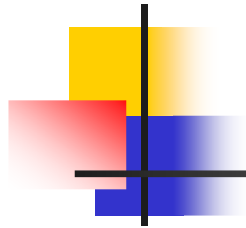


Java Enterprise Edition - JEE

13. Arquitetura MVC com Struts e Hibernate



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



Agenda

- Definição do estudo de caso;
- Criação do projeto;
- Importação das
- Criação dos beans;
- Configuração do struts;
- Configuração do filtro de sessões;
- Teste da aplicação;



Estudo de caso

- Atualização do projeto `hibernate`, criado na apresentação anterior;
- Criação das páginas `listar.jsp` e `index.html`;
- Criação da classe `ProdutoAction`;
- Atualização do `struts-config`;
- Criação da classe `SessionFilter`;
- Atualização do arquivo `web.xml`;
- Teste da aplicação



Exercício 01: listar.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head><title>Lista de produtos</title></head>
<body>
  <table border="1">
    <tr>
      <th>Nome</th><th>Descricao</th><th>Preco</th>
    </tr>
    <c:forEach var="produto" items="${listaDeProdutos}">
      <tr>
        <td>${produto.nome}</td>
        <td>${produto.descricao}</td>
        <td>${produto.preco}</td>
      </tr>
    </c:forEach>
  </table>
</body>
</html>
```



Trabalhando com Struts

- Até aqui, criamos uma classe filha de **Action** para cada ação do usuário
- Exemplo: **CadastrarProdutoAction**, **AlterarProdutoAction** etc;
- Cada classe possui apenas o método **execute()**;
- Mas, o que acontece em sistemas com grande número de classes bean?



Mudando a estratégia de controle

- E se, ao invés de criarmos uma classe de controle para cada ação, pudéssemos criar um controlador por caso de uso?
- O Struts oferece a classe **DispatchAction** que permite empacotar os métodos necessários a um determinado contexto
- Com isso, podemos criar uma classe de controle por caso de uso;



Considerações de DispatchAction

- É filha da classe Action;
- Requer o parâmetro **method**, que indica o método que deve ser executado;
- É configurada no **struts-config.xml** como uma Action comum;
- Seus métodos devem possuir os mesmos **tipo de retorno** e **lista de parâmetros** e lançar uma **Exception**;

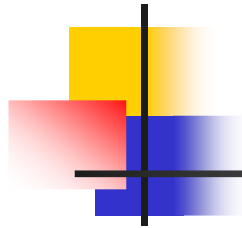
Exercício 02: ProdutoAction

```
public class ProdutoAction extends DispatchAction {  
    2 public ActionForward listar(ActionMapping mapping,  
                                3 ActionForm form,  
                                HttpServletRequest request,  
                                HttpServletResponse response)  
    4 throws Exception {  
        //Recupera uma conexão para acesso ao banco  
        Session session = (Session)request.getAttribute("session");  
        //Cria o objeto DAO  
        ProdutoDAO dao = new ProdutoDAO(session);  
        5 //Chamada ao metodo listar  
        List<Produto> lista = dao.listar();  
        //Coloca a colecao de produtos na requisicao  
        request.setAttribute("listaDeProdutos", lista);  
        //Retorna forwar para a tela de listagem  
        return mapping.findForward("telaListagem");  
    }  
}
```




Exercício 03: struts-config.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts-config PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 1.2//EN"
    "http://jakarta.apache.org/struts/dtds/struts-config_1_2.dtd">
<struts-config>
    <action-mappings>
        <action path="/listarProdutos" parameter="method" scope="request"
            type="br.fucapi.hibernate.control.action.ProdutoAction">
            <forward name="telaListagem" path="/listar.jsp"/>
        </action>
        <!-- Action que realiza apenas o redirecionamento -->
        <action path="/Welcome" forward="/welcomeStruts.jsp"/>
    </action-mappings>
</struts-config>
```



Acesso ao banco de dados

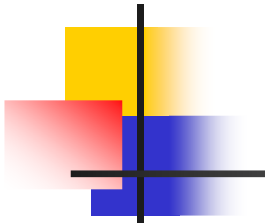
- Onde devemos abrir uma sessão?
- E como manter uma transação?
- Para executar 3 métodos em uma requisição, precisamos abrir e fechar 3 vezes a conexão?
- Esse tipo de arquitetura pode prejudicar o desempenho da aplicação;



Open Session in View

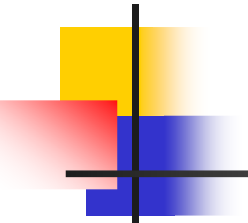
- Uma boa idéia é **interceptar** todas as requisições para o servidor e criar uma **transação** a cada nova requisição;
- Após a execução do fluxo, devemos realizar o **commit** da transação e fechar o objeto session;
- Para esse trabalho, utilizaremos objeto Filter, que ficará responsável por abrir e fechar as sessões de acesso ao BD;

Exercício 04: SessionFilter



```
public class SessionFilter implements Filter {  
    public void doFilter(ServletRequest request,  
        ServletResponse response, FilterChain chain)  
        throws IOException, ServletException {  
        1 //Cria a sessao  
        Session session = HibernateUtil.getSession();  
        try {  
            2 //Inicia a transacao  
            session.getTransaction().begin();  
  
            3 //Inclui a sessao na requisicao  
            request.setAttribute("session", session);  
            //Executa a requisicao  
            chain.doFilter(request, response);  
  
            4 //Fim da transacao  
            session.getTransaction().commit();  
        } catch (Exception e) {  
            5 // Cancelamento da transacao  
            session.getTransaction().rollback();  
            e.printStackTrace();  
        } finally{  
            6 //Fecha a sessao  
            session.close();  
        }  
    }  
}
```

Exercício 05: web.xml



```
<servlet><!-- Mapeamento do Struts -->
  <servlet-name>controller</servlet-name>
  <servlet-class>
    org.apache.struts.action.ActionServlet
  </servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>controller</servlet-name>
  <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

```
<filter><!-- Mapeamento do Filtro -->
  <display-name>SessionFilter</display-name>
  <filter-name>SessionFilter</filter-name>
  <filter-class>
    br.fucapi.hibernate.control.filter.SessionFilter
  </filter-class>
</filter>
<filter-mapping>
  <filter-name>SessionFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```



Exercício 06: index.html

```
<html>
<head><title>Tela inicial</title></head>
<body>
  <h1>Teste hibernate</h1>
  <a href="listarProdutos.do?method=listar">
    Listar produtos
  </a>
</body>
</html>
```

Teste da aplicação

- Página inicial:



- Listagem:

A screenshot of a web browser window titled 'Lista de produtos'. The main content area displays a table with three columns: 'Nome', 'Descricao', and 'Preco'. The table contains two rows of data. The first row shows 'Camisa regata' in the 'Nome' column, 'Camisa regata G' in the 'Descricao' column, and '100.5' in the 'Preco' column. The second row shows 'Bermuda masculina' in the 'Nome' column, 'Bermuda M' in the 'Descricao' column, and '50.0' in the 'Preco' column. The browser window has a standard title bar with a small icon on the left and a maximize button on the right.

Nome	Descricao	Preco
Camisa regata	Camisa regata G	100.5
Bermuda masculina	Bermuda M	50.0



Referências

- Hall, Marty, “Core Servlets and Java Server Pages”, Janeiro 2002, Sun Microsystems Press;
- <http://java.sun.com/>
- <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

13. Arquitetura MVC com Struts e Hibernate



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