A Tutorial on JavaScript DOM

4/25/2012

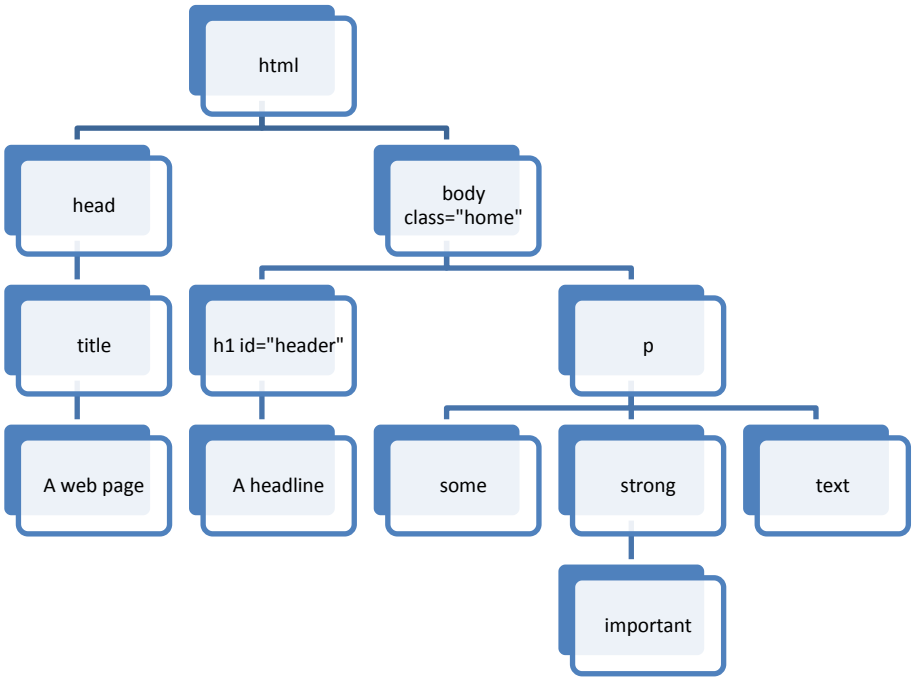
What is the Document Object Model (DOM)?

When a web browser loads an HTML file, it displays the contents of that file on the screen (styled with CSS). But at the same time the web browsers creates a "model" to memorize all the tags, attributes, and contents of the file, and the order in which they appear – this representation of the page is called the *Document Object Model*, or *DOM* for short.

The DOM provides the information needed for JavaScript to communicate with the elements on the web page. With the DOM, programmers can navigate through, change, and add to the HTML on the page. The DOM itself isn't actually JavaScript – it's a standard from the World Wide Web Consortium (W3C) that most browser manufacturers have adopted and added to their browsers. The DOM lets JavaScript communicate with and change a page's HTML.

Here is an example to see how the DOM actually works:

```
<html>

<head>

<title>A web page</title>

</head>

<body class=”home”>

<h1 id=”header”>A headline</h1>

<p>Some <strong>important</strong>text</p>

</body>

</html>
```



[Figure1]

On this and all other web sites, some tags wrap around other tags – like the <html> tag, which surrounds all other tags, or the <body> tag, which wraps around the tags and contents that appear in the browser window. You can represent the relationships between tags with a family tree diagram (see Figure1). The <html> tag is the "root" of the tree, while other tags represent different "branches" of the family tree.

In addition to HTML tags, web browsers also keep track of the text that appears inside a tag, as well as the attributes that are assigned to each tag. In fact, the DOM treats each of these tags/elements, attributes, and text as individual units called *nodes*.

Syntax

A web page is simply organized as a collection of tags, tag attributes, and text, or, a bunch of nodes. So for JavaScript to manipulate the contents of a page, it needs a way to communicate with a page's nodes. There are two main methods for selecting nodes: *getElementById()* and *getElementsByTagName()*. [there's an 's' in the second method]

1) getElementById()

Getting a tag by ID means locating a single node with a unique ID applied to it. In Figure 1, the <h1> tag has an ID attribute with the value of header. The following JavaScript selects that node:

```
document.getElementById('header')
```

This line means, "Search this page for a tag with an ID of 'header' assigned to it." The document part of *document.getElementById('header')* is a keyword that refers to the entire document. It is required, so you can't type *getElementById* by itself. The command *getElementById* is the method name and the *'header'* part is a string that is sent to the method.

Frequently, you will assign the results of this method to a variable to store a reference to the particular tag, so you can later manipulate it in your program. For example, if you want to use JavaScript to change the text of the headline, you can use this code:

```
var headline = document.getElementById('header');

headline.innerHTML = "JavaScript was here!"
```

The *getElementById()* command returns a reference to a single node, which is stored in a variable named headline. The second line of code uses the variable to access the tag's *innerHTML* property: *headline.innerHTML*, so you can reset the headline.

2) getElementsByTagName()

If you would like to find every link on a web page and do something to those links, you need to get a list of elements, not just one element marked with an ID. The command *getElementsByTagName()* will do.

This method works similarly to *getElementById()*, but instead of providing the name of an ID, you supply the name of the tag you're looking for. For example, to find all of the links on a page, you write this:

```
var pageLinks = document.getElementsByTagName('a');
```

This line in plain English means, "Search this document for every <a> tag and store the results in a variable named *pageLinks*." The *getElementsByTagName()* method returns a list of nodes, instead of just a single node. Therefore, the list acts more like an array. For example, the first item in the *pageLinks* variable from the code above is pageLinks[0] – the first <a> tag on the page – and *pageLinks.length* is the total number of <a> tags on the page.

You can also use *getElementById()* and *getElementsByTagName()* together. For example:

```
var banner = document.getElementById('banner');

var bannerLinks = banner.getElementsByTagName('a');

var totalBannerLinks = bannerLinks.length;
```

In the above code, the variable *banner* contains a reference to a tag, so the code *banner.getElementsByTagName('a')* only searches for <a> tags inside that instead of the whole HTML document.

JavaScript DOM in D3

The *getElementById()* and *getElementsByTagName()* methods in JavaScript DOM are not applied in any D3 code. Just like jQuery, DOM is another piece of the mental model to help programmers understand how D3 works in selection and data binding. We will talk about how selections are handled in D3 later.