Session 8 - Using the HTML DOM

Further AJAX topics

The main topic this week is the ***DOM*** but before we look at that, we just need to cover a couple of additional AJAX topics related to our Web Services and HTTP topic from last week, namely:

- How to retrieve an HTTP status code from an AJAX client;
- How to send POST requests and HTTP authentication from an AJAX client.

How to test the HTTP status code from an AJAX client

It is easy to test the HTTP status code from a web service client. If you are using AJAX you can simply use the ***status*** property of ***e.target*** (the original XMLHttpRequest object) in the callback, for example:

```
function ajaxCallback(e)
{
    if(e.target.status==404)
    {
        alert("The song was not found!");
    }
}
```

POST requests and sending HTTP authentication from AJAX

To send POST requests from AJAX, you need to create a ***FormData*** object and then ***append*** each item of POST data to it. The example below will create three items of POST data, ***flightnumber***, ***origin*** and ***destination***, and send them to ***newflight.php***. Note how the FormData object is supplied as a parameter to the ***xhr2.send()*** function. In the PHP script, we'd then read the data using ***$_POST["flightnumber"]***, ***$_POST["origin"]*** and ***$_POST["destination"]***.

```
var xhr2 = new XMLHttpRequest();
var data = new FormData();
data.append("flightnumber", "SA177");
data.append("origin", "London");
data.append("destination", "Denver");
xhr2.addEventListener("load", responseReceived);
xhr2.open("POST", "newflight.php");
xhr2.send(data);
```

When developing an ***AJAX*** client which uses HTTP authentication, you set this line in the HTTP header directly. The ***setRequestHeader()*** method of the XMLHttpRequest object allows you to add an instruction directly to the HTTP header. For example, to add the Authorization line (originally from http://coderseye.com/2007/how-to-do-basic-auth-in-ajax.html, site no longer up) with AJAX we could do the following:

```
var xhr2 = new XMLHttpRequest();
var data = new FormData();
data.append("flightnumber", "SA177");
data.append("origin", "London");
data.append("destination", "Denver");
xhr2.addEventListener("load", responseReceived);
xhr2.setRequestHeader("Authorization","Basic " + btoa(
xhr2.open("POST", "newflight.php");
xhr2.send(data);
```

Note that the ***btoa()*** (binary to ASCII) function is used to base64-encode the username and password. cURL did this automatically but with JavaScript you have to do it yourself.

Querying and Manipulating HTML documents using the DOM

So far, we've seen the ***innerHTML*** property which can be used to read, or change, the text within an HTML element. We've also looked at simple use of the ***Document Object Model (DOM)*** to access elements on a web page - specifically the use of ***getElementById()*** to access a specific page element using its ID.

However that's just the start: the DOM offers a whole range of ways to read and manipulate HTML pages or XML data. To understand how you can use DOM for document manipulation, you must understand the concept of ***nodes***, which we will discuss below. The examples below use the DOM to manipulate HTML documents, however, more generally, the DOM is used for accessing and manipulating XML documents. An HTML web page is a particular, specific type of XML document. So, as well as using the DOM to query and manipulate web pages, we can use it in a more general sense to query and manipulate XML. In AJAX, this latter use of the DOM is used extensively.

The Concept of Nodes

- Part of the W3C Document Object Model (DOM)
- A systematic way to navigate and manipulate the content of an HTML or XML document, not only the ***elements***, but also the ***text within them***
- An HTML or XML document consists of a series of hierarchical ***nodes***

- Each ***element*** (e.g ***p***, ***div***, or ***em*** in HTML; or a custom tag in XML) is treated as a ***node***
- However it's not just the elements themselves: the ***text*** within each element is also treated as a special kind of node, a ***text node***
- The nodes are a ***nested***, ***hierarchical*** structure
    - An element within an element is a ***child node*** of that element

Example of Nodes Terminology

```
<body>

<p> Welcome to the <em>wonderful!</em> world of dynamic
</body>
```

The paragraph is a ***child node*** of the ***body***
The paragraph contains **three** of its own ***child nodes***:

- The **text**: ***"Welcome to the"***
- The **em** element
- The **text**: ***"world of dynamic text!"***

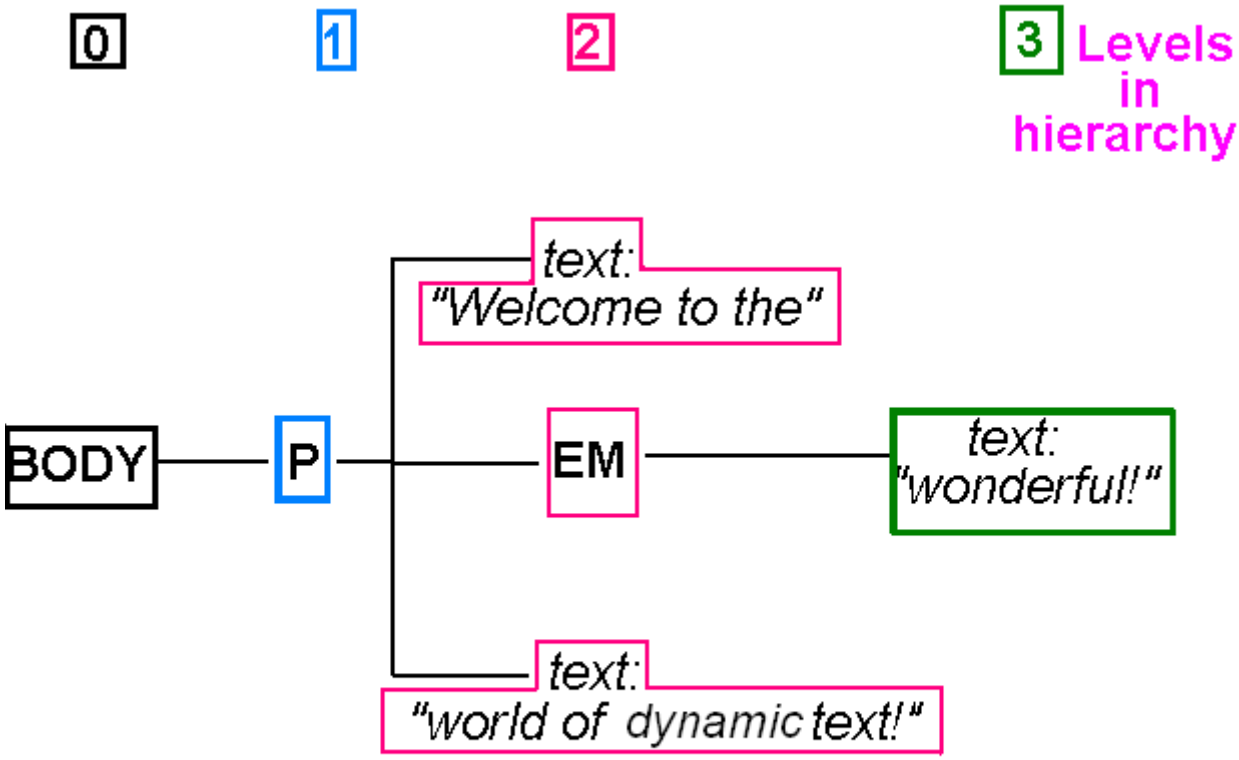
The **em** ***itself*** contains one child node:

- The **text**: ***"wonderful!"***
- This is a child of the **em** not the **p**

Nodes Hierarchy Diagram: Tree view

```
<body>
<p> Welcome to the <em>wonderful!</em> world of dynamic

</body>
```



Querying the Node Structure of a Document

- document.body.***childNodes*** is an ***array*** of all child nodes of the body
  - i.e. all elements and text within the page
- node.***childNodes*** is an array of all child nodes of a given node (i.e. element)
- node.***nodeName*** gives the ***name*** of a node
  - i.e. em, p, song, artist, etc; or ***#text*** for text nodes
- node.***nodeValue*** gives the ***text within a text node*** **(only)**
  - It has no meaning for page elements

Example

Adding a new node

- One of the keys to dynamic text
- ***document.createElement()*** allows us to ***create a brand new element!***
- ***document.createTextNode()*** allows us to create a brand new text node
- Having created an element we can then populate it with text nodes and child elements
- ... and finally add it to either:
  - The body

- Another element, which will act as its parent

Example

Replacing an existing node

- The other key to dynamic text
- We can ***replace a whole paragraph of text with another!!!***
- Steps:
  - Find the element to replace
  - Create a new element (as per the previous example)
  - Call ***replaceChild()*** to replace the old node with the new

Example

Getting all elements of a particular type

- It can make it easier to find a page element if we can collect together all elements of a particular type (e.g. all paragraphs)
- ***document.getElementsByTagName()*** allows us to do this
- This gives us an ***array*** of all elements of a particular type, which we can then index

Example

Removing nodes

- The final key concept of dynamic text is ***removing*** a particular node
- Use

```
parentElement.removeChild(childElement)
```

- Use IDs or ***getElementByTagName()*** once again to find the element to remove

Other useful features of the DOM

(Source: Quirksmode, a very useful reference site for JavaScript and the DOM)

- ***childNodes*** property - an array of all the child nodes of a given node.e.g.:

```
for (var count=0; count<parent.childNodes.length; c
{
    alert(parent.childNodes[count].nodeName); // sh
}
```

- ***firstChild*** and ***lastChild*** properties - the first and last child node of a given node. e.g.

```
// create a child node of the parent, assume that '
var p = document.createElement("p");
parent.appendChild(p);

// create a text node
var textNode = document.createTextNode("some text")

// append it to the first child of the node 'parent
// paragraph we created above. In other words 'p' a
// are the same in this case.
parent.firstChild.appendChild(textNode);
```

- ***parentNode*** property - the parent node of a given child node. So for the code:

```
var p = document.createElement("p");
parent.appendChild(p);
```

  the ***parentNode*** of ***p*** would be ***parent***.
- ***previousSibling*** and ***nextSibling*** - find the sibling nodes (siblings = Geschwister in German). Imagine that a given parent node has three child nodes, 0, 1 and 2. The ***previousSibling*** of child node 1 would be child node 0. The ***nextSibling*** of child node 1 would be child node 2.
- ***insertBefore*** method - inserts a child node into a parent node at an arbitrary position, e.g:

```
parentNode.insertBefore(newNode,parentNode.childNod
```

  This code would insert ***newNode*** before child node 3 of the parent.
- ***document.querySelector()*** and ***document.querySelectorAll()*** give the first element, and all elements, which match a CSS selector respectively, eg

```
document.querySelectorAll(".important")
```

  will give an array of all elements with a class of ***important***.

More advanced exercise - Using the DOM to add Download Functionality to your HitTastic! AJAX site

Do not attempt this unless you have done all exercises from the first AJAX topic last time, and are comfortable with the material on DOM nodes from this week. This exercise will also give you a preview of some later topics.

Inspecting elements

The ***inspect*** functionality in the browser will help you see what's going on - try it as you do this exercise - right click on an element. Inspecting helps you to see the actual current DOM structure of the page - including any dynamically-created elements.

Questions

You are going to extend the functionality of your AJAX-based HitTastic! website so that the user is able to download music. To do this, you need to ***make sure that you have done the "download" web service (session 4)***.

1. Make a copy, in a new file, of your existing AJAX page. Delete all the existing code in the ***for*** loop.
2. Add code to your AJAX callback function so that it ***creates a DOM paragraph (p) element*** for each hit returned.
3. Within this ***p***, create:
   - a text node containing the details of the song, and
   - a button node (i.e. an "input" with a type of "button") allowing the user to download the song. Add both to the paragraph node using ***appendChild()***.

   To make sure that the button is an input with a type of "button", you need to set the "type" property to "button", i.e. if ***btn*** is your button, then you can use:

```
btn.type = "button";
```

4. Specify the text on the button by setting the "value" property:

```
btn.value="Download!";
```

5. Give the button an ID equal to 'btn' plus the song id, i.e.

```
btn.id = "btn" + (the song ID from the JSON);
```

6. Set up a "click" event handler for the button. as an arrow function, e.g:

```
btn.addEventListener ("click", (e) =>
                {

                                    // fill in your arr
```

```
                }
);
```

This will set the "click" handler of the button to be an arrow function.

7. Add each paragraph to the results div using ***appendChild()***.
8. You can now complete the arrow function which runs when the user clicks the button. Use AJAX to call your download web service from session 4. You will need to send POST data to the service with AJAX (see above).
    - **You need to work out which button was pressed in order to download the correct song.**. e.target gives us the button which was clicked. So

      ```
      var songid = e.target.id.substring(3);
      ```

      will give the song ID; note that the button ID was 'btn' + the song ID so substring(3) gives us the characters from the fourth onwards, i.e the song id.

The callback to the AJAX download request should retrieve the output from the web service, interpret it and act accordingly. If the song was bought successfully, display a message (ideally in a dialog box , if not as an alert box) reading ***Bought successfully***, otherwise display ***Not enough money in your account!***

- How would you create a dialog box ? You could create a element, give it an absolute CSS position in JavaScript, e.g.

  ```
  element.style.position='absolute';
  element.style.left='100px';
  element.style.top='100px';
  element.style.backgroundColor = ...;
  ```

  and add it as a child of its parent (which might be the page body or the main of the page). You could add an "OK" button to the ***dialog*** div with a "click" event handler arrow function, as above, which makes the dialog box disappear (set its 'display' CSS property to 'none').