Document Object Model

Document Object Model

- The Document Object Model (DOM) is the tree-like structure used to represent an HTML document.
- It is through the DOM that we can interact not only with the browser but also with the HTML and CSS on a page.
- Recall the HTML tree from previous lectures.

DOM Levels

- The DOM exists through W3C specifications, sometimes called "Levels".
- However, the DOM existed before there were specifications for it!
- DOM Level 0, aka "Legacy DOM", is the name for the DOM that existed before specification.
- There are three levels, Levels 1 through 3.
- Browsers support the DOM with varying success. – IE9 and sooner do things differently.

DOM Properties

- There are several properties of the document object that are useful.
- title, doctype, and others
- You can also access elements through the forms collection.
- document.forms
- Numerous other collections available.
- See https://developer.mozilla.org/en-US/docs/Web/API/Document

Document Methods

- The methods found in the document object are where the magic happens.
- It is through these methods that we can access elements in the document.
- Through the elements we can change the document and change styles.

Some Basic Document Methods

- Retrieve an element by its "id" from the HTML:
- document.getElementById("idName");
- Retrieve elements by their class name:
- document.getElementsByClassName("name");
- Retrieve elements by HTML tag name:
- document.getElementsByTagName("name");
- See MDN URL from previous slide for others.

So You Have an Element...

- You can get access to the elements within the document using the previous methods.
- Now what?
- You use the **Element Interface** to query and change that element or that collection of elements.
- https://developer.mozilla.org/en-US/docs/Web/API/Element
- Following slides demo some properties and methods.

Changing the Content with innerHTML

- The innerHTML property is used to get or set the HTML within an element.
- Demo

Retrieve Element Attributes

- You can get a list of an elements current attributes with the attributes property:
- element.attributes;
- Demo
- Returns a list of all attributes.

Get and Set Individual Attributes

- You can retrieve an individual attribute within an element or set an individual attribute.
- getAttribute("attribute")
- setAttribute("attribute","newvalue")

View Current Classes

- The classList property returns a list of classes applied to the element – Returns a DOMTokenList.
- The className property returns a string with a list of classes.

Changing Styles - Inline

- Recall the window.getComputedStyle() method from earlier.
- You can change styles, one at a time, using the style property.
- The style property is essentially the same as using an inline style on the element(s). element.style.property.
- Be aware that when using this, the property names are not the same as in traditional CSS.
  - Example: backgroundColor instead of background-color.

Changing Styles By Name

- Avoid changing styles inline (the previous slide).
- It is better to apply an existing CSS class to the element(s).
- Doing so keeps styling where it belongs, in the CSS, and not in the JavaScript.
- The className property can be used for this purpose.

Creating HTML

- You can also create HTML elements within the document.
- document.createElement("elementName");
- Put the element into the document:
  - appendChild
  - insertBefore
  - ... others ...

appendChild()

- Use appendChild() to add an element within (at the end) of the chosen element:
- //add something to the bottom of the body:
- document.body.appendChild(yourElement);

insertBefore()

- Get the element that you want to insert before:

```
var currentSection =
document.getElementById("lastSection");
```

- Execute insertBefore():

```
document.body.insertBefore(newSection,
currentSection);
```

Adding Text to the Element

- You've added an element but not changed its text.
- Use createTextNode() for to add plain text within an element.

```
var content = document.createTextNode("This is some content.");

newSection.appendChild(content);

document.body.appendChild(newSection);
```

Removing Elements

- removeChild() is used to remove elements.
- Be sure to run removeChild on the parentNode:

```
var currentSection = document.getElementById("lastSection");

currentSection.parentNode.removeChild(currentSection);
```

HTML Collections

- There are various collections (think: array) that might be of use.
- We worked with document.forms, which is a collection of the forms in a given document.
- Also:
  - document.anchors – A collection of the named <a> elements in a document.
  - document.images – A collection of the <img> elements.
  - document.links – A collection of all <a> and <area> elements that contain an href attribute.

Next Steps

- The Document Object Model is a complex but highly useful way to interact with a web document.
- But what we've done is only part of the story.
- Events, which are tied to the window object, are a means to respond when something happens within the web browser environment.
- We will discuss events in an upcoming lecture.