Exibir relatórios gerados pelo Jasper Report em Flex

Autor: Roberto Lourenço de Oliveira Júnior

Email: robertojr at comp dot ufla dot br

Página pessoal: http://alunos.dcc.ufla.br/~robertojr

O Jasper Report é uma ótima ferramenta para geração de relatórios pois tem consigo o iReport, uma ferramenta para montar o layout do seu relatório vizualmente, sem a necessidade de se criar toda a estrutura atraves do jrxml.

Neste artigo, mostrarei como integrar os relatórios gerados pelo Jasper Report com uma aplicação Flex, utilizando para isso o xViewer, que nada mais é que um Flex Viewer para Jasper Report.

Os requisitos para fazer o que for passado aqui podem ser baixados nos links abaixo:

- iReport - http://jasperforge.org/projects/ireport
- xViewerSample - http://www.adobe.com/cfusion/exchange/index.cfm?event=extensionDetail&extid=1384018

Com iReport vamos desenvolver o modelo de um relatório e com a aplicação de exemplo xViewerSample exibiremos o relatorio.

Na aplicação xViewreSample tem uma biblioteca chamada jrvc.swc que é a responsável pela exibição do relatório. Para exibir devemos passar o XML do relatório gerado para uma das classes que estão nessa biblioteca e esta cuidará de criar toda a visualização.

Antes de começarmos a desenvolver o relatório vamos criar uma tabela para atravez dos dados continos nela criar o modelo para o relatório. Abaixo segue a SQL para criar a tabela e inserir alguns dados nela.

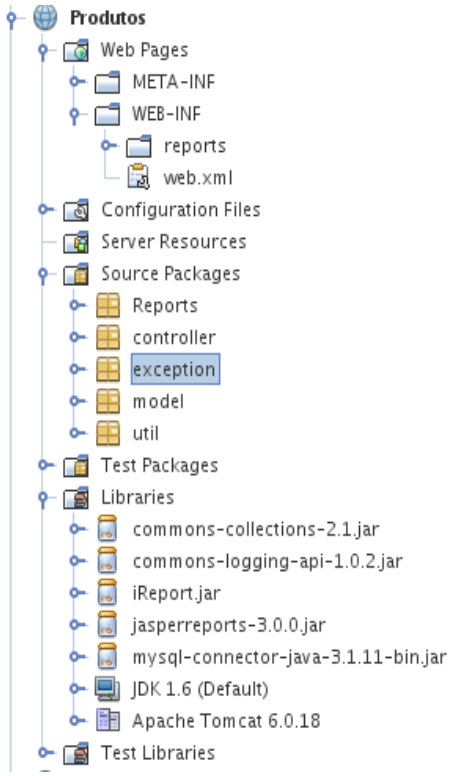
```
CREATE TABLE PRODUTO
(
  CODIGO   INTEGER NOT NULL,
  DESCRICAO VARCHAR(30) NOT NULL,
  PRECO NUMERIC(9, 2),
  PRIMARY KEY (CODIGO)
) TYPE = INNODB ;

INSERT INTO PRODUTO (CODIGO, DESCRICAO, PRECO) VALUES (1, 'produto1', 1);
INSERT INTO PRODUTO (CODIGO, DESCRICAO, PRECO) VALUES (2, 'produto2', 2);
INSERT INTO PRODUTO (CODIGO, DESCRICAO, PRECO) VALUES (3, 'produto3', 3);
INSERT INTO PRODUTO (CODIGO, DESCRICAO, PRECO) VALUES (4, 'produto4', 4);
INSERT INTO PRODUTO (CODIGO, DESCRICAO, PRECO) VALUES (5, 'produto5', 5);
INSERT INTO PRODUTO (CODIGO, DESCRICAO, PRECO) VALUES (6, 'produto6', 6);
INSERT INTO PRODUTO (CODIGO, DESCRICAO, PRECO) VALUES (7, 'produto7', 7);
```

```
INSERT INTO PRODUTO (CODIGO, DESCRICAO, PRECO) VALUES (8, 'produto8', 8);
INSERT INTO PRODUTO (CODIGO, DESCRICAO, PRECO) VALUES (9, 'produto9', 9);
INSERT INTO PRODUTO (CODIGO, DESCRICAO, PRECO) VALUES (10, 'produto10', 10);
```

Após executar essas SQL's, crie um modelo de relatório utilizando o iReport.

Vamos agora codificar algumas classes para comunicar com o banco de dados e gerar o relatório, e criaremos também um Servlet responsável pela impressão do XML que geraremos.

Cria uma aplicação Web e deixe a estrutura de pacotes da seguinte maneira:



Inclua na sua aplicação os seguinte pacotes:

- commons-collections-2.1.jar
- commons-logging-api-1.0.2.jar
- iReport.jar
- jasperreports-3.x.x.jar
- mysql-connector-java-3.1.11-bin.jar

Todos estes pacotes estão na pasta lib dentro do diretório onde foi instalado o iReport.

Agora criaremos a classe responsável por vincular o modelo do relatio, eventuais parâmetros necessário no relatório e a conexão com o banco:

```
public class RepositorioReports {

     public static JasperPrint generateReport(String pathModelReport, HashMap fieldReport, Connection connection) throws ReportException{

    JasperPrint reportPrint = null;

    try{

       reportPrint = JasperFillManager.fillReport(pathModelReport, fieldReport, connection);

    }catch(JRException e){

      throw new ReportException(e.getMessage());
```

```
      }
      return reportPrint;
   }
}
```

Esta classe deve ficar no pacote Reports.

A proxima classe é uma exceção que criaremos para ficar mais simplificar a manipulação das exceções no nosso Servlet. Como é de se esperar, esta classe deve ficar no pacote exception.

```
public class ReportException extends Exception {
   public ReportException(String msg){
      super(msg);
   }
}
```

A classe que tem os métodos para comunicação com o banco foi implementada por Peter Reutemann do Departamento de Ciência da Computação da Universidade de Waikato

```
public class Database {
 /** the driver. */
 public final static String DRIVER = "com.mysql.jdbc.Driver";

 /** the URL. */
 public static String URL = Constantes.URL_BANCO;
 /** the user. */
 public final static String USER = Constantes.USER_BANCO;

 /** the password. */
 public final static String PASSWORD = Constantes.PASS_BANCO;

 /** the actual connection. */
 protected Connection m_Connection;

 /** the singleton. */
 protected static Database m_Singleton;
 static {
  m_Singleton = null;
 }
 public Database(String url) {
```

```
  super();
  /** the URL. */
  URL = url;
  m_Connection = null;
}

public Database(){
   m_Connection = null;
}
public boolean connect() {
  boolean            result;

  try {
   Class.forName(DRIVER);
   m_Connection = DriverManager.getConnection(URL, USER, PASSWORD);
   result     = true;
  }
  catch (Exception e) {
   result = false;
  }

  return result;
}
public boolean disconnect() {
  boolean            result;

  try {
   if (m_Connection != null)
                    m_Connection.close();
   result = true;
  }
  catch (Exception e) {
   result = false;
  }

  return result;
```

```
}
public boolean isConnected() {
  return (m_Connection != null);
}
public int update(String sql) {
  int                   result;
  Statement        stmt;

  try {
    stmt   = m_Connection.createStatement();
    result = stmt.executeUpdate(sql);
  }
  catch (Exception e) {
    result = -1;
  }

  return result;
}
public ResultSet select(String sql) {
  ResultSet         result;
  Statement        stmt;

  try {
    stmt   = m_Connection.createStatement();
    result = stmt.executeQuery(sql);
  }
  catch (Exception e) {
    result = null;
  }

  return result;
}
public PreparedStatement prepare(String sql) {
  PreparedStatement result;

  try {
```

```
    result = m_Connection.prepareStatement(sql);
   }
   catch (Exception e) {
    result = null;
   }

   return result;
  }
  public void close(ResultSet rs) {
   try {
    Statement statement = rs.getStatement();
    rs.close();
    statement.close();
    statement = null;
    rs = null;
   }
   catch (Exception e) {
    // ignored
   }
  }
  public static Database getSingleton(String url) {
   if (m_Singleton == null)
    m_Singleton = new Database(url);

   return m_Singleton;
  }
  public static Database getSingleton() {
   if (m_Singleton == null)
    m_Singleton = new Database();

   return m_Singleton;
  }
  public Connection getConnection(){
    return this.m_Connection;
  }
}
```

Eu realizei algumas mudanças na classe, como por exemplo o último método getConnection() para que retorne o objeto Connection. Esta classe deve ficar no pacote model.

A classe a seguinte contém os dados para o acessoa ao banco de dados:

```
public class Constantes {
   public static String USER_BANCO = " ";
   public static String PASS_BANCO = " ";
   public static String URL_BANCO = "jdbc:mysql://localhost/xxx";
}
```

Esta classe deve ficar no pacote util.

Agora com a base da aplicação pronta, vamos desenvolver o Servlet que retornará o xml do relatório.

```
public class Produto extends HttpServlet {
     protected void processRequest(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {
      JasperPrint reportPrint = null;
      String reportXML = null; //String que guardará o xml do relatóroi
      Database conn = Database.getSingleton();
      conn.connect(); //Conecta o banco
      HashMap parametros = new HashMap(); //Parametos necessários ao relatório
      try{
           String pathModelReport = this.getServletContext().getRealPath("WEB-INF/reports/<nome_do_modelo>.jasper");
            reportPrint = RepositorioReports.generateReport(pathModelReport, parametros, conn.getConnection());
        reportPrint.setName("Produtos"); //Configura o nome do relatório
          reportXML = JasperExportManager.exportReportToXml(reportPrint); //Exporta o relatório para xml
      }catch(ReportException e){
        e.printStackTrace();
      }catch(JRException e){
        e.printStackTrace();
      }finally{
        conn.disconnect();
      }
      response.setContentType("text/xml"); //Configura o ContentType
           response.setCharacterEncoding("ISO-8859-1");//Configura  o  Character
```

```
Enconding
      PrintWriter outputStream = response.getWriter();
      outputStream.print(reportXML); // Imprime o xml
   }
      protected void doGet(HttpServletRequest request, HttpServletResponse
response)
   throws ServletException, IOException {
      processRequest(request, response);
   }
      protected void doPost(HttpServletRequest request, HttpServletResponse
response)
   throws ServletException, IOException {
      processRequest(request, response);
   }
   public String getServletInfo() {
      return "Short description";
   }
}
```

Mapeie o seu Servlet no arquivo web.xml, e coloque o Modelo do relatório compilado na pasta WEB-INF/reports. Agora execute o servlet e veja que apareceu um xml. É este o xml que passaremos para a aplicação Flex.

A parte mais difícil da aplicação está feita. Vamos agora criar a aplicação Flex e fazer tudo funcionar.

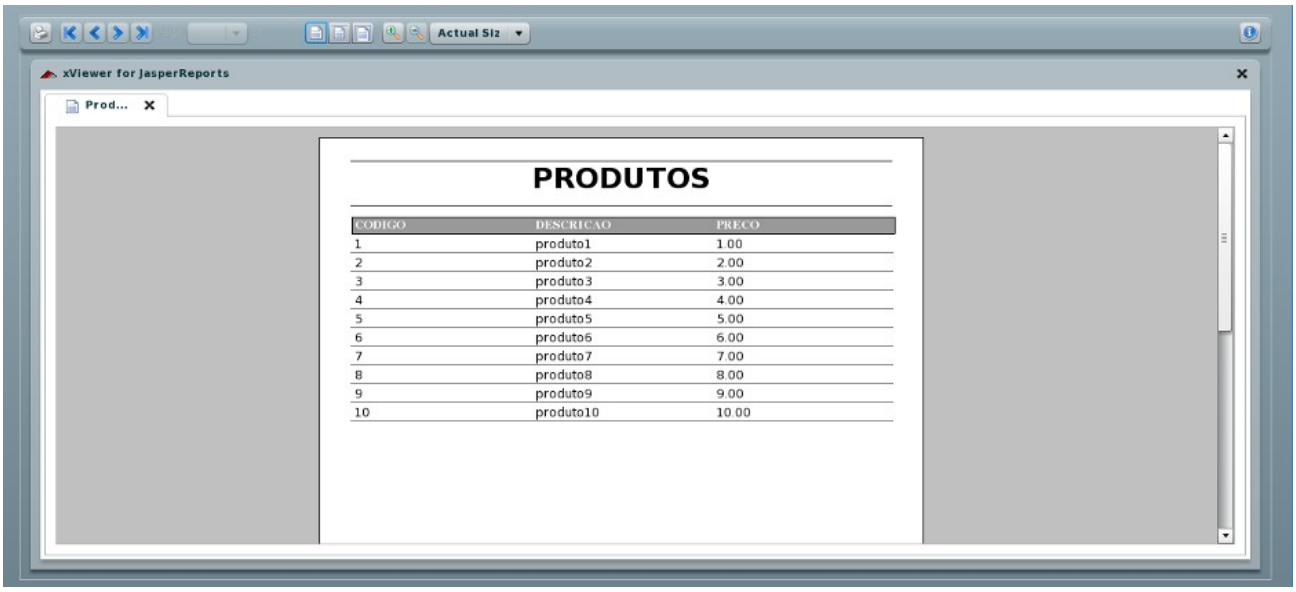
Importe o xViewerSample e abra o arquivo xViewerSample.mxml. Apague tudo o que estiver no arquivo e implemente os códigos a seguir:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
      xmlns:jrv="com.jaspersoft.reports.viewer.*">

      <!-- Area de visualizacao do relatorio -->
      <mx:Canvas left="20" right="20" top="20" bottom="8"
                  borderStyle="solid" creationComplete="{httpService(event)}">

            <jrv:Viewer id="viewer" label="teste"/>

      </mx:Canvas>

      <mx:Script>
            <![CDATA[
                  import mx.rpc.events.FaultEvent;
                  import mx.rpc.events.ResultEvent;
                  import mx.controls.Alert;
                  import mx.rpc.http.HTTPService;

                  /**
                   * Variavel que guarda um XML recebido por uma requisicao
```

```
                 */
                [Bindable]
            private var xmlData:XML = null;



                /**
                 * @description Esta funcao e' responsa'vel pela geracao da
visualizacao
                 * do relatorio
                 * @return void
                */
                private function viewReport():void {
                      viewer.disableJasperServer();
                      viewer.showClassicToolbar();
                      viewer.showReport(xmlData);
                }
                /**
                 * @description Esta funcao manipula o resultado da requisicao
                 * conventendo este resultado em um xml
                 * @param ResultEvent - Evento lancado pelo metodo
HTTPService.send()
                 *                                  caso a requisicao obteve
sucesso
                */
                private function resultHandler(event:ResultEvent):void{
                      xmlData = event.result as XML;
                      this.viewReport();
                }

                /**
                 * @description Esta funcao mostra um alerta caso ocorra algum
erro durante
                 * a requisicao
                 * @param FaultEvent - Evento lancado pelo metodo
HTTPService.send()
                 *                                em caso a requisicao falhar
                 */
                private function failRequest(event:FaultEvent):void{
                      Alert.show("Ops! Houve algum erro ao processar a
requisiÃƒÂ§ÃƒÂ£o.");
                }

                /**
                 * @description Esta funcao realiza uma requisicao HTTP
                 * @param Event - Evento lancado por alguma acao qualquer. Neste
contexto
                 * e um evento lancado por um Combobox que lista os relatorios
                 *                        disponiveis ao usuario.
                 */
                private function httpService(event:Event):void{
                      var httpService:HTTPService = new HTTPService();
                httpService.url= "http://localhost:8084/Produtos/Produto"
                      httpService.resultFormat = "e4x";
                      httpService.addEventListener(ResultEvent.RESULT,
this.resultHandler);
                      httpService.addEventListener(FaultEvent.FAULT,
this.failRequest);
```

```
                    httpService.send();
                }
            ]]>
        </mx:Script>
</mx:Application>
```

Rode o servlet que criamos e logo em seguida rode esta aplicação Flex, o resultado deve ser algo do tipo:



Conclusões:

A lógica para visualizar um relatório em flex é bem sim, e se resume nos seguintes passos:

- Gerar um relatório e exportá-lo para xml
- Realizar uma requisição html na aplicação Flex para o JSP / Servlet que “imprimiu” o xml do relatório
- Converter o resultado da requisição para xml e passar para o metódo viewReport

Depois de gerar a visualização do relatório, é deixar a criatividade comandar o incremento da aplicação.

É válido lembrar que esta biblioteca para flex oferece suporte para JasperServer e muitos outros recursos, portanto ainda se tem muito para estudar a respeito dela, uma vez que a sua documentação não é extensa.