5강

클라이언트 사이드 자바스크립트

WHATEVER YOU WANT, MAKE IT REAL.

강사 정길용

LIKE LION

{ DOM

Event

Ajax 프로그래밍

BOM

Web APIs }

웹 브라우저에서 실행되는 자바스크립트 환경

- ECMAScript: 자바스크립트 언어에 대한 표준
  - https://ecma-international.org/publications-and-standards/standards/ecma-262

- DOM(Document Object Model): 웹페이지 제어를 위한 표준
  - https://dom.spec.whatwg.org
  - window.document 등
  - Event

- BOM(Browser Object Model): 웹페이지 외부의 브라우저 기능 제어를 위한 표준
  - HTML 표준: https://html.spec.whatwg.org
  - window.navigator: 브라우저와 운영체제에 대한 정보 제공
  - window.location: 현재 페이지의 URL에 대한 제어(읽기, 수정)
  - window.history: 브라우저의 과거 페이지 이동 정보에 대한 제어(읽기, 수정)
  - alert, setTimeout 등

- Web APIs: 브라우저가 제공하는 웹 기능을 위한 표준
  - https://spec.whatwg.org
  - XMLHttpRequest: 서버와 통신에 사용되는 객체(Ajax)
  - Web Storage, Notifications API, WebSocket 등

참고: https://ko.javascript.info/browser-environment

▶ DOM(Document Object Model)

- 1998년 10월 W3C에서 DOM 레벨 1 발표
- 2000년 11월 DOM 레벨 2 발표
- 2004년 04월 DOM 레벨 3 발표
- 2015년 11월 DOM 레벨 4 발표
- 2019년 05월 W3C에서 WHATWG으로 이관
    - https://dom.spec.whatwg.org

- HTML, XML 등의 문서를 제어하기 위한 방법을 정의

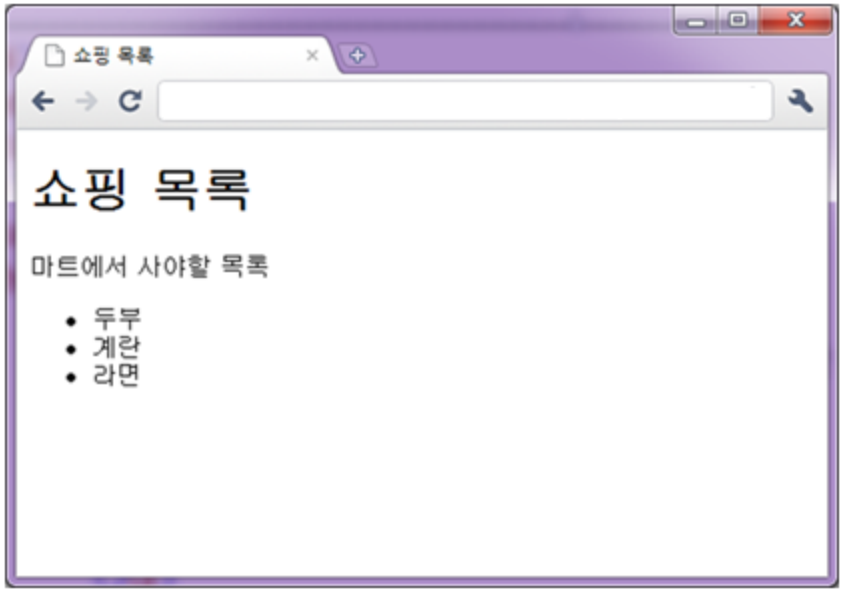
- 텍스트 기반의 HTML, XML 문서를 일정한 규칙에 의해 객체로 만들고 이를 이용하여 문서를 제어(특정 요소를 추출, 삽입, 삭제, 이동 등)

참고: https://ko.javascript.info/browser-environment

LIKE LION

ex05-01.html

```
<html>
<head>
<title>쇼핑목록</title>
</head>
<body>
  <h1>쇼핑 목록</h1>
  <p>마트에서 사야할 목록</p>
  <ul id="purchases" class="list">
    <li>두부</li>
    <li>계란</li>
    <li>라면</li>
  </ul>
</body>
</html>
```



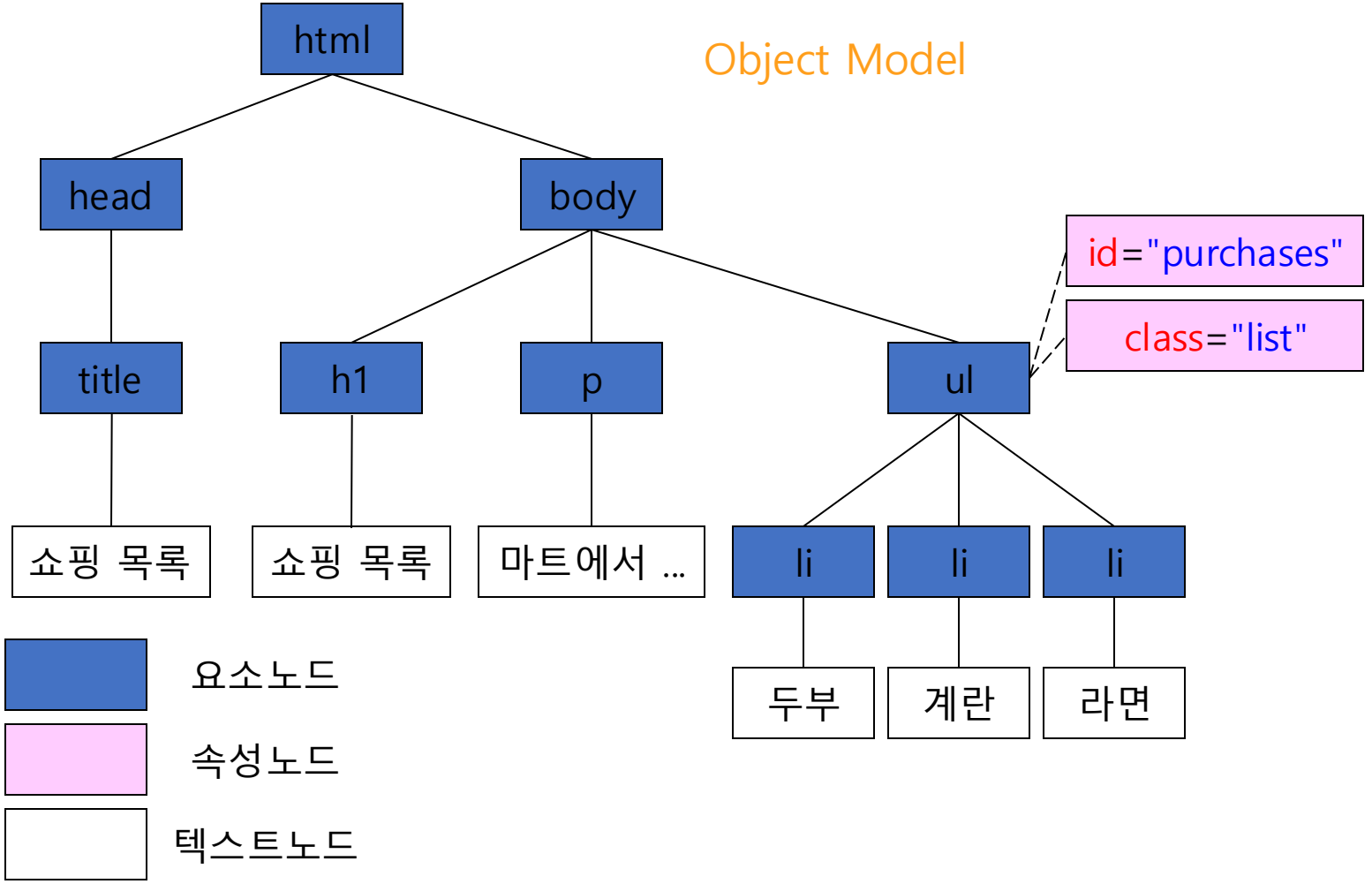
참고: https://ko.javascript.info/dom-nodes

LIKE LION

Document

```
<html>
<head>
<title>쇼핑목록</title>
</head>
<body>
  <h1>쇼핑 목록</h1>
  <p>마트에서 사야할 목록</p>
  <ul id="purchases" class="list">
    <li>두부</li>
    <li>계란</li>
    <li>라면</li>
  </ul>
</body>
</html>
```

Object Model



참고: https://ko.javascript.info/dom-nodes

노드(Node)

- DOM 트리구조는 모든 구성원이 각각의 객체로 인식되며 이러한 객체 하나하나를 노드라고 함

노드의 종류(주로 사용되는 노드)

- 문서노드(document node)
- 요소노드(element node)
- 속성노드(attribute node)
- 텍스트노드(text node)
- ……

종류	설명	nodeName	nodeType	nodeValue
문서 노드	문서	#document	9	null
요소 노드	태그	태그의 이름	1	null
속성 노드	요소의 속성	속성의 이름	2	속성의 값
텍스트 노드	요소의 내용	#text	3	문자열 값

참고: https://ko.javascript.info/dom-nodes

LIKE LION

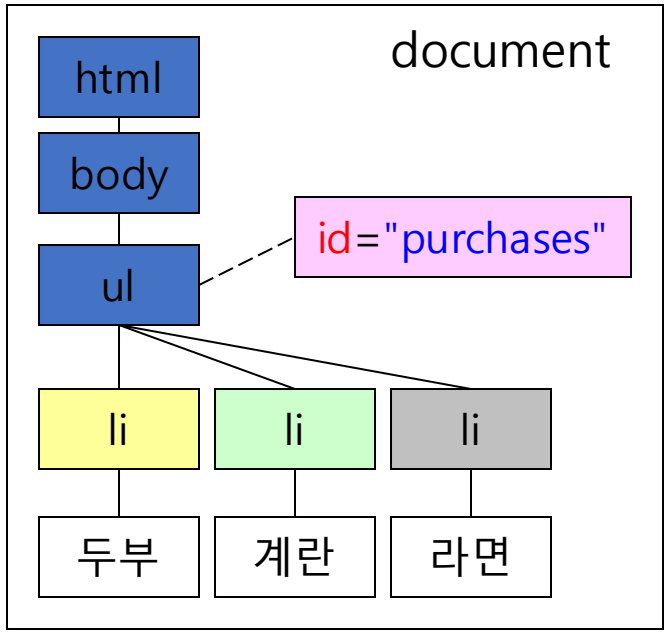
ex05-01.html

▶ 태그의 id를 이용하여 노드 찾기

- document.getElementById(id)
  - id 속성값에 해당하는 노드객체를 반환

```
const purchases = document.getElementById("purchases");
```

ul



참고: https://ko.javascript.info/searching-elements-dom

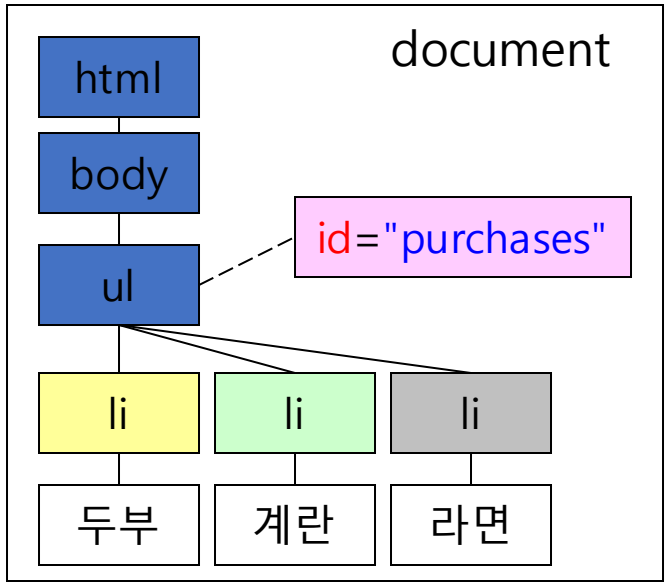
LIKE LION

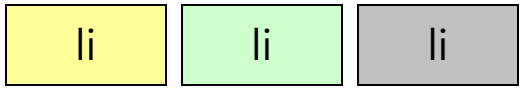
ex05-01.html

태그명을 이용하여 노드 찾기

- 요소노드.getElementsByTagName(tagName)
  - 지정한 요소노드의 하위 모든 요소를 대상으로 태그명(tagName)에 해당하는 요소노드를 배열로 반환(tagName에 "*"을 지정하면 모든 요소를 배열로 반환)

```
const liList = purchases.getElementsByTagName("li");
```





참고: https://ko.javascript.info/searching-elements-dom

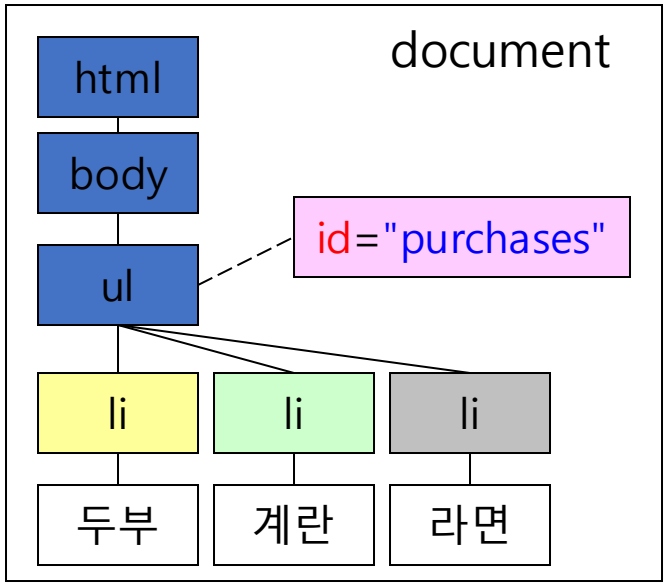
LIKE LION

ex05-01.html

트리구조를 이용하여 노드 찾기

- 부모/자식 노드 찾기

요소노드의 속성	설명
childNodes	자식 노드가 배열 형태로 저장
firstChild	첫번째 자식 노드(요소, 텍스트, 주석)
firstElementChild	첫번째 자식 요소 노드
lastChild	마지막 자식 노드(요소, 텍스트, 주석)
lastElementChild	마지막 자식 요소 노드
parentNode	부모 노드




```
const purchases = document.getElementById('purchases');
const firstItem = purchases.firstElementChild;
const lastItem = purchases.lastElementChild;
const liList = purchases.childNodes;
```

ul

li

li

li

li

li

참고: https://ko.javascript.info/dom-navigation

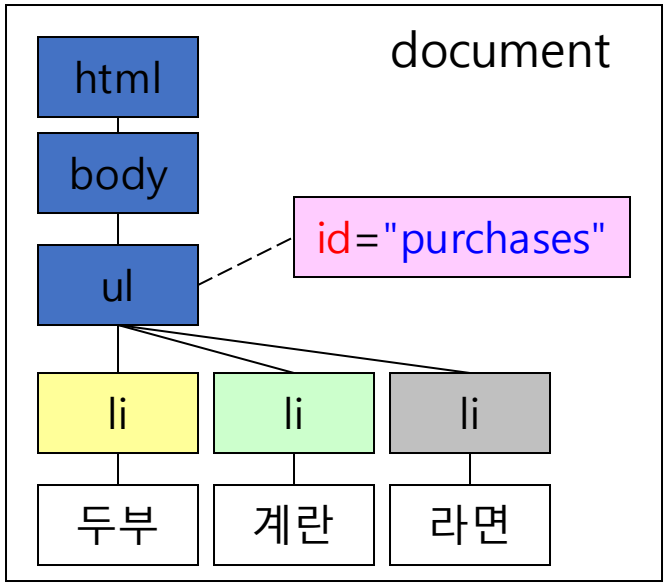
LIKE LION

ex05-01.html

▶ 트리구조를 이용하여 노드 찾기

- 형제 노드 찾기

속성	설명
previousSibling	바로 앞의 형제 노드(요소, 텍스트, 주석)
previousElementSibling	바로 앞의 형제 요소 노드
nextSibling	바로 뒤의 형제 노드(요소, 텍스트, 주석)
nextElementSibling	바로 뒤의 형제 요소 노드




```
const purchases = document.getElementById("purchases");
const secondItem = purchases.childNodes[3];
const firstItem = secondItem.previousElementSibling;
const lastItem = secondItem.nextElementSibling;
```

ul

li

li

li

참고: https://ko.javascript.info/dom-navigation

LIKE LION

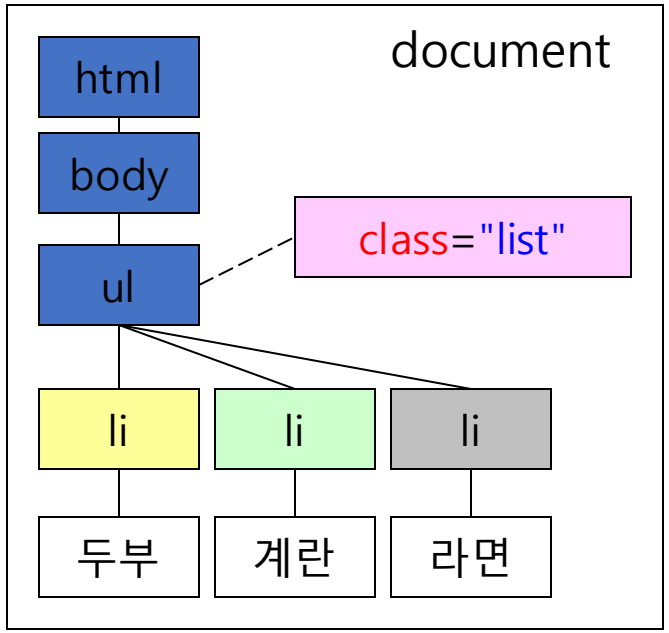
ex05-01.html

▶ class 속성으로 노드 찾기

- document.getElementsByClassName(className)
  - class 속성값이 className인 요소 노드의 목록을 반환

```
const purchases = document.getElementsByClassName("list")[0];
```

ul



참고: https://ko.javascript.info/searching-elements-dom

LIKE LION

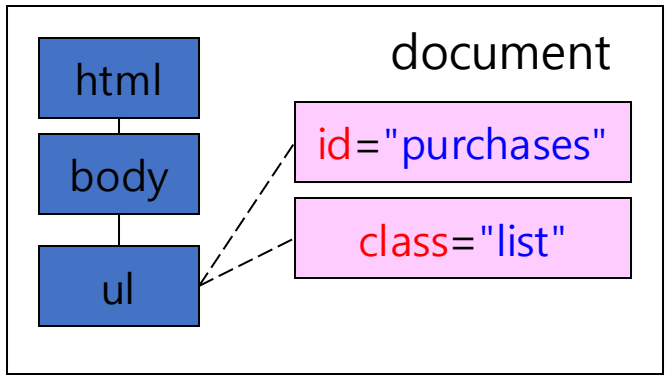
ex05-01.html

▶ CSS 셀렉터 이용

- Selector: CSS에서 사용하는 노드 선택 구문
  - https://www.w3.org/TR/css3-selectors/
- document.querySelector(selector)
  - 지정한 selector 구문에 매칭되는 노드 목록 중 첫번째 노드를 반환
- document.querySelectorAll(selector)
  - 지정한 selector 구문에 매칭되는 노드 목록을 반환

```
var purchases = document.querySelector(".list");
var purchases = document.querySelector("#purchases");
var purchases = document.querySelectorAll("ul")[0];
```

ul



참고: https://ko.javascript.info/searching-elements-dom

LIKE LION

ex05-03.html

elem.innerHTML

- elem의 내부 HTML 코드의 값을 조회하거나 수정
- elem 자신은 제외

```
const shoppingList =
document.querySelector('#purchases');
console.log(shoppingList.innerHTML);
```

```
'₩n    <li>두부</li>₩n    <li>계란</li>₩n    <li>
라면</li>₩n  '
```

elem.outerHTML

- elem의 내부 HTML 코드의 값을 조회하거나 수정
- elem 자신을 포함

```
const shoppingList =
document.querySelector('#purchases');
console.log(shoppingList.outerHTML);
```

```
'<ul id="purchases" class="list">₩n    <li>두부
</li>₩n    <li>계란</li>₩n    <li>라면</li>₩n
</ul>'
```

```
<ul id="purchases" class="list">
  <li>두부</li>
  <li>계란</li>
  <li>라면</li>
</ul>
```

참고: https://ko.javascript.info/basic-dom-node-properties

LIKE LION

ex05-03.html

elem.textContent

- elem의 내부 텍스트 노드의 값을 조회하거나 수정
- 소스코드의 값 그대로 조회

```
const secondLi =
document.querySelector('#purchases > li:nth-child(2)');
console.log(secondLi.textContent); // 계란✔
```

```
<ul id="purchases" class="list">
  <li>두부<span>✔</span></li>
  <li>계란<span hidden>✔</span></li>
  <li>라면<span>✔</span></li>
</ul>
```

elem.innerText

- elem의 내부 텍스트 노드의 값을 조회하거나 수정
- 브라우저에 의해서 실제 보이는 값으로 조회
- 화면에 보이지 않는 요소는 제외(hidden)

```
const secondLi =
document.querySelector('#purchases > li:nth-child(2)');
console.log(secondLi.innerText); // 계란
```

- 두부✔
- 계란
- 라면✔

참고: https://ko.javascript.info/basic-dom-node-properties

LIKE LION

ex05-03.html 

▶ 노드 생성

- document 객체의 createXxx() 메소드를 이용

메소드	설명
createElement(nodeName)	지정한 태그명으로 요소노드 생성
createTextNode(nodeValue)	지정한 내용으로 텍스트노드 생성
createAttribute(attributeName)	지정한 이름으로 속성노드 생성

```
const newLiNode = document.createElement("li");

const newTextNode = document.createTextNode("우유");
```

li

우유

참고: https://ko.javascript.info/modifying-document

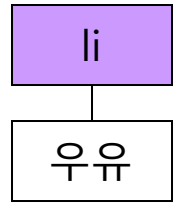
LIKE LION

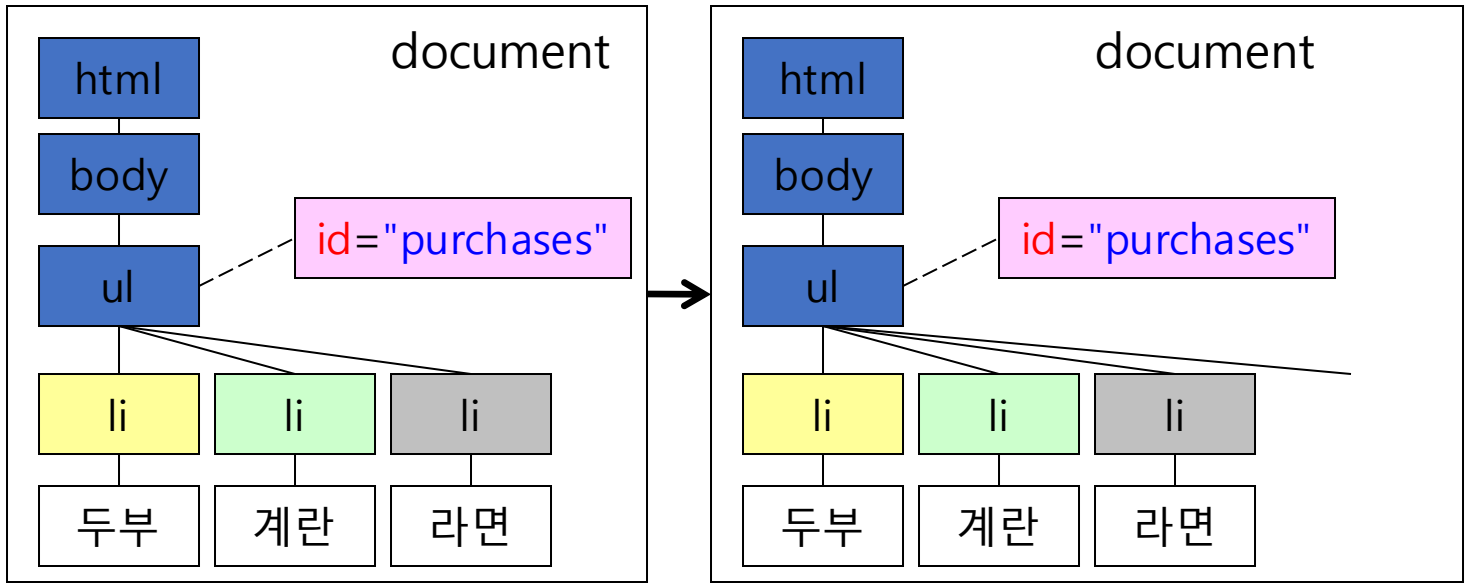
ex05-03.html

▶ 노드 추가

- 요소노드.appendChild(childNode)
  - 지정한 노드를(childNode) 요소노드의 마지막 자식노드로 추가

```
const newLiNode = document.createElement("li");
const newTextNode = document.createTextNode("우유
");
newLiNode.appendChild(newTextNode);
purchases.appendChild(newLiNode);
```





참고: https://ko.javascript.info/modifying-document

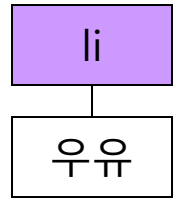
LIKE LION

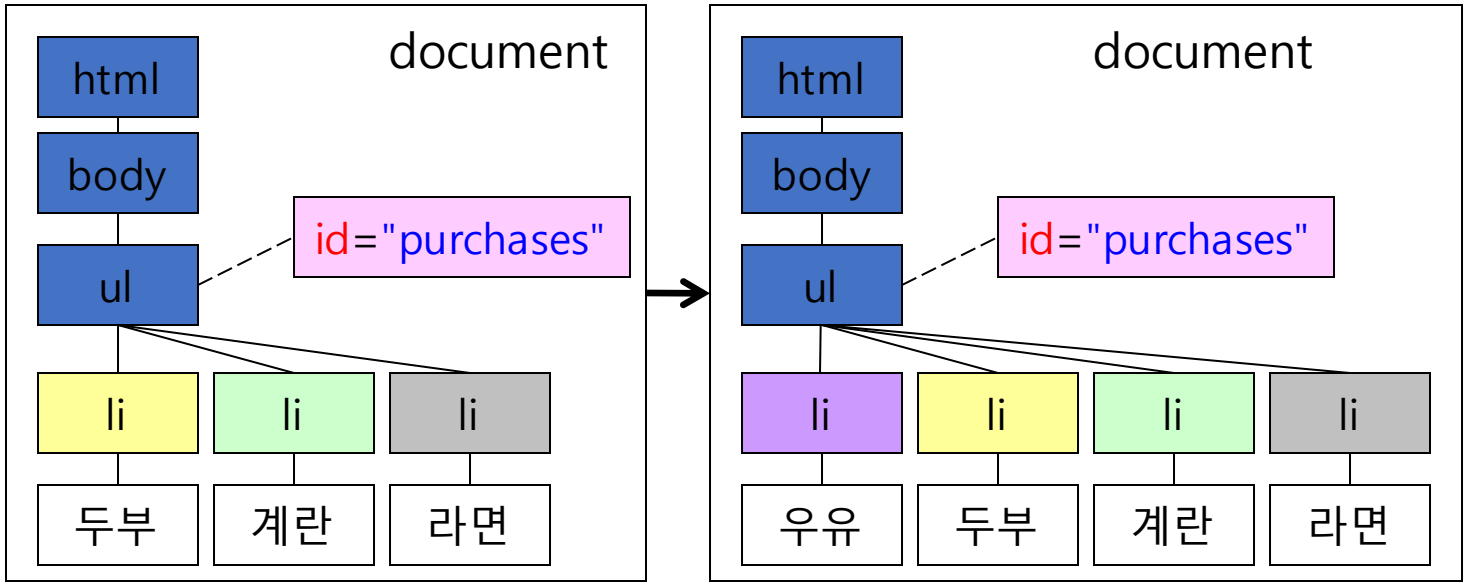
ex05-03.html

▶ 노드 삽입

- 요소노드.insertBefore(newNode, targetNode)
  - 지정한 노드를(newNode) targetNode 앞에 삽입

```
const purchases = document.getElementById("purchases");
purchases.insertBefore(newLiNode, purchases.firstChild);
```





참고: https://ko.javascript.info/modifying-document

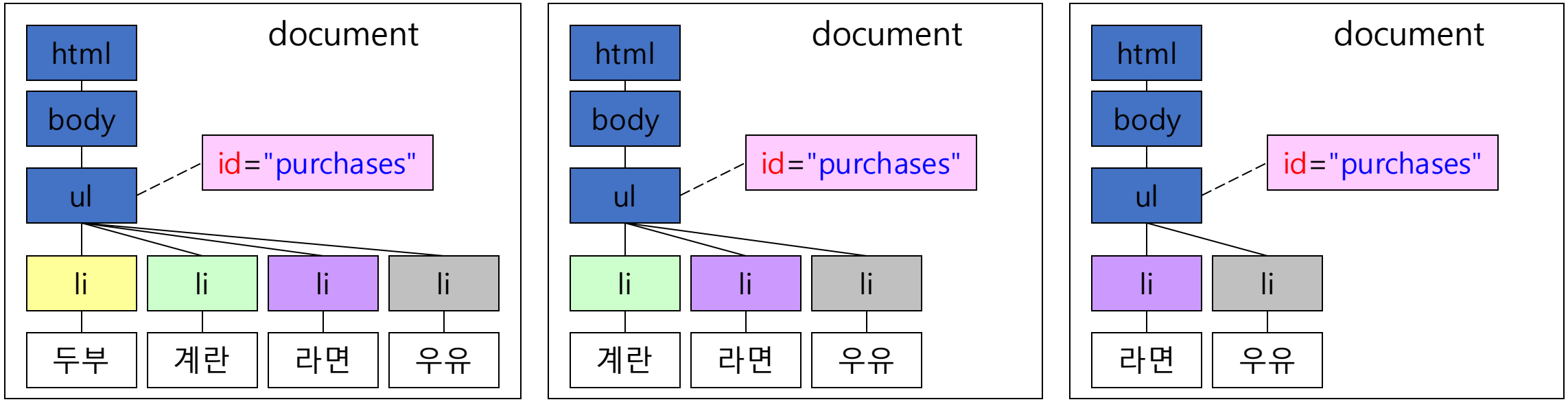
LIKE LION

ex05-03.html

▶ 노드 삭제

- 요소노드.removeChild(childNode)
  - 지정한 자식 노드를(childNode) 삭제한다.
- 요소노드.remove()
  - 자신을 삭제한다.

```
const purchases = document.getElementById("purchases");
purchases.removeChild(purchases.firstElementChild);
purchases.firstElementChild.remove();
```



참고: https://ko.javascript.info/modifying-document

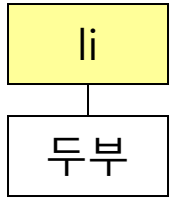
LIKE LION

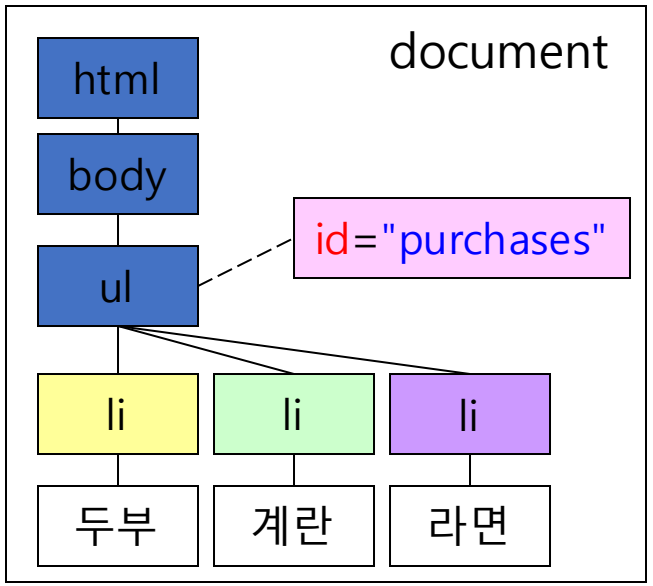
ex05-03.html

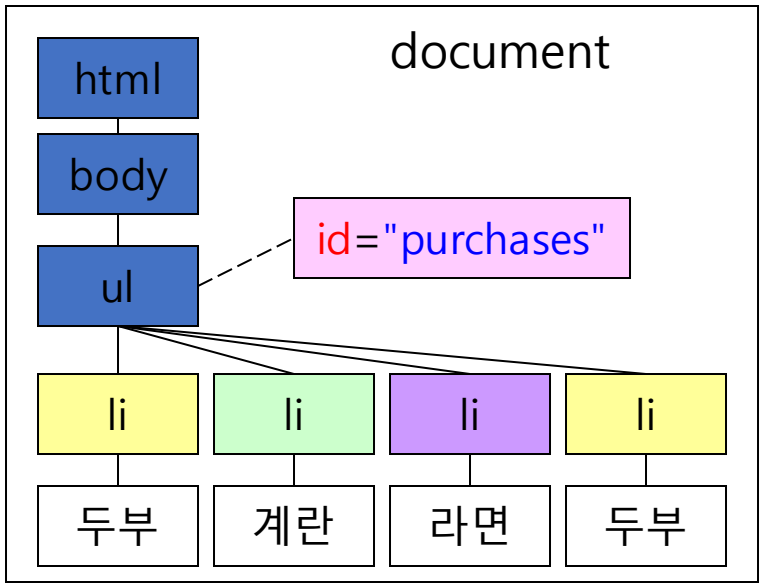
▶ 노드 복사

- 노드.cloneNode(haveChild)
  - 지정한 노드를 복사한다. haveChild가 true이면 하위 모든 노드를 같이 복사하고 false(기본값)이면 지정한 노드만 복사한다.

```
const purchases = document.getElementById("purchases");
const cloneLi = purchases.firstChild.cloneNode(true);
purchases.appendChild(cloneLi);
```







참고: https://ko.javascript.info/modifying-document

LIKE LION

ex05-06.html

HTML 표준 속성

- HTML 표준 속성은 DOM 객체의 속성으로 저장됨
  - elem.href -> "hello.html"
  - elem.src -> "hello.png"
  - elem.type -> "text"
  - elem.name -> "userName"

```
<a href="hello.html">눌러봐</a>
<img src="hello.png" format="png">
<input type="text" name="userName">
```

HTML 표준이 아닌 속성

- HTML 표준이 아닌 속성은 DOM 객체의 속성으로 저장되지 않음
  - elem.format -> undefined
- DOM 객체의 속성대신 elem.getAttribute(attrName) 메서드 이용
  - elem.getAttribute('format') -> png

custom 속성

- 개발자가 임의로 부여한 HTML 속성
- data-age, data-user-name 처럼 "data-" 접두어로 시작
- elem.dataset.age, elem.dataset.userName 처럼 dataset 객체의 속성으로 접근 가능
- 속성명을 -로 연결했을 경우 카멜케이스로 변환된 속성명 사용

참고: https://ko.javascript.info/dom-attributes-and-properties

LIKE LION

ex05-07.html

요소에 스타일 적용

- 요소의 style 속성에 직접 스타일 지정
  - `<span style="font-size: 40px;">😆</span>`

- 스타일을 명시한 클래스 작성 후 class 속성으로 적용

```
.pad100 {
  padding: 100px;
}
.size30 {
  font-size: 30px;
}
```

```
<ul class="pad100 size30">
  <li>두부</li>
  <li>계란</li>
</ul>
```

- 일반적으로 스타일 정의는 style 속성보다는 class를 사용하는 방식이 선호됨
  - 재사용성: 여러 요소에 적용 가능
  - 성능 향상: 브라우저의 캐싱
  - 유지보수: CSS 파일에서 일괄적인 스타일 관리

- 동적 스타일이 필요한 경우 제한적으로 style 사용

참고: https://ko.javascript.info/styles-and-classes

LIKE LION

ex05-07.html 

▶ 자바스크립트로 style 속성 접근

- 요소노드의 style 속성에 CSS 스타일 정보가 객체로 저장됨
- 스타일 속성에 접근할 경우 style.스타일속성명 형태로 사용
  - elem.style.top = "10px"
  - elem.style.left = "20px"

- font-size 같이 -로 연결한 스타일 속성은 카멜케이스로 변환된 속성명 사용
  - elem.style.fontSize = "10px"
  - elem.style.backgroundColor = "yellow"

- 요소노드의 style 속성은 객체이고 읽기 전용 전체 스타일을 한번에 바꾸기 위해서 문자열을 할당할 수 없음
  - elem.style = "font-size: 10px; background-color: yellow;" (x)

- 전체 스타일을 한번에 바꾸기 위해서 cssText 속성 사용
  - elem.style.cssText = "font-size: 10px; background-color: yellow;" 를 사용

참고: https://ko.javascript.info/styles-and-classes

LIKE LION

ex05-08.html 

자바스크립트로 class 접근

- elem.className 속성에 class 값이 저장되어 있음(class는 예약어라서 className을 대신 사용)
- class 값 전체를 바꿀때는 elem.className = "pad100 size30" 처럼 직접 값을 명시

class 속성을 하나씩 접근

- elem.classList
  - class 속성의 목록을 가지고 있는 유사 배열 객체
  - add("hello"): hello 클래스 추가
  - remove("hello"): hello 클래스 제거
  - toggle("hello"): hello 클래스가 있으면 제거하고 없으면 추가
  - contains("hello"): hello 클래스의 존재 여부 반환

```css
.pad100 {
  padding: 100px;
}
.size30 {
  font-size: 30px;
}
```

```html
<ul class="pad100 size30">
  <li>두부</li>
  <li>계란</li>
</ul>
```

class가 적용된 style 확인(최종 계산된 스타일)

- getComputedStyle(element, [pseudo])
  - 외부 css 파일, 내부 <style>, 인라인 스타일 등 모든 스타일 요소가 반영된 최종 계산된 스타일 반환
  - elem.style과 유사한 스타일 정보가 담긴 객체를 반환하지만 모든 속성은 읽기 전용
  - element: 스타일 값을 읽을 요소노드
  - pseudo: ::before 같은 pseudo-element의 스타일이 필요할 때 참고: https://ko.javascript.info/styles-and-classes

이벤트란?

- 브라우저에서 어떤 일이 일어 났음을 알려주는 신호
- 클릭, 키보드 입력, 마우스 이동, 스크롤 등의 작업
- 주로 요소 노드에서 발생

이벤트 핸들러

- 특정 이벤트가 발생했을 때 실행되는 함수
- 이벤트가 발생하는 대상에 이벤트와 이벤트 핸들러를 등록해서 처리

대표적인 이벤트 종류

- 마우스 이벤트
  - click, dblclick, mousemove, mouseover/mouseout, mousedown/mouseup, contextmenu
- 키보드 이벤트
  - keydown/keyup
- 폼 이벤트
  - focus/blur, input, change, submit
- 스크롤 이벤트
  - scroll
- 문서 로딩 이벤트
  - load, DOMContentLoaded, beforeunload/unload

참고: https://ko.javascript.info/introduction-browser-events

LIKE LION

ex05-04.html 

DOM 프로퍼티에 할당

- DOM Level 0 방식 (HTML 표준)
- 요소 노드의 on<**event**> 속성에 이벤트 핸들러를 등록하면 <**event**>가 발생했을 때 등록한 핸들러가 호출됨

```
<button>눌러봐</button>
```

```
var btn = document.querySelector('button');
btn.onclick = function(){
  console.log('버튼 클릭');
}
```

참고: https://ko.javascript.info/introduction-browser-events

LIKE LION

ex05-04.html 

HTML 인라인 방식

- DOM Level 0 방식 (HTML 표준)
- HTML 태그의 on<event> 속성에 <event>가 발생 했을 때 실행할 코드 지정
  - onclick, onmousemove, onkeydown 등
- 브라우저는 실행할 코드로 구성된 이벤트 핸들러를 만들어서 요소 노드의 on<event> 속성에 등록
- 아래의 두 코드는 동일한 효과

```
<button onclick="console.log('버튼 클릭');">눌러봐</button>
```

```
var btn = document.querySelector('button');
btn.onclick = function(){
  console.log('버튼 클릭');
}
```

참고: https://ko.javascript.info/introduction-browser-events

LIKE LION

ex05-04.html

DOM Level 0 방식의 불편한 점

- on<event> 속성의 값은 한개만 존재할 수 있기 때문에 이벤트 핸들러를 여러번 할당하면 기존 값이 덮어 씌워져서 이벤트 핸들러를 여러개 등록할 수 없음

```
var btn = document.querySelector('button');
btn.onclick = function(){
  console.log('버튼 클릭1');
}
btn.onclick = function(){
  console.log('버튼 클릭2');
}
```

참고: https://ko.javascript.info/introduction-browser-events

LIKE LION

ex05-04.html 

- elem.addEventListener(event, handler, [useCapture])
  - DOM Level 2 방식 (DOM 표준)
  - elem 요소노드에 event 발생시 실행할 handler 함수를 등록한다.

  - event: 이벤트 이름 (click, mousemove, keydown 등)
  - handler: 핸들러 함수
  - useCapture: 캡처링 단계의 이벤트 캐치 여부. 기본은 false이고 버블링 단계의 이벤트를 캐치함

```
var btn = document.querySelector('button');
btn.addEventListener('click', function(){
  console.log('버튼 클릭');
});
btn.addEventListener('click', function(){
  console.log('버튼 클릭');
});
```

참고: https://ko.javascript.info/introduction-browser-events

LIKE LION

ex05-04.html 

elem.removeEventListener(event, handler, [useCapture])

- elem 요소노드에 event 발생시 실행할 handler 함수를 제거한다.
- 핸들러를 등록할 때 지정했던 매개변수와 동일한 인자값의 핸들러가 삭제됨

```
var btn = document.querySelector('button');
btn.addEventListener('click', function(){
  console.log('버튼 클릭');
});
btn.removeEventListener('click', function(){ // 제거 안됨
  console.log('버튼 클릭');
});
```

```
var btn = document.querySelector('button');
function handleClick(){
  console.log('버튼 클릭');
}
btn.addEventListener('click', handleClick, true);
btn.removeEventListener('click', handleClick, true); // 제거됨
```

참고: https://ko.javascript.info/introduction-browser-events

LIKE LION

ex05-07.html 

Event 객체

- 발생한 이벤트의 상세 정보를 담고 있는 객체
- click 이벤트 였다면 마우스의 어떤 버튼이 눌렸는지, keydown 이벤트 였다면 어떤 키가 눌렸는지 같은 이벤트 상세 정보를 확인하고 싶을때 사용
- 이벤트 핸들러 함수의 첫번째 인자값으로 전달됨

```
var btn = document.querySelector('button');
document.addEventListener('mousemove', function(event){
  console.log('마우스 좌표', event.clientX, event.clientY);
});
```

```
마우스 좌표 103 104
마우스 좌표 112 93
마우스 좌표 123 81
```

참고: https://ko.javascript.info/introduction-browser-events

LIKE LION

ex05-09.html

Event 객체의 주요 속성과 메서드

- type: 발생한 이벤트 명
- target: 이벤트가 발생한 요소노드
- currentTarget: 이벤트를 처리중인 요소노드
  - <button>의 부모 에 click 이벤트를 등록하고 button을 누르면 내부가 눌렸으므로 이벤트 핸들러가 호출됨. 이때 target은 <button>이 되고 currentTarget은 가 됨

```
<div>
  <button>눌러봐</button>
</div>
```

```
var div = document.querySelector('div');
div.addEventListener('click', function(event){
  console.log(event.target, event.currentTarget);
});
```

- preventDefault(): 다음과 같은 브라우저의 기본 동작을 취소 할때 호출
  - <a> 태그를 누르면 href 주소로 페이지 이동
  - <button type="submit"> 버튼을 누르면 서버로 데이터 전송
- 그밖에 이벤트 종류별로 사용 가능한 속성 제공

참고: https://ko.javascript.info/introduction-browser-events

LIKE LION

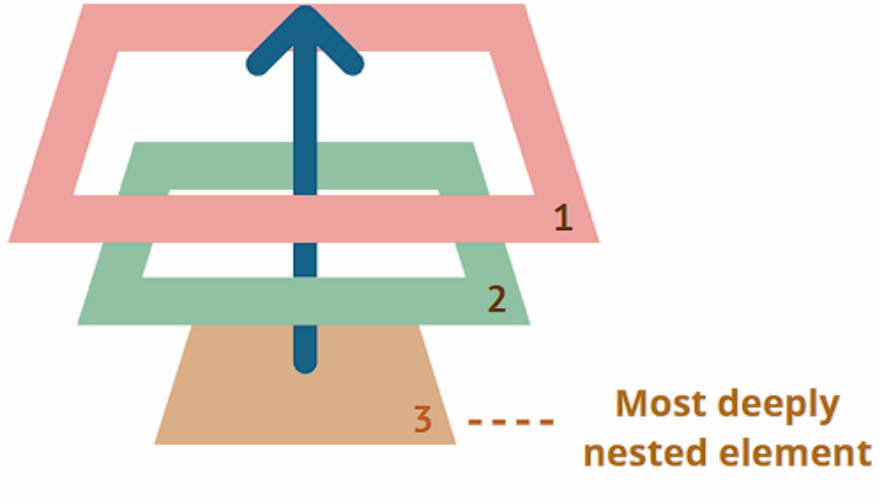
ex05-10.html

버블링(bubbling)

- 특정 요소에 이벤트가 발생하면 해당 요소의 이벤트 핸들러가 먼저 실행 된 후 부모 요소의 이벤트 핸들러가 연달아 실행됨
  - p의 onclick 핸들러 실행
  - div의 onclick 핸들러 실행
  - form의 onclick 핸들러 실행
  - document 객체를 만날 때까지 각 요소의 onclick 핸들러 실행

- event.stopPropagation() 호출시 이벤트 전파 중단
  - 대부분의 경우 버블링을 중단 시킬 일은 없음

```
<form onclick="alert('form')">FORM
  <div onclick="alert('div')">DIV
    <p onclick="alert('p')">P</p>
  </div>
</form>
```



참고: https://ko.javascript.info/bubbling-and-capturing

LIKE LION

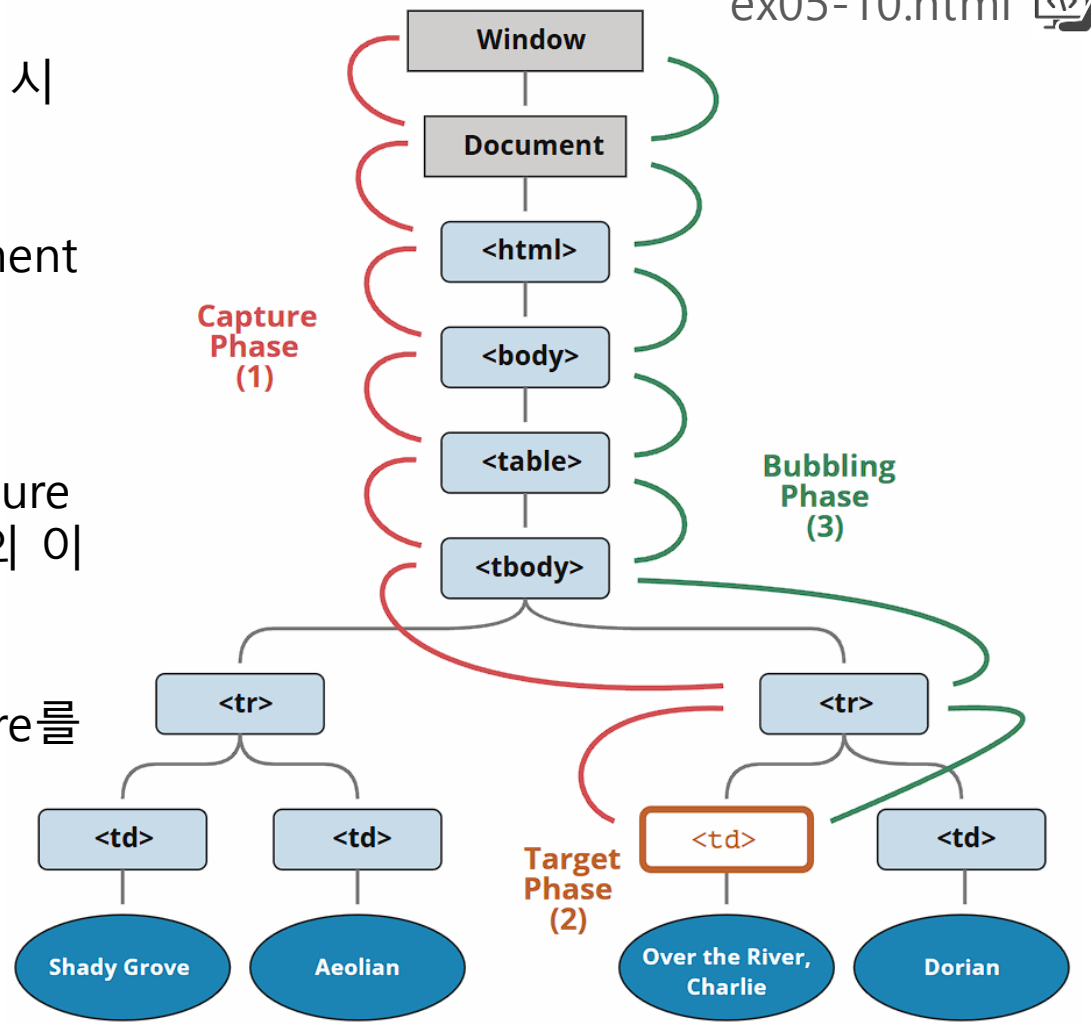
ex05-10.html

이벤트 전파 단계

- 캡처링(capturing) 단계 : 이벤트가 documen에서 시작해서 타겟 요소까지 하위 요소로 전파되는 단계
- 타깃 단계: 이벤트가 타겟 요소에 도달
- 버블링 단계:이벤트가 다시 타겟 요소에서 document까지 상위 요소로 전파되는 단계

캡처링

- addEventListener()의 세번째 매개변수인 useCapture를 생략하면 기본값은 false이고 이는 버블링 단계의 이벤트를 캐치
- 캡처링 단계의 이벤트를 캐치하기 위해서는 addEventListener()의 세번째 매개변수인 useCapture를 true로 지정해야 하지만 사용할 일은 거의 없음



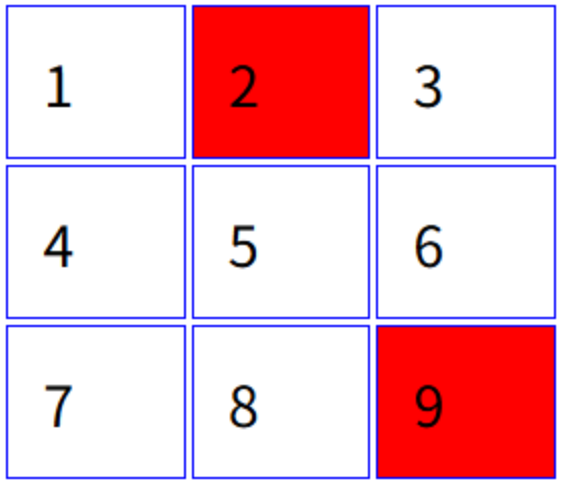
참고: https://ko.javascript.info/bubbling-and-capturing

LIKE LION

ex05-11.html

▶ 이벤트 위임이란?

- 이벤트 발생시 비슷한 처리를 해야하는 요소들이 여럿 있을 경우 각 요소에 하나씩 이벤트 핸들러를 할당하지 않고 공통의 부모 요소에 이벤트 핸들러를 하나만 할당해서 처리하는 방식
- 자식의 이벤트가 부모에게 전파되는 이벤트 버블링을 활용
- event.target 속성으로 실제 이벤트가 발생한 요소를 확인
- 동적으로 추가된 자식 요소에 따로 이벤트를 추가할 필요




```html
<table>
  <tr>
    <td>1</td><td>2</td><td>3</td>
  </tr>
  <tr>
    <td>4</td><td>5</td><td>6</td>
  </tr>
  <tr>
    <td>7</td><td>8</td><td>9</td>
  </tr>
</table>
```

```js
const table = document.querySelector('table');
table.addEventListener('click', function(event){
  if(event.target.tagName === 'TD'){
    event.target.style.backgroundColor = 'red';
  }
});
```

참고: https://ko.javascript.info/event-delegation

AJAX(Asynchronous Javascript + XML)란?

- 웹 어플리케이션 개발시에 클라이언트와 서버의 통신방법에 대한 형태로 자바스크립트와 XML에 기반한 비동기 통신기법을 사용
- 자바스크립트로 HTTP요청을 보내고 XML로 응답을 받아서 처리하는 개발 방식
- 현재는 XML 보다 JSON을 더 선호함
- 페이지 이동이나 새로고침 없이 서버에 HTTP 요청을 보내고 DOM API를 이용해 응답 데이터로 화면을 갱신하는 프로그래밍 방식

▶ XMLHttpRequest 생성자 함수

- 서버와 비동기 통신을 하는 객체
- new XMLHttpRequest()

▶ XMLHttpRequest 주요 메서드

메소드	설 명
open(method, url, async)	요청의 초기화로 요청방식, 요청 URL, 비동기 여부를 지정 method : HTTP 요청방식. GET, POST, PUT, PATCH, DELETE 등 url : 요청하는 자원의 URL async : true나 생략할 경우 비동기, false일 경우 동기방식
send(data)	요청 전송 data: POST 방식일 경우에 쿼리스트링으로 name1=value1&name2=value2 형태

속성	설명
readyState	요청의 상태를 나타내는 정수 값 - 0 : 초기화 이전 상태. open() 호출 전 - 1 : 로드되지 않은 상태. send() 메서드 호출 전 - 2 : 로드된 상태. send() 메소드 호출 후 응답헤더와 상태 받음 - 3 : 상호작용 상태. 데이터를 받고 있는 상태 - 4 : 완료 상태. 모든 데이터를 받은 상태
status	서버로부터 받은 응답의 상태를 나타내는 HTTP 상태코드 - 200, 404, 500 등
statusText	서버로부터 받은 응답의 상태 메세지
responseText	응답으로 받은 문자열
responseXML	응답으로 받은 XML DOM 객체
onload	응답이 도착하면 발생하는 이벤트 등록

LIKE LION

ajax/ex05-12.html

▶ XMLHttpRequest 콜백 함수

- 서버에 요청을 보내기 전에 XMLHttpRequest 객체의 load 이벤트를 등록해서 콜백 함수를 지정하면 응답이 도착한 후에 콜백 함수가 호출됨
- 응답 데이터는 responseText, responseXML 속성을 이용

```
const url = 'https://api.thecatapi.com/v1/images/search';
function getImages(){
  // 1. XMLHttpRequest 생성
  const xhr = new XMLHttpRequest();
  // 2. 요청준비(open())
  xhr.open('get', url, true);
  // 4. 응답 데이터 처리
  xhr.addEventListener('load', function(){
    const result = this.responseText;
    const data: Cat[] = JSON.parse(result);
    console.log(data);
  });
  // 3. 요청(send())
  xhr.send();
};
```

LIKE LION

ajax/ex05-13.html

fetch()

- ES6에서 추가
- 콜백 기반인 XMLHttpRequest 와 달리 Promise 기반으로 설계된 HTTP 클라이언트
- XMLHttpRequest를 대체해서 사용할 수 있는 표준 API
- XMLHttpRequest보다는 나은 선택이지만 응답 객체에서 본문을 바로 꺼내지 못하고 JSON이나 다른 데이터 타입으로 파싱해야하고 네트워크 에러를 제외한 HTTP 응답 에러에 대해서 오류가 발생하지 않으므로 따로 체크를 해야 하는 등 axios 라이브러리 대비 사용이 불편

```
const url = 'https://api.thecatapi.com/v1/images/search';
async function getImages() {
  try {
    const response = await fetch(url);
    if (response.ok) {
      const data: Cat[] = await response.json(); // json 파싱이 필요
      console.log(data);
    } else {
      console.error(response); // 4xx HTTP 응답 오류에 대한 처리
    }
  } catch (err) {
    console.error(err); // 네트워크 에러에 대한 처리
  }
}
```

LIKE LION

ajax/ex05-14.html 

▶ axios란?

- Node.js와 브라우저를 위한 Promise 기반 HTTP 클라이언트
- XMLHttpRequest 객체를 기반으로 동작하므로 Fetch API 보다 호환성 좋음
- 요청 및 응답 인터셉트
- JSON 데이터 자동 변환
- timeout 설정 가능

```
const url = 'https://api.thecatapi.com/v1/images/search';
async function getImages() {
  try{
    const response = await axios.get<Cat[]>(url);
    const data: Cat[] = response.data; // json 파싱이 필요 없음
    console.log(data);
  } catch(err) {
    // 네트워크 에러나 4xx HTTP 응답 에러 일괄 처리
    console.error(err);
  }
}
```

▶ BOM 이란?

- 웹페이지 외부의 브라우저 자체를 제어하기 위한 객체들의 집합을 정의한 표준
- HTML 표준: https://html.spec.whatwg.org
- window: BOM의 최상위 객체로 모든 전역 변수, 함수, 객체를 포함
    - alert(), setTimeout(), innerWidth, innerHeight 등
- window.navigator: 브라우저와 운영체제에 대한 정보 제공
- window.location: 현재 페이지의 URL에 대한 제어(읽기, 수정)
- window.history: 브라우저의 과거 페이지 이동 정보에 대한 제어(읽기, 수정)
- window.screen: 화면 해상도 등의 정보를 제공

LIKE LION

bom/ex05-15.html 

▶ navigator 객체

- 브라우저의 정보(버전, 언너, 플랫폼 등)에 접근할 수 있도록 해주는 객체
- 주로 사용자 환경을 파악하거나 기능 지원 여부를 확인할 때 사용
- navigator.userAgent: 사용자의 브라우저 및 OS 정보를 문자열로 제공
- navigator.language: 브라우저 기본 언어
- navigator.platform: 운영체제 정보
- navigator.onLine: 현재 온라인 상태 여부 확인
- navigator.geolocation: 위치 정보 확인 API

LIKE LION

bom/ex05-16.html 

location 객체

- 현재 문서의 URL 정보를 읽거나 변경할 수 있게 해주는 객체
- 페이지 이동, 새로고침, 리디렉션 등에 자주 사용
- location.assign(url): 주어진 URL 이동(history를 남김)
- location.replace(url): 현재 페이지를 새 URL로 교체(history를 남기지 않음)
- location.reload(): 페이지 새로고침
- 현재 url 관련 속성
    - https://example.com/about?category=book 일 경우
    - location.href: 현재 URL 확인 또는 변경
    - location.hostname: 도메인(example.com)
    - location.pathname: 경로(/about)
    - location.protocol: 프로토콜(https:)
    - location.search: 쿼리 스트링(?category=book)

▶ history 객체

- 브라우저의 방문 기록을 제어하는 객체
- history.back(): 이전 페이지로 이동(브라우저의 이전 페이지 버튼과 같은 효과)
- history.forward(): 다음 페이지로 이동(브라우저의 다음 페이지 버튼과 같은 효과)
- history.go(n): n만큼 앞(+), 뒤(-)로 이동
- history.pushState(state, title, url): 주소창의 url을 바꾸지만 페이지는 새로고침 하지는 않고 history에는 남김
    - data: 각 history entry에 저장되는 사용자 정의 객체. history.state 속성으로 꺼낼 수 있다.
    - title: history entry의 title(사용되지 않음)
    - url: 주소창에 보여질 url
- history.replaceState(state, title, url): 주소창의 url을 바꾸지만 새로고침 하지 않고 history에 남기지도 않음

screen 객체

- 브라우저가 실행 중인 디바이스 화면의 정보를 제공하는 객체
- 주로 해상도, 화면 크기, 색상 정보 등을 얻을 때 사용
- screen.width, screen.height: 전체 화면의 가로/세로 픽셀
- screen.availWidth, screen.availHeight: 작업표시줄 등을 제외한 사용 가능한 크기
- screen.colorDepth: 색상 깊이(보통 24비트이거나 32비트)
- screen.pixelDepth: 한 픽셀당 비트 수(colorDepth와 같거나 유사한 값)

그밖의 레이아웃 관련 속성

- window.innerWidth, window.innerHeight: 스크롤바를 포함한 뷰포트 너비/높이
- window.outerWidth, window.outerHeight: 툴바, 주소창을 포함한 브라우저 전체 창 크기
- document.documentElement.clientWidth: window.innerWidth와 비슷하지만 스크롤바 제외

Web APIs 란?

- 자바스크립트를 통해 다양한 기능을 활용할 수 있도록 웹 브라우저에서 제공하는 API들
- 브라우저의 내장 기능을 자바스크립트로 제어할 수 있게 해주는 도구들
- DOM API: HTML 문서의 구조와 내용을 동적으로 수정하고 제어
- XMLHttpRequest API, Fetch API: 비동기 HTTP 요청을 보내고, 서버와 통신에 사용
- Geolocation API: 사용자의 위치 정보를 가져오는 API
- Web Storage API: 웹 브라우저에 영구적으로 데이터를 저장하고 관리
- Notification API: 사용자에게 OS 수준의 알림 메세지를 보낼 수 있는 API
- WebSocket API: 서버와 실시간 양방향 통신을 위한 API
- Service Worker API: 백그라운드에서 실행되는 스크립트로, 오프라인 기능 및 푸시 알림 등을 처리

Geolocation API

- 사용자의 위치 정보를 가져오는 API
- GPS, 이통사 기지국, Wi-Fi, IP 등의 정보를 기반으로 위치 확인

Geolocation 객체

- window.geolocation 속성으로 사용
- getCurrentPosition(successCallback, errorCallback, options)
  - 현재 위치 정보를 비동기적으로 한번 확인
  - successCallback: 위치 정보 확인 성공시 호출(인자로 위치정보를 나타내는 Position 객체가 전달됨)
  - errorCallback: 에러 발생 시 호출(인자로 에러정보를 나타내는 PositionError 객체가 전달됨)
  - options: 옵션(PositionOptions 타입)
- watchPosition(successCallback, errorCallback, options)
  - 현재 위치 정보를 지속적으로 확인
  - watchId(정수형 값)가 반환
- clearWatch(watchId)
  - 위치 확인을 중지

▶ 콜백 함수의 인자

- 위치 정보 확인 성공 시 getCurrentPosition() 메소드에 첫번째 인자로 지정한 콜백 함수가 호출
- 콜백 함수의 인자로 Position 객체가 전달

▶ Position

- 위치 정보와 시간 정보를 가지고 있는 객체
- Position의 속성
  - coords: 위/경도 등의 위치 정보가 저장된 Coordinates 객체
  - timestamp: 위치 정보를 얻은 시각(1970/01/01 부터의 밀리세컨드)

▶ Coordinates

- 속성
  - latitude : 위도
  - longitude : 경도
  - altitude : 고도
  - accuracy : 위/경도의 오차(미터 단위)
  - altitudeAccuracy : 고도의 오차(미터 단위)
  - heading : 디바이스의 진행 방향(북쪽을 기준으로 시계방향의 각도값)
  - speed : 디바이스의 진행 속도(미터/초)

- 콜백 함수의 인자
  - 위치 정보 확인 실패 시 getCurrentPosition() 메소드에 두번째 인자로 지정한 콜백 함수가 호출
  - 콜백 함수의 인자로 PositionError 객체가 전달

- PositionError
  - 속성
    - code : 에러 코드. PositionError에 다음의 상수로 정의
      - PERMISSION_DENIED(1) : 사용자가 동의하지 않음
      - PERMISSION_UNAVAILABLE(2) : 네트워크 문제나 GPS 문제로 위치정보 확인 불가
      - TIMEOUT(3) : 지정 시간 초과
    - message : 에러 메세지

bom/ex05-17.html 

▶ 옵션 객체

- getCurrentPosition() 메소드에 세번째 인자로 지정한 옵션
  - enableHighAccuracy : 정확도 높은 위치 정보 요청(GPS > 기지국 > Wi-Fi > IP)
  - timeout : 시간제한(밀리 세컨드). 시간 초과시 에러발생
  - maximumAge : 위치 정보의 유효 기간 설정(밀리 세컨드). 0일 경우 항상 새로운 위치 정보 확인

Web Storage

- key-value 형태의 데이터를 저장하기 위한 스토리지
- 자바스크립트 객체를 다루듯 사용법이 간단
  - 저장 : 스토리지 속성에 값을 지정
  - 읽기 : 스토리지 속성에 접근
- 로컬 스토리지와 세션 스토리지로 구분
- 도메인별 각각 별도의 공간에 생성되기 때문에 다른 도메인에서는 접근 불가능

Web Storage vs. Cookie

- 기본 크기는 5M byte(쿠키는 4K byte)
- 서버로 데이터를 보내지 않음(쿠키는 요청 헤더에 자동으로 포함)
- 만료 기간이 없음(쿠키는 만료기간 지정)
- 자바스크립트 객체를 저장할 수 있음(쿠키는 문자열만 저장)

로컬 스토리지

- 프로그램이나 사용자가 삭제 하지 않는 이상 영구적인 데이터 저장
- "오늘 하루 이창을 열지 않음" 같이 보안이 필요하지 않은 데이터 저장에 적합

세션 스토리지

- 브라우저(window 객체)와 같은 생존 기간을 가지는 저장 영역
- 브라우저 탭이 닫히면 세션 스토리지 정보도 사라짐
- 새로고침이나 페이지 이동 시에는 세션 스토리지 정보는 유지됨
- 로그인한 사용자 정보 처럼 보안이 필요한 임시 데이터에 적합
- 다른 종류의 브라우저가 같은 도메인에 접속하더라도 로컬 스토리지나 세션 스토리지는 브라우저 별로 따로 생성

스토리지에 접근

- 로컬 스토리지 : window.localStorage 속성
- 세션 스토리지 : window.sessionStorage 속성

스토리지에 값 저장

- key값을 스토리지 객체의 속성명으로, value값을 속성값으로 직접 저장
  - localStorage.userId = 'haru';
  - localStorage['userId'] = 'namu';
- setItem() 메서드 이용
  - localStorage.setItem('userId', 'haru');

스토리지의 값 읽기

- 스토리지 객체의 속성명으로 읽기
  - const userId = localStorage.userId;
  - const userId = localStorage['userId'];
- getItem() 메서드 이용
  - const userId = localStorage.getItem('userId');

LIKE LION

bom/ex05-18.html 

스토리지의 데이터 삭제

- delete 연산자 이용
  - delete localStorage.userId;
  - delete localStorage['userId'];
- removeItem() 메서드 이용
  - localStorage.removeItem('userId');

스토리지의 모든 데이터 삭제

- clear() 메서드 이용
  - localStorage.clear();

기타 스토리지의 속성과 메서드

- length : 스토리지에 저장된 데이터의 수
- key(index) : 지정한 인덱스의 키를 반환(없으면 null)

websocket 프로토콜

- IETF http://tools.ietf.org/html/draft-ietf-hybi-thewebsocketprotocol
- 브라우저와 서버간 양방향 전이중 통신(Full Duplex)을 구현
- 일반 통신은 ws 프로토콜, 보안 통신은 wss 프로토콜 사용

websocket을 이용한 통신

- websocket 프로토콜 구현 서버 필요 (웹소켓 서버)
- websocket 프로토콜 구현 클라이언트 필요 (웹 브라우저)

- Web Socket Server
  - websocket 프로토콜 구현 서버
  - jWebSocket, pywebsocket, phpwebsocket, web-socket-ruby, socket.io 등

- socket.io
  - Node.js의 확장 모듈
  - 이벤트 기반의 통신 API 제공
  - 웹 브라우저가 지원하는 통신 방식으로 자동 접속(web socket, xhr-polling 등)

LIKE LION

bom/ex05-19.html 

WebSocket 생성자 함수

- 클라이언트 측의 자바스크립트 API
- websocket 프로토콜을 이용하여 서버와 통신하는 객체

주요 메서드

- new WebSocket(url) : 웹소켓 서버와 연결(ws, wss 프로토콜 사용)
- send(msg) : 서버에 메시지 전송, 서버와 연결된 상태이면 true, 서버와의 연결이 끊어진 상태이면 false 반환

주요 이벤트

- open : 서버와 연결됨
- message : 서버로부터 데이터 수신 시 발생. 인자값인 MessageEvent를 이용하여 데이터 수신
- close : 서버와 연결 해제