XI. Dokumentum Objektum Modell (DOM)

1. Mi a DOM?

Magában a JavaScript nyelvi alapjául szolgáló ECMAScript szabványban nincsen definiálva, hogy a HTML elemek hogyan érhetőek el. Ez érthető is, hiszen a JavaScript nyelv független az azt futtató környezettől. Ha a böngészőben megjelenített HTML dokumentumbeli elemek elérése a cél, akkor azt magának a JavaScriptet futtató környezetnek, azaz a böngészőnek kell biztosítania. **A HTML elemek eléréséhez** tehát **a böngésző tesz lehetővé olyan kapcsolódási pontokat (interfészeket), amelyeken keresztül a HTML dokumentum JavaScripttel manipulálható.**

A HTML elemekhez való hozzáférés JavaScriptben egy szabványos interfészen, a Dokumentum Objektum Modellen keresztül történik. A Document Object Model (DOM) mint az oldal szerkezetéért felelős HTML elemek JavaScript objektum reprezentánsai, amelyek ugyanolyan fastruktúrába szervezve a megjelenítés alapját képezik, és amelyen keresztül a dokumentum programozhatóvá válik. Magyarán szólva **a DOM nem más, mint az oldal** – általánosabb fogalmazva minden fastruktúra, pl. HTML és XML **– szerkezetének programozási felülete, így amikor egy weboldal betöltődik, a böngésző létrehozza az oldal dokumentumobjektum-modeljét**. *A DOM-on keresztül elérhetjük, módosíthatjuk, törölhetjük az elemeket, új elemeket hozhatunk létre.*

A DOM története[1]

A DOM története összefonódik a kilencvenes évek végén a Netscape Navigator és a Microsoft Internet Explorer közötti "böngészőháborúk" történetével, valamint a JavaScript és a JScript történetével, amelyek az első szkriptnyelvek, amelyeket széles körben alkalmaztak Webböngészők JavaScript motorjai. ***A JavaScriptet a Netscape Communications adta ki 1995-ben a Netscape Navigator 2.0-ban. A Netscape versenytársa, a Microsoft a következő évben kiadta az Internet Explorer 3.0-t a JavaScript JScript nevű újraimplementálásával.*** A JavaScript és a JScript segítségével a webfejlesztők weblapokat hozhatnak létre kliensoldali interaktivitással. ***A felhasználók által generált események észlelésének és a HTML-dokumentum módosításának korlátozott lehetőségei ezen nyelvek első generációjában végül „DOM Level 0" vagy „Legacy DOM" néven váltak ismertté.*** A 0. DOM-szinthez nem fejlesztettek ki független szabványt, de azt részben leírták a HTML 4 specifikációiban.

A régebbi DOM korlátozott volt az elérhető elemek tekintetében. Az űrlap-, hivatkozás- és képelemekre hierarchikus névvel lehetett hivatkozni, amely a gyökérdokumentum objektummal kezdődik. A hierarchikus név felhasználhatja a bejárt elemek neveit vagy szekvenciális indexét. (Például egy űrlap bemeneti eleme document.formName.inputName vagy document.forms[0].elements[0] néven érhető el.) ***A Legacy DOM lehetővé tette az ügyféloldali űrlapellenőrzést és az olyan egyszerű interaktivitást, mint például az eszköztippek létrehozása.***

1997-ben a Netscape és a Microsoft kiadta a Netscape Navigator, illetve az Internet Explorer 4.0-s verzióját, amely támogatja a Dynamic HTML (DHTML) funkciót, amely lehetővé teszi a betöltött HTML-dokumentumok módosítását. A DHTML kiterjesztéseket igényelt a kezdetleges dokumentumobjektumhoz, amely elérhető volt a régi DOM-megvalósításokban. Bár a Legacy DOM-megvalósítások nagyrészt kompatibilisek voltak, mivel a

[1] *forrás: https://en.wikipedia.org/wiki/Document_Object_Model#Levels_of_DOM*

JScript JavaScript-en alapult, a DHTML DOM-bővítményeket az egyes böngészőgyártók párhuzamosan fejlesztették, és továbbra is inkompatibilisek maradtak. A DOM e verziói „Köztes DOM” néven váltak ismertté.

Az ECMAScript szabványosítása után a W3C DOM munkacsoport megkezdte a szabványos DOM specifikáció kidolgozását. Az elkészült specifikáció, amelyet "DOM Level 1-nek" neveznek, 1998 végén a W3C ajánlása lett. 2005-re a W3C DOM nagy részét jól támogatták az általános ECMAScript-kompatibilis böngészők, köztük az Internet Explorer 6 (2001-től), az Opera, Safari és Gecko alapú böngészők (például Mozilla, Firefox, SeaMonkey és Camino).

A W3C DOM Munkacsoport közzétette végső ajánlását, majd 2004-ben feloszlott. A fejlesztési erőfeszítések a WHATWG-hez vándoroltak, amely továbbra is fenntartja az élő standardot. 2009-ben a Web Applications csoport átszervezte a DOM-tevékenységeket a W3C-n. 2013-ban az előrehaladás hiánya és a HTML5 közelgő megjelenése miatt a DOM 4. szintű specifikációját átadták a HTML-munkacsoportnak, hogy meggyorsítsák annak befejezését. Eközben 2015-ben a Web Applications csoportot feloszlatták, és a DOM felügyeleti jogkörre a Web Platform csoportra szállt. A DOM Level 4 2015-ös kiadásától kezdve a W3C új ajánlásokat hoz létre a WHATWG szabvány pillanatképei alapján.

- A ***DOM 1. szintje*** egy teljes modellt biztosított egy teljes HTML- vagy XML-dokumentumhoz, beleértve a dokumentum bármely részének megváltoztatásának eszközeit.
- A ***DOM Level 2*** 2000 végén jelent meg. Bevezette a getElementById függvényt, valamint egy eseménymodellt és támogatja az XML névtereket és a CSS-t.
- A 2004 áprilisában megjelent ***DOM Level 3*** támogatást nyújtott az XPath-hoz és a billentyűzetes eseménykezeléshez, valamint egy interfészt a dokumentumok XML-ként történő sorosításához.
- A HTML5 2014 okt-ében jelent meg. A HTML5 egy része felváltotta a DOM 2. szintű HTML-modulját.
- A ***DOM Level*** 4 2015-ben jelent meg.

A HTML dokumentum böngészőbe történő betöltésének folyamata

A DOM megértéséhez nézzük meg egy HTML dokumentum böngészőbe történő betöltésének folyamatát! **A HTML dokumentum egy fastruktúra, ahol HTML elemek egymás mellett vagy egymásba ágyazva jelennek meg.**

1. mintapélda

HTML-kód:

```
<!doctype html>
<html lang="hu">
    <head>
        <meta charset="utf-8">
        <title>Üdvözlés</title>
    </head>

    <body>
        <form>
            Név: <input type="text">
            <br>
            <input type="button" value="Köszönj!">
        </form>
    </body>
</html>
```

Megjelenés a böngészőben:

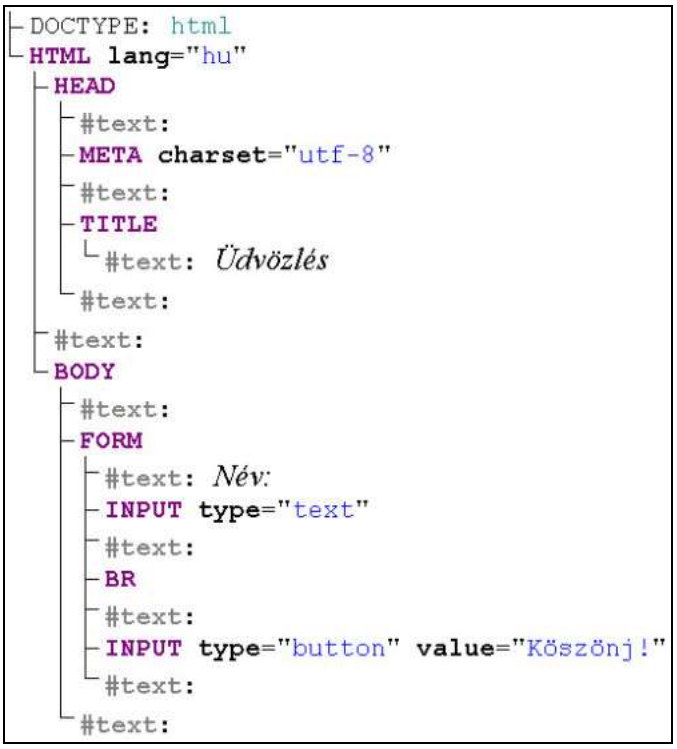
1. ábra

Minden elemnek legfeljebb egy szülője van, aki tartalmazza őt, és minden elemnek lehetnek gyerekei, akiket az ő nyitó- és záróeleme tartalmaz; az elemmel egy szinten lévő más elemeket pedig testvéreknek nevezzük. (Ugyancsak fastruktúrát alkot például a könyvtárszerkezet egy fájlrendszerben.) Az ***1. mintapéldában*** látható egy minta (***1. ábra***) teljes HTML oldal forráskódja, amelyben a **`head`** elem szülője a **`html`**, gyerekei a **`meta`** és a **`title`**, testvére pedig a **`body`** elem. A fastruktúrában a szöveges elemek is megjelennek.

A szöveges HTML állomány betöltésekor a böngésző minden egyes HTML elemnek (még a szövegeseknek is) létrehoz egy JavaScript objektumot, és ezeket az objektumokat ugyanolyan fastruktúrába rendezi, mint ahogy az az eredeti HTML elemeknél volt. A folyamat eredményeképpen kialakul egy objektumhierarchia, ez a DOM. Az ***1. mintapéldában*** megadott HTML kód betöltése után a ***2. ábrán*** látható objektumszerkezet alakul ki.

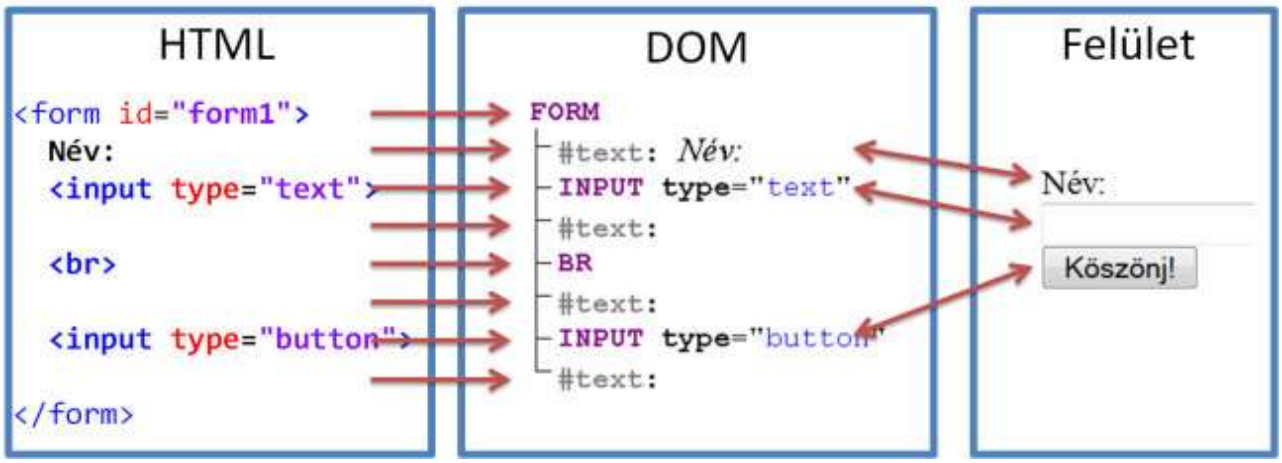
A böngésző az oldal megjelenítését a DOM alapján végzi el. A DOM és a kirajzolt oldal között nagyon szoros kapcsolat van. Ha változik a DOM, akkor az az oldalon azonnal érvényesül. Ugyanez visszafele is igaz: ha valamit változtatunk az oldalon, akkor a változások rögtön megjelennek a DOM-ban is. Ezt a leírt folyamatot a ***3. ábrán*** foglaltuk össze. (Az ábra jól szemlélteti azt, hogy az egyes HTML elemeknek milyen DOM objektumok felelnek meg.)

Az 1. mintapélda DOM hierarchiája:



2. ábra

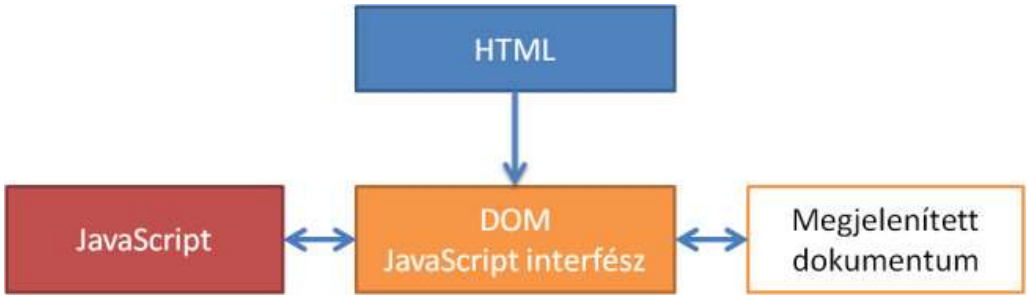
Ez a leképezés egyszeri, a betöltés során jön létre. A felület a DOM alapján rajzolódik ki, az oda-vissza nyilak azt szemléltetik, hogy a DOM és a felhasználói felület kölcsönös, élő kapcsolatban vannak egymással.



3. ábra

A DOM mint programozói interface

A HTML forráskód, a belőle létrejövő DOM, a felhasználói felület és a JavaScript viszonyát egy sematikus ábrán (***4. ábra***) szemléltethetjük. Az ábra a fent említett kapcsolaton túl azt is hangsúlyozza, hogy a DOM az az interfész, amelyen keresztül JavaScript kódból a felületi elemeket elérhetjük, módosíthatjuk. Mivel a DOM JavaScript objektumokból áll, olyanokból, mint amilyeneket a nyelvi részben láthattunk, a felület programozása nem áll másból, mint ezen objektumok adattagjainak lekérdezéséből, módosításából, metódusainak a meghívásából.



4. ábra

A DOM mint szabvány

Szerencsére az, hogy pontosan hogyan hívják a DOM egyes objektumait, milyen adattagjaik és metódusaik lehetnek, nem az egyes böngészőkre van bízva, hanem szabvány határozza meg. A DOM nemcsak HTML elemek modellezésére szolgál, hanem minden fastruktúra alapú hierarchiának kezelésére ír le egy szabványos interfészt. A HTML mellett ilyen például az XML dokumentumok szerkezete is. A DOM szabvány is – a HTML-hez hasonlóan – több lépésben fejlődött (részletesebben: https://hu.wikipedia.org/wiki/Document_Object_Model, ill. What are DOM levels?, teljes részletezettséggel: DOM Living Standard).

A DOM egy W3C szabvány, amely lehetőséget ad a dokumentumok elérésére: *"A W3C Document Object Model (DOM) egy platform- és nyelvsemleges interfész, amely lehetővé teszi a programok és szkriptek számára, hogy dinamikusan hozzáférjenek és frissítsék a dokumentumok tartalmát, szerkezetét és stílusát."*

A W3C DOM szabvány három különböző részre oszlik:

- Core DOM: szabványos modell minden dokumentumtípushoz
- XML DOM: szabványos modell XML dokumentumokhoz
- HTML DOM: szabványos modell HTML dokumentumokhoz

Amikor HTML oldalak esetében DOM-ról beszélünk, akkor több dolgot foglalunk össze ezen név alatt. **Maga a DOM egy általános fastruktúrával határozza meg annak felépítését, elemeinek típusát, azok adattagjait és működtető metódusait. Ezt egészíti a HTML DOM, amely minden egyes HTML elemnél meghatározza azokat a plusz tulajdonságokat és metódusokat, amelyek a HTML elemek teljes körű kezeléséhez szükségesek.** Végül **a DOM Events a DOM elemekhez tartozó események működtetését határozza meg**. A DOM tehát inkább a fastruktúrához tartozó műveleteket és tulajdonságokat határozza meg, a HTML DOM pedig a HTML specifikumokat tartalmazza. A böngészőben használva azonban ezek egyszerre jelennek meg, és válnak elérhetővé.

A DOM tehát minden egyes HTML elemről rengeteg információt tárol egy-egy JavaScript objektum képében, és nagyon sokféle műveletet enged elvégezni velük. Ezekkel gyakorlatilag az oldal tetszőleges megváltoztatása lehetséges. A főbb műveletcsoportok: faszerkezet bejárása (elemek elérése), új elemek létrehozása, meglévő elemek módosítása és törlése.

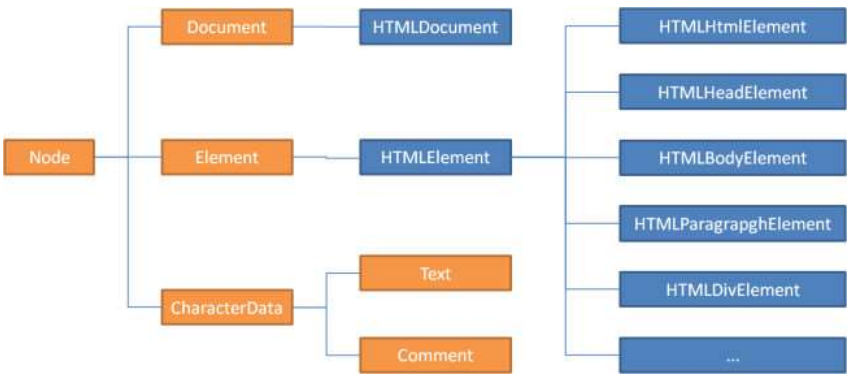
A DOM elnevezései

Ahogy arról korábban már volt szó, a böngészőkben elérhető DOM működését több szabvány határozza meg. A DOM alapját az ún. DOM Core szabvány jelenti, amelyben egy csomópontokból álló általános fastruktúra ábrázolásához és működtetéséhez szükséges információk és műveletek vannak leírva. A HTML dokumentum egy speciális elemekből álló fastruktúra, speciális többletfunkcionalitását a HTML DOM szabvány határozza meg. A következőkben DOM alatt a HTML DOM-ot értjük.

A DOM nem más, mint csomópontok fája. Sokféle típusú csomópont létezik, és ez meghatározza lehetséges tulajdonságaikat és metódusaikat is. Némelyik típusú csomópontnak lehetnek további gyerekei, mások csak levélelemként szerepelhetnek. A teljesség igény nélkül tekintsük át a fontosabb csomóponttípusokat:

- ***dokumentum (document):*** a dokumentum reprezentálja az egész HTML dokumentumot. Ez a dokumentumfa gyökere, ő biztosítja az elsődleges hozzáférést bármilyen dokumentumbeli adathoz. Mivel minden további csomóponttípus csak a dokumentumon belül létezhet, ezért ő biztosít néhány kényelmes metódust az elemek elérésére és létrehozására. JavaScriptben `window.document` vagy rövidebben `document` kulcsszóval hivatkozunk rá.
- ***elem (element):*** minden dokumentum egy vagy több elemből áll. Az elemek határát a nyitó és záró tagok jelölik ki, illetve léteznek üres elemek is, amelyek önmagukban tekinthetők lezártnak. Minden elemnek eltérő neve van, és lehetnek attribútumai is. Elem pl. a `<html>`, `<p>` stb.
- ***attribútum (attribute):*** elemhez kapcsolt további információ leírására szolgál. Minden attribútumnak van neve és értéke, pl. az `<img>` elemnek az `src` vagy az `alt` az attribútuma.
- ***szöveges csomópont (node):*** szöveges adatot tartalmazó csomópont. Gyakorlatilag az oldalakon megjelenő szövegek ábrázolására szolgál. A DOM feldolgozását nehezíti, hogy az egyes elemek elválasztására beírt karakterek (pl. újsor karakter) is szöveges csomópontként jelennek meg.

Az egyes csomóponttípusok viszonyát az alábbi ábra mutatja be. Jól látható, hogy az általános csomópont (Node) interfészt hogyan specializálják a belőle származó további interfészek. A DOM Core definiálja a Document interfészt, a HTML DOM ezt bővíti tovább a HTML-beli specialitásokkal a HTMLDocument interfészben. Ugyanígy az általános Element interfésznek



5. ábra

a HTMLElement a HTML-beli kiterjesztője, és ebből származnak az egyes HTML elemeknek megfelelő speciális interfészek: HTMLBodyElement, HTMLParagraphElement, HTMLDivElement, stb. A szöveges csomópontot (Text) megelőzi egy általánosabb CharacterData interfész, ugyanis a HTML-beli megjegyzéseknek is külön interfésze van (Comment), amely ebből származik.

Ha egy dokumentum felépítését tekintjük, akkor egy dokumentum általában elemekből és szöveges csomópontokból áll. (Ennél jóval többféle csomópontot definiál a szabvány, de ez számunkra csak feleslegesen bonyolítaná a képet). Az elemek általában további elemeket és szöveges csomópontokat tartalmazhatnak, a szöveges csomópontok viszont kizárólag levélelemek lehetnek.

A HTML DOM tulajdonságai és metódusai

A HTML DOM metódusok olyan műveletek, amelyeket a HTML elemeken hajthatunk végre.
A HTML DOM tulajdonságai a HTML elemek értékei, amelyeket beállíthatunk vagy ha már be vannak állítva, akkor módosíthatjuk (megváltoztathatjuk a már meglévő értékeit).

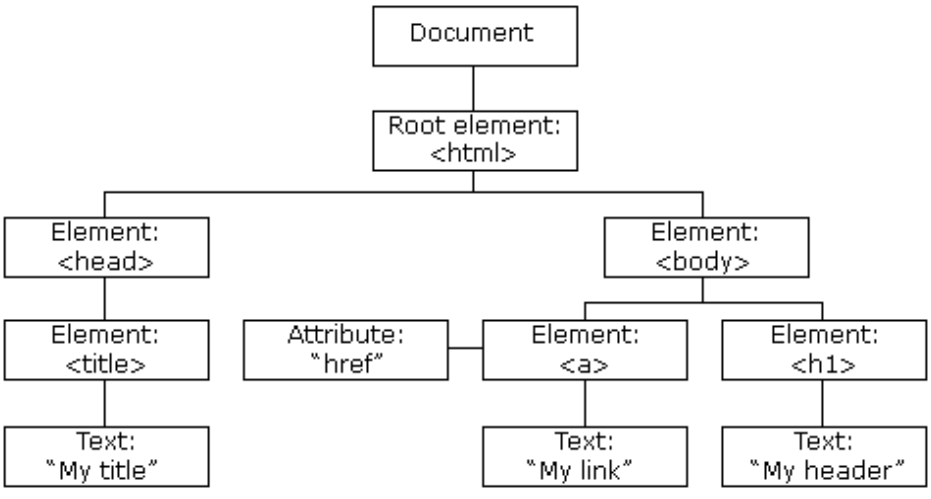
A DOM programozási lehetősége a JavaScripttel (és más programozási nyelvekkel) érhető el. Mivel a DOM-ban minden HTML elem objektumként van definiálva, ezért ez a programozási felület az egyes objektumok tulajdonságaival és metódusaival dolgozik:

- ***tulajdonság (property):*** egy olyan érték, amelyet kinyerhetünk (megkaphatjuk, hogy mi van aktuálisan megadva) vagy beállíthatunk, pl. egy kép (**`img`**) objektum elérési útja (**`src`**),
- ***metódus (method):*** egy olyan művelet, amelyet elvégezhetünk, pl. HTML elem hozzáadása.

2. mintapélda

HTML:
```
<p id="demo"></p>
```
szkript:
```
document.getElementById("demo").innerHTML = "Hello World!";
```
getElementById = metódus *innerHTML = tulajdonság*

Összefoglalás



6. ábra

Magában a JavaScript nyelvi alapjául szolgáló ECMAScript szabványban nincsen definiálva, hogy a HTML elemek hogyan érhetőek el. Ez érthető is, hiszen a JavaScript nyelv független az azt futtató környezettől. Ha a böngészőben megjelenített HTML dokumentumbeli elemek elérése a cél, akkor azt magának a JavaScriptet futtató környezetnek, azaz a böngészőnek kell biztosítania. A HTML elemek eléréséhez tehát a böngésző tesz lehetővé olyan kapcsolódási pontokat (interfészeket), amelyeken keresztül a HTML dokumentum JavaScript-

tel manipulálható. **A HTML elemekhez való hozzáférés JavaScriptben egy szabványos interfészen, a Dokumentum Objektum Modellen (röviden DOM) keresztül történik.** Amikor egy weboldal betöltődik, **a böngésző létrehozza az oldal dokumentumobjektum-modelljét: ez a HTML objektumok fájaként (fastruktúrájaként) épül fel** (***6. ábra***).

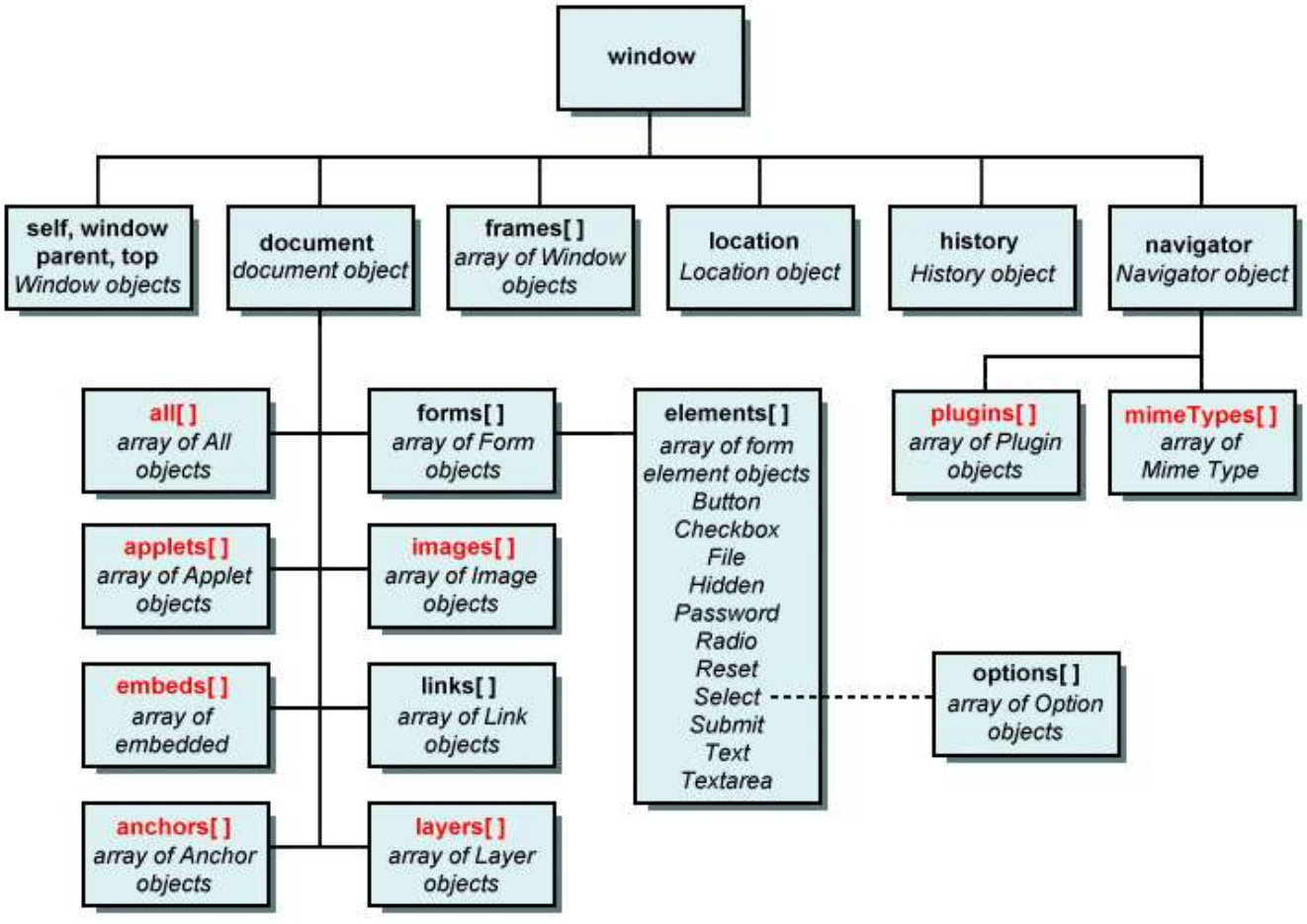
A JavaScript ennek alapján képes megváltoztatni az oldal összes HTML elemét, azok attribútumait és CSS-jellemzőit, új elemeket, attribútumokat és stílusjegyeket adhat a HTML dokumentumhoz, eltávolíthatja azokat, reagálhat az oldalon bekövetkező eseményekre, új eseményeket hozhat létre.

2. A HTML DOM objektum

A HTML DOM egy szabványos objektummodell és programozási felület a HTML számára. Meghatározza:

- a HTML elemeket objektumként,
- az összes HTML elem tulajdonságát,
- az összes HTML elem elérésének módszereit,
- az összes HTML elem eseményeit,

azaz **a HTML DOM a HTML elemek elérésének, módosításának, hozzáadásának és törlésének szabánya**.



7. ábra

Amikor a javascript böngészőn fut, van egy „root” objektum, amelyet „Window”-nak hívnak (***7. ábra***). A window egy globális objektum a JS-kódhoz és egyben a böngészőablakot is képviseli. A document objektum az oldal fő „belépési pontja”. Az oldalon bármit módosíthatunk vagy létrehozhatunk vele, és az oldal összes tartalmát módosítható objektumként ábrázolja. **A HTML DOM dokumentumobjektum a weboldalon található összes többi objektum tulajdonosa, így az a teljes weboldalt képviseli.** Ha egy HTML-oldal bármely eleméhez szeretne hozzáférni, mindig a dokumentumobjektum elérésével kell kezdenie.

HTML DOM dokumentumobjektum tulajdonságai

tulajdonság	funkció	DOM szint
`document.anchors`	visszaadja az összes **<a>** elemet, amelynek van **name** attribútuma	1
`document.baseURI`	visszaadja a dokumentum abszolút URL-jét	3
`document.body`	visszaadja a **<body>** elemet	1
`document.cookie`	visszaadja a dokumentum sütijeit	1
`document.doctype`	visszaadja a dokumentum doctype-ját	3
`document.documentElement`	visszaadja a **<html>** elemet	3
`document.documentMode`	visszaadja a böngésző által használt módot	3
`document.documentURI`	visszaadja a documentum URI[2]-ját	3
`document.domain`	visszaadja a dokumentumszerver tartománynevét	1
`document.embeds`	visszaadja az összes **<embed>** elemet	3
`document.forms`	visszaadja az összes **<form>** elemet	1
`document.head`	visszaadja a **<head>** elemet	3
`document.images`	visszaadja az összes **<img>** elemet	1
`document.implementation`	visszaadja a DOM implementációt	3
`document.inputEncoding`	visszaadja a dokumentum kódolását (azaz a character set beállítását)	3
`document.lastModified`	visszaadja a dokumentum utolsó módosításának dátumát és idejét	3
`document.links`	visszaadja az összes **<area>** és **<a>** elemet elemet, amelynek van **href** attribútuma	1

[2] *Az URI (Uniform Resource Identifier, egységes erőforrás-azonosító) egy rövid karaktersorozat, amelyet egy webes erőforrás azonosítására használunk. Közismert példái a webcímek, más néven URL-ek. - Az URI az erőforrást kétféleképp azonosíthatja: hely szerint (URL) vagy név szerint (URN). Az URL olyan URI, amely azzal határozza meg az erőforrást, miképp lehet azt elérni. Például a https://hu.wikipedia.org/ URL egy olyan URI, amely a magyar Wikipédia honlapját azonosítja, és magában foglalja azt is, hogy ennek az erőforrásnak valamilyen formája (a honlap aktuális HTML kódja) elérhető a hu.wikipedia.org nevű kiszolgálóról HTTPS protokollon keresztül. Az URN-re példa lehet a következő: urn:isbn:0-395-36341-1. Ez egy olyan URI, amely egy könyvet azonosít az ISBN-adata alapján. Ezzel az URN-nel azonosítottuk a könyvet anélkül, hogy bármit mondtunk volna a helyéről vagy az elérhetőségéről.*

tulajdonság	funkció	DOM szint
`document.readyState`	visszaadja a dokumentum (betöltődési) státuszát	3
`document.referrer`	visszaadja a dokumentum hivatkozó URI-ját	1
`document.scripts`	visszaadja az összes `<script>` elemet	3
`document.strictErrorChecking`	ez a tulajdonság beállítja vagy visszaadja, hogy a szigorú hibaellenőrzés kikényszeríthető-e a dokumentumon vagy sem (igaz vagy hamis logikai értékű, így ha igaz, akkor az azt jelenti, hogy szigorú ellenőrzést lehet alkalmazni)	3
`document.title`	visszaadja a `<title>` elemet	1
`document.URL`	visszaadja a dokumentum teljes URL-jét	1

Példa: Amikor pl. az oldal összes űrlapját el szeretnénk érni, akkor a `document.forms` kódot használjuk, amely egy tömbben adja vissza az oldal összes űrlapjának elérését. Ha ezek közül az elsőt akarjuk használni, akkor a tömb 0. indexű elemével dolgozunk: `document.forms[0]`. Ha az űrlapoknak azonosítót adtunk, akkor azokkal egyértelműbben elérhetővé tesszük az egyes formokat: például az `id="urlap1"` tulajdonsággal rendelkezőt a `document.forms["urlap1"]` hivatkozással a legegyszerűbb elérni, majd elemeit az `elements` tömb segítségével tudjuk lekérdezni. A ***3. mintapéldában*** az `urlap1` azonosítójú űrlap összes elemének értékét, azaz az egyes űrlapmezők `value` tulajdonságában tárolt értéket megjelenítő kód látható.

3. mintapélda

```
// megkeressük a dokumentumban lévő űrlapok közül az id="urlap1" tulajdonsággal  rendelkezőt
   const x = document.forms["urlap1"];
   let text = "";

// végiglépkedünk a megtalált űrlap összes elemén
   for (let i = 0; i < x.length; i++) {
      // és összegyűjtjük az egyes űrlapelemek értékeit
      text += x.elements[i].value + "<br>";
   }
// az id="demo" tulajdonságú helyre kiiratjuk az összegyűjtött értékeket
document.getElementById("demo").innerHTML = text;
```

HTML elemek elérése

A DOM programozása során elsődleges célunk az, hogy kiválasszuk azokat az elemeket, amelyekkel valamilyen módon dolgozni szeretnénk. Erre korábban már megismertünk egy metódus, a `document.getElementById` függvényt, amely azonosító alapján keresi meg és adja vissza az elemet. *A gyakorlat azonban azt mutatja, hogy sokszor nem azonosító, hanem egyéb információk alapján szükséges kiválasztani az elemeket, így erre számos további lehetőségünk van.*

4. mintapélda

```
<!doctype html>
<html>
    <head>
        <meta charset="utf-8">
        <title></title>
    </head>
    <body>
        <p id="par1">Bekezdés.</p>
        <p class="aktiv">Még egy bekezdés.</p>
        <ul>
            <li>első</li>
            <li class="aktiv">második</li>
        </ul>
        <form name="form1">
            <input type="radio" name="r1" value="elso">
            <input type="radio" name="r1" value="masodik">
        </form>
    </body>
</html>
```

A következőken azt mutatjuk be a ***4. mintapélda*** kódjának segítségével, hogy milyen módszereket használhatunk a weboldalon lévő egyes objektumok és azok tulajdonságainak elérésére, beállítására vagy módosítására.

metódus	funkció
`document.getElementById(`*id***`)`** `pl. document.getElementById("par1")`	a legegyszerűbb módszer egy elem beazonosítására, mert azt az ***azonosítója*** (***`id`***-ja) alapján keressük meg: • ha megtalálta (létezik ilyen elem), akkor azt objektumként adja eredményül • ha nem találta meg, akkor ***null*** értéket ad
`document.getElementsByName(`*name***`)`** `pl. document.getElementsByName("r1")`	elemek megkeresése a ***`name` attribútum értéke*** alapján – az elemeket tömbszerű objektumban (***`NodeList`***) adja vissza (*hasznos lehet rádiógombok keresésekor*)
`document.getElementsByTagName` **`(`***elementName***`)`** `pl. document.getElementByTagName("p")`	elemek megkeresése az ***azonos címkenév*** **(tagnév)** alapján – az elemeket tömbszerű objektumban (***`HTMLCollection`***) adja meg
`document.getElementsByClassName` **`(`***classname***`)`** `pl. document.getElementsByClassName("aktiv")`	elemek megkeresése az ***azonos osztálynév*** **(`class` érték)** alapján – az elemeket tömbszerű objektumban (***`HTMLCollection`***) adja meg

Ha például a megadott **`id`**-jú elemen belüli meghatározott HTML-tagokat szeretnénk megkeresni, akkor először **`id`** alapján, majd azon belül elemnév alapján kell meghatározni a keresés módját:

```
const elemIdAlapjan = document.getElementById(id);
const htmlTagIdBelsejebe = elemIdAlapjan.getElementsByTagName(elementName);
```

Gyakorlati példa: *összetartozó rádiógombokat vagy egy közös témára vonatkozó jelölőnégyzeteket akarunk lekérdezni (pl. legmagasabb végzettség – rádiógombokkal, nyelvismeret - jelölőnégyzetekkel), akkor célszerű az összetartozó elemeket egyetlen blokkban elhelyezni, amelyet azonosítóval látunk el, majd a benne található input elemeket (rádiógombokat / jelölőnégyzeteket), mivel azokat az <input> taggal hoztuk létre, már tagnév segítségével el tudjuk érni.*

metódus	funkció

`document.querySelector(`*css_selector***`)`**
`pl. document.querySelector("ul>li")`
`pl. document.querySelector("p.aktiv")`
	visszaadja ***az első olyan elemet, amely megfelel a megadott CSS-kijelölőnek***
	(a CSS-kijelölő utalhat elemre, azonosítóra, osztályra, összetett szelektorra)

`document.querySelectorAll(`*css_selector***`)`**
`pl. document.querySelector("ul>li")`
`pl. document.querySelector(".aktiv")`
	minden olyan elemet visszaad, amely megfelel a megadott szelektornak – az elemeket tömbszerű objektumban (**`NodeList`**) adja meg

Megjegyzés:
A NodeList és a HTMLCollection nagyon hasonlít egymáshoz. Mindkettő a dokumentumból kinyert egyes HTML elemekre utal és tömbszerű gyűjteményt (listát) ad eredményül, amelynek elemeit azok indexszámával érhetjük el (az index 0-tól kezdődik). Mindkettő rendelkezik egy hossz tulajdonsággal, amely a lista (gyűjtemény) elemeinek számát adja vissza. (Nem tömbök, ezért nem használható rájuk a tömbműveletek (kivéve, ha az **`Array.from(forras)`** *módon tömbbé nem alakítjuk. Feldolgozásuk csak* **`for`** *ciklussal ajánlott.)*

Alapvető különbségek:

- Amíg a **NodeList dokumentumcsomópontok** (elemcsomópontok, attribútumcsomópontok és szövegcsomópontok) gyűjteménye, addig a **HTMLCollection a dokumentumelemek** gyűjteménye.
- Emiatt a NodeList elemei csak az indexszámuk alapján érhetők el, a HTMLCollection elemek viszont a nevük, azonosítójuk vagy indexszámuk alapján érhetők el.
- a HTMLCollection-ra vonatkozó metódusok:
  - azonosítóval: **`namedItem(id-érték)`** – pl. lekérdezzük a bekezdéseket, azaz a **`<p>`** elemeket: **`let bek = document.getElementsByTagName("p")`** és azt követően ezek közül kiválasztjuk az egyiket az **`id`**-ja alapján: **`let bekAz = bek.namedItem("bevezeto")`** és ezután már annak bármely tulajdonságát (pl. **`bekAz.innerHTML`**) vagy metódusát is használhatjuk
  - indexszámmal: **`item(index)`** – pl. lekérdezzük a második szintű címsorokat, azaz a **`<h2>`** elemeket: **`let cimek2Szint = document.getElementsByTagName("h2")`** és azt követően ezek közül kiválasztjuk az egyiket az indexszáma alapján: **`let c0 = cimek2Szint.item(0)`**
  - indexszámmal történő megadás esetén használható a tömböknél már jól megszokott forma, azaz a [ ] jelek használata is: **`let c0B = document.getElementsByTagName("h2")[0];`**
- a NodeList-ra vonatkozó metódusok: (pl. lekérhető a **`document.body.childNodes`** segítségével a body összes gyermekcsomópontja)
  - **`entries()`**: egy iterátort ad vissza a listából a kulcs/érték párokkal
  - **`forEach(function(paraméterlista){kódsorok})`**: egy visszahívási függvényt hajt végre a NodeList minden csomópontjára
  - **`item(index)`**: visszaadja a csomópontok közül a megadott indexűt
  - **`keys()`**: egy iterátort ad vissza a listában szereplő kulcsokkal
  - **`values()`**: egy iterátort ad vissza a lista értékeivel
- A HTMLCollection mindig élő gyűjtemény, a NodeList viszont a leggyakrabban egy statikus gyűjtemény. (Pl. ha hozzáad egy <li> elemet egy listához a DOM-ban, akkor a HTMLCollection listája is megváltozik, a NodeList listája viszont nem változik.)

HTML elemek értékének, tulajdonságainak vagy stílusjegyeinek megváltoztatása

Ha kiválasztottunk vagy elértünk egy elemet, akkor a metódusain és tulajdonságain keresztül tudunk vele dolgozni. Ezeket a szabvány határozza meg, és függ attól, hogy milyen típusú elemről van szó. **Egy elem a tulajdonságait egyrészt mint csomópont, másrészt mint HTML elem kapja meg.** Az előbbiről a DOM Core, az utóbbiról a HTML DOM szabvány gondoskodik. De **a kiválasztott elemnek lehetnek HTML szerinti attribútumai is** (pl. az **<img>** tagnak az **src**, az **alt** vagy a **title** az attribútuma). Ezeknek a kezelésére a szabvány többféle lehetőséget is biztosít.

metódus (M) vagy tulajdonság (T)	funkció

(T) *element*.**innerHTML**
(T) *element*.**innerHTML** = *új-tartalom*
`pl. let k = document.getElementById("kezdet");`
`k.innerHTML = "<b>Új HTML tartalom!</b>";`
`pl. let t = document.getElementById("f1").innerHTML;`
	beállítja vagy visszaadja a megadott elem ***HTML tartalmát a belső címkékkel együtt***

(T) *element*.**innerText** (T) *node*.***innerText***	***visszaadja vagy beállítja*** az elem vagy a csomópont ***szövegtartalmát***: ekkor az összes gyermekcsomópont eltávolításra kerül, és csak egy új szövegcsomópont lép helyette
(T) *element*.**textContent**	***visszaadja*** az elem ***szöveges tartalmát és minden leszármazottját, szóközzel és rejtett CSS-szöveggel, de címkék nélkül***

(M) *element*.**hasAttribute**(*attribute*)
`pl. let k1=document.getElementById("kep1");`
`let vanTitle=k1.hasAttribute("title");`
	egy elem megadott nevű ***attribútuma létezésének vizsgálata***

(T) *element*.*attribute*
`pl. document.getElementById("kep1").src`
(M) *element*.**getAttribute**(*attribute*)
`pl. let altErtekVanE=k1.getAttribute("alt");`
	elem megadott ***attribútumának lekérdezése***

(T) *element*.*attribute* = *új-érték*
`pl. k1.src = "../ujk/maci.gif";`
(M) *element*.**setAttribute**(*attribute, value*)
`pl. k2.setAttribute("class","kiemelt");`
	módosítja az elem megadott ***attribútumának értékét***

(M) *element*.**removeAttribute**(*attribute*) `pl. k2.removeAttribute("alt");`	***eltávolítja*** az elem megadott nevű ***attribútumát***
(T) **attributes** `pl. document.getElementById("kep1").attributes`	visszaadja a meghatározott elem attribútumgyűjteményét
(T) *element*.**style**.*property* = *új-stílusérték* `pl. k1.style.width = "500px";`	***módosítja*** a megadott elem megadott ***stílusértékét***

Eseménykezelők hozzáadása

metódus	funkció
`document.getElementById(`*id*`).onclick = function() { `*utasítások*` }`	***eseménykezelőt hoz létre*** a megadott *id*-jú elemre történő kattintáshoz

3. Elemek létrehozása és törlése

HTML elemek hozzáadása

A dokumentum módosításának következő lépése az új elemek létrehozása. Erre korábban már láttunk megoldást az **`innerHTML`** tulajdonság személyében, amelynél a szövegesen megadott HTML részletből a böngésző létrehozza a megfelelő DOM elemeket és azokat elhelyezi a dokumentumfa megfelelő részére. (Az **`innerHTML`** a HTML5-tel kerül bele a szabványba.) **A formálisabb utat elemek létrehozására a `document.createElement(`*element*`)` biztosítja. Ez egy dokumentumszintű metódus, paraméterként a létrehozandó elem nevét kell megadni.** A metódus eredményeképpen a memóriában létrejön a megfelelő DOM elem. Ezt az elemet felruházhatjuk a szükséges attribútumokkal, beállíthatjuk bizonyos tulajdonságait, akár gyerekelemeket is csatolhatunk hozzá. Ha a memóriabeli elemet felkészítettük, akkor már csak a dokumentum megfelelő pontjára kell beszúrni azt.

metódus	funkció
`document.createElement(`*element*`)`** `pl. let ujelem = document.createElement("p")`	***létrehozza*** a HTML elemet
`document.createTextNode(`*érték*`)`** `pl. let ujertek = document.createTextNode ("Ez lesz az új elem tartalma!")`	***létrehozza*** a szövegcsomóponti tartalmat[3]

`appendChild(`*element*`)`**
`pl. let bek=document.createElement("p");`
`document.body.appendChild(bek);`
	a megadott ***szülőelem gyerekei közé utolsóként szúrja be*** az *újelem*-et

`insertBefore(`*újelem*`,`*refelem*`)`** `pl. szuloelem.insertBefore(ujelem,helyelem)`	***beszúrja az újelem csomópontot*** a megadott ***szülőelem gyerekei*** közül **a létező *helyelem* gyerekcsomópont elé** (ha ez utóbbi null, akkor a szülő gyermekelemei végére szúrja be)

[3] *Az elem tartalmának megadásához használható az innerHTML tulajdonság is. Pl. ha let blokk=document.createElement("div")-vel létrehoztuk a blokkot, akkor a blokk.innerHTML="Ez az új tartalom…" utasítással elhelyezhetjük benne a tartalmat is.*

A ***5. mintapéldában*** egy **új elem kialakításának és utolsó gyermekelemként való beszúrásának** lépéseire láthat példát.

5. mintapélda

```
// szükség esetén beolvassuk az új tartalmat (input elem) és annak kinyerjük a value értékét:
   let ujertek = document.getElementById("ujtartalom").value;
// létrehozzuk az új HTML-tagot:
   const ujelem = document.createElement("li");
// ha szöveget szeretnénk adni ehhez, akkor létrehozzuk a szöveges csomópontot:
   const ujtartalom = document.createTextNode(ujertek);
// létrehozzuk az új elemet a tagnévvel és a tartalmával – a taghoz (li) hozzáfűzzük a szöveges
   csomópontot
   ujelem.appendChild(ujtartalom);
// megadhatunk számára új attribútumokat is
   ujelem.setAttribute("class","friss");
//  beazonosítjuk a beszúrás helyét és hozzáfűzzük a létrehozott új elemet
   (li tag a szövegtartal-mával) annak utolsó gyermekelemeként
   document.getElementById("hely").appendChild(ujelem);
```

A ***6. mintapéldában*** egy **új elem kialakításának és egy megadott elem elé való beszúrásának** lépéseire láthat példát.

6. mintapélda

```
// szükség esetén beolvassuk az új tartalmat (input elem) és annak kinyerjük a value értékét:
   let ujertek2 = document.getElementById("ujtartalom2").value;
// létrehozzuk az új HTML-tagot:
   const ujelem2 = document.createElement("p");
// létrehozzuk az új HTML-tag tartalmát:
   const ujtartalom2 = document.createTextNode(ujertek2);
// létrehozzuk az új elemet a tagnévvel és a tartalmával
   ujelem2.appendChild(ujtartalom2);
// megkeressük a beszúrási hely szülőelemét
   let szuloelem = document.getElementById("szulo2");
// megkeressük a szülőelem azon gyermekelemét, amelyik elé szeretnénk majd beszúrni
   az új elemet
   let gyerekelem = document.getElementById("gyerek2");
// a szülőelembe a megadott gyermekelem elé beszúrjuk az új elemet
   szulo2.insertBefore(ujelem2,gyermekelem);
```