CS144: Document Object Model (DOM)

- JavaScript code interacts with a Web page through HTML DOM

Adding JavaScript to a Web page

```html
<script>
... javascript code ...
</script>
<script src="script.js"></script>
```

- `<script>` may appear anywhere on a page

HTML DOM Tree

- Every HTML element on a page becomes an `element node`
- Text inside an HTML element becomes a `text node`, which is a child of the element node
  - e.g., `<h1>Heading</h1>`
  - Most modern browsers preserve whitespaces starting from `<head>` through `<body>`
    - But many older browsers handle them slightly differently
- An attribute of an HTML element becomes an `attribute node`
  - An attribute node is `associated with` the element node, but is not a child node
- Example http://oak.cs.ucla.edu/classes/cs144/examples/dom.html

```html
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>
  <h1>Heading</h1>
  <a href="good/">Link</a>
</body>
</html>
```
CS144: Web Applications

Accessing DOM Tree in JavaScript

- In JavaScript, every node in the DOM tree becomes a JavaScript object with properties, methods, and associated events
  - An HTML document is parsed into a “DOM tree” and is made available as document object inside the browser
  - In the above example, document object has !DOCTYPE and html nodes as its children
- Traversing the DOM tree
  - A node’s children and parent are accessible through childNodes and parentNode properties, respectively
  - An element node’s attributes are accessible through attributes property
  - Alternatively, the following methods allow direct access to any node in the tree
    ```
    document.getElementById('id');
document.getElementsByTagName('h1');
document.getElementsByClassName('class');
    ```
- Each node’s type, name, and value are available through following properties:
  - `nodeType`: element (1), attribute (2), text (3), comment (8), …
  - `nodeName`: tag name for element node (e.g., HEAD), attribute names for and attribute node, #text for text node, …
  - `nodeValue`: inside text for text and comment nodes. attribute value for attribute nodes. null otherwise
- Traverse the DOM tree using Chrome Developer Console for our example page

Manipulating DOM Nodes

- JavaScript objects corresponding to DOM nodes have
  - Properties
  - Methods
  - Associated events
- By changing the property values, calling the methods, we can change the HTML element dynamically
  - Updating properties
document.body.style.background = "yellow";
document.body.innerHTML = "<p>new text</p>";
document.getElementById('warning1').style.color = "red";

Note:
* By setting `innerHTML` property, we can update the DOM tree below the object
  * Alternatively, `document.createElement()`, `document.createTextNode()` and `appendChild()`, `removeChild()`, `replaceChild()` can be used to modify the DOM tree

```javascript
var newP = document.createElement("p");
var newText = document.createTextNode("new text");
newP.appendChild(newText);
document.body.replaceChild(newP);
```

– Calling methods

```javascript
document.getElementById('myform1').reset();
document.getElementById('myform1').submit();
```

Basic Event Handling in JavaScript

* **Event-driven programming**: to dynamically update a Web page based on user action, JavaScript program must
  1. “wait for” relevant “events”
  2. take an appropriate actions given an event

* Dealing with events inside a Web page
  * Each DOM object is associated with a set of “events”
    * e.g., “load”, “click”, “input”, “keyup”, “mouseover”, …
  * An object has an *event handler* for each associated event
    * `onload`, `onunload`, `onclick`, `oninput`, `onmouseover`, …
    * When an event is fired on an object (= event target), the associated *event handler* (= event listener = callback function) is called
By setting an event handler to our own function, we can specify what actions to take when an event is triggered

```javascript
function ChangeColor(event) {
    document.body.style.color = "red";
}
document.body.onclick = ChangeColor;
```

or inside the HTML body element itself

```html
<body onclick="ChangeColor(event);">
```

- Read our previous example code and make sure that students understand it

```html
<html>
<meta charset="utf-8">
<head><title>JavaScript Example</title></head>
<body>Click on this document!</body>
<script>
    let colors = [ "yellow", "blue", "red" ];
    let i=0;
    function ChangeColor(event) {
        document.body.style.backgroundColor = colors[i++%3];
    }
    document.body.onclick = ChangeColor;
</script>
</html>
```

**Advanced JavaScript Event Handling**

- *Event object*
  - Event object contains details of the event and is passed as the (only) argument to the event handler function
  - Its `type` property specifies its event type
  - Its `target` property specifies the event target to which the event was originally triggered
• Event handler function
  – Even handlers are invoked with an event object as their single argument
  – Inside an event handler, `this` points to the “current” element (not the original event target)
  – If event handler returns `false`, browser does NOT perform the default action associated with the event
  – If event handler is specified as the value of `onXXX` attribute inside HTML page not in a script block, the specified code is wrapped into a function that is passed with the single parameter `event`

• *Event bubbling*
  – After the event handlers on the target element are invoked, most events “bubble up” the DOM tree
    * Target’s parent and grand parent get the event all the way through the `document` (and sometimes even `window`) object
    * Exceptions: focus, scroll, …
  – To stop event propagation, call `event.stopPropagation()`

• JavaScript Execution Timeline in Browser
  – A JavaScript code in a browser is executed as a *single thread*
    * No two event handlers will *never* run at the same time
    * Document content are never updated by two threads simultaneously
      ▶ No worries about locks, deadlock or race conditions
    * But web browser “stops” responding to user input while script is running
  1. `document` object is created and `document.readyState` is set to “loading”
  2. The document is parsed synchronously downloading and executing scripts in the order they appear (if no `async`)
    – `async` script starts to be downloaded and gets executed as soon as they are available
  3. Once the document is completely parsed, `document.readyState` is set to “interactive”
  4. Browser fires “DOMContentLoaded” event (calls `onload` callback) on `document` object
  5. `document.readyState` property is set to “complete”
  6. Browser waits for events and calls appropriate event handlers
Q: What will happen if we move the `<script>...` before `<body>...</body>` in the above example?

Note:
- HTML DOM object manipulation can be done only after the object has been parsed and loaded, not before
- To run some initialization code, set the `onload` handler with the initialization code
- To run final cleanup code, set the `onunload` handler
- Script with `async` attribute cannot use `document.write()` method

`window` as the Global Object

- `window` object is the “global object” within a browser
  - All global variables and functions become properties and methods of `window`
    * e.g., `document` is in fact `window.document`
- `window.location`: the URL of the current page
  - By setting this property, we can load a different page
- `window.history`: browsing history
  - `window.history.back()`, `window.history.forward()`
- `window.alert()`, `confirm()`, `prompt()`: open a dialog box

```javascript
alert("hello, world!");
response = confirm("Click OK to proceed, Cancel to return");
// boolean
name = prompt("Type your name"); // string
```

References

- DOM Technical Reports: https://www.w3.org/DOM/DOMTR
- DOM Level 3 Events: https://www.w3.org/TR/DOM-Level-3-Events/
• Reference for common JavaScript and DOM objects: https://www.w3schools.com/jsref/