HTML + CSS
&
javascript

基本

Chapter.1

用語解説

HTML

Hyper Text Markup Language の略語。

いわゆる「ウェブページ」を構成するパーツを、ひとつひとつ定義（マークアップ）していくための言語で、近年 HTML5 が正式勧告されている。

ファイル拡張子は *.html、あるいは *.htm が一般的。

CSS

Cascading Style Sheets の略語。

スタイルシート、あるいは単に CSS と呼ばれる。「スタイル」の語からもわかるように、CSS はウェブページの「外見・外観」を定義するために使われる。

近年では、CSS3 という新しい規格が徐々に浸透してきており、ただ単に見た目を変化させるだけでなく、動きをつけてアニメーションさせたりすることもできるようになった。

拡張子は *.css とする。

JavaScript

名前は似ているが「JAVA」とは無関係。

主にウェブブラウザで利用されるスクリプト言語で、最近ではサーバーサイドの実装にも利用されるようになり、活躍の場を広げている。

HTML、CSS が静的なものであるのに対し、JavaScript は動的なものであると説かれることが多い。

拡張子は *.js とする。

役割の違いをしっかり把握

- HTML は「ウェブページに意味を定義する」
- CSS は「ウェブページの外見を決める」
- JavaScript は、ウェブページで利用する場合は「動的にページやその他の処理を制御する」
- Node.js の登場により JavaScript はブラウザ以外のところでも使われるようになってきている

Chapter.2

HTML を書いてみる

タグ

HTML は先述の通り「意味を定義していく」ことが役割。

すごく大雑把な言い方をすると「見た目はどうでもいいからしっかり意味合いや構造がわかるように書く」ということが大事。

ではどのようにして意味を与えるのか、と考えたときに登場するのが「HTML タグ」と呼ばれるもの。不等号の記号などを使ってタグを書き、それにより意味のある構造を作っていくのが HTML を記述するということ。

タグの記述例

```
<html>
    <body>
        <h1>見出し</h1>
        <p>段落</p>
        <p>段落</p>
        <p>段落</p>
    </body>
</html>
```

※ 入れ子構造になっているのがポイント

HTML タグ

次のタグを書いた HTML ファイルを用意し、実際にブラウザで開いてみる。

- <h1> moji </h1>
- moji
- kyouchou 　←入れ子構造が維持されている
- list 　←特定のタグの中にしか書けないタグがある
- <inpur type=”text”>　← 閉じタグがないタグもある
- <head></head>　← ページ上に全く見えないタグもある

HTML はあくまでも意味や構造を定義するものであるということを意識する。

HTMLタグの属性値

タグには属性を持たせることができる。

属性を持たせる場合は、開始タグに属性を記述する。

- 属性付きの段落　← 任意の ID を属性として付加
- <input type=”checkbox”>　← ある意味 input は、type 属性が必須
- <canvas width=”500”>　← 一部大きさなどの外見に影響の及ぶ属性を持つ場合もある

※ ID は対象となるタグに一意の名前をつけるものなので、同じ ID を持つタグをページ内に複数置くことはできない。そういうことがしたい場合は ID ではなく、class を利用する

※クラスも ID も外見には一切影響を与えないが後で出てくる CSS や JavaScript との連携に役立つ

Chapter.3

CSS を書いてみる

CSS セレクタ

CSS を書く際は「セレクタ単位」でスタイルを決めていく。

セレクタは、以下の様なものを区分として設定できる。

- タグの種類によって区別
- タグに振られている ID を基準に区別
- タグに振られている class を基準に区別

CSS の記述例

```
body {
    font-size: 20pt;     /* フォントのサイズを 20pt に */
    font-weight: bold;   /* フォントを太字に */
    margin: 0;           /* マージンはタグの外側の余白 */
    padding: 0;          /* パディングはタグの内側にできる余白 */
}

p {
    color: red;          /* テキストの色を赤に */
}

#paragraph {                      /* シャープは ID 名を表す */
    text-decoration: underline; /* 下線を引く */
}
.paragraph {                      /* ピリオドは class 名を表す */
    text-align: right;            /* テキストを右寄せにする */
}
```

CSS の記述

CSS は奥深いので少しずつ覚えるのがよい。

ブロック要素、インライン要素、というのを意識することで理解が深まる。

ブロック要素は、単体で配置しても自然と改行されるように配置される。インライン要素はその名の通り、文章内に配置しても勝手に改行されることはなく、同じ行のなかに収まる。

両者の違いを意識するだけで CSS は格段に書きやすくなるが、かなり細かい仕様が多いので焦らず、よく使うものに絞って覚えるのがよい。

Chapter.4

JavaScript とは

JavaScript

JavaScript は、ウェブページ内で動的に処理を行いたい場合に利用するスクリプト言語。

たとえば、入力フォームの値が正しく入力されているかをチェックして分岐する処理は、HTML や CSS だけでは実現できない。JavaScript を使うことで、ウェブページに様々な機能を追加することが可能になる。

CSS を無効化してもページの機能は失われないが、JavaScript を無効化するとページそのものが表示されなくなるなど、場合により機能が失われる可能性がある。

JavaScript

最近では、JavaScript を利用していないウェブページはほとんど皆無。

また、ウェブブラウザの多くが、JavaScript を扱う上で便利な機能やツールを提供しており、JavaScript での開発、デバッグを行う際には、そのブラウザに組み込まれた便利機能を活用する。

Windows 版の Chrome なら F12 を、Mac 版なら Command＋Option＋I キー同時押しで表示でき、一般にデベロッパーツール、あるいは開発者ツールと呼ばれている。

※ 開発者ツールには、JavaScript 以外にも HTML や CSS の情報が網羅されており、ウェブ開発には欠かせない。

JavaScript コンソール

開発者ツールに付属するコンソールを使って、JavaScript をその場で実行することができる。

以下をコンソールに入力して Enter キーを叩くとどうなるでしょう。

```
100 * 100
alert('hello!');
Date.now()
```

※もしコンソールが出ていない場合は、開発者ツールにフォーカスがある状態で Esc キーを押すといつでも表示させられる

Chapter.5

JavaScript 変数と基本構文

JavaScript の変数

JavaScript では、変数は原則として型を持たず、動的に型が変化する。

そもそも、型を宣言する方法がないうえに、一度数値を格納した変数に、あとから文字列を格納しなおす、といったことも普通に可能。

```
var i; // 変数宣言は var をつける（つけなくてもエラーにはならないがグローバルになる）


i = 0;
i = 'aaa';
i = false; // 次々と格納されるデータの型が変化しているが問題なし


i = 0;
alert(i + 'AAA'); // どんなことが起こるでしょうか？
```

JavaScript の変数

内部的には一応型が別れており、数値型、文字列型、オブジェクト型などがあるが、C 言語などの言語とはかなり型の概念が違うので注意。

また、厳密な数値を扱いたい場合向けに、最近では「型付き配列」といったものを使うこともできるが、通常の開発では滅多に使われない。

ただし WebGL 開発のような、厳密なバイトレベルでのデータの管理が必要なケースでは型付き配列が逆に欠かせないものになる。

if 文

```
if (式) {
    処理;
}
// --------------------
if (式) {
    式が真の場合の処理;
} else {
    式が偽の場合の処理;
}
// --------------------
if (式１) {
    式１が真の場合の処理;
} else if(式２) {
    式２が偽の場合の処理;
}
```

やってみよう

（式）に数値が与えられる前提で、その数値が奇数か偶数かで分岐する処理を if 文を使って書くにはどうしたらいいだろうか？

switch 文

```
switch （式） {
    case 値１：
        式の結果が値１の場合の処理；
        break;
    case 値２：
        式の結果が値２の場合の処理；
        break;
    default ：
        式の結果が値１、値２以外の場合の処理；
}
```

やってみよう

（式）に 1 ～ 50 の数値が与えられるとして、十の位と一の位がゾロ目になる数値だけを判定する switch 文はどう書く？

for 文

```
for (初期化式; 比較式; 更新式) {
    処理;
}
```

※ break、continue が利用できる

※ 以下は記述例（100回実行される）

```
for (i = 0; i < 100; ++i) {
    j += 2;
}
```

やってみよう

0 ～ 100 までループする for 文を使って FizzBuzz に挑戦してみよう。

for in 文

```
for (変数 in オブジェクト) {
    処理;
}
```

※ break、continue が利用できる

※ 後述するオブジェクトを理解してから詳しく説明するが、for in ループではオブジェクトのプロパティ名を順番に取り出しながらループする処理が記述できる

while 文

```
while (式) {
    処理;
}
```

※ while は式が真である限り繰り返し実行され続けるので
　無限ループにならないように注意

※ break、continue が利用できる

try catch 文

```
try {
    トライする処理;
} catch (err) {
    例外（エラー）が発生した場合の処理;
} finally {
    最終処理;
}
```

※ エラーが起こった場合の詳細な情報は `err` に格納されている

※ 最終処理は、トライした結果が成功であろうがエラーであろうが関係なく実行される

Chapter.6

ウェブ開発とデバッグ

最低限覚えておきたい

開発者ツールは非常に強力なツールなので、使い方はある程度覚えておきたいところ。

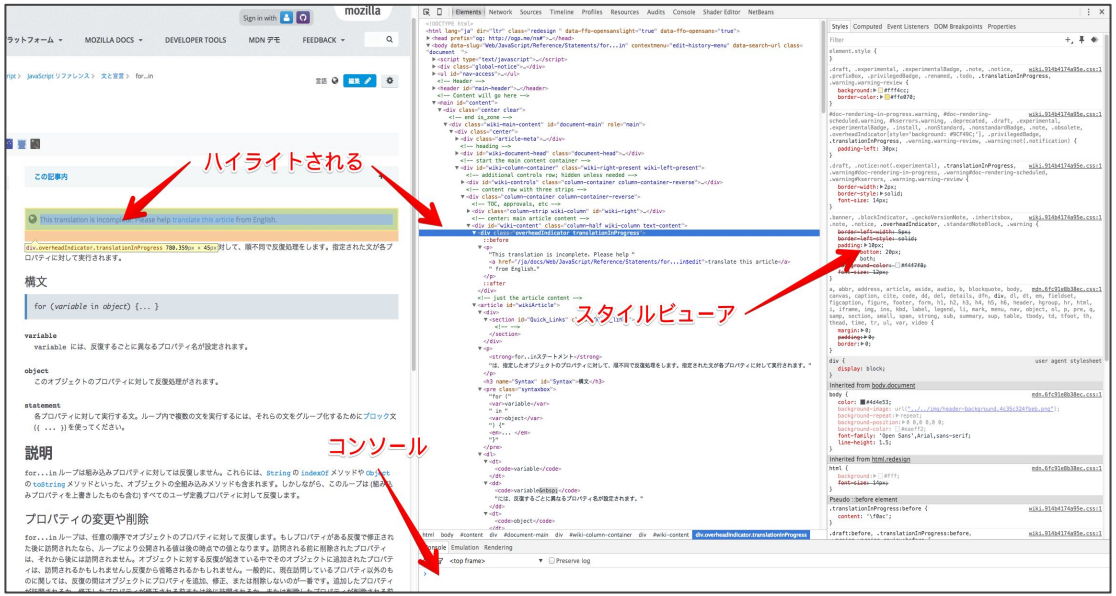
最低でも次に示す内容については、把握しておくと開発の助けとなる。

- エレメントパネルの使い方
- ソースパネルの使い方
- コンソールの使い方

Element パネル

element、つまり HTML タグで記述されたウェブページの構造をそのまま見ることができる。

スタイルビューアの部分では、現在あたっているスタイルが閲覧できる他、動的にスタイルを書き換えてウェブページ側へと反映させることも可能。

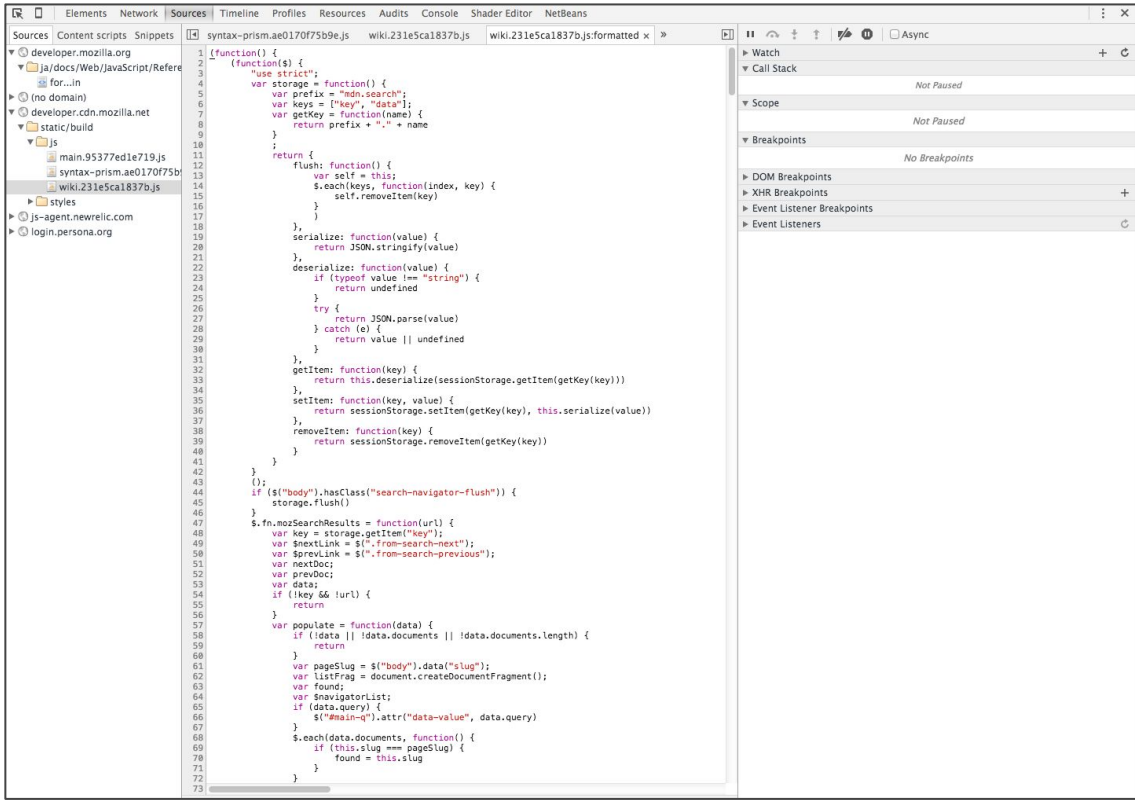


Source パネル

HTML のソース、JavaScript のソースが閲覧できる。

Minify されて難読化されてしまっているソースコードを整形する、ブラウザで編集した結果をローカルファイルに反映するなどの便利機能が大量に含まれている。

最低限、ステップ実行の使い方は押さえておくこと。



コンソールの使い方

コンソールでは、その場で即席のインラインコードを実行できる。

また、JavaScript のソースのほうに「console.log(xxx);」と記述されたものは、実行時にコンソールに出力されるので、いわゆる printf 的な用途として利用できる。

また、オブジェクトをログ出力すると、その構造を維持した状態で出力してくれる（言葉で説明しにくい……）

上矢印キーで履歴を遡れる。またオブジェクトが持つプロパティなどはサジェストされるので、ぱぱっと結果を確かめたい場合などにとても便利。

Chapter.7

関数定義

関数

JavaScript では関数は「 function 」で定義できる。

```
function myFunc ([引数]) {
    処理;
    return 戻り値;
}
```

変数に型がないので、当然引数にも型指定をする方法はない。

値を返す場合は「 return 」を使う。

関数の呼び出し

関数を呼び出す場合には、関数名に丸括弧を付加して呼び出す。

```
function myFunc ([引数]) {
    処理;
}

myFunc(); // ここで呼び出される
```

JavaScript 関数の不思議

JavaScript では、関数であっても、変数に格納できてしまう。

```
var myFunc = function ([引数]) {
    処理;
};
```

もちろん変数に格納した関数を呼び出すことも可能。

```
myFunc();

※ 上段のコードで定義した関数が呼び出される
```

JavaScript では全てがオブジェクト

なぜ関数が、変数に格納できてしまうのか。

これは、JavaScript ではあらゆるものが、内部的にはオブジェクトとして扱われているため。

関数でさえも、オブジェクトの一種として扱われているので、変数に代入することができてしまう。

（このへんは難しいので、今後も少しずつことあるごとに復習するようにしますので、現時点でよくわからなくても問題ありません）

Chapter.8

DOM

DOM (Document Object Model)

JavaScript から、HTML として記述されたタグにアクセスし、操作したり属性値を取得したりすることができる。

このような、JavaScript から HTML ドキュメントにアクセスするための仕組みを総称して DOM（どむ）と呼ぶ。

この DOM の概念が存在することにより、JavaScript は動的にウェブページにアクセスして要素を動かしたり、値をチェックしたりすることができる。

DOM (Document Object Model)

DOM は HTML のタグの構造と同じように、オブジェクトが階層構造を作っている巨大なオブジェクトの塊、というふうに言える。

たとえば、HTML で記述した <body> タグの中身をチェックしたいと考えた場合、まずは body エレメントに JavaScript でアクセスする。その際のコードは以下のようになる。

```
var b = document.body;
console.log(b.textContent);
```

DOM (Document Object Model)

```
var b = document.body;
console.log(b.textContent);
```

すべてのエレメントが、「 document 」というグローバル変数の中に階層構造を維持したまま格納されている。

JavaScript からは、これを深くたどっていけば、最終的にウェブページ上のあらゆるエレメントへといつかたどり着くことができる。まさに、HTMLドキュメントがそのまま JavaScript のオブジェクトになっていると考えるとよいかも。

document.getElementById

ID 属性が付加されているエレメントを、ID を指定して取得する。

※おさらいすると、同じ ID を同一ウェブページ上に複数設置することができないので、getElementById を用いれば特定のオブジェクトへの参照を簡単に得られる

```
var element = document.getElementById('target');

console.log(target.id);
```

※ 取得したエレメントが持つ属性に関する情報は、ピリオドを使ったプロパティ参照を行うことで自由に取得できる

やってみよう

HTML に固有の ID を持つ要素を配置し、JavaScript を使ってその ID を持つ要素の、textContent プロパティを書き変えてみよう。

※ textContent はタグの間に記述された文字列を表すプロパティ

element.parentNode プロパティ

parentNode は、element を格納している親要素を取得するときに使う。

実は DOM で参照できるエレメントには、自分自身を削除するというメソッドは存在しない。自分の子要素を削除するメソッドは存在するため、自分自身を削除したいときなどに使うことがある。

```
var target = document.getElementById('target');
var parent = target.parentNode;
parentNode.removeChild(target);
```

※ 要素同士が階層構造になっており、親 ⟷ 子 という関係になっていることを意識する

やってみよう

親要素への参照を取得し、子要素を削除してみよう。

子要素の削除には、親要素の removeChild が利用できる。

document.createElement
element.appendChild

removeChild は要素を削除するためのメソッドだったが、createElement は逆に要素を新しく生成するためのメソッド。引数に、文字列で生成したいタグ名を渡すと、新しいエレメントを生成してくれる。
生成したエレメントは、appendChild されることで対象エレメントの子要素としてドキュメント内に置かれる。

```
var target = document.getElementById('target');
var newElement = document.createElement('h1');
target.appendChild(newElement);
newElement.textContent = 'title string';
```

やってみよう

新しく要素を生成して、それを body の中に新しく配置してみよう。

element.style プロパティ

エレメントは、当然ながら CSS によってスタイルが当てられている。
JavaScript からは、このスタイルにアクセスすることも可能。
CSS の属性名は、DOM でアクセスする際はハイフンを消し、大文字化するというルールがあり、慣れれば簡単。
このスタイルを動的に変更できるテクニックは動的なサイトの構築にかなり便利。

```
var target = document.getElementById('target');
target.style.fontSize = '32px';
```

※ 上の例では、本来は「 font-size 」となる CSS の属性名を、DOM で参照するうえではハイフンを消し、大文字化する場合の例。

やってみよう

開発者ツールのコンソールを使って、様々なエレメントのスタイルを JavaScript で制御してみよう。

Chapter.9

DOM イベント

イベント

動的なウェブページを構築する上では、このイベントの概念がとても重要となる場合が多い。

イベント、とはユーザーが起こすあらゆるアクションと、なにかしらの動作に由来する事象として検知できるものすべてのことを言う。

- マウスが動いた、クリックされた、ダブルクリックされた、等
- ウィンドウがリサイズされた、ページのロード完了となった、等
- ボタンが押された、選択肢が変更された、等
- キーボードが押された、キーが離された、等

イベントを検知すると何ができるのか

イベントを検知する方法がわかると、様々な出来事に反応して動くスクリプトを記述することができるようになる。

たとえば、ユーザーが起こしたクリックというアクションに反応して処理を行ったり、ファイルが読み込まれ、完全にロード完了となったタイミングから初期化を開始したり、といったことができる。

JavaScript はイベントを検知して動く仕組みをうまく利用することが、ある意味では必須の言語と言える。

element.addEventListener

エレメントにイベント検知時に実行する関数を登録する。
どのようなイベントに関数を紐付けるのかは、引数に渡した文字列で決まる。
この addEventListener をいかに使いこなせるかはとても大事。
また、関数を変数に格納できるというのも、こういう時に便利。

```
function myFunc (){alert('click event!');}

var element = document.getElementById('target');
target.addEventListener('click', myFunc, false);
```

※ 第一引数が対象となるイベントの種類、第二引数が登録する関数、
第三引数はひとまず難しく考えず、`false` にしておけば問題なし

やってみよう

ページ内の要素をクリックすると、アラートが出るようにしてみよう。

event.currentTarget プロパティ

イベント発生時には、addEventListener に登録した関数の引数に、自動的に Event オブジェクトが格納された状態で呼び出しが行われる。この Event オブジェクトの currentTarget プロパティには、イベントが発火したエレメントが格納されている。

これを利用すると、イベントの発生の引き金となったエレメントが簡単に調べられる。

```
document.getElementById('target');
target.addEventListener('click', function(eve){
    eve.currentTarget.style.backgroundColor = 'red';
}, false);
```

※ JavaScript ではしばしばこんなふうに関数をわざわざ定義せずに即席で生成するような書き方をすることがある（いずれ詳しくやります）

やってみよう

クリックされた要素自身が、色を変化させるようなイベント処理を記述してみよう。