Document Object Model (DOM)

HTML DOM Tree

- An 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>` creates an `h1` element node and a text node "Heading" as its child
  - Different browsers handle "whitespace only" text nodes differently
- Aa 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

```
<html>
<head><title>Page Title</title></head>
<body><h1>Heading</h1><a href="good/">Link</a></body>
</html>
```

- Overall, an HTML document becomes a "DOM tree", whose root is a *document node*
  - The document node has the "HTML" element node as its child

Traversing DOM Tree in JavaScript

- In JavaScript, every node in the DOM tree becomes a JavaScript object with properties, methods, and associated "events"
- The root document node of the tree is accessible through the global variable `document`
- A node's children and parent are accessible through `childNodes` and `parentNode`, respectively
- An element node's attributes are accessible through `attributes` property

- Each object has the information on the type, name, and value of the node
  - Type (`nodeType`): element (1), attribute (2), text (3), comment (8), ...
  - Name (`nodeName`): tag and attribute names for element and attribute nodes, `#text` for text node, ...
  - Value (`nodeValue`): inside text for text and comment nodes. attribute value for attribute nodes. `null` otherwise
- Traverse the DOM tree using Chrome Developer Console for http://oak.cs.ucla.edu/classes/cs144/examples/dom.html
- Alternatively, the methods allow direct access to any node in the tree

```
document.getElementByID('id');
document.getElementByTagName('h1');
document.getElementByClassName('class');
```

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 `style` property, we can update CSS style of the object
* 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

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

- Calling methods

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

Basic Event Handling in JavaScript

- *Event-driven programming*
  - For updating a Web page dynamically based on user action, JavaScript program must
    1. "wait for" relevant "events"
    2. take an appropriate actions given an event
- Dealing with events on the DOM tree
  - Each DOM object are associated with a set of "events"
    - e.g., "load", "click", "mouseover", "keyup", ...
  - An object has an *event handler* for each associated event
    - `onload`, `onunload`, `onclick`, `onmouseover`, `onmouseout`, `onkeyup`, ...
    - 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 happens

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

or inside the body element itself

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

- Read example code and make sure that you understand it

```
<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 and `target` property specifies the event target
- Event handler function
  - Even handlers are invoked with an event object as their single argument
  - Inside an event handler, `this` points to the 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 `window`) object
    * Exceptions: focus, scroll, ...

- 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

- JavaScript Execution Timeline in Browser

  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>...</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` 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`
- `windows.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()`
- `alert()`, `confirm()`, `prompt()`: opens a dialog box

```
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 CSS property names in JavaScript: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Properties_Reference
- Reference for common JavaScript and DOM objects: https://www.w3schools.com/jsref/