Traversing the DOM with JavaScript

11TH APR 2018

A good JavaScript developer needs to know how to traverse the DOM—it's the act of selecting an element from another element.

But why do we need to learn to traverse the DOM? Isn't `document.querySelector` enough for most of our needs?

In this article, I'm going to show you why traversing is better that `document.querySelector`, and how to traverse like a pro. So sit back, relax, and enjoy the article!

Before we move on, note that this article is a sample lesson from Learn JavaScript—a course I built to help you learn JavaScript once and for all (even if you tried and failed previously). You can find out more about Learn JavaScript through this link.

Why we traverse

Let's say you want to go to your neighbor's house. What's the fastest and most efficient way to get there?

1. Move from your house to their house (since you already know their address)

2. Lookup their address on Google maps, then walk according to the directions Google gives you.

If you move directly from your house to their house, you're doing the equivalent of traversing the DOM—selecting one element from a neighboring element.

If you lookup their address on Google, you're doing the equivalent of `document.querySelector` to find elements.

Can you guess which method is more efficient?

```
<div class="neighborhood">
  <div class="your-house">😇</div>
  <div class="neighbor-house">🎉</div>
</div>
```

You probably know the answer—it's always easier to move from an element to another (compared do doing a full search). That's why we traverse the DOM.

You can traverse in three directions:

1. Downwards
2. Sideways
3. Upwards

(Note: there's a second reason—it's more reliable—but that's an advanced topic for another day.)

Traversing downwards

There are two methods to traverse downwards:

1. `querySelector` or `querySelectorAll`
2. `children`

querySelector or querySelectorAll

To traverse downwards from a specific element, you can use

To traverse downwards from a specific element, you can use `element.querySelector` or `element.querySelectorAll`.

If we put `element.querySelector` into the house analogy, we search for a specific room in your house. It's faster than searching for the same room from outer space (the document).

```
<div class="component">
  <h2 class="component__title">Component title</h2>
</div>

const component = document.querySelector('.component')
const title = component.querySelector('.component__title')

console.log(title) // <h2 class="component__title"> ... </h2>
```

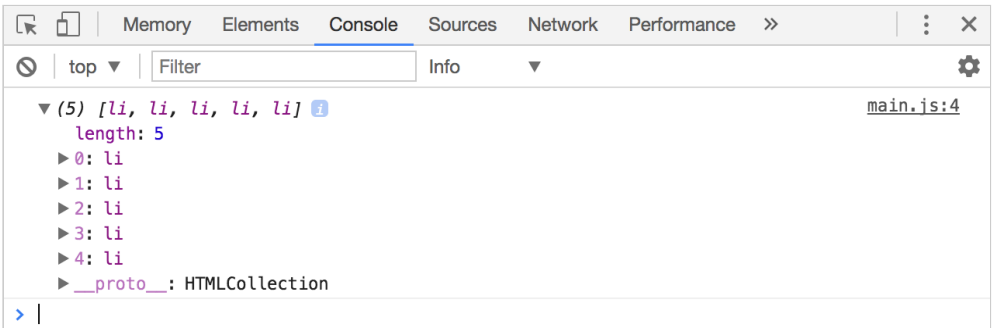
CHILDREN

`children` is a property that lets you select direct descendants (elements that are immediately nested in another element). It returns a HTML Collection that updates when children elements are changed.

```
<ul class="list">
  <li><a href="#">Link 1</a></li>
  <li><a href="#">Link 2</a></li>
  <li><a href="#">Link 3</a></li>
  <li><a href="#">Link 4</a></li>
  <li><a href="#">Link 5</a></li>
</ul>

const list = document.querySelector('.list')
const listItems = list.children

console.log(listItems)
```



> Selecting all list items with the children property

A HTML Collection is similar to a NodeList (that `querySelectorAll` returns). There are subtle differences that don't really matter for the context of this article.

What matters is—a HTML collection is an array-like object. If you want to loop over it with `Array.prototype.forEach`, you need to convert it into an array with `Array.from` first.

```
const array = Array.from(HTMLCollection)
array.forEach(el => { /* do whatever you want */})
```

Selecting a specific child

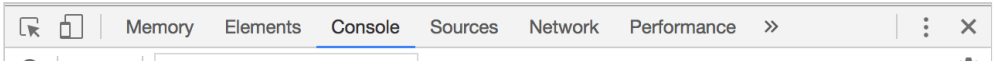
You can select the nth-item in the list from both NodeLists (result from `querySelectorAll`) and HTML Collections (result from `children`). To do so, you use the index of the element, just like how you select a specific item from an Array.

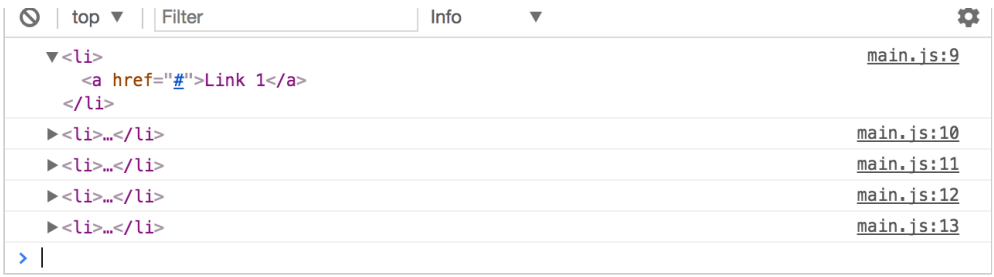
```
<ul class="list">
  <li><a href="#">Link 1</a></li>
  <li><a href="#">Link 2</a></li>
  <li><a href="#">Link 3</a></li>
  <li><a href="#">Link 4</a></li>
  <li><a href="#">Link 5</a></li>
</ul>
```

```
const listItems = document.querySelectorAll('li')

const firstItem = listItems[0]
const secondItem = listItems[1]
const thirdItem = listItems[2]
const fourthItem = listItems[3]
const fifthItem = listItems[4]

console.log(firstItem)
console.log(secondItem)
console.log(thirdItem)
console.log(fourthItem)
console.log(fifthItem)
```





> Select a specific child with [index]

Try the above code with a HTML Collection. You'll get the same result.

Traversing upwards

There are two methods to traverse upwards:

1. `parentElement`
2. `closest`

parentElement

`parentElement` is a property that lets you select the parent element. The parent element is the element that encloses the current element.

In the following HTML, `.list` is the parent element of all `<li>`. Each `<li>` is the parent element of their respective `<a>`.

```
<ul class="list">
  <li><a href="#">Link 1</a></li>
  <li><a href="#">Link 2</a></li>
  <li><a href="#">Link 3</a></li>
  <li><a href="#">Link 4</a></li>
  <li><a href="#">Link 5</a></li>
</ul>
```

```
const firstListItem = document.querySelector('li')
const list = firstListItem.parentElement

console.log(list)
// <ul class="list">...</ul>
```

closest

closest

`parentElement` is great for selecting one level upwards. To find an element that can be multiple levels above the current element, you use the `closest` method.

`closest` lets you select the closest ancestor element that matches a selector. Here's the syntax:

```
const closestAncestor = Element.closest(selector)
```

As you may suspect, `selector` is the same `selector` you pass to `querySelector` and `querySelectorAll`.

In the following HTML, you can select `.list` from the `<a>` effortlessly with `Element.closest`:

```
<ul class="list">
  <li><a href="#">Link 1</a></li>
  <li><a href="#">Link 2</a></li>
  <li><a href="#">Link 3</a></li>
  <li><a href="#">Link 4</a></li>
  <li><a href="#">Link 5</a></li>
</ul>
```

```
const firstLink = document.querySelector('a')
const list = firstLink.closest('.list')

console.log(list)
// <ul class="list"> ... </ul>
```

Note: `closest` **starts searching from the current element**, then proceeds upwards until it reaches the `document`. It stops returns the first element it finds.

```
const firstLink = document.querySelector('a')
const firstLinkThroughClosest = firstLink.closest('a')

console.log(firstLinkThroughClosest)
// <a href="#">Link 1</a>
```

`closest` is pretty new. It doesn't work on IE Edge 14 and below. It doesn't work on Opera mini too. If you need to support older

browsers, you may want to use a polyfill.

Traversing sideways

There are three methods to traverse sideways:

1. `nextElementSibling`
2. `previousElementSibling`
3. Combining `parentElement`, `children`, and `index`.

nextElementSibling

You can select the next element with `nextElementSibling`.

```
const nextElem = Node.nextElementSibling
```

```
<ul class="list">
  <li><a href="#">Link 1</a></li>
  <li><a href="#">Link 2</a></li>
  <li><a href="#">Link 3</a></li>
  <li><a href="#">Link 4</a></li>
  <li><a href="#">Link 5</a></li>
</ul>
```

```
const firstListItem = document.querySelector('li')
const secondListItem = firstListItem.nextElementSibling

console.log(secondListItem)
// <li><a href="#">Link 2</a></li>
```

previousElementSibling

Likewise, you can select the previous element with `previousElementSibling`.

```
<ul class="list">
  <li><a href="#">Link 1</a></li>
  <li><a href="#">Link 2</a></li>
  <li><a href="#">Link 3</a></li>
  <li><a href="#">Link 4</a></li>
```

```
  <li><a href="#">Link 5</a></li>
</ul>
```

```
const secondListItem = document.querySelectorAll('li')[1]
const firstListItem = secondListItem.previousElementSibling

console.log(firstListItem)
// <li><a href="#">Link 1</a></li>
```

Combining parentElement, children, and index

This method lets you select a specific sibling. It's easier to explain how it works with an example, so let's do that. Say you want to select the fourth item from the first item in this HTML.

```
<ul class="list">
  <li><a href="#">Link 1</a></li>
  <li><a href="#">Link 2</a></li>
  <li><a href="#">Link 3</a></li>
  <li><a href="#">Link 4</a></li>
  <li><a href="#">Link 5</a></li>
</ul>
```

Let's say you already have the first item:

```
const firstItem = document.querySelector('li')
```

To select the fourth item, you can use `firstItem.parentElement` to get the list, then `list.children` to get a HTML Collection. Once you have the HTML Collection, you can find the fourth item by using a index of 3. (Remember, zero-based index!).

```
const firstItem = document.querySelector('li')
const list = firstItem.parentElement
const allItems = list.children
const fourthItem = allItems[3]

console.log(fourthItem)
// <li><a href="#">Link 4</a></li>
```

Putting everything together in one step:

```
const firstItem = document.querySelector('li')
const fourthItem = firstITem.parentElement.children[3]

console.log(fourthItem)
// <li><a href="#">Link 4</a></li>
```

Exercise

Practice traversing the DOM with the methods taught in this lesson. With the HTML given below, do these tasks:

1. Select `.characters` with `document.querySelector`
2. Select `.humans` from `.characters`
3. Select all humans with `querySelectorAll`, starting from `.humans`
4. Select all hobbits with `children`
5. Select the Merry (the hobbit)
6. Select `.enemies` from Sauron
7. Select the `.characters` div from Nazgûl
8. Select Elrond from Glorfindel
9. Select Legolas from Glorfindel
10. Select Arwen from Glorfindel

```
<div class="characters">
  <ul class="hobbits">
    <li>Frodo Baggins</li>
    <li>Samwise "Sam" Gamgee</li>
    <li>Meriadoc "Merry" Brandybuck</li>
    <li>Peregrin "Pippin" Took</li>
    <li>Bilbo Baggins</li>
  </ul>
  <ul class="humans">
    <li>Gandalf</li>
    <li>Saruman</li>
    <li>Aragorn</li>
    <li>Boromir</li>
    <li>Faramir</li>
  </ul>
  <ul class="elves">
    <li>Legolas</li>
```

```
      <li>Glorfindel</li>
      <li>Elrond</li>
      <li>Arwen Evenstar</li>
    </ul>
    <ul class="enemies">
      <li>Sauron</li>
      <li>Nazgûl</li>
    </ul>
  </div>
```

Wrapping up

You learned how to traverse the DOM in three directions—downwards, upwards, and sideways—in this lesson. Here's a quick bullet point to summarize the methods you learned:

1. Traversing downwards
    1. `element.querySelector`
    2. `element.querySelectorAll`
    3. `element.children`
2. Traversing upwards
    1. `element.parentElement`
    2. `element.closest`
3. Traversing sideways
    1. `element.nextElementSibling`
    2. `element.previousElementSibling`
    3. Combine `parentElement`, `children`, and `index`

I hope this lesson helps clarify why we traverse the DOM and how you can do it. If this lesson has helped you, might enjoy Learn JavaScript, where share more JavaScript lessons—basics, intermediate, and even advanced lessons—in a step by step manner.

If you enjoyed this article, please tell a friend about it! Share it on Twitter. If you spot a typo, I'd appreciate if you can correct it on GitHub. Thank you!

Become a JavaScript expert with this free email course

Don't be afraid if you're stuck, overwhelmed or confused with JavaScript. Break your top 3 learning barriers and learn JavaScript quickly through an email course I built specifically for you – JavaScript Roadmap.

First Name

Email Address

Get your Javascript roadmap for free

< How to handle the "bad experience" question

Why I restructured Learn JavaScript >