JavaScript実践 - day1

今日やること

- ECMAScriptとJavaScriptの関係
- 信頼できる情報源
- 基本文法とHTML要素の操作

今日やること

- ECMAScriptとJavaScriptの関係
- 信頼できる情報源
- 基本文法とHTML要素の操作

JavaScriptとは？

JavaScript

https://developer.mozilla.org/ja/docs/Web/JavaScript

JavaScript (JS) は軽量でインタープリター型（あるいは実行時コンパイルされる）第一級関数を備えたプログラミング言語です。ウェブページでよく使用されるスクリプト言語として知られ、多くのブラウザー以外の環境、例えば Node.js や Apache CouchDB や Adobe Acrobat などでも使用されています。 JavaScript はプロトタイプベースで、マルチパラダイムで、シングルスレッドで、動的な言語であり、オブジェクト指向、命令型、宣言型（関数プログラミングなど）といったスタイルに対応しています。

JavaScript の動的な機能には、ランタイムオブジェクトの構築、可変引数リスト、関数変数、動的スクリプトの作成（`eval` で）、オブジェクトの内包（`for...in` と `Object` ユーティリティで）、ソースコードの復元（JavaScript 関数はそのソーステキストを格納し `toString()` で復元可能）が含まれます。

この章では JavaScript 言語自体について、すなわちウェブページや他のホスト環境に限定されないコアの部分に限定して解説しています。ウェブページ特有の API 群の情報を得たい場合は Web API と DOM を参照してください。

JavaScript の規格書は ECMAScript Language Specification および ECMAScript Internationalization

JavaScript

https://developer.mozilla.org/ja/docs/Web/JavaScript

JavaScript (JS) は軽量でインタープリター型（あるいは実行時コンパイルされる）第一級関数を備えたプログラミング言語です。ウェブページでよく使用されるスクリプト言語として知られ、多くのブラウザー以外の環境、例えば Node.js や Apache CouchDB や Adobe Acrobat などでも使用されています。 JavaScript はプロトタイプベースで、マルチパラダイムで、シングルスレッドで、動的な言語であり、オブジェクト指向、命令型、プログラミングなスタイルに対応しています。

JavaScript の動的な機能 オブジェクト 関数変数、動的スクリプトの作成（`eval` で `for..` リティで）、ソースコードの復元（JavaScript 格納 が含まれます。

この章では JavaScript わちウェブ 環境に限定されないコアの部分に限定して解説してい グ特有の API 群の い場合は Web API と DOM を参照してください。

JavaScript の規格書は ECM age Specificatio AScript Internationalization

JavaScript

https://developer.mozilla.org/ja/docs/Web/JavaScript

JavaScript (JS) は軽量でインタープリター型（あるいは実行時コンパイルされる）第一級関数を備えたプログラミング言語です。ウェブページでよく使用されるスクリプト言語として知られ、多くのブラウザー以外の環境、例えば Node.js や Apache CouchDB や Adobe Acrobat などでも使用されています。 JavaScript はプロトタイプベースで、マルチパラダイムで、シングルスレッドで、動的な言語であり、オブジェクト指向、命令型、宣言型（関数プログラミングなど）といったスタイルに対応しています。

JavaScript の動的な機能には、ランタイムオブジェクトの構築、可変引数リスト、関数変数、動的スクリプトの作成（`eval` で）、オブジェクトの内包（`for...in` と `Object` ユーティリティで）、ソースコードの復元（JavaScript 関数はそのソーステキストを格納し `toString()` で復元可能）が含まれます。

この章では JavaScript 言語自体について、すなわちウェブページや他のホスト環境に限定されないコアの部分に限定して解説しています。ウェブページ特有の API 群の情報を得たい場合は Web API と DOM を参照してください。

JavaScript の規格書は ECMAScript Language Specification および ECMAScript Internationalization

ECMAScriptとJavaScriptの関係

つまり？

JavaScriptには

- コアの部分
- ウェブページ特有のAPI群

の2つの区分がある

ECMAScriptとJavaScriptの関係

つまり？

JavaScriptには

- コアの部分 = **ECMAScript**
- ウェブページ特有のAPI群

の2つの区分がある

ECMAScriptとJavaScriptの関係

つまり？

JavaScriptには

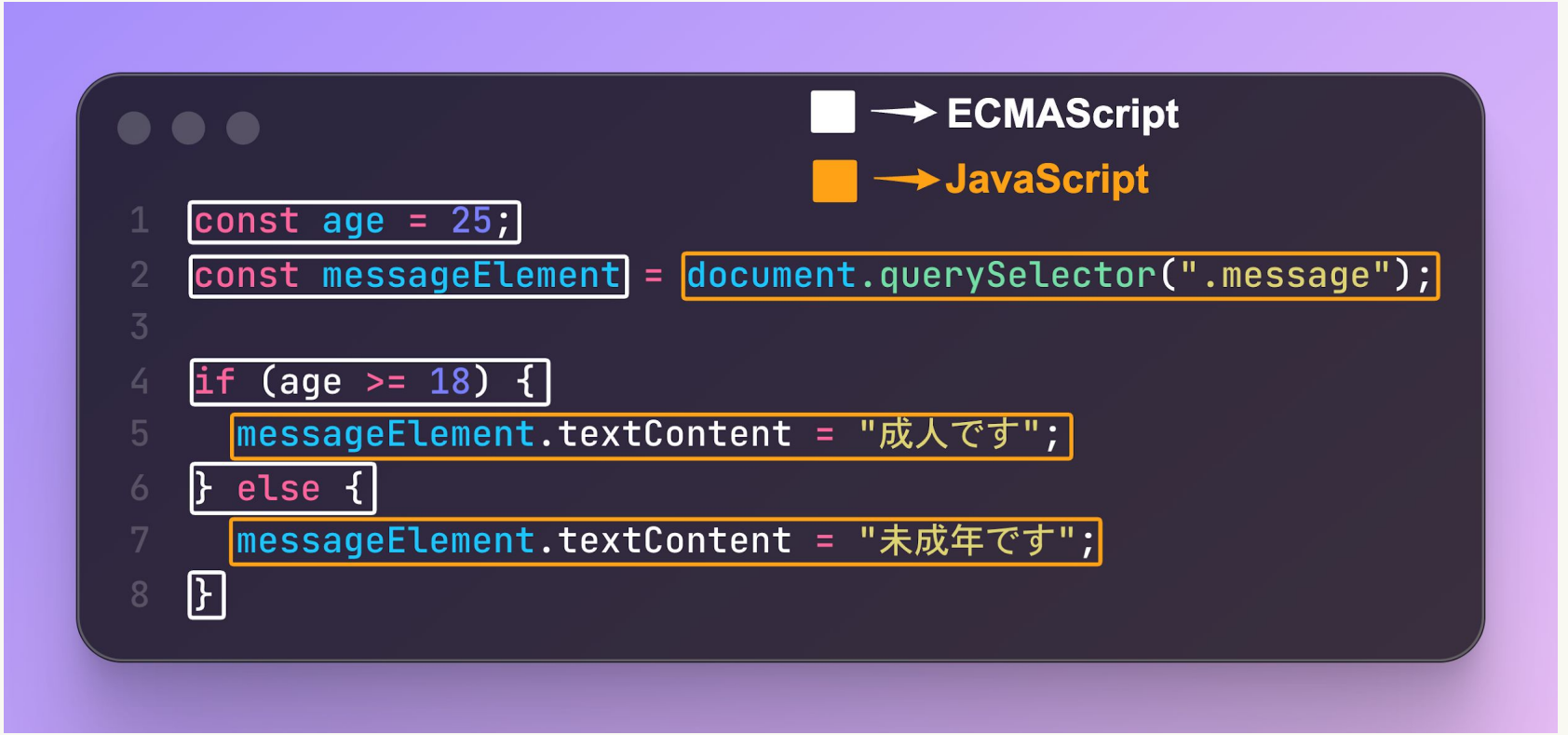
- コアの部分 = **ECMAScript**
- ウェブページ特有のAPI群 = **JavaScript**

の2つの区分がある

実例

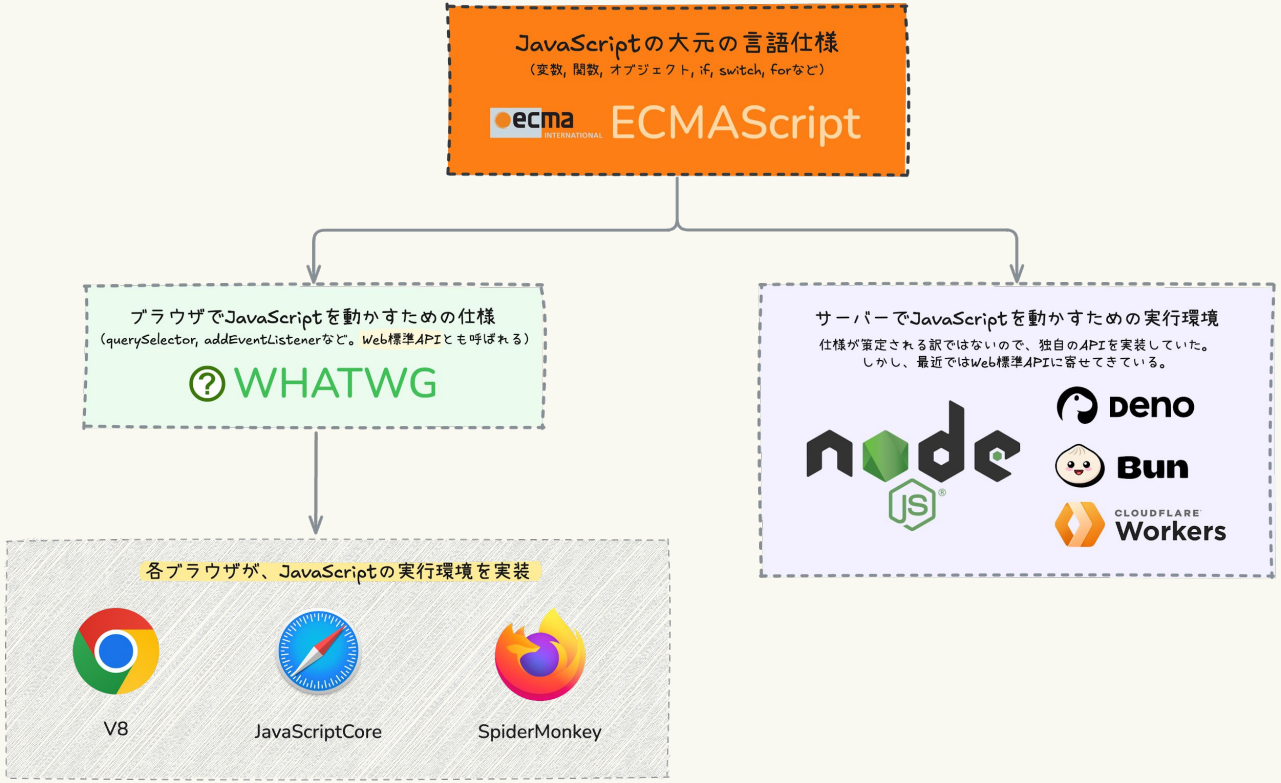
```
const age = 25;
const messageElement = document.querySelector(".message");

if (age >= 18) {
  messageElement.textContent = "成人です";
} else {
  messageElement.textContent = "未成年です";
}
```

実例


ECMAScript
JavaScript
1 const age = 25;
2 const messageElement = document.querySelector(".message");
3
4 if (age >= 18) {
5 messageElement.textContent = "成人です";
6 } else {
7 messageElement.textContent = "未成年です";
8 }

ECMAScriptとJavaScriptの関係

まとめ



補足

- Web標準API: ブラウザに標準装備された道具
- TC39: ECMAScript仕様を決める委員会
- ES2015（ES6）以降: 毎年アップデート（Living Standard）されるJSに変化
- Living Standard: バージョン番号を持たず、常に更新され続ける形式
  - HTML Living Standard など

今日やること

- ECMAScriptとJavaScriptの関係
- 信頼できる情報源
- 基本文法とHTML要素の操作

今日やること

ECMAScriptとJavaScriptの関係

信頼できる情報源

基本文法とHTML要素の操作

信頼できる情報源

信頼できる情報源

- MDN Web Docs
  - 解説、例、互換性データや仕様書へのリンクが豊富
- 自分で実験
  - ブラウザ開発者ツールのConsoleで検証
  - **実際に実行できるかどうか自分で確かめることが大事**

script.js

```
1  const target = document.querySelector(".container");
2
3  // 目的のHTMLが本当に取得できているか確かめる
4  console.log(target);
5
6  target.addEventListener("click", function() {
7    // .containerをクリックした時に実行されているか確かめる
8    console.log("Click!!");
9  }
```

今日やること

ECMAScriptとJavaScriptの関係

信頼できる情報源

基本文法とHTML要素の操作

今日やること

- ECMAScriptとJavaScriptの関係
- 信頼できる情報源
- 基本文法とHTML要素の操作

基本文法とHTML要素の操作

絶対的な決まり事

1. 大文字と小文字を区別する
2. 予約語は使えない
3. 行末のセミコロンは省略可能だが、統一して使用することを推奨

絶対的な決まり事

```
// 1. 大文字と小文字を区別する
const userName = "太郎";
const username = "花子";  // userNameとは別の変数
console.log(UserName); // 存在しない

// 2. 予約語は使えない
const class = "NG";  // エラー: classは予約語
// その他の予約語: break, case, const, newなどがある

// 3. セミコロンは省略可能だが統一すべき
let a = 1;  // セミコロンあり
let b = 2   // セミコロンなし
```

windowオブジェクトとdocumentオブジェクト

- window
  - ブラウザの機能を使用する時の入り口
  - `alert`, `location`, `scroll`など
- document
  - HTML要素にアクセスする時の入り口
  - `document.documentElement` = `<html>...</html>`
  - `document.head` = `<head>...</head>`
  - `document.body` = `<body>...</body>`
  - `querySelector`, `querySelectorAll`で任意のHTML要素を取得

window document

```
// windowオブジェクトはブラウザ自体の機能にアクセスする時の入り口
window.alert("アラート!!"); // アラートが表示される
window.location.href; // 今のURLが表示される
window.scroll(0, 200); // 200px下にスクロールされる

// documentオブジェクトはHTML要素にアクセスする時の入り口
document.documentElement; // <html>要素
document.head; // <head>要素
document.body; // <body>要素
documnet.querySelector(".button"); // .button要素
```

基本文法とHTML要素の操作

`querySelector`

- CSSを指定する時と同じ指定方法で1つのHTML要素を取得
- HTML上で一番最初に見つかった要素を1つだけ取得

`addEventListener`

- ブラウザ上で何かが起こるタイミングのことを**イベント**と呼ぶ
  - 特定の要素をクリック
  - スクロールする
  - キーボードを押す
  - ページの読み込みが完了する
  - ブラウザのサイズを変更する
- イベントが発生した時に実行される処理を設定することを**イベントを貼る**と表現する
  - HTML要素に対して、イベント処理という「機能」を貼り付けるというニュアンス

HTMLの操作

```
// class="button"のHTML要素の最初の1つを取得
const target = document.querySelector(".button");

// ボタンにクリックイベントを「貼る」
target.addEventListener("click", function() {
  // クリックイベントが起こった時に実行したい処理を記述
  console.log("クリックされました");
});
```

変数

- 必要な情報を保管する箱を用意できる
  - 箱に情報を入れておくことで再利用する際に便利になる
- 箱には名前がつけられる（変数名）
  - 変数名に意味のある名前を使用することで可読性が向上する
- `let`か`const`を使用
  - `let`は中身を入れ替えられる箱
  - `const`は中身を**入れ替えられない**箱
- 原則、`const`を優先
  - 中身が入れ替わらないという制約がバグを少なくする
  - 変数の中身を後で**意図的に**変更する場合や、プログラム上必要な場合に`let`を使用
- `var`は使用しない

条件分岐

- `if`を使用すると、特定の条件を満たすかどうかで行う処理を変更することが出来る
- 条件は開発者ツールの`Console`に入力して確かめることができる

条件分岐

```
if(条件) {
    // 条件がtrueの時の処理
} else {
    // 条件がfalseの時の処理
}

// 条件がtrue, falseどちらになるか確かめる
console.log(条件);
```

条件分岐

- `if`の条件で何かと何かを比較する場合に使用する記号を**比較演算子**と言う
- 比較演算子には色々な種類があるが、等しいか等しくないことを条件にする場合には
  `===`(厳密に等しい)か`!==`(厳密に等しくない)のどちらかを使用することを推奨

厳密等価演算子を推奨

```
// == の比較では、数字の5と文字の5が等しい
console.log(5 == "5"); // true

// 厳密等価演算子を使用すれば等しくない
console.log(5 === "5"); // false
```

関数

- 関数(Function)は、特定の処理をまとめて再利用するための機能
- 定義した関数は関数名();で実行できるが、定義しただけでは実行されない

関数

```
// hello関数を定義
const hello = function() {
  console.log("こんにちは！！");
}

// 定義しただけでは実行されないので、実行する必要がある
hello();
```

関数 - 引数(ひきすう)

- 関数名( )の( )に関数内で使用するパラメーターを指定できる
- このパラメーターのことを**引数**と呼ぶ

関数 - 引数

```
// 引数に name を指定
const hello = function(name) {
  console.log(`こんにちは！${name}さん`);
}

hello("田中"); // こんにちは！田中さん

// 複数の引数を指定する場合は ,（カンマ）で区切る
const hello_with_message = function(message, name) {
  console.log(`${message}！${name}さん`);
}

hello_with_message("こんばんわ", "田中"); // こんばんわ！田中さん
```

関数 - スコープ

- `{}`の中で設定された変数には関数の外からはアクセス出来なくなる
- この変数が有効な範囲のことを**スコープ**と呼ぶ
- 関数で使用する変数はなるべく関数のスコープの中で定義することを推奨

関数 - スコープ

```
const show_name = function() {
  // スコープ始まり
  const name = "山田";

  console.log(name); // 山田
  // スコープ終わり
};

console.log(name); // スコープ外なのでname変数はエラー
```

`querySelectorAll`

- CSSを指定する時と同じ指定方法で全てのHTML要素を取得

ループ処理

- 複数の要素などに順番に処理を実行する
  - 反復処理をする方法の1つが`for...of`文

for...ofの実例

index.html

```
<div class="item">田中</div>
<div class="item">佐藤</div>
<div class="item">渡辺</div>
```

for...ofの実例

複数の要素

```
for(変数名 of 複数の要素) {
    // 上で設定した変数はこの中で使用できる
}

// class="item" の要素を全て取得
const names = document.querySelectorAll(".item");

// class="item" の要素を1つずつ処理する
for(const name of names) {
  name.addEventListener("click", function() {
    console.log(name.textContent);
    // 田中、佐藤、渡辺
  }
}
```

実際に作ってみる

信号機🚥

- 赤、黄、緑色の要素をクリックしたら背景色をその色に変化させる
- 2回同じ色をクリックしたら背景色を初期状態に戻す

どこから手をつければいいのかわからない‥🧐

初めの一歩

- 何に対して
- どんな時に
- どんな状態になって欲しいのか

を考える

初めの一歩

- 何に対して→ `querySelector`
- どんな時に
- どんな状態になって欲しいのか

を考える

初めの一歩

- 何に対して→ `querySelector`
- どんな時に→ `addEventListener`
- どんな状態になって欲しいのか

を考える

初めの一歩

- 何に対して→ `querySelector`
- どんな時に→ `addEventListener`
- どんな状態になって欲しいのか→ 実際の処理

を考える