Document

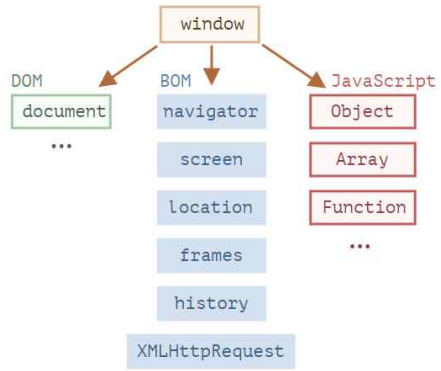
☑ Browser Environments, Specs

JavaScript language는 처음에 Web Browser 용으로 만들어졌다. 그 이후로 그것은 진화하여 많은 용도와 platform을 가진 language가 되었다.

Platform은 JavaScript를 실행할 수 있는 경우 Browser, Web Server 또는 다른 host트, 심지어 "smart" 커피 머신일 수 있다. 각각은 platform 별 기능을 제공한다. JavaScript specification은 이를 host environment(호스트 환경) 이라고 한다.

Host Environment은 language core에 추가로 자체 object 및 function을 제공한다. Web Browser는 Web Page를 제어하는 수단을 제공한다. Node.js는 Server-side 기능 등을 제공한다.

다음은 JavaScript가 Web Browser에서 실행될 때의 그림이다.



window 라는 "root" object가 있다. 두 가지 역할이 있다.
첫째는 JavaScript code의 global object 이다. 둘째는 "Browser window"를 나타내며 이를 제어하는 방법을 제공한다.

☑ attrubute와 property

Browser는 page를 load 할 때 HTML을 "read"(즉, "parsing") DOM object를 생성한다. element node의 경우 대부분의 표준 HTML attribute는 자동으로 DOM object의 property가 된다.

예를 들어 tag가 <body id="page">이면, DOM object에는 body.id="page"가 있다.

그러나 attribute-property mapping은 1:1이 아니다. 이 두 가지 개념을 분리하는 데 주의를 기울이고,

그것들을 어떻게 사용하는지, 언제 같을 때, 다를 때를 볼 것이다.

❖ DOM property

이미 built-in DOM property를 보았다. 많이 있다. 그러나 기술적으로 아무도 우리를 제한하지 않으며 충분하지 않으면 우리 자신을 추가할 수 있다. DOM node는 일반 JavaScript object 이다. 우리는 그것들을 변경할 수 있다.

예를 들어 document.body에 새로운 property를 생성해 보자.

```
document.body.myData = {
  name: 'Semyung',
  title: 'SmartIT'
};

alert(document.body.myData.title); // SmartIT
```

method도 하나 추가해 보자.

```
document.body.sayTagName = function() {
  alert(this.tagName);
};

document.body.sayTagName(); // BODY (sayTagName의 'this'엔 document.body가 저장된다.)
```

Element.prototype 같은 built-in prototype을 수정해 모든 element node에서 이 method를 사용하게 할 수도 있다.

```
Element.prototype.sayHi = function() {
  alert(`Hello, I'm ${this.tagName}`);
};

document.documentElement.sayHi();          // Hello, I'm HTML
document.body.sayHi();                     // Hello, I'm BODY
```

따라서 DOM property와 method는 일반 JavaScript object와 동일하게 작동한다:

☞ 어떤 value(값)이든 가질 수 있다.

☞ 대소문자를 구분한다. (elem.NoDeTyPe가 아닌 elem.nodeType 으로 작성)

❖ HTML attribute

HTML에서 tag는 attribute를 가질 수 있다. Browser가 HTML을 구문 분석하여 tag 용 DOM object를 생성할 때, standard attribute(표준 속성)을 인식하고 이 attribute에서 DOM property를 생성한다.

따라서 element에 id 또는 다른 standard attribute(표준 속성)이 있으면, 해당 property가 생성된다. 그러나 attribute가 non-standard(비표준)인 경우에는 발생하지 않는다.

예를 들어:

```
<body id="test" something="non-standard">
  <script>
    alert(document.body.id);       // test
    // 비표준 속성은 property로 전환되지 않습니다.
    alert(document.body.something); // undefined
  </script>
</body>
```

한 element에 대한 standard attribute(표준 속성)은 다른 element에 대해 알려지지 않을 수 있다. 예를 들어, "type"은 <input>(HTMLInputElement)에 대한 표준이지만, <body>(HTMLBodyElement)에 대해서는 표준이 아니다. standard attribute(표준 속성)은 해당 element class에 대한 사양에 설명되어 있다.

```
<body id="body" type="...">
  <input id="input" type="text">
  <script>
    alert(input.type); // text
    alert(body.type); // undefined: 비표준이기 때문에 DOM property가 생성되지 않았다.
  </script>
</body>
```

따라서 attribute가 비표준이면 해당 attribute에 대한 DOM property는 없다. 그러나 모든 attribute는 다음 방법을 사용하여 액세스할 수 있다.

- ☞ elem.hasAttribute(name) : 존재 여부를 확인한다.
- ☞ elem.getAttribute(name) : value(값)을 가져온다.
- ☞ elem.setAttribute(name, value) : value(값)을 설정한다.
- ☞ elem.removeAttribute(name) : attribute를 제거한다.

이 method는 HTML로 작성된 것과 정확히 작동한다. 또한 elem.attributes를 사용하여 모든 attribute를 읽을 수 있다. 이 attribute는 name 및 value property가 있는 built-in Attr class에 속하는 object collection 이다.

다음은 비표준 property를 읽는 데모이다. 비표준 property를 읽는 예시를 살펴보자.

```
<body something="non-standard">
  <script>
    alert(document.body.getAttribute('something')); // 비표준 속성에 접근
  </script>
</body>
```

HTML attribute는 아래와 같은 특징을 보인다.

☞ 대·소문자를 가리지 않는다. id와 ID는 동일하다.
☞ value(값)은 항상 string 이다.

HTML attribute를 어떻게 다루는지에 대한 예시를 살펴보자.

```
<body>
  <div id="elem" about="Elephant"></div>

  <script>
    alert( elem.getAttribute('About') );   // (1) 'Elephant', attribute 읽기

    elem.setAttribute('Test', 123);        // (2) attribute 추가하기

    alert( elem.outerHTML );               // (3) 추가된 attribute 확인하기

    for (let attr of elem.attributes) {    // (4) attribute 전체 나열하기
      alert( `${attr.name} = ${attr.value}` );
    }
  </script>
</body>
```

주의해서 볼 점은 다음과 같다.

① getAttribute('About') : 여기에서 첫 글자는 대문자이고 HTML에서는 모두 소문자이다. 그러나 그것은 중요하지 않다. attribute name은 대소문자를 구분하지 않는다.
② attribute에 무엇이든 할당할 수 있지만 string이 된다. 그래서 여기에 value(값)으로 "123"이 있다.
③ 우리가 설정한 attribte을 포함한 모든 ttribute은 outerHTML에서 볼 수 있다.
④ attribute collection은 iterable(반복) 가능하며 element의 모든 attribute(표준 및 비표준)를 name 및 value property가 있는 object로 포함한다.

❖ property-attribute synchronization

standard attribute(표준 속성)이 변하면 대응하는 property는 자동으로 갱신된다. 몇몇 경우를 제외하고 property가 변하면 attribute 역시 마찬가지로 갱신된다.

아래 예시에서 attribute id가 수정되면 이에 대응하는 property가 갱신되는 것을 확인할 수 있다. 그 반대도 마찬가지 이다.

```
<input>

<script>
  let input = document.querySelector('input');

  // attribute 추가 => property 갱신
  input.setAttribute('id', 'id');
  alert(input.id);        // id (갱신)

  // property 변경 => attribute 갱신
  input.id = 'newId';
  alert(input.getAttribute('id')); // newId (갱신)
</script>
```

그런데 아래 예시의 input.value 처럼 동기화가 attribute에서 property 방향으로만 일어나는 예외상황도 존재한다.

```
<input>

<script>
  let input = document.querySelector('input');

  // attribute 추가 => property 갱신
  input.setAttribute('value', 'text');
  alert(input.value);     // text (갱신)

  // property를 변경해도 attribute가 갱신되지 않음
  input.value = 'newValue';
  alert(input.getAttribute('value')); // text (갱신 안됨!)
</script>
```

위 예시에선 다음을 확인할 수 있다.

☞ attribute value를 수정하면 property도 수정된다.
☞ 하지만 property를 수정해도 attribute는 수정되지 않는다.

이런 'feature(특징)'은 유용하게 사용될 수도 있다. user의 어떤 행동 때문에 value가 수정되었는데 수정 전의 'original(원래)' value로 복구하고 싶은 경우, attribute에 저장된 value를 가지고 오면 되기 때문이다.

❖ DOM property의 type

DOM property는 항상 string이 아니다. checkbox에 사용되는 input.checked property의 경우 boolean 값을 가진다.

```
<input id="input" type="checkbox" checked> checkbox

<script>
  alert(input.getAttribute('checked'));    // attribute value : 빈 string
  alert(input.checked);                    // property value : true
</script>
```

몇 가지 다른 예를 살펴보다. style attribute의 경우 string 이지만, style property는 object 이다.

```
<div id="div" style="color:red;font-size:120%">Hello</div>

<script>
  // string
  alert(div.getAttribute('style')); // color:red;font-size:120%

  // object
  alert(div.style);               // [object CSSStyleDeclaration]
  alert(div.style.color);         // red
</script>
```

대부분의 property의 값은 string 이다.

아주 드물게 DOM property type이 string인 경우에도 attribute와 다를 수 있다. 예를 들어, attribute에 relative URL 이거나 #hash 만 포함된 경우에도 href DOM property는 항상 전체 URL 이다.

예시:

```
<a id="a" href="#hello">link</a>
<script>
  // attribute
  alert(a.getAttribute('href')); // #hello

  // property
```

```
  alert(a.href ); // http://site.com/page#hello 형식의 전체 URL
</script>
```

HTML에 작성된 것과 정확히 같은 href 또는 다른 attribute가 필요한 경우 getAttribute를 사용할 수 있다.

❖ Non-standard attribute, dataset

HTML을 작성할 때 우리는 대부분 표준 속성을 사용한다. 하지만 표준이 아닌 속성도 존재한다. 이런 비표준이 유용한 지 아닌지, 그리고 어떤 경우에 비표준 속성을 사용해야 하는지 알아본다.

non-standard attribute(비표준 속성)은 사용자가 직접 지정한 데이터를 HTML에서 JavaScript로 넘기고 싶은 경우나 JavaScript를 사용해 조작할 HTML element를 표시하기 위해 사용할 수 있다.

예시:

```
<!-- 이름(name) 정보를 보여주는 div 라고 표시 -->
<div show-info="name"></div>
<!-- 나이(age) 정보를 보여주는 div 라고 표시 -->
<div show-info="age"></div>

<script>
  // 표시한 element를 찾고, 그 자리에 원하는 정보를 보여주는 코드
  let user = {
    name: "Pete",
    age: 25
  };

  for(let div of document.querySelectorAll('[show-info]')) {
    // 원하는 정보를 field 값에 입력해 줌
    let field = div.getAttribute('show-info');
    div.innerHTML = user[field]; // 첫 번째 Pet는 "name"으로, 25는 "age"로
  }
</script>
```

비표준 속성은 element에 style을 적용할 때 사용되기도 한다.

아래 예시에선 주문 상태(order state)를 나타내는 custom attribute order-state를 사용해 주문 상태(order state)에 따라 style을 변경한다.

```
<style>
```

```
  /* style이 custom attribute 'order-state'에 따라 변한다. */
  .order[order-state="new"] {
    color: green;
  }

  .order[order-state="pending"] {
    color: blue;
  }

  .order[order-state="canceled"] {
    color: red;
  }
</style>

<div class="order" order-state="new">
  A new order.
</div>

<div class="order" order-state="pending">
  A pending order.
</div>

<div class="order" order-state="canceled">
  A canceled order.
</div>
```

.order-state-new, .order-state-pending, .order-state-canceled와 같은 class를 사용하는 것보다 attribute를 사용하는 것이 더 나은 이유는 무엇일까?
이유는 attribute은 class보다 다루기 편리하다는 점 때문이다. attribute의 state는 아래와 같이 쉽게 변경할 수 있다.

```
// new class를 adding(추가)하거나 removing(지우는 것)보다 더 쉽게 상태(state)를 바꿀 수 있다
div.setAttribute('order-state', 'canceled');
```

물론 custom attribute(커스텀 속성)에도 문제가 발생할 수 있다. non-standard attribute(비표준 속성)을 사용해 코드를 작성했는데 나중에 그 attribute가 표준으로 등록되게 되면 문제가 발생한다. HTML은 살아있는 언어이기 때문에 개발자들의 요구를 반영하기 위해 지속해서 발전한다. 따라서 이런 경우 예기치 못한 부작용이 발생할 수 있다.

이런 충돌 상황을 방지하기 위한 속성인 data-* 가 있다.

'data-'로 시작하는 attribute 전체는 개발자가 용도에 맞게 사용하도록 별도로 예약된다. dataset property를 사용하면 이 속성에 접근할 수 있다.

element인 elem에 "data-about" 인 이름의 attribute가 있다면, elem.dataset.about을 사용해 그 값을 얻을 수 있다.

[예시]

```
<body data-about="Elephants">
<script>
  alert(document.body.dataset.about); // Elephants
</script>
```

data-order-state 같이 여러 단어로 구성된 attribute는 camel-cased를 사용해 dataset.orderState으로 변환된다.

이런 특징을 사용해 주문 상태에 관한 예시를 다시 작성하면 다음과 같다.

```
<style>
  .order[data-order-state="new"] {
    color: green;
  }

  .order[data-order-state="pending"] {
    color: blue;
  }

  .order[data-order-state="canceled"] {
    color: red;
  }
</style>

<div id="order" class="order" data-order-state="new">
  A new order.
</div>

<script>
  // 읽기
  alert(order.dataset.orderState);         // new

  // 수정하기
  order.dataset.orderState = "pending";    // (*)
</script>
```

data-* attribute는 custom data(커스텀 데이터)를 안전하고 유효하게 전달해준다.
data-* 속성을 사용하면 읽기 뿐만 아니라 수정도 가능하다는 점에 유의한다. attribute이 수정되면 CSS가 해당 view를 자동으로 업데이트해 준다. 위 예시에서 (*)로 표시한 줄에서 색이 파란색으로 바뀐다.

☑ 요약

☞ attribute : HTML 안에 쓰임
☞ property : DOM object 안에 쓰임

❖ 비교표:

	property	attribute
Type	Any value, standard property의 type은 spec에 설명되어 있음	string
Name	case-sensitive	not case-sensitive

attribute와 함께 쓰이는 method:

☞ elem.hasAttribute(name) : 속성 존재 여부 확인
☞ elem.getAttribute(name) : attribute value를 가져옴
☞ elem.setAttribute(name, value) : attribute value을 변경함
☞ elem.removeAttribute(name) : attribute value을 지움
☞ elem.attributes : attributee의 collection을 반환함

대부분의 경우 DOM property를 사용하는 것이 좋다. DOM property가 적합하지 않을 때, 정확히 attribute가 필요할 때만 attribute를 참조해야 한다. 예를 들면 다음과 같다.

☞ non-standard attribute(비표준 속성)가 필요하다. 그러나 data- 로 시작한다면 dataset을 사용해야 한다.
☞ 우리는 HTML에서 "기록된 대로" value를 읽고 싶다. DOM property의 value는 다를 수 있다. 예를 들어 href property는 항상 전체 URL이고, 우리는 "original" value를 원할 수 있다.

☑ 과제

❖ attribute 가져오기

★ 중요도: 5

document에서 data-widget-name이라는 attribute를 가진 element를 찾고, 해당 element의 value를 읽는 코드를 작성해 보자.

```
<!DOCTYPE html>
<html>
<body>

  <div data-widget-name="menu">Choose the genre</div>

  <script>
    /* your code */
  </script>
</body>
</html>
```

[해답]

```
<!DOCTYPE html>
<html>
<body>

  <div data-widget-name="menu">Choose the genre</div>

  <script>
    // getting it
    let elem = document.querySelector('[data-widget-name]');

    // reading the value
    alert(elem.dataset.widgetName);
    // or
    alert(elem.getAttribute('data-widget-name'));
  </script>
</body>
</html>
```

❖ external link를 red로 만들기

★ 중요도: 3

style property를 변경해 모든 external link를 빨간색으로 만들어 보자.

external link가 되기 위한 조건은 아래와 같다.

☞ href에 ://가 포함되어 있어야 한다.

☞ 그러나 http://internal.com 으로 시작하지 않아야 한다.

[예시]

```
<a name="list">the list</a>
<ul>
  <li><a href="http://google.com">http://google.com</a></li>
  <li><a href="/tutorial">/tutorial.html</a></li>
  <li><a href="local/path">local/path</a></li>
  <li><a href="ftp://ftp.com/my.zip">ftp://ftp.com/my.zip</a></li>
  <li><a href="http://nodejs.org">http://nodejs.org</a></li>
  <li><a href="http://internal.com/test">http://internal.com/test</a></li>
</ul>

<script>
  // setting style for a single link
  let link = document.querySelector('a');
  link.style.color = 'red';
</script>
```

[결과]

The list:

- http://google.com
- /tutorial.html
- local/path
- ftp://ftp.com/my.zip
- http://nodejs.org
- http://internal.com/test

[해답]

먼저 모든 external link reference를 찾아야 한다. 두 가지 방법이 있다.

첫 번째는 document.querySelectorAll('a')을 사용하여 모든 링크를 찾은 다음 필요한 것을 필터링하는 것이다.

```
let links = document.querySelectorAll('a');

for (let link of links) {
  let href = link.getAttribute('href');
  if (!href) continue;    // no attribute

  if (!href.includes('://')) continue; // no protocol

  if (href.startsWith('http://internal.com')) continue; // internal

  link.style.color = 'red';
}
```

참고: link.getAttribute('href')를 사용한다. HTML의 value가 필요하기 때문에 link.href가 아니다.

...또 다른 간단한 방법은 CSS selector에 check(검사)를 추가하는 것이다.

```
// look for all links that have :// in href
// but href doesn't start with http://internal.com
let selector = 'a[href*="://"]:not([href^="http://internal.com"])';
let links = document.querySelectorAll(selector);

links.forEach(link => link.style.color = 'red');
```