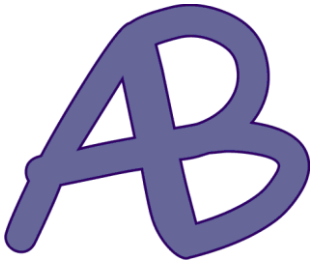
Aplikacje frontendowe



Wykład

Manipulowanie DOM-em
Tworzenie elementów HTML

Tworzenie i usuwanie elementów HTML

Metoda appendChild służy do dodawania nowego elementu jako ostatniego dziecka danego węzła (elementu) w drzewie DOM (Document Object Model).

```
parentNode.appendChild(newChild);
```

Przed dodaniem element należy utworzyć za pomocą metody createElement()

```
// Tworzymy nowy element <div>
let newDiv = document.createElement("div");
newDiv.textContent = "To jest nowy div";

// Znajdujemy element, do którego dodamy nowy element
let parentElement = document.getElementById("container");

// Dodajemy nowy element jako ostatnie dziecko elementu 'container'
parentElement.appendChild(newDiv);
```

Tworzenie i usuwanie elementów HTML

Dodawanie kilku elementów

Tworzy trzy nowe elementy listy <li> i dodaje je do elementu o id „myList”.

```
let list = document.getElementById("myList");
let items = ["Element 1", "Element 2", "Element 3"];

items.forEach((text) => {
  let listItem = document.createElement("li");
  listItem.textContent = text;
  list.appendChild(listItem);
});
```

Tworzenie i usuwanie elementów HTML

Przenoszenie istniejącego elementu

Można także przenosić istniejące element w inne miejsca strony

```
let item = document.getElementById("item");
let newParent = document.getElementById("newParent");

// Przenosimy element 'item' do nowego rodzica 'newParent'
newParent.appendChild(item);
```

- ✓ Jeśli newChild już istnieje w DOM, zostanie przeniesiony z aktualnej lokalizacji do nowej.
- ✓ Element newChild może mieć tylko jednego rodzica w danym czasie. Przeniesienie go spowoduje jego usunięcie z aktualnego rodzica.
- ✓ Metoda appendChild zwraca dodany element newChild.

Tworzenie i usuwanie elementów HTML

DocumentFragment

- ✓ DocumentFragment to węzeł dokumentu, który może służyć jako kontener do grupowania innych węzłów.
- ✓ Typowym zastosowaniem jest utworzenie go, złożenie w nim poddrzewa DOM, a następnie dołączenie lub wstawienie fragmentu do modelu DOM
- ✓ Jest to węzeł typu dokumentu, który nie jest częścią aktywnego dokumentu DOM, co oznacza, że operacje na nim są wydajniejsze, ponieważ nie powodują wielokrotnego renderowania strony.

Korzyści związane z wydajnością są często zawyżane. W rzeczywistości w niektórych silnikach użycie a jest wolniejsze niż dołączanie do dokumentu w pętli.

Tworzenie i usuwanie elementów HTML

Przykład:

Utworzenie fragmentu zawierającego podpunkty listy na podstawie tablicy i dodanie go w całości do listy

```
const ul = document.querySelector("ul");
const fruits = ["Apple", "Orange", "Banana", "Melon"];

const fragment = new DocumentFragment();

for (const fruit of fruits) {
  const li = document.createElement("li");
  li.textContent = fruit;
  fragment.append(li);
}

ul.append(fragment);
```

W tym przykładzie fragment tworzony jest w pamięci a potem dołączany do DOM przy użyciu metody append.

Tworzenie i usuwanie elementów HTML

Przykład:

Utworzenie fragmentu zawierającego podpunkty listy na podstawie tablicy i dodanie go w całości do listy

```
const ul = document.querySelector("ul");
const fruits = ["Apple", "Orange", "Banana", "Melon"];

const fragment = new DocumentFragment();

for (const fruit of fruits) {
  const li = document.createElement("li");
  li.textContent = fruit;
  fragment.append(li);
}

ul.append(fragment);
```

W tym przykładzie fragment tworzony jest w pamięci a potem dołączany do DOM przy użyciu metody append.

Tworzenie i usuwanie elementów HTML

Przykład: - wersja 2

```
const ul = document.querySelector("ul");
const fruits = ["Apple", "Orange", "Banana", "Melon"];

const fragment = document.createDocumentFragment();

for (const fruit of fruits) {
  const li = document.createElement("li");
  li.textContent = fruit;
  fragment.appendChild(li);
}

ul.appendChild(fragment);
```

Można też utworzyć fragment jako nie powiązaną gałąź dokumentu a następnie przenieść go do właściwego modelu DOM

Tworzenie i usuwanie elementów HTML

Metoda Element.remove() usuwa element z modelu DOM.

```
const element = document.getElementById(„id_elementu");
element.remove(); // Usuwa element
```

Tworzenie i usuwanie elementów HTML

Metoda removeChild() interfejsu usuwa węzeł podrzędny z modelu DOM i zwraca usunięty węzeł.

```
  <div id="parent">
    <div id="child"></div>
  </div>
//----------------------------------------------------
    const parent = document.getElementById("parent");
    const child = document.getElementById("child");
    const throwawayNode = parent.removeChild(child);
```

Tworzenie i usuwanie elementów HTML

Aby usunąć określony element bez konieczności określania jego węzła nadrzędnego:

```
const node = document.getElementById("child");
if (node.parentNode){
  node.parentNode.removeChild(node);
}
```

Sprawdzamy, czy odnaleziony element ma rodzica

Aby usunąć wszystkie elementy podrzędne z elementu:

```
const element = document.getElementById("idOfParent");
while (element.firstChild) {
  element.removeChild(element.firstChild);
}
```

Dopóki istnieje pierwszy potomek – usuwamy pierwszego potomka

Tworzenie i usuwanie elementów HTML

Aby usunąć określony element bez konieczności określania jego węzła nadrzędnego:

```
const node = document.getElementById("child");
if (node.parentNode){
  node.parentNode.removeChild(node);
}
```

Sprawdzamy, czy odnaleziony element ma rodzica

Aby usunąć wszystkie elementy podrzędne z elementu:

```
const element = document.getElementById("idOfParent");
while (element.firstChild) {
  element.removeChild(element.firstChild);
}
```

Dopóki istnieje pierwszy potomek – usuwamy pierwszego potomka

Manipulowanie DOM-em
Atrybuty i style

Atrybuty

Atrybuty:

Metoda getAttribute() interfejsu Element zwraca wartość określonego atrybutu

Jeśli dany atrybut nie istnieje, zwrócona wartość będzie równa .null

```
//w HTML
        <div id="div1">Hi Champ!</div>

//w konsoli
      const div1 = document.getElementById("div1");
      //=> <div id="div1">Hi Champ!</div>
      const exampleAttr = div1.getAttribute("id");
      //=> "div1"
      const align = div1.getAttribute("align");
      //=> null
```

Atrybuty

Atrybuty:

Metoda setAttribute() interfejsu Element ustawia wartość atrybutu dla określonego elementu. Jeśli atrybut już istnieje, wartość jest aktualizowana; W przeciwnym razie dodawany jest nowy atrybut o określonej nazwie i wartości.

Aby uzyskać bieżącą wartość atrybutu, należy użyć getAttribute(); Aby usunąć atrybut, wywołaj metodę removeAttribute().

```
const button = document.querySelector("button");

button.setAttribute("name", "helloButton");
button.setAttribute("disabled", "");
```

Style

Style:

W JavaScript możemy modyfikować style elementów HTML na kilka różnych sposobów:

1. Używanie właściwości `style`. Pozwala ona bezpośrednio modyfikować styl elementu. Korzystamy z niej, aby ustawić poszczególne właściwości CSS.

```
let element = document.getElementById("myElement");

element.style.color = "red";
element.style.backgroundColor = "yellow";
element.style.fontSize = "20px";
```

Style

2. **Używanie klasy CSS** - Stylowanie elementów za pomocą klas CSS jest bardziej przejrzyste i skalowalne. Możemy dodawać lub usuwać klasy za pomocą metod `classList.add` i `classList.remove`.

```css
//css
.tlo {
  color: white;
  background-color: blue;
  font-weight: bold;
}
```

```javascript
//JavaScript
    let element = document.getElementById("myElement");
    element.classList.add(„tlo");
```

Style

3. Możemy także ustawić wiele stylów jednocześnie za pomocą właściwości `cssText`.

```
let element = document.getElementById('myElement');
element.style.cssText =
    'color: purple; background-color: lightgray; border: 1px solid black;';
```

Style

4. Możemy dynamicznie tworzyć nowe reguły stylów i dodawać je do dokumentu

```
let style = document.createElement("style");
style.textContent = `
.myClass {
color: blue;
background-color: lightgreen;
padding: 10px;
}
`;
document.head.appendChild(style);
```

Powyższy kod tworzy nowy element `<style>`, definiuje styl dla klasy o nazwie`.myClass`, a następnie dodaje go do nagłówka dokumentu

Dzięki temu możemy oscylować wiele elementów jednocześnie

Słuchacze zdarzeń

Słuchacze zdarzeń

Metoda addEventListener() służy do dodawania obsługi zdarzeń do wybranych elementów DOM. Pozwala na przypisanie funkcji, która zostanie wywołana, gdy wystąpi określone zdarzenie na danym elemencie.

```
element.addEventListener(event, function, useCapture);
```

Parametry:

- event: Typ zdarzenia jako ciąg znaków, np. 'click', 'mouseover', 'keydown’.
- function: Funkcja, która zostanie wywołana, gdy zdarzenie wystąpi.
- useCapture (opcjonalnie): Wartość boolean, określająca, czy zdarzenie ma być obsługiwane w fazie wychwytu (capture phase). Domyślnie jest ustawiona na false

Słuchacze zdarzeń

Obsługa kliknięcia na przycisku:

```
let button = document.getElementById("myButton");
button.addEventListener("click", function () {
  alert("Przycisk został kliknięty!");
});
```

Obsługa najechania myszką na element::

```
let div = document.getElementById("myDiv");
div.addEventListener("mouseover", function () {
  div.style.backgroundColor = "yellow";
});
div.addEventListener("mouseout", function () {
  div.style.backgroundColor = "white";
});
```

Słuchacze zdarzeń

Obsługa wielu zdarzeń dla jednego elementu:

```
let input = document.getElementById("myInput");
input.addEventListener("focus", function () {
  console.log("Pole zostało zaznaczone.");
});
input.addEventListener("blur", function () {
  console.log("Pole zostało odznaczone.");
});
```

Przekazywanie danych do funkcji obsługi zdarzeń:

```
function handleEvent(event) {
  console.log(
    `Zdarzenie typu: ${event.type} na elemencie o id: ${event.target.id}`
      );
    }
    let link = document.getElementById("myLink");
    link.addEventListener("click", handleEvent);
```

Słuchacze zdarzeń

- ✓ Na ten sam element można dodać wiele słuchaczy zdarzeń dla tego samego lub różnych typów zdarzeń.
- ✓ Aby usunąć nasłuchiwanie zdarzenia, używa się metody removeEventListener, która wymaga tych samych argumentów co addEventListener.
- ✓ Wewnątrz funkcji wywoływanej przez addEventListener, this odnosi się do elementu, który wywołał zdarzenie.
- ✓ Używanie addEventListener jest preferowane w stosunku do atrybutów HTML, takich jak onclick, ponieważ pozwala na dodanie wielu słuchaczy do jednego elementu i lepszą organizację kodu.

Słuchacze zdarzeń

1. Zdarzenia dotyczące myszy (Mouse Events)

- `click`: Wyzwalane, gdy element zostanie kliknięty.
- `dblclick`: Wyzwalane, gdy element zostanie podwójnie kliknięty.
- `mousedown`: Wyzwalane, gdy przycisk myszy zostanie wciśnięty na elemencie.
- `mouseup`: Wyzwalane, gdy przycisk myszy zostanie zwolniony na elemencie.
- `mousemove`: Wyzwalane, gdy kursor myszy porusza się nad elementem.
- `mouseover`: Wyzwalane, gdy kursor myszy najedzie na element.
- `mouseout`: Wyzwalane, gdy kursor myszy opuści element.

Słuchacze zdarzeń

2. Zdarzenia dotyczące klawiatury (Keyboard Events)

- `keydown`: Wyzwalane, gdy klawisz zostaje wciśnięty.
- `keyup`: Wyzwalane, gdy klawisz zostaje zwolniony
- `keypress`: Wyzwalane, gdy klawisz zostaje wciśnięty i zwolniony (znak klawisza)

3. Zdarzenia formularzy (Form Events)

- `submit`: Wyzwalane, gdy formularz zostanie wysłany.
- `change`: Wyzwalane, gdy wartość elementu formularza zostanie zmieniona.
- `input`: Wyzwalane, gdy wartość pola formularza jest zmieniana.
- `focus`: Wyzwalane, gdy element otrzymuje fokus.
- `blur`: Wyzwalane, gdy element traci fokus.

Słuchacze zdarzeń

4. Zdarzenia okna (Window Events)

- `load`: Wyzwalane, gdy strona w pełni się załaduje.
- `resize`: Wyzwalane, gdy rozmiar okna zostanie zmieniony.
- `scroll`: Wyzwalane, gdy użytkownik przewija stronę.
- `unload`: Wyzwalane, gdy użytkownik opuszcza stronę

Słuchacze zdarzeń

Zapobieganie domyślnym zachowaniom:

Metoda preventDefault() anuluje zdarzenie, jeśli można je anulować, co oznacza, że domyślna akcja należąca do zdarzenia nie zostanie wykonana.

Może to być przydatne na przykład, gdy:

- Kliknięcie przycisku "Prześlij" uniemożliwia przesłanie formularza
- Kliknięcie linku uniemożliwia linkowi podążanie za adresem URL.

```
document
  .getElementById("myId")
  .addEventListener("click", function (event) {
    event.preventDefault();
  });
```

Literatura:

- Negrino Tom, Smith Dori, ***Po prostu JavaScriopt i Ajax, wydanie VI***, Helion, Gliwice 2007.
- Lis Marcin, JavaScript, Ćwiczenia praktyczne, wydanie II, Helion, Gliwice 2007.
- http://www.w3schools.com/JS/js_popup.asp
- Beata Pańczyk, wykłady opublikowane na stronie http://www.wykladowcy.wspa.pl/wykladowca/pliki/beatap