JavaScript基礎実習

04. HTML要素の操作

株式会社ジードライブ

今回学ぶこと

- WebブラウザとJavaScript
- HTML要素の操作
  - 要素の選択
  - 属性操作
  - 要素内テキストの操作
  - 幅や高さの取得

次章でこれらの処理をクリック等の
ユーザの操作やタイマーと関連付ける

JavaScriptの実行環境

- JavaScriptは、おもにWebブラウザ上で動作するプログラムを作成するための言語として発展してきた
  - ブラウザに搭載されているJavaScriptエンジンがJavaScriptを解析・実行する
  - ここまでは、Node.jsでJavaScriptを実行してきたが、以降はブラウザ上でJavaScriptを動作させる

JavaScriptにできること

- ブラウザ上で動作するJavaScriptでは、以下のようなことを行うことができる
  - イベント処理/タイマー処理
  - HTML操作
  - フォーム制御
  - Ajaxによる通信　…など

イベントとは

- マウス操作、キーボード操作、フォーム操作などの様々なタイミングで発生する「出来事」
  - マウスのクリック
  - キーの入力
  - フォームの送信、etc...
- JavaScriptではそのようなイベントの発生時にコードを実行する仕組みが用意されている
  - イベント発生時に実行する処理をイベントハンドラという

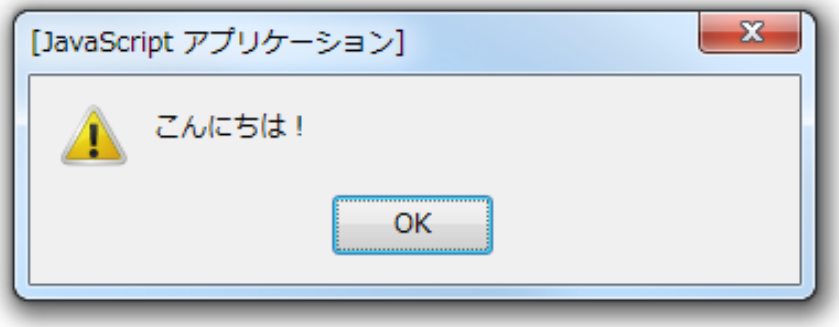
JavaScriptの記述方法

- JavaScriptの記述場所はCSSと同様、
  (1) タグの属性として記述
  (2) HTML内に記述
  (3) 別ファイルに記述して読み込む
  の３パターンがある
- ユーザによってはJavaScriptをオフにしていることがあるので配慮する

タグの属性として記述

- イベントハンドラ属性を利用する

```
<body>
<button onclick="alert('こんにちは！')">
ボタン
</button>
</body>
```



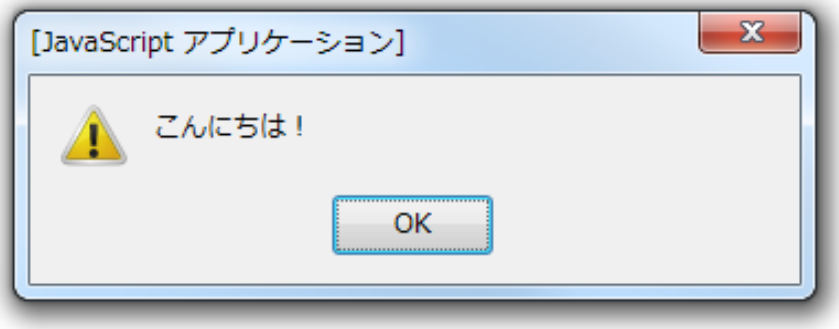
イベントハンドラ属性の例

イベントハンドラ属性	イベント発生タイミング
onclick	マウスをクリックした
onmouseover	マウスカーソルが要素の上に移動した
onmouseout	マウスカーソルが要素から外れた
onchange	フォーム部品の値が変わった
onsubmit	フォームの送信ボタンが押された
onkeydown	要素の上をキーで押した
onload	文書の読み込み完了
onscroll	スクロールバーでスクロールした時

HTML内に記述

- HTML内に<script>タグを使って埋め込む

```
<body>
<script>
alert("こんにちは！");
</script>
</body>
```



scriptタグは、head内/body内のどちらにでも記述できる

外部ファイルに記述

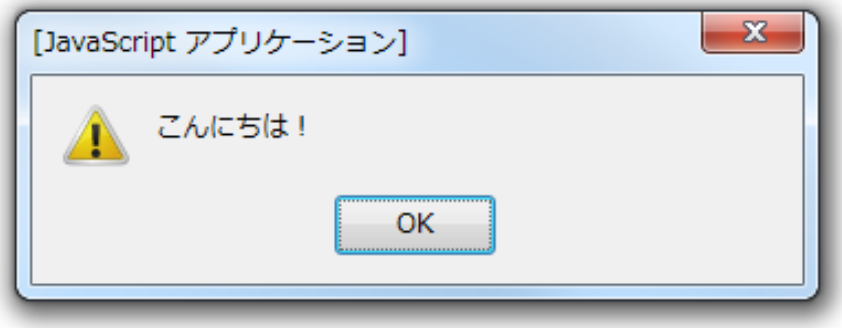
- scriptタグのsrc属性を利用して、別ファイルとして作成したJavaScriptを読み込む

index.html

```
<body>
<script src="message.js"></script>
</body>
```

message.js

```
alert("こんにちは");
```



JavaScriptファイルの拡張子は **.js**

scriptタグの記述位置

- scriptタグはbodyの終了タグ直前に記述する、または defer属性を付与する
  - HTMLの読み込み(DOMの構築)を妨げないため

```
<html>
<head>
   <title>JSの読み込み</title>
   <script src="script1.js" defer></script>
</head>
<body>
   <h1>JavaScriptの読み込み</h1>
   …
   <script src="script2.js"></script>
</body>
</html>
```

scriptタグと変数のスコープ

- scriptタグが分かれていても、ブロック{ }が分かれていなければ、変数は有効なので注意する必要がある

```
<script>
let price = 100;
</script>

<script>
console.log(price);
</script>

<script src="myscript.js"></script>
```

- 変数priceを参照できてしまう
- 新たに変数priceを定義しようとするとエラーになる

Windowオブジェクト

- ブラウザ上で実行されるJavaScriptでは、Windowオブジェクトを通じて、様々な操作を行うことができる
  - HTMLの書き換え
  - ページ遷移
  - タイマー処理
- 利用時は「window」で利用することができる
  - window は記述を省略することができる

```
window.alert('Hello');
window.document.getElementById('logo');
```

Windowオブジェクト

- プロパティ／メソッドの例

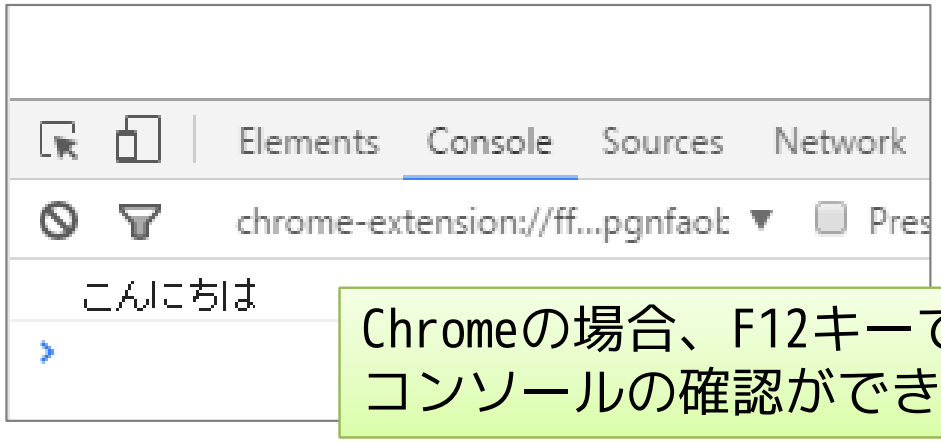
プロパティ名	説明
document	HTMLの情報をもつオブジェクト
outerWidth, outerHeight	ブラウザウィンドウの幅や高さ
innerWidth, innerHeight	HTMLを表示している領域の幅や高さ
scrollX, scrollY	現在のウィンドウのスクロール量

メソッド名	説明
alert()	警告ウィンドウの表示
prompt()	ユーザーに入力を求めるウィンドウの表示
confirm()	確認を求めるウィンドウを表示
setTimeout()	一定時間後に処理を実行する
setInterval()	一定時間ごとに処理を実行する

コンソール画面への出力

- window.console.log()を使うと、ブラウザのコンソール画面にメッセージや計算の結果などの情報を表示させることができる
  - 引数はカンマ区切りで列挙可能
  - window.console.table()も使用可能

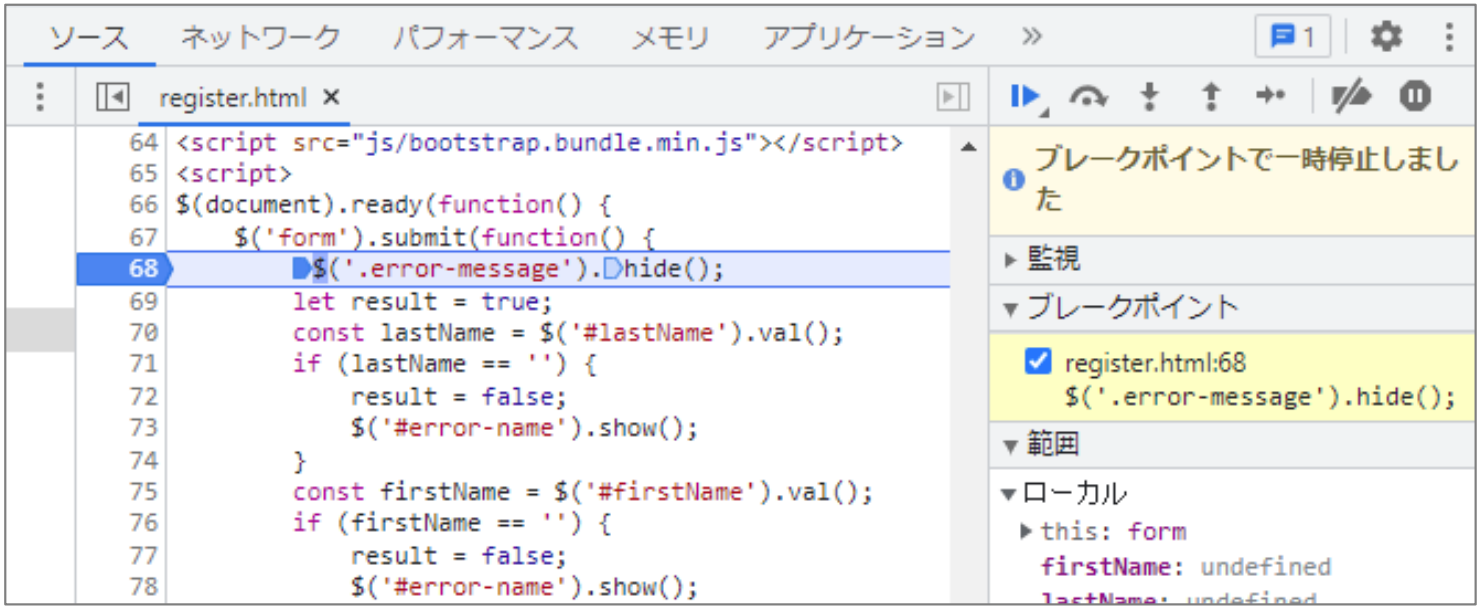
```
window.console.log('こんにちは');  // window.は省略可
```



Chromeの場合、F12キーで開発用ツールを起動し、コンソールの確認ができる

ブラウザによるJSのデバッグ

- ブラウザの開発用ツールには、デバッグ機能が備わっている
  - 使い方参考：https://ics.media/entry/190517/



JavaScriptが無効なブラウザへの対応

- noscript要素に記述した内容はJavaScriptが無効なブラウザでのみ表示される
  - ユーザーによってはJavaScriptをOFFにしている場合があるので配慮が必要

```
<body>
<script>
alert("こんにちは");
</script>
<noscript>
<h1>JavaScriptを有効にしてください。</h1>
</noscript>
</body>
```

JavaScriptが有効になっていない場合に表示されるHTML

JavaScriptを無効にする

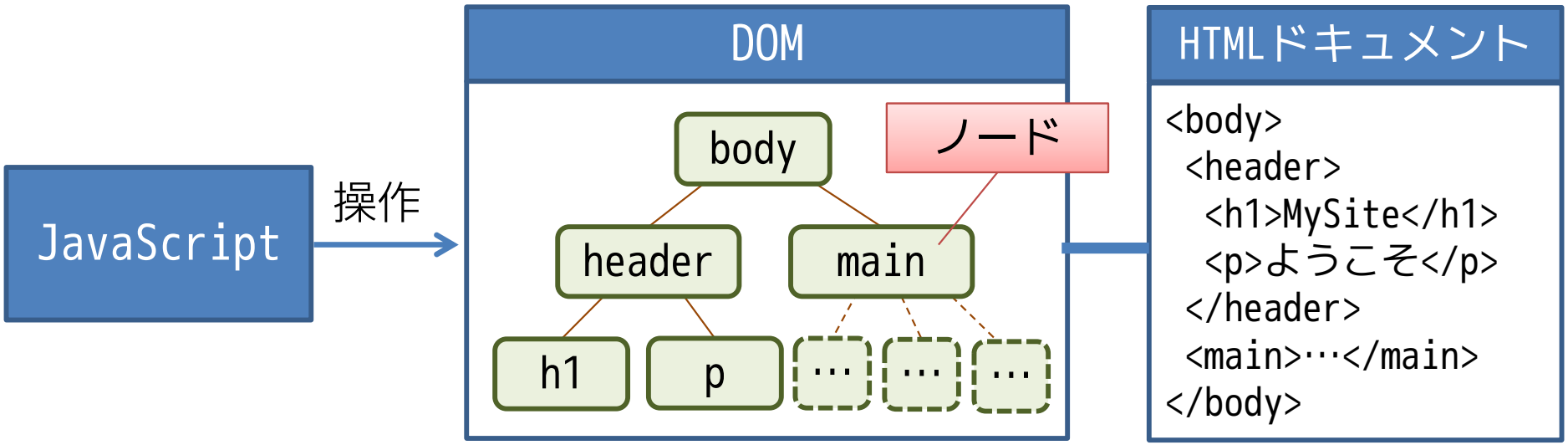
- Chromeの場合、以下のURLでJavaScriptの設定ページを開くことができる
  - chrome://settings/content/javascript

練習

- 練習04-1
- 練習04-2

DOM

- DOM(Document Object Model)はHTMLドキュメントを操作するためのインターフェース
- DOMはツリー状の構造をもっている
  - ツリー構造の一つ一つの節はノードと呼ばれる



HTML要素の操作

- HTML(DOM)を操作する場合は、①対象となる要素を選択し、②その対象がもっているプロパティ値を変更したり、メソッドを使用する

```
// ①操作対象となる要素を選択（idがlogoの要素を選択）
const logo = document.getElementById('logo');

// ②対象となる要素のプロパティを修正（style属性に文字色を設定）
logo.style.color = 'red';
```

要素選択のメソッド

- HTMLの要素選択には、documentのもつ以下のメソッドを利用する

メソッド	説明
getElementById('id名')	HTMLのid属性を元に要素を選択する。 戻り値は、HTMLElementオブジェクト
querySelector('セレクタ')	CSSセレクタに基づき、要素を選択する。 複数の要素が該当する場合は、最初のものが選択される。 戻り値は、Elementオブジェクト
querySelectorAll('セレクタ')	CSSセレクタに基づき、複数の要素を選択する。戻り値は、NodeListオブジェクト ⇒ Nodeオブジェクトの集合

これらメソッドの戻り値であるNode, Element, HTMLElementには継承関係がある(同じように扱うことができる)

NodeList

- NodeListは配列と同じように扱うことができる

```
const pList = document.querySelectorAll('p');

// インデックス番号を利用して最初のp要素を操作
pList[0].innerText = 'Hello';

// for文で全てのp要素を操作
for(let p of pList) {
  p.style.backgroundColor = '#000';
}

// forEachメソッドを使い、全てのp要素を操作
pList.forEach(p => p.style.color = '#fff');
```

練習

- 練習04-3

属性値の操作

- 基本的には、各HTML要素の属性に対応するプロパティが用意されている

```
<img src="images/cat.jpg" alt="猫">

<script>
// 犬の画像に変更し、幅や高さを設定する
const element = document.querySelector('img');
element.src = 'images/dog.jpg';
element.alt = '犬';
element.width = 100;
element.height = 200;
</script>
```

ブラウザの開発ツールで確認できる

```
▼<body>
  <img src="images/dog.jpg" alt="犬" width="100" height="200"> == $0
  ▶<script>…</script>
```

属性値の設定

- setAttribute()メソッドで属性値を設定することも可能

書式

setAttribute(属性名, 値**)**

```
<img src="images/cat.jpg" alt="猫">

<script>
// 犬の画像に変更し、幅や高さを設定する
const element = document.querySelector('img');
element.setAttribute('src', 'images/dog.jpg');
element.setAttribute('alt', '犬');
element.setAttribute('width', 100);
element.setAttribute('height', 200);
</script>
```

属性値の取得

- getAttribute()メソッドで、属性値を取得できる

書式

getAttribute(属性名**)**

```
<img src="images/cat.jpg" alt="猫">

<script>
const element = document.querySelector('img');
console.log(element.getAttribute('alt')); // 猫
console.log(element.getAttribute('width')); // null
</script>
```

width属性は設定していないので、nullになる

style属性の操作

- 要素のstyle属性に値を設定する場合は、以下のように記述する

書式

```
要素.style.CSSプロパティ = '値';
```

```
<h1>hello</h1>

<script>
const element = document.querySelector('h1');
element.style.color = '#00f';
element.style.fontSize = '30px';
</script>
```

font-size のようにハイフンが入るプロパティは、キャメルケースにする

class属性の操作

- class属性については、classListプロパティ(DOMTokenListオブジェクト)のメソッドを使い、操作することができる

書式

```
要素.classList.add(加えたいクラス名)
要素.classList.remove(取り除きたいクラス名)
要素.classList.toggle(加えたい／取り除きたいクラス名)
```

toggle()は、クラスのON/OFFを交互に切り替えることができるメソッド

```
<h1>タイトル</h1>

<script>
const element = document.querySelector('h1');
element.classList.add('title');
</script>
```

練習

- 練習04-4

要素内テキストの取得(1)

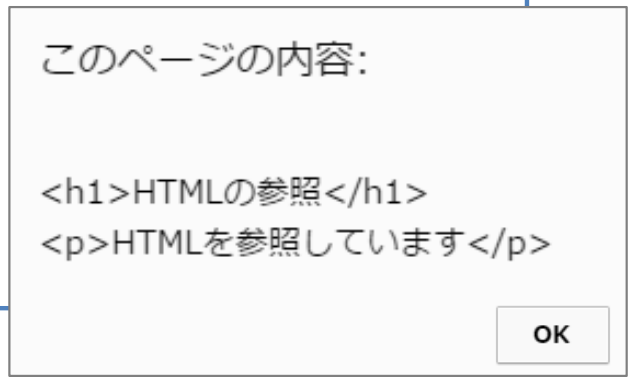
- HTMLを含むテキストを取得する場合には、innerHTMLプロパティを利用する

書式

```
要素.innerHTML
```

```
<div id="group">
  <h1>HTMLの参照</h1>
  <p>HTMLを参照しています</p>
</div>

<script>
const group = document.getElementById('group');
alert(group.innerHTML);
</script>
```



要素内テキストの取得(2)

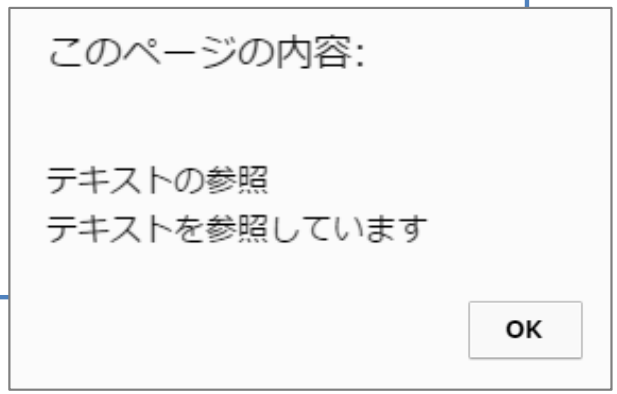
- HTMLタグを含めずにテキストを取得したい場合は、textContentプロパティを利用する

書式

```
要素.textContent
```

```
<div id="group">
  <h1>HTMLの参照</h1>
  <p>HTMLを参照しています</p>
</div>

<script>
const group = document.getElementById('group');
alert(group.textContent);
</script>
```



要素内テキストの変更(1)

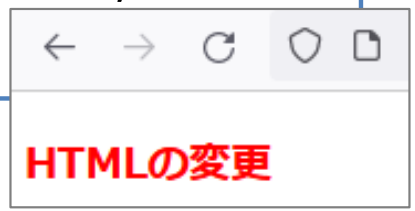
- 要素内のテキストをHTMLを含む文字列で上書きする場合には、innerHTMLプロパティを利用する

書式

```
要素.innerHTML = '文字列';
```

```
<div id="group">
  <h1>HTMLの参照</h1>
  <p>HTMLを参照しています</p>
</div>

<script>
const group = document.getElementById('group');
group.innerHTML = '<h3 style="color: red">HTMLの変更</h3>';
</script>
```



要素内テキストの変更(2)

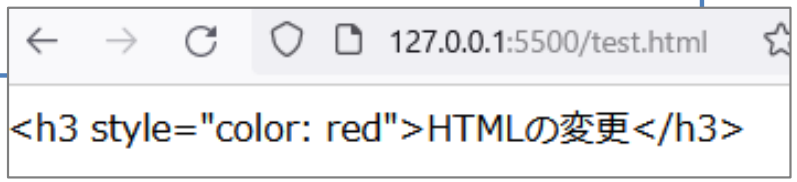
- < >といった記号をエスケープ処理して、要素内テキストを上書きする場合は、textContentプロパティを利用する

書式

```
要素.textContent = '文字列';
```

```
<div id="group">
  <h1>HTMLの参照</h1>
  <p>HTMLを参照しています</p>
</div>

<script>
const group = document.getElementById('group');
group.textContent = '<h3 style="color: red">HTMLの変更</h3>';
</script>
```



練習

- 練習04-5

幅や高さのプロパティ

- clientWidth, clientHeightなどのプロパティを使い、要素の幅や高さを取得することができる

プロパティ	説明
要素.clientWidth 要素.clientHeight	表示領域の幅や高さ(境界線を含まない)
要素.offsetWidth 要素.offsetHeight	表示領域の幅や高さ(境界線やスクロールバーを含む)
要素.scrollWidth 要素.scrollHeight	スクロール分も含めた幅や高さ(境界線を含まない)
window.innerWidth window.innerHeight	ブラウザウィンドウの幅や高さ(ブックマークバーなどを含まないHTML表示領域のサイズ) スクロール分を考慮する場合は、 document.documentElement.scrollWidth, document.documentElement.scrollHeight を使用する

幅や高さのプロパティ

利用例: 要素の高さをブラウザサイズに合わせる

```
<div id="contents">
  Contents
</div>

<script>
const contents = document.getElementById('contents');
contents.style.height = window.innerHeight + 'px';
</script>
```

練習

- 練習04-6