Javascript_1

HTML, CSS 의 관계에대한 이해

HTML 과 javascript

- javascript는 HTML 태그에 접근하고(select), 수정(update), 생성 및 추가(create, append) 그리고 삭제(remove)를 할수 있습니다.
- HTML 태그 자체뿐 아니라 해당 태그의 속성들에도 위와 같은 액션들을 수행 할 수 있습니다.
- HTML을 조작하는 액션에 대해서 HTML 을 기초로 동작하기 때문에 javascript 를 닫는 body 태그 바로 위에 위치하는것이 실수를 줄일 수 있습니다.

```
<!DOCTYPE html>
<html>
<head>
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" >
 <title>javascript</title>
 <script type="text/javascript"
src="http://m1.daumcdn.net/svc/original/U03/cssjs/jquery/jquery-1.9.0.min.js"></script>
 <script type="text/javascript"
src="http://m1.daumcdn.net/svc/original/U03/cssjs/userAgent/userAgent-1.0.14.min.js"></script>
 <script type="test/javascript">
  jQuery.noConflict(); //jQuery를 $ 변수의 충돌을 방지하기위해 $를 사용하지 않고 jQuery변수를 사용
</script>
</head>
<body>
<div id="wrap">
 ......
</div>
<script type="text/javascript">
 var ua = daumtools.userAgent();
 var wrap = document.getElementById('wrap'); // 기본방식
 var wrapJ = jQuery('#wrap'); // jQuery 방식
 console.log(wrap);
 console.log(wrapJ);
 console.log(ua);
</script>
</body>
</html>
```

- javascript 가 head 태그에 기술되어 있으면 해당 javascript 를 먼저 접근을 하기때문에 HTML태그 즉 화면의 출력결과 응답성에 영향을 줄 수 있습니다.

CSS 와의 관계

- HTML 적용된 style에 대해서 접근, 수정, 추가, 삭제를 할수 있습니다.
- CSS 대한 수정은 HTML 해당 태그에 독립적으로 적용되는 부분과 style 태그에 명시되어 있는 룰을 바꿔 영향받고 있는 태그들을 일괄적으로 적용하는 부분으로 나눌수 있습니다.

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" >
<title>javascript</title>
<script type="text/javascript"
src="http://s1.daumcdn.net/svc/original/U03/cssjs/jquery/jquery-1.9.0.min.js"></script>
<style id="css1st" type="text/css">
.class_div {background:#ccc;}
</style>
</head>
<body>
  <div id="wrap" class="class_div">
    ......
  </div>
  <div id="footer" class="class_div">
    ......
  </div>
  <div class="class_div">
    ......
  </div>
<script type="text/javascript">
  var wrap = document.getElementById('wrap');
  console.log('1 : ' + wrap.style.background); //wrap의 배경색출력
  wrap.style.background = '#ff0000';//wrap 배경색 설정
  console.log('2 : ' + wrap.style.background); //wrap 배경색출력

  var css1st = document.getElementById('css1st');
  console.log('3 : ' + css1st.sheet.rules[0].style.background); //css1st의 첫번째 셀렉터의 룰중 배경색 출력
  css1st.sheet.rules[0].style.background = '#f00'; // css1st의 첫번째 셀렉터의 룰중 배경색 입력
  console.log('4 : ' + css1st.sheet.rules[0].style.background); //css1st의 첫번째 셀렉터의 룰중 배경색 출력
</script>
</body>
</html>
```

FrameWork(jQuery)

- jQuery
  - 존 레식이 2006년 뉴욕 시 바캠프(Barcamp NYC)에서 공식적으로 소개하였다. jQuery는 오늘날 가장 인기있는 자바스크립트 라이브러리 중 하나다.
  - MIT 라이센스와 GNU 일반 공중 사용 허가서v2의 듀얼 라이선스를 가진 자유 오픈 소프트웨어이다.
  - 문법은 코드 보기, 문서 객체 모델 찾기, 애니메이션 만들기, 이벤트 제어, Ajax 개발을 쉽게 할 수 있도록 디자인 되었다. 또한, jQuery는 개발자가 플러그인을 개발할 수 있는 기능을 제공한다.
  - 마이크로소프트와 노키아는 자사 플랫폼에 jQuery를 포함하는 계획을 발표한 바 있다. 마이크로소프트는 비주얼스튜디오의 ASP.NET AJAX 프레임워크와 ASP.NET MVC 프레임워크에 적용했고, 노키아는 자사의 런타임 웹 위젯 개발 플랫폼에 통합하였다. 또한, jQuery는 미디어위키에도 1.16 버전부터 사용되고 있다.
  - ssizle 엔진을 참조한 자체 selector를 사용하며 css3 형식의 selector 사용가능
  - 리턴되는 결과물에는 jQuery 의 Element 및 관련 method들을 사용할 수 있음
  - selector -
    API
  - method -
    jQuery API

BOM, DOM 컨트롤

BOM & DOM

BOM 이란..?

- Browser Object Model

window 객체 및 메서드

- window 객체란 - 클라이언트 측 자바스크립트 프로그램의 전역 객체이며 브라우저의 창 및 프레임을 제어하는 프로퍼티와 메서드들이 존재한다.

Timer Control

JavaScript는 Script언어이기 때문에 코드가 완벽히 동기적으로 동작합니다.
순차적으로 실행된다는 뜻입니다. 위에서 아래로, 왼쪽에서 오른쪽으로 인터프리팅 되고 결과를 바로 보여줍니다.
한 라인의 실행이 완료되지 않으면 다음 라인으로 넘어가지 않죠.

그러나 이런 순차적인 실행방식에 비동기적 실행법을 추가할 수 있습니다.
바로 타이머를 사용하는 방식인데요 타이머를 사용하면 지정된 시간만큼의 딜레이 후 코드가 실행됩니다.

- 기본적인 Timer 사용법

보통 알고있는 Timer의 사용방법은 다음과 같다.

```
window.setTimeout("console.log('bomb!');", 1000);
```

그러나 이는 잘못된 사용방법이다. setTimeout의 첫번째 인자에는 기본적으로 함수를 할당하여야 한다.
그러나, 구형브라우저에서 잘못 사용해 왔기때문에 여전히 저 방식또한 유효하다.
위 코드를 아래와 같이 수정하여 사용한다.

```
function myBomb(){
    console.log('bomb!');
}
window.setTimeout(myBomb, 1000);
```

- 주기적으로 무한 반복하기

setTimeout 메소드를 재귀적으로 사용하면 일정 시간을 주기로 무한 반복하는 함수를 작성할 수 있다.

```
function myBomb(){
    console.log('bomb!');
    window.setTimeout(myBomb, 1000);
}
window.setTimeout(myBomb, 1000);
```

위 처럼 myBomb 함수 내부에서 setTimeout을 계속 호출하게 되면 1000ms, 즉 1초마다 한번씩 myBomb 함수를 호출하게된다.

그러나, JavaScript에서는 이처럼 계속적으로 반복할 수 있는 방법을 제공한다. 바로 setInterval 메소드 이다.

```
function myBomb(){
    console.log('bomb!');
}
window.setInterval(myBomb, 1000);
```

myBomb 함수의 내용은 본연의 내용에 충실하지만 setInterval 메소드를 통해 1초마다 한번씩 무한 반복하는 함수가 되었다.

- 타이머 멈추기

setTimout과 setInterval 메서드는 리턴값으로 unique한 타이머 ID를 반환하는데 이 타이머 ID를 이용해 타이머를 중단 할 수 있다.

```
function myBomb(){
    console.log('bomb!');
}
var timer = window.setInterval(myBomb, 1000);
```

timer 변수에 타이머 ID를 할당하고 다음처럼 타이머를 제거한다.

```
window.clearInterval(timer);
window.clearTimeout(timer);
```

location

- location - Location 객체를 가리키지만 이 프로퍼티에 문자열 값을 지정하면 브라우져는 이 문자열을 URL로 해석하여 URL에 위치한 문서를 불러온다.

  ```
  window.location = 'http://daum.net';
  ```

- location의 method중에 reload(), replace() 가 존재함
  - reload()는 현재 표시된 페이지를 다시 불러오는 데 사용된다.
  - replace(url)는 지정한 URL을 불러와 표시하는데 사용한다. 그러나 replace() 메서드를 호출하는 것은 Window객체의 location 프로퍼티에 URL을 대입하는것과는 다른 의미를 갖인다. replace()를 호출하면 지정된 URL이 브라우저의 최근 열어본 페이지 목록에 새항목으로 추가되는 대신 현재의 항목을 대신하게된다.
    따라서 현재의 문서를 다른 문서로 덮어쓰기 위해 replace()를 호출하면 브라우저의 '뒤로 이동' 버튼을 클릭하여 원래 문서로 돌아가는 것이 불가능하다.

history

- Window 객체의 history 프로퍼티는 브라우저 창에 대한 History 객체를 가리키며 본래 History 객체는 브라우저 창의 최근 열어본 페이지 목록을 최근 방문한 URL들의 배열로써 모델링하도록 설계되었으나 사생활 보고와 시스템 보안의 측면에서 사용자가 과거에 방문한 웹 사이트들의 리스트에 스크립트가 접근하게 하는 것은 부적절하기 때문에 History 객체의 배열 엘리먼트는 스크립트에서 절대로 접근할 수 없도록 되었다.

- method
  - back()

    ```
    history.back();
    ```

  - forward()

```
history.forward();
```

navigator

- 브라우져의 버전, 출력 가능한 데이터 포맷들의 목록 등 웹 브라우져의 전반에 대한 정보를 담고 있다.
- 프로퍼티
  - appName - 웹 브라우저의 간단한 이름이다. IE에서는 "Micorosoft Internet Explorer"이며 파이어폭스 등 네스케이프에서 유래한(모질라 네스케이프같은)다른 브라우저들은 "Netscape"이다.
  - appVersion - 브라우저의 버전 숫자 또는 버전과 관련된 키다 정보를 담은 프로퍼티다. 이프로퍼티는 사용자에게 표시되는 버전 숫자와 항상 일치하지는 않으며, 내부적으로 사용되는 버전 숫자임을 주의해야한다.
  - userAgent - 브라우저가 USER-AGENT HTTP 헤더에 넣어 전송하는 문자열이다. 이 프로퍼티는 보통 appName과 appVersion의 모든 정보를 포함하며 때로는 더 자세한 정보를 추가로 담고 있기도 하다
  - appCodeName - 브라우저의 코드네임이다. 네스케이프는 "Mozilla" 라는 네임을 사용한다. 호환성을 위하여 IE도 역시 같은 코드네임을 사용한다.

```
var browser = "Browser Info\n";
for (var name in navigator) {
  browser += name + " : " + navigator[name] + "\n";
}
console.log(browser);
```

- 사내 javascript userAgent Parser(
  javascript userAgent

)

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript"
src="http://m1.daumcdn.net/svc/original/U03/cssjs/userAgent/userAgent-1.0.14.min.js"></script>
<script type="text/javascript">

var ua = daumtools.userAgent();
console.log(ua); // 현재 브라우져 정보를 담은 object
console.log(ua.ua); // navigator.userAgent
console.log(ua.platform); // 현재 브라우져의 플랫폼 "pc" | "tablet" | "mobile"

</script>
</head>
```

open, close

- 새로운 웹 브라우져의 창을 열고 닫을 수 있는 메서드

```
var w = window.open(
  'http://daum.net', 'daum', 'width=400, height=350, status=yes, resizable=yes'
);
```

- http://www.w3schools.com/jsref/met_win_open.asp
- 해당 팝업윈도우를 닫을때

```
w.close();
```

opener, self, parent

- 새로 열린 브라우져의 창과 frame 등과의 관계를 정의 하는 객체
- opener - window.open 을 통해서 열린 브라우져에서의 자신을 열게한 브라우져 창의 window를 말함

Test page

```
var w = window.open(
  'http://daum.net', 'daum', 'width=400, height=350, status=yes, resizable=yes'
);

function showLog() {
  console.log("I'm opener");
}
```

New Window open page

```
opener.showLog();
```

- self - 현재 페이지의 window 즉 자기 자신을 의미한다.

New Window open page

```
<button type="button" onclick="self.close(); return false;">self 창닫기</button>
<button type="button" onclick="window.close(); return false;">window 창닫기</button>
```

- parent - 현재 페이지의 iframe통해서 내부 페이지를 생성했을경우 해당 iframe의 내부에서 부모 즉 상위의 window를 나타냄.

```
<script type="text/javascript">
function showLog() {
  console.log("I'm parent");
}
</script>
<iframe src="iframePage.html"></iframe>
```

iframePage.html

```
<script type="text/javascript">
parent.showLog();
</script>
```

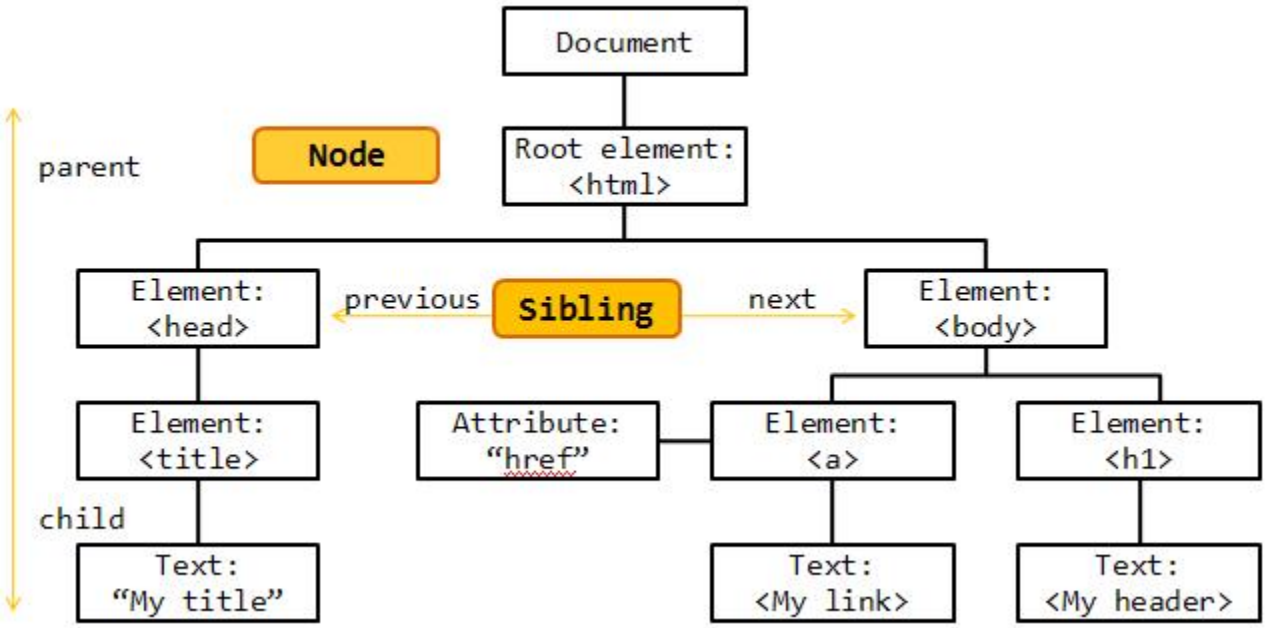
document

- window객체에는 Document객체를 참조하는 document 프로퍼티가(window.document) 있고 이 객체는 문서를 구성하는 프로퍼티와 메서드들로 구성되어 있다.
- window.document == document;

DOM (Document Object Model)

> 문서를 구성하는 객체에 어떻게 접근할 것인가를 정의하는 API다.
> W3C는 근래의 모든 웹 브라우저에서 잘 지원되는 표준 DOM을 정의 하고 있고 클라이언트 측 자바스크립트 프로그래밍은 DOM의 진화라고 해도 과언은 아니다.

DOM Tree 구조



HTML을 표현하는 DOM 은 태크와 원소들을 나타내는 노드와 텍스트 문자열을 표현하는 노드들을 가진다.

```
<html>
  <head>
    <title>My title</title>
  </head>
</html>
<body>
<h1>My header</h1>
<a href="http://daum.net">My link</a>
</body>
</html>
```

노드

> DOM 트리 구조는 다양한 타입의 Node 객체로 구성된 트리로 표현되고, 각 노드에는 트리를 순회하거나 조작하기 위한 프로퍼티와 메서드가 정의 되어 있다.

- 노드 타입

인터페이스	nodeType 상수	nodeType 값
Elementy	Node.ELEMENT_NODE	1
Text	Node.TEXT_NODE	3
Document	Node.DOCUMENT_NODE	9
DocumentFragment	Node.DOCUMENT_FRAGMENT_NODE	11

DOM API 활용

Element 가져오기

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" >
<title>javascript</title>
<script type="text/javascript" src="http://s1.daumcdn.net/svc/original/U03/cssjs/jquery/jquery-1.9.0.min.js"></script>
</head>
<body>
  <div id="container">
    <div id="title">
    </div>
    <div id="contents">
      <ul>
        <li>Tistory</li>
        <li>cafe</li>
        <li>blog</li>
      </ul>
    </div>
  </div>
<body>
</html>
```

- getElementById
  노드에 id 값을 이용해서 해당 Element를 가져오는 방법 (리턴값은 node object)

```
var container = document.getElementById('containter');
var title = document.getElementById('title');
var containerJ = jQuery('#container');

console.log(container);
console.log(title);
```

- getElementsByTagName
  노드의 tag name을 이용해서 Elements를 가져오는 방법 (리턴값은 Array)

```
var body = document.getElementsByTagName('body');
var bodyJ = jQuery('body');
var diveNodes = document.getElementsByTagName('div');

var contents= document.getElementById('contents');
var services = contents.getElementsByTagName('li');
var servicesJ = jQuery('#contents li');

console.log(body);
console.log(diveNodes);
console.log(services);
```

create / append

- createTextNode
  새로운 Text Type의 Node 객체를 생성한다.

```
var newTextNode= document.createTextNode('This is new text node');
```

- createElement
  새로운 Elementy Type의 Node 객체를 생성한다.

```
var newNode = document.createElement('div');
```

```
var newNodeJ = jQuery('<div>This is new text node</div>');
```

- appendChild
  부모 node가 되고자하는 node1에 자신이 넣고자 하는 node2를 자식 node로 넣을 수 있게한다.
  만약 node1에 다른 자식 node가 존재한다면 새로 추가되는 node2는 마지막 자식 node로 추가된다.

```
<body>
  <div id="title"></div>
  <span id="titleText">Title Text</span>
<script type="text/javascript">

var node1 = document.getElementById('title');
var node2 = document.getElementById('titleText');

node1.appendChild(node2);

jQuery('#title').append(jQuery('#titleText'));

</script>
</body>
```

다음 스크립트 실행되면 node1은 node2의 parentNode가 되고 node2는 node1의 childNodes에 속하게 된다.

- createDocumentFragment
  새로운 DocumentFragment Type의 Node 객체를 생성한다.
  DocumentFragment는 문서 내에서 실제로 나타나지는 않고 노드의 집합을 저장하고 저장된 노드들의 임시 저장소 역할을 담당한다.
  DocumentFragment에 저장된 node들은 DocumentFragment에 를 통해 마치 단일 객체인 것처럼 조작할수 있다.

```
var contentsUl = document.getElementById('contentsUl');
var contentArray = ['Tistory', 'cafe', 'blog', 'yozm'];
var contentsFragment = document.createDocumentFragment();
var textNode, nodeLi;

for (var i = 0, l = contentArray.length; i < l; i++) {
  textNode = document.createTextNode(contentArray[i]);
  var nodeLi= document.createElement('li');
  nodeLi.appendChild(textNode);
  contentsFragment.appendChild(nodeLi);
}

contentsUl.appendChild(contentsFragment);
```

Update

각 node들의 프로퍼티의 기존의 설정된 값을을 동적으로 수정가능하다.
(단. read only 만 가능한 설정값은 변경할 수 없으니 수정가능 유무를 판단하여야 한다.)

```
node.id = 'newId';
node.className = 'newClass';
node.checked = 'checked';
node.href = 'http://daum.net';
node.scrollTop = 100;

node.style.color = '#000000'; // update1
node.style.width = '300px';   // update2
node.style.height = '200px';  // update3
node.style.display = 'none';  // update4

jQuery(node).id = 'newId';
jQuery(node).attr('href', 'http://daum.net');
jQuery(node).class = 'newClass';
jQuery(node).css('color', '#000000');
jQuery(node).css({
  color: '#000000',
  display: 'none'
});
```

style과 관려된 속성을 변경할때 주의점으로는 브라우져가 변경을 감지하고 변경된 내용을 화면에 표시할때 update1, 2, 3, 4 와 같이 4번의 변경이 순차 적으로 이루어 짐으로 className 변경을 통해서 한번 style속성을 변경하는것이 바람직하다.

Remove

```
<body>
  <div>
    <div id="tistory">tistory</div>
    <div id="blog">blog</div>
    <div id="cafe">cafe</div>
    <div id="removeNode">removeNode</div>
  </div>
</body>
<script type="text/javascript">
var removeNode = document.getElementById('removeNode');
removeNode.parentNode.removeChild(removeNode);

jQuery('#removeNode').remove();
</script>
```

innerHTML

> innerHTML을 W3C에서 공식적으로 DOM의 일부분으로 승인한 적은 없지만 현대의 모든 브라우저가 지원하는 중요하고도 강력한 프로퍼티다.
> Element에 대해 이 프로퍼티의 값을 요청했을때 얻는것은 해당 Element의 자식을 표현하는 HTML 테스트 문자열이다.
> 이 프로퍼티를 설정하면 브라우저는 문자열을 파싱하기 위해 HTML 파서를 호출하고 해당 Element의 자식을 파서가 반환한 내용으로 교체한다.
> 대체적으로 HTML문서를 innerHTML을 사용하여 문자열로 기술하는 것이 createElement(), appendChild()의 호출을 나열하는 것보다 훨씬 간변하고 좋은 성능을 낸다.

innerHTML 사용법 소개

```
<body>
  <h1>InnerHTML Test</h1>
  <div id="container">
    <div id="title">Title</div>
  </div>
</body>
```

```
document.body.innerHTML = '
   <body>'
 + '<h1>InnerHTML Test</h1>'
 + '<div id="container">'
 + '<div id="title">Title</div>'
 + '</div>'
 + '</body>';
```

DOM API vs innerHTML 성능비교

http://www.quirksmode.org/dom/innerhtml.html 성능비교 테스트 페이지(quirksmode)
http://andrew.hedges.name/experiments/innerhtml/ 비교 테스트하기

어떤걸 써야할까...?

- 상황별로 알맞게 써야한다