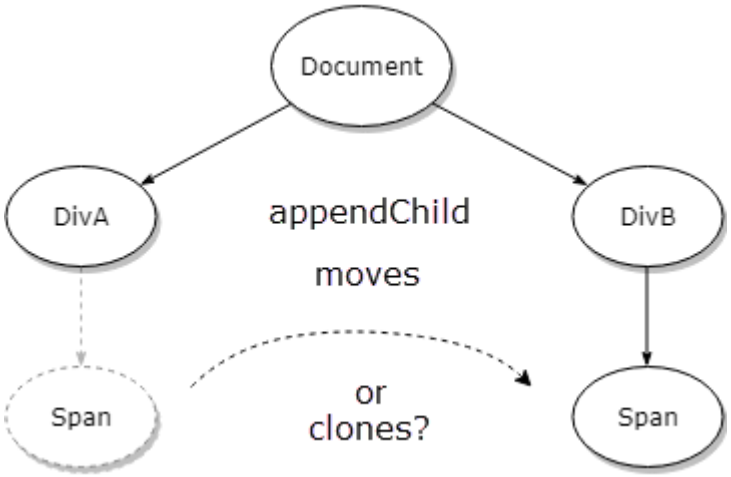
Here is why appendChild moves a DOM node between parents

Max Koretskyi aka Wizard Follow
Sep 18, 2018 · 5 min read

A light introduction into DOM fundamentals through an insightful quiz



. . .

I'm a firm believer of the need to know web fundamentals. So occasionally I ask interesting questions related to web development architecture or web platform API. These questions help me understand how passionate a developer is about their job; how far candidates ventured in their desire for knowledge.

So last week I asked the following question on Twitter:

Given this HTML

```
<div class="a">
    <span></span>
</div>
<div class="b"></div>
```

and this JavaScript that uses `appendChild` method:

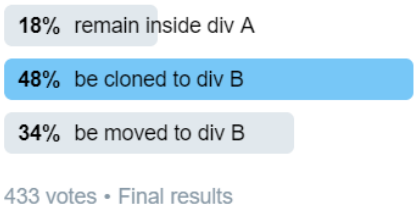
```
const span = document.querySelector('span');
const divB = document.querySelector('.b');

divB.appendChild(span);
```

is child `span` element going to:

- remain inside div A
- be cloned to div B
- be moved to div B

And I got pretty interesting results:



To be honest, I wasn't surprised to learn that majority of developers who answered my question thought that the span would be cloned. I ask this question often during job interviews and that is the most common answer I get. But let me say that I can see the logic behind this answer. It's a tricky question, really.

The correct answer is that the `span` will be moved to the new parent div B, not cloned. You can easily learn that by simply going to MDN and reading the documentation on the `appendChild` method. This is what the docs have to say:

> *The* `Node.appendChild()` *method adds a node to the end of the list of children of a specified parent node. If the given child is a reference to an existing node in the document,* `appendChild()` ***moves it from its current position to the new position*** *(there is no requirement to remove the node from its parent node before appending it to some other node).*

So that's where we can stop. However, since I like exploring stuff in-depth, I want to provide an elaborate explanation and teach you some DOM concepts along the way.

DOM nodes

Let's start with DOM nodes. What happens when you navigate to a website? A browser makes a request and gets the response that always includes HTML. HTML is a simple text, so how do we work with it in JavaScript? Well, a browser's rendering engine parses HTML and **creates JavaScript objects** corresponding to HTML tags. This is where the name "Document **Object Model**", commonly known as DOM, comes from. **One occurrence of an HTML tag results in one instance of a JavaScript object**.

So using the HTML in the question:

```
<div class="a">
    <span></span>
</div>
<div class="b"></div>
```

a browser will create two instances of HTMLDivElement and one instance of HTMLSpanELement. It will also add corresponding classes to both div elements. If you wanted to emulate the work a browser does, here is how you could do it:

```
// <div class="a">
const divA = document.createElement('div');
divA.classList.add('a');


// <div class="b"></div>
const divB = document.createElement('div');
divB.classList.add('b');
```

```
// <span></span>
const span = document.createElement('span');
```

```
// a few checks
divA.className; // "a"
divB.className; // "b"
divA instanceof HTMLDivElement // true
divB instanceof HTMLDivElement // true
span instanceof HTMLSpanElement // true
```

Node tree

Okay, we have JavaScript objects now. These objects reside in memory and are referred to as nodes. What's important is that nodes

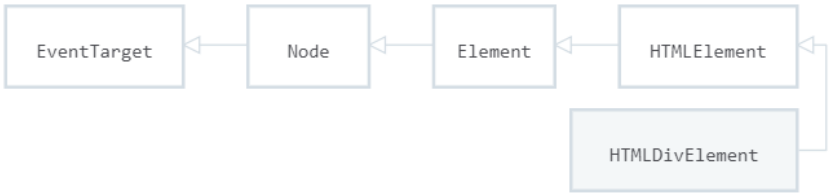
don't live separately. They form a particular data structure—a **tree**.

How do we know that? Well, most of the stuff developers use is defined by specifications. JavaScript is defined by the EcmaScript spec and web platform is defined by whatwg.

So if we go and read the spec here's what we'll find:

> `Document`, `DocumentType`, `DocumentFragment`, **`Element`**, `Text`, `ProcessingInstruction`, *and* `Comment` *objects (simply called nodes)* ***participate in a tree****, simply named the node tree.*

Note that both `div` and `span` elements are `Element` nodes. This class hierarchy diagram demonstrates that:



Now here is how the spec defines a tree:

> *A tree is a finite hierarchical tree structure... An object that participates in a tree has a parent, which is either null or an object, and has children...*

And in web we have a particular type of a node tree called Document tree:

> *A document tree is a node tree whose root is a document.*

This is where the *Document* part in the **Document** Object Model (DOM) comes from.

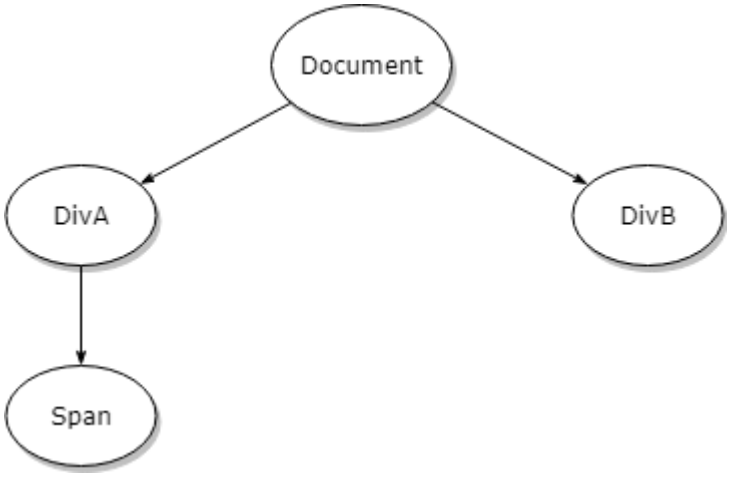
Creating document tree

So we've created DOM nodes earlier separately. We can now compose them into a tree by appending div elements to the Document and span element to the DivA.

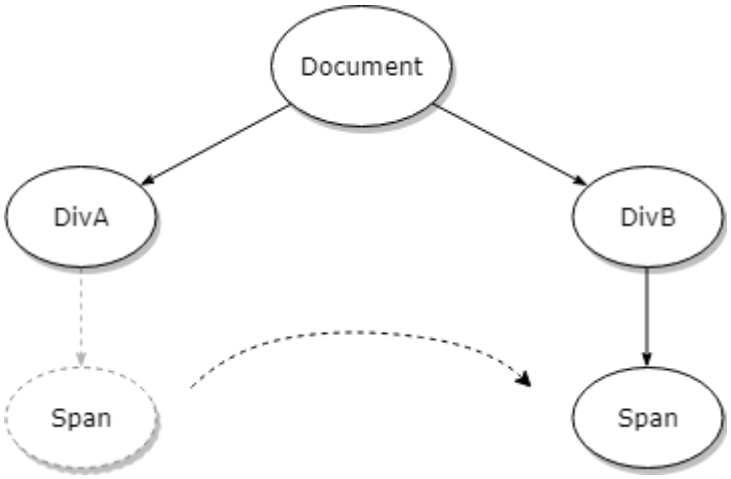
```
document.appendChild(divA);
document.appendChild(divB);
```

```
divA.appendChild(span);
```

This gives us the the following tree:



In the question that I asked we want to move the node to the other parent:



Using the appendChild method:

```
divB.appendChild(span);
```

Why is the node not cloned?

It's a reasonable to assume that the node could be cloned. However, there are a few implications that make this scenario very improbable:

- cloning a node through appendChild results in the node without a direct reference to it:

```
const span = document.querySelector('span');
const divB = document.querySelector('.b');

divB.appendChild(span);
```

In the code above if the node is cloned the `span` variable will point to either cloned or original instance, so we're losing a reference to either original or new DOM node instance

- It's not clear what to do if the element that is being moved has a very deeply nested children tree. Should it be cloned as well? Deep-cloning an object is very expensive and quite complex, especially if circular references are involved
- cloning a node can lead to duplicate IDs

Why doesn't node become a child of two parents?

Well, if we go back to the definition of a tree again and re-read it we can see the answer:

> *... An object that participates in a tree **has a parent, which is either null or an object**, and has children...*

So every node in a tree except the root node **has exactly one connection** upward to a node called parent. Not many, just zero or one.

It means that if we want to move a node to a different parent, it needs to be removed from the previous parent first, because a node can't have two parents!

. . .

This article is going to be the first in a series designed to teach you web fundamentals. I'll be asking interesting questions on Twitter and providing an

answer with in-depth explanations in the form of short articles. We'll glance into specifications and explore the basis on which web frameworks are built.

Take part in a journey to learn fundamentals with me—follow me on Twitter and on Medium.