JavaScript DOM remove element

Asked 12 years, 11 months ago Modified 4 years, 5 months ago Viewed 371k times

216

I'm trying to test if a DOM element exists, and if it does exist delete it, and if it doesn't exist create it.

```
var duskdawnkey = localStorage["duskdawnkey"];
var iframe = document.createElement("iframe");
var whereto = document.getElementById("debug");
var frameid = document.getElementById("injected_frame");
iframe.setAttribute("id", "injected_frame");
iframe.setAttribute("src", 'http://google.com');
iframe.setAttribute("width", "100%");
iframe.setAttribute("height", "400");

if (frameid) // check and see if iframe is already on page
{ //yes? Remove iframe
    iframe.removeChild(frameid.childNodes[0]);
} else // no? Inject iframe
{
    whereto.appendChild(iframe);
    // add the newly created element and it's content into the DOM
    my_div = document.getElementById("debug");
    document.body.insertBefore(iframe, my_div);
}
```

Checking if it exists works, creating the element works, but deleting the element doesn't. Basically all this code does is inject an iframe into a webpage by clicking a button. What I would like to happen is if the iframe is already there to delete it. But for some reason I am failing.

javascript dom

Share
Improve this question
Follow

edited Feb 8, 2015 at 16:31
Mark Amery
154k ● 89 ● 426 ● 469

asked Jan 12, 2012 at 6:06
Joshua Redfield
2,247 ● 2 ● 14 ● 10

possible duplicate of JavaScript: remove element by id – Zaz Dec 30, 2014 at 17:34

4 Answers

Sorted by: Highest score (default)

`removeChild` should be invoked on the parent, i.e.:

357

```
parent.removeChild(child);
```

In your example, you should be doing something like:

```
if (frameid) {
    frameid.parentNode.removeChild(frameid);
}
```

Share

Improve this answer

Follow

edited Dec 19, 2014 at 20:28

Mark Amery

154k ● 89 ● 426 ● 469

answered Jan 12, 2012 at 6:11

casablanca

70.6k ● 7 ● 138 ● 154

Thanks figured it out right before I read your post. Had to change it to whereto.removeChild(whereto.childNodes[0]); – Joshua Redfield Jan 12, 2012 at 6:18

6 That would also work assuming that your frame is always the first child of the `debug` div. Using `parentNode` is a more generic solution that will work with any element. – casablanca Jan 12, 2012 at 6:19

1 This solution might not be enough. If anyone is reading this please take a look at Glenn's suggestion – Sebas Jul 30, 2015 at 3:42

99

In most browsers, there's a slightly more succinct way of removing an element from the DOM than calling `.removeChild(element)` on its parent, which is to just call `element.remove()`. In due course, this will probably become the standard and idiomatic way of removing an element from the DOM.

The `.remove()` method was added to the DOM Living Standard in 2011 (commit), and has since been implemented by Chrome, Firefox, Safari, Opera, and Edge. It was not supported in any version of Internet Explorer.

If you want to support older browsers, you'll need to shim it. This turns out to be a little irritating, both because nobody seems to have made a all-purpose DOM shim that contains these methods, and because we're not just adding the method to a single prototype; it's a method of `ChildNode`, which is just an interface defined by the spec and isn't accessible to JavaScript, so we can't add anything to its prototype. So we need to find all the prototypes that inherit from `ChildNode` and are actually defined in the browser, and add `.remove` to them.

Here's the shim I came up with, which I've confirmed works in IE 8.

```
(function () {
    var typesToPatch = ['DocumentType', 'Element', 'CharacterData'],
        remove = function () {
            // The check here seems pointless, since we're not adding this
```

```
            // method to the prototypes of any any elements that CAN be the
            // root of the DOM. However, it's required by spec (see point 1 of
            // https://dom.spec.whatwg.org/#dom-childnode-remove) and would
            // theoretically make a difference if somebody .apply()ed this
            // method to the DOM's root node, so let's roll with it.
            if (this.parentNode != null) {
                this.parentNode.removeChild(this);
            }
        };

    for (var i=0; i<typesToPatch.length; i++) {
        var type = typesToPatch[i];
        if (window[type] && !window[type].prototype.remove) {
            window[type].prototype.remove = remove;
        }
    }
})();
```

This won't work in IE 7 or lower, since extending DOM prototypes isn't possible before IE 8. I figure, though, that on the verge of 2015 most people needn't care about such things.

Once you've included them shim, you'll be able to remove a DOM element `element` from the DOM by simply calling

```
element.remove();
```

Share

Improve this answer

Follow

edited Nov 16, 2019 at 13:13

answered Dec 19, 2014 at 21:46

Mark Amery

154k ● 89 ● 426 ● 469

1 Just leaving this here: polyfill.io/v2/docs/features/#Element_prototype_remove if you include that automagic polyfill service you will get support back to IE 7. – complistic Aug 1, 2016 at 0:39

5 This is definitely the way to do it in 2017 as long as you don't care about IE. See: developer.mozilla.org/en-US/docs/Web/API/ChildNode/remove – nevf May 31, 2017 at 21:37

48

Seems I don't have enough rep to post a comment, so another answer will have to do.

When you unlink a node using removeChild() or by setting the innerHTML property on the parent, you also need to make sure that there is nothing else referencing it otherwise it won't actually be destroyed and will lead to a memory leak. There are lots of ways in which you could have taken a reference to the node before calling removeChild() and you have to make sure those references that have not gone out of scope are explicitly removed.

Doug Crockford writes here that event handlers are known a cause of circular references in IE and suggests removing them explicitly as follows before calling removeChild()

```
function purge(d) {
    var a = d.attributes, i, l, n;
    if (a) {
        for (i = a.length - 1; i >= 0; i -= 1) {
            n = a[i].name;
            if (typeof d[n] === 'function') {
                d[n] = null;
            }
        }
    }
    a = d.childNodes;
    if (a) {
        l = a.length;
        for (i = 0; i < l; i += 1) {
            purge(d.childNodes[i]);
        }
    }
}
```

And even if you take a lot of precautions you can still get memory leaks in IE as described by Jens-Ingo Farley here.

And finally, don't fall into the trap of thinking that Javascript **delete** is the answer. It seems to be suggested by many, but won't do the job. Here is a great reference on understanding **delete** by Kangax.

Share Improve this answer Follow

answered Feb 15, 2013 at 3:14

Glenn Lawrence

3,063 ● 3 ● 33 ● 38

1 perhaps you can show some jsFiddle to proof this. Thanks – Muhaimin Feb 27, 2014 at 10:51

1 I confirm this behaviour. My framework uses a Javascript object mapping tree over the dom layout. Each js object references its dom element. Even though I call `element.parentNode.removeChild` to remove elements, they stay alive and can still get referenced. They just are not visible in the regular dom tree. – Sebas Jul 30, 2015 at 3:25

Yes, and then removing the global pointer to that js mapping object magically unlocks the garbage collector. This is an important addition to the accepted answer. – Sebas Jul 30, 2015 at 3:41

16

Using **Node.removeChild()** does the job for you, simply use something like this:

```
var leftSection = document.getElementById('left-section');
leftSection.parentNode.removeChild(leftSection);
```

In DOM 4, the remove method applied, but there is a poor browser support according to W3C:

> The method node.remove() is implemented in the DOM 4 specification. But because of poor browser support, you should not use it.

But you can use remove method if you using jQuery...

```
$('#left-section').remove(); //using remove method in jQuery
```

Also in new frameworks like you can use conditions to remove an element, for example `*ngIf` in Angular and in React, rendering different views, depends on the conditions...

Share Improve this answer Follow

answered Sep 21, 2017 at 5:43

Alireza

105k ● 27 ● 277 ● 173

1 If you are creating modal node in js using createElement, then just make sure to check if the node exists before removing it. *if(leftSection){ ..your code }* should do the job.
– Afsan Abdulali Gujarati Nov 19, 2018 at 23:55