

Санкт-Петербургский государственный университет

кафедра системного программирования

Свитков Сергей Андреевич

# Реализация библиотеки для потоковой обработки .xlsx файлов

Курсовая работа

Научный руководитель:  
ст. преп к.т.н. Литвинов Ю. В.

Санкт-Петербург  
2018

# Оглавление

<b>Введение</b>	<b>3</b>
<b>1. Обзор существующих решений</b>	<b>4</b>
1.1. Apache POI . . . . .	4
1.2. SJXLSX . . . . .	4
1.3. Итоги обзора . . . . .	6
<b>2. Постановка задачи</b>	<b>7</b>
<b>3. Анализ формата XLSX</b>	<b>8</b>
3.1. Workbook . . . . .	8
3.2. Worksheet . . . . .	9
<b>4. Реализация</b>	<b>10</b>
4.1. Алгоритм . . . . .	10
4.2. Архитектура . . . . .	10
<b>5. Апробация</b>	<b>13</b>
<b>Заключение</b>	<b>17</b>
<b>Листинги</b>	<b>18</b>
<b>Список литературы</b>	<b>22</b>

# Введение

В современном мире большой популярностью пользуются многопользовательские веб-приложения. Приложения такого рода могут использоваться для самых разнообразных целей — от совместного редактирования документов несколькими пользователями до анализа различной статистики операторами связи.

Одну из категорий веб-приложений представляют проекты в сфере телекоммуникаций и биллинга. Такие приложения используются операторами связи для анализа различной статистики по действиям абонентов: перемещения между зонами роуминга, количество входящих/исходящих вызовов, и т.д.

Для формирования отчётов требуется формат представления данных, предоставляющий строгое структурирование. К таким можно отнести JSON [4], XML [8], XLSX [7]. Однако, следует принять во внимание, что отчетность или статистика, представленная в таком формате, может использоваться как при взаимодействии различных компонент приложения или различных приложений, так и для анализа человеком. Преимущество формата XLSX в том, что для просмотра таких файлов существуют общеизвестные решения (Microsoft Excel, OpenOffice Calc, Google Sheets).

Таким образом, возникает необходимость в библиотеке, которая позволила бы формировать документы формата XLSX. Следует отметить, что, поскольку объем данных не ограничен, а веб-приложения являются многопользовательскими, то необходимо формировать файл в потоковом режиме, то есть, держать в оперативной памяти только ограниченное количество данных.

Исходя из сформулированных требований было принято решение проанализировать существующие решения в данной области и, в случае отсутствия подходящей реализации, создать свою библиотеку для решения подобных задач.

# 1. Обзор существующих решений

Задача формирования документов формата XLSX не является новой, имеется ряд существующих библиотек.

## 1.1. Apache POI

Библиотека Apache POI [1] (далее — просто POI) предоставляет средства как для формирования, так и для чтения файлов формата XLSX. До версии 3.8 в библиотеке отсутствовала поддержка потоковой обработки файлов. Несмотря на то, что, начиная с версии 3.8, появилась поддержка потоковой обработки, некоторые проблемы с использованием оперативной памяти остались. Так, некоторые операции над документами всё равно можно проводить только храня целый документ в памяти. К недостаткам библиотеки можно отнести отсутствие возможности задать условия автоматического создания новых страниц в документе, а так же отсутствие полной документации и примеров использования.

На момент начала данной работы не было известно о том, что в POI реализована поддержка потоковой обработки документов, что во многом и послужило мотивацией для постановки задачи о реализации своей библиотеки. После анализа последней версии библиотеки было решено реализовать собственный алгоритм потокового формирования документов и сравнить полученную реализацию с POI по производительности.

## 1.2. SJXLSX

Библиотека SJXLSX [6]— проект с открытым исходным кодом. Документации к проекту крайне мало, к тому же, последнее обновление в репозитории было сделано в 2015м году.

Был проведен эксперимент по проверке реализации на количество затрачиваемой операционной памяти. С использованием данной библиотеки был сгенерирован файл из 100 тысяч строк, запись произво-

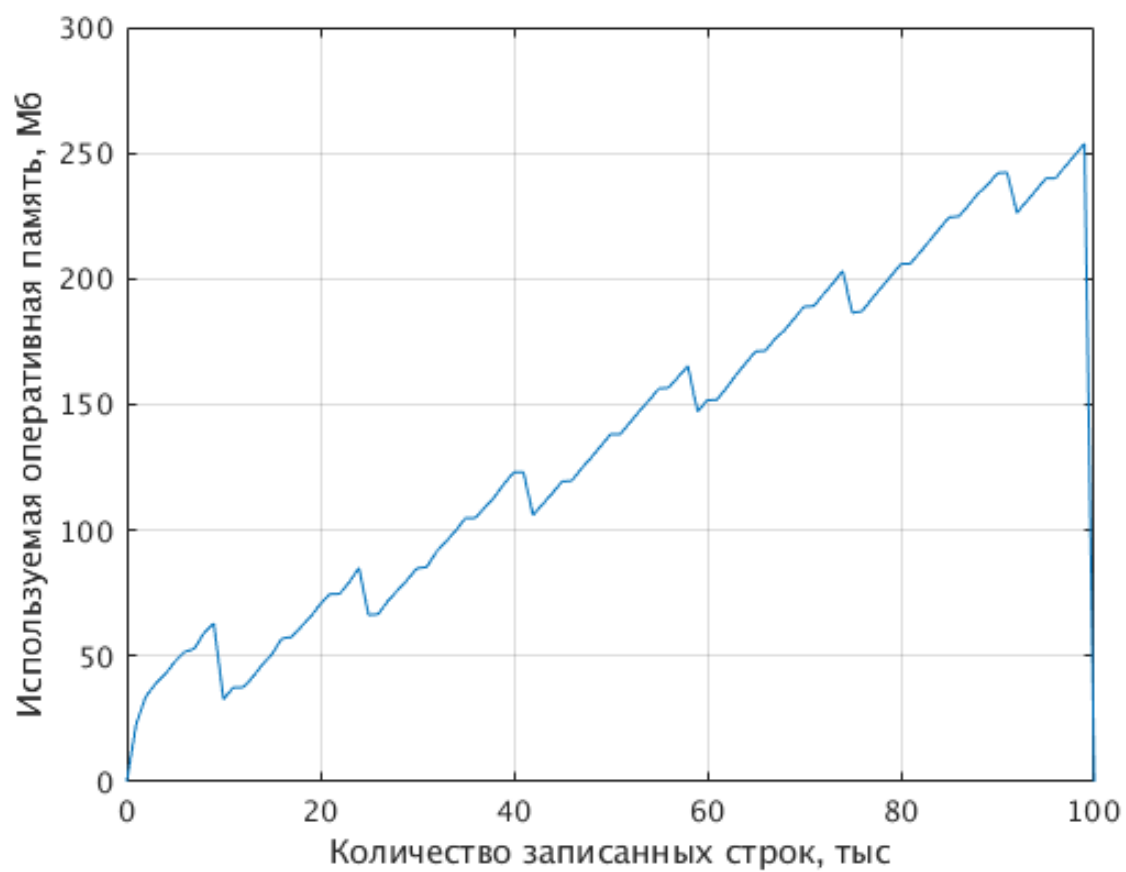


Рис. 1: График потребления оперативной памяти библиотекой SJXLXS при генерации файла XLSX размером в 100 тысяч рядов

дилась фрагментами по 1 тысячи строк с интервалом в 5 секунд. Объем используемой памяти измерялся с помощью JConsole. Из графика (рис. 1), полученного в результате эксперимента, можно сделать выводы о том, что по количеству используемой операционной памяти при генерации файла данная реализация является крайне неэффективной.

### **1.3. Итоги обзора**

Исходя из результатов обзора было принято решение о реализации библиотеки для потоковой генерации файлов и последующем сравнении её с рассмотренными существующими решениями.

## 2. Постановка задачи

Целью данной работы является разработка библиотеки для потоковой обработки файлов формата `xlsx` и сравнение её с существующими реализациями. Для достижения этой цели были поставлены следующие задачи:

- сформулировать подход, который будет использоваться для реализации библиотеки;
- реализовать библиотеку;
- провести апробацию полученной реализации;
- сравнить полученную реализацию с существующими библиотеками по потреблению RAM (оперативной памяти) и скорости работы при создании документа;
- разместить исходный код и примеры использования библиотеки на `github`;

### 3. Анализ формата XLSX

Для реализации библиотeki потребовалось изучить структуру формата XLSX.

Формат XLSX был создан в декабре 2006 года при участии Microsoft, Ecma, ISO/IEC. К сожалению, документации по стандарту MS-XLSX крайне мало, в открытом доступе можно найти два стандарта: ISO/IEC 29500 [3] и ECMA-376 [2]. Эти документы имеют крайне большой объем (порядка 7 тысяч страниц), и на их изучение ушло бы большое количество времени.

Для формата XLSX так же существует другой стандарт, OOXML [5]. В отличие от MS-XLSX, данный стандарт довольно хорошо документирован, поэтому было принято решение формировать документы в соответствии с данным стандартом.

Формат XLSX представляет собой ZIP-архив с XML файлами. Его структура (рис. 2) представляет собой следующий набор файлов и директорий:

- `Content_Types.xml` — типы контента в архиве и пути к ним;
- `_rels` — зависимости между файлами внутри архива;
- `docProps` — метаданные: имя автора, дата создания, ...;
- `xl` — директория с основными файлами архива: `workbook`, страницы, стили, таблицы;

Рассмотрим более подробно основные компоненты архива: `Workbook` и `Worksheet`.

#### 3.1. Workbook

`Workbook` представляет собой XML-файл, который не содержит данных файла, но содержит следующие мета-данные: ссылки на отдельные `Worksheet` и их свойства. В работе приведён пример файла `workbook.xml` (листинг 1). Контент документа содержится непосредственно в `Worksheets`.



```

example/
├── [Content_Types].xml
├── _rels
└── xl
    ├── drawings
    │   └── drawing1.xml
    ├── _rels
    │   └── workbook.xml.rels
    ├── sharedStrings.xml
    ├── styles.xml
    ├── workbook.xml
    └── worksheets
        ├── _rels
        │   └── sheet1.xml.rels
        └── sheet1.xml

```

6 directories, 8 files

Рис. 2: Структура формата XLSX

## 3.2. Worksheet

Worksheet содержат данные, из которых и состоит документ. Worksheet может иметь один из следующих форматов: grid, chart, dialog sheet. Наиболее популярным и хорошо задокументированным является grid, рассмотрим его более подробно.

Grid представляет собой "сетку" из "клеток" (cells) с данными. Каждая клетка может содержать какой-то определенный тип данных: числа, булевские переменные, формулы, и т.д.. Для оптимизации использования памяти строковые значения хранятся не в теле самой клетки, а в отдельной части документа. Это позволяет минимизировать дубликацию строк. Пример файла worksheet.xml (листинг 2)

Закончив анализ формата XLSX можно приступить к реализации библиотеки.

## 4. Реализация

В данной секции будут описаны алгоритм работы и архитектура библиотеки. Библиотека была реализована на языке Java с использованием системы сборки Maven. Исходный код реализации библиотеки опубликован на Github в репозитории организации "НТЦ Протей"<sup>1</sup>. Работа велась под учётной записью likeanowl<sup>2</sup>. Код реализации опубликован под лицензией MIT.

### 4.1. Алгоритм

Было принято решение реализовать следующий алгоритм для генерации файлов формата XLSX:

- для каждой страницы создавать временный файл;
- хранить в RAM только один ряд (во время создания);
- после создания добавлять ряд во временный файл страницы;
- после завершения формирования документа — записывать данные из временных файлов в основной файл;
- для экономии дискового пространства сжимать временные файлы.

### 4.2. Архитектура

Для реализации предлагаемого алгоритма была реализована следующая архитектура (рис. 3).

Основными элементами приложения являются классы `WorkbookWriter` и `Worksheet`. `StreamConsumer` служит для хранения списка временных файлов, генерируемых при создании документа, а так же позволяет создать `InputStream` из хранимых файлов. Класс `ConnInfo` служит для

---

<sup>1</sup><https://github.com/protei-rnd/oxml-doc>

<sup>2</sup><https://github.com/likeanowl>

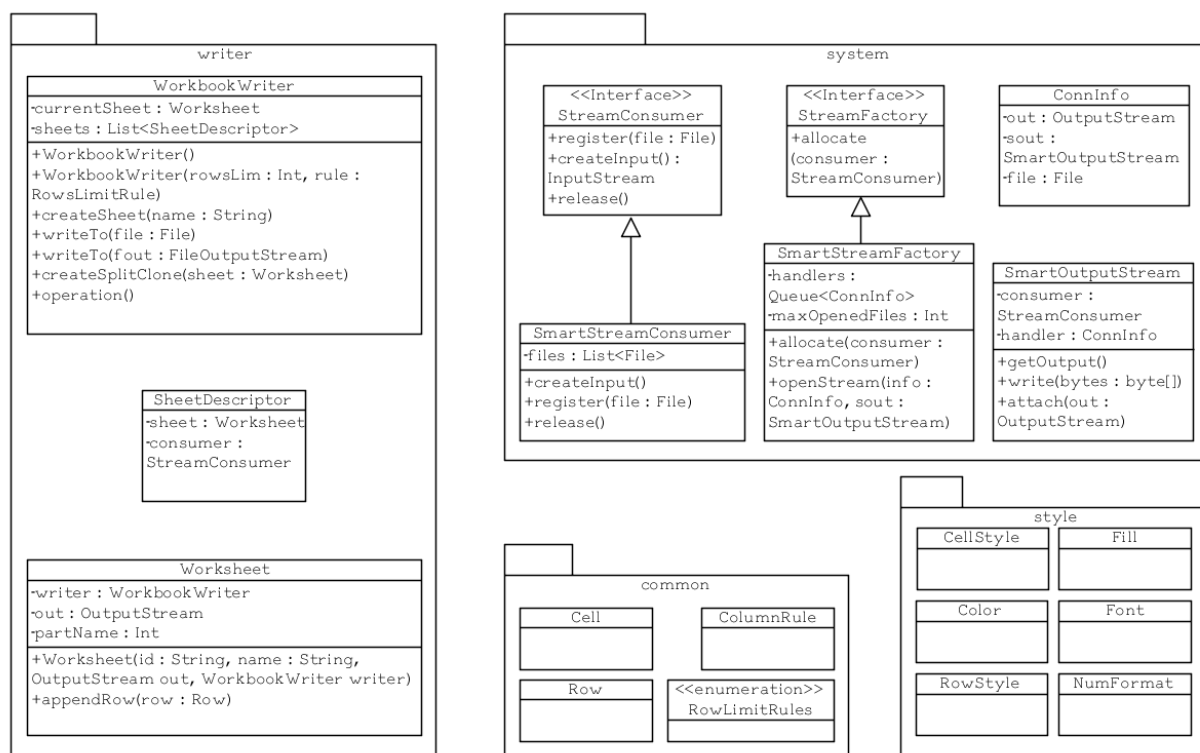


Рис. 3: Архитектура библиотеки, упрощённый вариант

связи SmartOutputStream и StreamConsumer. SmartOutputStream служит для создания OutputStream из временных файлов и записи в них. SheetDescriptor используется связи Worksheet и StreamConsumer. SmartStream используется для управления открытыми SmartOutputStream и создания новых экземпляров этого класса.

Таким образом, полный цикл формирования документа выглядит следующим образом:

- WorkbookWriter создаёт новую страницу, создавая для неё SheetDescriptor, StreamConsumer и SmartOutputStream. Из OutputStream, полученного из SmartOutputStream, создаётся ZipOutputStream для нового временного файла. Созданная страница помечается как готовая к записи.
- После создания страницы к ней можно добавлять ряды. Каждый ряд обрабатывается, применяются стили, если они были заданы для ряда, пишется заголовок файла, если он ещё не был записан. В том случае, когда задано ограничение на максимальное количе-

ство рядов для одной страницы, происходит автоматическое разбиение и в `WorkbookWriter` создается новый экземпляр `Worksheet`.

- После завершения формирования документа, происходит формирование архива с XML файлами. Это делается путём обхода списка `SheetDescriptor`, созданных для страниц документа. Из временных файлов при помощи `StreamConsumer` создается `SequenceInputStream`. Контент из данного потока пишется в соответствующий странице файл архива.

В репозитории с исходным кодом были размещены примеры (листинг 3, листинг 4) использования библиотеки.

## 5. Апробация

Было принято решение сравнить полученную реализацию библиотеки с существующими решениями по следующим метрикам: количество потребляемой при генерации файла оперативной памяти и скорость работы. Исходный код инструмента, который был применен для тестирования, опубликован на Github<sup>3</sup>. Эксперименты проводились на машине с конфигурацией:

- CPU — Intel i7-7700
- RAM — DDR4 2400 MHz, 16 Gb
- SSD — SATA 3, скорость записи 340 MB/s

В ходе тестирования библиотек на количество используемой оперативной памяти формировались файлы размером 100 тысяч рядов. Генерация производилась добавлением порций данных по 1000 строк с интервалом в 5 секунд между итерациями.

Результаты эксперимента (рис. 4, рис. 5) показали, что реализованная в рамках данной работы библиотека по количеству используемой RAM находится на одном уровне с Apache POI, при этом Apache POI и oxml-doc показывают гораздо более хорошие результаты, чем SJXLSX.

Результаты эксперимента по сравнению скорости работы библиотек представлены на рис. 6. Из результатов можно сделать вывод о том, что реализованная в рамках данной работы oxml-doc генерирует файлы быстрее, чем Apache POI.

---

<sup>3</sup><https://github.com/likeanowl/simpletestutil>

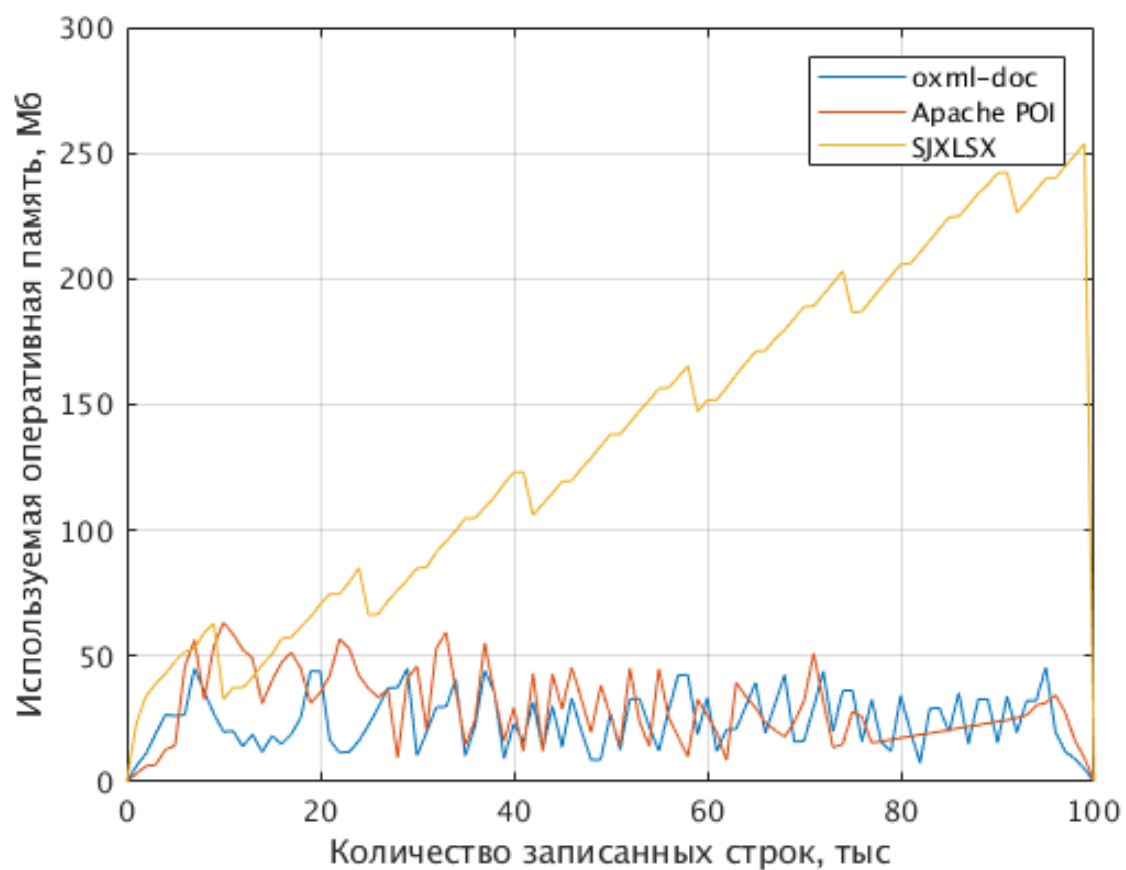


Рис. 4: Сравнение библиотек oxml-doc, Apache POI, SJXLSX по количеству потребляемой оперативной памяти при генерации файлов

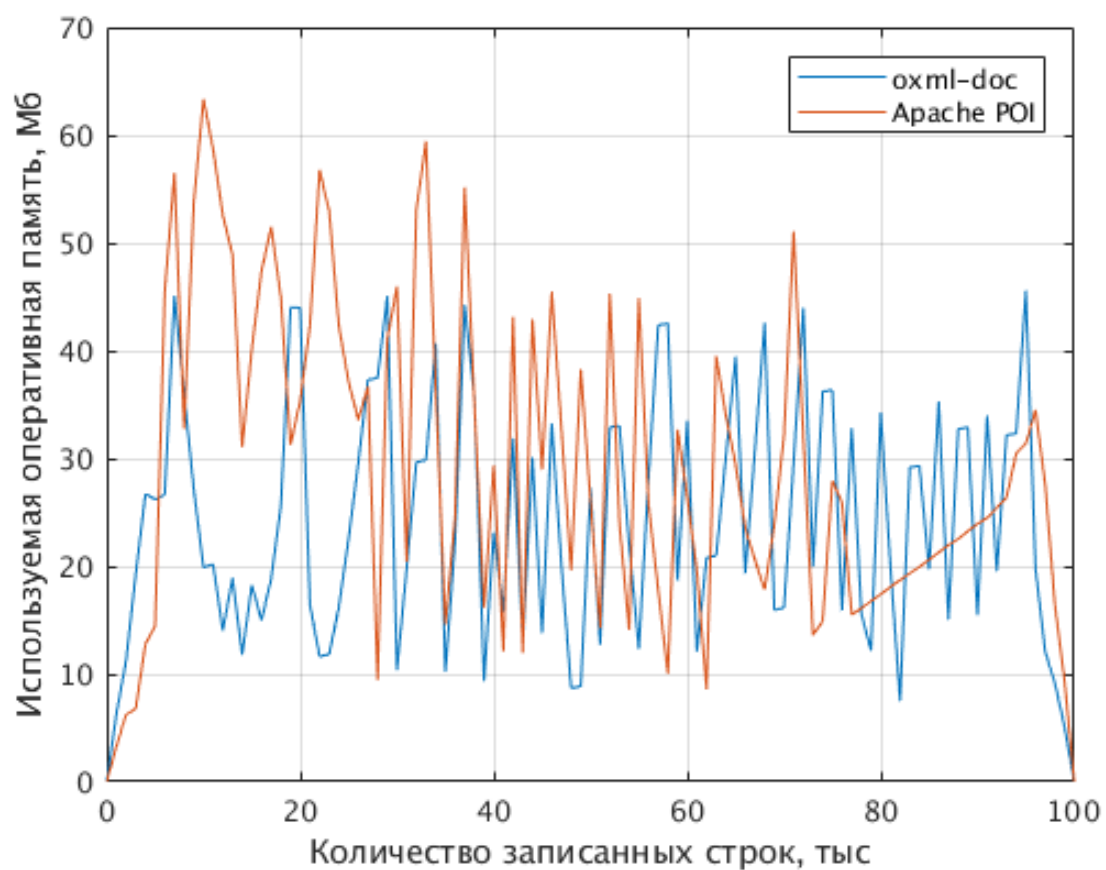


Рис. 5: Сравнение библиотек `oxml-doc`, `Apache POI` по количеству потребляемой оперативной памяти при генерации файлов

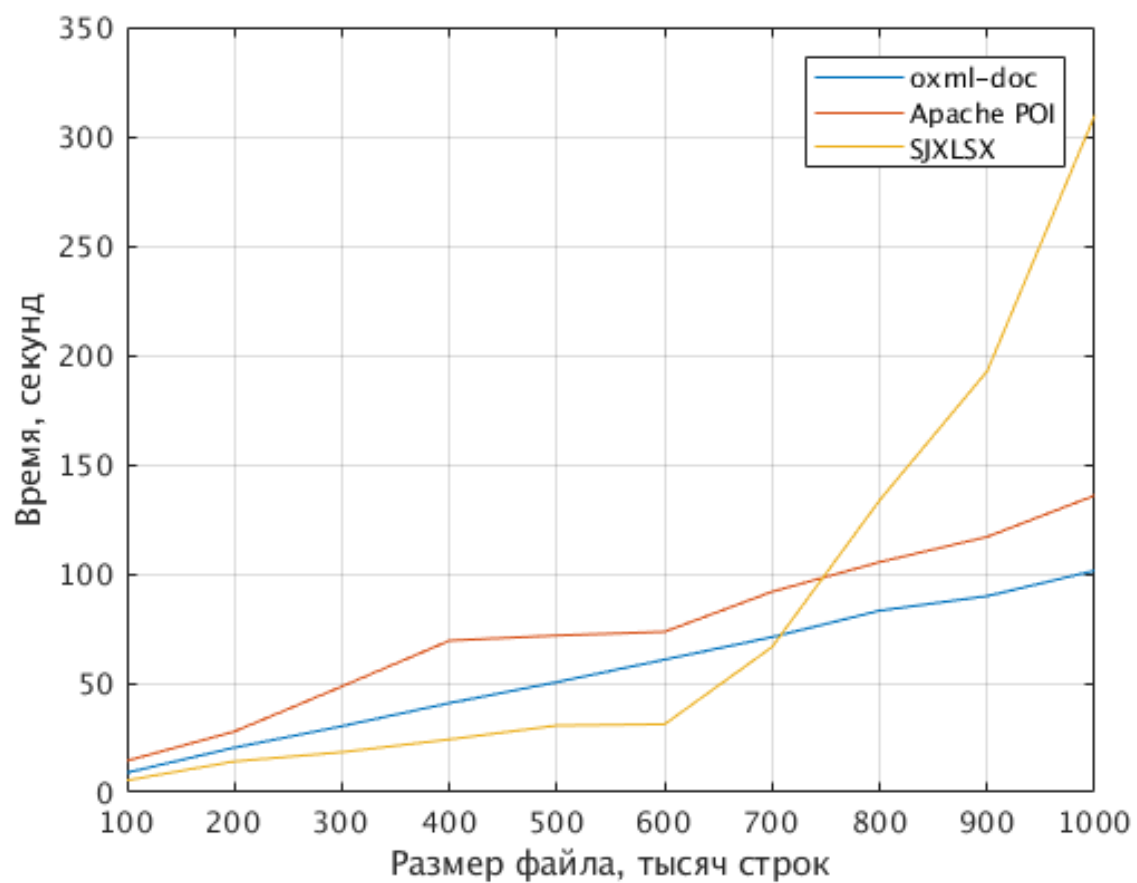


Рис. 6: Сравнение библиотек oxml-doc, Apache POI, SJXLSX по скорости работы



# Заключение

В ходе данной работы были достигнуты следующие результаты:

- реализована библиотека для потоковой записи файлов формата XLSX;
- проведена апробация реализации;
- проведено сравнение реализации с существующими библиотеками по количеству используемой RAM и по скорости работы при создании документа;
- исходный код и примеры использования библиотеки опубликованы на github;
- сборка библиотеки опубликована в maven-central.

# Листинги

```
1 <workbook . . .>
2   . . .
3   <workbookPr . . ./>
4   <sheets>
5     <sheet name='sheet1' r:id='rId1'>
6     <sheet name='sheet2' r:id='rId2'>
7     <sheet name='sheet3' r:id='rId3'>
8   </sheets>
9   . . .
10 </workbook>
```

Листинг 1: Пример файла workbook.xml

```
1 <worksheet . . .>
2   . . .
3   <cols>
4     <col min='1' max='1' width='26.140625' customWidth='1'>
5       . . .
6     </cols>
7
8   <sheetData>
9     <row r='1'>
10      <c r='A1' s='1' t='s'>
11        <v>0</v>
12      . . .
13      </c>
14    </row>
15    . . .
16  </sheetData>
17  . . .
18
19  <mergeCells count='1'>
20    <mergeCell ref='B12:J16'>
21    </mergeCells>
22
23  <pageMargins . . ./>
24  <pageSetup . . ./>
25
26  <tableParts ccount='1'>
27    <tableParts count='1'>
28  </tablePart r:id='rId2'>
29 </worksheet>
```

Листинг 2: Пример файла worksheet.xml

```

1 try (WorkbookWriter writer = new WorkbookWriter(1000, RowLimitRules.SPLIT)) {
2     final WorkSheet sheet = writer.createNewSheet("sheet");
3     final Row row = new Row();
4     for (int i = 0; i < 1000; i++) {
5         for (int j = 0; j < 55; j++) {
6             row.append("test");
7             sheet = sheet.appendRow(row);
8             row.clear();
9         }
10        writer.writeTo(new File("Test.xlsx"));
11    } catch (Throwable e) {
12        e.printStackTrace();
13    }

```

Листинг 3: Пример создания простого документа XLSX с помощью библиотеки oxml-doc

```

1 try (WorkbookWriter writer = new WorkbookWriter()) {
2     // base text font
3     final Font textFont = writer.createFont();
4     textFont.setFontSize("11");
5     textFont.setName("Calibri");
6     textFont.setColor("FF2020FF");
7
8     // headers text font
9     final Font headerFont = writer.createFont();
10    headerFont.setFontSize("11");
11    headerFont.setName("Calibri");
12    headerFont.setColor("FF2020F0");
13    headerFont.setBold(true);
14
15    // special font for the best sold items
16    final Font bestSoldFont = writer.createFont();
17    bestSoldFont.setFontSize("11");
18    bestSoldFont.setName("Calibri");
19    bestSoldFont.setColor("FFF02020");
20
21    // headers fill
22    final Fill emptyFill = writer.createFill();
23
24    final Fill headerFill = writer.createFill();
25    headerFill.setPattern(FillPattern.SOLID);
26    headerFill.setBackgroundColor(new Color(0xFFD0D0D0));
27    headerFill.setFrontColor(new Color(0xFFD0D0D0));
28
29    // special fill for price column

```

```

30 final Fill priceFill = writer.createFill();
31 priceFill.setPattern(FillPattern.SOLID);
32 priceFill.setBackgroundColor(new Color(0xFFFFF0));
33 priceFill.setFrontColor(new Color(0xFFFFF0));
34
35
36 // base style
37 final CellStyle baseStyle = writer.createCellStyle();
38 baseStyle.setFont(textFont);
39
40 // header style
41 final CellStyle headerStyle = writer.createCellStyle();
42 headerStyle.setFont(headerFont);
43 headerStyle.setFill(headerFill);
44 headerStyle.setAlignment(new Alignment(HorizontalAlignment.CENTER, false,
    VerticalAlignment.CENTER));
45
46 // header row style
47 final RowStyle headerRowStyle = new RowStyle(30, headerStyle, true, true);
48
49 // price column style
50 final CellStyle priceColumnStyle = writer.createCellStyle();
51 priceColumnStyle.setFont(headerFont);
52 priceColumnStyle.setFill(priceFill);
53 priceColumnStyle.setNumberFormat(writer.createNumFormat().asNumberFmt(2));
54
55 final CellStyle bestSoldStyle = writer.createCellStyle();
56 bestSoldStyle.setFont(bestSoldFont);
57 bestSoldStyle.setFill(emptyFill);
58
59 final RowStyle bestSoldRowStyle = new RowStyle(20, bestSoldStyle, true, true);
60
61 //Lets create a sheet
62 writer.createNewSheet("Market place");
63
64 //Column rules
65 writer.addColumnRule(new ColumnRule(0, 50));
66 writer.addColumnRule(new ColumnRule(1, 25));
67 writer.addColumnRule(new ColumnRule(2, 40));
68 writer.addColumnRule(new ColumnRule(3, 25, priceColumnStyle));
69 writer.addColumnRule(new ColumnRule(4, 20, priceColumnStyle));
70
71 writer.appendRow(new Row(headerRowStyle).append("Item", headerStyle).append("Count",
    headerStyle)
72     .append("Last changes", headerStyle)
73     .append("Price", headerStyle));

```

```

74
75 writer.appendRow(new Row().append("Oranges").append("2 kg").append(new Date()).append
    (10).append("10.0"));
76 writer.appendRow(new Row().append("Apples").append("1 kg").append(new Date()).append
    (20).append("20.0"));
77 writer.appendRow(new Row().append("Mangos").append("50 kg.").append(new Date()).
    append(30).append("30.0"));
78
79 writer.appendRow(new Row(bestSoldRowStyle).append("Bananas", bestSoldStyle)
80     .append("sold out", bestSoldStyle)
81     .append(new Date(), bestSoldStyle)
82     .append(25.3).append("25.3"));
83
84 final File testFile = new File("test_base.xlsx");
85 try (FileOutputStream fout = new FileOutputStream(testFile)) {
86     writer.writeTo(fout);
87 } catch (Throwable e) {
88     e.printStackTrace();
89 }
90 Assert.assertEquals(testFile.length(), 0);
91
92 //cleaning up
93 testFile.delete();
94 } catch (Throwable e) {
95     e.printStackTrace();
96 }

```

Листинг 4: Пример создания документа XLSX с форматированием с помощью библиотеки oxml-doc

## Список литературы

- [1] Apache. Apache POI // Apache official page. — URL: <https://poi.apache.org/> (online; accessed: 18.05.2018).
- [2] Ecma. ECMA-376 // web page. — URL: <https://www.ecma-international.org/publications/standards/Ecma-376.htm> (online; accessed: 18.05.2018).
- [3] ISO/IEC. ISO/IEC 29500 // web page. — URL: <https://www.iso.org/standard/71691.html> (online; accessed: 18.05.2018).
- [4] JSON. JSON // JSON official page. — URL: <https://www.json.org/> (online; accessed: 18.05.2018).
- [5] OOXML. OOXML // web page. — URL: <http://officeopenxml.com/SScontentOverview.php> (online; accessed: 18.05.2018).
- [6] Pelfree David. SJXLSX // Github repository. — URL: <https://github.com/davidpelfree/sjxlsx> (online; accessed: 18.05.2018).
- [7] XLSX. XLSX // Microsoft XLSX format description page. — URL: [https://msdn.microsoft.com/en-us/library/dd922181\(v=office.12\).aspx](https://msdn.microsoft.com/en-us/library/dd922181(v=office.12).aspx) (online; accessed: 18.05.2018).
- [8] XML. XML // XML specification page. — URL: <https://www.w3.org/TR/xml/> (online; accessed: 18.05.2018).