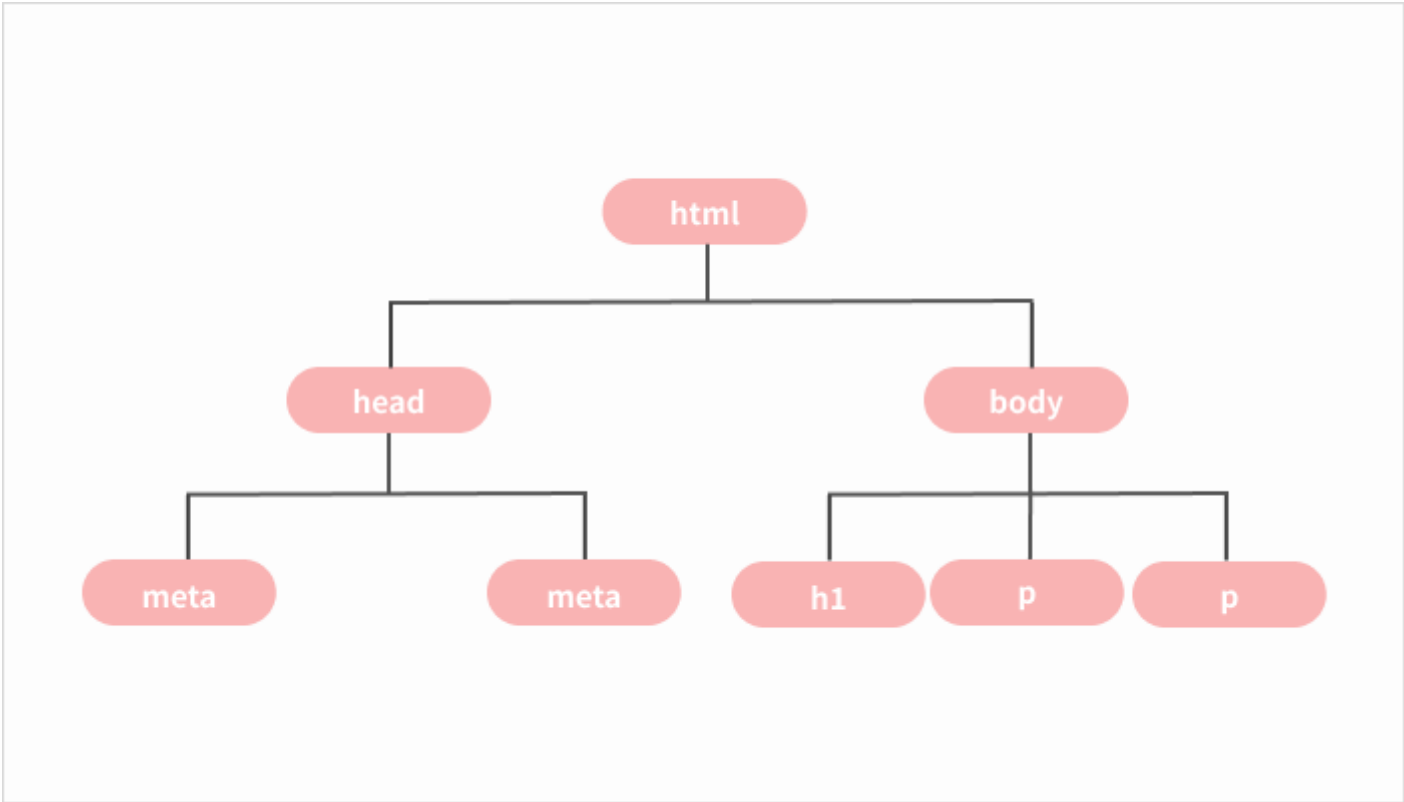




ブラウザはHTMLファイルを読み込むと、自動的に以下のような**DOMツリー**を生成します。

>



JavaScriptは各種メソッドを使って**ノード**にアクセスすることで、HTML要素の取得・作成・更新・削除などを行うことができます。これがDOM操作の仕組みです。

DOMツリーが生成されると `document` オブジェクトが使えるようになり、この `document` オブジェクトを使うことでさまざまなDOM操作を行えるようになります。

DOM操作の仕組みをまとめると、以下のとおりです。

1. ブラウザがHTMLファイルを読み込むと、自動的にDOMツリーを生成する
2. DOMツリーが生成されると、 `document` オブジェクトが使えるようになる
3. `document` オブジェクトを使うと、JavaScriptでDOM操作ができる

16.3 DOM操作でHTML要素を取得しよう

それでは、JavaScriptからDOM操作を行い、任意のHTML要素を取得する方法を学びましょう。

DOM操作を行うときはまず、「どのHTML要素に動きを加えるか」という「操作の対象」を取得しなければなりません。つまり、HTML要素を取得しないとDOM操作ができないということです。

前章では、 `document.HTML要素名` と記述してHTML要素を取得しました（例： `document.head` ）。しかし `document` オブジェクトのメソッドを使えば、取得するHTML要素をより細かく指定できます。

HTML要素を取得するための主なメソッドは以下のとおりです。このあと実際にコードを書くので、現時点ではイメージをつかむだけでOKです。

- `getElementById()` ：HTML要素を**id**で取得する
- `getElementsByClassName()` ：HTML要素を**class**で取得する
- `querySelector()` ：HTML要素を**CSSセレクタ**で取得する（**最初の1つ**）
- `querySelectorAll()` ：HTML要素を**CSSセレクタ**で取得する（**すべて**）

では順番に見ていきましょう。

getElementById() ：HTML要素をidで取得する



getElementById() は、指定したidを持つHTML要素を1つ取得するメソッドです。idは同じHTML内で1回しか設定できない固有の値なので、取得できるHTML要素も当然1つだけです。

実際に使ってみましょう。まずは dom.html を以下のように編集してください。

```
dom.html

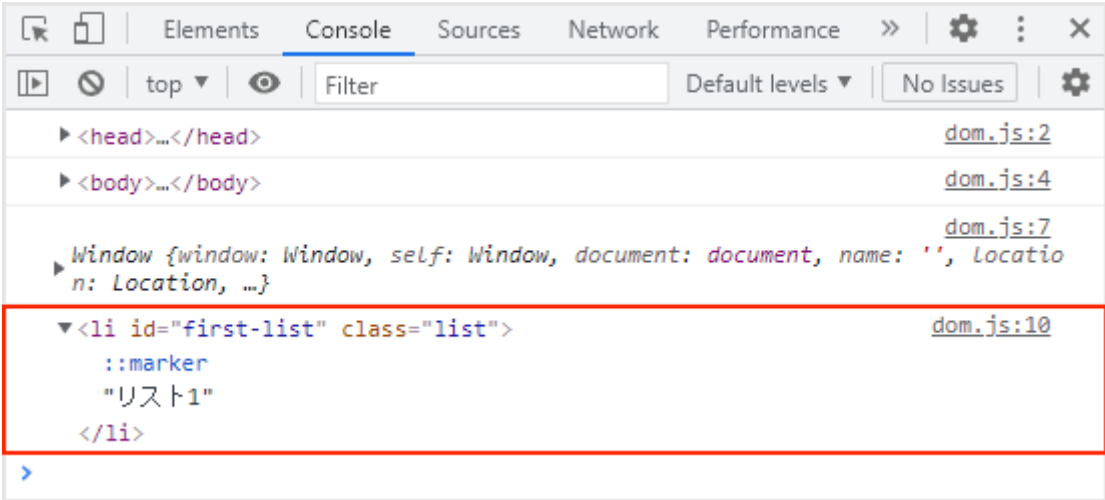
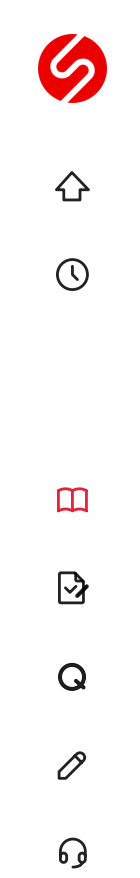
1  <!DOCTYPE html>
2  <html lang="ja">
3
4  <head>
5    <meta charset="UTF-8">
6    <title>DOM操作</title>
7  </head>
8
9  <body>
10 -   <h1>見出し</h1>
11 +   <h1 id="first-heading" class="heading">大見出し</h1>
12 +   <h2 id="second-heading" class="heading">中見出し</h2>
13 +   <ul>
14 +     <li id="first-list" class="list">リスト1</li>
15 +     <li id="second-list" class="list">リスト2</li>
16 +   </ul>
17   <script src="js/dom.js"></script>
18 </body>
19
20 </html>
21
```

このHTMLファイルの中から、 first-list というidを持つ1つ目のリストだけを取得してみましょう。 dom.js を以下のように編集してください。

```
dom.js

1  // head要素を取得し、中身を出力する
2  console.log(document.head);
3  // body要素を取得し、中身を出力する
4  console.log(document.body);
5
6  // windowオブジェクトの中身を出力する
7  console.log(window);
8
9 + // HTML要素をidで取得し、中身を出力する
10 + console.log(document.getElementById('first-list'));
11
```

では dom.html をブラウザで開き、デベロッパーツールのコンソールを確認してみましょう。以下のように、 first-list というidを持つ1つ目のリストだけが表示されていればOKです。



引数の値を変えて、他のHTML要素も取得してみてください。

getElementsByClassName() : HTML要素をclassで取得する

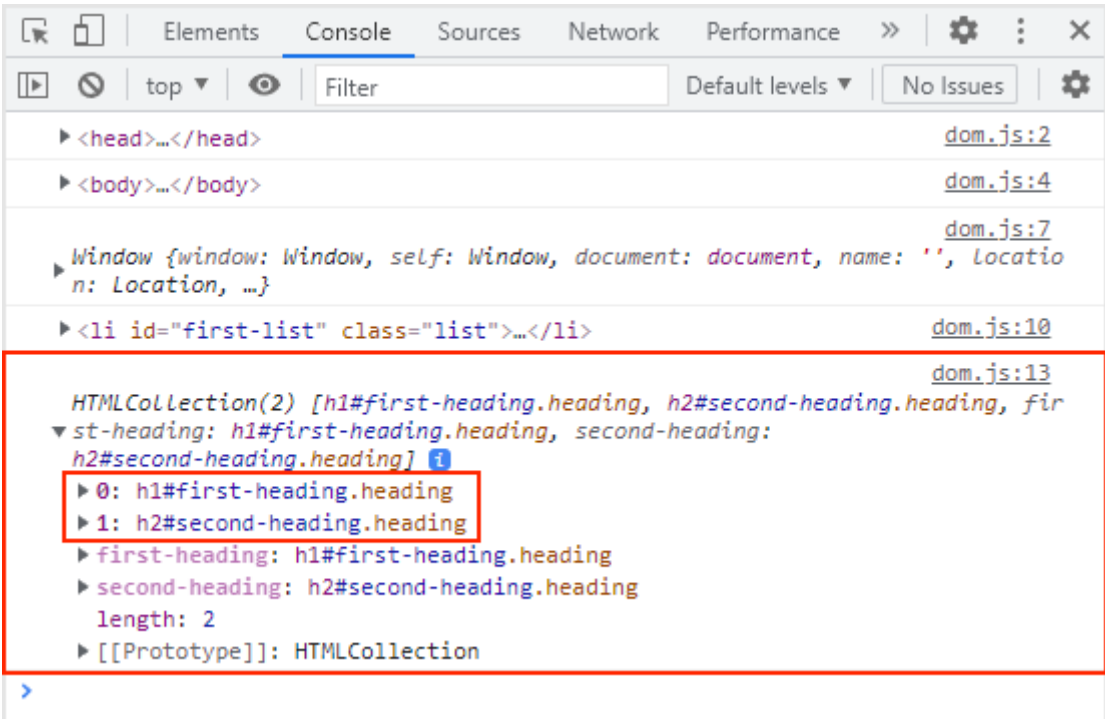
getElementsByClassName() は、指定したclassを持つHTML要素をすべて取得するメソッドです。idとは異なりclassは複数のHTML要素に設定できるので、そのすべてのHTML要素を取得します（そのため「getElement」ではなく「getElements」と複数形になっています）。

実際に使ってみましょう。今回は、heading というクラスを持つ2つのHTML要素（h1 と h2 ）を取得してみます。dom.js を以下のように編集してください。

dom.js

```
1 //===== 前略 =====
2
3 // HTML要素をidで取得し、中身を出力する
4 console.log(document.getElementById('first-list'));
5
6 + // HTML要素をclassで取得し、中身を出力する
7 + console.log(document.getElementsByClassName('heading'));
8
```

では dom.html をブラウザで開き、デベロッパーツールのコンソールを確認してみましょう。以下のように、heading というクラスを持つ2つのHTML要素（h1 と h2 ）が表示されていればOKです。



複数のHTML要素を1つずつ取得する方法





今回のように `getElementsByClassName()` メソッドを使って複数のHTML要素を取得した場合、 `HTMLCollection` という配列風のデータが戻り値として返ってきます。

なお、 `HTMLCollection` は厳密には配列ではないので「配列風」と表現していますが、細かく覚える必要はありません。「配列に似ているデータ」という認識でOKです。

この `HTMLCollection` からHTML要素を1つずつ取得するには、8章で学んだfor文を使って繰り返し処理を行えば簡単です。

```
1 for (カウンタ変数の初期値; 条件式; カウンタ変数の増減値) {
2   カウンタ変数が決まった値になるまで繰り返す処理
3 }
4
```

lengthプロパティ

ここで、今回新しく登場する `length` プロパティについて解説しておきます。 `length` プロパティは文字列の文字数や配列の要素数を取得するプロパティで、 `HTMLCollection` にも使うことができます。

今回は `HTMLCollection` の長さをfor文の条件式にすることで、 `HTMLCollection` の中に入っているHTML要素の数だけ繰り返し処理を行うことができます。

実際にやってみよう

では実際にやってみましょう。 `dom.js` を以下のように編集してください。

`dom.js`

```
1 //===== 前略 =====
2
3 // HTML要素をclassで取得し、中身を出力する
4 console.log(document.getElementsByClassName('heading'));
5
6 + // 複数のHTML要素を取得し、定数に代入する
7 + const headings = document.getElementsByClassName('heading');
8 +
9 + // 複数のHTML要素を1つずつ取得し、中身を出力する
10 + for (let i = 0; i < headings.length; i++) {
11 +   console.log(headings[i]);
12 + }
13
```

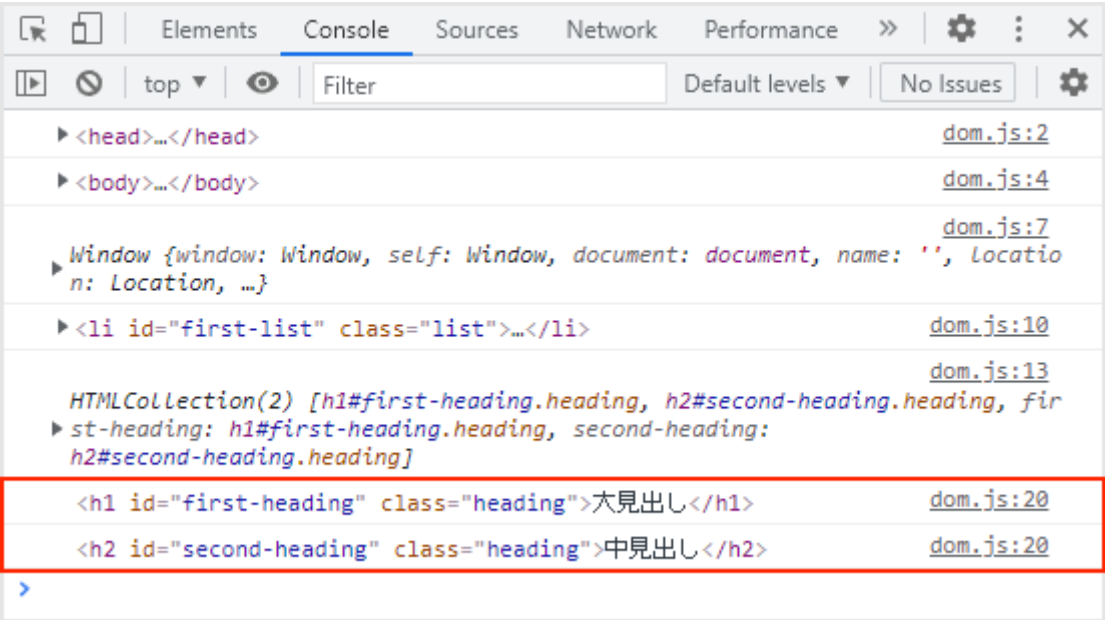
上記のコードでは `headings.length` の値が `2` なので、条件式は `i < 2` となり、カウンタ変数 `i` が `0` と `1` のときに処理が実行されます。つまり、以下の処理です。

- 1回目の処理： `console.log(headings[0]);`
- 2回目の処理： `console.log(headings[1]);`

このように、 `HTMLCollection` の中身を1つずつ取得できます。

では `dom.html` をブラウザで開き、デベロッパーツールのコンソールを確認してみましょう。以下のように、複数のHTML要素が1つずつ順番に表示されていればOKです。





querySelector() : HTML要素をCSSセレクタで取得する（最初の1つ）

querySelector() は、指定したCSSセレクタに**最初に合致したHTML要素**を取得するメソッドです。なお、CSSセレクタについて復習したい方は、以下の教材を参照してください。

- 『HTML/CSSの基礎を学ぼう11章 CSSのセレクタの概要を理解しよう（1）』

querySelector() メソッドの引数にはCSSセレクタを指定するので、以下のような書き方になります。

JSファイル（見本）

```
1 // 要素名で取得する例
2 document.querySelector('h1');
3 // id名で取得する例
4 document.querySelector('#second-heading');
5 // class名で取得する例
6 document.querySelector('.list');
7
```

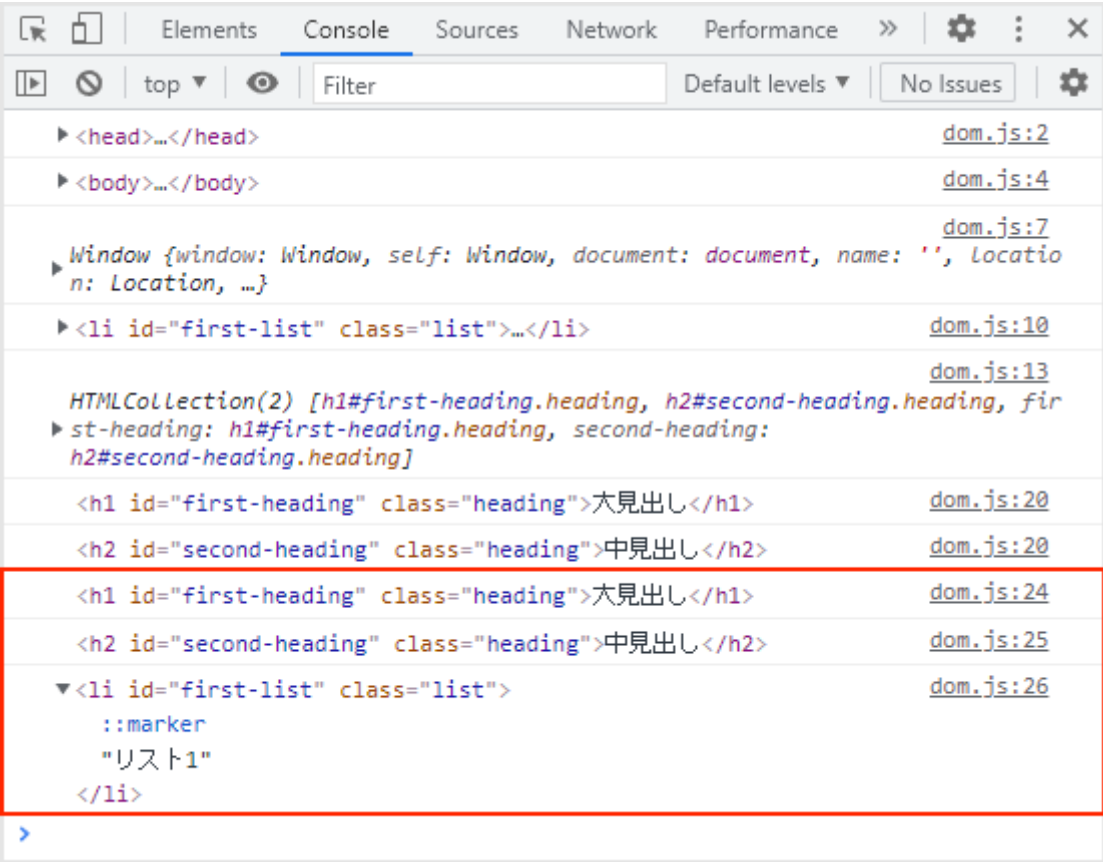
では実際に使ってみましょう。 dom.js を以下のように編集してください。

dom.js

```
1 //===== 前略 =====
2
3 // 複数のHTML要素を取得し、定数に代入する
4 const headings = document.getElementsByClassName('heading');
5
6 // 複数のHTML要素を1つずつ取得し、中身を出力する
7 for (let i = 0; i < headings.length; i++) {
8   console.log(headings[i]);
9 }
10
11 + // HTML要素をCSSセレクタで取得し、中身を出力する（最初の1つ）
12 + console.log(document.querySelector('h1'));
13 + console.log(document.querySelector('#second-heading'));
14 + console.log(document.querySelector('.list'));
15
```

続いて dom.html をブラウザで開き、デベロッパーツールのコンソールを確認してみましょう。以下のように、指定したCSSセレクタに該当するHTML要素を取得できていればOKです。





なお、`list` というクラスを持つHTML要素は2つあります。しかし、`querySelector()` は最初に合致したHTML要素を取得するメソッドなので、1つ目のHTML要素のみが表示されている点に注意してください。

querySelectorAll() : HTML要素をCSSセクタで取得する（すべて）

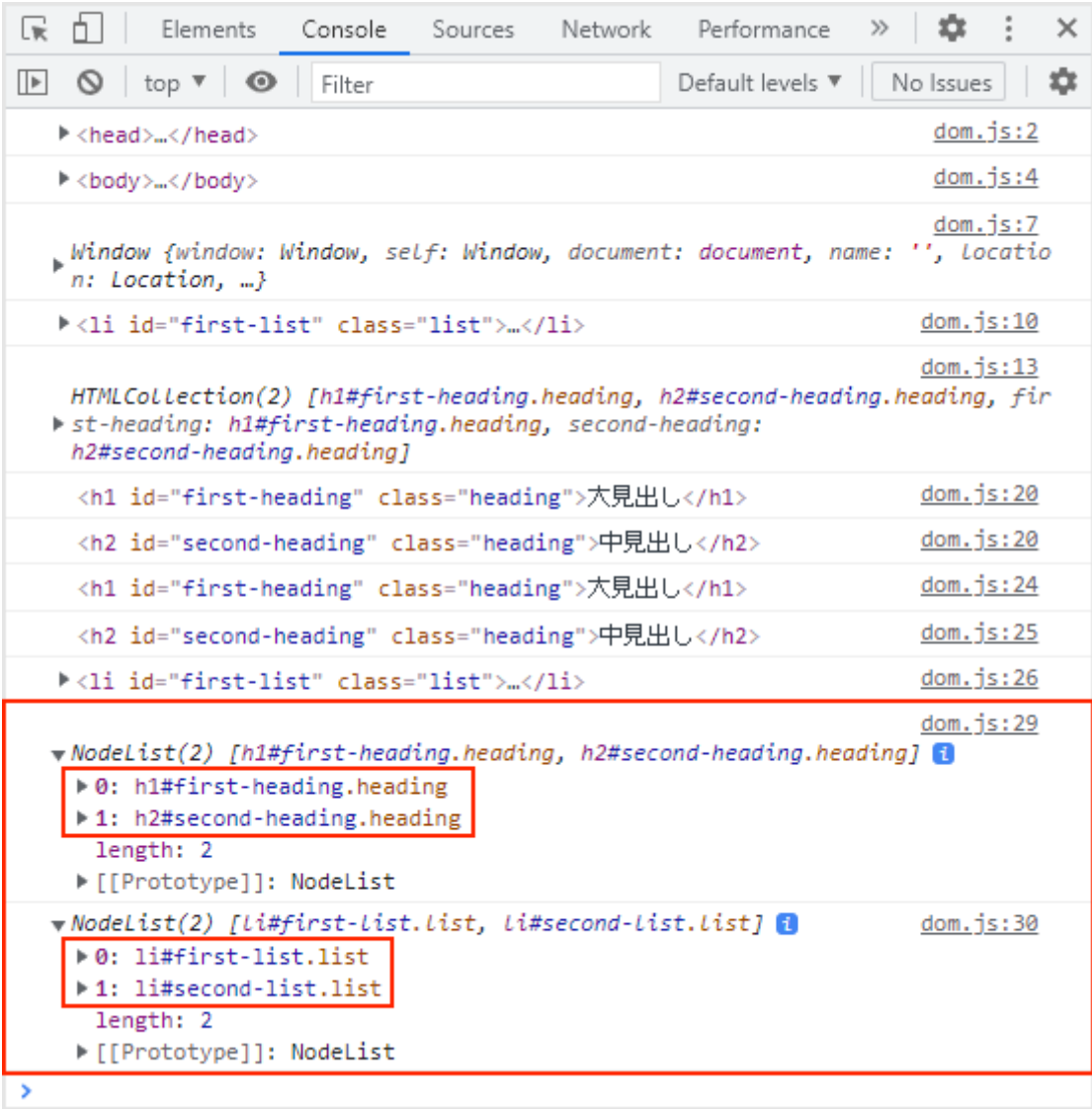
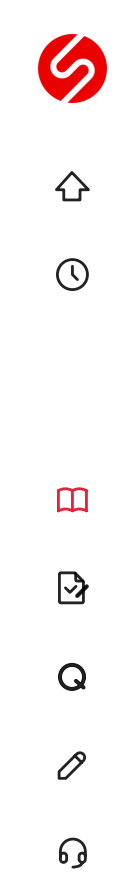
`querySelectorAll()` は、指定したCSSセクタに**合致したHTML要素すべて**を取得するメソッドです。

実際に使ってみましょう。`dom.js` を以下のように編集してください。

`dom.js`

```
1 //===== 前略 =====
2
3 // HTML要素をCSSセクタで取得し、中身出力する（最初の1つ）
4 console.log(document.querySelector('h1'));
5 console.log(document.querySelector('#second-heading'));
6 console.log(document.querySelector('.list'));
7
8 + // HTML要素をCSSセクタで取得し、中身出力する（すべて）
9 + console.log(document.querySelectorAll('.heading'));
10 + console.log(document.querySelectorAll('li'));
11
```

続いて `dom.html` をブラウザで開き、デベロッパーツールのコンソールを確認してみましょう。以下のように、指定したCSSセクタに該当するHTML要素をすべて取得できていればOKです。



なお、今回戻り値として返ってきた `NodeList` も、`HTMLCollection` と同じように配列風のデータです。 `querySelectorAll()` メソッドを使った場合は、この `NodeList` が戻り値として返ってきます。

`HTMLCollection` と同じようにfor文や `length` プロパティが使えるので、気になった方は先ほどと同じようにHTML要素を1つずつ取得してみてください。以下はサンプルコードです。

JSファイル（見本）

```
1 // 複数のHTML要素を取得し、定数に代入する
2 const cssHeadings = document.querySelectorAll('.heading');
3 const cssLists = document.querySelectorAll('li');
4
5 // 複数のHTML要素を1つずつ取得し、中身を出力する
6 for (let i = 0; i < cssHeadings.length; i++) {
7   console.log(cssHeadings[i]);
8 }
9 for (let i = 0; i < cssLists.length; i++) {
10  console.log(cssLists[i]);
11 }
12
```

16.4 DOM操作でHTML要素を作成しよう

続いて、JavaScriptからDOM操作を行い、新しくHTML要素を作成する方法を学びましょう。

現時点では、 `dom.html` をブラウザで開くと以下のように表示されます。







大見出し



中見出し

- リスト1
- リスト2

今回は、以下のようにリストを追加してみましょう。







大見出し

中見出し

- リスト1
- リスト2
- JavaScriptで新しく作成したリスト3

上記のようにリストを追加するには、以下のメソッドやプロパティを使います。

- createElement() メソッド：HTML要素を新しく作成する
- textContent / innerHTML プロパティ：HTML要素にテキストを追加する
- appendChild() メソッド：作成したHTML要素を子要素として末尾に追加する

順番に解説します。

createElement()メソッド

HTML要素を新しく作成するには createElement() メソッドを使います。引数には以下のように、作成したいHTML要素名を指定します。

JSファイル（見本）

```
1 // 新しくli要素を作成し、定数に代入する
2 const li = document.createElement('li');
3
```

textContent / innerHTMLプロパティ


作成したHTML要素にテキストを追加するには、textContent プロパティや innerHTML プロパティに値を代入します。どちらも文字列を追加する点では一緒ですが、innerHTML は通常の文字列に加えてHTMLタグも記述できます。

以下の例では、innerHTML を使ってリンク付きのテキストを追加しています。なお、以下は見本なので textContent プロパティと innerHTML プロパティの両方を記載していますが、実際にはどちらか1つのみを使います。

JSファイル（見本）







大見出し

中見出し

- リスト1
- リスト2
- JavaScriptで新しく作成したリスト3

https://terakoya.sejuku.net/programs/60/chapters/682

9/12



```
1 // 新しくli要素を作成し、定数に代入する
2 const li = document.createElement('li');
3
4 // 作成したli要素にテキストを追加する
5 li.textContent = 'JavaScriptで新しく作成したリスト3';
6 li.innerHTML = '<a href="#">JavaScriptで新しく作成したリスト3</a>';
7
```

上記のサンプルコードでは定数にJavaScriptで作成したHTML要素を代入しただけなので、ブラウザには表示されません。JavaScriptでHTML要素を作成した場合、ブラウザに表示するにはそのHTML要素を document オブジェクトに追加する必要があります。

appendChild()メソッド

HTML要素を document オブジェクトに追加するには、appendChild() メソッドを使います。appendChild() は、作成したHTML要素を子要素として末尾に追加するメソッドです。

3節で学んだHTML要素を取得するためのメソッドを使うことで、以下のように任意のHTML要素の末尾に追加できます。

JSファイル（見本）

```
1 // 新しくli要素を作成し、定数に代入する
2 const li = document.createElement('li');
3
4 // 作成したli要素にテキストを追加する
5 li.textContent = 'JavaScriptで新しく作成したリスト3';
6 li.innerHTML = '<a href="#">JavaScriptで新しく作成したリスト3</a>';
7
8 // ul要素の末尾にli要素を追加する
9 document.querySelector('ul').appendChild(li);
10
```

なお、appendChild() メソッドは**子要素として**末尾に追加するメソッドなので、上記のように ul 要素に使った場合、 ul 要素の子要素である li 要素の末尾に追加されます。

HTMLファイル（見本）

```
1 <!--===== 前略 =====>
2
3 <ul>
4   <li id="first-list" class="list">リスト1</li>
5   <li id="second-list" class="list">リスト2</li>
6   <!-- 以下の箇所にli要素が追加される -->
7   <li>JavaScriptで新しく作成したリスト3</li>
8 </ul>
9
10 <!--===== 後略 =====>
11
```

実際にやってみよう

では実際にやってみましょう。dom.js を以下のように編集してください。今回は文字列のみ追加するので、textContent プロパティを使います。



🔗

🏠

🕒

📖

📄

🔍

✎

🔊

dom.js

1 //===== 前略 =====

2

3 // HTML要素をCSSセクタで取得し、中身を出力する（すべて）

4 console.log(document.querySelectorAll('.heading'));

5 console.log(document.querySelectorAll('li'));

6

7 + // 新しくli要素を作成し、定数に代入する

8 + const li = document.createElement('li');

9 +

10 + // 作成したli要素にテキストを追加する

11 + li.textContent = 'JavaScriptで新しく作成したリスト3';

12 +

13 + // ul要素の末尾にli要素を追加する

14 + document.querySelector('ul').appendChild(li);

15

続いて、 dom.html をブラウザで開きましょう。以下のように、新しくリストが追加されていればOKです。

大見出し

中見出し

- リスト1
- リスト2
- JavaScriptで新しく作成したリスト3

まとめ

本章では以下の内容を学習しました。

メソッド	処理の内容
getElementById()	HTML要素をidで取得する
getElementsByClassName()	HTML要素をclassで取得する
querySelector()	HTML要素をCSSセクタで取得する（最初の1つ）
querySelectorAll()	HTML要素をCSSセクタで取得する（すべて）
createElement()	HTML要素を新しく作成する
appendChild()	HTML要素を子要素として末尾に追加する

プロパティ	値
length	文字列や配列の長さ
textContent	HTML要素内のテキスト
innerHTML	HTML要素内のテキスト（HTMLタグも可）

>

https://terakoya.sejuku.net/programs/60/chapters/682

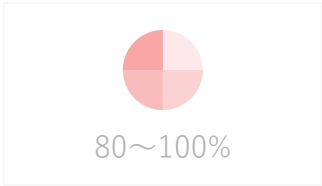
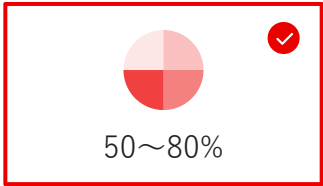
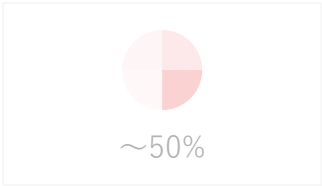
11/12



次章では、イベント処理について学びます。

理解度を選択して次に進みましょう

ボタンを押していただくと次の章に進むことができます



最後に確認テストを行いましょう

下のボタンを押すとテストが始まります。

教材をみなおす

テストをはじめる

前に戻る

20 / 26 ページ

次に進む

◀ 一覧に戻る

改善点のご指摘、誤字脱字、その他ご要望はこちらからご連絡ください。

© SAMURAI Inc.

[利用規約](#) [法人会員利用規約](#) [プライバシーポリシー](#) [運営会社](#)