DOM操作

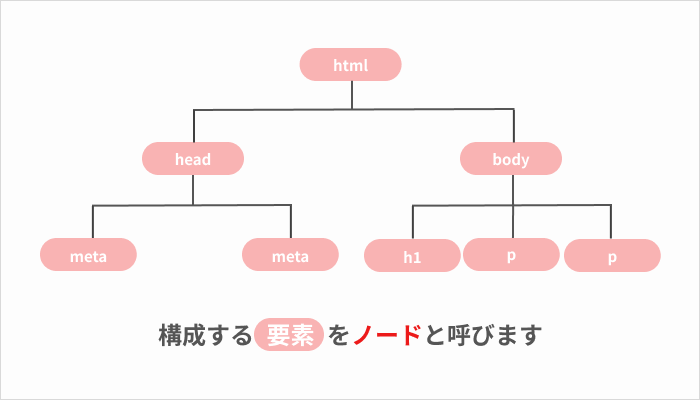
DOMはHTMLドキュメントをオブジェクトのように扱える仕組みのこと

JavaScriptからHTML要素を取得、作成、更新、削除するといった操作ができる



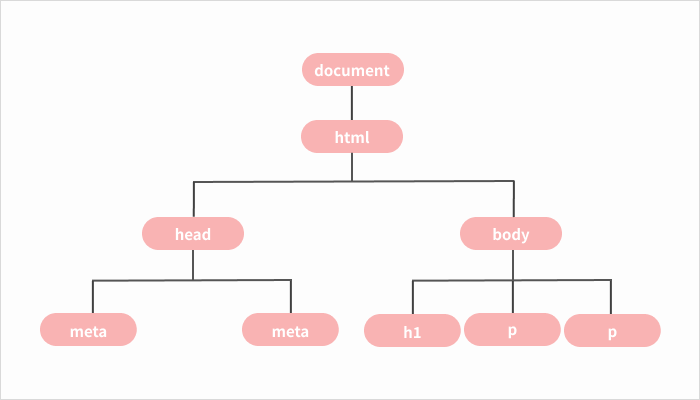
このような事が出来るのがDOM操作

ブラウザはHTMLファイルを読み込むと自動的にこんな感じのDOMツリーを生成する



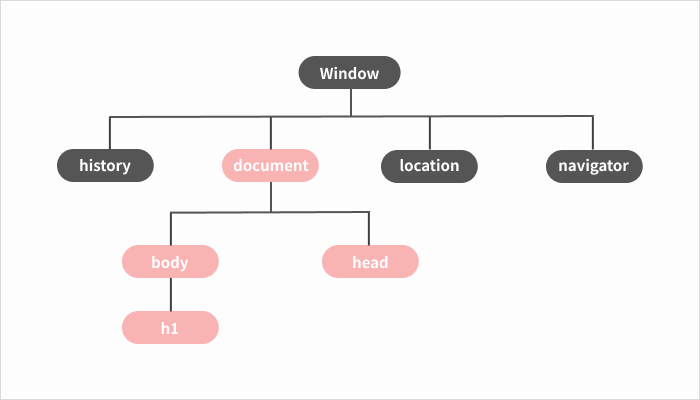
DOMツリーの先頭（htmlの上）

DOMツリーが生成されるとこのドキュメントオブジェクトが使えるようになり、さまざまなDOM操作を行えるようになる。



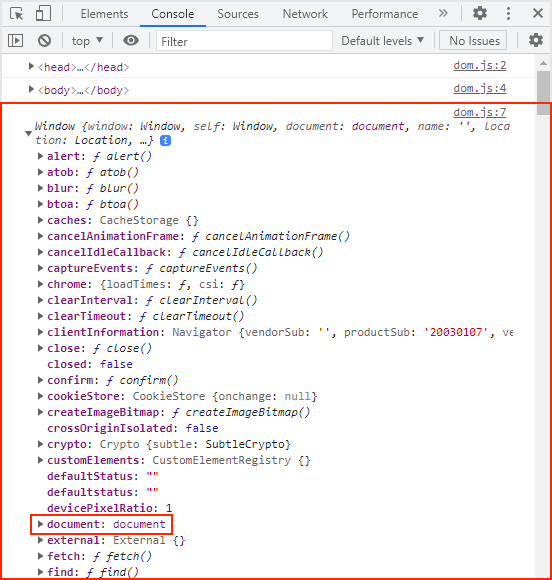
補足 windouwオブジェクトについて

ドキュメントオブジェクトより更に上の階層に存在するwindowオブジェクトがある



windowオブジェクトとはブラウザが自動的に生成するオブジェクトのこと、windowオブジェクトを使うことでJsからブラウザの情報のさまざまな情報にアクセスできるようになる

出力させるとwindowオブジェクトには大量のプロパティ（キーと値のセット）が存在する事が分かる



プロパティによって値にfと表示されているが、これは関数を意味するfunctionの頭文字、つまりメソッドが定義されている

このようにドキュメントオブジェクトはwindowオブジェクトの中に入っているため、以下の２つのコードは同じ動作になる

window.document.head;

document.head;

正確にはwindow.document.headだが、windowの記述は省略できるのでdocument.headと記述しても正しく実行できる

DOM操作でHTML要素を取得する

DOM操作を行うときはまず「どのHTML要素に動きを加えるか」という「操作の対象」を取得しないといけない

HTML要素を取得しないとDOM操作ができないということ

document.html 要素名を記述してHTML要素を取得できるが

しかしdocumentオブジェクトのメソッドを使えば取得するHTML要素をより細かく指定できる

HTML要素を取得するための主なメソッドは以下のとおり（イメージ程度でＯＫ

getElementById() HTML要素をidで取得する

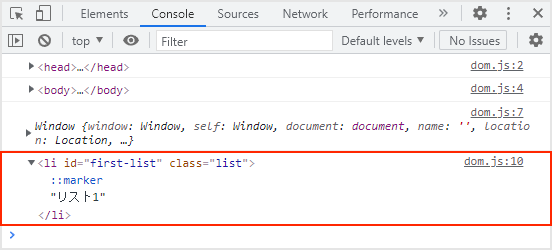
getElementsByClassName()　HTML要素をclassで取得する

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）

getElementById()　HTML要素をidで取得する

指定したidを持つHTML要素を１つ取得するメソッドです、idは同じHTML内で１回しか設定できない固有の値なので、取得できるHTML要素も当然１つだけ

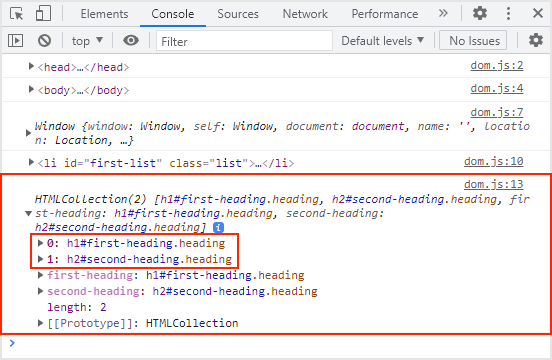


getElementsByClassName() HTML要素をclassで取得する

指定したclassを持つHTML要素をすべて取得するメソッド

idとは異なりclassは複数のHTML要素に設定できるので、そのすべてのHTML要素を取得する（そのため「getElement」ではなく「getElements」と複数形になっている

表示されるとこんな感じ



複数のHTML要素を１つずつ取得する方法

今回のようにgetElementsByClassName() メソッドを使って複数のHTML要素を取得した場合、HTMLcollectionという配列風のデータが戻り値として返ってくる

厳密にはHTMLcollectionは配列ではなく、配列に似ているデータ

lengthプロパティ

文字列の文字数や配列の要素数を取得するプロパティ、HTMLcollectionにも使う事ができる

querySelector() HTML要素をCSSセレクタで取得する（最初の一つ）

指定したCSSセレクタに最初に合致したHTML要素を取得するメソッド

要素名で取得する例

document.querySelector(“h1”);

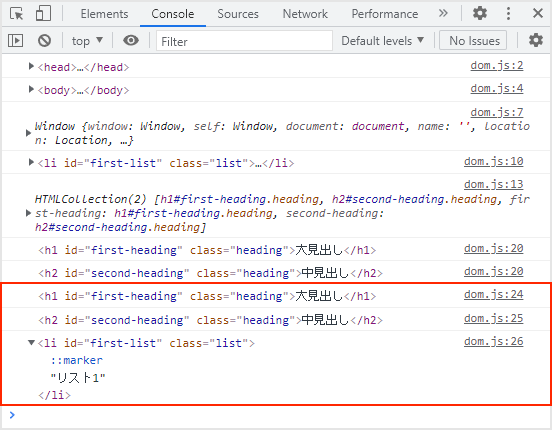
id名で取得する例

document.querySelector(“#second-heading”);

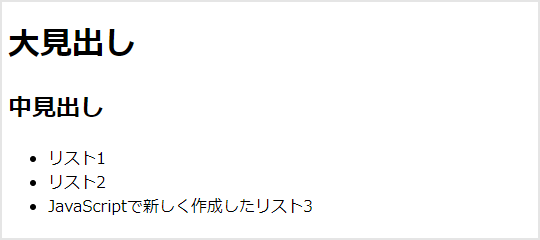
class名で取得する例

document.querySelector(“.list”);

出力するとこんな感じのが出る



DOM操作をする



こんな感じでリストを追加するには、以下のメソッドやプロパティを使う

1. createElement()メソッド：HTML要素を新しく作成する
2. textContent/ innerHTMLプロパティ：HTML要素にテキストを追加する
3. appendChild()メソッド：作成したHTML要素を子要素として末尾に追加する

### **createElement()メソッド**

HTML要素を新しく作成するにはcreateElement()メソッドを使います。引数には以下のように、作成したいHTML要素名を指定します。

// 新しくli要素を作成し、定数に代入する 2const li = document.createElement('li'); 3

### **textContent / innerHTMLプロパティ**

作成したHTML要素にテキストを追加するには、textContentプロパティやinnerHTMLプロパティに値を代入します。どちらも文字列を追加する点では一緒ですが、innerHTMLは通常の文字列に加えてHTMLタグも記述できます。

以下の例では、innerHTMLを使ってリンク付きのテキストを追加しています。なお、以下は見本なのでtextContentプロパティとinnerHTMLプロパティの両方を記載していますが、実際にはどちらか1つのみを使います。

// 新しくli要素を作成し、定数に代入する

const li = document.createElement('li');

// 作成したli要素にテキストを追加する

li.textContent = 'JavaScriptで新しく作成したリスト3'; li.innerHTML = '<a href="#">JavaScriptで新しく作成したリスト3</a>';

上記のサンプルコードでは定数にJavaScriptで作成したHTML要素を代入しただけなので、ブラウザには表示されません。JavaScriptでHTML要素を作成した場合、ブラウザに表示するにはそのHTML要素をdocumentオブジェクトに追加する必要があります。

### **appendChild()メソッド**

HTML要素をdocumentオブジェクトに追加するには、appendChild()メソッドを使います。appendChild()は、作成したHTML要素を子要素として末尾に追加するメソッドです。

3節で学んだHTML要素を取得するためのメソッドを使うことで、以下のように任意のHTML要素の末尾に追加できます。

// 新しくli要素を作成し、定数に代入する

const li = document.createElement('li');

// 作成したli要素にテキストを追加する

li.textContent = 'JavaScriptで新しく作成したリスト3';

li.innerHTML = '<a href="#">JavaScriptで新しく作成したリスト3</a>';

// ul要素の末尾にli要素を追加する

document.querySelector('ul').appendChild(li);

なお、appendChild()メソッドは子要素として末尾に追加するメソッドなので、上記のようにul要素に使った場合、ul要素の子要素であるli要素の末尾に追加されます。

<ul>

<li id="first-list" class="list">リスト1</li>

<li id="second-list" class="list">リスト2</li>

<!-- 以下の箇所にli要素が追加される -->

<li>JavaScriptで新しく作成したリスト3</li>

</ul>

イベント処理

「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

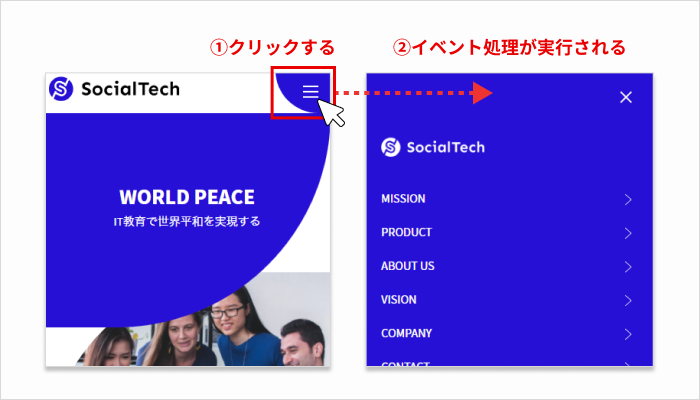
・画面をスクロールする

・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。



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。

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

});

「HTML要素」の部分には、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。

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

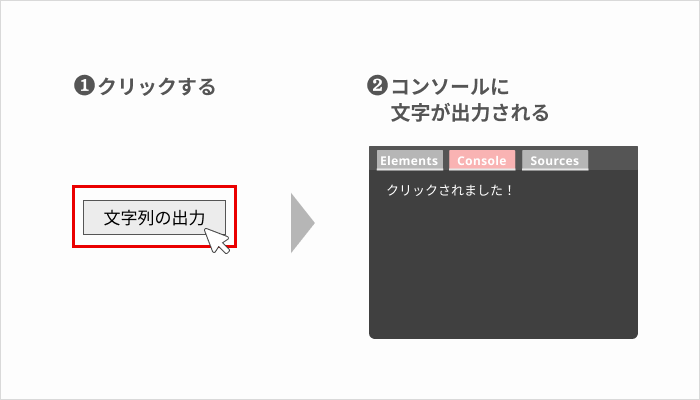
submit フォームを送信したとき

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

簡単なイベント処理を作成する

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てみる



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>

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');

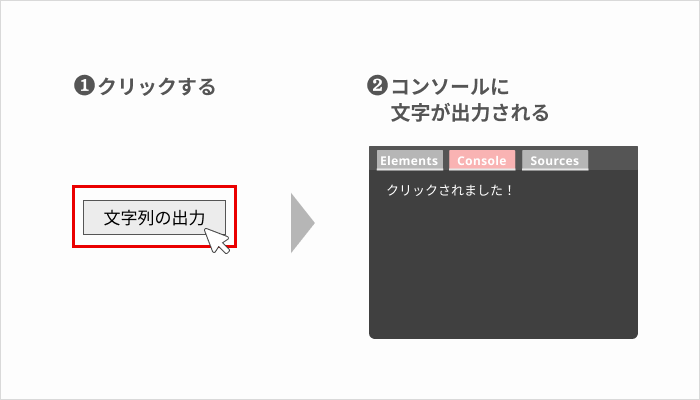
// HTML要素がクリックされたときにイベント処理を実行する

btn.addEventListener('click', () => {

console.log('クリックされました！');

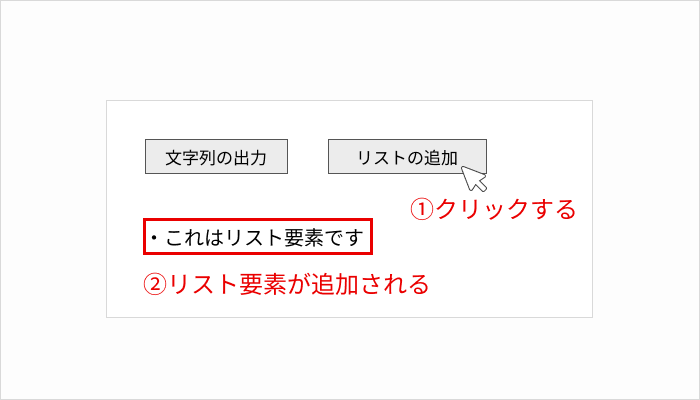
});

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。



クリック時にリストを追加してみよう

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。



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>

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');

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。

li要素を新しく作成する

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

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

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

正解

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

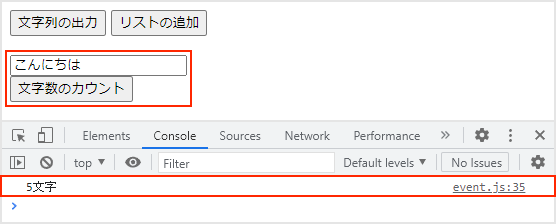
parentList.appendChild(childList);

});

フォームに入力された文字数をカウントしてみよう

1つ目は、フォームに入力された文字数をカウントするイベント処理を作成します。実際のWeb制作で使われる場面としては、「文字数制限を設け、指定した文字数を超えた場合に警告メッセージを表示する」などが考えられます。

今回は、以下のようにシンプルに文字数を出力する処理を作成します。



event.html

<!--======== 前略 ========-->

<button id="add-btn">リストの追加</button>

<ul id="parent-list"></ul>

<form name="textForm">

<input type="text" name="textBox" />

</form>

<button id="count-btn">文字数のカウント</button>

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

</body>

</html>

イベント処理の内容

続いて、JavaScriptで以下のイベント処理を作成します。処理が実行されるタイミングは「文字数のカウント」ボタンをクリックしたときです。

テキストボックスに入力された文字列を取得する

取得した文字列の文字数を出力する

1. テキストボックスに入力された文字列を取得する

1つ目ですが、JavaScriptからテキストボックスの値を取得するには、まずform要素を取得する必要があります。

これまでのようにgetElementById()メソッドなどを使って取得することもできますが、フォームの場合はname属性の値を使うことで簡単に取得できます。

JSファイル（見本）

// textFormというname属性を持つフォームを取得する

document.forms.textForm;

Copy

また、続けてテキストボックスのname属性の値を記述すれば、テキストボックス（input要素）も取得できます。

JSファイル（見本）

// textBoxというname属性を持つテキストボックスを取得する

document.forms.textForm.textBox;

Copy

入力された値を取得するには、続けてvalueプロパティを記述します。

JSファイル（見本）

// textBoxというname属性を持つテキストボックスの値を取得する

document.forms.textForm.textBox.value;

Copy

なお、先ほどevent.htmlを編集した際に、「なぜname属性の値がキャメルケースなのか」と疑問を抱いた方もいると思いますが、その理由はここで使うためです（キャメルケースについては4章を参照）。もしidのようにハイフン-を使ってしまうと、JavaScriptでエラーが発生してしまいます。

JSファイル（見本）

// ハイフンを使うとエラーが発生する

document.forms.text-form;

Copy

2. 取得した文字列の文字数を出力する

取得した文字列の文字数を出力するには、lengthプロパティを使います。復習になりますが、lengthプロパティは文字列や配列の長さを取得するプロパティです。

JSファイルを編集する

ではevent.jsを以下のように編集してください。

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「リストが追加されました」というテキストを追加する

childList.textContent = 'リストが追加されました';

// 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);

});

// count-btnというidを持つHTML要素を取得し、定数に代入する

const countBtn = document.getElementById('count-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

countBtn.addEventListener('click', () => {

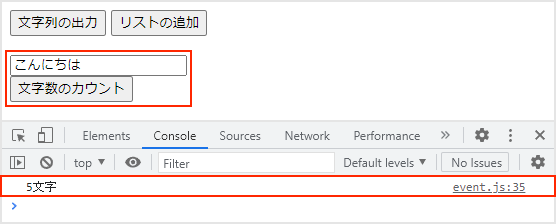
// テキストボックスに入力された文字列を取得する

const text = document.forms.textForm.textBox.value;

// 取得した文字列の文字数を出力する

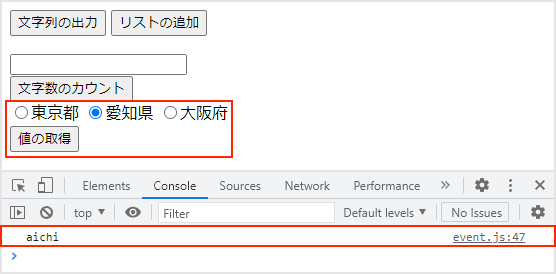
console.log(text.length + '文字');

});



選択されたラジオボタンの値を取得してみよう

2つ目は、選択されたラジオボタンの値を取得するイベント処理を作成します。



<!--======== 前略 ========-->

<form name="textForm">

<input type="text" name="textBox" />

</form>

<button id="count-btn">文字数のカウント</button>

<form name="areaForm">

<input type="radio" name="area" value="tokyo" />東京都

<input type="radio" name="area" value="aichi" />愛知県

<input type="radio" name="area" value="osaka" />大阪府

</form>

<button id="area-btn">値の取得</button>

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

</body>

</html>

イベント処理の内容

続いて、JavaScriptで以下のイベント処理を作成します。処理が実行されるタイミングは「値の取得」ボタンをクリックしたときです。

選択されたラジオボタンの値を取得する

取得した値を出力する

1. 選択されたラジオボタンの値を取得する

先ほどテキストボックスの値を取得したときと同様に、ラジオボタンの値も取得できます。

JSファイル（見本）

document.forms.form要素のname属性の値.input要素のname属性の値.value;

Copy

2. 取得した値を出力する

取得した値を出力するにはconsole.log()を記述するだけですが、取得した値は一度変数や定数に代入しておくことで、コードがスッキリして見やすくなります。

JSファイル（見本）

// 1. 変数や定数に代入しないパターン

// 引数が長いので見づらい

console.log(document.forms.form要素のname属性の値.input要素のname属性の値.value);

// 2. 変数や定数に代入しておくパターン

const 定数名 = document.forms.form要素のname属性の値.input要素のname属性の値.value;

// 引数が短くスッキリしていて見やすい

console.log(定数名);

Copy

JSファイルを編集する

上記を参考に、まずは自分でevent.jsを編集してみてください。編集したら、以下のコードと見比べて答え合わせをしましょう。

event.js

//======== 前略 ========

// count-btnというidを持つHTML要素を取得し、定数に代入する

const countBtn = document.getElementById('count-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

countBtn.addEventListener('click', () => {

// テキストボックスに入力された文字列を取得する

const text = document.forms.textForm.textBox.value;

// 取得した文字列の文字数を出力する

console.log(text.length + '文字');

});

// area-btnというidを持つHTML要素を取得し、定数に代入する

const areaBtn = document.getElementById('area-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

areaBtn.addEventListener('click', () => {

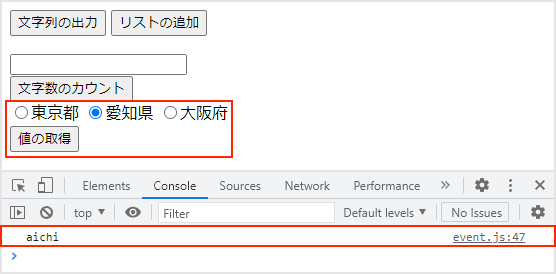
// 選択されたラジオボタンの値を取得する

const area = document.forms.areaForm.area.value;

// 取得した値を出力する

console.log(area);

});



選択されたチェックボックスの値を取得してみよう

最後に、選択されたチェックボックスの値を取得するイベント処理を作成します。



event.html

<!--======== 前略 ========-->

<form name="areaForm">

<input type="radio" name="area" value="tokyo" />東京都

<input type="radio" name="area" value="aichi" />愛知県

<input type="radio" name="area" value="osaka" />大阪府

</form>

<button id="area-btn">値の取得</button>

<form name="osForm">

<label>利用したことのあるOSを選択してください</label><br />

<input type="checkbox" name="os" value="windows" />Windows

<input type="checkbox" name="os" value="macos" />macOS

<input type="checkbox" name="os" value="linux" />Linux

<input type="checkbox" name="os" value="ios" />iOS

<input type="checkbox" name="os" value="android" />Android

</form>

<button id="os-btn">値の取得</button>

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

</body>

</html>

イベント処理の内容

続いて、JavaScriptで以下のイベント処理を作成します。処理が実行されるタイミングは「値の取得」ボタンをクリックしたときです。

すべてのチェックボックスを配列風のデータで取得する

繰り返し処理でチェックボックスを1つずつ取り出し、もし選択されていれば値を出力する

1. すべてのチェックボックスを配列風のデータで取得する

まずはすべてのチェックボックスを配列風のデータで取得します。値ではなく、チェックボックスを形作るinput要素そのものを取得します。

JSファイル（見本）

document.forms.form要素のname属性の値.input要素のname属性の値;

Copy

2. 繰り返し処理でチェックボックスを1つずつ取り出し、もし選択されていれば値を出力する

チェックボックスがラジオボタンと異なるのは、「複数の項目を選択できる」という点です。

よって、選択されたすべての値を出力するために、for文とif文を組み合わせて使います。以下のような流れです。

for文を使い、取得したチェックボックス（input要素）を1つずつ取り出す

for文の中でif文を使い、各チェックボックスがもし選択されていれば値を出力する

JSファイル（見本）

for (カウンタ変数の初期値; 条件式; カウンタ変数の増減値) {

カウンタ変数が決まった値になるまで繰り返す処理

}

Copy

JSファイル（見本）

if (条件式) {

条件が成り立つときの処理

}

Copy

チェックボックスの場合、ユーザーが選択する（チェックを入れる）たびにcheckedプロパティがtrueになる仕組みになっています。

この仕組みをif文の条件式に利用し、チェックボックス.checkedと記述すれば、チェックボックスが選択されていた場合にのみ処理を行うことができます。

なお、for文の条件式にはlengthプロパティを使い、チェックボックスの数だけ繰り返し処理を行います。

JSファイルを編集する

今回はfor文やif文を組み合わせて使うので、難易度が高いです。よって、難しいと感じたら以下のコードを見ながら編集してOKです。可能であれば、まずは自分でevent.jsを編集してみましょう。

event.js

//======== 前略 ========

// area-btnというidを持つHTML要素を取得し、定数に代入する

const areaBtn = document.getElementById('area-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

areaBtn.addEventListener('click', () => {

// 選択されたラジオボタンの値を取得する

const area = document.forms.areaForm.area.value;

// 取得した値を出力する

console.log(area);

});

// os-btnというidを持つHTML要素を取得し、定数に代入する

const osBtn = document.getElementById('os-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

osBtn.addEventListener('click', () => {

// すべてのチェックボックスを配列風のデータで取得する

const items = document.forms.osForm.os;

// 繰り返し処理でチェックボックスを1つずつ取り出し、もし選択されていれば値を出力する

for (let i = 0; i < items.length; i++) {

if (items[i].checked) {

console.log(items[i].value);

}

}

});

チェックボックスを複数選択してみましょう。以下のように、選択したチェックボックスの値がコンソールに出力されていればOKです。なお、チェックボックスを選択しなかった場合はボタンを押しても何も起こりません。

