TODO-DOM Lezione12-JS in the browser

Thursday, March 25, 2021 8:33 AM

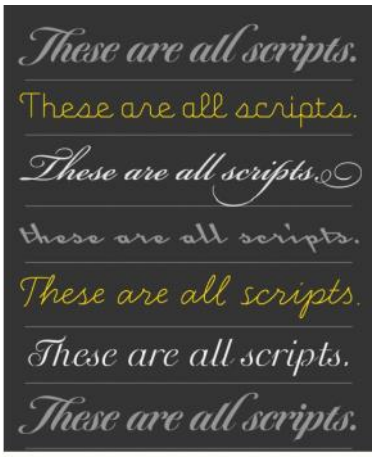
<WA1/>
<AW1/>
2021

JS In The Browser

Handling web document structure

Fulvio Corno
Luigi De Russis
Enrico Masala

Some slides adapted from Giovanni Malnati



Goal

Empower webapp with js

- Loading JavaScript in the browser
- Browser object model
- Document object model
- DOM Manipulation
- DOM Styling
- Event Handling
- Forms

Mozilla Developer Network: The Script element
https://developer.mozilla.org/en-US/docs/Web/HTML/Element/script

JS in the browser

LOADING JS IN THE BROWSER

Loading JavaScript In The Browser

- JS must be loaded from an HTML document
- `<script>` tag
  - Inline

```
...
<script>
alert('Hello');
</script>
...
```

  - External

```
...
<script src="file.js"></script>
...
```

https://developer.mozilla.org/en-US/docs/Web/HTML/Element/script

Anywhere in the page: when browser reads this instruction it stops the processing of the html and starts executing code immediately! Where should we put it?

JavaScript External Resources

- JS code is loaded from one or more external resources (files)
- Loaded with `src=` attribute in `<script>` tag
- The JS file is loaded, and immediately executed
  - Then, HTML processing continues

```
<script src="file.js"></script>
<!-- type="text/javascript" is the default: not needed -->
```


?

Where To Insert The <script> Tag?

- In the <head> section
  - "clean" / "textbook" solution
  - Very **inefficient**: HTML processing is stopped until the script is loaded and executed
  - Quite **inconvenient**: the script executes when the document's DOM does not exist yet
  - *But*: see after!
- Just before the end of the document
  - More efficient than the "textbook" solution

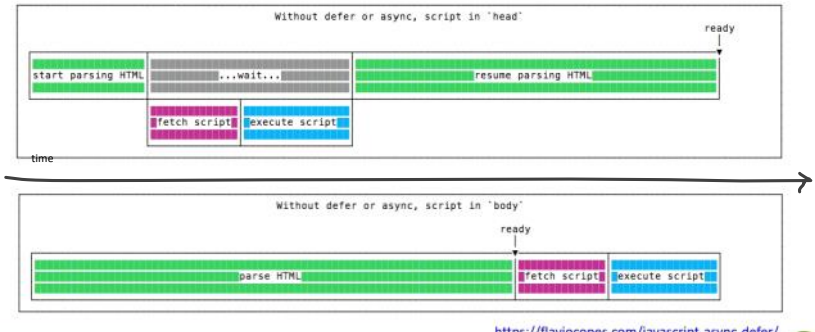
```
<!DOCTYPE html>
<html>
    <head>
        <title>Loading a script</title>
        <script src="script.js"></script>
    </head>
    <body>
        ...
    </body>
</html>
```

Clean! But inefficient!
You see a blank page until the script is loaded and executed!
Our js CANNOT access anything in the body because it is not loaded yet! It just loads things to do later!

```
<!DOCTYPE html>
<html>
    <head>
        <title>Loading a script</title>
    </head>
    <body>
        ...
        <script src="script.js"></script>
    </body>
</html>
```

Last instruction before </body>
CAN access anything in the body, because it is already loaded!
Multiple js script at the end are concatenated all together!

Performance Comparison In Loading JS



User see blank page and waits until all is executed

https://flaviocopes.com/javascript-async-defer/
User see webpage here, but can interact with it only when script is executed!

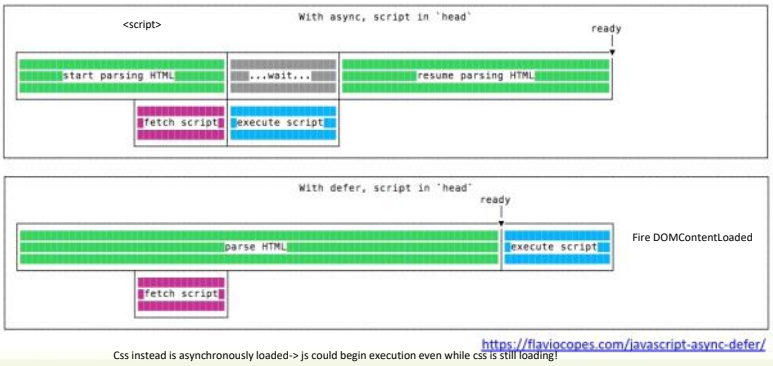
Traditional way!

New Loading Attributes

- `<script async src="script.js"></script>`
  - Script will be fetched in parallel to parsing and evaluated as soon as it is available
  - Not immediately executed, not blocking
- `<script defer src="script.js"></script>` (*preferred*)
  - Indicate to a browser that the script is meant to be executed after the document has been parsed, but before firing `DOMContentLoaded` (that will wait until the script is finished)
  - Guaranteed to execute in the order they are loaded
- Both should be placed in the <head> of the document

We do not stop html parsing, parallel fetching, but when the code is executed the parsing is stopped. Big problem: we can't know what portion of the page has been loaded when the script begins execution!

defer vs. async



https://flaviocopes.com/javascript-async-defer/

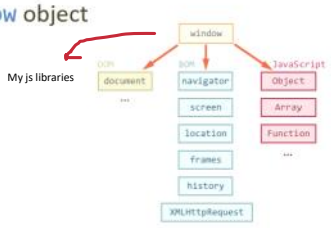
Css instead is asynchronously loaded-> js could begin execution even while css is still loading!

We don't know how long is the fetch of the script! We don't know when it will be execution! Non deterministic!
If many script: The first that is fetched completely is executed first! Even worse!

Wait until the whole page has been parsed entirely before executing it!
Browser can open new os (network) thread to fetch the script in parallel to the parsing of the HTML (as it would do when an image present in the html is required to be fetched) BUT it can't parse HTML and execute the script in parallel, otherways the js could try to modify data structure that are being created in that moment, creating an undefined behaviour depending on the speed of the browser parsing vs the speed of the browser js executing. We could do those in parallel only if we want to execute js that doesn't interact with the page, like loading libraries, but that is very rarely the case! JS is made for modifying the page!

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



Each js file is a different program, but they will all be concatenated as a big js file with a global memory space shared by all: the window object. (it is defined at the browser level). Every name defined in our js files become an attribute of that windows object. We need to verify the attributes declared by other libraries and doesn't have any malware since we will share our code together, to avoid overwriting of objects (I can't call my variable dayjs for example.) See their github!
We could technically use window.console.log, since everything is an attribute of the window global space, but since it is global for everything, we omit it.
From our code we can access the JS Standard library (using appropriate APIs), the BOM and the DOM, which are stored inside the browser itself.

Run chrome from vscode->sources we can see the code and have a debugger inside frontend: code executed FROM THE BROWSER! (Not from node/terminal)
Some information in the frontend will also be reported into vscode, but not all info!

N.B. const dayjs=require('dayjs'); DOES NOT WORK IN THE BROWSER because require is a function specific to node for loading modules. Browser with ES6 use different mechanism based on modules to do that. Import statement instead of require function.
In the browser we don't have the node modules folder! We just have our js file (+standard libraries..)!
We can't automatically load from node modules because then the browser won't know where to find that module.

We need to load the script of the library DIRECTLY IN THE HTML and than we are able to use them, using the definitions in the other files, because they become global (for ALL the next files! Not very clean! We'll see not global modules!) Very difficult for the programmers and the IDE to find where the declaration were!

Events and Event Loop

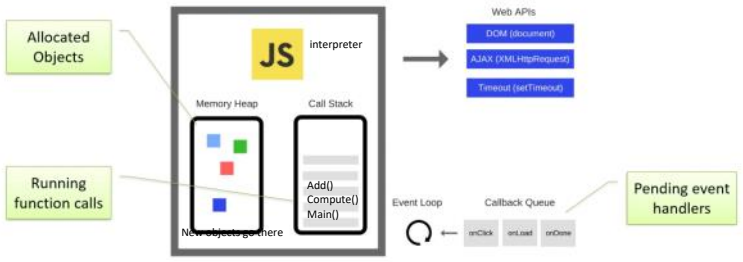
- Most phases of processing and interaction with a web document will generate Asynchronous *Events* (100's of different types)
- Generated events may be handled by:
  - Pre-defined behaviors (by the browser)
  - User-defined *event handlers* (in your JS)
  - Or just ignored, if no event handler is defined
- 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

Local vs external content distribution network -> which is faster? Is your website faster than the cdn? Which is faster for the user? Cdn is faster, reduces the load for our server and often client has them cached, but we will depend on others for our page to work.

After finish loading, all became asynchronous, based on the handling of the events! Browser generates events at EVERY EVENT (MOVE MOUSE, FINISH LOADING IMAGE,...)
2 handles:
1. Defined by the browser! (button has visual effect/ textbox has default handler)
2. Our custom handler for each event!

Event loop: JS interpreter executes a BIG EVENT LOOP, whenever an event wants to be executed it is added to a message queue of events that wants to be executed. After the end of the loading/execution no more synchronous operations are needed.

Execution Environment



Top of the call stack is the function executed right now. If other external event schedule asynch operations it will be loaded in the callback queue. When all synch functions in the callstack are completed, the next event of the queue begins execution, its event handler (callback) is added to the call stack and will execute synchronously! (adding eventually other callbacks to the call stack)
Only one function is being executed at a time!
Always create small functions. Db.all starts only after its call stack is emptied, because it is asynchronous!! IT DOESN'T STOP THE EXECUTION OF THE WHOLE CALLBACK QUEUE! When the db will return the result, a new event will be generated and added to the message queue The important part is that each function returns soon (create only small fucntions when possible!!)

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!

Synchronous execution

Asynchronous execution

https://developer.mozilla.org/en-US/docs/Web/JavaScript/EventLoop
https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/#what-is-the-event-loop

JS in the browser

BROWSER OBJECT MODEL

What objects can we access?

Browser Main Objects

We can interact with the browser itself using the BOM API.

- `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

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

Back and forward: list of locations

Storage for webpages! (dictionary) + getter/setter
Memory inside the browser saves info like login tokens (each LocalStorage is only accessible for that domain!->if you manage to hack the internal sandbox of the browser you could access other sites LocalStorage!)

SessionStorage only saves until end of session (close browser)

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

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

Create new custom event handlers! When i click there I want to execute my js code! Extends predefined event handler system!

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

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

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

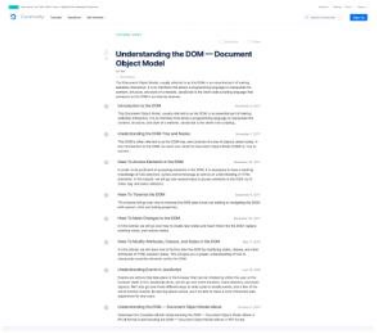
```
let storage = permanent ? window.localStorage :
                          window.sessionStorage;
if(!storage["name"]) {
   storage["name"] = "A simple storage"
}
alert("Your name is " + storage["name"]);
```

TU Informatics

JS in the browser

DOCUMENT OBJECT MODEL

Suggested Reading



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

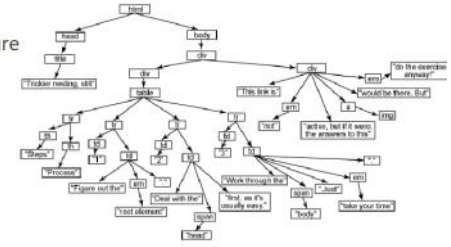
DOM Living Standard

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

Continuous evolution of functionalities

DOM

- Browser's internal representation of a web page
- Obtained through parsing HTML
  - Example of parsed HTML tree structure
- Browsers expose an API that you can use to interact with the DOM



https://flaviocopes.com/dom/

Crucial for understanding how stylesheets work!

Interaction With The DOM

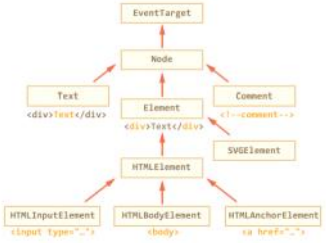
- Via JavaScript it is possible to
  - Access the page metadata and headers
  - Inspect the page structure
  - Edit any node in the page
  - Change any node attribute
  - Create/delete nodes in the page
  - Edit the CSS styling and classes
  - Attach or remove *event listeners*

Document is the global object to access for programming the dom

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



3 types of nodes:
Element(can also be children):
Attr (only children): src='example.jpg' (name+value)
Text(only children): Hello
Element text Element (3 elements)

Each tag is a different element

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 primitive
  - `for…of` for classical iteration

Data structure, array (very similar to array, little bit more powerful) for containing nodes. When we need to return many nodes.

JS in the browser

DOM MANIPULATION

1.

Finding DOM elements

Historically the first 4 where in the first dom version, jQuery popularized the css selector $('main h1').
JQuery less used than before because now dom has these methods integrated to access easily elements

- `document.getElementById(value)`
  - Node with the attribute id=value
- `document.getElement`**`s`**`ByTagName(value)`
  - NodeList of all elements with the specified tag name (e.g., 'div')
- `document.getElement`**`s`**`ByClassName(value)`
  - NodeList of all elements with attribute class=value (e.g., 'col-8')
- `document.querySelector(css)`
  - First Node element that matches the CSS selector syntax
- `document.querySelectorAll(css)`
  - NodeList of all elements that match the CSS selector syntax

'p' -> All the paragraphs

If only 1-> result[0] = first element

Same syntax as css! -> faster if I know that I only need the first/I know there is only 1 element

Most powerful: use the css engine to interpret selector, not css files!

https://flaviocopes.com/dom/

After this I can use a JS Object to access any element in the page that we want!
Query-> result[0].innerText extract text from the first result

Note

Query-> result[0].innerText extract text from the first result

Note

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

If a method returns an element

H1 inside header and h1 inside main:
To extract text inside h1 inside main multiple ways returns the same value:

1. Select all h1 in the page (getElementsByTagName h1)and select result[1]
2. Select h1 inside the main QuerySelector('main h1') (most readable, even better if I have ID)
3. Find the main section, and inside it I find h1! (getElementsByTagName('main'))
   [0].getElementsByTagName('h1')[0]

Documents are separated by the tab!

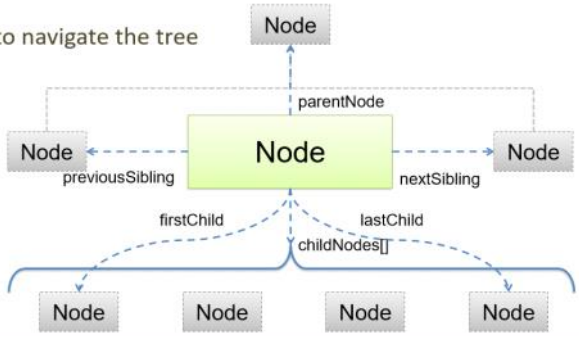
Accessing DOM Elements

```
<!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



DOM nodes have hundreds of attributes (html attributes, but also childNodes and other attributes to navigate the tree).
Node.attributes->id, class,...

For easy pages if i stop the js for debugging, the page is frozen!

Tag Attributes Exposed As Properties

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

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

- For manipulating attributes, use the methods in the next slide

Handling Tag Attributes

- `elem.hasAttribute(name)`
  - check the existence of the attribute
- `elem.getAttribute(name)`
  - check the value
- `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 tag
  - `document.createTextNode(text)` to create a text node with the 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>
```

Create Elements doesn't add it to the page immediately, we need to add it as a child before!

Let Newp=document.CreateElement('P');
Let Text=document.CreateTextNode("1234");
Newp.AppendChild(text);
FindelementsbyTag(Main).AppendChild(newp);

Long process!

Inserting Elements In The DOM Tree

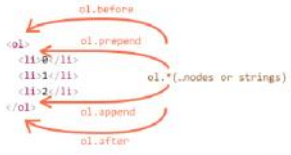
- If not inserted, they will not appear
  `document.body.appendChild(div)`

```
...
<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

Property of a node that contains all html inside that node!

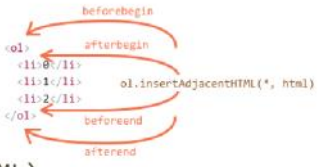
Unlike before, it is quicker to do (easier but slower)
Findelementbytag(main).innerHTML += " " + i + " "

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

N.B. innerHTML no newline, only paragraphs!

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


Inserting New Content



- `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

Add/remove classes easily!

- 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

String

List-> best use! We do not need to manipulate css, we just add/remove classes, then browser applies immediately the effects of the css to the elements

DOM Styling Elements

All style properties are inside the style property

- `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

JS in the browser

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
- JavaScript supports three ways of defining event handlers
  - Inline event handlers
  - DOM on-event handlers
  - **Using `addEventListener()`** ← *modern way*

Target of the event is actually its source...-.-

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

Event object, callback to execute when it is triggered with parameter=event type to handle!

```
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
})
```

Each event needs its own callback
For having multiple events doing the same thing we need 1 for each or 1 event for the container of all elements that we need to act on?

If i click on h1, the target property of the click event will show h1 (the element from which the event generated!)

Deciding what event to handle, anywhere, and decide where to act in response and what to do!

Only limitation: be fast! we do not want to block the page while computation happens! For example we should do query selector outside and use the result inside the callback thanks to closure property!

Define function and use it in multiple places if needed! (Do not declare in-place)

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

Event Object

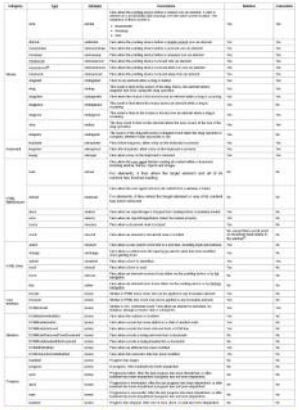
- Main properties:
  - `target`, the DOM element that originated the event
  - `type`, the type of event
  - `stopPropagation()` called to stop propagating the event in the DOM (onward)

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

Types of events that can be generated

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.)



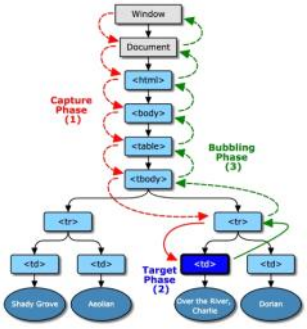
https://en.wikipedia.org/wiki/DOM_events

https://medium.com/prod-io/javascript-understanding-dom-event-life-cycle-49e1cf62b2ea

Event object propagates through the tree

Event Handling On The DOM Tree

- Something occurs (e.g., a mouse click, a button press)
- Capture phase
  - The event is passed to all DOM elements on the path from the Document to the parent of the target element
  - No event handlers are fired
    - Except if registered with useCapture=true
- Target phase
  - The event reaches the target
  - Event handlers are triggered
- Bubbling phase
  - Trace back the path towards the document root
  - Event handlers are triggered on any encountered node
  - Allows us to handle an event on any element by its parent elements
  - `event.stopPropagation()` interrupts the bubbling phase



All events handlers for the same type of events that are registered until the root of the tree are executed!
We can define the handler at the common ancestor of all elements that we want to manage event on, instead of copy-pasting the handler in all the descendants: thanks to the target attribute we know exactly who generated it!

When we handle event we usually want to stop the propagation!

Preventing Default Behavior

Some events are automatically handled by the browser! Click link/click submit button in forms. Other default action are very welcome! (click on text box>focus)
Some disable right click/select text-> not a good practice to modify this browser behaviour! Annoying!

- 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()`

Stopping Event Propagation

- Can be done with `event.stopPropagation()`
  - Typically in the event handler

```
const link = document.getElementById('my-link')
link.addEventListener('mousedown', event => {
  // process the event
  // ...

  event.stopPropagation()
})
```


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);
```

Best practice

Images, css styles not yet!

Usually all js files should wait for the right moment to execute (after elements have loaded completely..)
We should link all of our code to an event that tell us when the browser has correctly loaded the content that we need!

Define event handler to customize our html (or empty html (twitter))

Function callback{
//all code goes there!!! All eventlistener are defined here!
PageTitle=Select header h1
Articletitle=Select main h1

Articletitle.addeventlistener(click, (event)=>{

pagetitle.classlist.toggle('bg-primary'));
});
Document.Addeventlistener(domcontentloaded, callback); ->this is the only synchronous instruction!!!

Mozilla Developer Network:
Web forms — Collecting data from users
https://developer.mozilla.org/en-US/docs/Learn/Forms

Handling user input

FORM CONTROLS

Form Declaration

- `<form>` tag
- Specifies URL to be used for submission (attribute `action`)
- Specifies HTTP method (attribute `method`, default `GET`)

```
...
<form action="/new-task" method="POST" id="userdata">
       ...normal HTML content...
              and
       ...FORM Controls...
</form>
...
```

Form is a normal invisible container (can contain html content), but can also contain form controls (sort of widgets).
The form is identified by an id.
Each of its form controls (input, selection, button, label, datalist) must have its name, by which we can extract what the user input is, which is the value associated with that name.

Action to be carried on when the form is submitted, tipically send data to the server to do processing.
In our case we don't want to submit forms, since it would unload all of our js! We don't want the default behaviour of the browser, we want to handle forms by ourself on the frontend!

Form Controls

- A set of HTML elements allowing different types of user input/interaction. Each element should be uniquely identified by the value of the `name` attribute
- Several control categories
  - Input
  - Selection
  - Button
- Support elements
  - Label
  - Datalist

https://developer.mozilla.org/en-US/docs/Web/HTML/Element#Forms

Input Control

- `<input>` tag
- Text input example
- The `value` attribute will hold user-provided text

xxxxxxxxxxxx

```
...
<input type="text" name="firstname" placeholder="Your username"> </input>
...
```

Some tags (such as `<input>` and `<img>` and `<br />` and `<hr />`) are singular tags and have no legal closing tags

Your firstname

Locating a Form In The DOM

Quicker than query selector!

- `document.forms` is a collection of all forms in the page

  `const myForm = document.forms['form ID']`
- The `form` node has an **`elements`** properties, that collects all data-containing inner elements

  `const myElement = myForm.elements['element ID']`

For normal html elements with an id inside the form

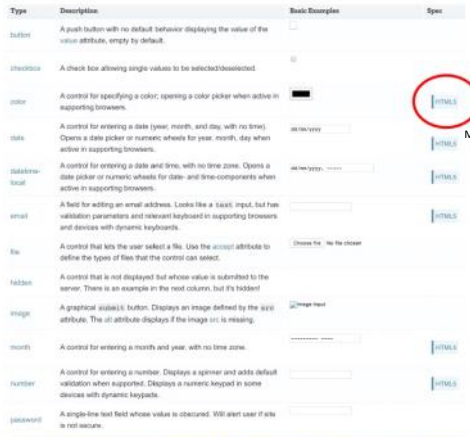
Or elements names for form controls

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

Input Control (1)

Form control input can be of different types

- `type` attribute
  - button
  - checkbox
  - color
  - date
  - email
  - file
  - hidden
  - month
  - number
  - password

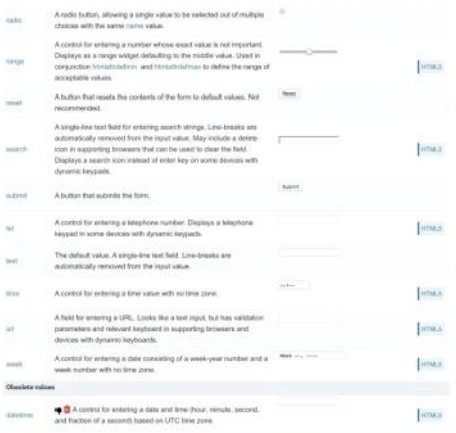


May not work, depends on the browser

https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input

Input Control (2)

- type attribute
  - radio (button)
  - range
  - submit/reset (button)
  - search
  - tel
  - text
  - url
  - week



https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input

Input Control: Commonly Used Attributes

Attributes of the form control input of type...

Attribute	Meaning
checked	radio/checkbox is selected
disabled	control is disabled
readonly	value cannot be edited
required	need a valid input to allow form submission
size	the size of the control (pixels or characters)
value	the value inserted by the user
autocomplete	hint for form autofill feature of the browser

https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input#Attributes

Input Control: Other Attributes

- Depends on the control

```
<input type="number" name="age" placeholder="Your age" min="18" max="110" />

<input type="text" name="username" pattern="[a-zA-Z]{8}" />

<input type="file" name="docs" accept=".jpg, .jpeg, .png" />
```

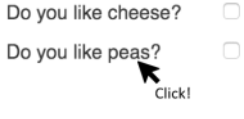
Constrain input data that could be inserted (easy validation)

https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input#Attributes

Label Tag

- The HTML `<label>` element represents a caption for an item in a user interface. Associated with `for` attribute and `id` on input
- Important for accessibility purposes (e.g. screenreader etc.), clicking the label activates the control (larger activation area e.g. in touch screens)

```
<div class="preference">
    <label for="cheese">Do you like cheese?</label>
    <input type="checkbox" name="cheese" id="cheese">
</div>
<div class="preference">
    <label for="peas">Do you like peas?</label>
    <input type="checkbox" name="peas" id="peas">
</div>
```



https://developer.mozilla.org/en-US/docs/Web/HTML/Element/label

Other Form Controls

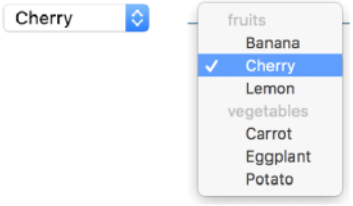
`<textarea>`:
a multi-line text field

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

Other Form Controls

Drop-down controls

```
<select id="groups" name="groups">
  <optgroup label="fruits">
    <option>Banana</option>
    <option selected>Cherry</option>
    <option>Lemon</option>
  </optgroup>
  <optgroup label="vegetables">
    <option>Carrot</option>
    <option>Eggplant</option>
    <option>Potato</option>
  </optgroup>
</select>
```



Select tag has name and options children are the various values

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

Button Control

- `<button>` tag
- Three types of buttons
  - `submit`: submits the form to the server
  - `reset`: reset the content of the form to the initial value
  - `button`: just a button, whose behavior needs to be specified by JavaScript

```
...
<button type="submit" value="Send data" />
...
```

Any type of html content can become a button!
Many types of buttons already exists but we want to override the predefined behaviour to customize ours

Button vs. input type=button

More flexible, can have content (markup, images, etc.)

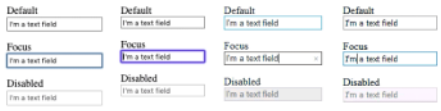
```
...
<button class="favorite styled"
        type="button">
    Add to favorites
</button>
...
<button name="favorite">
  <svg aria-hidden="true" viewBox="0 0 10 10"><path
d="M7 9L5 8 3 9V6L1 4h3l1-3 1 3h3L7 6z"/></svg>
  Add to favorites
</button>
...
```

https://developer.mozilla.org/en-US/docs/Web/HTML/Element/button

Bootstrap can make forms look nice!

Default Appearance May Vary

- Solve with CSS, but
- Some problems still remain
  - See: "Styling web forms" in MDN
  - Examples of controls difficult to manage:
    - Bad: Checkboxes, ...
    - Ugly: Color, Range, File: cannot be styled via CSS



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

The Road to Nicer Forms

- Useful libraries (frameworks) and polyfills
  - Especially for controls difficult to handle via CSS
  - Rely on JavaScript
- Suggestions
  - Bootstrap
  - Using libraries may improve accessibility

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

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

When clicking on form or modifying stuff the focus goes there (default handler).
Save button-> handle event to check data and save them/send them somewhere

Handling user input

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}`);
})
```


Handle the whole form or the individual inputs (elements property of the form object: UserForm.elements['id of the username input'].value contains all the text that the user is writing in that moment!)!
Validate AS the user is going trough the fields! (change event fires when form loses focus!)

Prevent default to avoid the automatic submission of the form!
For example show error message (add message or make it visible,..) when submitting without the mandatory fields!

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 is handled in the JavaScript: single page application

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


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

Handling user input

FORM VALIDATION

Form Validation?

- When entering data into a form, the browser will check to see if the data is in the correct format and with the constraints set by the application
  - Client-side validation: via HTML5 and JavaScript
  - Server-side validation: the application server will take care of it
- After client-side validation, data can be submitted to the server
- Why client-side validation?
  - We want to get the right data in the right format before processing the data
  - We want to protect users' data (e.g., enforcing secure passwords)
  - We want to protect the application (however, **NEVER TRUST** client-side validation on server side)

In html5 some attributes exists to perform form validation automatically by the browser itself!

Types Of Client-Side Validation

Since from the html form type the browser already know which kind of data will that form contain, it can perform easy validation of what is inserted.

- Built-in form validation by HTML5 input elements. Examples:
  - Email: check if the inserted value is a valid email (syntax only)
  - URL: check if it is a valid URL
  - Number: check if the text is a number
  - Attribute `required`: if a value is not present, form cannot be submitted
  - …
- JavaScript validation: custom code is used to check correctness of values

Built-In Form Validation

Browser itself give an error message! This is useful! You just need to add the "required" attribute:
<input type="text" class="form-control" name='username' id='username' required />

- Mainly relies on element attributes such as:
  - **`required`**: if a value is not present, form cannot be submitted
  - **`minlength maxlength`** for text
  - **`min max`** for numerical values
  - **`type`**: type of data (email, url, etc.)
  - **`pattern`**: regular expression to be matched
- When element is valid, the `:valid` CSS pseudo-class applies, which can be used to style valid elements, otherwise `:invalid` applies

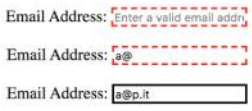
Element that automatically changes its color (style) whether its content respects certain rules or not!

Built-In Form Validation Styling

```
...
<form>
  <label for="e_addr">Email Address:<label>
  <input type="email" id="e_addr" id="email" required
placeholder="Enter a valid email address">
</form>
...
```

```
input:invalid {
  border: 2px dashed red;
}

input:valid {
  border: 2px solid black;
}
```



JavaScript Validation

- JavaScript must be used to take control over the look and feel of native error messages
- Approaches:
  - Constraint Validation API
  - **eventListeners** on some specific events

Constraint Validation API

Property/method	Function
`validationMessage`	a localized message describing the validation constraints that the control doesn't satisfy
`validity`	a ValidityState object, that includes sub-properties: patternMismatch, tooLong, tooShort, rangeOverflow, rangeUnderflow, typeMismatch, valid, valueMissing, …
`willValidate`	true if the element will be validated when the form is submitted
`checkValidity()`	true if the element's value has no validity problems. If invalid, it fires an *invalid* event.
`setCustomValidity(`*`message`*`)`	Adds a custom error message to the element: the element is treated as invalid, and the specified error is displayed

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

API that queries the browser about the validation status of the forms, determined by the browser validation, and depending on the status we can customize the error message!
We can set each property on each input element (set custom validationMessage) or we could read properties from each input element (if(validity.typeMismatch)...else if(validity.tooShort)...) or we could use methods of each input element.

For all object it includes all validity state, to know why the form input was invalid

Typically submit or button clicks action.
With forms use browser validation.
With buttons to show/remove/change the page just create a custom handler.
Since there are lots of events and dom nodes attributes, we could create incredible things!

References

- Web forms — Collecting data from users
  - https://developer.mozilla.org/en-US/docs/Learn/Forms
- Basic native form controls
  - https://developer.mozilla.org/en-US/docs/Learn/Forms/Basic_native_form_controls
- The HTML5 input types
  - https://developer.mozilla.org/en-US/docs/Learn/Forms/HTML5_input_types
- Client-side form validation
  - https://developer.mozilla.org/en-US/docs/Learn/Forms/Form_validation
- Constraint validation
  - https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5/Constraint_validation
- Constraint validation API
  - https://developer.mozilla.org/en-US/docs/Web/API/Constraint_validation

References

- Web Engineering SS20 - TU Wien, prof. Jürgen Cito, https://web-engineering-tuwien.github.io/
- Async and defer
  - Efficiently load JavaScript with defer and async, Flavio Copes, https://flaviocopes.com/javascript-async-defer/
  - https://hacks.mozilla.org/2017/09/building-the-dom-faster-speculative-parsing-async-defer-and-preload/