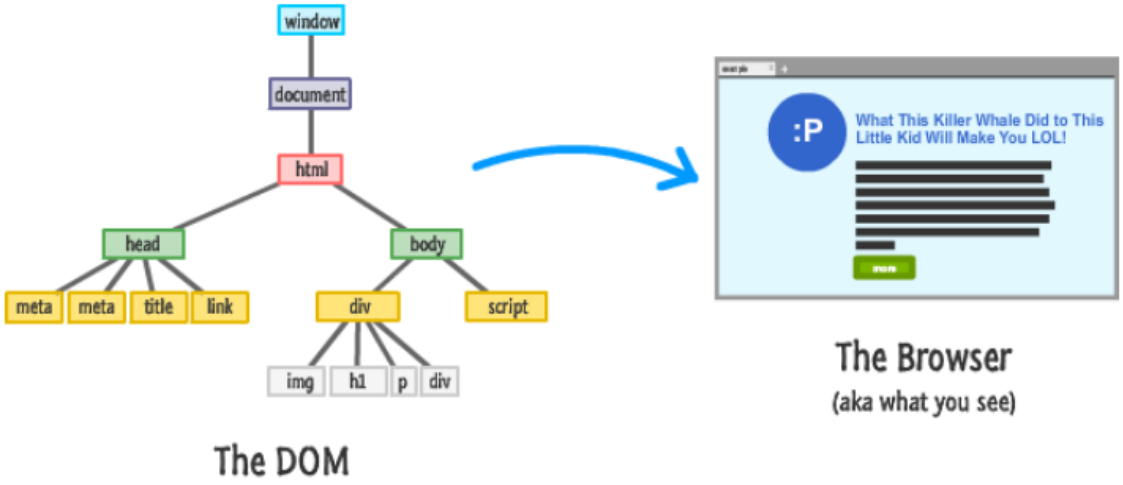
Temat 16

Temat: Model DOM 3- Przechodzenie przez DOM, tworzenie i usuwanie elementów

Przechodzenie przez DOM

Wiesz już, że DOM wygląda jak drzewo. Elementy w DOM są ułożone w hierarchii, która definiuje to, co ostatecznie widzisz w przeglądarce:



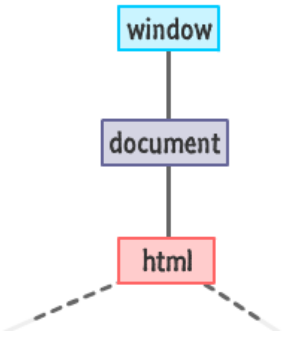
Aby operować na takich obiektach, musimy dobrze opanować sztukę "spacerowania" po nich.

Każdy element na stronie tworzy tak zwany **node** czyli pojedynczy węzeł drzewa. Takimi nodami są nie tylko elementy, ale także tekst w nich zawarty. Nas głównie będą interesować nody, które są elementami - np. buttony, divy itp.

Znalezienie właściwej drogi

Zanim znajdziesz elementy i wykonasz na nich operacje, musisz najpierw dotrzeć do miejsca, w którym znajdują się te elementy. Najprostszym sposobem rozwiązania tego problemu jest po prostu rozpoczynanie od góry i przesuwanie w dół.

Widok z góry DOM składa się z elementów okna , dokumentu i html :



Ze względu na to, jak ważne są te trzy rzeczy, DOM zapewnia łatwy dostęp do nich poprzez **`window`**, **`document`** i **`document.documentElement`**:

```
let windowObject = window;
let documentObject = document;
let htmlElement = document.documentElement;
```

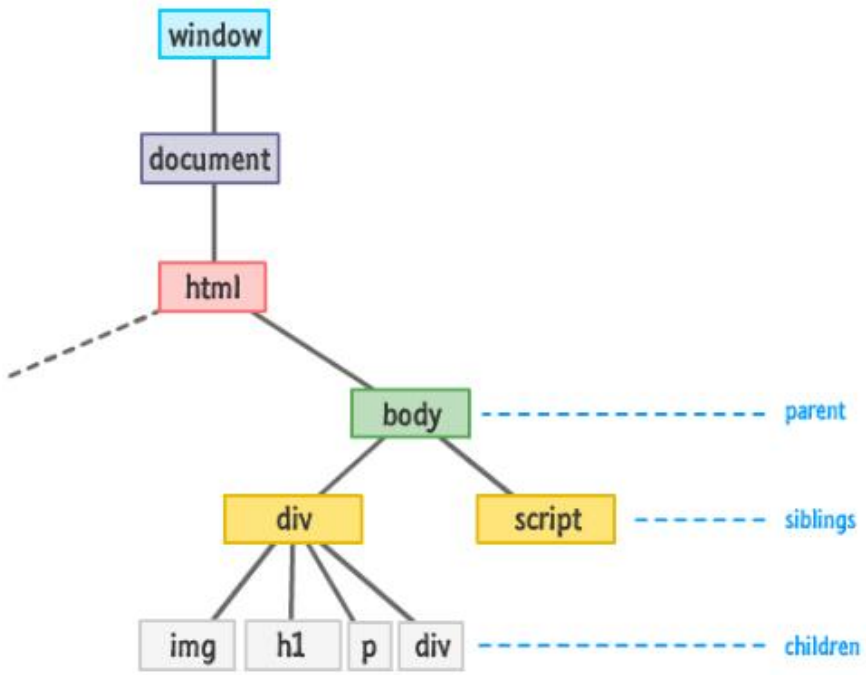
Należy zauważyć, że zarówno **`window`**, jak i **`document`** są właściwościami globalnymi. Nie musisz ich jawnie deklarować.

Najwyższe węzły drzewa są dostępne bezpośrednio jako właściwości document:

- **`<html> = document.documentElement`** – najwyższy węzeł dokumentu to document.documentElement. To jest węzeł DOM tagu <html>.
- **`<body> = document.body`** – szeroko stosowany węzeł DOM
- **`<head> = document.head`** –<head> Znacznik jest dostępny przez document.head.

Gdy przejdziesz poniżej poziomu elementu HTML, DOM zaczyna się rozgałęziać. W tym momencie istnieje kilka sposobów poruszania się. Jednym ze sposobów jest użycie znanych Ci już **`querySelector i querySelectorAll`** pozwalające precyzyjnie uzyskać pożądane elementy. Gdy jednak **nie wiesz, gdzie chcesz się udać**, metody querySelector i querySelectorAll nie będą wystarczające.

Warto pamiętać, że wszystkie elementy w DOM mają co najmniej jedną kombinację **rodziców** , **rodzeństwa** i **dzieci:**



Niemal każdy element, w zależności od punktu startowego, może odgrywać wiele rodzinnych ról.

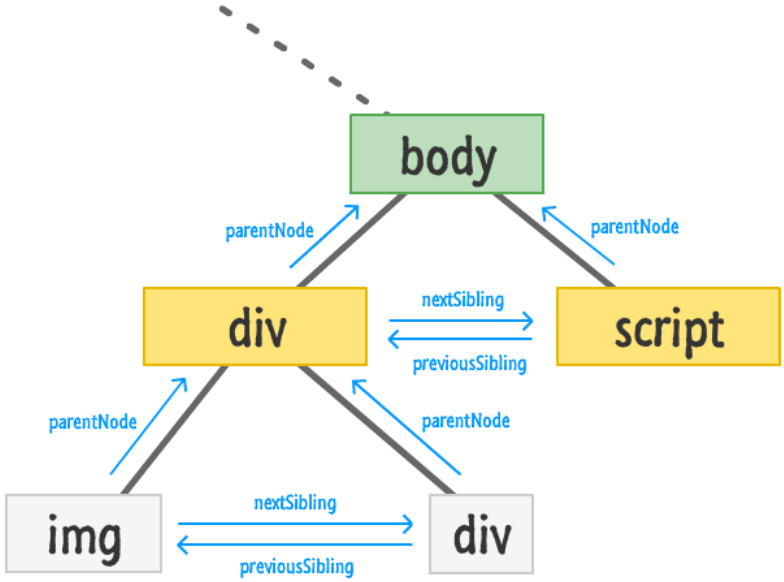
Do przechodzenia między nimi można użyć kilku właściwości, np.: **`firstChild, lastChild, parentNode, children, previousSibling, nextSibling`**

Rodzeństwo i rodzic

Z tych właściwości najłatwiej jest poradzić sobie z rodzicami i rodzeństwem. Odpowiednie właściwości to **`parentNode , previousSibling i nextSib`**ling .
Rodzeństwo to węzeł, który jest dzieckiem tego samego rodzica. Na przykład <head> i <body> są rodzeństwem:

- <body> - mówimy, że jest "następnym" lub "prawym" rodzeństwem <head>,
- <head> -mówimy, że jest "poprzednim" lub "pozostawionym" rodzeństwem <body>.

Poniższy schemat przedstawia sposób działania tych trzech właściwości:



Właściwość **`parentNod`**e wskazuje na element nadrzędny elementu. Właściwości **`previousSibling i nextSibling`** umożliwiają elementowi znalezienie poprzedniego lub następnego rodzeństwa.

Dzieci: `childNodes, firstChild, lastChild, children`

Są dwa terminy, z których będziemy od teraz korzystać:

- **Węzły potomne (lub dzieci)** - elementy będące bezpośrednimi dziećmi. Innymi słowy, są one zagnieżdżone dokładnie w danym elemencie. Na przykład, <head> i <body> są dziećmi elementu <html>.
- **Potomkowie** - wszystkie elementy, które są zagnieżdżone w danym, w tym dzieci, ich dzieci i tak dalej.

Na przykład w poniższym kodzie:

```
<html>
<body>
  <div>Begin</div>

  <ul>
    <li>
      <b>Information</b>
    </li>
  </ul>
</body>
</html>
```

<body> ma dzieci i <ul> (i kilka pustych węzłów tekstowych).
Z kolei wszyscy potomkowie <body>, to bezpośrednie dzieci: , <ul>, ale także więcej elementów zagnieżdżonych, takich jak <li> (jest dzieckiem <ul>) i <b> (jest dzieckiem <li>).

`childNodes` – Kolekcja, zapewnia dostęp do wszystkich węzłów potomnych, w tym węzły tekstowe.

Kolekcja ta ma również długość, czyli właściwość, która podaje liczbę dzieci.

Przykład 1. Przygotuj stronę html wykorzystującą poniższy kod.

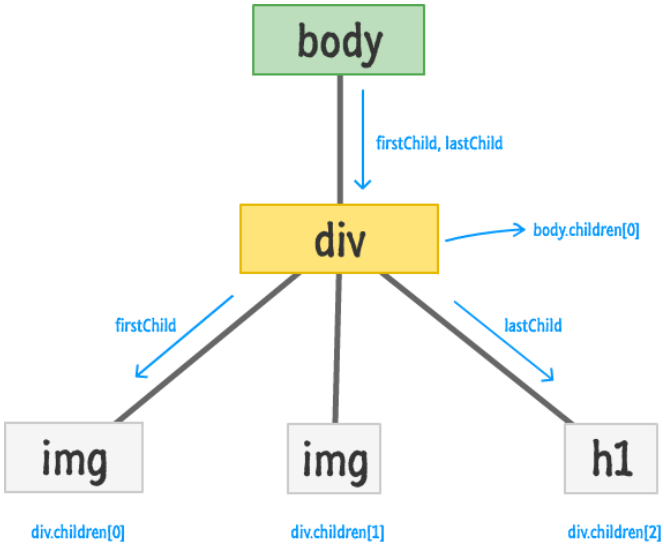
```
<body>
    <div>Początek</div>
    <ul>
        <li>Informacje</li>
    </ul>
    <div>Koniec</div>

    <script>
    for (let i = 0; i < document.body.childNodes.length; i++) {
      alert( document.body.childNodes[i] ); // Text, DIV, Text, UL, ...,
      SCRIPT
    }
    console.log(document.body.childNodes);
    </script>
    ...więcej rzeczy...
</body>
```

Zwróć uwagę na interesujący szczegół tutaj. Jeśli uruchomimy powyższy przykład, ostatni pokazany element to <script>. W rzeczywistości dokument ma więcej rzeczy poniżej, ale w momencie wykonania skryptu przeglądarka go jeszcze nie przeczytała, więc skrypt go nie widzi.

`childNodes` wygląda jak tablica. Ale tak naprawdę to nie jest tablica, ale raczej *kolekcja* - specjalny obiekt podobny do tablicy, który możemy przeglądać tak jak tablicę. Ta kolekcja ma również długość - właściwość, która podaje liczbę dzieci.

`firstChild , lastChild i children`



Właściwości **`firstChild i lastChild`** odnoszą się do pierwszego i ostatniego elementu potomnego rodzica. Jeśli rodzic ma tylko jedno dziecko, tak jak w przypadku elementu body w przykładzie na powyższym diagramie, to zarówno firstChild, jak i lastChild wskazują na to samo. Jeśli element nie ma elementów podrzędnych, wówczas te właściwości zwracają wartość **`null`** .

Sprawdzanie, czy dziecko istnieje

Aby sprawdzić, czy element ma dziecko, możesz wykonać:

Przykład 2. Przygotuj stronę html wykorzystującą poniższy kod.

```
<body>
    <div>Początek</div>
    <ul>
        <li>Informacje</li>
    </ul>
    <div id="div2"></div>
    <script>
    const bodyElement = document.body;
    console.log(bodyElement.childNodes);
    console.log(bodyElement.children);
    for (let i = 0; i < bodyElement.children.length; i++) {
      console.log(bodyElement.children[i]);
      console.log(bodyElement.children[i].tagName);
    }
```

```
    //Sprawdzanie, czy istnieje dziecko w body i wyświetlenie kilku
    dzieci
    if (bodyElement.firstChild) {
        console.log("są dzieci");
        console.log("Pierwsze dziecko:"+bodyElement.firstChild);
        console.log("Drugie dziecko:"+
        bodyElement.firstChild.nextSibling);
        console.log("Ostatnie dziecko:"+bodyElement.lastChild);
    }
    else {
        console.log("dzieci brak");
    }
    //Sprawdzanie, czy istnieje dziecko  w drugim div
    let divElement = document.getElementById('div2');
    if (divElement.firstChild) {
        console.log("W drugim divie są dzieci");
    }
    else {
        console.log("W drugim divie dzieci brak");
    }
    </script>
/body>
```

Przykład 3. Przygotuj stronę html wykorzystującą poniższy kod.

```
<body>
    <div class="text-cnt">
    <p id="text">
        Mała
        <strong style="color:red">Ala</strong>
        miała
        <span style="color:blue">kota</span>
    </p>
    </div>
    <script>
    const text = document.querySelector('#text');

        console.log(text.parentElement) //wskazuje na nadrzędny nod
        będący elementem - div.text-cnt
        console.log(text.parentNode) //wskazuje na nadrzędny nod -
        div.text-cnt

        console.log(text.firstChild) //"Mała "
        console.log(text.lastChild) //"" - html jest sformatowany,
        wiec ostatnim nodem jest znak nowej linii

        console.log(text.firstElementChild) //pierwszy element -
        <strong style="color:red">Ala</strong>
        console.log(text.lastElementChild) //ostatni element - <span
        style="color:blue">kota</span>

        console.log(text.children); //[strong, span] - kolekcja
        elementów
        console.log(text.children[0]) //wskazuje na 1 element -
        <strong style="color:red">Ala</strong>
```

```
    console.log(text.childNodes) //[text, strong, text] - kolekcja
    wszystkich dzieci - nodow
    console.log(text.childNodes[0]) //"Mała"

    console.log(text.firstElementChild.nextElementSibling)
    //kolejny brat-element pierwszego elementu - <span
    style="color:blue">kota</span>
    console.log(text.firstElementChild.nextSibling) //kolejny
    brat-node pierwszego elementu - "miała"

    console.log(text.firstElementChild.previousElementSibling)
    //poprzedni brat-element pierwszego elementu - null, bo przed
    pierwszym strong nie ma elementow
    console.log(text.firstElementChild.previousSibling)
    //poprzedni brat-node pierwszego elementu - "Mała"

    //powyższe możemy łączyć
    console.log("Połączenia właściwości:");
    console.log(text.children[0].firstChild) //pierwszy element i
    w nim pierwszy nod : "Ala"
    console.log(text.children[0].firstElementChild) //null - w
    pierwszym strong nie mamy juz elementów
    console.log(text.firstChild.firstElementChild) //undefined -
    nie ma elementu w pierwszym tekście
    console.log(text.firstElementChild.firstElementChild) //null -
    nie ma elementy w strong
    console.log(text.firstElementChild.firstChild) //"Ala"
</script>
ody>
```

Tworzenie i usuwanie elementów

Bardzo często aplikacje i aplikacje interaktywne dynamicznie tworzą elementy HTML i wstawiają je do DOM.

Tworzenie obiektu za pomocą `createElement`

Aby utworzyć nowy element na stronie możemy skorzystać z metody

```
document.createElement(typ)
```

Sposób działania createElement jest dość prosty. Wywołujesz go za pośrednictwem obiektu dokumentu i podajesz nazwę tagu elementu, który chcesz utworzyć. W poniższym fragmencie tworzysz element akapitu reprezentowany przez literę p :

```
document.createElement("p");
```

Po utworzeniu nowego elementu warto też ustawić jego właściwości:

```
let el = document.createElement("div");

el.id = "myDiv";
el.innerText = "Tekst w divie";
el.setAttribute("title", "To jest tekst w dymku");
el.classList.add("module");
el.style.setProperty("background-color", "#FF6633");
```

Jeśli uruchomisz tę linię kodu jako część aplikacji, zostanie ona wykonana i zostanie utworzony element. Tworzenie elementu jest prostą częścią. To jednak nie wystarczy. Utworzony w powyższy sposób element jest dostępny dla skryptu, ale nie ma go jeszcze w drzewie dokumentu. Musimy go tam wstawić metodą

```
parentElement.appendChild(nowyElement)
```

Przykład 4. Przygotuj stronę html wykorzystującą poniższy kod.

```
<!DOCTYPE html>
<html lang="pl">
<head>
    <meta charset="utf-8">
    <title>DOM, przechodzenie po drzewie - T16p4</title>
    <style>
    body{
    background-color: #025;
    color: #fff;
    font-size: 18px;
    }
    .test-first{
        background-color: #250;
        height: 200px;
        width: 200px;
        padding: 20px;
    }
    .module{
        background-color: #555;
        border: 3px solid #00f;
        color: #fff;
        height: 100px;
        width: 100px;
    }
    </style>
</head>
<body>
    <div class="test-first">
    </div>
    <script>
        //tworzymy element
        const el = document.createElement("div");
        el.id = "myDiv";
        el.innerText = "Tekst w divie";
        el.setAttribute("title", "To jest tekst w dymku");
        el.classList.add("module");
        //pobieramy miejsce docelowe
        const div = document.querySelector(".test-first");
        // wstawianmy element do drzewa dokumentu
        div.appendChild(el);
    </script>
</body>
</html>
```

Aby utworzyć węzły DOM, istnieją dwie metody:

- `document.createElement(tag)` – Tworzy nowy element z podanym znacznikiem:
  `let div = document.createElement('div');`
- `document.createTextNode(text)` –Tworzy nowy *węzeł tekstowy* z podanym tekstem:
  `let textNode = document.createTextNode('Jestem tu');`

Do wstawiania elementów na stronę używaliśmy dotychczas innerHTML. Za pomocą innerHTML wstawiamy kod html w dany element. Ta właściwość nie daje jednak referencji do elementów we wstawianym html. Bardzo często będziemy chcieli za chwilę wykonać jakąś akcję na wstawianych elementach - np podpiąć im kliknięcie, zmienić tekst itp. Jeżeli będziemy korzystać z innerHTML, będziemy musieli te elementy po wstawieniu dodatkowo pobrać.

Przykład 5. Przygotuj stronę html wykorzystującą poniższy kod.

```
<body>
    <h1 id="theTitle" class="summer">Co tu się dzieje?</h1>
    <button onclick =addText()>Dodaj tekst</button>
    <script>
        function addText(){
            const newElement = document.createElement("p");
            const text1 = prompt("podaj tekst do wstawienia");
            newElement.textContent = text1;
            document.body.appendChild(newElement);
        }
    </script>
</body>
```

Zwróć uwagę, że **`parentElem.appendChild(node)`** dołącza element **`node`** jako ostatnie dziecko **`parentElem.`**

Przykład 6. Przygotuj stronę html wykorzystującą poniższy kod.

```
<body>
    <h1 id="theTitle">Lista zakupów</h1>
    <button onclick =insert()>Dodaj pozycję</button>
    <ol>
        <li>pieczywo</li>
    </ol>
    <script>
        function insert(){
            const list1 = document.querySelector("ol");
            const newLi = document.createElement("li");
            const text1 = prompt("podaj co jeszcze kupić");
            newLi.textContent = text1;
            list1.appendChild(newLi);
        }
    </script>
</body>
```

Jeśli chcesz wstawić nowy element w innym miejscu, możesz to zrobić, wywołując funkcję **`insertBefore`**. Funkcja **`insertBefore`** przyjmuje dwa argumenty.

- Pierwszym argumentem jest element, który chcesz wstawić.
- Drugi argument jest odniesieniem do rodzeństwa (znanego również jako dziecko rodzica), które chcesz poprzedzić.

Przykład 7. Przygotuj stronę html wykorzystującą poniższy kod.

```
<body>
    <h1 id="theTitle">Lista zakupów</h1>
    <button onclick =insert1()>Dodaj pozycję na drugim miejscu</button>
    <button onclick =insert2()>Dodaj pozycję na przedostatnim miejscu
    </button>
    <button onclick =insert3()>Dodaj pozycję na drugim miejscu od końca
    </button>
    <ol>
        <li>pieczywo</li>
    </ol>
    <script>
        const list1 = document.querySelector("ol");
        function insert1(){
            const newLi = document.createElement("li");
            const text = prompt("podaj co jeszcze kupić");
            newLi.textContent = text;
            list1.insertBefore(newLi, list1.children[1]);
        }
        function insert2(){
            const newLi = document.createElement("li");
            const text = prompt("podaj co jeszcze kupić");
            newLi.textContent = text;
            list1.insertBefore(newLi, list1.lastChild);
        }
        function insert3(){
            const newLi = document.createElement("li");
            const text = prompt("podaj co jeszcze kupić");
            newLi.textContent = text;
            list1.insertBefore(newLi, list1.lastChild.previousSibling);
        }
    </script>
</body>
```

Usuwanie elementów

Aby usunąć element, możemy:

1. wywołać funkcję **`removeChild`** na rodzicu elementu, który chcemy usunąć.

Jeżeli nie mamy bezpośredniego dostępu do elementu nadrzędnego i nie chcemy tracić czasu na znajdowanie go, można usunąć ten element za pomocą właściwości **`parentNode.`**

```
Element.parentNode.removeChild(Element);
```

Przykład 8.

```
<body>
    <div class="test-first">
    </div>
    <button onclick =add()>Dodaj element</button>
    <button onclick =del()>Usuń element</button>
    <script>
        function add(){
            const el = document.createElement("div");
            el.id = "myDiv";
            el.innerText = "Tekst w divie";
            el.classList.add("module");
            const div = document.querySelector(".test-first");
            div.appendChild(el);
        }
        function del(){
            const div = document.querySelector(".test-first");
            const el = div.lastChild; // ostatnie dziecko
            el.parentNode.removeChild(el); // przechodzimy do rodzica,
            aby usunąć element
        }
    </script>
</body>
```

Usuwamy element , wywołując **`removeChild`** na jego obiekcie nadrzędnym, podając **`element.parentNode`** . Wygląda to okrężnie, ale działa dobrze.

2. wywołać funkcję **`node.remove()`**, która usuwa node z jej miejsca. Np.:
   `const p = document.querySelector("#paragraf");`
   `p.remove();`

W przeciwieństwie do **`removeChild`** metoda **`remove()`** nie jest wspierana przez przeglądarki IE. Jeżeli musisz je wspierać, wtedy powinieneś użyć removeChild.