Lesson Tuesday

Intermediate JavaScript (/intermediate-javascript) / Object-Oriented JavaScript (/intermediate-javascript/object-oriented-javascript) / Introduction to the Node Object

Text

In this lesson, we'll learn a bit more about the `Node` object. We've previously encountered the `Node` object when we discussed the series of objects that HTML element objects inherit from. Remember this graphic for the `HTMLHeadingElement` object?

In the image we can see that the `HTMLHeadingElement` object (and all other HTML element objects like `HTMLLIElement`, `HTMLAnchorElement`, and so on) inherits from 4 other objects:

- `HTMLElement`
- `Element`
- `Node`

- `EventTarget`

We've talked about every object listed above, except for `Node`, so in this lesson we'll take the time to understand what it is and what it does. However, we won't work with `Node` properties and methods much. Why? The `Element` object offers the same functionality, and it's more conceptually intuitive.

That said, it's still important to have a basic understanding of the `Node` object, and that's what this lesson is all about: developing a basic understanding of the `Node` object. Doing this will also give us an opportunity to review the DOM and other concepts related to the Web APIs we've learned about so far.

An Introduction to `Node` and a Review of the DOM

Let's revisit the object's that we've worked with so far:

- `HTMLElement` represents any HTML element in the HTML DOM. This is a generic object type for more specific objects like `HTMLLIElement`, `HTMLImageElement`, and so on.
- `Element` represents any element in a DOM. This is even more generic than `HTMLElement`, because this can include HTML elements in the HTML DOM, or SVG (https://developer.mozilla.org/en-US/docs/Web/SVG) elements in an SVG DOM.
- `EventTarget` represents any object that can be the target of events and that we can attach an event listener to.

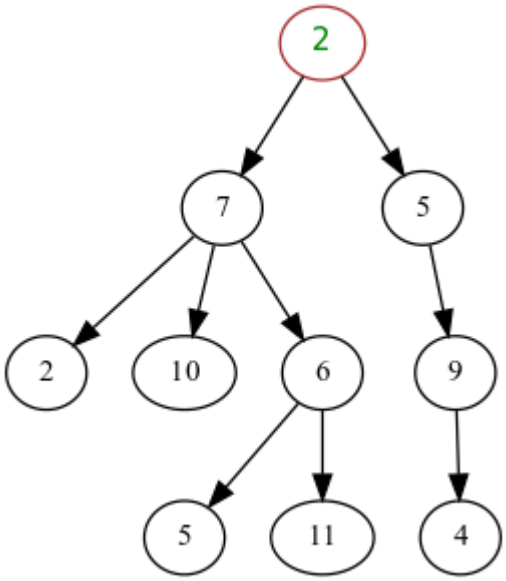
So, how does `Node` fit into this?

- `Node` represents a point within a DOM; this could be an element, text, a comment, and more.

We could describe `Node` as an even more generic representation of the `Element` object, but to get to the heart of it, we need to understand what a node in a tree is. Not a maple or pine tree, but the 'tree' data structure.

A tree is a collection of nodes that are organized hierarchically. There's always a root node, branches off of that node that lead to other nodes. A node is just a point along the tree that contains a value, any value.

The following image is an example of a tree from Wikipedia's entry on the tree data structure (https://en.wikipedia.org/wiki/Tree_(data_structure)).



There are many types of trees (with specific names), but we don't need to concern ourselves with the details. We care about trees because the Document Object Model is a tree. In technical terms, the DOM is a hierarchical collection of nodes, and each node in the DOM is an object that represents some aspect of the document, like text, a comment, an element, or the document itself!

So, for our browser Web APIs, the `Node` object represents a node in the DOM tree — not just any tree, but specifically the DOM! Because of this, `Node` contains properties and methods for adding

and removing nodes, and traversing the DOM. **Traversing the DOM** simply means to move to different elements within the DOM.

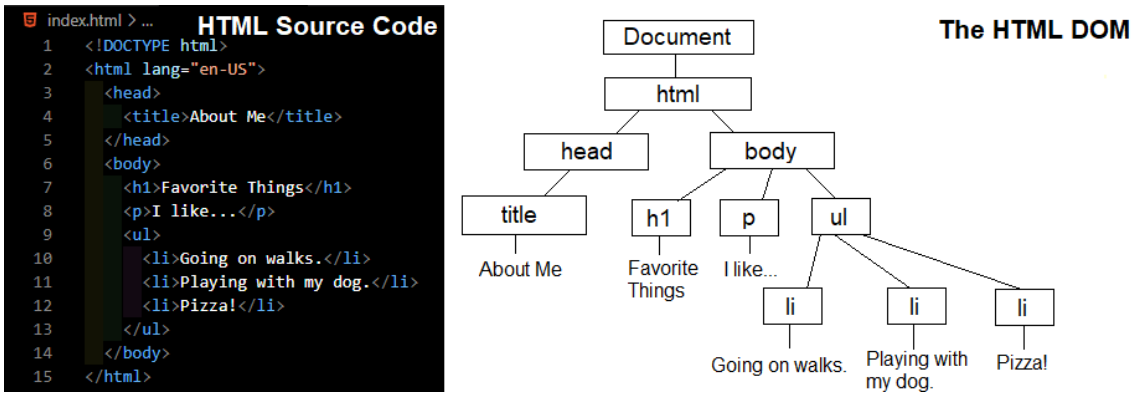
Keep in mind that `Node` is a generic object type that represents a generic node in the DOM tree. There are specific object types that represent specific types of nodes. To learn more about these other object types, visit the MDN documentation on `Node` (https://developer.mozilla.org/en-US/docs/Web/API/Node), which lists them all in the second paragraph.

Visualization of an HTML DOM Tree

Let's look at a previous example that compares HTML source code with a corresponding HTML DOM tree. As we can see the image on the right has a root node (`document`) and a series of branches that go on the child nodes.

In fact the terms "parent", "child or "children", and "siblings" are how we describe the hierarchical relationships in the DOM tree. In the image below, we can list out a few relationships:

- `<body>` is the child of `<html>` , and `<html>` is the parent of `<body>`
- The three `<li>` elements are all siblings, as well as children of the `<ul>` element.
- `<body>` is the parent of `<h1>` , `<p>` , and `<ul>`

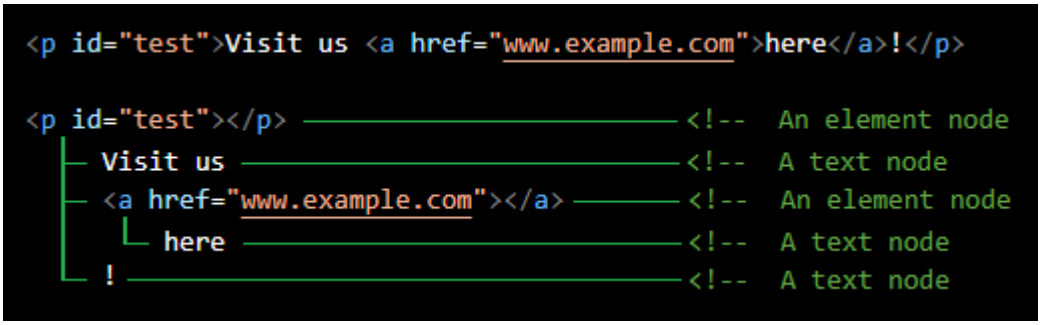


Take note that the above image is a bit deceiving. It only describes some of the nodes present in the HTML — the elements. Text and commentary, among others, are also nodes. Let's look at another example to better understand different types of nodes in the DOM tree.

Visualization of Node Types

The following image breaks down the different node types of the P tag with a nested anchor. A **node type** is just like an object type, having a name and representing a specific type of node in the DOM tree.

```
<p id="test">Visit us <a href="www.example.com">here</a>!</
p>
```



To see a full list of node types, visit the following documentation on MDN:

- `Node.nodeType` (https://developer.mozilla.org/en-US/docs/Web/API/Node/nodeType)

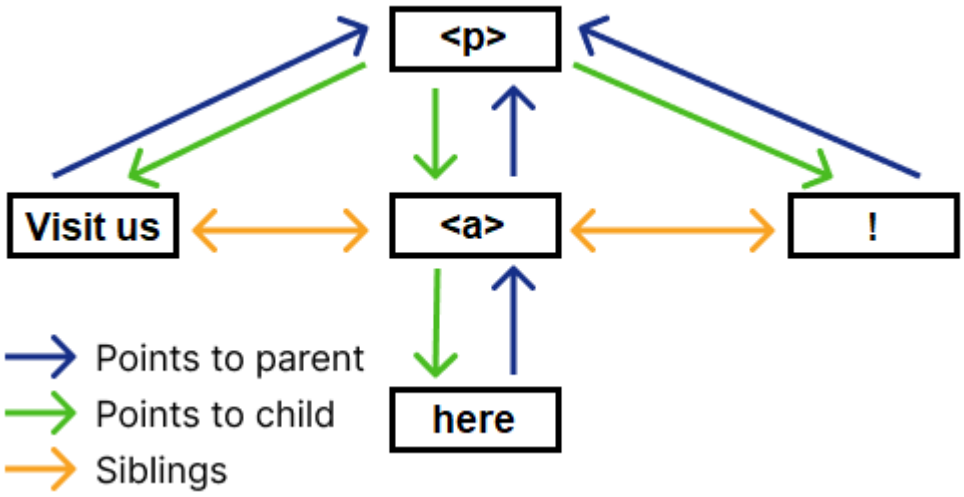
Take note that attributes are not counted as separate nodes in the DOM, though they do have a dedicated node type. Instead, they are counted as part of the element node they are attached to. This is an odd and specific detail that is due to the discrepancy between how JavaScript was originally written and structured, and how it's since

evolved. Don't worry about retaining this information, this is just a technical side-note for those who may be wondering about attributes.

Visualization of Node Relationships

Next, let's look at one more example that demonstrates the relationships between nodes and how we describe them with "parent", "child", and "siblings". Understanding this will help us understand the names of the properties and methods that the `Node` object (and others) offer us.

In the following image, we'll see the same paragraph element as in the previous example, still broken into a hierarchy of nodes, but this time with the relationships between each node labeled.



Here are a few relationships between the nodes in the above image:

- The `<p>` element node is the parent of the text node `"Visit us "`, the element node `<a>`, and the text node `"!"`.
- The text node `"Visit us "`, the element node `<a>`, and the text node `"!"` are all siblings.
- The text node `"here"` is the child of the element node `<a>`.

`Node` Properties and Methods

At this point a few things should be relatively clear:

- The data structure of the DOM is a tree, which is a hierarchical collection of nodes.
- Each node in the DOM is an object. Generally speaking, a node in a tree can be any value, but for the DOM they are all objects.
- The `Node` object type represents a node within the DOM.
- There are many node types, each of which serves to distinguish the nodes in a tree. Common ones are element nodes, text nodes, and comment nodes.
- `Node` is the most generic object type that represents node in the DOM tree, and there are other objects that represent specific node types.

So what's next? Let's look at a few `Node` properties and methods with the goal of exposing ourselves to what's out there. Actually implementing any `Node` methods or properties is further exploration. All of the DOM manipulation and traversal that we need to do, we can do with the `Element`, `Document`, and `HTMLElement` objects.

However, we can call the following `Node` properties and methods on the HTML element objects (`HTMLHeadingElement`, `HTMLImageElement`, and so on) that we work with. We'll also see some of these methods and properties used in code examples online. So, it's important to develop a basic familiarity with `Node` properties and methods, as well as know where to reference information about them.

Creating Nodes

There's a couple ways to create nodes, each of which belongs to the `document` object:

- `document.createComment()` (https://developer.mozilla.org/en-US/docs/Web/API/Document/createComment)
- `document.createTextNode()` (https://developer.mozilla.org/en-US/docs/Web/API/Document/createTextNode)
- `document.createAttribute()` (https://developer.mozilla.org/en-US/docs/Web/API/Document/createAttribute)

Notably `document.createElement()` (https://developer.mozilla.org/en-US/docs/Web/API/Document/createElement) does not return a node object.

If we wanted to create an attribute or text, we'd use these methods instead:

- Setting `HTMLElement.innerText` to change the value of the text for an element.
- Accessing attribute properties or calling `Element.setAttribute()` method.

Adding New Nodes

A few `Node` methods to add new nodes to the DOM are:

- `Node.appendChild()` (https://developer.mozilla.org/en-US/docs/Web/API/Node/appendChild)
- `Node.insertBefore()` (https://developer.mozilla.org/en-US/docs/Web/API/Node/insertBefore)
- `Node.removeChild()` (https://developer.mozilla.org/en-US/docs/Web/API/Node/removeChild)

If we wanted to do the above with `Element` methods, we might choose from the following:

- `[Element.before()`](https://developer.mozilla.org/en-US/docs/Web/API/Element/before)
- `[Element.after()`](https://developer.mozilla.org/en-US/docs/Web/API/Element/after)

- `[Element.append()]`(https://developer.mozilla.org/en-US/docs/Web/API/Element/append)
- `[Element.prepend()]`(https://developer.mozilla.org/en-US/docs/Web/API/Element/prepend)
- `[Element.remove()]`(https://developer.mozilla.org/en-US/docs/Web/API/Element/remove)
- `[Element.insertAdjacentHTML()]` (https://developer.mozilla.org/en-US/docs/Web/API/Element/insertAdjacentHTML)

DOM Traversal

To move along the DOM, accessing parent, child, sibling elements, we might use the following `Node` properties. Note that this is not an exhaustive list!

- `Node.childNodes` (https://developer.mozilla.org/en-US/docs/Web/API/Node/childNodes)
- `Node.firstChild` (https://developer.mozilla.org/en-US/docs/Web/API/Node/firstChild)
- `Node.nextSibling` (https://developer.mozilla.org/en-US/docs/Web/API/Node/nextSibling)
- `Node.previousSibling` (https://developer.mozilla.org/en-US/docs/Web/API/Node/previousSibling)
- `Node.parentNode` (https://developer.mozilla.org/en-US/docs/Web/API/Node/parentNode)
- `Node.textContent` (https://developer.mozilla.org/en-US/docs/Web/API/Node/textContent)
- `Node.nodeName` (https://developer.mozilla.org/en-US/docs/Web/API/Node/nodeName)

Notice how these property names reference "parent", "child", and "sibling". This makes guessing what information each property contains easy.

To access the same information via `Element` or `HTMLElement`, we might use some of the following properties:

- `[Element.children]`(https://developer.mozilla.org/en-US/docs/Web/API/Element/children)
- `[Element.firstElementChild]`(https://developer.mozilla.org/en-US/docs/Web/API/Element/firstElementChild)
- `[Element.lastElementChild]`(https://developer.mozilla.org/en-US/docs/Web/API/Element/lastElementChild)
- `[Element.nextElementSibling]`(https://developer.mozilla.org/en-US/docs/Web/API/Element/nextElementSibling)
- `[Element.previousElementSibling]`(https://developer.mozilla.org/en-US/docs/Web/API/Element/previousElementSibling)
- `[Element.tagName]`(https://developer.mozilla.org/en-US/docs/Web/API/Element/tagName)
- `[HTMLElement.innerText]`(https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/innerText)

To review an explanation of the differences between `Node.textContent` and `HTMLElement.innerText`, check out this article on MDN:

- Differences between `Node.textContent` and `HTMLElement.innerText` (https://developer.mozilla.org/en-US/docs/Web/API/Node/textContent#differences_from_innertext)

Summary

All in all, everything that we can do with `Node`, we can also do via `Element` and `HTMLElement`. The `Node` object may give us finer grained control over the DOM tree, but we don't particularly need that fine grained control. It's up to you whether you want to explore the `Node` object and use its properties and methods in your code.

Here is a summary of key concepts:

- The data structure of the DOM is a tree, which is a hierarchical collection of nodes.
- Each node in the DOM is an object. Generally speaking, a node in a tree can be any value, but for the DOM they are all objects.

- The `Node` object type represents a node within the DOM.
- There are many node types, each of which serves to distinguish the nodes in a tree. Common ones are element nodes, text nodes, and comment nodes.
- `Node` is the most generic object type that represents node in the DOM tree, and there are other objects that represent specific node types.

Previous (/intermediate-javascript/object-oriented-javascript/address-book-movie-tickets-bank-account)
Next (/intermediate-javascript/object-oriented-javascript/optional-accessing-stylesheets-in-the-cssom)

Lesson 24 of 33
Last updated March 23, 2023

disable dark mode

(http://www.epicodus.com)