01

Gerando PDF e outras saídas na aplicação Java

Transcrição

A versão utilizada do JasperReports Library se encontra nesse link (http://s3.amazonaws.com/caelum-online-public/FJ-24/jasperreports-4.7.0-project.zip). Também preparamos um **projeto com todas as bibliotecas necessárias** que vem preconfigurado para Eclipse. O projeto pode ser baixado pelo link: cap5-movimentacoes.zip (https://s3.amazonaws.com/caelum-online-public/FJ-24/cap5-movimentacoes.zip)

A versão mais atual do JasperReports Library se encontra aqui (http://community.jaspersoft.com/download). Caso queira baixar o driver do MySQL (Connector J) separado segue também o link (http://dev.mysql.com/downloads/connector/j/).

Introdução ao Report Engine JasperReports

Vimos como criar e exportar relatórios usando o designer *iReport*. Para esta aula focaremos na utilização do nosso relatório dentro de uma aplicação Java.

Utilizamos o *iReport* para definir os componentes do relatório. A definição foi salva dentro de um arquivo XML com a extensão `jrxml`. Os dados vieram do banco MySQL e no final tudo foi exportado em PDF para podemos visualizar.

Com a definição do relatório pronta podemos usar diretamente o *Report Engine*, o *JasperReport*. O iReport nada mais é do que o designer que define o `jrxml`, mas é o JasperReports o responsável pela geração e exportação de relatórios.

Antes de exportar com o *Report Engine* é preciso realizar um passo intermediário, que consiste na compilação do `jrxml` para um arquivo com a extensão `jasper`. O iReport sempre compilava e gerava o arquivo durante as exportações. O relatório compilado será passado para o *Report Engine* com os parâmetros da aplicação e a conexão que define a fonte de dados. Com todos esses informações, poderemos gerar o relatório e exportá-lo, por exemplo, no formato PDF.

Preparação do projeto

Vamos gerar o relatório programaticamente usando o Eclipse como IDE. Já temos um projeto preparado chamado gastos. A pasta `lib` é para as nossas bibliotecas e possui o driver do MySQL, que já foi definido em nosso `classpath`. Na pasta `src` já temos a classe `ConnectionFactory` para obter conexões com o banco MySQL (`jdbc:mysql://localhost/financas`).

Além disso, temos uma classe de testes que abre uma conexão através da *Report Engine*. Vamos chamar uma vez o método `main`. Executou sem problemas.

O próximo passo é copiar as definições dos relatórios, ou seja os arquivos `jrxml`. Vamos copiar os dois arquivos (o relatório principal e o sub-relatório) para a raiz do projeto.

Configuração das bibliotecas

Como vimos na introdução é preciso compilar esses dois arquivos em `jasper`, mas antes baixaremos os JARs do projeto JasperReports. Vamos acessar o site http://jasperforge.org (http://jasperforge.org), escolhendo o projeto *JasperReports Library*, clicando no botão *download* - sem logar no sistema. Na lista vamos escolher a versão mais recente, no nosso caso `4.7.0`, usando a extensão ZIP.

Na pasta apresentada já temos o ZIP baixado, então vamos extrair os arquivos. Entrando na pasta extraída encontramos o JAR principal na sub-pasta `dist`, o arquivo `jasperreports-4.7.0.jar`. Esse JAR ficará na pasta `lib` do projeto no Eclipse, vamos arrastá-lo realizando uma cópia.

Além do JAR principal há mais dependências que estão na pasta `lib` da distribuição do JasperReport. Nessa pasta encontramos diversos JARs. Queremos gerar o PDF em nosso projeto e para isso esses JARs serão necessários: o primeiro é o *itext*, que o JasperReport utiliza para gerar PDF. Como nosso relatório também gera um gráfico, utilizaremos o *jfreechart* e *jcommon*. Por último, utilizaremos alguns JARs dos Apache Commons.

Falta adicionar os JARs no classpath do projeto. Selecionando e usando *add to buildpath* dentro do Eclipse.

Compilação programática

Com as bibliotecas e com o classpath configurado podemos começar a implementar. Para compilar o arquivo `jrxml` usaremos a classe `JasperCompileManager`. Ela possui vários métodos estáticos para compilar o XML. Nós utilizaremos o método `compileReportToFile` que recebe o nome do XML. É preciso tratar a exceção checked `JRException`.

Vejo o código:

```
public class TesteRelatorio {

  public static void main(String[] args) throws SQLException, JRException {

    JasperCompileManager.compileReportToFile("gasto_por_mes.jrxml");
  }
}
```

Alterando a linguagem de script

A chamada do método `main` lança uma exceção, acusando um problema de compilação relacionada com a linguagem Groovy. É possível embutir scripts de Groovy dentro do relatório, mas não utilizaremos esse recurso. No relatório utilizaremos apenas a linguagem Java.

O problema é que o nosso relatório ainda está configurado para utilizar Groovy como linguagem padrão. Podemos alterar este padrão para a linguagem Java abrindo o arquivo `jrxml` no iReport e selecionando o elemento raiz no *Report Inspector*.

Em nosso caso, abriremos o `jrxml` diretamente no Eclipse e buscaremos pelo elemento *language*. Nesta opção, basta trocar de Groovy para Java. Isso deve ser feito nos dois arquivos `jrxml`.

```
<jasperReport ... name="gasto_por_mes" language="java" pageWidth="572" pageHeight="752" ...>
```

Com esta alteração, compilaremos o relatório mais uma vez. Agora temos o arquivo `jasper` gerado.

Preenchendo o relatório

Falta agora carregar este arquivo e preencher o relatório com os dados com base na conexão e os parâmetros da aplicação. Para essa tarefa existe uma classe `JasperFillManager` que possui o método `fillReport` que é sobrecarregado. Utilizaremos a versão que recebe o nome do arquivo `jasper`, o mapa de parâmetros e a conexão. Só falta inicializar o mapa de parâmetros. Criaremos um `HashMap` sem nenhum elemento, por enquanto:

```
public class TesteRelatorio {

  public static void main(String[] args) throws SQLException, JRException, FileNotFoundExceptio

    JasperCompileManager.compileReportToFile("gasto_por_mes.jrxml");

    Map<String, Object> parametros = new HashMap<String, Object>();
    Connection connexao = new ConnectionFactory().getConnection();

    JasperFillManager.fillReport("gasto_por_mes.jasper", parametros, connexao);

    connexao.close();
 }
}
```

O código compila, mas ao executá-lo recebemos uma exceção. O JasperReports acusa a falta do sub-relatório. É preciso compilar o sub-relatório também.

Alteraremos o nome do arquivo `jrxml` no código Java e chamaremos novamente o método `main` para gerar o arquivo `jasper` do sub-relatório. Atualizando o projeto, dois arquivos `jasper` aparecerão.

Com isso, preencheremos mais uma vez o relatório com o `JasperFillManager` , mas primeiro comentaremos a linha que gerou o arquivo `jasper` , já que não há a necessidade de recriá-lo.

Trabalhando com o objeto JasperPrint e várias saídas

Durante a execução nenhuma exceção foi lançada, mas também não vimos nenhum resultado. Olhando para assinatura do método `fillReport(..)` percebemos que ele devolve um objeto do tipo `JasperPrint` . Este objeto representa o relatório preenchido, mas é genérico, independente da saída concreta.

Agora só falta exportar o `JasperPrint` , mas isso é a responsabilidade de outra classe. Por esta razão existem `JRExporters` . `JRExporter` é uma interface e suas implementações definem a saída concreta. Há implementações para HTML, ODF, RTF, PDF entre várias outras.

Veja também o link para a documentação:
http://jasperreports.sourceforge.net/api/net/sf/jasperreports/engine/JRExporter.html
(http://jasperreports.sourceforge.net/api/net/sf/jasperreports/engine/JRExporter.html)

Exportando o relatório para PDF

Em nosso caso, exportaremos para PDF, usando a classe `JRPdfExporter` . O `JRPdfExporter` recebe como parâmetro o objeto `JasperPrint` (usando o método `setParameter(... , ...)`).

Além disso, através do mesmo método, definimos o `OutputStream` para configurar o nome do arquivo PDF. Nesse exemplo usamos um `FileOutputStream` como fluxo de saída e o arquivo se chamará `gasto_por_mes.pdf` .

Esse `OutputStream` também exige o tratamento da exceção `FileNotFoundException` . Por último é preciso chamar o método `export()` .

```
public class TesteRelatorio {

  public static void main(String[] args) throws SQLException, JRException, FileNotFoundExceptio
```

```
        JasperCompileManager.compileReportToFile("gasto_por_mes.jrxml");

        Map<String, Object> parametros = new HashMap<String, Object>();
        Connection connexao = new ConnectionFactory().getConnection();

        JasperPrint print = JasperFillManager.fillReport("gasto_por_mes.jasper", parametros, connexa

        JRExporter exporter = new JRPdfExporter();
        exporter.setParameter(JRExporterParameter.JASPER_PRINT, print);
        exporter.setParameter(JRExporterParameter.OUTPUT_STREAM, new FileOutputStream("gasto_por_mes
        exporter.exportReport();

        connexao.close();
    }
}
```

Após a chamada do `main` e depois de atualizar, aparecerá o PDF gerado na raiz do projeto. Abrindo o relatório podemos visualizar os dados das contas seguidas pelas movimentações. Como esperado, as informações encontram-se nas últimas páginas: o gráfico e o *crosstable*.