01

Jasper Reports dentro de uma aplicação web

Transcrição

Para esta aula é preciso ter instalado **Eclipse na versão Java EE**. Você pode baixar o Eclipse no link: http://www.eclipse.org/downloads (http://www.eclipse.org/downloads/). Além disso, usaremos o **Apache Tomcat 7** que se encontra no http://tomcat.apache.org (http://tomcat.apache.org/download-70.cgi). Também temos um **projeto preconfigurado** com todas as bibliotecas necessárias disponível: aqui (http://s3.amazonaws.com/caelum-online-public/FJ-24/cap6-movimentacoes-web.zip)

Caso você tenha dúvidas sobre a instalação do Tomcat verifique a apostila aberta do treinamento FJ-21 (http://www.caelum.com.br/apostila-java-web/o-que-e-java-ee/#3-10-configurando-o-tomcat-no-wtp).

Refatoração para simplificar a geração do PDF

Na última aula vimos como um relatório pode ser gerado programaticamente a partir de sua definição: o arquivo `jrxml`. Também preenchemos e exportamos o relatório utilizando código Java.

Foi necessário definir a entrada, o mapa de parâmetros e estabelecer uma conexão com o banco de dados, tudo como parâmetro do método `fillReport(..)` da classe `JasperFillManager`, que lê e preenche o relatório.

O relatório preenchido é um objeto do tipo `JasperPrint`, que precisa ser processado por um `JRExporter`. Escolhemos `JRPdfExporter` que transforma o objeto `JasperPrint` em PDF.

Nesta aula já temos preparada a classe `GeradorRelatorio` que encapsula os detalhes de geração de relatório. Essa classe recebe no construtor a entrada, ou seja, o nome do arquivo `jasper`, os parâmetros e a conexão. Além disso, possui um método público `geraPDFParaOutputStream(..)` que encapsula classes como `JasperFillManager` ou `JRPdfExporter`.

O uso do `GeradorRelatorio` simplifica o trabalho. Uma vez estabelecida a entrada podemos instanciar a classe e passar os parâmetros via construtor. Depois chamamos o método `geraPDFParaOutputStream` passando um `FileOutputStream`.

Introdução a aplicação Web

Neste vídeo mostraremos como gerar o relatório com uma aplicação web. A aplicação chama-se `gastos-web` e já possui classes como `ConnectionFactory` e `GerenciadorRelatorio`. Além disso, temos dentro da pasta `WEB-INF/lib` todas as bibliotecas necessárias que vimos até agora.

O *servlet-container* Tomcat, na versão 7, já esta configurado no Eclipse. Para baixar o Tomcat acesse: `http://tomcat.apache.org (http://tomcat.apache.org)`

Criação da Servlet para gerar o PDF

Nossa tarefa agora é criar uma página com um formulário para enviar uma requisição e uma servlet que gera o PDF como resposta. Começaremos com a servlet. Selecionaremos a pasta `src` do projeto usando o wizard para criar uma nova classe servlet dentro no pacote `br.com.caelum.relatorio.servlet`.

A servlet terá o nome `RelatorioServlet` e estenderá a classe `HttpServlet`. Também alteraremos o mapeamento padrão, usaremos a URL `/gastos`.

O único método que queremos gerar na servlet é o método `doPost(..)`.

Segue o esboço da servlet:

```
@WebServlet("/gastos")
public class RelatorioServlet extends HttpServlet {

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws Serv

    }
}
```

A nossa servlet está preparada, mas antes de usar o `GeradorRelatorio` é preciso copiar o arquivo `jasper` do projeto anterior. Para isso criaremos uma pasta `jasper` dentro da pasta `WEB-INF`. Nesta pasta colocaremos os dois relatórios compilados, o principal e o sub-report.

Voltando para a servlet, também copiamos o código que representa a entrada do relatório, ou seja, nome do relatório, parâmetros e a conexão. Já fecharemos a conexão. A chamada do método `close()` exige um `try-catch`, pois estamos dentro de uma servlet. Colocaremos todo o código relacionado com a geração do relatório nesse bloco `try-catch`. Dentro do bloco `catch` lançaremos uma `ServletException`.

Finalmente podemos usar o `GeradorRelatorio` chamando o construtor e passando os parâmetros. Depois usaremos o método `geraPDFParaOutputStream(..)` que recebe o fluxo de saída.

Como estamos em um aplicação web, o fluxo de saída é ligado à resposta. Através dela pegamos o `OutputStream`, um `ServletOutputStream`:

```
response.getOutputStream()
```

O problema é que o arquivo `jasper` está dentro da pasta `/WEB-INF/jasper`. Falta indicar o caminho até esta pasta. Isso é feito através do `ServletRequest` que possui o método `getServletContext().getRealPath(..)` que recebe o caminho que desejamos acessar. Passando o caminho a partir da raiz da aplicação web: `/WEB-INF/jasper/gasto_por_mes.jasper`.

Segue a Servlet completa:

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

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws Serv

        try {
            String nome = request.getServletContext().getRealPath("/WEB-INF/jasper/gasto_por_me
            Connection connection = new ConnectionFactory().getConnection();
            Map<String, Object> parametros = new HashMap<String, Object>();

            GeradorRelatorio gerador = new GeradorRelatorio(nome, parametros, connection);
            gerador.geraPDFParaOutputStream(response.getOutputStream());


            connection.close();
        } catch (SQLException e) {
            throw new ServletException(e);
```

```
        }
    }
}
```

Definição do formulário JSP

Como a nossa servlet usa o método `doPost(..)` é preciso criar um formulário HTML, que ficará dentro da pasta `WebContent` com o nome `relatorio.jsp` .

Dentro da tag body criaremos o formulário que já preparamos antes. Repare o atributo `action` que chama a URL da Servlet através do método POST. Também temos um botão para submeter o formulário.

Segue o formulário HTML:

```
<html>
  <body>

   <form action="gastos" method="POST">
       <input type="submit" value="Gera PDF" />
   </form>

  </body>
</html>
```

Ao subir o Tomcat e ao chamar nossa aplicação no endereço `http://localhost:8080/gastos-web/relatorio.jsp (http://localhost:8080/gastos-web/relatorio.jsp)` aparece o formulário com o botão. Submetendo o formulário recebemos o PDF gerado pela Servlet.

Podemos ver na segunda página do relatório que as movimentações foram geradas a partir de Junho até o mês de Agosto. Aparecem apenas 3 meses de movimentações. O problema é que em nosso relatório definimos valores padrões para as datas. Para esta aplicação queremos que o usuário final defina pelo formulário a data inicial e final do relatório.

Capturando as datas do relatório pelo formulário

Vamos voltar para o Eclipse e alterar a página JSP. Adicionaremos dois inputs. Novamente já temos o código preparado. O primeiro `input` define a data inicial e o segundo a data final. Repare que o nome dos inputs: `data_ini` e `data_fim` :

```
<html>
  <body>

    <form action="gastos" method="POST">

      Data inicial: <input type="text"  name="data_ini"/>  <br />

      Data final: <input type="text"  name="data_fim"/> <br />

      <input type="submit" value="Gera PDF" />

    </form>
</body>
</html>
```

Passando parâmetros da requisição para o relatório

Em nossa servlet leremos esses parâmetros. Voltando para Eclipse e abrindo a classe `RelatorioServlet` usaremos o método `getParameter(..)` do objeto request para receber o parâmetro `data_ini` , a mesma coisa para o parâmetro `data_fim` , copiando a linha anterior e renomeando as variáveis.

```
String dataIni = request.getParameter("data_ini");
String dataFim = request.getParameter("data_fim");
```

Os parâmetros são ainda do tipo `String` , porém o nosso relatório espera receber as datas como `java.util.Date` . É preciso parsear os dois parâmetros usando a classe `SimpleDateFormat` . Essa classe recebe um pattern no construtor que define o formato da data, em nosso caso `dd/MM/yyyy` . Com o `SimpleDateFormat` podemos usar o método `parse(dataString)` para criar um objeto `Date` a partir da `String` .

```
SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
Date dataInicial = sdf.parse(dataIni);
Date dataFinal = sdf.parse(dataFim);
```

Ao chamar o método `parse(..)` o compilador exige um tratamento de exceção. Vamos adicionar mais um bloco `catch` para capturar `ParseException` .

Por último é preciso adicionar os dois objetos `Date` no mapa que foi passado para a classe `GeradorRelatorio` . Chamaremos o método `put( .. , .. )` do mapa que recebe dois argumentos, uma chave e um valor. O valor é o objeto data que criamos pelo `SimpleDateFormat` e a chave definda na fase de design do relatório. As chaves se chamam `DATA_INI` para a data inicial e `DATA_FIM` para a data final:

```
Map<String, Object> parametros = new HashMap<String, Object>();
parametros.put("DATA_INI", dataInicial);
parametros.put("DATA_FIM", dataFinal);
GeradorRelatorio gerador = new GeradorRelatorio(nome, parametros, connection);
```

Segue a servlet completo:

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

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws Serv

        try {
            String nome = request.getServletContext().getRealPath("/WEB-INF/jasper/gasto_por_me
            Connection connection = new ConnectionFactory().getConnection();

            String dataIni = request.getParameter("data_ini");
            String dataFim = request.getParameter("data_fim");

            SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");

            Date dataInicial = sdf.parse(dataIni);
            Date dataFinal = sdf.parse(dataFim);
```

```
            Map<String, Object> parametros = new HashMap<String, Object>();
            parametros.put("DATA_INI", dataInicial);
            parametros.put("DATA_FIM", dataFinal);

            GeradorRelatorio gerador = new GeradorRelatorio(nome, parametros, connection);
            gerador.geraPDFParaOutputStream(response.getOutputStream());

            connection.close();
        } catch (SQLException e) {
            throw new ServletException(e);
        } catch (ParseException e) {
            throw new ServletException(e);
        }
    }
}
```

Com o formulário definido e a servlet preparados podemos testar a aplicação. Vamos reiniciar o Tomcat e chamar no navegador a aplicação. No formulário aparecem os dois inputs onde inserimos as datas, por exemplo `01/01/2012` e `31/12/2012` . Gerando o PDF o relatório aparece no navegador com a primeira movimentação em Janeiro seguida pelos outros meses até Dezembro.