Document Object Model (DOM) Notes

getElementById and innerHTML

Here is an example where `getElementById()` and `innerHTML` are used to display debugging information in an HTML document. Suppose the document has the following `div` element:

```
<div id="debugDiv">
</div>
```

Then the following function call would return a reference to that `div`:

```
document.getElementById("debugDiv");
```

Now suppose that you want to add some text to the `debugDiv` element. You can do so by using the innerHTML property as follows:

```
var debugDiv = document.getElementById("debugDiv");
debugDiv.innerHTML = "A line of debugging information<br>";
```

You can append text by using the `+=` assignment operator:

```
debugDiv.innerHTML += "Another line of debugging information<br>";
```

Suppose that you wanted to display the value of a variable, `name`. You could use a line of code like this:

```
debugDiv.innerHTML += "The value of name is: " + name + "<br>";
```

Using `getElementById` and `innerHTML` to add text to the debugging `div` takes two lines of code. Using basic DOM functions would require code to traverse the DOM tree, find the right element, create a new node, and append the new node to its parents list of child notes. That code could easily be five or ten times longer, and would be more difficult to understand, debug, and maintain.

When you set the value of the `innerHTML` property you can use arbitrary HTML. The browser will re-parse the element and format it accordingly, which is powerful and dangerous. It's powerful because you could build a table, or include a video player or do anything else you can do with HTML. It can be dangerous, however, if you put unfiltered user input into a web page. Doing so could the web page open to cross-site scripting attacks. You can read about cross-site scripting attacks in this Wikipedia article:
    http://en.wikipedia.org/wiki/Cross-site_scripting
and on this web site:
    http://www.thegeekstuff.com/2012/02/xss-attack-examples/

DOM Nodes

The Document Object Model (DOM) is a tree structure. The diagram (Figure 9.13) on page 335 of *Modern JavaScript* shows how the relationship between elements in an HTML page can be represented as a tree. Each node in the DOM tree is an object that has a property with a value that tells what kind of node it is. The most important kinds of nodes for our purposes are the document node, element nodes, attribute nodes, and text nodes.

The type of each node determines what kinds of nodes it can have as children. Here is a summary of the parent-child relationships that are most important for this class:

• A document node can have (at most) one element as a child.
• Elements can have other element nodes and text nodes as children.
• Text nodes don't have child nodes.

You can read more about node types on this w3schools page:
http://www.w3schools.com/dom/dom_nodetype.asp

In addition to the family-relationship terms mentioned in the book (parent, child, sibling), the terms ancestor and descendant are also used to describe relationships of nodes in a DOM tree.