DOM

Marco Torchiano

Version 2.4.1 - May 2020

License

Introduction

DOM

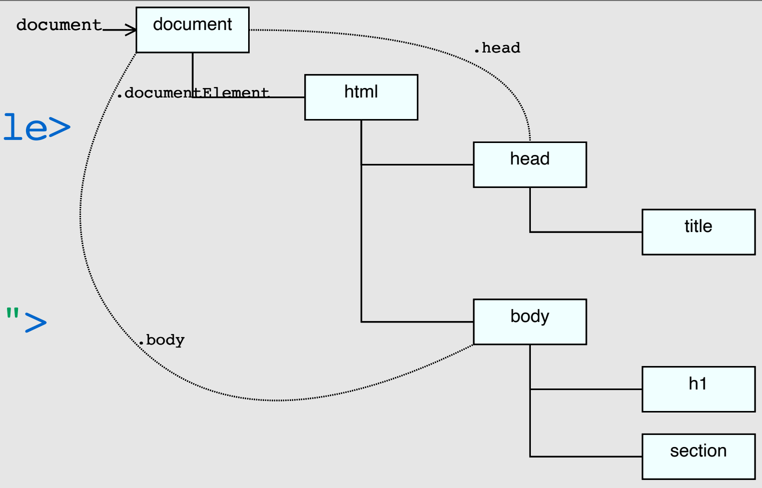
Document Object Model

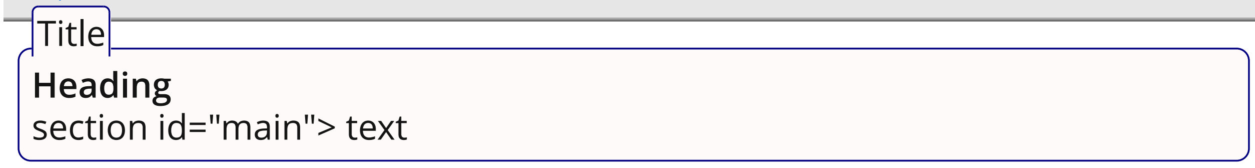
Set of objects that represent the structure of a document loaded in the browser. Can be used to:

- explore the content of the document
    - e.g. get lists of elements,
- modify the attributes of the elements
    - e.g. class or style,
- change the structure of the document
    - i.e. add and remove elements.

DOM Example

```
<html>
<head>
  <title>Title</title>
</head>
<body>
  <h1>Heading</h1>
  <section id="main">
  text
  </section>
</body>
</html>
```





Title

Heading
section id="main"> text

Type of objects

The DOM is a graph where the (Element) objects are **nodes** and direct containment is represented by **edges**

- `Text` nodes represent the text fragments
- `Comment` nodes represent comments
- `HTMLElement` nodes represent elements
  - There are specific types of nodes e.g. `HTMLBodyElement`, `HTMLDivElement`, …

Retrieve elements

- `getElementById()` the element with the given *id*
- `getElementsByTagName()` all the elements with a given tag
- `getElementsByClassName()` all the elements with a given tag
- `childNodes` the elements included in the current one
- `children` as above excluding text and comments

Node lists

Multiple nodes are collected in:

- `NodeList` when all nodes are required
  - e.g. through `childNodes`
- `HTMLCollection` for HTML elements only
  - e.g. through `children`

Both object types can be iterated using:

- `length`: number of elements
- `[]` or `item()`: return the element at given index

Node lists

```
var pars = document.getElementsByTagName("p");
console.log("Found " + pars.length+ " paragraphs");
```

↪
```
Found 40 paragraphs
```

```
for(let n of pars[2].childNodes)
  console.log(n.nodeName + " : " +
              n.textContent.slice(0,28)+
              (n.textContent.length>28?"...":""));
```

↪
```
#text : This work is licensed under ...
A : Creative Commons Attribution...
#text : .
```

Elements

Special collections inside `document`:

- `documentElement` the root element (i.e. `<html>`)
- `body` returns the `<body>` element
- `head` returns the `<head>` element
- `title` returns the content of `<title>` element
- `forms` returns all `<form>` elements
- `links` returns all `<a>` elements with `href`
- `scripts` returns all `<script>` elements
- `images` returns all `<img>` elements

Create content

- `innerHTML` sets the internal HTML of the element

```
var s=document.getElementById("create-content");
s.innerHTML += "<p>The new content!</p>";
```

- *attribute* access any attribute of element
  - `style` accesses the element's style, the individual style attributes can be accessed as sub-properties

```
s.style.color = "navy"; → "navy"
```

The new content!

DOM manipulation: creation

DOM nodes can be created using `document`:

- `createElement( type )` creates an element of the given type
- `createTextNode( content )` creates a text node

Elements and nodes are created as *detached*, they need to be attached and placed somewhere in the current DOM graph.

DOM manipulation: placement

The created items can be attached and placed with respect to an existing element:

- `appendChild( node )` appends the node as the last child of the element
- `insertBefore( node, ref)` appends the node after the `ref` node
- `removeChild( node )` removes the child node

DOM manipulation: placement

- `insertAdjacentHTML('pos',text)` add HTML code w.r.t. element
  - `pos` can be `beforebegin`, `afterend`, `afterbegin`, `beforeend`
  - `text` is HTML code that is parsed and then added
- `insertAdjacentElement(pos,element)` similar to above but adds an element

`appendChild` example

```
var t = document.getElementById('tgAppend');
var n = document.createElement('em');
n.appendChild( document.createTextNode(' Hi!') );
t.appendChild(n);
```

```
<p id="tgAppend">Content of paragraph</p>
```

Content of paragraph *Hi!*

`insertBefore` example

```
var t = document.getElementById('tgInsert');
var ref = document.getElementById('tgRef');
var n = document.createTextNode(' THE ');
t.insertBefore( n, ref );
```

```
<p id="tgInsert">Content of
   <b id="tgRef">paragraph</b></p>
```

Content of THE **paragraph**

Adjacent HTML example

```
var t = document.getElementById("tgAdjacent");
t.insertAdjacentHTML('beforebegin','<b>BB</b>');
t.insertAdjacentHTML('afterbegin','<b>AB</b> ');
t.insertAdjacentHTML('beforeend',' <b>BE</b>');
t.insertAdjacentHTML('afterend','<b>AE</b>');
```

```
<p id="tgAdjacent" style="background:aqua">Content</p
```

BB
AB Content **BE**

AE

Events

Events are triggered by specific actions performed on the elements of a page.

Events can be handled by means of specific methods called *handlers*:

- elements object expose special properties corresponding to events
- such properties can be assigned to functions that will handle them
  - in javascript e.g. `element.onclick = func`,
  - in html e.g. `<p onclick="func()">`

Events

In event handlers `this` refers to the element destination of the event.

```
var a = document.getElementsByTagName("h2")[17]
a.onmouseover=function(){
    this.style.color = "red";
    this.style.fontWeight = "normal";
};
a.onmouseout=function(){
    this.style.color = "black";
    this.style.fontWeight = "bold";
};
```

Common events

- `onclick` click on the element
- `onmouseover` mouse moving over the element
- `onmouseout` mouse exiting from element
- `onkeypress` keyboard key pressed
- `onchange` form input element value is changed
- `onload` the object (e.g. the page) has been loaded

The event are triggered for an element and all its containers.

BOM

Browser Object Model

The main browser object is `window`; it contains the properties:

- `document`
- `location`
- `innerHeight`, `innerWidth`
- `screen`
- `history`

Note: all properties of `window` are directly accessible, i.e. `window` is the *global* context.

Location

The `location` property object includes

- `href`: the full URL
- `hostname`, e.g. `softeng.polito.it`
- `protocol`, e.g. `http:`
- `pathname`, e.g. `/courses/VIQ/`
- `search`, e.g. `?par=val`

Can be assigned to navigate to a new URL in the same window/tab.

Window on load event

Operations on the DOM must be performed after its structure is complete and stable.

The `onload` event can be used to trigger operations on DOM once stable.

```
window.onload = function(){
  // place here the code working on DOM
}
```

Window: methods

Main methods:

- `open()` open a URL in a new window/tab
- `alert()` show notification window
- `confirm()` show a dialg to ask a question,
    - returns a boolean
- `prompt()` show a dialog to ask for information,
    - returns the string entered

Window: Timing

- `setTimeout(function, milliseconds)`
  Executes a function, after waiting a specified number of milliseconds.
- `setInterval(function, milliseconds)`
  Executes a function repeatedly every given milliseconds.
- `clearInterval( itvlObj )`,
  `clearTimeout( itvlObj )`
  Cancel the scheduled invocation
    - use the handle returned by `set..` function

Animation Example

Using an interval to change attributes:

```
(function(){  // using a closure
  var i=0;
  var step=1;
  var h = document.getElementsByTagName("h2")[25];
  setInterval(function(){ // function to be called
    if(i>=255) step=-1;
    if(i<=0) step=+1;
    h.style.color='rgb('+i+','+i+','+i+')';
    i+=step;
  },10); // every 10 ms
})();
```

References

- W3Schools. JavaScript Tutorial. http://www.w3schools.com/js/default.asp
- W3Schools- JavaScript Reference. http://www.w3schools.com/jsref/default.asp
- W3C - DOM Parsing. https://www.w3.org/TR/DOM-Parsing/
- DOM Examples:
  - On Codepen: http://codepen.io/mtorchiano/pen/VaEmOm?editors=1010
- W3Schools. jQuery Tutorial. https://www.w3schools.com/jquery/default.asp