FTCE

Faculdade de Tecnologia e Ciências Exatas

USJT - 2017 - Práticas de Programação

Professores Teoria: Bonato, Hamilton
Laboratório: Calvetti, Hamilton, Keity e Rodrigo

Aula: 10
Assunto: Filtros

Conceitos básicos

Há vários códigos em programação que não estão exatamente relacionados às regras de negócio, mas sim a tarefas como autenticação e autorização (login), geração de arquivos de logs, manutenção da conexão com o banco de dados, etc.

Se formos implementar estas funcionalidades sem uma técnica especial, veremos que teremos que repetir o mesmo código em quase todas as classes de negócio: código para verificar se o usuário esta logado, código para abrir e fechar conexões do banco, etc. Uma das técnicas que evitam esta repetição de código é a Programação Orientada a Aspectos (POA) [LADDAD 2009] [TORSTEN 2008], que está fora do escopo deste curso, mas a bibliografia mencionada está no final do documento para quem estiver interessado em se aprofundar.

Um outra forma de se implementar estas funcionalidades é por meio de Filtros de Servlet. O filtro é uma interface do pacote javax.servlet que tem a seguinte característica: **são invocados pelo contêiner antes e depois da execução do servlet ou JSP ao qual o filtro está aplicado.** Desta forma, é possível executar código antes e depois do servlet ou JSP (que é também um servlet) terminarem de executar para fazerem o que você precisa que seja feito.

Implementação

```
package filter;

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;
import javax.servlet.annotation.WebFilter;

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

```
    public ExemploFilter() {
        // Coloque aqui código que precisa ser inicializado no construtor
    }

    public void destroy() {
        // Coloque aqui código para ser executado quando o filtro for
        // descarregado pelo contêiner
    }

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

        // Coloque aqui código para ser executado antes da chamada do servlet

        chain.doFilter(request, response);

        // Coloque aqui código para ser executado depois da chamada do servlet
    }

    public void init(FilterConfig fConfig) throws ServletException {
        // Coloque aqui código para ser excutado quando o filtro for
        // carregado pelo contêiner
    }

}
```

Os métodos init e destroy tem a mesma função que os métodos análogos do Servlet: serem executados quando o contêiner carrega ou descarrega o filtro.

O método doFilter é onde a "ação" realmente acontece. Se quiser que o filtro verifique se o usuário está logado no sistema, faça isso antes de chamar o FilterChain.doFilter(HttpServletReqest, HttpServletResponse). Se quiser gravar um log do que foi feito pelo usuário, o faça depois que o doFilter retornar.

Portanto, as requisições passam por dentro do Filtro.

Endereçamento

Para definir quais requisições serão filtradas por um determinado filtro, use a anotação **@WebFilter**.

Alguns exemplos:

Para filtrar as requisições de qualquer URL da aplicação, use:

```
@WebFilter("/*")
```

Para filtrar as requisições de uma determinada URL da aplicação, use:

```
@WebFilter("/controller.do")
```

Para filtrar uma lista de URLs, use:

```
@WebFilter(name = "meu_filtro" urlPatterns = {
"/controller.do", "/cadastro.do"})
```

Para filtrar as requisições de uma lista de servlets, use:

```
@WebFilter(name = "meu_outro_filtro" servletNames = {
"Servlet1", "Servlet2"})
```

Exemplo

Para implementar um teste de login e uma geração de logs de auditoria na aplicação, pode-se criar dois filtros, ambos aplicados ao servlet **controller.do** (assumindo que você já está usando os patterns Command e Front Controller.

Para isso você deve criar:

- model: classes Usuario, UsuarioDAO e UsuarioService
- view: Login.jsp
- tabela: Usuario
- filtros: LogFilter e LoginFilter

Exemplo Model: Usuario.java

```
package model;

import java.io.Serializable;

public class Usuario implements Serializable {
   private static final long serialVersionUID = 1L;

   private String username;
   private String password;

   public String getPassword() {
      return password;
   }

   public void setPassword(String password) {
      this.password = password;
   }

   public String getUsername() {
      return username;
   }

   public void setUsername(String username) {
      this.username = username;
   }
```

```
   @Override
   public String toString() {
      return "Usuario [username=" + username + ", password=" + password
            + "]";
   }
}
```

Exemplo Service: UsuarioService.java

```
package service;

import model.Usuario;
import dao.UsuarioDAO;

public class UsuarioService {

   public boolean validar(Usuario usuario){
      UsuarioDAO dao = new UsuarioDAO();
      return dao.validar(usuario);
   }
}
```

Exemplo Model: Usuario.java

```
package model;

import java.io.Serializable;

public class Usuario implements Serializable {
   private static final long serialVersionUID = 1L;

   private String username;
   private String password;

   public String getPassword() {
      return password;
   }

   public void setPassword(String password) {
      this.password = password;
   }

   public String getUsername() {
      return username;
   }

   public void setUsername(String username) {
      this.username = username;
   }

   @Override
   public String toString() {
      return "Usuario [username=" + username + ", password=" + password
            + "]";
```

```
   }
}
```

Exemplo Command: FazerLogin.java

```
package command;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import model.Usuario;
import service.UsuarioService;

public class FazerLogin implements Command {

   @Override
   public void executar(HttpServletRequest request, HttpServletResponse
response)
         throws ServletException, IOException {
      String nome = request.getParameter("username");
      String senha = request.getParameter("passwd");

      Usuario usuario = new Usuario();
      usuario.setUsername(nome);
      usuario.setPassword(senha);
      UsuarioService service = new UsuarioService();

      if(service.validar(usuario)){
         HttpSession session = request.getSession();
         session.setAttribute("logado", usuario);
         System.out.println("Logou "+usuario);
      } else {
         System.out.println("Não Logou "+usuario);
         throw new ServletException("Usuário/Senha inválidos");
      }
      response.sendRedirect("index.jsp");
   }

}
```

Exemplo View: Login.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %><!DOCTYPE
html>
<html lang="pt-br">

<head>
    <meta charset="utf-8">
```

```
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Cadastros</title>

    <link href="css/bootstrap.min.css" rel="stylesheet">
    <link href="css/style.css" rel="stylesheet">
</head>

<body>
    <!-- Barra superior com os menus de navegacao -->
   <c:import url="Menu.jsp"/>
    <!-- Container Principal -->
    <div id="main" class="container">
       <h3 class="page-header">Login</h3>
       <!-- Formulario de Login -->
        <form action="controller.do" method="post">
          <div class="row col-md-12">
          <div class="form-group">
             <div class="input-group col-md-4">
               <div class="input-group-addon"><span class="glyphicon
glyphicon-envelope" aria-hidden="true"></span></div>
               <input type="email" name="username" id="username"
class="form-control" maxlength="60" placeholder="E-mail" required/>
             </div>
          </div>
          <div class="form-group">
             <div class="input-group col-md-4">
               <div class="input-group-addon"><span class="glyphicon
glyphicon-option-horizontal" aria-hidden="true"></span></div>
               <input type="password" name="passwd" id="passwd" class="form-
control" placeholder="Senha" required/>
             </div>
          </div>
          </div>
          <div class="row col-md-12">
             <button type="submit" class="btn btn-primary" name="command"
value="FazerLogin"><span class="glyphicon glyphicon-ok" aria-
hidden="true"></span> Ok</button>
          </div>
        </form>
        </div>
    <script src="js/jquery.min.js"></script>
    <script src="js/bootstrap.min.js"></script>
</body>

</html>
```

Exemplo Filtro: LogFilter.java

```
package filter;

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;
import javax.servlet.annotation.WebFilter;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;

import model.Usuario;

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

   public void destroy() {
      // TODO Auto-generated method stub
   }

   public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain) throws IOException, ServletException {
      // place your code here
      request.setCharacterEncoding("UTF-8");
      response.setCharacterEncoding("UTF-8");
      HttpServletRequest req = (HttpServletRequest)request;
      HttpSession session = req.getSession();
      Usuario usuario = (Usuario)session.getAttribute("logado");

      if(usuario == null){
         System.out.println(req.getParameter("command"));
      } else {
         System.out.println(usuario.getUsername()+ " -> " +
req.getParameter("command"));
      }
      // pass the request along the filter chain
      chain.doFilter(request, response);
      if(usuario == null){
         System.out.println(req.getParameter("command"));
      } else {
         System.out.println(req.getParameter("command")+" -> " +
usuario.getUsername());
      }
   }

   public void init(FilterConfig fConfig) throws ServletException {
      // TODO Auto-generated method stub
   }

}
```

Exemplo Filtro: LoginFilter.java

```
package filter;

import java.io.IOException;

import javax.servlet.Filter;
```

```java
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebFilter;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import model.Usuario;

@WebFilter("/controller.do")
public class LoginFilter implements Filter {

   public void destroy() {
   }

   public void doFilter(ServletRequest request, ServletResponse response,
         FilterChain chain) throws IOException, ServletException {
      // place your code here
      request.setCharacterEncoding("UTF-8");
      response.setCharacterEncoding("UTF-8");
      HttpServletRequest req = (HttpServletRequest) request;
      HttpSession session = req.getSession();
      Usuario logado = (Usuario) session.getAttribute("logado");
      String path = req.getContextPath();
      String uri = req.getRequestURI();
      String comando = req.getParameter("command");
      if(comando == null){
         comando = "";
      }

      if (logado == null && !uri.equals(path + "/Login.jsp")
            && !comando.equals("FazerLogin")) {
         ((HttpServletResponse) response).sendRedirect(path + "/Login.jsp");
      } else {
         // pass the request along the filter chain
         chain.doFilter(request, response);
      }
   }

   public void init(FilterConfig fConfig) throws ServletException {
   }

}
```

Exemplo Tabela: Usuario

```sql
USE DATABASE vendas;
CREATE TABLE usuario (
  username VARCHAR(100) NOT NULL,
  password VARCHAR(100) NOT NULL,
  PRIMARY KEY (username))
ENGINE = InnoDB
```

```
DEFAULT CHARACTER SET = utf8;
```

Exercícios

1. Revisão da Teoria

a. Importe o arquivo exemplo_clientes.war. Faça deployment no Tomcat e rode. Depois analise o código para entendê-lo.

2. Exercício para Nota (correção em aula)

a. Implemente a partir do código com o FrontController do projeto de Países da aula passada os filtros de Login e de Log.
Depois, **i**mplemente um filtro para abrir e fechar a conexão do banco de dados:

1. Pegue a conexão antes do chain.doFilter;
2. Implemente um Singleton usando a classe ConnectionFactory para armazenar lá a conexão; veja o apêndice.
3. Feche a conexão depois do chain.doFilter.

Obs: esta implementação não é thread safe e irá causar problemas de deadlock com as conexões. Para uma implementação thread safe, use a classe ThreadLocal<T> do pacote java.lang. Veja o exemplo no final deste documento.

Bibliografia

DEITEL, Harvey M.; DEITEL, Paul J; FURMANKIEWICZ, Edson. **Java: como programar.** 6. ed. atual. São Paulo: Pearson, 2005. xl, 1110 p. ISBN 8576050196 (Broch.)

BASHAM, Bryan; SIERRA, Kathy; BATES, Bert. **Use a cabeça!: Servlets & JSP**. 2. ed. Rio de Janeiro: Alta Books, 2008-2010. xxxii, 879 p. ISBN 9788576082941 (broch.)

FOWLER, Martin.; **Padrões de Arquitetura de Aplicações Corporativas.** Bookman, 2008.

LADDAD, R.; **AspectJ in Action.** Manning Publications, 2009.

TORSTEN, Nelson; **Programação Orientada a Aspectos com AspectJ.** Disponível online em <http://www.cin.ufpe.br/~acnlf/curso_poa.pdf>. Acessado em 09/05/2016.

Apêndice

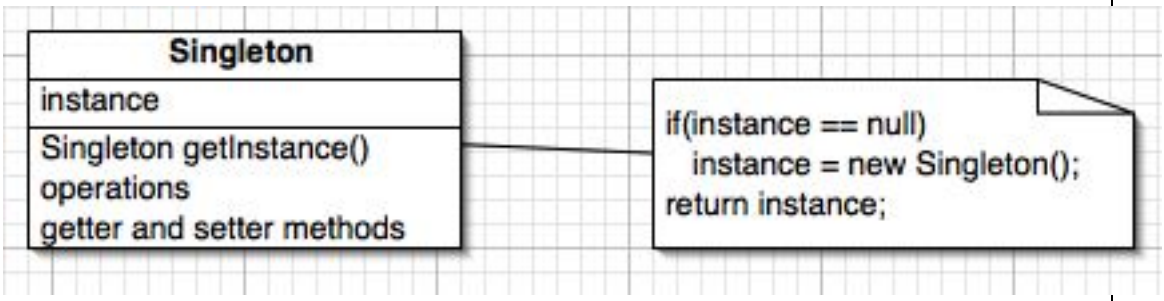
Pattern: Singleton

Intenção
– Garantir que uma classe tenha somente uma instância e fornecer um ponto global de acesso à mesma.

Motivação
– Em muitas situações é necessário garantir que algumas classes tenham uma e somente uma instância. Exemplo: o gerenciador de arquivos num sistema deve ser único.

Aplicabilidade
– Quando deva existir apenas uma instância de uma classe e essa instância deve dar acesso aos clientes através de um ponto bem conhecido.

Estrutura



Código Exemplo

```
public class ClassicSingleton {
        private static ClassicSingleton instance = null;
        private ClassicSingleton() {
        // Evita a instanciação por outra classe
        }
        public static ClassicSingleton getInstance() {
                if(instance == null) {
```

```
                        instance = new ClassicSingleton();
                }
        return instance;
        }
}
```

Colaborações
– Os clientes acessam uma única instância do Singleton pela operação getInstance()

Conseqüências
– Acesso controlado à instância única.

Padrões Correlatos
– AbstractFactory, Prototype, Builder

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class ConnectionFactory {
   //singleton da conexão
   private static final ThreadLocal<Connection> conn = new ThreadLocal<>();

   static {
      try {
         Class.forName("com.mysql.jdbc.Driver");
      } catch (ClassNotFoundException e) {
         throw new RuntimeException(e);
      }
   }

   // Obtém conexão com o banco de dados
   public static Connection obtemConexao() throws SQLException {
      if (conn.get() == null){
         conn.set(DriverManager

   .getConnection("jdbc:mysql://localhost/cerveja?user=alunos&password=alunos
"));
      }
      return conn.get();
   }
   //Fecha a conexão
   public static void fecharConexao() throws SQLException {
      if(conn.get() != null){
         conn.get().close();
         conn.set(null);
      }
```

```
    }
}
```