Element: getElementsByClassName() method

- Syntax
  - Parameters
  - Return value
- Usage notes
- Examples
  - Matching a single class
  - Matching multiple classes
  - Examining the results
  - Filtering the results using array methods
- Specifications
- Browser compatibility

The Element method `getElementsByClassName()` returns a live HTMLCollection which contains every descendant element which has the specified class name or names.

The method getElementsByClassName() on the Document interface works essentially the same way, except it acts on the entire document, starting at the document root.

Syntax

```
getElementsByClassName(names)
```

Parameters

- `names`
  - : A string containing one or more class names to match on, separated by whitespace.

Return value

An HTMLCollection providing a live-updating list of every element which is a member of every class in `names`.

Usage notes

As always, the returned collection is *live*, meaning that it always reflects the current state of the DOM tree rooted at the element on which the function was called. As new elements that match `names` are added to the subtree, they immediately appear in the collection. Similarly, if an existing element that doesn't match `names` has its set of classes adjusted so that it matches, it immediately appears in the collection.

The opposite is also true; as elements no longer match the set of names, they are immediately removed from the collection.

> **NOTE:** In quirks mode, the class names are compared in a case-insensitive fashion. Otherwise, they're case sensitive.

Examples

Matching a single class

To look for elements that include among their classes a single specified class, we just provide that class name when calling `getElementsByClassName()`:

```
element.getElementsByClassName("test");
```

This example finds all elements that have a class of `test`, which are also a descendant of the element that has the `id` of `main`:

```
document.getElementById("main").getElementsByClassName("test");
```

Matching multiple classes

To find elements whose class lists include both the `red` and `test` classes:

```
element.getElementsByClassName("red test");
```

Examining the results

You can use either the item() method on the returned `HTMLCollection` or standard array syntax to examine individual elements in the collection. However, the following code will not work as one might expect because `"matches"` will change as soon as any `"color-box"` class is removed.

```
const matches = element.getElementsByClassName("color-box");

for (let i = 0; i < matches.length; i++) {
  matches[i].classList.remove("color-box");
  matches.item(i).classList.add("hue-frame");
}
```

Instead, use another method, such as:

```
const matches = element.getElementsByClassName("color-box");

while (matches.length > 0) {
  matches.item(0).classList.add("hue-frame");
  matches[0].classList.remove("color-box");
}
```

This code finds descendant elements with the `"color-box"` class, adds the class `"hue-frame"`, by calling `item(0)`, then removes `"color-box"` (using array notation). Another element (if any are left) will then become `item(0)`.

Filtering the results using array methods

We can also use Array methods on any HTMLCollection by passing the HTMLCollection as the method's `this` value. Here we'll find all div elements that have a class of `test`:

```
const testElements = document.getElementsByClassName("test");
const testDivs = Array.prototype.filter.call(
  testElements,
  (testElement) => testElement.nodeName === "DIV",
);
```

Specifications

Browser compatibility

□□□ #3
□ □□□ □□□ 2026-10-31 03:27:53 UTC
□ cocobook-sync □□□ 2026-11-04 17:34:24 UTC