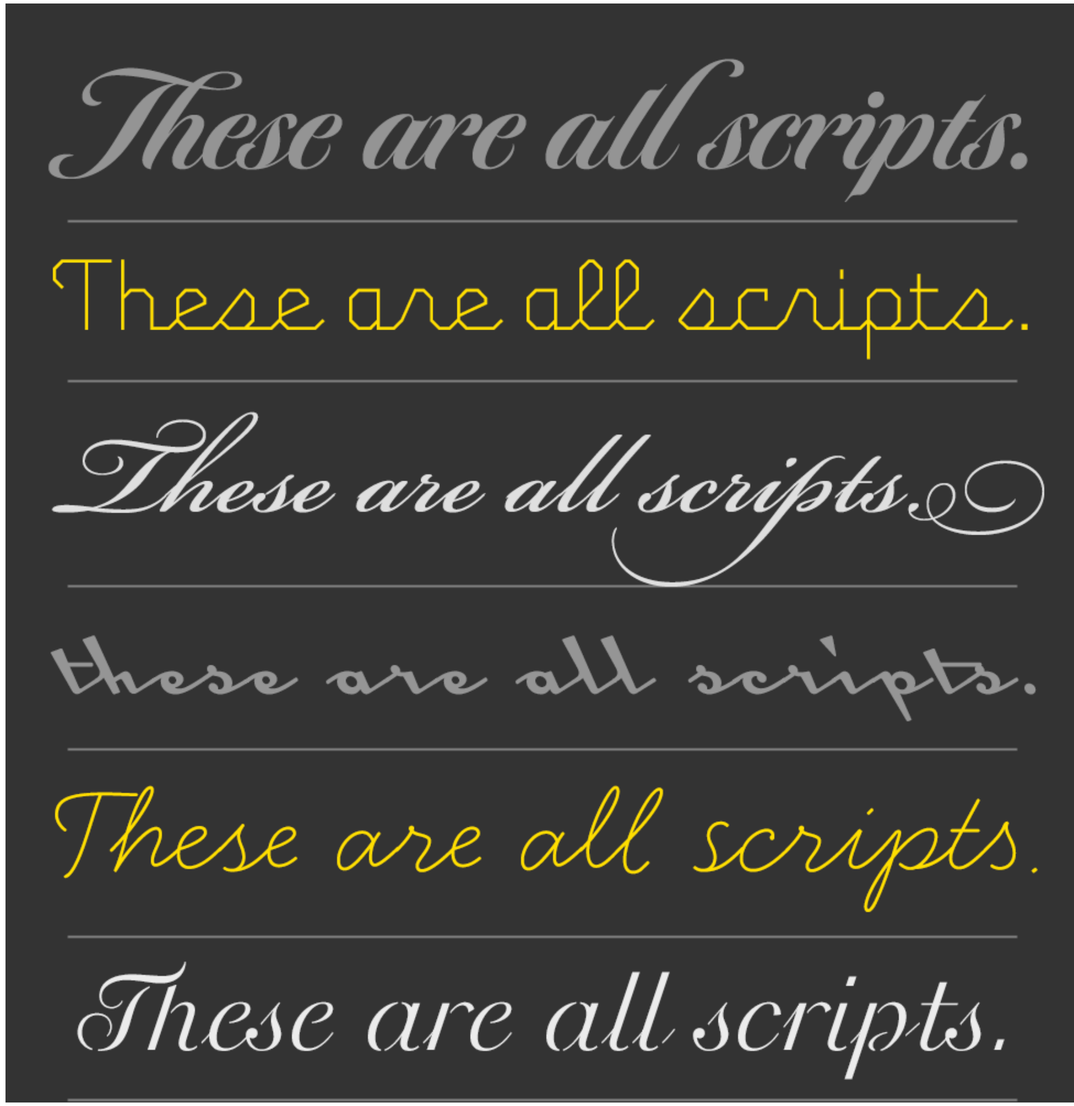
Intro
<AppWeb/>
2022

JavaScript in the Browser

Navigating and Changing the Web Document Structure

Luigi De Russis



Politecnico di Torino

Goal

- Revise the browser’s execution environment and event loop
- Browser object model
- Document object model
- DOM Manipulation
- DOM Styling
- Event Handling

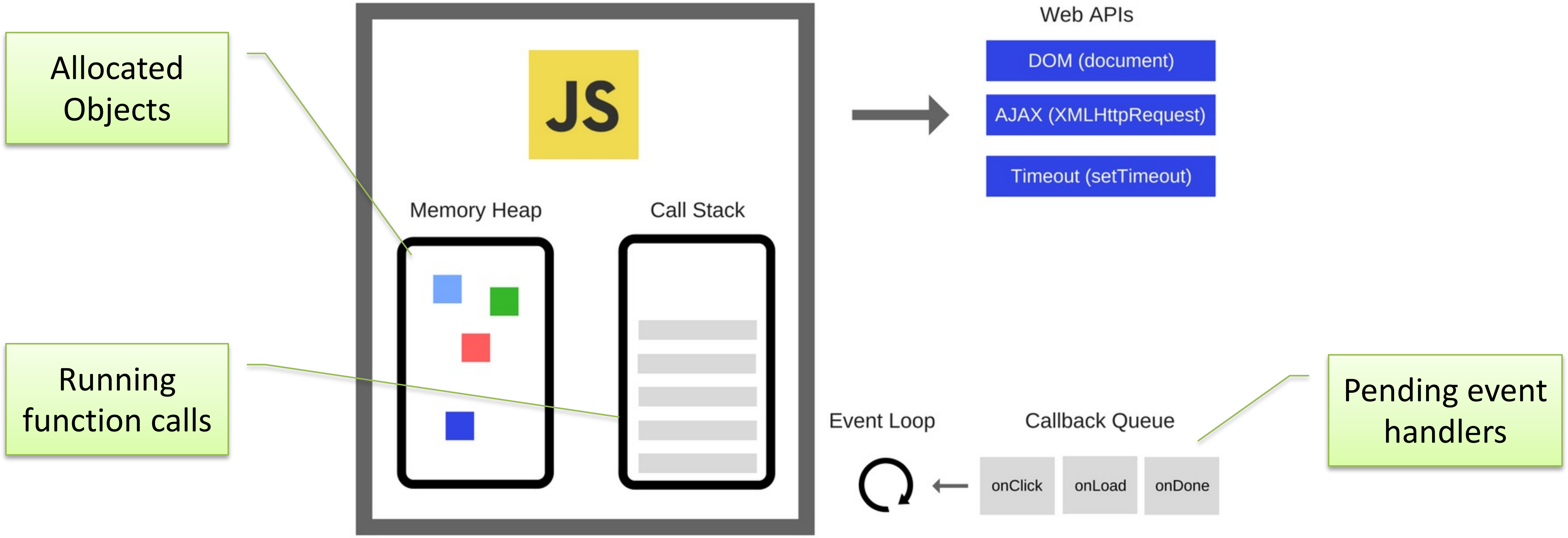
Recap: Where Does The Code Run?

- Loaded and run in the browser *sandbox*
- Attached to a *global context:* the `window` object
- May access only a limited set of APIs
  - JS Standard Library
  - Browser objects (BOM)
  - Document objects (DOM)
- Multiple `<script>`s are independent
  - They all access the same global scope
  - To have structured collaboration, *modules* are needed

Recap: Events and Event Loop

- Most phases of processing and interaction with a web document will generate Asynchronous *Events* (100's of different types)
- Generated events may be handled by:
  - Pre-defined behaviors (by the browser)
  - User-defined *event handlers* (in your JS)
  - Or just ignored, if no event handler is defined
- But JavaScript is single-threaded
  - Event handling is *synchronous* and is based on an *event loop*
  - Event handlers are queued on a *Message Queue*
  - The Message Queue is polled when the main thread is idle

Recap: Execution Environment



Recap: Event Loop

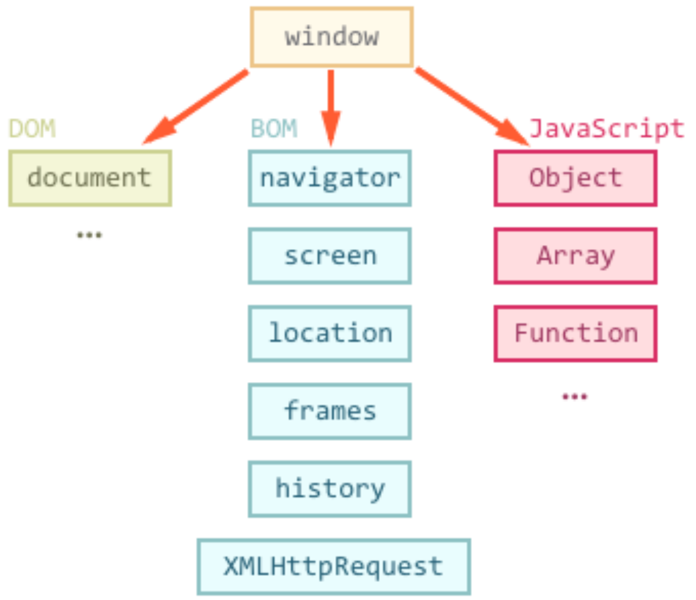
- During code execution you may
  - Call functions → the function call is pushed to the call stack
  - Schedule events → the call to the event handler is put in the Message Queue
    - Events may be scheduled also by external events (user actions, I/O, network, timers, …)
- At any step, the JS interpreter:
  - If the call stack is not empty, pop the top of the call stack and executes it
  - If the call stack is empty, pick the head of the Message Queue and executes it
- A function call / event handler is **never** interrupted
  - Avoid blocking code!

https://developer.mozilla.org/en-US/docs/Web/JavaScript/EventLoop

BROWSER OBJECT MODEL

Browser Main Objects

- `window` represents the window that contains the DOM document
  - allows to interact with the browser via the BOM: browser object model (not standardized)
  - global object, contains all JS global variables
    - can be omitted when writing JS code in the page
- `document`
  - represents the DOM tree loaded in a window
  - accessible via a `window` property: `window.document`



https://medium.com/@fknussel/dom-bom-revisited-cf6124e2a816

The *global* Scope

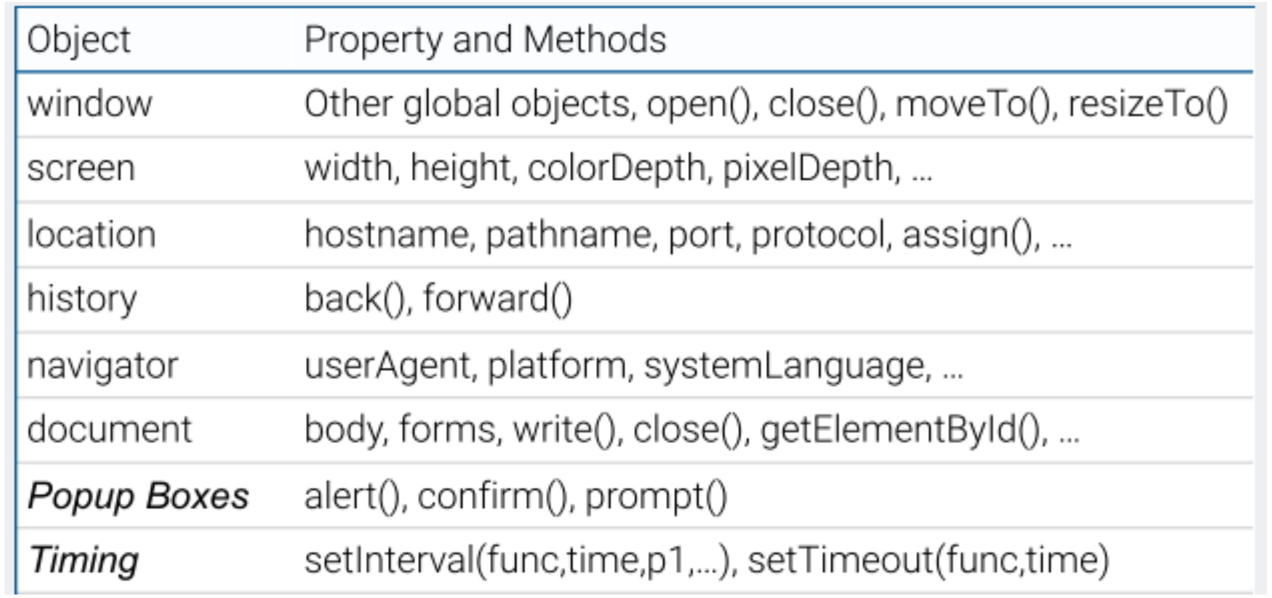
- `window` represents the global scope of the JS program
- Attributes may be added to `window`
  - Explicitly: `window.myprogram="nice";`
  - Implicitly: `let myprogram="nice";`
  - Beware name clashes with other scripts or predefined properties
- `window` attributes are automatically visible
  - `window.document` and `document` are equivalent

Browser Object Model

- `window` properties
  - `console`: browser debug console (visible via developer tools)
  - `document`: the document object
  - `history`: allows access to History API (history of URLs)
  - `location`: allows access to Location API (current URL, protocol, etc.). Read/write property, i.e., can be set to load a new page
  - `localStorage` and `sessionStorage`: allows access to the two objects via the Web Storage API, to store (small) info locally in the browser

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

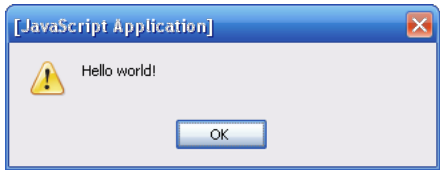
Frequently Seen Properties and Methods



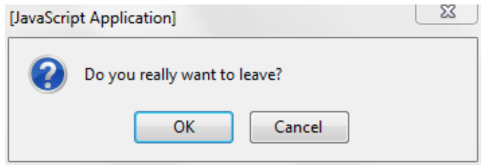
Object	Property and Methods
window	Other global objects, open(), close(), moveTo(), resizeTo()
screen	width, height, colorDepth, pixelDepth, ...
location	hostname, pathname, port, protocol, assign(), ...
history	back(), forward()
navigator	userAgent, platform, systemLanguage, ...
document	body, forms, write(), close(), getElementById(), ...
Popup Boxes	alert(), confirm(), prompt()
Timing	setInterval(func,time,p1,...), setTimeout(func,time)

Window Object: Main Methods

- Methods
  - `alert(),prompt(),confirm()`: handle browser-native dialog boxes
    Never use them – just for debug
  - `setInterval(),clearInterval(),setTimeout(), setImmediate()`: allows to execute code via the event loop of the browser
  - `addEventListener(),removeEventListener()`: allows to execute code when specific events happen to the document


[JavaScript Application]

Hello world!


[JavaScript Application]

Do you really want to leave?

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

Window Object: Main Methods

- `open()`: allows to open a **new** browser window
- `moveTo()`, `resizeTo()`, `minimize()`, `focus()`: allows to manipulate the browser window
- …

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

Storing Data

Cookies

- String/value pairs, Semicolon separated
- Cookies are transferred on to every request

```
document.cookie = "name=Jane Doe; nr=1234567;
expires="+date.toGMTString()
```

Web Storage (Local and Session Storage)

- Store data as key/value pairs on user side
- Browser defines storage quota

Local Storage (`window.localStorage`)

- Store data in users browser
- Comparison to Cookies: more secure, larger data capacity, not transferred
- No expiration date

Session Storage (`window.sessionStorage`)

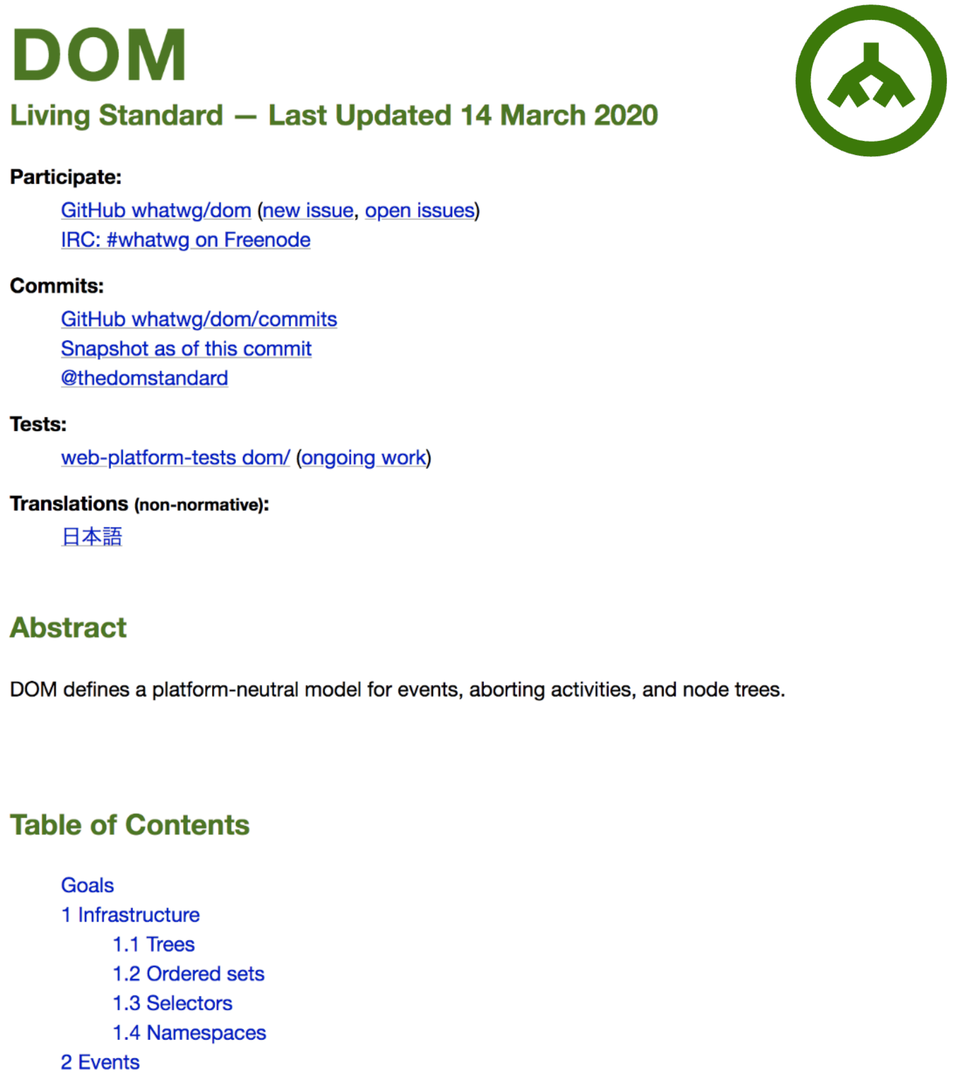
- Store data in session
- Data is destroyed when tab/browser is closed

```
let storage = permanent ? window.localStorage :
                          window.sessionStorage;
if(!storage["name"]) {
   storage["name"] = "A simple storage"
}
alert("Your name is " + storage["name"]);
```

DOCUMENT OBJECT MODEL

DOM Living Standard

- Standardized by WHATWG in the DOM Living Standard Specification
- https://dom.spec.whatwg.org



DOM

Living Standard — Last Updated 14 March 2020

Participate:

GitHub whatwg/dom (new issue, open issues)
IRC: #whatwg on Freenode

Commits:

GitHub whatwg/dom/commits
Snapshot as of this commit
@thedomstandard

Tests:

web-platform-tests dom/ (ongoing work)

Translations (non-normative):

日本語

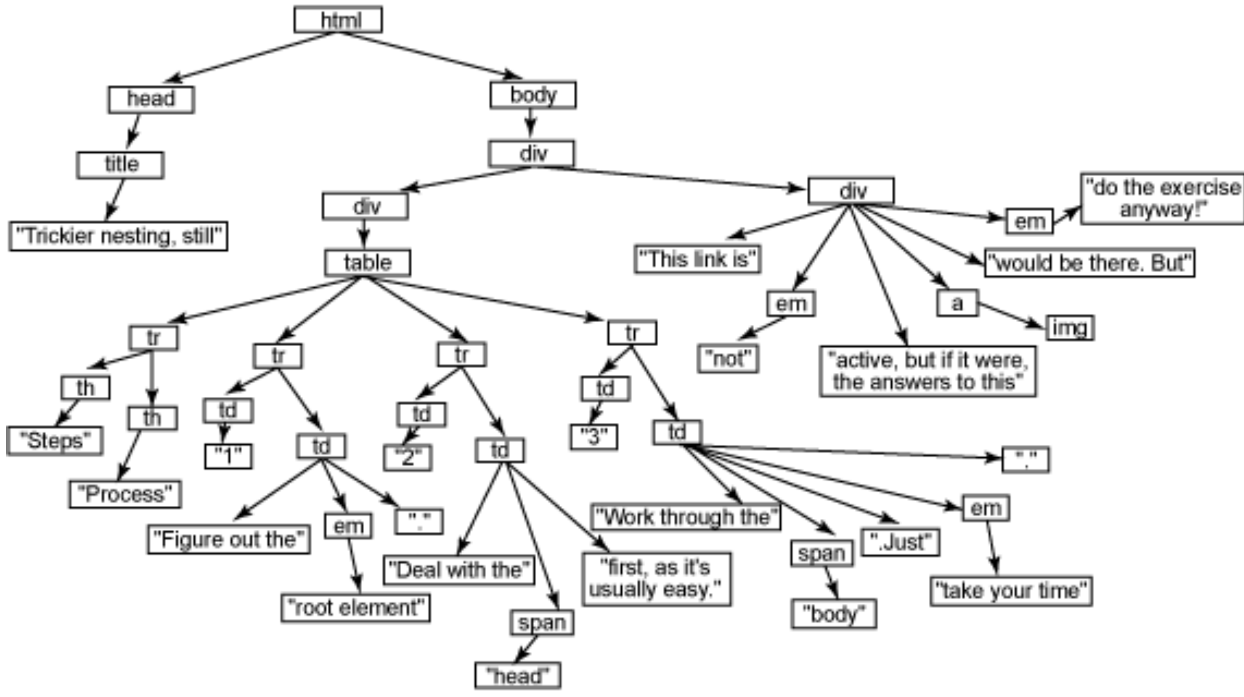
Abstract

DOM defines a platform-neutral model for events, aborting activities, and node trees.

Table of Contents

DOM

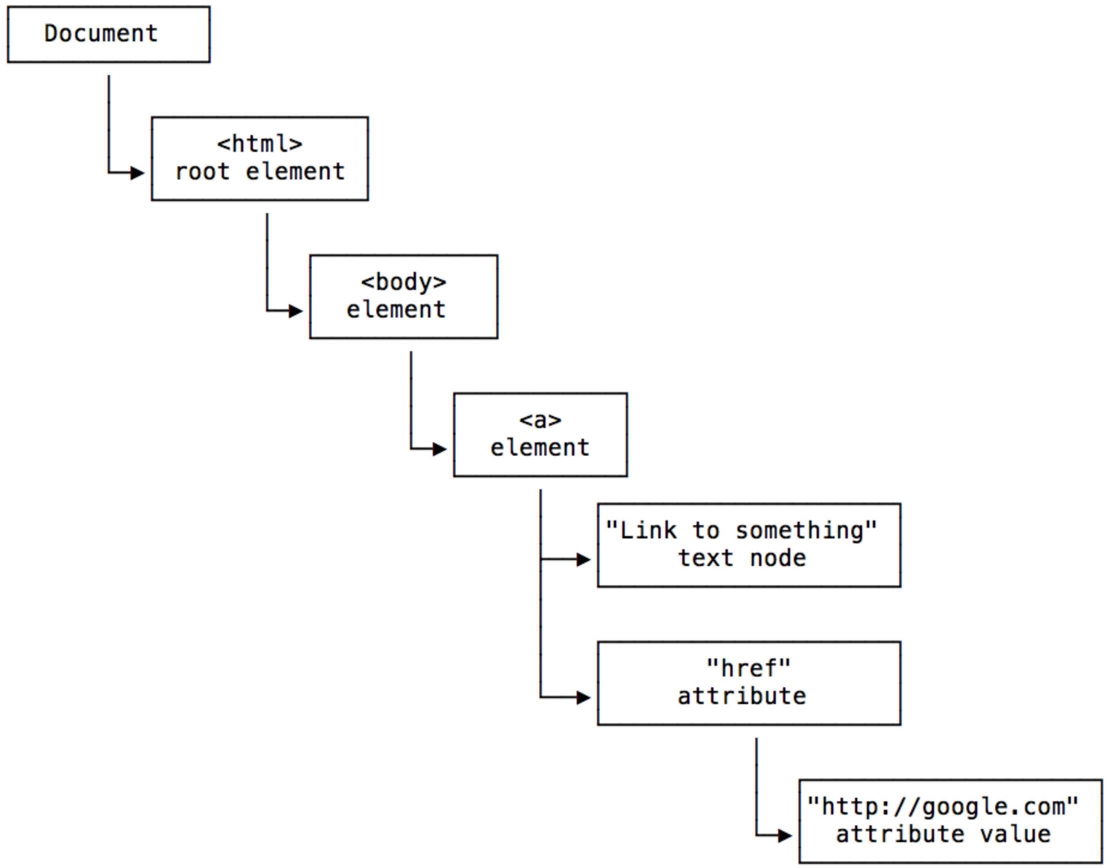
- Browser’s internal representation of a web page
  - Obtained through parsing HTML
- Browsers expose an API that you can use to interact with the DOM
  - Access the page metadata and headers
  - Inspect the page structure
  - Edit any node in the page
  - Change any node attribute
  - Create/delete nodes in the page
  - Edit the CSS styling and classes
  - Attach or remove *event listeners*



https://flaviocopes.com/dom/

Types Of Nodes

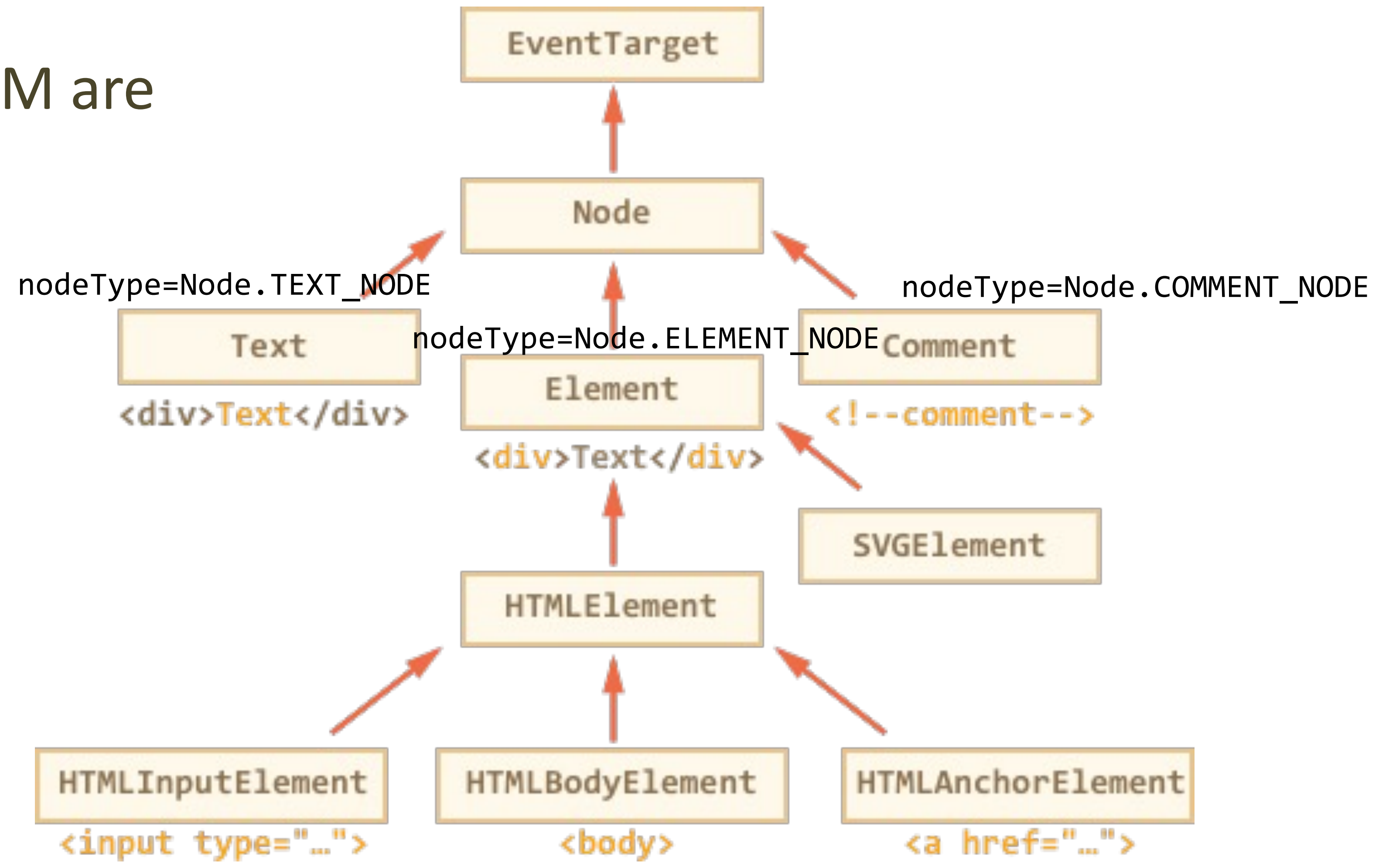
- **Document**: the document Node, the root of the tree
- **Element**: an HTML tag
- **Attr**: an attribute of a tag
- **Text**: the text content of an Element or Attr Node
- **Comment**: an HTML comment
- **DocumentType**: the Doctype declaration



https://flaviocopes.com/dom/

DOM Classes Hierarchy

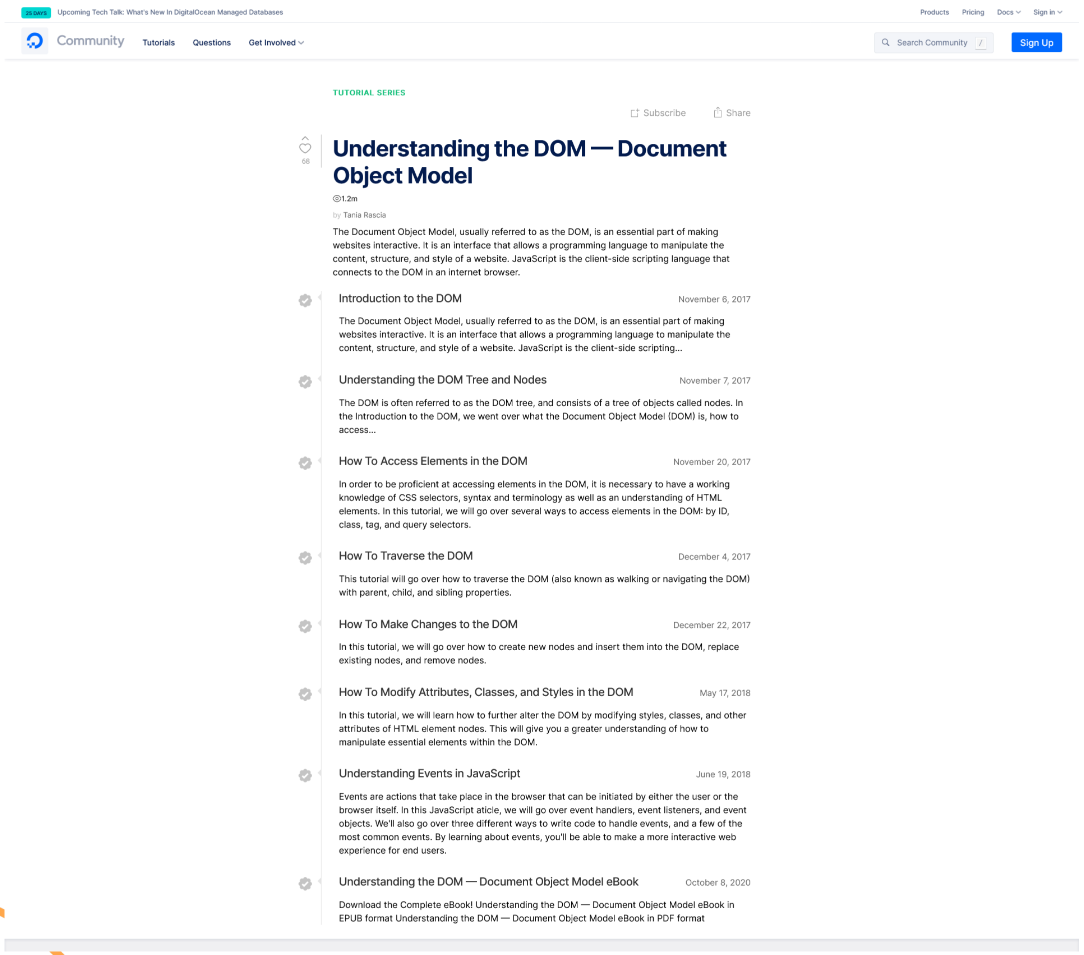
- Objects in DOM are instances of a hierarchy



Node Lists

- The DOM API may manipulate sets/lists of nodes
- The `NodeList` type is an array-like sequence of Nodes
- May be accessed as a JS Array
  - `.length` property
  - `.item(i)` , equivalent to `list[i]`
  - `.entries()`, `.keys()`, `.values()` iterators
  - `.forEach()` functional iteration
  - `for`…`of` classical iteration

Suggested Reading



Upcoming Tech Talk: What's New In DigitalOcean Managed Databases

TUTORIAL SERIES

Understanding the DOM — Document Object Model

1.2m

by Tania Rascia

The Document Object Model, usually referred to as the DOM, is an essential part of making websites interactive. It is an interface that allows a programming language to manipulate the content, structure, and style of a website. JavaScript is the client-side scripting language that connects to the DOM in an internet browser.

Introduction to the DOM — November 6, 2017

The Document Object Model, usually referred to as the DOM, is an essential part of making websites interactive. It is an interface that allows a programming language to manipulate the content, structure, and style of a website. JavaScript is the client-side scripting...

Understanding the DOM Tree and Nodes — November 7, 2017

The DOM is often referred to as the DOM tree, and consists of a tree of objects called nodes. In the Introduction to the DOM, we went over what the Document Object Model (DOM) is, how to access...

How To Access Elements in the DOM — November 20, 2017

In order to be proficient at accessing elements in the DOM, it is necessary to have a working knowledge of CSS selectors, syntax and terminology as well as an understanding of HTML elements. In this tutorial, we will go over several ways to access elements in the DOM: by ID, class, tag, and query selectors.

How To Traverse the DOM — December 4, 2017

This tutorial will go over how to traverse the DOM (also known as walking or navigating the DOM) with parent, child, and sibling properties.

How To Make Changes to the DOM — December 22, 2017

In this tutorial, we will go over how to create new nodes and insert them into the DOM, replace existing nodes, and remove nodes.

How To Modify Attributes, Classes, and Styles in the DOM — May 17, 2018

In this tutorial, we will learn how to further alter the DOM by modifying styles, classes, and other attributes of HTML element nodes. This will give you a greater understanding of how to manipulate essential elements within the DOM.

Understanding Events in JavaScript — June 19, 2018

Events are actions that take place in the browser that can be initiated by either the user or the browser itself. In this JavaScript aticle, we will go over event handlers, event listeners, and event objects. We'll also go over three different ways to write code to handle events, and a few of the most common events. By learning about events, you'll be able to make a more interactive web experience for end users.

Understanding the DOM — Document Object Model eBook — October 8, 2020

Download the Complete eBook! Understanding the DOM — Document Object Model eBook in EPUB format Understanding the DOM — Document Object Model eBook in PDF format

- https://www.digitalocean.com/community/tutorial_series/understanding-the-dom-document-object-model
- Complete and detailed tutorial
- Here, we *focus* on the core concepts and techniques

DOM MANIPULATION

Finding DOM Elements

- `document.getElementById(value)`
  - Returns the Node with the attribute id=value
- `document.getElementsByTagName(value)`
  - Returns the NodeList of all elements with the specified tag name (e.g., ‘div’)
- `document.getElementsByClassName(value)`
  - Returns the NodeList of all elements with attribute class=value (e.g., ‘col-8’)
- `document.querySelector(css)`
  - Returns the first Node element that matches the CSS selector syntax
- `document.querySelectorAll(css)`
  - Returns the NodeList of all elements that match the CSS selector syntax

https://flaviocopes.com/dom/

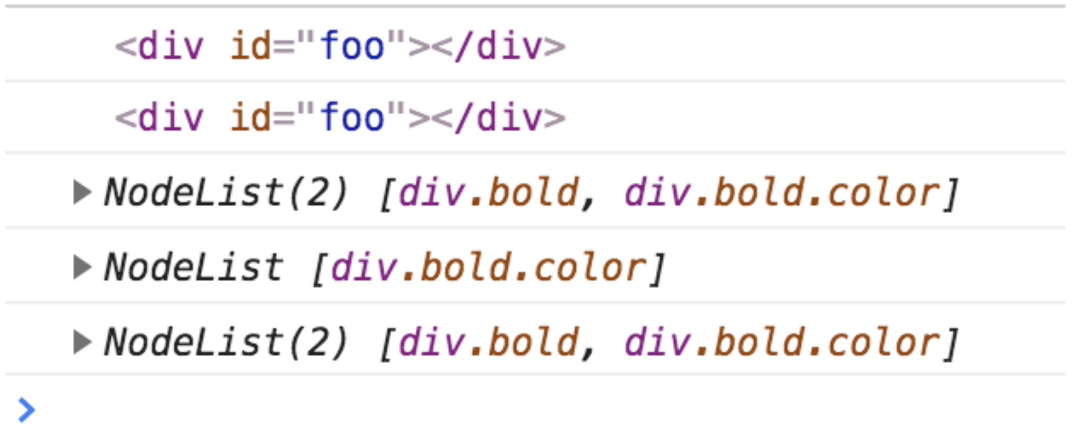
Note

- Node-finding methods also work on any Element node
- In that case, they only search through *descendant* elements
  - May be used to refine the search
- Example:

```
let main = document.getElementById('main');
let articletext = main.getElementsByTagName('p');
```

Accessing DOM Elements

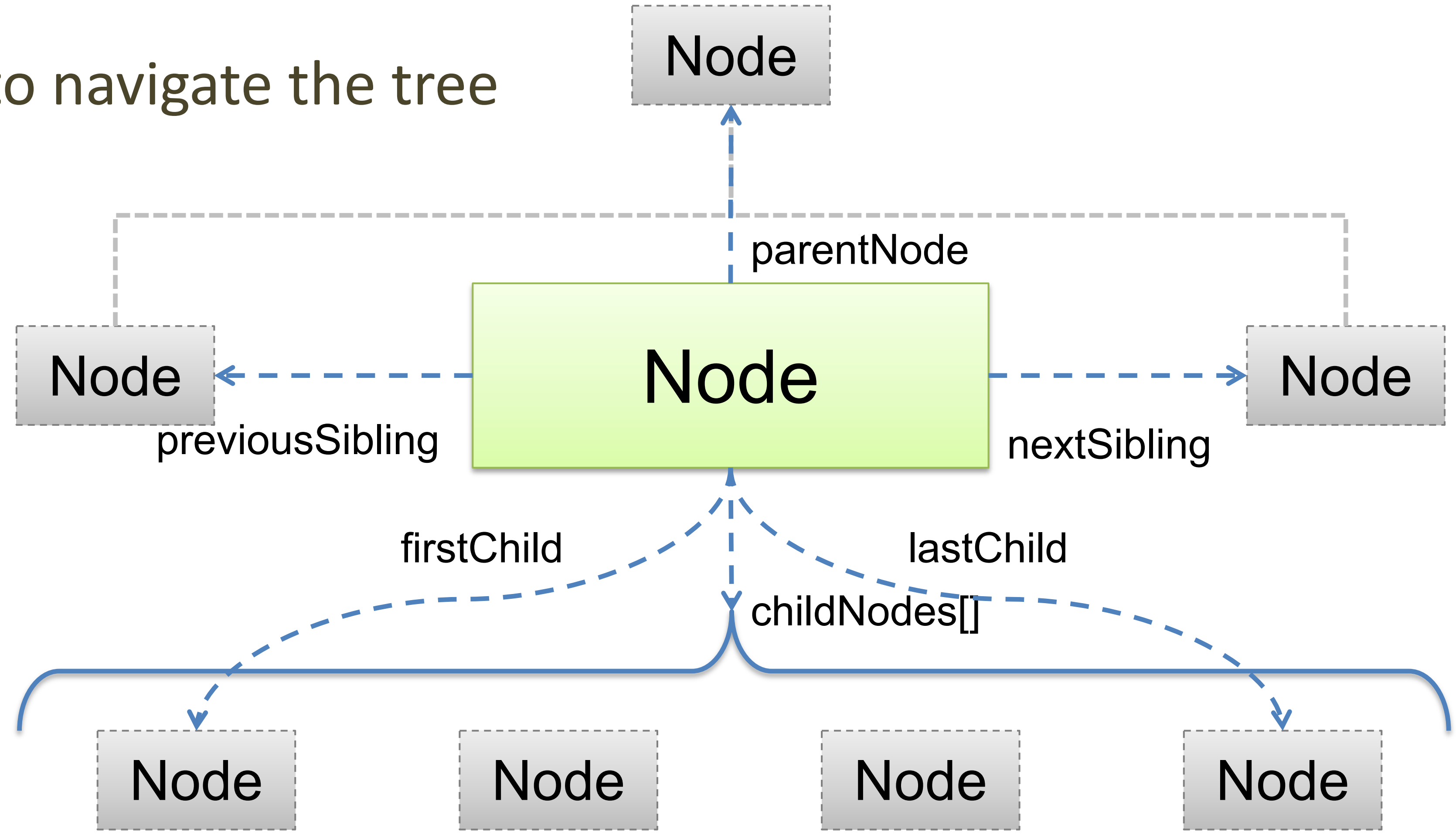
```
<!DOCTYPE html>
<html>
<head></head>
<body>
<div id="foo"></div>
<div class="bold"></div>
<div class="bold color"></div>
<script>
 document.getElementById('foo');
 document.querySelector('#foo');
 document.querySelectorAll('.bold');
 document.querySelectorAll('.color');
 document.querySelectorAll('.bold, .color');
</script>
</body>
</html>
```



```
   <div id="foo"></div>
   <div id="foo"></div>
 ▶ NodeList(2) [div.bold, div.bold.color]
 ▶ NodeList [div.bold.color]
 ▶ NodeList(2) [div.bold, div.bold.color]
>
```

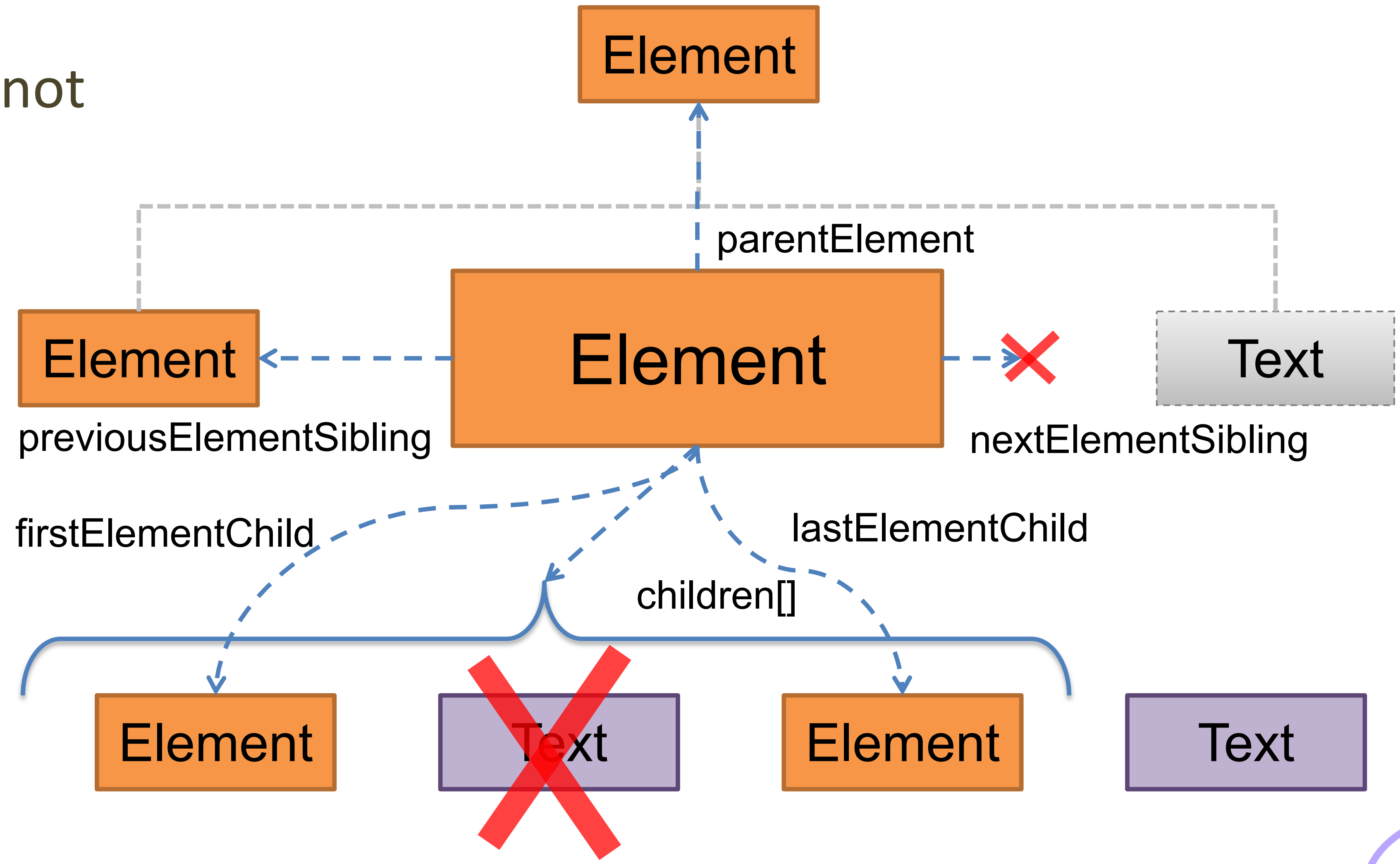
Navigating The Tree

- Properties to navigate the tree



Navigating The Tree

- "Elements" do not include text



Tag Attributes Exposed As Properties

- *Attributes* of the HTML elements become *object properties* of the DOM objects
- Example
  - `<body id="page">`
  - DOM object: `document.body.id="page"`
  - Also: `document["body"]["id"]`

  - `<input id="input" type="checkbox" checked />`
  - DOM object: `input.checked  // boolean`

Handling Tag Attributes

- `elem.hasAttribute(name)`
  - check the existence of the attribute
- `elem.getAttribute(name)`
  - check the value, like `elem[name]`
- `elem.setAttribute(name, value)`
  - set the value of the attribute
- `elem.removeAttribute(name)`
  - delete the attribute
- `elem.attributes`
  - collection of all attributes
- `elem.matches(css)`
  - Check whether the element matches the CSS selector

Creating Elements

- Use document methods:
  - `document.createElement(tag)` to create an element with a chosen tag
  - `document.createTextNode(text)` to create a text node with the given text
- Example: div with class and content

```
let div = document.createElement('div');
div.className = "alert alert-success";
div.innerText = "Hi there! You've read an important message.";
```

```
<div class="alert alert-success">
Hi there! You've read an important message.
</div>
```

Inserting Elements In The DOM Tree

- If not inserted, they will not appear

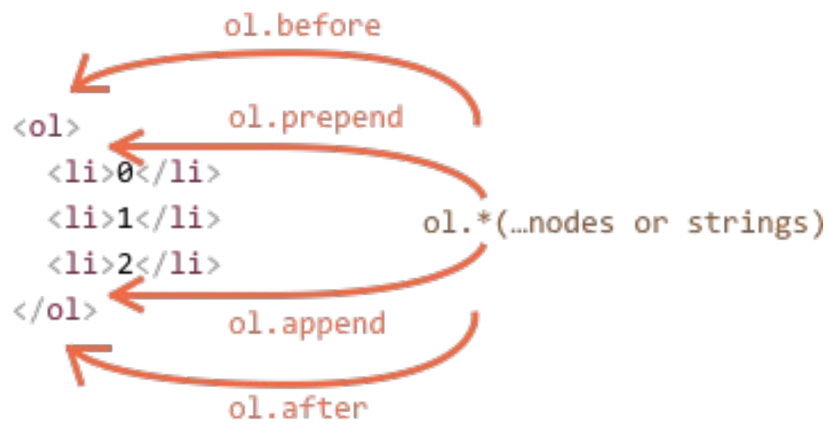
```
document.body.appendChild(div)
```

```
...
<body>
 <div class="alert alert-success">
 <strong>Hi there!</strong> You've read an important message.
 </div>
<body>
```

Inserting Children

- **parentElem.appendChild(node)**
- parentElem.insertBefore(node, nextSibling)
- parentElem.replaceChild(node, oldChild)

- node.append(…nodes or strings)
- node.prepend(…nodes or strings)
- node.before(…nodes or strings)
- node.after(…nodes or strings)
- node.replaceWith(…nodes or strings)



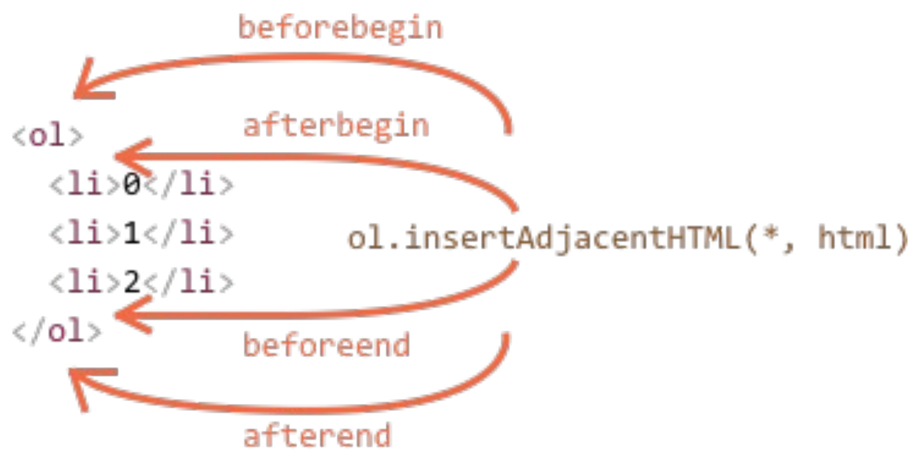
Handling Tag Content

- `.innerHTML` to get/set element content in textual form
- The browser will parse the content and convert it into DOM Nodes and Attrs

```
<div class="alert alert-success">
    <strong>Hi there!</strong> You've read an important message.
</div>
```

```
div.innerHTML  // "<strong>Hi there!</strong> You've read an important message."
```

Inserting New Content



```
<ol>
  <li>0</li>
  <li>1</li>
  <li>2</li>
</ol>
```

- `elem.innerHTML = "`*html fragment*`"`
- `elem.insertAdjacentHTML(where, HTML)`
  - where = `"beforebegin"` | `"afterbegin"` | `"beforeend"` | `"afterend"`
  - HTML = HTML fragment with the nodes to insert
- `elem.insertAdjacentText(where, text)`
- `elem.insertAdjacentElement(where, elem)`

Cloning Nodes

- `elem.cloneNode(true)`
  - Recursive (deep) copy of the element, including its attributes, sub-elements, …
- `elem.cloneNode(false)`
  - Shallow copy (will not contain the children)
- Useful to “replicate” some part of the document

DOM Styling Elements

- Via values of **class** attribute defined in CSS
- Change class using the property **`className`**
  - Replaces the whole string of classes
  - *Note*: `className`, not `class` (JS reserved word)
- To add/remove a single class use `classList`
  - `elem.classList.add("col-3")` add a class
  - `elem.classList.remove("col-3")` remove a class
  - `elem.classList.toggle("col-3")` if the class exists, it removes it, otherwise it adds it
  - `elem.classList.contains("col-3")` returns true/false checking if the element contains the class

DOM Styling Elements

- `elem.style` contains all CSS properties
  - Example: hide element

    ```
    elem.style.display="none"
    ```

  (equivalent to CSS declaration `display:none`)

- `getComputedStyle(element[,pseudo])`
  - `element`: selects the element of which we want to read the value
  - `pseudo`: a pseudo element, if necessary
- For properties that use more words the camelCase is used (`backgroundColor, zIndex...` instead of `background-color ...`)

Mozilla Developer Network: Event Reference
https://developer.mozilla.org/en-US/docs/Web/Events

EVENT HANDLING

Event Listeners

- JavaScript in the browser uses an *event-driven* programming model
  - Everything is triggered by the firing of an event
- Events are determined by
  - The Element generating the event (event ~~source~~ target)
  - The type of generated event

https://flaviocopes.com/javascript-events/

addEventListener()

- Can add as many listeners as desired, even to the same node
- Callback receives as first parameter an Event object

```
window.addEventListener('load', (event) => {
  //window loaded
})
```

```
const link = document.getElementById('my-link')
link.addEventListener('mousedown', event => {
  // mouse button pressed
  console.log(event.button) //0=left, 2=right
})
```

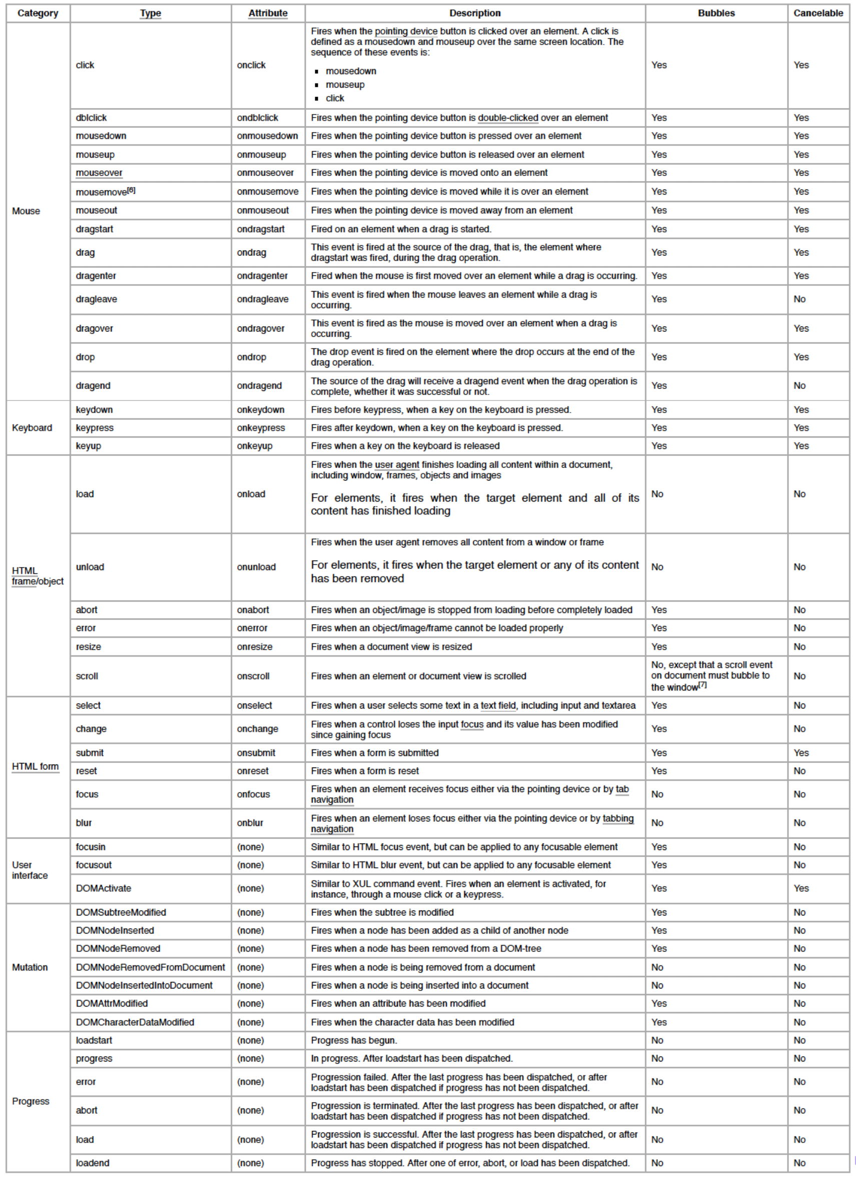
Event Object

- Main properties:
  - `target`, the DOM element that originated the event
  - `type`, the type of event

https://developer.mozilla.org/en-US/docs/Web/API/Event/type

Event Categories

- User Interface events (load, resize, scroll, etc.)
- Focus/blur events
- Mouse events (click, dblclick, mouseover, drag,
- Keyboard events (keyup, etc.)
- Form events (submit, change, input)
- Mutation events (DOMContentLoaded, etc.)
- HTML5 events (invalid, loadeddata, etc.)
- CSS events (animations etc.)



Category	Type	Attribute	Description	Bubbles	Cancelable
Mouse	click	onclick	Fires when the pointing device button is clicked over an element. A click is defined as a mousedown and mouseup over the same screen location. The sequence of these events is: ▪ mousedown ▪ mouseup ▪ click	Yes	Yes
	dblclick	ondblclick	Fires when the pointing device button is double-clicked over an element	Yes	Yes
	mousedown	onmousedown	Fires when the pointing device button is pressed over an element	Yes	Yes
	mouseup	onmouseup	Fires when the pointing device button is released over an element	Yes	Yes
	mouseover	onmouseover	Fires when the pointing device is moved onto an element	Yes	Yes
	mousemove[6]	onmousemove	Fires when the pointing device is moved while it is over an element	Yes	Yes
	mouseout	onmouseout	Fires when the pointing device is moved away from an element	Yes	Yes
	dragstart	ondragstart	Fired on an element when a drag is started.	Yes	Yes
	drag	ondrag	This event is fired at the source of the drag, that is, the element where dragstart was fired, during the drag operation.	Yes	Yes
	dragenter	ondragenter	Fired when the mouse is first moved over an element while a drag is occurring.	Yes	Yes
	dragleave	ondragleave	This event is fired when the mouse leaves an element while a drag is occurring.	Yes	No
	dragover	ondragover	This event is fired as the mouse is moved over an element when a drag is occurring.	Yes	Yes
	drop	ondrop	The drop event is fired on the element where the drop occurs at the end of the drag operation.	Yes	Yes
	dragend	ondragend	The source of the drag will receive a dragend event when the drag operation is complete, whether it was successful or not.	Yes	No
Keyboard	keydown	onkeydown	Fires before keypress, when a key on the keyboard is pressed.	Yes	Yes
	keypress	onkeypress	Fires after keydown, when a key on the keyboard is pressed.	Yes	Yes
	keyup	onkeyup	Fires when a key on the keyboard is released	Yes	Yes
HTML frame/object	load	onload	Fires when the user agent finishes loading all content within a document, including window, frames, objects and images For elements, it fires when the target element and all of its content has finished loading	No	No
	unload	onunload	Fires when the user agent removes all content from a window or frame For elements, it fires when the target element or any of its content has been removed	No	No
	abort	onabort	Fires when an object/image is stopped from loading before completely loaded	Yes	No
	error	onerror	Fires when an object/image/frame cannot be loaded properly	Yes	No
	resize	onresize	Fires when a document view is resized	Yes	No
	scroll	onscroll	Fires when an element or document view is scrolled	No, except that a scroll event on document must bubble to the window[7]	No
HTML form	select	onselect	Fires when a user selects some text in a text field, including input and textarea	Yes	No
	change	onchange	Fires when a control loses the input focus and its value has been modified since gaining focus	Yes	No
	submit	onsubmit	Fires when a form is submitted	Yes	Yes
	reset	onreset	Fires when a form is reset	Yes	No
	focus	onfocus	Fires when an element receives focus either via the pointing device or by tab navigation	No	No
	blur	onblur	Fires when an element loses focus either via the pointing device or by tabbing navigation	No	No
User interface	focusin	(none)	Similar to HTML focus event, but can be applied to any focusable element	Yes	No
	focusout	(none)	Similar to HTML blur event, but can be applied to any focusable element	Yes	No
	DOMActivate	(none)	Similar to XUL command event. Fires when an element is activated, for instance, through a mouse click or a keypress.	Yes	Yes
Mutation	DOMSubtreeModified	(none)	Fires when the subtree is modified	Yes	No
	DOMNodeInserted	(none)	Fires when a node has been added as a child of another node	Yes	No
	DOMNodeRemoved	(none)	Fires when a node has been removed from a DOM-tree	Yes	No
	DOMNodeRemovedFromDocument	(none)	Fires when a node is being removed from a document	No	No
	DOMNodeInsertedIntoDocument	(none)	Fires when a node is being inserted into a document	No	No
	DOMAttrModified	(none)	Fires when an attribute has been modified	Yes	No
	DOMCharacterDataModified	(none)	Fires when the character data has been modified	Yes	No
Progress	loadstart	(none)	Progress has begun.	No	No
	progress	(none)	In progress. After loadstart has been dispatched.	No	No
	error	(none)	Progression failed. After the last progress has been dispatched, or after loadstart has been dispatched if progress has not been dispatched.	No	No
	abort	(none)	Progression is terminated. After the last progress has been dispatched, or after loadstart has been dispatched if progress has not been dispatched.	No	No
	load	(none)	Progression is successful. After the last progress has been dispatched, or after loadstart has been dispatched if progress has not been dispatched.	No	No
	loadend	(none)	Progress has stopped. After one of error, abort, or load has been dispatched.	No	No

https://en.wikipedia.org/wiki/DOM_events

Preventing Default Behavior

- Many events cause a default behavior
  - Click on link: go to URL
  - Click on submit button: form is sent
- Can be prevented by `event.preventDefault()`

HTML Page Lifecycle: Events

- `DOMContentLoaded` (defined on `document`)
  - The browser loaded all HTML, and the DOM tree is ready
  - External resources are not loaded, yet
- `load` (defined on `window`)
  - The browser finished loading all external resources
- `beforeunload/unload`
  - The user is about to leave the page / has just left the page
  - Not recommended (non totally reliable)

```
document.addEventListener("DOMContentLoaded", ready);
```

Throttling

- Some events fire continuously (mousemove, scroll, etc.) providing coordinates, so that user behavior can be tracked
- Complex operations in the event handler result in sluggish user experience
- Use external libraries or set timers to process them only periodically

```
let cached = null ;
window.addEventListener('scroll', event => {
  if (!cached) {
    setTimeout(() => {
      // process event -- you can access the original event at `cached`
      cached = null ;
    }, 100) }
  cached = event ;
}) ;
```

Mozilla Developer Network:
Web forms — Form Validation
https://developer.mozilla.org/en-US/docs/Learn/Forms/Form_validation

FORM EVENTS

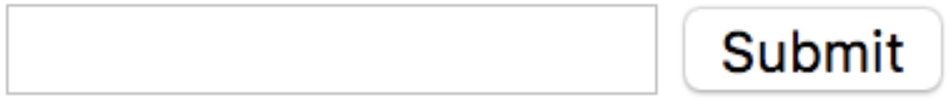
Events On Input Elements

Event	Meaning
`input`	the value of the element is changed (even a single character)
`change`	when something changed in the element (for text elements, it is fired only once when the element loses focus)
`cut copy paste`	when the user does the corresponding action
`focus`	when the element gains focus
`blur`	when the element loses focus
`invalid`	when the form is submitted, fires for each element which is invalid, and for the form itself

https://developer.mozilla.org/en-US/docs/Learn/Forms/Form_validation

Example

```
...
<form action="/add" method="POST">
  <input type="text">
  <input type="submit">
</form>
...
```



```
const inputField = document.querySelector('input[type="text"]')

inputField.addEventListener('input', event => {
  console.log(`The current entered value is: ${inputField.value}`);
})

inputField.addEventListener('change', event => {
  console.log(`The value has changed since last time: ${inputField.value}`);
})
```

Form Submission

- Can be intercepted with the `submit` event
- If required, default action can be prevented in eventListener with the `preventDefault()` method
  - A new page is NOT loaded, everything must be handled in JavaScript

```
document.querySelector('form').addEventListener('submit', event => {
    event.preventDefault();
    console.log('submit');
})
```

License