Laborator 09 - DOM scripting

Obiective: În urma parcurgerii acestui laborator studentul va:

- înţelege modul de funcţionare al DOM-ului în browser
- folosi câteva metode pentru manipularea DOM-ului
- înţelege beneficiile folosirii unui framework ca jQuery

DOM

Rolul Javascript în aplicaţiile Web este de a oferi programatorului un mod de a interacţiona cu browserul. Tot ce se întâmplă în pagina web **încărcată în browser**, se întâmplă prin execuţia de cod Javascript.

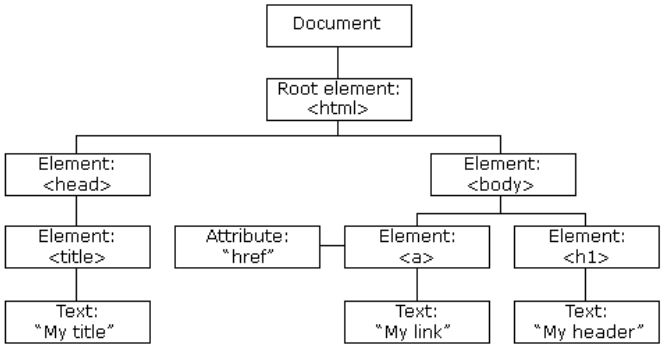
DOM-ul este o convenţie cross-platform şi cross-language de reprezentare a unui document XML, HTML sau XHTML ca obiect şi de manipulare a elementelor sale constitutive. El a fost standardizat de W3C ca urmare a "browser wars" dintre IE şi Netscape Navigator, de la sfârşitul anilor '90. Principalul beneficiu adus de adoptarea standardului a fost faptul că dezvoltatorii nu au mai fost nevoiți să scrie codul de două ori, odată pentru IE și odată pentru restul browserelor :).

DOM-ul are mai multe părți:

- **core**: care cuprinde constructele folosite pentru manipularea nodurilor în toate documentele care pot fi parsate în DOM;
- **html**: ce conține constructe specifice manipulării documentelor HTML (și al căror suport variază între implementările din diferite browsere);
- **css**: care se ocupă de componenta de reprezentare vizuală a paginilor web și de manipularea proprietăților CSS și alte componente care sunt în afara obiectivelor acestui laborator.

Mai multe detalii puteți vedea pe Mozilla Developer Core.

Figura de mai jos este un exemplu de DOM (x)HTML:



DOM-ul este alcătuit din **noduri**, care pot fi de mai multe tipuri. Dintre acestea, cele mai importante tipuri sunt element, atribut și text. Verificarea tipului unui nod se face verificând proprietatea `nodeType`. Valoarea returnată este un întreg, nu un string (1 pentru element, 2 pentru atribut, 3 pentru text). De asemenea, nodurile au o proprietate `nodeValue` prin care se poate manipula valoarea nodului.

Toate browserele oferă un API în Javascript de manipulare a DOM-ului.

Referenţierea nodurilor

În browser, toate elementele din DOM au un părinte comun, obiectul **`document`**. Acesta, la rândul lui, are ca părinte obiectul `window`, care se referă la fereastra(tabul) din browser în care e redat conţinutul paginii. Mededele enumerate mai jos sunt folosite pentru referenţierea elementelor din DOM, pornind de la `document`. Ele fac parte din standardul W3C:

- **`document.getElementById()`** - intoarce elementul cu id-ul precizat.
- **`document.getElementsByName()`** - intoarce o lista ce contine elementele care au atributul `name` corespunzător.
- **`document.getElementsByTagName()`** - intoarce o lista ce contine elementele cu tagurile corespunzătoare din document.

Un element, la rândul său, are proprietăţi şi metode ce pot fi folosite pentru referenţierea altor elemente cu care acesta relaţionează:

- `element.childNodes[]`
- `element.firstChild`
- `element.lastChild`

Search

- Repartizare teme
- Catalog PW

Laboratoare

- Laborator 01 - Introducere
- Laborator 02 - BD in PHP
- Laborator 03 - Arrays, Magic Methods
- Laborator 04 - (X)HTML, CSS
- Laborator 05 - Formulare HTML, Persistența Datelor
- Laborator 06 - Securitate I
- Laborator 07 - Securitate II
- Laborator 08 - JavaScript
- Laborator 09 - DOM scripting
- Laborator 10 - AJAX
- Laborator 11 - Web Frameworks

Teme

- Tema 01 - API pentru BD
- Tema 02 - Template System & Controller
- Tema 03 - Generator de formulare
- Tema 04 - Tree Web UI

Login

- `element.nextSibling`
- `element.parentNode`
- `element.previousSibling`
- `element.hasChildNodes`
- **`element.getAttribute(attrName)`**
- **`element.setAttribute(attrName, attrValue)`**

Toate metodele prezentate mai sus fac parte din DOM core și ar trebui să existe în toate browserele moderne.

Crearea, duplicarea și inserarea nodurilor din/în DOM

Următoarele metode sunt folosite pentru a crea elemente și a le integra într-un DOM existent:

- **`document.createElement(element)`**, crează un element de tipul tagului dat ca parametru funcției.
- **`document.createTextNode(text)`**, crează un nod text cu conținutul dat de parametrul primit.
- **`node.cloneNode(deep)`**, duplică nodul pe care este aplicată. Parametrul determină dacă se vor clona și copiii nodului.
- **`element.appendChild(newNode)`**, atașează `newNode` ca ultimul copil al `element`.
- **`element.insertBefore(newNode, targetNode)`**, atașează `newNode` ca și copil al `element`, înaintea `targetNode`. `targetNode` trebuie să fie copil al `element`. Dacă `targetNode` nu e specificat, funcția se comportă exact ca `appendChild()`.
- **`element.removeChild(node)`**, șterge `node`, dacă este copil al `element`.
- **`element.replaceChild(newChild, oldChild)`**, înlocuiește `oldChild` cu `newChild`.

Elementele pot fi create și cu ajutorul metodelor `document.write()` și `element.innerHTML()`. Problema lui `document.write()` este că este legat de locul în document unde este apelat. Problema lui `innerHTML` este că nu face parte din standard; deși poate părea mai facil să îl folosiți, soluția elegantă este folosirea constructelor standard puse la dispoziție în DOM. De asemenea, dacă serviți documentele XHTML mimetype-ul corect - `application/xhtml+xml` - nici `document.write()` nici `element.innerHTML()` nu vor mai funcționa.

Evenimente

Evenimentele se pot seta inline, prin atribute ale elementelor HTML:

```
<img src="event.jpg" id="eventImage" onmouseover="if (this.getAttribute('src') == 'event.jpg') a
```

Acest mod de a aplica evenimente elementelor din DOM nu este recomandat, deoarece amestecă structura unui document cu comportamentul său și nu este deloc lizibil. Dacă luăm în considerare faptul că în DOM evenimentele sunt de fapt proprietăți ale elementelor, avem la dispoziție și următorul tip de construcție:

```
var image = document.getElementById('eventImage');
image.onmouseover = function () {
  if (this.getAttribute('src') == 'event.jpg') {
    alert('ping');
  }
}
```

Modul de funcționare al evenimentelor din DOM Level 0 este simplu de înțeles. Obiectul `document` captează un eveniment, îl delegă la elementul țintă unde va fi executat de handlerul ce îi corespunde. DOM Level 2 a introdus un nou model pentru tratarea evenimentelor în browser. Modelul este suportat de majoritatea browserelor moderne (excepție făcând, bineînțeles, IE 6 și 7). Față de tratarea simplă a evenimentelor, acest model aduce în plus propagarea evenimentului și la *nodurile părinte*. Evenimentul se desfășoară în trei faze:

- faza de **capturare**, în care evenimentele se propagă de la obiectul `document` în jos, spre țintă. Dacă vreunul din părinții nodului țintă au vreun event handler specific înregistrat pentru această fază el este declanșat.
- **executarea** handlerelor pe nodul țintă
- propagarea "în sus" (**bubbling**), unde evenimentul se propagă de la țintă înapoi spre `document`, trecând din nou prin toți părinții.

Din motive de simplitate, în acest laborator vom lucra cu evenimente DOM Level 0, dar este important să știți că există deja definit un alt model și că are șanse mari să devină modul de utilizare preferat în viitor.

jQuery

Framework-urile Javascript au apărut din cauza inconsistențelor dintre modul de tratare al standardelor în diferite browsere și din cauza modului în care este proiectat API-ul DOM. jQuery este de departe cel mai folosit. A fost lansat în 2006 de către John Resig (Mozilla) și de atunci a câștigat constant în popularitate cu trecerea timpului.

jQuery aduce numeroase metode noi și "scurtături" în lucrul cu DOM-ul și cu Javascript în general, în contextul paginilor web. Deoarece documentația este extensivă și există numeroase site-uri care încearcă să prezinte cât mai bine frameworkul, nu voi trata aici în detaliu caracteristicile sale. Ce este important să rețineți este faptul că, prin folosirea jQuery aveți la dispoziție un API puternic, proiectat

mai bine și mai user-friendly decât cel al DOM-ului și care vă garantează o funcționare bună a codului scris de voi în 99% dintre situații, indiferent de browser.

Metodele puse la dispoziție de jQuery sunt grupate în funcție de obiectivele pe care își propun să le îndeplinească:

- Referențierea elementelor se face cu ajutorul obiectului jQuery (`jQuery()` sau `$()`) și a selectorilor CSS suportați. De asemenea, jQuery pune la dispoziție și numeroase metode de traversare a DOM-ului. De exemplu:

```
var element = $('div.container div#caption > img');
```

- Manipularea DOM-ului se face cu ajutorul unor metode care se aplică direct unui element referențiat prin obiectul jQuery. De exemplu:

```
$('div.article p:first-child').addClass('abstract').after('<hr />');
```

- jQuery permite atașarea de evenimente și oferă anumite funcționalități care nu sunt disponibile direct din DOM. De exemplu, metoda `.live()` atașează evenimente pe toate elementele care se potrivesc unui anumit selector, chiar dacă ele au fost inserate în DOM după ce metoda a fost apelată:

```
$('.clickme').live('click', function() {
  // Live handler called.
});
$('body').append('<div class="clickme">Another target</div>');
// divul inserat va avea setat evenimentul declarat mai sus
```

Principii generale pentru dezvoltarea cu Javascript în pagini Web

Faptul că programarea în Javascript are un "low barrier of entry" este o sabie cu două tăișuri. Pe de-o parte, este foarte ușor pentru oricine să facă scripturi simple care să adauge funcționalitate paginilor web. Pe de alta, există foarte multe pagini web ce conțin cod gândit prost și care le face inaccesibile pentru unii potențiali utilizatori. Trebuie să aveți în vedere că nu toată lumea are Javascript activat și că diferite browsere (și/sau terminale) nu suportă orice funcționalitate.

Cele mai importante trei principii de avut în vedere sunt: **graceful degradation**, **unobtrusive javascript** și **backwards compatibility**.

Graceful degradation

Graceful degradation se referă la faptul că Javascript nu ar trebui să fie elementul determinant pentru funcționarea unei pagini web. Totul ar trebui să funcționeze și atunci când Javascript este dezactivat, fie și fără o comportare identică. Secvența de mai jos este un exemplu de "așa nu" din punct de vedere al graceful degradation:

```
<a href="#" onclick="popUp('http://www.example.com/'); return false;">Example<a>
```

V-ați putea întreba cine nu are Javascript activat în zilele noastre. Un răspuns posibil ar fi un search-engine spider :). Alt răspuns ar putea fi un utilizator cu dizabilități. Codul de mai sus ar putea fi rescris pentru a permite urmarea linkului și atunci când Javascript nu este disponibil:

```
<a href="http://www.example.com/" onclick="popUp(this.getAttribute('href')); return false;">E
```

Unobtrusive Javascript

Unobtrusive Javascript se referă la separarea cât mai atentă a comportamentului (js) de structura documentului (html). Această stratificare conferă o flexibilitate mai mare atât codului cât și markupului și reduce posibilitatea apariției unor erori care să afecteze funcționarea paginii. Fie urmatorul exemplu in care codul `Javascript` este scris intr-un fisier separat:

example.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8" />
<script type="text/javascript" src="jsExample.js"></script>

</head>
<body>
 <a href="http://www.example.com/" class="popup">Example</a>
</body>
</html>
```

jsExample.js

```
var links = document.getElementsByTagName("a");
for (var i=0; i<links.length; i++) {
  if (links[i].classname == "popup") {
    links[i].onclick = function() {
```

```
      popUp(this.getAttribute("href"));
      return false;
    }
  }
}
```

Observati linia `<script type="text/javascript" src="jsExample.js"></script>` care include o sursa avand cod `Javascript`, intr-un fisier `html`.

Backwards compatibility

Backwards compatibility se referă la luarea în considerare a faptului că diferite medii de execuție oferă diferite grade de suport pentru diferite elemente ale DOM-ului. Există două tehnici pentru a verifica dacă un browser suportă metodele folosite. Prima se numește `object detection`. Constă în verificarea metodelor și proprietăților folosite din DOM și terminarea scripturilor în cazul în care ele nu sunt disponibile:

```
if (!document.getElementsByTagName) return false;
```

Cea de-a doua tehnică este `browser sniffing`, verificarea browserului și a versiunii pentru a determina ce cod (și dacă) va fi folosit. Este nerecomandată pentru că, uneori, browserele *mint* :) și, oricum, numărul mare de browsere și versiuni duc la branch-uri foarte complicate și ilizibile. Pentru backwards compatibility, object detection este metoda preferată.

Taskuri

Descărcați această arhivă.

1. Adăugați funcționalitate paginii `index.html` din folderul `dom` astfel încât să se comporte ca o galerie de imagini:
   - la orice moment să fie vizibilă o imagine în pagină.
   - fiecare link apăsat să determine afișarea altei imagini.
   - să se permită o navigare de bază (înainte / înapoi) între imagini.
   - folosiți thumbnailurile pentru a crea o navigare îmbunătățită.
2. Același lucru, dar folosind jQuery (folosiți folderul jQuery din arhivă).

Total: 12p (6p partea de DOM și 6p partea de jQuery).

Taskurile vă lasă libertatea de a implementa cum considerați voi mai bine această navigare. Trebuie să aveți în vedere, însă, principiile de dezvoltare enunțate în acest laborator.

Show pagesource Old revisions Back to top