Chap39 DOM

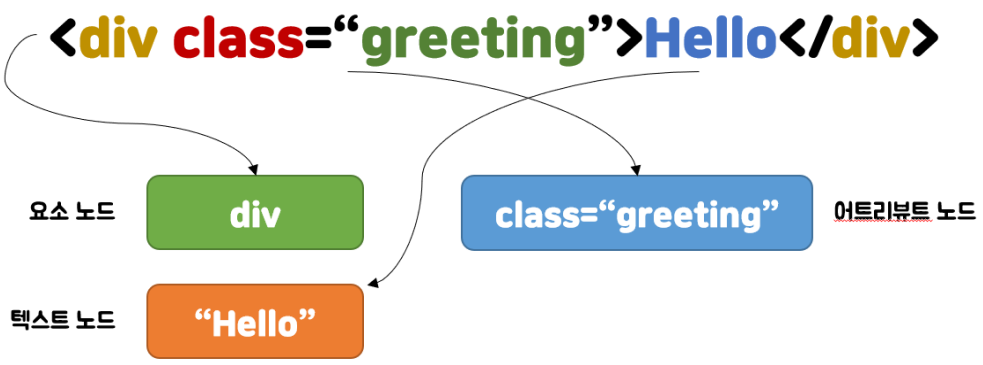
HTML문서 파싱 → DOM 생성

DOM ?

HTML 문서의 계층적 구조와 정보를 표현하며 이를 제어할 수 있는 API, 즉 프로퍼티와 메서드를 제공하는 트리 자료구조다.

▼ 노드

▼ HTML 요소와 노드 객체



HTML 요소 : HTML 문서를 구성하는 개별적인 요소. 렌더링 엔진에 의해 파싱되어 DOM을 구성하는 노드 객체로 변환. HTML의 어트리뷰트는 어트리뷰트 노드로, HTML 텍스트 콘텐츠는 텍스트 노드로 변환된다.

HTML 요소 간에는 중첩 관계에 의해 계층적인 부자 관계가 형성된다. 따라서 HTML문서의 구성 요소인 HTML 요소를 객체화한 모든 노드 객체들은 트리 자료구조로 구성한다.

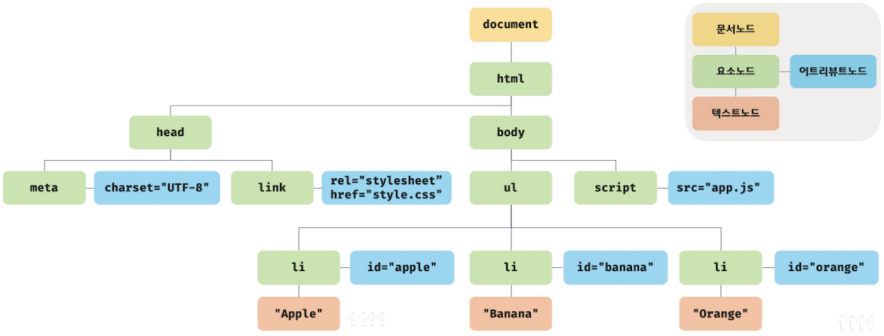
▼ 트리 자료구조

노드들의 계층 구조로 이뤄진 것

노드 객체들로 구성된 트리 자료구조를 DOM이라고 한다.

▼ 노드 객체의 타입

HTML문서 파싱 → DOM 생성



DOM

생성된 DOM은 위와 같은 형태다. DOM은 노드 객체의 계층적인 구조로 구성되고 노드 객체는 종류가 있고 상속 구조를 갖는다. 중요한 노드 타입에 대해 알아보겠다.

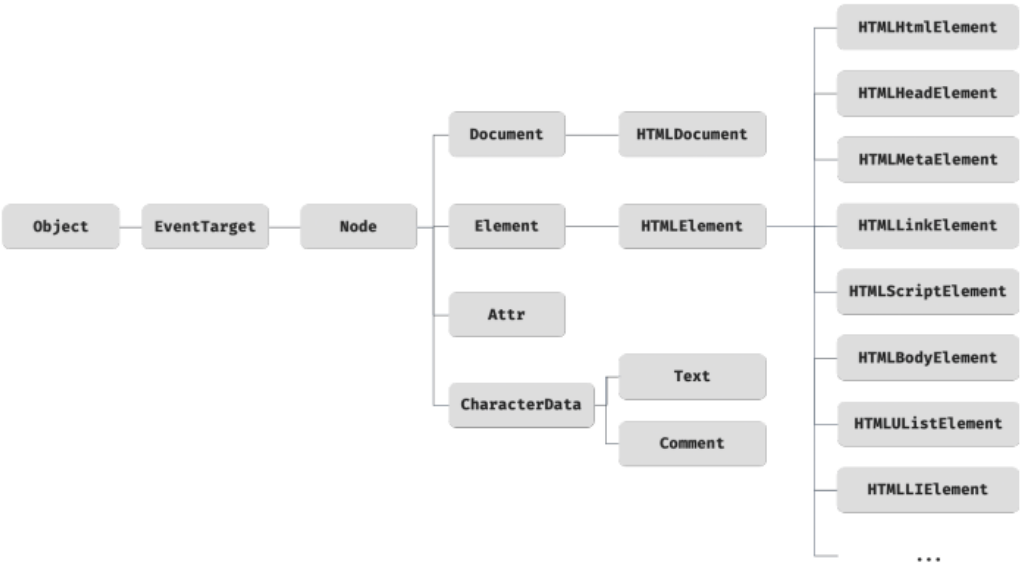
- 문서노드
  - DOM 트리의 최상위에 존재하는 루트노드로서 document 객체를 가리킨다.
  - window.document 또는 documet로 참조할 수 있다.
  - 진입점 역할 담당. 즉, 요소,어트리뷰트, 텍스트 노드에 접근하려면 문서노드를 통해야 한다.
- 요소 노드
  - HTML 요소를 가리키는 객체
  - HTML 요소 간의 중첩에 의해 부자 관계를 가지며, 이 부자 관계를 통해 정보를 구조화한다.
  - 문서의 구조를 표현한다.
- 어트리뷰트 노드
  - HTML 요소의 어트리뷰트를 가리키는 객체
  - 부모 노드가 없다. 다라서 요소 노드의 형재 노드는 아니다.
  - 따라서 어트리뷰트 노드에 접근하여 어트리뷰트를 참조하거나 변경하려면 먼저 요소 노드에 접근해야 한다.
- 텍스트 노드
  - HTML 요소의 텍스트를 가리키는 객체

- 요소노드의 자식 노드이면서 자식노드를 가질 수 없는 리프노드이다.
- 따라서 텍스트노드에 접근하려면 부도노드인 요소 노드에 접근해야 한다.
- DOM의 최종단

▼ 노드 객체의 상속 구조

DOM → 계층적 구조와 정보 표현, 이를 제어할 수 있는 API를 제공하는 트리 자료구조

따라서 DOM을 구성하는 노드 객체는 DOM API 사용 가능. 노드객체는 자신의 부모, 형제, 자식 탐색하여 자신의 어트리뷰트와 텍스트 조작 가능. → 상속 구조 갖음



▼ 요소 노드 취득

HTML 구조나 내용 또는 스타일 등을 동적으로 조작하려면?

→ 요소 노드를 취득해야한다.

▼ id를 이용한 요소 노드 취득

`Document.prototype.getElementById` 메서드 이용하여 인수로 전달한 id 어트리뷰트 값을 갖는 **하나의 요소 노드를 탐색하여 반환**한다.

```
<!DOCTYPE html>
<html>
  <body>
    <ul>
      <li id="apple">Apple</li>
      <li id="banana">Banana</li>
```

```
      <li id="orange">Orange</li> <!--글씨 색깔 오렌지 -->
      <li id="orange">Orange2</li>
    </ul>
    <script>
      const $elem = document.getElementById('orange');
      $elem.style.color = 'orange';
      let apple = '사과';
      //동일한 id가 있어도 에러가 나지 않고 첫번째 요소 노드만 반환
      console.log(orange); // orange 노드 객체
      //id값과 동일한 이름의 전역 변수가 이미 선언되어 있으면 이 전
      console.log(apple); // '사과'
      console.log(banana); // banana 노드 객체

    </script>
  </body>
</html>
```

- 중복된 id값을 갖는 HTML 요소가 여러 개 존재하더라도 에러가 발생하지 않고 첫번째 요소 노드만 반환한다.
- 인수로 전달된 id값을 갖는 HTML 요소가 존재하지 않는 경우 null 반환
- HTML요소에 id 어트리뷰트를 부여하면 id값과 동일한 이름의 전역 변수가 암묵적으로 선언되고 해당 노드 객체가 할당되는 부수 효과가 있다.
- 단, id값과 동일한 이름의 전역 변수가 이미 선언되어 있으면 이 전역 벼수에 노드 객체가 재할당 되지 않는다.

▼ 태그 이름을 이용한 요소 노드 취득

`Document.prototype/Element.prototype.getElementsByTagName` 메서드를 이용하여 인수로 전달한 태그 이름을 갖는 **모든 요소 노드를 탐색하여 반환한**다.

- 일치하는 모든 요소를 갖고온다.
- 따라서 여러개의 요소 노드 객체를 갖는 DOM 컬렉션 객체를 유사 배역 객체이면서 이터러블로 반환한다.
- 모든 요소 노드르 취득하려면 인수로 `*` 을 전달한다.
- 만약 인수로 전달된 태그 이름을 갖는 요소가 존재하지 않는 경우 빈 HTMLCollection 객체를 반환한다.

▼ class를 이용한 요소 노드 취득

`Document.prototype/Element.prototype.getElmentsByClassName` 메서드를 이용하여 인수로 전달한 class 어트리뷰트 값을 갖는 **모든 요소 노드들을 탐색하여 반환**한다.

만약 인수로 전달된 class값을 갖는 요소가 존재하지 않는 경우 빈 HTMLCollection 객체를 반환한다.

▼ css 선택자를 이용한 요소 노드 취득

CSS 선택자? 스타일을 적용하고자 하는 HTML 요소를 특정할 때 사용하는 문법

1. `Document.prototype/Element.prototype.querySelector` 메서드를 이용하여 인수로 전달한 CSS 선택자를 만족시키는 **하나의 요소를 탐색하여 반환**한다.
    - 인수로 전달한 CSS 선택자를 만족시키는 요소 노드가 여러 개인 경우 첫 번째 요소 노드만 반환한다.
    - 인수로 전달한 CSS 선택자를 만족시키는 요소 노드가 존재하지 않는 경우 null을 반환한다.
    - 인수로 전달한 CSS 선택자가 문법에 맞지 않는 경우 DOMException 에러가 발생한다.

2. `Document.prototype/Element.prototype.querySelectorAll` 메서드를 이용하여 인수로 전달한 CSS 선택자를 만족시키는 모든 요소 노드를 탐색하여 반환할 수 있다.
    - 인수로 전달한 CSS 선택자를 만족시키는 요소가 존재하지 않는 경우 빈 NodeList 객체를 반환한다.
    - 인수로 전달한 CSS 선택자가 문법에 맞지 않는 경우 DOMException 에러가 발생한다.
    - HTML 문서의 모든 요소 노드를 취득하려면 전체 선택자 `*` 를 인수로 전달한다.

id 어트리뷰트가 있는 요소 노드를 취득하는 경우는 getElementById 사용하고 그 외에만 querySelector 사용 (querySelector가 더 느림)

▼ 특정 요소 노드를 취득할 수 있는지 확인

`Element.prototype.matches` 메서드를 이용하면 인수로 전달한 CSS 선택자를 통해 특정 요소 노드를 취득할 수 있는지 확인할 수 있다.

- 이벤트 위임을 사용할 때 유용

▼ HTMLCollection과 NodeList

DOM 컬렉션 객체인 HTMLCollection, NodeList는 DOM API가 여러 개의 결과 값을 반환하기 위한 DOM컬렉션 객체다.모두 유사배열객체이면서 이터러블이다. 따라서 for...of문으로 순회할 수 있으며 스프레드 문법을 사용하여 간단히 배열로 변환할 수 있다.

특징

- 노드 객체의 상태 변화를 실시간으로 반영하는 살아있는 객체이다.
- HTMLColletcion은 언제나 live 객체로 동작한다.
- NodeList는 대부분의 경우 노드 객체의 상태 변화를 실시간으로 반영하지 않고 과거의 정적 상태를 유지하는 non-live객체로 동작하지만 경우에 따라 live 객체로 동작할 때가 있다.

▼ 노드 탐색

노드 탐색 프로퍼티는 모두 접근자 프로퍼티이다

getter만 있는 읽기 전용 접근자 프로퍼티

▼ 공백 텍스트 노드

공백 문자(탭,줄바꿈 등)는 공백 텍스트 노드를 생성한다.

노드를 탐색할 때 공백 문자에 의해 생성된 공백 텍스트 노드 주의해야한다.

▼ 자식 노드 탐색

- `Node.prototype.childNodes` : 텍스트 노드를 포함한 NodeList 객체 반환
- `Element.prototype.children` : 텍스트 노드 제외한 HTMLCollection 객체 반환
- `Node.prototype.firstChild` : 첫 번째 자식 노드 반환, 텍스트 노드 일수도
- `Node.prototype.lastChild` : 마지막 자식 노드 반환, 텍스트 노드 일수도
- `Element.prototype.firstElementChild` : 첫 번째 요소 자식 노드 반환
- `Element.prototype.lastElementChild` : 마지막 요소 자식 노드 반환

▼ 자식 노드 존재 확인

`Node.prototype.hasChildNodes` 메서드 사용

- 자식 노드가 존재하면 true, 존재하지 않으면 false를 반환한다.
- 자식 노드 중에 텍스트노드가 아닌 요소 노드가 존재하는지 확인하려면 hasChildNodes 메서드 대신 children.length 또는 Element 인터페이스의 childElementCount 프로퍼티를 사용한다.

▼ 요소 노드의 텍스트 노드 탐색

요소노드의 텍스트 노드는 요소노드의 자식노드!!

따라서 `firstChild` 프로퍼티로 접근할 수 있다.

`firstChild` 프로퍼티

- 첫번째 자식 노드를 반환한다.
- 반환한 노드는 텍스트 노드이거나 요소 노드다.

▼ 부모 노드 탐색

`Node.prototype.parentNode` 프로퍼티를 사용하여 탐색한다.

▼ 형제 노드 탐색

- `Node.prototype.previousSibling` : 텍스트 노드 포함 이전 형제 노드
- `Node.prototype.nextSibling` : 텍스트 노드 포함 다음 형제 노드
- `Node.prototype.previousElementSibling` : 이전 형제 요소 노드
- `Node.prototype.nextElementSibling` : 다음 형제 요소 노드

▼ 노드 정보 취득

▼ 요소 노드의 텍스트 조작

▼ nodeValue

`Node.prototype.nodeValue` 프로퍼티는 setter,getter 둘 다 존재하는 접근자 프로퍼티 → 값 갱신이 가능

- Node.prototype.nodeValue는 노드 객체의 값을 반환
- **노드 객체의 값 = 텍스트 노드의 텍스트**
- **즉, 텍스트 노드가 아니면 null 반환**

▼ textContent

`Node.prototype.textContent` 프로퍼티는 setter와 getter 모두 존재하는 접근자 프로퍼티로서 요소 노드의 텍스트와 모든 자손 노드의 텍스트를 모두 취득하거나 변경

한다.

textContent와 비슷한 `innerText`가 있는데 이는 사용하지 않는게 좋다. 왜냐하면

- innerText 프로퍼티는 CSS에 순종적이다. 예를 들어, innerText 프로퍼티는 CSS에 의해 비표시로 지정된 요소 노드의 텍스트를 반환하지 않는다.
- innerText 프로퍼티는 CSS 를 고려해야 하므로 textContent 프로퍼티보다 느리다.

▼ DOM 조작

DOM 조작 : 새로운 노드를 생성하여 DOM에 추가하거나 기존 노드를 삭제 또는 교체하는 것.

DOM을 조작하면 **리플로우와 리페인트가 발생되어 성능에 영향을 준다.** 따라서 성능 최적화를 위해 주의해서 다루어야한다.

▼ innerHTML

`Element.prototype.innerHTML` 프로퍼티

- setter와 getter 모두 존재하는 접근자 프로퍼티로서 요소 노드의 HTML 마크업을 취득하거나 변경한다.
- innerHTML프로퍼티를 참조하면 요소 노드의 콘텐츠 영역 내에 포함된 모든 HTML 마크업을 문자열로 반환한다.
- innerHTML프로퍼티에 문자열을 할당하면 요소 노드의 **모든 자식 노드가 제거되고** 할당한 문자열에 포함되어 있는 HTML마크업이 파싱되어 요소 노드의 자식노드로 DOM에 반영된다.
    - 모든 자식 노드가 제거되기 때문에 변경되지 않은 부분도 다시 새롭게 자식 노드를 생성하여 DOM에 반영된다. 즉 효율적이지 않다.
- 입력받은 데이터를 그대로 innerHTML 프로퍼티에 할당하는 것은 **크로스 사이트 스크립팅 공격에 취약하므로 위험하다.**

⇒ 복잡하지 않은 요소를 새롭게 추가할 때는 유용하지만 기존 요소를 제거하지 않으면서 위치를 지정해 새로운 요소를 삽입해야 할 때는 사용하지 않는 것이 좋다.

▼ insertAdjacentHTML메서드

`Element.prototype.insertAdjacentHTML` 메서드

- 기존 요소를 제거지 않으면서 위치를 지정해 새로운 요소를 삽입한다.
- 두 번째 인수로 전달한 HTML마크업 문자열을 파싱하고 그 결과로 생성된 노드를 첫 번째 인수로 전달한 위치에 삽입하여 DOM에 반영
  - 첫 번째 인수 - `beforebegin`, `afterbegin`, `beforeend`, `afterend`
- innerHTML 프로퍼티보다는 빠르고 효율적이지만 크로스 사이트 스크립팅 공격에 취약하다는 점은 동일

▼ 크로스 사이트 스크립팅 공격

공격자가 상대방의 브라우저에 스크립트가 실행되도록 해 사용자의 세션을 가로채거나, 웹사이트를 변조하거나, 악의적 콘텐츠를 삽입하거나, 피싱 공격을 진행하는 것

▼ 노드 생성과 추가

DOM은 노드를 직접 생성/삽입/삭제/치환하는 메서드를 제공한다.

아래는 단 하나의 요소 노드를 생성하여 DOM에 추가하는 방법.따라서 DOM은 한 번 변경. 리플로우 리페인트 실행된다.

- 요소 노드 생성 : `Docment.prototype.createElement(tagName)`
- 텍스트 노드 생성 : `Document.prototype.createTextNode(text)`
- 텍스트 노드를 요소 노드의 자식 노드로 추가(마지막 노드로 추가) : `Node.prototype.appendChild(childNode)`
  - 요소 노드에 자식 노드가 하나도 없는 경우에는 textContent 프로퍼티를 사용하는 편이 더욱 간편하다. 단, textContent는 모든 자식 노드가 제거되고 할당한 문자열이 텍스트로 추가되므로 주의해야한다.

▼ 복수의 노드 생성과 추가

```
<!DOCTYPE html>
<html>
  <body>
    <ul id='fruits'></ul>
    <script>
      const $fruits=document.getElementById('fruits');
      ['Apple','Banana','Orange'].forEach(text=>{
        const $li=document.createElement('li');
        const textNode=documnet.createTextNode(text);
        $li.appendChild(textNode);
       $fruits.appendChild($li);
```

```
      });
    </script>
  </body>
</html>
```

위 예제는 3개의 요소노드를 생성하요 DOM에 3번 추가하므로 DOM이 3번 변경된다. 따라서 리플로우, 리페인트가 3번 실행된다.

→ 비효율적

그렇다면 컨테이너 요소사용하면?

```
<!DOCTYPE html>
<html>
  <body>
    <ul id='fruits'></ul>
    <script>
      const $fruits=document.getElementById('fruits');
      const $contatiner=docment.createElement('div');
      ['Apple','Banana','Orange'].forEach(text=>{
        const $li=document.createElement('li');
        const textNode=documnet.createTextNode(text);
        $li.appendChild(textNode);
        $container.appendChild($li);
      });
      $fruits.appendChild($container);
    </script>
  </body>
</html>
```

DOM을 한번 변경하기는 하지만 불필요한 컨테이너 요소(div)가 DOM에 추가되는 부작용이 있다.

→바람직하지 않음

따라서 `DocumentFragment` 노드를 통해 해결할 수 있다.

- 기존 DOM고 별도로 존재하므로 DOM에 어떠한 변경도 발생하지 않는다.
- DOM 변경이 발생하는 것은 한번 뿐이며 리플로우와 리페인트오 한번만 실행된다.

- 따라서 **여러 개의 요소 노드를 DOM에 추가하는 경우** `DocumentFragment` **노드를 사용하는 것이 효율적**

▼ 노드 삽입

- 마지막 노드로 추가 : `Node.prototype.appendChild` 메서드
- 지정한 위치에 노드 삽입 : `Ndoe.prototype.insertBefore(newNode,childNode)` 메서드

▼ 노드 이동

DOM에 이미 존재하는 노드를 appendChild,insertBefore 메서드로 DOM에 추가하면 원래 있던 위치의 노드는 제거되고 새로운 위치로 노드를 추가함

▼ 노드 복사

`Node.prototype.cloneNode(true|false)` 메서드

- true : 깊은 복사하여 모든 자손 노드가 포함된 사본을 생성
- false : 얕은 복사하여 노드 자신만의 사본을 생성

▼ 노드 교체

`Node.prototype.replaceChild(newChild,oldChild)`

▼ 노드 삭제

`Node.prototype.removeChild(child)` 메서드

▼ 어트리뷰트

▼ 어트리뷰트 노드와 attributes 프로퍼티

HTML문서가 파싱될 때 HTML 요소의 어트리뷰트는 어트리뷰트 노드로 변환되어 요소 노드와 연결된다. 이때 **HTML 어트리뷰트당 하나의 어트리뷰트 노드가 생성**된다. 예를들어 input태그의 어트리뷰트가 3개이면 3개의 어트리뷰트 노드가 생성된다.

따라서 이러한 어트리뷰트 노드는 `Element.prototype.attributes` 프로퍼티로 취득할 수 있다. (getter만 존재한 읽기 전용 접근자 프로퍼티)

▼ HTML 어트리뷰트 조작

`Element.prototype.getAttribute/setAttribute` 메서드

- attributes 프로퍼티의 setter이 없는 단점 해결
- attributes프로퍼티를 통하지 않고 요소 노드에서 메서드를 통해 직접 HTML 어트리뷰트 값 취득하거나 변경할 수 있어서 편리

`Element.prototype.hasAttribute(attributeName)` 메서드 : 특정 어트리뷰트 존재 확인할 때 사용

`Element.prototype.removeAttribute(attributeName)` 메서드 : 특정 어트리뷰트 삭제

▼ HTML 어트리뷰트 vs DOM 프로퍼티

요소 노드 객체에는 HTML 어트리뷰트에 대응하는 프로퍼티가 존재

이러한 DOM 프로퍼티들은 HTML 어트리뷰트 값을 초기값으로 갖고 있음

DOM 프로퍼티는 setter,getter 둘다 있는 접근자 프로퍼티

▼ HTML 어트리뷰트

HTML 요소의 초기 상태를 지정하고 이는 변하지 않는다

▼ DOM 프로퍼티

요소 노드의 최신 상태를 관리

▼ HTML어트리뷰토와 DOM프로퍼티의 대응관계

대부분의 HTML어트리뷰트는 HTML어트리뷰트 이름과 동일한 DOM프로퍼티와 1:1로 대응한다.

단, 그렇지 않은 경우가 있다.

- id 어트리뷰트와 id프로퍼티는 1:1 대응하며 동일한 값으로 연동한다.
- input 요소의 value 어트리뷰트는 value프로퍼티와 1:1 대응한다. 하지만 value어트리뷰트는 초기상태를, value 프로퍼티는 최신 상태를 갖는다.
- class 어트리뷰트는 className,classList프로퍼티와 대응한다.
- for 어트리뷰트는 htmlFor프로퍼티와 1대1 대응한다.
- td요소의 colspan어트리뷰트는 대응하는 프로퍼티가 존재하지 않는다.
- textContent프로퍼티는 대응하는 어트리뷰트가 존재하지 않는다.
- 어트리뷰트 이름은 대소문자를 구별하지 않지만 대응하느 프로퍼티 키는 카멜 케이스르 따른다.

▼ DOM프로퍼티의 타입

getAttribute 메서드로 취득한 어트리뷰트의 값은 항상 문자열이다.

DOM 프로퍼티로 취득한 최신 값은 문자열이 아닐 수도 있다.

▼ data 어트리뷰트와 dataset 프로퍼티

data 어트리뷰트와 dataset 프로퍼티를 사용하면 HTML 요소에 정의한 사용자 정의 어트리뷰트와 자바스크립트 간에 데이터를 교환할 수 있다.

data 어트리뷰트

- data-user-id, data-role과 같이 data- 접두사 다음에 임의의 이름을 붙여 사용한다.
- data 어트리뷰트 값은 `HTMLElement.dataset` 프로퍼티로 취득 가능.
- HTMLElement.dataset 프로퍼티는 모든 data 어트리뷰트를 담은 DOMStringMap 객체 반환
- DOMStringMap 객체는 사용자정의 어트리뷰트 이름을 카멜케이스로 변환한 프로퍼티를 갖는다

▼ 스타일

▼ 인라인스타일조작

▼ 클래스 조작

▼ 요소에 적용되어 있는 CSS 스타일 참조

`window.getComputedStyle(element[,pseudo])` 메서드

- 첫번째 인수로 전달한 요소 노드에 적용되어 있는 평가된 스타일을 CSSStyleDeclaration 개체에 담아 반환한다.
- 두번째 인수로 :after, :before와 같은 의사요소를 지정하는 문자열을 전달할 수 있다.

▼ DOM 표준

HTML와 DOM표준은 W3C과 WHATWG이라는 두 단체가 협력하며 만들었다.

그러나 저점 두단체가 서로다른 결과물을 내놓기 시작했지만 2018년부터 구글, 애플, 마이크로소프트, 모질라로 구성된 WHATWG가 단일 표준을 제공하기로 두 단체가 합의

DOM은 현재 4개의 버전이 있다.