Javascript BOM 객체

키워드 : BOM (https://www.w3schools.com/js/js_window.asp)

강의 영상의 첫 시간에 살펴봤듯이, 우리가 작성하는 자바스크립트 코드는 브라우저라는 하나의 프로그램(매개체)을 통해서 동작하고 결과를 사용자에게 돌려주도록 설계합니다. 이러한 브라우저 단위에서 객체를 선언하고 조작하는 방법을 기술한 것이 BOM(Browser Object Model)이라고 할 수 있습니다. BOM 객체는 거의 모든 브라우저에서 동작하도록 설계되어 있습니다.

BOM 객체도 다른 객체와 마찬가지로 속성(Properties)과 메서드(Methods)를 갖습니다. 이번 장에서는 이러한 BOM 객체에 대해서 하나하나 알아보겠습니다.

Window

먼저, window 객체입니다. window 객체는 최상위 객체를 나타냅니다. 여기서 말하는 최상위 객체는 자바스크립트 코드에서 작성되는 모든 것은 window 객체의 하위 요소가 된다는 것을 의미합니다. e.g., 객체, 함수, 변수, etc

앞서 살펴본 DOM 객체도 사실이 window 객체의 하위 속성(Property)입니다.

```
//1번 코드
document.getElementById("id").innertHTML = "안녕!";
//2번 코드
window.document.getElementById("id").innertHTML = "안녕!";
```

위 코드 예시의 1번 코드와 2번 코드는 동일한 코드입니다. 코드에서 나타나듯이, window 객체는 최상위 객체이기 때문에 생략하고 사용할 수 있습니다.

window 객체에서 자주 사용되는 메서드는 다음과 같습니다.

메서드	의미
window.open()	새 창을 엽니다.
window.close()	현재 창을 닫습니다.
window.moveTo()	현재 창으로 이동합니다.
window.resizeTo()	현재 창의 크기를 조절합니다.

이 밖에도 브라우저 창에 크기를 픽셀(Pixel) 단위로 반환하는 innerHeight, innerHeight 등의 속성 값이 존재합니다.

단, IE(Internet Explorer) 하위 버전(5 ~ 8) 같은 경우에는 위 속성 값이 제대로 반환되지 않을 수 있습니다. 이러한 문제점(크로스 브라우징)을 해결하기 위해서 약간 다른 코드를 작성할 수 있습니다.

```
var width = document.body.clientWidth;
var height = document.body.clientHeight;
```

위 코드처럼 너비와 높이 값을 구하면, 하위 버전의 브라우저에서도 너비와 높이 값을 반환받을 수 있습니다. 단, 위에 작성된 모든 코드는 사용자 브라우저의 툴바(Toolbar)와 스크롤바(Toolbar)를 포함하지 않습니다.

Screen

다음은 screen객체입니다. screen객체는 이름 그대로 웹 페이지를 방문하는 사용자의 screen과 관련된 속성 값들을 제공합니다.

screen 객체에서 자주 사용되는 속성은 다음과 같습니다.

속성	의미
screen.width	스크린의 너비를 픽셀 단위로 반환합니다.
screen.height	스크린의 높이를 픽셀 단위로 반환합니다.
screen.colorDepth	하나의 색을 표현하기 위해 사용되는 비트수를 반환합니다. (스크린 해상도 e.g., 24bit)
screen.pixelDepth	하나의 색을 표현하기 위해 사용되는 비트수를 반환합니다. (스크린 해상도 e.g., 24bit)

Location

다음은 Location객체입니다. Location객체는 현재 페이지의 주소(URL)를 얻어오고, 새로운 페이지를 여는 동작을 할 수 있도록 해주는 객체입니다.

location 객체에서 자주 사용되는 속성과 메서드는 다음과 같습니다.

명칭	의미
location.href	현재 페이지의 주소(URL)를 얻어옵니다.
location.pathname	현재 페이지의 폴더 경로와 파일 이름을 얻어옵니다.
location.protocol	현제 페이지의 프로토콜 정보를 얻어옵니다. (e.g., http: or https:)
location.assign("주소")	주소 페이지로 연결합니다.

위 표에서 보이듯이, location 객체는 현재 페이지의 주소와 관련된 정보를 반환해주는 객체입니다.

History

다음은 history객체입니다. history 객체는 마찬가지로 이름에서 느껴지듯이, 사용자의 웹 페이지 방문 정보를 담고 있는 객체라고 할 수 있습니다.

location 객체에서 자주 사용되는 메서드는 다음과 같습니다.

메서드	의미
history.back()	뒤로 가기(현재 페이지 -> 이전 페이지)
history.forward()	앞으로 가기(이전 페이지 -> 기존 페이지)

위 사진에서 보이는 브라우저 왼쪽 상단에 있는 버튼을 생각하시면 이해하기 쉽습니다.

Popup

다음은 popup 관련 객체입니다. popup이라는 객체명이 따로 존재하는 것은 아닙니다. 자바스크립트에서 제공하는 팝업창과 관련 객체는 크게 세 가지입니다.

다음은 팝업창과 관련된 세 가지 메서드입니다.

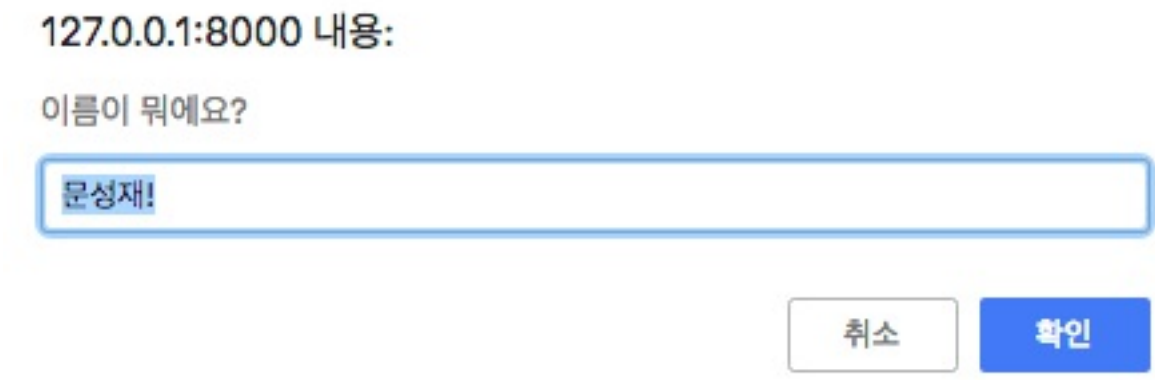
메서드	의미
window.alert("제목")	방문자에게 정보를 제공하기 위해 띄우는 팝업입니다.
window.confirm("제목")	방문자에게 확인 요청을 위해 띄우는 팝업입니다.
window.prompt("제목", "기본값")	방문자에게 입력 값을 요청하기 위해 띄우는 팝업입니다.

아래 코드는 팝업창을 띄우는 예를 보여줍니다.

```
//window 생략 가능
alert("경고!");
```


```
if(confirm("정말로 삭제하시겠습니까?")){
    //실행코드...
}else{
    //실행코드...
}
```


```
//처음 팝업창이 실행되면 기본값으로 입력창이 설정되어 있습니다.
var userName = prompt("이름이 뭐에요?", "문성재!");
```



위 코드와 같이 여러 팝업 창을 이용하면, 웹 페이지 방문자와 비교적 짧은 코드로 상호 작용을 할 수 있는 장점이 있습니다. 만약, 조금 더 디자인을 다듬고 싶다면, 직접 팝업 창을 코드로 작성해서 사용해야 합니다.

Timing

다음은 `timing`객체입니다. 강의에서 살펴봤던 `setInterval` 메서드가 바로 이 타이밍 관련 메서드입니다.

자바스크립트에서 타이머를 만들 수 있는 방법은 아래와 같이 크게 두 가지 방법이 있습니다.

메서드	의미
setTimeout("함수명", 밀리초)	밀리초 단위가 지날 때까지 기다렸다가 함수 실행(한번 실행)
setInterval("함수명", 밀리초)	밀리초 단위가 지날 때까지 기다렸다가 함수 실행(반복 실행)

아래 코드는 두 메서드를 선언하고 종료(Clear)하는 간단한 코드 예시입니다.

```
//1번 코드
var timer1 = setTimeout("function", 5000);
clearTimeout(time1);

//2번 코드
var timer2 = setInterval("function", 5000);
clearInterval(time2);
```

우선, 우리가 아직 함수를 다루지 않았기 때문에, 함수 부분은 "function"으로 대체했습니다. 먼저, 1번 코드와 2번 코드의 공통점은 해당 타이밍 함수를 종료(clear) 하기 위해서 변수를 선언하고, 해당 변수를 종료하는 메서드에 할당했다는 공통점이 있습니다. 반대로, 어떤 차이점이 존재하는지에 대해 코드를 각각 살펴보도록 하겠습니다.

- 1번 코드의 경우 setTimeout 메서드를 이용해 함수를 호출하여 5초 동안 기다렸다가 한 번만 실행하는 코드입니다. 만약, 코드가 아직 실행되지 않았다면, 함수가 호출되지 않습니다.
- 2번 코드의 경우 setInterval 메서드를 이용해 함수를 호출하여 5초 간격으로 계속해서 함수를 실행하는 코드입니다. 만약, setInterval 메서드가 이미 실행된 시점에서 clearInterval을 호출되면 중간에 함수 반복이 멈추게 됩니다.

특정 시간 이후에 한 번만 함수를 호출해서 실행할 땐 setTimeout을 사용하고, 계속해서 일정 간격으로 함수를 동작하도록 할 땐 setInterval을 이용합니다. 각각의 메서드를 종료시킬 땐, 각각 순서대로 clearTimeout("변수"), clearIntval("변수")에 대응시켜 종료합니다.

Cookie

다음은 cookie객체입니다. 쿠키(Cookie)는 위에서 다룬 내용 보다 조금 더 깊은 학습이 필요한 난이도 있는 부분입니다. 여기선 간단히 쿠키가 어떤 것인지에 대해서 알아보도록 하겠습니다.

우선, 쿠키를 한 마디로 정의하면 다음과 같습니다.

> 웹 페이지를 방문한 사용자의 정보를 담고있는 파일

아마 한 번쯤은 브라우저 고급 탭에서 '쿠키 삭제'를 진행 해보신 적이 있으실 겁니다. 우리가 어떤 웹 페이지를 방문하고, 해당 웹 페이지가 닫히는(종료되는) 시점에 우리의 정보도 함께 사라집니다. 만약, 방문자의 정보를 따로 기록으로 남겨둘 필요성이 있다면 쿠키를 사용할 수 있습니다.

쿠키 정보는 브라우저를 통해 웹 페이지를 실행한 방문자의 컴퓨터에 저장되며, 대표적인 예로 "오늘 하루동안 이 창을 열지 않음"를 예로 들 수 있습니다. 아래는 쿠키 정보에 이름을 담는 간단한 코드 예시입니다.

```
document.cookie = "user=seongjae; expires=Date();, path=/;";
```

위 코드에서 나타나듯이, 쿠기 정보는 데이터, 유효 기간, 쿠기 정보가 저장될 로컬 경로라는 세 가지로 이루어집니다.

단, 보안상에 이유로 중요한 정보(로그인 정보)와 같은 경우는 서버 측 코드로 관리하는 세션(Session)이라는 것을 이용하는 것을 권장하기도 합니다.

meta tag

이번 장에선 브라우저에 대한 내용을 다뤘습니다. 브라우저에 대한 내용과 더불어 <meta>태그에 대해서 조금 더 정리해보도록 하겠습니다. 우선, 우리는 HTML 템플릿을 작성할 때 메타 태그를 잠시 언급했었습니다. 메타 태그의 charset 속성을 통해 웹 페이지 인코딩 방식을 "UTF-8"로 설정했습니다.

우리가 사용했던 인코딩 설정뿐만 아니라, 메타 태그는 이름에서 느껴지듯 여러 메타 데이터(Meta data)라고 불리는 웹 페이지의 기본 정보를 표현할 수 있는 코드를 작성할 수 있습니다. 우선, 메타 태그는 두 가지 주요 특징이 있습니다.

- <meta> 태그는 항상 <head> 요소 안에 있습니다.
- 메타 데이터는 항상 name/value 쌍으로 전달됩니다.

다음으로 아래 코드는 웬만한 웹 페이지에선 거의 무조건 사용되는 메타 태그입니다. 결론적으로, HTML5에서는 아래와 같이 작성하면, 모바일 화면 크기와 일반 컴퓨터의 화면 크기 차이를 자동으로 인지하여 콘텐츠의 크기를 조절하도록 해줍니다.

```
<meta name="viewport" content="width=device-width, initial-scale=1.0, shrink-to-fit=no">
```

하나하나 살펴보도록 하겠습니다.

- name="viewport": 페이지의 크기와 배율을 제어하는 방법에 대해 나타냅니다.
- content="width=device-width, initial-scale=1.0, shrink-to-fit=no": width = device-width 부분은 페이지의 너비를 장치의 화면 너비에 따라 설정합니다
- content="width=device-width, initial-scale=1.0, shrink-to-fit=no": initial-scale = 1.0 부분은 브라우저가 페이지를 처음로드 할 때 초기 줌 레벨을 설정합니다. (0.0 ~ 10.0)까지 설정할 수 있습니다.
- content="width=device-width, initial-scale=1.0, shrink-to-fit=no": shrink-to-fit=no 부분은 사파리(Safari) 브라우저와의 호환을 위해 작성되는 코드입니다.

제목!

내용~ 냥펀치~

제목!

냐옹~ 냥펀치~

위 두 개의 사진의 코드는 메타 태그를 제외하고 모두 동일하게 작성되었고, 크롬 개발자 도구의 Toggle Device Toolbar 기능을 이용해 모바일 크기(430x716)를 확인한 결과입니다. 왼쪽은 메타 태그가 작성되지 않은 코드, 오른쪽은 메타 태그가 작성된 코드입니다. 감이 바로 오시죠? :)

이 밖에, maximum-scale, minimum-scale, user-scalable 등도 설정할 수 있으며 각각의 역할에 대해서는 직접 코드를 작성해보면 어렵지 않게 익힐 수 있을 겁니다!

이 메타태그를 통해 할 수 있는 일은 이것 뿐만이 아닙니다. 웹 페이지의 각종 기본 정보를 설정할 수 있습니다. 아래는 메타 태그의 주요 내용입니다.

선언방법	의미
`<meta name="keywords" content="html, css, js, python">`	웹 페이지의 주요 키워드를 나타냅니다.
`<meta name="description" content="Web client developemnt, Python crawling">`	웹 페이지의 특징을 나타냅니다.
`<meta name="author" content="Seongjae Moon.">`	웹 페이지의 작성자를 나타냅니다.

이게 어디에 사용될까요? 우리가 일반적으로 웹 페이지를 개발하고, 배포하여 누군가가 웹 페이지를 볼 수 있도록 하려면, 여러 사람들의 검색 키워드에 따라 검색 엔진에 노출되어야 합니다. 그 부분은 조금 더 복잡한 이야기이므로, 단순히 생각해서 우리의 웹 페이지의 대한 정보를 잘 표현할 수 있도록 마치 명찰처럼 메타 태그에 작성해두는 것이라고 생각하면 쉬울 것 같네요. :)

이 메타 태그의 정보를 담는 것을 확장시켜 페이스북이나 트위터와 같은 대형 IT 기업에서 개발한 여러 메타 정보 전용 프로토콜이 존재합니다. 특히, 페이스북에서 개발한 오픈 그래프(Open Graph)라는 프로토콜이 대중적으로 사용되고 있습니다. 이 프로토콜은 여러 메타 태그의 정보를 모든 웹 페이지에서 통일성 있게 사용할 수 있도록 개발되었습니다. 이 메타 태그의 활용을 확인할 수 있는 대표적인 예로 스마트폰 메신저 애플리케이션에서 URL 주소 첨부 시 보이는 미리보기를 들 수 있습니다.

예를 들면, 아래와 같습니다.

```
<meta name="msapplication-TileColor" content="#ffffff">
<meta name="msapplication-TileImage" content="/assets/icons/ms-icon-144x144.png">
<meta name="theme-color" content="#ffffff">
<!-- Facebook OGP cards -->
<meta property="og:description" content="Android, Git, Algorithm, IT, Firebase, Web, Database, My Story, etc.
">
<meta property="og:url" content="https://seongjaemoon.github.io">
<meta property="og:site_name" content="슈퍼문의 개발일지">
<meta property="og:title" content="슈퍼문의 개발일지">
<meta property="og:type" content="website">
<meta property="og:image" content="https://seongjaemoon.github.io/assets/logo.png">
<meta property="og:image:type" content="image/png">
<meta property="og:image:width" content="612">
<meta property="og:image:height" content="605">
<!-- Twitter: card tags -->
<meta name="twitter:card" content="summary">
<meta name="twitter:title" content="슈퍼문의 개발일지">
<meta name="twitter:description" content="Android, Git, Algorithm, IT, Firebase, Web, Database, My Story, etc.
">
<meta name="twitter:image" content="https://seongjaemoon.github.io/assets/logo.png">
<meta name="twitter:url" content="https://seongjaemoon.github.io">
```

위 사진은 저의 블로그 주소를 복사해서 메신저 앱에 붙여 넣기 한 결과입니다. 붙여 넣기를 하면, 메신저 앱에서 해당 문자열은 주소(URL)라는 것을 인지하고, 자동으로 하이퍼링크를 걸어줌과 동시에 메타 태그의 정보를 읽어와 표시해주는 것을 확인할 수 있습니다. 기존의 메타 태그와 선언 방법의 차이만 있을 뿐, 표현되어 있는 내용 자체는 큰 차이가 없기 때문에 어렵지 않게 이해하실 거라 생각합니다. :)

Open Graph에 대한 더 자세한 내용은 여기 (http://ogp.me/)에서 확인 가능하고, 메타 태그와 관련된 더 자세한 내용은 여기 (https://www.w3schools.com/tags/tag_meta.asp)를 참고해주세요! :)

우선, 이번 장에서 window 객체의 하위 객체인 navigator 객체는 따로 다루지 않았습니다. 그 이유는 여러 가지가 있지만, 위에 다룬 내용은 전반적으로 거의 모든 브라우저에서 지원되는 내용입니다. 하지만, navigator 객체는 전반적으로 그렇지 않은 부분이 많기 때문에, 혼란을 가중시킬 우려가 있어 포함하지 않았습니다.

간단하게 navigator 객체에 대해서 이야기하면, 웹 페이지에 방문한 사용자의 브라우저 버전, 명칭, 엔진, 플랫폼 등을 반환해주는 객체입니다.

이 세상엔 굉장히 많은 브라우저가 존재하고, 각종 이유로 해당 브라우저의 정보가 구분이 안 되는 경우가 많습니다. 개발 시에 이러한 정보를 반환받길 원하는 이유는 보통 브라우저 크기에 따른 웹 페이지의 화면 변화 때문입니다. `e.g.,` 스마트폰의 화면 크기, 태블릿의 화면 크기, 랩탑의 화면 크기, `etc`

이는, 먼저 살펴본 `<meta>`태그라던지 부트스트랩 혹은, `css` 고급 기술인 `media query` 등을 이용해 해결할 수 있습니다. 만약, 내용이 더 궁금하시다면 여기 (https://www.w3schools.com/js/js_window_navigator.asp)를 참고해주세요.

우리는 자바스크립트 객체를 시작으로 BOM 객체에 내용까지 다뤄보았습니다. 객체에 대해 이야기할 때 다뤄야 하는 내용들 중 많은 부분을 강의에서 다루지 못했습니다. 사실, 우리가 공부한 내용보다 더 많은 내용이 존재합니다. 하지만, 지금까지 다룬 내용에 대해 어느 정도 이해가 가셨다면, 직접 코드를 작성하며 금방 익숙해지실 수 있을 거라 생각합니다. (원래 처음이 어렵지요^^)

객체라는 꽤나 어렵고, 생소하고, 지루하고, 재미없는? 내용이 이제 조금은 여러분들 곁에 가까이 느껴지길 희망합니다! :)