SmartPad4i

SmartPad4i ExcelJSで簡単！ Excel活用テクニック

株式会社ミガロ.
プロダクト事業部　技術支援課
國元 祐二

略　歴

生年月日:1979年3月27日
最終学歴:2002年　追手門学院大学
　　文学部アジア文化学科卒業
入社年月:2010年10月　株式会社ミガロ. 入社
社内経歴:
2010年10月　RAD事業部(現プロダクト事業部)
　　配属

現在の仕事内容:

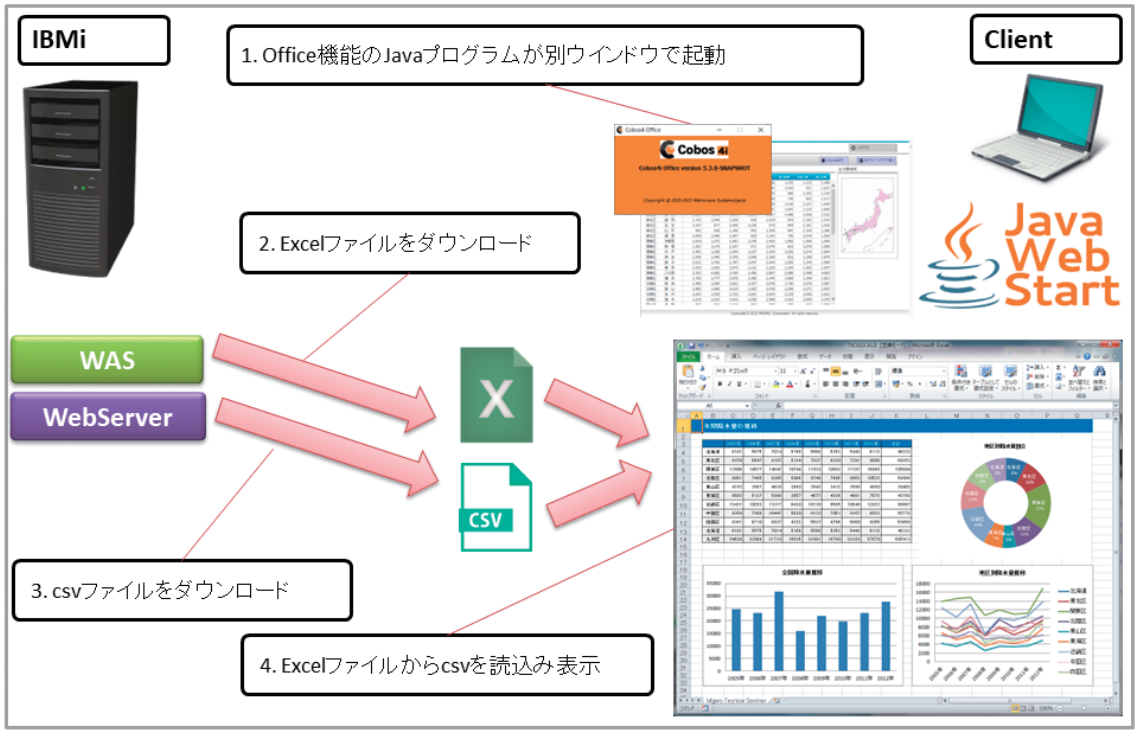
Cobos4i(SP4i)、Valenceの製品試験やサポート業務、導入支援などを担当している。

1.はじめに

SmartPad4i(Cobos4i)には、IBMiからExcelファイルをダウンロード後、CSVファイルと連携させてExcel/Wordを表示できるOffice機能が存在する。**【図1】**

Office機能はRPGを記述することでクライアント端末にExcelファイルを作成できる便利な機能である。

図 1　SmartPad4i(Cobos4i)　Office機能



Office機能はSmartPad4i(Cobos4i)の最新版では、Java Web Startで実装されている。
Java Web StartはJava11以降で廃止となったため、Office機能を使用するにはJava8(※1)をクライアント端末にインストールする必要がある。

(※1)2023年9月現在、Javaはバージョン21まで存在しており、Java8,Java11,Java17,Java21のみサポートが継続されている。

Java8は2030 年 12 月までサポートされるため、Office機能は使用し続けて頂けるが、クライアント端末にJava8を導入する必要があるため、動作環境が制限される。
また、Office機能はcsvファイルを表データとして取込むため、セル単位のレベルでデータをExcelファイルへ出力できない。

Webアプリケーションで Excelファイルを出力する場合、以前はサーバーサイドのプログラム開発(Java,PHP ,Perl等)が必要であった。多言語でのプログラム開発の知識や、サーバー環境の構築が必要なため、実装にはハードルが高い。
しかし、OSS(オープンソースソフトウェア)のExcelJSを使用すると簡単にExcelファイルが作成できる。
JavaScriptの実装は必要であるが、誰でも簡単に利用可能だ。
JavaScriptのみでExcelファイルが作成できれば、サーバー環境の構築や、多言語でのプログラム開発の必要はない。クライアント環境には、JavaScriptが動作するブラウザだけで済む。
また、ExcelJSの場合、セル単位でデータの出力が制御できるため、任意のレイアウトでExcelファイルを作成することが可能だ。さらに、JavaScriptはHTMLに追加するだけで動作するため、SmartPad4i(Cobos4i)に組込むことも容易である。

そこで、本稿ではExcelJSでExcelファイルを作成する方法について紹介する。

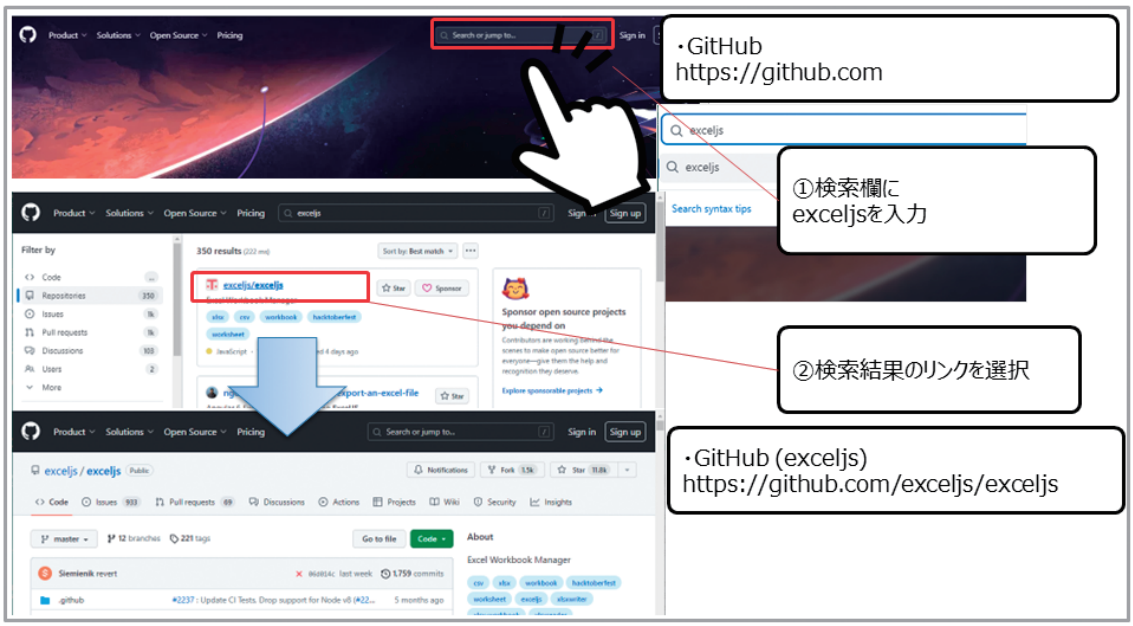
2.JavaScriptでExcelファイルを作成する方法

はじめに、JavaScriptでExcelファイルを作成できるOSS(オープンソースソフトウェア)のExcelJSについて説明する。

2-1. ExcelJSとは

ExcelJSはJavaScriptを使用してExcelファイルを操作するライブラリで、Node.js環境(サーバーサイド)で動作し、Excelファイルの生成/編集が可能である。【図2】

図 2 ExcelJS



● ExcelJS
https://github.com
検索欄にexceljsを入力して検索
または、https://github.com/exceljs/exceljs

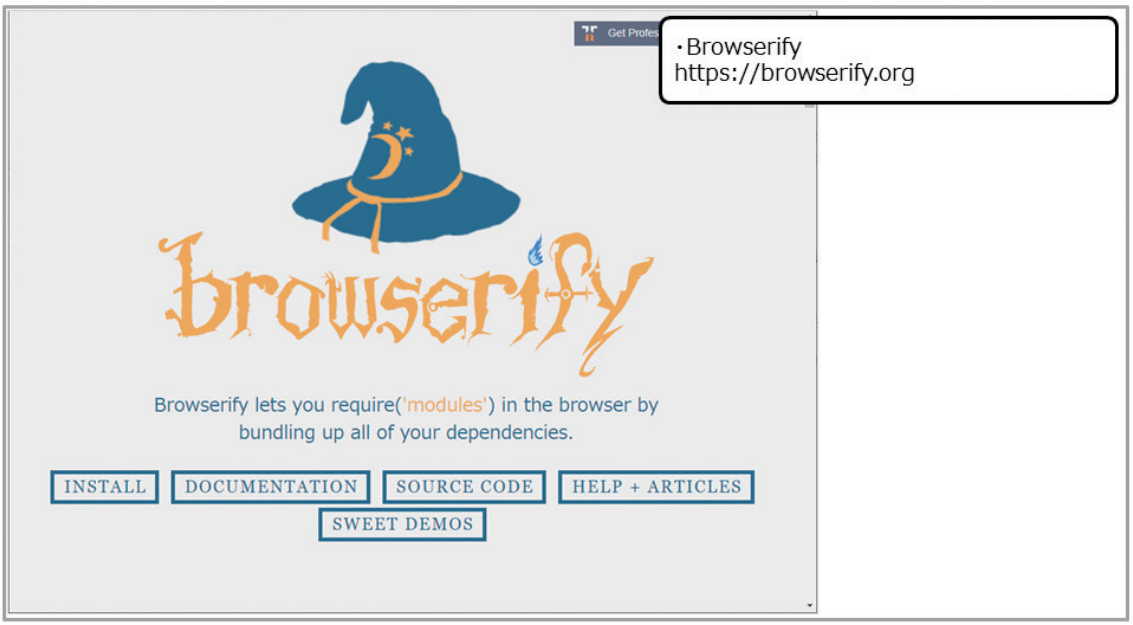
ExcelJSは通常、サーバーサイドで使用するライブラリであるが、Browserifyで変換されたJavaScriptファイルを使用することで、クライアントサイドのJavaScriptでも利用可能である。

2-1-1. Browserifyとは

BrowserifyはJavaScriptで開発をする際に使用するツールの一つで、Node.jsのモジュールをブラウザ上で実行できる形式に変換するためのツールである。**【図3】**

● Browserify
https://browserify.org

図 3 Browserify



Node.jsはサーバーサイドのJavaScript実行環境であるため、同じJavaScriptをブラウザ上で実行しても動作しない。Browserifyはこの制約を解消し、Node.jsで作成されたモジュールをブラウザ上で動作するものに変換することができる。

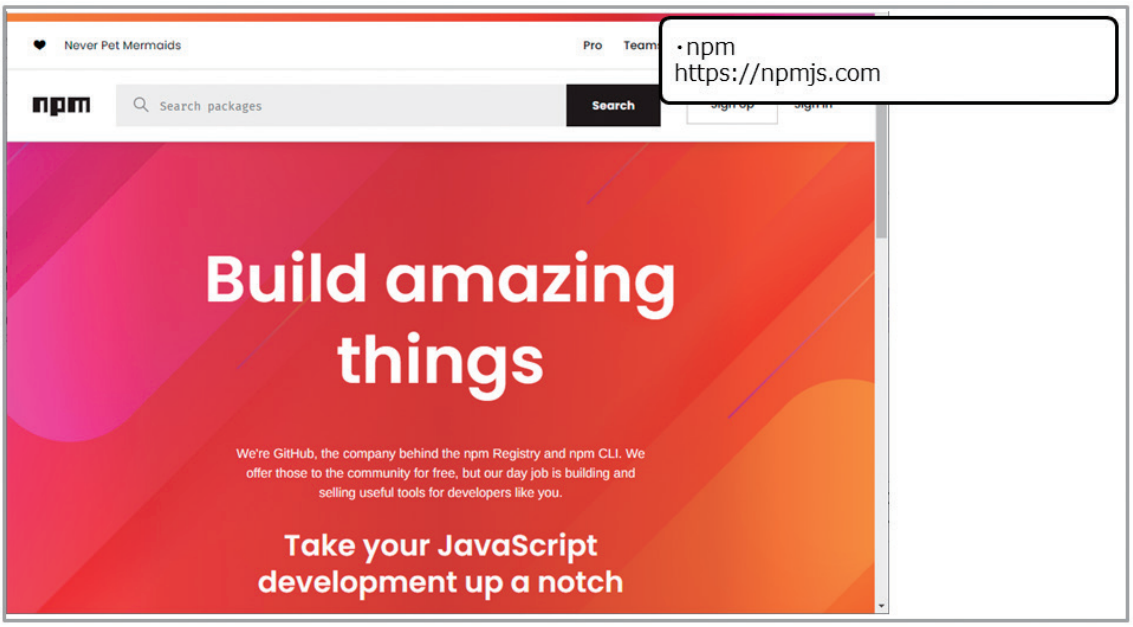
2-2. ExcelJSの導入方法

ExcelJSはnpm（Node Package Manager）と呼ばれる、パッケージ管理システムを利用してダウンロードする。npmは、Node.jsプロジェクトで使用されるさまざまなパッケージ（ライブラリやモジュール）を管理し、インストール、アップデート、削除などの操作を簡単に行えるようにするツールだ。

npmはNode.jsに含まれているため、Node.jsをインストールするとnpmもインストールされる。**【図4】**

● npm
https://npmjs.com

図 4 npm

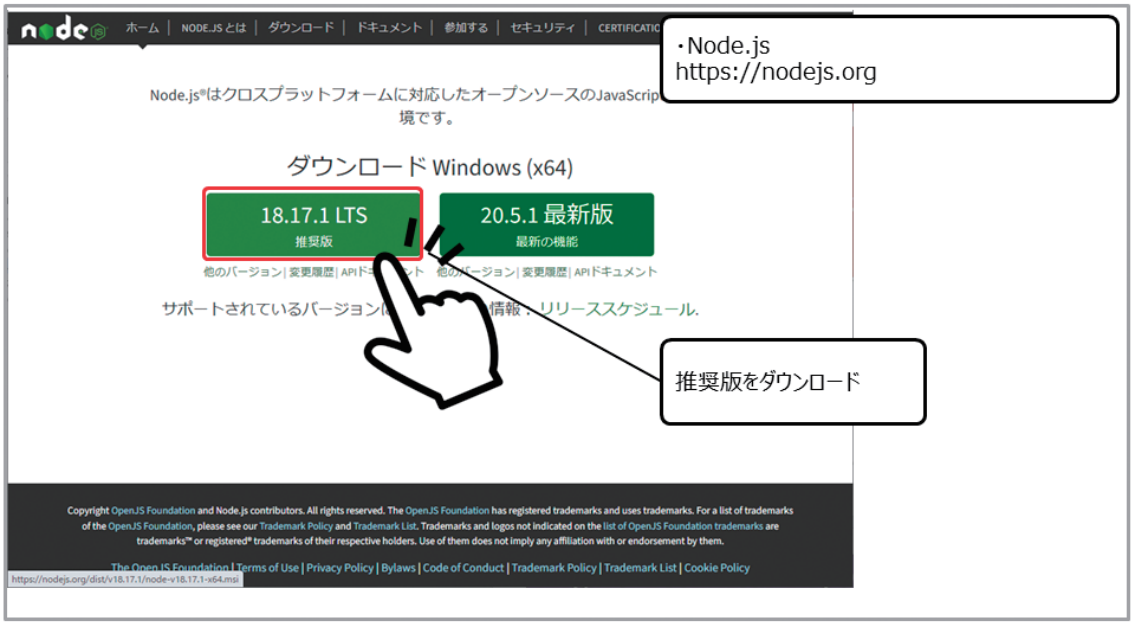


2-2-1. Node.jsの導入

Node.jsは、公式サイトからダウンロードしたインストーラーを実行することで導入可能である。**【図5】**

● Node.js
https://nodejs.org

図 5　Node.js



では、次の項で、npmを使用してExcelJSのファイルを取得する手順を説明する。

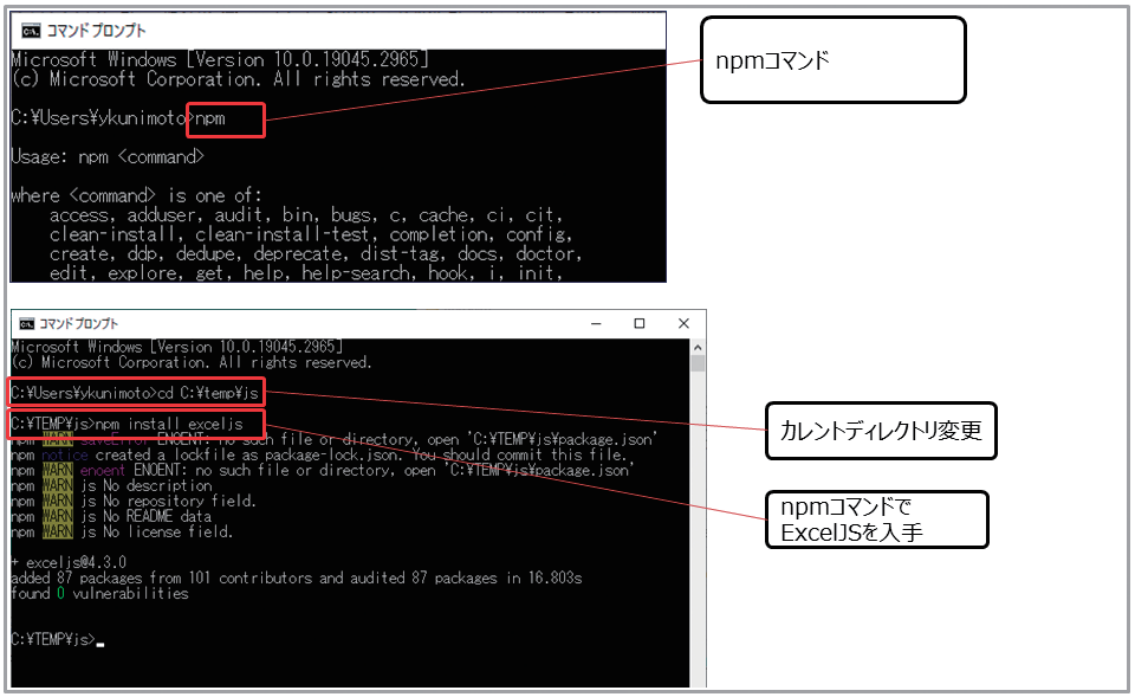
2-2-2. ExcelJSパッケージの導入

Node.jsのインストール後、コマンドプロンプトから、'npm'コマンドを入力することで、npmの導入を確認できる。

npm導入を確認した後、'cd'コマンドを使用して、ExcelJSパッケージをダウンロードするフォルダにカレントディレクトリを移動する。

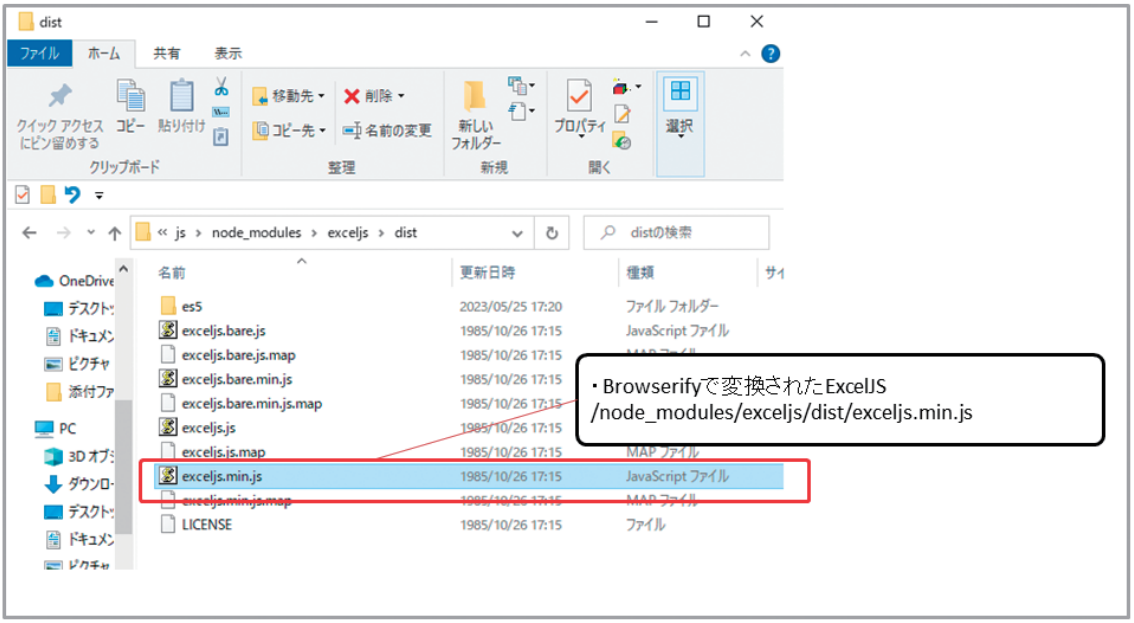
例えば、ExcelJSをダウンロードしたいフォルダがC:\temp\jsである場合、「cd　C:\temp\js」のコマンドを実行する。カレントディレクトリを変更後、次のコマンド'npm install　exceljs'を実行することで、ExcelJSパッケージがダウンロードされる。**【図6】**

図 6 npmコマンド



カレントディレクトリには、ExcelJSパッケージに関連する多くのファイルがダウンロードされる。ExcelJSパッケージには、あらかじめbrowserifyでまとめられたJavaScriptファイルが含まれており、クライアントサイドでExcelJSを使用する場合には、JavaScriptファイル(/node_modules/exceljs/dist/exceljs.min.js)を使用する【図7】。

図 7 Browserifyで変換されたExcelJS

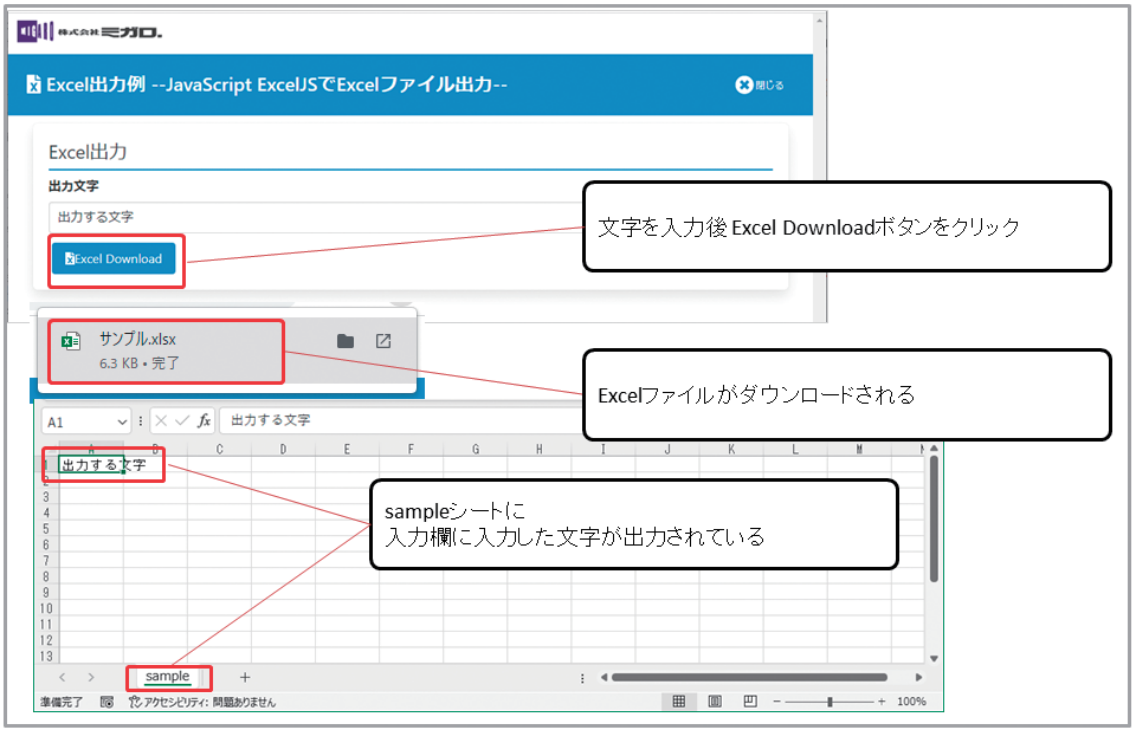


このJavaScriptファイルをブラウザ上で読込むことで、ExcelJSの機能をクライアントサイドで利用できるようになる。

2-3. ExcelJSの利用方法

ExcelJSの利用方法を簡単なサンプルアプリSMP010を元に説明する。SMP010では、HTML（SMP010.html）に配置したボタンをクリック時に、入力欄の文字列をExcelファイルに書込み、Excelファイルを作成してダウンロードする。**【図8】**

図 8　ExcelJSを使用した簡単な例 SMP010



2-3-1. exceljs.min.jsの配置と設定

前項でnpmから取得した'exceljs.min.js'をSmartPad4i（Cobos4i）環境に配置する。本稿では、サンプルのアプリを'/smartpad4i/html/SP4IREP23'ディレクトリに配置していると仮定し、同ディレクトリに新しい'js'フォルダを作成後'exceljs.min.js'を配置する。

次に、配置した'exceljs.min.js'を外部参照する設定をHTMLに追加する。**【ソース1】**

ソース 1

exceljs.min.jsの読込み（SMP010.HTML HTML 外部JavaScript）

```
～省略～
  <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-
polyfill/6.26.0/polyfill.js"></script>
  <script src="../../../smartpad4i/html/SP4IREP23/js/exceljs.min.js"></script>
  </body>
</html>
```

外部読込みのJavaScriptはHTML内のどこに追加しても動作するが、画面表示速度を向上させるために、通常は'</body>'タグの前に記述することを推奨している。

また、ExcelJSを読込む前に、CDNから'polyfill.js'を読込む。'polyfill.js'は、新しい機能やAPIを古いブラウザ環境でも動作させるためのコードであり、実行に必要な機能を持たない環境下では、不足機能を模倣したコードを実行することにより、ブラウザの互換性を保つ役割を果たす。

2-3-2.ボタンの配置と実行処理

Excelファイルをダウンロードするためのボタンを配置する。ボタンはアンカータグで設定し、ボタンの要素を識別するために、タグへカスタムデータ属性'data-exceldownload="true"'を設定する。**【ソース2】**

ソース 2

ボタンの配置（SMP010.HTML HTML タグ追加）

```
<a class="button is-info load-b ml-1" data-exceldownload="true" >
  <i class="fa fa-file-excel"></i>Excel Download
</a>
```

次に、ボタンをクリックした際の処理をJavaScriptで追加する。**【ソース3】**

ソース 3

ボタンクリック時の処理（SMP010.HTML内 JavaScript）

```
<script>
  /**
   * SmartPad4i(Cobos4i) 初期処理用のinitpage関数
   */
  function initpage(){
    // ダウンロード用ボタンの取得
    var excelButton = document.querySelector('[data-exceldownload]');
    /**
     * ダウンロードボタン クリックイベント
     */
    excelButton.addEventListener( "click" , function () {
      createExcel();
    } , false );

    /**
     *  xcelファイルを作成する非同期関数
     */
    async function createExcel(){
      // 新しいworkbookを作成
      const workbook = new ExcelJS.Workbook();
      // シート名 sampleのworksheetを追加
      workbook.addWorksheet("sample");
      // シート名 sampleのworksheetの取得
      const worksheet = workbook.getWorksheet("sample");
      // 1行目の1列目に入力欄(id="INP01")の文字を設定
      worksheet.getCell(1, 1).value =
         SP4i.getElementById('INP01').value;
      // 書込みした内容をwriteBufferメソッドで出力
      const uint8Array = await workbook.xlsx.writeBuffer();
      // Blobオブジェクトの作成
      const blob = new Blob([uint8Array],
        {type: 'application/octet-binary'});
      // ダウンロードするためにblobのURL作成
      const url = window.URL.createObjectURL(blob);
      // アンカーを作成してURL設定
      const a = document.createElement('a');
      a.href = url;
      // ダウンロードするファイル名を指定してダウンロード
      a.download = 'サンプル.xlsx';
    } a.click();
    }   a.remove();
  </
</script>
body>
```

①
②
③
④

■【ソース3】①

まず、【ソース3】①の処理でカスタムデータ属性'data-exceldownload'が設定された要素を取得する。'document.querySelector'はCSSセレクタに一致する最初の要素を取得するメソッドだ。このメソッドでボタンの要素を取得している。

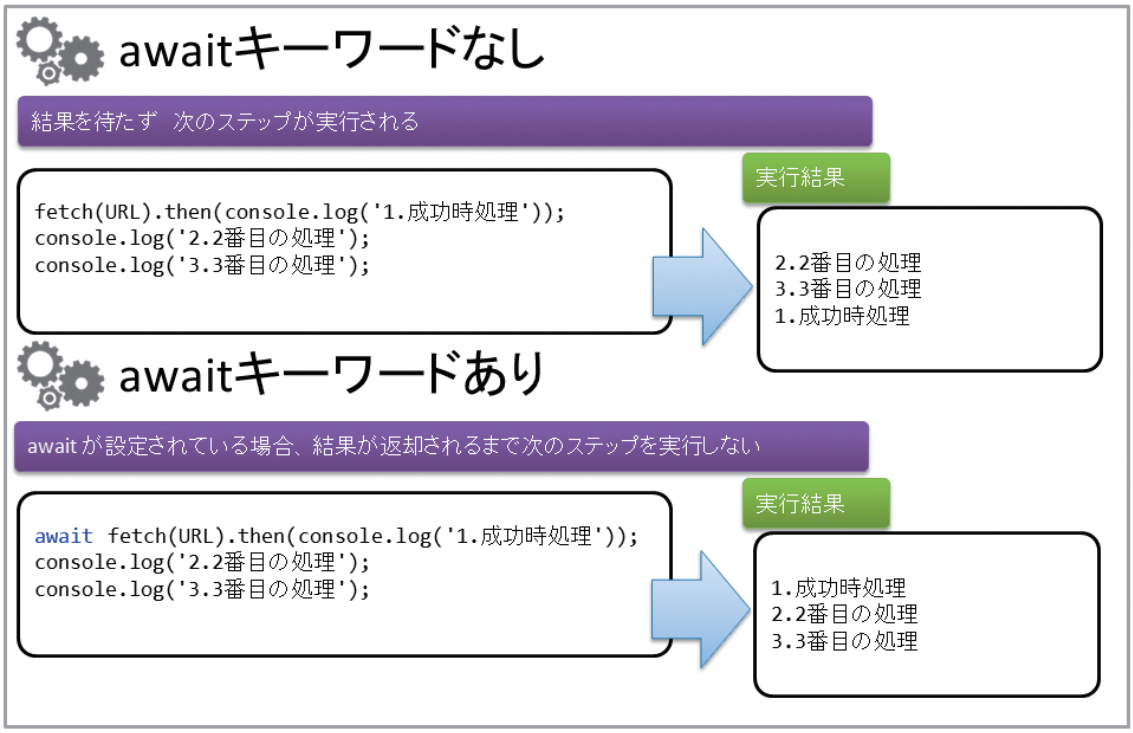
次に、取得したボタンの'addEventListener'メソッドを使用してイベントリスナー(イベントハンドラ)を追加する。ボタンがクリックされた際に非同期関数'createExcel'を実行するように設定する。

■【ソース3】②

'createExcel'は非同期関数として定義している。非同期関数の定義方法は、functionの前に'async'を記述するだけだ。非同期関数内で処理の終了を待機する'await'キーワードを利用するために使用している。'await'キーワードを利用することで、処理の終了を待機(同期的に処理)することが可能となり、非同期処理をより効果的に扱うことができる。

例えば、fetch APIは通信を実行する処理で、HTTPリクエストを行うXMLHttpRequestをより柔軟に使いやすくした機能である。fetchはPromiseオブジェクトを返却する非同期処理のため、'await'キーワードがない場合、fetchを実行後、処理結果が返却される前に次のステップへ進んでしまう。'await'キーワードがある場合、fetchの処理が完了するまで待機するため、fetchの処理結果を利用して後続処理を記述することが可能になる。【図9】

図 9　awaitキーワードでの同期処理



ExcelJSパッケージのExcelJSオブジェクトの'Workbook'メソッドを使用して、ワークブックを作成後、ワークブックオブジェクトの'addWorksheet'メソッドでワークブックに新しいシートを追加する。追加したシートオブジェクトはワークブックオブジェクトの'getWorksheet'メソッドを使用して取得する。

■【ソース3】③

追加したシートオブジェクトの'getCell'メソッドを使用して、シートの特定のセルにデータを設定する。'getCell'メソッドの第1引数は行番号、第2引数は列番号である。ワークブックオブジェクトに含まれるxlsxオブジェクトの'writeBuffer'メソッドを実行することで、オブジェクトに追加した内容をUnit8Array形式のデータとして取得している。

なお、'writeBuffer'メソッドは非同期処理されるため、'await'キーワードを設定して、同期的に処理している。

Unit8Array形式のデータは、JavaScriptのTypedArrayの一種で、8ビット符号なし整数（バイト）の配列を表現するために使用されるデータ型で、この型は主にバイナリデータやバイトストリームを操作するために使用される。

■【ソース3】④

次に、Unit8Array形式のデータをBlobに変換する。BlobはJavaScriptでバイナリデータや大きなデータの断片を表現するためのオブジェクトであり、Blobに変換することで、後続のダウンロード処理が可能となる。

ファイルのダウンロードを開始するために、URLオブジェクトの'createObjectURL'メソッドを使用して、URLを含む文字列を生成する。

この生成したURL文字列を動的に作成したアンカータグのhrefプロパティに設定し、クリック処理を実行すると、ExcelJSで生成したExcelファイルがブラウザを介してダウンロードされる。

この手順で、Excelファイルの生成とダウンロードが行える。

3.テンプレートを読込んで売上伝票を作成する方法

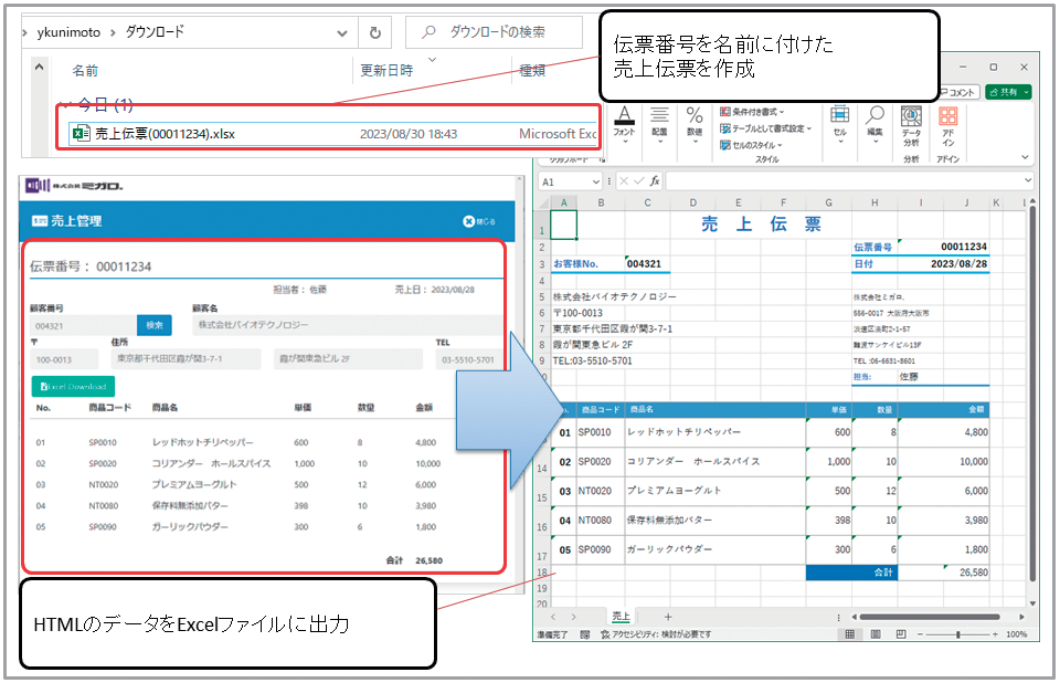
次は応用編として、Excelファイルのテンプレートを事前にサーバー上に配置しておき、ブラウザからテンプレートを読込み後、Excelファイルに売上情報を書込み、ダウンロードする方法を別のサンプルアプリSMP020で説明する。

サンプルアプリSMP020の画面は**【図10】**だ。

図 10　サンプルアプリ SMP020 画面①

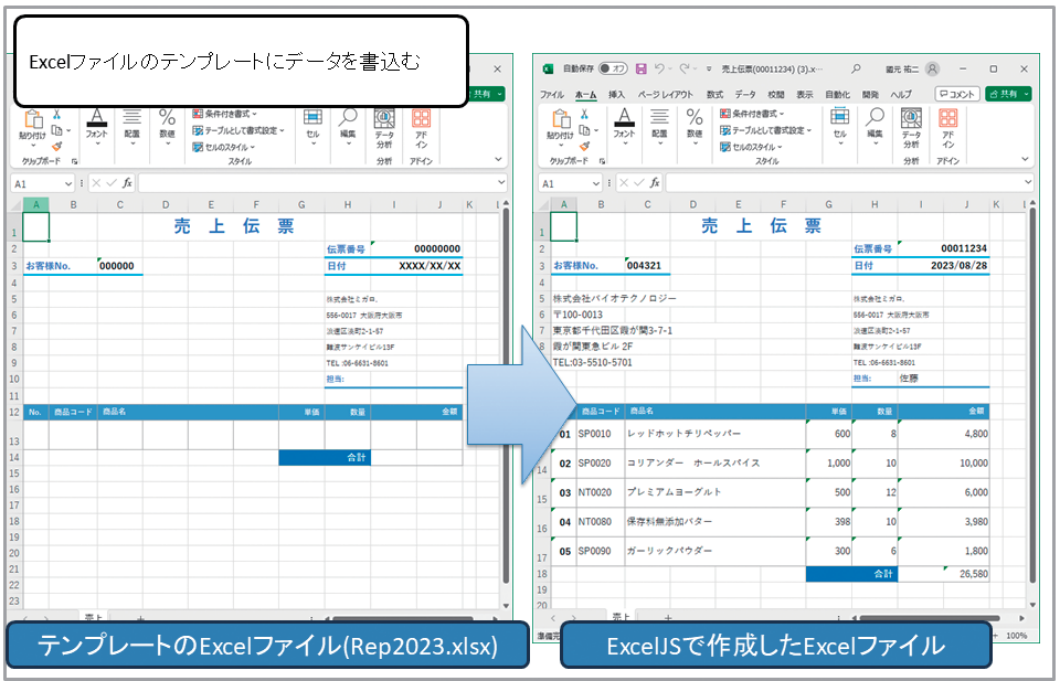
【図10】画面で「Excel　Download」のボタンをクリックすると、HTML上に出力されている情報を元に書込まれた売上伝票のExcelファイルが出力される。【図11】

図 11　サンプルアプリ SMP020 画面②



作成された売上伝票のExcelファイルはサーバー上に事前に作成したテンプレートのExcelファイル（Rep2023.xlsx）を元に作成する。【図12】

図 12　テンプレートのExcelファイルを元に作成



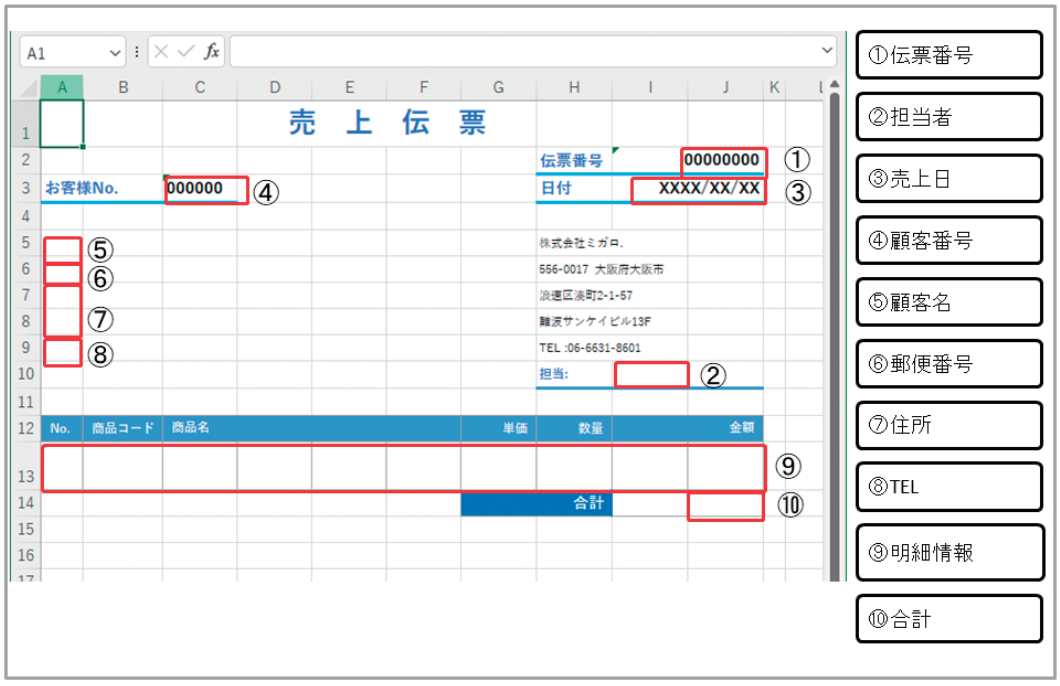
3-1. テンプレートの読込み方法

では、JavaScriptで読込むテンプレート（Rep2023.xlsx）を
Excelで作成する。HTMLから取得する情報は【図13】だ。

図 13　HTMLから取得する情報

【図13】の①～⑩の項目の情報をHTMLから取得する。取得した情報は【図14】テンプレートのExcelファイル（Rep2023.xlsx）の①～⑩の箇所に出力する。

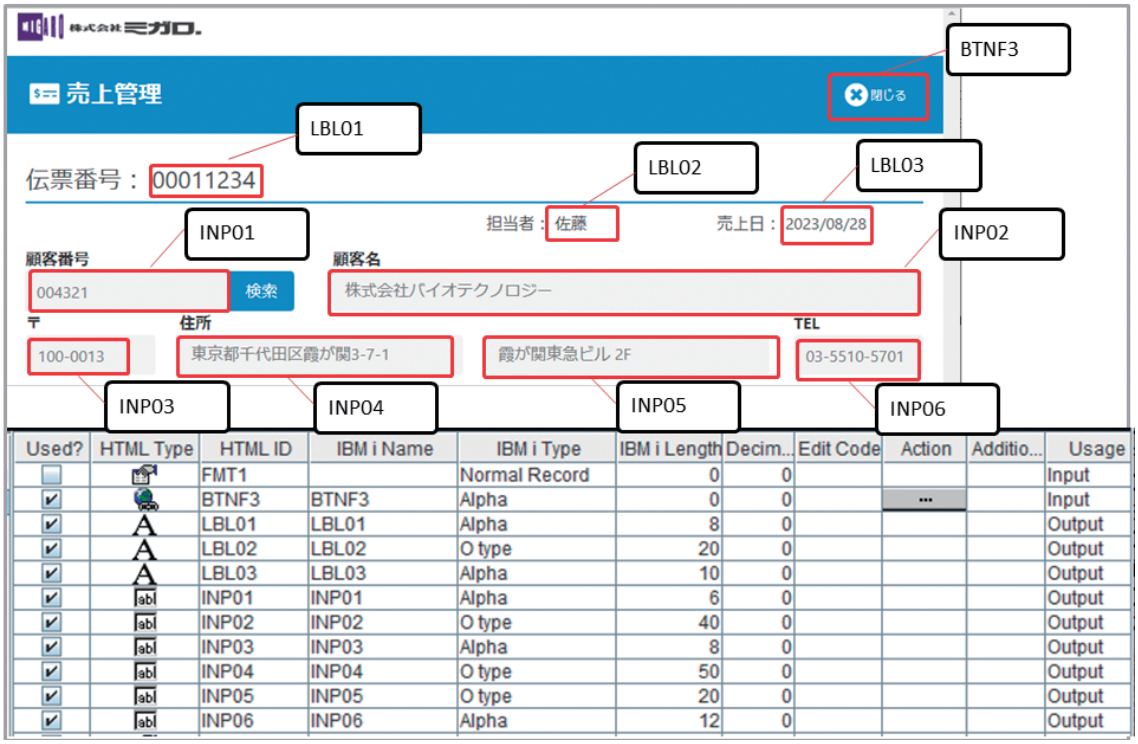
図 14　テンプレートのExcelファイル（Rep2023.xlsx）



【図14】⑨明細情報の箇所は、JavaScriptで行をコピーして挿入するため、書式設定などは1行のみ定義する。
作成したテンプレートのExcelファイル（Rep2023.xlsx）はJavaScriptで動的に読込むため、本稿ではHTMLファイルと同階層に配置する。

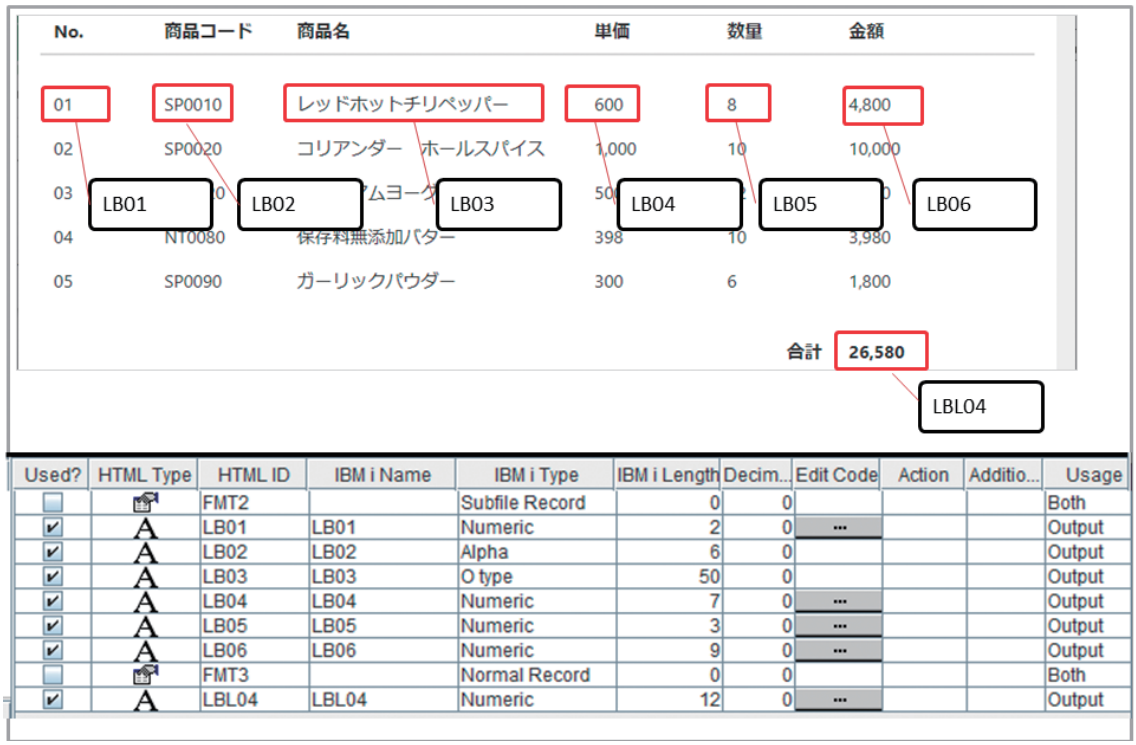
では、サンプルアプリSMP020のHTMLについて説明する。
サンプルアプリSMP020のDesignerの設定は**【図15】**
【図16】だ。

図 15　サンプルアプリSMP020のDesigner定義①



Used?	HTML Type	HTML ID	IBM i Name	IBM i Type	IBM i Length	Decim...	Edit Code	Action	Additio...	Usage
☐		FMT1		Normal Record	0	0				Input
☑		BTNF3	BTNF3	Alpha	0	0		...		Input
☑		LBL01	LBL01	Alpha	8	0				Output
☑		LBL02	LBL02	O type	20	0				Output
☑		LBL03	LBL03	Alpha	10	0				Output
☑		INP01	INP01	Alpha	6	0				Output
☑		INP02	INP02	O type	40	0				Output
☑		INP03	INP03	Alpha	8	0				Output
☑		INP04	INP04	O type	50	0				Output
☑		INP05	INP05	O type	20	0				Output
☑		INP06	INP06	Alpha	12	0				Output

図 16　サンプルアプリSMP020のDesigner定義②



Used?	HTML Type	HTML ID	IBM i Name	IBM i Type	IBM i Length	Decim...	Edit Code	Action	Additio...	Usage
☐		FMT2		Subfile Record	0	0				Both
☑	A	LB01	LB01	Numeric	2	0	...			Output
☑	A	LB02	LB02	Alpha	6	0				Output
☑	A	LB03	LB03	O type	50	0				Output
☑	A	LB04	LB04	Numeric	7	0	...			Output
☑	A	LB05	LB05	Numeric	3	0	...			Output
☑	A	LB06	LB06	Numeric	9	0	...			Output
☐		FMT3		Normal Record	0	0				Both
☑	A	LBL04	LBL04	Numeric	12	0	...			Output

設定した、項目にIBMiプログラム（RPG）から値を出力している。
HTML側の記述としては、ExcelJSを使用するために**【ソース1】**の外部読込みと、ボタンクリック時にファイルをダウンロードするため**【ソース2】**と同じようにボタンを追加している。
また、明細の各項目にJavaScriptでアクセスできるようにするため、カスタムデータ属性を設定した。**【ソース4】**

ソース 4

明細の定義　（SMP020.HTML HTML カスタム属性定義）

```
<table class="table is-fullwidth is-hoverable" id="SFL01">
  <tbody>
    <tr>
      <td id="LB01" data-dwn="saleno" style="width:100px">01</td>
      <td id="LB02" data-dwn="salecd" style="width:120px" > </td>
      <td id="LB03" data-dwn="salename"  > </td>
      <td id="LB04" data-dwn="saleprice" style="width:120px"> </td>
      <td id="LB05" data-dwn="saleunit" style="width:110px"> </td>
      <td id="LB06" data-dwn="saleamount" style="width:180px"> </td>
    </tr>
  </tbody>
</table>
```

'data-dwn="フィールド項目"'のように定義しており、カスタムデータ属性を設定することで、JavaScriptの'document.querySelectorAll'メソッドから要素にアクセスすることが可能となる。

サンプルアプリSMP020では【図10】のようにダウンロード用ボタンを配置している。これは前項のサンプルアプリSMP010と同じ仕組みで、ボタンがクリックされた際に、Excelファイルを出力する非同期関数を呼出すように設定した。クリック時の処理は【ソース5】だ。

ソース 5

ボタンクリック時の処理① (SMP020.HTML内 JavaScript)

```
<script>
  /**
   * SmartPad4i(Cobos4i) 初期処理用のinitpage関数
   */
  function initpage(){
    // ダウンロード用ボタンの取得
    var excelButton = document.querySelector('[data-exceldownload]');

    /**
     * ダウンロードボタン クリックイベント
     */
    excelButton.addEventListener( "click" , function () {
      createExcel();
    } , false );

    /**
     * Excelファイルを作成する非同期関数
     */
    async function createExcel(){
      // HTMLから情報取得
      let d_No     = SP4i.getElementById('LBL01').textContent; // ①伝票番号
      let d_Tanto  = SP4i.getElementById('LBL02').textContent; // ②担当者名
      let d_Date   = SP4i.getElementById('LBL03').textContent; // ③売上日
      let d_CCd    = SP4i.getElementById('INP01').value;       // ④顧客番号
      let d_CName  = SP4i.getElementById('INP02').value;       // ⑤顧客名
      let d_CZip   = '〒' + SP4i.getElementById('INP03').value;// ⑥郵便番号
      let d_CAddr1 = SP4i.getElementById('INP04').value;       // ⑦住所1
      let d_CAddr2 = SP4i.getElementById('INP05').value;       // ⑦住所2
      let d_CTel   = 'TEL:'+SP4i.getElementById('INP06').value;// ⑧TEL
      let d_All    = SP4i.getElementById('LBL04').textContent; // ⑩売上合計

      // テンプレートのExcelファイルパス
      const EXCEL_URL = '/smartpad4i/html/SP4IREP23/Rep2023.xlsx';
      // テンプレートのExcelファイルを読込み、バイト配列で取得
      const existingExcelBytes = await fetch(EXCEL_URL).then(res =>
         res.arrayBuffer());
      // バイト配列で取得したExcelファイルを8ビット符号なし整数値の配列に変換
      const exceldata = new Uint8Array(existingExcelBytes);
      // workbookを作成
      const workbook = new ExcelJS.Workbook();
      // テンプレートのExcelファイルのデータを読込み
      await workbook.xlsx.load(exceldata);
      // テンプレートで設定したシート取得
      const worksheet = workbook.getWorksheet('売上');
      // シートの用紙サイズ、縦横設定
      worksheet.pageSetup = { paperSize: 9, orientation:'portrait'};
      // ページ 余白設定 (単位inch)
      worksheet.pageSetup.margins = {
        left: 0.23, right: 0.23,
        top: 0.75 , bottom: 0.75 ,
        header: 0.31, footer: 0.31
      };

      // ソース6に続く
```

■【ソース5】①

SMP010サンプルアプリと同じ仕組みでボタンクリック時にExcelファイルを作成する非同期処理関数を呼び出している。

■【ソース5】②

Excelファイルに書込む内容を取得している。SmartPad4iのメソッド'SP4i.getElementById'を使用して、各id属性の要素のテキストや値を取得する処理だ。

spanタグの場合は'textContent'プロパティで出力されているテキストを取得し、inputタグの場合は、'value'プロパティで入力値を取得する。

■【ソース5】③

テンプレートのExcelファイルを読込む処理になる。
テンプレートの読込みは、JavaScriptのfetch　APIを使用する。
HTTPリクエストの処理はサーバーへのリクエスト処理後、HTTPレスポンスを取得するまでに時間が掛かる場合もあるため、'await'キーワードを使用して呼出すことで、HTTPレスポンスが返却されるまでの処理を同期的に実行している。
fetchの第1引数には取得したいリソースを指定する。テンプレートのExcelファイルをHTTP経由で取得するため、URLを指定する。fetchの処理はPromiseオブジェクトを返却する。

Promiseオブジェクトは非同期処理を簡潔かつ効果的に扱うための仕組みである。fetchのチェーンメソッドとして'then'を指定すると、処理が成功した場合（Promiseオブジェクトに処理成功のステータスが設定された場合）に'then'メソッド内の処理が実行される。
thenメソッド内ではレスポンスの結果を'arrayBuffer'メソッドでバイト配列として'existingExcelBytes'変数に取得している。
取得した、'existingExcelBytes'変数のデータをUnit8Arrayで変換後、ワークブックオブジェクトに含まれるxlsxオブジェクトの'load'メソッドを使用して、テンプレートのExcelファイルを読込む。

■【ソース5】④

テンプレートのExcelファイルに定義されている売上シートをワークブックオブジェクトの'getWorksheet'メソッドを使用して取得している。
シートを取得後、シートオブジェクトの'pageSetup'メソッドで、用紙サイズ（'paperSize'）と縦横（'orientation'）を指定後、さらに'worksheet.pageSetup.margins'オブジェクトを指定することで、ページの余白設定を変更する。Excelの余白表示はcm単位であるのに対して、'margins'に設定する単位はインチ（inch）単位であるため注意が必要だ。

3-2. 帳票出力方法

取得したデータをExcelファイルに書込む処理が**【ソース6】**だ。

ソース 6

ボタンクリック時の処理② (SMP020.HTML内 JavaScript)

```
<script>
      ～省略～
  /**
   * Excelファイルを作成する非同期関数
   */
  async function createExcel(){
        ～省略～
    worksheet.getCell(2 , 9).value = d_No;          // ①伝票番号
    worksheet.getCell(10, 9).value = d_Tanto;       // ②担当者名
    worksheet.getCell(3 , 9).value = d_Date;        // ③売上日
    worksheet.getCell(3 , 3).value = d_CCd;         // ④顧客番号
    worksheet.getCell(5 , 1).value = d_CName;       // ⑤顧客名
    worksheet.getCell(6 , 1).value = d_CZip;        // ⑥郵便番号
    worksheet.getCell(7 , 1).value = d_CAddr1;      // ⑦住所1
    worksheet.getCell(8 , 1).value = d_CAddr2;      // ⑦住所2
    worksheet.getCell(9 , 1).value = d_CTel;        // ⑧TEL
    worksheet.getCell(14 , 10).value = d_All;       // ⑩売上合計

    // ⑨明細情報 **********************

    let no_arr=document.querySelectorAll('[data-dwn="saleno"]');          //明細No.
    let cd_arr=document.querySelectorAll('[data-dwn="salecd"]');          //商品コード
    let name_arr=document.querySelectorAll('[data-dwn="salename"]');      //商品名
    let price_arr=document.querySelectorAll('[data-dwn="saleprice"]');    //価格
    let unit_arr=document.querySelectorAll('[data-dwn="saleunit"]');      //数量
    let amount_arr=document.querySelectorAll('[data-dwn="saleamount"]');//金額

    // データを書込む起点設定
    const startRow = 13; // 明細開始行番号
    const startCol =  1; // 明細開始列番号
    let row         = {};
    let rownum      =  0;
    // Excelの開始行をデータの数コピー
    worksheet.duplicateRow(startRow , no_arr.length - 1 , true);

    // シートに書き込み（明細行数でループ処理）
    for (var i=0;i < no_arr.length; i++) {
      let pos = startRow + i;
      row = worksheet.getRow(pos);
      // セルのマージ 商品名 3列目～6列目
      worksheet.mergeCells(pos , 3 , pos , 6);
      // セルのマージ 金額   9列目～10列目
      worksheet.mergeCells(pos , 9 , pos , 10);
      row.getCell(startCol).value     = no_arr[i].textContent;    //明細No.
      row.getCell(startCol + 1).value = cd_arr[i].textContent;    //商品コード
      row.getCell(startCol + 2).value = name_arr[i].textContent;  //商品名
      row.getCell(startCol + 6).value = price_arr[i].textContent; //価格
      row.getCell(startCol + 7).value = unit_arr[i].textContent;  //数量
      row.getCell(startCol + 8).value = amount_arr[i].textContent;//金額
    }

    // 書込みした内容をwriteBufferメソッドで出力
    const uint8Array = await workbook.xlsx.writeBuffer();
    // Blobオブジェクトの作成
    const blob = new Blob([uint8Array],
      {type: 'application/octet-binary'});
    // ダウンロードするためにblobのURL作成
    const url = window.URL.createObjectURL(blob);
    // アンカーを作成してURL設定
    const a = document.createElement('a');
    a.href = url;
    // ダウンロードするファイル名に伝票番号を設定してダウンロード
    a.download = '売上伝票(' + d_No  + ').xlsx';
    a.click();
    a.remove();
  }
}
</script>
```

■【ソース6】①
ワークシートオブジェクトの'getCell'メソッドを使用して、対象のセルに書込み処理を実行している。

■【ソース6】②
明細部分のデータを'document.querySelectorAll'メソッドで取得している。'document.querySelectorAll'はCSSセレクタを指定することで、対象のノードリストを取得することができる。

■【ソース6】③
明細データを書込む開始行や開始列を定義している。最後の処理では、ワークシートオブジェクトの'duplicateRow'メソッドを使用して、Excelの行をコピーしている。'duplicateRow'メソッドの第1引数は、コピーを開始する対象行番号、第2引数は、コピーする回数、第3引数は行のコピー時に新しい行を挿入するか、置換するかのフラグを指定する。サンプルでは、コピーした行を挿入したいため、第3引数に'true'を設定する。

■【ソース6】④
明細行を書込んでいる。ワークシートオブジェクトの'getRow'メソッドを使用して、指定した行番号の行オブジェクトを取得する。取得した行オブジェクトはrow変数に格納される。取得したrowオブジェクトのセルは全て1列ずつの状態のため、ワークシートオブジェクトの'mergeCells'メソッドでセルをマージ(結合)する。
'mergeCells'メソッドの第1引数は結合する開始セルの行番号、第2引数は、列番号、第3引数には結合する終了セルの行番号、第4引数には列番号を指定する。
この方法で、商品名の3列目～6列目と金額の9列目～10列目のセルが結合される。

データの書込みは、rowオブジェクトの'getCell'メソッドに列番号を指定することで対象のセルを取得することができるため、valueプロパティに各ノードリスト項目の'textContent'(テキスト)を設定することで、データを書込む。

■【ソース6】⑤
サンプルアプリSMP010と同様に、書込んだワークブックオブジェクトを配列データに変換後、ダウンロードする処理を記述している。また、サンプルアプリSMP020では、ダウンロードするファイル名に伝票番号を設定する。
このようにして、Excelファイルのテンプレートを活用し、データを書込み、ダウンロードする処理が実行できる。

4.おわりに

JavaScriptは進化し続けており、ブラウザで実現できる範囲が増えてきている。
本稿では、Excelファイルを出力するOSSを紹介したが、インターネット上には、クライアントサイドのJavaScriptのみでMicrosoft Wordファイルや、PDFファイルを出力できるOSSのライブラリも存在する。
クライアントサイドのJavaScriptのみで実装する場合、既存のSmartPad4i(Cobos4i)アプリにも簡単に組込むことができるため、OSSを活用して、システム開発に役立てて頂ければ幸いである。