Resurse suplimentare
pe care recomand să le parcurgi

Traversarea DOM-ului

Traversarea DOM se referă la navigarea prin arborele de elemente, pornind de la un element și deplasându-ne către elemente înrudite (părinți, copii sau frați). Acest lucru ne permite să accesăm elemente bazate pe relațiile lor structurale, nu doar prin selectori.

Navigarea către elementul părinte

```
const element = document.querySelector('#child')

// Accesarea elementului părinte
const parent = element.parentNode // Include orice tip de nod părinte
// sau
const parentElement = element.parentElement // Include doar noduri de tip Element
```

Diferența între parentNode și parentElement:

- `parentNode` poate returna orice tip de nod (inclusiv document)
- `parentElement` returnează doar noduri de tip Element (null dacă părintele este document)

Navigarea în jos spre elementele copil

```
const parent = document.querySelector('#parent')

// Obținerea tuturor nodurilor copil (inclusiv text și comentarii)
const childNodes = parent.childNodes

// Obținerea doar a elementelor copil (exclude nodurile text și comentariile)
const children = parent.children

// Accesarea primului și ultimului nod copil
const firstChild = parent.firstChild // Poate fi nod text
const lastChild = parent.lastChild // Poate fi nod text

// Accesarea primului și ultimului element copil
const firstElementChild = parent.firstElementChild
const lastElementChild = parent.lastElementChild

// Verificarea dacă un element are copii
if (parent.hasChildNodes()) {
  console.log('Acest element are noduri copil')
}

// Numărarea elementelor copil
console.log(`Număr de elemente copil: ${parent.children.length}`)
```

Navigarea către elementele frate (siblings)

```
const element = document.querySelector('#middle')

// Obținerea nodurilor frate
const prevSibling = element.previousSibling // Poate fi nod text
const nextSibling = element.nextSibling // Poate fi nod text

// Obținerea elementelor frate
const prevElementSibling = element.previousElementSibling
const nextElementSibling = element.nextElementSibling
```

Exemplu de traversare complexă

```
// Să presupunem că avem următoarea structură HTML
// <div id="container">
//    <h2>Titlu</h2>
//    <ul>
//      <li>Element 1</li>
//      <li id="target">Element 2</li>
//      <li>Element 3</li>
//    </ul>
//    <p>Paragraf final</p>
// </div>

// Începem cu elementul li cu id="target"
const targetElement = document.querySelector('#target')

// Navigăm la elementul părinte (ul)
const parentUl = targetElement.parentElement

// Accesăm primul element din listă (primul frate)
const firstLi = parentUl.firstElementChild
console.log(firstLi.textContent) // "Element 1"

// Navigăm la elementul părinte al listei (div#container)
const containerDiv = parentUl.parentElement

// Găsim elementul h2 (primul copil al div-ului)
const heading = containerDiv.firstElementChild
console.log(heading.textContent) // "Titlu"

// Găsim paragraful (ultimul copil al div-ului)
const paragraph = containerDiv.lastElementChild
console.log(paragraph.textContent) // "Paragraf final"
```

Găsirea elementelor relative

Pe lângă traversarea directă a DOM-ului, putem folosi și metode pentru a găsi elemente relative într-un mod mai flexibil.

Metoda closest()

Metoda `closest()` ne permite să căutăm în sus în arborele DOM pentru a găsi primul element care se potrivește unui selector specificat.

```
const item = document.querySelector('li.special')

// Găsește cel mai apropiat element părinte care are clasa 'container'
const container = item.closest('.container')

// Găsește cel mai apropiat element <ul> sau <ol>
const list = item.closest('ul, ol')
```

Această metodă este deosebit de utilă pentru delegarea evenimentelor, când trebuie să găsim un părinte specific.

Metoda matches()

Metoda `matches()` verifică dacă un element se potrivește cu un selector CSS specificat.

```
const element = document.querySelector('#item')

if (element.matches('.active')) {
  console.log("Elementul are clasa 'active'")
}

if (element.matches('li.special')) {
  console.log("Elementul este un <li> cu clasa 'special'")
}
```

Metoda contains()

Metoda `contains()` verifică dacă un nod conține un alt nod ca descendent.

```
const container = document.querySelector('#container')
const button = document.querySelector('#myButton')

if (container.contains(button)) {
  console.log('Containerul conține butonul')
}
```

Tehnici avansate pentru modificarea structurii DOM

După ce știm cum să traversăm DOM-ul, putem folosi aceste cunoștințe pentru a modifica structura paginii în moduri mai sofisticate.

Mutarea elementelor în DOM

Putem folosi metodele DOM pentru a muta elemente existente în DOM.

```
const element = document.querySelector('#moveMe')
const destination = document.querySelector('#destination')

// Mutarea unui element (va fi eliminat din poziția originală)
destination.appendChild(element)
```

Clonarea elementelor

Pentru a duplica elemente din DOM:

```
const original = document.querySelector('#original')

// Clonare simplă (fără copii)
const shallowClone = original.cloneNode(false)

// Clonare profundă (cu toți copiii)
const deepClone = original.cloneNode(true)

// Adăugarea clonei în DOM
document.querySelector('#destination').appendChild(deepClone)
```

Fragmente de document

Fragmentele de document oferă un container virtual pentru manipularea mai multor elemente simultan, îmbunătățind performanța.

```
// Creăm un fragment de document
const fragment = document.createDocumentFragment()

// Adăugăm elemente în fragment
for (let i = 0; i < 100; i++) {
  const li = document.createElement('li')
  li.textContent = `Element ${i}`
  fragment.appendChild(li)
}

// Adăugăm fragmentul în DOM (o singură operație de reflow)
const list = document.querySelector('#myList')
list.appendChild(fragment)
```

Inserarea HTML cu insertAdjacentHTML

Pe lângă manipularea structurată a DOM-ului, putem insera și HTML direct într-o poziție specifică:

```
const element = document.querySelector('#target')

// Inserează HTML înainte de elementul țintă (în exterior)
element.insertAdjacentHTML('beforebegin', '<p>Înainte de element</p>')

// Inserează HTML la începutul elementului țintă (în interior)
element.insertAdjacentHTML('afterbegin', '<p>Primul copil</p>')

// Inserează HTML la sfârșitul elementului țintă (în interior)
element.insertAdjacentHTML('beforeend', '<p>Ultimul copil</p>')

// Inserează HTML după elementul țintă (în exterior)
element.insertAdjacentHTML('afterend', '<p>După element</p>')
```

Traversarea DOM-ului cu comoditate: noduri Text și Comment

Când traversăm DOM-ul, deseori întâlnim și noduri Text și Comment care pot complica procesul:

```
const div = document.querySelector('#myDiv')

// Ignorarea nodurilor Text și Comment când parcurgem copiii
Array.from(div.childNodes)
  .filter((node) => node.nodeType === Node.ELEMENT_NODE)
  .forEach((element) => {
    console.log(element.tagName)
  })

// Alternativ, putem folosi children care include doar elemente
Array.from(div.children).forEach((element) => {
  console.log(element.tagName)
})
```

Constante pentru tipurile de noduri

```
// Tipuri de noduri
console.log(Node.ELEMENT_NODE) // 1
console.log(Node.TEXT_NODE) // 3
console.log(Node.COMMENT_NODE) // 8
console.log(Node.DOCUMENT_NODE) // 9
```

Găsirea poziției și dimensiunii elementelor

Pentru a modifica DOM-ul eficient, uneori trebuie să știm unde sunt elementele în pagină.

Metoda getBoundingClientRect()

```
const element = document.querySelector('#target')
const rect = element.getBoundingClientRect()

console.log(`Lățime: ${rect.width}px`)
console.log(`Înălțime: ${rect.height}px`)
console.log(`Distanța de la stânga viewport-ului: ${rect.left}px`)
console.log(`Distanța de la vârful viewport-ului: ${rect.top}px`)
console.log(`Distanța de la dreapta viewport-ului: ${rect.right}px`)
console.log(`Distanța de la baza viewport-ului: ${rect.bottom}px`)
```

Poziții relative la document (nu doar la viewport)

```
const element = document.querySelector('#target')
const rect = element.getBoundingClientRect()

// Adăugăm scroll-ul pentru a obține poziția absolută în document
const absoluteTop = rect.top + window.scrollY
const absoluteLeft = rect.left + window.scrollX

console.log(`Poziția absolută de sus: ${absoluteTop}px`)
console.log(`Poziția absolută din stânga: ${absoluteLeft}px`)
```

Optimizări de performanță pentru modificarea structurii DOM

Modificarea DOM-ului în afara fluxului normal

```
// Eliminarea temporară din DOM
const element = document.querySelector('#complex')
const parent = element.parentNode
const nextSibling = element.nextSibling

// Eliminăm elementul din DOM
parent.removeChild(element)

// Facem multiple modificări
for (let i = 0; i < 100; i++) {
  const div = document.createElement('div')
  div.textContent = `Div ${i}`
  element.appendChild(div)
}

// Reintroducem elementul în DOM
parent.insertBefore(element, nextSibling)
```

Folosirea proprietății display pentru a ascunde elementele în timpul modificărilor

```
const element = document.querySelector('#container')

// Ascundem elementul
element.style.display = 'none'

// Facem multiple modificări
for (let i = 0; i < 100; i++) {
  const paragraph = document.createElement('p')
  paragraph.textContent = `Paragraph ${i}`
  element.appendChild(paragraph)
}

// Afișăm elementul din nou
element.style.display = 'block'
```

Tehnici pentru verificarea și manipularea structurii DOM

Verificarea dacă un element conține un alt element

```
function isDescendant(parent, child) {
  let node = child.parentNode
  while (node !== null) {
    if (node === parent) {
      return true
    }
    node = node.parentNode
  }
  return false
}

// Utilizare
const container = document.querySelector('#container')
const button = document.querySelector('#button')
console.log(isDescendant(container, button))
```

Obținerea tuturor descendenților unui element

```
function getAllDescendants(element) {
  const descendants = []

  function traverse(node) {
    for (let child of node.children) {
      descendants.push(child)
      traverse(child)
    }
  }

  traverse(element)
  return descendants
}

// Utilizare
const allChildren = getAllDescendants(document.querySelector('#container'))
console.log(`Număr total de descendenți: ${allChildren.length}`)
```