

小小导出，我大前端足矣！

一、问题剖析

那是一个倾盆大雨的早上，花瓣随风雨落在我的肩膀上，是五颜六色的花朵。

我轻轻抚摸着它，随后拨开第一朵花瓣，她不爱我。

拨开第二朵，她爱我。

正当我沉迷于甜蜜的幻想中，后端小白👤喊道：这个导出你前端应该就能做的吧！

👤♂那是自然，有什么功能是我大前端做不了的，必须得让你们大开眼界。

二、为什么导出要前端做？

前端导出的场景：

1. 轻量级数据：如果要导出的表格数据相对较小，可以直接在前端生成和导出，避免服务器端的处理和通信开销。
2. 数据已存在于前端：如果表格数据已经以 JSON 或其他形式存在于前端，可以直接利用前端技术将其导出为 Excel、CSV 或其他格式。
3. 实时生成/计算：如果导出的表格需要根据用户输入或动态生成，可以使用前端技术基于用户操作实时生成表格，并提供导出功能。
4. 快速响应：前端导出表格可以提供更快的响应速度，避免等待服务器端的处理和下载时间。

后端导出的场景：

1. 大量数据：如果要导出的表格数据量很大，超过了前端处理能力或网络传输限制，那么在服务器端进行导出会更高效。
2. 安全性和数据保护：敏感数据不适合在前端暴露，因此在服务器端进行导出可以更好地控制和保护数据的安全。
3. 复杂的业务逻辑：如果导出涉及复杂的业务逻辑、数据处理或数据查询，使用服务器端的计算能力和数据库访问更合适。
4. 跨平台支持：如果需要在支持多个前端平台（如 Web、移动应用等），将导出功能放在服务器端可以提供一致的导出体验。

三、讲解一下在前端做的导出

xlsx、xlsx-style

如果是只做表格导出: www.npmjs.com/package/xls...

如果导出要包含样式: www.npmjs.com/package/xls...

```
1 import XLSX from "xlsx";
2
3 exportData() {
4   let tableName = '表格'
5
6   if(!getVal(this.dataList, 'length')){
7     this.$message.info("暂时数据");
8     return
9   }
10
11
12   // 处理头部
13   let headers = {
14     "B2": "字段-B2",
15     "E2": "字段-E2",
16   }
17   const props = [ "B2", "E2" ]
18   let tmp_dataListFilter = [
19     {
20       "B2": "字段-B2",
21       "E2": "字段-E2",
22     },
23     {
24       "E2": "2",
25       "B2": "2",
26     }
27   ]
28
29   tmp_dataListFilter.unshift(headers) // 将表头放入数据源前面
30   let wb = XLSX.utils.book_new();
31   let contentWs = XLSX.utils.json_to_sheet(tmp_dataListFilter, {
32     skipHeader: true, // 是否忽略表头,默认为false
33     origin: "A2" // 设置插入位置
34   });
35   // 单独设置某个单元格内容
36   contentWs["A1"]={
37     t:"s",
38     v:tableName,
39   };
40   // 设置单元格合并! merges为一个对象数组, 每个对象设定了单元格合并的规则,
41   // [{s:{r:0,c:},e:{r:0,c:2}}为一个规则, s:起始位置, e:结束位置, r:行, c:列
42   contentWs["!merges"]=[{ s:{r:0,c:0 },e:{r:0,c:props.length - 1 }}]
43
```

```

44 // 设置单元格的宽度
45 contentWs["!cols"] = []
46 props.forEach(p => contentWs["!cols"].push({wch: 35}))
47 XLSX.utils.book_append_sheet(wb, contentWs, tableName) // 表格内的下面的tab
48 XLSX.writeFile(wb, tableName + ".xlsx"); // 导出文件名字
49 },,

```

package.json

```

1 "xlsx": "^0.15.5",
2 "xlsx-style": "^0.8.13"

```

大概效果如下：

	A	B
1	汇总表	
2		1
3	灌灌灌灌	7
4		
5		
6		
7		
8		
9		
0		
1		
2		

感觉前端导出也很容易。

哦哦，那你别高兴太早。

四、需求升级：单元格要居中和加粗。

xlsx

尝试使用xlsx-style设样式。

官方文档：github.com/rockboom/SheetJS

文档说给单元格设置s为对象

v	原始值(查看数据类型部分获取更多的信息)
w	格式化文本(如果可以使用)
t	内行: b Boolean, e Error, n Number, d Date, s Text, z Stub
f	单元格公式编码为A1样式的字符串(如果可以使用)
F	如果公式是数组公式, 则包围数组的范围(如果可以使用)
r	富文本编码 (如果可以使用)
h	富文本渲染成HTML (如果可以使用)
c	与单元格关联的注释
z	与单元格关联的数字格式字符串(如果有必要)
l	单元格的超链接对象 (.Target 长联接, .Tooltip 是提示消息)
s	单元格的样式/主题 (如果可以使用)

如果 `w` 文本可以使用, 内置的导出工具(比如CSV导出方法)就会使用它。要想改变单元格的值, 在打算导出之前确保删除 `cell.w` (或者设置 `cell.w` 为 `undefined`)。工具函数会根据数字格式(`cell.z`)和原始值(如果可用)重新生成 `w` 文本。

真实的数组公式存储在数组范围中第一个单元格的 `f` 字段内。此范围内的其他单元格会省略 `f` 字段。

数据类型

原始值被存储在 `v` 值属性中, 用来解释基于 `t` 类型的属性。这样的区别允许用于数字和数字类型文本的展示。下面有6种有效的单元格类型。

Type	Description
b	Boolean: 值可以理解为JS <code>boolean</code>
e	Error: 值是数字类型的编码, 而且 <code>w</code> 属性存储共同的名称 **
n	Number: 值是JS <code>number</code> **
d	Date: 值是 JS <code>Date</code> 对象或者是被解析为Date的字符串 **
s	Text: 值可以理解为 JS <code>string</code> 并且被写成文本 **

```

1 let contentWs = XLSX.utils.json_to_sheet(tmp_dataListFilter, {
2   skipHeader: true, // 是否忽略表头,默认为false
3   origin: "A2", // 设置插入位置
4 });
5 // 单独设置某个单元格内容
6 contentWs["A1"] = {
7   t: "s",
8   v: tableName,
9   s: { // 这个是关键s
10     font: { bold: true },
11     alignment: { horizontal: 'center' }
12   }
13 };

```

发现设置无效。

有人说要改xlsx、xlsx-style源码：

大概的意思是：修改xlsx.extendscript.js、xlsx.full.min.js更改文件变量。

发现仍然无效。

使用binary方式保存

1. 首先保存的时候 type要改成 binary方式
2. 保存的时候需要使用 xlsx-style模块

```
1  var writingOpt = {
2    bookType: 'xlsx',
3    bookSST: true,
4    type: 'binary'    // <--- 1.改这里
5  }
6
7
8  /*
9   2.  type:'array'改为'binary' 后因为下面代码会报错，打不开excel
10  new Blob([wbout], { type: 'application/octet-stream' })
11  要文本转换成数组缓存后再生成二进制对象
12  */
13
14  // 添加String To ArrayBuffer
15  function s2ab(s) {
16    var buf = new ArrayBuffer(s.length);
17    var view = new Uint8Array(buf);
18    for (var i = 0; i < s.length; i++) {
19      view[i] = s.charCodeAt(i) & 0xFF;
20    }
21    return buf;
22  }
23
24  let blob = new Blob([s2ab(wbout)], { type: 'application/octet-stream' })
25
26  FileSaver.saveAs(blob, exportName)
```

可以下载了。但依然样式没起作用。

使用 xlsx-style 模块生成文件

首先安装模块

```
1 npm install xlsx-style
```

在项目里安装报好多错误直接强制安装，不检查依赖。

```
1 npm install xlsx-style -force
```

安装完成后 找不到cptable模块会报错

报错内容如下：

```
1 ./node_modules/xlsx-style/dist/cpexcel.js Module not found: Error: Can't
  resolve './cptable' in
2
```

这个问题在vue.config.js里配置一下就可以解决。

其他框架自己找找方法，反正只要不让他报错能启动就行。

```
1 module.exports = {
2   // ...其他配置省略
3   configureWebpack: {
4     // ...其他配置省略
5     externals:{
6       './cptable':'var cptable'
7     },
8   },
9 }
```

安装完xlsx-style后改代码

```
1 import XLSX2 from "xlsx-style";    // 1. 引入模块
2
3 // 2. 使用`xlsx-style` 生成。 XLSX.write => XLSX2.write
4 var wbout = XLSX2.write(wb, writingOpt)
```

仍然无效。

总结xlsx

大概的意思是说：默认不支持改变样式，想要支持改变样式，需要使用它的收费版本。

本着勤俭节约的原则，很多人使用了另一个第三方库：**xlsx-style[4]**，但是使用起来极其复杂，还需要改 node_modules 源码，这个库最后更新时间也定格在了 6 年前。还有一些其他的第三方样式拓展库，质量参差不齐。

使用成本和后期的维护成本很高，不得不放弃。

ExcelJS

ExcelJS终于可以了

ExcelJS[5] 周下载量 450k，github star 9k，并且拥有中文文档，对国内开发者很友好。虽然文档是以 README 的形式，可读性不太好，但重在内容，常用的功能基本都有覆盖。

最近更新时间是6个月内，试用了一下，集成很简单，再加之文档丰富，就选它了。

```
1 npm install exceljs
2 npm install file-saver // 下载到本地还需要另一个库：file-saver
```

基本操作

```
1 //导入ExcelJS
2 import ExcelJS from "exceljs";
3
4 //下载文件
5 download_file(buffer, fileName) {
6     console.log("导出");
7     let fileURL = window.URL.createObjectURL(new Blob([buffer]));
8     let fileLink = document.createElement("a");
9     fileLink.href = fileURL;
10    fileLink.setAttribute("download", fileName);
11    document.body.appendChild(fileLink);
12    fileLink.click();
13 }
```

导出xlsx表格的代码

```
1 //下面是导出的函数
2 async export() {
3     const workbook = new ExcelJS.Workbook();
4     const worksheet = workbook.addWorksheet("Sheet1");
5     //这里是数据列表
6     const data = [
7         { id: 1, name: "艾伦", age: 20, sex: "男", achievement: 90 },
```

```

8           { id: 2, name: "柏然", age: 25, sex: "男", achievement: 86 },
9       ];
10      // 设置列,这里的width就是列宽
11      worksheet.columns = [
12          { header: "序号", key: "id", width: 10},
13          { header: "姓名", key: "name", width: 10 },
14      ];
15
16      // 批量插入数据
17      data.forEach(item => worksheet.addRow(item));
18
19      // 写入文件
20      const buffer = await workbook.xlsx.writeBuffer();
21      //下载文件
22      this.download_file(buffer, "填报汇总.xlsx");
23  }

```

设置行高和列宽

列宽上面已经有了,这里说明一下行高怎么设置

worksheet.getRow(2).height = 30;

合并单元格

worksheet.mergeCells("B1:C1");

自定义表格样式

```

1  //设置样式表格样式, font里面设置字体大小, 颜色(这里是argb要注意), 加粗
2  //alignment 设置单元格的水平和垂直居中
3  const B1 = worksheet.getCell('B1')
4  B1.font = { size: 20, color:{ argb: 'FF8B008B' }, bold: true }
5  B1.alignment = { horizontal: 'center', vertical: 'middle' }

```

ExcelJS实战

```

1  import ExcelJS from "exceljs";
2
3  //下载文件
4  download_file(buffer, fileName) {
5      console.log("导出");
6      let fileURL = window.URL.createObjectURL(new Blob([buffer]));
7      let fileLink = document.createElement("a");
8      fileLink.href = fileURL;
9      fileLink.setAttribute("download", fileName);

```



```
10     document.body.appendChild(fileLink);
11     fileLink.click();
12 },
13 async exportClick() {
14     const loading = this.$loading({
15         lock: true,
16         text: "数据导出中,请耐心等待!",
17         spinner: "el-icon-loading",
18         background: "rgba(0, 0, 0, 0.7)",
19     });
20
21     this.tableData = [
22         { a: 1, b: 2 }
23     ]
24     const enterpriseVisitsColumns = [
25         {
26             prop: "a",
27             label: "银行",
28         },
29         {
30             prop: "b",
31             label: "企业数",
32         }
33     ]
34
35     // 表格数据: this.tableData
36     if (!(this.tableData && this.tableData.length)) {
37         this.$message.info("暂无数据");
38         loading.close();
39         return;
40     }
41
42     let tableName = this.tableName;          // 表格名
43     const workbook = new ExcelJS.Workbook();
44     const worksheet = workbook.addWorksheet(tableName);
45     const props = enterpriseVisitsColumns();
46     //这里是数据列表
47     const data = this.tableData;
48     // 设置列,这里的width就是列宽
49     let arr = [];
50     props.forEach((p) => {
51         arr.push({
52             header: p.label,
53             key: p.prop,
54             width: 25,
55         });
56     });
```

```

57     worksheet.columns = arr;
58
59     // 插入一行到指定位置, 现在我往表格最前面加一行, 值为表名
60     const rowIndex = 1; // 要插入的行位置
61     const newRow = worksheet.insertRow(rowIndex);
62     // 设置新行的单元格值
63     newRow.getCell(1).value = tableName; // 值为表名
64
65     // 批量插入数据, 上面插一条, 这里就是从第二行开始加
66     data.forEach((item) => worksheet.addRow(item));
67
68     // 设置样式表格样式, font里面设置字体大小, 颜色(这里是argb要注意), 加粗
69     // alignment 设置单元格的水平和垂直居中
70     // const B1 = worksheet.getCell("B1");
71     // B1.font = { size: 20, color: { argb: "FF8B008B" }, bold: true };
72     // B1.alignment = { horizontal: "center", vertical: "middle" };
73
74     // 合并单元格, 就是把A1开始到J1的单元格合并
75     worksheet.mergeCells("A1:J1");
76
77     // 批量设置所有表格数据的样式
78     worksheet.eachRow((row, rowNumber) => {
79         let size = rowNumber == 1 ? 16 : rowNumber == 2 ? 12 : "";
80         // 设置表头样式
81         row.eachCell((cell) => {
82             cell.font = {
83                 size,
84                 // color: { argb: 'FF8B008B' },
85                 bold: true,
86             };
87             cell.alignment = { horizontal: "center", vertical: "middle" };
88         });
89
90         // 设置所有行高
91         row.height = 30;
92     });
93
94     // 写入文件
95     const buffer = await workbook.xlsx.writeBuffer();
96     // 下载文件
97     this.download_file(buffer, tableName + ".xlsx");
98
99     loading.close();
100 },

```

导出功能并不是说都是前端或者后端实现，要具体情况，具体分析，我相信哪方都可以做，但谁适合做，这个才是我们需要去思考的。

就如同我们项目中，该例子后面也是前端实现的，大数据分页当然还是得后端同学来实现较好。

如果有其他更好的方法也欢迎评论区见，这里提供的只是诸多方法之一。

最后，祝君能拿下满意的offer。