Loop through a collection of DOM elements

Published: 2016.10.19 | 3 minutes read

It is easy to think of a collection of DOM elements as a regular JavaScript array. This is a gotcha that many beginners fall into (including myself). NodeLists don't share all of the `Array`'s prototype methods, but there are a heap of ways to achieve the desired result. Let's go through the list of possible methods and hacks. No frameworks or libraries today - it's pure js day fellaz!

NodeList.forEach()

Aha! You'll know this method mainly from the Array's prototype but actually some browsers contain this function in the prototype of NodeList too. However, because of the lack of sufficient browser support I wouldn't consider it the way to go. This list would have been incomplete without it though.

- Google Chrome - yeep
- Firefox >= 50
- IE - hazard a guess!
- Edge - nope
- Opera - yeep
- Safari (stable version) - nope
- Safari (Technology Preview) - yeep

- Android - nope
- Android (Chrome) - yeep
- Firefox Mobile - yeep
- iOS - nope

```
const articles = document.querySelectorAll('article');

articles.forEach(a => {
  a.style.fontFamily = 'Comic Sans MS';
});

// Chrome - 'Comic Sans MS' everywhere dudes! Sweet!
// Firefox - TypeError: articles.forEach is not a function
```

Array.prototype.forEach()

If `forEach()` doesn't exist in `NodeList`'s prototype, you can always ask your good friend `Array` to lend it to you — your browser definitely has this (if it's not Internet Explorer 8 or below).

```
const articles = document.querySelectorAll('article');

[].forEach.call(articles, a => {
  a.style.fontFamily = 'Comic Sans MS';
});

// or

Array.prototype.forEach.call(articles, a => {
  a.style.fontFamily = 'Comic Sans MS';
});
```

If you don’t like `call()` or `apply()` you can convert the DOM elements to an array first and then use `forEach()` as you intend to.

```
const articles = [].slice.call(document.querySelectorAll('article'));

// or

const articles = [...document.querySelectorAll('article')];

// or

const articles = Array.from(document.querySelectorAll('article'));

articles.forEach(a => {
  a.style.fontFamily = 'Comic Sans MS';
});
```

You can even go absolutely crazy and add Array’s `forEach()` to `NodeList.prototype`.

```
if (typeof NodeList.prototype.forEach === "undefined") {
  NodeList.prototype.forEach = Array.prototype.forEach;
}

if (typeof HTMLCollection.prototype.forEach === "undefined") {
  HTMLCollection.prototype.forEach = Array.prototype.forEach;
}

const articles = document.querySelectorAll('article');

articles.forEach(a => {
  a.style.fontFamily = 'Comic Sans MS';
});
```

All three of the snippets above will work just fine. They do feel a bit hacky though and I'm not the only one who thinks like this. Bear in mind that the spread operator presented above `[...]` and `Array.from()` are parts of the modern spec. To use them without worry equip yourself with Babel.

for loop

The good ol' `for` loop is a good candidate to do this job. It's a very well supported and reliable method. No hacks, no babels!

```
const articles = document.querySelectorAll('article');

for (let i = 0; i < articles.length; i++) {
  articles[i].style.fontFamily = 'Comic Sans MS';
}
```

for-of loop

The ECMAScript 2015 spec brought us a new tool to traverse through iterable objects. As we saw in the previous example, `NodeList` is definitely an iterable collection so we can easily add a for..of loop to our collection. Babel may be helpful in this instance as it is a part of the spec that is a bit more modern than your clients requirements.

```
const articles = document.querySelectorAll('article');

for (let a of articles) {
  a.style.fontFamily = 'Comic Sans MS';
}
```

Conclusions

Hopefully this list of methods (and hacks) helped you out. Use whatever feels right depending on your use case. My preferable method from the list above is the `for...of` loop as almost every single line of my code goes through a compiler that will translate it to a syntax that even old school browsers can handle. If I need to quickly smash an example out I use a `for` loop.

Let me know your thoughts. What is your preferred method to traverse through DOM elements? If you liked this article the share buttons are right below. Bye :*

If you liked this article, please share it on Twitter.

Comments Community Privacy Policy 1 Login

Recommend Tweet Share Sort by Best

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Chris Wijnia • 3 years ago • edited

I like to do it like this in es2015:

```
[...elements].forEach(element => func)
```

3 | • Reply • Share ›

Paweł Grzybek Mod Chris Wijnia • 3 years ago

Another great method! Thanks.

| • Reply • Share ›

rafi rafi • 6 months ago

Where is the HTML document;;;;

^ | ⌄ • Reply • Share ›

Ivanix • 4 years ago

Thanks for this post! Btw, I found that in Android chrome version 49.xx document.querySelectorAll does NOT have forEach. I was going nuts because my web app was failing silently on chrome mobile while working fine on desktop.

^ | ⌄ • Reply • Share ›

Paweł Grzybek Mod → Ivanix • 4 years ago

Yeah, you are right. It seems to work fine on Chrome for Android from version 51.

https://developer.mozilla.o...

^ | ⌄ • Reply • Share ›

PuckRockGrrl • 4 years ago • edited

Give Array.from a try:

```
<p>something</p>
 <p>sumthing</p>
 <p>something</p>

    <script type="text/javascript">
    // Select all the paragraphs, convert to Array
    const items = Array.from(document.querySelectorAll('p')
    // Filter for only the elements that contain the word
    const filtered = items
      .filter(item => item.textContent.includes('sumthing')
      .map(item => item.textContent = item.textContent + '
    </script>
```

^ | ⌄ • Reply • Share ›

Paweł Grzybek Mod → PuckRockGrrl • 4 years ago

Works like a charm :)

View — uploads.disquscdn.com

^ | ⌄ • Reply • Share ›

PuckRockGrrl → Paweł Grzybek
• 4 years ago

Learned that trick from Wes Bos' ES6 course recently! :-) Really enjoying it so far.

^ | ⌄ • Reply • Share ›

Daniel Nass • 4 years ago

For of did not compatible with Internet Explorer through babel, sadly. Babel use a Symbol operator to make for of available and IE did not support this feature.
To make compatible you need to use a 'loose' option, as described here http://babeljs.io/docs/plug...

Reply • Share ›

Paweł Grzybek Mod → Daniel Nass • 4 years ago

I think the same applies to Safari or some "non-current" version of iOS Safari.
Thanks **@Daniel Nass** , very good point. I will update article late on.

Reply • Share ›

Šime Vidas • 4 years ago

But how do you usually test in IE and Edge? I'm curious 😅

Reply • Share ›

Paweł Grzybek Mod → Šime Vidas • 4 years ago

I normally spin up a VirtualBox. I have a machine set up that runs on IE9 and Windows 7. Maybe it's not the most comprehensive way of testing but does the job - allows me to catch the bugs on my day to day projects.

Any Edge next to you to test one thing for me my friend? 🙂

Reply • Share ›

Šime Vidas → Paweł Grzybek • 4 years ago

Ah, I forgot. So, in Edge, NodeList doesn't