Accessing the DOM is not equal accessing the DOM – live vs. static element collections

Published at
Mar 22 2018

Updated at
Jun 30 2018

Reading time
4 min

When a browser receives an HTML document, it creates the Document Object Model (DOM) which is a tree representation of the document. Then there are DOM methods that allow us as Frontend developers to programmatically access parts of the parsed documents and add functionality to websites. So far so good!

A method you come across quickly is `querySelectorAll` which is used to access elements in the DOM. Let's have a quick look at how it works.

Accessing the DOM using `querySelectorAll`

JavaScript

```javascript
// <html>
// <head>...</head>
```

```
//     <li>foo</li>
//     <li>bar</li>
//     <li>baz</li>
//   </ul>
// </body>
// </html>

const listItems = document.querySelectorAll('li');
console.log(listItems);        // NodeList(3) [li, li, li]
console.log(listItems.length); // 3

for (let i = 0; i < listItems.length; i++) {
  console.log(listItems[i].innerText);
}

// foo
// bar
// baz
```

Thanks to good developer tooling these days browsers show the type of an object when you log it to the console. As you see above the return value of `document.querySelectorAll` is a `NodeList` .

Dealing with `NodeLists` meant some surprises for me in the past. They look like Arrays but they are not, and a big warning box in the particular MDN article describes this fact clearly.

> *Although NodeList is not an Array, it is possible to iterate on it using forEach(). Several older browsers have not implemented this method yet. You can also convert it to an Array using Array.from.*

What surprises me is that `NodeLists` have a defined `forEach` method today because this method was missing when I started in web development and this was exactly one of the pitfalls I ran into a lot of times years ago. Other methods that are provided by `NodeLists` are `item` , `entries` , `keys` , and `values` . In case you want to read more about these I recommend to check out the MDN article.

The magic of live collections

When I read the documentation for `NodeLists` last week, I noticed something that I've never seen before:

> *In some cases, the NodeList is a live collection [...]*

Wait, what? A live collection? In some cases?

It turns out that `NodeLists` behave differently depending on how you access them. Let's have a look at the same document and retrieve elements differently.

JavaScript

```
// <html>
// <head>...</head>
// <body>
//   <ul>
//     <li>foo</li>
//     <li>bar</li>
//     <li>baz</li>
//   </ul>
// </body>
// </html>

// retrieve element using querySelectorAll
const listItems_querySelectorAll = document.querySelectorAll('li');
console.log(listItems_querySelectorAll); // NodeList(3) [li, li, li

// retrieve element using childNodes
const list  = document.querySelector('ul');
const listItems_childNodes = list.childNodes;
console.log(listItems_childNodes); // NodeList(7) [text, li, text,
```

The apparent difference is that there are more elements included when you access elements via `childNodes`. The text nodes in this collection are the spaces and line breaks that you see in the HTML.

But that's not what I discovered. The big difference between the two `NodeLists` is that **one is live and one is static** which becomes visible when I add another list item to the `ul` element.

JavaScript

```
list.appendChild(document.createElement('li'));

// static NodeList via querySelectorAll
console.log(listItems_querySelectorAll); // NodeList(3) [li, li, li
// live NodeList via childNodes
console.log(listItems_childNodes);       // NodeList(8) [text, li,
```

😲 As you see `listItems_childNodes` (the `NodeList` accessed via `childNodes`) reflects the elements of the DOM even when elements were added or removed. The collection that is returned by `querySelectorAll` stays the same. **That's entirely new news to me!**

Not every method to query the DOM returns a NodeList

It gets even more confusing... you might know that there are also methods like `getElementsByClassName` and `getElementsByTagName` that let you access DOM elements, too. It turns out these methods return something different.

JavaScript

```
// <html>
// <head>...</head>
// <body>
//   <ul>
//     <li>foo</li>
```

```
//   </ul>
// </body>
// </html>

const listItems_getElementsByTagName = document.getElementsByTagNam
console.log(listItems_getElementsByTagName); // HTMLCollection(3) [
```

Oh well... an `HTMLCollection` . So what is this other type? It only includes the matching elements and does not include text nodes, it provides only two methods (`item` and `namedItem`) and **it is live** which means that it will also include added elements.

JavaScript

```
listItems_getElementsByTagName[0].parentNode.appendChild(document.c

// live HTMLCollection via getElementsByTagName
console.log(listItems_getElementsByTagName); // HTMLCollection(4) [
```

And to make it even more complicated, `HTMLCollections` are also returned when you use `document.forms` (yes – you can access forms without querying the DOM) or access child elements via the `children` property of an element.

JavaScript

```
// <html>
// <head>...</head>
// <body>
//   <ul>
//     <li>foo</li>
//     <li>bar</li>
//     <li>baz</li>
//   </ul>
// </body>
// </html>

const list = document.querySelector('ul');
```

When you look at the specification of `HTMLCollection` you'll find the following sentence:

> *HTMLCollection is a historical artifact we cannot rid the web of. While developers are of course welcome to keep using it, new API standard designers ought not to use it [...]*

This statement makes clear that for a certain amount of time `NodeList` and `HTMLCollection` where competing standards and now we're stuck with both of them.

Evolving the web is complicated

So, today we have `childNodes` (live `NodeList`) next to `children` (live `HTMLCollection`), `querySelectorAll` (static `NodeList`) next to `getElementsByTagName` (live `HTMLCollection`) and some unexpected edge-cases depending on how you access elements.

Personally, I'm surprised that I never heard of live and static collections before and I think the discovery of this detail when dealing with the DOM will save me a lot of time someday because finding a bug that is caused by a live collection is definitely very hard to find.

If you want to play around with the described behavior you can check out this CodePen.

Share article on Twitter

Related Topics

JS JavaScript

See 1 comment.

Load comments

Get the latest content right in your inbox ...

(View past newsletters)

Email address

janedoe@example.com

Subscribe

OR

... say Hi somewhere in the internet!

Twitter GitHub Instagram

This page is built with the CNN stack (Contentful, Nuxt & Netlify) and loaded in 1.15s.