Javascript for Web Browser

정리자: 이태혁 (Lee, Tae Hyuk)

[HTML에서 Javascript를 어떻게 인식할수 있는가?]

1. inline방식

eg) <input type = “button” onclick =”alert(’Hello world’)” value = “Hello World”)

여기서 alert (Hellow world’)는 javascript문법으로 onclick다음에 오는걸 이미 약속되어 있는 것.

onclick은 HTML문법이고 alert는 JS문법이다 이미 약속이 되어 있는 것이다.

(정보, html+제어 JS) → 단순하지만 integarted system으로 따로 제어하는게 쉽지않다.

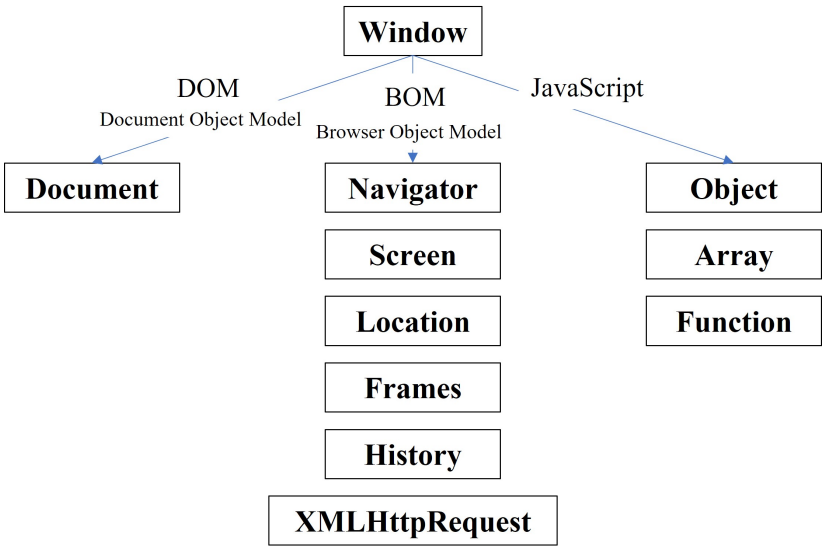
2. script (외부에서 불러오는 방식)

eg) <script src =”main.js”></script> 이런식으로 하게 된다. defer 병렬적으로 돌아간다.

3. html 내부에서 <script> Js code </script> 할수 있다.

[Window 전역객체에 대한 개념]

-전역객체 Window



[Figure Window Object]

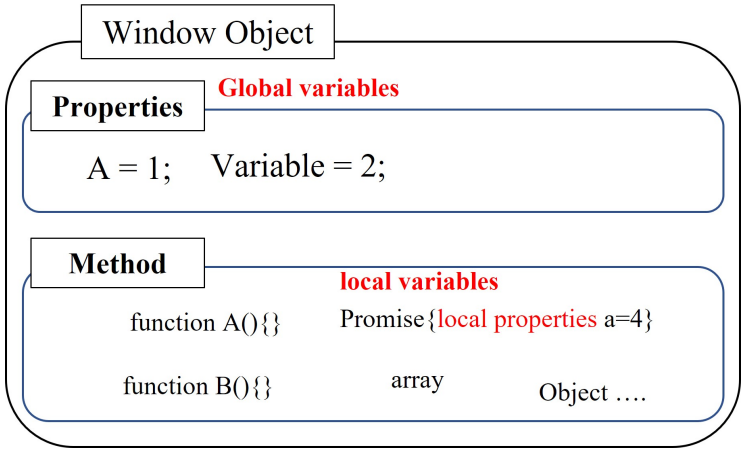
DOM, BOM, JS에 속해있는 어떤 객체든 **모두 window객체에 소속**되어 있다.

→ window객체에 접근하기 위해서는 console을 실행시켜서 window라고 입력하면 window객체에 접근할 수 있다. 그 아래 객체에 접근하고 싶으면 window.document, window.navigator 등을 치면 알수 있다.

이 **Window객체**는 모든 객체들이 소속되어 있는 객체이면서 **'전역 객체'**라고도 불린다.

*window객체가 전역객체라고 불리는 이유는?

-내가 어떤 작업을 해서 function을 만들어도 모두 window객체 안에 소속되게 되어 있다.



즉, 다시말해서 내가 함수를 만들든 짜여져있는 내장 함수를 가지고오든 심지어 a=1; 이라는 할당을 하든, 또는 선언을 하든간에 모두 Window Object내에 있는 method를 사용한 것 과 같다. 그림상에서 Window Object밖으로 나갈수 없다.

ex) a=1; → console.log(a) //// result = 1 나옴

그러나 실제로는 window.a 임 (window 객체의 constructor 또는 method 이기때문에)

→ 여기서 의미상 전역변수 (not local variable)를 만든다는 것은 즉, window의 property를 만든다는 의미와 같다. (아! 완벽히 이해됐어!)

→ 여기서 DOM, BOM, JavaScript 등은 분류만 해 놓았을뿐이지 실제로 저렇게 프로그램상 명명되어있지 않을거고 그냥 다 window객체 내에 있을 것이다.

*여기서 window객체에 소속되어있다 (자식이란 말은 잘못되었다는데 나중에 왜 잘못되었는지 알아보자)

<Dreamcoding -엘리 에서는 JS를 배운 것과 같다>

[BOM - Browser Object Model]

BOM이란?

Web Browser를 제어하기 위해서 Browser가 제공해주는 객체들(Object)을 의미한다.

우리는 이것들을 JS를 통해서 제어한다. 다음 5가지 개념에 대해서 알수 있다.

-사용자와 커뮤니케이션

-Location 객체

-Navigator 객체

-창 제어

▼ 사용자와 커뮤니케이션하기

▼ *alert* (경고창 만들어주기)

(경고창이라고 부른다 - 사용자에게 정보를 제공하거나 디버깅등의 용도로 많이 사용한다.)

alert('Hello world')' → 경고창이 뜬다.

▼ *confirm*

(확인과 취소 버튼을 만들어준다)

```
<body>
    <input type = "button" value="confirm" onclick="func_confirm();"/>
        <script>
            function func_confirm(){
                if(confirm('ok?')){
                    alert('ok');}
                else{
                    alert('cancel');}
            }
            </script>
            </body>
```

▼ *Prompt*

(사용자가 입력한 값을 JS가 얻어낼수 있는 방법)

```
<body>
    <input type = "button" value="confirm" onclick="func_confirm();"/>
        <script>
            function func_prompt(){
            if(prompt('id?')==='egoing'){
                alert('welcome');
            }else {
```

```
    alert('fail');}
}
</script>
</body>
```

▼ Location 객체

-Location객체는 현재 윈도우의 문서가 위치하는 URL을 알아내는 방법.

▼ *현재 윈도우의 URL 알아내기*

```
console.log(location.toString(), location.href);
```

위의 code를 console창에 입력하면 결과가 두 개가 나온다. (둘 다 똑같은 결과가 나온다)

-하지만 href를 쓰는 것을 선호한다.

*참고로 console.log(location)하면 location객체를 보여준다. 그러나 alert(location)을 출력하면 객체를 보여주는 것이 아니라 URL을 alert창에 보여주는데 그 이유는 *alert의 경우 property를 string화 해서 들어가게 하기때문에 문자화 한 결과로서 location.toString()과 같은 결과를 보인다*.

▼ *URL Parsing*

```
console.log(location.protocol, location.host, location.port, location.pathname, location.search, location.hash)
```

location객체는 URL을 의미에 따라서 별도의 프로퍼티로 제공하고 있다.

https://opentutorials.org/course/2136/11854?id=10#bookmark

console.log(location.protocol);* → *http

//Web page 통신규약

console.log(location.host);* →*opentutorials.org* *(domain)

//host란? 서비스를 식별하는 주소를 의미한다. 서버 컴퓨터를 식별하는 주소가 host

console.log(location.port);* → *domain:8080

//port란 것은 domain:하고 나타난 식별 번호를 의미한다. host는 컴퓨터를 식별하는 것이라면 port는 컴퓨터에서 돌아가는 여러 서버소프트웨어들을 식별하는 번호이다.

console.log(location.pathname);* →*domain:8080/course/2136/11854

//pathname은 어느 웹서버에 접속했을때 그 웹서버가 가지고 있는 정보중에 어떤 구체적인 정보를 요청하는 정보를 의미한다. (A leture의 1강의 3부 정보 이런 것)

console.log(location.search);* →*domain:port/pathname?id=10

//이 서비스에 전달된 id값이 10이라는 뜻이다.

console.log(location.hash);* →*domain:port/pathname?search#bookmark

//id뒤에 #하고 나온 것이 hash를 의미한다. (문서 안에 특정한 위치에 북마킹을 할 수 있다)

▼ *URL 변경하기*

location.href = 'http://egoing.net'; (좀더 명시적인 방법)

location = 'http://egoing.net';

현재 문서를 https://egoing.net주소를 할당함으로써 이동시킨다 (location.href-현재 주소)

-JS Application에서 다른곳으로 사용자를 옮겨줘야할때 사용된다.

▼ *URL Reload하기*

현재 보고있는 URL을 reload하고 싶다

location.href = location.href;

location.reload();

▼ Navigator 객체

-Navigator는 JS를 통해서 브라우저의 정보 (버전 등)을 알수 있는 기능을 제공한다

<주로 호환성 문제등을 위해서 사용한다>

브라우저에 맞는 코딩을 할 수 있게 하는 것이 우선적인 목적이다.

우선 ***Cross Browsing***이란 것에 대해서 알아야 한다.

전세계적으로 봤을때 정말 많은 브라우저들이 있다 **Internet Explore, Fire Fox, Chrome, Safari, Opera** 등이 있다. 이런 다양한 브라우저들의 동작 방법은 **W3C(국제 표준화 기구) - HTML, CSS** 그리고 **ECMA(표준화 기구) - JavaScript**를 규정하고 있는 가이드를 따라간다.

결론적으로 브라우저 업체들은 위의 W3C, ECMA(표준화 기구)에서 정의한 스펙에 따라서 브라우저를 만든다. 따라서 전반적인 Frame은 거의 비슷하게 구현되지만, 스펙이 정의하지 않는 아주 디테일한 부분들은 각자의 상황에 맞게 전략에 맞게 구현을 하게 되어있다.

- 이것을 위해 각 브라우저들의 특징을 알려주는 Navigator라는 객체이다.

- 브라우저마다 다르게 동작한다는 이슈를 **Cross Browsing Issue**라고 한다.

▼ *App 이름 출력하기*

console.dir(navigator.appName); →

//IE - Microsoft Internet Explorer, FireFox, Chrome - Netscape로 표시한다.

▼ *App version 확인*

console.dir(navigator.appVersion); →

//5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) hrome/98.0.4758.102 Safari/537.36

*여기서AppleWebkit은 HTML layout Engine을 의미한다. Window는 내가 사용하는 운영체제

▼ *useAgent 확인*

F12에서 Network창에 들어가서 서버랑 주고받은 내용을 열람할수 있다. 여기서 목록중 하나를 클릭하고 header를 보면 User-agent를 볼수 있는데 google의 경우는 다음과 같았다.

User-Agent:
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/98.0.4758.102 Safari/537.36

*설명: Web Browser가 Web Server에게 정보를 request할때, Web Browser Web server에게 어떤 정보가 필요한지와 Web Browser에 대한 정보를 전달한다. 그 정보에는 현재 Web Browser가 어떤 브라우저인지를 알려주는 정보도 포함되어 있다. (위의 정보를 준다)

▼ *Platform 확인*

-현재 브라우저가 사용하고 있는 OS를 확인한다

navigator.platform

▼ 기능테스트 (쓸지는 모르겠음)

기능테스트라는 더 유용한 기능이 존재한다. 이 기능테스트가 무엇인고 하니? 내가 작성한 코드가 실행될 브라우저가 특정 기능을 가지고 있는지를 테스트해볼수 있는 기능이다.

Javascript에서 공부했던 것처럼 상위 객체중 Object객체가 존재한다. 이 객체는 keys라는 method를 지원한다.

```
if(!Object.keys){
//Object라는 객체가 (즉 JS의 뿌리가되는 객체가 keys를 가지고 있지 않다면 다음을 실행하게 된다.
Object.keys = (function() {
    'use strict';
    var haswOwnProperty = object.prototype.hasOwnPropery,
        hasDontEnumBug = !({toString: null}).propertyIsEnumerable('toString'),
        dontEnums = [
            'toString',
            'toLocaleString',
```

```
            'valueof',
            'hasOwnProperty',
            'isPrototypeOf',
            'propertyIsEnumerable',
            'constructor'
            ],
            dontEnusmLength = dontEnums.length;
        return function(obj)
```

→ 만약에 그 기능을 가지고 있지 않다면 추가해주는 코드라고 한다.

▼ 창 제어

Window객체는 전역 객체이다. Window라고하는 가장 큰 단위를 제어하는 객체라고 앞서 말했다. 이에 따라 Window를 제어하는 것들이 많이 있다. 그중 대표적인 것이 window를 열고 닫는 window.open과 window.close가 있다.

▼ Window open

Full-Code는 실습 파일에 있다. 창을 JS로 열고 닫는것을 제어한다. 새 창을 열거나 기존의 창에서 열거나 등

```
<script>
    function open1(){
        window.open('demo2.html');
    };
    function open2(){
        window.open('demo2.html', '_self'); //_self는 현재 창에서 load되게 된다. 따라서 새로운 창이 나오지는 않는다.
    };
        //open이란 것은 html에서 <a></a> anchor tag와 동일한 효과를 줄수 있게 해준다 (JS에서).
    function open3(){
        window.open('demo2.html', '_blank');//아무것도 없는 것과 같다 새창이 계속 열림.
    };
    function open4(){
        window.open('demo2.html', 'ot'); //이미 ot라는 이름으로 열린 창이 계속 열리게 된다. 창에 이름을 지정하게 되면 ot(임의의 이름임)
        //그 이름으로 열린 창을 새고침하듯 계속 열리게 됨 _self => ot가 된거라고 생각하면 됨 즉 두번째 argument에는 창의 이름이 들어가네
    }
    function open5(){
        window.open('demo2.html', '_blank', 'width=200, height=200, resizable=yes'); //창을 열때 width, height을 정해서 열어준다.
        //resizable =yes하면 그 창의 크기를 조절할수 있다.
    }
</script>
```

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/0c6c78c6-2c54-499a-9a4a-e48217b9d839/demo2.html

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/47af8423-07d7-4877-9b1c-22777fe5fa47/Window_control_open.html

▼ Window와 상호작용

창간의 상호작용을 하게 되는데 첫 번째 창에서 입력을 통해서 다른 창에서 출력이 바뀌는 것을 제어하는 것을 예시로 들었다.

```
<ul>
    <input type = "button" onclick="winopen();" value="open">
    <input type = "text" onkeypress="winmessage(this.value);">
    <input type = "button" onclick="winclose();" value="close">
</ul>
<script>
    function winopen(){
        win = window.open('demo2.html', 'ot', 'width=300, height=500, resizable=yes');
        //여기서 html객체를 win으로 가져와서 (html도 객체화가 됨 html객체는 document를 가지고 있어 )
        };
    function winmessage(msg){
        win.document.getElementById('message').innerText = msg;
    };
    function winclose(){
        win.close();
    };
</script>
```

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/737596ca-4380-4f78-b13c-7b20e7661786/Window_control_open.html

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/522b163e-be28-4ade-a6ac-dc6476889e59/demo2.html

▼ Popup 차단하기 (보안에 관한 이야기)

-브라우저에서 보안은 상당히 중요하다.

어떤 웹사이트에 들어가 그 웹사이트가 가지고 있는 기능이 내 웹사이트를 제어해서 특정한 파일을 자신의 웹사이트에 전송할수 있게 되면 이게 바로 보안 취약점이 된다.

서로 도메인이 같으면 같은 경로상에 있으니까 JS를 가지고 충분히 같은 도메인상의 다른 페이지를 제어할수 있게됨. 하지만 다른 도메인이라면 그것이 불가능해진다.

Pop-up이라는게 사용자들에게 불편함을 많이 줬기때문에 브라우저 업체들이 기본적으로 팝업을 차단하게 만들었다. (사용자가 허용할때만 가능)

따라서, 다음과 같이 자동으로 Pop-up되게 해놓은 것은 차단이 자동적으로 된다 (오른쪽 상단에 팝업 차단) 아래와 같은건 자동으로 Pop-up되게 만들었기때문에 차단됨

```
<html>
    <body>
        <script>
            window.open('demo2.html');
        </script>
    </body>
</html>
```

하지만 위에서 button 클릭해서 열리는건 괜찮음 <상호작용쪽에서>

[DOM - Document Object Model]

앞에서는 BOM 브라우저와 사용자간의 상호작용이 목적이었다면, 이번에는 내부적으로 문서를 제어하는 기능들을 목표로 하고 있는 DOM(Document Object Model)을 살펴볼 것이다. 정말 중요하다.

[제어하기 위한 단계는 두 단계로 이루어져 있다]

Step1. 제어의 대상을 찾는다

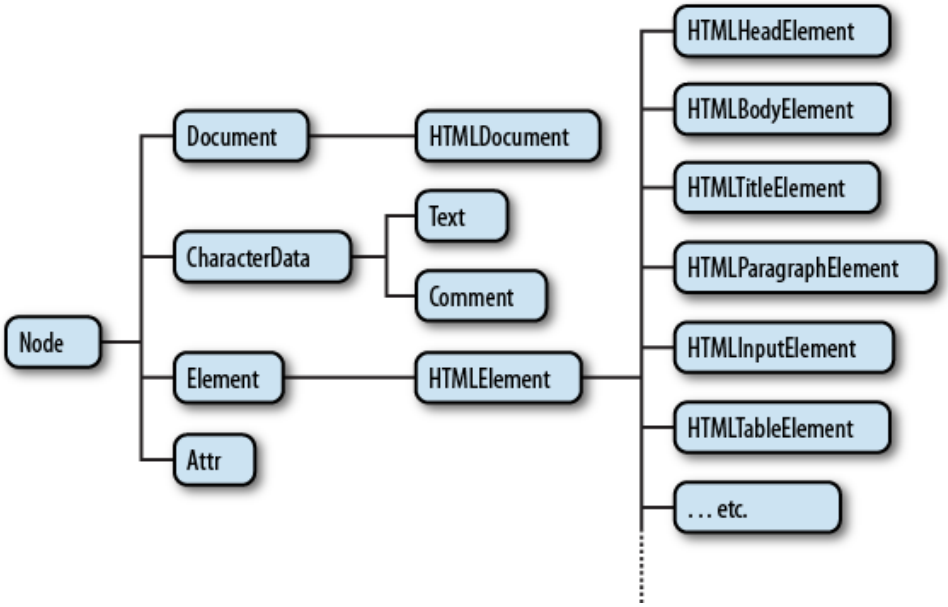
Step2. 대상을 조정한다(Manipulation)

[메커니즘의 이해]

기본적인 메커니즘은 우리가 만든 HTML의 tag를 기반으로 브라우저가 웹페이지를 만드는데 그 과정에서 각각의 tag에 해당되는 객체들을 만들어놓는다 (브라우저가) 우리가 **웹페이지에서 작업을 할려면 브라우저가 만들어 놓은 객체를 찾아야 한다**.

그 객체를 찾아내는 것이 Step1. 제어 대상을 찾는 방법이다. 아래에서 그 방법들을 제시해 놓았다. 하지만 대상을 찾기 전에 우선 그 찾는 대상 HTML내의 Element들에 대해서 알아보도록 하자.

[DOM Tree structure]



[Figure DOM-Tree]

HTML내에서 <a> 등 많은 tag들을 가지고 있다. 이런 tag하나 하나가 HTMLElement객체에 속한 요소들이다. 위의 DOM-Tree에서 보면 알겠지만, 위의 tag 요소를 나타내는 객체 HTMLHeadElement, HTMLLIElement, HTMLAnchorElement 들 모두 HTMLElement부모 객체를 상속한다.

[Object → Node → Element → HTMLElement → 각 tag들의 객체 (eg. HTMLHeadElement 등)]

즉, 우리가 Step1에서 해야하는 것은 우리가 제어해야 할 HTMLElement 요소들을 찾고 무엇으로 되어 있는 객체인지 알아내는 것이다. (예를 들어 id가 'AA'인 것을 찾고 그 id가 'AA'인 tag요소는 어떤 객체로 되어 있는지 HTMLLIElement인지, HTMLHeadElement인지를 직별하는 것까지가 제어 대상을 찾는 부분이다)

[Step1. 제어 대상찾기]

▼ 1. document.getElementsByTagName

Tag의 이름을 통해서 모든 요소들을 받아오는 방법 (실습)

```
<body>
    <ul>
        <li>HTML</li>
        <li class="active">CSS</li>
        <li class="active">JavaScript</li>
    </ul>
```

```
    <ol>
        <li>HTML</li>
        <li class="active">CSS</li>
        <li class="active">JavaScript</li>
    </ol>
    <script>
        let lis = document.getElementsByTagName('li');  //document객체는 HTML문서 전체를 의미한다.
        //여기서 lis는 유사배열이라고 한다 배열과 비슷하게 작동하지만 그렇다고 배열은 아니다 type은 object HTMLCollection 이네
        //https://devsoyoung.github.io/posts/js-htmlcollection-nodelist
        console.log(lis)
        for (let i=0; i<lis.length; i++){
        lis[i].style.color = 'red';
        }
        //근데 만약 내가 ul안에만 있는 li에 대한것만 색을 바꾸고 싶다?
        let ul = document.getElementsByTagName('ul')[0];  //[0]은 첫 번째 ul에 관한 것만 가져 오겠다는 것 getElements이므로 모든 tag를 가져올거임
        let lisoful = ul.getElementsByTagName('li');
        //첫 번째 ul에 해당되는 객체를 찾고 그 객체에 대해서 li를 찾으면 된다.
        console.log(lisoful)
        for (let i=0; i<lisoful.length; i++){
        lisoful[i].style.color = 'green';
        }
        //style은 css에 관한건데 이건 어떻게 control하는 것인지?
    </script>
</body>
```

▼ 2. document.getElementsByClassName

Class(그룹)이름을 통해서 해당 요소들을 받아오는 방법 (실습)

```
<body>
    <ul>
        <li>HTML</li>
        <li class="active">CSS</li>
        <li class="active">JavaScript</li>
    </ul>
    <ol>
        <li>HTML</li>
        <li class="active">CSS</li>
        <li class="active">JavaScript</li>
    </ol>
    <script>
        let lis = document.getElementsByClassName('active');
        console.log(lis)
        for (let i=0; i<lis.length; i++){
        lis[i].style.color = 'red';
```

```
        }
    </script>
</body>
```

▼ 3. document.getElementsById

특정 요소를 찾기위해 많이 사용되는 방법중 하나

특정 id만을 찾는 것이기때문에 가장 성능이 좋다 이걸로 할 수 있으면 이걸로 해야함.

```
<body>
    <ul>
        <li>HTML</li>
        <li class="active" id="active">CSS</li>
        <li class="active">JavaScript</li>
    </ul>
    <script>
        let li = document.getElementById('active');
        //얘는 특징이 앞서 받아온 것과 다르게 s가 붙지 않는다. 하나의 결과만 가져온다는 특징이 있다.
        //가장 성능이 좋다 얘를 쓸수 있다면 무조건 얘를 써야한다.
        //본래 id용도는 class와 다르게 (같은 class에서도) 각 각 element들을 식별하기위한 용도이다.
        //따라서 하나만 있는 것이 맞다.
        console.log(li)
        li.style.color='red';
    </script>
</body>
```

▼ 4. document.querySelector

querySelector는 CSS선택자로 내가 원하는 Element들의 객체를 찾아서 우리에게 return해주는 역할을 한다. (*querySelector는 맨 처음에 나온 하나만을 반환하기때문에 모든 요소들을 유사배열로 반환하고 싶다면 querySelectorAll을 써야한다)

```
<script>
//CSS에서 선택자라는 것이 있다. 선택자를 통해서 우리가 꾸며주고자 하는 Element를 design할수 있는 기능이 있다.
//바로 선택자를 인자로 받아서 선택자에 해당되는 element들의 객체를 찾아서 우리에게 return해주는 method
    let li1 = document.querySelector('li');
//querySelector는 조건에 맞는 인자 하나만을 return해 준다. (가장 먼저 발견된 것 하나만)
//querySelectorAll은 해당 인자를 모두 유사배열에 담아서 return한다.
    li1.style.color = 'red';
    let li2 = document.querySelector('.active');
    li2.style.color='blue';
        let lisall = document.querySelectorAll('li');
    for(let name in lisall){
    //JS에서 in은 index를 of는 그 index에 있는 데이터 인자를 받아온다
    lisall[name].style.color = 'blue';
}
// * CSS선택자의 역할: CSS 선택자는 CSS 규칙을 적용할 요소를 정의합니다.
```

```
//.querySelector()는 CSS 선택자로 요소를 선택하게 해줍니다.
</script>
```

지금까지 DOM programming (웹 브라우저의 문서를 제어하기 위해서)을 하기 위해서 우선 제어 대상인 객체를 찾아내는데, 찾아낸 것일뿐 아직 그 객체의 Identity를 모르는 상태이다. 도대체 어떤 객체인가에 대한 이해가 필요하다.

→ 즉, 이번에는 찾아낸 객체에 대한 정체성을 알아보는 작업을 할 것이다.

▼ 1. HTMLElement

-HTML 요소들의 객체에 관해서

아래의 코드에서 설명을 적어놓았지만, document.getElementById('list');, document.getElementByTagName('anchor');은 각 ()안에 해당되는 tag들의 객체를 반환해 준다.

(위의 document.getElementById('list');의 경우 id가 list인 tag의 객체를 반환해준다. 이 경우는 <li> tag로 되어있기때문에 HTMLLIElement 객체를 반환해 줄 것이다. 하나 의문 같으 li의 경우 어떻게 식별하지? numbering하나?)

따라서 아래 코드를 기준으로 봤을때 반환된 객체 target.constructor.name을 console.log에 찍어보면 HTMLElement가 반환된다. target을 출력해보면 HTMLElement{id, type} 객체가 반환된다.

```
<body>
    <a id="anchor" href="https://opentutorials.org">opentutorials</a>
    <!--a객체의 속성들을href 등을 위와같이 표현할수 있게 해준거임-->
    <ul>
        <li>HTML</li>
        <li>CSS</li>
        <li id="list">JavaScript</li>
    </ul>
    <input type="button" id="button" value="button"/>
    <script>
        let target = document.getElementById('list');
        console.log(target.constructor.name); //return HTMLLIElement (Object)
        let target1 = document.getElementsByTagName('anchor');
        console.log(target1.constructor.name); //return HTMLAnchorElement (Object)
        let target2 = document.getElementsByTagName('button');
        console.log(target2.constructor.name); //return HTMLInputElement (Object)
        //HTMLElement는 공통으로 들어가지만 각 tag에 따라 name이 다르다.
        //https://opentutorials.org/course/1375/6665 각 객체 name에 따라 어떻게 되어 있는지 속을 까볼수 있다.
        //하나의 예시로 HTMLLIElement 객체는 속성으로 type과 value를 가지고 있다 type은 list의 네모, 원 을 조절하고
        //value는 그 안에 있는 값을 변경해준다.
        /*(까보니까 interface로 되어있네)
        interface HTMLIElement : HTMLElement{ //여기서 알수 있는 것은 : HTMLElement는 HTMLElement를 상속받는다 라는 것
        attribute DOMstring type;
```

```
        attribute long value;}
        interface HTMLAnchorElem : HTMLElement{
        ...}
        상속이란 객체가 다른 객체의 property를 그대로 물려받으면서 동시에 자기 자신에게 필요한 property를 추가할수 있는 것
        자식객체 부모객체 개념
        */
        target.type = "square";  //뭐 보통 CSS에서 함.
        /*https://web.stanford.edu/class/cs98si/slides/the-document-object-model.html
        Dom-tree*/
    </script>
</body>
```

▼ 2. HTMLCollection

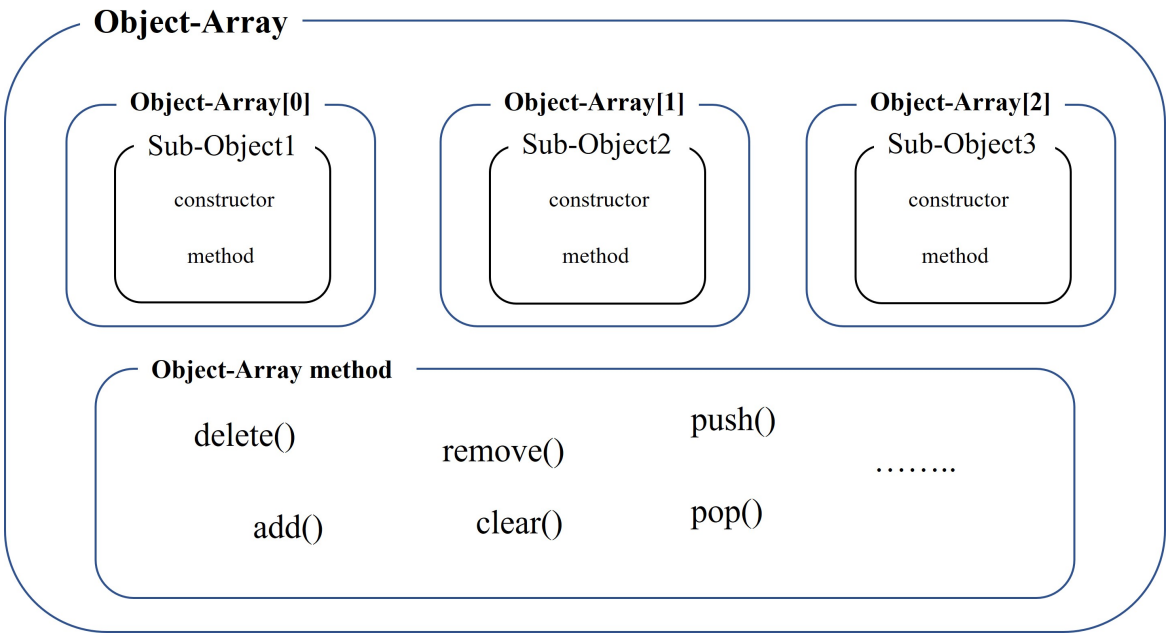
HTMLElement를 조회할때 만약 여러개를 조회해서 반환한다면, 다음과 같다

```
let li1 = document.getElementByTagName('li');  //맨 위에서 찾은거 하나를 반환
let li2 = document.getElementsByTagName('li');  //찾은거 모두 반환 Elements (s)가 붙음
```

li1은 ***console.log(li1.constructor.name)*** 출력시 HTMLElement가 반환되는데

li2은 ***console.log(li2.constructor.name)*** 출력시 HTMLCollection(유사배열)이 반환된다.

여기서 알수 있듯이 여러개의 HTMLElement객체를 담은 HTMLCollection (유사배열 not 배열)을 반환한다. HTMLCollection또한 객체이다.



위의 그림은 내가 그려놓은 그림으로 만약 Sub-Object1에 접근해서 method를 쓰고싶으면, Object-Array[0].method로 사용하면 될 것이고 Array자체의 method가 필요하다면, Object-Array.clear() 등을 사용하는게 아닐까 내 경험상으로는 그렇다.

```
<script>
    console.group('before');
    let lis = document.getElementsByTagName('li');
    for(let i=0; i<lis.length; i++){  //이 문법들은 도대체 뭘로 짜여지는 것일까? 궁금함
    console.log(lis[i]);}  //배열 Access operator를 사용해서 배열을 사용할수 있다.
    console.groupEnd();  //Tip console.group(group name)시점부터 groupEnd까지 모두 grouping해서 보여줌 시각적으로 좋음
    console.group('after');
    lis[1].parentNode.removeChild(lis[1]);
    for(let i=0; i<lis.length; i++){  //이 문법들은 도대체 뭘로 짜여지는 것일까? 궁금함
    console.log(lis[i]);}  //배열 Access operator를 사용해서 배열을 사용할수 있다.
    console.groupEnd();
</script>
```

위의 코드에서 보면 HTMLCollection으로 여러 객체를 반환해 오는 것을 알수있다. 또한 removeChild로 두 번째 요소를 제거하여 출력한 것이다.

*하나 더 추가하자면, console.group('name') - console.groupEnd()는 이 사이에 출력된것을 'name'이라는 group으로 grouping해서 출력해준다.

▼ [DOM Tree - (Element Object)]

위의 DOM-Tree를 보면 한 가지 의문이 들만한 점이 있을 것이다. Element바로 아래 HTMLElement가 있는데 왜 이렇게 세분화 해놨을까 이다.

→ DOM이라고 하는 것이 HTML에만을 목적으로 만들어진 것이 아니기 때문이다. 마크업 언어들을 제어하기 위한 규격이 DOM이기때문에 비단 HTML만이 아니라 XML, SVG, XUL 등 다른 마크업 언어들에도 해당된다.

여기서 Element 객체는 전반적인 마크업 언어들을 제어하기 위한 것이다. 그리고 HTMLElement는 HTML에서 부가적으로 가지고 있어야 할 사항들을 추가시켜 놓은 것이라고 볼수 있다. (DOM을 공부하면 HTML, XML, SVG 등을 제어할수 있는 방법을 알수있다.

지금하고 있는 Element만이 아니라 다른 CharacterData, Dcoument, Attr 등까지 다루고 전반적인 DOM의 구조를 알아보는 것을 목표로한다.

[Element객체의 기능을 정리해보자]

1. 식별자 - 문서내에서 특정한 Element를 식별하기 위한 용도로 사용되는 API

   (Element.classList, Element.className, Element.id, Element.tagNae)

2. 조회 - Element 하위 Element를 조회하는 API

(Element.getElementsByClassName, ...)

3. 속성 - Element의 속성을 알아내고 변경하는 API

   (Element.getAttribute(name), Element.setAttribute(name, value))

앞서 사용했더 dcoument object는 document자체를 의미한다.

Element가 가지고 있는 API중에 Element를 식별할수 있는 식별자와 관련있는 API에 대한 것을 알아보도록 할 것이다. 앞서서 getElementById등으로 우린 Element들을 찾았다. 바로 그때 ***각 Element들이 자신을 찾아낼수 있도록 이름을 갖는 것 이것이 식별자***이다. 이에 해당되는 것은 id, class 등이 있다. ***식별자 API는 이 식별자를 가져오고 변경하는 역할을 한다*** (id, class 등을 가져오고 변경하는 역할을 한다)

▼ *1. Element.tagName*

```
<ul>
    <li>HTML</li>
    <li>CSS</li>
    <li id="active" class="important current">JavaScript</li>
</ul>
<script>
    console.log(document.getElementById('active').tagName);
    //여기서document.getElementById('active') 는 문서객체에서 HTMLElement를 찾아온다.
    //그러면 결국 HTMLLIElement.tagName (HTMLElement는 tagName이라는 객체가 있다)
    //tagname은 li가 될것이다.
    document.getElementById('active').tagName = 'a';  //실행 안됨, tagName읽기전용이다.
</script>
```

tagName은 Element의 tag이름을 반환한다. LI, Anchor 등

▼ *2. Element.id*

객체가 가지고 있는 Property중 id라는 것을 알아보자. 이것은 그룹을 나눌때 사용되는 class와 다르게 각 element를 분간할수 있는 고유한 식별자이다. (하나의 tag만이 가지고 있는 식별자)

- id의 경우는 찾는것부터 새롭게 할당까지 가능하다 -

```
let active = document.getElementById('active');
//그 특정 Element id로 검색해서 객체 반환
console.log(active.id);  //그 특정 id 출력 (active출력)
active.id = 'deactive';  //새로운 id를 할당해준다
console.log(active.id);  //deactive가 출력될 것
```

▼ *3. Element.className*

class같은경우 html속성이름(class)과 JS에서 쓰이는 property(className)의 이름이 다르다.

class는 JS의 예약어인경우에 다를수 있다는데

```
<script>
    let active = document.getElementById('active');  //그 특정 Element id로 검색해서 객체 반환
    active.className = "important current";
    console.log(active.className);  //그 특정 id 출력 (active출력)
    //클래스 이름을 추가할때는 아래와 같이 문자열을 더한다
    active.className += " readed";
    console.log(active.className);
</script>
```

className으로 이름을 변경할수 있고 삭제할수 있지만, 매우 불편하다 추가할려면 기존에 이름 있는지를 확인해서 추가해야 하는 등 번거롭다 이걸 해결할수 있느 것이 다음에 볼 Element.classList 이다.

▼ *4. Element.classList*

```
<li id=”active” class ="marked current"> JavaScript</li>
let active = document.getElementById('active');
```

일때 `active.classList`를 출력해보면

Result → `DOMTokenList{0: “marked current”, length:2, ithem: function, contains: function, add: function, remove: function...}` 이 반환된다.

즉, Element.classList - classList property는 DOMTokenList 객체를 반환한다. DOMTokenList는 유사배열로 class안의 요소들을 배열처럼 가지고 있다. 예를 들어 위의 경우 class로 makred current 두 단어를 가지고 있다.

`active.classList[0]` ⇒ marked

`active.classList[1]` ⇒ current

그리고 active.classList는 위와같이 배열로 저장된다.

그 말은 우리가 for loop로 배열을 볼수 있다는 의미가 된다. 그에 대한 코드는 다음과 같다.

```
for(let i=0; i<active.classList.length; i++){
    console.log(active.classList[i]);}
```

그리고 만약 위의 marked current에 ‘important’라는 단어를 추가하고 싶다면 배열 method와 같이 `active.classList.add('important');`를 하면 추가가 되고 다시 지우고 싶다면 `active.classList.remove('important');`를 쓰면 된다.

그리고 여기서 **toggle** 기능에 대해서 소개를 해보자면, toggle은 switch역할을 한다 (껐다 켰다) 있으면 없애고, 없으면 생성하고 이에 대한 문법은 다음과 같다.

`active.classList.toggle('important');` 시행할때마다 생성됐다 제거됐다 함.

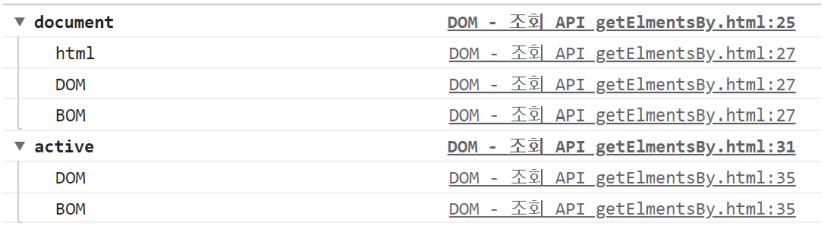
▼ *[차이점] document.getElementsBy* & Element.getElementsBy**

* 한 가지 구분하고 가야할 것이 있다 바로 앞단에서 **document.getElementsBy* 방법을 사용한 적이 있었다. 그런데 Element.getElementsBy*와 무슨차이**가 있을까? 알아보도록 하자.

결론부터 말하자면, *document.get*는 HTML문서 전체를 객체로 받아와서 범위가 HTML전체 범위 중에 찾게 되는 것*이고 *Element.get*는 내가 찾아낸 Element하위의 객체들중에서 내가 원하는 것을 찾는 방법*인 것이다.

```
<script>
    let list = document.getElementsByClassName('marked');
    console.group('document');
    for(let i=0; i<list.length; i++){
        console.log(list[i].textContent);}
    console.groupEnd();

    console.group('active');
    let active = document.getElementById('active');
    let list1 = active.getElementsByClassName('marked');
    for(let i =0; i<list1.length; i++){
        console.log(list1[i].textContent);}
    console.groupEnd();
</script>
```



위의 결과를 보면 group ‘document’는 html DOM, BOM 현 HTML문서에 존재하는 모든 class가 marked인 객체들을 반환했다. 반면에 group ‘active’는 id가 active인 Element안에서 class이름이 marked인 객체들만 반환했다.

[Step2. 제어대상 제어하기]

여태까지 Element의 tag이름 또는 식별자(id, Class)를 이용해 조회하는 것을 알아보았다.

*식별자는 Element를 그룹핑하고 구분하기 위해 사용된다.

자 이제 Element를 찾았으니까 그 Element의 속성(Atrribute)를 어떻게 제어하는지 알아보도록 하자.

▼ Element.setAttribute

찾은 Element의 속성을 바꿔줄수 있다.

t.setAttribute(’id’, ‘B’); ↔ ***t.id = ‘B’;*** 를 사용

t.setAttribute('class', 'C'); ↔ ***t.class = 'C';*** 를 사용

t.setAttribute('href', 'D'); ↔ ***t.href = 'D';*** 를 사용

```
<a id="target" href="http://opentutorials.org"> opentutorials</a>
<script>
    let t = document.getElementById('target'); //HTMLAnchorElement객체가 들어오겠지
    //oop인 객체지향언어는 직접 접근이 아니라 reference로 정보를 주고받는다
    console.log(t.getAttribute('href')); // href 속성값을 반환한다.
    //t.id, t.href를 통해서 쉽게 가져올수도 있지만 getAttribute도 가져올수 있다
    t.setAttribute('href', 'http://www.naver.com'); // 속성을 바꿀때 set을 사용한다
    t.setAttribute('title', 'opentutorials.org'); // 없는 속성이라면 만들어진다.
    t.removeAttribute('title'); //다음과 같이 삭제도 된다.
    //속성이 있는지 없는지 확인하고 싶다?
    t.hasAttribute('title'); // 이게 나온다. boolean으로
    t.href = 'http://www.kakao.com' // 사실 이렇게 해도 됨.
</script>
```

*속성방식, Attribute 방식

t.setAttribute('class', 'important');

*프로퍼티 방식, Property 방식

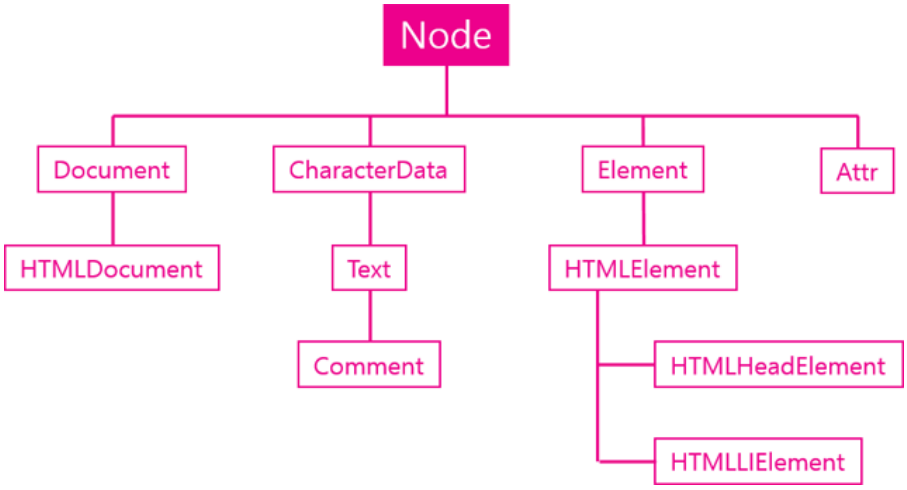
t.className = 'important'

위에서 보면 알겠지만, Property방식이 좀 더 간편하고 속도도 빠르지만 실제 html속성 이름과 다른 이름을 갖는 경우가 있다 (Attribute와 Property이름이 다른 경우)

Attribute	class	readonly	rowspan	maxlength
Property	className	readOnly	rowSpan	maxLenght

그리고 출력이 다른 경우도 존재한다. href에 Property방식으로 접근할경우 웹페이지 https//부터 전체 주소가 나오는데, getAttribute로 href에 접근하면 ./demo.html만 나온다.

[DOM Tree - (*Node Object*)]



Node객체는 Object객체의 자식이자 DOM에서 시조와 같은 역할을 한다. ***다시 말해 모든 DOM 객체는 Node 객체를 상속 받는다***. (DOM의 모든 객체들은 Node객체를 상속받기 때문에 서로 다른 Branch끼리 서로의 관계를 알아볼수 있는 통로가 된다).

구체적으로 Node 자식은 여태까지 우리가 다루어봤던 Element(각 tag), 각 tag의 속성(Attr), Text (
Javascript, <!— comment —> 등), 문서 전체를 의미하는 document 가 있다.

Node객체가 가지고 있는 주요한 기능에 대해서 알아보자. (Node객체의 메소드, 속성 등)

```
* Element는 서로 부모, 자식, 형제자매 관계로 연결되어 있다. Node객체는 각 Node가 다른 Node와 연결된 정보를 프로그래밍적으로 탐색할수 있게 해주는 API를 가지고 있다.
```

부모랑 자식의 관계의 예는 다음과 같다.

```
<ul>  //부모 Element(또는 Node)
    <li>Javascript</li> //자식 Element(또는 Node)
    <li>html</li> //자식 Element(또는 Node)
</ul>
```

위와 같이 있을때 <ul> Element에 속해있는 자식 Element들은 <li> 2개가 존재한다.

▼ [Node관계 API]

- ***Node.childNodes*** → Element의 자식 Element들이 <li>, #text 등인 것을 알 수 있다.
- ***Node.firstChild*** → 첫 번째 자식이 누구인지 알수 있게 해준다.
- ***Node.lastChild*** → 마지막 자식이 누구인지 알수 있게 해준다.
- ***Node.nextSibling*** → 위의 예제에서 첫<li>의 다음 형제 Element (두 번째<li>)인것을 알려준다.
- ***Node.previousSibling*** → 두 번째 <li> Element에서 전 형제 Element가 첫째 임을 알려준다.
- ***Node.contains()*** → 자식 Element가 있는지를 확인해준다.
- ***Node.hasChoildNodes()***→ contain과 비슷한 용도로 사용된다.

→ 위 method또는 속성은 Node객체에 정의되어 있기때문에, Text 정보, Element든, 문서전체이든 상관없이 서로 관계(Text, Element 등 모두 Node객체를 상속하기때문에)를 알수 있다.

```
<body id="start">
    <ul>
        <li><a href="./532">html</a></li>
        <li><a href="./532">css</a></li>
        <li><a href="./532">JavaScript</a>
            <ul>
                <li><a href="./532">JavaScript Core</a></li>
                <li><a href="./532">DOM</a></li>
                <li><a href="./532">BOM</a></li>
            </ul>
        </li>
    </ul>


<script>
let s = document.getElementById('start');
console.log(1, s.firstChild); // #text <li>일것 같지만, 띄어쓰기, 줄바꿈 등 모두 자식 Node로 인식한다.
/*<ul> Element type의 node를 받고싶다면 <body id="start"><ul> 이렇게 붙여써야한다
참고로 #text는 DOM-tree상에서 Text객체에 해당된다*/
let ul = s.firstChild.nextSibling;
console.log(2, ul);  //body element의 첫 자식 #text(<body id="start"> 이부분) 의 다음 형제는 <ul>이 지목된다
console.log(3, ul.nextSibling); //<ul>의 다음 형제는 </ul>다음에 있는 #text가 된다
console.log(4, ul.nextSibling.nextSibling); //</ul>다음의 #text의 다음 형제는 비로소 <script>가 된다
console.log(4.1, ul.firstChild); //<ul>다음 #text가 바로 첫 자식으로 들어온다
console.log(4.2, ul.firstChild.nextSibling); //<ul> -> #text의 첫 형제가 비로소 <li>가 들어온다
console.log(4.3, ul.firstChild.nextSibling.parentElement);  //<ul> element가 된다.
console.log(5, ul.childNodes); //현재 시점 node내부에 있는 모든 자식 노드들을(각 객체들) 유사배열로 출력하게 된다.
console.log(6, ul.childNodes[1]); //유사배열이라 내부에 있는것은 index로 접근이 가능하다
console.log(7, ul.childNodes.length); //길이를 뽑을수 있다.
</script>
</body>
```

▼ [Node 종류및 값 API]

위에처럼 Node들 사이의 관계만이 아니라 노드의 종류, 값 등을 알려주는 역할도 한다. Node들에는 어떤 이름이 있는지 알아보기 위히서는 다음과 같은 Code를 실행시킨다.

```
for(let name in Node){
    console.log(name, Node[name]};
```

위의 Code를 실행시키면 각 Node의 이름과 부여받은 숫자(상수)를 알수 있다. 예를 들어

ELEMENT_NODE 1, TEXT_NODE 3, COMMENT_NODE 8 등과 같이 출력된다.

- ***Node.nodeType*** → Element인지, Text인지, Comment인지 등 종류를 알려준다.
- ***Node.nodeName*** → Node 이름을 알려준다.
- ***Node.nodeValue*** → Node가 가지고 있는 값을 알려준다.
- ***Node.textContent*** → Node 하위 Text가 어떤것인지 알려준다.

```
//Node Type에 대해서
console.log("body_type", s.nodeType); // body Element에 대해서 nodeType을 출력하니 // 상수 1이 나온다.
console.log("#text", s.firstChild.nodeType); // body의 첫 Child는 #text이고 text type은 상수 3이다.
console.log("document", document.nodeType); //문서 전체를 의미하는 document의 type은 9번이다.
console.log(document.nodeType ===9); //true가 출력될 것이다. 나중에 식별할때 쓸수 있겠다 논리형으로
console.log(document.nodeType ===Node.DOCUMENT_NODE); //숫자 기억하기 힘들면 이름으로 기억하자.
 //Node객체의 속성들 (구성요소들 이름으로 넣어도 된다.)
//결론 nodeType은 각 node 객체 종류에 부여된 숫자가 출력된다. 그렇다면 nodeName은?
console.log("body_type Name", s.nodeName); //BODY가 나온다.
console.log("Text Name", s.firstChild.nodeName); //#text가 나온다.
console.log("Text Name", s.firstChild.nextSibling.nodeName); //UL이 나온다.
```

추가적으로 Node객체의 자식을 추가하는 방법에 대한 API도 제공을한다

- ***Node.appendChild()*** → 임의의 Node가 하위 Node(자식)를 추가할수 있게 해준다.
- ***Node.removeChild()*** → 임의의 Node가 하위 Node(자식)를 제거해주기도 한다.

▼ [Node 추가 API] - 중요

```
<ul>
    <li>HTML</li>
    <li>CSS</li>
    <li id="target">JavaScript</li>
</ul>
<input type = "button", onclick="callRemoveChild();" value ="removeChild()">
<input type = "button", onclick="callReplaceChild();" value ="replaceChild()">
<script>
    function callRemoveChild(){
    let target = document.getElementById('target');
    target.parentNode.removeChild(target);  // target의 부모를 알아야 자식을 삭제할수 있다.
    //삭제할 권한이 부모가 가지고 있기때문에 그렇다
    }
    //노드 바꾸기 replaceChild
    function callReplaceChild(){
    let a = document.createElement('a');
    a.setAttribute('href', 'http://opentutorials.org/module/904/6701');
    a.appendChild(document.createTextNode("Web Browser JavaScript"));
    let target = document.getElementById('target');
    target.replaceChild(a, target.firstChild);
    }
</script>
```

▼ [Node 삭제 및 교체 API] - 중요

```
<ul>
    <li>HTML</li>
    <li>CSS</li>
    <li id="target">JavaScript</li>
</ul>
<input type = "button", onclick="callRemoveChild();" value ="removeChild()">
<input type = "button", onclick="callReplaceChild();" value ="replaceChild()">
<script>
    function callRemoveChild(){
    let target = document.getElementById('target');
    target.parentNode.removeChild(target);  // target의 부모를 알아야 자식을 삭제할수 있다.
    //삭제할 권한이 부모가 가지고 있기때문에 그렇다
    }
    //노드 바꾸기 replaceChild
    function callReplaceChild(){
    let a = document.createElement('a');
    a.setAttribute('href', 'http://opentutorials.org/module/904/6701');
    a.appendChild(document.createTextNode("Web Browser JavaScript"));
    let target = document.getElementById('target');
    target.replaceChild(a, target.firstChild);
    }
</script>
```

[DOM Tree - (Document *Object*)]

앞서 얘기한 것 처럼 Document 객체는 Node객체 아래 있고 최상위에는 Window객체가 존재한다.

여기서 document는 문서 (html)전체를 객체로 받아오는 역할을 한다.

그렇다면 document의 자식들은 어떻게 구성이 되어 있을지 알아보도록 하자.

document.childNodes → [<!DOCTYPE html>, <html>...</html>]

document.childNodes[0] → <!DOCTYPE html>

document.childNodes[1] → <html>...</html>

//html안에 있는 문서 전체가 document 객체의 2번째 자식이 된다.

참고로 DOCTYPE html과 <html></html>은 형제 자매이다.

Document 객체가 하는 일은 무엇이 있을까?

1. ***노드 생성 API*** → 문서에 소속될수 있는 Node나 Element를 생성해주는 역할을 한다.

#이후 Node객체가 가지고 있는 appendChild등을 가지고 추가해주거나 한다.

2. ***문서 정보 API*** → document.title 등이 있다.

[DOM Tree - (Text *Object*)]

만약 생활코디 가 있다면 는 Element이고 생활코딩은 Text객체에 속한다.

▼ Event Programming

이벤트란 사건을 의미한다. 즉, 이벤트 프로그래밍이란 이벤트가 발생했을때 프로그래밍적으로 어떠한 작업을 할수 있게 해주는 것.

- 브라우저에서의 사건이랑 사용자가 클릭했을 “때!”, 스크롤을 했을 “때!”, 필드의 내용을 바꾸었을 “때!” 와 같이 어떤 행동을 할때를 이벤트가 발생했다고 한다.

```
<input type = “button” onclick="alert(window.location)” value ="alert(window.href)”/>
<input type = “button” onclick="window.open('bom.html’)” value =”window.open(‘bom.html)’/>
<input type = “button” onchange="alert(this.value)” />
```

버튼을 눌렀을때 alert가 뜨게 한다 이것 또한 Event Programming이다.

Event와 관련된 용어들을 정리해보자

1. ***Event target*** - event의 대상이 되는 것을 event target이라 한다. 위에서는 Button이 된다. ***(즉 이벤트 행동이 행해지는 대상을 의미한다)***
2. ***Event type*** - ***이벤트 행동의 유형을 의마한다.*** 위에서는 onclick 클릭했을 때!, onchange 무언가 변화했을때!
3. ***Event handler (Event Listener)*** - 이벤트가 발생했을 때 동작하는 코드를 의미한다. 즉, 이벤트가 발생한 것에 의해 일어나는 (후속)동작을 의미한다. 위에서는

   “alert(window.location”, “window.open(’bom.html)’과 같은 것을 의미한다. 보통은 function을 Event handler라고 한다.

   http://www.tcpschool.com/javascript/js_event_eventListenerRegister

이벤트를 동작하는 방식은 다음과 같이 분류할수 있다.

▼ Inline 방식

위에서 Event용어를 살펴봤을때 Element안에 넣었던 (일렬로 짠 코드)가 Inline방식이다.

```
<input type=”button” id=”target” onclick=”alert(‘Hello world,
‘+document.getElementById(‘target’).value);” value=”button”/>
```

```
<input type="button" id="target" onclick="alert('Hello world, '+this.value);" value="button"/>
//this는 객체 자신을 의미하기때문에 this.value를 써도 된다.
```

▼ PropertyListener 방식

<input type="button" id="target" value="button"/>

//위에서의 Property는 type, id, value, onclick 등의 Element의 property들을 의미한다.

```
<script>
    let t = document.getElementById('target');
    t.onclic = function(event){
        //input의 property를 직접 가져와서 function을 넣어주는 방식
        alert('Hello world');
    }
</script>
```

▼ addEVentListener 방식

```
<body>
    <input type = "button" id ="target1" value = "button1"/>
    <input type = "button" id ="target2" value = "button2"/>
<script>
    let t1 = document.getElementById('target1');
    let t2 = document.getElementById('target2');
    function btn_listener(event){
    switch(event.target.id){
    case 'target1':
    alert(1);
    break;
    case 'target2':
    alert(2);
    break;}
    }
    t1.addEventListener('click', btn_listener);  //(event.type, event.handler)
    t2.addEventListener('click', btn_listener);  //(event.type, event.handler)
    //Event listener를 하나 만들어 놓고 여러개의 event target에 적용시킬수 있다. (Event Listener 재활용)
    //그 이벤트가 어느 이벤트 target에서 발생했는지를 알아보기 위해서 event.target을 쓰면 된다.
    //addEventListener를 쓰는 이유는? property Listener는 위와 같이 두개를 쓰면 덮어쓰게 된다.
//하지만 addEventListener는 두 개 따로 실행이 된다.
</script>
```