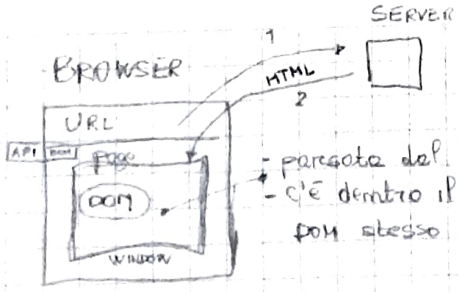
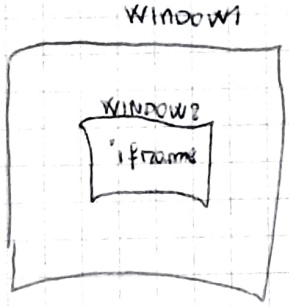
AIR TORQUE
Head Offices: Via dei Livelli di Sopra, 11
Factory: Via dei Livelli di Sopra, 8
24060 COSTA DI MEZZATE (Bergamo) ITALY
Tel. +39 035 682299 - Fax +39 035 687791
www.airtorque.it - e-mail: info@airtorque.it



- parsata dal browser che crea il DOM
- c'è dentro il global-object Window che permette di interagire con DOM stesso e BOM e l'environment-host di JS

server → HTML → Browser lo parsa → DOM.

OSS



WINDOW1 ≠ WINDOW2
go1 ≠ go2

Tuttavia questi si possono "parlare"!
e scambiarsi cose.

OSS ϟ

- DOMContentLoaded : HTML caricato
- load : anche tutte le risorse esterne caricate. (es. img)

0 caricamento pagina

ASAP (*) Syncrono.

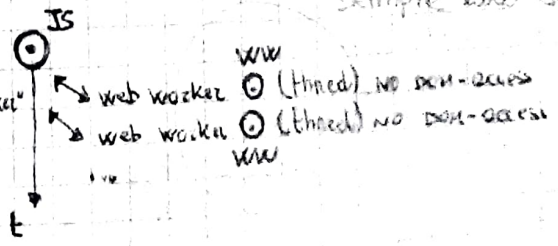
FASE 1 JS caricamento del DOM e delle funzioni JS con defer, async o niente, in sostanza caricamento dei JS utili per fase 2

DOMContentLoaded ϟ DOM ok, ma magari mancano immagini e contenuti esterni. Inoltre possono esserci script async/defer non ancora eseguiti

load ϟ (*) → FASE 2 JS ho il contenuto completo della pagina
Di solito qui si registrano tutti gli eventHandler perché c'è tutto.
Quindi qui associo a tutti gli elementi i loro eventi.
Più tutti gli async/defer sono stati eseguiti

JAVASCRIPT è Single-thread — sempre lato client

però può delegare alcuni compiti ai "Web-worker" cioè thread

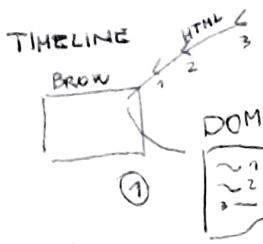


MAIN-THREAD
SS

dove fare calcoli SS (li delega a un web-worker)

qui non c'è una chiamata asincrona a un server e tutto lato Client

---



Mentre ~~gli~~ analizza l'HTML crea il DOM passo a passo

window.document.readyState → "LOADING"

---

② becca uno <script> senza async o defer e quindi lo esegue subito!

SINCRONO (download, run)

Dato che è sincrono può usare document.write() per inserire cose nello stream che diventeranno parte del DOM.
Può vedere il DOM che è stato creato fino a quel punto.

③ becca <script> con async

lo scarica

... nel frattempo va avanti

- esegue ASAP lo script scaricato.

NON può usare document.write()

④ becca <script> con defer (lo scarica asincronamente)

- eseguito alla fine
- ha accesso a tutto il DOM
- non può usare docum.write()

---

⑤ Quando tutto parsato: window.docum.readyState → "INTERACTIVE"

⑥ DOMContentLoaded (NB: Possono esserci async non ancora eseguiti)

⑦ ~~window.load~~, window.docum.readyState → "COMPLETE" → load()

- ERRORS

window.onerror = function (msg, url, line) { push-server }.
return true;

- SICUREZZA
  - JS non può cancellare o creare file/directory su PC utente
  - JS ~~[illegible]~~ ha capacità limitate di chiamate HTTPS/HTTP.

1) SOP

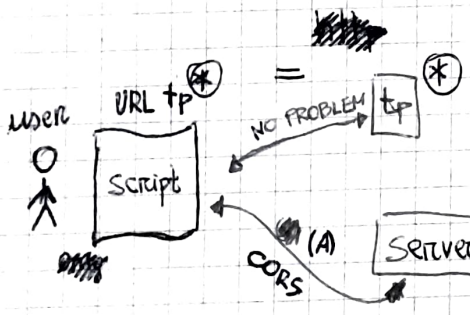
3° 2° 1° TopLevelDomain

blog. miosito. it

NOME A DOMINIO

SOP = Same Origin Policy

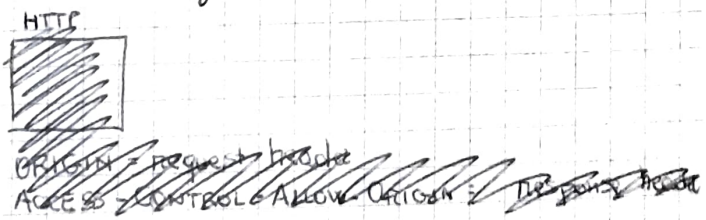
ORIGIN = ~~[illegible]~~ PROTOCOL | HOST | PORT dell' URL

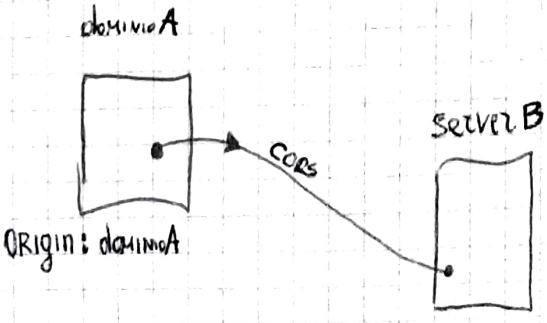


(A) Se faccio una XMLHTTPREQUEST il client deve specificare nell'header Origin e il server deve permetterlo

(Se voglio comunicare devo impostare i CORS)

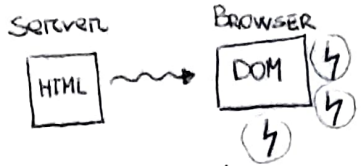
CORS = Cross Origin Resource Sharing. (Servono per rilassare la SOP)





EVENTI ϟ

- Li genera il browser.



gli eventi si registrano sul DOM.

- non sono solo legati a modi HTML ma un po' a tutto!

- **EVENT TYPE** (event name) : "mouseover", "keydown", "load", ...
  STRINGA

- **EVENT TARGET** : Window, Element (Button), Document, ...
  OGGETTO
  Oggetto sul quale "accade" l'evento, e a cui è associato l'evento.

- **EVENT HANDLER** (event listener) : funzione che risponde a un evento (E-TYPE) su un oggetto (E-TARGET) quando "triggerato" dal browser.
  FUNZIONE

  NB: Posso gestire la stop-propagation (bubbling)

  NÌ 1) ETarget.Etype — window.onload = function(event){...} — form.onsubmit = function(event){...}
  Questo modo ha il difetto di sovrascrivere (eventualmente) gli eventi.
  NO 2) Diretto in HTML: <button onclick="... code ...">OK</button> ; deprecato!
  ok 3) addEventListener: b.addEventListener("click", handler, options) → {capture:V, once:V, passive:V} (once: TRUE)

- **EVENT OBJECT** : Oggetto (passato all'handler) con le caratteristiche specifiche dell'evento
  OGGETTO
  { type: "mouseover"; target: Button ; ... }

EVENT PROPAGATION : Processo con il quale il browser decide quali handler chiamare e su quali target.

Alcuni eventi hanno associato delle azioni di default.
Es. clic su link → carico nuova pagina.
Se non voglio l'azione di default devo cancellare l'evento di default associato, cioè facciamo un "cancelling".

AIR TORQUE®
AIR TORQUE S.P.A.
Società Unipersonale soggetta ad attività di direzione e coordinamento da parte della Società Samson A.G.
Head Offices: Via dei Livelli di Sopra, 11
Factory: Via dei Livelli di Sopra, 8
24060 COSTA DI MEZZATE (Bergamo) ITALY
Tel. +39 035 682299 - Fax +39 035 687791
www.airtorque.it - e-mail: info@airtorque.it

DOM **obj**.addEventListener("~~on~~click", handler, options) (1)

options → true

options → { capture: true, once: true, passive: true }

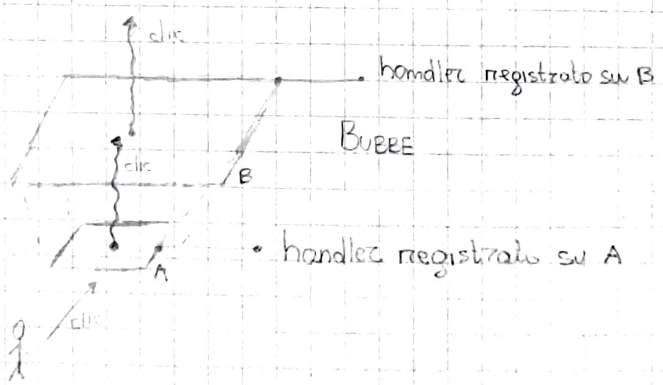
capture: TRue ⇝ l'handler verrà registrato come "capturing" e non come "bubbling" NB: DEFAUT è false, cioè bubbling

once: TRue ⇝ ~~~~ handler rimosso dopo che viene triggerato.

passive: True ⇝ handler NON chiama preventDefault() per concedere l'azione di default.

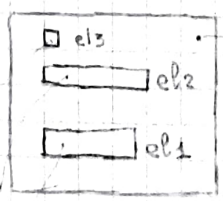
(1)

```
function handler (event) {        1 solo Argomento
  ...
  this? oggetto del DOM su cui è registrato l'evento
  ...
  ...
  event.preventDefault();  ≈ return false.   Significa di non eseguire
  }                          preferibile  legacy   l'azione di default.
```



{ NON Tutti gli eventi fanno BUBBLE, es "focus"

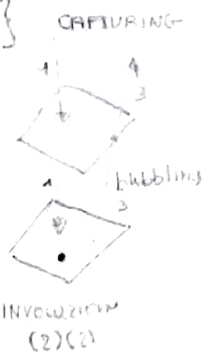
Esempio:



(1) Invece di registrare un evento "change" per ogni element

(2) registro un solo evento "change" sul FORM. Dato che gli element fanno BUBBLE.

FASE 1 : "capturing" {capture : true}



FASE 2 : Invocazione handler

FASE 3 : Bubbling

EVENT CANCELLATION

1) event.preventDefault() → cancella azione default

2) event. stopPropagation() / stopImmediatePropagation() } → cancella propagazione capturing & bubbling

EVENTI CUSTOM

A) document.dispatchEvent(new CustomEvent("busy")) T, F

B) document.addEventListener("busy", showSpinner)

OLD: