

Desenvolvimento Web I

Aula 06 - MVC, Servlets e JavaServer Pages

Apresentação

Olá, novamente, meu caro aluno! Vamos recapitular o que estudamos juntos até agora. No início desta disciplina, você utilizou Servlets no desenvolvimento de pequenos programas Web. Em seguida, viu que o código HTML, embutido dentro dos Servlets, dificultava o entendimento e a manutenção desses arquivos. Para minimizar esse problema, você estudou o uso de JSP, um arquivo que parece um HTML, mas que possui marcadores especiais. Um deles, inclusive, permite escrever código Java no JSP. Pois bem, já lhe ocorreu que agora com o JSP corremos o risco de ter o problema inverso, o de ter arquivos JSP sobrecarregados com comandos Java?

Dentro desse novo contexto, a aula de hoje apresenta uma forma de minimizar o problema de se ter muitos comandos Java dentro de arquivos JSP, facilitando a construção e evolução desses arquivos.

Tenha uma boa leitura!



Vídeo 01 - Apresentação

Objetivos

- Aplicar o conceito de arquitetura MVC (Model/View/Controler), utilizando Servlets e JSP.

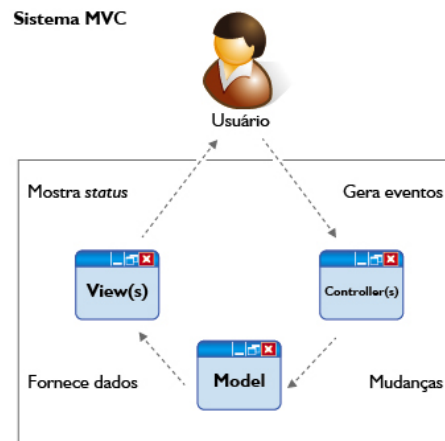
Separação em camadas

Assim como os Servlets não devem ter código HTML embutido, os arquivos JSP não devem ter código Java - pelo menos não em excesso. Então, como fazer com que estruturamos nosso código de forma a não sobrecarregar nem os Servlets com código HTML, nem os arquivos JSP com código Java? Existem várias formas de se fazer isso e você irá estudar aqui uma das mais simples, porém eficiente e bastante utilizada na prática, chamada de arquitetura MVC.

A arquitetura de um sistema de software indica, entre outras coisas, a forma em que o software é dividido em termos de componentes, suas funções, relacionamentos, etc. No caso, a arquitetura MVC (Model – View – Controller) é uma arquitetura que estrutura a separação do código de um software nos seguintes componentes/camadas:

- **Controlador:** componente de software responsável por receber os comandos do usuário e acionar as ações corretas a serem realizadas no sistema, incluindo a seleção de qual componente de apresentação da resposta deverá utilizado.
- **Modelo:** representa as regras de negócio. Fazem parte dessa camada de software as classes responsáveis por realizar transações de negócio (venda de produtos, cadastro de informações, etc.). Essa parte do código é responsável também por se comunicar com o banco de dados ou com outros sistemas existentes.
- **Visão:** componente de software responsável pela apresentação da resposta, ou seja, pela montagem das telas de resposta do sistema. Ele é ativado pelo controlador e recebe informações da camada de modelo.

Figura 01 - Arquitetura MVC



Fonte: <<http://douglasmiranda.wordpress.com/2009/03/25/includes-frameworks-php-utilizando-mvc/>>. Acesso em: 10 jul. 2012.

Vejamos como isso funciona na prática. Um usuário, ao acessar uma URL, aciona um controlador (Servlet) executando no servidor Web, o qual é responsável por realizar a mudança no modelo de negócio (classes Java de negócio) e redirecionar a montagem da resposta para um componente de apresentação apropriado (arquivo JSP). Vamos entender isso através de um exemplo simples. Considere que você é uma pessoa interessada em ter um sistema para registrar todos os encontros que você tem com seus colegas. Vamos então desenvolver um sistema para registrar esses encontros.

Esse sistema é bem simples e será construído ao longo das seções seguintes, de acordo com o padrão arquitetural MVC e utilizando as tecnologias Servlet e JSP. No entanto, lembre-se: as ideias trazidas pelo MVC são independentes de tecnologia. Essa arquitetura pode ser utilizado em qualquer linguagem de programação (Java, C, Delphi, etc.) e com qualquer tipo de sistema (Web, desktop, mobile, etc.).

Elemento Modelo

A modelagem do sistema, geralmente, começa com a modelagem das classes que representam o negócio do sistema. No nosso exemplo, o negócio basicamente é o registro de encontros que você (ou outro usuário do sistema) realizar. Dessa forma, é necessário criar uma classe que represente as informações de cada um dos seus encontros. Segue na Listagem 1 uma possível implementação para essa classe:

```
1 package aula06;
2
3 import java.util.Date;
4
5 public class Encontro {
6     private String nomePessoa;
7     private String local;
8     private Date data;
9     private String motivo;
10
11     public String getNomePessoa() {
12         return nomePessoa;
13     }
14
15     public void setNomePessoa(String nomePessoa) {
16         this.nomePessoa = nomePessoa;
17     }
18
19     public String getLocal() {
20         return local;
21     }
22
23     public void setLocal(String local) {
24         this.local = local;
25     }
26
27     public Date getData() {
28         return data;
29     }
30
31     public void setData(Date data) {
32         this.data = data;
33     }
34
35     public String getMotivo() {
36         return motivo;
37     }
38
39     public void setMotivo(String motivo) {
40         this.motivo = motivo;
41     }
42 }
```

Listagem 1 - Classe para representar os dados de um encontro

Note os atributos dessa classe: nome da pessoa com quem você se encontrou; local do encontro; data e motivo do encontro. Além disso, precisamos de uma classe para realizar o cadastro e a consulta dos encontros registrados. Isso será feito pela classe SistemaEncontros, mostrada na Listagem 2.

```
1 package aula06;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class SistemaEncontros {
7     private List <Encontro> encontros = new ArrayList<Encontro>();
8
9     public void adicionar(Encontro e) {
10         encontros.add(e);
11     }
12
13     public List<Encontro> listarEncontros() {
14         return encontros;
15     }
16 }
```

Listagem 2 - Classe para representar o sistema de registro de encontros

Note que a implementação mostrada para o sistema de encontros utiliza uma lista (array) para armazenar em memória os registros de encontros realizados. Caso você não esteja familiarizado com as classes `java.util.List` e `java.util.ArrayList`, estude na API Java <<http://docs.oracle.com/javase/tutorial/collections/index.html>> o uso e as funções disponíveis nessas classes.

Atividade 01

1. Implemente as classes mostradas para a camada de modelo (negócio) do sistema de encontro.
2. Adicione ao sistema a função de remover um encontro previamente cadastrado.
3. Adicione ao sistema a função de consultar os encontros realizados com uma dada pessoa, ou seja, dado o nome de uma pessoa, o sistema deverá retornar a lista (um array) de encontros que você realizou com ela.

Elemento Controle

Para o elemento controle, temos em essência a tecnologia de Servlets. Em primeiro lugar, vamos utilizar uma classe para instanciar e guardar uma referência para o sistema de encontros, permitindo assim que qualquer Servlet a ser criado possa acessar o sistema. Isso pode ser feito através da classe mostrada na Listagem 3:

```
1 package aula06;
2
3 import javax.servlet.ServletContext;
4 import javax.servlet.ServletContextEvent;
5 import javax.servlet.ServletContextListener;
6 import javax.servlet.annotation.WebListener;
7
8 @WebListener
9 public class ContextListener implements ServletContextListener {
10     public static final String SISTEMA_ENCONTROS = "sistemaEncontros";
11     public void contextInitialized(ServletContextEvent event) {
12         ServletContext context = event.getServletContext();
13         context.setAttribute("SISTEMA_ENCONTROS", new SistemaEncontros());
14     }
15
16     public void contextDestroyed(ServletContextEvent event) {
17         ServletContext context = event.getServletContext();
18         context.removeAttribute("SISTEMA_ENCONTROS");
19     }
20 }
21
```

Listagem 3 - Classe que inicializa e que cria uma referência para o sistema de registro de encontros

Essa classe herda o código (uma subclasse) de uma classe especial chamada `ServletContextListener` e implementa métodos que vão ser invocados basicamente quando o contêiner dos Servlets for criado (`contextInitialized`) e para quando ele for destruído (`contextDestroyed`). Nesses dois momentos, temos a oportunidade de, respectivamente, adicionar e remover uma referência para o sistema dentro do chamado contexto dos Servlets. Note nas linhas 9 e 14 que ambos os métodos recebem como parâmetro um objeto do tipo `ServletContextEvent`. Esse objeto possui o método `getServletContext()`, o qual retorna um objeto `ServletContext`, e seu uso pode ser visto nas linhas 10 e 15. De posse desse objeto, você pode criar e

acessar atributos, similarmente ao que acontece com os objetos que representam sessões de usuário. No caso, utilizam-se os métodos `setAttribute()` e `removeAttribute()` para adicionar e remover atributos do `ServletContext`. Na linha 11, temos a criação de um atributo de nome “sistemaEncontros”, cujo valor é a referência ao sistema de encontros. Já na linha 16, temos a remoção desse atributo.

Elemento controle II

Para que o `ContextListener` que criamos seja executado automaticamente foi adicionado a anotação `@WebListener` antes da classe, como pode ser visto na Listagem 3. Uma outra forma de fazer isso é adicionar o seguinte trecho de código no arquivo `web.xml` da aplicação (não é necessário se você criou a classe com a anotação `@WebListener`):

```
1 <listener>
2   <listener-class>aula06.ContextListener</listener-class>
3 </listener>
```

Uma vez que garantimos a inicialização e destruição da referência ao sistema nos momentos corretos, nosso próximo passo é a implementação de um Servlet controlador, como mostrado no código da Listagem 4.


```

1 package aula06;
2
3 import java.io.IOException;
4 import java.util.Date;
5 import javax.servlet.ServletException;
6 import javax.servlet.annotation.WebServlet;
7 import javax.servlet.http.HttpServlet;
8 import javax.servlet.http.HttpServletRequest;
9 import javax.servlet.http.HttpServletResponse;
10
11 @WebServlet("/ServletControlador")
12 public class ServletControlador extends HttpServlet {
13
14     private static final long serialVersionUID = 1L;
15
16     protected void doGet(HttpServletRequest request,
17         HttpServletResponse response)
18         throws ServletException, IOException {
19         doPost(request, response);
20     }
21
22     protected void doPost(HttpServletRequest request,
23         HttpServletResponse response)
24         throws ServletException, IOException {
25         String acao = request.getParameter("acao");
26         SistemaEncontros sistema = (SistemaEncontros) getServletContext().getAttribute("SISTEMA_E
27         String resposta = "index.jsp";
28         if ("cadastrar".equals(acao)) {
29             resposta = "cadastrar.jsp";
30         } else if ("confirmarCadastro".equals(acao)) {
31             Encontro e = new Encontro();
32             e.setNomePessoa(request.getParameter("nomePessoa"));
33             e.setMotivo(request.getParameter("motivo"));
34             e.setLocal(request.getParameter("local"));
35             e.setData(new Date());
36             sistema.adicionar(e);
37             request.setAttribute("lista", sistema.listarEncontros());
38             resposta = "listar.jsp";
39         } else if ("listar".equals(acao)) {
40             request.setAttribute("lista", sistema.listarEncontros());
41             resposta = "listar.jsp";
42         }
43         request.getRequestDispatcher(resposta).forward(request, response);
44     }
45 }

```

Listagem 4 - Servlet exercendo papel de controlador, dentro da arquitetura MVC

A classe ServletControlador atende tanto requisições através do método GET, como através do método POST. Na implementação do método doPost(), fazemos na linha 21 o acesso ao parâmetro que indica a ação a ser executada

(request.getParameter("acao")), e, nas linhas 22-23, o acesso ao sistema através do contexto do Servlet (getServletContext().getAttribute("sistemaEncontros")). Baseado nesses dados, é trabalho do controlador verificar qual a ação requisitada e atuar de acordo com ela. No caso, temos uma sequência de comandos if/else if (linhas 25 a 39) que verificam quatro possíveis casos, cada um representando uma ação diferente:

- **Valor do parâmetro “acao” é igual a “cadastrar”**

- Deve-se montar a tela de cadastro de encontros. Para isso, basta redirecionar a requisição para o arquivo cadastro.jsp. Esse arquivo será apresentado na próxima seção dessa aula. O comando request.getRequestDispatcher(resposta).forward(request, response) é responsável por fazer com que a página de resposta seja montada pela página JSP, indicada na variável resposta.

- **Valor do parâmetro “acao” é igual a “confirmarCadastro”**

- Uma vez preenchido o formulário de cadastro, o mesmo será submetido para o servlet controlador, o qual construirá o objeto encontro (linha 28), pegará as informações do encontro através das chamadas ao método request.getParameter() (linhas 29 a 31) e adicionará no sistema o objeto criado (linha 33).
- Em seguida, a lista de encontros registrados é colocada como atributo da requisição request.setAttribute() (linha 34). Essa informação, como veremos, será acessada pelo arquivo JSP a ser executado, indicado na linha 35 (listar.jsp).

- **Valor do parâmetro “acao” é igual a “listar”**

- Nesse caso, o arquivo JSP a ser executado é o listar.jsp, e a lista contendo os encontros registrados é passada como atributo da requisição (linhas 37).

- **Valor do parâmetro “acao” é diferente dos casos citados**

- A página index.jsp será apresentada (valor inicial da variável resposta definido na linha 24).

Lembrando que todo Servlet tem que estar referenciado no arquivo web.xml, caso não esteja se utilizando anotações, como visto na aula sobre Servlets. Como a Listagem 4 utiliza anotações essa referência não é necessária, mas caso não seja o caso seria necessário adicionar o seguinte código nesse arquivo:

```
1 <servlet>
2   <description></description>
3   <display-name>ServletControlador</display-name>
4   <servlet-name>ServletControlador</servlet-name>
5   <servlet-class>aula06.ServletControlador</servlet-class>
6 </servlet>
7 <servlet-mapping>
8   <servlet-name>ServletControlador</servlet-name>
9   <url-pattern>/ServletControlador</url-pattern>
10</servlet-mapping>
```

Atividade 02

1. Implemente e configure a aplicação de acordo com o mostrado para a camada de controle.
Para as atividades a seguir, ainda não é necessário escrever o código JSP relativo às operações propostas.
2. Altere o Servlet controlador para dar suporte à operação de remover encontro.
3. Altere o Servlet controlador para dar suporte à operação de alterar local de encontro.
4. Altere o Servlet controlador para dar suporte à operação de consultar encontro, dado o nome da pessoa a se encontrar.

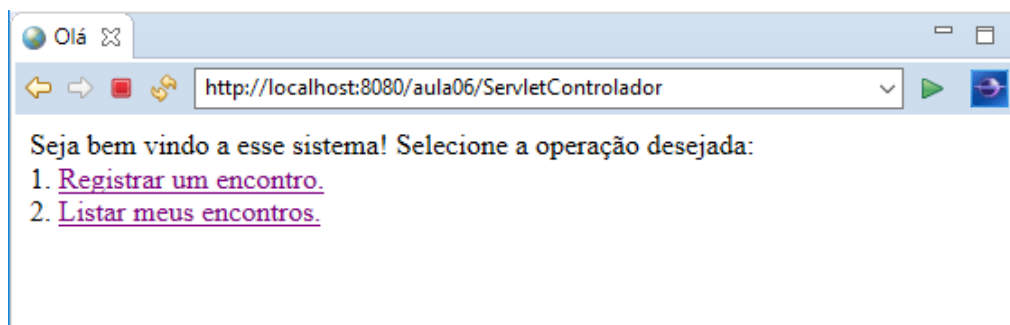
Elemento Visão

Para o elemento visão da arquitetura MVC, temos os arquivos JSP. O primeiro deles é o index.jsp, cujo código é mostrado abaixo e cuja tela é vista na Figura 2. Note os links do menu (linhas 8 e 9), utilizando o ServletControlador e o parâmetro acao para indicar a ação a ser realizada.

```
1 <html>
2   <head>
3     <title>Olá</title>
4   </head>
5   <body>
6     Seja bem vindo a esse sistema! Selecione a operação desejada:
7     <BR>
8     1. <a href="ServletControlador?acao=cadastrar">Registrar um
9       encontro.</a><BR>
10    2. <a href="ServletControlador?acao=listar">Listar meus encontros.</a><BR>
11  </body>
12 </html>
```

Listagem 5 - Código do arquivo index.jsp

Figura 02 - Tela gerada pelo arquivo index.jsp



A tela de registro de encontros é gerada pelo arquivo cadastrar.jsp, cujo código-fonte é mostrado a seguir. Note que ele é, na verdade, um arquivo puramente HTML. A tela gerada para esse arquivo é mostrada na Figura 3.

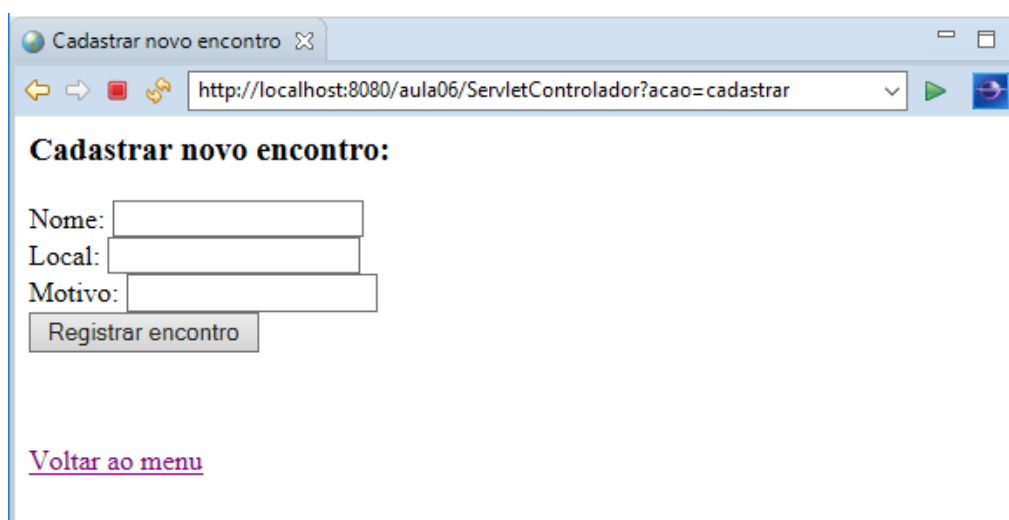
```

1 <html>
2   <head>
3     <title>Cadastrar novo encontro</title>
4   </head>
5   <body>
6     <h3>Cadastrar novo encontro:</h3>
7     <form action="ServletControlador" method="post">
8       <input type="hidden" name="acao" value="confirmarCadastro"/>
9       Nome: <input type="text" name="nomePessoa" value=""/><br>
10      Local: <input type="text" name="local" value=""/><br>
11      Motivo: <input type="text" name="motivo" value=""/><br>
12      <input type="submit" value="Registrar encontro"/><br>
13    </form><BR><BR>
14    <a href="ServletControlador">Voltar ao menu</a>
15  </body>
16 </html>

```

Listagem 6 - Código do arquivo cadastrar.jsp.

Figura 03 - Tela gerada pelo arquivo cadastrar.jsp



The screenshot shows a web browser window with the title 'Cadastrar novo encontro'. The address bar shows the URL 'http://localhost:8080/aula06/ServletControlador?acao=cadastrar'. The main content area displays the form titled 'Cadastrar novo encontro:'. The form contains three text input fields labeled 'Nome:', 'Local:', and 'Motivo:'. Below these fields is a button labeled 'Registrar encontro'. At the bottom of the form, there is a link labeled 'Voltar ao menu'.

Elemento Visão II

Por fim, temos, a seguir, o código-fonte do arquivo listar.jsp, responsável por mostrar a lista de encontros registrados. Note o uso das diretivas de import logo no início do arquivo. Também temos o uso de um scriptlet que envolve o comando for para percorrer a lista de encontros armazenada como atributo da requisição e identificada pelo nome "lista" (veja que esse foi o nome utilizado no código do Servlet controlador). Nesse caso, o comando for vai imprimir na saída o conteúdo HTML que está entre suas chaves { e } (linhas 18 a 27). Esse conteúdo é, basicamente, a informação de cada encontro cadastrado.

```

1 <%@ page import="java.util.List" %>
2 <%@ page import="aula06.Encontro" %>
3 <html>
4   <head>
5     <title>Lista de encontros</title>
6   </head>
7   <body>
8     <h3>Lista de encontros:</h3>
9     <table border="1">
10      <tr>
11        <td>Nome</td>
12        <td>Local</td>
13        <td>Motivo</td>
14        <td>Data</td>
15      </tr>
16      <%
17      List<Encontro> lista = (List<Encontro>) request.getAttribute("lista");
18      for (Encontro encontro : lista) {
19        <%
20          <tr>
21            <td><%= encontro.getNomePessoa() %></td>
22            <td><%= encontro.getLocal() %></td>
23            <td><%= encontro.getMotivo() %></td>
24            <td><%= encontro.getData().toLocaleString() %></td>
25          </tr>
26          <%
27        }
28        <%
29      </table>
30      <BR><BR>
31      <a href="ServletControlador">Voltar ao menu</a>
32    </body>
33  </html>

```

Listagem 7 - Código do arquivo listar.jsp

Atividade 03

1. Implemente a aplicação de acordo com o mostrado para a camada de apresentação. Finalmente, execute o sistema e verifique se o seu comportamento está correto.
2. Adicione o arquivo JSP necessário para dar suporte à operação de remover encontro.
3. Adicione o arquivo JSP necessário para dar suporte à operação de alterar local de encontro.

4. Adicione o arquivo JSP necessário para dar suporte à operação de consultar encontro a partir do nome da pessoa a se encontrar.

Conclusão

Chegamos ao final da nossa Aula 6, mas, nas próximas aulas, você voltará a estudar recursos de JSP que ajudam no desenvolvimento para Web. Serão apresentados recursos que, inclusive, aumentam a produtividade de desenvolvimento. Até lá.

Leitura complementar

Realize uma busca na Internet pela arquitetura MVC e faça uma leitura das primeiras páginas encontradas sobre o assunto. Faça também a leitura sobre os marcadores JSP aprendidos hoje através da seguinte URL:

- <<http://docs.oracle.com/javaee/5/tutorial/doc/bnaln.html>>

Resumo

Nesta aula, você estudou o padrão arquitetural chamado MVC, responsável por separar a lógica de negócio do controle do fluxo de execução e do modelo de negócio (regras de negócio). Essa arquitetura é muito importante para que o código de sua aplicação como um todo fique bem organizado. Na verdade, existem diversas bibliotecas de desenvolvimento para Web que seguem essa arquitetura. Você viu que essa arquitetura se divide em três camadas: de controle, modelo e visão. Por fim, você estudou as camadas de modelo, controle e de apresentação mais detalhadamente, observando que a primeira representa a lógica do negócio, a segunda representa o fluxo de controle da aplicação e a última é responsável pelo código de apresentação do sistema (telas de entrada e saída).

Autoavaliação

Descreva como é possível aplicar o conceito de arquitetura MVC (Model/View/Controller) utilizando Servlets e JSP.

Referências

AHMED, K. Z.; UMRYSH, C. E. **Desenvolvendo aplicações comerciais em Java com J2EE e UML**. Rio de Janeiro: Ciência Moderna, 2003. 324 p.

BASHAM, Bryan; SIERRA, Kathy; BATES, Bert. **Head First Servlets and JSP: passing the Sun Certified Web Component Developer Exam (SCWCD)**. 2nd ed. O'Reilly Media, 2008.

CATTELL, Rick; INSCORE, Jim. **J2EE: criando aplicações comerciais**. Rio de Janeiro: Editora Campus, 2001.

HALL, Marty. **More Servlets and JavaServer Pages (JSP)**. New Jersey: Prentice Hall PTR (InformIT), 2001. 752 p. Disponível em: <<http://pdf.moreservlets.com/>>. Acesso em: 11 maio 2012.

HALL, Marty; BROWN, Larry. **Core Servlets and JavaServer Pages (JSP)**. 2nd ed. New Jersey: Prentice Hall PTR (InformIT), 2003. 736p. (Core Technologies, 1). Disponível em: <<http://pdf.coreservlets.com/>>. Acesso em: 11 maio 2012.

HYPERTEXT Transfer Protocol: HTTP/1.1 – Methods Definition. Disponível em: <<http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html>>. Acesso em: 11 maio 2012.

J2EE Tutorial. Disponível em: <<http://java.sun.com/javaee/reference/tutorials/>>. Acesso em: 11 maio 2012.