Document Object Model(DOM)

Presenter:
Agubata Odinaka

Without a document, JavaScript would have no way to make its presence felt. It’s HTML that creates the tangible interface through which JavaScript can reach its users. This relationship makes it vital that JavaScript be able to access, create, and manipulate every part of the document.

Within this model, each element in the HTML document becomes an object, as do all the attributes and text. JavaScript can access each of these objects independently, using built-in functions that make it easy to find and change what we want on the fly.

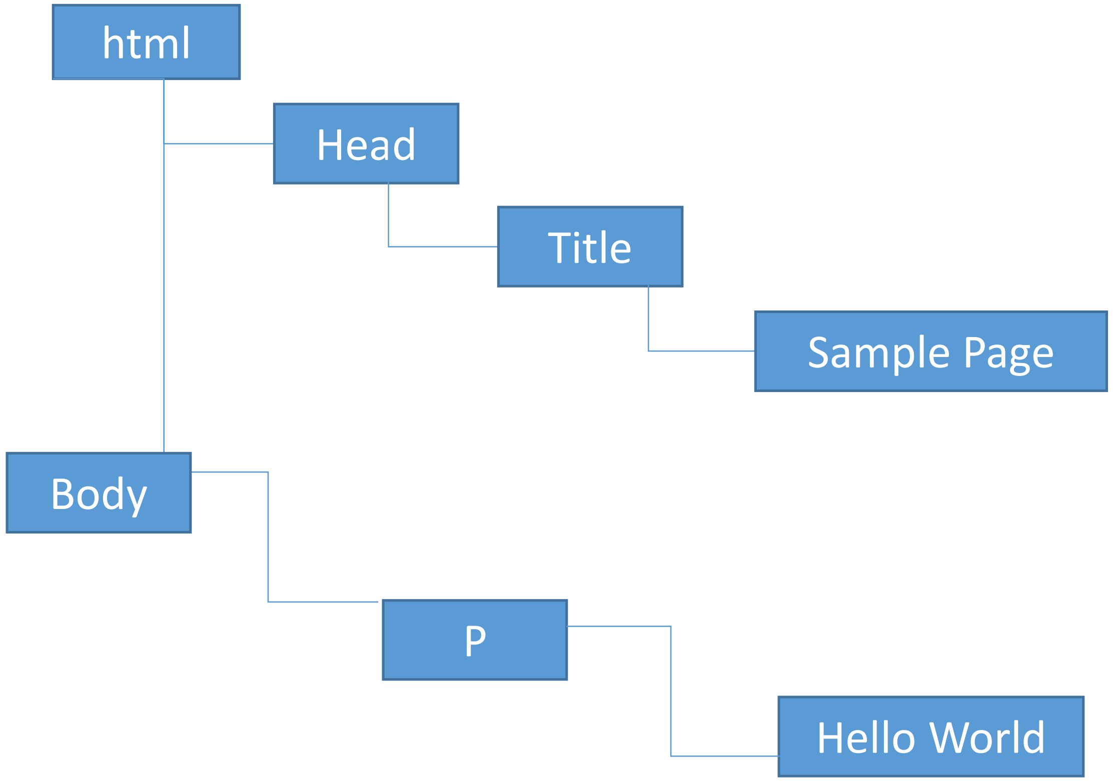
The DOM maps out an entire page as a document composed of a hierarchy of nodes. Each part of an HTML or XML page is a derivative of a node.

Consider the following HTML page:

```
<html>
    <head>
            <title>Sample Page</title>
    </head>
    <body>
            <p>Hello World!</p>
    </body>
</html>
```

Document Object Model(DOM)

This code can be diagrammed into a hierarchy of nodes using the DOM:



Document Object Model(DOM)

To create the DOM for a document, each element in the HTML is represented by what’s known as a node. A node’s position in the DOM tree is determined by its parent and child nodes.

Element Nodes

Element nodes represents one type of node, and they define most of the structure of the DOM, but the actual content of a document is contained in two other types of nodes: text nodes and attribute nodes.

An element node is distinguished by its element name (head, body, h1, etc.), but this doesn’t have to be unique. Unless you supply some identifying characteristic—like an id attribute—one paragraph node will appear much the same as another.

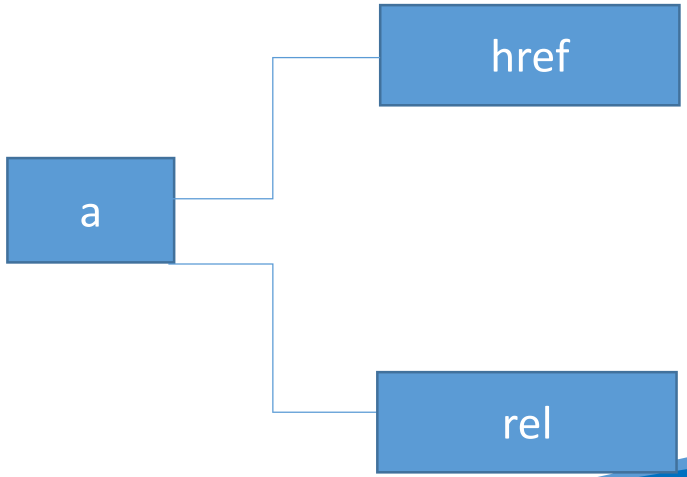
Text Nodes

In HTML code, any text that’s not contained between angled brackets will be interpreted as a text node in the DOM.

Structurally, text nodes are treated almost exactly like element nodes: they sit in the same tree structure and can be reached just like element nodes; however, they cannot have children.

Attributes Nodes

With tags and text covered by element and text nodes, the only pieces of information that remain to be accounted for in the DOM are attributes. At first glance, attributes would appear to be part of an element—and they are, in a way—but they still occupy their own type of nodes, handily called attribute nodes.



Accessing Specific Nodes

Finding an Element by ID

The most direct path to an element is via its id attribute. id is an optional HTML attribute that can be added to any element on the page, but each ID you use has to be unique within that document:

Example:

```
<p id="uniqueElement">
⋮
</p>
document.getElementById(‘uniqueElement’);
```

Finding an Element by TagName

Using IDs to locate elements is excellent if you want to modify one element at a time, but if you want to find a group of elements, getElementsByTagName is the method for you.

Example:

```
<ul>
<li>paragraph</li>
<li>unordered list</li>
<li>list item</li>
</ul>
```

Finding an Element by TagName

Restricting Tag Name Selection

```
<p>There are 3 different types of element in this body:</p>
<ul>
<li>paragraph</li>
<li>unordered list</li>
<li>list item</li>
</ul>

<p>There are 2 children of html:</p>
<ul>
<li>head</li>
<li>body</li>
</ul>
```

Finding an Element by TagName

We can retrieve all these list item elements using one line of JavaScript:

var listItems = document.getElementsByTagName("li");

By executing that code, you’re telling your program to search through all of the descendants of the document node, get all the nodes with a tag name of "li", and assign that group to the listItems variable.

Finding an Element by TagName

```
var lists = document.getElementsByTagName("ul");

var secondList = lists[1];

var secondListItems = secondList.getElementsByTagName("li");
```

Finding an Element by ClassName

It’s quite often very handy to find elements based on a class rather than a tag name.

```
document.getElementsByCLassName(‘className’);
```

Navigating the DOM Tree

Finding a Parent

Every element node—except for the document node—has a parent. Consequently, each element node has a property called parentNode. When we use this property, we receive a reference to the target element's parent.

Consider this HTML:

```
<p>
<a id="oliver" href="index.php">Oliver Twist</a>
</p>
```

Finding a Parent

Once we have a reference to the anchor element, we can get a reference to its parent paragraph using parentNode like so:

```
var oliver = document.getElementById("oliver");
var paragraph = oliver.parentNode;
```

Finding Children

The parent-child relationship isn’t just one way. You can find all of the children of an element using the childNodes property.
An element can only have one parent, but it can have many children, so childNodes is actually a node list that contains all of the element’s children, in source order.

Take, for instance, a list like this:

Finding Children

```
<ul id=“Johnson">
    <li>Okoro</li>
    <li>Daniel</li>
    <li>Isiguzo</li>
    <li>Chibuzor</li>
</ul>
```

Finding Children

To get the “Okoro” list item, we could just use:

var johnson = document.getElementById(‘johnson’);

var okoro = johnson.**firstChild**;

And to get the “Chibuzor” list item, we can use:

var buzor = johnson.**lastChild**;

Finding Siblings

To get the “Daniel” list item, we could just use:

var dan= okoro.**nextSibling**;

And to get the “Isiguzo” list item, we can use:

var guzo = buzor.**previousSibling**;

Interacting with Attributes

Attributes are more focused on reading and modifying the data related to an element.

As such, the DOM only offers two methods related to attributes, and both of them can only be used once you have an element reference.

Getting an Attribute

With a reference to an element already in hand, you can get the value of one of its attributes by calling the method getAttribute with the attribute name as an argument. Let's get the href attribute value for this link:

```
<a id="lelo" href="http://www.mylelojobs.com/">Let's all go there</a>
<a id="secDiv">Push it to the limit</div>
```

Getting an Attribute

We need to create a reference to the anchor element, then use getAttribute to retrieve the value:

```
var lelo = document.getElementById(“lelo");
var leloHref = lelo.getAttribute("href");
```

The value of leloHref will now be "http://www.mylelojobs.com/".

Setting an Attribute

As well as being readable, all HTML attributes are writable via the DOM.

To write an attribute value, we use the setAttribute method on an element, specifying both the attribute name we want to set and the value we want to set it to:

```
var sec = document.getElementById(“secDiv");
sec.setAttribute(“class", “secClass");
    OR
sec.className = ‘secClass’;
```

Changing Styles

Each element node has a property called style. style is an object that lets you change every aspect of an element’s appearance, from the color of its text, to its line height, to the type of border that’s drawn around it. For every CSS property that’s applicable to an element, style has an equivalent property that allows us to change that property’s value.

Suppose you have the following HTML page:

```
<html>
    <head>
            <title>Changing Styles</title>
    </head>
    <body>
             <div id="white">My name is Mr. Gregory</div>
    </body>
</html>
```

To change the text color of the div element whose id is “white”, we’d use style.color:

```
var white = document.getElementById(“white");
white.style.color = "#FFFFFF";
```

To change its background color, we’d use style.backgroundColor:

```
white.style.backgroundColor = "#000000";
```

Creating and manipulating nodes

Creating new nodes

So far, you’ve learned how to access various nodes inside of a document, but that’s just the beginning of what can be done using the DOM.

You can also add, remove, replace, and otherwise manipulate nodes within a DOM document.

This functionality is what makes the DOM truly dynamic.

We’ll be seeing createElement(), createTextNode(), appendChild() a lot in this section.

Creating new nodes

Suppose you have the following HTML page:

```
<html>
    <head>
        <title>createElement() Example</title>
    </head>
    <body>
    </body>
</html>
```

And you want to add the following code using the DOM:

```
<p>Hello World!</p>
```

Creating new nodes

The createElement() and createTextNode() methods can be used to accomplish this. Here’s how.

The first thing to do is create the <p> element:

```
var op = document.createElement(“p”);
```

Secondly, create the text node:

```
var oTxt = document.createTextNode(“Hello World!”);
```

Creating new nodes

Next you need to add the text node to the element. To do this, you can use the appendChild() method.

The appendChild() method exists on every node type and is used to add a given node to the end of another’s childNodes list. In this case, the text node should be added to the <p> element:

```
op.appendChild(oTxt);
```

Creating new nodes

```
<script type="text/javascript">
    function createMessage() {
            var op = document.createElement("p");
            var oTxt = document.createTextNode("Hello World! ");
            op.appendChild(oTxt);
            document.body.appendChild(op);
    }
createMessage();
</script>
```

Creating new nodes

When you run this code, the message “Hello World!” is displayed as if it were part of the HTML document all along.

Removing a HTML Element

Naturally, if you can add a node you can also remove a node, which is where the removeChild() method comes in.

This method accepts one argument, the node to remove, and then returns that node as the function value.

So if, for instance, you start out with a page already containing the "Hello World!" message and you wanted to remove it, you can use the method like this:

Suppose you have the following HTML page:

```
<html>
    <head>
            <title>createElement() Example</title>
    </head>
    <body>
            <p id="pTag">Following fundamentals</p>
    </body>
</html>
```

Removing a HTML Element

```
<script type=”text/javascript”>
    function removeMessage() {
        var oP = document.body.getElementById(“pTag”);
        document.body.removeChild(oP);
    }
    removeMessage();
</script>
```

Replacing a HTML Element

If you want to replace a message with a new one? In that case, you can use the replaceChild() method.

The replaceChild() method takes two arguments: the node to add and the node to replace. In this case, you create a new element with a new message and replace the <p> element with the “Hello World!” message.

Replacing a HTML Element

```
<script type="text/javascript">
    function replaceMessage() {
            var newP = document.createElement("p");
            var oTxt = document.createTextNode("Hello Universe! ");
            newP.appendChild(oTxt);
            var old = document.getElementById("pTag");
            old.parentNode.replaceChild(newP, old);
    }
    replaceMessage();
    </script>
```

Inserting a HTML Element

Suppose you have the following HTML page:

```
<html>
    <head>
            <title>createElement() Example</title>
    </head>
    <body>
            <div id="div1">
                    <p id="p1">This is a paragraph.</p>
                    <p id="p2">This is another paragraph.</p>
            </div>
    </body>
</html>
```

```
<script type=”text/javascript”>
    function insertCheck(){
            var para = document.createElement(“h1");
            var node = document.createTextNode("This is new.");
            para.appendChild(node);
            var element = document.getElementById("div1");
            var child = document.getElementById("p1");
            element.insertBefore(para,child);
    }
insertCheck();
</script>
```

Questions

Assignment

Create a Form using JavaScript