JS Training 2020: DOM Manipulation

March 17, 2020

Tamás Faragó
Junior Software Engineer

Who am I?

Tamás Faragó

Junior Software Engineer

- Working at EPAM since 2019 August
- This is my fourth job
  - EPAM is awesome, join us :)
- I love badminton

Agenda

1	DOM and its structure
2	How will you get a rendered page out of an HTML/XML?
3	Get and set simple attributes of DOM elements
4	DOM events

Data Object Model

The Document Object Model (DOM) is a programming **API** for HTML and XML documents. It **defines the logical structure of documents and the way a document is accessed and manipulated**. In the DOM specification, the term "document" is used in the broad sense - increasingly, XML is being used as a way of representing many different kinds of information that may be stored in diverse systems, and much of this would traditionally be seen as data rather than as documents. Nevertheless, XML presents this data as documents, and the DOM may be used to manage this data.

As a W3C specification, one important objective for the Document Object Model is **to provide a standard programming interface** that can be used in a **wide variety of environments and applications**. The Document Object Model **can be used with any programming language**. In order to provide precise, language-independent specification of the Document Object Model interfaces...

source: https://www.w3.org/TR/WD-DOM/introduction.html

Document as an abstract concept

A Document is an **abstract concept**, which has no definition. A document is described by a markup language. The most typically used markup languages are XML and **HTML**.

As DOM is an Application Programming **Interface**, it needs to be implemented somewhere. Each browser builds its own concrete document and uses its own implementation for DOM. This DOM API provides the possibility to change the [concrete] documents' content and functionality in "runtime".

IMPORTANT NOTE: Document != HTML && Document != XML

These languages are tools for describing a document. But the document itself is an abstract thing. (This means, if you change something in the [concrete] document with the help of the DOM it won't be reflected in the HTML or XML file.)

DOM manipulation (usage of API)

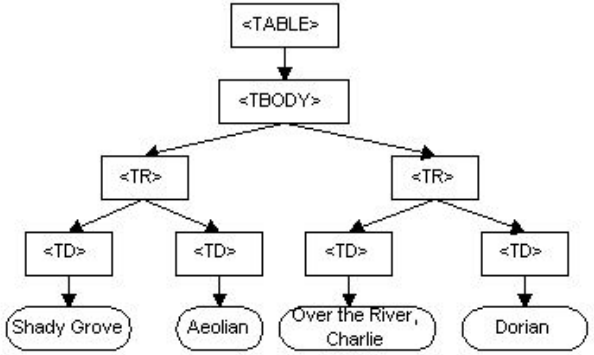
The API provides a way to access and modify the attributes of the DOM elements.

I.e. we can get the size of the window of the browser width or get an element by id, and set it disabled. In the following slides we'll look at a bunch of examples.

DOM structure

The DOM has its own structure. As a defacto standard the tree data model has become the structure of the DOM.

```
1.   <TABLE>
2.    <TBODY>
3.      <TR>
4.        <TD>Shady Grove</TD>
5.        <TD>Aeolian</TD>
6.      </TR>
7.      <TR>
8.        <TD>Over the River, Charlie</TD>
9.        <TD>Dorian</TD>
10.     </TR>
11.   </TBODY>
12.  </TABLE>
```



graphical representation of the DOM of the example table

Source:
https://www.w3.org/TR/DOM-Level-2-Core/introduction.html

How will you get a rendered page out of an HTML/XML?

1. **Parsing HTML/XML:**

   First the browser will read the content of the HTML/XML character-by-character, and “tokenize” it. Tokenization means that we split up the content into smaller pieces that are understandable for the browser.

   After that, out of these pieces it will build nodes, then connect them into a tree, called DOM-tree.

How will you get a rendered page out of an HTML/XML?

2. **Parsing stylesheets:**

   While parsing the HTML/XML we might bump into tokens, that are about styling. I.e.

   - style attribute on a tag: *style=”background-color: red”*
   - *<style>* tag,
   - *<link>* tag with *rel=”stylesheet”* attribute

     <link href="/media/examples/link-element-example.css" rel="stylesheet">

   Basically we will have a very similar process: tokenization → nodes → tree

   This tree is called CSS Object Model (CSSOM) tree.

How will you get a rendered page out of an HTML/XML?

3. **Generating rendering tree:**

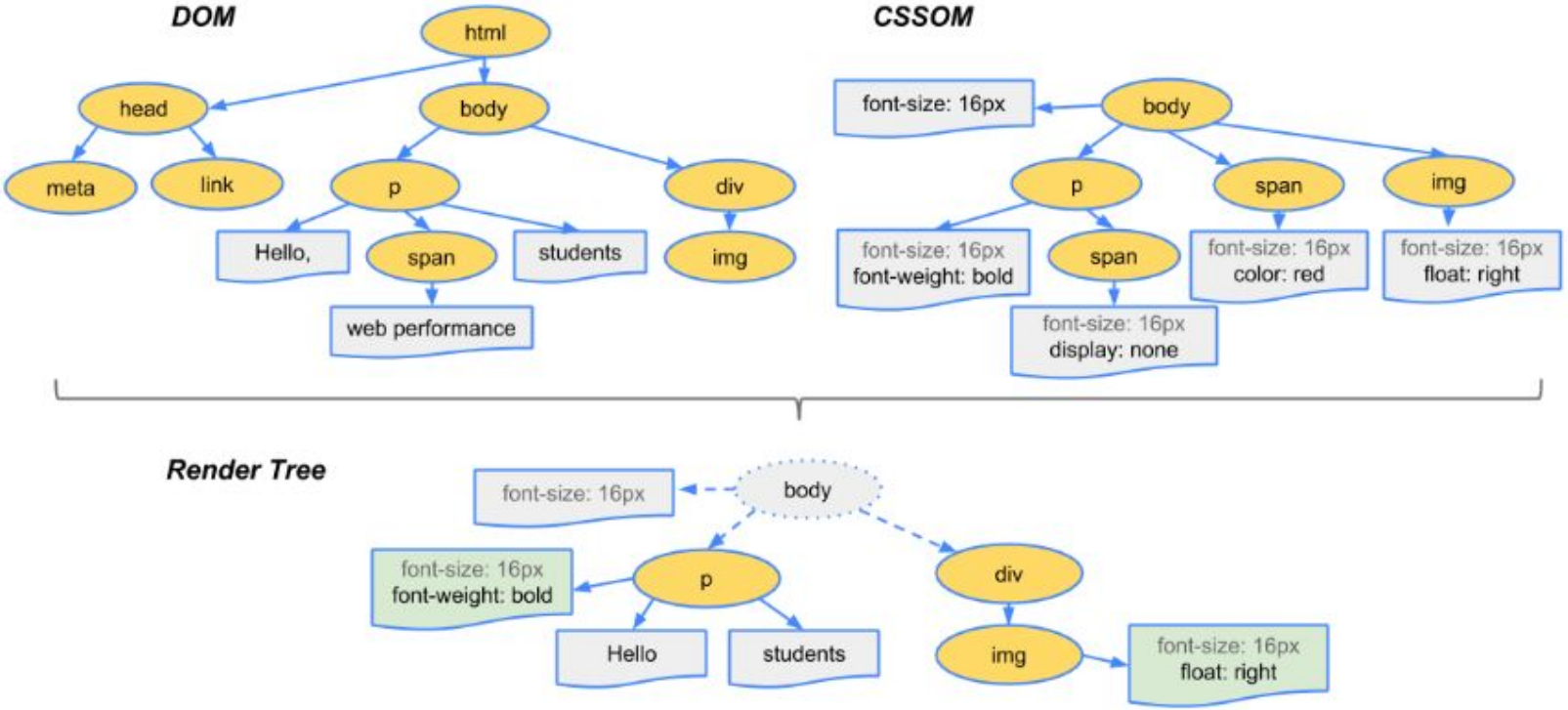
   Rendering tree is eventually a combination of DOM and CSSOM. This tree will contain all the elements that should be rendered.

   The elements that are part of the DOM, but in the CSSOM they have *Display: none* will be ultimately left out.

   Also metadata is left out.

   Sidenote: If an element has *visibility: hidden* it will be included in rendering tree.

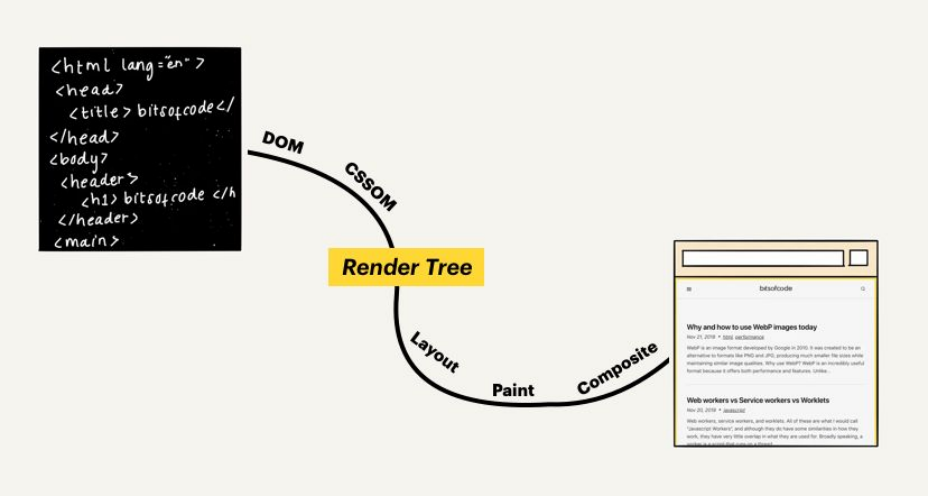
How will you get a rendered page out of an HTML/XML?



Source: https://developers.google.com/web/fundamentals/performance/critical-rendering-path/render-tree-construction

How will you get a rendered page out of an HTML/XML?

4. **Layout:** Calculating size of the elements
5. **Paint:** Fill the pixels
6. **Composite:** Draw the elements in the correct order



Source: https://bitsofco.de/what-exactly-is-the-dom/

Render blocking resources

During the HTML/XML parsing process we can bump into **external resources** such as styles, images, scripts.

While fetching images are not blockers, styles and scripts can block the rendering process.

DEMO TIME

The element that is outside of the document: window

The window object represents an open window in a browser that contains a document. You can access the document through window object: `window.document`

This object also contains:

- attributes about the window (size, location, local storage, access to navigation)
- All global
  - variables (these are properties of the window)
  - functions (these are methods of the window)

The root element of the document: document

The Document **interface** represents the entire HTML or XML document. Conceptually, it is the root of the document tree, and provides the **primary access** to the document's data.

The document object implements the Document interface.

Often used methods to manipulate the DOM:

- Document.getElementById(id)
- Document.getElementsByClassname(classname)
- Document.querySelector(querySelector)
- Document.querySelectorAll(querySelector)

Inheritance

As in the previous lesson you've learnt, we have the concept '**inheritance**'. This inheritance appears as well among the interfaces of the DOM.

Source: https://developer.mozilla.org/en-US/docs/Web/API/Document

This means the *document* object (that implements the Document interface) can use the the methods and variables that are in those interfaces which are predecessors of the Document interface.
(I.e. in the *EventTarget* interface, there's a method called *dispatchEvent*, so we can use this on the *document* object: `document.dispatchEvent(...)`)

Check resolution

Exercises One: Alert the user, what kind of video can be displayed in the user's window.

The following attributes are used for checking the resolution of window:

window.innerWidth, window.innerHeight, window.outerWidth, window.outerHeight

Notice the difference between the "inner" and "outer" size properties. The "inner"s give you back the area that are usable for placing elements. The "outer"s give you back the real size of the browser (minus the elements that are out of scope of the window, for instance tab bar at the top of the browser).

Solution

Disable an element

Exercises Two: Get the *maiden name* input from the DOM and set it disabled. Also set its wrapper's opacity to 0.5

```
1.  <div id="app">
2.      <div class="input-wrapper">
3.          <p>Gender</p>
4.          <label for="gender-male-input">male</label>
5.          <input id="gender-male-input" type="radio" name="gender" value="male">
6.          <label for="gender-female-input">female</label>
7.          <input id="gender-female-input" type="radio" name="gender" value="female">
8.      </div>
9.      <div class="input-wrapper">
10.         <label for="name-input">Name</label>
11.         <input id="name-input">
12.     </div>
13.     <div class="input-wrapper">
14.         <label for="maiden-name-input">maiden name</label>
15.         <input id="maiden-name-input">
16.     </div>
17. </div>
```

Solution

Set style for disabled elements

Exercise Three: Set the border color to blue for those elements that are disabled

```
1.  <div id="app">
2.      <div class="input-wrapper">
3.          <p>Gender</p>
4.          <label for="gender-male-input">male</label>
5.          <input id="gender-male-input" type="radio" name="gender" value="male">
6.          <label for="gender-female-input">female</label>
7.          <input id="gender-female-input" type="radio" name="gender" value="female">
8.      </div>
9.      <div class="input-wrapper">
10.         <label for="name-input">Name</label>
11.         <input disabled id="name-input">
12.     </div>
13.     <div class="input-wrapper">
14.         <label for="maiden-name-input">maiden name</label>
15.         <input disabled id="maiden-name-input">
16.     </div>
17.     <button disabled>submit</button>
18. </div>
```

Solution

kebab-case vs camelCase --- naming conventions

The purpose of both of them is to make a long variable name readable.

kebab-case:

- words are splitted with a dash
- it is used in html, css and often for filenames

camelCase:

- start of the new word is marked with capital letter
- it is used in many programming languages (also in javascript)

DOM Events

DOM provides reactive programming in that way, that if something happens (user clicks on button, document load finishes, animation ends, etc-etc…) we can have a reaction on it.

Each event has its type:

- click, mousemove, mousewheel, mouseup, mousedown
- onload
- animationstart, animationend
- focus, focusout
- ...

Two ways to listen to events

- EventTarget.addEventListener(<event name>, <listener>, <options?>)
  - We can attach more listeners to one element
- GlobalEventHandlers.on<event> = function(event) {...}
  - Only one can be attached

Event listeners

In order to be able to react to the DOM events, we need to create event listeners. That means that we **subscribe** for these events through elements. I.e. we have a button that if we click, then we'll increment number.

Exercise Four: Create a click counter application.

Optional version: create the elements with DOM instead of putting them into the HTML file.

Solution

Change background-color on click

Exercises Five: Create a mouse on the website, without placing it in the HTML file. On click it should change the background-color of the document to a random one.

Solution

Disable maiden-name if the user is male

Exercises Six: Disable maiden-name input, and set the wrapper's opacity to 0.5 if the user selects "male" radio button

Solution

Submit the form

Exercises Seven: Submit a form, then show a “Thank you, <name>” message to the user. (Name should come from the form’s input)

Solution

Show a popup when user probably wants to leave

Exercises Eight: Show a popup when the mouse leaves the document (set the modal's display to flex), and remove it if the mouse comes back from the DOM

Solution

Detailed description:

1. listen to mouse leave event, if it happens show the modal, and add a listener to mouseenter
2. on mouseenter remove the modal

Optional: Add <p> and set its content with the proper description (hover in, hover out) so the user know what's going on =)

Removing events

- In case of using EventTarget.addEventListener(<event name>, <handler>, <options?>):
  EventTarget.removeEventListener(<event name>, <handler (same reference)>)

- In case of using GlobalEventHandlers.on<event> = function() {...}
  GlobalEventHandlers.on<event> = null;

Window events

DOMContentLoaded event

VersuS

load event

DEMO TIME

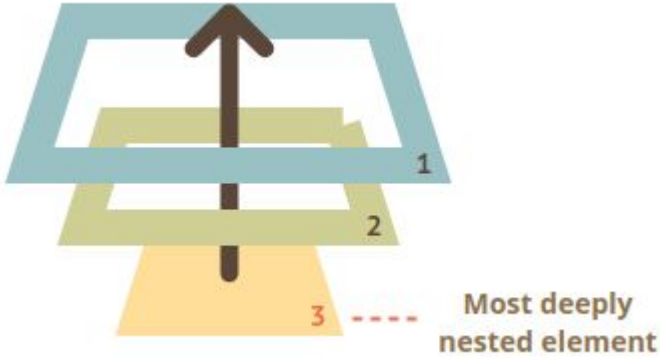
Event bubbling and event cancellation

Bubbling simple means that when an event occurs on an element, then this event will occur on its predecessors as well.

Simple example for bubbling

Exercise Nine: create a menu, that is opened on click, but when clicking on one of its menupoint, then it won't close

Solution



Cancel submitting form, when it’s invalid

Exercise Ten: cancel submit when data is invalid in form

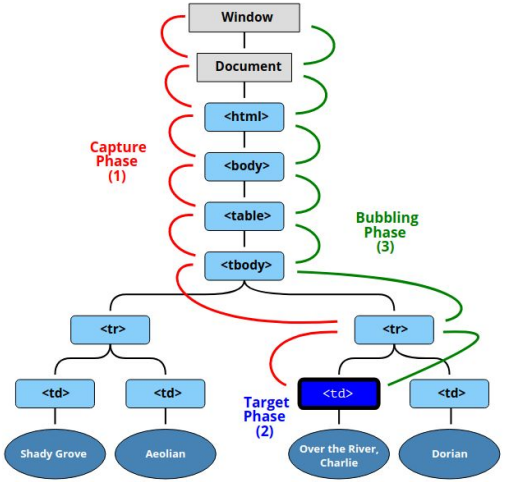
Solution

Event capturing

Capturing is very similar to bubbling, the only difference is the direction:

The chain starts at the elements most distant predecessor and ends at the element.

Simple example



Homework

Exercises Eleven: Show every product in the table (check the table in `index.html`) in the table you need to show the following datas:

- name
- first category
- number of pictures
- price
- quantity

If the user clicks on a row, then in the "product-details" (check `index.html`) show the following datas:

- name
- each categories
- pictures
- prices
- quantity

There are some tips around. (In `index.html` and in the `index.js`.)

You can modify everything in `index.js`, but you are forbidden to change anything in the `index.html`.

I advise not to touch style.css unless you want to add a shiny style for your website 😁

Thank You for your attention!