JS

JavaScript Fundamentals

By:Ahmed ElMahdy

JS

Events

Form Events

A form event is fired when a form control receive or loses focus or when the user modify a form control value such as by typing text in a text input, select any option in a select box etc
Here're some most important form events and their event handler.

The Focus Event (onfocus)

The focus event occurs when the user gives focus to an element on a web page.
You can handle the focus event with the onfocus event handler.

Note: The value of this keyword inside an event handler refers to the element which has the handler on it (i.e. where the event is currently being delivered).

The Blur Event (onblur)

The blur event occurs when the user takes the focus away from a form element or a window.
You can handle the blur event with the onblur event handler.

Events

Form Events

The Change Event (onchange)

The change event occurs when a user changes the value of a form element.
You can handle the change event with the onchange event handler.

The Submit Event (onsubmit)

The submit event only occurs when the user submits a form on a web page.
You can handle the submit event with the onsubmit event handler.

JS

Events

Document/Window Events

Events are also triggered in situations when the page has loaded or when user resize the browser window, etc. Here're some most important document/window events and their event handler.

The Load Event (onload)

- The load event occurs when a web page has finished loading in the web browser.
- You can handle the load event with the onload event handler.

The Unload Event (onunload)

- The unload event occurs when a user leaves the current web page.
- You can handle the unload event with the onunload event handler.

The Resize Event (onresize)

- The resize event occurs when a user resizes the browser window.
- The resize event also occurs in situations when the browser window is minimized or maximized.
- You can handle the resize event with the onresize event handler.

JS

DOM

JavaScript DOM Styling

Styling DOM Elements in JavaScript

Adding CSS Classes to Elements

- You can also get or set CSS classes to the HTML elements using the className property.
- Since, class is a reserved word in JavaScript, so JavaScript uses the className property to refer the value of the HTML class attribute.
- You can use the classList property to get, set or remove CSS classes easily from an element.
- Property classList is supported in all major browsers except Internet Explorer prior to version 10.

JS

DOM

JavaScript DOM Get Set Attributes

Working with Attributes

The attributes are special words used inside the start tag of an HTML element to control the tag's behavior or provides additional information about the tag.
In the following sections we will learn about these methods in detail.

Getting Element's Attribute Value

- The getAttribute() method is used to get the current value of a attribute on the element.
- If the specified attribute does not exist on the element, it will return null.

Setting Attributes on Elements

- The setAttribute() method is used to set an attribute on the specified element.
- If the attribute already exists on the element, the value is updated; otherwise a new attribute is added with the specified name and value.
- you can use the setAttribute() method to update or change the value of an existing attribute on an HTML element.

Removing Attributes from Elements

- The removeAttribute() method is used to remove an attribute from the specified element

JS

DOM

JavaScript DOM Manipulation

Manipulating DOM Elements in JavaScript

Now that you've learnt how to select and style HTML DOM elements. In this chapter we will learn how to add or remove DOM elements dynamically, get their contents, and so on.

Adding New Elements to DOM

- Use the document.createElement() method to create new element in an HTML document.
- This method creates a new element, but it doesn't add it to the DOM.
- The appendChild() method adds the new element at the end of any other children of a specified parent node.
- Use the insertBefore() method to add the new element at the beginning of any other children.

JS

DOM

JavaScript DOM Manipulation

Manipulating DOM Elements in JavaScript

Getting or Setting HTML Contents to DOM

- Use the innerHTML property to get or set the contents of the HTML elements easily.
- This property sets or gets the HTML markup contained within the element i.e. content between its opening and closing tags.
- The innerHTML property replaces all existing content of an element.
- Use the insertAdjacentHTML() method to insert the HTML into the document without replacing the existing contents of an element.
- The insertAdjacentHTML() accepts two parameters: the position in which to insert and the HTML text to insert.
- The position must be one of the following values: "beforebegin", "afterbegin", "beforeend", and "afterend". This method is supported in all major browsers.

Note: The beforebegin and afterend positions work only if the node is in the DOM tree and has a parent element. Also, when inserting HTML into a page, be careful not to use user input that hasn't been escaped, to prevent XSS attacks.

DOM

JavaScript DOM Manipulation

Manipulating DOM Elements in JavaScript

Removing Existing Elements from DOM

- Use the removeChild() method to remove a child node from the DOM.
- This method also returns the removed node.
- It is also possible to remove the child element without exactly knowing the parent element. Simply find the child element and use the parentNode property to find its parent element.
- The parentNode property returns the parent of the specified node in the DOM tree.

Replacing Existing Elements in DOM

- replace an element in HTML DOM with another using the replaceChild() method.
- This method accepts two parameters: the node to insert and the node to be replaced. It has the syntax like parentNode.replaceChild(newChild, oldChild);.

DOM

JavaScript DOM Navigation

Navigating Between DOM Nodes

There are many occasions where you need to access a child, parent or ancestor element. See the JavaScript DOM nodes chapter to understand the logical relationships between the nodes in a DOM tree. DOM node provides several properties and methods that allow you to navigate or traverse through the tree structure of the DOM and make changes very easily.

In the following section we will learn how to navigate up, down, and sideways in the DOM tree using JavaScript.

Accessing the Child Nodes

- Use the firstChild and lastChild properties of the DOM node to access the first and last direct *child node* of a node, respectively. If the node doesn't have any child element, it returns null .

Note: The nodeName is a read-only property that returns the name of the current node as a string. For example, it returns the tag name for element node, #text for text node, #comment for comment node, #document for document node, and so on.

DOM

JavaScript DOM Navigation

Navigating Between DOM Nodes

Accessing the Parent Nodes

- Use the parentNode property to access the parent of the specified node in the DOM tree.
- The parentNode will always return null for document node, since it doesn't have a parent.

Tip: The topmost DOM tree nodes can be accessed directly as document properties. For example, the <html> element can be accessed with document.documentElement property, whereas the <head> element can be accessed with document.head property, and the <body> element can be accessed with document.body property.

- To get only element nodes you can use the parentElement.

Accessing the Sibling Nodes

- Use the previousSibling and nextSibling properties to access the previous and next node in the DOM tree, respectively.
- Use the previousElementSibling and nextElementSibling to get the previous and next sibling element skipping any whitespace text nodes. All these properties returns null if there is no such sibling.
- The textContent property represents the text content of a node and all of its descendants.

JS

DOM

JavaScript DOM Navigation

Navigating Between DOM Nodes

Accessing the Child Nodes

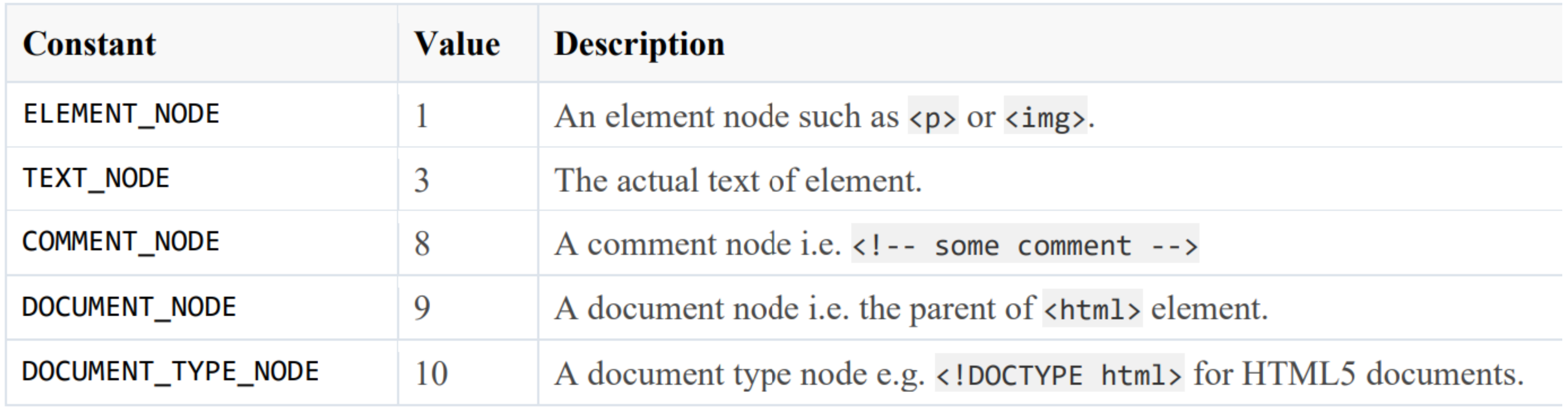
- whitespace such as spaces, tabs, newlines, etc. are valid characters and they form #text nodes and become a part of the DOM tree.
- Since the <div> tag contains a newline before the <h1>tag, so it will create a #text node.
- To avoid the issue with firstChild and lastChild returning #text or #comment nodes, you could alternatively use the firstElementChild and lastElementChild properties to return only the first and last element node, respectively. But, it will not work in IE 9 and earlier.
- Use the childNodes property to access all child nodes of a given element, where the first child node is assigned index 0.
- The childNodes returns all child nodes, including non-element nodes like text and comment nodes. To get a collection of only elements, use children property instead

DOM

JavaScript DOM Navigation

Types of DOM Nodes

The DOM tree is consists of different types of nodes, such as elements, text, comments, etc.
Every node has a nodeType property that you can use to find out what type of node you are dealing with. The following table lists the most important node types:



Constant	Value	Description
`ELEMENT_NODE`	1	An element node such as `<p>` or `<img>`.
`TEXT_NODE`	3	The actual text of element.
`COMMENT_NODE`	8	A comment node i.e. `<!-- some comment -->`
`DOCUMENT_NODE`	9	A document node i.e. the parent of `<html>` element.
`DOCUMENT_TYPE_NODE`	10	A document type node e.g. `<!DOCTYPE html>` for HTML5 documents.

DOM

JavaScript Window

The Window Object

The window object represents a window containing a DOM document. A window can be the main window, a frame set or individual frame, or even a new window created with JavaScript.
If you remember from the preceding chapters we've used the alert() method in our scripts to show popup messages. This is a method of the window object.
In the next few chapters we will see a number of new methods and properties of the window object that enables us to do things such as prompt user for information, confirm user's action, open new windows, etc. which lets you to add more interactivity to your web pages.

Calculating Width and Height of the Window

- The window object provides the innerWidth and innerHeight property to find out the width and height of the browser window viewport (in pixels) including the horizontal and vertical scrollbar, if rendered.
- To find out the width and height of the window excluding the scrollbars you can use the clientWidth and clientHeight property of any DOM element (like a div)

Note: The document.documentElement object represents the root element of the document, which is the <html> element, whereas the document.body object represents the <body> element. Both are supported in all major browsers.

DOM

JavaScript Window Screen

The Screen Object

The window.screen object contains information about the user's screen such as resolution (i.e. width and height of the screen), color depth, pixel depth, etc.

Since window object is at the top of the scope chain, so properties of the window.screen object can be accessed without specifying the window. prefix, for example window.screen.width can be written as screen.width.

The following section will show you how to get information of the user's display using the screen object property of the window object.

Getting Width and Height of the Screen

Use the screen.width and screen.height property obtains the width and height of the user's screen in pixels.

DOM

JavaScript Window Screen

The Screen Object

Getting Available Width and Height of the Screen

- The screen.availWidth and screen.availHeight property can be used to get the width and height available to the browser for its use on user's screen, in pixels.
- The screen's available width and height is equal to screen's actual width and height minus width and height of interface features like the taskbar in Windows.

Getting Screen Pixel Depth

- Get the pixel depth of the screen using the screen.pixelDepth property.
- Pixel depth is the number of bits used per pixel by the system display hardware.

JS

DOM

JavaScript Window Screen

The Screen Object

Getting Screen Color Depth

- use the screen.colorDepth property to get the color depth of the user's screen.
- Color depth is the number of bits used to represent the color of a single pixel.
- Color depth indicates how many colors a device screen is capable to produce. For example, screen with color depth of 8 can produce 256 colors (28).
- Currently, most devices has screen with color depth of 24 or 32. In simple words more bits produce more color variations, like 24 bits can produce 2^24 = 16,777,216 color variations (true colors), whereas 32 bits can produce 2^32 = 4,294,967,296 color variations (deep colors).

Tip:As of now virtually every computer and phone display uses 24-bit color depth. 24 bits almost always uses 8 bits of each of R, G, B. Whereas in case of 32-bit color depth, 24 bits are used for the color, and the remaining 8 bits are used for transparency.

DOM

JavaScript Window Location

The Location Object

The location property of a window (i.e. window.location) is a reference to a Location object; it represents the current URL of the document being displayed in that window.
Since window object is at the top of the scope chain, so properties of the window.location object can be accessed without window.
prefix, for example window.location.href can be written as location.href.

The following section will show you how to get the URL of page as well as hostname, protocol, etc. using the location object property of the window object.

Getting the Current Page URL

Use the window.location.href property to get the entire URL of the current page.

JS

DOM

JavaScript Window Location

The Location Object

Getting Different Part of a URL

Use other properties of the location object such as protocol, hostname, port, pathname, search, etc. to obtain different part of the URL.

Note: When you visit a website, you're always connecting to a specific port (e.g. http://localhost:3000). However, most browsers will simply not display the default port numbers, for example, 80 for HTTP and 443 for HTTPS.

Loading New Documents

- Use the assign() method of the location object i.e. window.location.assign() to load another resource from a URL provided as parameter.
- Use the replace() method to load new document which is almost the same as assign(). The difference is that it doesn't create an entry in the browser's history, meaning the user won't be able to use the back button to navigate to it.
- Use the window.location.href property to load new document in the window. It produce the same effect as using assign() method.

DOM

JavaScript Window Location

The Location Object

Reloading the Page Dynamically

The reload() method can be used to reload the current page dynamically. You can optionally specify a Boolean parameter true or false.

If the parameter is true, the method will force the browser to reload the page from the server. If it is false or not specified, the browser may reload the page from its cache.

Note: The result of calling reload() method is different from clicking browser's Reload/Refresh button. The reload() method clears form control values that otherwise might be retained after clicking the Reload/Refresh button in some browsers

DOM

JavaScript Window History

The History Object

The history property of the Window object refers to the History object. It contains the browser session history, a list of all the pages visited in the current frame or window.
Since Window is a global object and it is at the top of the scope chain, so properties of the Window object i.e. window.history can be accessed without window. prefix, for example window.history.length can be written as history.length.

The following section will show you how to get the information of user's browsing history. However, for security reasons scripts are not allowed to access the stored URLs.

JS

DOM

JavaScript Window History

The History Object

Getting the Number of Pages Visited

- Use The window.history.length property to get the number of pages in the session history of the browser for the current window.
- It also includes the currently loaded page. You can use this property to find out how many pages a user has visited during the current browser session.

Going Back to the Previous Page

- You can use the back() method of the History object i.e. history.back() to go back to the previous page in session history. It is same as clicking the browser's back button.
- If your browser back button is active then clicking this Go Back link takes you one step back.

JS

DOM

JavaScript Window History

The History Object

Going Forward to the Next Page

- Use the forward() method of the History object i.e. history.forward() to go forward to the next page in session history. It is same as clicking the browser's forward button.
- If your browser forward button is active then clicking this Go Forward link takes you one step forward.

Going to a Specific Page

- Use the go() method of the History object i.e. history.go() to load specific page from the session history.
- This method takes an integer as a parameter.
- A negative integer moves backward in the history, and a positive integer moves forward in the history.

Tip:If you attempt to access the page that does not exist in the window's history then the methods back(), forward() and go() will simply do nothing.

DOM

JavaScript Window Navigator

The Navigator Object

The navigator property of a window (i.e. window.navigator) is a reference to a Navigator object; it is a read-only property which contains information about the user's browser.
Since Window is a global object and it is at the top of the scope chain, so properties of the Window object such as window.navigator can be accessed without window. prefix, for example window.navigator.language can be written as navigator.language.

The following section will show you how to get various information about user's browser.

Detect Whether the Browser is Online or Offline

- Use the navigator.onLine property to detect whether the browser (or, application) is online or offline.
- This property returns a Boolean value true meaning online, or false meaning offline.
- Browser fires online and offline events when a connection is establish or lost.
- You can attach handler functions to these events in order to customize your application for online and offline scenarios.

JS

DOM

JavaScript Window Navigator

The Navigator Object

Check Whether Cookies Are Enabled or Not

- Use the navigator.cookieEnabled to check whether cookies are enabled in the user's browser or not.
- This property returns a Boolean value true if cookies are enabled, or false if it isn’t.

Tip: You should use the navigator.cookieEnabled property to determine whether the cookies are enabled or not before creating or using cookies in your JavaScript code.

Detecting the Browser Language

Use the navigator.language property to detect the language of the browser UI.

This property returns a string representing the language, e.g. "en", "en-US", etc

JS

DOM

JavaScript Window Navigator

The Navigator Object

Getting Browser Name and Version Information

- appName — Returns the name of the browser. It always returns "Netscape", in any browser.
- appVersion — Returns the version number and other information about the browser.
- appCodeName — Returns the code name of the browser. It returns "Mozilla", for all browser.
- userAgent — Returns the user agent string for the current browser. This property typically contains all the information in both appName and appVersion.
- platform — Returns the platform on which browser is running (e.g. "Win32", "WebTV OS", etc.)

As you can see from the above descriptions, the value returned by these properties are misleading and unreliable, so don't use them to determine the user's browser type and version.

Check Whether the Browser is Java Enabled or Not

- Use the method javaEnabled() to check whether the current browser is Java-enabled or not.
- This method simply indicates whether the preference that controls Java is on or off, it does not reveal whether the browser offers Java support or Java is installed on the user's system or not.

JS

DOM

JavaScript Dialog Boxes

Creating Dialog Boxes

In JavaScript you can create dialog boxes or popups to interact with the user. You can either use them to notify a user or to receive some kind of user input before proceeding.

You can create three different types of dialog boxes alert, confirm, and prompt boxes.

The appearance of these dialog boxes is determined by the operating system and/or browser settings, they cannot be modified with the CSS. Also, dialog boxes are modal windows; when a dialog box is displayed the code execution stops, and resumes only after it has been dismissed.

In the following section we will discuss each of these dialog boxes in detail.

Creating Alert Dialog Boxes

- An alert dialog box is the most simple dialog box. It enables you to display a short message to the user.
- It also includes OK button, and the user has to click this OK button to continue.
- create alert dialog boxes with the alert() method.

DOM

JavaScript Dialog Boxes

Creating Dialog Boxes

Creating Confirm Dialog Boxes

- A confirm dialog box allows user to confirm or cancel an action.
- A confirm dialog looks similar to an alert dialog but it includes a Cancel button along with the OK button.
- Create confirm dialog boxes with the confirm() method.
- This method simply returns a Boolean value (true or false) depending on whether the user clicks OK or Cancel button. That's why its result is often assigned to a variable when it is used.

Creating Prompt Dialog Box

- The prompt dialog box is used to prompt the user to enter information.
- A prompt dialog box includes a text input field, an OK and a Cancel button.
- Create prompt dialog boxes with the prompt() method. This method returns the text entered in the input field when the user clicks the OK button, and null if user clicks the Cancel button.
- If the user clicks OK button without entering any text, an empty string is returned. For this reason, its result is

Tip: To display line breaks inside the dialog boxes, use newline character or line feed (\n); a backslash followed by the character n

DOM

JavaScript Timers

Working with Timers

A timer is a function that enables us to execute a function at a particular time. Using timers you can delay the execution of code so that it does not get done at the exact moment an event is triggered or the page is loaded. For example, you can use timers to change the advertisement banners on your website at regular intervals, or display a real-time clock, etc.

There are two timer functions in JavaScript: setTimeout() and setInterval().

The following section will show you how to create timers to delay code execution as well as how to perform one or more actions repeatedly using these functions in JavaScript.

Executing Code After a Delay

The setTimeout() function is used to execute a function or specified piece of code just once after a certain period of time. Its basic syntax is setTimeout(function, milliseconds).

This function accepts two parameters: a *function*, which is the function to execute, and an optional *delay* parameter, which is the number of milliseconds representing the amount of time to wait before executing the function

Note: If the *delay* parameter is omitted or not specified, a value of 0 is used, that means the specified function is executed "immediately", or, as soon as possible

JS

DOM

JavaScript Timers

Working with Timers

Executing Code at Regular Intervals

- Use the setInterval() function to execute a function or specified piece of code repeatedly at fixed time intervals. Its basic syntax is setInterval(*function, milliseconds*).
- This function also accepts two parameters: a function, which is the function to execute, and interval, which is the number of milliseconds representing the amount of time to wait before executing the function (1 second = 1000 milliseconds).

Stopping Code Execution or Cancelling a Timer

- Both setTimeout() and setInterval() method return an unique ID (a positive integer value, called timer identifier) which identifies the timer created by the these methods.
- This ID can be used to disable or clear the timer and stop the execution of code beforehand. Clearing a timer can be done using two functions: clearTimeout() and clearInterval().
- The setTimeout() function takes a single parameter, an ID, and clear a setTimeout() timer associated with that ID.
- The clearInterval() method is used to clear or disable a setInterval() timer.