

4.3 JavaScript 基礎

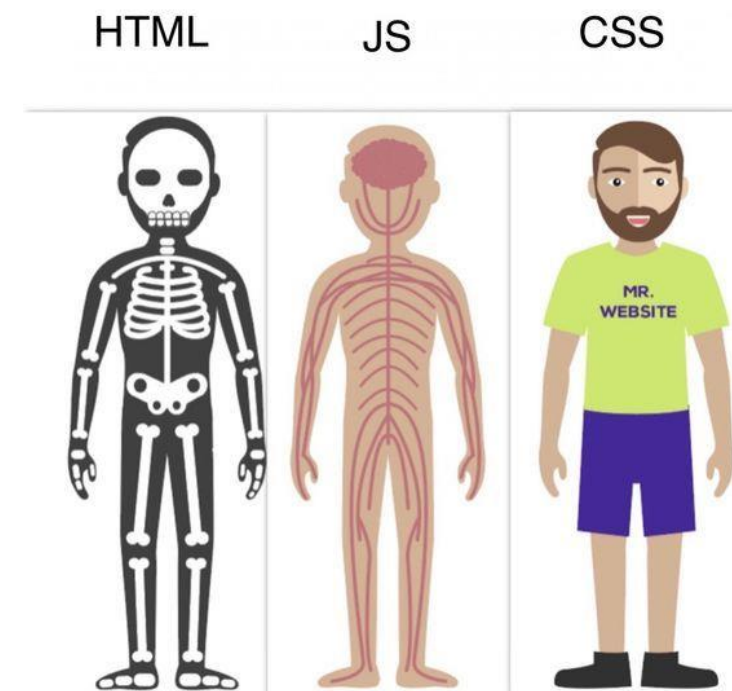
- JavaScript
- JavaScript の基本文法
- DOM
- jQuery

目次

- 1 JavaScript
- 2 JavaScript の基本文法
- 3 DOM
- 4 jQuery

JavaScript

- **JavaScript** はウェブ関連の開発で最も人気のある言語の一つです。主な応用は、ウェブ上の**動作**を制御することです。
- JavaScript は、HTML ページ上の要素と直接やり取りして、ユーザーの操作に動的にフィードバックを与えることができます。



JavaScript の応用

- JavaScript は**スクリプト言語**[Scripting Language]の一種です。Java に比べ基本文法が簡単で、HTML に直接埋め込むことができます。そのため、ウェブページ上の動作など、**小規模なプログラム**の開発に非常に効率的に利用できます。
- しかし、JavaScript は、サーバーやデスクトップアプリケーションなど、大規模なアプリケーション開発にも利用できます。
- 例えば、**Node.js** という人気のあるサーバ開発フレームワークがあります。近年日本のインターネット企業の中でも、サーバ開発に Node.js を利用している会社が多数存在します（合同会社 DMM.com、株式会社ディー・エヌ・エーとか）。

JavaScript と Java

- JavaScript と Java は、名前は似ていますが、**全く異なる言語**です。
- JavaScript の命名は、1990 年代の Netscape 社と Microsoft 社のブラウザ戦争に遡ります。より大きな市場を獲得するため、Sun と提携した Netscape 社は、当時 Java が最もホットな新興言語であったことから、自分のブラウザで使えるスクリプト言語を JavaScript と名付けることにしたのです。
- したがって、違いはあっても、**JavaScript の構文は Java を意識して設計され**、Java プログラマが JavaScript を学ぶのはかなり簡単です。

外部 JavaScript ファイルの読み込み

- CSS と同様に、**外部**の JavaScript ファイルを取り込むか、HTML 文書**内部**に直接 JavaScript コードを埋め込むかのどちらかも可能です。
- 外部コードを取り込むには、HTML の **<script>** タグを使用し、その **src** 属性にコードファイルの URL を設定します：

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

Note

終了タグを書くのを忘れないでください。

- JavaScript のコードファイルには、一般的に「.js」という拡張子が付きます。

JavaScript コードの埋め込み

- JavaScript のコードを埋め込むには、`<script>` タグの内容に直接 JavaScript のコードを記述してください：

```
1 <script>
2   let str = "Hello, world!";
3   console.log(str);
4 </script>
```

- CSS と同様、`<script>` は `<head>` タグ内に記述することが推奨されます。

JavaScript の実行順序

- 同じ文書内に複数外部・内部 JavaScript コードがある場合、それらは読み込まれた・埋め込まれた順で（つまり、上から下の順で）順番に実行されます：

```
1 <script src="1.js"></script>
2 <script>
3   console.log("2");
4 </script>
5 <script src="3.js"></script>
```


外部・内部コードの選択

- 特別な要求がない場合は、**外部コードファイル**を使用することをお勧めします。外部コードファイルを使用するのは次のようなメリットがあります：
 - ページの構造を記述する HTML コードと、動作を記述する JavaScript コードが分離され、読みやすく、拡張しやすいようになります。
 - JavaScript のコードだけを、ブラウザがキャッシュしてくれて、ページの読み込みを高速化することが可能です。

Q&A

目次

- 1 JavaScript
- 2 JavaScript の基本文法
- 3 DOM
- 4 jQuery

Java から JavaScript へ

- JavaScript と Java は異なる言語ではありますが、JavaScript の基本的な文法の多くが、Java の文法と一貫しています。JavaScript を学ぶ際には、Java との文法の違いだけを覚えることから始めるといいでしょう。

JavaScript と Java の主な違い

- JavaScript において、Java と違うの重要な文法は：

文法	区別
変数	var、let または const を使って定義。異なるタイプの値が格納可能
文字列	シングルクォート「'」で定義できる
配列	「[]」または new Array() で定義。異なるタイプのデータが格納可能
オブジェクト	「{}」で定義。通常は連想配列として使う
繰り返し文	for-of、for-in 文がある
メソッド	function で定義。ラムダ式は「=>」で定義

JavaScript の基本的なプログラム

- 次の JavaScript コードを HTML 文書に取り込むか埋め込んで、HTML ページをブラウザで開いてください：

```
console.log("Hello, world!");
```

- ブラウザの**コンソール**を開いてください。コンソールを開くショートカットキーは、ブラウザの種類により異なります：
 - Chrome : Shift + Ctrl + J (Windows) 、 ⌘ + ⌘ + J (macOS) 。
 - Firefox : Shift + Ctrl + J (Windows) 、 ⌘ + ⌘ + J (macOS) 。
 - Safari : ⌘ + ⌘ + C。
 - Edge : Shift + Ctrl + I。
- コードが正しく書かれていれば、コンソールに「Hello, world!」が見えるはずです。

基本文法

- JavaScript の文は基本的に Java と同じように動きます：
 - 上から下へ順番に実行。
 - 各文がセミコロン「;」で区切られる。
 - 単語の区切りやフォーマッティングには、スペースや Tab を使用します。
- 以下の違いにご注意ください：
 - 単語の区切りに**改行を使用しないでください**。
 - インデントの 1 レベルにつき、スペース **2 つ**を使うこと。

標準出力

- JavaScript は様々な方法でデータを出力できます：
 - `document.write()` を使って HTML 文書に書き込みます。
 - `window.alert()` でアラートを表示させます：



- **`console.log()`** を使用し、ブラウザのコンソールに出力します。



変数

- JavaScript で**変数**を宣言するには、**var**、**let**、**const** のキーワードを使用します：

```
var str = "This is a string.";
let num = 120;
const PI = 3.14;
```

- **var** は、初期化しないままにしておくことができる変数を宣言します。変数は自動的に「undefined」に初期化されます。
- **let** は、初期化が必要な変数を宣言します。
- **const** は、定数を宣言します。
- 可読性と効率を高めるために、**可能な限り const と let 宣言を使ってください。**

変数のタイプ

- Java とは異なり、変数の宣言時に型を指定する**必要はありません**。同じ変数で**異なるタイプの値**を保持することも可能です。
- 以下の例では、変数 a は、最初は数字を保持し、後に文字列を保持するために使用されます：

```
1 let a = 42;
2 console.log(a); // => 42
3 a = "This is the answer.";
4 console.log(a); // => This is the answer.
```


変数名

- JavaScript の識別子の命名規則は Java と似ています：
 - 名前に使用できる文字は、**英文字**、**アラビア数字**、アンダースコア「_」、ドル記号「\$」のみです。
 - 名前が**数字で始まることはできません**。
 - JavaScript の**キーワード**にすることはできません。
 - 変数名とメソッド名は**小文字のキャメルケース**を使用します。
- JavaScript のキーワードは Java のキーワードと異なり、以下の資料で確認することができます：

 https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Lexical_grammar#reserved_words

算術演算子

- **算術演算子**は、数値に対する演算を行うために使用されます：

演算子	機能
+	足し算
-	引き算
*	掛け算
**	べき乗
/	小数割り算
%	剰余演算
++	1 だけ増加
--	1 だけ減少

代入演算子

- **代入演算子**は変数の値を変更します：

演算子	機能	等価な表現
=	代入	
+=	増加	$a = a + b$
-=	減少	$a = a - b$
*=	掛け	$a = a * b$
**=	何乗になる	$a = a ** b$
/=	割り	$a = a / b$
%=	剰余になる	$a = a \% b$

比較演算子

- **比較演算子**は、2 つの変数を互いに比較します：

演算子	機能
<code>==</code>	値が等しい
<code>===</code>	値と種類のどちらも等しい
<code>!=</code>	値が等しくない
<code>!==</code>	値と種類のどちらが等しくない
<code>></code>	大なり
<code><</code>	小なり
<code>>=</code>	以上
<code><=</code>	以下

=== と ==

- JavaScript の比較演算子「==」と「===」は、若干異なる動作をします：

- == 値の等しさの比較。
- === 値と種類の等しさの比較。

```
1 let a = "321";
2 let b = 321.0;
3
4 console.log(a == b); // => true
5 console.log(a === b); // => false
```

- == は、Java におけるオブジェクトの equals() メソッドに相当しています。
- Java における equals() メソッドに相当する JavaScript のメソッドが Object.is() メソッドになります。

論理演算子

- **論理演算子**は、ブール値を組み合わせて複雑な論理を表現するために使用されます：

演算子	機能
&&	論理積（AND）
	論理和（OR）
!	論理否定（NOT）

タイプ演算子

- **typeof** 演算子は変数の種類を取得できます：

```
1 let a = "123";
2 console.log(typeof a); // => string
3 console.log(typeof 10); // => number
```

- **instanceof** (スペースがないことに注意)は、オブジェクトが特定の種類であるかどうかを判断するために使用されます：

```
1 let a = new String("123");
2 console.log(a instanceof String); //true
3 console.log(a instanceof Number); //false
```

数字

- JavaScript の変数は、任意の**整数または小数**が格納できます：

```
let num1 = 1.0;  
const num2 = -123;
```

- 指数表現、16 進表現、8 進表現、2 進表現とかも、Java と同様の書き方で表すことができます：

```
1 console.log(1.23E5); // => 123000  
2 console.log(0xCAFE); // => 51966  
3 console.log(036021); // => 15377  
4 console.log(0b1101); // => 13  
5 console.log(123_456_789); // => 123456789
```

ブール値

- JavaScript では、条件判断に**ブール値** **true** または **false** を使用することもできます：

```
1 let bool1 = 1 + 1 == 2;  
2 console.log(bool1); // => true  
3  
4 let bool2 = false;  
5 console.log(bool2); // => false  
6  
7 console.log(bool2 || true) // => true
```

文字列

- JavaScript でも、テキストデータを格納するために文字列を使用します。
- ただし、JavaScript の文字列は、二重引用符「"」または一重引用符「'」の**どちらでも**定義することができます：

```
let str1 = "quoted by double";
let str2 = 'quoted by single';
```

- この文法のメリットの一つは、「'」で文字列を囲むと、「"」を文字列に直接書いて、エスケープする必要がなくなります：

```
let str1 = 'Tom says: "Hello!"';
```

- Java にあるようなキャラクター型「char」は JavaScript には**存在しません**。単独の文字も文字列で表現されます。

配列

- JavaScript の**配列**の使い方は、Java とは大きく異なります。
 1. **作成方法**が異なります。JavaScript の配列は 2 つの作成文法があり、どちらも Java とは異なります。
 2. **保存ルール**が異なります。JavaScript の配列は異なる種類のデータを格納できます。
 3. **メソッド**が異なります。JavaScript の配列には便利なメソッドがいくつも用意されています。

配列の作成

- 初期値が定義された配列を作成するには「`[]`」を使用します：

```
1 let arr = [1, 2, 3];
2 console.log(arr); // => [1, 2, 3]
```

- 定義された長さを持つ配列を作成するには、`new Array()` コンストラクタを使用します：

```
1 let arr = new Array(5);
2 arr[1] = 0;
3 console.log(arr); // => [empty, 0, empty x 3]
```

Try `arrays.html`

データの格納

- Java とは違って、**異なる種類のデータ**を同じ配列に格納できます：

```
1 let arr = ["string", 10, 1.5, [1, 2, 3]];
2 console.log(arr); // => ['string', 10, 1.5, Array(3)]
```

- Java と同様に、**添え字**を介して「**[]**」で配列のデータをアクセスします：

```
1 let arr = ["string", 10, 1.5, [1, 2, 3]];
2
3 console.log(arr[0]); // => string
4 console.log(arr[3][1]); // => 2
5
6 arr[1] = "alpha";
7 console.log(arr[1]); // => alpha
```

配列の属性とメソッド

- Java と同様、**length** 属性は配列の長さを取得します：

```
console.log([1, 2, 3].length); // => 3
```

- **push()** メソッドで、配列の末尾にデータを挿入します（配列は「長くなる」のです）。**pop()** メソッドで、最後のデータを削除します（配列が「短くなる」のです）：

```
1 let arr = ['a', 'b', 'c', 'd'];
2
3 arr.push('e');
4 console.log(arr); // => ['a', 'b', 'c', 'd', 'e']
5 console.log(arr.pop()); // => 'e'
6 console.log(arr); // => ['a', 'b', 'c', 'd']
```

- Java とは異なり、JavaScript の配列は可読性の高い文字列として直接出力できます。

オブジェクト

- Java と似ていますが、JavaScript の**オブジェクト**は、データを格納するための連想配列として使われるのは普通です。
- オブジェクトを作成するには、「**{ }**」符号で、「**,**」で区切られたいくつかの**キー**[key]（プロパティ [Property]ともいう）と**値**[Value]のペアを囲みます：

```
1 let obj = {name: "Alice", id: 148, isStudent: true};
2 console.log(obj); // => {name: 'Alice', id: 148, isStudent: true}
```

- オブジェクトのキーは、文字列のみです。値は、任意のタイプにすることができます。

Try 01011
11010
01011
objects.html

オブジェクトの使用

- Java と同様、「`.`」演算子を使って、オブジェクトの値をキーで直接アクセスできます：

```
1 let obj = {name: "Alice", id: 148, isStudent: true};
2 obj.id = 120;
3 console.log(obj.id); // => 120
```

- また、「`[]`」記号を使って、キーの文字列でアクセスすることも可能です：

```
1 let obj = {name: "Alice", id: 148, isStudent: true};
2 obj["id"] = 120;
3 console.log(obj["id"]); // => 120
```

- JavaScript のオブジェクトは、Java と同様にオブジェクト指向を実現するためのメソッドを保存することも可能ですが、ここではオブジェクト指向の部分を省略します。

Q&A

分岐文

- JavaScript の分岐文の文法は、Java と同じです :
 - if 文、if-else 文、およびそれらのネスティング。
 - switch-case 文とその中の break 文。
 - 三項演算子「?:」。

繰り返し文

- 以下の繰り返し文は、Java と同じです：
 - for 文。
 - while 文。
 - do-while 文。
- しかし、JavaScript には 2 種類の異なる繰り返し文があります：
 1. for-of 文。
 2. for-in 文。

for-of 文

- **for-of** 文は、Java の **for-each** 文と同様に、配列や文字列などの繰り返し可能なオブジェクトのすべての値を繰り返し処理するために使用されます：

```
1 const arr = [2, 3, 5, 7];
2 let sum = 0;
3 for (let x of arr) {
4     sum += x;
5 }
6 console.log(sum); // => 17
```

```
1 const str = "abc";
2 for (let c of str) {
3     console.log(c); // => a b c
4 }
```

Note !

変数名の前に let
を書くのを忘れ
ないこと。

for-in 文

- **for-in** 文は、オブジェクト内の全ての**キーの名前**を繰り返し処理します：

```
1 const obj = {name: "Alice", id: 148, isStudent: true};
2 for (let prop in obj) {
3   console.log(prop); // => name id isStudent
4 }
```

- また、配列の全ての**インデックス名**（0 から length - 1 までの数字）を繰り返し処理することも可能です：

```
1 const arr = [1, 2, 3, 4];
2 for (let i in arr) {
3   console.log(i); // => 0 1 2 3
4 }
```

Try 01011
11010
01011
loops.html

break と continue

- Java と同様に、ループの中で **break** 文や **continue** 文を使うこともできます：
 - break 文はループ全体をスキップします。
 - continue 文は 1 回の繰り返しをスキップします。
- 同様に、**ラベル**を使って、どのレベルのループから飛び出すかを指定します：

```

1 outer: for (let i = 0; i < 10; i++) {
2   for (let j = 0; j < 10; j++) {
3     console.log(i + " " + j); // => 0 0  0 1  0 2  0 3
4     if (j == 3) break outer;
5   }
6 }

```

メソッド

- JavaScript のメソッドは、Java と使い方が似ていますが、定義の文法は違います。
- メソッドを定義するには、**function** を使います：

```
1 function sum(a, b) {
2   return a + b;
3 }
```

Try 01011
11010
01011
functions.html

- Java との相違点は：
 1. **function** キーワードを使用します。
 2. 戻り値のタイプを定義しない。
 3. 引数のタイプを定義しない。

デフォルト引数

- JavaScript のメソッドの引数は、比較的簡単にデフォルト値が設定できます。デフォルト値を設定するには、定義時に引数の名に「=デフォルト値」を追加します：

```
1 function print(str1, str2="World", str3="!") {
2   console.log(str1 + " " + str2 + str3);
3 }
```

- 使用する時、パラメータを渡さなかった場合、デフォルト値が使用されます：

```
1 print("Hello"); // => Hello World!
2 print("Hello", "Bob"); // => Hello Bob!
3 print("Bye", "Alice", "...") // Bye Alice...
```


デフォルトパラメーターに関する注意事項

- 注意：最後のいくつかの引数のみ、デフォルト値に設定できます：

Example ✓

```
1 function method(a, b, c=0)
2 function method(a, b=0, c=0)
3 function method(a=0, b=0, c=0)
```

Example ✕

```
1 function method(a, b=0, c)
2 function method(a=0, b=0, c)
3 function method(a=0, b, c=0)
```

変数のスコープ

- Java と同様、JavaScript の変数の**スコープ**は、その変数がどの中括弧「`{}`」で定義されたのかによって決まります：

```

1 let a = 5;
2
3 function method() {
4     let a = 10;
5     console.log(a); // => 10
6 }
7
8 method();
9 console.log(a); // => 5

```

Tips💡

グローバル変数は、同じページ内の他の JavaScript コードでも使用できます！

- オブジェクト指向の前提を考えずに、単にメソッド内の変数を**ローカル変数**、メソッド外の変数を**グローバル変数**[Global Variable]と呼べばよいのでしよう。

ラムダ式

- JavaScript にも Java と同じようなラムダ式があり、メソッドを引数として渡すことができます。
- Java と異なるのは、「=>」記号を使用します：

```
1 const arr = ["Alice", "Bob", "Dave", "Alexander"];
2 arr.sort((a, b) => {
3   return a.length - b.length;
4 });
5 console.log(arr); // => ["Bob", "Dave", "Alice", "Alexander"]
```

- メソッド名を使って、既存のメソッドを直接渡すこともできます：

```
1 function compare(a, b) {return a.length - b.length;}
2 arr.sort(compare);
```

コメント

- Java と同様、「//」記法で1行のコメント、「/* */」記法で複数行の**コメント**を書くことができます：

```
1 let a = 120 + 240; // a becomes 360
2 /*
3  * This is a multi-line comment.
4  * これは複数行コメントです。
5  */
6 console.log(a) // => 360
```

おさらい : JavaScript と Java の重要な違い

Sum Up

文法	区別
変数	<code>var</code> 、 <code>let</code> または <code>const</code> を使って定義。異なるタイプの値が格納可能
文字列	シングルクォート「 <code>'</code> 」で定義できる
配列	「 <code>[]</code> 」または <code>new Array()</code> で定義。異なるタイプのデータが格納可能
オブジェクト	「 <code>{}</code> 」で定義。通常は連想配列として使う
繰り返し文	<code>for-of</code> 、 <code>for-in</code> 文がある
メソッド	<code>function</code> で定義。ラムダ式は「 <code>=></code> 」で定義

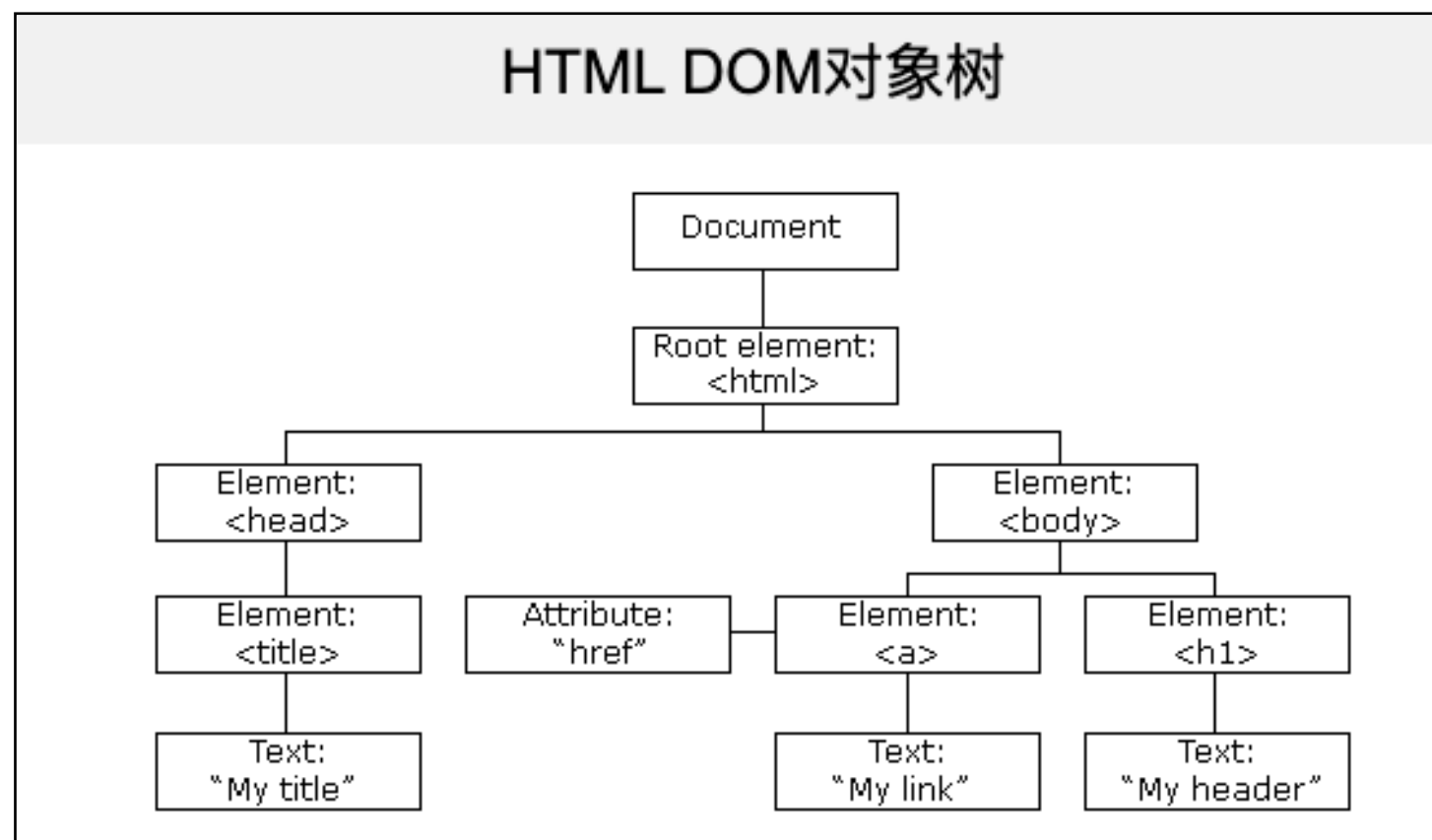
Q&A

目次

- 1 JavaScript
- 2 JavaScript の基本文法
- 3 DOM
- 4 jQuery

DOM とは

- HTML ページが読み込まれると、JavaScript は各要素に対応する 1 つのオブジェクトを作成します。これらのオブジェクトは、**DOM** (Document Object Model) と呼ばれます。
- DOM の要素は**木**を形成しています。各ノードは、1 つの HTML 要素を表します：



document オブジェクト

- **document** オブジェクトは、DOM オブジェクト木全体のルートノードである。ページ上の全ての要素は、document の子孫として保存されます。
- JavaScript の中で、「document」という名前のオブジェクトを直接使用できます：

```
document.getElementById("text-1").innerHTML = "Hello";
```

HTML 要素の取得

- ページ上のある要素の DOM オブジェクトを取得するには、document オブジェクトの以下のメソッドを使用します：

メソッド	機能
getElementById(id)	ID による要素の指定
getElementsByTagName(tag)	タグ名による要素の指定
getElementsByClassName(class)	クラス名による要素の指定

- CSS のセレクターと似ている方法、つまり、**ID**（例：#text-1）、**タグ名**（例：p）、**クラス名**（例：.class-1）で要素を指定することがわかります。

要素の内容を操作

- 要素を取得したら、その**内容**（開始タグ `<` と終了タグ `>` の間の部分）は **innerHTML** 属性でアクセスできます：

HTML :

```
1 <body>
2   <p id="text-1">fuga</p>
3 </body>
```

Try <sup>01011
11010
01011</sup>
DOM/edit-html.html

JavaScript :

```
1 console.log(document.getElementById("text-1").innerHTML); // => "fuga"
2
3 document.getElementById("text-1").innerHTML = "hoge";
```

要素の属性を操作

- **同名の属性**を使って、要素の**属性**にアクセスできます。例えば、ある要素が href 属性を持つ場合、その DOM オブジェクトの href 属性で属性を取得したり変更したりできます：

HTML :

```
1 <a id="link-1" href="https://developer.mozilla.org">link</a>
```

JavaScript :

```
1 // => https://developer.mozilla.org
2 console.log(document.getElementById("link-1").href);
3 document.getElementById("link-1").href = "https://google.com";
```

Note  例外として、**class** 属性はオブジェクトの **className** という属性で対応しています。

リストを取得した場合

- `getElementsByTagName()`、`getElementsByClassName()` を使うと、**複数の DOM オブジェクト**の指定になることに注意してください。要素のリストが返されたことになります。
- 繰り返し文でそのリストの要素を繰り返さなければなりません:

HTML :

```
1 <p id="text-1">fuga</p>
2 <p id="text-2">hoge</p>
3 <p id="text-3">piyo</p>
```

Try  [DOM/edit-multiple.html](#)

JavaScript :

```
1 const ps = document.getElementsByTagName("p");
2 for (let p of ps) {
3   console.log(p.innerHTML); // => fuga hoge piyo
4 }
```

要素の作成

- 新しい要素をページに追加するには、まず **document.createElement()** メソッドで新しい要素の DOM オブジェクトを**作成**します：

```
let newTitle = document.createElement("h1");
```

- ここで、渡されるパラメータは、要素のタグ名（p、h1、div など）です。
- そして、先ほど説明した方法で、その**内容や属性を設定**します：

```
1 newTitle.title = "Hi!";
2 newTitle.innerHTML = "This Is A Title";
```


要素の追加

- 次に、新しく作成した要素をページに**追加**するために、ページ上の既存の要素の様々なメソッドが使用できます：

メソッド	機能
append()	要素（内部）の末尾に追加する
prepend()	要素（内部）の先頭に追加する
after()	要素の後ろに追加する
before()	要素の前に追加する

- 例えば、以下のコードでは、ID が「text-1」の要素の後に新しく作成した要素を挿入します：

```
document.getElementById("text-1").after(newTitle);
```

要素の削除

- また、**remove()** メソッドを使用し、要素を削除することもできます：

```
document.getElementById( "text-1" ).remove( );
```

Try 

DOM/add-remove.html

DOM イベント

- また、DOM では、ユーザーの操作に対する要素の反応を設定できます。これは、**イベント**^[Event]と呼ばれる仕組みで実現されています。それぞれのタイプのイベントは、ユーザーが異なる動作を起こす時に発生します。よく使われるイベントは以下の通り：

イベント名	発生するタイミング
click	要素がクリックされた時
dblclick	要素がダブルクリックされた時
mouseover	マウスを要素の上に移入させた時
mousemove	マウスを要素の上で移動させた時
mouseout	マウスを要素の上に移出させた時
focus	要素がフォーカスされた時
blur	要素がフォーカスを失った時

イベント動作の設定

- ある要素でイベントが発生したら、どのような JavaScript コードが実行されるかを設定することができます。
- イベントの動作を設定するには、要素の「**on[イベント名]**」属性を使います。例えば、click イベントは、onclick 属性に対応します：

```
1 document.getElementById("text-1").onclick = () => {
2   console.log("Hello");
3 };
```

- この属性に、メソッドを代入することが必要です。上の例では、ラムダ式を使ってメソッドを渡しています。既存のメソッド名をそのまま使用することもできます：

```
1 function printHello() { console.log("Hello"); }
2 document.getElementById("text-1").onclick = printHello;
```

HTML でのイベント動作の設定

- HTML タグの「**on[イベント名]**」属性を使って、イベントの動作を直接定義することもできます：

```
<p id="text-1" onmouseover="printHello( )">fuga</p>
```

イベント動作の追加

- 「onイベント名」属性でイベントの動作を設定すると、以前に設定した動作が上書きされます。したがって、**最後の1つ**の設定だけが有効です。
- 上書きを避け、新しいイベント動作を**追加**するだけなら、要素の **addEventListener()** メソッドを使用します：

```
1 const text = document.getElementById("text-1");
2 text.addEventListener("dblclick", () => console.log("Hello"));
3 text.addEventListener("dblclick", () => console.log("World"));
```

- ここで、1つ目の引数はイベント名、2つ目の引数は追加された動作（メソッド）です。

load イベント

- **load** イベントは、要素が読み込まれた時に発生します。
- 特殊なオブジェクト window の load は、ページ全体がロードされた後に発生します。したがって、実行が必要なコードをこのイベントに追加して、**JavaScript コードが必ずページがロードされた後に実行されることを確保**できます：

```
1 window.addEventListener("load", () => {
2   const text = document.getElementById("text-1");
3   text.addEventListener("dblclick", () => console.log("Hello"));
4   text.addEventListener("dblclick", () => console.log("World"));
5 });
```

Try 01011
11010
01011
DOM/events.html



目次

- 1 JavaScript
- 2 JavaScript の基本文法
- 3 DOM
- 4 jQuery

jQuery

- **jQuery** は、HTML を操作するための JavaScript ツールライブラリです。
- jQuery の目標は、「**より少なく書き、より多く行う**」ことを実現することです。元の DOM API は面倒で重複なコードをたくさん書かなければなりませんが、jQuery は最低限のコードで同じ操作を実現できることを目的としています。
- jQuery は、Google や Microsoft など、多くの有名企業で使用されています。また、Bootstrapをはじめ、多くの JavaScript ライブラリも jQuery に依存しています。



jQuery の読み込み

- jQuery をウェブページに読み込ませるには、ファイルをダウンロードする方法と、**CDN 版**（オンライン版）を使用する方法があります。
- jQuery ライブラリは、以下の公式リンクからダウンロードできます：
 - 🌐 <https://code.jquery.com/jquery-3.6.0.min.js>
 - ダウンロードできたら、`<script>` タグを使用して読み込みます。
- また、このような CDN リンクを使用して `<script>` タグで直接読み込むこともできます：

```
<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
```

JavaScript コードの軽量化

多くの JavaScript ライブラリをダウンロードする際、コードの特別なバージョンを提供しています。例えば、jQuery 3.6.0 では、jquery-3.6.0.js と jquery-3.6.0.min.js という 2 つのバージョンが提供されています。両者の違いは何ですか？

「.min.js」で終わるものは、通常、元のコードの**圧縮された**[\[Minified\]](#)ものです。元のコードと全く同じ機能を備えていますが、軽量化された最小限のコードを使用することで、ブラウザがコードを読み込む時間を短縮し、ユーザー体験を最適化できます。

また、Google Closure Compiler などのツールを使用し、コードを簡単に最小化することもできます。

基本の文法

- jQuery の全ての操作は、特別なオブジェクト「\$」に基づいて行われます。これまで DOM で行っていた操作のほとんどを「\$」のメソッドで置き換えることができます。
- 基本的な使い方は次のような形になります：

```
$("selector").method();
```

- "selector" の部分はセレクターを書いて操作する要素を選択し、様々なメソッドで操作を行います。

セレクター

- jQuery は、DOM の「getElementById()」のような面倒なメソッドを CSS **セレクター** で置き換えます。
- 例えば、先ほどの DOM メソッドで要素を指定するコード：

```
const text = document.getElementById( "text-1" );
const ps = document.getElementsByTagName( "p" );
const divs = document.getElementsByClassName( "main" );
```

- jQuery を使用する場合は、以下のように記述します：

```
const text = $( "#text-1" );
const ps = $( "p" );
const divs = $( ".main" );
```

要素内容の取得

- セレクターで要素を取得したら、その内容を `html()` メソッドで取得できます：

```
1 let text = $("#text-1");  
2 console.log(text.html()); // => fuga
```

要素の内容の修正

- `html()` メソッドは、**内容の修正**にも使用される。同じメソッドに、パラメータを 1 つ渡すだけです：

```
1 let text = $("#text-1");  
2 text.html("hoge");
```


要素の属性の取得

- 要素の**属性**を取得するには、**attr()** メソッドを使用します：

HTML :

```
<a id="link-1" href="https://google.com">fuga</a>
```

JavaScript :

```
1 let link = $("#link-1");  
2 console.log(link.attr("href")); // => https://google.com
```

- このメソッドは 1 つの引数があり、指定したい属性名を表します。

要素の属性の修正

- 要素の属性を修正するにも、**attr()** メソッドを使用します：

```
1 let link = $("#link-1");  
2 link.attr("href", "https://developer.mozilla.com");
```

- このメソッドの 2 番目の引数は、変更後の値を指定します。

要素のプロパティの取得・変更

- 要素の**プロパティ**を取得したり変更したりするには、**css()** メソッドを使用します：

```
1 console.log(text.css("color")); // => rgb(0, 0, 0)
2 text.css("color", "red");
```

- attr() と同様に、パラメータを 1 つ渡したらプロパティの値を取得し、2 つ渡したらプロパティの値を変更します。

Try 01011
11010
01011
[jquery/edit-html.html](#)

複数の要素の場合

- jQuery のほとんどのメソッドでは、要素のリストを繰り返さなくても、**一括して複数の要素を操作**することができます。
- 例えば、以下のコードでは、全ての `<p>` 要素の文字の色を一括に赤に設定しています：

```
1 let allText = $("p");
2 allText.css("color", "red");
```

Try 01011
11010
01011
[jquery/edit-multiple.html](#)

要素の追加

- 新しい要素を**作成**する jQuery メソッドも非常に簡単で、「\$()」の中に要素の**タグを渡す**だけです：

```
let newText = $( "<p>" );  
newText.html( "Added text" );  
newText.css( "color", "red" );
```

- 新しく作成されたら、DOM と同様のメソッド「append()、prepend()、before()、after()」を使ってページに追加できます：

```
$( "div.main" ).append( newText );
```

要素の削除

- **remove()** メソッドを使用し、要素を削除することもできます：

```
1 $( "#text-1" ).remove( );
```

- **empty()** メソッドを使用すると、要素の内容を空にする（すべての子要素を削除する）ことも可能です：

```
$( "div.main" ).empty( );
```

Try  jquery/add-remove.html

イベント動作の設定

- イベントの動作を設定するには、**イベント名と同じメソッド**を使用します：

```
1 $( "#text-1" ).click( ( ) => {  
2     console.log( "Fizz" );  
3 } );
```

イベント動作の追加

- イベント動作の追加（DOM では `addEventListener()` メソッド）は、jQuery では **`on()`** メソッドで行います：

```
1 $( "#text-1" ).on( "click", ( ) => {  
2     console.log( "Buzz" );  
3 } );
```


load イベント

- DOM の `window.load()` は、jQuery の `$()` メソッドで直接設定できます：

```
$(method);
```

```
1 $(( ) => {
2     $( "#text-1" ).click(( ) => {
3         console.log( "Fizz" );
4     });
5 });
```

- jQuery のコードのほとんどをこのラムダ式に記述することで、ページの読み込みが終了した後に実行されるようにすることができます。

Try `jquery/events.html`


エフェクト

- jQuery ではまた多くの要素に動的な**エフェクト**も提供しています。例えば：

メソッド	効果
hide()	要素を隠す
show()	要素を示す
fadeOut()	要素をフェードアウトする
fadeIn()	要素をフェードインする
slideUp()	要素を上にはスライドする
slideDown()	要素を下にはスライドする
animate()	アニメーション効果

Try 01011
11010
01011
jquery/
effects.html

jQuery UI

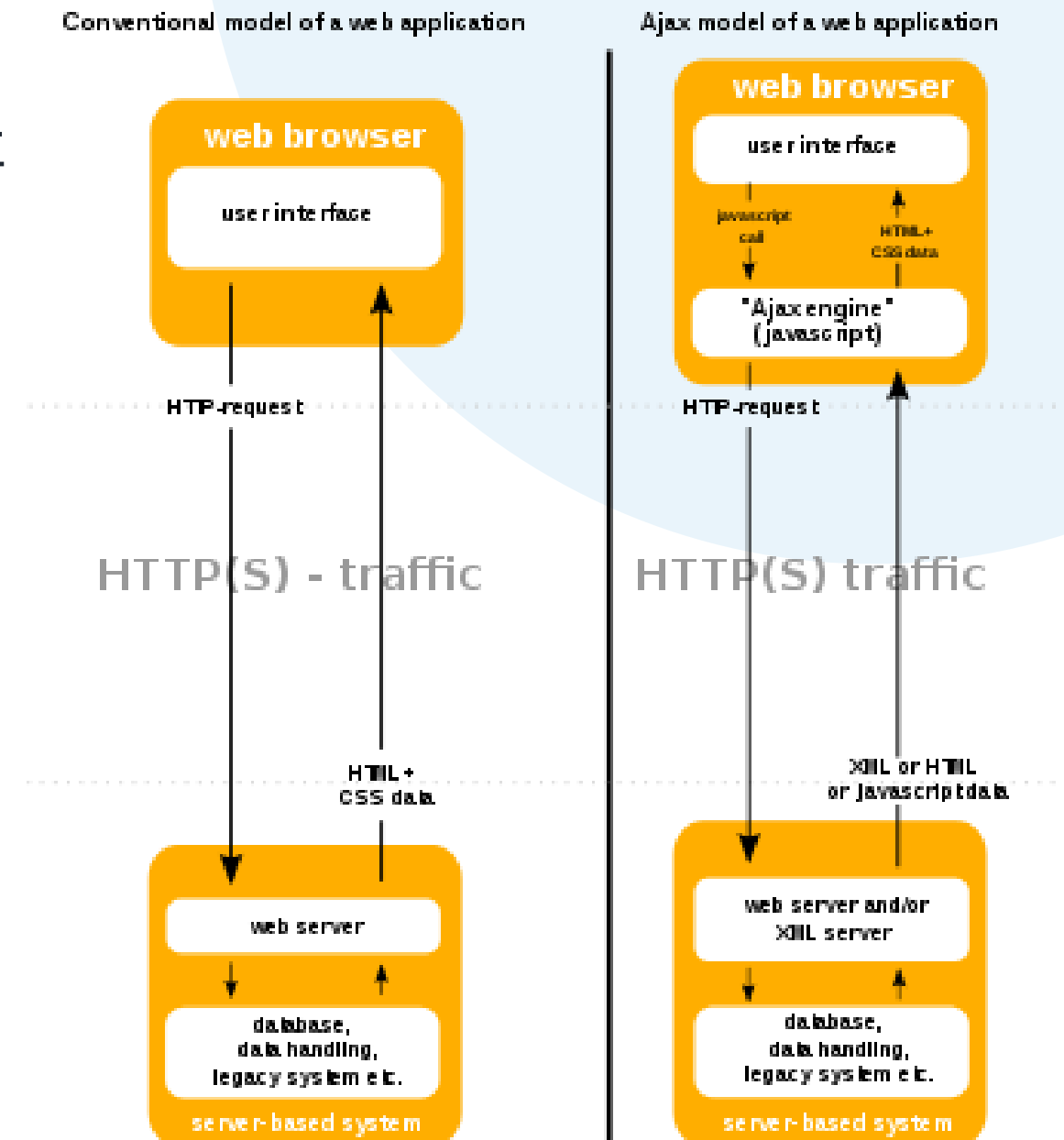
- また、jQuery の拡張ライブラリである **jQuery UI** を使えば、より多くのアニメーション効果を簡単に追加できます（別の JavaScript ファイルを読み込む必要があります）：
 <https://jqueryui.com/>



Q&A

AJAX

- **AJAX** (Asynchronous Javascript and XML) とは、JavaScript などの組み込み言語により、サーバとのデータ交換でウェブページの一部更新を行う技術のことです。
- 従来の動的ウェブモデルとは異なり、ユーザーはサーバが HTML ページ全体を作成・配信するのを待つ必要がなく、クライアント側でページの**一部のみを更新**するため、サーバの負荷とフィードバック速度が大幅に向上します。



jQuery の ajax() メソッド

- jQuery には、サーバにアクセスするための HTTP リクエストを送信し、サーバからレスポンスを返す **ajax()** メソッドが用意されています。このレスポンスを使って、ページ上の特定の情報を変更したり、動的な効果を実装したりすることができます：

```
1 $.ajax({  
2   url: "http://ip-api.com/json/",  
3   data: {fields: "country,regionName,city,lat,lon"},  
4   success: (result) => {  
5       console.log(result);  
6   }  
7 });
```

その他の AJAX メソッド

- jQuery は、AJAX を使用するための便利な方法を他にも数多く提供しています：
 - **get()** メソッドは、簡単な HTTP GET リクエストをサーバに送信。
 - **post()** メソッドは、簡単な HTTP POST リクエストをサーバに送信。
 - **load()** メソッドは、サーバからのレスポンスを HTML コードとして直接に要素に読み込む。

Try 01011
11010
01011
jquery/ajax.html

Q&A

まとめ

Sum Up



1. JavaScript の 2 つの利用方法。
2. JavaScript の基本文法 : Java との違いを覚えることが重要。
3. JavaScript で HTML を操作する方法 : DOM。
4. jQuery で DOM 操作を簡略化する方法 :
 - ① セレクターで要素の取得
 - ② 要素の内容、属性、プロパティの変更
 - ③ イベントの設定



Light in Your Career.

THANK YOU!