**Rukovanje DOM-om**

**00:28:15**

Jedinica: 27 od 30

**Rezime**

* DOM je skraćenica koja označava pojam Document Object Model, što je objektna reprezentacija jednog HTML dokumenta.
* DOM gradi objektnu strukturu u formi stabla.
* Kako bi se rukovalo elementima (čvorovima) DOM strukture, neophodno je prethodno dobaviti referencu na objekte koji ih predstavljaju.
* Funkcija getElementById obezbeđuje referencu na element sa prosleđenom vrednošću id atributa.
* Funkcija getElementsByClassName obezbeđuje referencu na sve elemente koji imaju prosleđenu vrednost class atributa.
* Funkcija getElementsByTagName obezbeđuje referencu na sve elemente na stranici koji su određenog tipa.
* Funkcija querySelector obezbeđuje referencu na prvi element koji zadovoljava prosleđeni CSS selektor.
* Funkcija querySelectorAll obezbeđuje referencu na sve elemente koji zadovoljavaju kriterijum prosleđenog CSS selektora.
* Svojstvo koje se koristi za rukovanje tekstualnim sadržajem HTML elementa je innerHTML.
* Za postavljanje vrednosti atributa nekog elementa, može se koristiti metoda setAttribute.
* Stilizacijom jednog elementa se može rukovati korišćenjem style svojstva, koje svaki DOM objekat poseduje.
* Funkcija createElement kreira HTML element.
* Funkcija removeChild uklanja HTML element.
* Funkcija appendChild dodaje HTML čvor.
* Funkcija replaceChild menja postojeći HTML element.
* Funkcija insertBefore omogućava dodavanje elementa pre nekog drugog elementa u DOM strukturi

U dosadašnjim lekcijama, u kojima je bilo reči o osnovnim načelima jezika JavaScript, mogla je da se stekne slika o mogućnostima jezika. Ipak, predstavljene mogućnosti jezika nisu do sada iskorišćene za jednu od najkorisnijih uloga koju JavaScript ima. Reč je o manipulisanju strukturom, stilizacijom i podacima.

JavaScript je u svom nastajanju i zamišljen kao jednostavan jezik, čija je primarna uloga bila manipulacija strukturom i stilizacijom. Tako je bilo omogućeno da se na jednostavan način validiraju podaci koje je korisnik uneo na stranici, i da se rezultat validacije prikaže korisniku, bez potrebe za upošljavanjem servera, korišćenjem u to vreme veoma sporih internet konekcija. Kasnije JavaScript dobija mogućnost da manipuliše i podacima, čime se stvara široko poznati akronim AJAX.  
  
Korišćenjem JavaScript jezika, moguće je obaviti sledeće:

* kreiranje, dodavanje, izmene i uklanjanje HTML elemenata
* kreiranje, dodavanje, izmene i uklanjanje HTML atributa elemenata
* kreiranje, dodavanje, izmene i uklanjanje CSS opisa
* kreiranje i reagovanje na događaje nastale na HTML elementima
* rukovanje podacima, odnosno sadržajem HTML elemenat
* asinhrono slanje zahteva serveru i prijem odgovora

Da bi pobrojano bilo ostvarivo, JavaScript se oslanja na jednu specijalnu objektnu strukturu dokumenta. Reč je o strukturi koja sa naziva DOM.

**Šta je DOM?**

DOM je skraćenica koja označava pojam *Document Object Model*. DOM je objektna reprezentacija jednog HTML dokumenta.

HTML se sastoji iz tagova, odnosno elemenata koji grade strukturu dokumenta. Takvi elementi za JavaScript nemaju posebno značenje. Ali, kada se takvi elementi prevedu u objekte, svi znamo da je JavaScript na svom terenu. Tako je DOM potpuno objektno orijentisana reprezentacija strukture dokumenta, koja omogućava skriptnim jezicima da na lak način vrše modifikaciju dokumenta.

DOM je standard [W3C](https://www.link-elearning.com/linkdl/opisPojma.php?id=144319) organizacije i nije ograničen na korišćenje sa HTML dokumentima. DOM standard se zapravo sastoji iz tri različite celine:

* Core DOM – standardni objektni model za sve tipove dokumenata
* XML DOM – objektni model za XML dokumente
* HTML DOM – objektni model za HTML dokumente

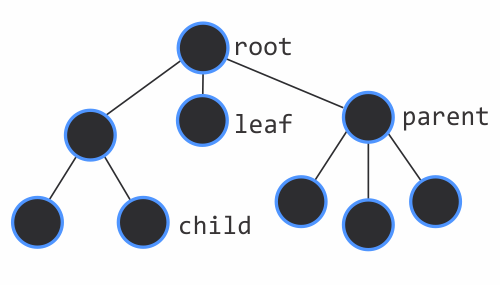
U ovom kursu ćemo se baviti isključivo HTML DOM standardom. Pored ovog standarda, i poznavanje XML DOM standarda prilikom razvoja HTML5 aplikacija može biti veliki plus, pogotovu ukoliko se u aplikaciji koristi [SVG](https://www.link-elearning.com/linkdl/opisPojma.php?id=144273).

**Kako izgleda DOM struktura?**

DOM gradi objektnu strukturu u formi stabla. Stablo je specijalna vrsta strukture podataka, kod koje svaki element, osim korenog, ima svog roditelja. Takođe, svaki element može imati i proizvoljan broj potomaka. Tako su osobine stabla sledeće:

* stablo poseduje koreni element
* svaki element osim korenog ima svog roditelja
* jedan element može imati proizvoljan broj potomaka
* element ne mora da ima potomke
* roditelji i potomci međusobno grade jednu granu
* poslednji element u jednoj grani se naziva list

Osobine DOM strukture, odnosno jednog stabla, mogu se ilustrovati slikom 27.1.

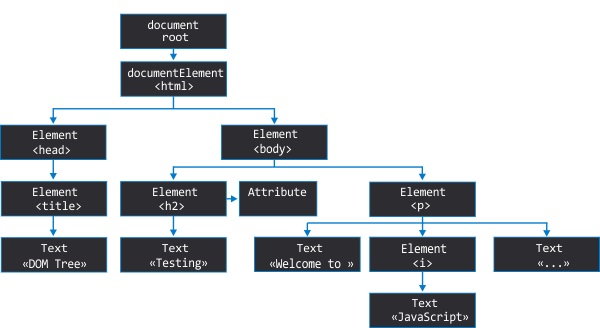


*Slika 27.1 – Struktura stabla*

S obzirom da je i sam HTML dokument slične strukture, veoma je lako njega transformisati u prikaz sličan ovom na slici 27.1. U nastavku će biti prikazano kako jedan HTML dokument izgleda transformisan u DOM. Kao ogledni dokument će poslužiti sledeći HTML:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <!DOCTYPE html>  <html>  <head>   <title>DOM Tree</title>  </head>  <body>   <h2>Testing</h2>   <p>Welcome to <i>JavaScript</i>...</p>  </body>  </html> |

Kada se prikazani HTML transformiše u DOM, izgleda kao na slici 27.2.

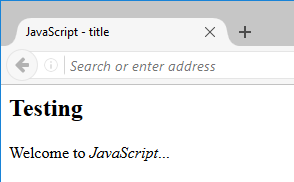


*Slika 27.2 – DOM struktura jednog HTML dokumenta*

Slika 27.2 ilustruje način na koji se HTML struktura konvertuje u objektnu DOM strukturu. Vrlo je važno primetiti da je koreni element DOM strukture objekat document. To je objekat koji će sadržati sve ostale objekte u ovoj strukturi. Tako se može reći da je npr. element title svojstvo objekta document. Ukoliko je to tačno, onda se na veoma lak način može izmeniti naslov jednog dokumenta:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <html>  <head>  <title>DOM Tree</title>  </head>  <body>  <h2>Testing</h2>  <p>Welcome to <i>JavaScript</i>...</p>    <script>              document.title = "JavaScript - title";  </script>    </body>  </html> |

U HTML strukturu dodat je script element sa jednom linijom koda, koja za cilj ima postavljanje vrednosti title svojstva document objekta. Kada se ovakav dokument otvori u nekom browseru, dobija se efekat kao na slici 27.3.



*Slika 27.3 – DOM struktura jednog HTML dokumenta*

Na slici 27.3 se može videti da je naslov dokumenta *JavaScript – title*, iako je u samoj strukturi u fajlu naslov postavljen na *DOM Tree*. Dokument uistinu i ima naslov *DOM Tree*, sve do onoga trenutka kada se linija JavaScript koda za promenu vrednosti svojstva title ne izvrši. Na sličan način, moguće je manipulisati i ostalim elementima DOM strukture.

**Elementi DOM strukture koji se nalaze na dnu hijerarhije i ne poseduju potomke nazivaju se:**

 grane

 roditelji

 listovi

 stabla

**Selektovanje elemenata u DOM strukturi**

Kako bi se rukovalo elementima (čvorovima) DOM strukture, neophodno je prethodno dobaviti referencu na objekte koji ih predstavljaju. Drugim rečima, potrebno je takve elemente selektovati na sličan način na koji se to postiže u CSS jeziku.

Postoji nekoliko načina na koje je moguće doći do željenog objekta u DOM strukturi. U svakom slučaju, za pristup elementima se koristi globalni objekat koji se kreira od strane browsera. Nad ovim globalnim objektom, mogu se pozvati različite metode koje omogućavaju dobavljanje reference na jedan ili više objekata. Te metode su prikazane tabelom 27.1.

|  |  |
| --- | --- |
| **Metoda** | **Opis** |
| getElementById(id) | obezbeđuje referencu na element sa prosleđenom vrednošću idatributa |
| getElementsByClassName(name) | obezbeđuje referencu na sve elemente koji imaju prosleđenu vrednost class atributa |
| getElementsByTagName(name) | obezbeđuje referencu na sve elemente na stranici koji su određenog tipa |
| querySelector(cssSelector) | obezbeđuje referencu na prvi element koji zadovoljava prosleđeni CSS selektor |
| querySelectorAll(cssSelector) | obezbeđuje referencu na sve elemente koji zadovoljavaju kriterijum prosleđenog CSS selektora |

*Tabela 27.1 – Funkcije za selektovanje elemenata*

**Pretraga elemenata po ID-ju – getElementById()**

Za pronalazak elementa po vrednosti atributa id, koristi se metoda getElementById, koja se poziva nad document objektom. Na taj način se govori browseru da je potrebno da pronađe element koji se nalazi u dokumentu i koji ima id koji je naveden kao parametar ove metode.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | <html>  <head>   <title>DOM</title>  </head>  <body>   <div id="div\_1">    <p id="p\_1">Text 1</p>    <p id="p\_2">Text 2</p>    <p>Text 3</p>   </div>   <script>    var paragraph\_1 = document.getElementById('p\_2');   </script>    </body>  </html> |

U primeru je kreirano nekoliko HTML elemenata koji poseduju definisane vrednosti id atributa:

* div\_1 – odnosi se  na prvi div element
* p\_1 – odnosi se na prvi p element
* p\_2 – odnosi se na drugi p element

Nakon izvršavanja definisanog JavaScript koda, unutar promenljive paragraf\_1, biće smeštena referenca na element sa id-jem p\_2, odnosno na drugi paragraf element.

|  |
| --- |
| **Napomena**    *Treći paragraf u primeru ne poseduje definisanu vrednost id atributa. Stoga, ne bi bilo moguće njega selektovati na ovaj način.* |

**Pretraga elemenata po nazivu klase – getElementsByClassName()**

DOM omogućava da se izvrši selektovanje više elemenata koji imaju zajedničku vrednost class atributa:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | <html>  <head>   <title>DOM</title>  </head>  <body>   <p id="myClass">Text 1</p>   <p class="myClass">Text 2</p>   <p class="myClass someOtherClass">Text 3</p>   <p>Text 4</p>   <p>Text 5</p>   <p class="yourClass">Text 6</p>     <script>    var paragraphs = document.getElementsByClassName('myClass');    console.log(paragraphs.length);   </script>      </body>  </html> |

Struktura body dela se sastoji iz 6 paragrafa. Neki paragraf elementi imaju definisanu vrednost idatributa, drugi class atributa, dok su ostali paragrafi bez ikakvih atributa.

JavaScript kodom se selektuju svi elementi koji za vrednost class atributa imaju myClass.

S obzirom da više elemenata može imati jednu istu klasu, metoda getElementsByClassNamevraća niz elemenata, pa čak i kada se selektuje samo jedan element.

Nakon selektovanja, unutar konzole u primeru se ispisuje broj selektovanih elemenata:

|  |  |
| --- | --- |
| 1 | 2 |

Ovo znači da su ukupno selektovana dva elementa sa klasom myClass. Ti elementi su drugi i treći paragraf.

Jedan element može imati više klasa, odvojenih praznim mestom, baš kao što je u primeru to slučaj sa trećim paragrafom. Metoda getElementsByClassName selektovaće element ukoliko pronađe podudaranje makar jednog klasnog imena sa nazivom klase koji se pretražuje.

**Selektovanje elemenata na osnovu naziva taga – getElementsByTagName()**

Elemente je moguće selektovati i na osnovu njihovog tipa, odnosno naziva taga koji se koristi za izgradnju elementa:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | <html>  <head>   <title>DOM</title>  </head>  <body>   <ul>    <li>Item 1</li>    <li>Item 2</li>    <li>Item 3</li>   </ul>     <script>    var items = document.getElementsByTagName('li');    console.log(items.length);   </script>    </body>  </html> |

U primeru je kreirana jedna neuređena lista sa tri stavke. Da bi se selektovale sve tri stavke, odnosno tri elementa koja reprezentuju stavke, koristi se metoda getElementsByTagName, kojoj se prosleđuje naziv tipa elementa.

Metoda getElementsByTagName vraća niz elemenata. Stoga JavaScript kod u konzolu upisuje broj 3, s obzirom da je to dužina niza koji se dobija metodom getElementsByTagName.

**Selektovanje elementa korišćenjem CSS selektora – querySelector()**

Korišćenjem metode querySelector(cssSelector), moguće je selektovati prvi element koji zadovoljava kriterijum prosleđenog CSS selektora:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | <html>  <head>   <title>DOM</title>  </head>  <body>   <ul>    <li>Item 1</li>    <li id="myId\_1">Item 2</li>    <li>Item 3</li>   </ul>     <script>    var item = document.querySelector('#myId\_1');      var oneMoreItem = document.querySelector("li");    console.log(oneMoreItem.innerHTML);   </script>    </body>  </html> |

Primer prikazuje neuređenu listu sa tri stavke. Korišćenjem metode querySelector, navodi se CSS selektor koji selektuje element sa myId id vrednošću. Na taj način promenljiva item dobija referencu na drugi li element.

Kako bi dokazali da metoda querySelector uvek vraća samo prvi element koji zadovoljava prosleđeni CSS kriterijum, definisan je drugi primer, kojim se selektuju elementi korišćenjem CSS selektora li. Na ovaj način se selektuju elementi po tipu. U dokumentu postoje tri elementa takvog tipa, ali će metoda querySelector vratiti samo prvi element. Tako će promenljiva oneMoreItemimati referencu na prvi li element. To dokazuje i ispis koji proizvodi prikazani kod:

|  |  |
| --- | --- |
| 1 | Item 1 |

|  |
| --- |
| **Napomena**  *Metoda innerHTML se koristi za manipulaciju tekstualnim sadržajem elementa, i biće detaljno opisana u nastavku.* |

**Selektovanje elemenata korišćenjem CSS selektora – querySelectorAll()**

Veoma slično prethodnoj metodi funkcioniše i metoda querySelectorAll(), ali sa tom razlikom što će u ovom slučaju biti selektovani svi elementi koji zadovoljavaju uslov naveden u zagradi.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | <html>  <head>   <title>DOM</title>  </head>  <body>   <ul>    <li>Item 1</li>    <li id="myId">Item 2</li>    <li>Item 3</li>   </ul>     <script>    var oneMoreItem = document.querySelectorAll("li");    console.log(oneMoreItem.length);   </script>    </body>  </html> |

S obzirom da vraća više elemenata, povratna vrednost metode querySelectorAll je niz vrednosti. Zbog toga prikazani kod proizvodi rezultat:

|  |  |
| --- | --- |
| 1 | 3 |

**Izmene na HTML elementima**

Nakon što se obavi selektovanje jednog ili više elemenata, može se preći na manipulaciju njihovim sadržajem i vrednostima atributa. Na taj način se sprovode izmene na HTML elementima. U nastavku lekcije će biti prikazane tehnike za:

* izmenu sadržaja elemenata
* dodavanje, brisanje i izmenu vrednosti atributa
* kreiranje nove i izmenu postojeće stilizacije

**innerHTML**

Svojstvo koje se koristi za rukovanje tekstualnim sadržajem HTML elementa je innerHTML. Omogućava da se tekstualni sadržaj pročita, ili da se upiše novi umesto postojećeg:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | <html>  <head>   <title>DOM</title>  </head>  <body>   <p id="the-paragraph">Text 1</p>     <script>    var paragraph = document.getElementById('the-paragraph');    //reading innerHTML    console.log("Original paragraph content was: " + paragraph.innerHTML);    //writing innerHTML    paragraph.innerHTML = "New text set using innerHTML property";   </script>      </body>  </html> |

Primer ilustruje korišćenje innerHTML svojstva za čitanje i postavljanje teksta HTML elementa. U primeru, rukuje se paragraf elementom sa id-jem *the-paragraph*.

Prvo se čita originalni tekst ovog paragrafa, i informacija o tome se upisuje u konzolu:

|  |  |
| --- | --- |
| 1 | Original paragraph content was: Text 1 |

Nakon čitanja, vrednost innerHTML svojstva se menja, dodeljivanjem nove vrednosti, tako da paragraf na stranici izgleda ovako:

|  |  |
| --- | --- |
| 1 | New text set using innerHTML property. |

**Čitanje vrednosti atributa**

Čitanje vrednost atributa nekog elementa može se obaviti na veoma jednostavan način, navođenjem naziva atributa kao svojstva objekta:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | <html>  <head>   <title>DOM</title>  </head>  <body>   <p id="the-paragraph" class="my-class">Text 1</p>     <script>    var paragraph = document.getElementById('the-paragraph');    console.log("Id: " + paragraph.id);    console.log("Class: " + paragraph.className);   </script>    </body>  </html> |

Primer će emitovati rezultat:

|  |  |
| --- | --- |
| 1  2 | Id: the-paragraph  Class: my-class |

**Postavljanje vrednost atributa**

Za postavljanje vrednosti atributa nekog elementa, može se koristiti metoda setAttribute. Ova metoda ne samo da ima mogućnost da promeni vrednost atributa već ga i kreira u situacijama kada takav atribut ne postoji. Njena sintaksa je sledeća:

|  |  |
| --- | --- |
| 1 | element.setAttribute(attributename,attributevalue) |

Prvi parametar se odnosi na naziv atributa čija se vrednost postavlja, dok je drugi parametar vrednost atributa.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <html>  <head>   <title>DOM</title>  </head>  <body>   <input type="text" name="firstname" id="first-name" value="I want to be button!">     <script>    var input = document.getElementById('first-name');    input.setAttribute('type', 'button');   </script>    </body>  </html> |

U primeru je definisan jedan input element tipa text. Ipak, JavaScript kodom se menja vrednost njegovog atributa type sa *text na button*. Pre izvršavanja linije koda u kojoj se to postiže, element na stranici izgleda kao na levoj polovini slike 27.4. Nakon izvršavanja linije koda u kojoj se poziva metoda setAttribute, kontrola, zbog nove vrednosti type atributa, dobija novu vizualnu reprezentaciju na stranici, pa tako izgleda kao na desnoj polovini slike 27.4.

https://www.link-elearning.com/linkdl/coursefiles/1166/PH5JC_27_04.jpg

*Slika 27.4 – Promena type vrednosti input elementa korišćenjem JavaScripta*

**Rukovanje stilizacijom**

Stilizacijom jednog elementa se može rukovati korišćenjem style svojstva koje svaki DOM objekat poseduje. Sledeći primer ilustruje način na koji je moguće promeniti boju teksta jednog paragrafa:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <html>  <head>   <title>DOM</title>  </head>  <body>   <p id="the-paragraph" class="my-class">Text 1</p>     <script>    var paragraph = document.getElementById('the-paragraph');    paragraph.style.color = "blue";   </script>    </body>  </html> |

U primeru se koristi svojstvo style, nad kojim se koristi svojstvo color i njemu postavlja vrednost na blue. Na taj način se tekst paragrafa boji u plavo tako što JavaScript postavlja inline stilizaciju na element, kao na slici 27.5.

https://www.link-elearning.com/linkdl/coursefiles/1166/PH5JC_27_05.jpg

*Slika 27.5 – JavaScript postavlja inline stilizaciju na element*

Upravo prikazan način za rukovanje stilizacijom podrazumeva direktnu intervenciju nad CSS opisima i njihovo upisivanje u style atribut elementa (*inline stilizacija*). Pored ovog pristupa, za promenu stilizacije jednog elementa često se pribegava rešenju koje podrazumeva dodavanje i uklanjanje klasa sa HTML elementa. Jednostavno, definiše se nekoliko CSS opisa koji selektuju elemente najčešće na osnovu klase. Zatim se na elementima jednostavno vrši postavljanje i uklanjanje vrednosti atributa class.

Malopređašnji primer, ovoga puta dodavanjem i uklanjanjem klasa:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | <html<  <head>   <title>DOM</title>  </head>  <style type="text/css">   .blue-text{    color: blue;   }  </style>  <body>   <p id="the-paragraph" class="my-class">Text 1</p>     <script>    var paragraph = document.getElementById('the-paragraph');    paragraph.classList.add("blue-text");   </script>    </body>  </html> |

Ovoga puta je unutar style sekcije kreiran CSS opis koji selektuje sve HTML elemente sa klasom blue-text. Kada se ova klasa doda paragraf elementu, boja njegovog teksta postaje plava.

Da bi se jedan naziv klase uklonio sa nekog elementa, moguće je koristiti sledeći pristup:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | <html>  <head>   <title>DOM</title>  </head>  <style type="text/css">   .blue-text{    color: blue;   }  </style>  <body>   <p id="the-paragraph" class="my-class blue-text">Text 1</p>     <script>    var paragraph = document.getElementById('the-paragraph');    paragraph.classList.remove("blue-text");   </script>    </body>  </html> |

Ovoga puta, paragraf element inicijalno ima kao jednu od klasa i klasu blue-text. Naravno, zbog toga je boja teksta paragrafa plava. Da bi se boja teksta vratila na podrazumevanu, moguće je ukloniti klasu blue-text, i to se u primeru postiže pozivanjem metode remove nad svojstvom classList.

**Dodavanje i brisanje elemenata**

U dosadašnjim primerima prikazano je samo kako se rukuje elementima koji već postoje unutar DOM strukture. Vrlo moćna funkcionalnost DOM API-ja jeste mogućnost da se elementi dinamički dodaju i uklanjanju. Ovo otvara mogućnost za čitav splet različitih interaktivnosti koje je moguće izgraditi dodavanjem i uklanjanjem elemenata. Dodavanje i uklanjanje elemenata je osnova i AJAX pristupa, jer je podatke koji asinhrono pristignu sa servera neophodno ubaciti u postojaću stranicu, a neretko je potrebno određene elemente prethodno i ukloniti.

Za kreiranje i uklanjanje elemenata,  koriste se funkcije prikazane tabelom 27.2.

|  |  |
| --- | --- |
| **Metoda** | **Opis** |
| createElement(element) | kreira HTML element |
| removeChild(element) | uklanja HTML element |
| appendChild(element) | dodaje HTML čvor |
| replaceChild(newChild,oldChild) | menja postojeći HTML element |
| insertBefore(newNode, referenceNode); | omogućava dodavanje elementa pre nekog drugog elementa u DOM strukturi |

*Tabela 27.2 – Funkcije za kreiranje i uklanjanje elemenata*

Da bi se neki HTML element dodao u postojeći dokument, njega je prvo potrebno kreirati korišćenjem metode createElement.

|  |  |
| --- | --- |
| 1 | var newParagraph = document.createElement("p"); |

Na ovaj način, kreira se jedan novi paragraf element. Ipak, ovakav paragraf još nije dodat stranici, a i ne poseduje nikakav sadržaj. Kreirajmo njegov sadržaj:

|  |  |
| --- | --- |
| 1 | var newContent = document.createTextNode("This is dynamically generated paragraph!"); |

U DOM strukturi, uglavnom se zasebnim čvorovima predstavlja element i njegov tekstualni sadržaj. Upravo zbog toga se čvor tekstualnog sadržaja zasebno kreira. Ovako kreiran tekstualni sadržaj još nije povezan ni sa jednim konkretnim elementom. To se postiže upotrebom metode appendChild:

|  |  |
| --- | --- |
| 1 | newParagraph.appendChild(newContent); |

Na kraju, preostaje da se paragraf sa sadržajem doda stranici:

|  |  |
| --- | --- |
| 1 | document.body.appendChild(newParagraph); |

Kod kompletnog primera je sledeći:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | <html>  <head>  <title>DOM</title>  </head>  <body>    <p id="the-paragraph" class="my-class blue-text">Non-dynamic paragraph.</p>    <script>  var newParagraph = document.createElement("p");  var newContent = document.createTextNode("This is dynamically generated paragraph!");  newParagraph.appendChild(newContent);  document.body.appendChild(newParagraph);    </script>    </body>  </html> |

Primer proizvodi rezultat:

|  |  |
| --- | --- |
| 1  2 | Non-dynamic paragraph.  This is dynamicly generated paragraph! |

Analizom primera se može videti da se korišćenjem metode appendChild element dodaje na začelje roditeljskog elementa. Ukoliko je potrebno preciznije odrediti lokaciju ubacivanja elementa, može se koristiti metoda insertBefore. Ova metoda prihvata dva parametra, odnosno čvor koji se dodaje i referentni čvor pre koga će element biti ubačen.

Identičan primer prethodnom, ali ovoga puta sa ubacivanjem elementa na početak roditeljskog elementa:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | <html>  <head>   <title>DOM</title>  </head>  <body>     <p id="the-paragraph" class="my-class">Non-dynamic paragraph.</p>     <script>    var newParagraph = document.createElement("p");    var newContent = document.createTextNode("This is dynamically generated paragraph!");    newParagraph.appendChild(newContent);      var oldParagraph = document.getElementById("the-paragraph");    document.body.insertBefore(newParagraph, oldParagraph);     </script>    </body>  </html> |

Ovoga puta primer proizvodi rezultat:

|  |  |
| --- | --- |
| 1  2 | This is dynamically generated paragraph!  Non-dynamic paragraph. |

**Uklanjanje elemenata**

Da bi se neki element uklonio iz DOM strukture, moguće je koristiti metodu removeChild:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | <html>  <head>   <title>DOM</title>  </head>  <body>     <ul id='myList'>    <li>Item 1</li>    <li>Item 2</li>    <li>Item 3</li>   </ul>       <script>    var list = document.getElementById("myList");    list.removeChild(list.children[1]);     </script>    </body>  </html> |

Primer proizvodi rezultat:

* Item 1
* Item 3

U primeru je kreirana jedna HTML lista sa tri stavke. U JavaScript kodu se dolazi do reference na listu u DOM strukturi, a zatim se nad takvim objektom poziva metoda removeChild. Ovoj metodi se prosleđuje objekat koji predstavlja element koji je potrebno ukloniti iz strukture. U primeru se uklanja središnja stavka, a referenca na nju se dobija korišćenjem svojstva children.

|  |
| --- |
| **Napomena**    *Svojstvo children vraća kolekciju elemenata koji su naslednici nekog elementa. Pritom se pod pojmom elemenata misli na isključivo HTML objekte, a ne čvorove. Tako ovo svojstvo ima različit efekat od svojstva childNodes, koje vraća kolekciju čvorova jednog elementa. Već je rečeno da se elementi i njihov tekstualni sadržaj u DOM strukturi uglavnom predstavljaju različitim čvorovima.* |

**Zamena elemenata**

Pored metoda za kreiranje, dodavanje i uklanjanje, DOM API poseduje i metode za zamenu čvora nekim drugim. Reč je o metodi replaceChild:

|  |  |
| --- | --- |
| 1 | replaceChild(newChild, oldChild); |

Metoda replaceChildprihvata dva parametra. Prvi se odnosi na element koji se dodaje, a drugi na element umesto koga se novi element dodaje:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | <html>  <head>   <title>DOM</title>  </head>  <body>     <p id="the-paragraph" class="my-class">Non-dynamic paragraph.</p>     <script>    var newParagraph = document.createElement("p");    var newContent = document.createTextNode("This is dynamically generated paragraph!");    newParagraph.appendChild(newContent);      var oldParagraph = document.getElementById("the-paragraph");    document.body.replaceChild(newParagraph, oldParagraph);     </script>    </body>  </html> |