SmartPad4i

SmartPad4iで電子サインを実現する方法

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、JC/400)、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

最近、携帯電話の契約手続きを行う機会があった。
店員からタブレットを見せられながら、料金プランや契約関連の説明を受けた後、タブレットに指でサイン(名前の記入)を求められた。過去、生命保険の手続き時にも同様にタブレットを使用してサインを行ったケースもある。
今まで、紙媒体で実施されてきた手続きが、スマートデバイスに置き換わっていることや、スマートデバイスでの電子サインが、意思確認の手段として普及してきていることを日々の生活の中で実感している。

電子サインを実装できれば、対面の手続き、報告業務などで業務効率化やコスト削減、ペーパーレス化なども実現可能である。
もちろん、SmartPad4iアプリケーションでも電子サインの機能を実装することは可能だ。本稿では、SmartPad4iを使用して、電子サインの機能を実装する方法について説明する。

2.canvas要素の基本

2-1.HTML5のcanvas要素

電子サインのインターフェースにはcanvas要素を使用する。canvas要素はHTML5で追加された要素だ。HTMLでは、canvasタグで描画領域を定義して、canvas要素への描画はJavaScriptで記述する必要がある。タグの例は**【ソース1】**だ。

ソース 1

HTML　canvasタグの例

```
<canvas id="CANVAS" width="600px" height="300px">
  <p>お使いのブラウザではCanvasが使用できません</p>
</canvas>
```

canvas要素の描画領域はwidth属性とheight属性で指定する。未指定の場合width属性は300px,height属性は150pxが設定される。CSSのwidthとheight設定では描画領域を指定できず、見た目上のサイズが変更されてしまうため、引き伸ばされた画像となるため注意が必要だ。

また、canvas要素に未対応のブラウザでは、描画した内容を表示することができない。canvasタグの中にhtmlを記述することで、未対応のブラウザで表示するための代替コンテンツを設定できる。

2-2. canvas要素への描画

2-2-1. 図形の描画

まずは、canvas要素で定義した領域に描画をする簡単な例を紹介する。canvas要素への描画は「Canvas API」をJavaScriptから使用する。
四角形を描画する例は**【ソース2】**だ。

ソース2

JavaScript canvasへの描画例(四角形)

```
<script>
 window.onload = function(){
  //canvas要素の取得
  var canvas = document.getElementById('CANVAS'); ----------①
  //描画コンテキストの取得
  var context = canvas.getContext('2d'); ----------②
  //左から250 , 上から75 の位置へ 横幅100 , 縦幅150の四角形を描画
  context.fillRect(250,75,100,150); ----------③
 }
</script>
```

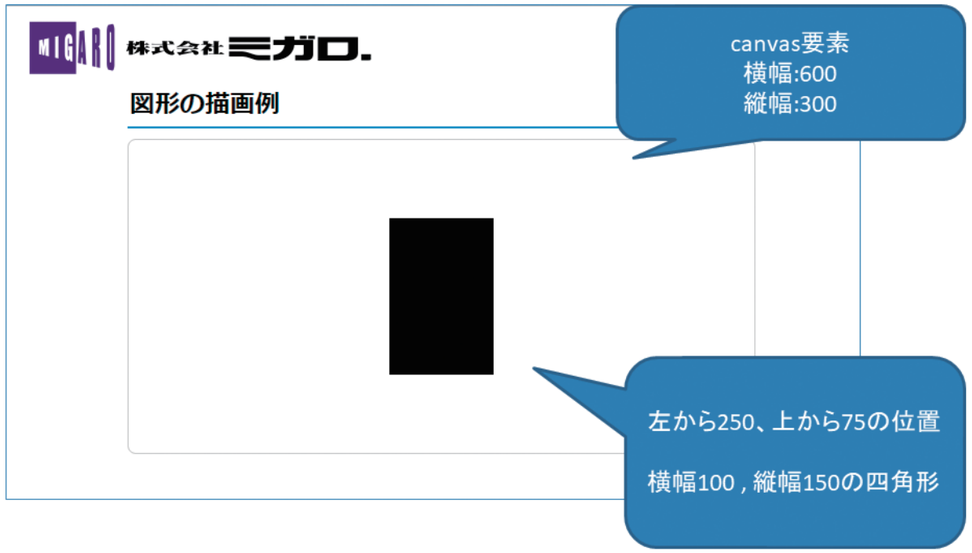
【ソース2】 1行目のwindow.onload = function(){}はHTMLを読み込んだ際に処理を実施するための記述だ。HTMLを読込後、【ソース2】①のdocument.getElementByIdでHTML上のcanvas要素を取得する。

次に、【ソース2】②canvas要素のgetContextメソッドで描画コンテキストを取得する。描画コンテキストはグラフィック描画するためのメソッドや、プロパティを持つオブジェクトだ。

getContextメソッドの引数には、contextType（文字列）を設定する。"2d"を指定すると、2次元のレンダリングを行う描画コンテキストが作成される。"webgl"または"webgl2"では3次元のレンダリングを行う描画コンテキストが作成され、"bitmaprenderer"はcanvas要素の内容を特定のフォーマットに置き換える機能を持つオブジェクトが作成される。電子サインや四角形の描画では2次元レンダリングを使用するため、getContextの引数に"2d"を指定している。context変数には、「CanvasRenderingContext2D」という描画コンテキストが格納される。

四角形を描画する場合には、【ソース2】③のように描画コンテキストのfillRectメソッドを使用する。引数は4つで、1つ目、2つ目が描画開始点の横座標と縦座標、3つ目が横幅、4つ目が縦幅だ。表示される四角形は【図1】になる。

図1　ソース2で描画される内容



2-2-2. 線の描画

次は線を描画する方法について説明する。線の描画は【ソース3】だ。

ソース 3

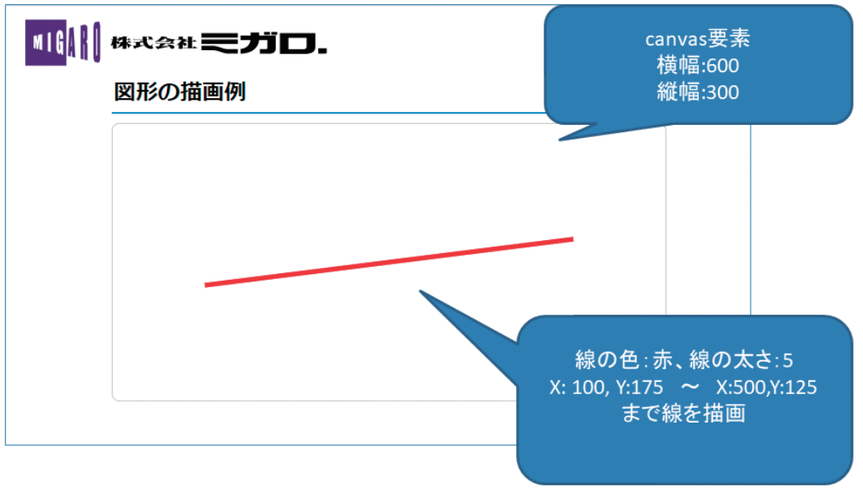
JavaScript canvasへの描画例（線）

```
<script>
 window.onload = function(){
  //canvas要素の取得
  var canvas = document.getElementById('CANVAS');
  //描画コンテキストの取得
  var context = canvas.getContext('2d');
  //線の色
  context.strokeStyle = 'red'; ----------①
  //線の太さ
  context.lineWidth = 5; ----------②
  //線の始点
  context.moveTo( 100 , 175 ); ----------③
  //線の終点
  context.lineTo( 500 , 125 ); ----------④
  //描画
  context.stroke(); ----------⑤
 }
</script>
```

描画コンテキストを取得後、【ソース3】①strokeStyleで線の色を指定する。デフォルトは黒で、色の文字列（red, yellow等）、または、RGB値で指定できる。【ソース3】②lineWidthでは線の太さを指定する。

線の描画は、【ソース3】③～⑤になる。moveToメソッドで線の始点を設定し、次にlineToメソッドで線の終点を指定する。最後にstrokeメソッドを実行するとcanvas要素に、指定した線の色、太さで始点から終点までの線が描画される。【図2】

図2　ソース3で描画される内容



3.電子サインの実装方法

3-1. 電子サインの実装例

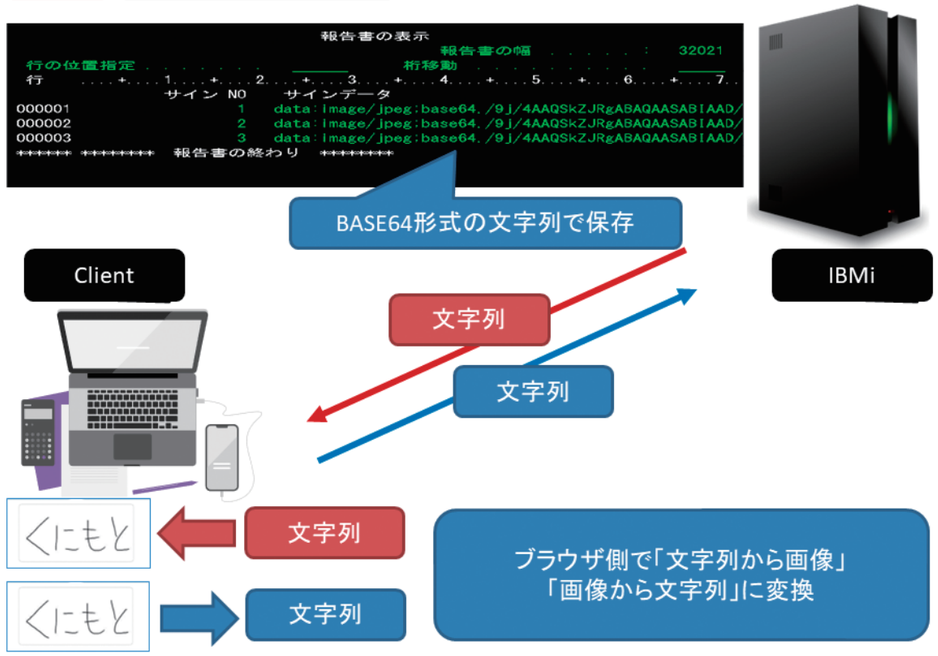
本節では、SmartPad4iで電子サイン機能を実装する方法について説明する。電子サインを実装したアプリケーションの例は【図3】、【図4】だ。

図3　電子サインの実装(1)

図4　電子サインの実装(2)

例では、電子サインを記入するcanvas要素と操作用の「クリア」ボタン、「登録」ボタンを配置した。
画面が読み込まれると、canvas要素に描画するためのイベント処理が追加されて、canvas要素に指やタブレットペンでサインが記入できる状態になる。サイン記入後「登録」ボタンをクリックすると、記入した電子サインの情報が文字列として、IBMi側に送信され、IBMiのファイルに電子サインが文字列として登録される。【図5】

図5　電子サインの実装(3)



また、ファイルに登録した最後のサインを画面に表示できるように実装した。

3-2. フロントエンド実装

3-2-1. HTMLの作成

では、フロントエンドのHTMLとJavaScriptの実装方法から説明する。HTMLの例は**【ソース4】**、**【ソース5】**だ。

ソース 4

HTML 電子サイン 例(1)

```
<form method="post" >
  <div id="container">
    <div id="content">
      <header>
        <div>
          <div>
            <span><img src="img/migaro_logo.png"></span>
          </div>
        </div>
      </header>
      <main>
        【ソース5】
      </main>
    </div>
  </div>
</form>
```

【ソース4】では、画面上部ヘッダーに表示するアイコンの画像を定義している。mainタグの中は**【ソース5】**になる。

ソース 5

HTML 電子サイン 例(2)

```
<div class="box" >
  <section style="border-bottom:solid 2px #3e8ed0">
    <h1>署名</h1>
  </section>
  <canvas id="signCanvas" width="600px" height="300px">
    <p> お使いのブラウザでは電子サインができません</p>
  </canvas>
</div>
<div style="width:100%"></div>
<div>
  <div>
    <input type="button" class="button" style="margin-right: 10px;"
           data-clear="true"  value="クリア">
    <input type="button"  class="button"  value="登録"  data-save="true">
    <input type="button"  style="display:none"  value="登録"  id="BTN01">
    <input type="hidden"  id="SIGN">
  </div>
</div>
<img src="" data-img="true">
```

①canvas要素 横幅:600 縦幅:300

②操作用ボタン

③隠しボタン 隠し入力欄

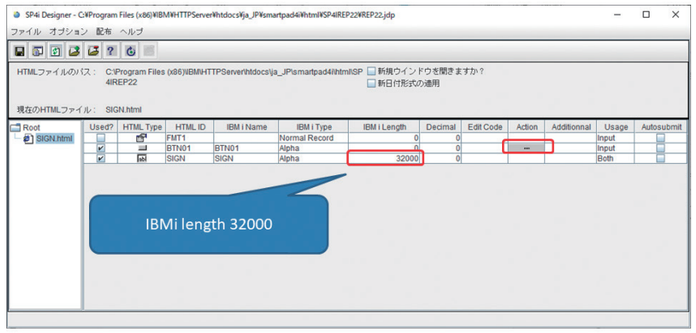
④登録済み電子サイン表示用画像

【ソース5】①でcanvas要素を横幅600縦幅300で定義した。**【ソース5】**②では、操作用ボタンを定義している。canvas要素の描画内容を消去するための「クリア」ボタン、canvas要素の描画内容を登録するための「登録」ボタンだ。クリックされた際に、canvas要素の電子サインを文字列に変換する処理を行い、隠し入力欄に電子サインの文字列を設定後、IBMi側へデータを送信するため、隠しボタンをクリックする。**【ソース5】**③はIBMiにデータを送信するための隠しボタンと隠し入力欄だ。隠し入力欄に文字列データを設定、隠しボタンをクリックすることでIBMi側へ送信している。**【ソース5】**④はIBMi側ファイルに登録された電子サインを表示するための画像を定義した。

3-2-2. Designerの設定

SmartPad4iの開発では、作成したHTMLをDesignerで読み込み、各フィールドのデータ型やデータ長を設定後、IBMiへ配布する。電子サインのデータは、画像データを文字列に変換して入出力する。そのため、入出力する入力欄のIBMi Lengthには、最大長の32,768byteに近い値を設定する。サンプルでは、32,000byteを設定した。**【図6】**

図 6 Designer設定



3-2-3. JavaScriptの作成

次に、canvas要素へ電子サインを描画するためのJavaScriptを説明する。SmartPad4iのinitpage関数実行時にcanvas要素へ電子サインを描画するための処理を記述している**【ソース6】**。

SmartPad4iのinitpage関数は画面が読み込まれて、SmartPad4iの初期処理が呼び出される前に実行される関数だ。

ソース 6

JavaScript 電子サインcanvasへの描画(1)

```
<script>
  function initpage(){
  ～ 省略 ～
    //描画コンテキストの取得
    var canvas = document.getElementById('signCanvas');
    var context = canvas.getContext("2d");
    //線形, 太さ, 色 ,キャンバスクリア
    context.lineCap  = "round";
    context.lineWidth = 3;
    context.strokeStyle = "black";
    clearCanvas();
    var isDragging = false;
    var lastPosition = { x: null, y: null };
    //PC
    canvas.addEventListener("mousedown", dragStart);
    canvas.addEventListener("mouseup",   dragEnd);
    canvas.addEventListener("mouseout",  dragEnd);
    canvas.addEventListener("mousemove", dragMovePC);
    //SmartDevice
    canvas.addEventListener("touchstart",  dragStart);
    canvas.addEventListener("touchcancel", dragEnd);
    canvas.addEventListener("touchend",    dragEnd);
    canvas.addEventListener("touchmove",   dragMoveSD);
  }
</script>
```

①コンテキスト取得

②描画設定と初期化

③制御用

④PC用イベント

⑤Smartデバイス用イベント

【ソース6】は電子サインを描画するための準備処理に当たる箇所だ。initpage関数の前半に「～省略～」と記載した箇所には、**【ソース6】**の中で実行される関数が定義されている。

関数は**【ソース7】**、**【ソース8】**、**【ソース9】**になる。

ソース 7

JavaScript 電子サインcanvasへの描画(2)

```
//canvasクリア
function clearCanvas(){
  context.clearRect(0,  0, canvas.width,  canvas.height);  //canvasクリア
  context.fillStyle  = "#FFFFFF"; //塗りを白色に設定
  context.fillRect(0,0,canvas.width,canvas.height);  //四角を描く
}

//描画の開始
function dragStart(event) {
  context.beginPath();
  isDragging = true;
}

//描画の終了
function dragEnd(event) {
  isDragging = false;
  //座標リセット
  lastPosition.x = null;
  lastPosition.y = null;
}
```

①canvas初期化
②描画の開始
③描画の終了

ソース 8

JavaScript 電子サインcanvasへの描画(3)

```
//描画呼び出し(PC)
function dragMovePC(event) {
  draw(
    event.layerX - canvas.getBoundingClientRect().left - window.scrollX ,
    event.layerY - canvas.getBoundingClientRect().top  - window.scrollY
  );
}

//描画呼び出し(SD)
function dragMoveSD(event) {
  event.preventDefault();
  let x = event.touches[0].clientX - canvas.getBoundingClientRect().left
    - document.body.scrollLeft;
  let y = event.touches[0].clientY - canvas.getBoundingClientRect().top
    - document.body.scrollTop;
    draw(x, y);
  }
```

①PC用描画処理
②Smartデバイス用描画処理

ソース 9

JavaScript 電子サインcanvasへの描画(4)

```
//描画処理
function draw(x, y) {
  if (!isDragging) {
    return;
  }
  if (lastPosition.x === null || lastPosition.y === null) {
    context.moveTo(x, y);
  } else {
    context.moveTo(lastPosition.x, lastPosition.y);
  }
  context.lineTo(x, y);
  context.stroke();
  // 座標保持
  lastPosition.x = x;
  lastPosition.y = y;
}
```

①ドラッグ処理
②開始点
③終了点
④座標保持

【ソース6】①で描画コンテキストを取得し、**【ソース6】**②で線の描画設定をしている。また、**【ソース7】**①のclearCanvas関数を呼び出し、canvas要素をクリアする。clearCanvas関数では、描画コンテキストのclearRectメソッドを呼び出し、canvas要素をクリアしている。clearRectメソッドは矩形の形状で描画内容を消去する処理だ。引数はクリアする開始点x,y座標とクリアする横幅、縦幅の4つを指定する。

また、消去後にfillStyleへ白を設定、fillRectでcanvas要素全体に背景用の白い四角形を描画することでcanvas要素をクリアしている。

【ソース6】③は制御用のオブジェクトで、isDraggingはドラッグ処理中のフラグ、lastPositionは描画時の最終x,y座標位置を保持している。

【ソース6】④と⑤はマウス操作や、タッチ操作時の処理になる。マウス操作のPC用と、タッチ操作のSmartデバイス用を定義した。

addEventListenerは要素にイベントを追加する際に使用するメソッドだ。**【ソース6】**④では、canvas要素上のマウスダウン(mousedown)イベントで**【ソース7】**②で定義しているdragStart関数を呼び出す。

dragStart関数は描画の開始処理になり、描画コンテキストのbeginPathメソッドを呼び出し、現在描画中のパスをクリア後、制御用変数のisDraggingフラグにtrueを設定する。

【ソース6】④では、マウスのボタンが離された際のマウスアップ(mouseup)イベント、マウスが要素上から外れた場合に発生するマウスアウト(mouseout)イベントで、**【ソース7】**③で定義しているdragEnd関数を呼び出す。

dragEnd関数では、isDraggingフラグをfalseに設定することで、ドラッグ操作が終了したことを設定している。また、lastPositionオブジェクトのx,y座標にnullを設定して初期化する。

【ソース6】④のマウスが移動している際に発生するマウスムーブ(mousemove)イベントでは、**【ソース8】**①のdragMovePC関数を呼び出している。

dragMovePC関数では、**【ソース9】**の実際に描画処理を実施している、draw関数を呼び出す際の座標を取得する。eventオブジェクトに格納されているlayerXからマウスのx座標、layerYからマウスのy座標を取得、マウス位置からcanvas要素のgetBoundingClientRectメソッドでcanvas要素のx,y座標を取得して、それぞれ減算することで、canvas要素上の描画位置を算出している。window.scrollX,scrollYは画面がスクロールしている場合の計算処理だ、Smartデバイスでは、document.body.scrollLeftと document.body.scrollTopになる。

【ソース6】⑤は**【ソース6】**④のSmartデバイス向けのタッチ操作イベントにそれぞれ関数を設定している。PCブラウザと座標位置の取得が異なるため、タッチ操作(touchMove)イベント時には、**【ソース8】**②のdragMoveSD関数を呼び出している。PCブラウザとの違いは、eventオブジェクトのpreventDefaultを呼び出して、イベントの伝搬を終了している点だ。この処理により、電子サイン記入時には、タブレットや、スマートフォンの画面が上下左右に動かず、canvas要素上でサインができる。

最後に**【ソース9】**のdraw関数を呼び出してcanvas要素に描画を行う。**【ソース9】**のdraw関数では、**【ソース9】**①でドラッグ操作中であるかを判断して、ドラッグ操作が終了している場合にはreturnで関数処理を終了している。

ドラッグ操作時には**【ソース9】**②で線を描画する開始点を指定する。

【ソース9】③では、線の描画の終了点を設定後、描画コンテキストのstrokeメソッドを呼び出して、canvas要素に描画を行う。最後に、**【ソース9】**④で最後の座標位置を保存している。

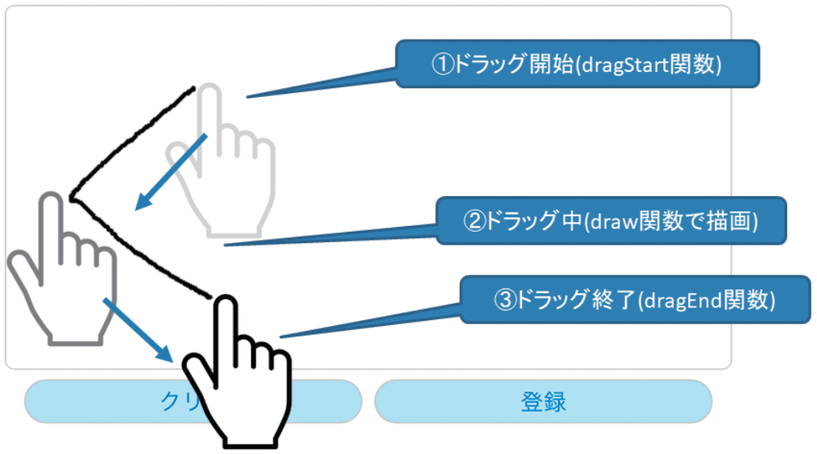
要するに、タッチ操作/マウスダウン操作時に描画開始の処理が実行されて、canvas要素上をタッチ操作/マウス操作で移動中に常にdraw関数で描画を繰り返している。

タッチ操作の終了(指やタッチペンをcanvas要素から離す)/マウス操作の終了(マウスボタンを離す)時に、描画処理が終了することになる。**【図7】**

図 7 電子サインの実装(4)

株式会社ミガロ.

署名



canvas要素へ描画した内容を、クリアボタンでクリアする処理と、登録ボタンでIBMi側プログラムへ送信する処理は【ソース10】だ。

ソース 10

JavaScript 電子サインcanvasへの描画(5)

```
//クリアボタン処理
var clearButton = document.querySelector('[data-clear]');
clearButton.onclick = function(){
  clearCanvas();
}

//登録ボタン処理
var saveButton = document.querySelector('[data-save]');
saveButton.onclick = function(){
  var RESIZE_RATIO = 0.5;
  var JPG_QUORITY = 0.5;
  var resizeCanvas = document.createElement('canvas');
  var rCtx = resizeCanvas.getContext('2d');
  //サイズを調整 1で同サイズ/ 0.5で半分
  rCtx.width = canvas.width * RESIZE_RATIO;
  rCtx.height = canvas.height * RESIZE_RATIO;
  rCtx.drawImage(canvas, 0, 0, canvas.width, canvas.height,
      0, 0, rCtx.width, rCtx.height);
  //レコードデータ更新
  var sing   = SP4i.getElementById('SIGN');
  sing.value = resizeCanvas.toDataURL('image/jpeg' ,JPG_QUORITY);
  var btn01  = SP4i.getElementById('BTN01');
  btn01.click();
}
```

①クリアボタン
②リサイズ用コンテキスト
③半分のサイズに変換
④データ変換と送信処理

【ソース10】①「クリア」ボタンの要素取得には、カスタムデータ属性を使用している。カスタムデータ属性は、data-*の形式で、*の箇所に独自で命名した名前を指定して属性を追加できる便利な機能だ。document.querySelectorメソッドでカスタムデータ属性にdata-clearが設定されている要素を取得している。

document.querySelectorメソッドはCSS セレクターを引数に設定することにより、条件に一致する最初の要素を取得することのできるメソッドだ。

取得したボタンは、clearButton変数に格納され、クリックイベント処理時に、【ソース7】①のclearCanvas関数を呼び出すことで、canvas要素をクリアしている。

【ソース10】「登録」ボタンの処理では、document.querySelectorメソッドでカスタムデータ属性にdata-saveが設定されている要素を取得している。要素は、saveButton変数に格納後、クリック時の処理を追加している。

「登録」ボタンのクリック時には、【ソース10】②の処理でリサイズ用の描画コンテキストを作成する。

リサイズ用の描画コンテキストの横幅、縦幅を半分に設定後、リサイズ用描画コンテキストのdrawImageメソッドで電子サインの描画内容をリサイズ用のコンテキストへコピーしている。

drawImageメソッドは、第1引数に描画するイメージを設定する。img要素、canvas要素、video要素を指定可能だ。第2引数から、第5引数には第1引数に設定された要素(コピー元)の使用範囲を指定する。第2引数は、開始x座標、第3引数は開始y座標、第4引数は横幅、第5引数は縦幅になる。第6引数から、第9引数は描画するイメージを配置する座標を指定する。

つまり、【ソース10】③の処理でリサイズ用の描画コンテキストに、半分のサイズに圧縮した内容をコピーしている。

【ソース10】④の処理では、IBMiへデータを送信するために、描画した電子サインの情報をtoDataURLメソッドで文字列に変換する。jpegデータ形式のデータをbase64エンコードした文字列に変換して、id属性SIGNが設定された入力欄に格納後、id属性"BTN01"の隠しボタン要素をクリックすることでIBMi側にデータを送信している。

最後に、登録した電子サインを表示するためのJavaScriptは【ソース11】だ。

ソース 11

JavaScript BASE64エンコードされた電子サインを画像として表示

```
//登録電子サイン表示
var imgsrc  = SP4i.getElementById('SIGN').value;
var img = document.querySelector('[data-img]');
if(imgsrc){
  img.src = imgsrc;
  img.style.display = 'block';
}
```

【ソース11】は、initpage関数内の最後に追加する。SP4i.getElementByIdメソッドを使用して、id属性"SIGN"の入力欄に設定された値をimgsrc変数に格納している。

次に、document.querySelectorメソッドでデータ属性"data-img"の要素をimg変数に格納する。imgsrc変数に値が設定されている場合、つまり、バックエンド側から電子サインの文字列データが送信されてきている場合には、img変数の画像ソースに電子サインの文字列を設定して、cssのdisplayプロパティに"block"を設定することで画面に表示している。以上が、フロントエンドの処理になる。

3-3. バックエンド実装

フロントエンドから送信された、電子サインの情報は文字列データとしてIBMiプログラムに送信される。
本稿のサンプルでは、JPEG画像データをBASE64エンコードした。
電子サインを保存するファイルのDDSは**【ソース12】**になる。

ソース 12

IBMi DDS 電子サインファイル

```
     A*****************************************************************
     A* ID         : F_SIGN
     A* NAME       : 電子サインファイル
     A* CREATE     : 2022/09/06 Y.KUNIMOTO
     A* UPDATE     : 9999/99/99
     A*****************************************************************
     A*
     A                                      UNIQUE
     A          R F_SIGNR                   TEXT('電子サイン')
     A            SIGNSNO       14S 0       COLHDG('サインNO')
     A            SIGNDTA    32000A         COLHDG('サインデータ')
     A          K SIGNSNO
```

電子サインを保存するフィールドは32,000桁で定義している。
バックエンド側では、送信されてきた電子サインの文字列データをファイルに入出力するだけだ。RPGプログラムは**【ソース13】**になる。

ソース 13

IBMi RPGプログラム(F仕様書)

```
  *---------------------------------------------------------------
  * DATABASE FILE DEFINITION
  *---------------------------------------------------------------
  * <YOURCODE>
  * YOUR FILES
 FF_SIGN    IF A E           K DISK      USROPN
  * </YOURCODE>
```

IBMi RPGプログラム(書込み呼び出し)

```
  * <YOURCODE>
  * CHECK ACTION CODE SPACTN HERE AND PROCESS.
 C     SPACTN        IFEQ      'A1'
 C                   EXSR      SB0020
 C                   ENDIF
 C     SPACTN        IFEQ      'F3'
 C                   GOTO      ENDPGM
 C                   ENDIF
  * YOUR CODE
  * </YOURCODE>
```

IBMi RPGプログラム(読込み呼び出し)

```
 C     YRDATA        BEGSR
  *
 C                   EXSR      SB0010
```

ソース 13

IBMi RPGプログラム(読込み)

```
  *---------------------------------------------------------------
  * SB0010:電子サインの読込み
  *---------------------------------------------------------------
 C     SB0010        BEGSR
 C                   OPEN      F_SIGN
 C     *HIVAL        SETGT     F_SIGNR
 C                   READP     F_SIGNR                                70
 C     *IN70         IFEQ      *OFF
 C                   MOVEL(P)  SIGNDTA       OSIGN
 C                   ENDIF
 C                   CLOSE     F_SIGN
 C                   ENDSR
```

IBMi RPGプログラム(書込み)

```
  *---------------------------------------------------------------
  * SB0020:電子サインの書込み
  *---------------------------------------------------------------
 C     SB0020        BEGSR
 C                   OPEN      F_SIGN
 C     *HIVAL        SETGT     F_SIGNR
 C                   READP     F_SIGNR                                70
 C     *IN70         IFEQ      *OFF
 C* サイン番号取得
 C     SIGNSNO       ADD       1             SIGNSNO
 C                   ELSE
 C                   Z-ADD     1             SIGNSNO
 C                   ENDIF
 C* 電子サイン書込み
 C                   MOVEL(P)  ISIGN         SIGNDTA
 C                   WRITE     F_SIGNR
 C                   CLOSE     F_SIGN
 C                   ENDSR
```

【ソース13】では、F仕様書ではファイルを読書き可能として定義している。【ソース13】(書込み呼び出し)ではフロントエンド側からアクションコード"A1"が送信された場合、つまり、登録ボタンが押下された場合に、サブルーチンSB0020を実行してファイルに書込んでいる。
また、読込みは、アプリケーション起動時にSmartPad4iのYRDATAでサブルーチンSB0010を呼び出し読込んでいる。 RPGプログラムは本稿用サンプルのため、バックエンド側のプログラムは使用する業務にあわせて作成してほしい。

4.実装時の注意点

電子サイン情報は文字列データとしてIBMi側で扱う、文字列データは大文字、小文字を区別するため注意が必要だ。また、電子サインのサイズやデータ形式によっては大容量の文字列データとなる場合がある。
SmartPad4iでは、一つのフィールドでデータを送受信することのできる最大長は32,768byteのため、大容量のデータを入出力する場合には、フォーマットを分けて、複数のフィールドを使用して送受信する必要がある。
また、IBMi側ファイルで保存できる文字列の最大長も32,768byteのため、特定のbyte長でデータを区切り、複数のレコードに情報を保存する処理などが必要となるため注意してほしい。

5.おわりに

本稿では、SmartPad4iで電子サインの機能を実装する方法についてまとめた。実装の中でSmartPad4i/Cobos4iでは、HTMLを使用して、自由にインターフェースを作成できることを再認識できた。
電子サインの機能をSmartPad4iアプリケーションで実装することで、業務効率化やコスト削減、ペーパーレス化を進めることの手助けになれば幸いだ。