BOM. Events

softserve

Agenda

- BOM
  - Window object
  - History object
  - Screen object
  - Location object
  - Navigator object
  - Geolocation object
- Events
  - Events handlers
  - Events object

softserve

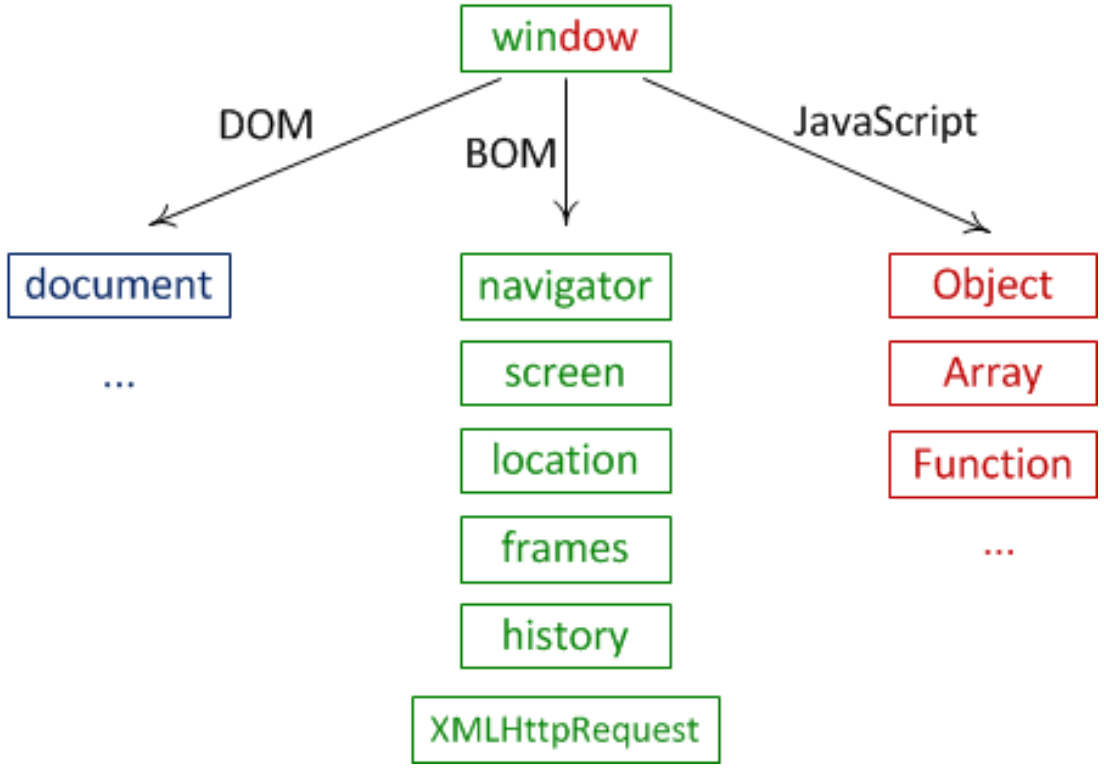
BOM

softserve

Browser Object Model

Of great importance in JavaScript is working with a Web browser and the objects that it provides. For example, using browser objects allows you to manipulate the HTML elements that are on the page or interact with the user.

From the point of view of JavaScript, a browser is a collection of objects organized into a hierarchical structure. It is this hierarchy of objects that is available for use in scripts that is called the **Browser Object Model**, or simply the **BOM**.



softserve

BOM. Window object

A window object is a Web browser window that hosts web pages. **window** is a **global object**, so it is **not necessary to use its name** when accessing its properties and methods:

```
alert("New message");
window.alert("New message");
```

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

Global variables are properties of the window object.

```
var company = "SoftServe";
alert(window.company); // SoftServe
```

Global functions are methods of the window object.

```
function showMessage() {
    alert("Some message");
}
window.showMessage(); // Some message
```

softserve

BOM. Window object. Working with browser windows

The window object provides a number of properties and methods for managing browser windows:

- window.**innerHeight** - the inner height of the browser window (in pixels)
- window.**innerWidth** - the inner width of the browser window (in pixels)
- window.**open(**url, name, params**)** – opens a specific resource in a new browser window
- **close()** – closes the browser window
- **moveTo(**x, y**)** – moves the window to a new position
- **resizeTo(**width, height**)** – resizes the window

Example:

```
let openTab = window.open("https://www.w3schools.com", "w3schools",
"width=200,height=200,left=700,top=200");
```

BOM. 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. All user visit information is stored in a special history stack.

The history object provides a number of methods for **navigating through visit history**:

- **history.back()** – moving to the last viewed page
- **history.forward()** – moves to the next viewed page
- **history.go()** – allows you to scroll back and forth through the history to a certain number of pages (negative number - back, positive - forward)

```
history.go(-1); // back to 1 page, analogue history.back()
history.go(2); // 2 pages forward
```

With the **length** property, you can get the number of pages visited by a user:

```
console.log(history.length);
```

softserve

BOM. 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.

Window.screen object properties:

- **availHeight** - the height of the screen available for the purpose of displaying windows (excluding toolbars, menubars and so on)
- **availWidth** - the width of the screen available for the purpose of displaying windows (excluding toolbars, menubars and so on)
- **colorDepth** - to color depth of the screen
- **height** - to height if the screen
- **width** - to width of the screen

```
alert(screen.availHeight);
alert(screen.height);
alert(screen.width);
alert(screen.colorDepth);
```

softserve

BOM. Location object

The **location object** contains information about the **location of the current Web page**. **Properties**:

- **location.href** – the full line of the request to the resource
- **location.pathname** – the path to the resource
- **location.origin** – general request scheme (root domain)
- **location.protocol** – protocol
- **location.port** – the port used by the resource
- **location.host** – host
- **location.hostname** – hostname
- **location.hash** – if the query string contains a lattice **symbol (#)**, then this property returns that part of the string that goes after that character
- **location.search** – if the query string contains a **question mark (?)**, for example, then this property returns that part of the string that goes after the question mark

BOM. Location object

The **location object** also provides a number of **methods** that you can use **to control the query**:

- **location.assign(**url**)** – loading resource
- **location.replace(**url**)** – replaces the current web page with another resource. Unlike assign(), the replace() method does not save the previous Web page in the history transition history stack
- **location.reload()** – reloads the current web page therefore, it will be impossible to return to it
- **location.resolveURL**(url) - to resolve the current URL to the specified one.

```
<script>
    location.reload();        // reload from the browser's cache
    location.reload(true);    //reload from the server
    location.reload(false);   // reload from the browser's cache. default value is false
</script>
```

BOM. Navigator object

The JavaScript **window** object property of **navigator** is used refer to the **Navigator object**, which contains all **information related to browser**, it vendor, version, plugins etc.

Javascript Navigator Object Properties:

- **navigator.appCodeName -** the code name of the browser
- **navigator.appName** - the name of the browser.
- **navigator.appVersion** - specifies the version of the browser.
- **navigator.platform** - the operation system on which the browser runs.
- **navigator.userAgent -** the HTTP user-agent header sent from the browser to the server
- **navigator.onLine** property to detect whether the browser (or, application) is online or offline

```
console.log(navigator.platform); // win32
```

softserve

BOM. Navigator object. Geolocation object

The navigator object stores the **geolocation property**. The **geolocation property** returns a **Geolocation object** that can be used to **locate the user's position**. The Geolocation object allows the user to provide their location to web applications. For privacy reasons, the user is asked for **permission** to report location information.

Geolocation Object Properties

Property	Description
coordinates	Returns the position and altitude of the device on Earth
position	Returns the position of the concerned device at a given time
positionError	Returns the reason of an error occurring when using the geolocating device
positionOptions	Describes an object containing option properties to pass as a parameter of Geolocation.getCurrentPosition() and Geolocation.watchPosition()

Geolocation Object Methods

Method	Description
clearWatch()	Unregister location/error monitoring handlers previously installed using Geolocation.watchPosition()
getCurrentPosition()	Returns the current position of the device
watchPosition()	Returns a watch ID value that then can be used to unregister the handler by passing it to the Geolocation.clearWatch() method

softserve

BOM. Geolocation object. getCurrentPosition()

The **getCurrentPosition()** method is used to return the user's position.

The getCurrentPosition() method returns an object on success

- **coords.latitude** - the latitude as a decimal number (always returned)
- **coords.longitude** - the longitude as a decimal number (always returned)
- **coords.accuracy** - the accuracy of position (always returned)
- **coords.altitude** - the altitude in meters above the mean sea level (returned if available)
- **coords.speed** - the speed in meters per second (returned if available)

```
if(navigator.geolocation){
  navigator.geolocation.getCurrentPosition(function(pos){
    alert( 'latitude: '+pos.coords.latitude+'  longitude: '+pos.coords.longitude+' altitude:
    '+pos.coords.altitude+' accuracy: '+pos.coords.accuracy+' speed: '+pos.coords.speed );
  },function(e){alert(e);},
  {enableHighAccuracy: true, timeout: 4000, maximumAge: 0 } );
      } else alert(" Your browser does not support geolocation.");
```

softserve

Events

softserve

Events

JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page.

Javascript Events makes the webpages interactive and responsive. These events are ***asynchronous*** (i.e it can occur anytime).

JavaScript has the following **event types**:

- **Mouse events** (cursor movement, mouse click, etc.)
- **Keyboard events** (pressing or releasing a keyboard key)
- **Element life cycle events** (for example, web page load event)
- **Events of form elements** (clicking a button on a form, selecting an element in a drop-down list, etc.)
- Events that occur when **DOM elements change**
- Events that occur when **touched on touch screens**
- Events that occur when **errors occur**

softserve

Events. Common events

Here is a list of some **common HTML DOM events**:

- **change** – an HTML element has been changed
- **click** - the user clicks an HTML element
- **mouseover** - the user moves the mouse over an HTML element
- **mouseout** - the user moves the mouse away from an HTML element
- **keypress** - the event occurs when the user presses a key
- **keydown** - the user pushes a keyboard key
- **load** - the browser has finished loading the page
- **submit** – the visitor submitted the <form>
- **focus** – the visitor focuses on the element, such as clicking on <input>
- **transitionend** – when the CSS animation is complete

HTML DOM Events

softserve

Events handlers

An event can be assigned a **handler**, that is, a function that will fire as soon as the event occurs.

Thanks to handlers, JavaScript code can respond to user actions.

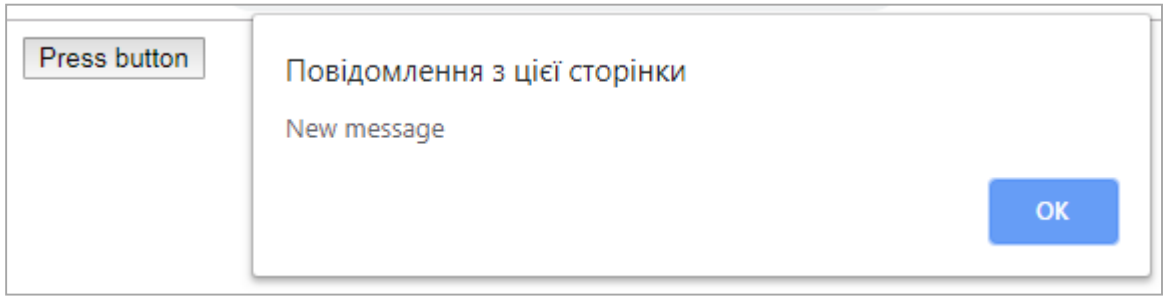
JavaScript has three methods for assigning **event handlers**:

1) Using the **HTML attribute**
2) Using the **DOM element property**
3) Using the **EventListener methods**

Events handlers. HTML attribute

The most easiest and the simplest way to handle events is by using **Inline Events**.
To create an Inline Event you need a ***simple event handler*** and ***event attribute***, which is called "**on<event>**"

```
<button onclick="alert('New message')">Press button</button>
```



An onclick attribute is created in a regular button, which sets the event handler for clicking the button. That is, in order **to process any event, we need to define a handler for it**. The handler is JavaScript code. When you click on the button, the code specified in the "onclick“ attribute will be executed.

Please note that inside the "alert" single quotes are used, since the attribute itself is in double

softserve

Events handlers. HTML attribute

However, to keep the JavaScript seperate from HTML, you can set up the event handler in an external JavaScript file or within the <script> and </script> tags. You can assign an element attribute to a function:

```
<button type="button" onclick="sendMessage()" id="myBtn">Send message</button>
<script>
    function sendMessage() {
        alert('New message from function!');
    }
</script>
```

Note: Since HTML attributes are case-insensitive so onclick may also be written as onClick, OnClick or ONCLICK. But its *value* is case-sensitive.

softserve

Events handlers. HTML attribute. Limitations

Although this approach works great, it has many disadvantages:

- Html code mixes with JavaScript code, making it harder to develop, debug, and maintain an application
- Event handlers can only be set for items already created on the web page. Dynamically created elements in this case lose the ability to handle events
- Only one handler can be attached to an element for a single event.
- Cannot delete handler without changing code

Events handlers. DOM element property

Most of these drawbacks can be avoided by assigning a handler using the **property of the DOM element "on<event>"**.

```
<button type="button" id="myBtn">Send message</button>
<script>
    function sendMessage() {
        alert('New message from DOM element property!');
    }
    document.getElementById("myBtn").onclick = sendMessage;
</script>
```

Note: the function name without brackets.

Events handlers. DOM element property

In general, the job handler through an attribute is similar to the handler job because of the element property. Because in the first case, the browser reads the HTML markup, creates a new function from the attribute contents and writes to the property.

A handler is always stored in the property of a DOM object, and an attribute is just one way to initialize it.

These two code examples work the same way:

HTML only :

```
<button type="button" onclick="alert('Click!')" id="myBtn">Press me!</button>
```

HTML + JS:

```
<button type="button" id="myBtn">Press me!</button>
<script>
   document.getElementById("myBtn").onclick = function() {
      alert("Click!");
   }
</script>
```

softserve

Events handlers. DOM element property

Since the DOM element can have only one property with the name onclick, you cannot assign more than one handler (both in the attribute and in the property).

```
<button type="button" onclick="alert('First!')" id="myBtn">Press me!</button>
<script>
   document.getElementById("myBtn").onclick = function() {
      alert("Second!");
   }
</script>
```

Assignment via JavaScript will **overwrite** the handler from the attribute.

You can remove the handler by setting:

```
elem.onclick = null
```

Events. Access to an element through this

Inside the event handler, this refers to the current element, that is, the one on which, as they say, the handler "hangs" (ie, is assigned).

In the code below, the button displays its contents using this.innerHTML:

```
<input type="button" onclick="alert(this.value)" value="Send data">
```

The names of the DOM properties are case sensitive. Use elem.onclick, not elem.ONCLICK. The ONCLICK property will not work.

softserve

Events handlers. addEventListener

The main drawback of the above methods for assigning a handler is the inability to hang several handlers on one event.

This problem can be solved using special methods for adding the **addEventListener** event and removing the **removeEventListener** event.

Syntax:

elem.**addEventListener**(event, handler[, phase]);

```
event - event name e.g. "click"
handler - handler function reference
phase - additional object with properties
```

elem.**removeEventListener**(event, handler[, phase]);

```
event - event name e.g. "click"
handler - handler function reference
phase - additional object with properties
```

softserve

Events handlers. addEventListener

The **addEventListener** method allows you to **add multiple handlers to a single event of one element** :

```
<button id="myBtn">Send mail</button>
<script>
  function mailSender1() {
    alert('Mail_1 was sent');
  };
  function mailSender2() {
    alert('Mail_2 was sent');
  }
  myBtn.onclick = function() { alert("Prepare to sending"); }
  myBtn.addEventListener("click", mailSender1); // Mail_1 was sent
  myBtn.addEventListener("click", mailSender2); // Mail_2 was sent
</script>
```

Thus, you can simultaneously assign handlers both through the DOM property and through addEventListener. However, in order to avoid confusion, it is recommended to choose one method.

softserve

Events handlers. Event object

When an event occurs, the browser creates an **event object**, writes the details to it, and passes it as an argument to the handler function.

Event Object Properties:

- **bubbles**: returns true if the event is up. For example, if an event occurred on a nested element, then it can be processed on the parent element.
- **cancelable**: returns true if you can cancel the standard event handling
- **currentTarget**: defines the element to which the event handler is attached
- **defaultPrevented**: returns true if the preventDefault() method was called on the Event object
- **eventPhase**: defines the stage of event processing
- **target**: indicates the element on which the event was fired
- **timeStamp**: stores the time the event occurred
- **type**: indicates the name of the event

Events handlers. Event object

Event object usage example:

```
<input type="button" value="Button" id="elem">
<script>
    function eventHandler(event){
        console.log("Event type: " + event.type);
        console.log(event.target);
    }
    let btn = document.getElementById("elem");
     btn.addEventListener("click", eventHandler);
</script>
```

And in this case, the target property is an element, so we can manipulate it like any other nodes and DOM elements. For example, change the background color:

```
function eventHandler(event){
    event.target.style.backgroundColor = "orange";
}
```

softserve

Useful links

https://www.w3schools.com/js/js_window.asp

https://learn.javascript.ru/events

softserve

THANKS

softserve