ワンポイント 返し処理の中断(break文)

「汚れが消えるまで拭く」プログラムを考えたとき、もし汚れが強力で、いくら吹いても落とすことができないとしたら、このプログラムは無限ループになってしまいますよね。こうしたケースも含めて、繰り返し処理を途中で中断したいときに便利なのが「break文」です。「break文」は以前、switch文のレッスンで処理を中断するために使いましたが、while文やfor文などの繰り返し処理の中断にも使うことができます。以下の例では「汚れが消えるまでこする」プログラムが無限ループにならないよう「効果がない」と判断された時点でbreak文を使い、繰り返し処理を中断しています。

```
while (汚れが消えていない) {
  拭く;
  if (効果がない) {
    break; ····· 繰り返し処理を中断
  }
}
```

ワンポイント 繰り返しをスキップする(continue文)

繰り返し処理全体は中断したくないけれど、今回の処理はスキップしたい、という場合に使用するのがcontinue文です。
今回も「汚れが消えるまで拭く」プログラムを例に考えてみましょう。このプログラムを実行したところ、連続して拭く処理を行うと、摩擦熱で素材が傷んでしまうことがわかったとします。そこでプログラムを改善して、摩擦で熱いときには休憩を入れて処理をスキップするようにしました。continue文をうまく使うことで、繰り返し処理の条件をさらに細かく設定することができます。

```
while (汚れが消えていない) {
  if (摩擦で熱い) {
    10秒休む;
    continue; ····· while文の先頭にジャンプ
  }
  拭く;
}
```


Chapter

6

HTML/CSSを操作する方法を学ぼう

JavaScriptを使ってHTMLやCSSを操作してWebページのコンテンツを変更するには、「DOM（ドム）」を利用します。DOMを理解するために「オブジェクト」から順に学びましょう。

Lesson 33 [オブジェクトの概要]

オブジェクトとは何かを知りましょう

このレッスンのポイント

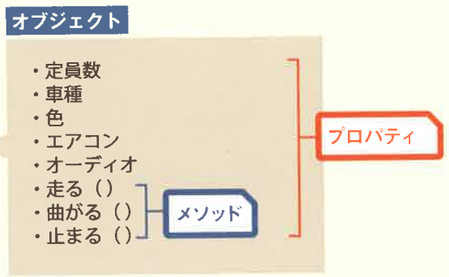
HTMLやCSSを操作するには「オブジェクト」という概念の理解が必要です。このレッスンでは「オブジェクト」の意味と、一緒に使われることの多い「プロパティ」「メソッド」という言葉を理解できるようにしましょう。

オブジェクトとは？

「オブジェクト」とは、変数や関数などのデータを扱いやすくまとめたものです。

例えば「車」に関するプログラムを作りたいとき、さまざまなデータをバラバラに管理するのは大変ですよね。そんなときに便利なのがオブジェクトです。車には「定員数」「車種」「色」などの情報や、「エアコン」「オーディオ」などの構成物、「走る」「曲がる」「止まる」といった機能があります。こうした車に関するデータをまとめて「車オブジェクト」というものを作ることができます。オブジェクトの持つデータを、そのオブジェクトの「プロパティ（所有物や特性）」といい、プロパティのうち、オブジェクトに対する操作を記述した関数を「メソッド」といいます。

▶車オブジェクト



オブジェクトを利用する

せっかくオブジェクトを作っても、実際に利用できなければ意味がありません。

先ほどの「車オブジェクト」は、自分に関連するデータ（プロパティ）を持っていて、その中には「走る」「曲がる」「止まる」といった操作（メソッド）も含まれています。

こうしたオブジェクトの持つプロパティを利用するには「オブジェクト名.プロパティ名」という形でドット記号「.」でつないで記述します。

例えば「車オブジェクト」に右に曲がってほしいときには「車.曲がる(右)」という具合にプログラムを書くことができます。

▶オブジェクトとプロパティの構文

```
element.innerHTML
```

オブジェクト名　プロパティ名

▶プロパティの書き方のイメージ

```
//_右に曲がってほしいとき
車.曲がる(右);

//_定員数を教えてほしいとき
車.定員数;
```

実際のオブジェクトやプロパティ、メソッドの名前は英語ですが、ここでは書き方のイメージをつかんでください。

オブジェクトの中のオブジェクト

車オブジェクトのプロパティには「エアコン」や「オーディオ」が含まれていますが、これらもそれぞれ操作できるオブジェクトです。オブジェクトの中にオブジェクトがある場合、例えば、エアコンオブジェクトの持つ「温度調整()」というメソッドを利用するには、「車.エアコン.温度調整()」という具合に書きます。

▶車に付いているエアコンの温度を28度にしたい場合は？

```
車.エアコン.温度調整(28);
```

Lesson 34 [windowオブジェクト]

Webページとオブジェクトの関係について知りましょう

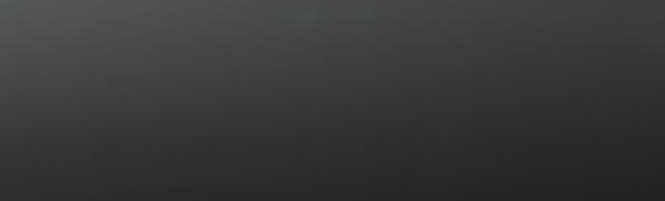
このレッスンのポイント

JavaScriptでは、ブラウザに関するあらゆるデータをまとめた「windowオブジェクト」がはじめから用意されています。windowオブジェクトを利用すると、現在表示しているWebページのHTMLやCSSの情報を取得したり、操作したりすることができます。

windowオブジェクトがすべての起点になる

JavaScriptでは、ブラウザのウィンドウ自体を表す「windowオブジェクト」がはじめから用意されており、windowオブジェクトを使ってブラウザの機能やWebページのデータにアクセスしたり、操作を行ったりすることができるようになっています。windowオブジェクトのプロパティには、Webページの情報をまとめた「documentオブジェクト」や、コンソールを表す「consoleオブジェクト」など、ブラウザを構成するあらゆるデータが含まれています。

▶windowオブジェクトとブラウザの対応



JavaScriptで利用できる多くの機能がwindowオブジェクトによって提供されています。

「window.」は省略できる

windowオブジェクトは、他のオブジェクトと同様に「オブジェクト名.プロパティ」の形で利用することができますが、常に利用されるオブジェクトなので「window.」を省略可能になっています。実は、過去のレッスンで利用した「console.log()」や「alert()」などの関数も、windowオブジェクトから提供されているものなんです。

▶「window」は省略できるので、どちらも同じ結果になる

```
console.log('Hello_World');
window.console.log('Hello_World');
```

「window.」は省略できることを覚えておきましょう。

Webページもオブジェクトの集まり

Webページに関する情報は、「documentオブジェクト」にまとめられています。documentオブジェクトのプロパティには、Webページを構成する要素を表す「elementオブジェクト」や、それらを操作するメソッドが含まれています。JavaScriptでは、このdocumentオブジェクトを利用することで、WebページのHTML/CSSを自由に操作することができます。

▶ブラウザとWebページを表すオブジェクト

表示中のWebページに関するデータは、documentオブジェクトからアクセスして、操作することができます。

Lesson 35 [DOM操作:内容の書き替え]

HTMLの要素の内容を変更してみましょう

このレッスンのポイント

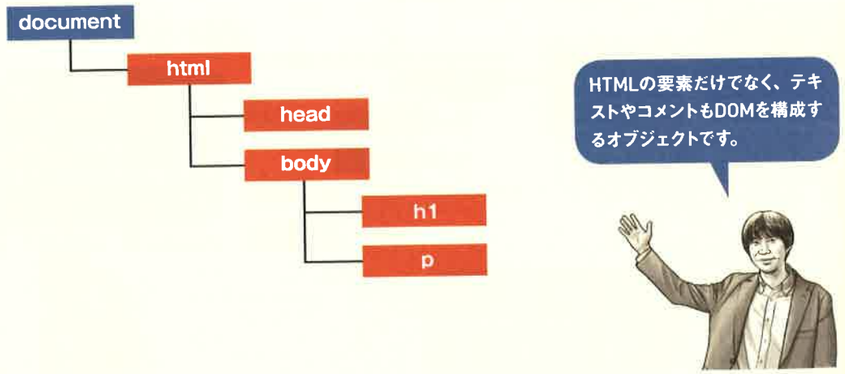
ブラウザで表示しているHTML文書の情報は、windowオブジェクトのプロパティであるdocumentオブジェクトから利用することができます。このレッスンでは、documentオブジェクトからHTML要素を取り出して、実際にその内容を変更してみましょう。

HTMLを操作するときは「DOM」という仕組みを利用する

documentオブジェクトのプロパティには、ブラウザで表示しているHTML文書の情報と、それらを利用するためのさまざまなメソッドが用意されています。このように、オブジェクトを通じてHTML文書にアクセスするための仕組みを、専門用語で「DOM(Document Object Model)」といいます。

DOMはHTML文書の構造を、オブジェクトをツリー状につなげた構造で表したものです。JavaScriptでHTML文書を操作することを「DOM(ドム)を操作する」ともいうので、あわせて覚えておくといいでしょう。

▶ DOMの構造



特定のIDを持つHTML要素を取得する

HTML要素の内容を変更するには、まず変更したいHTML要素をオブジェクトとして取得する必要があります。特定の要素をオブジェクトとして取得するにはgetElementByIdメソッドが便利です。このメソッドは名前の通り、HTML要素が持つid属性の値を手がかりに、HTML要素を取得します。

▶ getElementByIdメソッド

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

element: オブジェクトを入れる変数
'sample': id属性の値

▶ id="sample"の要素を取得する例

```
<!-- 取得する要素 -->
<div id="sample">サンプル</div>
```

HTML

```
//id="sample"の要素を取得して変数elementに代入
var element = document.getElementById('sample');
```

JavaScript

HTML要素の内容を書き替える

取得したHTML要素の内容は「innerHTML」というプロパティで確認することができます。内容を書き替えるには、この「innerHTML」の値を上書きして変更すればOKです。内容は文字列で、HTMLと同じ形式で記述できます。

▶ innerHTMLプロパティ

```
element.innerHTML = 'サンプル';
```

element: HTML要素
'サンプル': 書き替えたい内容

▶ id="sample"の要素の内容を「 こんにちは 」に書き替える場合

```
var element = document.getElementById('sample');
element.innerHTML = '<p>こんにちは</p>';
```

NEXT PAGE

HTMLを書き替えてみる

1 HTMLファイルを編集する　06/change/practice/index.html

まずは、書き替えるHTML要素を準備しましょう。このレッスンのindex.htmlをBracketsで開いて、以下のコードを記述して上書き保存してください。bodyの開始タグの下にdiv要素を追記して、後からJavaScriptで操作できるよう、idを「"practice"」と指定しておきましょう❶。このファイルをブラウザで開くと「練習」と表示されます。

```
008 <body>
009 ␣␣<div␣id="practice">練習</div>
010 ␣␣<script␣src="js/app.js"></script>
011 </body>
```

❶ div要素を追加

練習

小さな字で「練習」と表示されている

2 JavaScriptファイルを編集する　06/change/practice/js/app.js

次に、HTML要素の内容を書き替えるプログラムを記述していきましょう。このレッスンのapp.jsファイルをBracketsで開いて、以下のプログラムを記述してください。1行目で、先ほどHTMLファイルに追記したdiv要素を、id属性の値を頼りに取得しています❶。2行目で、取得した要素の内容を示すinnerHTMLプロパティを書き替えています❷。

```
001 var␣practice␣=␣document.getElementById('practice');
002 practice.innerHTML␣=␣'<h1>れんしゅう</h1>';
```

❶ 要素を取得
❷ 内容を書き替え

3 プログラムが完成した

プログラムが完成したら上書き保存して、index.htmlを再読み込みしてみましょう。読み込み時にJavaScriptが実行された結果、「練習」という文字ではなく、h1要素の「れんしゅう」が表示されます。

れんしゅう

JavaScriptで文字が書き替えられた

Point　innerHTMLにHTMLタグを代入すると要素になる

<h1>のようなHTMLタグを含む文字列をinnerHTMLに代入すると、ブラウザがタグを解釈して新しいHTML要素が作られます。そのまま「<h1>」と表示させたい場合は、innerTextプロパティを使用します。

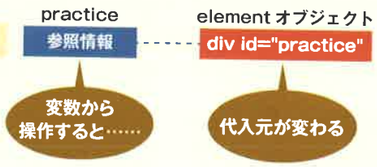
ワンポイント　オブジェクトの参照渡し

鋭い人は、HTML要素を変数から操作できることに疑問を持ったかもしれません。通常、変数に代入された値は「コピー」になるので、それを変更しても代入元に影響することはありません。しかし、HTML要素の場合、変数に代入された値を変更すると、コピー元のHTML要素も変更されました。

実はデータ型が「オブジェクト」のデータを変数に代入したときだけは、データがコピーされるのではなく、元のデータを参照するための情報が変数に記憶されます。そして、変数を使用して操作すると、元のオブジェクトが変更されます。

▶ 変数から代入元のオブジェクトを参照する

```
var practice = document.getElementById('practice');
```



Lesson 36 [DOM操作:要素へのアクセス]

要素を自在に取得できるようになりましょう

このレッスンのポイント

要素の内容は変更できましたか？ 先のLesson 35ではid属性を元に要素を取得しましたが、特定のタグやclass名で要素を取得したいこともあると思います。このレッスンでは、そうした要素の取得方法を学び、要素を自在に取得できるようになりましょう。

CSSセレクタで要素を取得する

CSSでは装飾する要素を「セレクタ」を使って指定することができます。
JavaScriptで取得する要素を指定する場合も、このCSSセレクタを使用することができます。CSSセレクタで1つの要素を取得するにはquerySelectorメソッドを使います。HTML文書の先頭から順に探しはじめて、引数に指定したセレクタに最初にマッチした要素が取得されます。

▶ querySelectorメソッド

```
document.querySelector('.about');
```

CSSセレクタ

▶ クラス名と要素名を手がかりにして取得

```
<section class="about">
  <h2>このサイトについて</h2>
  <p>このサイトは...です。</p>
</section>
```

```
var element = document.querySelector('.about h2');
```

……h2要素を取得

ワンポイント 複数の要素を取得・操作する

JavaScriptには同時に複数の要素を取得するメソッドも用意されています。少し難しい内容になるので、コラムとして紹介します。
複数の要素を取得するメソッドとして、例えばquerySelectorAllメソッドを利用すれば、引数に指定したCSSセレクタにマッチするすべての要素を取得することができます。
このとき戻り値が、マッチした要素をまとめた「NodeListオブジェクト」になるので、要素を操作する際は、事前にNodeListオブジェクトから取り出す必要があります。
NodeListオブジェクトから要素を取り出すには、オブジェクト名の後に[ ]（角カッコ）を付けて、取り出したい要素の番号を指定します。番号は、0を先頭に、要素が見つかった順で振られています。
取得したすべての要素を操作するには、for文などの繰り返し処理を用います。
NodeListオブジェクトのプロパティ「length」の値を見ると、含まれている要素数を確認することができます。要素の番号がlengthの値未満になるようにfor文を記述することで、取得したすべての要素に処理を行うことができます。

▶ CSSセレクタが="a.button"の要素をすべて取得

```
var nodeList = document.querySelectorAll('a.button');
```

▶ 最初にマッチした要素を取得

```
var firstElement = nodeList[0];
```

▶ 取得したすべての要素をコンソールに表示

```
for (var i = 0; i < nodeList.length; i++) {
  console.log(nodeList[i]);
}
```

▶ 要素の取得に用いられるメソッド

メソッド	引数	戻り値
getElementById()	id属性値	Elementオブジェクト
querySelector()	CSSセレクタ	Elementオブジェクト
querySelectorAll()	CSSセレクタ	NodeListオブジェクト
getElementsByClassName()	クラス属性値	NodeListオブジェクト
getElementsByName()	name属性値	NodeListオブジェクト
getElementsByTagName()	タグ名	NodeListオブジェクト

Lesson 37 [DOM操作:CSSの変更]

要素のスタイルを変更してみましょう

このレッスンのポイント

Webページで要素の見た目を整える場合はCSSを利用しますが、JavaScriptでもstyleプロパティを使って変更できます。ここでは文字色や文字サイズなどの基本的な変更のみを説明しますが、CSSの知識があればもっと複雑なスタイル変更も可能です。

要素のスタイルを変更する

JavaScriptから要素のスタイルを変更したい場合、各要素が持つstyleプロパティを利用します。styleプロパティにはCSSの情報を管理するオブジェクトが代入されており、そのオブジェクトがCSSプロパティとほぼ同名のプロパティを持っています。それらに代入することで、CSSプロパティを上書きすることができます。

▶ styleプロパティ

```
element.style.color = '#FF0000';
```

element: 要素　color: CSSプロパティ名　'#FF0000': プロパティ値

▶ id="sample"のHTML要素の文字色を「#FF0000」に変更

```
var element = document.getElementById('sample');
element.style.color = '#FF0000';
```

JavaScriptではプロパティ名にハイフン「-」が使用できないので、「backgroud-color」のようなハイフンが入るCSSプロパティの場合は名前が変更されています。

要素のスタイルを変更してみよう

1 JavaScriptファイルを編集する 06/change/practice/js/app.js

前回のレッスンで使用したapp.jsを編集して、テキストと背景色のスタイルを変更してみましょう。

app.jsファイルをBracketsで開いて、要素のスタイルを変更する文を追加します❶。

```
001 var practice = document.getElementById('practice');
002 practice.innerHTML = '<h1>れんしゅう</h1>';
003 practice.style.backgroundColor = '#999999';
004 practice.style.fontSize = '30px';
005 practice.style.color = '#FFFFFF';
```

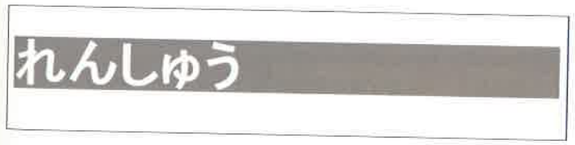
❶ スタイルを変更

Point ハイフン「-」の書き替え

ハイフンを含むCSSプロパティは、JavaScriptではハイフンを外してその後の単語の頭文字を大文字にします。「backgroud-color」は「backgroudColor」に、「font-size」は「fontSize」になります。

2 プログラムが完成した

プログラムが完成したら、内容を上書き保存して、index.htmlをブラウザで開いて動作を確認しましょう。文字サイズが30pxになり、背景がグレーの白抜き文字になります。



「#999999」は濃いグレー
「#ffffff」は白を意味します。

Lesson 38 [DOM操作:要素の追加]

要素を追加してみましょう

このレッスンのポイント

これまでは、すでに存在する要素の内容やスタイルを変更してきましたが、JavaScriptでは新たに要素を作って追加することもできます。このレッスンでは、要素を追加するcreateElementメソッドやinsertBeforeメソッドの使い方を学んでいきましょう。

要素を作って属性と内容を設定する

HTMLの要素は「タグ」と「属性」と「内容」の3つで構成されていましたね。

まず、タグだけが指定された要素を作るには、createElementメソッドを使い、引数に作りたい要素のタグ名を指定します。次に属性を指定するためには、setAttributeメソッドを使用します。引数には、設定したい属性名と、その値を指定します。最後に、内容を設定するには、Lesson 35で学んだinnerHTMLプロパティを使用します。

▶ createElementメソッド

```
var element = createElement('h1');
```

変数　タグ名

▶ setAttributeメソッド

```
element.setAttribute('id','first');
```

要素　属性名　属性値

新しい要素をHTML文書に追加する

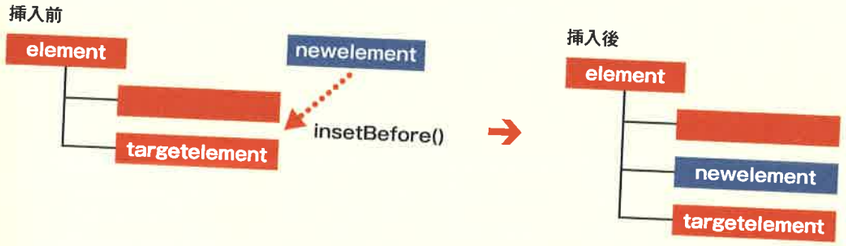
createElementメソッドで作成した要素は、まだどこにも所属していない宙ぶらりんの状態です。Webページに表示するには、要素をHTML文書に追加する必要があります。

要素を追加するにはinsertBeforeメソッドを使用します。要素は親要素の子になり、第2引数で指定したターゲット要素の前に追加されます。最後の子要素にしたい場合は第2引数を「null」にします。

▶ insertBeforeメソッド

```
element.insertBefore(newelement, targetelement)
```

親要素　追加する要素　ターゲット要素



nullは「オブジェクトの値が存在しないこと」を表す特殊な値です。関数／メソッドの引数にnullを指定した場合、「その引数に指定するものはないよ」と伝えていることになります。

ワンポイント Webページを構成する最も基本的な単位「ノード(Node)」とは？

DOMの観点で、Webページを構成する最も基本的なオブジェクトを「ノード：Node」といいます。要素や、属性、内容となるテキストなどもノードです。本書で「要素」として説明している部分は、書籍によっては「ノード」と説明している場合もありますが「要素はノードの一種なので、どちらの説明も正しい」と覚えておけば、混乱なく理解することができます。あわせて覚えておくといいでしょう。

要素を追加してみよう

1 JavaScriptファイルを編集する

06/change/practice/js/app.js

前回のレッスンで使用したapp.jsを編集して、新しい要素を追加してみましょう。先に取得した「practice」の要素の子要素として、新たに作成した要素を追加します。

まずdiv要素を作成して、変数firstに代入しています❶。

次に、内容や属性を設定します❷。最後にdiv要素をHTML文書に追加します❸。ここではinsertBeforeメソッドの第2引数を「null」にしているので、親要素の最後の子要素になります。

```
001 var practice = document.getElementById('practice');
002 practice.innerHTML = '<h1>れんしゅう</h1>';
003 practice.style.backgroundColor = '#999999';
004 practice.style.fontSize = '30px';
005 practice.style.color = '#FFFFFF';
006
007 //要素を追加します
008 var first = document.createElement('div');
009 first.setAttribute('id','first');
010 first.innerHTML = '<p>要素を追加</p>';
011 practice.insertBefore(first, null);
```

1 新しいdiv要素を作成
2 属性と内容を設定
3 要素をHTMLに追加

2 さらに要素を追加する

追加した要素の前にさらに要素を追加してみましょう。新しい要素を作成して属性と内容を設定します❶。insertBeforeメソッドの第2引数に先ほど追加した「first」を指定して、「first」の前に要素が追加されるようにします❷。

```
007 //要素を追加します
008 var first = document.createElement('div');
009 first.setAttribute('id','first');
010 first.innerHTML = '<p>要素を追加</p>';
011 practice.insertBefore(first, null);
012
013 //さらに要素を追加します
014 var second = document.createElement('div');
015 second.setAttribute('id','second');
016 second.innerHTML = '<p>さらに要素を追加</p>';
017 practice.insertBefore(second, first);
```

1 要素を作成
2 要素を追加

3 プログラムが完成した

プログラムが完成したら、内容を上書き保存して、index.htmlをブラウザで開いて動作を確認しましょう。要素が2つ追加されていれば成功です。

要素は追加できましたか？　CSSの変更や要素の追加を利用すれば、ユーザーの操作に合わせた画面の表示切り替えもできるようになりますね。

Lesson 39 [DOM操作：要素の削除]

要素を削除してみましょう

このレッスンのポイント

要素の変更、追加ときたら、最後は「削除」ですね。このレッスンを終えれば、HTML/CSSの基本操作をひと通りできるようになります。この章で学んだことを整理しながら、プログラムを記述していきましょう。HTML/CSSの操作は、この後の章でより実践的に利用していきます。

要素を削除するにはまず親要素を探す

要素を削除するには、まず削除したい要素の親要素を取得する必要があります。親要素のオブジェクトを取得するにはparentElementプロパティを使用します。

親要素のオブジェクトが取得できたら、子要素を削除することができるremoveChildメソッドを利用します。引数には、削除したい要素のオブジェクトを指定します。

▶ parentElementプロパティ

```
var parent = targetelement.parentElement
```

parent：親要素を代入する変数　targetelement：子要素

▶ removeChildメソッド

```
parent.removeChild(targetelement);
```

parent：親要素　targetelement：削除したい要素

削除したい要素の親要素がわからない場合は、parentElementプロパティから取得することができます

要素を削除してみよう

1 JavaScriptファイルを編集する

06/change/practice/js/app.js

Lesson 38で使用したapp.jsを編集して、追加した要素を削除してみましょう。app.jsファイルをBracketsで開いて、以下の記述を追加してください。

今回は、最初に追加した「first」の要素を削除してみます。

要素を削除するには、削除したい要素と、その要素の親要素の情報が必要です。今回は「first」の親要素が「practice」とわかっていますが、親要素がわからない場合も調べられるように、練習も兼ねて「first」から親要素の情報を取得します❶。親要素の情報が取得できたら、親要素の持つremoveChildメソッドで要素を削除します❷。

```
017 practice.insertBefore(second, first);
018
019 // 要素を削除します
020 var parent = first.parentElement;   ← 1 親要素を取得
021 parent.removeChild(first);         ← 2 要素を削除
```

2 プログラムが完成した

プログラムが完成したら、内容を上書き保存して、index.htmlをブラウザで開いて動作を確認しましょう。Lesson 38で追加したはずの「要素を追加」という文字が消えています。

れんしゅう

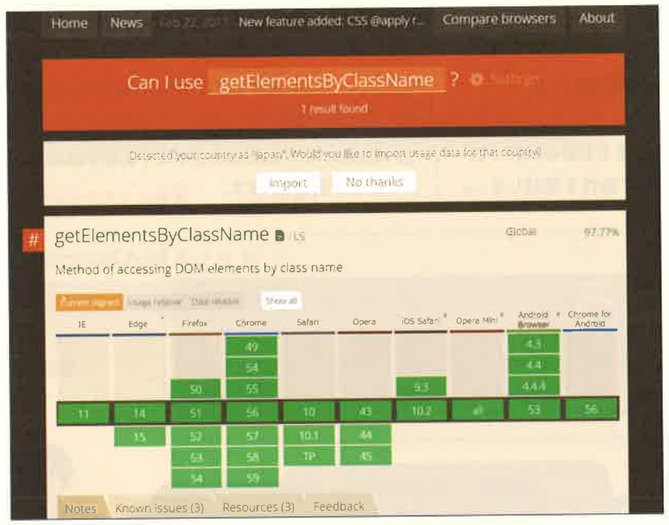
さらに要素を追加

お疲れさまでした。
これで、HTML/CSS操作の基本は終了です。

ワンポイント ブラウザごとにDOMに微妙な違いがある

実は、DOMはJavaScriptの言語仕様として決められたものではなく、ブラウザによって提供される仕組みです。そのため、ブラウザによって一部のメソッドやプロパティが利用できなかったり、挙動が違ったりする場合があります。本書では、Chromeをはじめとした主要なブラウザで利用できるものを厳選して学んでいきます。
そのため、HTML/CSSを操作するプログラムを書くと、ブラウザによってはうまく動かないというケースが出てきます。
でも、ブラウザの違いを調べながらプログラムを書くのは大変ですよね。そんなときに便利なのが、ブラウザ間の違いを修正してくれる「ライブラリ」です。ライブラリとは、汎用性の高いプログラムを再利用しやすくまとめたもので、中にはブラウザ間の違いを修正してくれるものもあります。本書でも、ブラウザ間の違いを吸収しつつ、便利な機能を提供してくれるライブラリ「jQuery」を扱います。
jQueryはGoogleやYahoo! などのサイトでも利用され、JavaScriptを扱う人なら必ず知っているといってもいいほど人気の高いライブラリです。
この章ではライブラリに頼らずHTML/CSSを操作する方法を学びますが、複雑な操作を行う際は「jQuery」などのライブラリを使用するほうが、ブラウザごとの違いに対応することもでき、より一般的です。どちらも重要な知識なので、後のChapter 10では、jQueryを利用したHTML/CSSの操作方法も学んでいきます。

▶ Can I use...



http://caniuse.com
JavaScriptのメソッドやCSSプロパティのブラウザごとの対応状況を確認できるWebサイト。

Chapter

7

ユーザーの操作に対応させよう

この章では、ユーザーの操作にあわせてプログラムを実行するイベント処理について学んでいきます。操作に応じて処理を切り替えることで、より実用的なプログラムを記述することができるようになります。