JasperReports with Spring

Last updated: January 8, 2024

Written by: baeldung (https://www.baeldung.com/author/baeldung)

Reviewed by: Grzegorz Piwowarek (https://www.baeldung.com/editor/grzegorz-author)

Spring (https://www.baeldung.com/category/spring) +

1. Overview

JasperReports (https://community.jaspersoft.com/knowledgebase/getting-started/getting-started-jasperreports-library/) is an open
In this article, we'll explore its key features and classes, and implement examples to showcase its capabilities.

2. Maven Dependency

First, we need to add the *jasperreports* dependency to our *pom.xml*:

```
<dependency>
    <groupId>net.sf.jasperreports</groupId>
    <artifactId>jasperreports</artifactId>
    <version>6.20.0</version>
</dependency>
```

The latest version of this artifact can be found here (https://mvnrepository.com/artifact/net.sf.jasperreports/jasperreports).

3. Report Templates

Report designs are defined in JRXML files. These are ordinary XML files with a particular structure that JasperReports engine can interp
Let's now have a look at only the relevant structure of the JRXML files – to understand better the Java part of the report generation pro
Let's create a simple report to show employee information:

```
<jasperReport ... >
    <field name="FIRST_NAME" class="java.lang.String"/>
    <field name="LAST_NAME" class="java.lang.String"/>
    <field name="SALARY" class="java.lang.Double"/>
    <field name="ID" class="java.lang.Integer"/>
    <detail>
        <band height="51" splitType="Stretch">
            <textField>
                <reportElement x="0" y="0" width="100" height="20"/>
                <textElement/>
                <textFieldExpression class="java.lang.String">
                  <![CDATA[$F{FIRST_NAME}]]></textFieldExpression>
            </textField>
            <textField>
                <reportElement x="100" y="0" width="100" height="20"/>
                <textElement/>
                <textFieldExpression class="java.lang.String">
                  <![CDATA[$F{LAST_NAME}]]></textFieldExpression>
            </textField>
            <textField>
                <reportElement x="200" y="0" width="100" height="20"/>
                <textElement/>
                <textFieldExpression class="java.lang.String">
                  <![CDATA[$F{SALARY}]]></textFieldExpression>
            </textField>
        </band>
    </detail>
</jasperReport>
```

3.1. Compiling Reports

JRXML files need to be compiled so the report engine can fill them with data.

Let's perform this operation with the help of the *JasperCompilerManager* class:

```
InputStream employeeReportStream
  = getClass().getResourceAsStream("/employeeReport.jrxml");
JasperReport jasperReport
  = JasperCompileManager.compileReport(employeeReportStream);
```

To avoid compiling it every time, we can save it to a file:

```
JRSaver.saveObject(jasperReport, "employeeReport.jasper");
```

4. Populating Reports

The most common way to fill compiled reports is with records from a database. This requires the report to contain a SQL query the eng

First, let's modify our report to add a SQL query:

```
<jasperReport ... >
    <queryString>
        <![CDATA[SELECT * FROM EMPLOYEE]]>
    </queryString>
    ...
</jasperReport>
```

Now, let's create a simple data source:

```
@Bean
public DataSource dataSource() {
    return new EmbeddedDatabaseBuilder()
      .setType(EmbeddedDatabaseType.HSQL)
      .addScript("classpath:employee-schema.sql")
      .build();
}
```

Now, we can fill the report:

```
JasperPrint jasperPrint = JasperFillManager.fillReport(
  jasperReport, null, dataSource.getConnection());
```

Note that we are passing *null* to the second argument since our report doesn't receive any parameters yet.

4.1. Parameters

Parameters are useful for passing data to the report engine that it can not find in its data source or when data changes depending on d

We can also change portions or even the entire SQL query with parameters received in the report filling operation.

First, let's modify the report to receive three parameters:

```
<jasperReport ... >
    <parameter name="title" class="java.lang.String" />
    <parameter name="minSalary" class="java.lang.Double" />
    <parameter name="condition" class="java.lang.String">
        <defaultValueExpression>
          <![CDATA["1 = 1"]]></defaultValueExpression>
    </parameter>
    // ...
</jasperreport>
```

Now, let's add a title section to show the *title* parameter:

```
<jasperreport ... >
    // ...
    <title>
        <band height="20" splitType="Stretch">
            <textField>
                <reportElement x="238" y="0" width="100" height="20"/>
                <textElement/>
                <textFieldExpression class="java.lang.String">
                  <![CDATA[$P{title}]]></textFieldExpression>
            </textField>
        </band>
    </title>
    ...
</jasperreport/>
```

Next, let's alter the query to use the *minSalary* and *condition* parameters:

```
SELECT * FROM EMPLOYEE
  WHERE SALARY >= $P{minSalary} AND $P!{condition}
```

Note the different syntax when using the *condition* parameter. This tells the engine that the parameter should not be used as a standar

Finally, let's prepare the parameters and fill the report:

```
Map<String, Object> parameters = new HashMap<>();
parameters.put("title", "Employee Report");
parameters.put("minSalary", 15000.0);
parameters.put("condition", " LAST_NAME ='Smith' ORDER BY FIRST_NAME");

JasperPrint jasperPrint
  = JasperFillManager.fillReport(..., parameters, ...);
```

Note that the keys of *parameters* correspond to parameter names in the report. If the engine detects a parameter is missing, it will obta

5. Exporting

To export a report, first, we instantiate an object of an exporter class that matches the file format we need.

Then, we set our previous filled report as input and define where to output the resulting file.

Optionally, we can set corresponding report and export configuration objects to customize the exporting process.

5.1. PDF

```
JRPdfExporter exporter = new JRPdfExporter();

exporter.setExporterInput(new SimpleExporterInput(jasperPrint));
exporter.setExporterOutput(
  new SimpleOutputStreamExporterOutput("employeeReport.pdf"));

SimplePdfReportConfiguration reportConfig
  = new SimplePdfReportConfiguration();
reportConfig.setSizePageToContent(true);
reportConfig.setForceLineBreakPolicy(false);

SimplePdfExporterConfiguration exportConfig
  = new SimplePdfExporterConfiguration();
exportConfig.setMetadataAuthor("baeldung");
exportConfig.setEncrypted(true);
exportConfig.setAllowedPermissionsHint("PRINTING");

exporter.setConfiguration(reportConfig);
exporter.setConfiguration(exportConfig);

exporter.exportReport();
```

5.2. XLS

```
JRXlsxExporter exporter = new JRXlsxExporter();

// Set input and output ...
SimpleXlsxReportConfiguration reportConfig
  = new SimpleXlsxReportConfiguration();
reportConfig.setSheetNames(new String[] { "Employee Data" });

exporter.setConfiguration(reportConfig);
exporter.exportReport();
```

5.3. CSV

```
JRCsvExporter exporter = new JRCsvExporter();

// Set input ...
exporter.setExporterOutput(
  new SimpleWriterExporterOutput("employeeReport.csv"));

exporter.exportReport();
```

5.4. HTML

```
HtmlExporter exporter = new HtmlExporter();

// Set input ...
exporter.setExporterOutput(
  new SimpleHtmlExporterOutput("employeeReport.html"));

exporter.exportReport();
```

6. Subreports

Subreports are nothing more than a standard report embedded in another report.

First, let's create a report to show the emails of an employee:

```
<jasperReport ... >
    <parameter name="idEmployee" class="java.lang.Integer" />
    <queryString>
        <![CDATA[SELECT * FROM EMAIL WHERE ID_EMPLOYEE = $P{idEmployee}]]>
    </queryString>
    <field name="ADDRESS" class="java.lang.String"/>
    <detail>
        <band height="20" splitType="Stretch">
            <textField>
                <reportElement x="0" y="0" width="156" height="20"/>
                <textElement/>
                <textFieldExpression class="java.lang.String">
                  <![CDATA[$F{ADDRESS}]]></textFieldExpression>
            </textField>
        </band>
    </detail>
</jasperReport>
```

Now, let's modify our employee report to include the previous one:

```
<detail>
    <band ... >
        <subreport>
            <reportElement x="0" y="20" width="300" height="27"/>
            <subreportParameter name="idEmployee">
                <subreportParameterExpression>
                  <![CDATA[$F{ID}]]></subreportParameterExpression>
            </subreportParameter>
            <connectionExpression>
              <![CDATA[$P{REPORT_CONNECTION}]]></connectionExpression>
            <subreportExpression class="java.lang.String">
              <![CDATA["employeeEmailReport.jasper"]]></subreportExpression>
        </subreport>
    </band>
</detail>
```

Note that we are referencing the subreport by the name of the compiled file and passing it the *idEmployee* and current report connect

Next, let's compile both reports:

```
InputStream employeeReportStream
  = getClass().getResourceAsStream("/employeeReport.jrxml");
JasperReport jasperReport
  = JasperCompileManager.compileReport(employeeReportStream);
JRSaver.saveObject(jasperReport, "employeeReport.jasper");

InputStream emailReportStream
  = getClass().getResourceAsStream("/employeeEmailReport.jrxml");
JRSaver.saveObject(
  JasperCompileManager.compileReport(emailReportStream),
  "employeeEmailReport.jasper");
```

Our code for filling and exporting the report doesn't require modifications.

7. Conditional Display With *printWhenExpression*

In addition, we can use *printWhenExpression* to conditionally display report elements based on certain criteria. This means that elemer

Below is an example of how to modify the JRXML file to include a null check using *printWhenExpression*. We'll check for non-null value

```
<jasperReport ... >
    <field name="SALARY" class="java.lang.Double"/>
    <!-- other fields -->

    <detail>
        <band height="51" splitType="Stretch">
            <printWhenExpression><![CDATA[$F{FIRST_NAME} != null && $F{LAST_NAME} != null && $F{SALARY} != null]]></printWhenExpre

            <!-- Existing text fields and subreport-->

        </band>
    </detail>
</jasperReport>
```

This expression ensures that the entire content of the band will only be displayed if all these fields have valid (non-null) values. After up

8. Conclusion

In this article, we had a brief look at the core features of the JasperReports library.

We were able to compile and populate reports with records from a database; we passed parameters to change the data shown in the r

Complete source code for this article can be found over on GitHub (https://github.com/eugenp/tutorials/tree/master/libraries-report