Browser Object Model

UA Resource Development Unit
2020

AGENDA

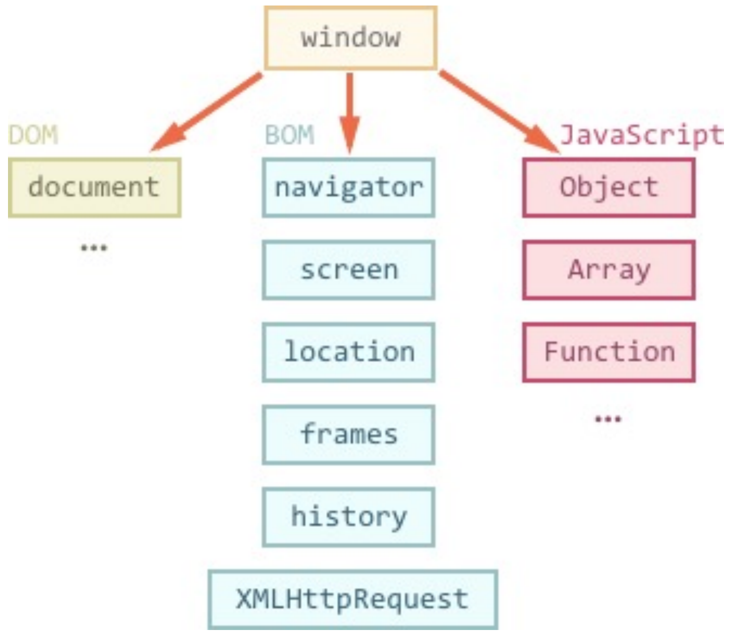
1. BOM
2. Window
3. Navigator & Location
4. Browser. Timing Events. Delayed Execution
5. DOM

INTRODUCTION

WHAT IS BOM?

BOM stands for the *Browser Object Model*.

BOM consist of lots of information about user,
for example their browser’s vendor, coordinates…
and methods for work with browser, like moving through
browser‘s history, changing the URL and so on…



BOM

- The Browser Object Model (BOM) allows JavaScript to "talk to" the browser.
- alert/confirm/prompt
- delayed execution (setTimeout, setInterval)
- Location
- Navigator
- Data storage (cookies/LocalStorage)
- Local/Session storage
- Dimensions

NAVIGATOR & LOCATION

NAVIGATOR

“navigator” is an object that consist of information about user’s operation system and browser, and methods for work with a device.

For example we can get access to device’s microphone, camera, battery, force device to vibrate and so on…

Also you can access the information about user’s geolocation, network connection, browser’s language and so on…

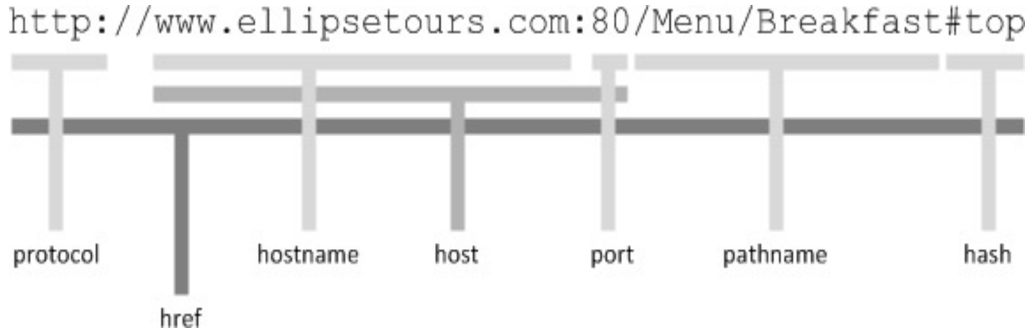
EXAMPLE

```
const positionOptions = {
 enableHighAccuracy: true, timeout: 5000,
 maximumAge: 0
};
const success = ({ coords: { latitude, longitude, accuracy } }) => {
 console.log('Your current position is:');
 console.log(`Latitude : ${latitude}`);
 console.log(`Longitude: ${longitude}`);
 console.log(`Accuracy : ${accuracy / 1000} km.`);
};
const error = ({ code, message }) => console.log(`ERROR(${code}):
 ${message}`);

navigator.geolocation.getCurrentPosition(success, error, positionOptions);
```

LOCATION

"location" is an object that has the information about current URL and allows us to change it and *refresh the page.*

In location object you can find:



LOCATION USAGE EXAMPLES

EXAMPLE #1

```
const query = 'sort=desc;tag=23';
location.search = encodeURIComponent(query);
// example.com/find?sort%3Ddesc%3Btag%3D23
```

EXAMPLE #2

```
location.reload(true);
// login redirect example
location.replace(`http://example.com/login#${location.pathname}`);
```

LOCATION DETAILS

Location object used to change URL in browser.

```
location.href; // get full URL
location.protocol; // get protocol
location.host; // get host, w/o url
location.pathname; // get pathname, w/o hostname
location.port; // get port
location.hash; // get hash (#)
location.search; // get search (?value=...)
// you may set each of this properties, like
location.href = 'http://google.com'; // redirect user browser to
google.com
location.hash = '#11';
// this event fire when hash was changed
window.addEventListener('hashchange', function (e) {
  console.log(e.oldURL + ' -> ' + e.newURL);
});
```

Encoding / Deconding

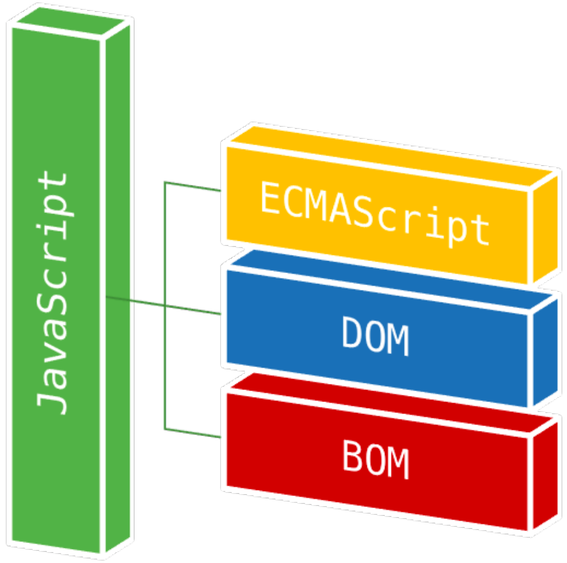
Since some characters (like, /, ?, #, :, =, etc) are reserved for using as URL delimiters, as well as non-ascii characters (non-latin alphabet), you need to encode values in order to get correct result. Browsers use escape-sequences, containing '%xx', where xx is a symbol code. Up to 3 '%xx%yy%zz' may be used to encode some wide characters (like japanese symbols).

```
// Encode only non-latin characters, leave all url-specific symbols in place
encodeURI(value);
// Encode all not-allowed characters, including /, ?, ...
encodeURIComponent(value);
// Decode only non-latin characters
decodeURI(value);
// Decode all delimiter characters
decodeURIComponent(value);
```

[Article Explaining URL and encodings]

Browser and its objects

- Browser
- Browser global objects (window, navigator, location, history)
- Timers
- Document
- DOM



Browser

The Browser Object Model (BOM)
There are no official standards for the **B**rowser **O**bject **M**odel (BOM).
Since modern browsers have implemented (almost) the same methods and properties for JavaScript interactivity, it is often referred to, as methods and properties of the BOM.

- Window Object
- Screen Object
- Location Object
- History Object
- Navigator Object
- Timing

Browser. Window object

The window object is supported by all browsers. It represents the browser's window.

All global JavaScript objects, functions, and variables automatically become members of the window object.

Global variables are properties of the window object.
Global functions are methods of the window object.

Even the document object (of the HTML DOM) is a property of the window object:

```
window.document.getElementById("header");
```

is the same as:

```
document.getElementById(“header");
```

Some other methods:
window.open() - open a new window
window.close() - close the current window
window.moveTo() - move the current window
window.resizeTo() - resize the current window

Browser. Navigator object

The window.navigator object contains information about the visitor's browser.

The window.navigator object can be written without the window prefix.
Some examples:

- navigator.appName
- navigator.appCodeName
- navigator.platform

The cookieEnabled property returns true if cookies are enabled - **navigator.cookieEnabled**
The userAgent property returns the user-agent header sent by the browser to the server - **navigator.userAgent**
The platform property returns the browser platform (operating system) - **navigator.platform**
The language property returns the browser's language - **navigator.language**
The onLine property returns true if the browser is online - **navigator.online**

Navigator (cont.)

```
// Get user-agent string, which can be parsed and detect browser
navigator.userAgent;
// Array of installed plugins. May be used to detect installed plugins, like Adobe Flash
navigator.plugins;
// Current system language
navigator.language;
// If the user allows you to track his position you may use geolocation to retrieve
// user current coordinates (HTML5)
navigator.geolocation.getCurrentPosition(success, error);
var positionChangeWatch = navigator.geolocation.watchPosition(success, error);
navigator.geolocation.clearWatch(positionChangeWatch);
```

Browser. History

The window.history object can be written without the window prefix.
To protect the privacy of the users, there are limitations to how JavaScript can access this object.

Some methods:

- history.back() - same as clicking back in the browser
- history.forward() - same as clicking forward in the browser

Browser. Timing Events

The window object allows execution of code at specified time intervals.

These time intervals are called timing events.

The two key methods to use with JavaScript are:

```
// Executes a function, after waiting a specified number of milliseconds.
setTimeout(function, milliseconds);

// Same as setTimeout(), but repeats the execution of the function continuously.
setInterval(function, milliseconds);
```

The setTimeout() and setInterval() are both methods of the HTML DOM Window object.

Browser. Timing Events

How to Stop the Execution?

The **clearTimeout()** method stops the execution of the function specified in **setTimeout()**.

```
window.clearTimeout(timeoutVariable)
```

The **window.clearTimeout()** method can be written without the window prefix.

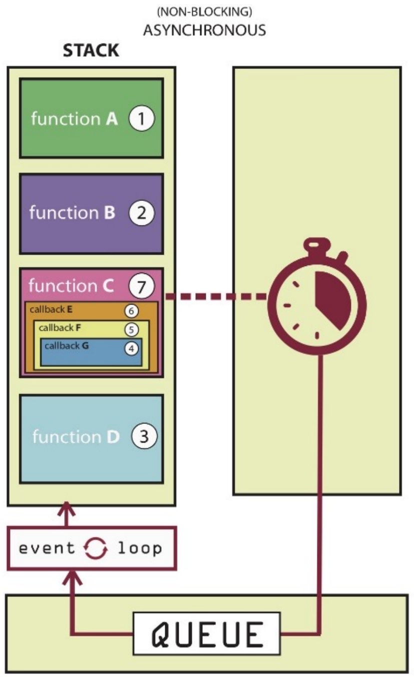
The **clearTimeout()** method uses the variable returned from **setTimeout()**:

```
myVar = setTimeout(function, milliseconds);

clearTimeout(myVar);
```

If the function has not already been executed, you can stop the execution by calling the **clearTimeout()** method

Browser. Timing Events

```
const bar = () => console.log('bar');
const baz = () => console.log('baz');

const foo = () => {
  console.log('foo');
  setTimeout(bar, 0);
  new Promise((resolve, reject) => {
    resolve('should be right after baz, before bar');
  }).then(resolve => console.log(resolve));
  baz();
}

// OUTPUT:
// foo
// baz
// should be right after baz, before bar
// bar
```



Delayed Execution

Time of code execution can be changed with "WindowTimers"

```
var timer = setTimeout(fn, ms); // execute fn after ms milliseconds
typeof timer; // "number" setTimeout returns an identificator
clearTimeout(timer); // clears the timeout
var interval = setInterval(fn, ms); // execute fn after every ms – multiple executions
clearInterval(interval); // stop interval execution
var animFrame = requestAnimationFrame(fn); // execute fn after 1/60 of second.
cancelAnimationFrame(animFrame);
```

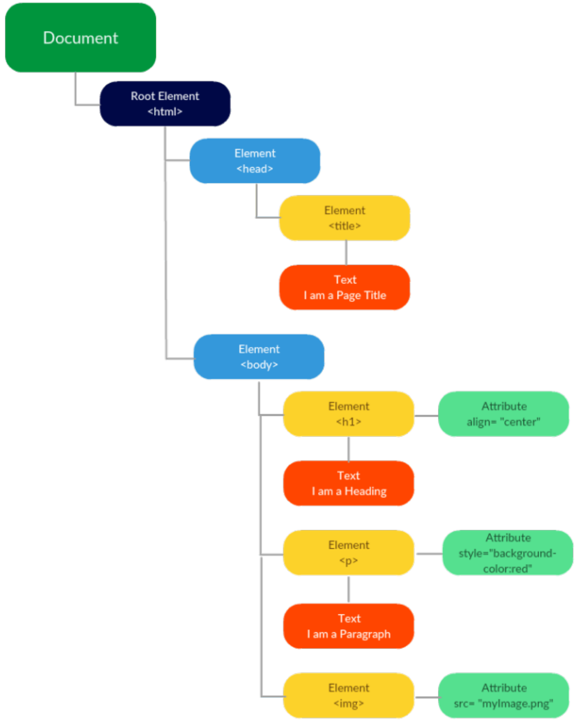
Dimensions

```
// An integer value indicating the width of the window's layout viewport in pixels.
var intFrameWidth = window.innerWidth;

// An integer value indicating the window's layout viewport height in pixels.
var intFrameHeight = window.innerHeight;

// The Screen interface represents a screen, usually the one on
// which the current window is being rendered
window.screen;
```

Browser. Document. DOM

With the HTML DOM, JavaScript can access and change all the elements of an HTML document

When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.
The **HTML DOM** model is constructed as a tree of **Objects**:



With the object model, JavaScript gets all the power it needs to create dynamic HTML:

- JavaScript can change all the HTML elements in the page
- JavaScript can change all the HTML attributes in the page
- JavaScript can change all the CSS styles in the page
- JavaScript can remove existing HTML elements and attributes
- JavaScript can add new HTML elements and attributes
- JavaScript can react to all existing HTML events in the page
- JavaScript can create new HTML events in the page

Browser. Document. DOM

The DOM Programming Interface

The HTML DOM can be accessed with JavaScript (and with other programming languages).

In the DOM, all HTML elements are defined as **objects**.

The programming interface is the properties and methods of each object.

A **property** is a value that you can get or set (like changing the content of an HTML element).

document.getElementById("demo").innerHTML = "Hello World!";

A **method** is an action you can do (like add or deleting an HTML element).

In the example above, getElementById is a **method**, while innerHTML is a **property**.

Browser. Document. DOM

If you want to access any element in an HTML page, you always start with accessing the document object. Below are some examples of how you can use the document object to access and manipulate HTML.

Finding HTML Elements

Method	Description
document.getElementById(*id*)	Find an element by element id
document.getElementsByTagName(*name*)	Find elements by tag name
document.getElementsByClassName(*name*)	Find elements by class name

Browser. Document. DOM

Changing HTML Elements

Property	Description
element.innerHTML = *new html content*	Change the inner HTML of an element
element.attribute = new value	Change the attribute value of an HTML element
element.style.*property = new style*	Change the style of an HTML element
Method	Description
element.setAttribute*(attribute, value)*	Change the attribute value of an HTML element

Browser. Document. DOM

Adding and Deleting Elements

Method	Description
document.createElement(*element*)	Create an HTML element
document.removeChild(*element*)	Remove an HTML element
document.appendChild(*element*)	Add an HTML element
document.replaceChild(*new, old*)	Replace an HTML element
document.write(*text*)	Write into the HTML output stream

Further reading

BOM:

- https://developer.mozilla.org/en-US/docs/Web/API/WindowTimers
- https://developer.mozilla.org/en-US/docs/Web/API/Location
- https://developer.mozilla.org/en-US/docs/Web/API/Navigator
- https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage

Q&A

epam

DRIVEN CANDID CREATIVE ORIGINAL INTELLIGENT EXPERT

UA Frontend Online LAB