JavaScript:
DOM and Dynamic HTML

Overview

- The Document Object Model (DOM)
- Element Access in JavaScript
- Events and Event Handling
  - Handling events from Body Elements
  - Handling events from Button Elements
  - Handling events from Text Box and Password Elements
- Dynamic HTML
  - Element positioning and moving
  - Changing Colours and Fonts
  - Dynamic Content
  - Reacting to a Mouse Click

What is DOM ?

Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."

DOM

The DOM is separated into 3 different parts / levels:

Core DOM - standard model for any structured document

XML DOM - standard model for XML documents

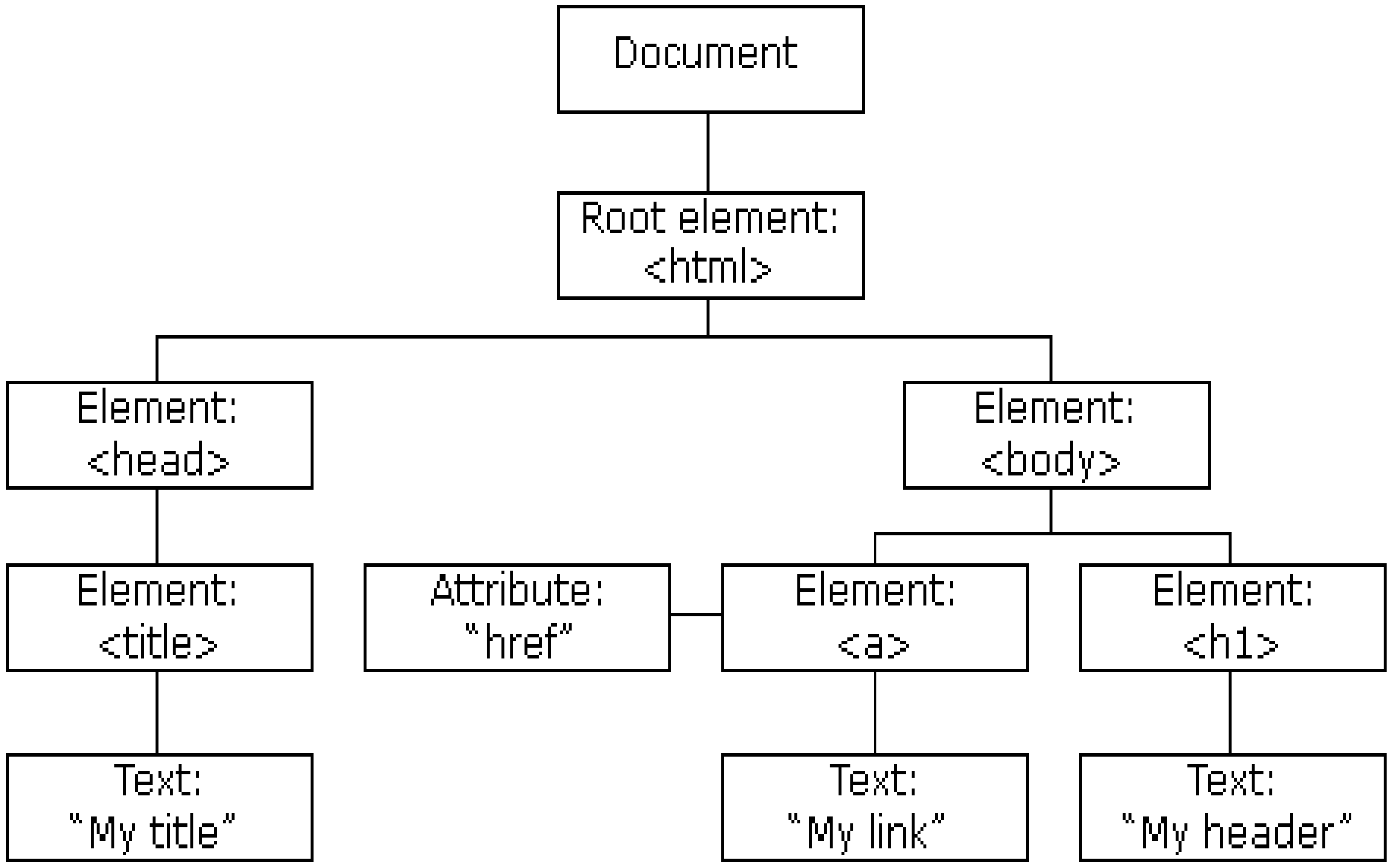
XHTML DOM - standard model for XHTML documents

JavaScript Execution Environment

- Every `Document` object has:
  - `forms` - an array of references to the forms of the document
    - Each `forms` object has an `elements` array, which has references to the form's elements
  - `Document` also has property arrays for anchors, links, & images

The Document Object Model

- A language that supports the DOM must have a binding to the DOM constructs
- In the JavaScript binding, XHTML elements are represented as objects and element attributes are represented as properties
- e.g., `<input type = "text" name = "address">`
  - would be represented as an object with two properties, `type` and `name`, with the values `"text"` and `"address"`

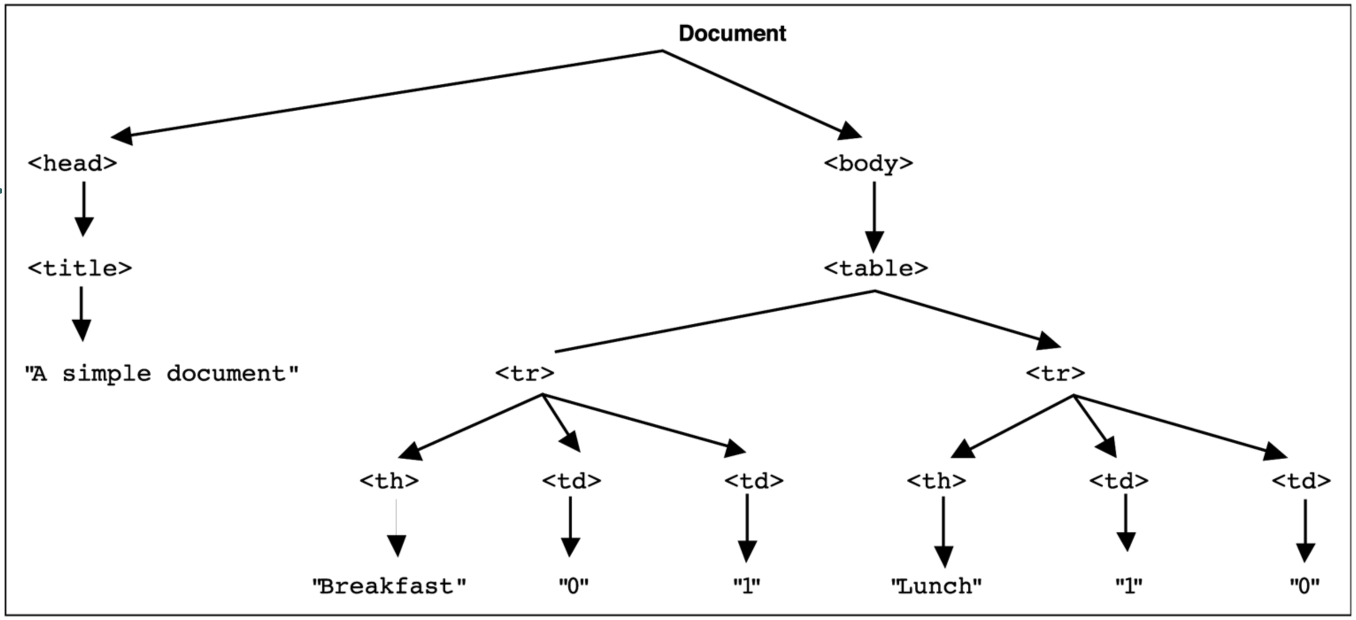

Document
Root element: <html>
Element: <head>
Element: <body>
Element: <title>
Attribute: "href"
Element: <a>
Element: <h1>
Text: "My title"
Text: "My link"
Text: "My header"

- JavaScript can change all the HTML elements in the page
- JavaScript can change all the HTML attributes in the page
- JavaScript can change all the CSS styles in the page
- JavaScript can remove existing HTML elements and attributes
- JavaScript can add new HTML elements and attributes
- JavaScript can react to all existing HTML events in the page
- JavaScript can create new HTML events in the page

DOM Structure

- Documents in the DOM have a tree like structure

```
<html>
  <head> <title> A simple document </title>
  </head>
  <body>
    <table>
        <tr>
           <th>Breakfast</th>
           <td>0</td>
           <td>1</td>
        </tr>
        <tr>
           <th>Lunch</th>
            <td>1</td>
            <td>0</td>
        </tr>
    </table>
  </body>
</html>
```



The Document Object Model

- DOM 0 is supported by all JavaScript-enabled browsers (no written specification)
- DOM 1 was released in 1998
- DOM 2 issued in 2000
  - Nearly completely supported by NS7
  - IE6’s support is lacking some important things
- DOM 3 is the latest W3C specification
- The DOM is an abstract model that defines the interface between HTML documents and application programs—an API

DOM1 AND DOM2

DOM Level 1 provided a complete model for an entire HTML or XML document, including means to change any portion of the document. Non-conformant browsers such as Internet Explorer 4.x and Netscape 4.x were still widely used as late as 2000.

DOM Level 2 was published in late 2000. It introduced the "getElementById" function as well as an event model and support for XML namespaces and CSS.

DOM3 AND DOM4

DOM Level 3, the current release of the DOM specification, published in April 2004, added support for XPath and keyboard event handling, as well as an interface for serializing documents as XML.

DOM Level 4 is currently being developed. Draft version 6 was released in December 2012

DOM IN HTML

The HTML elements as objects

The properties of all HTML elements

The methods to access all HTML elements

The events for all HTML elements

Element Access in JavaScript

- There are several ways to do it

1. DOM address

- Example (a document with just one form and one widget):

```
<form action = "">
   <input type = "button" name = "CLICK">
</form>
```

- `document.forms[0].elements[0]`
- Problem: document changes

Element Access in JavaScript

2. **Element names**
   - requires the element and all of its ancestors (except body) to have `name` attributes
   - Example:

```
<form name = "myForm"  action = "">
   <input type = "button" name = "click">
</form>
```

   - `document.myForm.click`

Element Access in JavaScript

3. getElementById Method (defined in DOM 1)

- Example:

```
<form action = "">
  <input type = "button"  id = "click">
</form>
```

- `document.getElementById("click")`

- Form elements often have `ids` and names both set to the same value

Element Access in JavaScript

- Checkboxes and radio button have an implicit array, which has their name

```
<form id = "toppingGroup">
   <input type = "checkbox" name = "toppings"
         value = "olives" />
   ...
   <input type = "checkbox"  name = "toppings"
         value = "tomatoes" />
 </form>
 ...
 var numChecked = 0;
 var dom = document.getElementById("toppingGroup");
 for (index = 0; index < dom.toppings.length; index++)
   if (dom.toppings[index].checked]
     numChecked++;
```

Events and Event Handling

- An *event* is a notification that something specific has occurred, either with the browser or an action of the browser user
- An *event handler* is a script that is implicitly executed in response to the appearance of an event
- The process of connecting an event handler to an event is called registration
- Don’t use `document.write` in an event handler, because the output may go on top of the display

Events and their Tag Attributes

Event	Tag Attribute
blur	onblur
change	onchange
click	onclick
focus	onfocus
load	onload
mousedown	onmousedown
mousemove	onmousemove
mouseout	onmouseout
mouseover	onmouseover
mouseup	onmouseup
select	onselect
submit	onsubmit
unload	onunload

Events, Attributes and Tags

- The same attribute can appear in several different tags
  - e.g., The `onclick` attribute can be in `<a>` and `<input>`
- A text element gets focus in three ways:

1. When the user puts the mouse cursor over it and presses the left button
2. When the user tabs to the element
3. By executing the `focus` method

Attribute	Tag	Description
`onblur`	`<a>`	The link loses the input focus.
	`<button>`	The button loses the input focus.
	`<input>`	The input element loses the input focus.
	`<textarea>`	The text area loses the input focus.
	`<select>`	The selection element loses the input focus.
`onchange`	`<input>`	The input element is changed and loses the input focus.
	`<textarea>`	The text area is changed and loses the input focus.
	`<select>`	The selection element is changed and loses the input focus.
`onclick`	`<a>`	The user clicks on the link.
	`<input>`	The input element is clicked.
`onfocus`	`<a>`	The link acquires the input focus.
	`<input>`	The input element receives the input focus.
	`<textarea>`	A text area receives the input focus.
	`<select>`	A selection element receives the input focus.
`onload`	`<body>`	The document is finished loading.
onmousedown	Most elements	The user clicks the left mouse button.
onmousemove	Most elements	The user moves the mouse cursor within the element.
`onmouseout`	Most elements	The mouse cursor is moved away from being over the element.
`onmouseover`	Most elements	The mouse cursor is moved over the element.
onmouseup	Most elements	The left mouse button is unclicked.
`onselect`	`<input>`	The mouse cursor is moved over the element.
	`<textarea>`	The text area is selected within the text area.
`onsubmit`	`<form>`	The Submit button is pressed.
`onunload`	`<body>`	The user exits the document.

Registration of Event Handler

- By assigning the event handler script to an event tag attribute

```
<input type “button” name = “myButton”
onclick = "alert('Mouse click!');“ />

<input type “button” name = “myButton”
onclick = "myHandler();" />
```

Handling Events from Body Elements

- Events most often created by body elements are `load` and `unload`
- Example:
  - the `load` event - triggered when the loading of a document is completed

Handling Events from Button Elements

- Plain Buttons – use the `onclick` property
- Radio Buttons
  - Example 1:
    - The handler is registered in the markup, so the particular button that was clicked can be sent to the handler as a parameter
  - Exampe 2:
    - The handler is registered by assigning it to a property of the JavaScript objects associated with the HTML elements
    - This registration must follow both the handler function and the XHTML form

Handling Events from Textbox and Password Elements

- Checking Form Input
  - A good use of JavaScript, because it finds errors in form input before it is sent to the server for processing
- Things that must be done:
  1. Detect the error and produce an alert message
  2. Put the element in focus (the focus function) - puts the cursor in the element
  3. Select the element (the select function) - highlights the text in the element
- To keep the form active after the event handler is finished, the handler must return `false`

Handling Events from Textbox and Password Elements

- Example 1 – comparing passwords
  - The form just has two password input boxes and Reset and Submit buttons
  - The event handler is triggered by the Submit button

- Example 2 – checking the format of a name and phone number
  - The event handler will be triggered by the change event of the text boxes for the name and phone number

Dynamic HTML

- An HTML document whose tag attributes, tag contents, or element style properties can be changed after the document has been and is still being displayed by a browser
- Such changes are made with an embedded script (JavaScript) that accesses the elements of the document as objects in the associated DOM structure

Element Positioning

- The position of any element is dictated by the three style properties: `position`, `left`, and `top`
  - The three possible values of position are `absolute`, `relative`, and `static`

```
<p style = "position: absolute; left: 50px;
               top: 100px;">
```

- If `position` is set to either `absolute` or `relative`, the element can be moved after it is displayed
  - Just change the `top` and `left` property values with a script

Changing Colours and Fonts

- Colour example:

  - The actual parameter `this.value` works because at the time of the call, `this` is a reference to the text box (the element in which the call is made)
    - So, `this.value` is the name of the new colour
- Changing fonts example

  - We can change the font properties of a link by using the `mouseover` and `mouseout` events to trigger a script that makes the changes

Event propagation

- The node of the document tree where the event is created is called the *target node*

- The first phase is called the *capturing phase*

- Events begin at the root and move toward the target node

  - If there are registered event handlers at nodes along the way (before the target node is reached), if one is enabled, it is run

- The second phase is at the target node

  - If there are registered handlers there for the event, they are run

- The third phase is the *bubbling phase*
  - Event goes back to the root; all encountered registered handlers are run

5.5 The DOM 2 Event Model (continued)

- Not all events bubble

- Any handler can stop further propagation by calling the `stopPropagation` method of the `Event` object

- DOM2 model uses the `Event` object method, `prevent Default` to stop default operations, such as submission of a form, even though an error has been detected

- Event handler registration is done with the addEventListener method

  - Three parameters:

    1. Name of the event, as a string literal

    2. The handler function

    3. A Boolean value that specifies whether the event is enabled during the capturing phase

5.5 The DOM 2 Event Model (continued)

- A temporary handler can be created by registering it and then unregistering it with `removeEventListener`

- The `currentTarget` property of `Event` always references the object on which the handler is being executed

- The MouseEvent object (a subobject of Event) has two properties, clientX and clientY, that have the x and y coordinates of the mouse cursor, relative to the upper left corner of the browser window

- An example: A revision of validator, using the DOM 2 event model

→ SHOW `validator2.html`

- Note: DOM 0 and DOM 2 event handling can be mixed in a document

5.6 The `navigator` object

- Indicates which browser is being used

- Two useful properties

   1. The `appName` property has the browser’s name

   2. The `appVersion` property has the version #

- Microsoft has chosen to set the `appVersion` of IE6 to 4 (?)

- Netscape has chosen to set the `appVersion` of NS6 to 5.0 (?)

→ SHOW `navigator.html` & Figures 5.9 & 5.10

Summary

- The Document Object Model (DOM)
- Element Access in JavaScript
- Events and Event Handling
  - Handling events from Body Elements
  - Handling events from Button Elements
  - Handling events from Text Box and Password Elements