プログラミング演習(JavaScript)④【実践2】 WebAPI

WebAPIとはインターネット経由で使えるAPI(＝機能やデータの提供窓口)のことで、Webアプリ、モバイルアプリなどから呼び出せる共通のサービス窓口のことである。

WebAPI を使うと、①サーバー側にリクエストをしなくても、直接データを取得して表示ができたり(例:天気予報、ニュースなどをその場で表示)、②fetch()や XMLHttpRequest を使って、リロードせずに非同期で API からデータを取ってこれたり(例:XのTLが自動で更新される)、③豊富な既存のデータベースが利用できたり(例:地図表示のGoogleMapsAPI、天気情報のOpenWeatherAPI、ユーザーやリポジトリ情報のGitHubなど)・・・、と、これらを利用できるようにすることは大きなメリットとなる。

16 気象庁の天気予報データ(JSON 形式)を取得して表示するプログラム

ここでは、気象庁の天気データを使って、天気予報を表示させるプログラムを作る。

js16-1.htm　静岡県の今日と明日の天気情報を表示する

```
001 <!DOCTYPE HTML>
002 <html>
003 <head>
004     <meta charset = ”utf-8”>
005 <script>
006 let url = ”https://www.jma.go.jp/bosai/forecast/data/forecast/220000.json”;
007 fetch(url)
008     .then(function(response) {
009         return response.json();
010     })
011     .then(function(weather) {
012         console.log(weather);
013         let area = weather[0].timeSeries[0].areas[0];
014         console.log(area);
015         document.getElementById(”publishingOffice”).lastElementChild.textContent =
    weather[0].publishingOffice;
016         document.getElementById(”reportDatetime”).lastElementChild.textContent =
    weather[0].reportDatetime;
017         document.getElementById(”targetArea”).lastElementChild.textContent =
    area.area.name;
018         document.getElementById(”today”).lastElementChild.textContent =
    area.weathers[0];
019         document.getElementById(”tomorrow”).lastElementChild.textContent =
020 area.weathers[1];
```

```
021     });
022 </script>
023 </head>
024 <body>
025     <h1>気象庁 JSON データを利用して<br>
026         静岡の天気を表示させる</h1>
027     <table>
028         <tr id="publishingOffice">
029             <th>発表者</th><td></td>
030         </tr>
031         <tr id="reportDatetime">
032             <th>報告日時</th><td></td>
033         </tr>
034         <tr id="targetArea">
035             <th>対象地域</th><td></td>
036         </tr>
037         <tr id="today">
038             <th>今日の天気</th><td></td>
039         </tr>
040         <tr id="tomorrow">
041             <th>明日の天気</th><td></td>
042         </tr>
043     </table>
044 </body>
045 </html>
```

【解説】

６行　変数 url に静岡県の天気予報データを提供する気象庁の API の URL を入れる

７行　fetch(url)は、指定された URL からデータを非同期で取得する

９行　response.json()で、取得したデータを JSON 形式として処理する

１１行　weather 変数に取得した JSON データが格納される。これで、この後の処理が可能になる。

１２行　データ取得成功後、デバッグ用にコンソールに wether の中身を表示させる

１３行　area に静岡エリアのデータを格納する。wether[0]が複数の予報の中から最初のデータ全体（「今日・明日・明後日までの短期予報（1～3 日分)」）を指し、timeSeries は時間ごとの天気予報（ [0]が天気（今日・明日・明後日など)、[1]が風・波、[2]が気温（最高・最低))、areas は地域ごとの天気情報で、ここでは、両方とも０なので、今日の中部地区の天気情報を取得している。

１５行　取得したデータの publishingOffice の値を HTML 上の id が publishingOffice の最後の子要素（<td></td>の中）に入れる。

１６、１７行　１５行と同様に指定した ID の最後の子要素にデータを入れる
１８行　area.wethers[0]である今日の天気を today に入れて表示する
１９行　area.wethers[1]である明日の天気を tomorrow に入れて表示する
２７～４３行　表示する項目とデータの表

表１６　データの内容と ID

項目名	ID
発表者	publishingOffice
報告日時	reportDatetime
対象地域	targetArea
今日の天気	today
明日の天気	tomorrow

【実行結果】

js16-2.htm　東京都と神奈川県と静岡県の今日と明日と明後日の天気情報を表示する

```
005    <script>
006        const areaCodes = {
007            "東京": "130000",
008            "神奈川": "140000",
009            "静岡": "220000"
010        };
011        function fetchWeather(areaCode) {
012            let url = `https://www.jma.go.jp/bosai/forecast/data/forecast/${areaCode}.json`;
013            fetch(url)
```

```
014                     .then(response => response.json())
015                     .then(weather => {
016                         let area = weather[0].timeSeries[0].areas[0];
017                         document.getElementById("publishingOffice").lastElementChild.textContent
    = weather[0].publishingOffice;
018                         document.getElementById("reportDatetime").lastElementChild.textContent    =
    weather[0].reportDatetime;
019                         document.getElementById("targetArea").lastElementChild.textContent          =
    area.area.name;
020                         document.getElementById("today").lastElementChild.textContent               =
    area.weathers[0];
021                         document.getElementById("tomorrow").lastElementChild.textContent            =
    area.weathers[1];
022                         document.getElementById("dayAfterTomorrow").lastElementChild.textContent
    = area.weathers[2];
023                     })
024                     .catch(error => {
025                         alert("天気情報の取得に失敗しました: " + error);
026                     });
027             }
028             window.addEventListener("DOMContentLoaded", () => {
029                 fetchWeather(areaCodes["東京"]);
030                 document.getElementById("areaSelect").addEventListener("change", function() {
031                     const selected = this.value;
032                     fetchWeather(areaCodes[selected]);
033                 });
034             });
035         </script>
036     </head>
037     <body>
038         <h1>気象庁の天気予報、3エリアから選択して表示</h1>
039         <label for="areaSelect">地域を選んでください:</label>
040         <select id="areaSelect">
041             <option>東京</option>
042             <option>神奈川</option>
043             <option>静岡</option>
044         </select>
045         <table>
```

```
            <tr id="publishingOffice">
                <th>発表者</th><td></td>
            </tr>
            <tr id="reportDatetime">
                <th>報告日時</th><td></td>
            </tr>
            <tr id="targetArea">
                <th>対象地域</th><td></td>
            </tr>
            <tr id="today">
                <th>今日の天気</th><td></td>
            </tr>
            <tr id="tomorrow">
                <th>明日の天気</th><td></td>
            </tr>
            <tr id="dayAfterTomorrow">
                <th>明後日の天気</th><td></td>
            </tr>
        </table>
    </body>
    </html>
```

【解説】

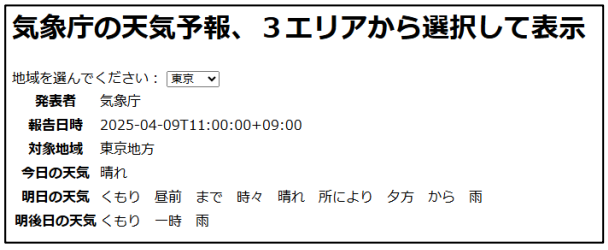
6～9行目　areaCodes オブジェクトを使って、地域名と気象庁のエリアコードを関連付ける

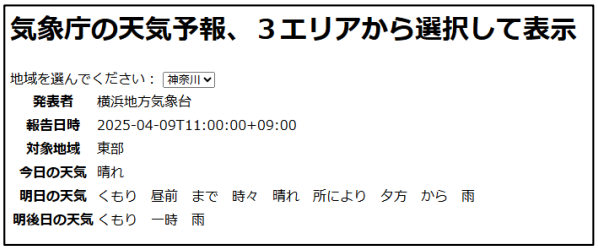
12行目　選択したエリアコードを url に入れて、指定した地域のデータを読み込む

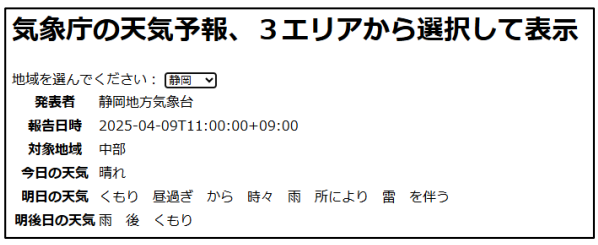
29行目　最初に表示させておく文字を「東京」にする

39～43行目　areaSelect で3つから選択できるようにする

【実行結果】







js16-3.htm　各エリアの対象地域を変更できるようにする

```
005     <script>
006         const areaCodes = {
007             "東京": "130000",
008             "神奈川": "140000",
009             "静岡": "220000"
010         };
011         let currentWeatherData = null;
012         function fetchWeather(prefCode) {
013             let url = `https://www.jma.go.jp/bosai/forecast/data/forecast/${prefCode}.json`;
014             fetch(url)
015                 .then(response => response.json())
016                 .then(weather => {
017                     currentWeatherData = weather;
018                     const areas = weather[0].timeSeries[0].areas;
019                     const regionSelect = document.getElementById("regionSelect");
020                     regionSelect.innerHTML = "";
021                     areas.forEach((area, index) => {
022                         const opt = document.createElement("option");
023                         opt.value = index;
024                         opt.textContent = area.area.name;
025                         regionSelect.appendChild(opt);
026                     });
027                     showWeather(0);
028                 })
029                 .catch(err => {
030                     alert("データ取得に失敗しました: " + err);
031                 });
032         }
033         function showWeather(areaIndex) {
034             if (!currentWeatherData) return;
035             const area = currentWeatherData[0].timeSeries[0].areas[areaIndex];
036             document.getElementById("publishingOffice").lastElementChild.textContent =
    currentWeatherData[0].publishingOffice;
037             document.getElementById("reportDatetime").lastElementChild.textContent =
    currentWeatherData[0].reportDatetime;
038             document.getElementById("targetArea").lastElementChild.textContent =
    area.area.name;
```

```
039                 document.getElementById("today").lastElementChild.textContent = area.weathers[0];
040                 document.getElementById("tomorrow").lastElementChild.textContent = area.weathers[1];
041                 document.getElementById("dayAfterTomorrow").lastElementChild.textContent = area.weathers[2];
042             }
043             window.addEventListener("DOMContentLoaded", () => {
044                 const prefSelect = document.getElementById("prefSelect");
045                 const regionSelect = document.getElementById("regionSelect");
046                 fetchWeather(areaCodes["東京"]);
047                 prefSelect.addEventListener("change", function () {
048                     fetchWeather(areaCodes[this.value]);
049                 });
050                 regionSelect.addEventListener("change", function () {
051                     showWeather(this.value);
052                 });
053             });
054         </script>
055     </head>
056     <body>
057         <h1>天気予報  対象地域の変更</h1>
038         <label for="prefSelect">都道府県:</label>
039         <select id="prefSelect">
040             <option>東京</option>
041             <option>神奈川</option>
042             <option>静岡</option>
043         </select>
044         <label for="regionSelect">対象地域:</label>
045         <select id="regionSelect"></select>
```

【実行結果】

12行目 fetchWether()で指定したエリアのデータを格納する

33行目 showWether()で指定した地域のデータを表示する

46行目 東京を初期表示にする

47行目 都道府県変更時の処理

50行目 地域変更時の処理

【実行結果】

17　地図情報を取得して表示する

地図情報の API は Google Maps API が有名だが、ここでは完全フリーで API キーがいらない API を使用して、ルート検索をする。地図の表示は、Leaflet.js のライブラリを使い、API から取得した情報を使って表示させる。

js17-1.htm　指定した場所の地図を表示させる

```
001 <!DOCTYPE html>
002 <html lang=”ja”>
003 <head>
004   <meta charset=”UTF-8”>
005   <title>駿府城公園 東御門</title>
006   <meta name=”viewport” content=”width=device-width, initial-scale=1.0”>
007   <!-- Leaflet CSS -->
008   <link rel=”stylesheet” href=”https://unpkg.com/leaflet@1.9.4/dist/leaflet.css” />
009   <style>
```

```
010       #map {
011         height: 600px;
012         width: 100%;
013       }
014     </style>
015   </head>
016   <body>
017     <h2>駿府城公園 東御門の地図</h2>
018     <div id="map"></div>
019     <!-- Leaflet JS -->
020     <script src="https://unpkg.com/leaflet@1.9.4/dist/leaflet.js"></script>
021     <script>
022       // 東御門の座標
023       const eastGate = [34.977715, 138.384739];
024       // 地図の初期化
025       const map = L.map('map').setView(eastGate, 15);
026       // OpenStreetMap タイルレイヤーの追加
027       L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
028         attribution: '© OpenStreetMap contributors'
029       }).addTo(map);
030       // 東御門にマーカーを追加
031       L.marker(eastGate).addTo(map).bindPopup("駿府城公園 東御門");
032     </script>
033   </body>
034   </html>
```

【解説】

6行　name="viewport" は、「ビューポート」という、ブラウザがページを表示するための見える範囲の設定を指定する、という意味

content="width=device-width" は、画面の幅(device-width)に合わせて表示を調整するという設定で、スマホならスマホの幅、タブレットならタブレットの幅に合わせて、ページのレイアウトが調整される

initial-scale=1.0 は、最初のズーム倍率を「1 倍」に設定するという意味で、ページが読み込まれたときに、ズームせずにそのままの倍率で表示される。

よって、モバイル端末でも読みやすく表示され、ユーザーが最初に見たとき、変な拡大・縮小が起きないという意味になる。これがないとページがすごく縮小されて読みにくくなることがあり、今ではモバイル対応には必須のタグになっている。

8行　Leaflet(リーフレット)という地図ライブラリの CSS(スタイル)ファイルを読み込むためのタグ。これがないと、マーカーの見た目が崩れたり、拡大ボタンが焼死されなくなったり、地図の枠やレイアウトがおかしくなったり、不具合が生じる

10～13行 の部分を高さ 600px、幅100%にしている。なので、親要素のブラウザの幅に合わせて、地図が画面いっぱいに横に広がるようになる。
20行 Leaflet(リーフレット)という地図表示ライブラリの JavaScript ファイルを読み込む。これがないと、L.map() や L.marker() など、Leaflet 特有の関数が使えなくなる。
23行 東御門の緯度と経度
25行 map という変数を作り、「作成した地図オブジェクト」を入れておくことで、あとで地図にマーカーを追加したり、動かしたりできるようにしている。L.map('map')の L は Leaflet ライブラリのメインオブジェクト。map() は「地図を作る」という関数。'map' は HTML の中にある の「id」を指定している。つまり:「id が'map'の場所に地図を作る」という意味になる。
.setView(eastGate, 15)
地図の「中心の座標」と「ズームレベル」を設定している。eastGate は [34.977715, 138.384739] という 駿府城公園 東御門の緯度・経度で、15 は ズームレベル(数字が大きいほどズームイン)。
まとめると、「 id="map" の場所に、中心を「東御門」に設定し、ズームレベル 15 で地図を表示する 」という意味になる。
27行 L.tileLayer(...)の tileLayer は 地図のタイル画像(地図の背景)を設定する関数で、地図は「タイル」と呼ばれる小さな画像をたくさん並べて表示されている。
'https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png' は OpenStreetMap(OSM) の地図画像を読み込むための URL パターン。
それぞれの意味
{s}:サブドメイン(a, b, c など)。読み込みを分散するために使われる。
{z}:ズームレベル(0～20 くらい)
{x} & {y}:表示したい地図の位置(タイルの座標)
28行 attribution: '© OpenStreetMap contributors' は地図の著作権表示(クレジット)で、OpenStreetMap の地図を使うときは、この表示が必須。
29行 .addTo(map) は、「このタイルレイヤーを、先ほど作った地図オブジェクト(map)に追加する」という意味で、これがないと、背景地図は表示されない。
という意味になる。この URL のテンプレートを使うことで、Leaflet が自動で必要な画像を読み込んで、地図を表示する。
31行 東御門の位置にピン(マーカー)を立てて、ピンをクリックすると「駿府城公園 東御門」と表示する吹き出しを追加する
【実行結果】

次に、マウスポインタで座標を取得させるプログラムを作る

js17-2.htm　地図の上でマウスポインタの座標を表示させ、クリックしたらコピーする

```
001 <!DOCTYPE html>
002 <html lang="ja">
003 <head>
004   <meta charset="UTF-8">
005   <title>駿府城公園 東御門＋右上に座標、クリックしてコピー</title>
006   <meta name="viewport" content="width=device-width, initial-scale=1.0">
007   <!-- Leaflet CSS -->
008   <link rel="stylesheet" href="https://unpkg.com/leaflet@1.9.4/dist/leaflet.css" />
009   <style>
010     #map {
011       height: 600px;
012       width: 100%;
013     }
014     #coords-display {
015       position: absolute;
```

```
      top: 10px;
      right: 10px;
      background-color: rgba(255,255,255,0.9);
      padding: 6px 12px;
      border: 1px solid #ccc;
      border-radius: 6px;
      font-family: monospace;
      font-size: 14px;
      z-index: 1000;
      box-shadow: 0 0 5px rgba(0,0,0,0.2);
    }
  </style>
</head>
<body>
  <h2>駿駿府城公園 東御門＋右上に座標、クリックしてコピー</h2>
  <div id="coords-display">座標: --</div>
  <div id="map"></div>
  <!-- Leaflet JS -->
  <script src="https://unpkg.com/leaflet@1.9.4/dist/leaflet.js"></script>
  <script>
    const eastGate = [34.977715, 138.384739];
    const map = L.map('map').setView(eastGate, 15);
    // OSM レイヤー
    L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
      attribution: '© OpenStreetMap contributors'
    }).addTo(map);
    // 東御門マーカー
    L.marker(eastGate).addTo(map).bindPopup("駿府城公園 東御門");
    // 座標表示用ラベル
    const coordsDisplay = document.getElementById('coords-display');
    // マウス移動時に座標表示
    map.on('mousemove', function(e) {
      const lat = e.latlng.lat.toFixed(6);
      const lng = e.latlng.lng.toFixed(6);
      coordsDisplay.textContent = `座標: ${lat}, ${lng}`;
    });
    // クリック時に座標をクリップボードにコピー
    map.on('click', function(e) {
```

```
        const lat = e.latlng.lat.toFixed(6);
        const lng = e.latlng.lng.toFixed(6);
        const coordsText = `${lat}, ${lng}`;
        navigator.clipboard.writeText(coordsText)
          .then(() => {
            coordsDisplay.textContent = `コピーしました: ${coordsText}`;
          })
          .catch(err => {
            coordsDisplay.textContent = `コピー失敗: ${err}`;
          });
      });
    </script>
</body>
</html>
```

【実行結果】

次に、地図にルート検索の機能を追加する。

js17-3.htm　クリックした場所から、東御門までのルートを検索して表示させる。

```
<!DOCTYPE html>
<html lang="ja">
<head>
  <meta charset="UTF-8">
  <title>駿府城公園 東御門ルート表示</title>
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <!-- Leaflet CSS -->
  <link rel="stylesheet" href="https://unpkg.com/leaflet@1.9.4/dist/leaflet.css" />
  <style>
    #map {
      height: 600px;
      width: 100%;
    }
    #coords-display {
      position: absolute;
      top: 10px;
      right: 10px;
      background-color: rgba(255,255,255,0.95);
      padding: 6px 12px;
      border: 1px solid #ccc;
      border-radius: 6px;
      font-family: monospace;
      font-size: 14px;
      z-index: 1000;
      box-shadow: 0 0 5px rgba(0,0,0,0.2);
    }
  </style>
</head>
<body>
  <h2>駿府城公園 東御門までの徒歩ルート表示</h2>
  <div id="coords-display">座標: --</div>
  <div id="map"></div>
  <!-- Leaflet JS -->
  <script src="https://unpkg.com/leaflet@1.9.4/dist/leaflet.js"></script>
  <script>
    const eastGate = [34.977715, 138.384739]; // 駿府城公園 東御門
    const map = L.map('map').setView(eastGate, 15);
    // タイルレイヤー追加
```

```
039     L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
040       attribution: '© OpenStreetMap contributors'
041     }).addTo(map);
042     // 東御門マーカー
043     const goalMarker = L.marker(eastGate).addTo(map).bindPopup("駿府城公園 東御門
        ").openPopup();
044     // 座標表示用
045     const coordsDisplay = document.getElementById('coords-display');
046     let routeLayer = null;
047     let startMarker = null;
048     // マウス移動時の座標表示
049     map.on('mousemove', function(e) {
050       const lat = e.latlng.lat.toFixed(6);
051       const lng = e.latlng.lng.toFixed(6);
052       coordsDisplay.textContent = `座標: ${lat}, ${lng}`;
053     });
054     // 地図クリック時の処理
055     map.on('click', function(e) {
056       const lat = e.latlng.lat.toFixed(6);
057       const lng = e.latlng.lng.toFixed(6);
058       const coordsText = `${lat}, ${lng}`;
059       // クリップボードにコピー
060       navigator.clipboard.writeText(coordsText)
061         .then(() => {
062           coordsDisplay.textContent = `コピーしました: ${coordsText}`;
063         })
064         .catch(err => {
065           coordsDisplay.textContent = `コピー失敗: ${err}`;
066         });
067       // 既存マーカー・ルートを削除
068       if (startMarker) map.removeLayer(startMarker);
069       if (routeLayer) map.removeLayer(routeLayer);
070       // 出発マーカー
061       startMarker = L.marker([lat, lng]).addTo(map).bindPopup("出発地点").openPopup();
062       // OSRM ルート API で徒歩ルート取得
063       const osrmUrl = `https://router.project-osrm.org/route/v1/foot/${lng},${lat};${eastGate[1]},
        ${eastGate[0]}?overview=full&geometries=geojson`;
064       fetch(osrmUrl)
```

```
065         .then(res => res.json())
066         .then(data => {
067           const coords = data.routes[0].geometry.coordinates;
068           const latlngs = coords.map(c => [c[1], c[0]]);
069           // ルート表示
070           routeLayer = L.polyline(latlngs, {
061             color: 'green',
062             weight: 5,
063             opacity: 0.8
064           }).addTo(map).bindPopup("徒歩ルート");
065         })
066         .catch(err => {
067           alert("ルート取得に失敗しました。");
068           console.error(err);
069         });
070     });
061   </script>
062 </body>
063 </html>
```

【実行結果】

最後に、出発地点と目的地を自分で決めて、ルート検索できるようにする。右クリックするとリセットする機能も付ける。

17-4.htm　クリックした場所から、東御門までのルートを検索して表示させる。

```
<!DOCTYPE html>
<html lang="ja">
<head>
  <meta charset="UTF-8">
  <title>クリックでルート表示(START / GOAL 文字のみ)</title>
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="https://unpkg.com/leaflet@1.9.4/dist/leaflet.css" />
  <style>
    #map {
      height: 600px;
      width: 100%;
    }
    #coords-display {
```

```
014         position: absolute;
015         top: 10px;
016         right: 10px;
017         background-color: rgba(255,255,255,0.95);
018         padding: 6px 12px;
019         border: 1px solid #ccc;
020         border-radius: 6px;
021         font-family: monospace;
022         font-size: 14px;
023         z-index: 1000;
024         box-shadow: 0 0 5px rgba(0,0,0,0.2);
025       }
026       .label-text {
027         font-weight: bold;
028         font-size: 16px;
029         white-space: nowrap;
030         color: black;
031         background: none;
032         border: none;
033         padding: 0;
034         box-shadow: none;
035       }
036     </style>
037   </head>
038   <body>
039     <h2>クリックで徒歩ルートを表示(START / GOAL 文字のみ)</h2>
040     <div id="coords-display">座標: --</div>
041     <div id="map"></div>
042     <script src="https://unpkg.com/leaflet@1.9.4/dist/leaflet.js"></script>
043     <script>
        const map = L.map('map').setView([34.977715, 138.384739], 15); // 駿府城公園周辺
044       L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
045         attribution: '© OpenStreetMap contributors'
046       }).addTo(map);
047       const coordsDisplay = document.getElementById('coords-display');
048       let startCoord = null;
049       let endCoord = null;
050       let routeLine = null;
```

```
051     let startMarker = null;
052     let endMarker = null;
053     let startLabel = null;
054     let endLabel = null;
055     // 座標表示
056     map.on('mousemove', function(e) {
057       coordsDisplay.textContent = `座標：${e.latlng.lat.toFixed(6)}, ${e.latlng.lng.toFixed(6)}`;
058     });
059     // ラベル生成用関数(文字のみ)
060     function createLabel(latlng, text) {
061       return L.marker(latlng, {
062         icon: L.divIcon({
063           className: 'label-text',
064           html: text,
065           iconAnchor: [-10, -10], // マーカーの上・左寄りに表示
066         }),
067         interactive: false
068       }).addTo(map);
069     }
070     // ピンマーカー
071     function createPinMarker(latlng) {
072       return L.marker(latlng).addTo(map);
073     }
074     // 左クリック → START と GOAL を設定
065     map.on('click', function(e) {
066       const latlng = e.latlng;
067       if (!startCoord) {
068         startCoord = latlng;
069         startMarker = createPinMarker(latlng);
070         startLabel = createLabel(latlng, 'START');
061         coordsDisplay.textContent = `出発地点：${latlng.lat.toFixed(6)}, ${latlng.lng.toFixed(6)}`;
062       } else if (!endCoord) {
063         endCoord = latlng;
064         endMarker = createPinMarker(latlng);
065         endLabel = createLabel(latlng, 'GOAL');
066         coordsDisplay.textContent = `目的地：${latlng.lat.toFixed(6)},
```

```
      ${latlng.lng.toFixed(6)}`;
067       const url = `https://router.project-osrm.org/route/v1/foot/${startCoord.lng},${startCoord.lat};${endCoord.lng},${endCoord.lat}?overview=full&geometries=geojson`;
068       fetch(url)
069         .then(res => res.json())
070         .then(data => {
071           const coords = data.routes[0].geometry.coordinates;
072           const latlngs = coords.map(c => [c[1], c[0]]);
073           routeLine = L.polyline(latlngs, {
074             color: 'blue',
075             weight: 5,
076             opacity: 0.8
077           }).addTo(map).bindPopup("徒歩ルート");
078         })
079         .catch(err => {
080           alert("ルートの取得に失敗しました。");
070           console.error(err);
071         });
072     }
073   });
074   // 右クリック → リセット
075   map.on('contextmenu', function(e) {
076     startCoord = null;
077     endCoord = null;
078     if (routeLine) {
079       map.removeLayer(routeLine);
080       routeLine = null;
070     }
071     [startMarker, endMarker, startLabel, endLabel].forEach(m => {
072       if (m) map.removeLayer(m);
073     });
074     startMarker = null;
075     endMarker = null;
076     startLabel = null;
077     endLabel = null;
078     coordsDisplay.textContent = "リセットしました。座標: --";
079   });
```

```
080    </script>
070  </body>
071  </html>
```

【実行結果】