DOM

DOM (Document Object Model) は、HTML や XML ドキュメントのためのプログラミングインターフェースです。

■DOM の html ツリー構造

ドキュメントをツリー構造として表現します。このツリー構造により、プログラムがドキュメントの内容を読み取り、変更することが可能になります。

◆DOM のツリー構造

・ルートノード

通常は<html>要素がルートノードとなります。この要素からすべての他の要素が派生します。

・親ノード、子ノード、兄弟ノード

各要素は親ノードを持ち、一つまたは複数の子ノードを持つことができます。
同じ親を持つノードは兄弟ノードとなります。

・要素ノード

HTML のタグ（例えば<body>, , <p>など）は要素ノードとして表されます。

・テキストノード

HTML 要素のテキスト内容はテキストノードとして表されます。

・属性ノード

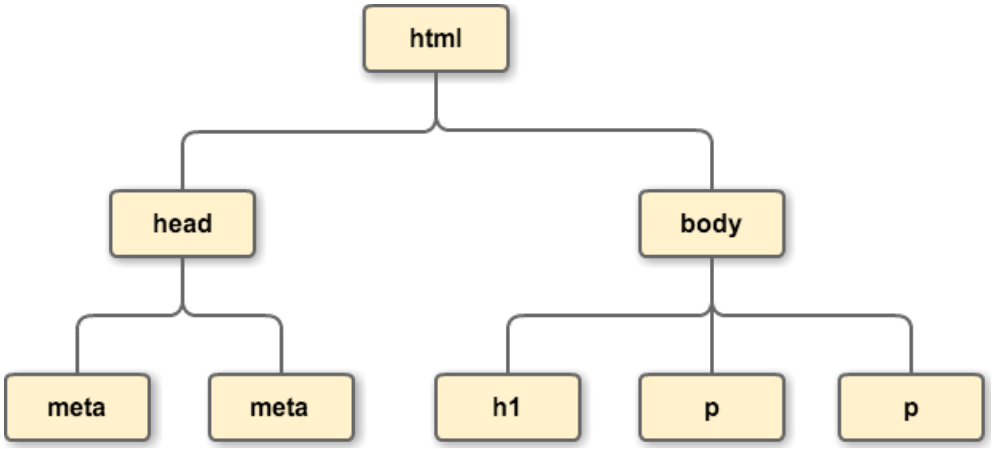
HTML 要素の属性（例えば class, id, style など）は属性ノードとして表されます。

例：簡単な HTML ドキュメントの DOM 構造

```
<!DOCTYPE html>
<html>
<head>
    <title>DOM の例</title>
</head>
<body>
    <h1 id="header">こんにちは</h1>
    <p>これは段落です。</p>
</body>
</html>
```

この HTML ドキュメントの DOM 構造は以下のようになります：

・html 要素はルートノードです。
・head と body 要素は html の子ノードで、互いに兄弟ノードです。
・title 要素は head の子ノードです。
・h1 と p 要素は body の子ノードで、互いに兄弟ノードです。
・h1 要素はテキストノード「こんにちは」を持ちます。
・p 要素はテキストノード「これは段落です。」を持ちます。
・h1 要素には属性ノード（id="header"）があります。



このツリー構造によって、JavaScript を使用して DOM にアクセスし、HTML ドキュメントの内容を動的に変更することができます。

■DOM でできること

DOM を利用することで、ウェブページはよりインタラクティブで、ユーザーにフレンドリーなものになり、開発者はクリエイティブなウェブアプリケーションを実現することができます。

1. 要素の選択と変更

特定の要素を ID、クラス名、タグ名、CSS セレクタなどで選択し、その内容（テキストや HTML）を変更できます。

2. 要素のスタイルの操作

要素のスタイル（色、サイズ、フォント、配置など）を動的に変更できます。

3. DOM の構造の変更

新しい HTML 要素を作成し、それを DOM に追加するか、既存の要素を削除または置換できます。

4. イベントのハンドリング

要素に対するユーザーのアクション（クリック、ホバー、キープレスなど）に反応して、特定のコードを実行できます。

5. フォームデータの操作

ユーザー入力を取得し、フォームの送信を制御し、バリデーションを実行できます。

6. アニメーションと相互作用

要素の表示や非表示、移動、サイズ変更などのアニメーションを作成できます。

7. ウェブページの動的更新

ページ全体をリロードせずに特定の部分だけを更新できます（例：AJAX を使用したコンテンツの更新）。

8. クライアントサイドのデータストレージ

ローカルストレージやセッションストレージを使用して、ブラウザ上でデータを保存し、取得できます。

9. ウェブページのアクセシビリティ向上

DOM を通じて、ウェブページのアクセシビリティを改善するための属性や構造を調整できます。

10. 動的なコンテンツ生成

ユーザーの操作や外部データソースに基づいて、リアルタイムでコンテンツを生成、表示できます。

＜実例＞

「3.」の例:新しい<p>を追加する

DOM を利用して新しい**<p>**（段落）要素を作成し、それをウェブページに追加するための JavaScript のサンプルコードを以下に示します。このコードは、新しい**<p>**要素を作成し、その要素にテキストを追加してから、指定された親要素（例えば**<body>**タグ）にこの新しい要素を挿入します。

```
// 新しい<p>要素を作成
var newParagraph = document.createElement("p");

// 新しいテキストノードを作成し、それを<p>要素に追加
var text = document.createTextNode("これは新しく追加された段落です。
");
newParagraph.appendChild(text);

// <body>要素(または他の要素)に新しい<p>要素を追加
document.body.appendChild(newParagraph);
```

＜コードの説明＞

・document.createElement("p")を使用して新しい<p>要素を作成します。

・document.createTextNode("...")を使用して、新しい段落に追加するテキストノードを作成します。

・appendChild メソッドを使用して、作成したテキストノードを<p>要素に追加します。

・最後に、document.body.appendChild(newParagraph)（または他の親要素を指定）を使用して、新しい<p>要素をドキュメントのボディ（または指定された別の親要素）に追加します。

このコードを実行すると、指定されたテキストを含む新しい段落がページのボディの最後に追加されます。また、document.getElementById や document.querySelector などを使用して、新しい<p>要素を追加する特定の親要素を選択することもできます。

「4.」を使ったボタン操作

新しい**<p>**要素を追加する機能とボタンをクリックしたときにアクションを実行する機能を組み合わせた HTML サンプルこの HTML ファイルは、ボタンをクリックすると新しい段落をページに追加する機能を持ちます。

```
<!DOCTYPE html>
<html>
<head>
    <title>DOM 操作のサンプル</title>
</head>
<body>

<button id="myButton">新しい段落を追加</button>

<script>
    // ボタン要素を選択
    var button = document.getElementById('myButton');

    // ボタンにクリックイベントリスナーを追加
    button.addEventListener('click', function() {
        // 新しい<p>要素を作成
        var newParagraph = document.createElement("p");

        // 新しいテキストノードを作成し、それを<p>要素に追加
        var text = document.createTextNode("これは新しく追加された段落です。");
        newParagraph.appendChild(text);

        // <body>要素に新しい<p>要素を追加
        document.body.appendChild(newParagraph);
    });
</script>

</body>
</html>
```

＜コードの説明＞

この HTML ファイルには<button>要素が含まれており、このボタンには id="myButton"が設定されています。

<script>タグ内で、ボタン要素を選択し、そのボタンにクリックイベントリスナーを追加しています。

ボタンがクリックされると、新しい<p>要素が作成され、その要素にテキストノードが追加され、最終的に新しい段落がドキュメントのボディに追加されます。

このファイルをウェブブラウザで開くと、ボタンをクリックするたびに新しい段落がページの末尾に追加されます。

「6.」を使ったアニメーション

DOM を使用して基本的なアニメーションを作成する一つの方法は、JavaScript で要素のスタイルを時間の経過とともに変更することです。以下に、単純なアニメーションのサンプルを示します。この例では、ボタンをクリックすると、ある要素が水平方向に動くアニメーションを実行します。

＜コードの説明＞

HTML では、動かす要素（ここでは div 要素）とアニメーションを開始するボタンを定義しています。

CSS スタイルを使用して、div 要素に赤色の背景、サイズ、初期位置を設定します。

JavaScript では、startAnimation 関数内でアニメーションのロジックを定義しています。

この関数では、setInterval を使用して一定間隔で frame 関数を呼び出し、div 要素の left プロパティを増加させています。

pos 変数が特定の値（ここでは 350 ピクセル）に達すると、clearInterval を使用してアニメーションを停止します。

最後に、ボタンにクリックイベントリスナーを追加し、クリックされたときにアニメーションを開始します。

このサンプルを実行すると、ボタンをクリックすると div 要素が右に移動するアニメーションが実行されます。これは DOM を使用した基本的なアニメーションの例であり、この原理を応用してより複雑なアニメーションを作成することも可能です。

```
<!DOCTYPE html>
<html>
<head>
    <title>DOM アニメーションのサンプル</title>
    <style>
        #animateBox {
            width: 50px;
            height: 50px;
            background-color: red;
            position: absolute;
            left: 0px;
        }
    </style>
</head>
<body>
<button id="startButton">アニメーション開始</button>
<div id="animateBox"></div>
<script>
    // アニメーションを実行する関数
    function startAnimation() {
        var elem = document.getElementById("animateBox");
        var pos = 0;
        var id = setInterval(frame, 5);

        function frame() {
            if (pos == 350) {
                clearInterval(id);
            } else {
                pos++;
                elem.style.left = pos + 'px';
            }
        }
    }
    // ボタンにクリックイベントリスナーを追加
    document.getElementById("startButton").addEventListener("click",
startAnimation);
</script>
</body>
</html>
```

「9.」を使った、アクセシビリティ

DOM（Document Object Model）を使ったアクセシビリティの向上は、ウェブページやアプリケーションをより多くの人々、特に障害を持つユーザーにとって使いやすくすることを意味します。アクセシビリティを考慮した DOM の操作には、以下のような要素が含まれます。

・意味のあるセマンティック HTML の使用

適切な HTML 要素（例：<header>, <nav>, <main>, <footer>）の使用は、スクリーンリーダーなどの支援技術がページの構造を正しく理解できるようにします。

・ARIA（Accessible Rich Internet Applications）の利用

ARIA は、スクリーンリーダーやその他の支援技術に対して、ウェブコンテンツやアプリケーションの役割、状態、プロパティをより明確に伝えるための技術です。例えば、aria-label, aria-hidden, role 属性などがあります。

・キーボードナビゲーションのサポート

JavaScript を使用して、キーボードだけでウェブサイトを操作できるようにします。これには、タブ順序の管理やフォーカス制御が含まれます。

・動的コンテンツのアクセシビリティの改善

AJAX や JavaScript で動的に更新されるコンテンツに対し、スクリーンリーダーが変更を検出して伝えるための対策を施します。

・視覚的なコントラストの強化

色覚障害を持つユーザーや視力の低いユーザーのために、十分なコントラストを提供するようにスタイルを調整します。

・エラーハンドリングとバリデーション

フォームのエラーが発生した場合、明確でわかりやすいフィードバックを提供し、スクリーンリーダーがエラーメッセージを読み上げられるようにします。

・動的要素のアクセシビリティ対応

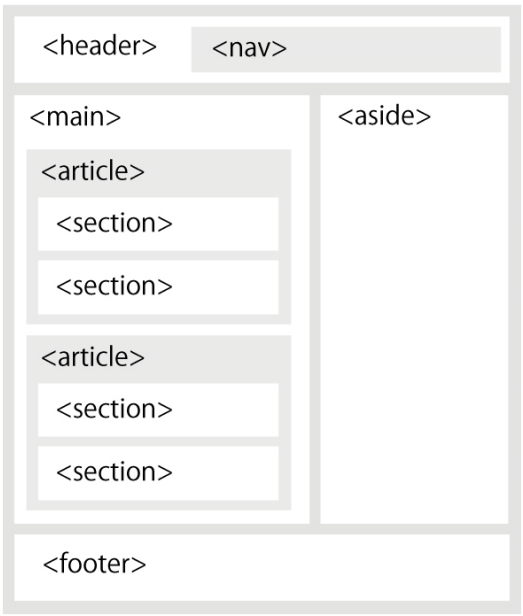
モーダルウィンドウ、ドロップダウンメニューなどの動的要素に、キーボードナビゲーションとスクリーンリーダー対応を実装します。

・言語属性の適切な設定

lang 属性を適切に設定することで、スクリーンリーダーが正しい言語でコンテンツを読み上げます。

■セマンティック html 要素

セマンティック HTML 要素は、その要素が持つ意味や内容の種類を明確に表す HTML 要素です。これらの要素を使用することで、ウェブページの構造がより明確になり、読みやすくなるだけでなく、検索エンジンの最適化（SEO）やスクリーンリーダーによるアクセシビリティの向上にも寄与します。以下は、よく使用されるセマンティック HTML 要素の例です。



<header>

ページやセクションのヘッダー部分に使用されます。通常、タイトル、ロゴ、ナビゲーションリンクなどが含まれます。

<nav>

ナビゲーションリンクのグループを示すために使用されます。主にメインメニュー、目次、索引などのナビゲーションに使用されます。

<main>

ページの主要なコンテンツを示すために使用されます。各ページには一つの<main>要素が含まれるべきです。

<section>

ページ内の一つのセクションやトピックを表します。通常、関連するコンテンツのグループを囲むために使用されます。

<article>

独立して完結しているコンテンツを示します。例えば、ブログの投稿、ニュース記事、フォーラムのポストなどに使用されます。

`<aside>`

メインコンテンツとは別の、関連する補足コンテンツを示します。サイドバー、広告、注釈などに使用されます。

`<footer>`

ページやセクションのフッター部分に使用されます。著者情報、著作権情報、関連ドキュメントへのリンクなどが含まれることが多いです。

`<figure>` と `<figcaption>`

画像、図表、コードスニペットなど、独立したコンテンツを示します。<figcaption>はその図表や画像のキャプションを提供します。

セマンティック HTML 要素を正しく使用することで、ウェブページの内容が構造化され、明確になります。これにより、ユーザーエージェントや検索エンジンはページの内容をより正確に解釈し、ユーザーに提供することが可能になります。また、アクセシビリティの観点からも重要であり、スクリーンリーダーを使用するユーザーにとって、ページの内容を理解しやすくなります。

■JavaScript DOM 操作のサンプルコード

JavaScript を使用して DOM を操作することで、Web ページの内容、構造、スタイルを動的に変更することができます。以下に、基本的な DOM 操作の例を示します。

1. 要素の選択と内容の変更

```
document.getElementById('demo').innerHTML = 'こんにちは、世界！';
```

このコードは、ID が'demo'の要素を見つけ、その内容を「こんにちは、世界！」に変更します。

2. 要素のスタイルの変更

```
document.getElementById('demo').style.color = 'red';
```

このコードは、ID が'demo'の要素のテキスト色を赤に変更します。

3. イベントリスナーの追加

```
document.getElementById('myButton').addEventListener('click',
function() {
    alert('ボタンがクリックされました！');
});
```

このコードは、ID が'myButton'のボタンにクリックイベントリスナーを追加し、ボタンがクリックされるとアラートを表示します。

4. 新しい HTML 要素の作成と追加

```
var newElement = document.createElement('p');
newElement.innerHTML = 'これは新しい段落です。';
document.body.appendChild(newElement);
```

このコードは、新しい<p>要素を作成し、その内容にテキストを追加してから、その要素をドキュメントの<body>に追加します。

これらの例は DOM を操作するための基本的な方法を示しています。JavaScript と DOM を組み合わせることで、より複雑でインタラクティブな Web ページを作成することが可能になります。

■定義

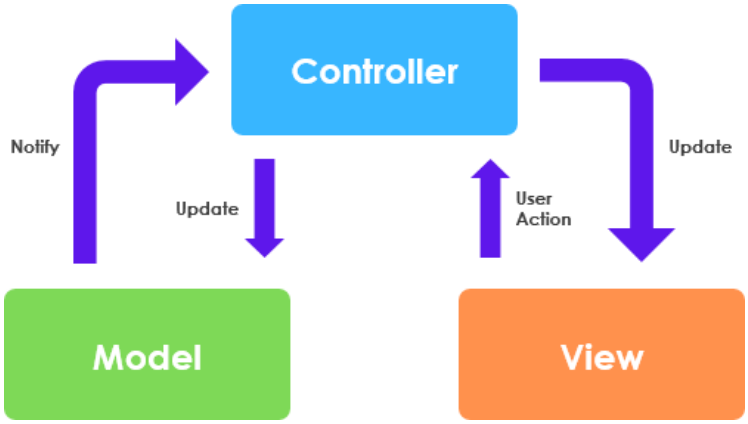
公式ドキュメントによると、「DOM(Document Object Model)」とは**HTML、あるいは XML ドキュメントに関連する API で、ドキュメントの構造や操作を定義したもの**を意味する。DOM を使うことで、私たちプログラマーは文書を作成したり、閲覧したり、要素や内容を追加・変更・削除したりできる。HTML や XML 文書にあるものはすべて DOM を使って操作できる。

> Document Object Model(DOM)は、HTML または XML 文書のためのプログラミング API です。ドキュメントの論理構造と、ドキュメントへのアクセスや操作の方法を定義しています。
> (中略)
> Document Object Model を用いることで、プログラマーは文書を作成・構築し、その構造を閲覧し、要素や内容を追加・変更・削除することができます。HTML や XML 文書にあるものはすべて、Document Object Model を使ってアクセス、変更、削除、追加することができます。特に、内部サブセットと外部サブセットの DOM インターフェースはまだ規定されていません。

DOM をより具体的に述べると MVC モデルのようなものである。MVC モデルは、MVC Framework - Introduction - Tutorialspoint の記事によると、以下のように定義されている。

> モデル・ビュー・コントローラー（MVC）は、アプリケーションをモデル、ビュー、コントローラーの 3 つの主要な論理コンポーネントに分離するアーキテクチャパターンです。これらのコンポーネントはそれぞれ、アプリケーションの特定の開発局面を処理するために構築されています。MVC は、拡張性のあるプロジェクトを作成するために、最も頻繁に使用される業界標準の Web 開発フレームワークの 1 つです。

要は、MVC(Model-View-Controller)モデルとはアプリケーションをモデル(Model)、ビュー(View)とコントローラ(Controller)の三つのロジックに分離するアーキテクチャ[1]を意味する。



ただし、MVC モデルはアプリケーションを Model、View、Controller の三つに分割するという定義があるだけで実態はない。プログラマーがその定義に従って構築し、その構築したものも同様に MVC モデルと呼ばれる。DOM も同じで実態はいない。定義されたとおりに実装すると、それも DOM と呼ばれる。

・文章を作成したり、閲覧したり、要素や内容を追加・変更・削除できる
・HTML や XML にあるものすべて DOM を使って操作できる
・MVC モデルと同様に、定義されたとおりに実装されたものも DOM と呼ばれる

＜Zenn 改＞