

Web講習会2021 ワールドワイドウェブ基礎

第6回: JavaScript



Arthur

この回の目標

- JavaScriptによって実現できるWebの機能について理解する
- JavaScriptをブラウザで読み込み実行する方法を知る
- DOM APIの使い方を知る
- イベントハンドラによる非同期処理を書けるようになる

JavaScript



JavaScript

JavaScript

ブラウザ上で動く唯一のスクリプト言語
≠ Java

近年はC, Rust, Goなどで生成した
バイナリコードも実行できる
→WebAssembly

言語の特徴

- 手続き型・オブジェクト指向・関数型のパラダイムに対応
- オブジェクト指向はプロトタイプベース
- 非同期処理 (≠ 並列処理)
読み込み中などの処理待ちのときに、他の処理を行えるように
- 動的型付け

Web言語の役割分担

HTML



文書に構造と意味を与える

c.f.) 第2回

CSS



文書に装飾を与える

c.f.) 第3回～第5回

JS

動的に内容を更新など
多機能な処理を行う

JavaScriptでできること

- 計算
- 動的なDOM (HTMLの木構造) の操作
Document Object Model
- ボタンなどのインタフェースに触れたときの処理を記述
- 非同期通信
- CookieやLocal Storageの読み書き
ステートレスなWebに状態を与える
- ファイルの読み込み
- 動画や音声などのメディアの制御
- 動的な画像の描画(Canvas)

Web API

Webをリッチにする様々な機能をJavaScriptから簡単に呼び出せる
→ **API** (Application Programming Interface)

ブラウザAPI

ブラウザに組み込まれている

- DOM API
HTMLやCSSを動的に変更
- Canvas API
2Dグラフィック
- Web Storage API
端末に情報を保存

サードパーティAPI

自分でソースコードを組み込む必要がある

- Twitter API
Twitterのタイムラインを表示
- Google Map API
地図の埋め込み、ピンを立てる

<https://developer.mozilla.org/ja/docs/Web/API>

ブラウザ上での実行

スクリプトの埋め込み

HTML内に<script>タグを使って書く

実はhead内でもbody内でも動くが、bodyの最後尾に書くことを推奨

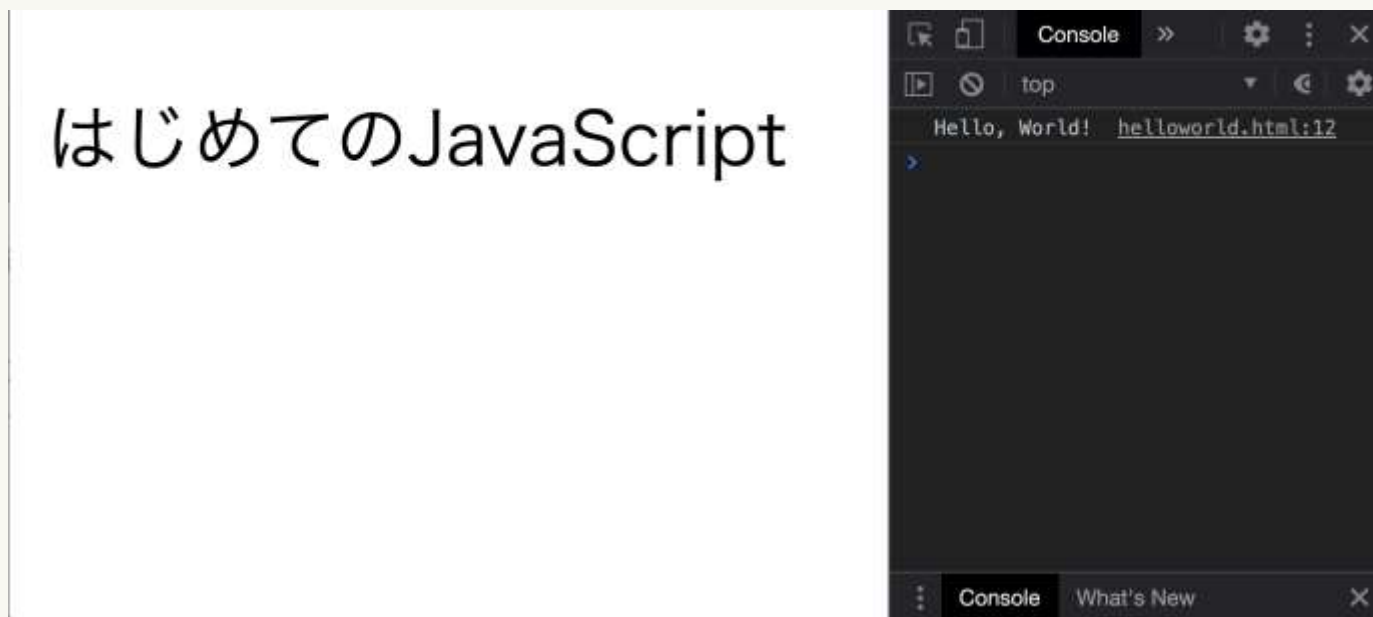
c.f.) https://qiita.com/impl_kawa/items/45dbd431a1f90bd93de6

```
<body>  
  <p>はじめてのJavaScript</p>  
  <script>  
    console.log('Hello, World!');  
  </script>  
</body>
```

はじめてのJavaScript

実行ログの確認

JavaScriptの実行結果はブラウザの描画で確認できない
→開発者ツールの「Console」をチェック



また、ConsoleはREPL (Read-Eval-Print Loop, 対話的実行環境)になっている

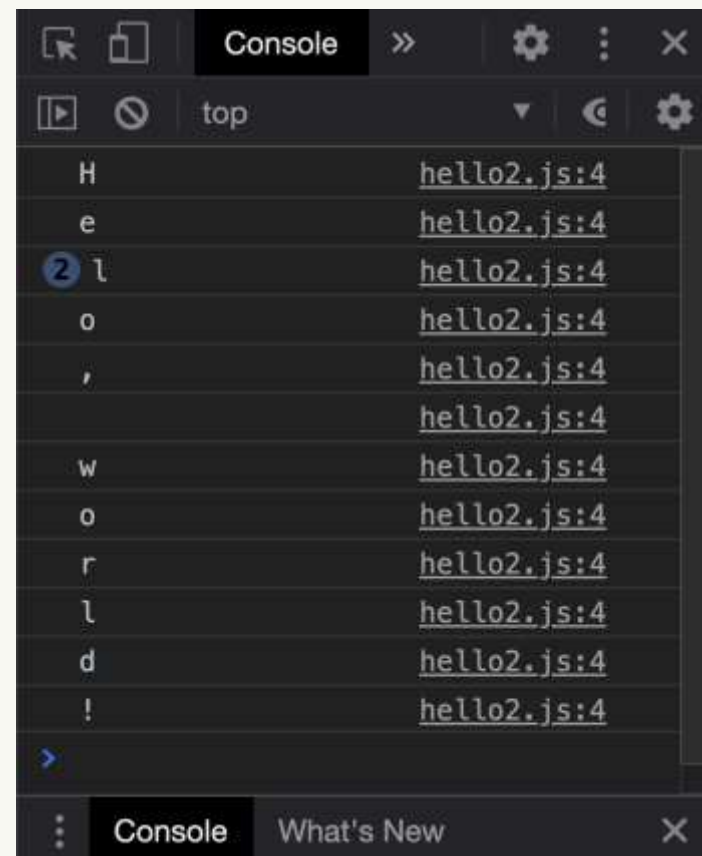
スクリプトの外部ファイル化

CSS同様に、JavaScriptも外部ファイルとして読み込み可能

<script>のsrc属性にパスを指定

```
<body>
  <p>2回目のJavaScript</p>
  <script src="hello2.js"></script>
</body>
```

```
var message = 'Hello, world!';
var length = message.length;
for (var i = 0; i < length; i++) {
  console.log(message[i]);
}
```



JavaScriptの歴史とバージョン

JavaScriptの誕生

1995年

LiveScriptの登場→JavaScriptに名称変更

Netscape Navigatorというブラウザに実装される

当時はJavaが大人気で、ブラウザ上で動くJavaアプレットもこの時期に登場

1996年

Internet Explorerに搭載され、急速に普及

以後、多くのブラウザにも実装される

各ブラウザが言語を独自拡張→**互換性の問題**

JavaScriptの標準化

1997年

ECMAScript (ES)

標準規格として定められたJavaScript

各ブラウザに実装されているJavaScriptはECMAScriptのスーパーセット

2009年

ES5

現存するほぼすべてのブラウザで動作するバージョン

※今回のスライドでES5をベースに説明

最近のJavaScript

ES2015 (ES6)

モダンな記法がたくさん追加された
基本どのブラウザでも動作するが、Internet Explorerはダメ
以降、毎年6月にESの新バージョンがリリースされる

ES2016

...

ES2020

古いブラウザでも最新のES構文を使いたいときは
ポリフィルやトランパイルツールを利用する

ポリフィル…まだサポートされていない機能を互換実装
トランスパイル…あるコードから別の言語のコードを生成する

JavaScriptの文法と特徴



以降の説明で前提とする知識

- 変数の宣言
- 基本的な演算子
 - 四則演算、否定、比較
- 関数の定義と呼び出し
- 配列、オブジェクト

おすすめの教材

体系的に学びたいなら、いつものMDNに加えて、以下のオープンソースの教材がおすすめ。

JavaScript Primer

<https://jsprimer.net/>



手続き型の要素

手続き型・オブジェクト指向・関数型のパラダイムに対応
非同期処理 (≠ 並列処理)
動的型付け

式 (expression)

値を評価でき、変数に代入できるもの

ex) 1 1 + 2 'Hello!' a = 1 add(1, 2)

文 (statement)

手続きの1ステップを記述

式の最後に;をつけると式文(expression statement)になる

ex) 1; add(1, 2); a = 1; if (hoge) { poyo(); poyo(); }

複数の文を{ }で囲み、1つの文として扱える
(compound statement)

※JavaScriptでは、構文解析上不都合がない場合に式文の;を省略できる

OOPの要素

手続き型・オブジェクト指向・関数型のパラダイムに対応
非同期処理 (≠ 並列処理)
動的型付け

JavaScriptではほとんどのものが**オブジェクト**
整数(Number)、文字列(String)、関数(Function)...

オブジェクトは**プロパティ**を持つ
HashMap、Dictionaryのような構造
object.propertyNameのようにアクセス

メソッドはプロパティの特殊な呼び方だけ→

```
> var hello = 'Hello'
< undefined
> hello.length
< 5
> hello.charAt(4)
< "o"
> hello.charAt
< f charAt() { [native code] }
```

関数型の要素

手続き型・オブジェクト指向・関数型のパラダイムに対応
非同期処理 (≠ 並列処理)
動的型付け

高階関数で副作用を減らした見通しの良いコードが書ける
関数オブジェクトを引数に渡す

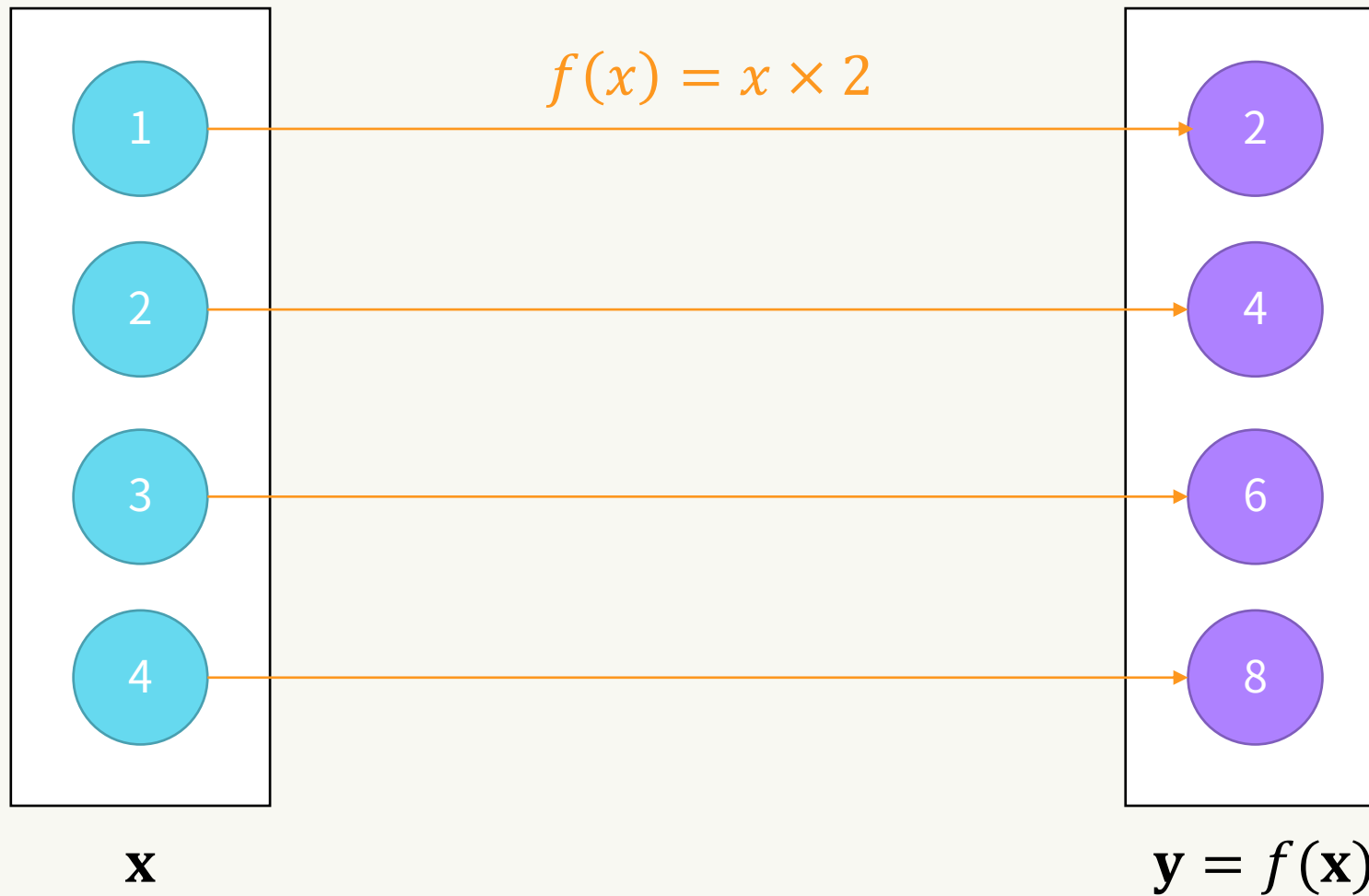
配列は以下の3つの高階関数を持つ

map: 各要素に関数を適用して新たな配列を生成

filter: 各要素に関数を適用して、値がtrueのものだけ残した配列を生成

reduce: 各要素に関数を適用して、1つの値に集約する (合計など)

map()のイメージ



手続き型との比較

手続き型・オブジェクト指向・関数型のパラダイムに対応
非同期処理 (≠ 並列処理)
動的型付け

手続き型の記述

```
var words = [  
  'Now', 'I', 'need', 'a', 'drink',  
  'alcoholic', 'of', 'course',  
  'after', 'the', 'heavy', 'lectures',  
  'involving', 'quantum', 'mechanics',  
];  
  
var count = 0;  
for (var i = 0; i < words.length; i++) {  
  count += words[i].length;  
}  
  
console.log(count); // 77
```

※ES2015以降では、関数をもっと手軽に定義できる 関数型を取り入れた記述

```
var words = [  
  'Now', 'I', 'need', 'a', 'drink',  
  'alcoholic', 'of', 'course',  
  'after', 'the', 'heavy', 'lectures',  
  'involving', 'quantum', 'mechanics',  
];  
  
var count = words.map(function(word) {  
  return word.length;  
}).reduce(function(acc, cur) {  
  return acc + cur;  
});  
  
console.log(count); // 77
```

非同期処理

手続き型・オブジェクト指向・関数型のパラダイムに対応
非同期処理 (≠ 並列処理)
動的型付け

JavaScriptはブラウザのUIスレッドで実行される
無限ループなど、処理をブロックする手続きがあるとフリーズ

非同期処理 … その処理が終わるのを待たずに次の処理を行う
非同期処理が終わった後に実行する関数: コールバック

非同期処理は並行ではあるが並列ではない
1つのスレッド内で、一定のクロックで処理を切り替えて並行実行する

非同期処理の例

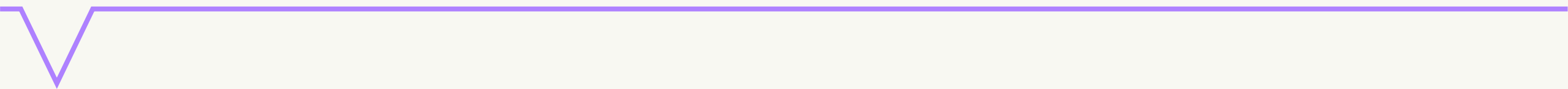
手続き型・オブジェクト指向・関数型のパラダイムに対応
非同期処理 (≠ 並列処理)
動的型付け

`setTimeout(callback, delay)`
delay [ms]後にcallbackを実行する

```
console.log('A');  
  
setTimeout(function() {  
  console.log('B');  
}, 1000);  
  
console.log('C');
```

A	<u>setTimeout.js:1</u>
C	<u>setTimeout.js:7</u>
B	<u>setTimeout.js:4</u>

DOM APIとイベントハンドラ



DOMとは

DOM (Document Object Model)

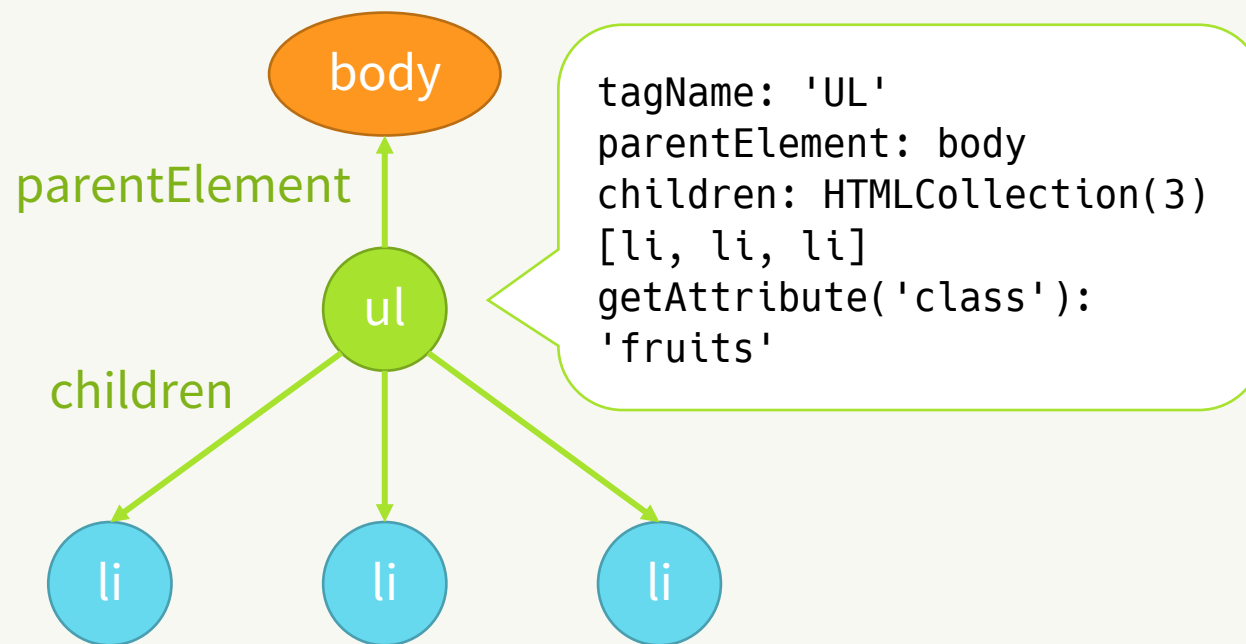
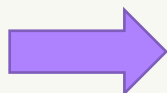
HTML文書をオブジェクト指向で記述できるようにしたインタフェース

子要素への参照を持つデータ構造

HTMLの木構造の各要素をノードとして表現

```
<body>
  <ul class="fruits">
    <li>Apple</li>
    <li>Banana</li>
    <li>Orange</li>
  </ul>
</body>
```

DOM Parser



ブラウザ実行時のグローバル変数

グローバルオブジェクト: Window

グローバルスコープで常時存在するオブジェクト、windowでアクセス

windowの中身

- 自分で定義したグローバル変数
グローバルスコープにおいてhogeとwindow.hogeは等価
- DOMのParser
- ブラウザのタブに関する様々な情報やメソッド
location (URL)、inner{Height,Width} (ウィンドウサイズ) etc...
- 各種APIへのインタフェースとなるオブジェクト
console、document、localStorage etc...

イベントハンドラ

HTMLファイル読み込みからDOM生成には時間差がある
JavaScriptの実行タイミングは制御できない

ボタンを押したときの処理はどこに書くのか

→ イベントハンドラによる非同期処理を利用

```
[対象のDOM オブジェクト].addEventListener([イベント名], function() {  
    // イベントが発火したときの処理  
})
```

DOMContentLoadedイベント

Documentオブジェクト

ブラウザに読み込まれたWebページを表すオブジェクト

window.document (あるいはdocument)でアクセス

documentはDOMContentLoadedというイベントを持つ
DOMツリーの生成が終わったら発火

```
document.addEventListener('DOMContentLoaded', function() {  
  console.log('DOM Generated');  
  // DOMの操作に関する手続き  
});  
console.log('JavaScript Loaded');
```

JavaScript Loaded	event_handler.html:16
DOM Generated	event_handler.html:13

ノードの取得

document.getElementById([idName])

id名から対象要素のノードを取得する

```
<h1 id="title">Fruits</h1>
<ul class="fruits">
  <li>Apple</li>
  <li>Banana</li>
</ul>
```

根ノードから木を辿っていくのは大変なので、
検索するメソッドが用意されている

```
document.addEventListener('DOMContentLoaded', function() {
  var title = document.getElementById('title');
  console.log(title.tagName); // 'H1'
});
```

ノードのリストを取得

document.getElementsByClassName([className])
class名から対象要素のノードのリストを取得する

```
<h1 id="title">Fruits</h1>
<ul class="fruits">
  <li>Apple</li>
  <li>Banana</li>
</ul>
```

```
document.addEventListener('DOMContentLoaded', function() {
  var fruits = document.getElementsByClassName('fruits');
  console.log(fruits[0].tagName); // 'UL'
});
```


ノード(のリスト)を取得できる他の関数

document.`getElementsByTagName`([tagName])

タグ名が一致する要素のノードのリスト

document.`querySelector`([query])

セレクタが一致する最初の要素のノード

document.`querySelectorAll`([query])

セレクタが一致するすべての要素のノードのリスト

ここまでで紹介したもののうち、`getElementById`以外は、`document`以外のノードオブジェクトからも呼び出し可能

ex) `document.querySelector('ul').getElementsByTagName('li')`

clickイベント

各ノードはclickイベントを持つ
要素がクリックされたときに発火

イベントハンドラはEventオブジェクトを引数として受け取る
マウスでクリックしたならクリックした位置など
クリック対象の要素も取れる

```
document.addEventListener('DOMContentLoaded', function() {  
  var button = document.getElementById('btn');  
  button.addEventListener('click', function(event) {  
    console.log(event);  
    console.log('クリックされました');  
  });  
});
```

```
DOM.html:22  
MouseEvent {isTrusted: true, scr  
▶ eenX: 758, screenY: 724, client  
  X: 60, clientY: 193, ...}  
クリックされました DOM.html:23
```

ノードの生成と追加

```
<body>
  <input id="form_name" type="text">
  <button id="btn">Add</button>
  <ul id="people"></ul>
  <script>
    document.addEventListener('DOMContentLoaded', function() {
      var button = document.getElementById('btn');
      button.addEventListener('click', function(event) {
        // フォームの入力値を取得
        var formName = document.getElementById('form_name');

        // <li>{ personName }</li>
        var person = document.createElement('li');
        var personName = document.createTextNode(formName.value);
        person.appendChild(personName);

        // <ul>内の子要素として追加
        var people = document.getElementById('people');
        people.appendChild(person)
      });
    });
  </script>
</body>
```

document.createElement([tagName])
指定したタグのノードの生成(まだ表示されない)

[親ノード].appendChild([子ノード])
親ノードの子要素としてノードを追加(これで表示される)



oiyo Add

- aaa
- aaaaaa
- hoge
- oiyo

次回予告

- Webサイトの公開
 - GitHub Pagesへのデプロイ

以下の準備をしてきてください

- gitの基本的な操作を身につける
 - add, commit, pushなど
 - ITSPの動画参照 <https://www.youtube.com/watch?v=WMIIPcgGC4Q>
- (無い人は)GitHubのアカウント作成
 - 学生であれば、GitHub Educationを適用することを推奨