GrapeCity

葡萄城表格产品

性能测试报告

目录

本报告的目的

通过设计常见的性能测试用例，从执行时间、文件体积、内存占用等方面给出具体的数据指标。帮助用户在产品评估以及架构设计时，根据业务特性，合理的规划架构和选择产品。同时，也可以通过数据指标，快速了解葡萄城表格产品在性能及容量方面的能力。

本报告将对 SpreadJS 以及 GcExcel for Java（以下简称 GcExcel）两个产品进行性能测试。

前提声明

1. 本性能测试报告仅为了在模拟客户场景中高频操作后，仅作为产品的执行性能提供参考性的指标，并不代表葡萄城产品在第三方运行环境下的具体性能表现，也不代表产品性能说明书！
2. 本报告中的数据均为人工在特定的环境中获得，并不代表所有运行环境，数据仅供参考。
3. 本报告中建议的实现架构仅提供参考意义，是否适合您的具体业务场景，请联系我们的技术顾问具体评估

内容导引

- 如果想具体了解 SpreadJS 或 GcExcel 在某个具体功能中的性能数据，可以通过浏览具体的测试用例了解详情，包括测试方法、测试数据等细节信息
- 如果想快速直观的了解葡萄城表格产品的性能表现，可以直接查看两个产品对财务三张表的测试用例，可以快速了解打开、保存时的性能表现
    - 快速跳转：对典型表格文件的测试用例：SpreadJS / GcExcel
- 如果想从应用场景的角度关注对应测试性能，可以通过下表快速了解应用场景中对应的测试用例：

测试用例编号 / 应用场景		企业自定义报表	计量检测与检定	金融保险精算	实验室管理	自动化报表	企业预算与决算	数据指标补录	项目投资分析测算
SpreadJS	1#	✓	✓		✓		✓	✓	
	2#	✓	✓	✓	✓	✓	✓	✓	✓
	3#			✓			✓	✓	✓
	4#	✓					✓	✓	
	6#	✓		✓			✓	✓	
GcExcel	7#		✓	✓	✓	✓		✓	✓
	8#	✓				✓			✓
	9#	✓		✓		✓	✓		✓
	10#		✓	✓	✓	✓		✓	
	11#	✓	✓	✓	✓	✓	✓	✓	
	12#	✓	✓		✓	✓	✓		✓
	13#	✓				✓	✓		✓
	14#	✓		✓		✓			

注：以上表格中的 1# 代表测试用例编号，如：3# 表示**测试用例 3#：使用 SpreadJS 进行公式计算性能测试**

- 如果需要在应用架构考虑性能因素，可参考报告中“典型应用场景架构推荐”，快速了解两个产品对于常见场景的推荐方案

性能测试：SpreadJS

测试环境（配置）说明

- **产品测试版本：**SpreadJS v16.1.2
- **测试环境配置：**

浏览器	操作系统	内存	CPU
Chrome 114.0.5735.110 (Official Build) (64-bit)	Windows 11 21H2 OS build 22000.1936	32 GB	Intel(R) Core (TM) i7-7820HQ CPU @ 2.90GHz

- **测试方法及数据：**见各个用例
- **框架使用：**为了获取真实的数据，使用纯 JS 方式完成测试，未使用如 Vue、React 等框架
- **SpreadJS 库加载：**为了更真实的模拟用户环境，除非用例中特别说明，以下为测试时加载的库：

```
<link href="./lib/css/gc.spread.sheets.excel2013white.16.1.2.css" rel="stylesheet" />

<script src="./lib/scripts/gc.spread.sheets.all.16.1.2.min.js"></script>
<script src="./lib/scripts/plugins/gc.spread.sheets.io.16.1.2.min.js"></script>
<script src="./lib/scripts/plugins/gc.spread.sheets.print.16.1.2.min.js"></script>
<script src="./lib/scripts/plugins/gc.spread.sheets.pdf.16.1.2.min.js"></script>
<script src="./lib/scripts/plugins/gc.spread.sheets.charts.16.1.2.min.js"></script>
<script src="./lib/scripts/plugins/gc.spread.sheets.shapes.16.1.2.min.js"></script>
<script src="./lib/scripts/plugins/gc.spread.sheets.slicers.16.1.2.min.js"></script>
<script src="./lib/scripts/plugins/gc.spread.pivot.pivottables.16.1.2.min.js"></script>
<script src="./lib/scripts/plugins/gc.spread.sheets.barcode.16.1.2.min.js"></script>
<script src="./lib/scripts/plugins/gc.spread.sheets.tablesheet.16.1.2.min.js"></script>
<script src="./lib/scripts/plugins/gc.spread.calcengine.languagepackages.16.1.2.min.js">
```

测试用例 1#：使用 SpreadJS 在浏览器中导入本地 Excel 文件

用例说明

作为 SpreadJS 的核心功能，导入本地 Excel 文件功能几乎应用在所有用户的实际场景中，此示例将使用多个不同大小（数据量）的 Excel 文件进行导入操作。

用例性能指标

- 导入 Excel 文件所需时间
- 导入 Excel 文件使用的 JS 内存

测试方法

- 每个文件会执行三次测试代码，均会重新刷新浏览器
- 所需时间的测量方法：

- 使用 SpreadJS v16 版本的 import 方法，使用默认参数（未开启 lazyload），如下图：

```
document.getElementById("loadExcel").addEventListener("click", function () {
    var excelFile = document.getElementById("fileDemo").files[0];
    timeStart("import excel time");
    memoryStart("import excel memory");
    workbook.import(excelFile, function () {
        memoryEnd();
        timeEnd();
    }, function (message) {
        console.log(message);
    }, {
        fileType: GC.Spread.Sheets.FileType.excel
    });
});
```

```
function timeStart(flag) {
    window.timeFlag = flag;
    console.time(timeFlag);
}
function timeEnd() {
    if (window.timeFlag) {
        console.timeEnd(window.timeFlag);
        delete window.timeFlag;
    }else{
        console.timeEnd();
    }
}
```

- 使用内存测量方法：
  - 调用浏览器支持的 performance 对象，从而获取执行导入代码前后内存变化，如下图：

```
function memoryStart(flag) {
    window.memoryFlag = flag;
    window.memoryStart = performance.memory.totalJSHeapSize;
}
function memoryEnd() {
    if (window.memoryFlag) {
        var memoryEnd = performance.memory.totalJSHeapSize;
        console.log(window.memoryFlag, (memoryEnd - window.memoryStart) / Math.pow(1024, 2));
    }
}
```

注：为了获取内存信息，需要 Chrome 开启 --enable-precise-memory-info 开关

测试数据说明

- 测试文件中的数据如下图：其中 B 和 K 列设置了样式（边框+底色）

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	列头1	列头2	列头3	列头4	列头5	列头6	列头7	列头8	列头9	列头10	列头11	列头12	列头13	列头14	列头15	列头16	列头17	列头18	列头19	列头20
2	1	中文r1c1	中文r1c2	中文r1c3	中文r1c4	中文r1c5	中文r1c6	中文r1c7	中文r1c8	中文r1c9	中文r1c10	1-1月-00	15-8月-72	13	14	15	16	17	18	19
3	2	中文r1c1	中文r1c2	中文r1c3	中文r1c4	中文r1c5	中文r1c6	中文r1c7	中文r1c8	中文r1c9	中文r1c10	1-1月-00	15-8月-72	13	14	15	16	17	18	19
4	3	中文r3c1	中文r3c2	中文r3c3	中文r3c4	中文r3c5	中文r3c6	中文r3c7	中文r3c8	中文r3c9	中文r3c10	1-1月-00	15-8月-72	53	54	55	56	57	58	59
5	4	中文r3c1	中文r3c2	中文r3c3	中文r3c4	中文r3c5	中文r3c6	中文r3c7	中文r3c8	中文r3c9	中文r3c10	1-1月-00	15-8月-72	53	54	55	56	57	58	59
6	5	中文r5c1	中文r5c2	中文r5c3	中文r5c4	中文r5c5	中文r5c6	中文r5c7	中文r5c8	中文r5c9	中文r5c10	1-1月-00	15-8月-72	93	94	95	96	97	98	99
7	6	中文r5c1	中文r5c2	中文r5c3	中文r5c4	中文r5c5	中文r5c6	中文r5c7	中文r5c8	中文r5c9	中文r5c10	1-1月-00	15-8月-72	93	94	95	96	97	98	99
8	7	中文r7c1	中文r7c2	中文r7c3	中文r7c4	中文r7c5	中文r7c6	中文r7c7	中文r7c8	中文r7c9	中文r7c10	1-1月-00	15-8月-72	133	134	135	136	137	138	139
9	8	中文r7c1	中文r7c2	中文r7c3	中文r7c4	中文r7c5	中文r7c6	中文r7c7	中文r7c8	中文r7c9	中文r7c10	1-1月-00	15-8月-72	133	134	135	136	137	138	139
10	9	中文r9c1	中文r9c2	中文r9c3	中文r9c4	中文r9c5	中文r9c6	中文r9c7	中文r9c8	中文r9c9	中文r9c10	1-1月-00	15-8月-72	173	174	175	176	177	178	179
11	10	中文r9c1	中文r9c2	中文r9c3	中文r9c4	中文r9c5	中文r9c6	中文r9c7	中文r9c8	中文r9c9	中文r9c10	1-1月-00	15-8月-72	173	174	175	176	177	178	179
12	11	中文r11c1	中文r11c2	中文r11c3	中文r11c4	中文r11c5	中文r11c6	中文r11c7	中文r11c8	中文r11c9	中文r11c10	1-1月-00	15-8月-72	213	214	215	216	217	218	219
13	12	中文r11c1	中文r11c2	中文r11c3	中文r11c4	中文r11c5	中文r11c6	中文r11c7	中文r11c8	中文r11c9	中文r11c10	1-1月-00	15-8月-72	213	214	215	216	217	218	219
14	13	中文r13c1	中文r13c2	中文r13c3	中文r13c4	中文r13c5	中文r13c6	中文r13c7	中文r13c8	中文r13c9	中文r13c10	1-1月-00	15-8月-72	253	254	255	256	257	258	259
15	14	中文r13c1	中文r13c2	中文r13c3	中文r13c4	中文r13c5	中文r13c6	中文r13c7	中文r13c8	中文r13c9	中文r13c10	1-1月-00	15-8月-72	253	254	255	256	257	258	259
16	15	中文r15c1	中文r15c2	中文r15c3	中文r15c4	中文r15c5	中文r15c6	中文r15c7	中文r15c8	中文r15c9	中文r15c10	1-1月-00	15-8月-72	293	294	295	296	297	298	299
17	16	中文r15c1	中文r15c2	中文r15c3	中文r15c4	中文r15c5	中文r15c6	中文r15c7	中文r15c8	中文r15c9	中文r15c10	1-1月-00	15-8月-72	293	294	295	296	297	298	299
18	17	中文r17c1	中文r17c2	中文r17c3	中文r17c4	中文r17c5	中文r17c6	中文r17c7	中文r17c8	中文r17c9	中文r17c10	1-1月-00	15-8月-72	333	334	335	336	337	338	339
19	18	中文r17c1	中文r17c2	中文r17c3	中文r17c4	中文r17c5	中文r17c6	中文r17c7	中文r17c8	中文r17c9	中文r17c10	1-1月-00	15-8月-72	333	334	335	336	337	338	339
20	19	中文r19c1	中文r19c2	中文r19c3	中文r19c4	中文r19c5	中文r19c6	中文r19c7	中文r19c8	中文r19c9	中文r19c10	1-1月-00	15-8月-72	373	374	375	376	377	378	379
21	20	中文r19c1	中文r19c2	中文r19c3	中文r19c4	中文r19c5	中文r19c6	中文r19c7	中文r19c8	中文r19c9	中文r19c10	1-1月-00	15-8月-72	373	374	375	376	377	378	379

性能数据结果

性能指标	50 万 2.5 万行 * 20 列	100 万 5 万行 * 20 列	500 万 25 万行 * 20 列	1,000 万 50 万行 * 20 列	1,500 万 75 万行 * 20 列
Excel 文件大小	2.554 MB	5.094 MB	25.471 MB	51.332 MB	77.196 MB
导入时间	1,803 ms	2,951 ms	13,206 ms	30,437 ms	不支持
所需内存	48.3 MB	86.6 MB	774.6 MB	1,408 MB	不支持

注：1,500 万数量文件在测试机器环境中出现内存不足，浏览器崩溃

测试用例 2#：将 SpreadJS 中的表格保存为本地文件

用例说明

文件导出也是 SpreadJS 作为表格控件在各个应用场景的核心功能，此用例将使用 SpreadJS 将在浏览器中的表格保存为支持的文件类型，测试：Excel vs SJS。

注：SJS 为 SpreadJS v16.0 版本开始支持的全新文件存储结构，目的是提供更快的加载、更小的保存体积、更高效的计算性能，更多有关 SJS 介绍，请点击这里。

用例性能指标

- 导出文件所需时间
- 导出文件使用的 JS 内存

测试方法

- 每个文件会执行三次测试代码，均会重新刷新浏览器
- 所需时间测量方法：
  - 使用 SpreadJS v16 版本的 export 方法（导出 Excel），使用默认参数，如下图：

```
document.getElementById("saveExcel").addEventListener("click", function () {
document.getElementById("saveSJS").addEventListener("click", function () {
    timeStart("export sjs time");
    memoryStart("export sjs memory");
    workbook.save(function (data) {
        memoryEnd();
        timeEnd();
    }, function (message) {
        console.log("error:", message);
    })
});
});
```

  - 使用 Save 方法保存 SJS 文件，使用默认参数，如下图：
- 使用内存测量方法：
  - 方法参考测试用例 #1

测试数据说明

- 导出测试数据采用测试用例 #1 中导入的文件数据，具体参考测试用例 #1

性能数据结果

文件类型	性能指标（单元格数量）	50 万 2.5 万行 * 20 列	100 万 5 万行 * 20 列	500 万 25 万行 * 20 列	1,000 万 50 万行 * 20 列
Excel	导出时间	2,004 ms	3,802 ms	18,814 ms	42,786 ms
	所需内存	91.9 MB	258.8 MB	1,010.9 MB	1286.9 MB
SJS	导出时间	742 ms	1,443 ms	7,554 ms	15,729 ms
	所需内存	61.6 MB	88.3 MB	479.6 MB	966.4 MB

测试用例 3#：使用 SpreadJS 进行公式计算性能测试

用例说明

公式计算性能也是电子表格产品的核心指标之一，此用例旨在不同数量的公式函数下，获取 SpreadJS 实际执行全量计算所需时间以及 JS 内存使用。从而在一些高负载的公式计算场景，让用户对 SpreadJS 的计算能力有清晰的参考指标。

用例性能指标

- 全量计算所需时间
- 全量计算使用的 JS 内存

测试方法

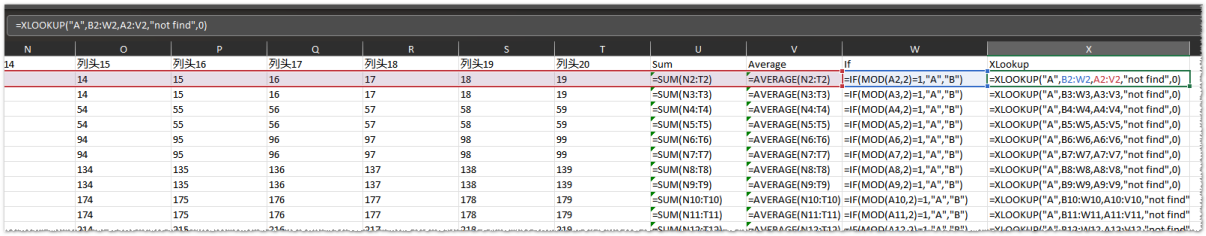
- 每个文件会执行三次测试代码，均会重新刷新浏览器
- 所需时间测量方法：
  - 使用开关计算引擎，达到全量计算，使用默认参数，如下图：

```
document.getElementById("calc").addEventListener("click", function () {
    timeStart("calculation time");
    memoryStart("calculation memory");
    workbook.suspendCalcService(false);
    workbook.resumeCalcService(true);
    memoryEnd();
    timeEnd();
});
```

- 使用内存测量方法：
  - 方法参考测试用例 #1

测试数据说明

- 此用例测试数据基于用例 #1 的数据，在最后增加 4 列计算列，分别是：
  - U 列：=SUM(N2:T2) / 计算同行 7 列和
  - V 列：=AVERAGE(N2:T2) / 计算同行 7 列平均数
  - W 列：=IF(MOD(A2,2)=1,"A","B") / 对同行 A 列取余数，并使用 IF 判断分别输出 A 或 B
  - X 列：=XLOOKUP("A",B2:W2,A2:V2,"not find",0) / 在同行中查找匹配字符“A”
  - 示例数据如下图：



14	列头15	列头16	列头17	列头18	列头19	列头20	Sum	Average	If	XLookup
	14	15	16	17	18	19	=SUM(N2:T2)	=AVERAGE(N2:T2)	=IF(MOD(A2,2)=1,"A","B")	=XLOOKUP("A",B2:W2,A2:V2,"not find",0)
	14	15	16	17	18	19	=SUM(N3:T3)	=AVERAGE(N3:T3)	=IF(MOD(A3,2)=1,"A","B")	=XLOOKUP("A",B3:W3,A3:V3,"not find",0)
	54	55	56	57	58	59	=SUM(N4:T4)	=AVERAGE(N4:T4)	=IF(MOD(A4,2)=1,"A","B")	=XLOOKUP("A",B4:W4,A4:V4,"not find",0)
	54	55	56	57	58	59	=SUM(N5:T5)	=AVERAGE(N5:T5)	=IF(MOD(A5,2)=1,"A","B")	=XLOOKUP("A",B5:W5,A5:V5,"not find",0)
	94	95	96	97	98	99	=SUM(N6:T6)	=AVERAGE(N6:T6)	=IF(MOD(A6,2)=1,"A","B")	=XLOOKUP("A",B6:W6,A6:V6,"not find",0)
	94	95	96	97	98	99	=SUM(N7:T7)	=AVERAGE(N7:T7)	=IF(MOD(A7,2)=1,"A","B")	=XLOOKUP("A",B7:W7,A7:V7,"not find",0)
	134	135	136	137	138	139	=SUM(N8:T8)	=AVERAGE(N8:T8)	=IF(MOD(A8,2)=1,"A","B")	=XLOOKUP("A",B8:W8,A8:V8,"not find",0)
	134	135	136	137	138	139	=SUM(N9:T9)	=AVERAGE(N9:T9)	=IF(MOD(A9,2)=1,"A","B")	=XLOOKUP("A",B9:W9,A9:V9,"not find",0)
	174	175	176	177	178	179	=SUM(N10:T10)	=AVERAGE(N10:T10)	=IF(MOD(A10,2)=1,"A","B")	=XLOOKUP("A",B10:W10,A10:V10,"not find"
	174	175	176	177	178	179	=SUM(N11:T11)	=AVERAGE(N11:T11)	=IF(MOD(A11,2)=1,"A","B")	=XLOOKUP("A",B11:W11,A11:V11,"not find"

性能数据结果

性能指标	公式数量（基于 100 万单元格）				
	1,000	1 万	5 万	10 万	50 万
计算时间	91 ms	343 ms	1,194 ms	2,196 ms	11,289 ms
使用内存	3.1 MB	6.2 MB	23.7 MB	42.5 MB	44.0 MB

测试用例 4#：对表格进行大量设置/获取单元格操作

用例说明

在用户的场景中，经常会碰到需要大量设置或者获取单元格数据的操作，例如：从后台获取的数据需要填充到表格中，或者用户录入了大量数据，需要遍历并提交后台保存。

虽然 SpreadJS 提供了诸如数据绑定完成此类操作的高级功能，但设置/获取单元格作为最基本的 API，其性能指标能代表产品在数据存储方面的能力。

用例性能指标

- 完成不同数量等级单元格设置/获取所需时间
- 设置/获取过程使用的 JS 内存

测试方法

- 每个文件会执行三次测试代码，均会重新刷新浏览器

- 所需时间测量方法：
  - 使用双重循环完成制定数据量的操作，并在数据准备时增加了随机数据，如下图：

```
document.getElementById("set_get_value").addEventListener("click", function () {
    var dataCount = +document.getElementById("dataRows").value;
    var sheet = workbook.getActiveSheet(), columnCount = 20, rowCount = parseInt(dataCount / columnCount) + 1;
    sheet.setRowCount(rowCount);
    sheet.setColumnCount(columnCount);
    var seed0 = "列头", seed1 = "中文";
    timeStart(`set ${dataCount} value time`);
    memoryStart(`set ${dataCount} value memory`);

    workbook.suspendPaint();
    workbook.suspendCalcService(true);
    workbook.suspendEvent();
    sheet.suspendDirty();

    for (var row = 0; row < rowCount; row++) {
        for (var col = 0; col < columnCount; col++) {
            if (row === 0) {
                sheet.setValue(row, col, seed0 + col);
            } else if (col === 1) {
                sheet.setValue(row, col, row)
            } else if (col > 1 && col < 10) {
                sheet.setValue(row, col, `${seed1}r${row}c${col}`);
            } else if (col >= 10 && col < 12) {
                sheet.setValue(row, col, new Date(2000, 12, Math.random() * 31))
            } else {
                sheet.setValue(row, col, row + "" + col);
            }

        }
    }

    sheet.resumeDirty();
    workbook.resumeEvent();
    workbook.resumeCalcService(false);
    workbook.resumePaint();

    memoryEnd();
    timeEnd();

    timeStart(`get ${dataCount} value time`);
    memoryStart(`get ${dataCount} value memory`);
    for (var row = 0; row < rowCount; row++) {
        for (var col = 0; col < columnCount; col++) {
            sheet.getValue(row, col);
        }
    }

    memoryEnd();
    timeEnd();
});
```

- 使用内存测量方法：
  - 方法参考测试用例 #1

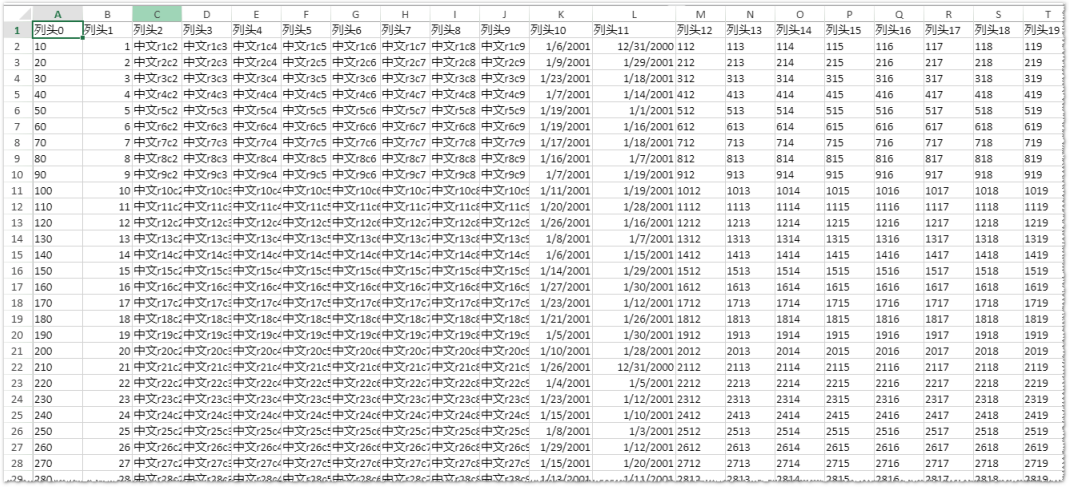
性能数据结果

操作类型	性能指标	操作单元格数量			
		50 万	100 万	1,000 万	1,500 万
设置操作	使用内存	29.0 MB	54.2 MB	580.2 MB	925.8 MB
	执行时间	750 ms	1,488 ms	15,193 ms	24,137 ms
获取操作	使用内存	1.5 MB	1.5 MB	1.5 MB	1.5 MB
	执行时间	134 ms	249 ms	2,456 ms	3,836 ms

注：因 getValue 并未赋予任何变量，因此执行内存与单元格数量无关，仅供参考

操作后表格数据示例：

（见下页）



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	列头0	列头1	列头2	列头3	列头4	列头5	列头6	列头7	列头8	列头9	列头10	列头11	列头12	列头13	列头14	列头15	列头16	列头17	列头18	列头19
2	10	1	中文r1c2	中文r1c3	中文r1c4	中文r1c5	中文r1c6	中文r1c7	中文r1c8	中文r1c9	1/6/2001	12/31/2000	112	113	114	115	116	117	118	119
3	20	2	中文r2c2	中文r2c3	中文r2c4	中文r2c5	中文r2c6	中文r2c7	中文r2c8	中文r2c9	1/9/2001	1/29/2001	212	213	214	215	216	217	218	219
4	30	3	中文r3c2	中文r3c3	中文r3c4	中文r3c5	中文r3c6	中文r3c7	中文r3c8	中文r3c9	1/23/2001	1/18/2001	312	313	314	315	316	317	318	319
5	40	4	中文r4c2	中文r4c3	中文r4c4	中文r4c5	中文r4c6	中文r4c7	中文r4c8	中文r4c9	1/7/2001	1/14/2001	412	413	414	415	416	417	418	419
6	50	5	中文r5c2	中文r5c3	中文r5c4	中文r5c5	中文r5c6	中文r5c7	中文r5c8	中文r5c9	1/19/2001	1/1/2001	512	513	514	515	516	517	518	519
7	60	6	中文r6c2	中文r6c3	中文r6c4	中文r6c5	中文r6c6	中文r6c7	中文r6c8	中文r6c9	1/19/2001	1/16/2001	612	613	614	615	616	617	618	619
8	70	7	中文r7c2	中文r7c3	中文r7c4	中文r7c5	中文r7c6	中文r7c7	中文r7c8	中文r7c9	1/17/2001	1/18/2001	712	713	714	715	716	717	718	719
9	80	8	中文r8c2	中文r8c3	中文r8c4	中文r8c5	中文r8c6	中文r8c7	中文r8c8	中文r8c9	1/16/2001	1/7/2001	812	813	814	815	816	817	818	819
10	90	9	中文r9c2	中文r9c3	中文r9c4	中文r9c5	中文r9c6	中文r9c7	中文r9c8	中文r9c9	1/7/2001	1/19/2001	912	913	914	915	916	917	918	919
11	100	10	中文r10c2	中文r10c3	中文r10c4	中文r10c5	中文r10c6	中文r10c7	中文r10c8	中文r10c9	1/11/2001	1/19/2001	1012	1013	1014	1015	1016	1017	1018	1019
12	110	11	中文r11c2	中文r11c3	中文r11c4	中文r11c5	中文r11c6	中文r11c7	中文r11c8	中文r11c9	1/20/2001	1/28/2001	1112	1113	1114	1115	1116	1117	1118	1119
13	120	12	中文r12c2	中文r12c3	中文r12c4	中文r12c5	中文r12c6	中文r12c7	中文r12c8	中文r12c9	1/26/2001	1/16/2001	1212	1213	1214	1215	1216	1217	1218	1219
14	130	13	中文r13c2	中文r13c3	中文r13c4	中文r13c5	中文r13c6	中文r13c7	中文r13c8	中文r13c9	1/8/2001	1/7/2001	1312	1313	1314	1315	1316	1317	1318	1319
15	140	14	中文r14c2	中文r14c3	中文r14c4	中文r14c5	中文r14c6	中文r14c7	中文r14c8	中文r14c9	1/6/2001	1/15/2001	1412	1413	1414	1415	1416	1417	1418	1419
16	150	15	中文r15c2	中文r15c3	中文r15c4	中文r15c5	中文r15c6	中文r15c7	中文r15c8	中文r15c9	1/14/2001	1/29/2001	1512	1513	1514	1515	1516	1517	1518	1519
17	160	16	中文r16c2	中文r16c3	中文r16c4	中文r16c5	中文r16c6	中文r16c7	中文r16c8	中文r16c9	1/27/2001	1/30/2001	1612	1613	1614	1615	1616	1617	1618	1619
18	170	17	中文r17c2	中文r17c3	中文r17c4	中文r17c5	中文r17c6	中文r17c7	中文r17c8	中文r17c9	1/23/2001	1/12/2001	1712	1713	1714	1715	1716	1717	1718	1719
19	180	18	中文r18c2	中文r18c3	中文r18c4	中文r18c5	中文r18c6	中文r18c7	中文r18c8	中文r18c9	1/21/2001	1/26/2001	1812	1813	1814	1815	1816	1817	1818	1819
20	190	19	中文r19c2	中文r19c3	中文r19c4	中文r19c5	中文r19c6	中文r19c7	中文r19c8	中文r19c9	1/5/2001	1/30/2001	1912	1913	1914	1915	1916	1917	1918	1919
21	200	20	中文r20c2	中文r20c3	中文r20c4	中文r20c5	中文r20c6	中文r20c7	中文r20c8	中文r20c9	1/10/2001	1/28/2001	2012	2013	2014	2015	2016	2017	2018	2019
22	210	21	中文r21c2	中文r21c3	中文r21c4	中文r21c5	中文r21c6	中文r21c7	中文r21c8	中文r21c9	1/26/2001	12/31/2000	2112	2113	2114	2115	2116	2117	2118	2119
23	220	22	中文r22c2	中文r22c3	中文r22c4	中文r22c5	中文r22c6	中文r22c7	中文r22c8	中文r22c9	1/4/2001	1/5/2001	2212	2213	2214	2215	2216	2217	2218	2219
24	230	23	中文r23c2	中文r23c3	中文r23c4	中文r23c5	中文r23c6	中文r23c7	中文r23c8	中文r23c9	1/23/2001	1/12/2001	2312	2313	2314	2315	2316	2317	2318	2319
25	240	24	中文r24c2	中文r24c3	中文r24c4	中文r24c5	中文r24c6	中文r24c7	中文r24c8	中文r24c9	1/15/2001	1/10/2001	2412	2413	2414	2415	2416	2417	2418	2419
26	250	25	中文r25c2	中文r25c3	中文r25c4	中文r25c5	中文r25c6	中文r25c7	中文r25c8	中文r25c9	1/8/2001	1/3/2001	2512	2513	2514	2515	2516	2517	2518	2519
27	260	26	中文r26c2	中文r26c3	中文r26c4	中文r26c5	中文r26c6	中文r26c7	中文r26c8	中文r26c9	1/29/2001	1/12/2001	2612	2613	2614	2615	2616	2617	2618	2619
28	270	27	中文r27c2	中文r27c3	中文r27c4	中文r27c5	中文r27c6	中文r27c7	中文r27c8	中文r27c9	1/15/2001	1/20/2001	2712	2713	2714	2715	2716	2717	2718	2719
29	280	28	中文r28c2	中文r28c3	中文r28c4	中文r28c5	中文r28c6	中文r28c7	中文r28c8	中文r28c9	1/13/2001	1/11/2001	2812	2813	2814	2815	2816	2817	2818	2819

测试用例 5#：边界容量测试（SpreadJS）

用例说明

在大型应用系统中，用户想了解产品支持的最大容量，例如：最大表单数量、单元格数量上限、最多包含多少公式计算等，本用例将用代码的方式，不断增加表单、单元格、公式等数量，直到浏览器崩溃或执行时间超过 30 分钟。

用例性能指标

- 达到停止条件时创建的对象数量
- 停止条件：
  - 浏览器因内存不足而崩溃
  - 执行时间达到 30 分钟

测试方法

- 创建对象代码示例，如下图：

```
document.getElementById("sheet_count").addEventListener("click", function () {
    var crashCount = 1;
    try {
        var time = Date.now();
        workbook.suspendPaint();
        workbook.suspendCalcService(true);
        workbook.suspendEvent();
        while (Date.now() - time < 30 * 60 * 1000) {
            workbook.setSheetCount(crashCount++);
        }
        workbook.resumeEvent();
        workbook.resumeCalcService(false);
        workbook.resumePaint();
    } catch {
        console.log(crashCount);
    }
    finally {
        console.log(crashCount);
    }
});
```

创建空表单

```
var time = Date.now();
var seed = 123.456, rowCount = 100000, colCount = 50, sheet = workbook.getActiveSheet();
sheet.setRowCount(rowCount);
sheet.setColumnCount(colCount);
var crashCount = 0;
try {
    workbook.suspendPaint();
    workbook.suspendCalcService(true);
    workbook.suspendEvent();
    sheet.suspendDirty();
    for (var row = 0; row < rowCount; row++) {
        for (var col = 0; col < colCount; col++) {
            crashCount++;
            if (Date.now() - time < 30 * 60 * 1000)
                return;
            sheet.setValue(row, col, seed);
        }
        if (rowCount - row < 10) {
            rowCount += 50000;
            sheet.setRowCount(rowCount);
            console.log(crashCount);
        }
    }
    sheet.resumeDirty();
    workbook.resumeEvent();
    workbook.resumeCalcService(false);
    workbook.resumePaint();
} catch { }
finally {
    console.log(crashCount);
}
```

设置浮点数单元格

```
var time = Date.now();
var seed = 'AVERAGE(10,20,30)', rowCount = 100000, colCount = 50, sheet = workbook.getActiveSheet();
sheet.setRowCount(rowCount);
sheet.setColumnCount(colCount);
var crashCount = 0;
try {
    workbook.suspendPaint();
    workbook.suspendCalcService(true);
    workbook.suspendEvent();
    sheet.suspendDirty();
    for (var row = 0; row < rowCount; row++) {
        for (var col = 0; col < colCount; col++) {
            crashCount++;
            if (Date.now() - time < 30 * 60 * 1000)
                return;
            sheet.setFormula(row, col, seed);
        }
        if (rowCount - row < 10) {
            rowCount += 50000;
            sheet.setRowCount(rowCount);
            console.log(crashCount);
        }
    }
    sheet.resumeDirty();
    workbook.resumeEvent();
    workbook.resumeCalcService(false);
    workbook.resumePaint();
} catch { }
finally {
    console.log(crashCount);
}
```

设置公式单元格

```
var time = Date.now();
var seed = 'test', rowCount = 100000, colCount = 50, sheet = workbook.getActiveSheet();
sheet.setRowCount(rowCount);
sheet.setColumnCount(colCount);
var crashCount = 0;
try {
    workbook.suspendPaint();
    workbook.suspendCalcService(true);
    workbook.suspendEvent();
    sheet.suspendDirty();
    for (var row = 0; row < rowCount; row++) {
        for (var col = 0; col < colCount; col++) {
            crashCount++;
            if (Date.now() - time < 30 * 60 * 1000)
                return;
            sheet.setValue(row, col, seed);
        }
        if (rowCount - row < 10) {
            rowCount += 50000;
            sheet.setRowCount(rowCount);
            console.log(crashCount);
        }
    }
    sheet.resumeDirty();
    workbook.resumeEvent();
    workbook.resumeCalcService(false);
    workbook.resumePaint();
} catch { }
finally {
    console.log(crashCount);
}
```

设置文字单元格

```
var time = Date.now();
var seed = new Date(), rowCount = 100000, colCount = 50, sheet = workbook.getActiveSheet();
sheet.setRowCount(rowCount);
sheet.setColumnCount(colCount);
var crashCount = 0;
try {
    workbook.suspendPaint();
    workbook.suspendCalcService(true);
    workbook.suspendEvent();
    sheet.suspendDirty();
    for (var row = 0; row < rowCount; row++) {
        for (var col = 0; col < colCount; col++) {
            crashCount++;
            if (Date.now() - time < 30 * 60 * 1000)
                return;
            sheet.setValue(row, col, seed);
        }
        if (rowCount - row < 10) {
            rowCount += 50000;
            sheet.setRowCount(rowCount);
            console.log(crashCount);
        }
    }
    sheet.resumeDirty();
    workbook.resumeEvent();
    workbook.resumeCalcService(false);
    workbook.resumePaint();
} catch { }
finally {
    console.log(crashCount);
}
```

设置日期单元格

测试数据结果

注：以上测试代码，在 30 分钟内均未造成浏览器崩溃，以下数据为测试代码运行到 30 分钟时，创建的对象数量：

	空表单	浮点数单元格	字符单元格	日期单元格	公式单元格
创建数量（个）	49,572	46,968	46,769	46,692	46,636

测试用例 6#：对典型财务表格文件：财务三张表，分别测量导入、导出所需时间和内存占用

用例说明

为了让更多用户对产品性能有更直观的认识，挑选了最常见的财务表格文件，进行导入、导出操作，获取执行数据以及代码执行内存。

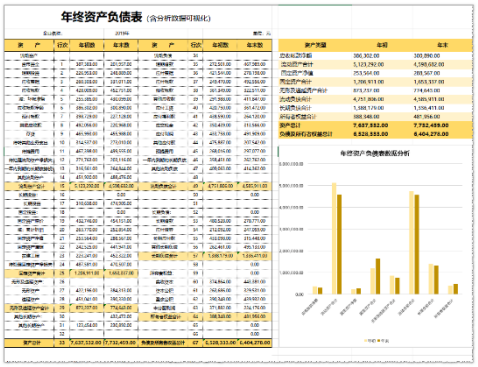
用例性能指标

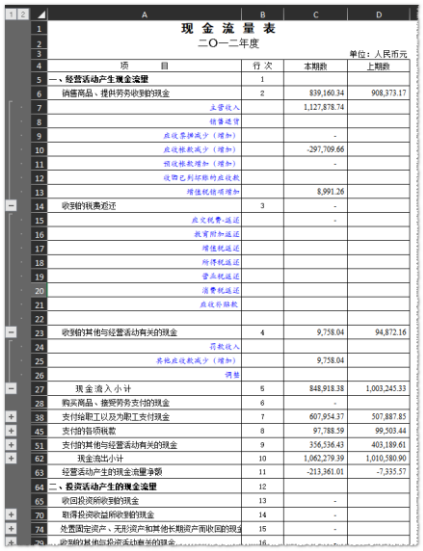
- 导入典型文件所需时间
- 导入、导出 SJS 文件使用的 JS 内存及时间

测试方法

- 测试方法参考用例 #1

测试文件说明





测试数据结果

典型表格文件	导入 Excel		导出 SJS		
	耗时	使用内存	耗时	使用内存	导出体积
利润表模板	418 ms	3.9 MB	43 ms	0.8 MB	7.0 KB
年终资产负债表(含分析)	462 ms	6.7 MB	88 ms	2.9 MB	22.0 KB
现金流量表模板	470 ms	4.4 MB	155 ms	2.6 MB	51.0 KB

性能测试：GcExcel

测试环境（配置）说明

- **产品测试版本：**GcExcel for Java v6.1.1
- **测试环境配置：**

JDK 版本	操作系统	CPU	内存
JDK 17.0.2, OpenJDK 64-Bit Server VM, 17.0.2+8-86	Windows 11 Professional	Intel(R) Core (TM) i7-9750H CPU @ 2.60GHz	32 GB

- **测试方法及数据：**见各个用例
- **框架使用：**为了获取更准确 Java 代码执行时间和内存使用，引入了 Java Microbenchmark Harness (JMH)，示例代码如下图：

```
Options opt = new OptionsBuilder()
        // Specify which benchmarks to run.
        // You can be more specific if you'd like to run only one benchmark per test.
        .include( regexp: App.class.getPackage().getName() + ".tests.ExcelReadTest_50w_withStyle")
        // Set the following options as needed
        .mode(Mode.SingleShotTime)
        .timeUnit(TimeUnit.MICROSECONDS)
        .warmupIterations(0)
        .measurementIterations(1)
        .forks(1)
        .shouldFailOnError(true)
        .addProfiler(GCProfiler.class)
        .build();

Collection<RunResult> result = new Runner(opt).run();
ValidationErrorHandler.HandleValidationErrors(
        TimeLimitValidator.CheckTimeLimits(
                new TestReport(result)));
```

注：具体依赖使用可通过报告最后的工程下载查看

测试用例 7#：GcExcel 加载不同大小的 Excel 文件

用例说明

作为后端 Java 组件，GcExcel 从设计初就需要频繁的加载 Excel 文件进行后续的编辑和处理，因此加载 Excel 文件的性能决定了是否能满足客户用户场景的重要指标之一。

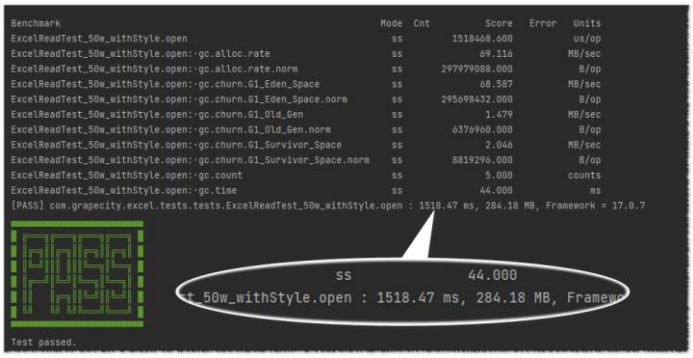
用例性能指标

- 加载 Excel 文件所需时间
- 加载 Excel 文件过程中申请的 JVM 内存

测试方法

（见下页）

- 使用 JMH 框架给出的结果作为指标数据，如下图：



注：此处的内存计数为累计申请，并非最大内存占用

测试数据说明

- 测试文件中的数据如下图：其中 B 和 K 列设置了样式（边框+底色）

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	列头1	列头2	列头3	列头4	列头5	列头6	列头7	列头8	列头9	列头10	列头11	列头12	列头13	列头14	列头15	列头16	列头17	列头18	列头19	列头20
2	1	中文r1c1	中文r1c2	中文r1c3	中文r1c4	中文r1c5	中文r1c6	中文r1c7	中文r1c8	中文r1c9	中文r1c10	1-1月-00	15-8月-72	13	14	15	16	17	18	19
3	2	中文r1c1	中文r1c2	中文r1c3	中文r1c4	中文r1c5	中文r1c6	中文r1c7	中文r1c8	中文r1c9	中文r1c10	1-1月-00	15-8月-72	13	14	15	16	17	18	19
4	3	中文r3c1	中文r3c2	中文r3c3	中文r3c4	中文r3c5	中文r3c6	中文r3c7	中文r3c8	中文r3c9	中文r3c10	1-1月-00	15-8月-72	53	54	55	56	57	58	59
5	4	中文r3c1	中文r3c2	中文r3c3	中文r3c4	中文r3c5	中文r3c6	中文r3c7	中文r3c8	中文r3c9	中文r3c10	1-1月-00	15-8月-72	53	54	55	56	57	58	59
6	5	中文r5c1	中文r5c2	中文r5c3	中文r5c4	中文r5c5	中文r5c6	中文r5c7	中文r5c8	中文r5c9	中文r5c10	1-1月-00	15-8月-72	93	94	95	96	97	98	99
7	6	中文r5c1	中文r5c2	中文r5c3	中文r5c4	中文r5c5	中文r5c6	中文r5c7	中文r5c8	中文r5c9	中文r5c10	1-1月-00	15-8月-72	93	94	95	96	97	98	99
8	7	中文r7c1	中文r7c2	中文r7c3	中文r7c4	中文r7c5	中文r7c6	中文r7c7	中文r7c8	中文r7c9	中文r7c10	1-1月-00	15-8月-72	133	134	135	136	137	138	139
9	8	中文r7c1	中文r7c2	中文r7c3	中文r7c4	中文r7c5	中文r7c6	中文r7c7	中文r7c8	中文r7c9	中文r7c10	1-1月-00	15-8月-72	133	134	135	136	137	138	139
10	9	中文r9c1	中文r9c2	中文r9c3	中文r9c4	中文r9c5	中文r9c6	中文r9c7	中文r9c8	中文r9c9	中文r9c10	1-1月-00	15-8月-72	173	174	175	176	177	178	179
11	10	中文r9c1	中文r9c2	中文r9c3	中文r9c4	中文r9c5	中文r9c6	中文r9c7	中文r9c8	中文r9c9	中文r9c10	1-1月-00	15-8月-72	173	174	175	176	177	178	179
12	11	中文r11c1	中文r11c2	中文r11c3	中文r11c4	中文r11c5	中文r11c6	中文r11c7	中文r11c8	中文r11c9	中文r11c1	1-1月-00	15-8月-72	213	214	215	216	217	218	219
13	12	中文r11c1	中文r11c2	中文r11c3	中文r11c4	中文r11c5	中文r11c6	中文r11c7	中文r11c8	中文r11c9	中文r11c1	1-1月-00	15-8月-72	213	214	215	216	217	218	219
14	13	中文r13c1	中文r13c2	中文r13c3	中文r13c4	中文r13c5	中文r13c6	中文r13c7	中文r13c8	中文r13c9	中文r13c1	1-1月-00	15-8月-72	253	254	255	256	257	258	259
15	14	中文r13c1	中文r13c2	中文r13c3	中文r13c4	中文r13c5	中文r13c6	中文r13c7	中文r13c8	中文r13c9	中文r13c1	1-1月-00	15-8月-72	253	254	255	256	257	258	259
16	15	中文r15c1	中文r15c2	中文r15c3	中文r15c4	中文r15c5	中文r15c6	中文r15c7	中文r15c8	中文r15c9	中文r15c1	1-1月-00	15-8月-72	293	294	295	296	297	298	299
17	16	中文r15c1	中文r15c2	中文r15c3	中文r15c4	中文r15c5	中文r15c6	中文r15c7	中文r15c8	中文r15c9	中文r15c1	1-1月-00	15-8月-72	293	294	295	296	297	298	299
18	17	中文r17c1	中文r17c2	中文r17c3	中文r17c4	中文r17c5	中文r17c6	中文r17c7	中文r17c8	中文r17c9	中文r17c1	1-1月-00	15-8月-72	333	334	335	336	337	338	339
19	18	中文r17c1	中文r17c2	中文r17c3	中文r17c4	中文r17c5	中文r17c6	中文r17c7	中文r17c8	中文r17c9	中文r17c1	1-1月-00	15-8月-72	333	334	335	336	337	338	339
20	19	中文r19c1	中文r19c2	中文r19c3	中文r19c4	中文r19c5	中文r19c6	中文r19c7	中文r19c8	中文r19c9	中文r19c1	1-1月-00	15-8月-72	373	374	375	376	377	378	379
21	20	中文r19c1	中文r19c2	中文r19c3	中文r19c4	中文r19c5	中文r19c6	中文r19c7	中文r19c8	中文r19c9	中文r19c1	1-1月-00	15-8月-72	373	374	375	376	377	378	379

注：以上数据文件与 SpreadJS 测试用例相同，以便后续在架构选型时方便对比

性能数据结果

性能指标	50 万 2.5 万行 * 20 列	100 万 5 万行 * 20 列	500 万 25 万行 * 20 列	1,000 万 50 万行 * 20 列	1,500 万 75 万行 * 20 列
Excel 文件大小	2.554 MB	5.094 MB	25.471 MB	51.332 MB	77.196 MB
导入时间	1,403 ms	1,704 ms	3,550 ms	6,450 ms	9,442 ms
所需内存	287.6 MB	499.9 MB	2,181 MB	2,286 MB	6,567 MB

测试用例 8#：GcExcel 将内存中的表格保存为 Excel 或 CSV 文件

用例说明

此用例为了测试在 GcExcel 中，将内存中的表格示例通过代码导出为 Excel 结果文件，或者是 CSV 格式（纯数据，在后端数据交换中经常使用）。

注：GcExcel for Java 将从 v6.2 中开始支持 SJS 文件格式，因此此报告中不包含 SJS 格式性能数据

用例性能指标

- 导出 Excel/CSV 文件所需时间
- 导出 Excel/CSV 文件过程中申请的 JVM 内存

测试方法

- 加载用例 7# 中的 xlsx 文件后，调用 save 方法保存
- 内存与时间获取与用例 7# 相同

测试数据

- 参见用例 1 中的数据说明

性能数据结果

性能指标（单元格数量）		50 万 2.5 万行 * 20 列	100 万 5 万行 * 20 列	500 万 25 万行 * 20 列	1,000 万 50 万行 * 20 列	1,500 万 75 万行 * 20 列
文件类型	导入 Excel 文件大小	2.554 MB	5.094 MB	25.471 MB	51.332 MB	77.196 MB
Excel	导出时间	2,403 ms	3,985 ms	15,294 ms	28,664 ms	46,692 ms
	所需内存	884 MB	1,688 MB	8,049 MB	16,018 MB	24,169 MB
CSV	导出时间	1,262 ms	1,868 ms	5,749 ms	10,742 ms	15,347 ms
	所需内存	949 MB	1,806 MB	8,047 MB	17,246 MB	25,890 MB

测试用例 9#：GcExcel 对内存中加载的表格进行全量公式函数计算

用例说明

GcExcel 在全栈表格解决方案中常常充当后端计算的功能，因此 GcExcel 对公式函数的计算性能也是产品核心能力之一。

用例性能指标

- 全量计算所需时间
- 计算过程中申请的 JVM 内存总数

测试方法

- 加载用例 3# 中的 xlsx 文件（100 万单元格）后，调用 calculate 方法进行全量计算
- 内存与时间获取与用例 7# 相同

性能数据结果

性能指标	1,000 公式	1 万公式	5 万公式	10 万公式	50 万公式
计算执行时间	51 ms	174 ms	341 ms	467 ms	1,118 ms
所需内存	511 MB	532 MB	627 MB	742 MB	1,781 MB

测试用例 10#：GcExcel 批量打开 Excel 文件，并读取其中区域的值

用例说明

在后端表格文件处理中，经常会碰到处理大量文件的场景，一个典型的应用就是提取批量文件中的数据进行保存和分析。

用例性能指标

- 完成批量文件提取数据所花费时间
- 过程中申请的 JVM 内存总数

测试方法

- 通过循环模拟多次打开文件并读取表格区域数据

测试数据说明

- 通过循环模拟多次打开文件并使用 GcExcel 产品的 importData 接口读取表格区域数据
- 文件及数据区域如下图：


年终资产负债表（含分析数据可视化）

企业名称：　　2019年　　单位：元

资　产	行次	年初数	年末数	资　产	行次	年初数	年末数
流动资产：				流动负债：	34		
货币资金	1	387,383.00	201,957.00	短期借款	35	272,501.00	467,988.00
短期投资	2	256,953.00	248,388.00	应付票据	36	421,544.00	278,198.00
应收票据	3	260,303.00	331,011.00	应付帐款	37	348,470.00	492,556.00
应收帐款	4	428,009.00	452,751.00	预收帐款	38	361,348.00	322,511.00
减：坏帐准备	5	255,385.00	430,099.00	其他应收款	39	291,988.00	411,847.00
应收帐款净额	6	386,302.00	300,890.00	应付工资	40	420,750.00	361,472.00
预付帐款	7	398,729.00	227,128.00	应付福利费	41	438,580.00	264,120.00
其他应收款	8	492,056.00	220,968.00	应交税金	42	350,439.00	313,566.00
存货	9	465,983.00	455,988.00	应付利润	43	433,758.00	491,909.00
待摊其他业务支出	10	314,537.00	273,010.00	其他应付款	44	475,887.00	207,542.00
待摊费用	11	467,398.00	499,555.00	预提费用	45	268,016.00	297,077.00
待处理流动资产净损失	12	271,763.00	203,116.00	一年内到期的长期负债	46	358,451.00	262,762.00
一年内到期的长期债券投	13	316,561.00	264,844.00	其他流动负债	47	409,063.00	414,362.00
其他流动资产	14	451,920.00	488,476.00		48		
流动资产合计	15	5,123,292.00	4,598,682.00	流动负债合计	49	4,751,806.00	4,585,911.00
长期投资：	16		0.00		50		0.00
长期投资	17	310,638.00	474,905.00		51		
固定资产：	18		0.00	长期负债：	52		0.00

性能数据结果

性能指标	100 次	500 次	1,000 次	5,000 次
读取所用时间	1,828 ms	6,413 ms	12,789 ms	56,425 ms
所需内存	111 MB	301 MB	627 MB	742 MB

测试用例 11#：GcExcel 导出文件（PDF / Excel）

用例说明

在后端表格文件处理中，经常以 Excel 作为模板，从数据库中取出批量数据，根据数据产生多份 Excel 或者 PDF 的结果文件，诸如计量检测中的证书生成、自动化报表系统中的多维度报表生成等。

用例性能指标

- 完成批量文件导出数据所花费时间，每条数据产生一个结果文件
- 通过 GcExcel 模板功能，完成单文件导出（每条记录产生一个工作表）
- 过程中申请的 JVM 内存之和

测试方法

- 批量导出：加载模板后，通过向模板指定位置设置数据，导出单独 Excel 或 PDF 文件，代码类似下图所示：

```
@Setup
public void testInit() {
    _workbook = new Workbook("GrapeCity-Internal-Use-Only,GrapeCity-Internal-Use-.......TM9QONycTMxkzOinj");
    // 加载模板
    _filePath = FileHelper.GetResourceFilePath("工资单模板2.xlsx");
    _workbook.open(_filePath);
    String json = null;
    try {
        // 读取测试数据
        InputStream stream = new FileInputStream(FileHelper.GetResourceFilePath("SalaryList100.json"));

        ByteArrayOutputStream result = new ByteArrayOutputStream();
        byte[] buffer = new byte[1024];
        int length;
        while ((length = stream.read(buffer)) != -1) {
            result.write(buffer, 0, length);
        }
        json = result.toString("UTF-8");
    } catch (IOException e) {
        e.printStackTrace();
    }
    _salaryList = new Gson().fromJson(json, new TypeToken<List<SalaryItem>>(){}.getType());
}

@TimeLimit
@Benchmark
public void calculate() {
    IWorksheet sheet = _workbook.getActiveSheet();
    for (SalaryItem item :
            _salaryList) {
        // 初始化以名字
        sheet.getRange("C3").setValue(item.Name);
        sheet.getRange("I1").setValue(item.DateTime);
        sheet.getRange("E6").setValue(item.SalaryInfo.Basic);

        //.......

        sheet.getRange("C23").setValue(item.ManagerName);
        sheet.getRange("J23").setValue(item.WhoMakeSheet);

        _workbook.save("output\\salary\\100\\" + item.Name +".pdf");
    }
}
```

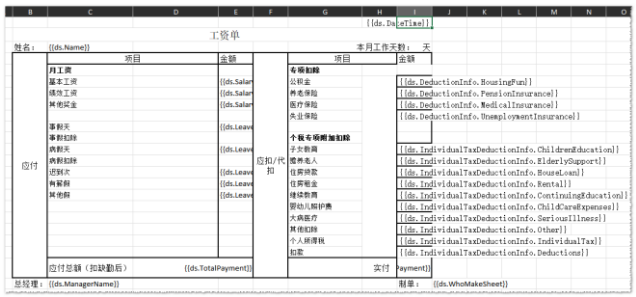
- 单文件导出，示例如下图：

```
@Setup
public void testInit() {
    _workbook = new Workbook( s: "GrapeCity-Internal-Use-Only,GrapeCity-Internal-Use-Only,791172449162884#A0Lx
    _filePath = FileHelper.GetResourceFilePath("工资单模板.xlsx");
    _workbook.open(_filePath);
    String json = null;
    try {
        InputStream stream = new FileInputStream(FileHelper.GetResourceFilePath("SalaryList100.json"));

        ByteArrayOutputStream result = new ByteArrayOutputStream();
        byte[] buffer = new byte[1024];
        int length;
        while ((length = stream.read(buffer)) != -1) {
            result.write(buffer, off: 0, length);
        }

        json = result.toString( charsetName: "UTF-8");
    } catch (IOException e) {
        e.printStackTrace();
    }

    _workbook.addDataSource( s: "ds", new JsonDataSource(json));
}

@TimeLimit
@Benchmark
public void calculate() {
    _workbook.processTemplate();
    _workbook.save( s: "output\\" + "salary_single_100.xlsx");
}
```

测试数据说明

Excel 模板（批量文件）

填充数据示例



Excel 模板（单文件）

性能数据结果

	文件类型	性能指标	100 个文件/表单	500 个文件/表单	1,000 个文件/表单	10,000 个文件/表单
	PDF	导出所有文件时间	8,410 ms	29,983 ms	59,652 ms	617,264 ms
		使用内存	3,520 MB	17,111 MB	34,101 MB	339,605 MB
	Excel	导出所有文件时间	1,927 ms	6,222 ms	13,645 ms	104,615 ms
		使用内存	568 MB	2,528 MB	4,974 MB	48,950 MB
	PDF	导出单文件时间	3,297 ms	9,900 ms	18,623 ms	286,385 ms
		使用内存	728 MB	3,062 MB	5,967 MB	78,546 MB
	Excel	导出单文件时间	1,098 ms	4,258 ms	6,695 ms	187,277 ms
		使用内存	638 MB	2,867 MB	5,620 MB	62,321 MB

注：这里的使用内存指整个执行过程中申请的内存之和，不代表峰值内存消耗

测试用例 12#：GcExcel 对表格进行大量设置/获取操作

用例说明

在用户的场景中，经常会碰到需要大量设置或者获取单元格数据的操作，例如：从后台获取的数据需要填充到表格中，或者需要提取用户上传的 Excel 文件中的数据。

用例性能指标

- 完成不同数量等级单元格设置/获取所需时间
- 设置/获取过程中申请的 JVM 内存之和

测试方法

- 通过提前读取准备好的不同重复度的数据文件（xlsx），使用 getRange().setValue()方法设置数据
- 通过读取准备好的不同重复度的数据文件（xlsx），使用 getRange().getValue()方法读取数据

注：具体测试办法和测试数据，可下载本报告最后的示例数据及测试工程详细了解

性能数据结果

单元格数量	数据不重复程度	设置数据		获取数据	
		执行时间	使用内存	执行时间	使用内存
50 万	100%	14 ms	361.1 MB	217 ms	374.8 MB
	30%	12 ms	282.4 MB	210 ms	294.3 MB
	70%	13 ms	333.2 MB	200 ms	350.2 MB
100 万	100%	17 ms	645.0 MB	250 ms	667.7 MB
	30%	15 ms	488.4 MB	220 ms	511.3 MB
	70%	16 ms	597.8 MB	181 ms	620.6 MB
1000 万	100%	45 ms	5,614.2 MB	574 ms	5,851.3 MB
	30%	54 ms	4,104.9 MB	459 ms	4,338.3 MB
	70%	20 ms	5,132.7 MB	508 ms	5,299.6 MB
1500 万	100%	65 ms	8,604.1 MB	675 ms	8,955.0 MB
	30%	83 ms	6,284.5 MB	650 ms	6,636.8 MB
	70%	74 ms	7,888.9 MB	927 ms	8,240.8 MB

测试用例 13#：GcExcel 对大数量表格进行排序 + 筛选操作

用例说明

模拟用户在对大量数据做过滤及排序的性能，进行过滤筛选及排序。

用例性能指标

- 完成排序 + 筛选操作的代码执行时间

- 设置/获取过程中申请的 JVM 内存之和

测试方法

使用以下代码示例完成排序 + 筛选：

```
@TimeLimit
@Benchmark
public void open() {
    IWorksheet sheet = _workbook.getWorksheets().get(0);
    sheet.getRange( s: "A1:T50000").sort(sheet.getRange( s: "A2:A50000"), SortOrder.Descending, SortOrientation.Columns);
    sheet.getRange( s: "A1:T50000").autoFilter( i: 1,  o: "*17*");
}
```

测试数据

使用测试用例 12# 中的数据

性能数据结果

指标 \ 单元格数量	50 万	100 万	500 万	1,000 万	1,500 万
读取所用时间	422 ms	451 ms	1,249 ms	1,514 ms	2,988 ms
所需内存	386 MB	697 MB	2,995 MB	5,778 MB	8,764 MB

测试用例 14#：边界容量测试（GcExcel）

用例说明

在大型应用系统中，用户想了解产品支持的最大容量，例如：最大表单数量、单元格数量上限、最多包含多少公式计算等，本用例将用代码的方式，不断增加表单、单元格、公式等数量，直到应用崩溃或执行时间超过 30 分钟。

用例性能指标

- 达到停止条件时创建的对象数量
- 停止条件：
  - Java 应用因内存不足而崩溃
  - 执行时间达到 30 分钟

性能数据结果

注：除空表单达到执行 30 分钟停止，其他指标均未达到最大内存后应用崩溃

	空表单	浮点数单元格	字符单元格	日期单元格	公式单元格
创建数量（个）	69,957	5 亿 3 千万	10 亿 7 千万	5 亿 3 千万	1 亿 3 千万

测试用例 15#：对典型财务表格文件：财务三张表，分别测量导入、导出所需时间和内存占用

用例说明

类似在 SpreadJS 产品用进行的测试，也使用 GcExcel 对最常见的财务表格文件，进行导入、导出操作，获取执行数据以及代码执行内存。

具体测试文件请参考测试用例 6#

用例性能指标

- 导入典型文件所需时间
- 导出 SSJSON 文件使用的时间、内存使用及文件体积

性能数据结果

典型表格文件	导入 Excel		导出 SSJSON		
	耗时	使用内存	耗时	使用内存	导出体积
利润表模板	305 ms	153.7 MB	153 ms	78.2 MB	30 KB
年终资产负债表(含分析)	690 ms	85.8 MB	225 ms	113.8 MB	110 KB
现金流量表模板	514 ms	67.5 MB	248 ms	110.2 MB	149 KB

典型应用场景架构推荐（性能角度）

通过下表可以快速了解典型应用场景中对应的性能测试用例：

测试用例编号 / 应用场景		企业自定义报表	计量检测与检定	金融保险精算	实验室管理	自动化报表	企业预算与决算	数据指标补录	项目投资分析测算
SpreadJS	1#	✓	✓		✓		✓	✓	
	2#	✓	✓	✓	✓	✓	✓	✓	✓
	3#			✓			✓	✓	✓
	4#	✓					✓	✓	
	6#	✓		✓			✓	✓	
GcExcel	7#		✓	✓	✓	✓		✓	✓
	8#	✓				✓			✓
	9#	✓		✓		✓	✓		✓
	10#		✓	✓	✓	✓		✓	
	11#	✓	✓	✓	✓	✓	✓	✓	
	12#	✓	✓		✓	✓	✓		✓
	13#	✓				✓	✓		✓
	14#	✓		✓		✓			

注：以上表格中的 1# 代表测试用例编号，如：3# 表示**测试用例 3#：使用 SpreadJS 进行公式计算性能测试**

场景：企业自定义报表

场景特点及和性能相关的痛点

- 在浏览器中加载报表模板的响应速度
- 当企业报表中存在大量公式计算，前端修改数据后的计算性能
- 如果是通过后台传输数据，前端完成报表模板与数据的绑定，需要关注前端对大数据量的加载性能
- 如果是后端完成报表数据填充，需要关注后端对模板的加载速度，以及模板中加载数据的性能

推荐架构方案

- 方案 1：纯前端 SpreadJS 完成报表模板定义、数据加载、文件导出、打印
  - 适用条件：
    - 针对企业轻量化的日常报表，数据量（行数）在 1 万行以内的报表场景
    - 报表中没有高负荷公式计算（超过 1 万公式单元格）
    - 对前端 PDF 导出没有很高的性能要求（1 秒内完成）
- 方案 2：
  - 纯前端 SpreadJS 完成报表模板定义，数据加载功能
  - 使用 GcExcel 在后端完成文件导出（Excel 或 PDF）、服务端存储、向网络打印机发送批量打印命令等功能
  - 适用条件：
    - 生产制造、物流、零售、电商等企业中报表数据量比较大，而且产生报表的频次高
    - 有个别报表存在高负荷公式计算场景，例如：日报表统计分析、预测等
    - 有大量 PDF、Excel 导出需求，而且对导出性能要求高
    - 对业务数据敏感，不方便网络传输

场景：计量检测及检定

场景特点及和性能相关的痛点

- 系统中存在大量表格模板，需要经常切换、打开、复用模板
- 对批量打印、批量 PDF 导出有高性能要求

推荐架构方案

- 纯前端 SpreadJS 完成模板定义，预览、修改
- 后端 GcExcel 完成文件数据入库、批量导出、批量打印
- 适用条件：
  - 有批量文件导出、批量打印的强需求
  - 对导出的性能有很高的要求
  - 有大量的模板复用、合并需求

场景：金融保险精算

场景特点及和性能相关的痛点

- 表格文件为高负荷公式函数计算
- 计算模板多为固定只读模板，仅对某几个关键数据或者某个表单的基础数据进行调整
- 对表格修改数据后的计算性能有高性能需求，如在 1~2 秒内获得计算后结果，或生成对应的分析数据
- 计算模板为敏感信息，对最终用户隔绝

推荐架构方案

- 纯前端 SpreadJS 完成加载模板，或仅加载可修改部分的表单，供用户修改及查看
- 后端 GcExcel 获取用户修改后，完成计算，并向前端发送计算后结果，供前端展示

场景：数据指标补录

场景特点及和性能相关的痛点

- 整个系统旨在完成对应指标数据的收集、审核、整理、汇总
- 对最终用户提供多种数据填报表单
- 常存在需要后端处理大量 Excel 数据文件的需求
- 需要对用户填报的数据进行处理，例：合并、修改、计算等批量操作

推荐架构方案

- 纯前端 SpreadJS 完成填报表单的设计与展示
- 后端 GcExcel 获取用户提交的表单后，抽取数据并入库
- 后端 GcExcel 批量操作大量文件，进行数据或文件的合并

如需更多具体应用场景的方案推荐，欢迎联系我们：400-657-6008，或在线预约技术顾问

对性能测试可能的疑问？

问题：为什么不使用浏览器内存作为性能指标，而是 JS 内存？

答：浏览器内存因浏览器在不同执行阶段执行垃圾回收策略不稳定，无法稳定获取一个可比较的值，同时考虑到 JS 内存更能反应 SpreadJS 产品在执行相关代码前后的内存变化，因此本次性能测试报告中前端内存指标采用 JS 内存而不是浏览器内存。

问题：为什么可以设置 1,500 万单元格，但无法导入同样数量单元格的 Excel 文件？

答：Excel 的本质是一个 XML 的压缩文件，所以整个读取文件的前提是能够将此文件解压，并读取其中的 XML 内容。受限于前端技术限制，目前的解压缩方案会将 XML 读取为 String 类型内容，且这个过程并非流

式读取。那么一旦当 1500 万单元格的 Excel 文件中包含大量的非重复内容，会导致读取的 XML 内容超过了浏览器的可表达字串长度限制，因此会出现浏览器因内存不足或 JS 字符串长度超长而崩溃或出错。

测试用例及工程下载

- 测试用文件：导入的 Excel 文件、数据填充模板、模拟数据、典型财务表格文件 > 下载地址
- 性能测试工程：SpreadJS / GcExcel

欢迎您对此测试报告反馈您的意见或建议

反馈链接：https://www.wenjuan.com/s/UZBZJvl735H/#

也可扫描下方二维码进行反馈：