# 17章 イベント処理を理解しよう

Webサイトでユーザーの操作によって発生するイベントについて解説します。

120分 - 読了

## 17.1 本章の目標

本章では以下を目標にして学習します。

- イベント処理とは何か、概要をつかむこと
- イベント処理の作り方を知ること
- イベント処理を実際に作成すること

これまで2章にわたってDOM操作について学び、JavaScriptからHTML要素を取得したり、作成したりしました。

では、「ボタンをクリックしたとき」「画面をスクロールしたとき」など、特定のタイミングでDOM操作を行いたい場合はどうすればよいのでしょうか。

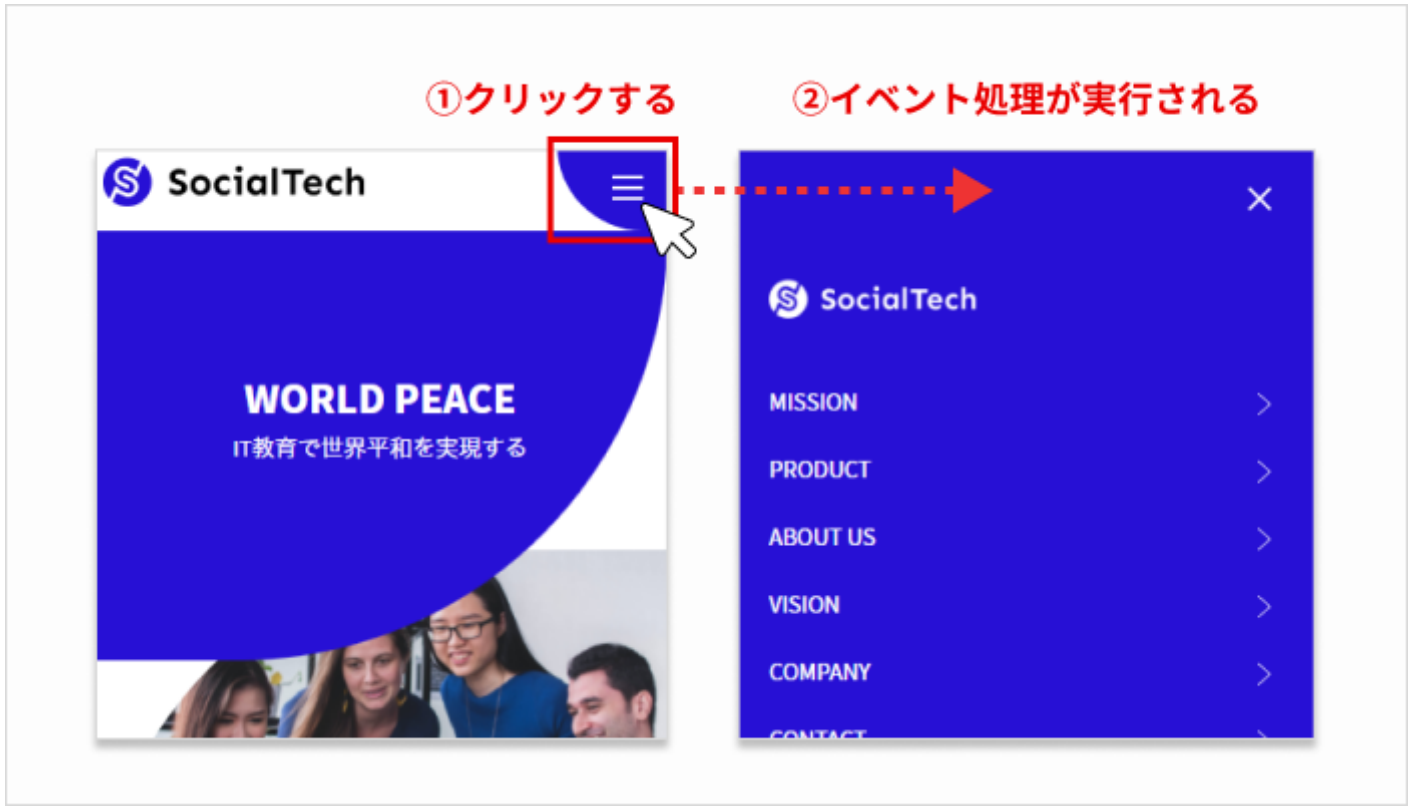
実際にWebサイトを閲覧する際、ユーザーは以下のような行動をします。

- ボタンをクリックする
- マウスカーソルを任意の場所に移動する
- キーボードで文字を入力する
- フォームの内容を送信する
- 画面をスクロールする
- ファイルを画面にドラッグ＆ドロップする

ユーザーがこのような行動をする理由はいくつかあります。

例えば、ナビゲーションメニューを表示するためにハンバーガーメニュー（横線3本のアイコン）をクリックしたり、ユーザー登録を行うためにフォームの内容を送信したりします。

このようなユーザーの行動に合わせて、あらかじめプログラムしていたDOM操作を行うのが**イベント処理**です。




本章ではイベント処理について学び、好きなタイミングでDOM操作を行えるようになりましょう。

## 17.2 HTML要素を取得・作成するメソッドを復習しよう

本章では、前章で学んだHTML要素を取得・作成するメソッドを多用します。そこで、まずはこれらのメソッドについて復習しておきましょう。暗記する必要はないので、ざっと確認してください。

| メソッド | 処理の内容 |
| --- | --- |
| `getElementById()` | HTML要素を**id**で取得する |
| `getElementsByClassName()` | HTML要素を**class**で取得する |
| `querySelector()` | HTML要素を**CSSセレクタ**で取得する（**最初の１つ**） |
| `querySelectorAll()` | HTML要素を**CSSセレクタ**で取得する（**すべて**） |
| `createElement()` | HTML要素を新しく作成する |
| `appendChild()` | HTML要素を子要素として末尾に追加する |

| プロパティ | 値 |
| --- | --- |
| `length` | 文字列や配列の長さ |
| `textContent` | HTML要素内のテキスト |
| `innerHTML` | HTML要素内のテキスト（HTMLタグも可） |

## 17.3 イベント処理とは

そもそもイベントとは何でしょうか。

イベントを直訳すると「出来事」という意味ですが、DOM操作におけるイベントとは、ブラウザを利用する以下のような「**ユーザーの行動**」を意味します。

- ボタンをクリックする
- マウスカーソルを任意の場所に移動する
- キーボードで文字を入力する
- フォームの内容を送信する
- 画面をスクロールする
- ファイルを画面にドラッグ＆ドロップする

つまりイベント処理とは、上記のような**ユーザーの行動に合わせてDOM操作を行うこと**です。

## 17.4 イベント処理の作り方

イベント処理は、主に `addEventListener()` メソッドを使って以下のように作成します。

```
HTML要素.addEventListener('イベントの種類', () => {
  イベント処理
});

```

とても複雑なコードに見えますが、以下のように2つの引数を渡しているだけです。「関数」の部分にアロー関数を記述して見やすいように改行すると、上記のコードになります。

```
HTML要素.addEventListener('イベントの種類', 関数);

```

「HTML要素」の部分には、前章で学んだ `getElementById()` メソッドなどを使って取得したHTML要素を記述します。

また、「イベントの種類」の部分には以下のように、どのようなタイミングで処理を実行するのか指定します。以下の表を見てイメージをつかんでおきましょう（覚える必要はありません）。

| イベントの種類 | 処理が実行されるタイミング |
| --- | --- |
| click | クリックしたとき（マウスボタンを押して離したとき） |
| mousedown | マウスボタンを押したとき |
| mouseup | マウスボタンを離したとき |
| mousemove | マウスカーソルを移動したとき |
| keydown | キーボードのキーを押したとき |
| keyup | キーボードのキーを離したとき |
| submit | フォームを送信したとき |
| focus | HTML要素にフォーカスしたとき |
| scroll | 画面をスクロールしたとき |

### イベント処理を作成してみよう

では、実際にイベント処理を作成してみましょう。今回は、ボタンがクリックされたときにコンソールに文字列を出力するイベント処理を作成します。


まずはVisual Studio Codeを開き、 `javascript-basic` フォルダ直下（ `index.html` と同じ階層）に `event.html` というファイルを作成してください。

続いて、 `event.html` を以下のように編集しましょう。 `script` タグ内で読み込んでいる `event.js` は、このあと作成します。

`event.html`

```
<!DOCTYPE html>
<html lang="ja">

<head>
  <meta charset="UTF-8">
  <title>イベント処理</title>
</head>

<body>
  <button id="output-btn">文字列の出力</button>
  <script src="js/event.js"></script>
</body>

</html>

```

ここで `event.html` をブラウザで開くと、以下のように「文字列の出力」ボタンが表示されます。しかし、まだイベント処理を作成していないため、クリックしても何も起こりません。

文字列の出力

では次に、 `js` フォルダ内に新しく `event.js` というファイルを作成してください。 `event.js` を作成したら、以下のように編集しましょう。

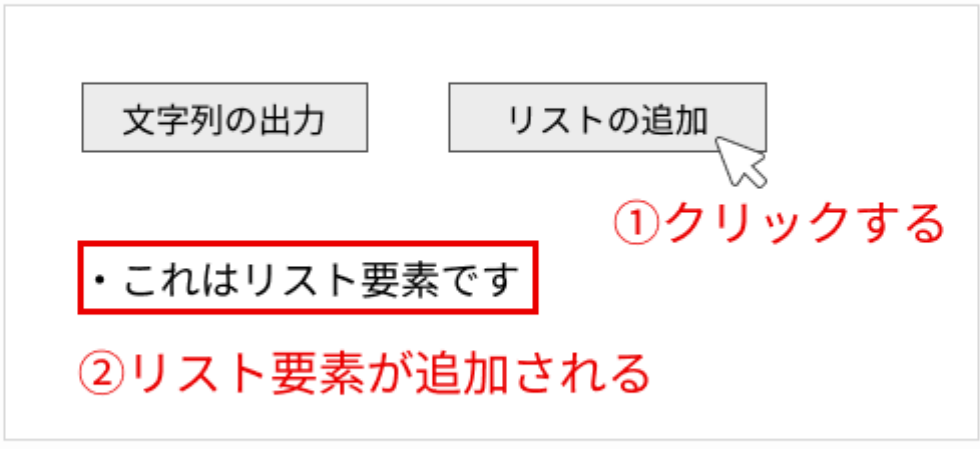
`event.js`

```
// output-btnというidを持つHTML要素を取得し、定数に代入する
const btn = document.getElementById('output-btn');

// HTML要素がクリックされたときにイベント処理を実行する
btn.addEventListener('click', () => {
  console.log('クリックされました！');
});

```

続いて `event.html` をブラウザで開き、デベロッパーツールのコンソールを確認してみましょう。開いた時点では何も表示されていませんが、画面左上の「実行」ボタンをクリックすると、文字列が出力されます。


## 17.5 クリック時にリストを追加してみよう

続いて、もう少し複雑なイベント処理を作ってみましょう。ボタンをクリックしたときに、リスト（ `li` 要素）が1つ追加されるイベント処理です。




まずは `event.html` を以下のように編集し、「リストの追加」ボタンを作成しましょう。また、あとから `appendChild()` メソッドを使って子要素に `li` 要素を追加するため、あらかじめ `ul` 要素を作成しておきます。

event.html

```
<!DOCTYPE html>
<html lang="ja">

<head>
  <meta charset="UTF-8">
  <title>イベント処理</title>
</head>

<body>
  <button id="output-btn">文字列の出力</button>

+ <button id="add-btn">リストの追加</button>
+ <ul id="parent-list"></ul>

  <script src="js/event.js"></script>
</body>

</html>

```

ここで `event.html` をブラウザで開くと、以下のように「リストの追加」ボタンが表示されます。しかし、まだイベント処理を作成していないため、クリックしても何も起こりません。


続いて `event.js` を以下のように編集し、 `button` 要素と `ul` 要素を取得しましょう。

event.js

```
// output-btnというidを持つHTML要素を取得し、定数に代入する
const outputBtn = document.getElementById('output-btn');

// HTML要素がクリックされたときにイベント処理を実行する
outputBtn.addEventListener('click', () => {
  console.log('クリックされました！');
});

+ // add-btnというidを持つHTML要素を取得し、定数に代入する
+ const addBtn = document.getElementById('add-btn');
+ // parent-listというidを持つHTML要素を取得し、定数に代入する
+ const parentList = document.getElementById('parent-list');

```

ではここで、どのようにイベント処理を記述すればよいか考えてみましょう。イベント処理の内容は以下のとおりです。

1. `li` 要素を新しく作成する
2. 作成した `li` 要素に「これはリスト要素です」というテキストを追加する
3. 作成した `li` 要素を `ul` 要素の子要素として末尾に追加する

以下のメソッドとプロパティを使い、まずは自分でコードを書いてみてください。

| メソッド | 処理の内容 |
| --- | --- |
| `addEventListener()` | イベント処理を実行する |

| メソッド | 処理の内容 |
| --- | --- |
| `createElement()` | HTML要素を新しく作成する |
| `appendChild()` | HTML要素を子要素として末尾に追加する |

| プロパティ | 値 |
| --- | --- |
| `textContent` | HTML要素内のテキスト |

では順番に正解のコードを見ていきましょう。

## 1. li要素を新しく作成する

event.js

```
//======== 前略 ========

// add-btnというidを持つHTML要素を取得し、定数に代入する
const addBtn = document.getElementById('add-btn');
// parent-listというidを持つHTML要素を取得し、定数に代入する
const parentList = document.getElementById('parent-list');

+ // HTML要素がクリックされたときにイベント処理を実行する
+ addBtn.addEventListener('click', () => {
+   // li要素を新しく作成する
+   const childList = document.createElement('li');
+ });

```

## 2. 作成したli要素に「これはリスト要素です」というテキストを追加する

event.js

```
//======== 前略 ========

// add-btnというidを持つHTML要素を取得し、定数に代入する
const addBtn = document.getElementById('add-btn');
// parent-listというidを持つHTML要素を取得し、定数に代入する
const parentList = document.getElementById('parent-list');

// HTML要素がクリックされたときにイベント処理を実行する
addBtn.addEventListener('click', () => {
  // li要素を新しく作成する
  const childList = document.createElement('li');

+   // 作成したli要素に「これはリスト要素です」というテキストを追加する
+   childList.textContent = 'これはリスト要素です';
});

```

## 3. 作成したli要素をul要素の子要素として末尾に追加する

event.js

```
//======== 前略 ========

// add-btnというidを持つHTML要素を取得し、定数に代入する
const addBtn = document.getElementById('add-btn');
// parent-listというidを持つHTML要素を取得し、定数に代入する
const parentList = document.getElementById('parent-list');

// HTML要素がクリックされたときにイベント処理を実行する
addBtn.addEventListener('click', () => {
  // li要素を新しく作成する
  const childList = document.createElement('li');

  // 作成したli要素に「これはリスト要素です」というテキストを追加する
  childList.textContent = 'これはリスト要素です';

+ // 作成したli要素をul要素の子要素として末尾に追加する
+ parentList.appendChild(childList);
});

```

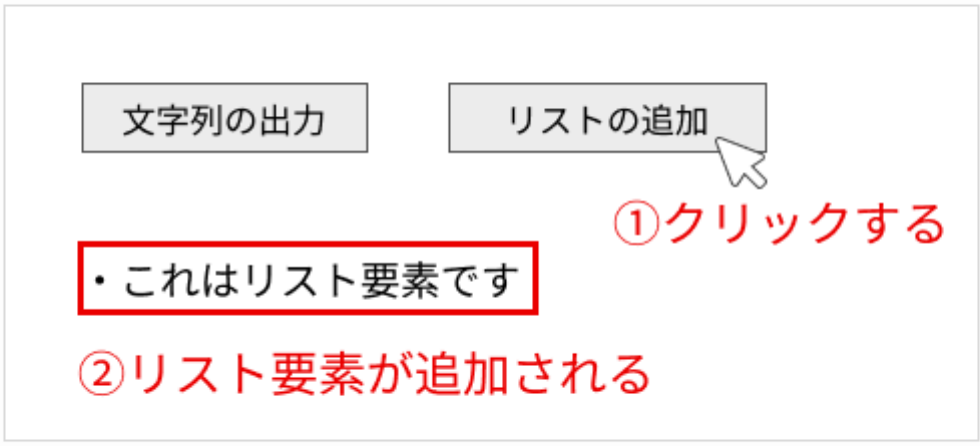
最終的なコード最終的なコードは以下のとおりです。

event.js

```
//======== 前略 ========

// add-btnというidを持つHTML要素を取得し、定数に代入する
const addBtn = document.getElementById('add-btn');
// parent-listというidを持つHTML要素を取得し、定数に代入する
const parentList = document.getElementById('parent-list');

// HTML要素がクリックされたときにイベント処理を実行する
addBtn.addEventListener('click', () => {
  // li要素を新しく作成する
  const childList = document.createElement('li');

  // 作成したli要素に「これはリスト要素です」というテキストを追加する
  childList.textContent = 'これはリスト要素です';

  // 作成したli要素をul要素の子要素として末尾に追加する
  parentList.appendChild(childList);
});

```

## ブラウザで確認してみよう

では `event.html` をブラウザで開き、デベロッパーツールのコンソールを確認してみましょう。「リストの追加」ボタンをクリックしたときに、以下のようにリストが追加されればOKです。




## まとめ

本章では以下の内容を学習しました。

- イベントは「**ユーザーの行動**」を意味する
- イベント処理とは、**ユーザーの行動に合わせてDOM操作を行うこと**である

```
// イベント処理の作り方
HTML要素.addEventListener('イベントの種類', () => {
  イベント処理
});

```

| イベントの種類 | 処理が実行されるタイミング |
| --- | --- |
| click | クリックしたとき（マウスボタンを押して離したとき） |
| mousedown | マウスボタンを押したとき |
| mouseup | マウスボタンを離したとき |
| mousemove | マウスカーソルを移動したとき |
| keydown | キーボードのキーを押したとき |
| keyup | キーボードのキーを離したとき |
| submit | フォームを送信したとき |
| focus | HTML要素にフォーカスしたとき |
| scroll | 画面をスクロールしたとき |

次章では、イベント処理でフォームを操作する方法を学びます。

## 理解度を選択して次に進みましょう

ボタンを押していただくと次の章に進むことができます

～50%　50～80%　80～100%

## 最後に確認テストを行いましょう

下のボタンを押すとテストが始まります。

教材をみなおす　テストをはじめる

前に戻る　21 / 26 ページ　次に進む

一覧に戻る

改善点のご指摘、誤字脱字、その他ご要望はこちらからご連絡ください。