3

创建第一份报表

本章我们将创建、编译、预览我们的第一份报表。

通过本章学习，我们将能够：

- 创建一份简单的 JRXML 报表模板
- 用 JasperReports 自定义 ANT 目标来预览报表模板
- 编译 JRXML 文件生成 Jasper 二进制报表模板
- 编写从 JasperReports 模板生成报表的代码
- 通过 JaspreReports 提供的工具浏览生成的 JasperReports 本地格式报表
- 生成能够显示在 web 浏览器中的报表
- 识别与报表中各片段相对应的 JRXML 元素

创建 JRXML 报表模板

创建报表的第一步是创建一份 JRXML 模板，我们在第 1 章中介绍了，JasperReports 的 JRXML 模板是标准的 XML 文件。但是，依照惯例，其扩展名为.jrxml，称为 JRXML 文件或 JRXML 模板。JRXML 模板可以手工编写，也可以使用可视化报表模板生成工具。时下，最流行的 JRXML 报表模板生成工具是 iReport。我们将在第 10 章中介绍它。

所有的JRXML文件都包含一个<jasperReport>根元素,它又包含若干子元素。所有的子元素都是可选的。因为而我们在这一章里主要是感受一下怎样设计报表，所以略掉了大多数<jasperReport>子元素，而只用了一个名为<detail>的子元素。

我们的第一份报表将显示一个静态的字符串，其 JRXML 如下：

```
<?xml version="1.0"?>
<jasperReport
     xmlns="http://jasperreports.sourceforge.net/jasperreports"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://jasperreports.sourceforge.net/
                                jasperreports http://jasperreports.
                                sourceforge.net/xsd/jasperreport.xsd"
     name="FirstReport">
   <detail>
      <band height="20">
        <staticText>
            <reportElement x="20" y="0" width="200" height="20"/>
            <text>
                    <![CDATA[If you don't see this, it didn't work]]>
            </text>
        </staticText>
      </band>
   </detail>
</jasperReport>
```

在这个 JRXML 文件的元素中，有些我们在前面的章节中还没有接触过。例如：

- <staticText> 定义了静态文本，它不需要依赖于任何数据源、变量、参数或报表表达式。
- <reportElement> 定义了<staticText>元素的位置和宽度。
- <text> 定义了显示在报表上的实际的静态文本。

在上面的例子中，我们看到了<band>元素。<detail>元素只能包含一个单独的<band>元素作为其子元素，而<band>元素可以包含许多不同的元素，用来显示文本、图表、图片或几何图形。本例中，它只包含了一个<staticText>元素。

对于<staticText>元素和所有<band>元素的子元素，<reportElement>元素都是必需的。<reportElement>中的定义的 x 和 y 相对于包含它的父元素（本例中为<staticText>）。

预览 XML 报表模板

JasperReports 里有个工具可以用来预览报表的设计效果，它使得报表的设计更加快速。通过它，我们不用编译和填充就能立即对报表进行预览。

该工具是一个独立的 Java 应用程序，它包含在 JasperReports 的 JAR 文件中。需要执行的类是 net.sf.jasperreports.view.JasperDesignViewer，执行它的最简单方法是把所有需要的库都添加到 CLASSPATH 中，然后使用 ANT 目标来运行它，这也是在工程 JAR 文件中的 JasperReports 范例所使用的方法，下面的 ANT 构建文件将加载 JasperDesignViewer 来预览我们刚完成的报表：

```
<project name="FirstReport XML Design Preview" default="viewDesignXML"
         basedir=".">
  <description>
     Previews our First Report XML Design
  </description>
  <property name="file.name" value="FirstReport" />
<!-- Directory where the JasperReports project file was extracted,
     needs to be changed to match the local environment -->
  <property name="jasper.dir"
            value="/opt/jasperreports-3.5.2"/>
  <property name="classes.dir" value="${jasper.dir}/build/classes" />
  <property name="lib.dir" value="${jasper.dir}/lib" />
  <path id="classpath">
  <pathelement location="./"/>
  <pathelement location="${classes.dir}" />
  <fileset dir="${lib.dir}">
     <include name="**/*.jar"/>
  </fileset>
  </path>
  <target name="viewDesignXML"
          description="Launches the design viewer to preview the XML
                       report design.">
```

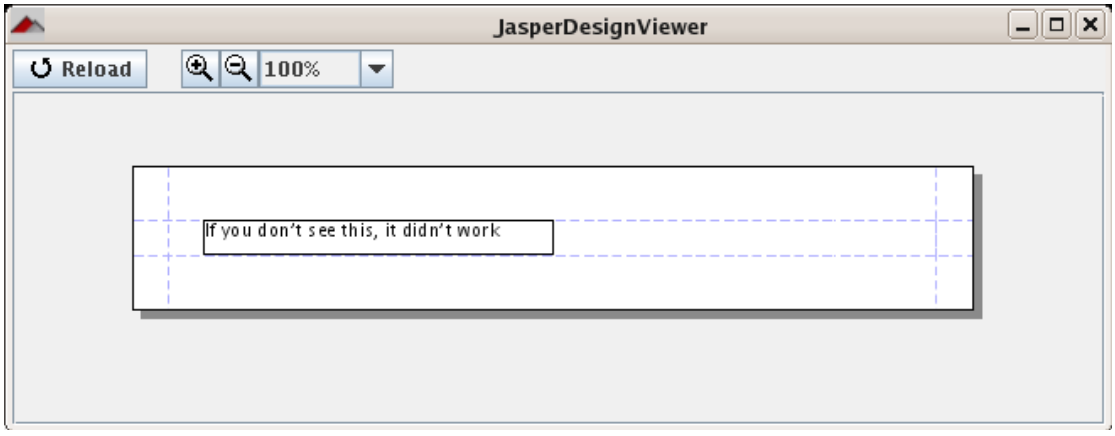
```
        <java classname="net.sf.jasperreports.view.JasperDesignViewer"
                fork="true">
            <arg value="-XML"/>
            <arg value="-F${file.name}.jrxml"/>
            <classpath refid="classpath"/>
        </java>
    </target>
</project>
```

这个 ANT 构建文件必须和 JRXML 文件保存在相同的目录中，建议把 JRXML 文件的报表名和文件名使用一致的名称。报表名称在<jasperReport>根元素中定义，在这里，我们用 FirstReport 作为报表名称。因此，建议报表模板的文件名使用 FirstReport.jrxml。

如果我们保存 ANT 构建文件为标准名称 build.xml，就不需要在命令行中再指定构建文件名。此处的示例中，构建文件有一个名为 viewDesignXML 的<target>目标元素。由于它是缺省的目标，我们就不需要在命令行中指定它。只需要在命令行输入 ant，缺省的目标就会被执行，并显示出预览报表。

```
$ ant
Buildfile: previewReportDesignXML.xml
viewDesignXML:
```

执行完 viewDesignXML 目标后，我们会看到一个窗口显示出预览状态的报表模板，其标签名称为 JasperDesignViewer。



通过关闭窗口，或在命令行窗口中输入 Ctrl-c，这个 JasperDesignViewer 可以被安全地关闭。

在这个特殊的例子中中，我们能在预览状态下看到全部文本，这是因为此报表只包含了静态文本。如果报表的数据是来源于数据库或报表参数，此处不会显示实际文本，而是显示包含了要显示的数据的表达式。这是因为 JasperDesignViewer 不会对数据源或报表参数进行实际访问。

创建二进制报表模板

JRXML 不能用来直接生成报表，它们需要被编译成 JasperReports 的本地二进制格式，编译后的报表模板称为 Jasper 文件。有两种不同的方法来把 JRXML 文件编译成 Jasper 文件：通过编程实现，或通过 JasperReports 提供的自定义 ANT 任务来实现。

通过编程来编译 JRXML 模板

通过调用 net.sf.jasperreports.engine.JasperCompileManager 类的 compileReportToFile()方法，可以把 JRXML 模板编译成 Jasper 文件。JasperCompileManager.compileReportToFile()方法有三个重载版本，如下所示：

- JasperCompileManager.compileReportToFile(String sourceFileName)
- JasperCompileManager.compileReportToFile(String sourceFileName, String destFileName)
- JasperCompileManager.compileReportToFile(JasperDesign jasperDesign, String destFileName)

下面的表格对这些方法的参数进行了说明：

参数	说明
String sourceFileName	此参数指定 JRXML 模板的位置，它被用于编译生成报表模板。它可以是绝对路径，也可以相对路径。编译生成

	的报表模板会以相同的文件名保存到和它相同的位置，并以.jasper 作为扩展名。
String destFileName	此参数指定在文件系统中用于保存编译生成的报表模板的文件名，它可以是绝对路径或相对路径。
JasperDesign jasperDesign	这是报表在内存中的存在形式，net.sf.jasperreports.engine.design.JasperDesign 实例可以通过在 net.sf.jasperreports.engine.xml.JRXmlLoader 中调用相应的方法来创建。

下面的代码段演示了怎样使用 JasperCompileManager.compileReportToFile()方法：

```
package net.ensode.jasperbook;
import net.sf.jasperreports.engine.JRException;
import net.sf.jasperreports.engine.JasperCompileManager;
public class FirstReportCompile
{
    public static void main(String[] args)
    {
        try
        {
            System.out.println("Compiling report...");
            JasperCompileManager.compileReportToFile(
                                                "reports/FirstReport.jrxml");
            System.out.println("Done!");
        }
        catch (JRException e)
        {
            e.printStackTrace();
        }
    }
}
```

编译执行上面的代码，我们会得到一个名为 FirstReport.jasper 的文件。此文件就是编译生成的 JasperReports 本地格式的模板。在本例中，我们使用前面讨论的第一个版本的 JasperCompileManager.compileReportToFile()方法，源文件名会被用于编译生成的报表。如果我们希望编译后的报表为不同的文件名，就需要使用第二个版本的

JasperCompileManager.compileReportToFile()方法，并在其第二个参数中指定要生成的名字。

预览编译后的报表模板

前面讨论的 net.sf.jasperreports.view.JasperDesignViewer 可用于预览编译后的报表模板，就象 JRXML 模板一样。执行此工具的最简单的方法是在 ANT 目标中封装一个调用。下面，我们在 build.xml 文件中添加第二个 ANT 目标，并把它命名为 viewDesign，它将使我们能够预览编译后的报表。

```
<project name="FirstReport XML Design Preview" default="viewDesignXML"
        basedir=".">
   <description>Previews and compiles our First Report</description>
   <property name="file.name" value="FirstReport" />
<!-- Directory where the JasperReports project file was extracted,
     needs to be changed to match the local environment -->
   <property name="jasper.dir"
             value="/opt/jasperreports-3.5.2" />
   <property name="classes.dir" value="${jasper.dir}/build/classes" />
   <property name="lib.dir" value="${jasper.dir}/lib" />
   <path id="classpath">
      <pathelement location="./" />
      <pathelement location="${classes.dir}" />
      <fileset dir="${lib.dir}">
         <include name="**/*.jar" />
      </fileset>
   </path>
   <target name="viewDesignXML"
           description="Launches the design viewer to preview the XML
                        report design.">
      <java classname="net.sf.jasperreports.view.JasperDesignViewer"
            fork="true">
         <arg value="-XML" />
         <arg value="-F${file.name}.jrxml" />
         <classpath refid="classpath" />
      </java>
   </target>
   <target name="viewDesign"
           description="Launches the design viewer to preview the
                        compiled report design.">
```

```
        <java classname="net.sf.jasperreports.view.JasperDesignViewer"
              fork="true">
           <arg value="-F${file.name}.jasper" />
           <classpath refid="classpath" />
        </java>
    </target>
    <target name="compile"
            description="Compiles the XML report design and produces the
                         .jasper file.">
    <taskdef name="jrc"
             classname="net.sf.jasperreports.ant.JRAntCompileTask">
       <classpath refid="classpath" />
    </taskdef>
    <jrc destdir=".">
       <src>
          <fileset dir=".">
             <include name="**/*.jrxml" />
          </fileset>
       </src>
       <classpath refid="classpath" />
    </jrc>
    </target>
    <target name="view"
            description="Launches the report viewer to preview the
                         report stored in the .jrprint file.">
       <java classname="net.sf.jasperreports.view.JasperViewer"
             fork="true">
          <arg value="-F{file.name}.jrprint" />
          <classpath refid="classpath" />
       </java>
    </target>
</project>
```

我们可以从命令行调用这个新目标：

ant viewDesign

调用完成后，我们会看到一个和前面的 JRXML 模板预览相同的窗口。

用 ANT 来编译 JRXML 模板

JasperReports 包含一个自定义的 ANT 任务，用它可以来编译报表模板。其编译方式非常便利，我们不需要编写执行编译的代码。然而，对于一些特殊的应用程序，还是需要通过编写相应的代码来编译报表的（比如 JRXML 文件是在运行时进行才创建的情况）。JasperReports 中的自定义 ANT 任务在 net.sf.jasperreports.ant.JRAntCompileTask 类中进行定义，名为 jrc。下面，让我们在 build.xml 中添加第三个自定义的目标来调用 jrc 任务。

```
<project name="FirstReport XML Design Preview" default="viewDesignXML"
        basedir=".">
   <description>Previews and compiles our First Report</description>
   <property name="file.name" value="FirstReport" />
<!-- Directory where the JasperReports project file was extracted,
     needs to be changed to match the local environment -->
   <property name="jasper.dir"
             value="/opt/jasperreports-3.5.2" />
   <property name="classes.dir" value="${jasper.dir}/build/classes" />
   <property name="lib.dir" value="${jasper.dir}/lib" />
   <path id="classpath">
      <pathelement location="./" />
      <pathelement location="${classes.dir}" />
      <fileset dir="${lib.dir}">
         <include name="**/*.jar" />
      </fileset>
   </path>
   <target name="viewDesignXML"
           description="Launches the design viewer to preview the XML
                        report design.">
      <java classname="net.sf.jasperreports.view.JasperDesignViewer"
            fork="true">
         <arg value="-XML" />
         <arg value="-F${file.name}.jrxml" />
         <classpath refid="classpath" />
      </java>
   </target>
   <target name="viewDesign"
           description="Launches the design viewer to preview the
                        compiled report design.">
      <java classname="net.sf.jasperreports.view.JasperDesignViewer"
            fork="true">
         <arg value="-F${file.name}.jasper" />
         <classpath refid="classpath" />
```

```
        </java>
    </target>
    <target name="compile"
            description="Compiles the XML report design and produces the
                         .jasper file.">
    <taskdef name="jrc"
              classname="net.sf.jasperreports.ant.JRAntCompileTask">
        <classpath refid="classpath" />
    </taskdef>
    <jrc destdir=".">
        <src>
            <fileset dir=".">
                <include name="**/*.jrxml" />
            </fileset>
        </src>
        <classpath refid="classpath" />
    </jrc>
    </target>
</project>
```

可以从命令行来调用这个新目标：

ant compile

编译目标会产生如下的输出：

Buildfile: build.xml

compile:

[jrc] Compiling 1 report design files.

[jrc] File : /home/heffel/NetBeansProjects/Code_8082/jasper_book_chapter_3/reports/FirstReport.jrxml ... OK.

BUILD SUCCESSFUL

Total time: 3 seconds

目标编译完成后，我们会在文件系统里得到一个 FirstReport.jasper 文件。

这个文件和通过调用 net.sf.jasperreports.engine.JasperCompileManager.compileReportToFile()方法生成的文件是一样的。

这里生成的 Jasper 文件也可以用 JasperReports 中包含的 JasperDesign 工具进行预览，其方法和输出与前面章节介绍的是相同的。

生成报表

按照 JasperReports 的习惯，从报表模板或 Jasper 文件生成报表的处理过程，称为填充(filling)报表。报表通过编写程序来进行填充，具体调用的是 net.sf.jasperreports.engine.JasperFillManager 类的 fillReportToFile()方法。此方法会将填充完报表并把它保存到磁盘。

fillReportToFile()方法有六个重载版本，如下所示：

- JasperFillManager.fillReportToFile(JasperReport jasperReport, String destFileName, Map parameters, Connection connection)
- JasperFillManager.fillReportToFile(JasperReport jasperReport, String destFileName, Map parameters, JRDataSource datasource)
- JasperFillManager.fillReportToFile(String sourceFileName, Map parameters, Connection connection)
- JasperFillManager.fillReportToFile(String sourceFileName, Map parameters, JRDatasource dataSource)
- JasperFillManager.fillReportToFile(String sourceFileName, String destFileName, Map parameters, Connection connection)
- JasperFillManager.fillReportToFile(String sourceFileName, String destFileName, Map parameters, JRDataSource dataSource)

下面的表格对这些方法的参数进行了说明：

参数	说明

JasperReport jasperReport	此参数用于报表模板，net.sf.jasperreports.engine.JasperReport 实例是编译后的报表模板在内存中的存在形式。
String destFileName	指定用于保存报表的目标文件名。
Map parameters	这是一个 java.util.Map 接口实现类的实例，用于初始化报表模板中定义的所有报表参数。
Connection connection	此参数用于连接数据库，以便执行报表模板中定义的 SQL 查询。
JRDataSource dataSource	这是一个 net.sf.jasperreports.engine.JRDataSource接口实现类的实例。

可以看出，在大多数情况下，我们通过一个实现了 net.sf.jasperreports.engine.JRDataSource 接口的类实例来传送填充报表的数据。报表模板可以包含内嵌的 SQL 查询，它们定义在 JRXML 文件的 <queryString>元素中。包含有 SQL 查询的报表不会传递 JRDataSource，而是传递一个实现了 java.sql.Connection 接口的类实例。JasperReports 再用这个 Connection 对象执行查询，从数据库获取报表数据。

尽管在报表模板中嵌入 SQL 查询可以简化开发，但传递 JRDataSource 还有其它的好处。即相同的报表可以使用不同的数据源，如数据库、CSV 文件、Java 对象，等等。我们将在第 5 章中详细讨论 JasperReports 对其它数据源的支持。

在这里的报表例子中，我们只包含了静态文本，不需要显示动态数据。由于没有 JRDataSource 或 Connection 就没办法填充报表，因此 JaspreReports 提供了一个不包含任何数据的 JRDataSoruce 实现，其名称为 JREmptyDataSource。另外，由于我们这里的报表没有参数，所以只需传递一个 java.util.HashMap 的空实例。我们将遵循 JasperReports 推荐的方法，把报表命名为与报表模板相同的名字（不包括文件的扩展名）。考虑到所有这些因素，最适当的 fillReportToFile()版本是上面列出的第 4 个，如下所示：

JasperFillManager.fillReportToFile(String sourceFileName, Map parameters, JRDataSource dataSource)

下面的 Java 类将添充报表，并把它保存在磁盘里：

```
package net.ensode.jasperbook;
import java.util.HashMap;
import net.sf.jasperreports.engine.JREmptyDataSource;
import net.sf.jasperreports.engine.JRException;
import net.sf.jasperreports.engine.JasperFillManager;
public class FirstReportFill
{
   public static void main(String[] args)
   {
      try
      {
         System.out.println("Filling report...");
         JasperFillManager.fillReportToFile("reports/FirstReport.jasper",
                                            new HashMap(),
                                            new JREmptyDataSource());
         System.out.println("Done!");
      }
      catch (JRException e)
      {
         e.printStackTrace();
      }
   }
}
```

执行这个类，我们会在编译 FirstReport.jasper 报表模板的目录里得到一个名为 FirstReport.JRprint 的文件。

查看报表

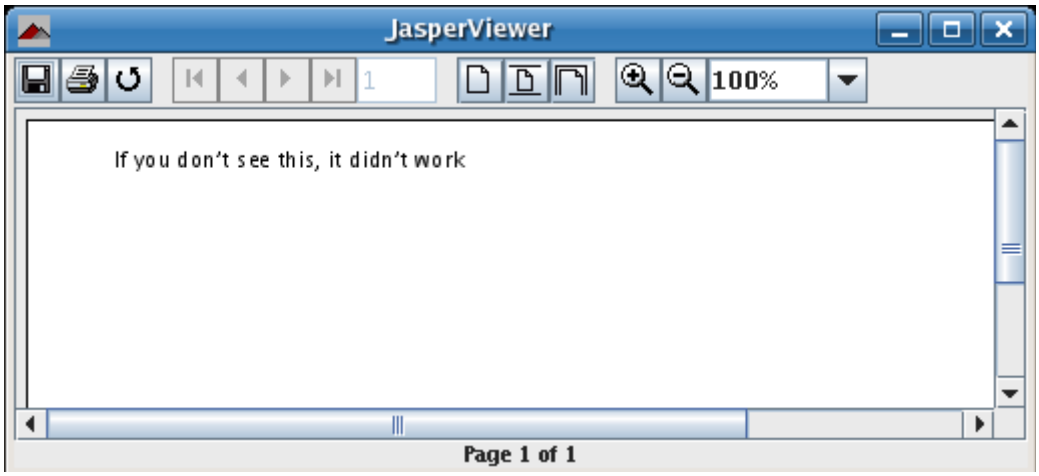
JasperReports 中有个工具名为 net.sf.jasperreports.view.JasperViewer 的类，可以用来查看生成的报表。和报表设计预览的工具一样，其最简使用方法是:把它包装到 ANT 目标里。这是 JasperReprots 内含示例程序所使用的方法，也是我们这里将要使用的方法。现在让我们添加一个新的目标到 ANT 构建文件中，按照 JasperReports 范例中的惯例，我们将此目标命名为“view”。

```
<project name="FirstReport XML Design Preview" default="viewDesignXML"
         basedir=".">
   <description>Previews and compiles our First Report</description>
   <property name="file.name" value="FirstReport" />
<!-- Directory where the JasperReports project file was extracted,
```

```
  needs to be changed to match the local environment -->
<property name="jasper.dir"
          value="/opt/jasperreports-3.5.2" />
<property name="classes.dir" value="${jasper.dir}/build/classes" />
<property name="lib.dir" value="${jasper.dir}/lib" />
<path id="classpath">
   <pathelement location="./" />
   <pathelement location="${classes.dir}" />
   <fileset dir="${lib.dir}">
      <include name="**/*.jar" />
   </fileset>
</path>
<target name="viewDesignXML"
        description="Launches the design viewer to preview the XML
                     report design.">
   <java classname="net.sf.jasperreports.view.JasperDesignViewer"
         fork="true">
      <arg value="-XML" />
      <arg value="-F${file.name}.jrxml" />
      <classpath refid="classpath" />
   </java>
</target>
<target name="viewDesign"
        description="Launches the design viewer to preview the
                     compiled report design.">
   <java classname="net.sf.jasperreports.view.JasperDesignViewer"
         fork="true">
      <arg value="-F${file.name}.jasper" />
      <classpath refid="classpath" />
   </java>
</target>
<target name="compile"
        description="Compiles the XML report design and produces the
                     .jasper file.">
<taskdef name="jrc"
          classname="net.sf.jasperreports.ant.JRAntCompileTask">
   <classpath refid="classpath" />
</taskdef>
<jrc destdir=".">
   <src>
      <fileset dir=".">
         <include name="**/*.jrxml" />
      </fileset>
   </src>
```

```
        <classpath refid="classpath" />
    </jrc>
    </target>
    <target name="view"
            description="Launches the report viewer to preview the
                         report stored in the .jrprint file.">
        <java classname="net.sf.jasperreports.view.JasperViewer"
              fork="true">
            <arg value="-F{file.name}.jrprint" />
            <classpath refid="classpath" />
        </java>
    </target>
</project>
```

执行这个新的 ANT 目标，我们将看到一个如下图所示的窗口：



对，就是它！这就是我们成功创建的第一份报表。

在 web 浏览器中显示报表

前面我们讨论了怎样创建报表，并用 JasperReports 的本地格式把它保存到磁盘。下面我们介绍如何在 web 浏览器里显示报表，这需要借助于 Servlet API，如下面的例子所示：

```
package net.ensode.jasperbook;
import java.io.IOException;
import java.io.InputStream;
import java.io.PrintWriter;
```

```
import java.io.StringWriter;
import java.util.HashMap;
import javax.servlet.ServletException;
import javax.servlet.ServletOutputStream;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import net.sf.jasperreports.engine.JREmptyDataSource;
import net.sf.jasperreports.engine.JRException;
import net.sf.jasperreports.engine.JasperRunManager;
public class FirstReportSendToBrowserServlet extends HttpServlet
{
   protected void doGet(HttpServletRequest request, HttpServletResponse
                                                              response)
   throws ServletException, IOException
   {
      ServletOutputStream servletOutputStream = response.
                                                    getOutputStream();
      InputStream reportStream =getServletConfig().getServletContext()
                     .getResourceAsStream("/reports/FirstReport.jasper");
      try
      {
         JasperRunManager.runReportToPdfStream(reportStream,
                                               servletOutputStream,
                                               new HashMap(),
                                               new JREmptyDataSource());
         response.setContentType("application/pdf");
         servletOutputStream.flush();
         servletOutputStream.close();
      }
      catch (JRException e)
      {
         // display stack trace in the browser
         StringWriter stringWriter = new StringWriter();
         PrintWriter printWriter = new PrintWriter(stringWriter);
         e.printStackTrace(printWriter);
         response.setContentType("text/plain");
         response.getOutputStream().print(stringWriter.toString());
      }
   }
}
```

由于 Web 浏览器不能显示 JasperReports 的本地格式的报表(至少在不使用 applet 的情况下是这样)，我们必须把报表导出为浏览器可以识别的格式。JasperReports 允许我们把报表导出为 PDF 等多种格式，由于 PDF 是最流行的格式之一，我们在此处的示例中就选择它作为导出格式。

上例中的 Servlet 调用了静态方法 JasperRunManager.runReportToPdfStream()，其格式为：

runReportToPdfStream (java.io.InputStream inputStream,
java.io.OutputStream outputStream,
java.util.Map parameters,
JRDataSource dataSource)

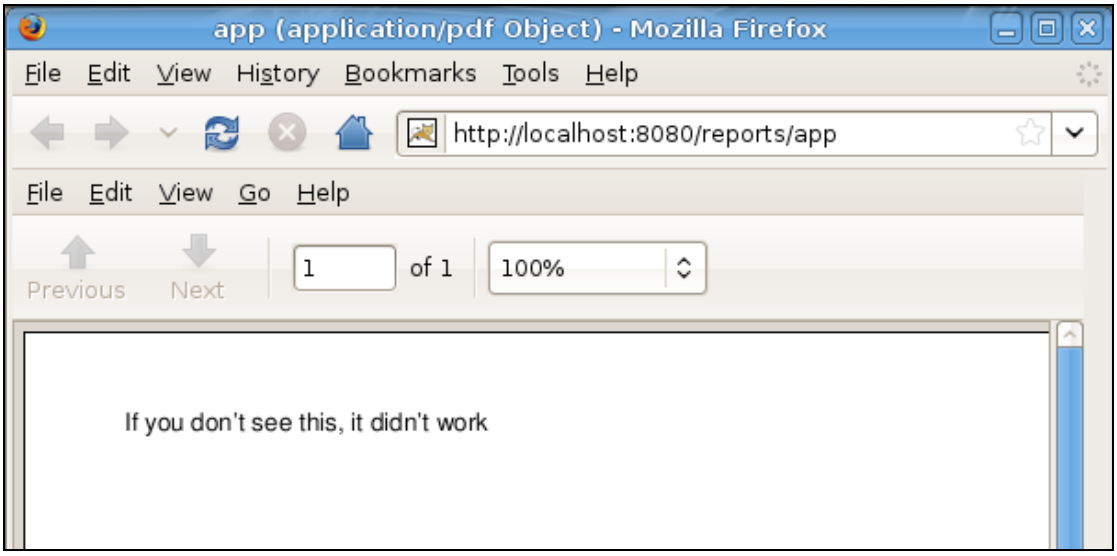
要想在浏览器中显示报表，我们需要在第一个参数中以流的形式传入二进制报表模板，也就是 Jasper 文件。我们可以通过调用 javax.servlet.ServletContext.getResourceAsStream()方法达到这个目的，它需要传入一个包含 Jasper 文件位置信息的 String 作为参数。此方法返回一个 java.io.InputStream 实例，我们可以把它作为 JasperRunManager.runReportToPDFStream()方法的第一个参数。

JasperRunManager.runReportToPDFStream()方法需要一个 java.io.OutputStream()实例来对编译后的报表执行写操作。我们可以简单地使用 Servlet 的缺省的输出流，可以通过调用 javax.servlet.http.HttpServletResponse.getOutputStream()方法得到它。

JasperRunManager.runReportToPDFStream()方法的另外两个参数是 java.util.Map 和 JRDataSource，前者用于传递报表的任意参数，后者用于以 net.sf.jasperreports.engine.JRDataSource 形式传递数据。这个例子中，我们不需要传递任何参数和数据，所以使用一个空的 HashMap 和 JREmptyDataSource。

为了确保浏览器正确地显示报表，我们必须把内容类型设置为 application/pdf，通过调用 javax.servlet.http.HttpServletResponse.setContentType()方法可以完成这件事。

最后的示例代码需要部署到一个 Servlet 容器中，我们可以通过 ANT 脚本来自动地完成这一过程。该脚本作为本书源代码的一部分，可以从 http://www.packtpub.com/files/code/8082_Code.zip 下载得到。下面的屏幕截图显示了浏览器中的 PDF 格式报表：



JRXML 报表模板的元素

在前面的袋子中，我们用 JRXML 报表模板的<detail>元素来生成了一个显示静态文本的报表，<detail>元素表示报表的主区域。当然，JRXML 模板还可以包含许多其它元素，用来在报表中显示次级辅助数据或完成其它一些任务，如：导入 Java 包、控制数据在报表中如何显示等。

下面就来介绍一下<jasperReporot>根元素的全部子元素。除个别已特别指出的元素外，大部分的元素都可以在模板中使用多次。

<property>

此元素用来给报表模板输入任意的信息。

<property name="someproperty" value="somevalue"/>

加载了报表的 Java 应用程序通过调用 JasperReport.getProperty()方法可以载入这些属性。JRXML 模板可以包含零个或多个<property>元素。

<import>

此元素用于导入个别的 Java 类或完整的包。

<import value="java.util.HashMap"/>

<template>

报表的样式可以定义在独立的报表模板中，这样可以方便地在报表之间进行重用。这一机制类似于 HTML 中的级联样式表可定义在独立的 CSS 文件中。报表样式模板可以用 XML 文件进行定义。按照惯例，这是一个 JRTX 文件。另外，报表样式模板也可以用一个实现了 net.sf.jasperreports.engine.JRTemplate 接口的类的实例进行定义，此方法比较少见。

<template>"my_template.jrtx"</template>

<style>

此元素用于定义报表元素的风格，可设置字体、尺寸、背景色、前景色、等等。其它的大部分报表元素都有一个 style 属性，用于指定其特定的样式。<stype>元素有一个 isDefault 属性，可用于指定该样式为缺省样式。当其它元素不特殊指定其 style 属性时，就使用这个缺省样式。

<style name="Arial_Normal" isDefault="true"
fontName="Arial" fontSize="10"
isBold="false" isItalic="false"

```
isUnderline="false" isStrikeThrough="false"/>
```

<subDataset>

报表模板中可以定义图表和交叉表，<subDataset>元素用于在报表中间接地为它们提供数据。

```
<subDataset name="Client_Data">
  <parameter name="Client" class="java.lang.String"/>
  <queryString>
    <![CDATA[SELECT foo, bar, temp
             FROM some_table
             WHERE client_code = $P{Client}]]>
  </queryString>
  <field name="foo" class="java.lang.String"/>
  <field name="bar" class="java.lang.String"/>
  <field name="temp" class="java.lang.String"/>
</subDataset>
```

subdatasets 只能被交叉表和图表引用。

<parameter>

此元素用于定义报表参数，参数值通过调用 JasperReports API 中的相应方法以 java.uitl.Map 来提供。

```
<parameter name="SomeParameter"
           class="java.lang.String"/>
```

<queryString>

此元素用于定义从数据库获取数据的 SQL 查询。

```
<queryString>
```

```
    <![CDATA[SELECT column_name FROM table_name]]>
  </queryString>
```

JRXML 模板可以包含零个或一个<queryString>元素。如果我们希望在报表模板中嵌入一个 SQL 查询，就必需使用此元素。

<field>

此元素用于把从数据源或查询获取的数据映射到报表模板。Field 可以嵌入到报表表达式中，从而获得所需的输出。

```
<field name="FieldName" class="java.lang.String"/>
```

<sortField>

此元素用于对报表中的数据按照 name 属性进行排序。排序可以能过 order 属性来指定是升序或降序。如果不指定 order，缺省使用升序。

```
<sortField name="BirthDate" order="Descending"/>
```

一个 JRXML 模板中，可以有一个或多个对应于报表模板中域的<sortField>元素。

<variable>

报表中多次使用的表达式可以赋值给变量，从而简化模板。

```
<variable name="VariableName"
          class="java.lang.Double"
          calculation="Sum">
  <variableExpression>
    $F{FieldName}
  </variableExpression>
</variable>
```

<filterExpression>

此元素用于在报表中对数据进行过滤。

```
<filterExpression>
<![CDATA[$F{status}.equals("active") ?
                Boolean.TRUE :Boolean.FALSE]]>
</filterExpression>
```

如果嵌在<filterExpression>元素中的表达式的结果为 Boolean.TRUE，数据源中的当前行就被包含在报表中。如果其结果为 Boolean.FALSE 或 null，数据源的当前行就会被滤掉。

请注意，此元素的初衷是用于不支持一般过滤功能的数据源类型，比如 CSV 数据源。

一个报表模板中，可以有零个或一个<filterExpression>元素。

<group>

此元素用于对连续的纪录进行分组，分组的依据是数据源的一些共同特征。

```
<group name="GroupName">
    <groupExpression>
        <![CDATA[$F{FieldName}]]>
    </groupExpression>
</group>
```

JRXML 模板可以包含零个或多个<group>元素。

<background>

此元素定义页的背景，它对报表的所有页有效。它可以显示图片、文本或水印。

```
<background>
```

```
<band height="745">
    <image scaleImage="Clip" hAlign="Left"
           vAlign="Bottom">
        <reportElement x="0" y="0" width="160"
                       height="745"/>
        <imageExpression>"image.gif"</imageExpression>
    </image>
</band>
</background>
```

一个 JRXML 模板中，最多可以包含一个<background>元素。

<title>

这是报表的标题，它只在报表起始处显示一次。

```
<title>
    <band height="50">
        <staticText>
            <reportElement x="180" y="0" width="200"
                           height="20"/>
            <text>
                <![CDATA[Title]]>
            </text>
        </staticText>
    </band>
</title>
```

<pageHeader>

此元素定义页眉，它在报表每一页的起始处打印。

```
<pageHeader>
    <band height="20">
```

```
        <staticText>
            <reportElement x="180" y="30" width="200"
                           height="20"/>
            <text>
                <![CDATA[Page Header]]>
            </text>
        </staticText>
    </band>
</pageHeader>
```

一个 JRXML 模板中，最多可以包含一个<pageHeader>元素。

<columnHeader>

此元素定义列眉的内容。如果报表只有一列，此元素将被忽略掉。

```
<columnHeader>
    <band height="20">
        <staticText>
            <reportElement x="180" y="50" width="200"
                           height="20"/>
            <text>
                <![CDATA[Column Header]]>
            </text>
        </staticText>
    </band>
</columnHeader>
```

如果在 JRXML 模板中使用此元素，那么<columnHeader>元素的个数必须和列数相同。

<detail>

此元素定义报表的 detail 段，<detail>段对于报表数据源的每条纪录都会重复显示。

```
<detail>
    <band height="20">
        <textField>
            <reportElement x="10" y="0" width="600"
                           height="20"/>
            <textFieldExpression class="java.lang.String">
                <![CDATA[$F{FieldName}]]>
            </textFieldExpression>
        </textField>
    </band>
</detail>
```

一个 JRXML 模板中，最多只能包含一个<detail>元素。通常情况下，大多数据报表模板都会包含一个<detail>元素，这是显示报表主要数据的地方。

<columnFooter>

此元素定义列脚内容。如果报表只有一列，此元素将被忽略掉。

```
<columnFooter>
    <band height="20">
        <staticText>
            <reportElement x="0" y="0" width="200"
                           height="20"/>
            <text>
                <![CDATA[Column Footer]]>
            </text>
        </staticText>
    </band>
</columnFooter>
```

一个 JRXML 模板中，可以包含零个或多个<columnFooter>元素，如果在模板中使用该元素，那么<columnFooter>元素的数量必需和列数相同。

<pageFooter>

此元素定义页脚，它在报表每一页的末尾处打印。

```
<pageFooter>
    <band height="20">
        <staticText>
            <reportElement x="0" y="5" width="200"
                           height="20"/>
            <text>
                <![CDATA[Page Footer]]>
            </text>
        </staticText>
    </band>
</pageFooter>
```

一个 JRXML 模板中，可以包含零个或一个<pageFooter>元素。

<lastPageFooter>

此元素定义的数据显示在末页的页脚，优先于<pageFooter>元素定义的页脚内容。

```
<lastPageFooter>
    <band height="20">
        <staticText>
            <reportElement x="0" y="5" width="200"
                           height="20"/>
            <text>
                <![CDATA[Last Page Footer]]>
```

```
            </text>
        </staticText>
    </band>
</lastPageFooter>
```

一个 JRXML 模板中，最多可以包含一个<lastPageFooter>元素。

<summary>

此元素仅在报表末尾打印一次。

```
<summary>
    <band height="20">
        <staticText>
            <reportElement x="0" y="5" width="200"
                           height="20"/>
            <text>
                <![CDATA[Summary]]>
            </text>
        </staticText>
    </band>
</summary>
```

一个 JRXML 模板中，最多可以包含一个<summary>元素。

<noData>

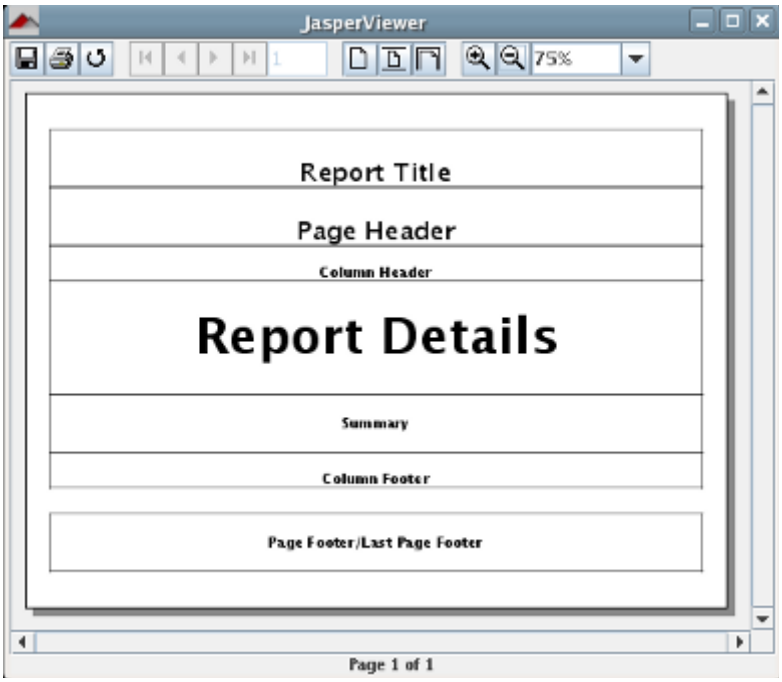
<noData>元素可以用来控制当数据源中没有数据时，报表中生成的内容。

```
<noData>
    <band height="20">
        <staticText>
            <reportElement x="0" y="5"
                           width="200" height="20"/>
```

```
            <text>
                <![CDATA[No data found]]>
            </text>
        </staticText>
    </band>
</noData>
```

和<detail>元素一样，前面讨论的大部分元素也都包含一个单独的<band>元素作为其唯一的子元素。我们将在后续的章节中详细讨论这个特殊的<band>元素。

从下面的截图中，我们可以直观地看到一个报表中的不同区域所处的相应位置：



正如我们所见，页脚标签为 Page Footer/Last Page Footer。如果报表的 JRXML 模板包含一个<lastPageFooter>元素，它将在最后一页代替<pagefooter>元素进行显示。在一些特殊情况下，需要特别指出的是，如果报表只有一页，并且在模板中既有<pageFooter>元素，又有<lastPageFooter>元素，那么，<lastPageFooter>的内容将被显示。这种情况下，<pageFooter>元素永远不会被显示。

此外，我们要记住<columnHeader>和<columnFooter>元素只有报表多于一列时才会被显示。如何在报表中添加列的详细讨论，我们将在第 6 章中进行。

小结

本章我们学习了通过编辑 XML 文件来创建 JRXML 报表模板，以及通过 JasperReports 提供的工具来预览这些模板。同时我们还讨论了通过编码方式或自定义的 ANT 任务来编译 JRXML 模板。

编译完报表后，我们通过调用 JasperFillManager 类提供的相应方法，把数据填充到报表中。还用 JaserViewer 工具对生成的要地 JasperReports 格式报表进行预览。我们还了解了 JRXML 模板中不同的报表区域。

最后，我们创建了基于 Web 的报表，并在 Web 浏览器中显示了生成的报表。