



Woodland  
Academy

## 4.5 デバッグ

- デバッグの操作方法



*Shape Your Future*

# 目次

## 1 デバッグの操作方法



# デベロッパーツールの表示

- tryのソースをChromeブラウザでウェブサイトを表示し、[右クリック] → [検証]を選択することで、デベロッパーツールが表示されます。

ショートカットキーはF12キー（macOSの場合はCommand + Option + Iキー）です。



1 + 2 =

要素

コンソール

ソース

ネットワーク

パフォーマンス

メモリ

アプリケーション

スタイル

計算済み

レイアウト

イベントリスナー

フィルタ

:hov .cls +

element.style {

}

body {

display: flex;

justify-content: center;

align-items: center;

background-color: #EEEEEE;

padding: 0;

overflow: hidden;

}

html, body {

height: 100%;

}

body {

display: block;

margin: 8px;

}

style.css:6

style.css:2

ユーザー エージェント スタイルシート

margin 8

border -

padding -

642.400x721.600

8

8

8

- まず、JavaScript(.js)のファイルを開くために、画面上にある[Sources]タブを選択します。





- ここで画面左にある[ソース]→[ページ]ツリーから、対象のJSのファイルを選択します。するとメインエリアに選択したファイルの内容が表示されます。

The screenshot shows the 'Sources' panel of a web browser. The file tree on the left shows the following structure:

- top
  - 127.0.0.1:5503
    - .vscode
      - index.html
      - main.js** (selected)
      - style.css

The main area displays the content of 'main.js':

```

1 // DOM要素を参照
2 const elementSelect = document.querySelector("#calcT
3 const elementNum1 = document.querySelector("#num1");
4 const elementNum2 = document.querySelector("#num2");
5 const elementResult = document.querySelector("#resul
6
7 // イベントを登録
8 elementSelect.addEventListener("change", update);
9 elementNum1.addEventListener("change", update);
10 elementNum2.addEventListener("change", update);
11
12 /** 計算し画面に結果を表示します。 */
13 function update() {
14     // 計算結果を求める
15     const result = calculate(
16         Number(elementNum1.value), // 1番目のテキスト
17         Number(elementNum2.value), // 2番目のテキスト
18         elementSelect.value // セレクトボックスの値 (
19     );
20
21     // 画面に表示
22     elementResult.innerHTML = result; // テキストを代
23 }
24
25 /** 計算します。 */
  
```

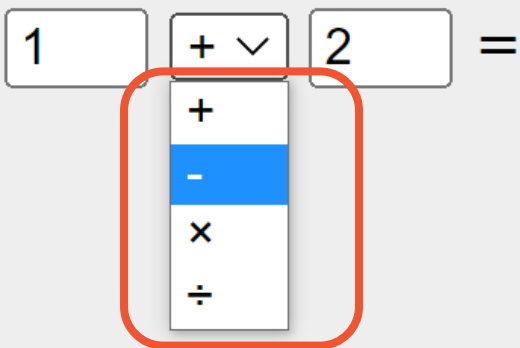
The right sidebar shows the 'Breakpoints' panel with the following items:

- 監視
- ブレークポイント
  - ブレークポイントがありません
- 範囲
  - 一時停止されていません
- コールスタック
  - 一時停止されていません
- XHR / フェッチ ブレークポイント
- DOM ブレークポイント
- グローバルリスナー
- イベントリスナー ブレークポイント
- CSP 違反ブレークポイント

# ブレークポイントの設定

- ブレークポイントは、プログラムの実行中に任意の行で処理を止める機能です。この機能を使用すれば、使用している変数の現在の状態が確認できます。
- メインエリアで表示されているソースコードの行番号の部分をクリックすると、青い矢印が表示されブレークポイントが設定されます。ブレークポイントを設定した状態でプログラムの実行中にこの行を通ると実行が一時的に止まります。青い矢印を再度クリックすることで設定したブレークポイントが削除されます。

- 試しに、先ほど表示したソースコードの27行目（let resultの箇所）にブレークポイントを設定し、サンプルのプルダウンの中から「-」（引き算）を選択しましょう。



main.js x

```

1 // DOM要素を参照
2 const elementSelect = document.querySelector("#calcT
3 const elementNum1 = document.querySelector("#num1");
4 const elementNum2 = document.querySelector("#num2");
5 const elementResult = document.querySelector("#resul
6
7 // イベントを登録
8 elementSelect.addEventListener("change", update);
9 elementNum1.addEventListener("change", update);
10 elementNum2.addEventListener("change", update);
11
12 /** 計算し画面に結果を表示します。 */
13 function update() {
14     // 計算結果を求める
15     const result = calculate(
16         Number(elementNum1.value), // 1番目のテキスト
17         Number(elementNum2.value), // 2番目のテキスト
18         elementSelect.value // セレクトボックスの値 (
19     );
20
21     // 画面に表示
22     elementResult.innerHTML = result; // テキストを代
23 }
24
25 /** 計算します。 */
26 function calculate(num1, num2, calcType) {
27     let result;
28     // 計算の種類で処理を分岐
29     switch (calcType) {
30         case "type-add": // 足し算の場合

```

ブレークポイント: main.js:27 let result;



- すると、ブレークポイントの箇所でスクリプトの実行が一時的に停止します。

デバッガ内で一時停止

ブレークポイントで一時停止しました

main.js:27  
let result;

ローカル  
this: Window  
calcType: "type-subtract"  
num1: 1  
num2: 2  
result: undefined

スクリプト  
グローバル Window

コールスタック  
calculate main.js:27  
update main.js:15  
XHR / フェッチ ブレークポイント  
DOM ブレークポイント  
グローバルリスナー

```

1 // DOM要素を参照
2 const elementSelect = document.querySelector("#calcType");
3 const elementNum1 = document.querySelector("#num1");
4 const elementNum2 = document.querySelector("#num2");
5 const elementResult = document.querySelector("#result");
6
7 // イベントを登録
8 elementSelect.addEventListener("change", update);
9 elementNum1.addEventListener("change", update);
10 elementNum2.addEventListener("change", update);
11
12 /** 計算し画面に結果を表示します。 */
13 function update() {
14     // 計算結果を求める
15     const result = calculate(
16         Number(elementNum1.value), // 1番目のテキスト
17         Number(elementNum2.value), // 2番目のテキスト
18         elementSelect.value // セレクトボックスの値
19     );
20
21     // 画面に表示
22     elementResult.innerHTML = result; // テキストを代入
23 }
24
25 /** 計算します。 */
26 function calculate(num1, num2, calcType) {
27     let result;
28     // 計算の種類で処理を分岐
  
```

# 変数の確認

- ブレークポイントでプログラムの実行を止めると、プログラム内の変数の現在の値を見ることができます。
- 変数上にマウスポインターを乗せることで変数の値の確認が可能です。数値や文字列などの簡単な情報であればこの簡易的な表示で十分でしょう。

```

23 }
24
25 /** 計算します。 */
26 function calculate(num1, num2, calcType) { num1 = 1, num2 = 2, calcType = "type-subtract"
27     let result;
28     // 計算の種類で処理を分岐

```

1

# スコープ内の変数の値を一覧で確認

- 複雑なオブジェクトの変数の詳細を確認したい場合、[Scope]パネルが役に立ちます。ここに現在実行中のJavaScriptのスコープの変数が表示されます。関数内のローカル変数であれば、Localツリーに表示されます。

The screenshot shows the VS Code editor with a file named `main.js` open. The code is as follows:

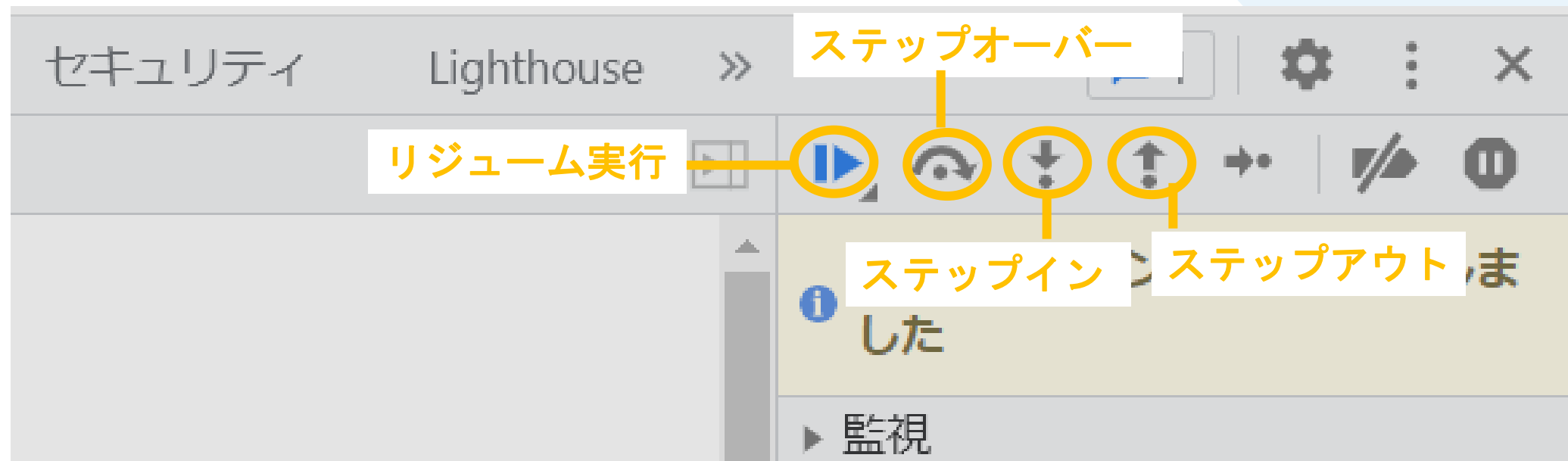
```
1 // DOM要素を参照
2 const elementSelect = document.querySelector("#calcType");
3 const elementNum1 = document.querySelector("#num1");
4 const elementNum2 = document.querySelector("#num2");
5 const elementResult = document.querySelector("#result");
6
7 // イベントを登録
8 elementSelect.addEventListener("change", update);
9 elementNum1.addEventListener("change", update);
10 elementNum2.addEventListener("change", update);
11
12 /** 計算し画面に結果を表示します。 */
13 function update() {
14     // 計算結果を求める
15     const result = calculate(
16         Number(elementNum1.value), // 1番目のテキスト入力フォームの値
17         Number(elementNum2.value), // 2番目のテキスト入力フォームの値
18         elementSelect.value // セレクトボックスの値 (計算の種類)
19     );
20
21     // 画面に表示
22     elementResult.innerHTML = result; // テキストを代入
23 }
24
25 /** 計算します。 */
26 function calculate(num1, num2, calcType) {
27     let result;
28     // 計算の種類で処理を分岐
```

A breakpoint is set at line 27. The right-hand side of the image shows the **Scope** panel. It displays the following information:

- ブレークポイントで一時停止しました** (Paused at breakpoint)
- 監視** (Watch)
- ブレークポイント** (Breakpoints): `main.js:27` is checked.
- 範囲** (Scope):
  - ローカル** (Local):
    - `this: Window`
    - `calcType: "type-subtract"`
    - `num1: 1`
    - `num2: 2`
    - `result: undefined`
  - スクリプト** (Script)
  - グローバル** (Global): `Window`
- コールスタック** (Call Stack):
  - `calculate` at `main.js:27`
  - `update` at `main.js:15`
  - `XHR / フェッチ ブレークポイント`
  - `DOM ブレークポイント`
  - `グローバルリスナー`

# プログラムのステップ実行

- ブレークポイントで止まったプログラムは画面右上のボタンで続きを実行できます。この機能を使えばプログラムが実行される処理の流れを確認できます。



- リジューム実行

ブレークポイントで止まっていたプログラムの続きを実行します。複数のブレークポイントを設定することで想定通りの処理順になっているか確認できます。

- ステップイン

ブレークポイントで処理が止まっている時、その行のプログラムはまだ実行されていない状態です。ステップインを実行すると、現在の行のプログラムを実行し、次の処理の行へ移動してそこでまたプログラムが止まります。プログラムが実行される処理の流れを1つひとつ確認できます。

- ステップインのように1行ずつ確認するのに対して、ステップオーバーとステップアウトは大まかな処理の流れを確認するのに役立ちます。
- ステップオーバーは現在止まっている行で関数が実行されている場合に、その関数の内部の処理をすべて実行して次の行で止まります。
- ステップアウトは現在処理が止まっている関数内の残りの処理をすべて実行し、関数を呼び出している行へ移動します。



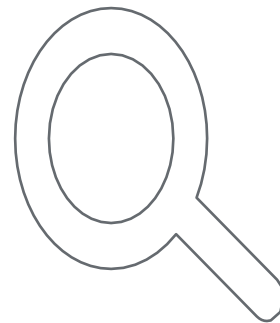
- デバック操作は、JAVAのEclipseの操作でも取り扱いました。デバック操作ができることで、様々なプログラムの問題に対応することができます。

“デバッグを制するものはプログラムを制す” by 私

- 実際の現場でも、エラーや問題があった際には、デバッグを使用し、自分の力で問題点を解決できるようになってください。



Q&A



# まとめ

Sum Up



## 1. デバックの操作方法及び各用語の意味

- ・ ステップイン
- ・ ステップオーバー
- ・ ステップアウト

# Thank you!

From Seeds to Woodland — Shape Your Future.



*Shape Your Future*