39___ DOM

1. DOM(Document Object Model)이란

-. HTML 문서의 계층적 구조와 정보를 표현

-. 이를 제어할 수 있는 DOM API, 즉 메서드와 프로퍼티 제공

-. 트리 자료구조

2. HTML 요소와 노드 객체

-. HTML 문서는 HTML 요소들의 집합

-. HTML 요소는 중첩 관계, 부자 관계 존재

-. HTML 요소들은 렌더링 엔진에 의해 요소 노드 객체로 변환

3. 노드 객체 타입

-. 총 12개의 노드 타입 존재. 이중 중요한 노드 타입은 4가지

(1) 문서 노드(docuent node)

-. 최상위에 존재하는 루트 노드

-. 루트 노드이므로, DOM 트리의 노드에 접근하기 위한 진입점(entry point) 역할

-. HTML 문서 전체를 가리키는 객체

-. 전역 객체 wnodw의 document 프로퍼티에 바인딩되어, window.document나 document로 접근 가능

-. script 태그가 분리되어 있어도, 하나의 window를 공유. 즉, HTML 문서당 document 객체는 유일

(2) 요소 노드(element node)

-. HTML 요소를 가리키는 객체

-. 문서의 구조를 표현

(3) 어트리뷰트 노드(attribute node)

-. 요소 노드와 형제 관계

-. 부모 노드와 연결되어 있지 않음.

그러므로, 어트리뷰트 노드에 접근하기 위해서는 요소 노드에 먼저 접근해야 함

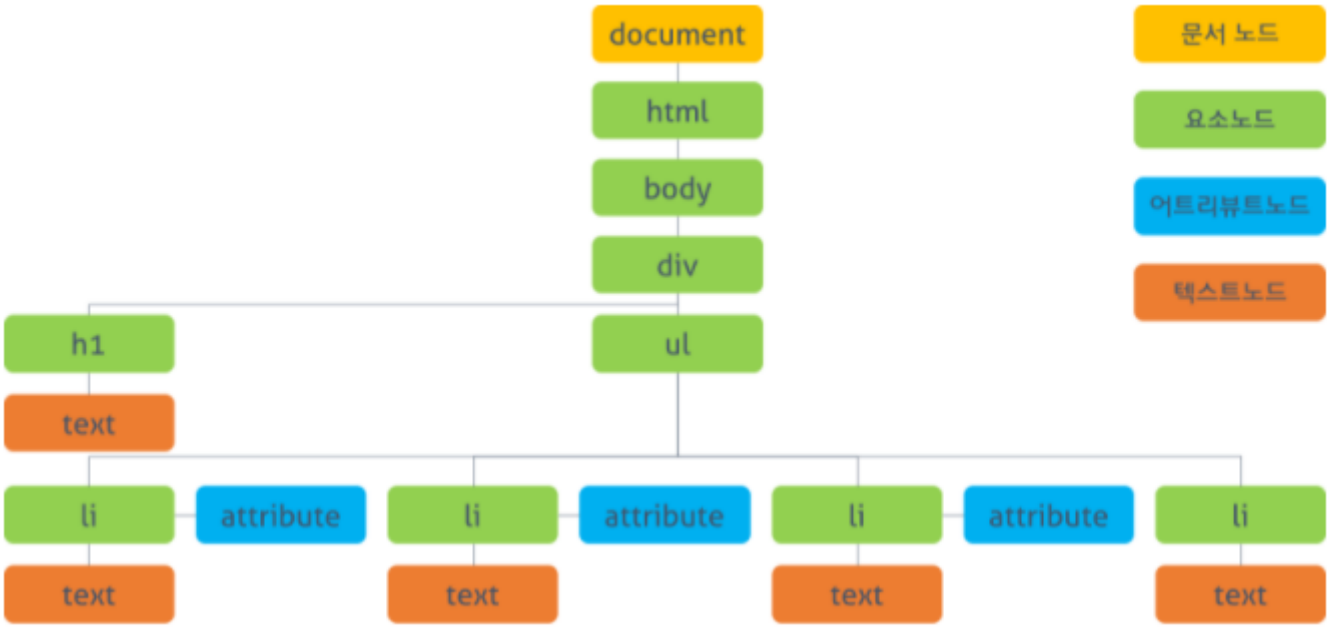
(4) 텍스트 노드

___ 문서의 정보를 표현

___ 요소 노드의 자식 노드이며 리프 노드

___ 텍스트 노드에 접근하기 위해서는 요소 노드에 먼저 접근해야 함

추가적으로 DocumentFragment 노드도 존재.



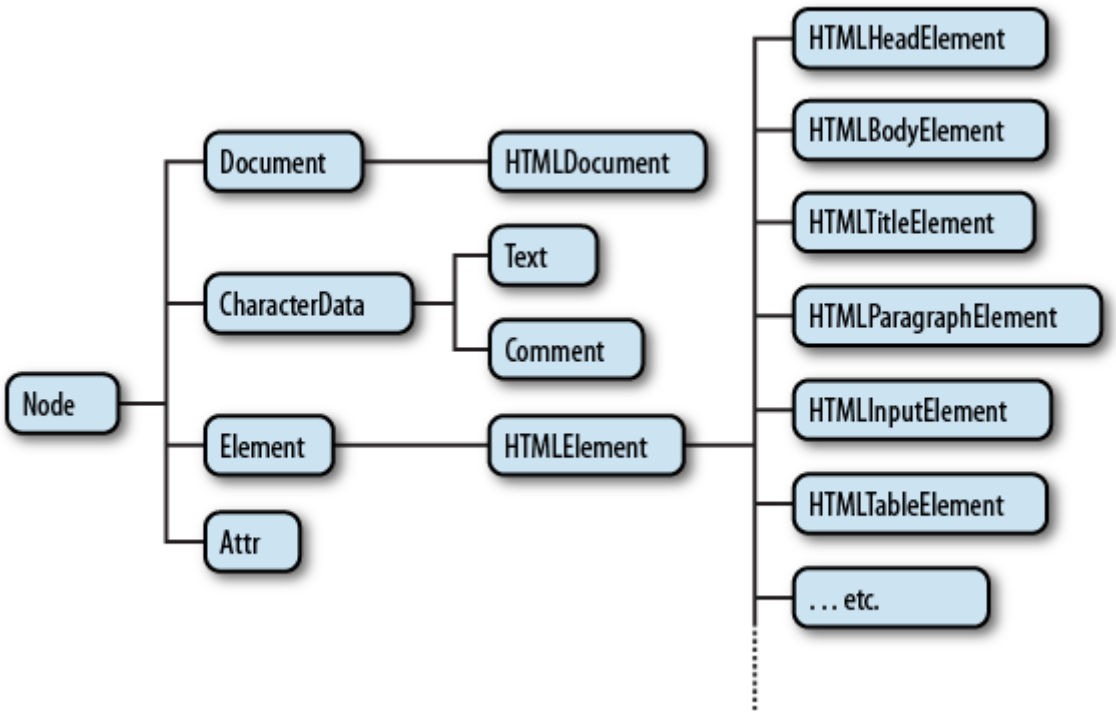
사진출처

4. 노드 객체의 상속 구조

-. 노드 객체는 DOM API를 사용할 수 있다.

-. 노드 객체는 ECMAScript 사양에 정의된 표준 빌트인 객체가 아닌, 브라우저 환경에서 추가적으로 제공하는 호스트 객체

-. 객체이므로 프로토타입에 의한 상속 구조 존재



사진참조

(사진상 잘려진 부분 존재. Node 이전에, Object — EventTarget — Node 순으로 인터페이스 존재)

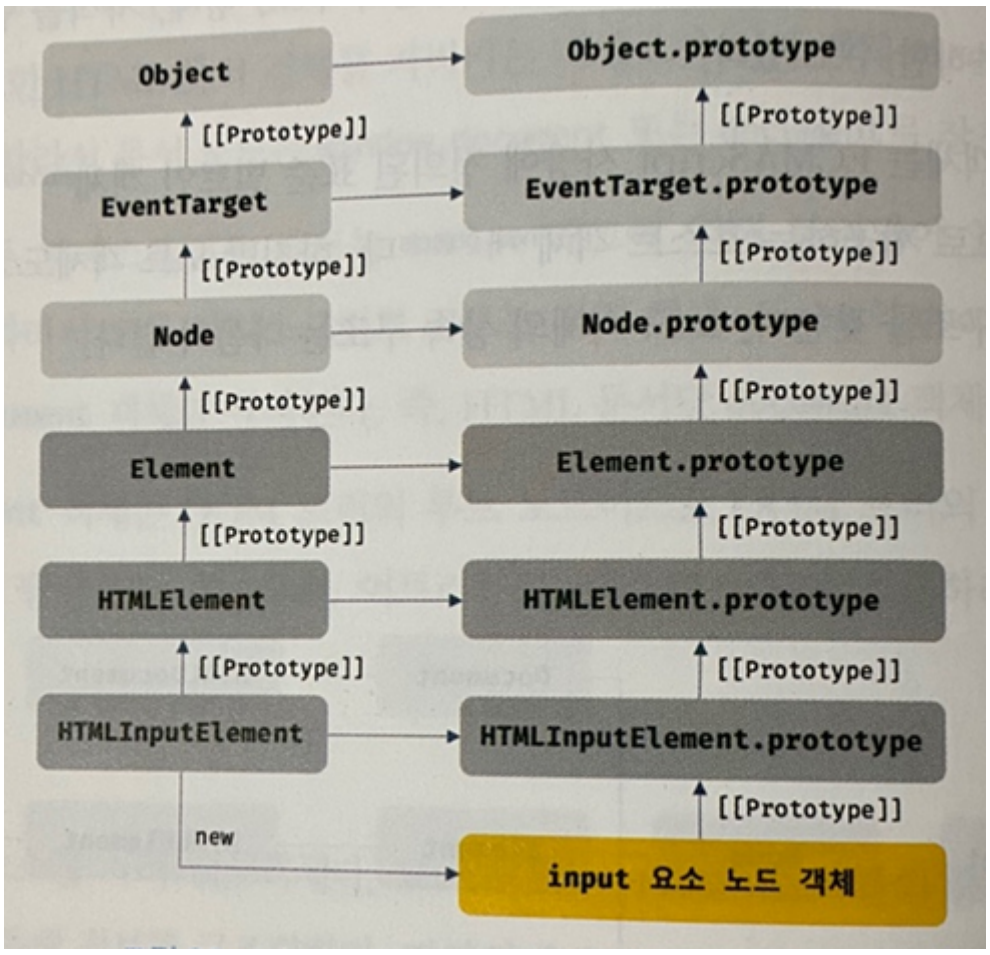
-. 모든 노드 객체는 Object, EventTarget, Node 인터페이스 상속받음

-. 문서 노드는 Document, HTMLDocument 인터페이스 상속받음

-. 요소 노드는 Element 인터페이스 상속받음

-. 어트리뷰트 노드는 Attr 인터페이스 상속받음

-. 텍스트 노드는 CharacterData 인터페이스 상속받음



-. input 요소 노드 객체는 프로토타입 체인에 있는 모든 프로토타입의 프로퍼티나 메서드 상속

-. 배열이 객체인 동시에 배열인 것처럼, input 태그도 여러 특성을 갖는 객체

input 요소 노드 객체의 특성	프로토타입을 제공하는 객체
객체	Object
이벤트를 발생시키는 객체	EventTarget
트리 자료구조의 노드 객체	Node
브라우저가 렌더링할 수 있는 웹 문서의 요소(HTML, XML, SVG)를 표현하는 객체	Element
웹 문서의 요소 중에서 HTML 요소를 표현하는 객체	HTMLElement
HTML 요소 중에서 input 요소를 표현하는 객체	HTMLInputElement

-. 노드 객체에는 모든 노드 객체가 공통적으로 갖는 기능과 노드 타입에 따라 고유하게 갖는 기능이 존재

___ 모든 노드 객체가 공통적으로 갖는 기능

1. 이벤트와 관련된 기능

-. EventTarget 인터페이스가 제공

-. EventTarget.addEventListener, EventTarget.removeEventListener

2. 트리 탐색 기능

-. Node 인터페이스가 제공

-. 트리 탐색 기능(Node.parentNode, Node.childNodes, ...)

-. 노드 정보 제공 기능(Node.nodeType, Node.nodeName)

___ 요소 노드 객체가 갖는 기능

-. HTMLElement 인터페이스가 제공, input 요소 노드 객체와 div 요소 노드 객체는 style 프로퍼티 존재.

즉, 노드 객체는 공통된 기능일수록 프로토타입 체인의 상위에, 개별적인 기능일수록 프로토타입 체인의 하위에 프로토타입 체인을 구축.

프로퍼티와 메서드는 상속 구조로 제공

5. 요소 노드 취득

HTML 요소를 조작하기 위해서는 요소 노드를 취득해야 함

1. id를 이용한 요소 노드 취득

____ id 어트리뷰트 값을 갖는 하나의 노드 반환

____ getElementById는 Document.prototype의 프로퍼티

____ id 값은 문서 내에 유일한 값(여러 개가 존재해도 에러 발생은 안함)

____ class처럼 공백 문자 구분 불가

____ 요소가 존재하지 않으면 null 반환

```
//html
<ul>
  <li id="banana">Apple</li>
  <li id="banana">Banana</li>
  <li id="banana">Orange</li>
</ul>

// script
const $elem = document.getElementById('banana'); // $elem = <li id="banana">Apple</li>
$elem.style.color = 'red';
```

```
//html
<ul>
  <li id="apple">Apple</li>
  <li id="banana">Banana</li>
  <li id="orange">Orange</li>
</ul>

// script
const $elem = document.getElementById('grape'); // null
$elem.style.color = 'red';
```

____ id 어트리뷰트를 부여하면, id값과 동일한 이름의 전역 변수가 암묵적으로 선언되고 해당 노드 객체 할당

```
// html
<div id="foo"></div>

// script
// id 값과 동일한 이름의 전역 변수가 암묵적으로 선언되고 해당 노드 객체가 할당
console.log(foo === document.getElementById('foo')); // true

// 암묵적 전역으로 생성된 전역 프로퍼티는 삭제되지만, 전역 변수는 삭제되지 않는다.
delete foo;
console.log(foo); // <div id="foo"></div>
```

____ id값과 동일한 전역 변수가 이미 선언되어 잇으면, 이 전역 변수에 노드 객체 재할당되지 않음

```
// html
<div id="foo"></div>

// script
let foo = 1;

delete foo;
console.log(foo); // 1
```

2. 태그 이름을 이용한 요소 노드 취득

____ Document.prototype/Element.prototype.getElementsByTagName 메서드는 태그 이름을 갖는 모든 요소 노드들을 탐색하여 반환

____ 여러 개의 요소 노드 객체를 갖는 DOM 컬렉션 객체인 HTMLCollection 객체 반환

```
//html
<ul>
  <li id="banana">Apple</li>
  <li id="banana">Banana</li>
  <li id="banana">Orange</li>
</ul>

// script
const $elem = document.getElementsByTagName('li'); // [li, li, li]
```

____ HTMLCollection 객체는 유사배열객체이면서 이터러블

____ 문서의 모든 요소 노드를 취득하려면 인수로 * 를 전달.

```
const $all = document.getElementsByTagName('*');
// [html, head, body ....]
```

____ Document.prototype을 통해서 호출하면, DOM 전체에서 요소 노드 탐색하여 반환

____ Element.prototype을 통해서 호출하면, 특정 요소 노드의 자손 노드 중에서 요소 노드를 탐색하여 반환

____ 요소가 존재하지 않는 경우, 빈 HTMLCollection 객체 반환

```
// html
<ul id="fruits">
  <li>Apple</li>
  <li>Banana</li>
  <li>Orange</li>
</ul>
<ul>
  <li>HTML</li>
</ul>

// script
const $lisFromDocument = document.getElementsByTagName('li');
// [li,li,li,li]

const $fruits = document.getElementBtyId('fruits');
const $lisFromFruits = $fruits.getElementsByTagName('li');
// [li,li,li]
```

3. class를 이용한 요소 노드 취득

____ Document.prototype/Element.prototype.getElementsByClassName 메서드는 class 이름을 갖는 모든 요소 노드들을 탐색하여 반환

____ 인수로 전달한 class 값은 공백으로 구분하여, 여러 개의 class 지정 가능

____ HTMLCollection 객체 반환

```
// html
<ul>
  <li class="fruit apple">Apple</li>
  <li class="fruit banana">Banana</li>
  <li class="fruit orange">Orange</li>
</ul>

// script
const $elems = document.getElementsByClassName('fruit');
//[<li>Apple</li>, <li>Banana</li>, <li>Orange</li>]

const $apples = document.getElementsByClassName('fruit apple');
// [<li>Apple</li>
```

____ 요소가 존재하지 않는 경우, 빈 HTMLCollection 객체 반환

4. CSS 선택자를 이용한 요소 노드 취득

스타일을 적용하고자 하는 HTML 요소를 특정할 때 사용

```
// 전체 선택자, 모든 요소 선택
* { ... }

// 태그 선택자, 모든 p 태그 요소 선택
p { ... }

// id 선택자, id가 foo인 요소 모두 선택
#foo { ... }

// class 선택자, class 값이 foo인 요소 모두 선택
.foo { ... }

// 어트리뷰트 선택자, input 요소 중에 type 어트리뷰트 값이 'text'인 요소 모두 선택
input[type=text] { ... }

// 후손 선택자, div 요소 후손 요소 중, p 요소 모두 선택
div p { ... }

// 자식 선택자, div 요소 자식 요소 중, p 요소 모두 선택
div > p { ... }
// 자식은 dpeth가 1, 후손은 depth가 1 이상.

// 인접 형제 선택자, p 요소의 형제 요소 중, p 요소 바로 뒤에 위치한 ul 요소
p + ul { ... }

// 일반 형제 선택자, p 요소 형제 요소 중, p 요소 뒤에 위치한 ul 요소 모두 선택
p ~ ul { ... }

// 가상 클래스 선택자, hover 상태인 a 요소 모두 선택
a:hover { ... }

// 가상 요소 선택자, p 요소 콘텐츠 앞에 위치하는 공간을 선택. 일반적으로 content 프로퍼티와 함께 사용
p::before { ... }
```

Document.prototype/Element.prototype.querySelector 메서드는, 인수로 전달한 CSS 선택자를 만족시키는 하나의 요소 노드 탐색하여 반환

____ 인수로 전달한 CSS 선택자를 만족하는 요소가 여러개인 경우, 가장 첫 번째 요소 노드만 반환

____ 인수로 전달된 CSS 선택자를 만족시키는 요소가 존재하지 않으면, null 반환

____ 인수로 전달한 CSS 선택자가 문법에 맞지 않는 경우, DOMException 에러 발생

Document.prototype/Element.prototype.querySelectorAll 메서드는, 인수로 전달한 CSS 선택자를 만족시키는 여러개의 요소 노드 탐색하여 반환

____ 여러 개의 요소 노드 객체를 갖는 DOM 컬렉션 객체인 NodeList 객체 반환, 유사배열객체이면서 이터러블

____ 요소가 존재하지 않는 경우, 빈 NodeList 반환

____ 인수로 전달한 CSS 선택자가 문법에 맞지 않는 경우, DOMException 에러 발생

```
<ul>
  <li class="apple">Apple</li>
  <li class="banana">Banana</li>
  <li class="orange">Orange</li>
</ul>

const $elems = document.querySelectorAll('ul > li');
const $elem = document.querySelector('.apple');
```

모든 요소를 탐색하고 싶은 경우, * 이용

```
cosnt $all = document.querySelectorAll('*');
// [html, head, body, ...]
```

querySelector와 querySelectorAll은, getElementById, getElementsBy*** 메서드보다 다소 느리다고 알려짐.

하지만, CSS 선택자 문법 덕분에 구체적인 조건과 일관된 방식으로 요소 노드 취득 장점 존재.

그러므로, id 요소를 취득 하는 경우 getElementById를 이용하고, 이외의 경우에는 querySelector와 querySelectorAll을 이용하는 것 추천

6. 특정 요소 노드 취득 가능 여부 확인

```
<ul id="fruits">
  <li class="apple">Apple</li>
  <li class="banana">Banana</li>
  <li class="orange">Orange</li>
</ul>

const $apple = document.querySelector('.apple');

console.log($apple.matches('#fruits > li.apple')); // true
console.log($apple.matches('#fruits > li.banana')); // false
```

7. HTMLCollection과 NodeList

```
<ul>
  <li class="red">Apple</li>
  <li class="red">Banana</li>
  <li class="red">Orange</li>
</ul>

const $elems = document.getElementsByClassName('red');

for(let i = 0; i < $elems.length; i++) {
  $elems[i].className = 'blue';
}

console.log($elems); // $elems = [<li class="red">Banana</li>
// $elems =  [<li>Banana</li>]
```

위 코드 동작에 대한 이해

1. i === 0인 경우,

   red에서 blue로 변경되는 순간, $elems HTMLCollection 객체에서 제거됨

2. i === 1인 경우,

   첫 번째 요소가 제거되었으므로, 세 번째 li요소를 가리키게 되고, red에서 blue로 변경된 뒤, 마찬가지로 $elems HTMLCollection 객체에서 제거됨

3. i === 2인 경우,

   i = 2인 인덱스가 존재하지 않으므로, 반복이 종료됨. 두 번째 요소는 className이 변경되지 않고 종료

이러한 문제를 해결하는 방법은,

1. 역순으로 순회하기

```
for (let i = $elems.length - 1; i >= 0; i--) {
  $elems[i].className = 'blue';
}
```

2. while문으로, HTMLCollection 요소가 남아있지 않을 때 까지 반복

```
let i = 0;
while($elems.length > i) {
  $elems[i].className = 'blue';
}
```

3. HTMLCollection 객체 대신 배열 사용하기

```
[...$elems].forEach(elem => elem.className = 'blue');
```

4. NodeList 객체 사용하기

____ NodeList 객체는 NodeList.prototype의 forEach, item, entries, keys, values 메서드 사용 가능

____ cildNodes 프로퍼티가 반환하는 NodeList 객체는 HTMLCollection 객체와 마찬가지로 live 객체로 동작함.

____ HTMLCollection과 NodeList 객체는 예상과 다르게 동작하므로, DOM 컬렉션을 사용하려면, 배열로 변환하여 사용하는 것을 권장. 추가적으로 배열의 고차 함수도 사용이 가능해진다.

```
const { childNodes } = $elems;

[...childNodes].forEach(childNode => {
  $elems.removeChild(childNode);
});
```