HTML BOM in JS

The Browser Object Model (BOM) is a collection of browser-specific objects provided by JavaScript to control and interact with the web browser itself, outside the content of the web page. This allows developers to handle window-related operations, navigate between pages, retrieve device information, and manage the browser’s history.

**window Object**

* **Description**: The window object is the top-level object in the BOM and represents the browser window or tab. It’s the global object, meaning all global variables and functions are part of window.
* **Properties and Methods**:
  + **window.open()**: Opens a new browser window or tab. It accepts parameters for the URL, window name, and window features (size, toolbar visibility, etc.).

window.open("https://example.com", "\_blank", "width=500, height=500");

* + **window.close()**: Closes the current window, typically only for windows opened via window.open().
  + **Dialog Boxes**:
  + **window.alert()**: Shows a simple alert dialog.
  + **window.prompt()**: Displays a dialog box that allows user input.
  + **window.confirm()**: Shows a confirmation dialog with "OK" and "Cancel" buttons.
  + **Timers**:
  + **window.setTimeout()**: Executes code after a specified delay (in milliseconds).

window.setTimeout(() => alert("Hello!"), 2000);

* + **window.setInterval()**: Repeats execution of code at specified intervals

const intervalId = window.setInterval(() => console.log("Interval"), 1000);

**navigator Object**

* **Description**: The navigator object provides information about the browser itself, such as the browser’s name, version, language, and whether the user is online.
* **Properties**:
  + **navigator.userAgent**: A string containing details about the browser, operating system, and device.

console.log(navigator.userAgent); // "Mozilla/5.0 (Windows NT 10.0; Win64; x64)..."

* **navigator.language**: Indicates the default language setting of the user’s browser.
* **navigator.platform**: Gives the operating system information, like "Win32", "MacIntel", etc.
* **navigator.onLine**: Boolean value indicating if the browser is connected to the internet.

**Geolocation**:

* **navigator.geolocation**: Provides the user’s current location with methods:
  + **navigator.geolocation.getCurrentPosition()**: Gets the user’s latitude and longitude, provided the user consents.

navigator.geolocation.getCurrentPosition((position) => {

console.log(position.coords.latitude, position.coords.longitude);

});

**navigator.geolocation.watchPosition()**: Tracks the user’s location continuously.

**screen Object**

* **Description: The screen object holds information about the user’s display screen, like its dimensions and color depth, which can be useful for optimizing layouts and handling different screen resolutions.**
* **Properties:**
  + **screen.width and screen.height: Return the screen width and height in pixels.**

**console.log(`Screen Width: ${screen.width}, Height: ${screen.height}`);**

* + **screen.availWidth and screen.availHeight: Width and height of the screen available for the browser, excluding the taskbar or any other system UI elements.**
  + **screen.colorDepth: The bit depth of the screen’s color palette. Common values include 24 and 32.**
  + **screen.orientation: Provides information about the screen orientation (portrait or landscape).**

**4. location Object**

* **Description: The location object provides detailed information about the URL of the current document and methods for manipulating or navigating the URL.**
* **Properties:**
  + **location.href: Contains the entire URL. Assigning a new URL to location.href navigates to that new page.**

**console.log(location.href); // "https://example.com/page"**

* + **location.hostname: The domain name of the web host.**
  + **location.pathname: The path within the URL, representing the specific page.**
  + **location.protocol: Specifies the protocol used (e.g., "http:" or "https:").**
  + **location.port: The port number, if specified.**
* **Methods:**
  + **location.assign(): Loads a new document. The old page is retained in session history.**

**location.assign("https://anotherpage.com");**

* + **location.reload(): Reloads the current page. Accepts a true argument for a forced reload, ignoring cached content.**

**location.reload(true);**

* + **location.replace(): Replaces the current page with a new one without adding the old page to the history, so users cannot go back to it.**

**5. history Object**

* **Description: The history object allows developers to navigate the user’s browser history, which is particularly useful for creating single-page applications where manual history control is required.**
* **Properties:**
  + **history.length: Returns the number of entries in the user’s history list.**
* **Methods:**
  + **history.back(): Navigates to the previous page in history (similar to pressing the browser’s back button).**

**history.back();**

* + **history.forward(): Moves one entry forward in the history stack.**

**history.forward();**

* + **history.go(): Moves to a specific page in the history list. Accepts an integer argument representing the number of steps forward or backward.**

**history.go(-2); // Go back two pages**

* **HTML5 API:**
  + **history.pushState(): Adds a new history entry without reloading the page. Useful in single-page applications to manage URL state.**

**history.pushState({ page: 1 }, "title 1", "?page=1");**

* + **history.replaceState(): Replaces the current history entry without adding a new one.**

**JS Web API’s**

**JavaScript Web APIs are collections of built-in functions and interfaces provided by the browser to enable interactions between JavaScript and the various components of a web application, such as the browser window, the user’s device, network, and more. They make web applications dynamic, interactive, and able to access a wide range of browser and system features.**

**1. Document Object Model (DOM) API**

* **Description: The DOM API represents the structure of an HTML or XML document as a tree of objects (nodes) that JavaScript can manipulate.**
* **Core Methods:**
  + **document.getElementById(id): Returns an element by its id.**
  + **document.querySelector(selector): Selects the first element matching a CSS selector.**
  + **document.createElement(tagName): Creates a new element node.**
* **Event Handling:**
  + **Events like click, mouseover, input, and submit are added to DOM elements using addEventListener().**

**document.getElementById("btn").addEventListener("click", function() {**

**alert("Button Clicked");**

**});**

* **Manipulating Elements:**
  + **Changing element properties, inner content, and styling dynamically.**

**const elem = document.getElementById("text");**

**elem.innerHTML = "Updated content";**

**elem.style.color = "blue";**

**2. Fetch API**

* **Description: The Fetch API is used for making network requests, like retrieving data from a server, and it replaces the older XMLHttpRequest.**
* **Usage:**
  + **Fetching a resource (e.g., JSON, text) from a URL, with methods to handle responses and errors.**

**fetch("https://api.example.com/data")**

**.then(response => response.json())**

**.then(data => console.log(data))**

**.catch(error => console.error("Error:", error));**

* **Features:**
  + **Supports Promises, making it simpler to handle asynchronous requests.**
  + **Can be used for GET, POST, PUT, and DELETE requests by configuring options.**

**3. Geolocation API**

* **Description: The Geolocation API provides access to the user’s current location, useful for applications that need to provide location-specific services.**
* **Core Methods:**
  + **navigator.geolocation.getCurrentPosition(successCallback, errorCallback): Gets the user’s current position.**

**navigator.geolocation.getCurrentPosition(**

**(position) => {**

**console.log(`Latitude: ${position.coords.latitude}`);**

**console.log(`Longitude: ${position.coords.longitude}`);**

**},**

**(error) => console.error("Error:", error)**

**);**

* + **watchPosition(): Tracks the user’s location continuously, triggering a callback every time the position changes.**
* **Usage Notes:**
  + **Requires user permission.**
  + **Typically used in mapping and navigation applications.**

**4. Canvas API**

* **Description: The Canvas API allows drawing graphics on a web page via the <canvas> element. It’s widely used for animations, games, and image editing.**
* **Core Elements:**
  + **<canvas> tag: Defines a canvas area.**
  + **getContext("2d"): Returns a drawing context for 2D graphics.**
* **Basic Drawing:**
  + **Drawing shapes, lines, and text:**

**const canvas = document.getElementById("myCanvas");**

**const ctx = canvas.getContext("2d");**

**ctx.fillStyle = "blue";**

**ctx.fillRect(10, 10, 100, 50); // Draws a blue rectangle**

* **Advanced Usage:**
  + **Supports image manipulation, pixel-level editing, and transformations.**

**5. Web Storage API**

* **Description: The Web Storage API allows the storage of data in the browser, providing two main types: localStorage and sessionStorage.**
* **localStorage:**
  + **Stores data with no expiration date; persists even after the browser is closed.**

**localStorage.setItem("name", "John");**

**console.log(localStorage.getItem("name")); // Output: John**

* **sessionStorage:**
  + **Stores data for the duration of the page session, cleared when the page is closed.**
* **Key Features:**
  + **Only stores strings, but JSON can be used to store objects.**
  + **Useful for persisting small amounts of user data without cookies.**

**6. Notification API**

* **Description: Allows web applications to display desktop notifications to the user.**
* **Usage:**
  + **Requesting Permission: Notifications require user permission.**

**Notification.requestPermission().then((result) => {**

**if (result === "granted") {**

**new Notification("Hello, user!");**

**}**

**});**

* **Creating Notifications:**
  + **With custom text, icons, and actions.**

**new Notification("New Message", {**

**body: "You have a new message!",**

**icon: "message\_icon.png"**

**});**

* **Applications:**
  + **Useful for notifying users of updates, messages, reminders, and alerts.**

**7. Device Orientation and Motion API**

* **Description: This API provides access to the physical orientation and movement of the device, which is often used in games, VR, and AR applications.**
* **Events:**
  + **deviceorientation: Provides orientation of the device.**

**window.addEventListener("deviceorientation", (event) => {**

**console.log("Alpha:", event.alpha);**

**console.log("Beta:", event.beta);**

**console.log("Gamma:", event.gamma);**

**});**

* + **devicemotion: Provides acceleration and rotation rate of the device.**
* **Usage:**
  + **Commonly used for controlling elements based on device orientation.**

**8. IndexedDB API**

* **Description: IndexedDB is a low-level API for storing large amounts of structured data, including files and blobs. It’s an asynchronous API that operates via a transaction-based database.**
* **Usage:**
  + **Creating a database and storing/retrieving data.**

**const request = indexedDB.open("myDatabase", 1);**

**request.onsuccess = (event) => {**

**const db = event.target.result;**

**console.log("Database opened", db);**

**};**

**request.onupgradeneeded = (event) => {**

**const db = event.target.result;**

**db.createObjectStore("myStore", { keyPath: "id" });**

**};**

* **Applications:**
  + **Ideal for offline applications that require persistent, complex data storage.**

**9. Service Workers and Cache API**

* **Service Workers:**
  + **Background scripts that run separately from the main browser thread. They enable features like offline caching, push notifications, and background sync.**
* **Cache API:**
  + **Works with service workers to store assets locally for offline usage.**

**caches.open("my-cache").then((cache) => {**

**cache.add("/offline-page.html");**

**});**

* **Applications:**
  + **Essential for Progressive Web Apps (PWAs) to enable offline capabilities and improve loading times.**

**10. WebRTC API**

* **Description: WebRTC (Web Real-Time Communication) is used to establish peer-to-peer connections, enabling audio, video, and data transfer directly between browsers.**
* **Usage:**
  + **Setting up a video call, sharing files, and creating real-time chat applications.**

**const pc = new RTCPeerConnection();**

**pc.createOffer().then((offer) => pc.setLocalDescription(offer));**

* **Applications:**
  + **Video conferencing, screen sharing, and P2P file transfer applications.**

**11. Push API**

* **Description: The Push API enables web applications to receive notifications from a server, even when the application is not active in the browser.**
* **Usage:**
  + **Requires Service Worker registration and push notification permission.**

**navigator.serviceWorker.register("/sw.js").then((registration) => {**

**registration.pushManager.subscribe({ userVisibleOnly: true });**

**});**

* **Applications:**
  + **Commonly used for sending real-time notifications in messaging, news, or social media applications.**

**12. Bluetooth API**

* **Description: The Bluetooth API allows web applications to connect to nearby Bluetooth devices, such as heart-rate monitors, smartwatches, and other IoT devices.**
* **Usage:**
  + **Interacting with Bluetooth devices by requesting permission to access them.**

**navigator.bluetooth.requestDevice({ filters: [{ services: ["heart\_rate"] }] })**

**.then(device => console.log("Device:", device))**

**.catch(error => console.log("Error:", error));**

* **Applications:**
  + **Fitness applications, medical monitoring, IoT device control.**