Chapter 12: Document Object Model

CS 80: Internet Programming

Instructor: Mark Edmonds

Introduction

- The Document Object Model gives you access to every HTML element on the page
- So far, we've only written new elements to the page using `document.writeln()`
  - But this is restrictive and unintuitive
- We learned all of this information on how to build HTML documents by writing actual HTML
  - We want the best of both worlds: dynamic changes to the page while having a default, interpret to use HTML structure

DOM Nodes and Trees

- DOM Tree represents the document
  - The tree is built based on the nesting of HTML tags in the document
    - Nested nodes are children of the containing HTML element in the tree
      - E.g. if this the `<p>` tag is inside of a `<body>` tag, the `<body>` node is the parent of the child `<p>` node
  - You've been looking at the DOM Tree in the "Elements" view of the developer tools

DOM Nodes and Trees

- DOM nodes are elements in the tree (which directly correspond to HTML elements)
  - Every piece of an HTML5 page (elements, attributes, text) is modeled by a DOM node
- Let's take a look at what the DOM looks like and take a look in the debugger

DOM Basics

- A critical, simple, and effective method
  - `document.getElementById("html_id")`
    - This returns an object representation of an HTML element, specifically, the HTML element with the specified "html_id"
    - This object is a DOM node in the DOM tree

    * Now we have direct, Javascript access to HTML elements!
        · Means we can interact with existing HTML elements through Javascript

DOM Basics

- Let's dig into the object that `getElementById` returns:
  - An "Element"
  - https://developer.mozilla.org/en-US/docs/Web/API/Element

DOM Basics

Important Methods

- `currentNode.getAttribute(attr_name)` - gets an attribute from the node (specifically `attr_name`). Attributes are HTML attributes, such as **`class`** or `src`, etc.
- `currentNode.setAttribute(attr_name, value)` - sets an attribute from the node (specifically `attr_name` to `value`). Attributes are HTML attributes, such as **`class`** or `src`, etc.
- `currentNode.removeAttribute(attr_name)` - removes an attribute from the node (specifically `attr_name`). Attributes are HTML attributes, such as **`class`** or `src`, etc.
- `document.createElement("HTMLtag")` - creates an HTML tag of type `HTML_tag`. E.g. `var node = document.createElement("p");` creates a paragraph tag

DOM Basics

Important Methods

- `currentNode.appendChild(child_node)` - appends the DOM node `child_node` to to the `node`. Note that `child_node` must be a constructed DOM node
- `currentNode.insertBefore(newNode, referenceNode)` - inserts the new node before the reference node as a child of the current node
- `currentNode.replaceChild(newChild, oldChild)` - replaces current node's the old child with the new child
- `currentNode.removeChild(child)` - removes a child node (note that this function returns the child node)

DOM Basics

Important Attributes

- `innerHTML` - accesses this node's inner HTML. This is the text to markup. E.g. if `<p>` is the `currentNode`. `currentNode.innerHTML` accesses the text within the `<p>` tag
- `parentNode` - accesses this node's parent HTML node.
- `length` - tells how many children node this node has

DOM Basics

- Changing/Setting the innerHTML of a node

```
// Case 1: create new node
var h1_node = document.createElement("h1"); // create new h1_node
h1_node.innerHTML = "Hello, World!"; // change the inner HTML
    using innerHTML
document.body.appendChild(h1_node); // insert our new node to the
    document's body (document.body gives you the required &lt;body&
    gt; HTML tag)
// Case 2: modify an existing node
var h1_node = document.getElementById("h1_ele"); // assume there
    is an &lt;h1 id="h1_ele"&gt; tag in the document
h1_node.innerHTML = "Hello, World!"; // change the inner HTML
    using innerHTML
```

Example: `editing_dom.html`

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <title>Using Javascript to edit HTML</title>
  <script>
    // wait until the DOM is loaded; we can't interact with it until it
        is loaded
    document.addEventListener('DOMContentLoaded', function() {
      // change the title
      updateHeader("header");

      // change the image
      updateImage("img");

```

```
16        // create new p element
17        createElement("p", document.body, "Hello, World!");
18
19        // create new element without innerHTML
20        createElement("div", document.body);
21
22        // change the title again
23        updateHeader("header");
24      });
25
26      // update header with new text
27      function updateHeader(id) {
28        var new_header = window.prompt("Enter a new title: ");
29        var header = document.getElementById(id);
30        if (header) {
31          header.innerHTML = new_header;
32        } else {
33          console.log("No header with id " + id);
34        }
35      }
36
37      // update the image
38      function updateImage(id) {
39        var new_img_src = window.prompt("Enter a new image URL: ");
40        var new_img_alt = window.prompt("Enter a new image alt: ");
41        var img = document.getElementById(id);
42        img.setAttribute("src", new_img_src);
43        img.setAttribute("alt", new_img_alt);
44      }
45
46      // create a new element
47      function createElement(element, parentNode, innerHTML) {
48        var new_node = document.createElement(element);
49        if (innerHTML){
50          new_node.innerHTML = innerHTML;
51        }
52        parentNode.appendChild(new_node);
53      }
54    </script>
55  </head>
56
57
58  <body>
```

```
59
60    <h1 id="header">We can edit HTML using DOM</h1>
61    <img id="img" src="http://www.menucool.com/slider/jsImgSlider/images/
          image-slider-2.jpg" alt="scene from movie Up">
62
63  </body>
64
65  </html>
```

Traversing DOM

- A complicated, intimidating, and informative example of how you can interact with a DOM tree
  - The javascript (`traversing_dom/dom.js`) is intimidating. I encourage you to dive into it. It will make more sense by the end of the lecture.

Example: `traversing_dom/dom.html`

```
1  <!DOCTYPE html>
2  <!-- Fig. 12.4: dom.html -->
3  <!-- Basic DOM functionality. -->
4  <html>
5
6  <head>
7    <meta charset="utf-8">
8    <title>Basic DOM Functionality</title>
9    <link rel="stylesheet" type="text/css" href="style.css">
10   <script src="dom.js"></script>
11 </head>
12
13 <body>
14   <h1 id="bigheading" class="highlighted">
15 [bigheading] DHTML Object Model</h1>
16   <h3 id="smallheading">[smallheading] Element Functionality</h3>
17   <p id="para1">[para1] The Document Object Model (DOM) allows for
         quick, dynamic access to all elements in an HTML5 document for
         manipulation with JavaScript.</p>
18   <p id="para2">[para2] For more information, check out the "JavaScript
          and the DOM" section of Deitel's
19     <a id="link" href="http://www.deitel.com/javascript">
20 [link] JavaScript Resource Center.</a></p>
```

```
21    <p id="para3">[para3] The buttons below demonstrate:(list)</p>
22    <ul id="list">
23      <li id="item1">[item1] getElementById and parentNode</li>
24      <li id="item2">[item2] insertBefore and appendChild</li>
25      <li id="item3">[item3] replaceChild and removeChild</li>
26    </ul>
27    <div id="nav" class="nav">
28      <form onsubmit="return false" action="#">
29        <p><input type="text" id="gbi" value="bigheading">
30          <input type="button" value="Get By id" id="byIdButton"></p>
31        <p><input type="text" id="ins">
32          <input type="button" value="Insert Before" id="insertButton"></
              p>
33        <p><input type="text" id="append">
34          <input type="button" value="Append Child" id="appendButton"></p
              >
35        <p><input type="text" id="replace">
36          <input type="button" value="Replace Current" id="replaceButton
              "></p>
37        <p><input type="button" value="Remove Current" id="removeButton
             "></p>
38        <p><input type="button" value="Get Parent" id="parentButton"></p>
39      </form>
40    </div>
41  </body>
42
43  </html>
```

Event Listeners

- Event listeners trigger Javascript code when the event fires
- This enables the webpage to react to a users's actions
  - Similar to the `:hover` we saw trigger different CSS rules based on the mouse's position
- Conceptual example: the event could be clicking a button in the document, and the function could be updating a table based on the input

Event Listeners

- `addEventListener()`
  - Enables you link a function to an action in the HTML document

  - The "action" is an event on the webpage
  - Events include things like:
    * `mouseenter` (mouse enters the corresponding HTML element)
    * `click` (a button has been pressed and released
    * `submit` (form submit button is pressed)
  - A list of events available is here: https://developer.mozilla.org/en-US/docs/Web/Events

Event Listeners

- Syntax:

```
target.addEventListener(event_type, callback);
```

- `target` = HTML element to listen for `event_type`
  - When `target` (DOM node) has `event_type` (event) occur, `callback` (function) is called

Example: `loading_event_listeners.html`

```
<!DOCTYPE html>
<html>

<head>
  <meta charset=utf-8 />
  <title>Proper Loading</title>
  <script>

    function start(){
      document.getElementById("p1").addEventListener("mouseenter",
          addHighlight);
      document.getElementById("p1").addEventListener("mouseleave",
          removeHighlight);
    }

    document.addEventListener("DOMContentLoaded", start);

    // the following line will not work! DOM is not loaded (yet)
    //start();

    function addHighlight() {
      var p1 = document.getElementById("p1");
```

```
21        p1.setAttribute("style", "background-color: #3F6");
22      }
23
24      function removeHighlight() {
25        var p1 = document.getElementById("p1");
26        p1.removeAttribute("style");
27      }
28    </script>
29  </head>
30
31  <body>
32    <p id="p1">
33      Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque
          faucibus augue in risus tempus viverra. Etiam gravida augue a
          venenatis sollicitudin. Praesent varius ex varius, accumsan
          libero vel, bibendum eros. Aenean tristique mattis sem id
          scelerisque. In cursus ultrices massa nec tristique. Phasellus
          efficitur ac neque eu suscipit. Donec volutpat pretium justo,
          eget fringilla sapien. Integer vitae metus eget lorem auctor
          vestibulum non ut risus. Aenean hendrerit iaculis sapien. Nunc
          vestibulum purus quam, nec consequat sem cursus a. Aenean
          interdum euismod dui id dapibus. Curabitur vel placerat purus.
          Etiam dolor turpis, dictum in augue sit amet, suscipit suscipit
          leo. Aliquam auctor fringilla ligula, vitae sodales ligula
          facilisis quis. Donec consequat molestie tempus. Donec faucibus
          elit ullamcorper ante accumsan congue.
34    </p>
35  </body>
36
37  </html>
```

DOM Collections

- DOM contains a lot of information, most of which we won't usually care about
- Collections are groups of related objects on a page
- Each collection contains all of the elements of the corresponding type on the page.
- Each collection is stored under the `document` object
- Collections:
  - images
  - links

  - forms

Images Collection

- Stores all images on a page
- Accessed through `document.images` (returns a list of images)
- Each element in the list is an Image object
  - http://www.w3schools.com/jsref/dom_obj_image.asp
- First image's link can be accessed through `document.images[0].src`
  - `document.images[0].alt` is the alt display
- `document.images.length` returns the number of images
- Others are accessed similarly

Links Collection

- Stores all links on a page
- Accessed through `document.links` (returns a list of links)
- Each element in the list is a Link object
  - http://www.w3schools.com/jsref/dom_obj_link.asp
- First URL's link can be accessed through `document.links[0].href`
  - `document.links[0].innerHTML` is the displayed text
- `document.links.length` returns the number of images
- Others are accessed similarly

Forms Collection

- Stores all forms on a page
- Accessed through `document.forms` (returns a list of links)
- Each element in the list is a Form object
  - http://www.w3schools.com/jsref/dom_obj_form.asp
- First form's inputs can be accessed through `document.forms[0].elements` (this is a list as well!)
  - Take a look at what is available for forms here
  - http://www.w3schools.com/jsref/coll_form_elements.asp
- `document.forms.length` returns the number of forms
- Others are accessed similarly

Exercise

- Write a Javascript function named `getFormvalue()` to print the values of First and Last name on the following form to `console`
  - Note that the `onsubmit` attribute specifies a Javascript function to call upon submission (one way to do form validation!)
  - Use form object documentation to help http://www.w3schools.com/jsref/coll_form_elements.asp

Exercise: Corresponding HTML

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset=utf-8 />
    <title>Return first and last name from a form - w3resource</title>
  </head>
  <body>
    <form method="get" id="form1" onsubmit="getFormvalue()">
      First name:
      <input type="text" name="fname" value="David"><br>
      Last name:
      <input type="text" name="lname" value="Beckham"><br>
      <input type="submit" value="Submit">
    </form>
  </body>
</html>
```

Example: `first_last_name.html`

```
<!DOCTYPE html>
<html>

<head>
  <meta charset=utf-8 />
  <title>Return first and last name from a form - w3resource</title>
  <script>
    function getFormvalue() {
      var form = document.getElementById("form1");
```

```
      for (var i = 0; i < form.length; i++) {
        if (form.elements[i].value != 'Submit') {
          console.log(form.elements[i].value);
        }
      }
    }
  </script>
</head>

<body>
  <!-- action is intentionally left blank for demonstration purposes
     -->
  <form method="get" id="form1" onsubmit="getFormvalue()">
    First name: <input type="text" name="fname" value="David"><br>
    Last name: <input type="text" name="lname" value="Beckham"><br>
    <input type="submit" value="Submit">
  </form>
</body>

</html>
```

Exercise

- How could we have done this using `addEventListener()`?

Example: `first_last_name_listener.html`

```
<!DOCTYPE html>
<html>

<head>
  <meta charset=utf-8 />
  <title>Return first and last name from a form - w3resource</title>
  <script>
    function start(){
      // listen for the submit event on the form
      document.getElementById("form1").addEventListener("submit",
         getFormvalue);
    }

```

```
13      // the window object listens for the "load" event and executes
            start() when "load" triggers
14      window.addEventListener("load", start);
15
16      function getFormvalue() {
17        var form = document.getElementById("form1");
18        for (var i = 0; i < form.length; i++) {
19          if (form.elements[i].value != 'Submit') {
20            console.log(form.elements[i].value);
21          }
22        }
23      }
24    </script>
25  </head>
26
27  <body>
28    <!-- action is intentionally left blank for demonstration purposes
          -->
29    <form method="get" id="form1">
30      First name: <input type="text" name="fname" value="David"><br>
31      Last name: <input type="text" name="lname" value="Beckham"><br>
32      <input type="submit" value="Submit">
33    </form>
34  </body>
35
36  </html>
```

Model, View, Controller

- Model, View, Controller (MVC) is a common design pattern used for webpages (it is also used elsewhere in computer science)
- Goal: modularize and separate distinct components of an application into three broad categories: a model, a view, and a controller

Model, View, Controller

- Consists of three parts:
  - *Model*: data and corresponding logic of this application
  - *View*: interface/presentation to the user
  - *Controller*: logic that updates model and/or view in response to user input

Model, View, Controller

- Consists of three parts:
  - *Model*: the backend (business logic)
  - *View*: the frontend (user interface)
  - *Controller*: the orchestrator between the backend and frontend (controller between the two)

Model, View, Controller

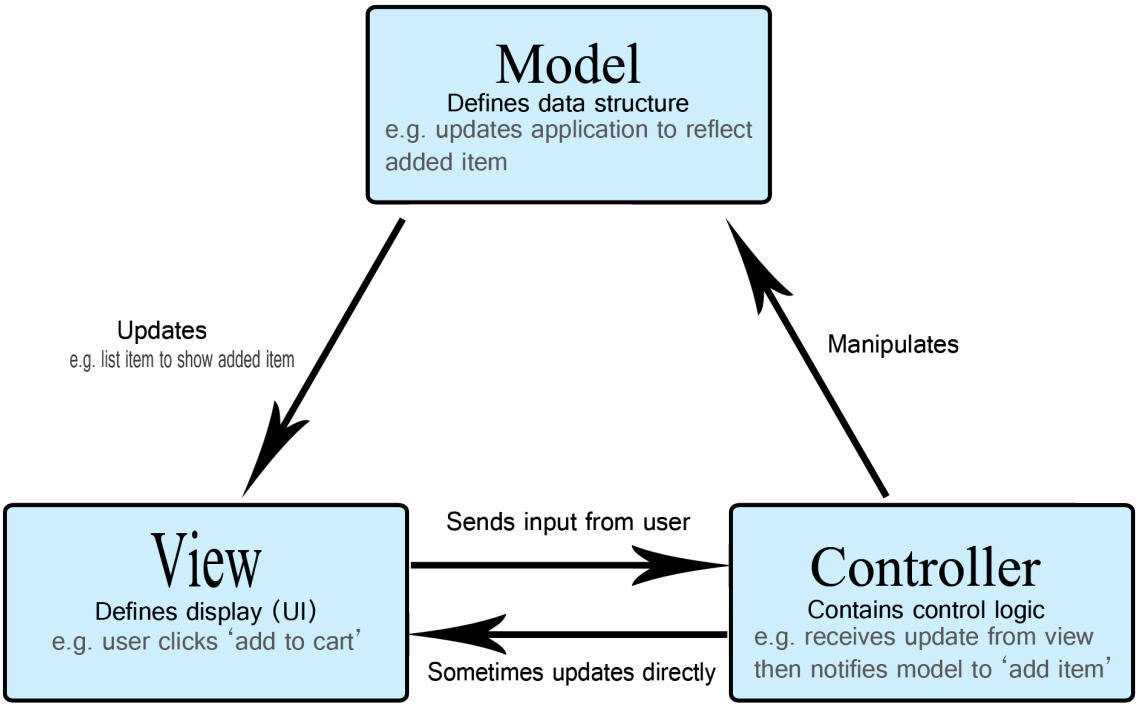

Figure 1: MVC architecture

Example: `model_view_controller.html`

```
1 <!DOCTYPE html>
2 <html>
```

```
3
4  <head>
5    <meta charset="utf-8">
6    <title>Basic Model, View, Controller Example</title>
7    <script>
8
9      function start() {
10       /* Model Handles the Data */
11       var model = {
12         data: {
13           userName : "John Doe",
14           userBirthYear : "1900",
15           userLoginID : "johndoe1900",
16         },
17         // set the model's data
18         setData: function(d){
19           this.data.userName = d.userName;
20           this.data.userBirthYear = d.userBirthYear;
21           this.generateLoginID()
22         },
23         getData: function(){
24           return this.data;
25         },
26         generateLoginID: function(){
27           // concatenate userName with userBirthYear
28           this.data.userLoginID = this.data.userName + this.data.
                userBirthYear;
29           // remove whitespace and make lowercase
30           this.data.userLoginID = this.data.userLoginID.replace(/ /g,''
                ).toLowerCase();
31         }
32       }
33
34       /* View Handles the Display */
35       var view = {
36         userName: document.getElementById("username"),
37         userBirthYear: document.getElementById("userbirthyear"),
38         userLoginID: document.getElementById("userloginid"),
39         getData: function(){
40           data = {};
41           data.userName = this.userName.value;
42           data.userBirthYear = this.userBirthYear.value;
43           return data;
```

```
44          },
45          updateView: function(userloginid) {
46            this.userLoginID.innerHTML = userloginid;
47          }
48        }
49
50        /* Controller Handles the Events */
51        var controller = {
52          model: model,
53          view: view,
54          handler: function(){
55            // get data from view (user)
56            viewData = this.view.getData();
57            // update the data in the model (and the model will update)
58            this.model.setData(data);
59            // get the updated model data (userLoginID has changed)
60            modelData = this.model.getData();
61            // update the view with the model's new data
62            this.view.updateView(modelData.userLoginID);
63          }
64        }
65
66        document.getElementById("update").addEventListener("click",
              function(){
67              controller.handler();
68        });
69      }
70
71      window.onload = start
72
73    </script>
74  </head>
75
76  <body>
77    <h1>Basic Model, View, Controller</h1>
78    <h3>Login ID generator</h3>
79    <p>
80      <label>Name:
81        <input type='text' id="username" value="John Doe">
82      </label>
83    </p>
84    <label>Birth Year:
85      <input type='text' id="userbirthyear" value="1900">
```

```
86    </label>
87    <button type="button" id="update">Update</button>
88    <p>
89      Your login ID is: <span id="userloginid"></span>
90    </p>
91  </body>
92
93  </html>
```