

7장 DOM과 이벤트

수업목표

- DOM의 기본 개념을 이해할 수 있다.
- DOM 노드의 노드 종류별 속성을 이해할 수 있다.
- DOM 노드를 생성, 수정, 삭제하는 메서드를 이해할 수 있다.
- DOM 엘리먼트 노드의 클래스 속성을 관리할 수 있다.
- DOM 이벤트를 다룰 수 있다.
 - 이벤트 캡처링과 버블링에 대해서 이해할 수 있다.
 - 자바스크립트의 다양한 이벤트 종류를 알고 이벤트를 등록하고 이벤트 발생 시 처리할 수 있다.
 - HTML DOM을 제어할 수 있는 시점을 알 수 있는 이벤트인 DOMContentLoaded 이벤트에 대해서 이해할 수 있다.
 - 이벤트를 취소하는 메서드인 stopPropagation()과 preventDefault()의 차이를 이해할 수 있다.

DOM(Document Object Model)

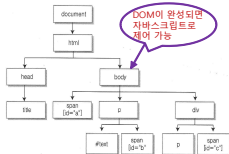
- DOM(문서 객체 모델)은 트리 형태로 구조화된 데이터이다.
- HTML 또는 XML 페이지의 구조와 요소(스타일, 내용)들을 제어할 수 있도록 제공하는 프로그래밍 인터페이스(API)이다.

html 파일

```
<html>
<head>
<title>DOM API</title>
<style>
p, span, div {
  border: 1px solid green;
  margin: 10px; }
</style>
</head>
<body>
<span id="a">span</span>
<p>p<span
id="b">span</span></p>
<div><p>p</p><span
id="c">span</span></div>
</body>
</html>
```

문서
구조
파악

DOM tree



렌더링

DOM tree를 화면에 그림

실습 dom/dom.html

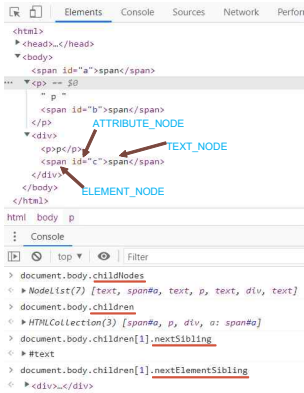
DOM의 구조와 노드, 태그의 이해

- HTML 태그구조를 **데이터 구조**로 표현
 - 유효하지 않은 HTML 태그를 교정해서 완전한 형태의 DOM 트리구조로 표현
- 자바스크립트는 **DOM을 제어**하는 것이지 HTML 페이지의 태그를 제어하는 것이 아니다.
 - 태그를 추가하거나 이동하는 것은 모두 DOM을 변경
 - DOM을 변경하면 브라우저가 다시 렌더링 함
- HTML 태그와 DOM 노드는 1:1로 매칭 되지 않음.
 - **노드와 태그는 다르다.**
 - 노드는 엘리먼트(태그) 노드 외에 텍스트 노드, 주석 노드로 분류됨
 - 노드는 엘리먼트노드 외에도 줄바꿈, 텍스트 내용 또는 브라우저에서 렌더링 되지 않는 시각적이지 않는 요소(주석)들도 모두 각각의 노드로 표현
- DOM에는 CSS에 관련된 내용은 없다
 - CSS는 CSSOM으로 표현하며 DOM과는 완전히 분리된 데이터 구조

노드 종류

- 노드는 DOM 트리를 구성하는 최소한의 단위

상수	노드 종류	HTML 용도
1	ELEMENT_NODE	<a> <div> 와 같은 html 태그
2	ATTRIBUTE_NODE	id='myelement' class='align:right'
3	TEXT_NODE	텍스트 문자열 노드. 태그내용
8	COMMENT_NODE	<!-- 주석 표시 -->
9	DOCUMENT_NODE	document 루트 노드
10	DOCUMENT_TYPE_NODE	<!DOCTYPE html>



```

<html>
  <head>...</head>
  <body>
    <span id="a">span</span>
    ...
    <p> -- $0
      " p "
      <span id="b">span</span>
    </p>
    <div>
      <p>p</p>
      <span id="c">span</span>
    </div>
  </body>
</html>

```

Annotations in the DOM tree:

- ATTRIBUTE_NODE points to `id="b"` in `span`
- TEXT_NODE points to `" p "` in `<p> -- $0`
- ELEMENT_NODE points to `span`

Console output:

```

> document.body.childNodes
< ▶ NodeList(7) [text, span#a, text, p, text, div, text]
> document.body.children
< ▶ HTMLCollection(3) [span#a, p, div, a: span#a]
> document.body.children[1].nextSibling
< ▶ #text
> document.body.children[1].nextElementSibling
< ▶ <div>...</div>

```

엘리먼트 선택

메서드	설 명
NodeList getElementsByTagName()	태그이름으로 태그에 해당하는 모든 엘리먼트의 목록을 구함
NodeList getElementsByName()	name 속성의 값이 일치하는 모든 엘리먼트의 목록을 구함
Element getElementById()	Id 속성으로 하나의 엘리먼트를 구함
NodeList getElementsByClassName()	class 속성으로 모든 엘리먼트 목록을 구함
Element querySelector()	css 선택자에 해당하는 하나의 엘리먼트를 구함
NodeList querySelectorAll()	css 선택자에 해당하는 모든 엘리먼트를 구함

엘리먼트 선택

- `element.querySelector("id")`
- `element.querySelectorAll("class")`
- `element.querySelectorAll("div.section > p#comment span.bold")`

NodeList

프로퍼티 타입	프로퍼티 이름	설 명
number	length	노드 리스트에 저장된 노드의 개수
node	item(i)	인덱스 i에 저장된 노드를 구함

```
var anchorTags = document.getElementsByTagName("a");
```

NodeList



DOM과 JavaScript

document :
Document 인터페이스를 구현하는 개체.
HTML 페이지에 있는 모든 것을 지님.

anchorTags.length : NodeList 인터페이스의
length 속성을 정의

```
var anchorTags = document.getElementsByTagName("a");  
  
for (var i = 0; i < anchorTags.length; i++) {  
    alert(`Href of ${i}th element is ${anchorTags[i].href} \n`);  
}
```

anchorTags[i].href :
"href"는 DOM1 HTML 명세에 정의된 HTMLAnchorElement 인터
페이스의 속성으로서 anchor 요소의 href 속성값을 반환.
= anchorTags.item(i).href

property와 attribute 속성의 이해

파워북 p.264

- DOM의 속성은 property(프로퍼티)

```
element.속성이름 = "속성값"
```

← 속성이 있는지 확인

- HTML 태그의 속성은 attribute(어트리뷰트)

- 속성이름은 대소문자 구분 없음

```
element.hasAttribute("속성이름")
```

← 속성이 있는지 확인

```
element.getAttribute("속성이름")
```

← 속성값 얻기

```
element.setAttribute("속성이름", "속성값")
```

← 속성값 변경

```
element.removeAttribute("속성이름")
```

← 속성 제거

실습 - property와 attribute 속성

//1. 표준속성은 HTML attr과 DOM의 prop가 속성값을 공유함.

```
menu.setAttribute('type', 'square');
console.log('1:', menu.type );
```

HTML attr로 표준속성 지정
DOM prop로 조회(0)

menu.type="square"도 동일

```
menu.title = '메인메뉴';
console.log('1:', menu.getAttribute('title'));
```

DOM prop로 표준속성 지정
HTML attr로 조회(0)

//2. 표준속성이지만 속성명이 다른 경우가 있음

```
menu.setAttribute('class', 'wide');
console.log('2:', menu.className );
```

HTML attr => class
DOM prop => className

표준이면 .과 set,get을 섞어서 사용가능

//3. 비표준(커스텀)속성을 HTML attr로 지정하면 HTML에서만 접근가능

```
menu.setAttribute('info', '메뉴');
console.log('3:', menu.info );
```

HTML 커스텀속성 지정
DOM prop로 조회안됨(X)

비표준이면 .과 set,get을 섞어서 못씀

//4. 비표준 (커스텀) 속성을 DOM prop로 지정하면 DOM에서만 접근가능.

```
menu.author = '관리자';
console.log('4:', menu.getAttribute('author'));
```

DOM 커스텀 속성지정
HTML attr로 조회안됨(X)

비표준이면 .과 set,get을 섞어서 못씀

```
menu.obj = {id:1, name:'등록'};
console.log('4:', menu.obj.name );
```

DOM attr은 객체타입도 가능

실습 dom/attr_prop.html

▶ 스크립트 실행 전

```
<ul id="menu">
  <li>_</li>
```

▶ 스크립트 실행 후 DOM tree

```
<ul id="menu" type="square"
  title="메인메뉴" class="wide"
  info="메뉴">
  <li>_</li>
```

▶ 스크립트 실행 후 console 창

```
1: square
1: 메인메뉴
2: wide
3: undefined
4: null
4: 등록
```

실습 – innerHTML, innerText 속성



dom/ .html

```
<head>
<title>innerHTML 내용 출력</title>
<script type="text/javascript">
function view() {
    var p1 = document.getElementById("p1");
    alert(p1.innerHTML);
}
</script>
</head>

<body>
    <p id="p1"><strong>Ajax</strong> 프로그래밍,<br/>기초</p>
    <input type="button" value="보기" onclick="view()"/>
</body>
```

클래스 추가

- 클래스 1개를 추가

dom으로 쓸때 앞에 .name이 아니라 .className으로 해야됨

```
var view = document.getElementById("view");
view.className = 'blackSkin';

view.setAttribute('class', 'blackSkin');
```

- 클래스 2개를 추가

```
view.className = 'blackSkin border';
view.className += 'border'
```

```
view.setAttribute('class', view.setAttribute('class') + 'border');
```

set아니라 get임

- 실습예제

https://www.w3schools.com/howto/howto_js_toggle_hide_show.asp

https://www.w3schools.com/howto/tryit.asp?filename=tryhow_js_tabs

클래스 삭제

- 클래스 속성 자체를 삭제

```
var view = document.getElementById("view");  
view.removeAttribute('class');
```

- 클래스 값만 삭제 : 속성에 빈값 대입

```
var view = document.getElementById("view");
```

```
view.className = '';
```

```
view.setAttribute('class', '');
```

classList를 이용한 편리한 클래스 관리

- 클래스 속성을 컬렉션(배열) 형태로 제공
 - 클래스 추가 : 나머지 파라미터 지원하므로 여러 개의 클래스명을 한거번에 추가 가능

```
element.classList.add('class1', 'class2', 'class3' ...)
```

```
let arrClass = [ 'class1', 'class2', 'class3'];  
view.classList.add( ... arrClass )
```

- 클래스 삭제

```
element.classList.remove('class1', 'class2', 'class3' ...)
```

```
let arrClass = [ 'class1', 'class2', 'class3'];  
view.classList.remove( ... arrClass )
```

classList를 이용한 편리한 클래스 관리 토글은 없으면 넣고 있으면 넣는다

- 클래스 토글 : 나머지 파라미터 지원하므로 여러 개의 클래스명을 한꺼번에 추가 가능

```
element.classList.toggle('class1')
```

- 클래스가 있는지 확인 : T/F 리턴

```
boolean = element.classList.contains('class1')
```

- 클래스 대체(교환)

```
boolean = element.classList.replace('기존클래스', '새클래스')
```

- 실습

https://www.w3schools.com/howto/howto_js_toggle_like.asp

classList를 이용한 편리한 클래스 관리

```
<body>
  <div id="title" class="">title</div>
  <div id="content" class="">content</div>
  <div id="footer" class="">
    <ul>
      <li>home</li>
      <li>admin</li>
    </ul>
  </div>
```

```
<script>
  title.classList.add('bold', 'padding10')
  document.querySelectorAll('div').forEach(function(el){
    if(el.classList.contains('padding10')){
      el.classList.remove('padding10');
    }
    el.classList.add('padding10', 'red');
  })

  title.classList.toggle('red')
  if( footer.classList.contains('padding10') ) {
    footer.classList.replace('padding10', 'margin10')
  }

  const arrClass = ['copyright', 'inline']
  const infos = document.querySelectorAll("#footer ul li");
  infos.forEach(function(el){
    el.classList.add( ...arrClass)
  })
</script>
```

classList를 이용한 편리한 클래스 관리

title
content

- home
- admin



```
<!DOCTYPE html>
<html lang="en">
  <head>...</head>
  ... <body> == $0
    <div id="title" class="bold padding10">title</div>
    <div id="content" class="padding10 red">content</div>
    <div id="footer" class="margin10 red">
      <ul>
        <li class="copyright inline">...</li>
        <li class="copyright inline">...</li>
      </ul>
    </div>
```

스타일(CSS) 변경

- 요소의 위치 변경

- position:absolute // 절대 위치

```
obj.style.left = '100px';  
obj.style.top = '100px';
```

- position : relative // 상대위치
 - 상대적인 거리 값을 고려해서 left와 top 속성을 명시
 - position : static // 기본설정으로 입력한 순서대로
 - relative로 변경하여 위치 지정

스타일(CSS) 변경

- css 스타일 값 구하기 문제 및 해결 방법

```
var view = document.getElementById("view");  
var color = view.style.color ;
```

- 태그에 직접 style 속성을 지정하지 않은 경우에는 값을 조회 못함.

```
var color = document.defaultView.getComputedStyle(div, '')  
                .getPropertyValue('background-color');
```

엘리먼트 생성/삭제

DOM 메서드 구분	메소드	설명
생성	<code>Element createElement(String tagName)</code>	지정한 태그 이름을 갖는 Element 노드를 생성한다.
	<code>Text createTextNode(String text)</code>	Text를 값으로 갖는 Text 노드를 생성한다.
삭제	<code>Node removeChild(Node oldChild)</code>	현재 노드의 자식 노드인 oldChild를 현재 노드에 제거
	<code>Node remove()</code>	현재 엘리먼트 자신을 삭제

엘리먼트 생성

```
<script>
  let div = document.createElement('div');
  let span = document.createElement('span');
  span.innerText = "HTML 문서 샘플";
  div.appendChild(span);
  document.body.append(div);
</script>
```

```
document.body.innerHTML += '<div><span>innerHTML DOM 생성</span></div>'
```

엘리먼트 삭제

```
<body>
  <div id="delid">삭제예정</div>
  <div id="parentdiv">
    <span id="childspan">삭제예정 child</span>
  </div>
  <script>
    document.getElementById('delid').remove()

    let parent = document.getElementById("parentdiv")
    let child = document.getElementById("childspan")
    parent.removeChild(child)

    parent.innerHTML = ""
  </script>
</body>
```

노드 삽입/이동/교체

자바스크립트 메서드 구분	메소드	설명
자바스크립트 메서드	before()	선택한 엘리먼트의 뒤에 추가
	after()	선택한 엘리먼트 앞에 추가
	prepend()	현재 엘리먼트의 첫 번째 자식 엘리먼트 앞에 추가
	append()	현재 엘리먼트의 마지막 자식 엘리먼트 뒤에 추가
DOM 메서드	Node insertBefore(Node new, Node old)	현재 노드의 자식 노드인 refChild 노드의 previousSibling 자리에 newChild 노드를 삽입
	Node appendChild(Node newChild)	NewChild 노드를 현재 노의 마지막 자식 노드로 추가한다.
	Node replaceChild(Node newChild, Node refChild)	현재 노드의 자식 노드인 oldChild 노드를 새로운 newChild노드로 교체
	cloneNode()	현재 노드를 복사해서 반환

실습 - 엘리먼트 삽입/이동

```
let li=document.createElement("li")
let litext=document.createTextNode("무지")
li.appendChild(litext);
```

```
let targetul=document.getElementById("friends");
targetul.appendChild(li);
```

```
let sourceul=document.querySelector("ul#friends li:first-child");
let targetli=document.querySelector("ul#friends li:last-child");
targetli.after(sourceul);
```

```
var appendli=document.createElement("li")
appendli.append("왕눈이")
document.querySelector("ul li ul").append(appendli);
```

```
let selected=document.querySelector(".icons li span:last-child");
let newsp=document.createElement("span");
let newspantext=document.createTextNode("빅 ");
newspan.appendChild(newspantext);
selected.parentNode.insertBefore(newspan, selected);
```

```
let items=document.querySelectorAll(".animal");
document.querySelector("#newfriends").prepend(...items);
```

실습

dom/ append_node.html



스크립트 실행 전

- 라이언
- 어피치
- 프로도
- 콘
 - 3세
 - 숫다리
 - 초록괴수
- 스몰 미디엄 빅



스크립트 실행 후

- 어피치
- 콘
 - 3세
 - 숫다리
 - 초록괴수
 - 왕눈이
- 무지
- 스몰 미디엄 빅 거대
- 프로도
- 라이언

실습 - 엘리먼트 교체

실습

dom/ .html

```
<body>
  <p id="pid">교체 될 내용</p>
<script>
  //<p> 엘리먼트 생성
  let p=document.createElement('p');
  // 문단 텍스트 생성
  let ptext = document.createTextNode('문단 텍스트 내용');
  // 문단 텍스트를 태그에 채움
  p.appendChild(ptext);

  //교체할 엘리먼트 선택
  let originalp = document.getElementById('pid')

  //부모태그에서 자식 엘리먼트 교체
  let oldp = originalp.parentNode.replaceChild(p, originalp);
</script>
</body>
```

노드 탐색

프로퍼티 타입	메서드와 속성	설명
String	nodeName	노드의 이름
String	nodeValue	노드의 값
Unsigned short	nodeType	1:Element, 2:Attribute, 3:Text, 8:Comment, 9:Document
Node	parentNode	부모 노드
NodeList	childNodes	자식 노드 목록
Node	firstChild	첫번째 자식 노드
Node	lastChild	마지막 자식 노드
Node	previousSibling	현재 노드와 같은 부모를 갖는 자식 노드 중 현재 노드 이전의 자식 노드
Node	nextSibling	현재 노드와 같은 부모를 갖는 자식 노드 중 현재 노드 다음의 자식 노드
Document	ownerDocument	이 노드가 포함된 Document 객체

실습 - 노드탐색

```
<input type="button" value="확인" onclick="node_access()"/>
<div id="a">a</div>
<div id="b">b</div>
<div id="c">c</div>

<script>
    function node_access() {
        var htmlNode = document.documentElement;
        var bodyNode = htmlNode.lastChild;
        var lastDivNode = bodyNode.lastChild;
        var textNode = lastDivNode.firstChild;
        var strValue = textNode.nodeValue;
        alert(strValue);
    }
</script>
```



dom/ .html

이벤트 등록 p274

- 인라인으로 이벤트 등록 : HTML 태그에 직접 이벤트 핸들러 등록

```
<script>
  function doProcess(){ console.log('click!!'); }
</script>
<input type="button" onclick="doProcess()"/>
```

- onXXX 속성을 이용하여 이벤트 핸들러 등록 : 자바스크립트 이용

```
<input type="button" value="클릭" id="btn1"/>
<script>
  function doProcess(){ console.log('click!!'); }
  let btn1 = document.getElementById("btn1");

  btn.onclick = doProcess;
</script>
```

이벤트 등록

매우 중요

- 이벤트 리스너 방식 처리

- addEventListener() 메서드

- 장점

- 하나의 이벤트유형에 다수의 핸들러를 부착 가능
 - 캡처링, 버블링 동작을 제어
 - html, svg 요소가 아니어도 이벤트 대상으로 사용가능

```
target.addEventListener(type, listener);
```

target: 이벤트타겟- 이벤트가 발생한 대상(요소)

type: 이벤트타입- 사용자가 발생시킨 액션(이벤트종류). 마우스클릭, 키보드클릭 등

listener: 이벤트수신기(핸들러)-이벤트가 발생하면 실행되는 콜백함수. 매개변수로 이벤트객체를 전달.

```
document.addEventListener('이벤트타입', function(event) {  
  console.log( event.type, event.target )  
})
```

이벤트 등록

- 이벤트 리스너 방식 처리 : `addEventListener()` 메서드

```
<input type="button" id="btn" value="등록" />
<script>
function doProcess(e) {
    console.log(e.target.value + 'click~~')
}
let btn = document.getElementById("btn");
btn.addEventListener("click", doProcess);
</script>
```

```
let btn = document.getElementById("btn");
btn.addEventListener("click", function(e) {
    console.log(e.target.value + 'click~~')
});
```

이벤트 등록

• 이벤트 등록 옵션 추가

```
target.addEventListener(type, listener, options);  
target.addEventListener(type, listener, useCapture);
```

useCapture: 이벤트를 캡처링 방식으로 전파할지를 결정

options: capture -

once - 이벤트가 한번만 호출되고 등록된 이벤트 리스너가 삭제됨

passive - 기본값은 false. true 로 설정하면 preventDefault() 메서드를 사용할 수 없다.

```
document.addEventListener('이벤트타입', function(event) {  
  console.log( event.type, event.target )  
}, true )
```

```
document.addEventListener('이벤트타입', function(event) {  
  console.log( event.type, event.target )  
}, {capture:true, passive:false, once:true} )
```

event는 type와 target
target은 document

이벤트타입: 키보드(p.291-292)
마우스(p.290) 등

등록 가능한 이벤트들

- DOMContentLoaded
- load
- reset
- submit
- resize
- scroll
- focus
- blur
- keydown
- keyup
- mouseenter
- mouseover
- mousedown
- mouseup
- click
- contextmenu

이벤트 삭제

- 등록한 이벤트 타입과 콜백함수 이름을 알아야 삭제 가능

```
function callbackfunc(e) {  
  console.log(e.target.value + 'click~~');  
}  
document.addEventListener('click' , callbackFunc )  
document.removeEventListener('click' , callbackFunc )
```

- 이벤트 삭제함수는 별도 반환 값이 없기 때문에 삭제 성공 여부를 알수 없음

이벤트 객체 얻기 매우중요

- 이벤트 발생시킨 대상이 누구인지, 또는 이벤트의 발생 위치가 어떻게 되는지 정보를 구하기 위해서는 이벤트 객체를 먼저 구해야한다.

```
//IE
function doClickOnBtn1() {
    var e = window.event;
}

//not IE
function doClickOnBtn1(e) {
}

//All
function doClickOnBtn1(e) {
    var e = window.event || e;
}
```

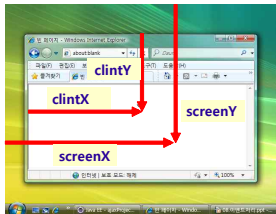
마우스 이벤트

이벤트	설 명
click	마우스 버튼 클릭시 발생
dblclick	마우스 버튼 더블 클릭 시 발생(비표준)
mousedown	마우스 버튼을 눌렀을 때
mouseup	마우스 버튼을 뗐을 때
mouseover	마우스가 영역 안으로 들어왔을 때
mousemove	마우스가 영역 안으로 이동할 때
mouseout	마우스가 영역 밖으로 나갈 때

마우스 이벤트 발생 위치 구하기

- clintX, clintY : 문서가 출력되는 영역에서
- screenX, screenY : 전체화면에서
- scrollLeft, scrollTop : 스크롤된 값

p.291



마우스 클릭 버튼 판별하기

프로퍼티값	의미
0	일반 버튼(왼쪽 버튼)
1	추가버튼(가운데, 휠)
2	컨텍스트 버튼(오른쪽 버튼)

```
var isRight = event.button == 2;
```

키보드 이벤트

• 이벤트 종류

이벤트	설명
keydown	키보드를 누를 때 발생한다.
keyup	키보드에서 떼를 때 발생한다.
keypress	키보드를 누를 때 발생한다.(비표준)

• 이벤트 객체

프로퍼티	의미
keyCode	키코드 값
ctrlKey	Ctrl 키를 누른 경우 true
shiftKey	Shift 키를 누른 경우 true
altKey	Alt 키를 누른 경우 true

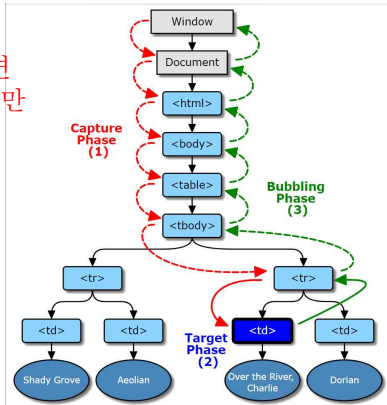
이벤트 캡처링 버블링 비교

p.279

- 이벤트 캡처링
 - 최상위노드에서 이벤트가 발생한 타겟 노드까지 내려가는 탐색
- 이벤트 버블링
 - 타겟노드에서 최상위노드까지 올라오는 탐색과정
 - 버블링 과정을 통해서 이벤트가 전파
- 이벤트 전파 중지
 - 상위 요소의 이벤트 콜백 함수가 실행되는 것을 막음
 - `event.stopPropagation()`

이벤트 캡처링 버블링 비교

부모에게 걸어놓으면
자자손손 다 걸리게 만
들어 전파함

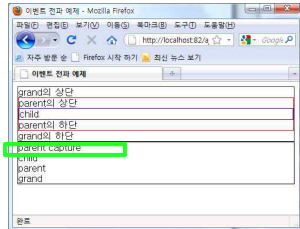
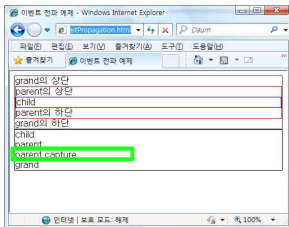


이벤트 캡처링 버블링 비교

```
<div id="grand" style="border: 1px solid #333">grand의 상단  
  <div id="parent" style="border: 1px solid red">parent의 상단  
    <div id="child" style="border: 1px solid blue">child  </div>  
  </div>  
</div>
```

```
<script>  
var grand = document.getElementById("grand");  
var parent = document.getElementById("parent");  
var child = document.getElementById("child");  
  
grand.addEventListener("mousedown", grandHandler, false);  
parent.addEventListener("mousedown", parentHandlerCapture, true);  
parent.addEventListener("mousedown", parentHandler, false);  
child.addEventListener("mousedown", childHandler, false);  
</script>
```

이벤트 흐름



DOMContentLoaded 이벤트

- DOM 제어의 시작점으로 HTML 페이지가 화면에 표시(렌더링)되기 전에
- DOM 생성 완료 직후 발생
- 자바스크립트를 이용해 DOM에 접근해서 제어할 수 있음

```
document.addEventListener('DOMContentLoaded', function(){  
    //여기에 DOM 제어에 필요한 코드와 실행할 함수를 호출 ex)AJAX  
})
```

마우스 중복 클릭 방지

- 버튼 비활성화

```
<body>
  <button id="saveBtn">저장</button>
  <script>
    saveBtn.addEventListener('click', function(e){
      this.setAttribute('disabled', 'disabled')
    })

    saveConfirmBtn.addEventListener('click', function(e){
      if(confirm('저장하시겠습니까?')){
        //폼전송
        console.log('등록 됨')
      } else {
        return;
      }
    })
  </script>
```

마우스 중복 클릭 방지

- confirm() 메시지 창으로 중복 클릭 차단

```
<body>
  <button id="saveConfirmBtn">확인후저장</button>
  <script>
    saveConfirmBtn.addEventListener('click',
function(e){
  if(confirm('저장하시겠습니까?')){
    //폼전송
    console.log('등록됨')
  } else {
    return;
  }
})
</script>
```

마우스 중복 클릭 방지

- 클릭 체크 변수로 중복 클릭 차단

```
<button id="savecheckBtn">(상태변경)저장</button>
<script>
  let submitFlag = false;
  function submitCheck(){
    if(submitFlag) {
      return submitFlag;
    } else {
      submitFlag = true;
      return false;
    }
  }
  savecheckBtn.addEventListener('click', function(e){
    if(submitCheck()) {
      //폼전송
      console.log('등록됨')
    }
  })
</script>
```

preventDefault()

- 기본 이벤트 전파 중지
 - 미리 정의된 방식으로 동작하는 이벤트 : submit 버튼, <a>
 - event.preventDefault()

stopPropagation()

- 타겟의 콜백 함수 및 기본 동작을 실행하지만 상위로의 이벤트 전파는 차단.