Document Object Model (DOM) Manipulation: Objects and Collections

Vijesh M. Nair
Assistant Professor
Dept. of CSE (AI-ML)

Introduction

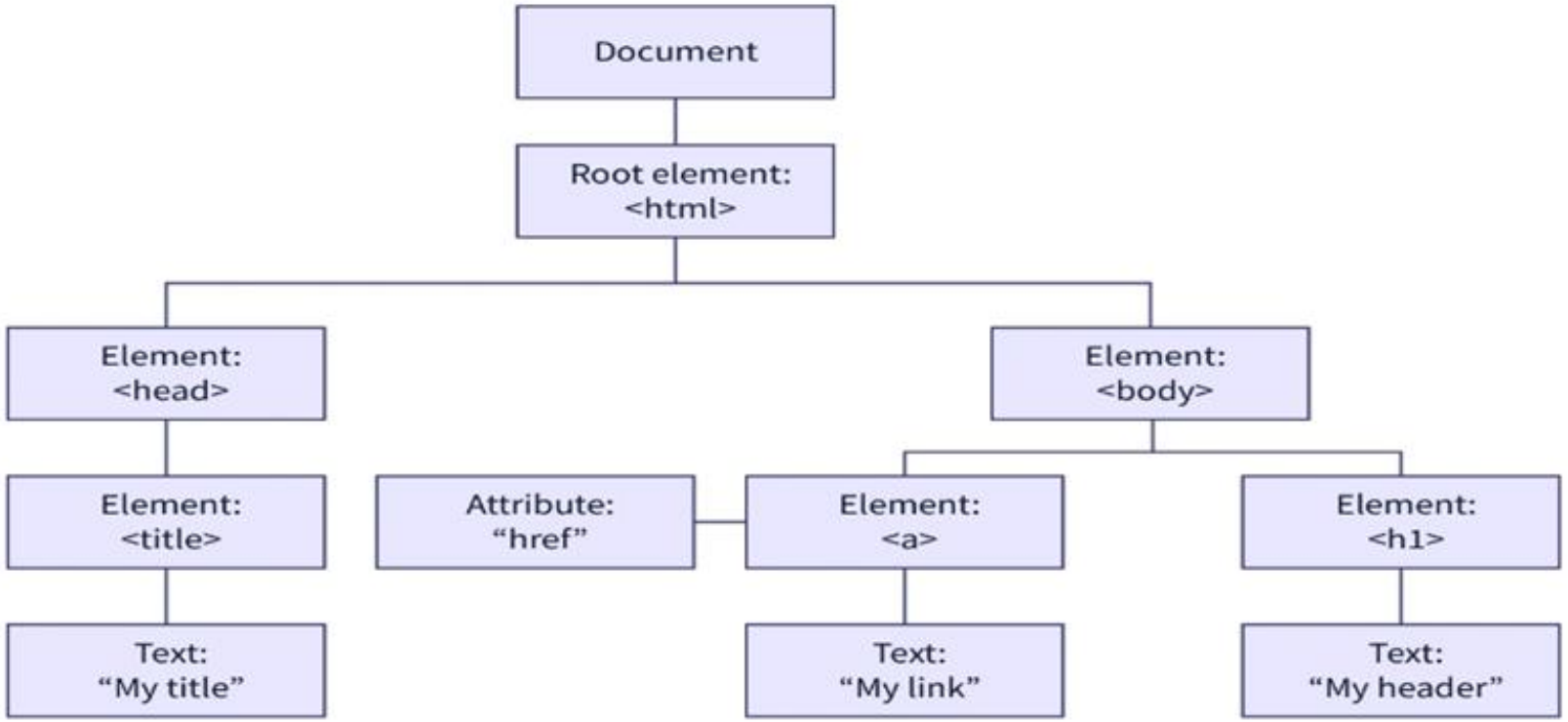
- The Document Object Model gives scripting access to all the elements on a web page.
- Using JavaScript, we can create, modify and remove elements in the page dynamically.

Software Engineering Observation 12.1

With the DOM, HTML5 elements can be treated as objects, and many attributes of HTML5 elements can be treated as properties of those objects. Then objects can be scripted with JavaScript to achieve dynamic effects.

Modeling a Document: DOM Nodes and Trees

- `getElementById` method
  - Returns objects called DOM nodes
  - Every piece of an HTML5 page (elements, attributes, text, etc.) is modeled in the web browser by a DOM node
- The nodes in a document make up the page's DOM tree, which describes the relationships among elements
- Nodes are related to each other through child-parent relationships
- A node can have multiple children, but only one parent
- Nodes with the same parent node are referred to as siblings
- The `html` node in a DOM tree is called the root node, because it has no parent


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”

```
1  <!DOCTYPE html>
2
3  <!-- Fig. 12.2: domtree.html -->
4  <!-- Demonstration of a document's DOM tree. -->
5  <html>
6     <head>
7        <meta charset = "utf-8">
8        <title>DOM Tree Demonstration</title>
9     </head>
10    <body>
11       <h1>An HTML5 Page</h1>
12       <p>This page contains some basic HTML5 elements. The DOM tree
13          for the document contains a DOM node for every element</p>
14       <p>Here's an unordered list:</p>
15       <ul>
16          <li>One</li>
17          <li>Two</li>
18          <li>Three</li>
19       </ul>
20    </body>
21 </html>
```

Demonstration of a document's DOM tree.

Demonstration of a document's DOM tree.

Traversing and Modifying a DOM Tree (Cont.)

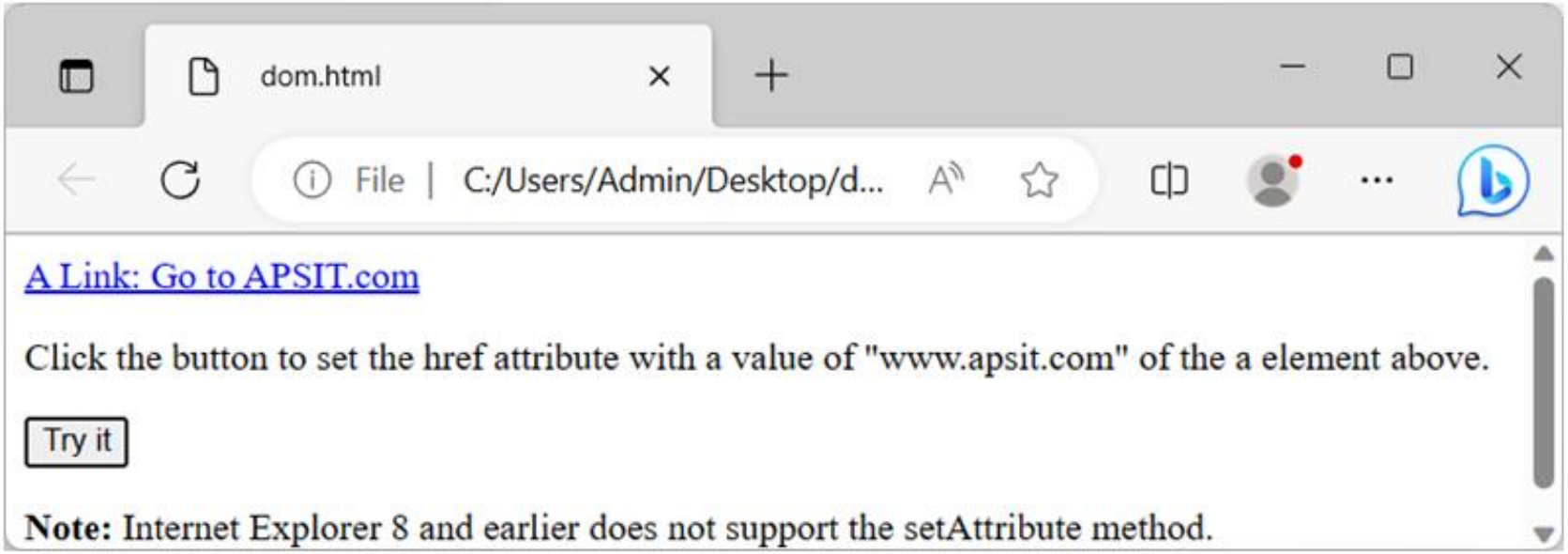
- The DOM element methods `setAttribute` and `getAttribute` allow you to modify an attribute value and get an attribute value, respectively.

- ```
  document.getElementById("myAnchor").
  getAttribute("target");
  ```

- ```
  document.getElementById("myAnchor").
  setAttribute("href",
  "https://www.apsit.edu.in/");
  ```

```
<!DOCTYPE html><html>
<head>
<script>function myFunction() {
document.getElementById("myAnchor").setAttribute("href"
, "https://www.apsit.edu.in/");}
</script></head>
<body>
<a id="myAnchor">A Link: Go to APSIT.com</a>
<p>Click the button to set the href attribute with a
value of "www.apsit.com" of the a element
above.</p>
<button onclick="myFunction()">Try it</button>
<p><strong>Note:</strong> Internet Explorer 8 and
earlier does not support the setAttribute method.</p>
</body></html>
```

Before clicking



After clicking

Traversing and Modifying a DOM Tree (Cont.)

- document object `createElement` method
  - Creates a new DOM node, taking the tag name as an argument. It does not insert the element on the page.
- document object `createTextNode` method
  - Creates a DOM node that contains only text. Given a string argument,
    - `createTextNode` inserts the string into the text node.
- Method `appendChild`
  - Inserts a child node (passed as an argument) after any existing children of the node on which it's called
- Property `parentNode` contains the node's parent
- `insertBefore` method
  - Inserts a node as a child, right before an existing child, which you specify.
  - node.insertBefore(newnode,existingnode)
- `replaceChild` method
  - Receives as its first argument the new node to insert and as its second argument the node to replace. node.replaceChild(newnode,oldnode)
- `removeChild` method
  - removes a specified child node of the specified element. Returns the removed node as a Node object, or null if the node does not exist. node.removeChild(node)

```
<!DOCTYPE html><html>
<head><script>
var para=document.createElement("p");

var node = document.createTextNode("This is new.");

para.appendChild(node);


var element = document.getElementById("div1");
element.appendChild(para);

</script></head>
<body>
<div id="div1"><p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p></div>

</body></html>
```

```
<!DOCTYPE html><html>
<head> <script>
function myFunction() {
var newItem = document.createElement("LI");
var textnode = document.createTextNode("Water");
newItem.appendChild(textnode);
var list = document.getElementById("myList");
list.insertBefore(newItem, list.childNodes[0]);}
</script> </head>
<body><ul id="myList">
  <li>Coffee</li>  <li>Tea</li></ul>
<p>Click the button to insert an item to the
list.</p><button onclick="myFunction()">Try
it</button><p><strong>Example
explained:</strong><br>First create a LI node,<br>
then create a Text node,<br> then append the Text
node to the LI node.<br>Finally insert the LI node
before the first child node in the list.</p>
</body></html>
```

- Coffee
- Tea

Click the button to insert an item to the list.

Try it

Example explained:
First create a LI node,
then create a Text node,
then append the Text node to the LI node.
Finally insert the LI node before the first child node in the list.

- Water
- Coffee
- Tea

Click the button to insert an item to the list.

Try it

Example explained:
First create a LI node,
then create a Text node,
then append the Text node to the LI node.
Finally insert the LI node before the first child node in the list.

```
<!DOCTYPE html><html><body>
<ul id="myList">
<li>Coffee</li><li>Tea</li><li>Milk</li></ul>

<p>Click the button to remove the first item
from the list.</p>
<button onclick="myFunction()">Try it</button>

<script>function myFunction() {
var list = document.getElementById("myList");
list.removeChild(list.childNodes[0]);}</script>

</body></html>
```

- Coffee
- Tea
- Milk

Click the button to see if the ul element has any child nodes. If so, remove its first child node (index 0).

Try it

- Tea
- Milk

Click the button to see if the ul element has any child nodes. If so, remove its first child node (index 0).

Try it

DOM Collections

- DOM has collections—groups of related objects on a page
- DOM collections are accessed as properties of DOM objects such as the document object or a DOM node
- The document object has properties containing the images collection, links collection, forms collection and anchors collection
  - Contain all the elements of the corresponding type on the page
- The collection's length property specifies the number of items in the collection

DOM Collections (Cont.)

- We access the elements of the collection using indices in square brackets
- `item` method of a DOM collection
  - An alternative to the square bracketed indices
  - Receives an integer argument and returns the corresponding item in the collection.
- `namedItem` method
  - receives an element `id` as an argument and finds the element with that `id` in the collection.
- `href` property of a DOM link node
  - Refers to the link's `href` attribute
- Collections allow easy access to all elements of a single type in a page
  - Useful for gathering elements into one place and for applying changes across an entire page

```
1   /* Fig. 12.12: style.css */
2   /* CSS for collections.html. */
3   body            { font-family: arial, helvetica, sans-serif }
4   h1              { font-family: tahoma, geneva, sans-serif;
5                     text-align: center }
6   p a             { color: DarkRed }
7   ul              { font-size: .9em; }
8   li              { display: inline;
9                     list-style-type: none;
10                    border-right: 1px solid gray;
11                    padding-left: 5px; padding-right: 5px; }
12  li:first-child { padding-left: 0px; }
13  li:last-child  { border-right: none; }
14  a               { text-decoration: none; }
15  a:hover         { text-decoration: underline; }
```

CSS for `collections.html`.

```
1  <!DOCTYPE html>
2
3  <!-- Fig. 12.13: collections.html -->
4  <!-- Using the links collection. -->
5  <html>
6     <head>
7        <meta charset="utf-8">
8        <title>Using Links Collection</title>
9        <link rel = "stylesheet" type = "text/css" href = "style.css">
10       <script src = "collections.js"></script>
11    </head>
12    <body>
13       <h1>Deitel Resource Centers</h1>
14       <p><a href = "http://www.deitel.com/">Deitel's website</a>
15          contains a growing
16          <a href = "http://www.deitel.com/ResourceCenters.html">list
17          of Resource Centers</a> on a wide range of topics. Many
18          Resource centers related to topics covered in this book,
19          <a href = "http://www.deitel.com/books/iw3htp5">Internet &
20          World Wide Web How to Program, 5th Edition</a>. We have
21          Resource Centers on
22          <a href = "http://www.deitel.com/Web2.0">Web 2.0</a>,
23          <a href = "http://www.deitel.com/Firefox">Firefox</a> and
24          <a href = "http://www.deitel.com/IE9">Internet Explorer 9</a>,
```

Using the `links` collection. (Part 1 of 2.)

```
25            <a href = "http://www.deitel.com/HTML5">HTML5</a>, and
26            <a href = "http://www.deitel.com/JavaScript">JavaScript</a>.
27            Watch for related new Resource Centers.</p>
28         <p>Links in this page:</p>
29         <div id = "links"></div>
30      </body>
31   </html>
```

Using the `links` collection. (Part 2 of 2.)

```
1  // Fig. 12.14: collections.js
2  // Script to demonstrate using the links collection.
3  function processLinks()
4  {
5     var linksList = document.links; // get the document's links
6     var contents = "<ul>";
7
8     // concatenate each link to contents
9     for ( var i = 0; i < linksList.length; ++i )
10    {
11       var currentLink = linksList[ i ];
12       contents += "<li><a href='" + currentLink.href + "'>" +
13          currentLink.innerHTML + "</li>";
14    } // end for
15
16    contents += "</ul>";
17    document.getElementById( "links" ).innerHTML = contents;
18 } // end function processLinks
19
20 window.addEventListener( "load", processLinks, false );
```

Script to demonstrate using the links collection.

Dynamic Styles

- An element's style can be changed dynamically
  - E.g., in response to user events
  - Can create mouse-hover effects, interactive menus and animations
- The document object's body property
  - Refers to the body element
- The `setAttribute` method is used to set the `style` attribute with the user-specified color for the `background-color` CSS property.
- If you have predefined CSS style classes defined for your document, you can also use the `setAttribute` method to set the `class` attribute.

```
1  <!DOCTYPE html>
2
3  <!-- Fig. 12.15: dynamicstyle.html -->
4  <!-- Dynamic styles. -->
5  <html>
6     <head>
7        <meta charset="utf-8">
8        <title>Dynamic Styles</title>
9        <script src = "dynamicstyle.js"></script>
10    </head>
11    <body>
12       <p>Welcome to our website!</p>
13    </body>
14 </html>
```

Dynamic styles. (Part 1 of 2.)

Fig. 12.15 | Dynamic styles. (Part 2 of 2.)

```
1  // Fig. 12.16: dynamicstyle.js
2  // Script to demonstrate dynamic styles.
3  function start()
4  {
5     var inputColor = prompt( "Enter a color name for the " +
6        "background of this page", "" );
7     document.body.setAttribute( "style",
8        "background-color: " + inputColor );
9  } // end function start
10
11 window.addEventListener( "load", start, false );
```

Fig. 12.16 | Script to demonstrate dynamic styles.

Thank You!