Lecture 4: Introduction to DOM Manipulation

Building Modern Web Applications
Vancouver Summer Program 2018 (Package E)

Karthik Pattabiraman, Julien Gascon-Samson

The Univerity of British Columbia
Department of Electrical and Computer Engineering
Vancouver, Canada

ece Electrical and Computer Engineering

UBC

Wednesday, July 25, 2018

DOM: Recap

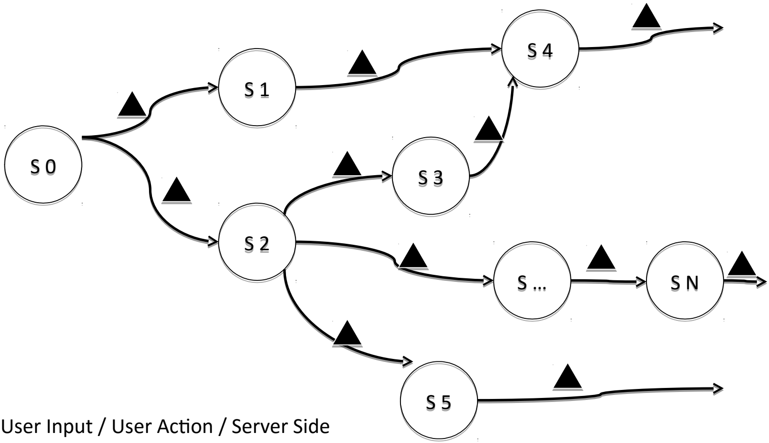
1. DOM: Recap
2. DOM Access APIs
3. DOM Traversal
4. Adding and removing nodes

DOM: Recap

- Hierarchical representation of the contents of a web page – initialized with static HTML
- Can be manipulated from within the JavaScript code (both reading and writing)
- Allows information sharing among multiple components of web application

DOM as an evolving entity

DOM is highly dynamic!



▲ User Input / User Action / Server Side

Why Study DOM Interactions?

- Needed for JS code to have any effect on webpage (without reloading the page)
- Uniform API/interface to access DOM from JS
- Does not depend on specific browser platform

NOTE

- We'll be using the native DOM APIs for many of the tasks in this lecture
- Though many of these can be simplified using frameworks such as jQuery, it is important to know what's "under the hood"
- We assume a standards compliant browser !

DOM Access APIs

1. DOM: Recap
2. DOM Access APIs
3. DOM Traversal
4. Adding and removing nodes

Motivation: Selecting Elements

- You can access the DOM from the object window.document and traverse it to any node
- However, this is slow – often you only need to manipulate specific nodes in the DOM
- Further, navigating to nodes this way can be error prone and fragile
  - Will no longer work if DOM structure changes
  - DOM structure changes from one browser to another

Methods to Select DOM Elements

- With a specified id
- With a specified tag name
- With a specified class
- With generalized CSS selector

Method 1: *getElementById*

- Used to retrieve a **single** element from DOM
  - IDs are unique in the DOM (or at least must be)
  - Returns null if no such element is found

Example

```
var name = "Section1";
var id = document.getElementById(name);
if (id == null)
    throw new Error("No element found: " + name);
```

Method 2: *getElementsByTagName*

- Retrieves multiple elements matching a given tag name ('type') in the DOM
- Returns a *read-only* array-like object (empty if no such elements exist in the document)

Example: Hide all images in the document

```
var imgs = document.getElementsByTagName("img");
for (var i=0; i<images.length; i++) {
    imgs[i].display = "none";
}
```

Method 3: *getElementsByClassName*

- Can also retrieve elements that belong to a specific CSS class
  - More than one element can belong to a CSS class<

Example

```
var warnings = document.getElementsByClassName("warning");
if (warnings.length > 0) {
    // do something with the warnings list here
}
```

Important point: Live Lists

- Both getElementsByClassName and getElementsByTagName return live lists
  - List can change after it is returned by the function if new elements are added to the document
  - List cannot be changed by JavaScript code adding to it or removing from it directly though
- Make a copy if you're iterating thro' the lists

Selecting elements by CSS selector

- Can also select elements using generalized CSS selectors using querySelectorAll() method
  - Specify a selector query as argument
  - Query results are not “live” (unlike earlier)
  - Can subsume all the other methods
- querySelector() returns the first element matching the CSS query string, null otherwise

CSS selector syntax: Examples (Recap)

```
"#nav"            // Any element with id=nav

"div"           // Any <div> element

".warning"        // Any element with "warning" class

"#log span"         // Any <span> descendant of id="log"

"#log > span"      // Any span child element of id="log"

"body>h1:first-child" // first <h1> child of <body>

"div, #log"       // All div elements, element with id="log"
```

Invocation on DOM subtrees

- All of the above methods can also be invoked on DOM elements not just the document
    - Search is confined to subtree rooted at element
- Example: Assume element with id="log" exists

```
var log = document.getElementById("log");
var error = log.getElementsByClassName("error");
if (error.length ==0) { ... }
```

Class Activity

- Assume the page contains a div element with id id, which contains a series of images (img nodes).
- Write a function that takes two arguments, id and interval. At each interval, the images must be "rotated", i.e., image0 will become image1, image1 will become image2, etc.
- See HTML and JS (icons at the top-right) for boilerplate code to use for this exercise.

```
function changeImages(id, interval) {

}
```

Helper

To repeat the execution of a given function f at a specific interval (e.g. 1000 ms): setInterval(1000, f);

DOM Traversal

1. DOM: Recap
2. DOM Access APIs
3. DOM Traversal
4. Adding and removing nodes

Traversing the DOM

- Since the DOM is just a tree, you can walk it the way you'd do with any other tree
  - Typically using recursion
- Every browser has minor variations in implementing the DOM, so should not be sensitive to such changes
  - Traversing DOM this way can be fragile

Before accessing or manipulating the DOM...

Problem

- When your JS code executes, the page might not have finished loading
  - ⇒The DOM tree might not be fully instanciated / might change!

window.onload

- *Event* that gets fired when the DOM is fully loaded (we'll get back to events later...)
- You can give a callback function to execute upon proper loading of the DOM.
- Your DOM manipulation code should go inside that function

```
// DOM Level 1 way shown below -- not recommended!. How to do it with DOM Level 2?
window.onload = function() { /* Access the DOM here... */ }
```

Properties for DOM Traversal

parentNode

Parent node of this one, or null

childNodes

A read only array-like object containing all the (live) child nodes of this one

firstChild, lastChild

The first and lastChild of a node, or null if it has no children

nextSibling, previousSibling

The next and previous siblings of a node (in the order in which they appear in the document)

Other node properties

nodeType: 'kind of node'

- Document nodes: 9
- Element nodes: 1
- Text nodes: 3
- Comment node: 8

nodeValue

Textual content of Text of comment node

nodeName

Tag name of a node, converted to upper-case

Example: Find a Text Node

- We want to find the DOM node that has a certain piece of text, say “text”
- Return true if text is found, false otherwise
- We need to recursively walk the DOM looking for the text in all text nodes

```
function search(node, text) {
    /* ... */
};

var result = search(window.document, "Hello world!");
```

Solution to Exercise

```
function search(node, text) {
    var found = false;
    if (node.nodeType==3) {
        if (node.nodeValue === text) found = true;
    } else { // textNodes cannot have children
        var cn = node.childNodes;
        if (cn) {
            for (var i=0; i < cn.length; i++) {
                found = found || search(cn[i], text);
            }
        }
    }
    return found;
};

var result = search(window.document, "Hello world!");
```

Class Activity

- Write a function that will traverse the DOM tree rooted at a node with a specific id, and checks if any of its sibling nodes and itself in the document is a text node, and if so, concatenates their text content and returns it.
- Can you generalize it so that it works for the entire subtree rooted at the sibling nodes ?

Adding and removing nodes

1 DOM: Recap

2 DOM Access APIs

3 DOM Traversal

4 Adding and removing nodes

Adding and removing nodes

- DOM elements are also JavaScript Objects (in most browsers) and consequently can have their properties read and written to
  - Can extend DOM elements by modifying their prototype objects
  - Can add fields to the elements for keeping track of state (E.g., visited node during traversals)
  - Can modify HTML attributes of the node such as width etc. – changes reflected in browser display

Creating New and Copying Existing DOM Nodes

Creating New DOM Nodes

- Using either document.createElement(“element”) OR document.createTextNode(“text content”)

```
var newNode = document.createTextNode("hello");
var elNode = document.createElement("h1");
```

Copying Existing DOM Nodes: use *cloneNode*

- Single argument can be true or false
  - True: deep copy (recursively copy all descendants)
- new node can be inserted into a different document

```
var existingNode = document.getElementByid("my");
var newNode = existingNode.cloneNode( true );
```

Inserting Nodes

appendChild

Adds a new node as a child of the node it is invoked on. node becomes *lastChild*

insertBefore

Similar, except that it inserts the node before the one that is specified as the second argument (*lastChild* if it's null)

```
var s = document.getElementByID("my");
s.appendChild(newNode);
s.insertBefore(newNode, s.firstChild);
```

Removing and replacing nodes

Removing a node *n*: *removeChild*

```
n.parentNode.removeChild(n);
```

Replacing a node *n* with a new node: *replaceChild*

```
n.parentNode.replaceChild(
    document.createTextNode("[redacted]"),
    n);
```

Class Activity

Class Activity

Write a function newdiv that takes two parameters: a node n and a string id. The function should replace node n by making it a child of a new div element with id = “id”.

```javascript
// function to replace a node n by making it a child of a
    new "div" element with id = "id"
function newdiv(n, id) {
   // Write your code here
};
```

Class Activity – Solution

Class Activity

Write a function newdiv that takes two parameters: a node n and a string id. The function should replace node n by making it a child of a new div element with id = “id”.

```javascript
// function to replace a node n by making it a child of a new "div" element with id = "id"
function newdiv(n, id) {
    var div = document.createElement("div");
    div.id = id;
    n.parentNode.replaceChild(div, n);
    div.appendChild(n);
};
```

Table of Contents

1. DOM: Recap
2. DOM Access APIs
3. DOM Traversal
4. Adding and removing nodes