

Java Enterprise Edition - JEE

10. Filter API



Esp. Márcio Palheta
gtalk: marcio.palheta@gmail.com



Agenda

- Interceptors
- API Filter
- Ciclo de vida
- Mapeamento
- Exercícios



Pedágio

- Duas cidades vizinhas;
- Única via de acesso;
- Instalação de pedágio;
- Controle do fluxo;
- Interceptação;



Filtros

- A API de **servlets** oferece um design pattern que permite a execução de tarefas, sem se preocupar com o que aconteceu ou vai acontecer, um **filtro**.
- A idéia consiste em um método de interceptação chamado **doFilter**, responsável por executar uma atividade qualquer e, em seguida, seguir o fluxo normal da aplicação

Estrutura padrão – Filter

```
//Mapeamento para interceptar TODAS as requisicoes
@WebFilter("/*")
//Classe que implementa a interface Filter
public class Filtro implements Filter {
    //Metodo executado quando o filtro for carregado
    public void init(FilterConfig fConfig) throws ServletException {
    }
    //Metodo executado quando uma requisicao for interceptada
    public void doFilter(ServletRequest request, ServletResponse response,
        FilterChain chain) throws IOException, ServletException {
        //Codigo a ser executado ANTES da requisicao

        //Executando a requisicao
        chain.doFilter(request, response);

        //Codigo a ser executado APOS a requisicao
    }
    //Metodo executado quando o filtro for destruido
    public void destroy() {
    }
}
```



Utilidade dos filtros

- O uso clássico utilizado para filtros é para
- logar informações referentes a requisição
- abrir e fechar transações,
- descriptografar informações que foram enviadas (antes) e criptografar informações antes de enviar (depois)
- compactar os dados a serem enviados ao cliente (depois).



Exercício 07

- Criação da classe `FiltroDeTempoRequisicao` que implementa `javax.servlet.Filter`;
- Implemente o método `doFilter()`;
- Configure o arquivo `web.xml` para que todas as páginas `.jsp` sejam monitoradas pelo novo filtro;
- Inicie o servidor e acessa as páginas JSP;

FiltroDeTempoRequisicao.java

```
package br.fucapi.curso.jee.control;
import java.io.IOException;
@WebFilter("*.jsp")
public class FiltroDeTempoRequisicao implements Filter {

    public void init(FilterConfig fConfig) throws ServletException {
        System.out.println("Inicio do filtro");
    }
    public void destroy() {
        System.out.println("Fim do filtro");
    }
    public void doFilter(ServletRequest request, ServletResponse response,
        FilterChain chain) throws IOException, ServletException {
        //Processamento antes da requisicao
        long horaInicio = System.currentTimeMillis();
        //Execucao da requisicao do usuario
        chain.doFilter(request, response);
        //Processamento apos a requisicao do usuario
        long horaFim = System.currentTimeMillis();
        System.out.println("URL: "+((HttpServletRequest)request).getRequestURL());
        System.out.println("Tempo em millis: "+(horaFim-horaInicio));
    }
}
```

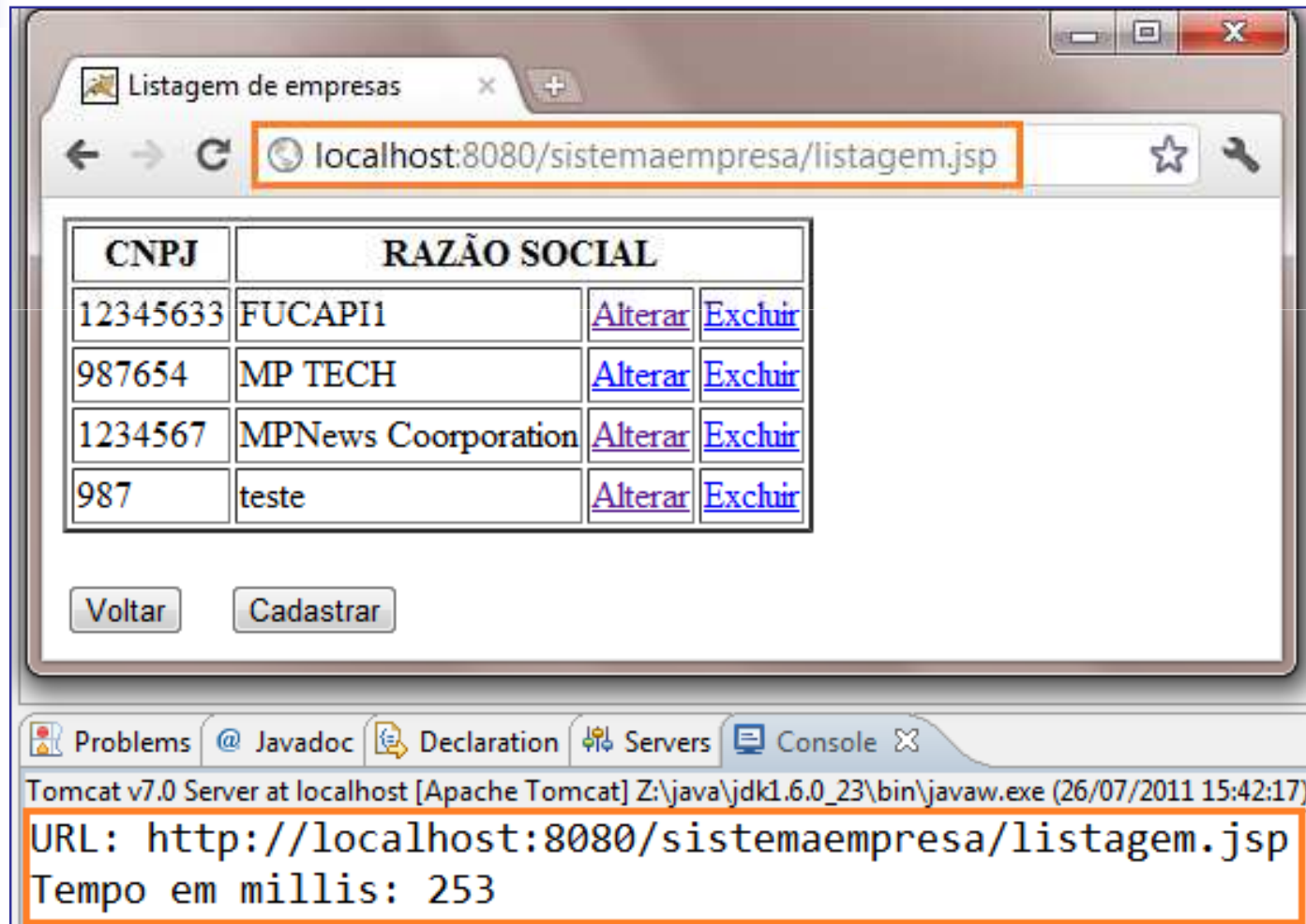



Como era o mapeamento no arquivo web.xml ?

```
<filter>
  <filter-name>FiltroPaginas</filter-name>
  <filter-class>br.fucapi.cpge.jee.view.LogFiltro</filter-class>
</filter>

<filter-mapping>
  <filter-name>FiltroPaginas</filter-name>
  <url-pattern>*.jsp</url-pattern>
</filter-mapping>
<servlet>
```

Teste do filtro



The screenshot shows a web browser window titled "Listagem de empresas" with the address bar displaying "localhost:8080/sistemaempresa/listagem.jsp". The page contains a table with company data and two buttons at the bottom.

CNPJ	RAZÃO SOCIAL		
12345633	FUCAPI1	Alterar	Excluir
987654	MP TECH	Alterar	Excluir
1234567	MPNews Corporation	Alterar	Excluir
987	teste	Alterar	Excluir

Below the table are two buttons: "Voltar" and "Cadastrar".

The console window at the bottom shows the following log message:

```
Tomcat v7.0 Server at localhost [Apache Tomcat] Z:\java\jdk1.6.0_23\bin\javaw.exe (26/07/2011 15:42:17)  
URL: http://localhost:8080/sistemaempresa/listagem.jsp  
Tempo em millis: 253
```



A criação de conexões

- Até aqui, cada método dos nosso objetos DAOs **abrem** e **fecham** suas conexões;
- O que acontece quando executamos vários métodos em uma mesma requisição?
- Como melhorar o uso de conexões?



Uma conexão por requisição

- É uma boa estratégia:
 - Criar conexão no início da requisição;
 - Usar a conexão pelos DAOs chamados;
 - Fechar a conexão no fim da requisição;
- Com isso, aproveitamos o mesmo objeto connection em várias chamadas à camada de modelo;



Uma conexão por requisição

- Neste cenário, os métodos do DAO **não** precisam **gerenciar a conexão**, mas apenas utilizar seus recursos;
- O objeto Connection a ser usado pelo DAO será **injetado** em seu método construtor;



Inversão de controle – IoC e Injeção de Dependência - DI


- IoC - A classe **EmpresaDAO** não cria mais seu objeto **Connection**, mas o recebe via método construtor;
- A classe **EmpresaDAO** depende de uma **Connection**;
- **DI** – a dependência de **EmpresaDAO** será injetada via construtor;



Implementando IoC e DI

- Podemos utilizar a API Filter para implementar IoC e DI;
- Com isso, a abertura e fechamento da **Connection** necessária em **DAO** fica a cargo do filtro de conexões;
- A seguir, vamos atualizar o projeto sistemaempresa, para incluir **IoC** e **DI**

Filtro de conexão



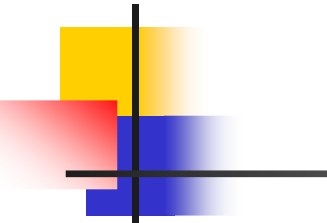
```
package br.fucapi.curso.jee.control;
import java.io.IOException;
@WebFilter("/*")
public class ConnectionFilter implements Filter {
    public void init(FilterConfig fConfig) throws ServletException {
        System.out.println("Inicio do filtro de conexoes");
    }
    public void destroy() {
        System.out.println("Fim do filtro de conexoes");
    }
    public void doFilter(ServletRequest request, ServletResponse response,
        FilterChain chain) throws IOException, ServletException {
        Connection connection = ConnectionFactory.getConnection();
        // Colocando a connection na requisicao
        request.setAttribute("connection", connection);
        // Execucao da requisicao
        chain.doFilter(request, response);
        // Fechamento da conexao
        try {
            connection.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```




Atualizações nas camadas de modelo e controle

- Crie na classe **EmpresaDAO** um atributo **Connection** e um construtor que receba esse objeto;
- Remova os controles para abrir e fechar conexões de EmpresaDAO;
- Atualize a **ServletController** para que seja recuperada a Connection criada pelo Filter e passada à EmpresaDAO;

Atualização da EmpresaDAO



```
public class EmpresaDAO {  
    private Connection connection;  
  
    public EmpresaDAO(Connection connection) {  
        this.connection = connection;  
    }  
    //O metodo NAO gerencia a conexao  
    public void excluir(Long id) {  
        // criação da String SQL a ser executada  
        String sql = "Delete from empresa Where id = ?";  
        // Criação do objeto a executar o comando SQL  
        PreparedStatement stmt = null;  
        try {  
            // Usa o atributo connection  
            stmt = connection.prepareStatement(sql);  
            // Carga dos parâmetros da instrução SQL  
            stmt.setLong(1, id);  
            // Execução do comando de exclusão  
            stmt.execute();  
        } catch (SQLException e) {  
            e.printStackTrace();  
        }  
    }  
    // Outros métodos aqui
```



Atualização da ServletController

```
@WebServlet("/controller")
public class ServletController extends HttpServlet {
    private static final long serialVersionUID = 1L;

    //Objeto DAO utilizado para invocar servicos da camada de modelo
    private EmpresaDAO dao;

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

        Connection connection = (Connection)request.getAttribute("connection");
        dao = new EmpresaDAO(connection);

        //Continuacao do código do metodo service()...
```



O que vem a seguir?

- Struts framework;
- MOR com hibernate 3;
- MVC usando Struts e Hibernate;
- Novos padrões de projeto;
- Modelo de arquitetura JEE;



Web Archive (.war)

- O processo padrão de **deploy** de uma aplicação web é a criação de um arquivo de extensão war, que é um arquivo zip com o diretório base da aplicação sendo a raiz do zip.
- No projeto **sistemaempresa**, todo o conteúdo do diretório web deveria ser incluído em um arquivo **sistemaempresa.war**.
- Após compactar o diretório web com esse nome, efetuaremos o deploy, copiando o arquivo **.war** para o diretório TOMCAT/webapps/;
- O novo contexto chamado **sistemaempresa** estará disponível.



Atividades do projeto final

- Definição de padrão de telas;
- Implementação de protótipo;
- Teste de navegabilidade;
- Validação do protótipo;



Referências

- www.caelum.com.br
- Hall, Marty, "Core Servlets and Java Server Pages", Janeiro 2002, Sun Microsystems Press;
- <http://java.sun.com/j2ee/1.6/docs/tutorial/doc/index.html>
- <http://java.sun.com/products/jndi/docs.html>
- <http://java.sun.com/blueprints/corej2eepatterns/Patterns/index.html>



Java Enterprise Edition - JEE

10. Filter API



Esp. Márcio Palheta
gtalk: marcio.palheta@gmail.com