CONTENTS

RELATED

CodeIgniter: Getting Started With a Simple Example

View

How To Install Express, a Node.js Framework, and Set Up Socket.io on a VPS

View

Tutorial Series: Understanding the DOM — Document Object Model

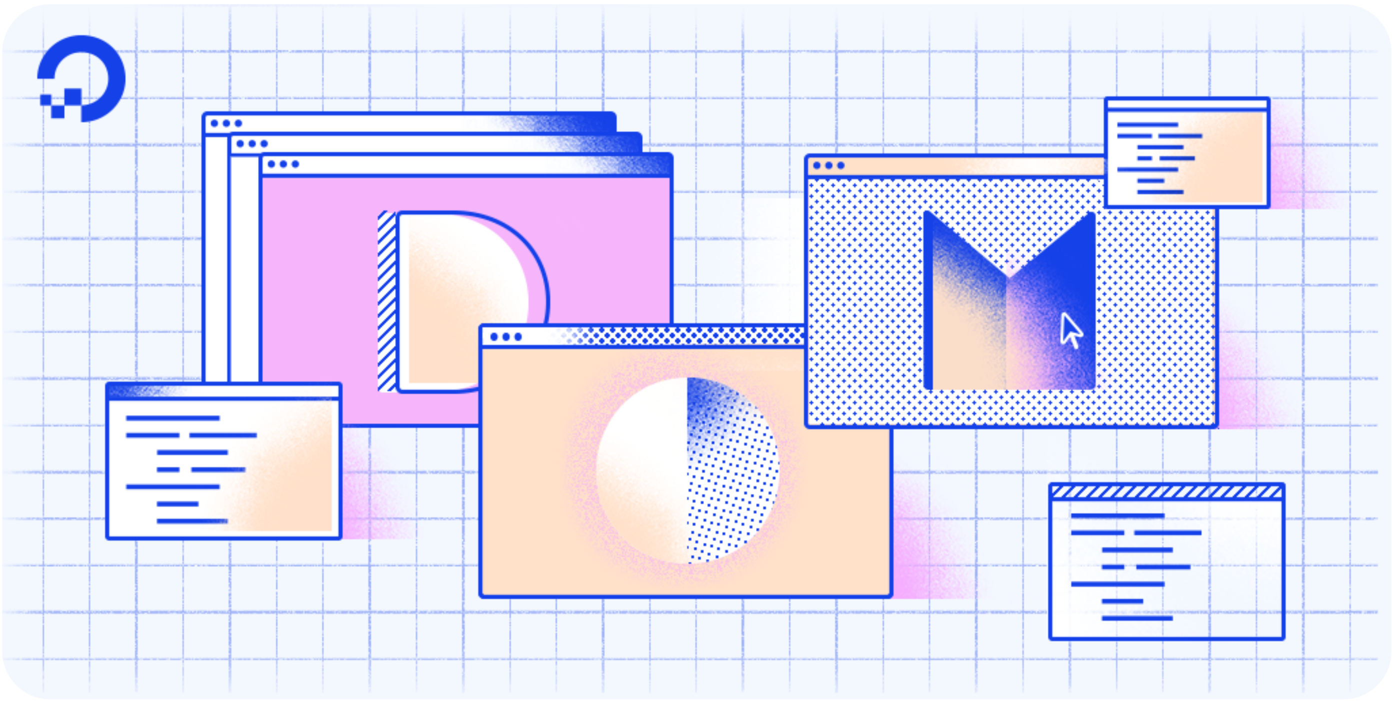
// Tutorial //

How To Access Elements in the DOM

Published on November 20, 2017 · Updated on June 23, 2022

JavaScript Development

By Tania Rascia



# Introduction

In Understanding the DOM Tree and Nodes, we went over how the DOM is structured as a tree of objects called nodes, and that nodes can be text, comments, or elements. Usually when we access content in the DOM, it will be through an HTML element node.

In order to be confident in accessing elements in the DOM, it's good to have a working

Here is a table overview of the five methods we will cover in this tutorial.

Gets	Selector Syntax	Method
ID	`#demo`	`getElementById()`
Class	`.demo`	`getElementsByClassName()`
Tag	`demo`	`getElementsByTagName()`
Selector (single)		`querySelector()`
Selector (all)		`querySelectorAll()`

It is helpful when studying the DOM to work with the examples on your own to ensure that you are understanding and retaining the information you learn.

Create a new file, `access.html`, in your own project to work through the examples along with this article. If you are unsure how to work with JavaScript and HTML locally, review our How To Add JavaScript to HTML tutorial.

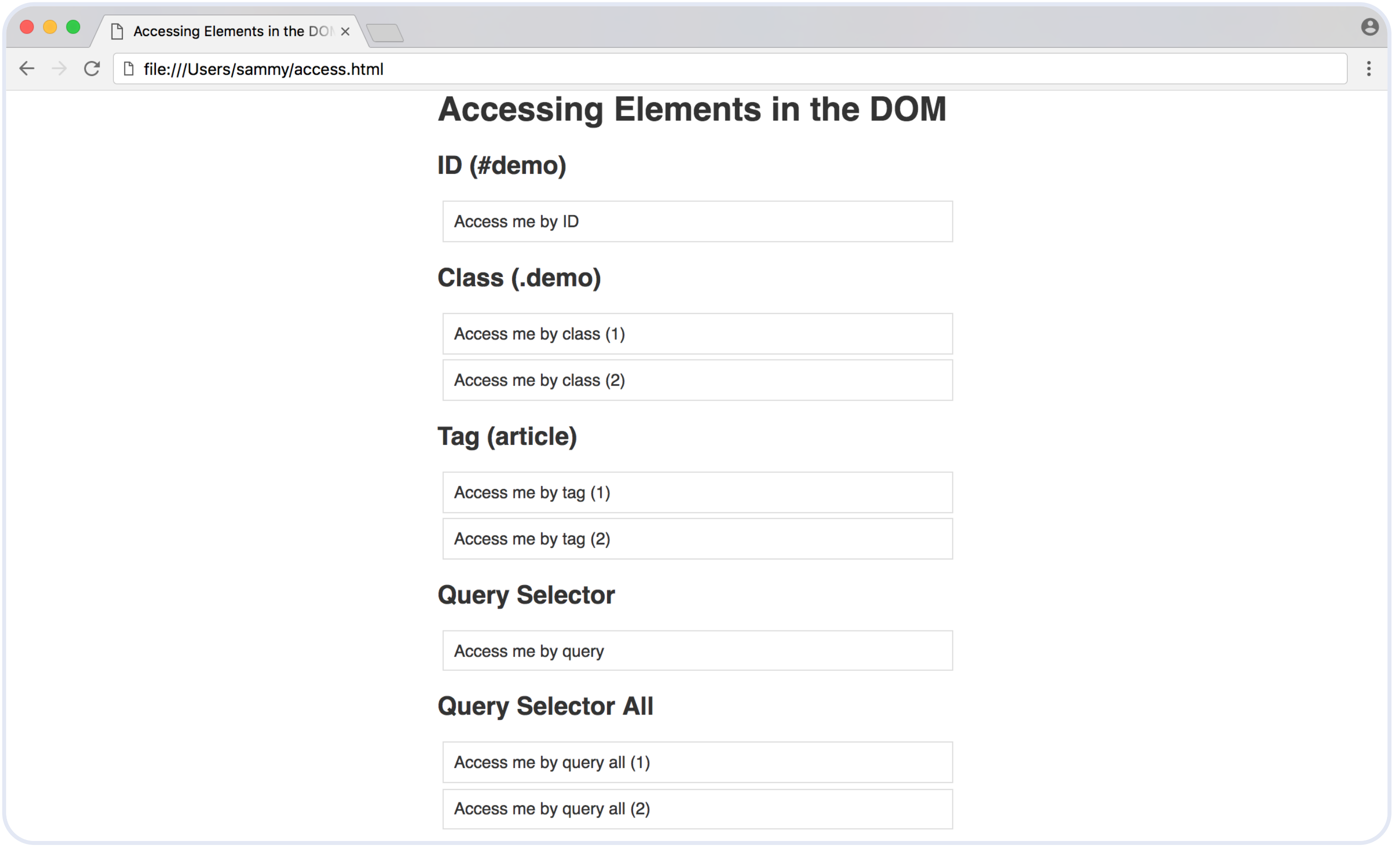
access.html

Copy

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>Accessing Elements in the DOM</title>
```

```
</head>

<body>

  <h1>Accessing Elements in the DOM</h1>

  <h2>ID (#demo)</h2>
  <div id="demo">Access me by ID</div>

  <h2>Class (.demo)</h2>
  <div class="demo">Access me by class (1)</div>
  <div class="demo">Access me by class (2)</div>

  <h2>Tag (article)</h2>
  <article>Access me by tag (1)</article>
  <article>Access me by tag (2)</article>

  <h2>Query Selector</h2>
  <div id="demo-query">Access me by query</div>

  <h2>Query Selector All</h2>
  <div class="demo-query-all">Access me by query all (1)</div>
  <div class="demo-query-all">Access me by query all (2)</div>

</body>

</html>
```

In this HTML file, we have many elements that we will access with different `document` methods. When we render the file in a browser, it will look similar to this:


Accessing Elements in the DOM
file:///Users/sammy/access.html

Accessing Elements in the DOM

ID (#demo)

Access me by ID

Class (.demo)

Access me by class (1)

Access me by class (2)

Tag (article)

Access me by tag (1)

Access me by tag (2)

Query Selector

Access me by query

Query Selector All

Access me by query all (1)

Access me by query all (2)

We'll be using the different methods that we outlined in the Overview above to access the available elements in the file.

# Accessing Elements by ID

The easiest way to access a single element in the DOM is by its unique ID. You can get an element by ID with the `getElementById()` method of the `document` object.

```
document.getElementById();
```
Copy

In order to be accessed by ID, the HTML element must have an `id` attribute. You have a `div` element with an ID of `demo` you can use:

```
<div id="demo">Access me by ID</div>
```
Copy

In the *Console*, get the element and assign it to the `demoId` variable.

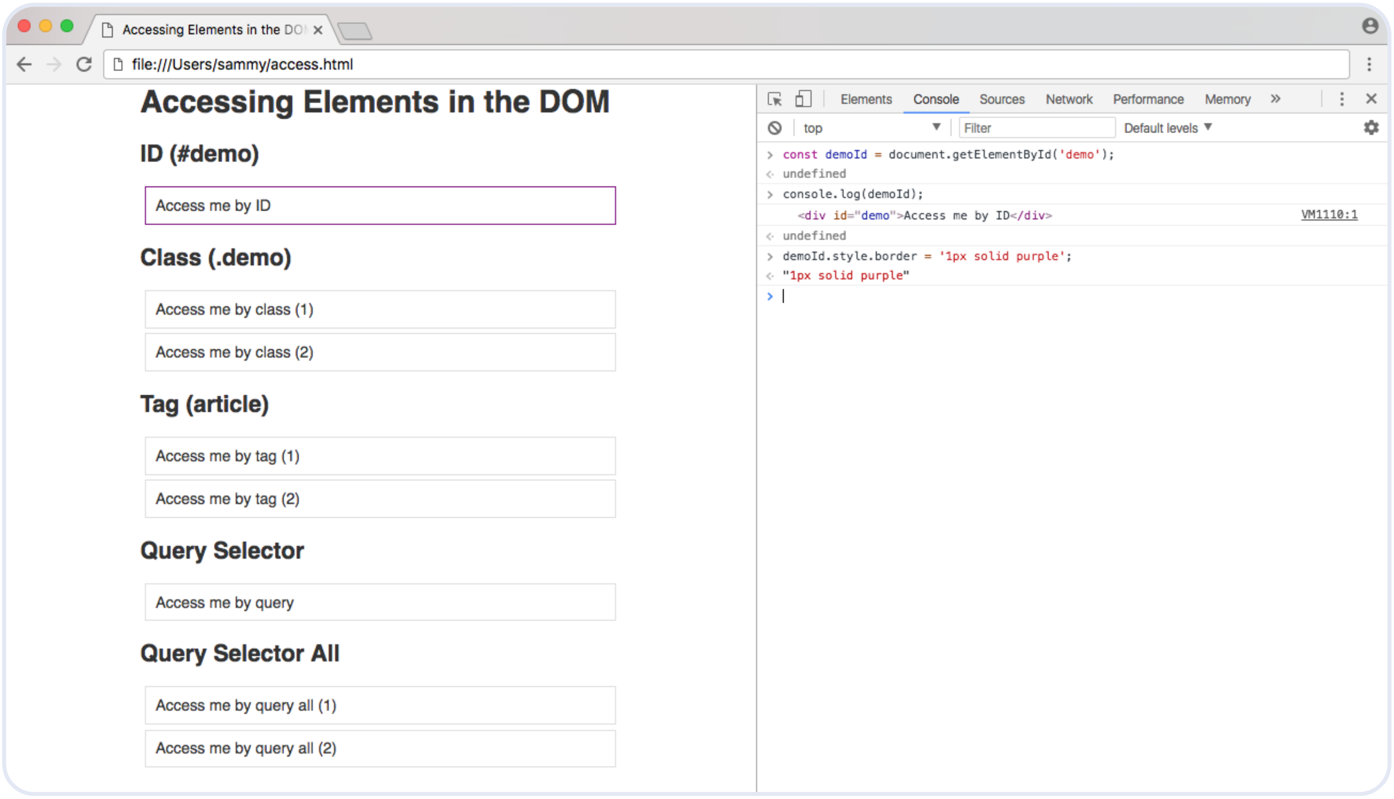
```
> console.log(demoId);
```
Copy

```
Output
<div id="demo">Access me by ID</div>
```

You can be sure you're accessing the correct element by changing the `border` property to `purple`.

```
> demoId.style.border = '1px solid purple';
```
Copy

Once you do so, your live page will look like this:



Accessing Elements in the DOM

ID (#demo)

Access me by ID

Class (.demo)

Access me by class (1)

Access me by class (2)

Tag (article)

Access me by tag (1)

Access me by tag (2)

Query Selector

Access me by query

Query Selector All

Access me by query all (1)

Access me by query all (2)

```
> const demoId = document.getElementById('demo');
< undefined
> console.log(demoId);
    <div id="demo">Access me by ID</div>                                  VM1110:1
< undefined
> demoId.style.border = '1px solid purple';
< "1px solid purple"
> |
```

Accessing an element by ID is an effective way to get an element quickly in the DOM. However, it has drawbacks: an ID must always be unique to the page, and therefore you will only ever be able to access a single element at a time with the `getElementById()` method. If you wanted to add a function to many elements throughout the page, your code would quickly become repetitious.

```
document.getElementsByClassName();
```
Copy

Now we want to access more than one element, and in our example we have two elements with a `demo` class.

```
<div class="demo">Access me by class (1)</div>
<div class="demo">Access me by class (2)</div>
```
Copy

Access these elements in the *Console* and put them in a variable called `demoClass`.

```
> const demoClass = document.getElementsByClassName('demo');
```
Copy

At this point, it might be tempting to modify the elements the same way you did with the ID example. However, if you try to run the following code and change the `border` property of the class demo elements to orange, you will get an error.

```
> demoClass.style.border = '1px solid orange';
```
Copy

```
Output
Uncaught TypeError: Cannot set property 'border' of undefined
```

The reason this doesn't work is because instead of just getting one element, you have an array-like object of elements.

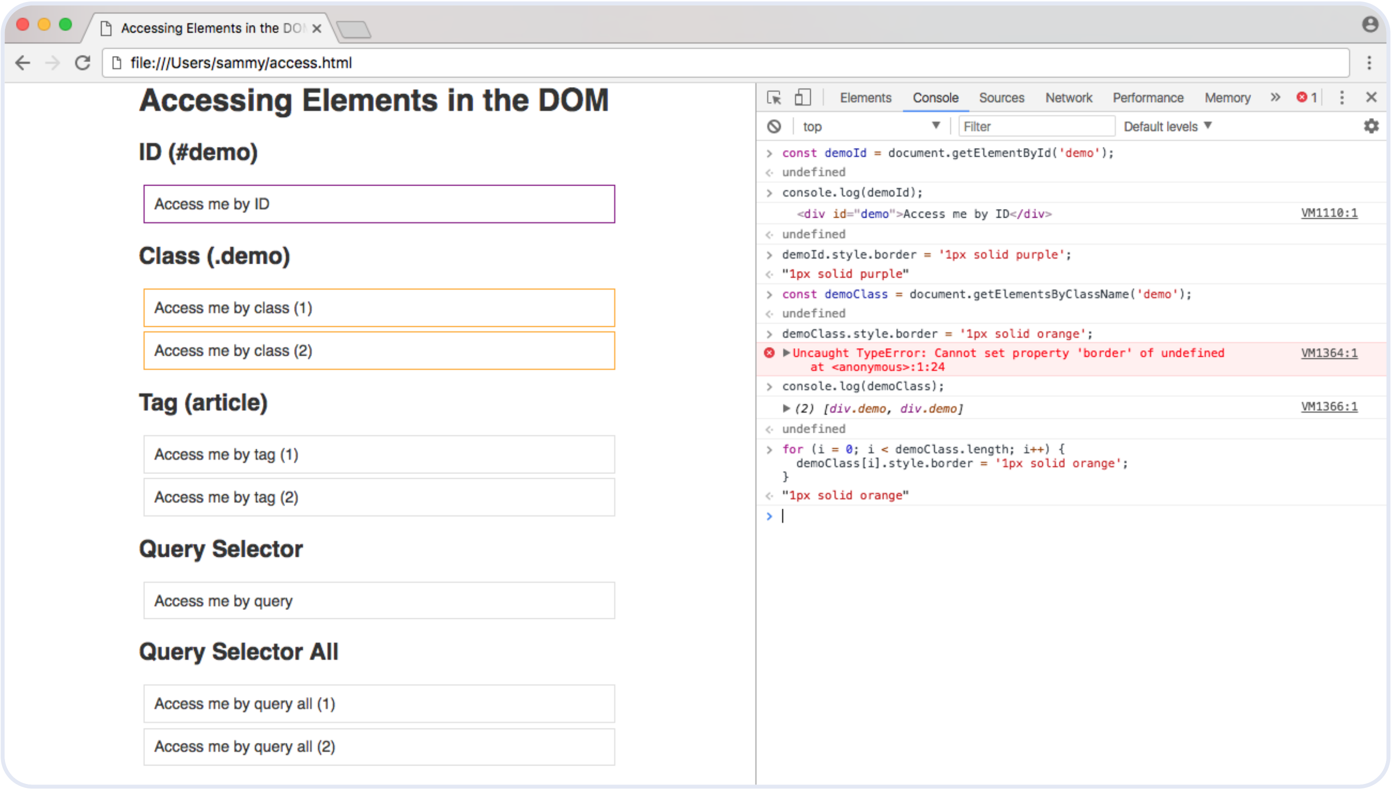
```
> console.log(demoClass);
```
Copy

```
Output
(2) [div.demo, div.demo]
```

JavaScript arrays must be accessed with an index number. You can change the first element of this array by using an index of `0`.

```
> for (i = 0; i < demoClass.length; i++) {
>   demoClass[i].style.border = '1px solid orange';
> }
```
Copy

When you run this code, your live page will be rendered like this:



Accessing Elements in the DOM

ID (#demo)

Access me by ID

Class (.demo)

Access me by class (1)

Access me by class (2)

Tag (article)

Access me by tag (1)

Access me by tag (2)

Query Selector

Access me by query

Query Selector All

Access me by query all (1)

Access me by query all (2)

```
> const demoId = document.getElementById('demo');
< undefined
> console.log(demoId);
    <div id="demo">Access me by ID</div>                         VM1110:1
< undefined
> demoId.style.border = '1px solid purple';
< "1px solid purple"
> const demoClass = document.getElementsByClassName('demo');
< undefined
> demoClass.style.border = '1px solid orange';
✖ ▶Uncaught TypeError: Cannot set property 'border' of undefined   VM1364:1
      at <anonymous>:1:24
> console.log(demoClass);
  ▶ (2) [div.demo, div.demo]                                     VM1366:1
< undefined
> for (i = 0; i < demoClass.length; i++) {
    demoClass[i].style.border = '1px solid orange';
  }
< "1px solid orange"
> |
```

You have now selected every element on the page that has a `demo` class, and changed the `border` property to `orange`.

# Accessing Elements by Tag

A less specific way to access multiple elements on the page would be by its HTML tag name. You access an element by tag with the `getElementsByTagName()` method.

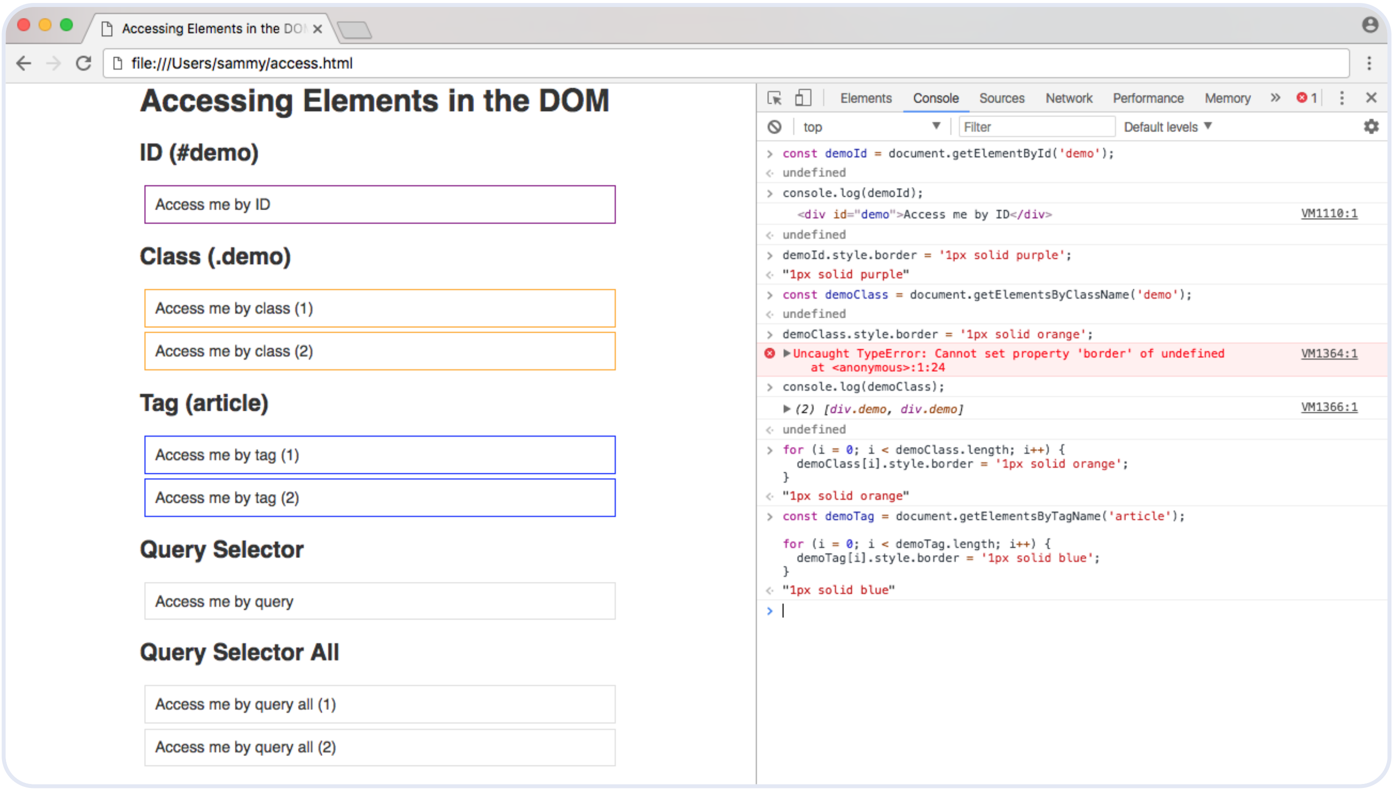
```
document.getElementsByTagName();
```
Copy

For our tag example, we're using `article` elements.

```
> const demoTag = document.getElementsByTagName('article');
>
> for (i = 0; i < demoTag.length; i++) {
>   demoTag[i].style.border = '1px solid blue';
> }
```

Copy

Upon running the code, the live page will be modified like so:



Accessing Elements in the DOM

ID (#demo)

Access me by ID

Class (.demo)

Access me by class (1)

Access me by class (2)

Tag (article)

Access me by tag (1)

Access me by tag (2)

Query Selector

Access me by query

Query Selector All

Access me by query all (1)

Access me by query all (2)

```
> const demoId = document.getElementById('demo');
< undefined
> console.log(demoId);
    <div id="demo">Access me by ID</div>                          VM1110:1
< undefined
> demoId.style.border = '1px solid purple';
< "1px solid purple"
> const demoClass = document.getElementsByClassName('demo');
< undefined
> demoClass.style.border = '1px solid orange';
✖ ▶Uncaught TypeError: Cannot set property 'border' of undefined   VM1364:1
      at <anonymous>:1:24
> console.log(demoClass);
  ▶ (2) [div.demo, div.demo]                                      VM1366:1
< undefined
> for (i = 0; i < demoClass.length; i++) {
    demoClass[i].style.border = '1px solid orange';
  }
< "1px solid orange"
> const demoTag = document.getElementsByTagName('article');

  for (i = 0; i < demoTag.length; i++) {
    demoTag[i].style.border = '1px solid blue';
  }
< "1px solid blue"
> |
```

The loop changed the `border` property of all `article` elements to `blue`.

# Query Selectors

If you have any experience with the jQuery API, you may be familiar with jQuery's method of accessing the DOM with CSS selectors.

```
$('#demo'); // returns the demo ID element in jQuery
```

Copy

You can do the same in plain JavaScript with the `querySelector()` and

To access a single element, you can use the `querySelector()` method. In our HTML file, we have a `demo-query` element

```
<div id="demo-query">Access me by query</div>
```

Copy

The selector for an `id` attribute is the hash symbol (`#`). You can assign the element with the `demo-query` id to the `demoQuery` variable.

```
> const demoQuery = document.querySelector('#demo-query');
```

Copy

In the case of a selector with multiple elements, such as a class or a tag, `querySelector()` will return the first element that matches the query. You can use the `querySelectorAll()` method to collect all the elements that match a specific query.

In the example file, you have two elements with the `demo-query-all` class applied to them.

```
<div class="demo-query-all">Access me by query all (1)</div>
<div class="demo-query-all">Access me by query all (2)</div>
```

Copy

The selector for a `class` attribute is a period or full stop (`.`), so you can access the class with `.demo-query-all`.

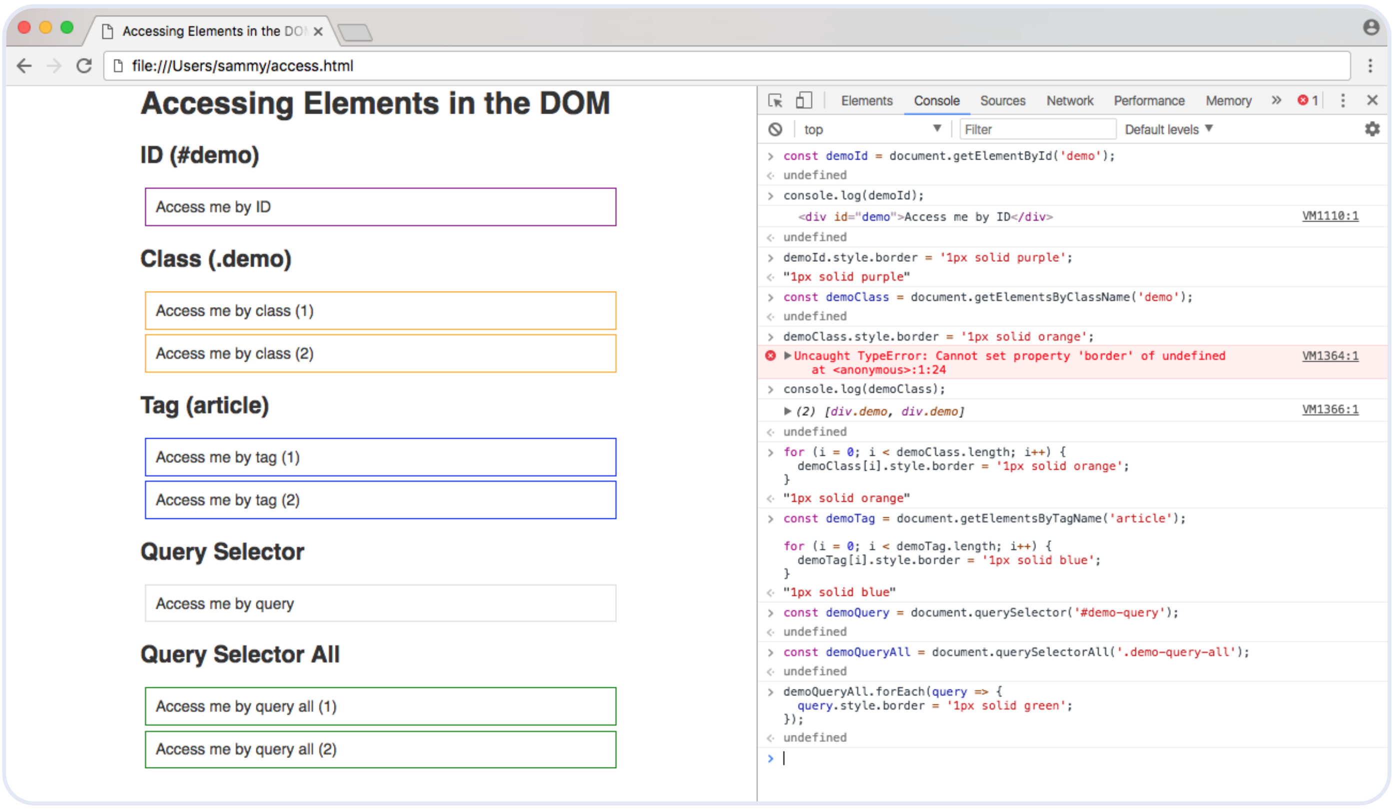
```
> const demoQueryAll = document.querySelectorAll('.demo-query-all');
```

Copy

Using the `forEach()` method, you can apply the color `green` to the `border` property of all matching elements.

```
> demoQueryAll.forEach(query => {
>   query.style.border = '1px solid green';
> });
```

Copy



Accessing Elements in the DOM

ID (#demo)

Access me by ID

Class (.demo)

Access me by class (1)

Access me by class (2)

Tag (article)

Access me by tag (1)

Access me by tag (2)

Query Selector

Access me by query

Query Selector All

Access me by query all (1)

Access me by query all (2)

```
> const demoId = document.getElementById('demo');
< undefined
> console.log(demoId);
    <div id="demo">Access me by ID</div>                         VM1110:1
< undefined
> demoId.style.border = '1px solid purple';
< "1px solid purple"
> const demoClass = document.getElementsByClassName('demo');
< undefined
> demoClass.style.border = '1px solid orange';
✖ ▶Uncaught TypeError: Cannot set property 'border' of undefined VM1364:1
      at <anonymous>:1:24
> console.log(demoClass);
   ▶ (2) [div.demo, div.demo]                                    VM1366:1
< undefined
> for (i = 0; i < demoClass.length; i++) {
    demoClass[i].style.border = '1px solid orange';
  }
< "1px solid orange"
> const demoTag = document.getElementsByTagName('article');

  for (i = 0; i < demoTag.length; i++) {
    demoTag[i].style.border = '1px solid blue';
  }
< "1px solid blue"
> const demoQuery = document.querySelector('#demo-query');
< undefined
> const demoQueryAll = document.querySelectorAll('.demo-query-all');
< undefined
> demoQueryAll.forEach(query => {
    query.style.border = '1px solid green';
  });
< undefined
> |
```

With `querySelector()`, comma-separated values function as an OR operator. For example, `querySelector('div, article')` will match `div` *or* `article`, whichever appears first in the document. With `querySelectorAll()`, comma-separated values function as an AND operator, and `querySelectorAll('div, article')` will match all `div` *and* `article` values in the document.

Using the query selector methods is extremely powerful, as you can access any element or group of elements in the DOM the same way you would in a CSS file. For a complete list of selectors, review CSS Selectors on the Mozilla Developer Network.

# Complete JavaScript Code

Below is the complete script of the work you did above. You can use it to access all the elements on our example page. Save the file as `access.js` and load it in to the HTML file right before the closing `body` tag.

access.js

```
// Change border of ID demo to purple
demoId.style.border = '1px solid purple';

// Change border of class demo to orange
for (i = 0; i < demoClass.length; i++) {
  demoClass[i].style.border = '1px solid orange';
}

// Change border of tag demo to blue
for (i = 0; i < demoTag.length; i++) {
  demoTag[i].style.border = '1px solid blue';
}

// Change border of ID demo-query to red
demoQuery.style.border = '1px solid red';

// Change border of class query-all to green
demoQueryAll.forEach(query => {
  query.style.border = '1px solid green';
});
```

Your final HTML file will look like this:

access.html

Copy

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>Accessing Elements in the DOM</title>

  <style>
    html { font-family: sans-serif; color: #333; }
    body { max-width: 500px; margin: 0 auto; padding: 0 15px; }
    div, article { padding: 10px; margin: 5px; border: 1px solid #dedede; }
  </style>

</head>
```

```
    <h2>Class (.demo)</h2>
    <div class="demo">Access me by class (1)</div>
    <div class="demo">Access me by class (2)</div>

    <h2>Tag (article)</h2>
    <article>Access me by tag (1)</article>
    <article>Access me by tag (2)</article>

    <h2>Query Selector</h2>
    <div id="demo-query">Access me by query</div>

    <h2>Query Selector All</h2>
    <div class="demo-query-all">Access me by query all (1)</div>
    <div class="demo-query-all">Access me by query all (2)</div>

    <script src="access.js"></script>

  </body>

  </html>
```

You can continue to work on these template files to make additional changes by accessing HTML elements.

# Conclusion

In this tutorial, we went over 5 ways to access HTML elements in the DOM — by ID, by class, by HTML tag name, and by selector. The method you will use to get an element or group of elements will depend on browser support and how many elements you will be manipulating. You should now feel confident to access any HTML element in a document with JavaScript through the DOM.

Thanks for learning with the DigitalOcean Community. Check out our offerings for compute, storage, networking, and managed databases.

Learn more about us →

Tutorial Series: Understanding the DOM – Document Object Model

The Document Object Model, usually referred to as the DOM, is an essential part of making websites interactive. It is an interface that allows a programming language to manipulate the content, structure, and style of a website. JavaScript is the client-side scripting language that connects to the DOM in an internet browser.

Subscribe

JavaScript Development

Browse Series: 8 articles

Expand to view all

Tania Rascia Author

Lisa Tagliaferri Editor

Madison Scott-Clary Editor
Tech Writer at DigitalOcean

Still looking for an answer? Ask a question

Search for more help

Was this helpful? Yes No

Comments

3 Comments

This textbox defaults to using `Markdown` to format your answer.

You can type `!ref` in this text area to quickly search our full set of tutorials, documentation & marketplace offerings and insert the link!

Sign In or Sign Up to Comment

jantjehofje • September 3, 2020

Thanks, really helped me out. Great tutorial and easy to follow

Reply

Peter Roche • April 25, 2019

Hi Tania, I too thought it was a great tutorial – all of them are so far. Really helpful. There is one thing, though, that I think could be improved. When you are processing elements that were accessed using .getElementsByClassName, or .getElementsByTagName, you handle the returned variable with a for loop, but when you are processing elements that were accessed using .querySelectorAll you handle the returned variable with the .forEach method. I think it would have been good to have some explanation for why – which, if I understand correctly, is because the "getElementsBy...()" methods return "HTMLCollection" objects, which cannot be processed with .forEach like arrays, while the querySelectorAll() method returns "NodeList" objects, which, in modern browsers, CAN be processed with .forEach like arrays. I think a little explanation about that would have been helpful.

Reply

J Miguel Aguilera • March 9, 2019

Reply

Try DigitalOcean for free

Click below to sign up and get **$200 of credit** to try our products over 60 days!

Sign up →

Popular Topics

Ubuntu

Linux Basics

JavaScript

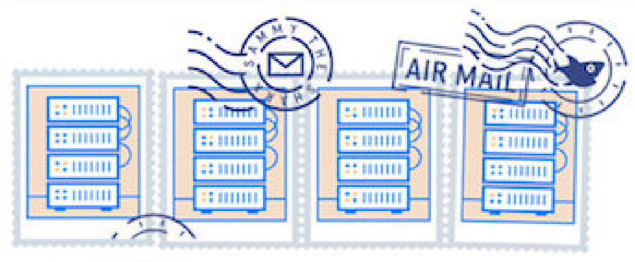
Python

MySQL

Docker

Kubernetes

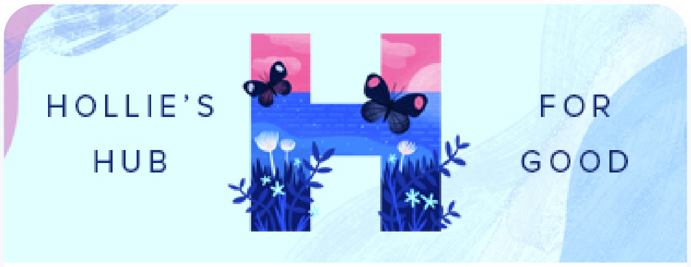
All tutorials →



Get our biweekly newsletter

Sign up for Infrastructure as a Newsletter.

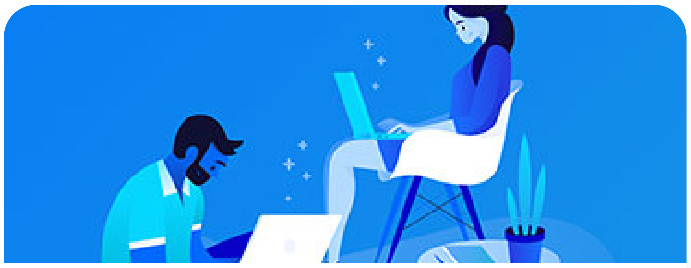
Sign up →



Hollie's Hub for Good

Working on improving health and education, reducing inequality, and spurring economic growth? We'd like to help.

Learn more →



Become a contributor

You get paid; we donate to tech nonprofits.

Learn more →

Kubernetes Course Learn Python 3 Machine Learning in Python
Getting started with Go Intro to Kubernetes

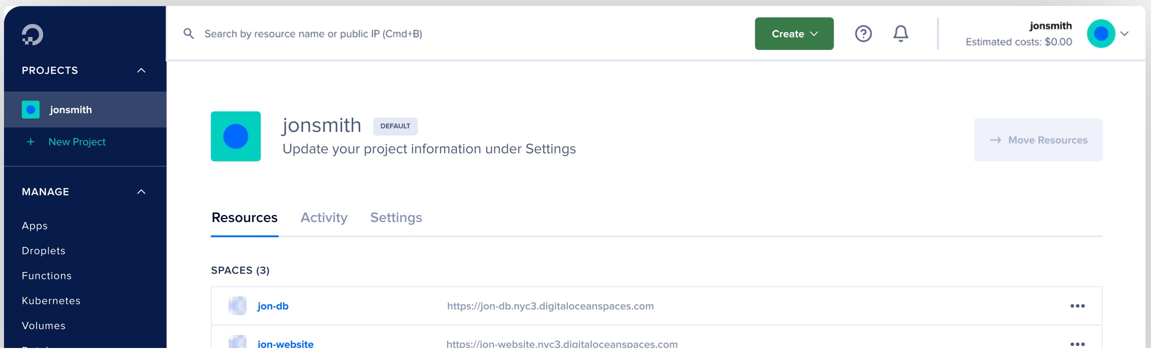
DigitalOcean Products

Cloudways Virtual Machines Managed Databases Managed Kubernetes
Block Storage Object Storage Marketplace VPC Load Balancers

Welcome to the developer cloud

DigitalOcean makes it simple to launch in the cloud and scale up as you grow – whether you're running one virtual machine or ten thousand.

Learn more →



Company	Products	Community	Solutions	Contact
About	Products	Tutorials	Website Hosting	Support

Partners

Channel Partners

Referral Program

Affiliate Program

Press

Legal

Security

Investor Relations

DO Impact

Functions

Cloudways

Managed Databases

Spaces

Marketplace

Load Balancers

Block Storage

Tools & Integrations

API

Pricing

Documentation

Release Notes

Uptime

Currents Research

Hatch Startup Program

deploy by DigitalOcean

Shop Swag

Research Program

Open Source

Code of Conduct

Newsletter Signup

Meetups

Streaming

VPN

SaaS Platforms

Cloud Hosting for Blockchain

Startup Resources