

Web Interactions

Https - Html - Javascript

arnaud.nauwynck@gmail.com

Outline

Https → Load Html + Js + Css + Json + *

Js → Call Http : SPA, Rest + Json

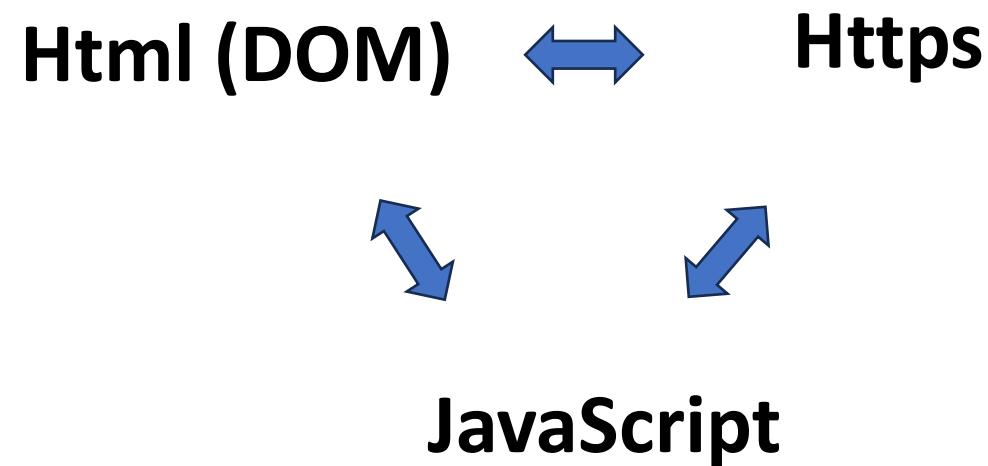
Html → execute Js <script>

Html → trigger events to Js

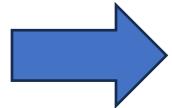
Js → Query/Update Html DOM

Custom Elements, Shadow DOM, Frameworks

Interactions



Outline



Https → Load Html + Js + Css + Json + *

Js → Call Http : SPA, Rest + Json

Html → execute Js <script>

Html → trigger events to Js

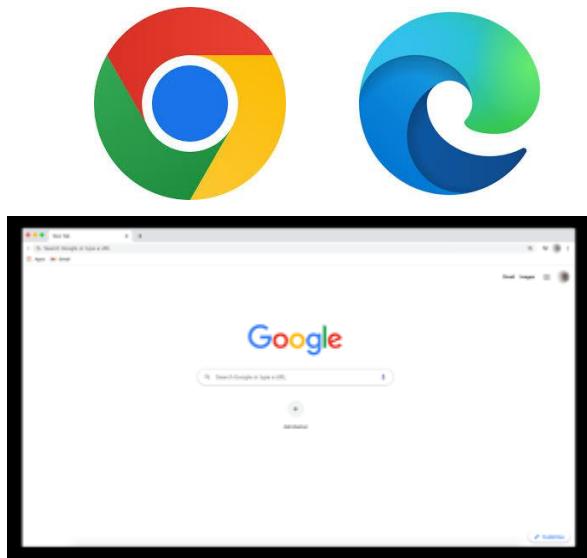
Js → Query/Update Html DOM

Custom Elements, Shadow DOM, Frameworks

Web Browser = Http Client

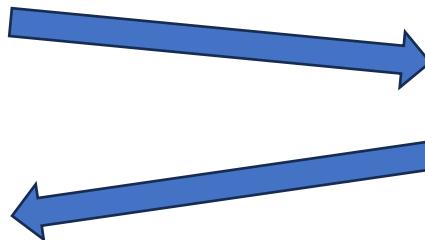
Web Site = Http Server

Your PC
mydesktop.home
192.168.0.1



Http Server
DNS hostname: www.google.com
Port : 8443
Protocol: Http + TLS
Resource path: some-path
Url: https://www.google.com:8443/some-path

http Request



http Response



Http Request (GET, POST, PUT, ...)

<https://en.wikipedia.org/wiki/HTTP>

HTTP/1.1 example of request / response transaction [\[edit\]](#)

Below is a sample HTTP transaction between an HTTP/1.1 client and an HTTP/1.1 server running on [www.example.com](#), port 80. [\[note 5\]](#)[\[note 6\]](#)

Client request [\[edit\]](#)

```
GET / HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

A client request (consisting in this case of the request line and a few headers that can be reduced to only the "Host: hostname" header) is followed by a blank line, so that the request ends with a double end of line, each in the form of a [carriage return](#) followed by a [line feed](#). The "Host: hostname" header value distinguishes between various [DNS](#) names sharing a single [IP address](#), allowing name-based [virtual hosting](#). While optional in HTTP/1.0, it is mandatory in HTTP/1.1. (A "/" (slash) will usually fetch a [/index.html](#) file if there is one.)

Http Response (200, 300, 400, 500 ...)

<https://en.wikipedia.org/wiki/HTTP>

Server response [edit]

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 155
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
ETag: "3f80f-1b6-3e1cb03b"
Accept-Ranges: bytes
Connection: close

<html>
  <head>
    <title>An Example Page</title>
  </head>
  <body>
    <p>Hello World, this is a very simple HTML document.</p>
  </body>
</html>
```

Step (after TCP Connect + SSL Handshake)

client write http Request text

Client WRITE request text, then flush socket buffer



Line1 =

« GET /path/foo?bar HTTP/1.1 »

Line 2,3...N= http headers:

« key1 : value1 »
« key2 : value2 »
« keyN : valueN »

Empty Line: « »

Then : body... N bytes

« binary-content,
maybe ascii html, json, image, ... »

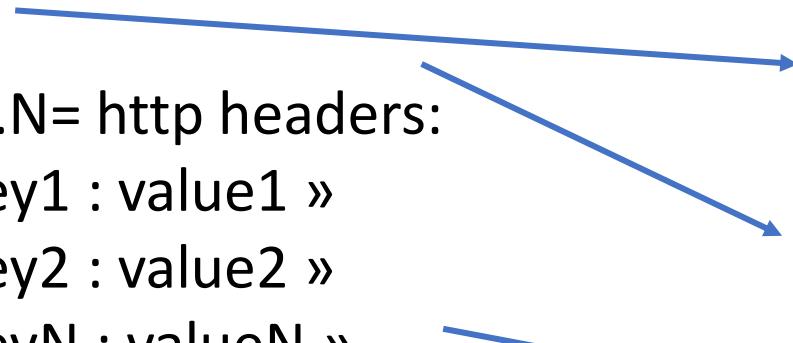
Request = VERB, Path, Headers, Body

Client WRITE request text, then flush socket buffer



Line1 =

« GET /path/foo?bar HTTP/1.1 »



GET, PUT, POST, DELETE, HEAD, PATCH,...
are http « **Method Verbs** »

request Path (and query parameters)
are relative to server host

Empty Line: « »

Then : body... N bytes

« binary-content,
maybe ascii html, json, image, ... »

http Request Headers

some are standards headers: Host, Content-Type, Accept, ..

http Request Body

optional (generally none in GET and DELETE,
one in PUT and POST)

Client WAIT response,
and do not write any more to socket

possible strategies:

- 1/ negotiate in http Header to keep socket open, and re-use
- 2/ half-close the socket on client-side (socket is now read-only)

server write http Response



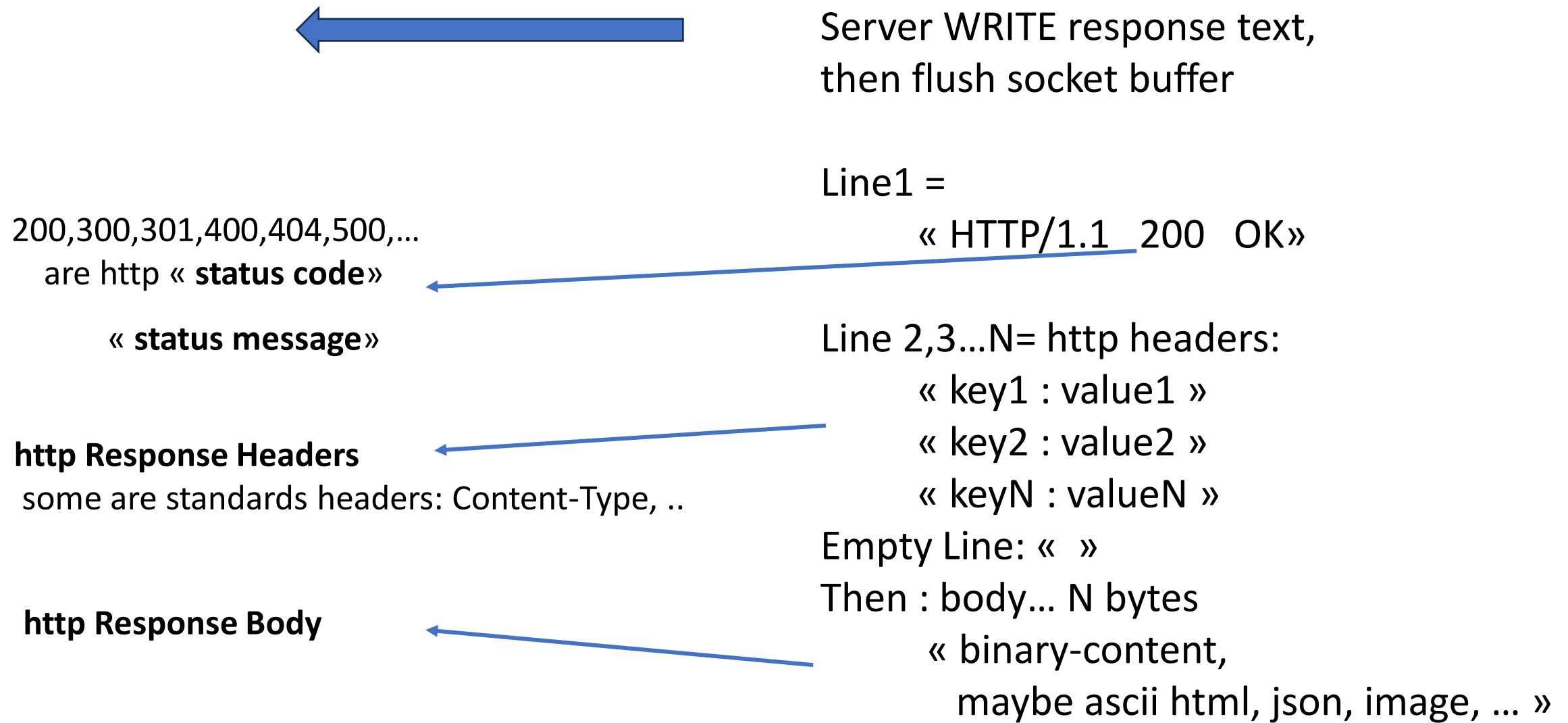
Server WRITE response text,
then flush socket buffer

Line1 =
« HTTP/1.1 200 OK»

Line 2,3...N= http headers:
« key1 : value1 »
« key2 : value2 »
« keyN : valueN »

Empty Line: « »
Then : body... N bytes
« binary-content,
maybe ascii html, json, image, ... »

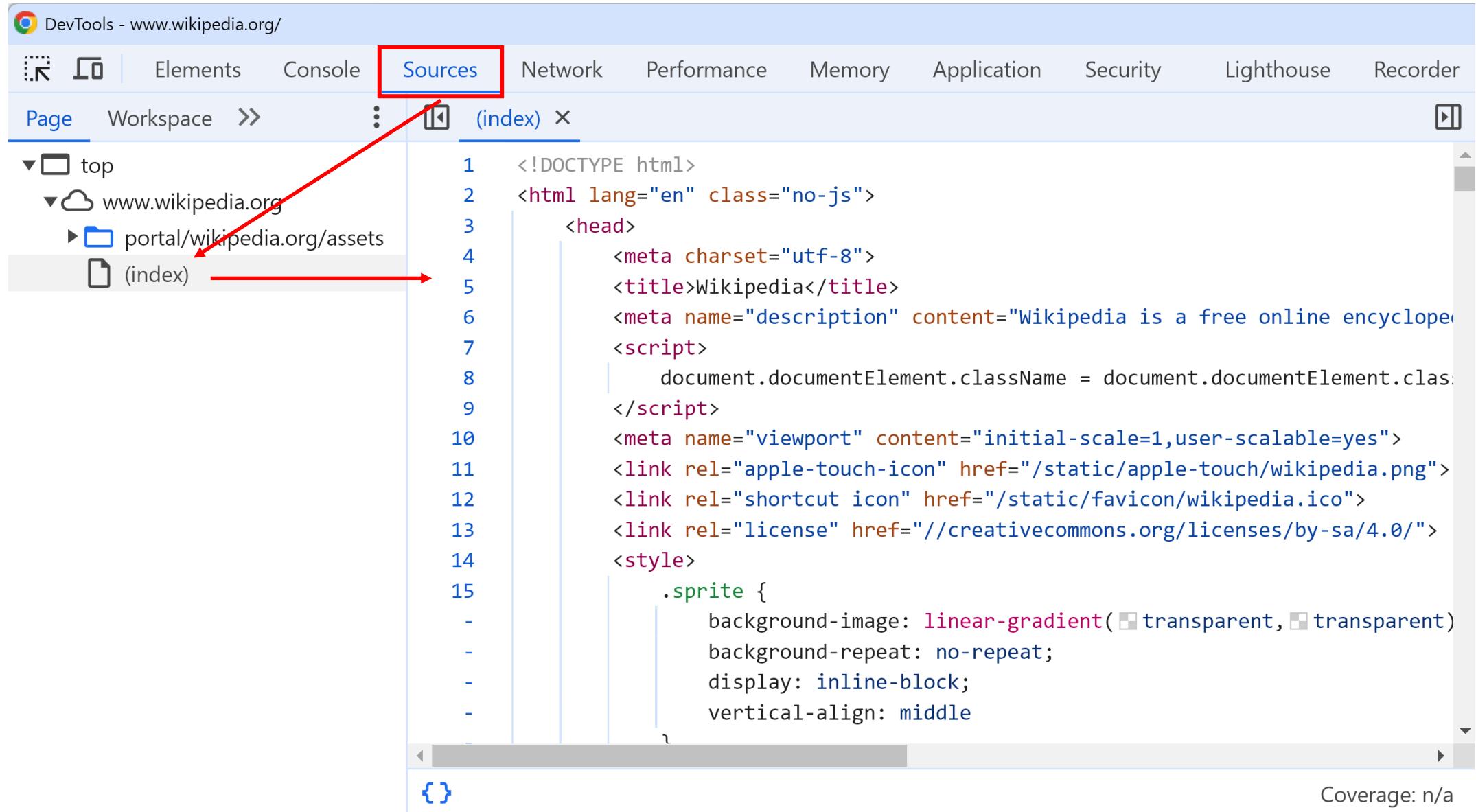
Response = status, message, Headers, Body



example:

<https://www.wikipedia.org/>

Loaded Source Html Page



loading a page = N Requests: html, png, js, svg, css..

example:

<https://www.wikipedia.org/>

The screenshot shows the Network tab in Google Chrome DevTools for the URL <https://www.wikipedia.org/>. A red arrow points from the text above to the 'Network' tab in the top navigation bar. The Network timeline at the top shows several requests: one long green bar (the main document) followed by several smaller bars at 40ms, 60ms, 80ms, 100ms, and 120ms intervals. Below the timeline is a table listing the resources loaded:

Name	Status	Type	Initiator	Size	T...
www.wikipedia.org	304	document	Other	797 B	2...
Wikipedia-logo-v2@2x.png	200	png	(index):30	(memory cache)	0...
index-24c3e2ca18.js	200	script	(index):859	(memory cache)	0...
gt-ie9-ce3fe8e88d.js	200	script	(index):860	(memory cache)	0...
sprite-de847d1a.svg	200	svg+xml	(index):859	(memory cache)	0...
Wikinews-logo_sister@2x.png	200	png	(index):859	(memory cache)	0...
wikipedia.ico	200	vnd.microsoft....	Other	(disk cache)	1...

At the bottom of the DevTools interface, there is a summary: 7 requests | 797 B transferred | 195 kB resources | Finish: 115 ms | DOMContentLoaded: 95 ms | Load: 104 ms. The Console tab is selected in the bottom navigation bar.

How to GET/POST Binary Content ?

Http Requests/Responses have header+body parts:

part 1 in text:

line 1 = VERB (GET/PUT/POST..) + path

line 2, 3, ..N = http header name=value

Requests headers include

"Accept="

"Compression = ..", "Connection = keep-alive"

Responses headers include

"Content-Type="

"Content-Length= <length>"

separator line : "" (empty)

part 2 in any format (byte[])

Content-Length

How can Http knows the full binary content response, then close the socket ?

... By using a Http Header "Content-Length"

Name	X Headers	Preview	Response	Initiator	Timing
www.wikipedia.org	▼General				
Wikipedia-logo-v2@2x.png	Request URL:	https://www.wikipedia.org/portal/wikipedia.org/assets/img/Wikipedia-l ogo-v2@2x.png			
index-24c3e2ca18.js	Request Method:	GET			
gt-ie9-ce3fe8e88d.js	Status Code:	200 OK (from memory cache)			
sprite-de847d1a.svg	Remote Address:	[2a02:ec80:600:ed1a::1]:443			
Wikinews-logo_sister@2x.png	Referrer Policy:	strict-origin-when-cross-origin			
wikipedia.ico	▼Response Headers				
	Accept-Ranges:	bytes			
	Access-Control-Allow-Origin:	*			
	Age:	13120			
	Cache-Control:	s-maxage=86400, max-age=86400			
7 requests 797 B transferred 195 kB resou	Content-Length:	37479			

Debug with Curl

to reproduce Http request manually from "Curl"

The screenshot shows the Chrome DevTools Network tab. The table lists various resources loaded by the page. One resource, 'Wikipedia-logo-v2@2x.png', is selected and has a context menu open. The menu includes options like 'Open in Sources panel', 'Copy URL', 'Copy as cURL (cmd)', and 'Copy as cURL (bash)'. The 'Copy as cURL (bash)' option is highlighted with a red box and a red arrow points to it from the left. Another red box highlights the 'Copy' option in the bottom-left of the menu. The right side of the screen shows the response details for the selected resource.

Name	X Headers	Preview	Response	Initiator	Timing
www.wikipedia.org	▼General				
Wikipedia-logo-v2@2x.png			pedia.org/portal/wikipedia.org		
index-24c3e2ca18.js			memory cache)		
gt-ie9-ce3fe8e88d.js			d1a::1]:443		
sprite-de847d1a.svg			n-cross-origin		
Wikinews-logo_sister@2x.png					
wikipedia.ico					

7 requests | 797 B transferred | 195 ms

Console Rendering Search

Copy as cURL (bash)

Copy

Copy URL

Copy as cURL (cmd)

Copy as PowerShell

Copy as fetch

Copy as fetch (Node.js)

Copy response

Copy all URLs

Copy all as cURL (cmd)

Copy all as cURL (bash)

Copy all as PowerShell

Curl

```
$ curl 'https://www.wikipedia.org/portal/wikipedia.org/assets/img/Wikipedia-logo-v2@2x.png'  
Warning: Binary output can mess up your terminal. Use "--output -" to tell  
Warning: curl to output it to your terminal anyway, or consider "--output  
Warning: <FILE>" to save to a file.  
$  
$ curl 'https://www.wikipedia.org/portal/wikipedia.org/assets/img/Wikipedia-logo-v2@2x.png'  
--output file.png  
% Total    % Received % Xferd  Average Speed   Time     Time      Time  Current  
          Dload  Upload   Total   Spent    Left  Speed  
100 37479  100 37479     0      0  330k      0 --:--:-- --:--:-- --:--:--  355k  
$
```

Curl -v

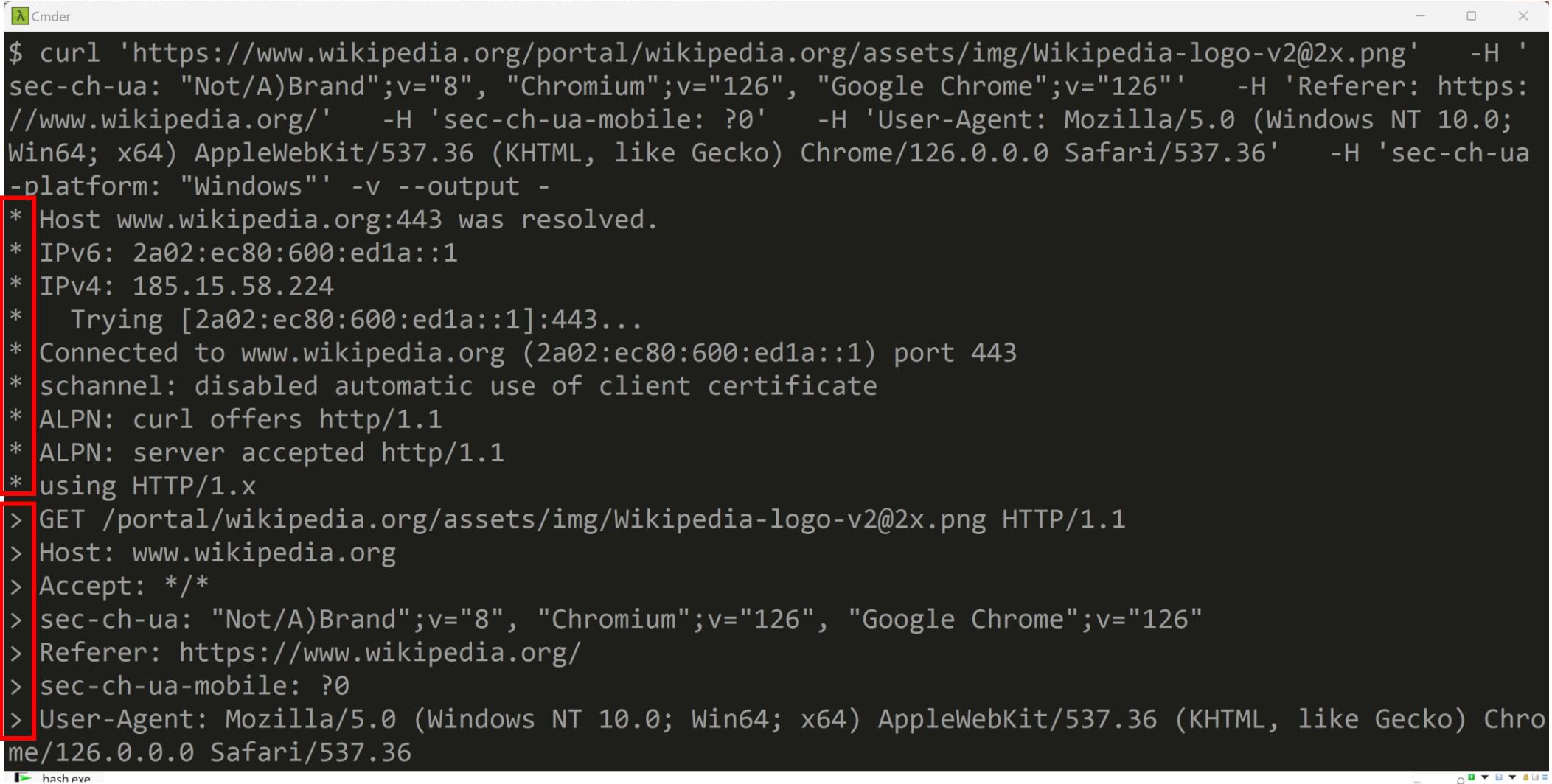
part 1/3: connection "*" + sent ">" Http headers

connection

```
* Host www.wikipedia.org:443 was resolved.  
* IPv6: 2a02:ec80:600:ed1a::1  
* IPv4: 185.15.58.224  
* Trying [2a02:ec80:600:ed1a::1]:443...
```

sent
http headers

```
> GET /portal/wikipedia.org/assets/img/Wikipedia-logo-v2@2x.png HTTP/1.1  
> Host: www.wikipedia.org  
> Accept: */*  
> sec-ch-ua: "Not/A)Brand";v="8", "Chromium";v="126", "Google Chrome";v="126"  
> Referer: https://www.wikipedia.org/  
> sec-ch-ua-mobile: ?0  
> User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/126.0.0.0 Safari/537.36
```



The screenshot shows a terminal window titled 'Cmder' running a curl command to download the Wikipedia logo. The output is divided into two main sections by red boxes:

- connection**: This section contains the initial connection steps, such as resolving the host, trying different IP addresses, and connecting to port 443.
- sent http headers**: This section contains the HTTP headers sent by the client, including the method (GET), path (the logo URL), and various user agent and security headers.

The terminal also shows a taskbar at the bottom with icons for hash.exe and other applications.

Curl -v

part 2/3: "<" received http headers

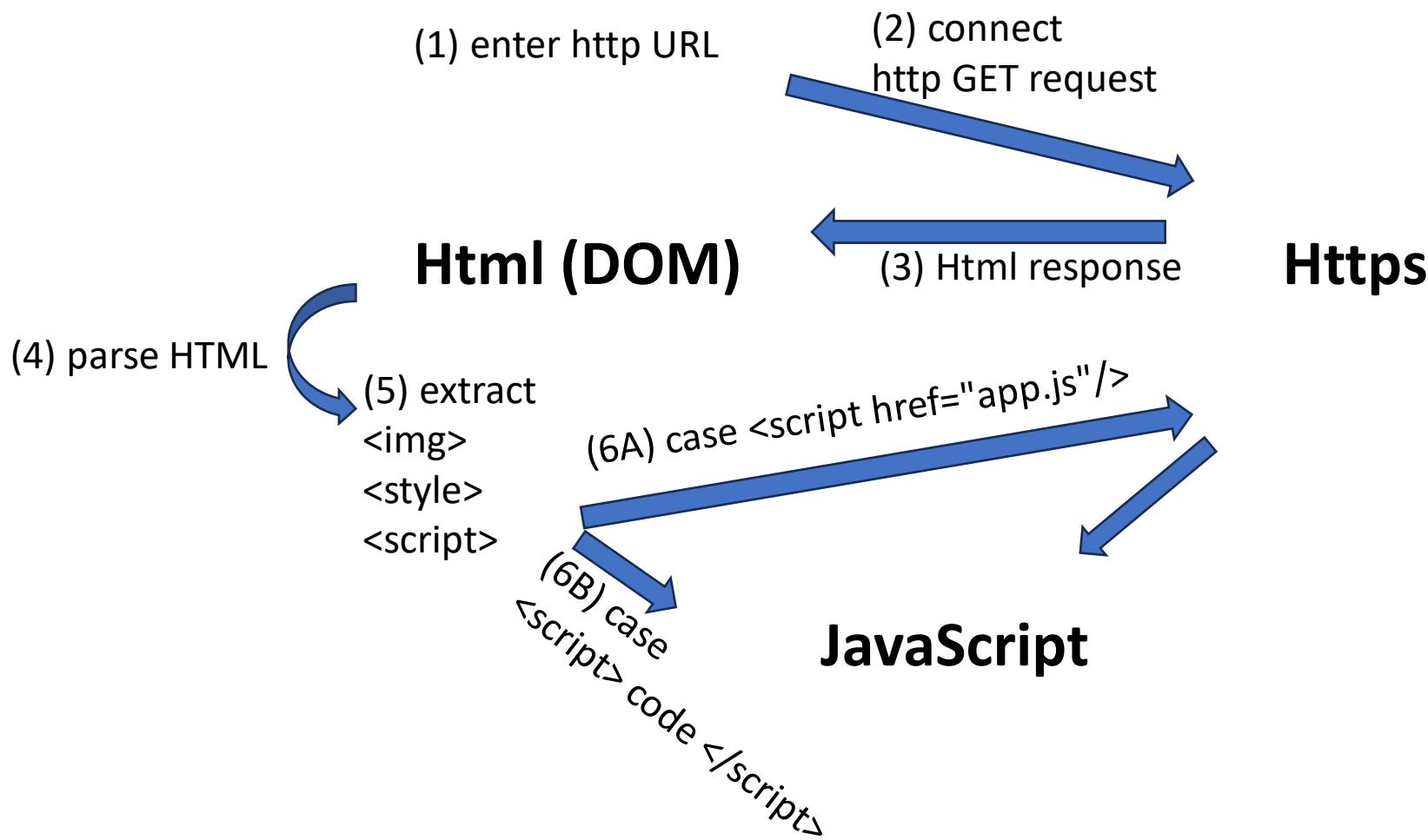
received
http headers

```
* schannel: failed to decrypt data, need more data
* Request completely sent off
< HTTP/1.1 200 OK
< date: Sat, 13 Jul 2024 11:19:03 GMT
< etag: "9267-617c8293c9c40"
< expires: Sat, 12 Jul 2025 12:18:17 GMT
< cache-control: s-maxage=86400, max-age=86400
< server: ATS/9.1.4
< last-modified: Mon, 06 May 2024 12:25:13 GMT
< content-length: 37479
< access-control-allow-origin: *
< content-type: image/png
< age: 16587
< x-cache: cp6016 hit, cp6016 hit/2647
< x-cache-status: hit-front
< server-timing: cache;desc="hit-front", host;desc="cp6016"
< strict-transport-security: max-age=106384710; includeSubDomains; preload
< report-to: { "group": "wm_nel", "max_age": 604800, "endpoints": [{ "url": "https://intake-logging.wikimedia.org/v1/events?stream=w3c.reportingapi.network_error&schema_uri=/w3c/reportingapi/network_error/1.0.0" }] }
< nel: { "report_to": "wm_nel", "max_age": 604800, "failure_fraction": 0.05, "success_fraction": 0.0 }
```

Curl -v

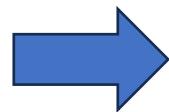
part 3/3: received body

Summary Interaction: Browser URL -> Http -> Loaded Resources



Outline

Https → Load Html + Js + Css + Json + *



Js → Call Http : SPA, Rest + Json

Html → execute Js <script>

Html → trigger events to Js

Js → Query/Update Html DOM

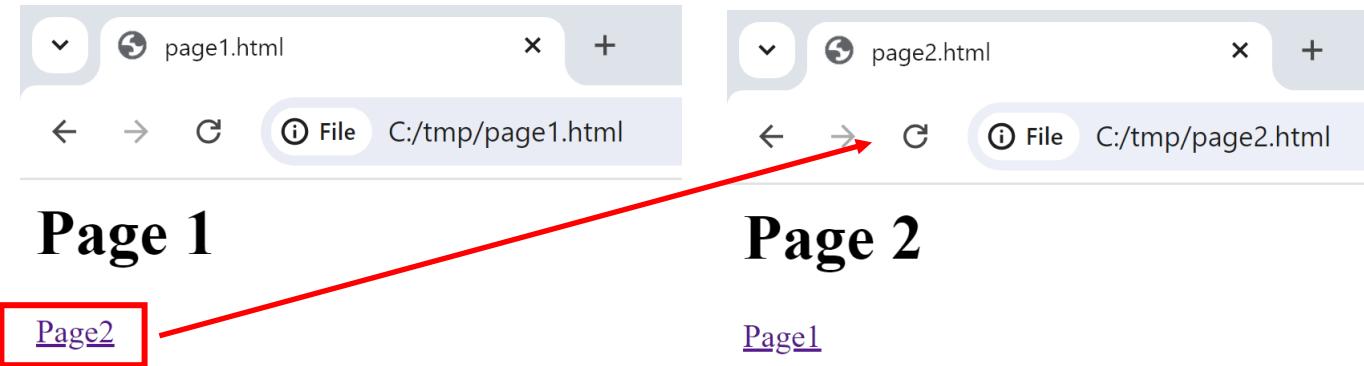
Custom Elements, Shadow DOM, Frameworks

Old Ways to Change Pages

1/ change page with ``



```
1 <!html>
2 ● <body>
3 <H1>Page 1</H1>
4
5 <a href="page2.html">Page2</a>
6
7 </body>
```



Old Ways to Change Refreshed Pages

2/ server-side generate pages

/htdocs / cgi-bin / *.sh

Index of /

Name	Last modified	Size	Description
cgi-bin/	2017-10-09 03:04	-	



BASH
THE BOURNE AGAIN SHELL



*.SP

node js
express



Java Servlet & Jsp

(html in java / java in pseudo-html)



Servlets

= plain old java code printing « html » text

```
void toHtml(ServletOutputStream out) {  
    if (cond) {  
        out.print("<div> <span \"attr\"=\"v1\"> ..");  
    } else {  
        out.print("<div> <span \"attr\"=\"v2\"> ..");  
    }  
  
    out.print( new Date() );  
}
```



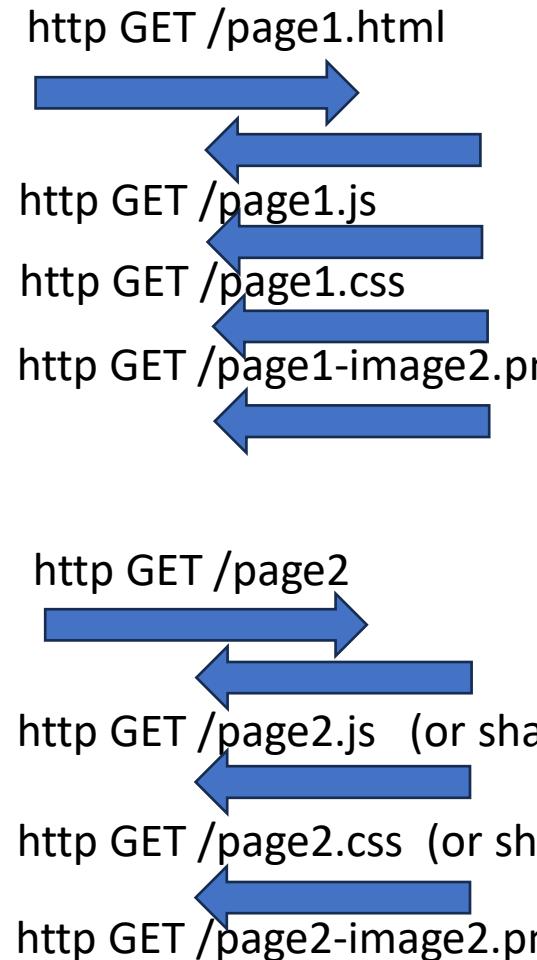
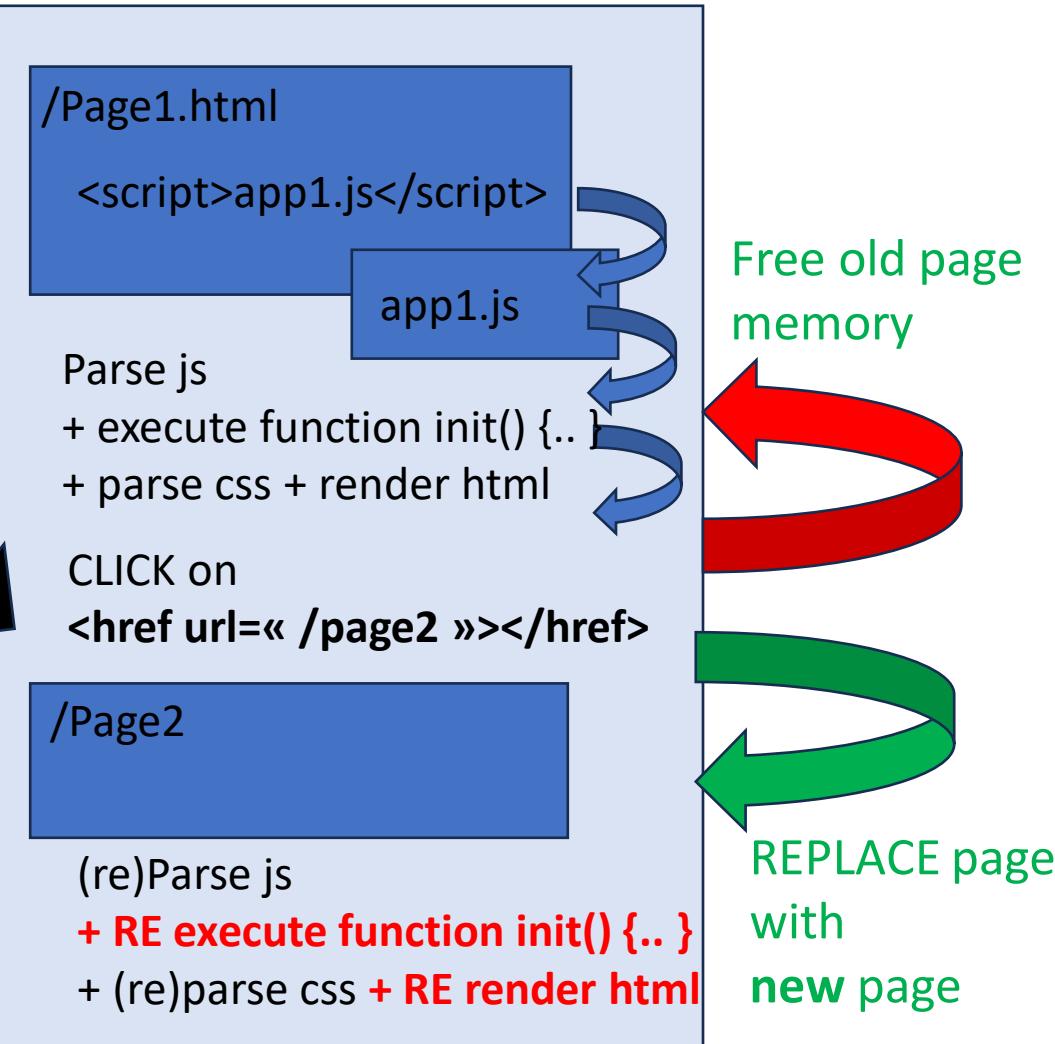
= pseudo html/xml/markup code containing java

```
<% if (cond) { %>  
    <div><span "attr"="v1"> .. </span></div>  
<% } else { %>  
    <div><span "attr"="v1"> .. </span></div>  
<% } %>
```

```
<%= new Date() %>
```

Click <href> ... RE-Loading new html page

Web Browser



Dynamic Server Pages Server

Not only a Developper problem...
Html+Js+CSS re-Rendering slowness + Network + ..
for reloading pages links

SLOW ... SLOW

NETWORK Bandwidth CONSUMING
+ Network Latency
+ CPU CONSUMING

Solution to
Multiple Pages Application Problems ?

caching + optims + workarounds ... ?

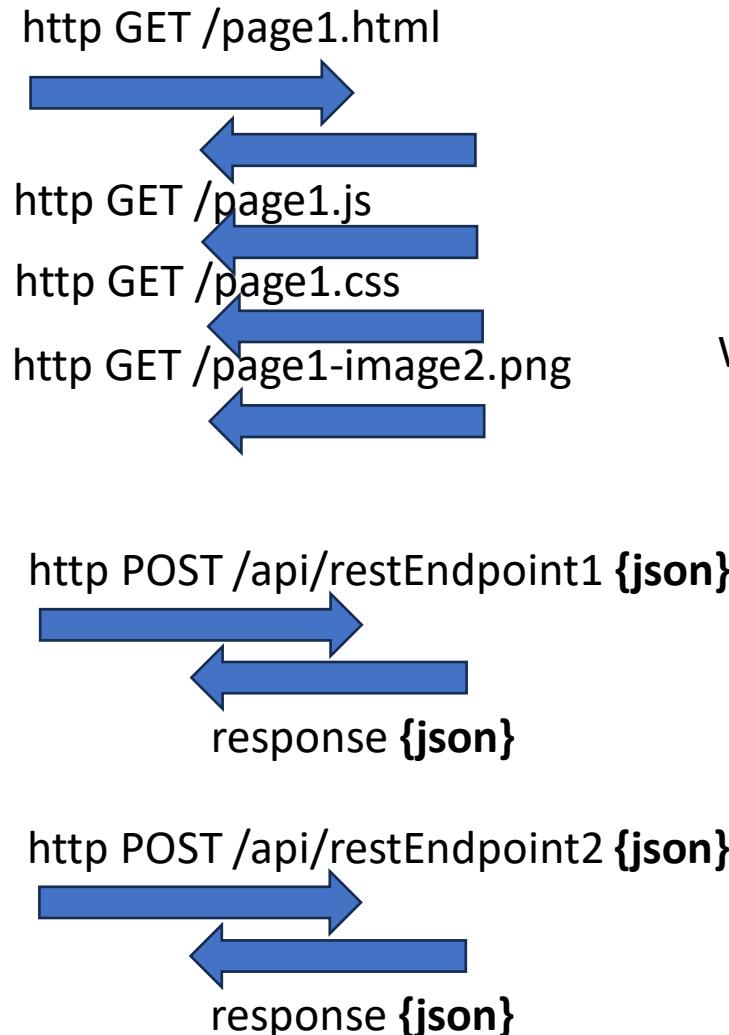
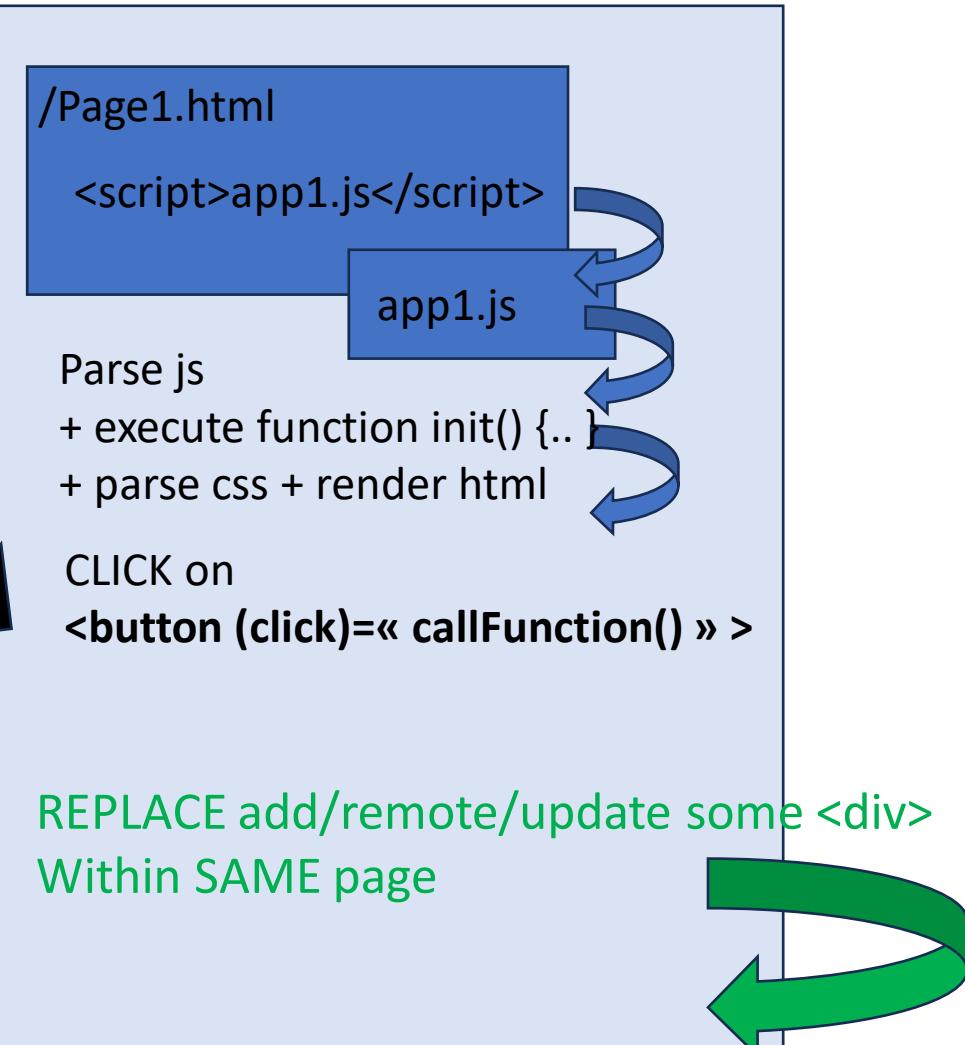
=>

SPA = **Single** Page Application !!

SPA = Single Page Application

... still dynamic, still Multiple Rest Json requests

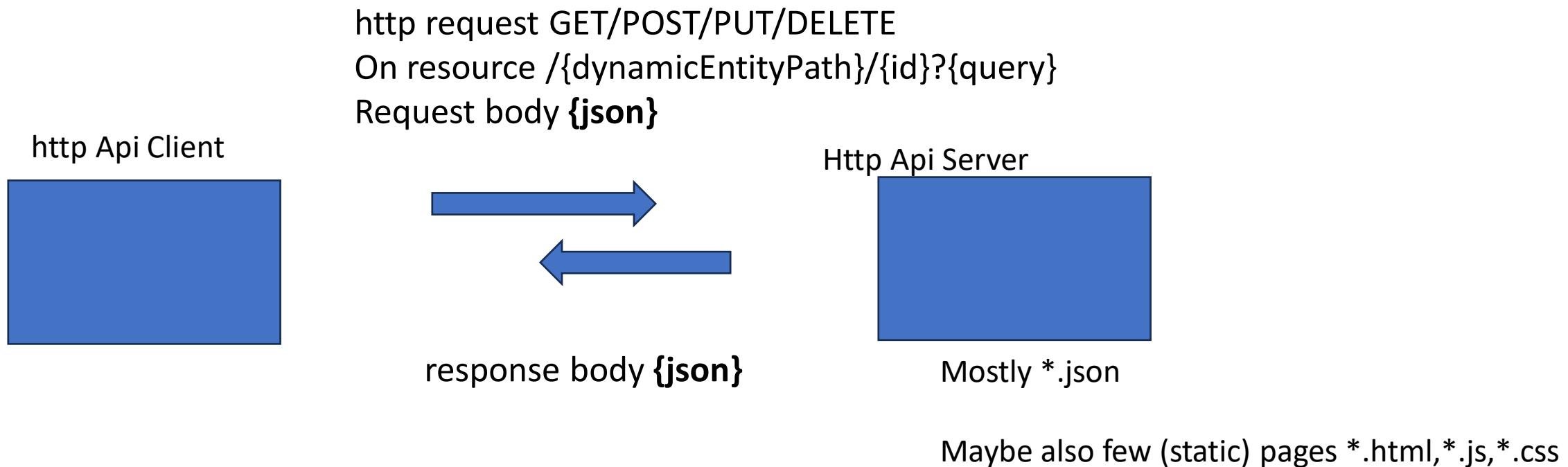
Web Browser



Web Server

http API Server (cf Rest Api)

Terminology: **Http Api Server**



How to Code High-Level Http Request in JavaScript ?

Spoiler Alert: using non-blocking Async IO
... more complex than regular system calls on Socket

Request as "Blocking" (Low-Level) Code / SystemCalls

Code (ex: java)

```
Socket s = // new Socket(...)  
          SSLSocketFactory.getDefault().createSocket(..)  
s.connect(...)
```

```
OutputStream out = s.getOutputStream()  
out.write("GET / HTTP 1.0\n")  
out.flush()
```

```
InputStream in = s.getInputStream()  
byte[] buffer = new byte[4096]  
int readCount = in.read(buffer)
```

```
s.close()
```

System Calls

← **connect()**

← **write()**

← **flush()**

← **read()**

← **close()**

all blocking calls

(thread/process suspended to wait)

In the Browser (Javascript Web Spec)
... only 1 "event loop" main Thread

JavaScript API from the Web Browser
implement ONLY
High-Level
NON-Blocking calls
(with Callbacks or Promises)

XML(?) XMLHttpRequest

developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest

mdn web docs References Guides Plus Curriculum Blog Tools NEW Theme

References > Web APIs > XMLHttpRequest

Filter

XMLHttpRequest

XMLHttpRequest API

XMLHttpRequest

Constructor XMLHttpRequest()

Instance properties

- channel ⚠
- mozAnon ⚠
- mozBackgroundRequest ⚠

Note: This feature is available in [Web Workers](#), except for [Service Workers](#).

XMLHttpRequest (XHR) objects are used to interact with servers. You can retrieve data from a URL without having to do a full page refresh. This enables a Web page to update just part of a page without disrupting what the user is doing.

```
graph LR; EventTarget --> XMLHttpRequestEventTarget; XMLHttpRequestEventTarget --> XMLHttpRequest
```

Despite its name, XMLHttpRequest can be used to retrieve any type of data, not just XML.



ChatGPT ▾



Explore GPTs

Today

HTTP Client Request Example ...

javascript

```
// Create a new XMLHttpRequest object
var xhr = new XMLHttpRequest();

// Configure it: GET-request for the URL /article/.../load
xhr.open('GET', 'https://jsonplaceholder.typicode.com/posts/1', true);
```

Sample Code

give me sample javascript for http request using XMLHttpRequest

```
xhr.onreadystatechange = function () {
    // Check if the request is complete
    if (xhr.readyState === 4) {
        // Check if the request was successful
        if (xhr.status === 200) {
            // Parse the JSON response
            var response = JSON.parse(xhr.responseText);
            // Update the content div with the fetched data
            var contentDiv = document.getElementById('content');
            contentDiv.innerHTML =
                <h2>${response.title}</h2>
                <p>${response.body}</p>
            `;
        } else {
            // Handle the error
            console.error('Error fetching data: ' + xhr.statusText);
        }
    };
};

// Send the request
xhr.send();
```

Callback Principles for Async

```
function asyncFunc( input, resultCallback ) {  
    ...  
}  
  
yourApp() {  
    asyncFunc(..., resultCallback ); // <= return immediatly,  
                                // and will call later: resultCallback( ...someResult )  
  
    ... // next... called before resultCallback  
}  
  
function resultCallback( res ) {  
    ...  
}
```

Promises for Async

```
function asyncFunc( input ) { // return Promise
    ...
}

yourApp() {
    var promise = asyncFunc(...); // <= return immediately, will put result in promise later
    promise.then(resultCallback);
    promise.catch(errorCallback);

    ... // next... called before resultCallback
}

function resultCallback( res ) { ... }
function errorCallback( res ) { ... }
```

Equivalent "Cascaded Functional" Code

```
yourApp() {  
    asyncFunc(...) // <= return immediately, will put result in promise later  
    .then( res => { // inlined lambda  
        ...  
    })  
    .catch( error => { // inlined lambda, when failure  
        ...  
    });  
  
    ... // next... called before lambda  
}
```

Equivalent Code using Angular RxJs Observable (deprecated: using 2 callbacks)

```
yourApp() {  
    asyncFunc(...) // <= return immediately, will put result in promise later  
    .subscribe(res => { // inlined lambda  
        ...  
    }, error => { // inlined lambda, when failure  
        ...  
    } );  
  
    ... // next... called before lambda  
}
```

Equivalent Code using Angular RxJs Observable (1 object with 2 callbacks)

```
yourApp() {
  asyncFunc(...) // <= return immediately, will put result in observable later
  .subscribe({
    next: res => { .. }, // inlined lambda
    error: error => {.. } // inlined lambda, when failure
  });
  ...
  // next... called before lambda
}
```

The screenshot shows the RxJS API documentation for the `Observer` interface. The URL in the browser bar is `rxjs.dev/api/index/interface/Observer`. The page has a red header with the RxJS logo and navigation links for OVERVIEW, REFERENCE, and TEAM. The left sidebar lists various sections: OVERVIEW, Observables, Observer (which is highlighted), Operators, Subscription, Subjects, Scheduler, Marble Testing, and INSTALLATION. The right panel contains the `Observer` interface definition:

```
interface Observer<T> {
  next: (value: T) => void
  error: (err: any) => void
  complete: () => void
}
```

Http Request as Async (=Non-Blocking) Code

The screenshot shows the ChatGPT interface. In the top bar, there are icons for a window, a refresh, and the text "ChatGPT". Below the bar, there are two tabs: "ChatGPT" and "Explore GPTs". On the left side, there's a sidebar with "Today" and "HTTP Client Request Example ...". The main area has a message from ChatGPT: "give me sample javascript code for http client request". ChatGPT responds: "Sure! Here are some sample JavaScript codes for making HTTP client requests using various popular methods:

Using Fetch API

The Fetch API is a modern way to make HTTP requests. It is built into the browser and is very straightforward to use.

GET Request

```
javascript Copy code
fetch('https://jsonplaceholder.typicode.com/posts')
  .then(response => response.json()) // Parse the JSON from the response
  .then(data => console.log(data)) // Log the data to the console
  .catch(error => console.error('Error:', error)); // Log any errors
```

test-fetch.html

C:\tmp\test-fetch.html - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

test-fetch.html

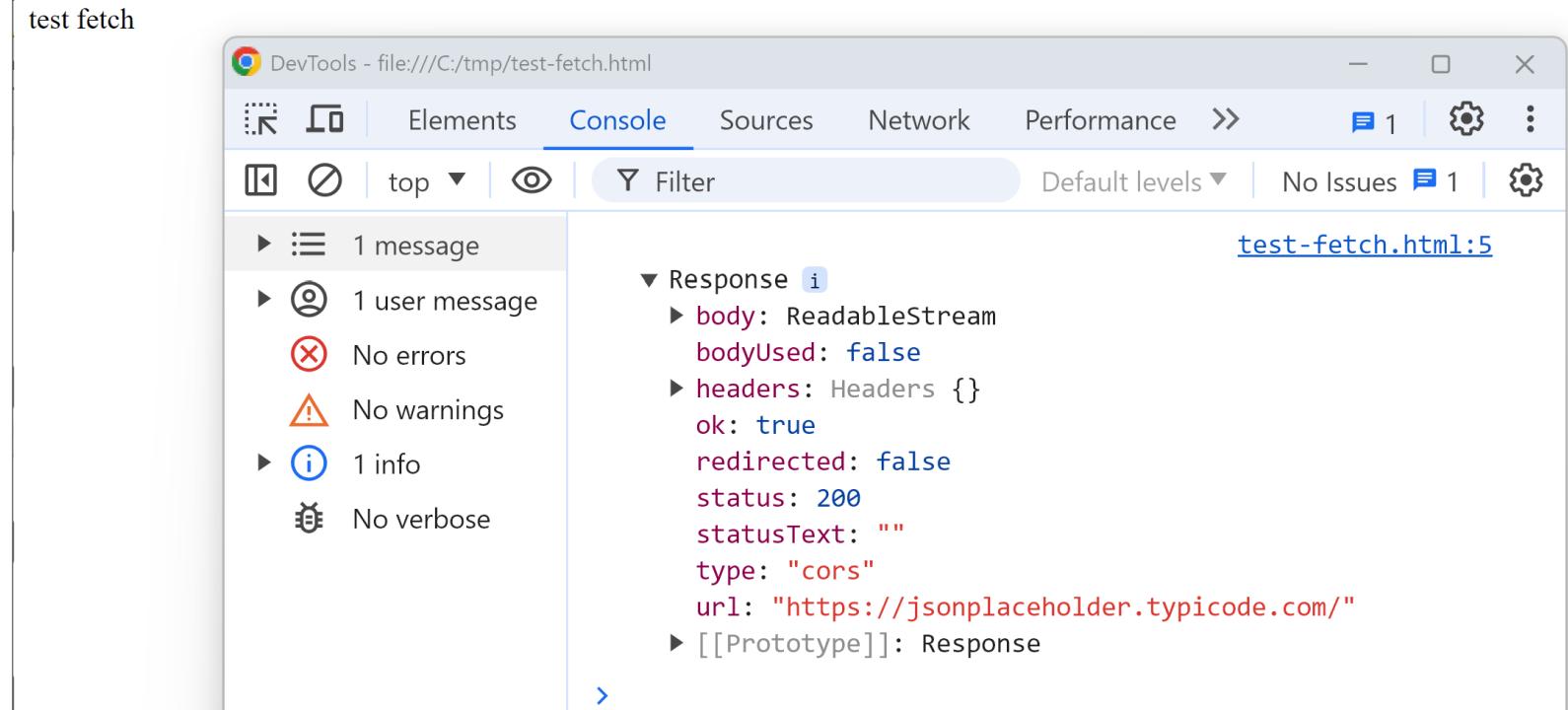
```
1 <!html>
2 <head>
3   <script>
4     fetch("https://jsonplaceholder.typicode.com/")
5       .then(resp => console.log(resp));
6   </script>
7 </head>
8 <body>
9   test fetch
10 </body>
```

(1/3) Edit Html code, save file

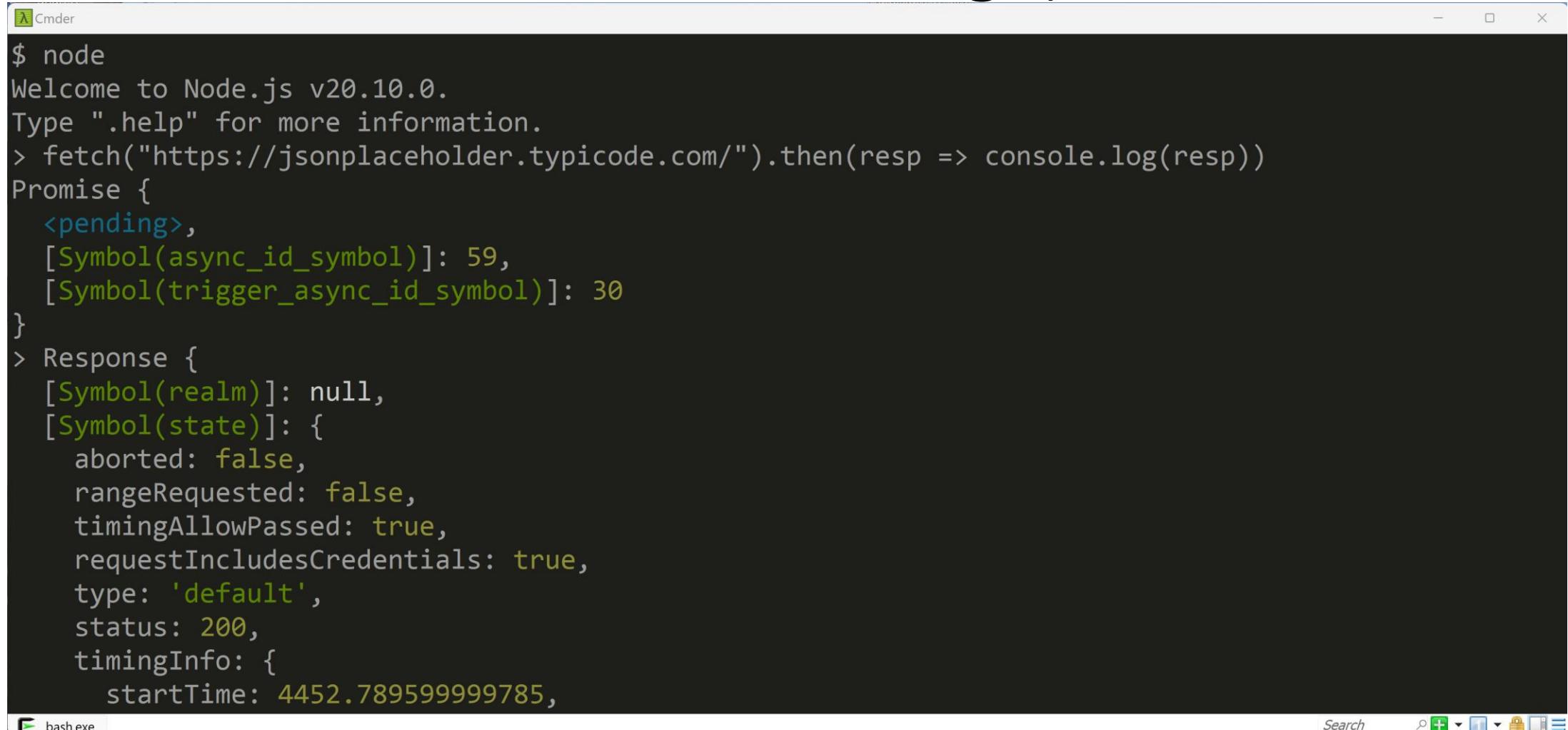


(2/3) open browser

(3/3) Open DevTools (F12)
open Console



using NodeJS (same Javascript interpreter V8 as Chrome/Edge)



A screenshot of a Cmder terminal window titled "Cmder". The terminal is running a Node.js session. The output shows the Node.js welcome message, a fetch request to jsonplaceholder.typicode.com, and the resulting Response object.

```
$ node
Welcome to Node.js v20.10.0.
Type ".help" for more information.
> fetch("https://jsonplaceholder.typicode.com/")
Promise {
  <pending>,
  [Symbol(async_id_symbol)]: 59,
  [Symbol(trigger_async_id_symbol)]: 30
}
> Response {
  [Symbol(realm)]: null,
  [Symbol(state)]: {
    aborted: false,
    rangeRequested: false,
    timingAllowPassed: true,
    requestIncludesCredentials: true,
    type: 'default',
    status: 200,
    timingInfo: {
      startTime: 4452.789599999785,
```



ChatGPT

Explore GPTs

Today

Angular REST HTTP Example ...

sample angular code for rest http json call

typescript

Copy code

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class DataService {

  private apiUrl = 'https://jsonplaceholder.typicode.com/posts'; // Sample API URL

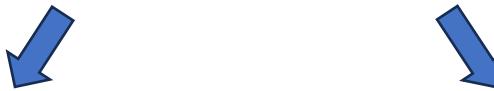
  constructor(private http: HttpClient) { }

  getData(): Observable<any> {
    return this.http.get<any>(this.apiUrl);
  }
}
```

```
this.dataService.getData().subscribe(
  (response) => {
    this.data = response;
  },
  (error) => {
    console.error('Error fetching data', error);
  }
);
```

Easy Interactions between Javascript / Json !

JavaScript Json
or TypeScript (= Javascript value)



```
var jsonRequestBody = { json: "is data" };
```

```
this.httpClient.post("url", jsonRequestBody)
```



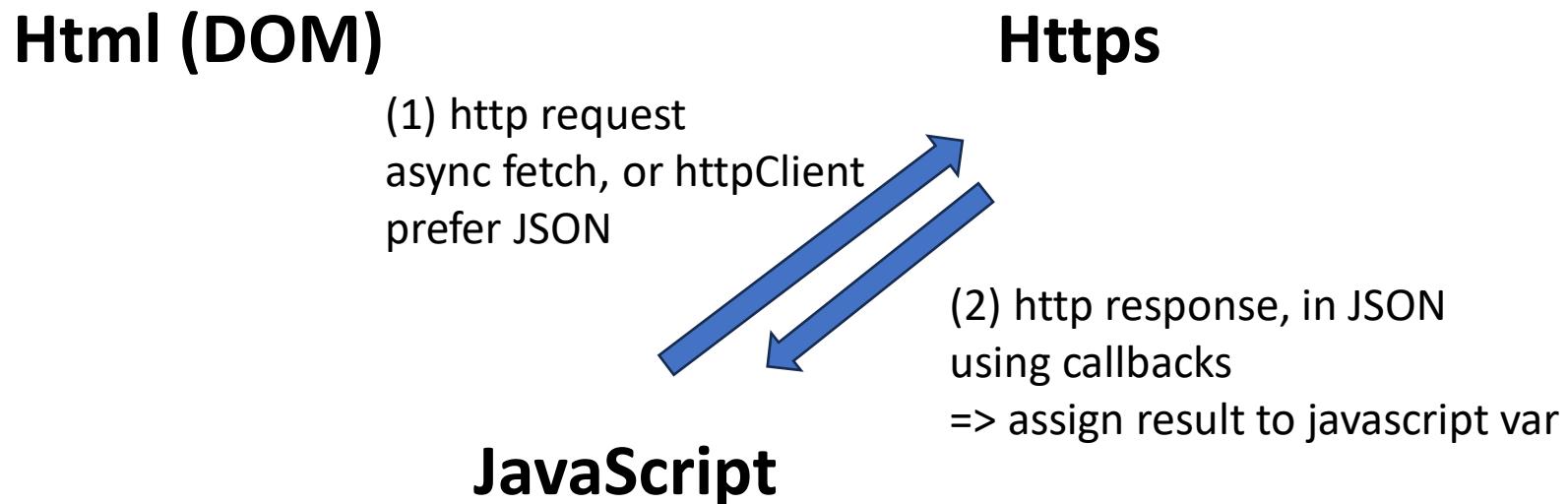
JS -> Json -> Http EASY

```
.subscribe( jsonResponseBody => {  
    this.jsResult = jsonResponseBody;  
})
```



Http -> Json -> Js EASY

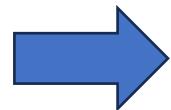
Summary Interaction: JavaScript call (Async) Http Rest JSON Api



Outline

Https → Load Html + Js + Css + Json + *

Js → Call Http : SPA, Rest + Json



Html → execute Js <script>

Html → trigger events to Js

Js → Query/Update Html DOM

Custom Elements, Shadow DOM, Frameworks

<script>..</script> & <script src=".." />

The screenshot shows a code editor with two tabs: "test-scripts.html" and "script2.js".

test-scripts.html:

```
1  <!html>
2  <head>
3  <script>
4      console.log("start script1...");
5
6      function f1() { console.log("call f1()"); }
7
8      console.log(.. done script1);
9  </script>
10
11 <script src="script2.js"></script>
12
13
14
15 </head>
16 <body>
17     test scripts
18
19     <script>
20         console.log("start script3...");
21
22         function f3() { console.log("call f3()"); }
23
24         f1();
25         f2();
26
27         console.log(.. done script3);
28     </script>
29
30 </body>
```

script2.js:

```
1  console.log("start script2...");
2
3  function f2() { console.log("call f2()"); }
4
5  console.log(.. done script2);
```

embedded <script> in head

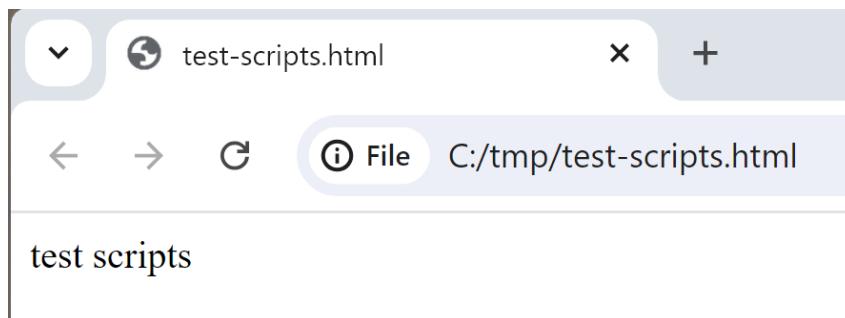
external <script src=".." />

The screenshot shows the content of the "script2.js" file.

```
1  console.log("start script2...");
2
3  function f2() { console.log("call f2()"); }
4
5  console.log(.. done script2);
```

embedded <script> in body

Sample <script>



DevTools - file:///C:/tmp/test-scripts.html

Sources

Page Workspace > :

top
file:///
C:/tmp

Scope Watch

Breakpoints

Pause on uncaught exceptions
 Pause on caught exceptions

Console Rendering Search

Filter Default levels 1 Issue: 1 Issue:

8 messages 8 user messages No errors No warnings 8 info No verbose

start script1... [test-scripts.html:4](#)
.. done script1 [test-scripts.html:8](#)
start script2... [script2.js:1](#)
.. done script2 [script2.js:5](#)
start script3... [test-scripts.html:21](#)
call f1() [test-scripts.html:6](#)
call f2() [script2.js:3](#)
.. done script3 [test-scripts.html:28](#)

<script (async | defer) src="..">

with many <script>, load (+parse) in parallel using async

see https://www.w3schools.com/tags/tag_script.asp

Attribute	Value	Description
<u>async</u>	async	Specifies that the script is downloaded in parallel to parsing the page, and executed as soon as it is available (before parsing completes) (only for external scripts)
<u>crossorigin</u>	anonymous use-credentials	Sets the mode of the request to an HTTP CORS Request
<u>defer</u>	defer	Specifies that the script is downloaded in parallel to parsing the page, and executed after the page has finished parsing (only for external scripts)
<u>integrity</u>	filehash	Allows a browser to check the fetched script to ensure that the code is never loaded if the source has been manipulated
<u>nomodule</u>	True False	Specifies that the script should not be executed in browsers supporting <u>ES2015</u> modules
<u>referrerpolicy</u>	no-referrer no-referrer-when-downgrade origin origin-when-cross-origin same-origin strict-origin strict-origin-when-cross-origin unsafe-url	Specifies which referrer information to send when fetching a script
<u>src</u>	URL	Specifies the URL of an external script file
<u>type</u>	scripttype	Specifies the media type of the script

window.onload = callbackReference;

```
1  <!html>
2   <head>
3     <script>
4       console.log("start script1...");
5
6       window.onload = function() {
7         console.log("window.onload()");
8       };
9
10      console.log(.. done script1");
11    </script>
12
13    <script async src="async-script2.js"></script>
14    <script defer src="defer-script3.js"></script>
15
16  </head>
17  <body>
18    test scripts
19  </body>
```

Filter

start script1...

[Default levels](#) | 1 Issue: 1 |

[test-document-onload.html:4](#)

.. done script1

[test-document-onload.html:10](#)

start async-script2...

[async-script2.js:1](#)

.. done async-script2

[async-script2.js:5](#)

start defer-script3...

[defer-script3.js:1](#)

.. done defer-script3

[defer-script3.js:5](#)

window.onload()

[test-document-onload.html:7](#)

Window ... global singleton

Can NOT set several ".onload" callbacks !

```
test-window-onload-overrides.html ✘
1  <!html>
2   <head>
3     <script>
4       console.log("start script1...");
5
6       window.onload = function() {
7         console.log("window.onload1() .... NOT CALLED");
8       };
9
10    window.onload = function() {
11      console.log("window.onload2()");
12    };
13
14    console.log(.. done script1);
15  </script>
16
17  </head>
18  <body>
19    test scripts
20  </body>
```

Y Filter

start script1...

.. done script1

window.onload2()

Default levels ▾

1 Issue: 1



[test-window-onload-overrides.html:4](#)

[test-window-onload-overrides.html:14](#)

[test-window-onload-overrides.html:11](#)

<body onload="someCode()">

```
1  <!html>
2  <head>
3  <script>
4      console.log("start script1...");
5
6      function bodyOnLoad() {
7          console.log("bodyOnLoad()");
8      }
9
10     console.log(.. done script1");
11 </script>
12
13 </head>
14 <body onload="bodyOnLoad()">
15     test scripts
16 </body>
```

Filter

Default levels ▾

1 Issue: 1



start script1...

[test-elements-onload.html:4](#)

.. done script1

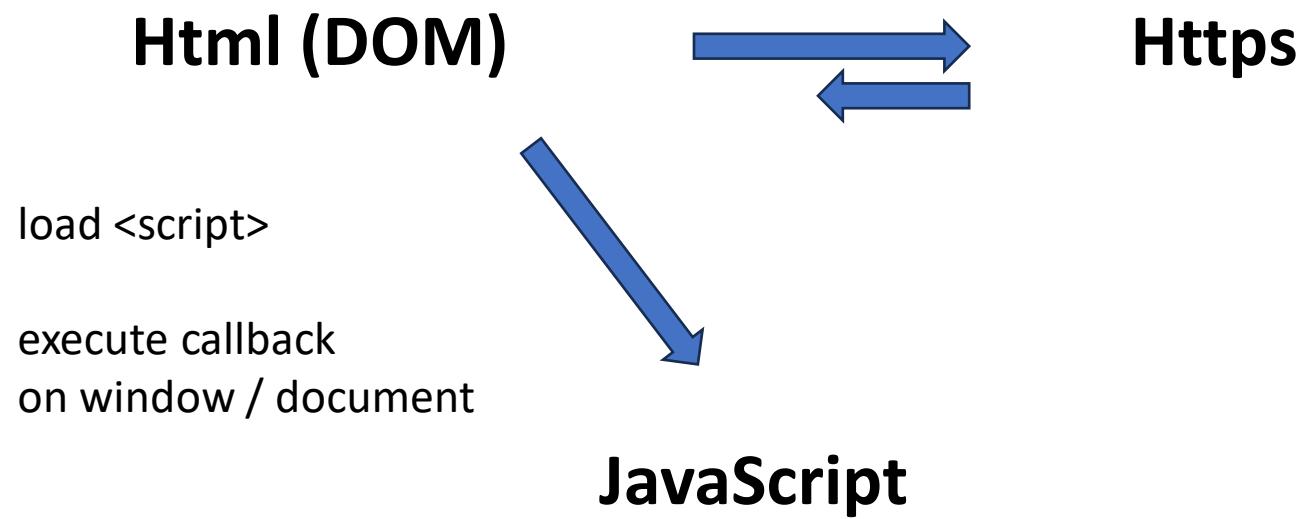
[test-elements-onload.html:10](#)

bodyOnLoad()

[test-elements-onload.html:7](#)



Summary Interaction: Html execute <script>

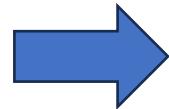


Outline

Https → Load Html + Js + Css + Json + *

Js → Call Http : SPA, Rest + Json

Html → execute Js <script>



Html → trigger events to Js

Js → Query/Update Html DOM

Custom Elements, Shadow DOM, Frameworks

w3schools Javascript > JS Events

← → ⌂

w3schools.com/js/js_events.asp



Tutorials ▾

Exercises ▾

Certificates ▾

Services ▾

Search...



HTML CSS

JAVASCRIPT

SQL

PYTHON

JAVA

PHP

HOW TO

W3.CSS

C

C++

C#

BOOTSTRAP REA

JS Output

JS Statements

JS Syntax

JS Comments

JS Variables

JS Let

JS Const

JS Operators

JS Arithmetic

JS Assignment

JS Data Types

JS Functions

JS Objects

JS Object Properties

JS Object Methods

JS Object Display

JS Object Constructors

JS Events

JS Strings

JS String Methods

JS String Search

JS String Templates

JS Numbers

JS BigInt

JS Number Methods

JS Number Properties

JS Arrays

JS Array Methods

JavaScript Events

◀ Previous

HTML events are "**things**" that happen to HTML elements.

When JavaScript is used in HTML pages, JavaScript can "**react**" on these events.

HTML Events

An HTML event can be something the browser does, or something a user does.

Here are some examples of HTML events:

- An HTML web page has finished loading
- An HTML input field was changed
- An HTML button was clicked

Often, when events happen, you may want to do something.

JavaScript lets you execute code when events are detected.

HTML allows event handler attributes, **with JavaScript code**, to be added to HTML elements.

With single quotes:

```
<element event='some JavaScript'>
```

w3schools HTML Events

← → ×  w3schools.com/tags/ref_eventattributes.asp

Tutorials ▾ Exercises ▾ Certificates ▾ Services ▾ Search...  

HTML CSS JAVASCRIPT SQL PYTHON JAVA PHP HOW TO W3.CSS C C++ C# BOOTSTRAP REACT MYSQL JQUERY

HTML References

- [HTML by Alphabet](#)
- [HTML by Category](#)
- [HTML Browser Support](#)
- [HTML Attributes](#)
- [HTML Global Attributes](#)

HTML Events

- [HTML Colors](#)
- [HTML Canvas](#)
- [HTML Audio/Video](#)
- [HTML Character Sets](#)
- [HTML Doctypes](#)
- [HTML URL Encode](#)
- [HTML Language Codes](#)
- [HTML Country Codes](#)
- [HTTP Messages](#)
- [HTTP Methods](#)
- [PX to EM Converter](#)
- [Keyboard Shortcuts](#)

HTML Tags

- [<!-->](#)
- [<!DOCTYPE>](#)
- [<a>](#)
- [<abbr>](#)

Global Event Attributes

HTML has the ability to let events trigger actions in a browser, like starting a JavaScript when a user clicks on an element.

To learn more about programming events, please visit our [JavaScript tutorial](#).

Below are the global event attributes that can be added to HTML elements to define event actions.

Window Event Attributes

Events triggered for the window object (applies to the <body> tag):

Attribute	Value	Description
<u>onafterprint</u>	<i>script</i>	Script to be run after the document is printed
<u>onbeforeprint</u>	<i>script</i>	Script to be run before the document is printed
<u>onbeforeunload</u>	<i>script</i>	Script to be run when the document is about to be unloaded
<u>onerror</u>	<i>script</i>	Script to be run when an error occurs
<u>onhashchange</u>	<i>script</i>	Script to be run when there has been changes to the anchor part of the a URL
<u>onload</u>	<i>script</i>	Fires after the page is finished loading
<u>onmessage</u>	<i>script</i>	Script to be run when the message is triggered
<u>onoffline</u>	<i>script</i>	Script to be run when the browser starts to work offline

Types of Events

Window Event Attributes

Events triggered for the window object (applies to the <body> tag):

[onafterprint](#)
[onbeforeprint](#)
[onbeforeunload](#)
[onerror](#)
[onhashchange](#)
[onload](#)
[onmessage](#)
[onoffline](#)
[ononline](#)
[onpagehide](#)
[onpageshow](#)
[onpopstate](#)
[onresize](#)
[onstorage](#)
[onunload](#)

Mouse Events

[onclick](#)
[ondblclick](#)
[onmousedown](#)
[onmousemove](#)
[onmouseout](#)
[onmouseover](#)
[onmouseup](#)
[onmousewheel](#)
[onwheel](#)

Misc Events

[ontoggle](#)

Drag Events

[ondrag](#)
[ondragend](#)
[ondragenter](#)
[ondragleave](#)
[ondragover](#)
[ondragstart](#)
[ondrop](#)
[onscroll](#)

Clipboard Events

[oncopy](#)
[oncut](#)
[onpaste](#)

Form Events

Events triggered by actions inside a HTML form

[onblur](#)
[onchange](#)
[oncontextmenu](#)
[onfocus](#)
[oninput](#)
[oninvalid](#)
[onreset](#)
[onsearch](#)
[onselect](#)
[onsubmit](#)

Media Events

Events triggered by medias like videos, images and audio

[onabort](#)
[oncanplay](#)
[oncanplaythrough](#)
[oncuechange](#)
[ondurationchange](#)
[onemptied](#)
[onended](#)
[onerror](#)
[onloadeddata](#)
[onloadedmetadata](#)
[onloadstart](#)
[onpause](#)
[onplay](#)
[onplaying](#)
[onprogress](#)
[onratechange](#)
[onseeked](#)
[onseeking](#)
[onstalled](#)
[onsuspend](#)
[ontimeupdate](#)
[onvolumechange](#)
[onwaiting](#)

HTML "onclick" Event Attribute (attribute, with value executed as script)

The screenshot shows the W3Schools website interface. The top navigation bar includes links for Tutorials, Exercises, Certificates, Services, a search bar, and a help icon. Below the bar, a secondary navigation menu lists various programming languages: HTML, CSS, JAVASCRIPT, SQL, PYTHON, JAVA, PHP, HOW TO, W3.CSS, C, C++, and C#. On the left side, there's a sidebar with 'HTML References' containing links like 'HTML by Alphabet', 'HTML by Category', 'HTML Browser Support', 'HTML Attributes', etc., and a 'HTML Tags' section. The main content area has a title 'HTML onclick Event Attribute' with a back-link '◀ HTML Event Attributes'. It features an 'Example' section with the code `<button onclick="myFunction()">Click me</button>`, a 'Try it Yourself' button, and a note about more examples below. The bottom section is titled 'Definition and Usage' with a descriptive paragraph.

HTML onclick Event Attribute

◀ HTML Event Attributes

Example

Execute a JavaScript when a button is clicked:

```
<button onclick="myFunction()">Click me</button>
```

[Try it Yourself »](#)

More "Try it Yourself" examples below.

Definition and Usage

The onclick attribute fires on a mouse click on the element.

onclick

```
test-onclick.html
1 <!html>
2 <head>
3   <script>
4     function myOnClick() {
5       console.log("myOnClick");
6     }
7   </script>
8
9 </head>
10 <body>
11   test button on click
12   <br/>
13   <button onclick="myOnClick()"> Click </button>
14
15 </body>
```



test button on click

Click



onclick event

```
1 <!html>
2  <head>
3    <script>
4      function myOnClick2(e) {
5        console.log("myOnClick2");
6        console.log("  e.srcElement:", e.srcElement);
7        console.log("  e.target:", e.target);
8        console.log("  e", e);
9      }
10    </script>
11
12  </head>
13  <body>
14    test button on click
15    <br/>
16    <button onclick="myOnClick2(event)"> Click
17  </body>
```

Filter Default levels ▾ | 1 Issue: 1 |

myOnClick2	test-onclick-2.html:5
e.srcElement:	test-onclick-2.html:6
<button onclick="myOnClick2(event)"> Click2(Event) </button>	
e.target:	test-onclick-2.html:7
<button onclick="myOnClick2(event)"> Click2(Event) </button>	
e	test-onclick-2.html:8
▼ PointerEvent {isTrusted: true, pointerId: 1, width: 1, height: 1, pressure: 0, ...} ⓘ	
isTrusted: true	
altKey: false	
altitudeAngle: 1.5707963267948966	
azimuthAngle: 0	
bubbles: true	
button: 0	
buttons: 0	
cancelBubble: false	
cancelable: true	
clientX: 75	
clientY: 35	
composed: true	

Event Bubbles Up ..

Event
Bubbling

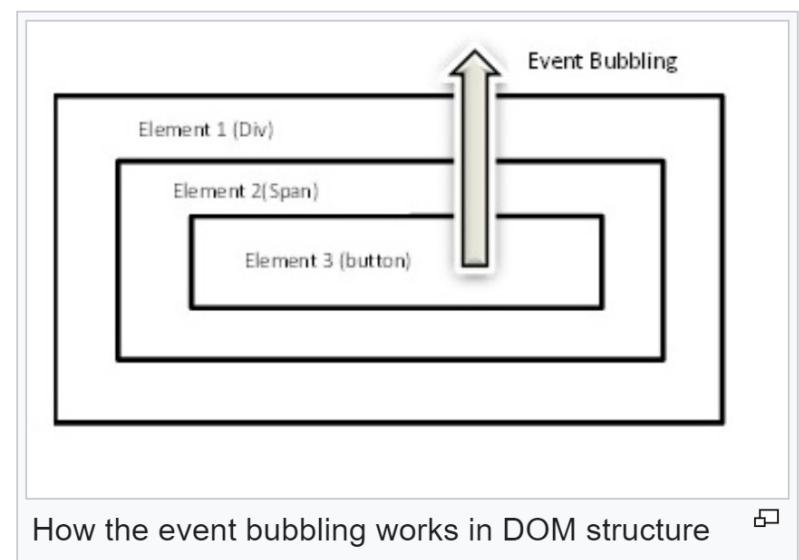


https://en.wikipedia.org/wiki/Event_bubbling

```
<div id="Element1" onclick="handler1()">  
  <span id="Element2" onclick="handler2()">  
    <input type="button" id="Element3" onclick="handler3()"/>  
  </span>  
</div>
```

When the Element3 button is clicked, an event handler for Element 3 is triggered first, then event bubbles up and the handler for immediate parent element - Element 2 is called, followed by the handler for Element 1 and so on till it reaches the outermost DOM element.

Event handling order: handler3() -> handler2() -> handler1()



Event Bubble: Sample Code

```
1  <!html>
2  <head>
3      <script>
4          function handler1(e) {
5              console.log("handler1", e);
6              console.log("  e.srcElement:", e.srcElement);
7              console.log("  e.target:", e.target);
8          }
9          function handler2(e) {
10             console.log("handler2", e);
11             console.log("  e.srcElement:", e.srcElement);
12             console.log("  e.target:", e.target);
13         }
14         function handler3(e) {
15             console.log("handler3", e);
16             console.log("  e.srcElement:", e.srcElement);
17             console.log("  e.target:", e.target);
18         }
19     </script>
20 </head>
21 <body>
22     test event bubbles up
23     <br/>
24     <div id="Element1" onclick="handler1(event)">
25         <span id="Element2" onclick="handler2(event)">
26             <button id="Element3" onclick="handler3(event)">click</button>
27         </span>
28     </div>
29 </body>
```

Event Bubble: Sample Execution

handler3 ► PointerEvent

```
e.srcElement: <button id="Element3" onclick="handler3(event)">Click</button>
e.target:    <button id="Element3" onclick="handler3(event)">Click</button>
```

handler2 ► PointerEvent

```
e.srcElement: <button id="Element3" onclick="handler3(event)">Click</button>
e.target:    <button id="Element3" onclick="handler3(event)">Click</button>
```

handler1 ► PointerEvent

```
e.srcElement: <button id="Element3" onclick="handler3(event)">Click</button>
e.target:    <button id="Element3" onclick="handler3(event)">Click</button>
```

Stopping Propagation: Sample Code

```
1  <!html>
2  <head>
3      <script>
4          function handler1(e) {
5              console.log("handler1 NOT CALLED!!", e);
6          }
7          function handler2(e) {
8              console.log("handler2", e);
9              console.log(" e.stopPropagation");
10             e.stopPropagation();
11             console.log(" e:", e);
12         }
13         function handler3(e) {
14             console.log("handler3", e);
15             console.log(" e.target:", e.target);
16             console.log(" e.bubbles:", e.bubbles);
17         }
18     </script>
19 </head>
20 <body>
21     test event bubbles up
22     <br/>
23     <div id="Element1" onclick="handler1(event)">
24         <span id="Element2" onclick="handler2(event)">
25             <button id="Element3" onclick="handler3(event)">Click</button>
26         </span>
27     </div>
28 </body>
```

Stopping Propagation: Sample Execution

```
handler3 ► PointerEvent {isTrusted: true, pointerId: 1, width: 1, height: 1, pressure: 0, ...}
```

```
  e.target: <button id="Element3" onclick="handler3(event)">Click</button>
```

```
  e.bubbles: true
```

```
handler2 ► PointerEvent {isTrusted: true, pointerId: 1, width: 1, height: 1, pressure: 0, ...}
```

```
  e.stopPropagation
```

```
  e: ► PointerEvent {isTrusted: true, pointerId: 1, width: 1, height: 1, pressure: 0, ...}
```

>

Script vs Function Callback ?

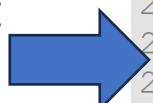
```
1  <!html>
2  <head>
3      <script>
4          function handler1(e) {
5              console.log("handler1", e);
6          }
7          function handler2(args) {
8              console.log("handler2", args);
9          }
10
11         function onLoad() {
12             e1 = document.getElementById("id1");
13             console.log("e1.onclick", e1.onclick);
14             e2 = document.getElementById("id2");
15             console.log("e2.onclick", e2.onclick);
16             e3 = document.getElementById("id3");
17             console.log("e3.onclick", e3.onclick);
18         }
19     </script>
20 </head>
21 <body onload="onLoad()">
22     test event bubbles up
23     <br/>
24     <button id="id1" onclick="handler1(event)">Click</button>
25     <button id="id2" onclick="handler2()">Click</button>
26     <button id="id3" onclick="console.log('handler3')">Click</button>
27 </body>
```



What do we get ?

3 different scripts

- function with event
- no event
- script code



Scripts are "compiled"/"wrapped" similarly in Function with 1 event param

```
e1.onclick f onclick(event) {  
    handler1(event)  
}
```

[test-onclick-querycallback.html:13](#)

```
e2.onclick f onclick(event) {  
    handler2()  
}
```

[test-onclick-querycallback.html:15](#)

```
e3.onclick f onclick(event) {  
    console.log('handler3')  
}
```

[test-onclick-querycallback.html:17](#)

Set in Code: with Function (not String)

```
test-onclick-5-set.html
1  <!html>
2  <body>
3      test DOM set onclick
4      <br/>
5      <button id="id1">Click</button> <br/>
6      <button id="id2">Click</button> <br/>
7
8      <script>
9          var handlerFunc1 = function(event) {
10              console.log('handler1', event);
11          }
12          document.getElementById('id1').onclick = handlerFunc1;
13
14          document.getElementById('id2').onclick = (event) => {
15              console.log('handler2', event);
16          }
17      </script>
18
19  </body>
```

test DOM set onclick
Click
Click

handler1 ► PointerEvent

handler2 ► PointerEvent

[test-onclick-5-set.html:10](#)

[test-onclick-5-set.html:15](#)

Set in Code ... Silent error with String !!

```
1  <!html>
2  <body>
3      test DOM set onclick error
4      <br/>
5      <button id="id1">Click</button> <br/>
6
7      <script>
8          button1 = document.getElementById('id1');
9
10         button1.onclick = "alert('handler1 DOES NOT WORK!');";
11         console.log("button1.onclick", button1.onclick); // => still null !
12
13     </script>
14
15 </body>
```

test DOM set onclick



NOTHING Happen when click (not even error)

setting attribute was silently ignored

Filter

button1.onclick null

Default levels ▾

1 Issue: 1



[test-onclick-6-set-error.html:12](#)

In Angular: (click)="script..."

```
// app.component.ts
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  template: `
    <button (click)="handleClick()">Click me</button>
  `
})
export class AppComponent {
  handleClick() {
    alert('Button clicked in Angular!');
  }
}
```

instead of native JavaScript :

onclick=<<Window script (event**)>>**

=> in Angular :

(click)=<<Component script (\$event**)>>**

script binded to the component context

\$event instead of event

No Bubbling of event

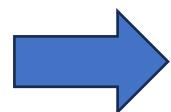
Outline

Https → Load Html + Js + Css + Json + *

Js → Call Http : SPA, Rest + Json

Html → execute Js <script>

Html → trigger events to Js



Js → Query/Update Html DOM

Custom Elements, Shadow DOM, Frameworks

D.O.M. ?

Document

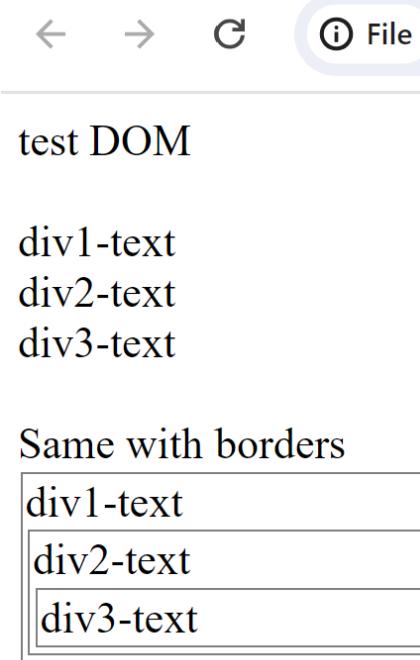
Object

Model

= API to handle HTML Document in JavaScript

HTML Source <-> Tree of Elements

```
1  <!html>
2  <body>
3    test DOM
4    <br/>
5    <br/>
6
7    <div id="id1">
8      div1-text
9      <div id="id2">
10        div2-text
11        <div id="id3">
12          div3-text
13        </div>
14      </div>
15
16    <BR/>
17    Same with borders
18    <br/>
19    <div id="bid1" style="border-
20      div1-text
21      <div id="bid2" style="borde
22        div2-text
23        <div id="bid3" style="bor
24          div3-text
25            </div>
26          </div>
27        </div>
28      </div>
29    </body>
```



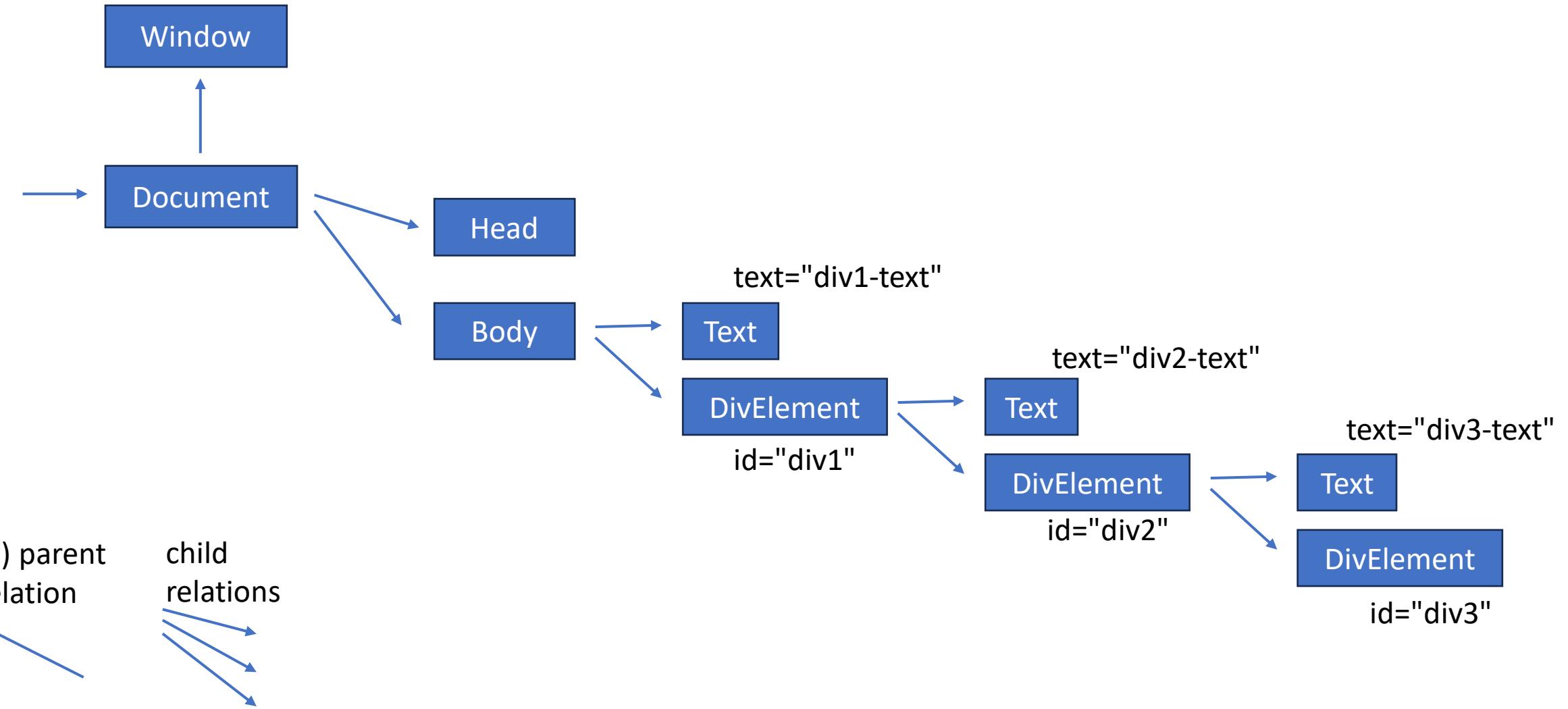
Rendering:
may add border to see container/content

well formatted xml: <element> .. </element>



DevTool Inspect:
can shrink/expand any element

DOM Tree



DOM Tree: navigable per ".getElementById", ".parentElement" ".children"

```
1  <!html>
2   <head>
3     <script>
4       function inspectDOM() {
5         e2 = document.getElementById('id2');
6         e1 = e2.parentElement;
7         e3 = e2.children[0]
8
9           console.log("e2", e2);
10          console.log("e1 = e2.parentElement", e1);
11          console.log("e3 = e2.children[0]", e3);
12      }
13    </script>
14  </head>
15  <body>
16    test DOM
17    <br/>
18    <div id="id1">
19      <span id="id2">
20        <button id="id3" onclick="inspectDOM()">Click</button>
21        </span>
22      </div>
23    </body>
```

e2 ► ...

e1 = e2.parentElement ► <div id="id1"> ... </div>

e3 = e2.children[0] <button id="id3" onclick="inspectDOM()">Click</button>

parentNode <-> parentElement

childNodes <-> children

HTML contains also "Node"

"**text**"

<!-- comment -->

<![CDATA[...]]>

..

Element is abstract sub-class of Node.

<h1>, <p>, <div>, , <...> are all Elements.

HTML DOM Tree can be navigate by **Node**
(more general then Element)

HTML DOM Classes

=> TS interface pseudo-declaration

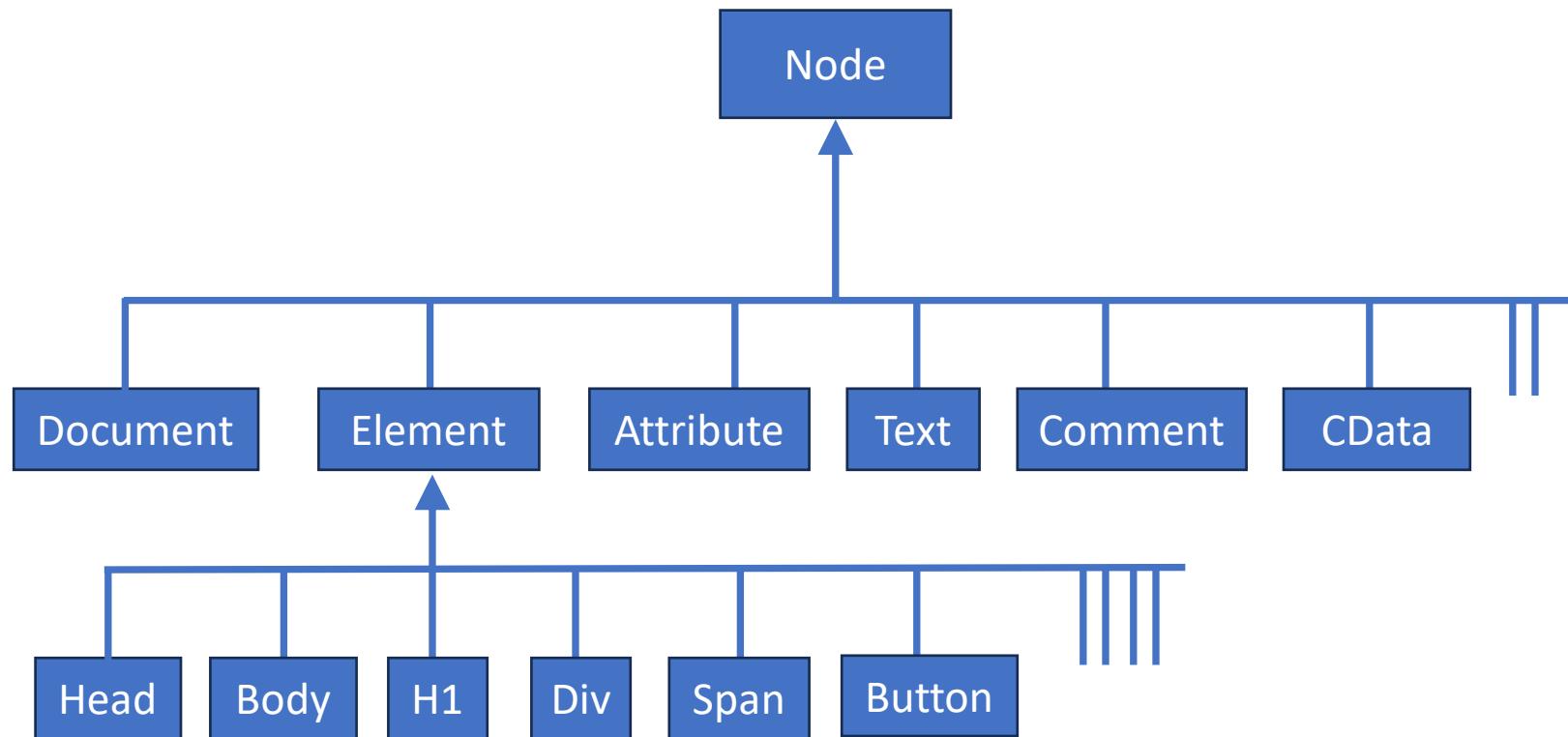
```
interface Node {  
    parentNode;  
    childNodes;  
    // parentElement, children .. computed from parent|child Node  
}
```

```
interface Text extends Node {  
    text: string;  
}
```

```
interface Element extends Node {  
    readonly tagName: string;  
    attributes: Map<string,string>;  
}
```

```
interface Div extends Element {  
    readonly tagName= "div";  
}  
...
```

HTML DOM Interfaces => UML Class Diagram



HTML DOM API Reference

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

The screenshot shows the MDN Web Docs website for the HTML DOM API. The page has a header with navigation links like 'mdn web docs', 'References', 'Guides', 'Plus', 'Curriculum', 'Blog', 'Tools', 'Theme', 'Log in', and 'Sign up for free'. Below the header, the breadcrumb navigation shows 'References > Web APIs > The HTML DOM API'. A language selector indicates 'English (US)'. On the left, there's a sidebar titled 'HTML DOM API' with a 'Filter' button. Under 'Interfaces', a list includes 'BeforeUnloadEvent', 'DOMStringMap', 'ErrorEvent', 'HashChangeEvent', 'ImageData', 'HTMLAnchorElement', 'HTMLAreaElement', 'HTMLAudioElement', 'HTMLBaseElement', 'HTMLBodyElement', and 'HTMLBRElement'. The main content area features a large heading 'The HTML DOM API'. It explains that the API consists of interfaces for HTML elements and form data, and lists functional areas like interacting with 2D images, managing media, and dragging/dropping. A sidebar on the right titled 'In this article' lists related topics: 'HTML DOM concepts and usage', 'HTML DOM target audience', 'HTML DOM API interfaces', 'Examples', 'Specifications', 'Browser compatibility', and 'See also'.

The HTML DOM API

The **HTML DOM API** is made up of the interfaces that define the functionality of each of the [elements](#) in [HTML](#), as well as any supporting types and interfaces they rely upon.

The functional areas included in the HTML DOM API include:

- Access to and control of HTML elements via the [DOM](#).
- Access to and manipulation of form data.
- Interacting with the contents of 2D images and the context of an HTML [`<canvas>`](#), for example to draw on top of them.
- Management of media connected to the HTML media elements ([`<audio>`](#) and [`<video>`](#)).
- Dragging and dropping of content on webpages.
- Access to the browser navigation history
- Supporting and connective interfaces for other APIs such as [Web Components](#), [Web Storage](#), [Web Workers](#), [WebSocket](#), and [Server-sent events](#).

w3schools ... DOM Classes List: page 1/3

← → ⌂ w3schools.com/jsref/default.asp

 Tutorials ▾ Exercises ▾ Certificates ▾ Services ▾ Search... ⚡ Plus ↗

HTML CSS JAVASCRIPT SQL PYTHON JAVA PHP HOW TO W3.CSS C C++ C# BOOTSTRAP REACT MYSQL JQUERY EXCEL XML DJANGO NUMPY PANDAS NO

Search...

JavaScript and HTML DOM

JS Reference

JS by Category

JS by Alphabet

JavaScript

- JS Arrays
- JS Boolean
- JS Classes
- JS Dates
- JS Error
- JS Global
- JS JSON
- JS Maps
- JS Math
- JS Numbers
- JS Objects
- JS Operators
- JS Precedence
- JS Promises
- JS RegExp
- JS Sets
- JS Statements
- JS Strings
- JS TypedArray

◀ Home Next ▶

Complete JavaScript and HTML DOM References

All Properties and Methods with Full Examples

Revised April 2024

JavaScript Objects with Examples

Array	String	Number	Math
Date	Global	Object	Set
Map	Classes	RegExp	Error
Boolean	Operators	Statements	Typed Array
Promise	JSON	Precedence	

DOM Classes List: page 2/3

← → ⌂ w3schools.com/jsref/default.asp

w3 schools Tutorials ▾ Exercises ▾ Certificates ▾ Services ▾ Search... ⚡ Plus

HTML CSS JAVASCRIPT SQL PYTHON JAVA PHP HOW TO W3.CSS C C++ C# BOOTSTRAP REACT MYSQL JQUERY EXCEL XML DJANGO NUMPY PANDAS

Search...

JS Reference

JS by Category

JS by Alphabet

JavaScript

JS Arrays
JS Boolean
JS Classes
JS Dates
JS Error
JS Global
JS JSON
JS Maps
JS Math
JS Numbers
JS Objects
JS Operators
JS Precedence
JS Promises
JS RegExp
JS Sets
JS Statements
JS Strings
JS TypedArray

Window Objects with Examples

Window History Location Navigator

Screen

HTML DOM Objects with Examples

Document Element Attributes Events

Event Objects HTMLCollection Style

Web APIs with Examples

Canvas Console Fetch Fullscreen

Geolocation History MediaQueryList Storage

Validation

DOM Classes List: page 3/3

W3 Schools

Tutorials ▾ Exercises ▾ Certificates ▾ Services ▾ Search... ⚡

HTML CSS JAVASCRIPT SQL PYTHON JAVA PHP HOW TO W3.CSS C C++ C# BOOTSTRAP REACT MYSQL JQUERY EXCEL XML DJANGO NUMPY PANDAS NO

Search...

JS Reference

JS by Category

JS by Alphabet

JavaScript

JS Arrays
JS Boolean
JS Classes
JS Dates
JS Error
JS Global
JS JSON
JS Maps
JS Math
JS Numbers
JS Objects
JS Operators
JS Precedence
JS Promises

HTML Elements with Examples

a abbr address area article aside audio b base bdo blockquote body br button canvas caption cite code col colgroup datalist dd del details dfn dialog div dl dt em embed fieldset figcaption figure footer form head header h1 - h6 hr html i iframe img ins input button input checkbox input color input date input datetime input datetime-local input email input file input hidden input image input month input number input password input radio input range input reset input search input submit input text input time input url input week kbd label legend li link map mark menu menuitem meta meter nav object ol optgroup option output p param pre progress q s samp script section select small source span strong style sub summary sup table tbody td tfoot th thead tr textarea time title track u ul var video

Other

CSSStyleDeclaration JS Conversion

Focus on class "Button" (page 1/2)

w3schools.com/jsref/dom_obj_pushbutton.asp

Tutorials ▾ Exercises ▾ Certificates ▾ Services ▾ Search...  

HTML CSS JAVASCRIPT SQL PYTHON JAVA PHP HOW TO W3.CSS C C++ C# BOOTSTRAP

API Validation
API Web

HTML DOM Button Object

<a>
<abbr>
<address>
<area>
<article>
<aside>
<audio>

<base>
<bdo>
<blockquote>
<body>

<button>
<canvas>
<caption>
<cite>
<code>
<col>
<colgroup>
<datalist>
<dd>

<details>
<dfn>
<dialog>
<div>
<dl>

Button Object

The Button object represents an HTML <button> element.

Access a Button Object

You can access a <button> element by using getElementById():

Example

```
var x = document.getElementById("myBtn");
```

[Try it Yourself »](#)

Create a Button Object

You can create a <button> element by using the document.createElement() method:

Example

```
var x = document.createElement("BUTTON");
```

to get an existing <button id="..">

to create a "new Button()" ??
document.createElement("button")

class "Button" (page 2/2)

The screenshot shows a web browser displaying the W3Schools website at w3schools.com/jsref/dom_obj_pushbutton.asp. The page title is "Button Object Properties". On the left, there is a sidebar with a list of HTML objects, and the "<button>" object is highlighted with a green background. The main content area contains a table with 12 rows, each representing a property of the Button object with its description.

Property	Description
autofocus	Sets or returns whether a button should automatically get focus when the page loads, or not
disabled	Sets or returns whether a button is disabled, or not
form	Returns a reference to the form that contains a button
formAction	Sets or returns the value of the formaction attribute of a button
formEnctype	Sets or returns the value of the formenctype attribute of a button
formMethod	Sets or returns the value of the formmethod attribute of a button
formNoValidate	Sets or returns whether the form-data should be validated or not, on submission
formTarget	Sets or returns the value of the formtarget attribute of a button
name	Sets or returns the value of the name attribute of a button
type	Sets or returns the type of a button
value	Sets or returns the value of the value attribute of a button

Standard Properties and Events

The Button object also supports the standard [properties](#) and [events](#).

Related Pages

HTML reference: [HTML <button> tag](#)

Reference HTMLButtonElement interface

mdn web docs References Guides Plus Curriculum Blog Tools NEW Theme Log in Sign up for free

References > Web APIs > HTMLButtonElement English (US)

Filter

HTMLButtonElement

The `HTMLButtonElement` interface provides properties and methods (beyond the regular `HTMLElement` interface it also has available to it by inheritance) for manipulating `<button>` elements.

```
graph TD; EventTarget --> Node; Node --> Element; Element --> HTMLElement; HTMLElement --> HTMLButtonElement
```

Instance properties

Inherits properties from its parent, `HTMLElement`.

`HTMLButtonElement.disabled`

A boolean value indicating whether or not the control is disabled, meaning that it does not accept any clicks.

`HTMLButtonElement.form` Read only

An `HTMLFormElement` reflecting the form that this button is associated with. If the button is a descendant of a form element, then this attribute is a reference to that form's associated `HTMLFormElement`. If the button is not a descendant of a form element, then the attribute can be a

HTML DOM API

- HTMLButtonElement
- ▼ Instance properties
 - disabled
 - labels
 - popoverTargetAction
 - popoverTargetElement
 - type
- ▼ Inheritance
 - HTMLElement
 - Element
 - Node
 - EventTarget
- ▼ Related pages for HTML DOM

In this article

- Instance properties
- Instance methods
- Specifications
- Browser compatibility
- See also

API to Add/Replace/Remove Dom Node

The screenshot shows the MDN Web Docs interface. At the top, there's a navigation bar with links for developer.mozilla.org, mdn web docs (which is underlined), References, Guides, Plus, Curriculum, Blog, and Tools (with a 'NEW' badge). Below the navigation is a breadcrumb trail: References > Web APIs > Node. On the left, there's a 'Filter' button. The main content area has a title 'Instance methods' with a dropdown arrow icon above it. The URL in the address bar is developer.mozilla.org/en-US/docs/Web/API/Node#instance_methods.

▼ Instance methods

`appendChild()`

`cloneNode()`

`compareDocumentPosition()`

`contains()`

`get rootNode()`

`hasChildNodes()`

`insertBefore()`

`isDefaultNamespace()`

`isEqualNode()`

`isSameNode()`

`lookupNamespaceURI()`

`lookupPrefix()`

`normalize()`

`removeChild()`

`replaceChild()`

Sample Code to Add/Remove HTML

```
1  <!html>
2  <head>
3  <script>
4      function onClickAddToList() {
5          inputElement = document.getElementById("id1");
6          listElement = document.getElementById("id2");
7
8          inputTextValue = inputElement.value;
9          console.log("adding value", inputTextValue);
10
11         itemElement = document.createElement("li");
12         itemElement.appendChild(document.createTextNode(inputTextValue));
13
14         listElement.appendChild(itemElement);
15     }
16 </script>
17
18 </head>
19 <body>
20     <input id="id1" type="text">
21     <br/>
22     <button onclick="onClickAddToList()"> Add To List </button>
23     <ul id="id2">
24         <li> ... </li>
25         <!-- filled dynamically by onClickAddToList() -->
26     </ul>
27 </body>
```

bar
Add To List

- ...
- foo
- bar
- bar
- bar
- bar

Debugging Elements (in-memory differ from Sources, refresh by "F5")

The screenshot shows a browser window with the URL `file:///C:/tmp/test-onclick/test-input-add-list.html`. The page contains a text input field with the value "bar", a button labeled "Add To List", and a list of items below it.

The DevTools Elements tab is active, displaying the DOM structure:

```
<body>
  <input id="id1" type="text">
  <br>
  <button onclick="onClickAddToList()"> Add To List </button>
  <ul id="id2">
    <li>...</li>
    <!-- filled dynamically by onClickAddToList() -->
    <li>...</li>
    <li>...</li>
    ...
    <li> == $0
      ::marker
      "bar"
    </li>
    <li>...</li>
    <li>...</li>
  </ul>
</body>
</html>
```

The list item containing "bar" is highlighted with a blue selection bar. A tooltip above the list item shows its dimensions: `1480 × 18.8`.

The bottom status bar of the DevTools shows the message `4 adding value bar`.

Outline

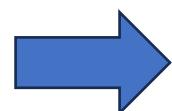
Https → Load Html + Js + Css + Json + *

Js → Call Http : SPA, Rest + Json

Html → execute Js <script>

Html → trigger events to Js

Js → Query/Update Html DOM



Custom Elements, Shadow DOM, Frameworks

Code Maintainability of <div> soup ?

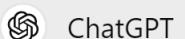
The screenshot shows the Gmail inbox interface on a computer screen. The left sidebar lists various sections like 'Boîte de réception' (14), 'Messages suivis' (12), and 'Spam' (72). The main area displays a list of messages. On the right, the developer tools' Elements tab is open, showing the hierarchical DOM structure of the page. The structure is heavily nested with many 'div' elements, each assigned unique class names such as 'nH', 'aeN WR baA nH oy8Mbf', 'Ls77Lb aZ6', etc. This visual representation highlights the complexity of the code, which is the subject of the slide's question.

```
<div tabindex="0"></div>
<div class="nH" style="width: 1536px;">
  <div class="nH" style="position: relative;">
    <div class="nH aqk aql bkl" flex="flex">
      <div class="aeN WR baA nH oy8Mbf" role="navigation" jslog="88024; u014N:xr6bB;" style="width: 187px; height: 665px;" flex="flex">
        <div class="aic" flex="flex">...</div>
        <div class="V3 aam ada" flex="flex">
          <div class="at9" flex="flex">
            <div class="Ls77Lb aZ6" flex="flex">
              <div jscontroller="DUNnfe" class="pp" style="user-select: none;">
                <div id=":m5" flex="flex">
                  <div class="nM" flex="flex">
                    <div id=":lw" class="aic" flex="flex"></div>
                  <div class="yJ" flex="flex">
                    <div class="ajl aib aZ6" aria-labelledby=":mh" flex="flex">
                      <h2 class="awk" id=":mh" flex="flex">Libellés</h2>
                    <div class="wT" flex="flex">
                      <div id=":m6" class="n3" flex="flex">
                        <div class="byl" flex="flex">
                          <div class="TK" flex="flex">
                            <div class="aim ain" flex="flex">
                              <div class="TO aBP nZ aiq" id=":mk" data-tooltip="Boîte de réception" data-tooltip-align="r" jslog="36893; u014N:cOuCgd; 14:WzZd" flex="flex"> == $0
                            <div class="TN hzz aHS-hnt" style="margin-left: 0px;" flex="flex">

```



ChatGPT ▾



ChatGPT



Today

HTTP Client Request Example ...



HTML Custom Elements allow you to create your own HTML tags with custom behavior, properties, and lifecycle methods. This feature is part of the Web Components standard and provides a way to encapsulate functionality and promote code reuse. Here's an overview of how to create and use custom elements.

Key Concepts of Custom Elements

1. **Custom Element:** A new HTML tag that you define.
2. **Shadow DOM:** Encapsulates the internal structure of the custom element to provide style and DOM isolation.
3. **Template:** Defines the HTML structure of the custom element.
4. **Lifecycle Callbacks:** Methods that allow you to run code at specific points in the element's lifecycle.

explain html custom elements

Reference Custom Elements

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

The screenshot shows the MDN Web Docs interface. The top navigation bar includes links for 'mdn web docs', 'References', 'Guides', 'Plus', 'Curriculum', 'Blog', 'Tools NEW', 'Theme', a search bar, 'Log in', and 'Sign up for free'. The left sidebar has a 'Web components' section with 'Using custom elements' highlighted in purple. Other sections include 'Web components', 'Guides' (with 'Using custom elements'), 'Interfaces' (with 'CustomElementRegistry', 'CustomStateSet', 'ElementInternals', 'HTMLSlotElement', 'HTMLElementTemplate', 'ShadowRoot'), and 'Properties'. The main content area features a large heading 'Using custom elements'. Below it, a paragraph explains that custom elements are HTML elements whose behavior is defined by the developer. A note states that this article introduces custom elements and walks through some examples. A sub-section titled 'Types of custom element' discusses the two types: customized built-in elements and registered custom elements. A note at the bottom cautions about implementation reality. The right sidebar lists 'In this article' topics: 'Types of custom element', 'Implementing a custom element', 'Registering a custom element', 'Using a custom element', 'Responding to attribute changes', and 'Examples'.

Using custom elements

One of the key features of web components is the ability to create *custom elements*: that is, HTML elements whose behavior is defined by the web developer, that extend the set of elements available in the browser.

This article introduces custom elements, and walks through some examples.

Types of custom element

There are two types of custom element:

- **Customized built-in elements** inherit from standard HTML elements such as [HTMLImageElement](#) or [HTMLParagraphElement](#). Their implementation extends the behavior of select instances of the standard element.

Note: Please see the `is` attribute reference for caveats on implementation reality of custom built-in elements.

In this article

- Types of custom element
- Implementing a custom element
- Registering a custom element
- Using a custom element
- Responding to attribute changes
- Examples

Custom Element Sample (Feature 1/4)

Window.customElements.define("new-tag")

```
1  <!html>
2  <head>
3    <script>
4      // Create a class for the element
5      class PopupInfo extends HTMLElement {
6        constructor() {
7          super();
8        }
9
10      }
11
12      // Define the new element
13      customElements.define("popup-info", PopupInfo);
14    </script>
15  </head>
16  <body>
17    test Custom Element 1:
18    <br/>
19
20    <popup-info>
21      popup content
22    </popup-info>
23
24  </body>
```

DOM Tree ... <popup-info>

The screenshot shows a browser window with DevTools open, displaying the DOM tree under the 'Elements' tab. The page content is:

```
<!--html-->
<html>
  <head> ...
  </head>
  <body>
    " test popup-info: "
    <br>
    ...
    <popup-info> popup content </popup-info> == $0
  </body>
</html>
```

A floating inspection panel is shown for the element `<popup-info>`. The panel title is `test popup-info:` and the sub-section is `popup content`. The panel displays the following information:

<code>popup-info</code>		91.09 × 18
Color	<input type="color" value="#000000"/>	
Font	16px "Times New Roman"	
ACCESSIBILITY		
Contrast	Aa	21 <input checked="" type="checkbox"/>
Name		
Role	generic	
Keyboard-focusable	<input type="checkbox"/>	

Not calling ".define()" ... same DOM anyway!?

```
1  <!html>
2  <head>
3  <script>
4  // Create a class for the element
5  class PopupInfo extends HTMLElement {
6  constructor() {
7    super();
8  }
9
10 }
11 // Define the new element
12 // customElements.define("xxx-popup-info", PopupInfo);
13 </script>
14 </head>
15 <body>
16   test Custom Element 2: ERROR(?) no define
17   <br/>
18
19   <popup-info>
20     popup content
21   </popup-info>
22
23 </body>
24
```

test Custom Element 2: ERROR(?) no define
popup content

The screenshot shows the Google DevTools interface with the 'Elements' tab selected. The DOM tree on the left displays the following structure:

- <!--html-->
- <html>
- > <head> ... </head>
- > <body>
- " test Custom Element 2: ERROR(?) no define "
-

- ... <popup-info> popup content </popup-info> == \$0
- </body>
- </html>

Below the DOM tree, the 'Console' tab is selected in the bottom navigation bar. The console area shows the following messages:

- html body popup-info
- ⋮ Console Rendering Search
- [] top ⌂ ⌂ Filter
- No messages >
- ⌚ No user messag...
- ✖ No errors
- ⚠ No warnings
- ⓘ No info
- 챗 No verbose

Custom Element (Feature 2/4)

Shadow DOM - encapsulate internal elements, with style and DOM isolation

```
11 connectedCallback() {  
12     // Create a shadow root  
13     const shadow = this.attachShadow({ mode: "open" });  
14  
15     // Create spans  
16     const wrapper = document.createElement("span");  
17     wrapper.setAttribute("class", "wrapper");  
18  
19     const icon = document.createElement("span");  
20     icon.setAttribute("class", "icon");  
21     icon.setAttribute("tabindex", 0);  
22     const info = document.createElement("span");  
23     info.setAttribute("class", "info");  
24     const text = this.getAttribute("data-text");  
25     info.textContent = text;  
26     const img = document.createElement("img");  
27     img.src = (this.hasAttribute("img"))?  
28         this.getAttribute("img") : "img/default.png";  
29     icon.appendChild(img);  
30  
31     wrapper.appendChild(icon);  
32     wrapper.appendChild(info);  
33  
34     // Attach the created elements to the shadow dom  
35     shadow.appendChild(style);  
36     shadow.appendChild(wrapper);  
37 }  
38 }
```

```
31 // Create some CSS to apply to the shadow dom  
32 const style = document.createElement("style");  
33  
34 style.textContent = `  
35     .wrapper {  
36         position: relative;  
37     }  
38  
39     .info {  
40         font-size: 0.8rem;  
41         width: 200px;  
42         display: inline-block;  
43         border: 1px solid black;  
44         padding: 10px;  
45         background: white;  
46         border-radius: 10px;  
47         opacity: 0;  
48         transition: 0.6s all;  
49         position: absolute;  
50         bottom: 20px;  
51         left: 10px;  
52         z-index: 3;  
53     }  
54  
55     img {  
56         width: 1.2rem;  
57     }  
58  
59     .icon:hover + .info, .icon:focus + .info {  
60         opacity: 1;  
61     }  
62`;
```

Navigable HTML Structure

<--differs from --> Styled, Internal Shadow DOM

The screenshot shows a browser developer tools interface with the following components:

- HTML Tree View:** On the left, a tree view of the HTML structure. A node labeled '#shadow-root (open)' is highlighted with a red box. The node contains a style element and a span element with class="wrapper". The span element contains an icon and an info section.
- Style Editor:** On the right, a panel showing the CSS properties for the selected element:
 - display: inline
 - height: auto
 - width: auto
- Console Tab:** At the bottom, the 'Console' tab is active. It shows the following output:

```
> popupElt = $0
<- > <popup-info img="img/alt.png" data-text="This is a long popup message - but the corresponding HTML elements (and style)
      are in the shadow DOM, not accessible from DOM node children">...</popup-info>
> popupElt.childElementCount
<- 0
```

Custom Element Sample (Feature 3/4) using HTML Template

```
test-custom-elt-4-template.html x
1  <!html>
2  <head>
3    <script src="app4.js"></script>
4  </head>
5  <body>
6    test Custom Element 4: using template
7    <br/>
8
9    <my-custom-elt>
10   </my-custom-elt>
11
12 </body>
```

```
app4.js x
1  // Create a class for the element
2  class MyCustomElement extends HTMLElement {
3    constructor() {
4      super();
5      this.attachShadow({ mode: 'open' });
6
7      const template = document.createElement('template');
8      template.innerHTML = `
9        <style>
10          .container {
11            padding: 10px;
12            background-color: #f0f0f0;
13            border: 1px solid #ccc;
14          }
15        </style>
16        <div class="container">
17          <p>This is a custom element!</p>
18        </div>
19      `;
20
21      this.shadowRoot.appendChild(template.content.cloneNode(true));
22    }
23
24
25    // Define the new element
26    customElements.define("my-custom-elt", MyCustomElement);
```

result of HTML Template

The screenshot shows a browser's developer tools with the "Elements" tab selected. The page title is "test Custom Element 4: using template". On the left, the rendered content is displayed: "This is a custom element!". On the right, the DOM tree is shown:

```
<!--html-->
<html>
  > <head> ...
  > <body>
    " test Custom Element 4: using template "
    <br>
    > <my-custom-elt>
      > #shadow-root (open)
        ... > <style> ... </style> == $0
        > <div class="container">
          <p>This is a custom element!</p>
        </div>
      </my-custom-elt>
    </body>
</html>
```

The shadow root of the custom element is expanded, showing its internal structure: a style element and a container div containing a single paragraph.

Custom Element (Feature 4/4) using Lifecycle callbacks & properties

3. Lifecycle Callbacks:

- `connectedCallback()`: Called when the element is added to the document. Useful for setup tasks.
- `disconnectedCallback()`: Called when the element is removed from the document. Useful for cleanup tasks.
- `adoptedCallback()`: Called when the element is moved to a new document.
- `attributeChangedCallback(name, oldValue, newValue)`: Called when an observed attribute changes. Useful for reacting to attribute changes.
- `static get observedAttributes()`: Specifies the attributes to observe for changes.

Lifecycle callbacks

```
24 // Lifecycle callbacks (optional)
25 connectedCallback() {
26   console.log('Custom element added to the page.');
27 }
28
29 disconnectedCallback() {
30   console.log('Custom element removed from the page.');
31 }
32
33 adoptedCallback() {
34   console.log('Custom element moved to a new page.');
35 }
36
37 attributeChangedCallback(name, oldValue, newValue) {
38   console.log(`Attribute: ${name} changed from ${oldValue} to ${newValue}`);
39 }
40
41 static get observedAttributes() {
42   return ['data-custom']; // Specify which attributes to observe.
43 }
```

Sample attributeChangedCallback() + getAttribute() + Shadow DOM update

```
test-custom-elt-6.html app6.js
1  <!html>
2  <head>
3    <script src="app6.js"></script>
4    <script>
5      function incrDataCustomAttribute() {
6        var customElt = document.getElementById("id1");
7        var newVal = "x" + customElt.getAttribute("data-custom");
8        customElt.setAttribute("data-custom", newVal);
9      }
10   </script>
11 </head>
12 <body>
13   test Custom Element 5: update attribute
14   <br/>
15
16   <my-custom-elt id="id1" data-custom="a">
17   </my-custom-elt>
18
19   <br/>
20   <button onclick="incrDataCustomAttribute()"> Update </button>
21
22 </body>
```

```
test-custom-elt-6.html app6.js
1  // Create a class for the element
2  class MyCustomElement extends HTMLElement {
3    constructor() {
4      super();
5      this.attachShadow({ mode: 'open' });
6
7      const template = document.createElement('template');
8      template.innerHTML = `
9        <style>
10          .container {
11            padding: 10px;
12            background-color: #f0f0f0;
13            border: 1px solid #ccc;
14          }
15        </style>
16        <div class="container">
17          <p>This is a custom element!</p>
18          <p id="id1"></p>
19        </div>
20      `;
21
22      this.shadowRoot.appendChild(template.content.cloneNode(true));
23    }
24
25    attributeChangedCallback(name, oldValue, newValue) {
26      console.log(`Attribute: ${name} changed from ${oldValue} to ${newValue}`);
27      const pElt = this.shadowRoot.getElementById("id1");
28      pElt.textContent = newValue;
29    }
30
31    static get observedAttributes() {
32      return ['data-custom']; // Specify which attributes to observe.
33    }
34
35  }
36
37  // Define the new element
38  customElements.define("my-custom-elt", MyCustomElement);
```

attributeChangedCallback=> update Shadow DOM

The screenshot shows a browser window with the title "test Custom Element 5: update attribute". Inside, a custom element displays the text "This is a custom element!" and "xxxxxa". Below it is a button labeled "Update". The DevTools sidebar shows the file structure: top → file:/// → C:/tmp/test-custom-elements → test-custom-elt-6.html and app6.js. The "Sources" tab is active, displaying the code for app6.js. The code includes an attributeChangedCallback function and a static get observedAttributes() method. The console tab shows seven messages, all related to attribute changes:

- Attribute: data-custom changed from null to a
Custom element added to the page.
- Attribute: data-custom changed from a to xa
- Attribute: data-custom changed from xa to xxa
- Attribute: data-custom changed from xxa to xxxx
- Attribute: data-custom changed from xxxx to xxxx
- Attribute: data-custom changed from xxxx to xxxxxa
- Attribute: data-custom changed from xxxxxa to xxxxxa

.. not broadly used ?

why are custom elements not broadly used ?

Custom Elements:

- **HTML Standard:** Built on web standards, no need for a framework-specific runtime.
- **Encapsulation:** Strong encapsulation with Shadow DOM.
- **Reusability:** Can be used across different frameworks and libraries.
- **Complexity:** Requires understanding of new concepts and more verbose API.

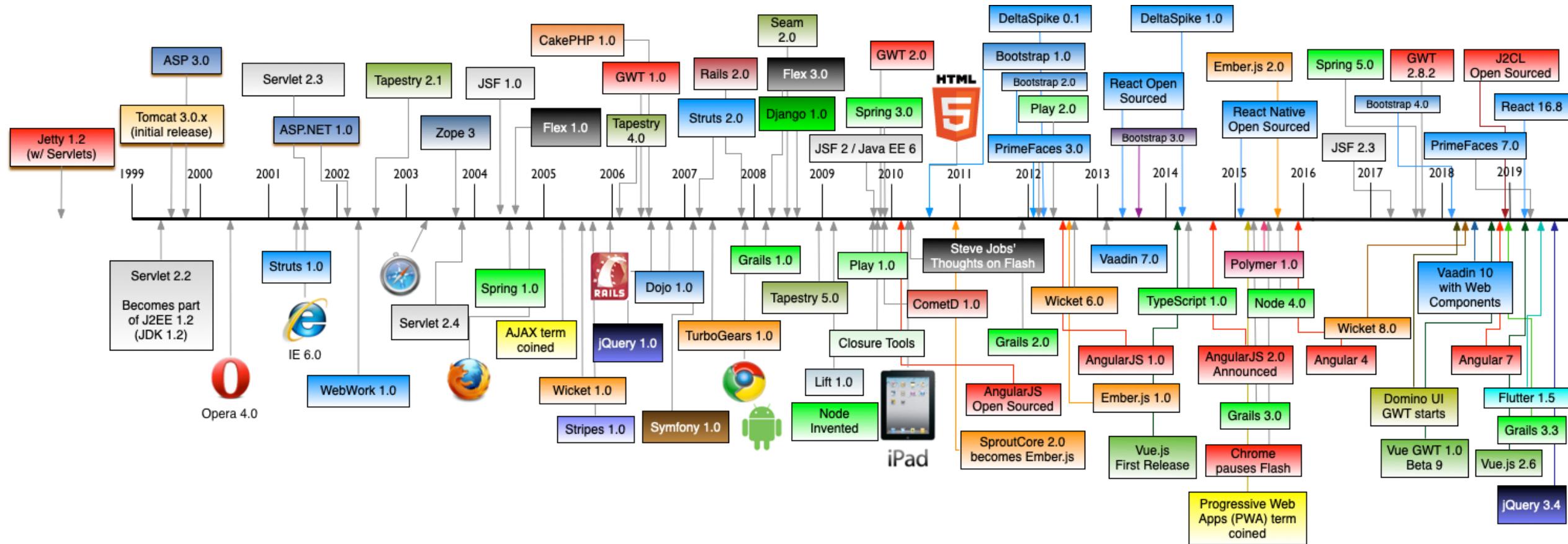
Conclusion

While custom elements offer powerful and flexible capabilities, their adoption is limited by factors such as the learning curve, compatibility issues, ecosystem maturity, and the established dominance of frameworks like React, Angular, and Vue. These frameworks provide robust tooling, community support, and features that simplify development, making them more appealing choices for many developers and organizations. However, as the web standards and browser support for custom elements continue to improve, their usage may increase, especially for scenarios requiring highly reusable and framework-agnostic components.



(Which - HowLong) Web Frameworks?

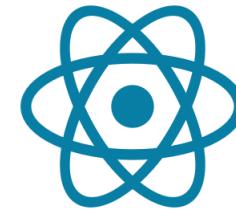
<https://github.com/mraible/history-of-web-frameworks-timeline>



Frameworks trends as of 2024



<https://angular.dev/>



React

<https://react.dev/>



<https://vuejs.org/>

Angular 18 (may 2024)
very different/superior to Angular 17
>> to Angular 16,15, .. 2
>> completely different from Angular"JS" 1



React : only rendering TSX
... Need many additional libraries,
for Router
+ State Management
+ Http
+ Async + ...

example for State Management:

- Redux (The Infamous)
- Mobx
- React Context API
- Zustand
- Recoil
- etc

Angular vs React



"All in one" Framework

using Typescript for code

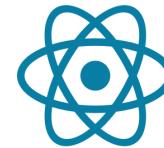
html templating with 2-way Bindings

previously with "*ng" directives
(example *ngIf="" now replaced with "@if{ }")

very powerfull and consistent

using RxJs for async,
and now "signals"

... long & complex to learn
pre-requisite: Typescript + RxJs



React

a minimalist "rendering only" Framework
with Virtual DOM (to compensate very old browser bugs)

TSX = mix of Html + Typescript interleaved
+ escaping symbols

NO binding (or only 1-way, to update yourself)

all complex features need to be added anyway
in additional libraries.

Redux -the legacy "state management"- is very complex and
now replaced

... react de facto minimalist standard,
but too many long & complex libraries to choose and learn

