Appunti: Introduzione a Javascript - Temi Trasversali (Parte I)

Navigazione sul DOM

- Il DOM (HTML/XML) non ha metodi propri per la navigazione libera sull'albero.
- Navigazione ufficiale: iterazione attraverso la proprietà `children`.
- **Esempio:** Funzione `getCell(tId, row, cell)` per trovare una cella specifica in una tabella: javascript function getCell(tId, row, cell) { let items = document.body.children; let i = 0; do { while (items[i].id !== tId) { i++; } } while (items[i].children[j].nodeName !== 'TBODY') { j++; } let rows = items[i].children[j].children; return rows[row].children[cell]; }

```
```

XPath

- Sintassi standard (W3C 1998) per identificare frammenti di documenti XML/HTML.
- Opera sulla struttura logica, non sintattica.
- Sintassi testuale non XML, utilizzabile in URI e attributi.
- **Esempio:** `/doc/chapter//para` (equivalente a `/child::doc/child::chapter/descendant::para`)
- **Assi XPath:**
  - `child`, `descendant`: figlio diretto/a qualunque livello.
  - `parent`, `ancestor`: genitore/antenato a qualunque livello.
  - `self`, `namespace`: nodo corrente/namespace.
  - `attribute`: attributi del nodo.
  - `preceding-sibling`, `following-sibling`: fratelli precedenti/successivi.
  - `preceding`, `following`: nodi precedenti/successivi (fuori dal nodo corrente).
  - `descendant-or-self`, `ancestor-or-self`: includono il nodo corrente.
- **Sintassi abbreviata:**
  - `child::x` → `x`
  - `attribute::a` → `@a`
  - `descendant::x` → `//x`
  - `self::*` → `./`
  - `parent::*` → `../`
- **Esempio:**
  - `/doc/chapter[5]/section[2]`
  - `chapter//para`
  - `//para`
- **Utilizzo in Javascript:** `document.evaluate(xpath)` (supporta XPath 1.0).

```
    function getCell(tId,row,cell) {
      let xpath = `//table[@id="${tId}"]//tr[${row}]/td[${cell}]`
        return document.evaluate(xpath);
    }
```

Selettori HTML Base

- Metodi DOM classici:
  - `getElementById` : se l'elemento ha un `id` .
  - `getElementsByName` : se l'elemento ha un attributo `name` .
  - `getElementsByTagName` : tutti gli elementi con un dato nome.
- Applicabili al documento intero o a un sottoalbero.

```
function getCell(tId,row,cell) {
    let table = document.getElementById(tId);
    let tbody = table.getElementByTagName('TBODY')
    return tbody.children[row].children[cell]
}
```

jQuery (cenni)

- Framework Ajax popolare (89% dei siti top nel 2017).
- Licenza open source.
- Leggero e veloce.
- Supporta navigazione/modifica DOM, eventi, comunicazioni asincrone (Ajax).
- **Sintassi:** Uso di `$` per i comandi.
  - `$()` è un alias di `jQuery()` , accetta selettori CSS.

```
$("a").click(function() { alert("Hello world!"); })
$("#clickMe").click(function() {
    $('tr .clickable').classList.toggle('d-none');
});
```

  - Esempio:

```
function getCell(tId,row,cell) {
  let css = `#${tId} tr[${row}] td[${cell}]`
  return $(css);
}
```

Selettori HTML Aggiuntivi (ispirati da jQuery)

- `getElementsByClassName` .
- `querySelector` : primo elemento che corrisponde a un selettore CSS.
- `querySelectorAll` : tutti gli elementi che corrispondono a un selettore CSS (restituisce una `NodeList` , non un array).
  - Per avere un array da una nodeList: `Array.from(document.querySelectorAll(css))`

```
function getCell(tId,row,cell) {
    let css = `#${tId} tr[${row}] td[${cell}]`
    return document.querySelectorAll(css);
}
function getCell(tId,row,cell) {
```

```
        let css = `#${tId} tr[${row}] td[${cell}]`
        return Array.from(document.querySelectorAll(css));
    }
```

"jQuery dei poveri" (esempio didattico)

- Funzione `$$` che supporta selettori CSS e XPath.

```
function $$(selector, context) {
  context = context || document.body;
  try {
    if (selector.includes("/")) { // XPath
      let f = document.evaluate(selector, context, null,
XPathResult.ANY_TYPE, null);
      let results = [];
      let res = f.iterateNext();
      while (res) {
        results.push(res);
        res = f.iterateNext();
      }
      return results;
    } else { // Selettore CSS
      return Array.from(context.querySelectorAll(selector));
    }
  } catch (e) {
    console.log("$$ responds: " + e.message);
    return [];
  }
}
```

Confronto CSS e XPath (esempi)

Selezione	CSS	XPath
Tutti i `p`	`p`	`//p`
Tutti i `p` di classe 'foo'	`p.foo`	`//p[@class="foo"]`
Tutti gli elementi di classe 'foo'	`.foo`	`//*[@class="foo"]`
Tutti gli `a` con href a google.com	`a[href~="google.com"]`	`//a[contains(@href,"google.com")]`
Tutti i `tr` in tabelle di classe "special"	`table.special tr`	`//table[@class="special"]//tr`
Il terzo `tr` nella tabella con id "id1"	`table#id1 tr:nth-child(3)`	`//table[@id="id1"]//tr[3]`
Tabelle con almeno 3 `tr`	`--`	`//table[count(.//tr)>=3]`

Tabella che contiene il `tr` con id="id3"	`table:has(tr#id3)`	`//table[.//tr[@id="id3"]]` o `//tr[@id="id3"]/ancestor::table`
Il `p` precedente all'`h1` con id="id3"	`--`	`//h1[@id="id3"]/preceeding::p[1]`

AJAX (Asynchronous JavaScript And XML)

- Tecnica per applicazioni web interattive.
- Aggiornamenti asincroni di porzioni di pagina.
- Migliora interattività, velocità, usabilità.
- **Non** è un linguaggio di programmazione, ma una combinazione di tecnologie:
  - HTML, CSS.
  - DOM (modificato con JavaScript).
  - `XMLHttpRequest` (XHR) per lo scambio di messaggi asincroni.
  - XML o JSON per i dati.

Architettura AJAX

1. L'utente interagisce con la pagina.
2. JavaScript crea una richiesta `XMLHttpRequest` .
3. La richiesta viene inviata al server (asincrona).
4. Il server elabora la richiesta e invia una risposta.
5. JavaScript elabora la risposta e aggiorna il DOM.

Pregi di AJAX

- **Usabilità:** Interattività, elimina l'attesa "click-and-wait".
- **Velocità:** Meno dati scambiati, computazione parzialmente sul client.
- **Portabilità:** Supportato dai browser principali, platform-independent, non richiede plugin.

Difetti di AJAX

- **Usabilità:** Problemi con il pulsante "back" e i segnalibri, indicizzazione difficile per i motori di ricerca.
- **Accessibilità:** Problemi con browser non visuali, richiede alternative.
- **Configurazione:** Richiede JavaScript abilitato (e ActiveX su IE).
- **Compatibilità:** Test su diversi browser necessari, alternative per browser senza supporto JavaScript.

Creare un'applicazione AJAX (passaggi chiave)

1. Creazione/configurazione della richiesta (XHR, librerie, `fetch()`).
2. Invio della richiesta.
3. *Attesa asincrona.*
4. Ricezione/analisi della risposta (o errore).
5. Modifica del DOM.

XMLHttpRequest (XHR) - Dettagli

- **`open()` :** Prepara la connessione (metodo HTTP, URL, asincrono/sincrono).
  - Asincrono: `http_request.onreadystatechange = myHandler;`

```
http_request.open('GET','http://www.example.org/file.json',true);
http_request.open('GET','http://www.example.org/file.json',false);
//Sconsigliatissimo
```

```
if (window.XMLHttpRequest) {
http_request = new XMLHttpRequest();
...
}
```

- **`send()`**: Invia la richiesta.
  - Parametro: body della richiesta (query string per POST, `null` per GET, o altri dati con `setRequestHeader`).

```
http_request.send(null);
http_request.setRequestHeader('Content-Type', 'mime/type');
```

- **Gestione della risposta (asincrona):**
  - `onreadystatechange`: funzione chiamata a ogni cambio di stato.
  - `readyState`: stato della richiesta (0: uninitialized, 1: loading, 2: loaded, 3: interactive, 4: complete).
  - `status`: codice di stato HTTP (200: OK, 404: Not Found, 500: Internal Server Error, ecc.).
  - `responseText`: risposta come testo.
  - `responseXML`: risposta come XMLDocument.

```
function myHandler() {
    if (http_request.readyState == 4) {
        if (http_request.status == 200) {
// risposta ricevuta e corretta

        } else {
// errore
        }
    }
}
```

- **Esempio XHR completo (sincrono - SCONSIGLIATO):**

```
function getData(){
  try {
  // Attiva la connessione Ajax
  myXHR = new XMLHttpRequest();
  myXHR.open("GET", "http://www.server.it/server ", false);
  myXHR.send(null);
    //legge la risposta
  let d = JSON.parse(myXHR.responseText);
    // modifica il documento corrente
    let data = prepareHTML(d);
    document.querySelector('#result').innerHTML = data ;
  }catch(error) {
      alert("Caricamentoimpossibile");
  }
}
```

- **Esempio XHR completo (asincrono - CONSIGLIATO):**

```
    function getData(){
  // Attiva la connessione Ajax
  myXHR = new XMLHttpRequest();
  myXHR.onreadystatechange = myHandler;
  myXHR.open("GET", "http://www.server.it/server ", true);
  myXHR.send(null);
}

function myHandler() {
if (myXHR.readyState == 4) {
if (myXHR.status == 200) {
   //legge la risposta
   let d = JSON.parse(myXHR.responseText);
   // modifica il documento corrente
    let data = prepareHTML(d);
    document.querySelector('#result').innerHTML = data ;
}
}
    }
```

Alternative a XMLHttpRequest

- **jQuery:** Funzione `$.ajax()` semplificata (e varianti `$.get()`, `$.post()`, `$.put()`). Supporta anche le promesse.

```
$.ajax({
    method: "GET",
    url: "http://www.server.it/server",
    success: function(d) {
        let data = prepareHTML(d);
        $('#result').html(data);
        alert("Caricamentoeffettuato");
    }, error: function(data) {
        alert("Caricamento impossibile");
    }
})

//Con promesse:

let good = (d) => {
    let data = prepareHTML(d);
    $('#result').html(data);
    alert("Caricamentoeffettuato");
};
let bad = () => {
    alert("Caricamento impossibile");
};
$.get("http://www.server.it/server").then(good, bad);
```

- **Axios:** Libreria open source basata su promesse (usata con React, Vue, Angular).

```
const axios = require('axios');
axios.get("http://www.server.it/server").then((d) => {
    let data = prepareHTML(d);
    document.querySelector('#result').innerHTML = data;
    alert('Caricamentoeffettuato');
}).catch((error) => {
    alert("Caricamentoimpossibile");
});

//Con async/await
async function getData() {
    try {
        let response = await axios.get("http://www.server.it/server");
        let data = prepareHTML(response.data);
        document.querySelector('#result').innerHTML = data;
        alert('Caricamentoeffettuato');
    } catch ((error) => {
        alert("Caricamentoimpossibile");
    }
}
getData();
```

- **Fetch:** API standard dei browser, basata su promesse.

```
fetch("http://www.server.it/server")
    .then(response => response.json())
    .then(d => {
        let data = prepareHTML(d);
        document.querySelector('#result').innerHTML = data;
        alert('Caricamentoeffettuato');
    }).catch((error) => {
        alert("Caricamentoimpossibile");
    });

      //Con parametro opzionale
    fetch("http://www.server.it/server", {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify(formData),
    }).then(response => response.json()).then(d => {
        let data = prepareHTML(d);
        document.querySelector('#result').innerHTML = data;
        alert('Caricamentoeffettuato');
    }).catch((error) => {
        alert("Caricamentoimpossibile");
    });
```

Programmazione Asincrona in JavaScript

- Caratteristica fondamentale di JavaScript.
- Esempi:
  - Richieste Ajax.
  - Gestione dei dati ricevuti via Ajax.
  - Gestione degli eventi utente.
  - `setTimeout()` . *Le callback sono funzioni indipendenti eseguite dopo il flusso principale.*

Esempi e Problemi

- **Esempio 1:** `setTimeout(..., 0)` esegue la funzione *dopo* il codice sincrono.
- **Esempio 2:** Ordine di esecuzione in chiamate Ajax (risposta corretta: "Ho messo nel forno l'arrosto"). Le funzioni di callback delle chiamate asincrone sono eseguite *dopo*.
- **Problema:** Codice bloccante (sincrono) può rendere l'applicazione non responsiva.
- **Problema:** Chiamate a catena (es. query a più database) diventano complesse.

Soluzioni per la Programmazione Asincrona

1. **Codice asincrono semplice:** Non funziona per chiamate a catena.
2. **Codice bloccante:** Sconsigliato, blocca l'interfaccia utente.
3. **Logica server-side:** Sposta la logica sul server (meno vantaggi da Ajax, complessità di gestione).
4. **Callback:** Comuni, ma JavaScript è single-thread. Le callback vengono eseguite *dopo* il flusso principale, non hanno accesso alle variabili locali della funzione chiamante (solo globali e closure). "Callback hell" con chiamate a catena.
5. **Promesse:** Oggetti che conterranno un valore in futuro. Semplificano il "callback hell". Create dalla funzione chiamante, mantenute dalla funzione chiamata.

```
    function searchSomeProducts(query) {
  promiseSearch('orders',query).then( (orders) => {
    let output = prepareTable(orders);
    document.getElementById("display").appendChild(output);
  })}

function promiseSearch(db, query) {
  let async = true;
  return new Promise(function(resolve) {
    let DB = dbConnect("http://wwww.site.com",db,acct,pwd)
        let DB = DB.search(query, async, function(data) {
      resolve(data);
    }
  });}
```

6. **Generator/yield:** (ES2016) Funzioni che possono essere interrotte e riprese. `yield` mette in pausa l'assegnazione. `next()` fa proseguire l'esecuzione. Utili con gli iteratori.
7. **async/await:** (ES2017) Il modo più semplice. `async` indica una funzione asincrona. `await` sospende l'esecuzione fino al completamento di una promessa. Mantiene lo stack e il contesto.

```
    async function searchBetterProducts(query) {
    let async = true;
    let orderDB = dbConnect("http://wwww.site.com", "orders", acct, pwd)
```

```
    let orders = await orderDB.search(query, async);
    let productDB = dbConnect("http://wwww.site.com", "products", acct, pwd)
    let products = await productDB.search(orders, async);
    let opinionDB = dbConnect("http://wwww.site.com", "opinion", acct, pwd)
    let opinions = await opinionDB.search(products, async);
    let output = prepareTable(orders, products, opinions);
    document.getElementById(“display”).innerHTML = output;
}
```

8. **Promise.all():** Per chiamate asincrone indipendenti. Restituisce una promessa che si risolve quando tutte le promesse in input si sono risolte. Si può usare l'operatore spread (`...`) per passare i risultati.
9. **`for await...of...`:** (ES2023) Combina generatori, iteratori e `await/async`. Meno diffuso.

- **Esercizio:** Sistema di connessioni parallele (link fornito).

Questo riassunto copre tutti i punti principali del PDF, organizzati in modo logico e con esempi di codice per chiarire i concetti. Sono evidenziati i punti chiave e le differenze tra le varie tecniche.