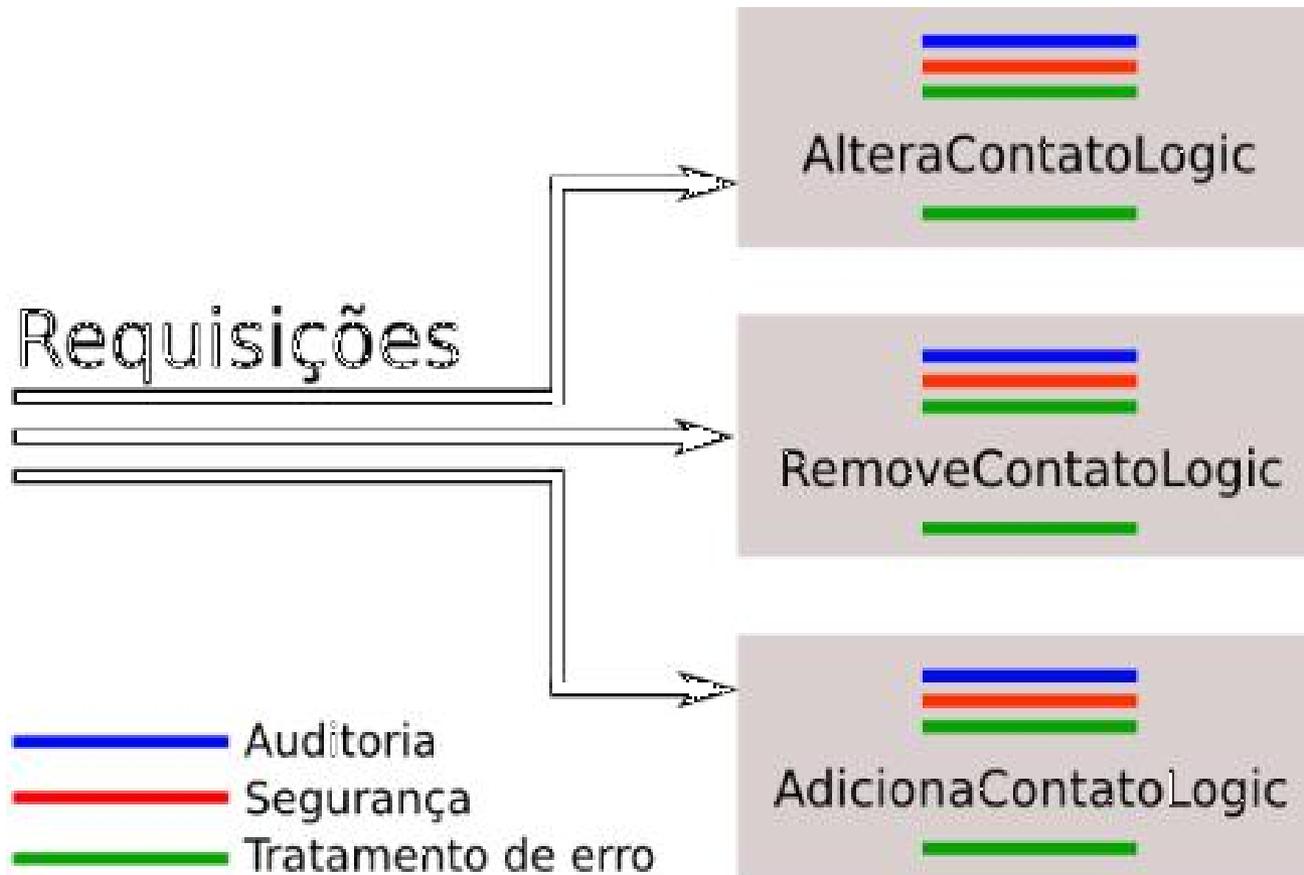
Filtros

Conceitos Iniciais

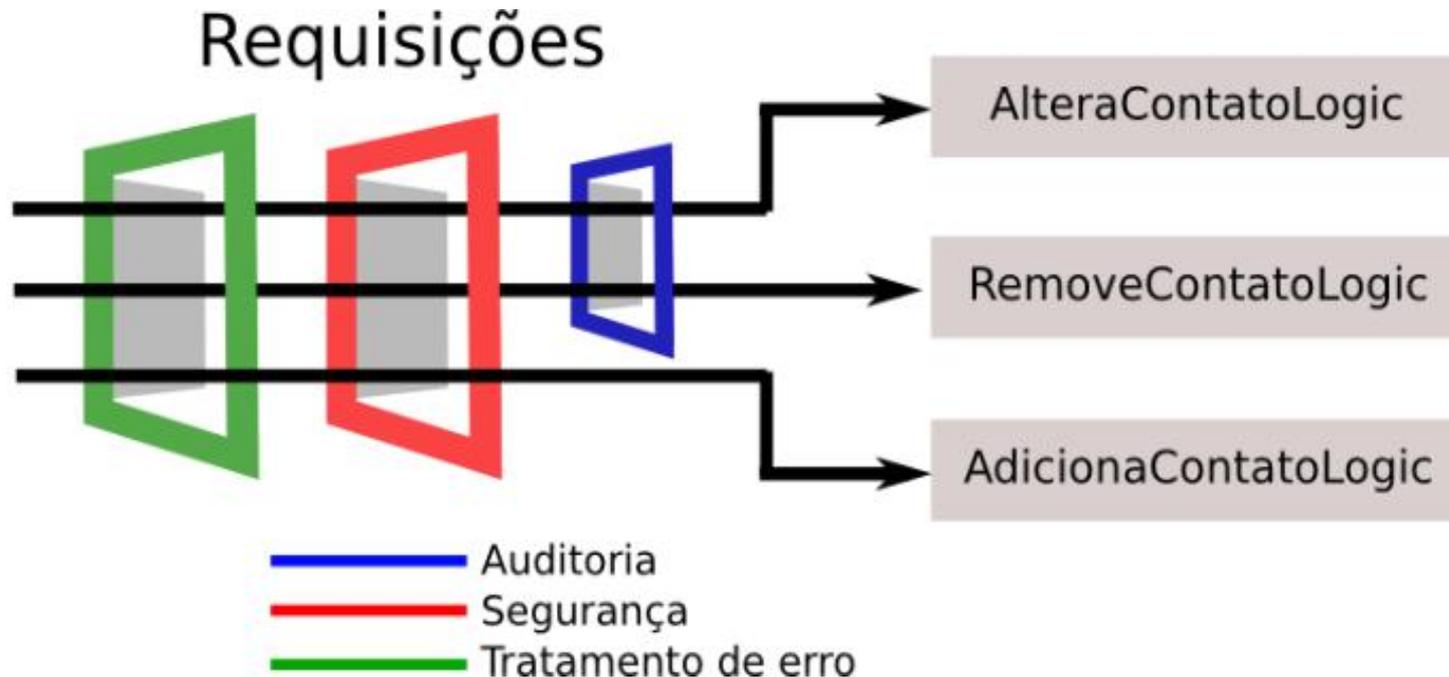
Auxilia o tratamento de requisitos que não são diretamente relacionados com a regra de negócio (não-funcionais)

- Ex.: Auditoria, Autenticação, Tratamento de erro, etc.

Conceitos Iniciais



Conceitos Iniciais



Filtros

“São classes que permitem que executemos código antes da requisição e também depois que a resposta foi gerada”

(CAELUM, 2016)

Conceitos Iniciais

“A grande vantagem é que cada requisito fica em um lugar só e conseguimos desacoplar nossas lógicas”.

(CAELUM, 2016)

Criando Filtros

Para cada filtro a ser criado:

Criar uma classe que implementa a interface `javax.servlet.Filter`

Declarar o filtro no `web.xml`, informando a quais URL's o filtro será aplicado

Criando Filtros

Exemplo de implementação da classe “MeuFiltro”

```
@WebFilter("/oi")
public class MeuFiltro implements Filter {
    public void doFilter(ServletRequest req,
        ServletResponse res, FilterChain chain) {
        // ...
    }
}
```

Desta forma indicamos que **todas as requisições vindas a partir de /oi serão filtradas** e, portanto, o filtro será aplicado em cada requisição.

Criando Filtros

Um filtro mais especifico...

Filtrar todas as requisições para paginas JSPs:

```
@WebFilter("/*.jsp")
public class MeuFiltro implements Filter {
    public void doFilter(ServletRequest req,
        ServletResponse res, FilterChain chain) {
        // ...
    }
}
```

Criando Filtros

Criando um filtro mais amplo

Filtra TODAS as requisições da aplicação:

```
@WebFilter("/*")
public class MeuFiltro implements Filter {
    public void doFilter(ServletRequest req,
        ServletResponse res, FilterChain chain) {
        // ...
    }
}
```

Criando Filtros

Configuração de um filtro no web.xml:

```
<filter>  
  <filter-name>meuFiltro</filter-name>  
  <filter-class>br.com.caelum.filtro.MeuFiltro</filter-  
    class>  
</filter>
```

```
<filter-mapping>  
  <filter-name>meuFiltro</filter-name>  
  <url-pattern>/*</url-pattern>  
</filter-mapping>
```

A Interface Filter

Possui 3 métodos

Init

Destroy

doFilter

doFilter: fará todo o processamento do filtro.

Recebe três parâmetros:

ServletRequest

ServletResponse

FilterChain

A Interface Filter

```
@WebFilter("/*")
public class FiltroTempoDeExecucao implements Filter {

    // implementação do init e destroy

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

        // todo o processamento vai aqui
    }
}
```

Filter Chain

FilterChain (a cadeia de filtros): permite indicar ao container que o request deve prosseguir seu processamento

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

    // processamento do filtro

    // continua a (chamando, por exemplo, o servlet ou JSP)
    chain.doFilter(request, response);
}
```

“A Volta...”

“Qualquer código colocado antes da chamada `chain.doFilter(request,response)` será executado na ida, qualquer código depois na volta”.

Pode-se com isso, por exemplo, abrir um recurso (conexão ou transação) na ida e na fechar/liberar o recurso na volta.

“A Volta...”

```
@WebFilter("/*")
public class FiltroTempoDeExecucao implements Filter {
    public void doFilter(ServletRequest request,
        ServletResponse response, FilterChain chain)
        throws IOException, ServletException {
        long tempoInicial = System.currentTimeMillis();

        chain.doFilter(request, response);

        long tempoFinal = System.currentTimeMillis();
        String uri = ((HttpServletRequest)request).getRequestURI();
        String parametros = ((HttpServletRequest) request).getParameter("logica");
        System.out.println("Tempo da requisicao de " + uri
            + "?logica="
            + parametros + " demorou (ms): "
            + (tempoFinal - tempoInicial));
    }
    // métodos init e destroy omitidos
}
```

Exemplo de Uso: Autenticação

```
/**
 * @see Filter#doFilter(ServletRequest, ServletResponse, FilterChain)
 */
public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
throws IOException, ServletException {

    RequestDispatcher rd = request.getRequestDispatcher("index.html");
    HttpSession sessao = ((HttpServletRequest)request).getSession();

    if (sessao == null)
        rd.forward(request, response);
    else{
        String status = (String) sessao.getAttribute("logado");
        if (status == null || !status.equals("OK"))
            rd.forward(request, response);
        else{
            // pass the request along the filter chain
            chain.doFilter(request, response);
        }
    }
}
```

Exemplo de Uso: Logging

```
/**
 * @see Filter#doFilter(ServletRequest, ServletResponse, FilterChain)
 */
public void doFilter(ServletRequest request, ServletResponse response, FilterChain
    chain) throws IOException, ServletException {

    // 1) Abrir ou criar um arquivo de Log.

    // 2) Gravar informações no arquivo de log. Ex.: Usuário logado, ação
    //executada (campo hidden operação) e valor dos atributos do form.

    // pass the request along the filter chain
    chain.doFilter(request, response);
}
```

Exemplo de Uso: Controle de Transação

Sempre que chegar uma requisição para a nossa aplicação, uma conexão deve ser aberta.

Depois que essa requisição for processada, a conexão deve ser fechada.

Adicionar também o tratamento de transação.

Exemplo de Uso

```
@WebFilter("/*")
public class FiltroConexao implements Filter {
    // implementação do init e destroy, se necessário

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

        // abre uma conexão
        Connection connection = new ConnectionFactory()
            .getConnection();

        // indica que o processamento do request deve prosseguir
        chain.doFilter(request, response);

        // fecha conexão
        connection.close();
    }
}
```

Exemplo de Uso

```
public class FiltroJPA implements Filter {
```

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

```
    Session session = HibernateUtil.getSession();
```

```
    try {
```

```
        session.beginTransaction();
```

```
        chain.doFilter(request, response);
```

```
        session.getTransaction().commit();
```

```
    } catch (Exception e) {
```

```
        session.getTransaction().rollback();
```

```
        e.printStackTrace();
```

```
    } finally {
```

```
        if (session != null && session.isOpen()) {
```

```
            session.close();
```

```
        }
```

```
    }
```

```
}
```



```
Session session =  
sFactory.openSession();
```

Exemplo de Uso

```
/**
 * @see Filter#init(FilterConfig)
 */
public void init(FilterConfig fConfig) throws ServletException {

    sFactory = new
    AnnotationConfiguration().configure().buildSessionFactory();

}
```

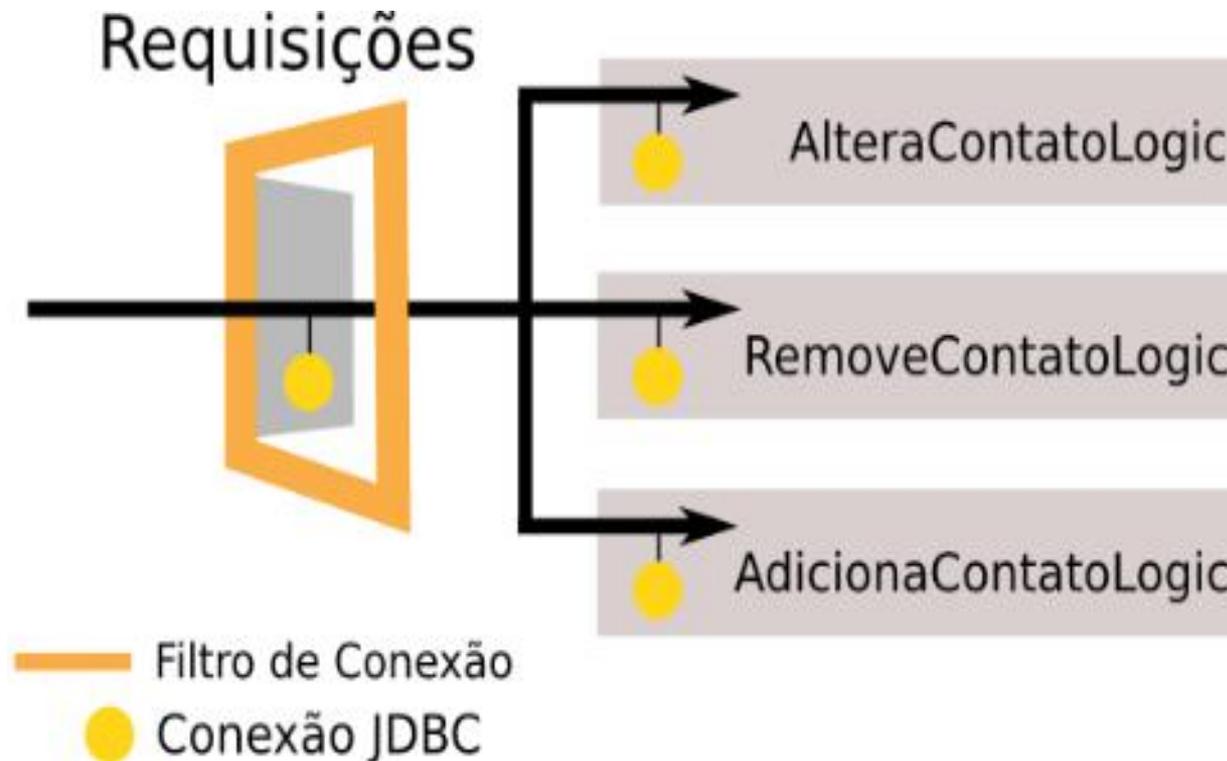
Exemplo de Uso

Abrimos uma conexão no começo dos requests, mas...

Como acessá-la? Como, dentro de uma Servlet, pegar um objeto criado dentro de um filtro, uma outra classe?

A ideia é associar (pendurar) a conexão criada ao request atual. Tanto o filtro quanto a Servlet estão no mesmo request e as conexões vão ser abertas por requests.

Exemplo de Uso



Exemplo de Uso

Método **setAttribute** do request: guarda algo na requisição

Recebe como parâmetro: uma identificação para o objeto (chave) que estamos guardando e também o próprio objeto para ser guardado no request.

Exemplo de Uso

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

```
        Session session = sFactory.openSession();  
        try {  
            session.beginTransaction();  
            request.setAttribute("sessaoBD", session);  
            chain.doFilter(request, response);  
            session.getTransaction().commit();  
        } catch (Exception e) {  
            session.getTransaction().rollback();  
            e.printStackTrace();  
        } finally {  
            if (session != null && session.isOpen()) {  
                session.close();  
            }  
        }  
    }
```

```
}
```

Exemplo de Uso

O Filtro será o único ponto da nossa aplicação que criará conexões.

Na sequência, servlets e JSPs podem obter a conexão guardada no request (usando “getAttribute”) e utilizá-la como necessário.