Explore ways to render SAS datasets through Web

Pascal Verrecchia

Sokaris Informatique

Abstract

In a modern world where web is becoming increasingly significant personally and professionally, more and more, **web applications** are replacing old-fashioned softwares installed on computer.

Main benefits are an easy and a common way to access for end-users and the possibility to externalize your IT in dedicated infrastructure like **Cloud Computing**.

With that in mind and focusing on study biostatistical programming needs and how it could be possible to offer full services through web technologies, an important first question will have to be raised: What are the technical possibilities to **render a SAS dataset in a web-browser** ? Is that even possible ?

The aim of these poster and paper is to answer to this question, analyze different ways to succeed and compare them.

Using Java

A standard web application is composed of multiple components and most of the time at least the following ones:

- ✓ User interface part (GUI) displayed in a web browser (like Internet Explorer, Google Chrome or Firefox).
- ✓ Back-end server (web server) that will contain the business logic. It is the brain of the application.
- ✓ Database to store the data needed for the application to work.

To be able to render SAS datasets in the user interface, we first need to be able to read the SAS dataset data with the back-end server.

We decided to use Java language[1] to perform that.

Why Java ?

Because it is a **polyvalent** programmatic language, very **popular** for server side management when it comes to work on web application development.

Moreover, there are a lot of **bridges** between SAS and Java, like Java material provided by SAS (jar files) or SAS Stored Process (exposed as web services).

Four methods have been tested to be able to read SAS datasets with Java. Two JDBC[2] drivers provided by SAS :

- ✓ Using SAS/SHARE[3]
- ✓ Using IOM[4] server

And two Open source Java libraries:

- ✓ SassyReader[5]
- ✓ Parso[6]

The next parts will describe the results of tests performed.

SassyReader

SassyReader is an Open source Java library that simply offer a way of reading the data that is stored in SAS dataset files.

It is a portage of the **SAS7BDAT** project that have been implemented in R language.

Though a unique interface, you can precisely define what you want to retrieve from the SAS dataset : structure, columns type, row data, …

Unfortunately, SassyReader **was not working correctly** when we test it on multiple **recent SAS datasets** (and more especially datasets created with SAS release v9.2, v9.3 and v9.4).

Another concern is that SassyReader doesn't seem to be updated and maintained anymore.

To conclude about SassyReader, it is an interesting project but it doesn't fit all the requirements we have and is not the best solution.

---

[1] https://www.oracle.com/java/index.html
[2] http://www.oracle.com/technetwork/java/javase/jdbc/index.html
[3] http://support.sas.com/software/products/connect_share/index.html
[4] http://support.sas.com/rnd/itech/doc9/overview/iom_server.html
[5] http://kasper.eobjects.org/2011/06/sassyreader-open-source-reader-of-sas.html
[6] http://grepcode.com/snapshot/repo1.maven.org/maven2/com.ggasoftware/parso/1.2.1/

Here is an example of Java code using SassyReader:

```
File file = new File("C:/testsDatasets/dataset.sas7bdat");

CountingSasReaderCallback callback = new CountingSasReaderCallback(true);

//Read SAS dataset
SasReader sasReader = new SasReader(file);
sasReader.read(callback);

List<Object[]> data = callback.getSampleRows();

//Retrieve data
for (Object[] objs : data) {
    for (int i = 0; i < objs.length + 1; i++) {
        Object obj = objs[i];
        System.out.println(obj.toString());
    }
}
```

Using SAS/SHARE

SAS/SHARE software is a **data server** that allows multiple users to gain simultaneous access to SAS files.

SAS provides Java DataBase Connectivity (**JDBC**) drivers to connect and interacts with SAS datasets stored on SAS/SHARE servers.

Using the com.sas.net.sharenet.ShareNetDriver Java class, you can **create a connection** to your SAS/SHARE server and use **SQL queries** to retrieve information from your SAS datasets.

With access to the complete features of JDBC, it is easy to create Java programs to store SAS dataset data in Java objects.

The main interest of this method is that interacting using SQL query is very convenient and offer flexibility.

The main concerns about using JDBC drivers through SAS/SHARE is, of course, the need to have SAS/SHARE in your IT environment (need a specific license). Another minor one is using SAS resources. SAS process is expensive (license price regarding CPU usage). If it is possible to read SAS datasets data in Java without consuming SAS process, it would be quite better.

Here is an example of Java code using SAS/SHARE JDBC driver:

```
try {
    //JDBC URL connection
    String jdbcUrl = "jdbc:sharenet://<your_server_host>:" +
            "<your_server_port>";

    Properties prop = new Properties();
    prop.setProperty("shareUser", "your_user");
    prop.setProperty("sharePassword", "your_password");
    prop.setProperty("librefs", "saslib c:\\testsDatasets';");

    ShareNetDriver snd = new ShareNetDriver();
    Connection conn = snd.connect(jdbcUrl, prop);

    //Create and execute the query to read SAS datasets
    Statement stmt = conn.createStatement();
    ResultSet rs = stmt.executeQuery("SELECT sup_id, " +
            "sup_name FROM saslib.suppliers");

    //Browse the results and store them in Java
    while (rs.next()) {
         Double id = rs.getDouble(1);
         String name = rs.getString(2);
         System.out.println(id + " " + name);
    }

    stmt.close();
    conn.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

Using IOM server

Integrated Object Model (IOM) in SAS Integration Technologies provides distributed object interfaces to SAS software features. There are four types of IOM servers:

- ✓ SAS Workspace server
- ✓ SAS Stored Process server
- ✓ SAS OLAP server
- ✓ SAS Metadata server

Following the same logic than what has been presented for SAS/SHARE in the previous part, SAS provides Java DataBase Connectivity (**JDBC**) drivers to connect and interacts with SAS datasets through IOM servers.

Using the com.sas.rio.MVADriver Java class, you can **create a connection** to your IOM server and use **SQL queries** to retrieve information from your SAS datasets.

With access to the complete features of JDBC, it is easy to create Java programs to store SAS dataset data in Java objects.

The main interest of this method, as for SAS/SHARE is that interacting using SQL query is very convenient and offer flexibility.

The main concerns about using JDBC drivers through an IOM server is also the need to have an IOM server in your IT environment (need a specific license). Another minor one is using SAS resources. SAS process is expensive (license price regarding CPU usage). If it is possible to read SAS datasets data in Java without consuming SAS process, it would be quite better.

Here is an example of Java code using IOM JDBC driver:

```
try {
    //JDBC URL connection
    String jdbcUrl = "jdbc:sasiom://<your_server_host>:" +
            "<your_server_port>";

    Class.forName("com.sas.rio.MVADriver");
    Properties prop = new Properties();
    prop.setProperty("shareUser", "your_user");
    prop.setProperty("sharePassword", "your_password");
    prop.setProperty("librefs", "saslib c:\\testsDatasets';");

    Connection conn = DriverManager.getConnection(jdbcUrl,
            prop);

    //Create and execute the query to read SAS datasets
    Statement stmt = conn.createStatement();
    ResultSet rs = stmt.executeQuery("SELECT sup_id, " +
            "sup_name FROM saslib.suppliers");

    //Browse the results and store them in Java
    while (rs.next()) {
         Double id = rs.getDouble(1);
         String name = rs.getString(2);
         System.out.println(id + " " + name);
    }

    stmt.close();
    conn.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

Parso

Parso is an Open source (GNU License 3) **Java lightweight library** to parse SAS dataset files.

It offers a lot of functionalities to access to the SAS dataset properties like row and columns count, OS type, compression methods, SAS Release version, …

To read the rows, you have two main methods:

- ✓ Read all rows at the same time and store everything in Java objects
- ✓ Read data row by row

In case you're working on **big datasets** (more than 1 GB), you could have **troubles with Java memory** if you try to read all rows in one step.

We have **successfully tested** Parso with datasets generated from **different OS** and from **several SAS Releases** (and more especially datasets created with SAS release v9.2, v9.3 and v9.4).

Regarding what have been said previously, one of the main benefit of Parso is that it doesn't use any SAS resources.

The main concern that has been raised testing Parso is the lack of methods to load only part of the SAS dataset. There is nothing to retrieve rows with delimiters (from line… to line…) so it means it is not possible to perform lazy loading (load data asynchronously) out of the box.

It would be interesting to improve this Java library to add this feature.

Here is an example of Java code using Parso:

```
File f = new File("C:/testsDatasets/dataset.sas7bdat");

try {
    InputStream is = new FileInputStream(f);
    SasFileReader sfr = new SasFileReader(is);
    SasFileProperties sfp = sfr.getSasFileProperties();
    System.out.println("Name : " + sfp.getName());
    System.out.println("File type : " + sfp.getFileType());
    System.out.println("Row count : " + sfp.getRowCount());
    System.out.println("Row length : " + sfp.getRowLength());
    System.out.println("OS Name : " + sfp.getOsName());
    System.out.println("OS Type : " + sfp.getOsType());
    System.out.println("Page count : " + sfp.getPageCount());
    System.out.println("Page length : " + sfp.getPageLength());
    System.out.println("Creation date : " + sfp.getDateCreated());
    System.out.println("Modification date : " + sfp.getDateModified());
    System.out.println("Server type : " + sfp.getServerType());
    System.out.println("SAS Release : " + sfp.getSasRelease());
    System.out.println("Columns count : " + sfp.getColumnsCount());
    System.out.println("Compression method : " + sfp.getCompressionMethod());
    List<Column> columns = sfr.getColumns();
    int i = 1;
    System.out.println("Column list : ");
    for (Column c : columns) {
        System.out.println("  Column " + i + " : ");
        i++;
        System.out.println("    Format : " + c.getFormat());
        System.out.println("    Id : " + c.getId());
        System.out.println("    Label : " + c.getLabel());
        System.out.println("    Length : " + c.getLength());
        System.out.println("    Name : " + c.getName());
    }

    Object[][] allRows = sfr.readAll();

    for (int j = 0; j <= allRows.length -1; j++) {
        Object obj = allRows[j];
        System.out.println(obj);
    }

} catch (Exception e) {
    e.printStackTrace();
}
```

From Java to User Interface

When SAS datasets data will be stored in Java objects, it will be necessary to communicate with client side of your web application (interface) to load these data and display it.

Most of the web application interfaces are now implemented using **HTML 5 and JavaScript framework** (like AngularJS[7]). The most common way to do that is to convert your Java data in **JSON**[8] (JavaScript Object Notation) format, which is an open standard format pretty well adapted for JavaScript environment. There are multiple ways to convert your data from Java to JSON, like **Jackson**[9] library.

[7] https://angularjs.org/
[8] http://json.org/json-fr.html
[9] http://jackson.codehaus.org/