

Javascript – deel 08

Deze les behandelt DOM-tree nodes en custom attributen.

Verslag

In dit deel van de cursus staan verschillende vragen die je moet beantwoorden en opdrachten om iets te maken of uit te proberen. Het is belangrijk dat je alle opdrachten zorgvuldig uitvoert!

Documenteer je werk in een verslag document 'javascript deel 08' waarin je

- voor elke uitprobeer opdracht een entry maakt met screenshots ter staving van wat je deed
- je antwoorden op de gestelde vragen neerschrijft

Oplossingen van grotere opdrachten (met veel code) bewaar je aparte folders in een Webstorm project.

DOM-tree node vs. DOM-tree element

Als we het over de bestanddelen van de DOM-tree hebben, moeten we eigenlijk over **DOM-tree nodes** spreken en niet over DOM-tree elementen.

De meeste nodes stellen weliswaar HTML-elementen voor (bv. `` en `` elementen), maar de DOM-tree bevat ook nodes voor andere zaken zoals bv. de tekstuele inhoud van een element of de commentaar stukken in het HTML document.

Voorheen hebben we het onderscheid niet gemaakt omdat we steeds met nodes werkten die elementen voorstelden. De namen van de methods die we hiervoor gebruikten gaven dit ook al aan : `getElementById`, `getElementsByClassName`, etc.

In wat volgt is het echter belangrijk dat je je van het verschil bewust bent.

DOM-tree nodes

Elke node 'n' uit de DOM-tree heeft een aantal properties die te maken hebben met de relatie van 'n' tot sommige andere nodes in de DOM-tree :

- **n.parentNode**
 - geeft de parent van n
 - geeft null indien er geen is, bv. omdat n nog geen parent heeft (bv. als we de node nog maar net hebben aangemaakt, zie verderop) of omdat n een document node is
- **n.childNodes**
 - geeft een lijst die alle children van n bevat
 - dit lijst object is geen array, maar je kan wel `.length` en `[idx]` erop toepassen
- **n.firstChild / n.lastChild**
 - geeft het eerste/laatste child van n
 - geeft null indien er geen is
- **n.nextSibling / n.previousSibling**
 - geeft de volgende/vorige sibling van n
 - geeft null indien er geen is

Meestal zijn we enkel geïnteresseerd in de DOM-tree nodes die HTML-elementen voorstellen. Bovenstaande methods geven ons echter alle soorten nodes. We moeten dan ietwat omslachtige code schrijven om enkel de element nodes eruit te pikken en de overige soorten te negeren.

Gelukkig heeft een node heeft ook enkele methods die dit werk voor ons doen :

- **n.children**
- **n.firstElementChild** / **n.lastElementChild**
- **n.nextElementSibling** / **n.previousElementSibling**

Deze werken op dezelfde manier als de eerdere methods maar leveren enkel de element nodes op.

Let op n.firstElementChild en n.lastElementChild, deze komen vaak van pas bij het gebruik van insertAdjacentHTML() op posities "afterbegin" en "beforeend", als verwijzing naar het nieuwe element!

De DOM-tree kan veel verschillende soorten nodes bevatten, bijvoorbeeld

- element nodes (voor HTML-elementen zoals <h1>, <a>, <p>, etc.)
- text nodes (voor de teksten die in en tussen de elementen staan)
- comment nodes (voor HTML commentaren tussen <!-- en -->)
- ...

Elke node heeft een aantal properties die ons vertellen wat voor soort node het is :

- **n.nodeName**
 - geeft het soort element (als string) indien n een element node is, bv. "IMG", "H1", etc.
 - Let op, deze string zal altijd in hoofdletters staan
 - geeft "#text" indien n een text node is
- **n.nodeType**
 - geeft een getal naargelang welk soort node n is
 - bv. 1 indien n een element node is
 - bv. 3 indien n een text node is
 - zie <https://developer.mozilla.org/en-US/docs/Web/API/Node/nodeType>
- **n.nodeValue** (enkel voor text nodes!)
 - geeft de tekst van de node als string, indien n een text node is
 - geeft null indien n een element node is

De text nodes waarvan hierboven sprake is, zijn extra nodes in de DOM-tree die de teksten bevatten die tussen de element tags staan.

Voorbeeld

Het HTML fragment

```
<p>
    hallo
    <span>dit is een</span>
    tekst
</p>
```

ziet er in de DOM-tree als volgt uit

```
P ELEMENT NODE
  TEXT NODE met nodeValue="hallo" (*)
  SPAN ELEMENT NODE
    TEXT NODE met nodeValue="dit is een"
    TEXT NODE met nodeValue="tekst" (*)
```

() in de nodeValue strings zit ook nog whitespace en newlines, maar da's moeilijk om hier te tonen.*

Je ziet dus dat element nodes nooit zelf 'hun' tekst bevatten, deze wordt altijd in een text node verpakt!

Voor eenvoudige element nodes die enkel tekst bevatten en geen kinderen hebben, is het natuurlijk veel handiger om de **.textContent** property van element nodes te gebruiken i.p.v. de onderliggende text nodes te manipuleren.

Uitprobeeropdracht

Stop het HTML fragment uit bovenstaand voorbeeld in een geldig (!) HTML bestand en geef de paragraaf een id="abc".

Voorzie de pagina van een setup() functie waarin je volgende code zet :

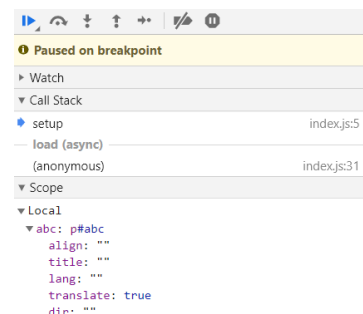
```
let abc=document.getElementById("abc");
console.log("Plaats je breakpoint op deze regel");
```

Open deze pagina in Chrome en open vervolgens de Chrome Developer Tools. Plaats een breakpoint op de console.log() regel en herlaad de pagina.

In de 'Scope' sectie (rechts naast de source code op het 'Sources' tabblad) onder 'Local' kun je de lokale variabelen bekijken.

Hiernaast zie je variabele 'abc' die naar de paragraaf met id="abc" verwijst. Als je die openklapt met het zwarte driehoekje kun je alle properties van dit element bekijken, bv. 'align', 'title', 'lang', etc.

Scroll verder naar beneden en je zult properties voor parentNode en childNodes tegenkomen.



Bekijk in de debugger een deel van de boomstructuur door de parentNode/childNodes properties op te zoeken en open te klappen.

Je zult ook TEKST nodes tegenkomen, ga na wat hun precieze inhoud is.

Bekijk nu het document object (dit is ook een node trouwens). Je kunt dit bv. in de 'Watch' sectie doen (2 secties boven 'Scope') door daar de globale variabele 'document' toe te voegen. Volg weerom de parentNode/childNodes. Welke soorten nodes kom je tegen?

Werken met DOM-tree nodes

In deze sectie wordt uitgelegd hoe we rechtstreeks met node objecten kunnen werken, dit is het krachtigere alternatief voor `.innerHTML` of `insertAdjacentHTML()` waarnaar in de vorige les verwezen werd.

Node aanmaken

De eigenlijke node objecten aanmaken gebeurt via het document object :

- **document.createElement(s)**
 - retourneert een nieuwe element node (dit zit nog niet in de DOM-tree!)
 - de 's' parameter is een string die aangeeft om wat voor element het gaat
 - bv. "img", "h1", etc.
- **document.createTextNode(t)**
 - retourneert een nieuwe text node
 - de 't' parameter is een string met de tekst voor de node
 - (doorgaans is `.textContent` echter een handiger alternatief!)

Bijvoorbeeld

```
let elementNode = document.createElement("a");
let textNode = document.createTextNode("Hello world!");
```

Eenmaal zo'n node is aangemaakt kunnen we er de gewoonlijke zaken mee doen zoals attributen een waarde geven, CSS-classes instellen of event listeners koppelen.

Node toevoegen

Een node 'c' kan aan een parent 'p' worden toegevoegd als laatste kind met

```
p.appendChild(c)
```

Indien 'p' deel uitmaakt van de DOM-tree, zal node 'c' nu ook deel uitmaken van de DOM-tree.

Node verwijderen

Een node 'c' kan uit een parent 'p' worden verwijderd (en dus niet langer een kind zijn van 'p') met

```
p.removeChild(c)
```

Indien 'p' deel uitmaakte van de DOM-tree (en 'c' als kind van 'p' dus ook), zal 'c' nu niet meer in de DOM-tree voorkomen.

Voorbeeld

In de HTML code

```
<div class="gallery">
  
  
  
  
</div>
```

In de Javascript code

```
let gallery = document.querySelector(".gallery");
...
let image = document.createElement("img");
image.setAttribute("src", "images/e.png");
gallery.appendChild(image);           // image toevoegen na laatste kind
...
let someImage=gallery.children[3];
gallery.removeChild(someImage);      // <img> met src="images/d.png"
                                     // d.png wordt verwijderd uit de gallery
```

Conclusie

Rechtstreeks met DOM-nodes werken is heel erg krachtig maar de code wordt al snel onoverzichtelijk.

Indien mogelijk geef je beter de voorkeur aan `.insertAdjacentHTML()` met `first/lastElementChild` of met slimme CSS-selectoren.

Bijvoorbeeld

Om een HTML-fragment als

```
<li class="menu-item"><a href="#">Details</a></li>
```

aan de DOM-tree toe te voegen door zelf nodes aan te maken, moet je vele regels code schrijven. Dit kan veel makkelijker met `.insertAdjacentHTML()` en dit geniet dus de voorkeur.

Stel echter dat je aan de 'Details' hyperlink een click event listener wil koppelen.

Als je zelf de nodes aanmaakt, dan heb je natuurlijk een verwijzing naar het nieuwe `<a>` element en kun je makkelijk de listener eraan koppelen.

Als je `.insertAdjacentHTML()` gebruikt, heb je echter geen verwijzing naar het nieuwe `<a>` element!

Veronderstel de volgende toevoeging, waarbij we toevoegen aan een `` met `id="menu"` :

```
let menu = document.querySelector("#menu");
menu.insertAdjacentHTML("beforeend", '<li class="menu-item"><a href="#">Details</a></li>');
```

We hebben een nieuw list item toegevoegd als laatste kind, maar hebben geen verwijzing naar de link die daarin vervat zit.

Je kan de gewenste verwijzing relatief eenvoudig opvissen met een slimme CSS-selector :

```
let link = document.querySelector("#menu>li:last-of-type a"); // let op de spatie!
```

Of je kan het met `.lastElementChild` proberen :

```
let nieuweListItem = menu.lastElementChild; // nieuwe <li> is laatste kind van #menu
let link = nieuweListItem.lastElementChild; // link is laatste kind van nieuwe list item
```


Opdracht 01

Unzip 'beginsituatie opdracht 01.zip' in een nieuw Webstorm project of folder.

De HTML en CSS code voor deze opdracht is al gegeven :

Oplossing opdracht 01

Ingredienten :

- boter
- kaas
- eieren

Instructies :

- klik op een ingredient om het te verwijderen

Voeg de nodige Javascript code toe zodat een klik op een ingredient, dit ingredient verwijdert.

Opdracht 02

Unzip 'beginsituatie opdracht 02.zip' in een nieuw Webstorm project of folder.

De HTML en CSS code voor deze opdracht is al gegeven :

Oplossing opdracht 02

Ingredienten :

- boter [verwijder](#)
- kaas [verwijder](#)
- eieren [verwijder](#)

Instructies :

- klik op een 'verwijder' link om een ingredient te verwijderen

Voeg de nodige Javascript code toe zodat een klik op een 'verwijder' link, het corresponderende ingredient verwijdert (incl. de 'verwijder' link natuurlijk).

Opdracht 03

Unzip 'beginsituatie opdracht 03.zip' in een nieuw Webstorm project of folder.

De HTML en CSS code voor deze opdracht is al gegeven :

Oplossing 01 voor opdracht 03

Type hier een ingredient

Ingredienten :

- boter [verwijder](#)
- kaas [verwijder](#)
- eieren [verwijder](#)

Instructies :

- typ een ingredient in het tekstveld
- klik vervolgens op de "Voeg toe" knop
- klik op een 'verwijder' link om een ingredient te verwijderen

Voeg de nodige Javascript code toe zodat

- een klik op een 'verwijder' link, het corresponderende ingredient verwijdt (incl. de link)
- er een ingredient bijkomt als men de naam in het tekstveld typt en op 'Voeg toe' klikt

Je kunt je hierbij baseren op je oplossing van opdracht 02 in combinatie met de oplossing van een gelijkaardige opdracht uit de vorige les.

Custom attributen

Het is toegelaten om zelfverzonnen attributen aan een element toe te voegen, op voorwaarde dat hun naam met "**data-**" begint.

Custom attributen zijn handig om extra data aan specifieke elementen uit de DOM-tree te hangen en komen in de begin tag terecht (net als voorgedefinieerde attributen, zoals src of href).

Bijvoorbeeld :

```

```

Ze kunnen gewoon in het HTML-document staan, maar ze komen ook voor in HTML-strings als er elementen dynamisch worden toegevoegd met bv. insertAdjacentHTML().

Als je een DOM-tree node hebt, kun je diens custom attributen met de gekende `getAttribute()` en `setAttribute()` methods manipuleren.

Bijvoorbeeld :

```
let img = ...  
let idx = img.getAttribute("data-index");  
...  
img.setAttribute("data-index", "5");
```

De waarde van een custom attribuut is, net zoals bij voorgedefinieerde attributen, altijd een **string**!

Waarvoor kun je custom attributen gebruiken?

Als je extra informatie in een DOM-tree element wil bijhouden doe je dit met 1 of meerdere custom attributen in dat element.

Stel dat je een webapplicatie wil schrijven om personen en vergaderingen te beheren.

De gegevens van een persoon zullen in je programma netjes in Javascript objecten worden bijgehouden, net als de gegevens over een vergadering (Javascript objecten komen in een volgende les aan bod). Dus per persoon 1 object, alsook per vergadering 1 object.

Ergens in je UI zal een aantal van die personen gevisualiseerd worden, bv. de lijst van genodigden voor een specifieke vergadering. Veronderstel dat we per persoon een `` voorzien in die lijst.

Stel nu dat we deelnemers uit de vergadering kunnen verwijderen door erop te klikken. De code van de click event listener moet kunnen achterhalen wat het bijbehorende persoon object is, zodat we dit uit het vergadering object kunnen verwijderen. Het is dus niet voldoende om enkel het `` element uit de UI te verwijderen, er moet ook iets gebeuren met de achterliggende data in Javascript.


Een mogelijke oplossing hiervoor is een zelfgekozen attribuut "data-persoon" aan elk `` element toe te voegen. In de click event listener kun je dan de waarde van dit attribuut opvragen en zo het persoon object terugvinden.

Wat de waarde van dit "data-persoon" attribuut is moet je zelf bedenken, bijvoorbeeld

- indien elk persoon object een uniek nummer heeft zou je dit nummer kunnen gebruiken. Dan noem je het attribuut beter gelijk ook "data-persoonNummer"
- indien je geen nummer hebt maar alle persoon objecten zitten in een array, dan zou je de index in dit array kunnen bijhouden in een attribuut "data-persoonIndex".

Merk op dat het geen goed idee is om hiervoor het id-attribuut van de `` elementen te misbruiken

- indien er meerdere vergaderingen op de pagina staan kunnen er nml. meerdere `` items voor die ene persoon voorkomen en zitten we met dubbele waarden voor het id-attribuut.
- een vergaderingsobject zal wellicht ook een id hebben en als we daar ook het id-attribuut voor gebruiken kunnen we ook conflicten krijgen tussen id's van personen en id's van vergaderingen.

Al bij al gebruik je in de praktijk trouwens veel minder vaak id-attributen dan de kleine oefeningen in deze cursus doen vermoeden 

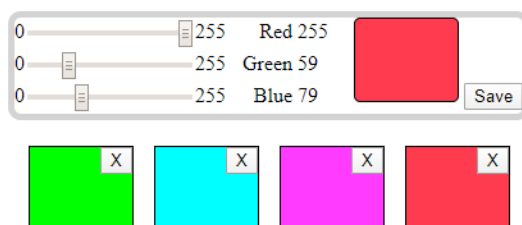
Opdracht : Colorpicker uitbreiding

Bekijk eerst het korte filmpje dat het eindresultaat demonstreert.

Maak een nieuw project dat een kopie is van de voorbeeldoplossing van de colorpicker opdracht.

Voeg rechts aan de colorpicker component een 'Save' knop toe.

Een klik op de 'Save' knop voegt onderaan een kopie van de swatch toe en een bijbehorende delete knop (met 'X' als opschrift). De swatch is dit keer rechthoekig (niet afgerond) en de delete knop staat netjes in de rechterbovenhoek van deze gekleurde rechthoek.



Een klik op de toegevoegde swatch, stelt de colorpicker component in op de bewaarde kleur.

Een klik op de bijbehorende delete knop verwijdert de bewaarde swatch weer.

Tips

Dit is een behoorlijk pittige opdracht waarin veel van de geziene leerstof gecombineerd wordt. Voordat je eraan begint, bestudeer eerst nog eens aandachtig de oplossing van de colorpicker opdracht.

Hieronder duidt b-swatch op een rechthoekige swatch die toegevoegd werd als de gebruiker op 'Save' klikte. Dit om verwarring te vermijden met de swatch bovenaan naast de sliders.

Voeg de functionaliteit best in deze volgorde toe

1. Een klik op 'Save' voegt een b-swatch toe met de juiste achtergrondkleur
 - Voorzie al de delete knop maar koppel nog geen click gedrag aan de b-swatch of de delete knop.
 - De layout is geen prioriteit maar als je je geroepen voelt : de delete knop wordt absoluut gepositioneerd en je krijgt hem in de rechterbovenhoek door de 'top' en 'right' CSS-properties in te stellen.
2. Een klik op een b-swatch herstelt de waarden van 3 sliders naar de bewaarde kleur
 - De waarden van de sliders voor een bewaarde kleur, stop je in drie custom attributen (data-red, data-green en data-blue) van het b-swatch element.
 - In de click event listener achterhaal je met event.target het b-swatch element, vraagt de waarden van diens 3 custom attributen op, gebruikt ze om de sliders te herstellen en roept tenslotte update() op.

3. Een klik op een delete knop verwijdert de corresponderende b-swatch
 - Hier is er een moeilijkheid omdat de delete knop een child is van het b-swatch element. Een klik op de delete knop zal dus (wegens event bubbling) ook de click event listener van het b-swatch element activeren.
 - Je zult dus in die event listener een onderscheid moeten maken tussen een directe klik op de b-swatch en een klik op de delete knop.