CS Study 22주차

Event Propagation

김신아

HTML 태그

```
<html>
    <head>
        <style>
            html{border:5px solid red; padding:30px;}
            body{border:5px solid green; padding:30px;}
            fieldset{border:5px solid blue; padding:30px;}
            input{border:5px solid black; padding:30px;}
        </style>
    </head>
    <body>
        <fieldset>
            <legend>event propagation</legend>
            <input type="button" id="taget" value="target">
        </fieldset>

        <script>
        ...
        </script>

    </body>
</html>

```

HTML 태그는 중첩되어 있다. 따라서 특정한 태그에서 발생하는 이벤트는 중첩되어 있는 태그들 모두가 대상이 될 수 있다.
이런 경우 중첩된 태그들에 이벤트가 등록 되어 있다면 어떻게 처리 될까?

이벤트 등록

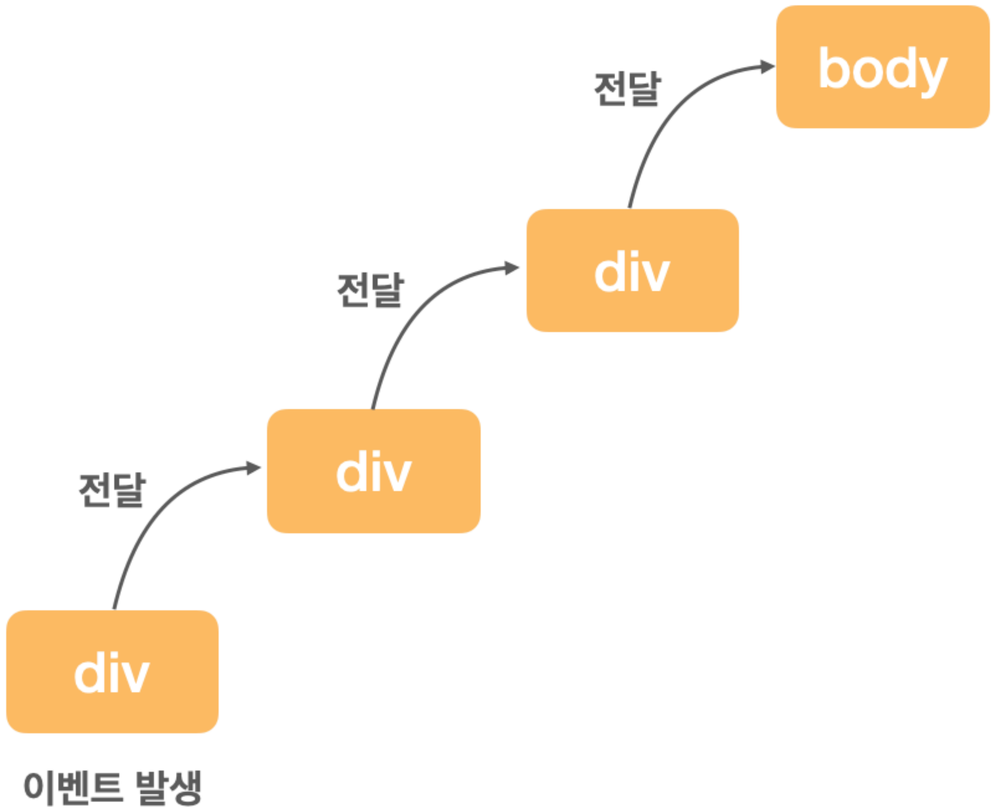
```html
<button>add one item</button>
```

```javascript
var button = document.querySelector('button');
button.addEventListener('click', addItem);

function addItem(event){
    console.log(event);
}
```

addEventListener() 웹 API는 웹 개발자들이 화면에 동적인 기능을 추가하기 위해 자연스럽게 접하게 되는 기본적인 기능
사용자의 입력에 따라 추가 동작을 구현할 수 있는 방법

이벤트 버블링 - Event Bubbling

특정 화면 요소에서 이벤트가 발생했을 때 해당이벤트가 더 상위의 화면 요소들로 전달되어 가는 특성을 의미



상위의 화면 요소란? HTML 요소는 기본적으로 트리 구조를 갖는다.
여기서는 트리 구조상으로 한 단계 위에 있는 요소를 상위 요소라고 하며 body 태그를 최상위 요소라 하겠다.

이벤트 버블링 - Event Bubbling

특정 화면 요소에서 이벤트가 발생했을 때 해당이벤트가 더 상위의 화면 요소들로 전달되어 가는 특성을 의미

<코드>

```
<body>
    <div class="one">
        <div class="two">
            <div class="three">
            </div>
        </div>
    </div>
</body>


var divs = document.querySelectorAll('div');
divs.forEach(function(div){
    div.addEventListener('click', logEvent);
});

function logEvent(event){
    console.log(event.currentTarget.className);
}
```

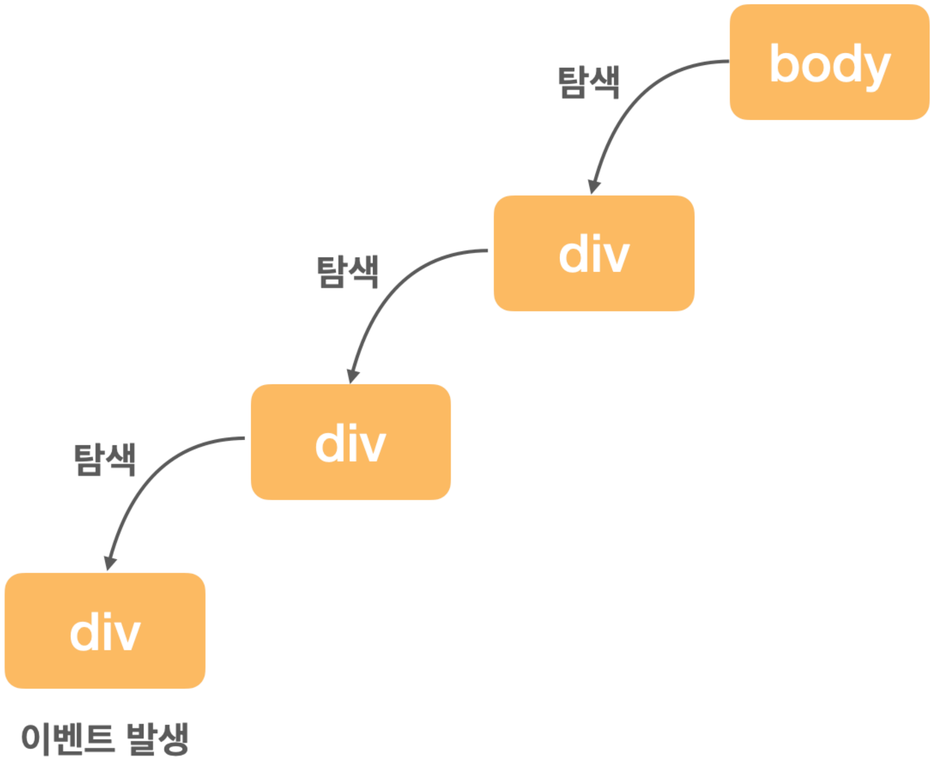
<실행결과>

// three 클래스를 갖는 div 태그를 클릭

```
three
two
one
```

이벤트 캡쳐링 - Event Capturing

특정 화면 요소에서 이벤트가 발생했을 때 최상위 요소인 body 태그에서 해당 태그를 찾아 내려감



이벤트 캡쳐링 - Event Capturing

특정 화면 요소에서 이벤트가 발생했을 때 최상위 요소인 body 태그에서 해당 태그를 찾아 내려감

<코드>

```
<body>
    <div class="one">
        <div class="two">
            <div class="three">
            </div>
        </div>
    </div>
</body>


var divs = document.querySelectorAll('div');
divs.forEach(function(div){
    div.addEventListener('click', logEvent, {
        capture: true // default 값은 false
    });
});

function logEvent(event){
    console.log(event.currentTarget.className);
}
```

addEventListener() API에서 옵션 객체에 capture: true를 설정해주면 해당 이벤트를 감지하기 위해 이벤트 버블링과 반대 방향으로 탐색

<실행결과>

// three 클래스를 갖는 div 태그를 클릭

```
one
two
three
```

event.stopPropagation()

이벤트 버블링과 캡쳐를 피하기 위한 방법으로 사용 아래 API는 해당 이벤트가 전파되는 것을 막음
이벤트 버블링의 경우 클릭한 요소의 이벤트만 발생시키고 상위 요소로 이벤트를 전달하는 것을 방해
이벤트 캡쳐링의 경우 클릭한 요소의 최상위 요소의 이벤트만 동작시키고 하위 요소들로 이벤트를 전달하지 못함

```
function logEvent(event){
    event.stopPropagation();
}
```

event.stopPropagation()

```
// 이벤트 버블링 예제
divs.forEach(function(div) {
    div.addEventListener('click', logEvent);
});

function logEvent(event){
    event.stopPropagation();
    console.log(event.currentTarget.className); // three
}
```

```
// 이벤트 캡쳐링 예제
divs.forEach(function(div) {
    div.addEventListener('click', logEvent, {
        capture: true // default 값은 false
    });
});

function logEvent(event){
    event.stopPropagation();
    console.log(event.currentTarget.className); // one
}
```

event.target & event.currentTarget

event.currentTarget == the element to which the event handler has been attached.

> The currentTarget read-only property of the Event interface identifies the current target for the event, as the event traverses the DOM. **It always refers to the element to which the event handler has been attached, as opposed to Event.target**, which identifies the element on which the event occurred and which may be its descendant.

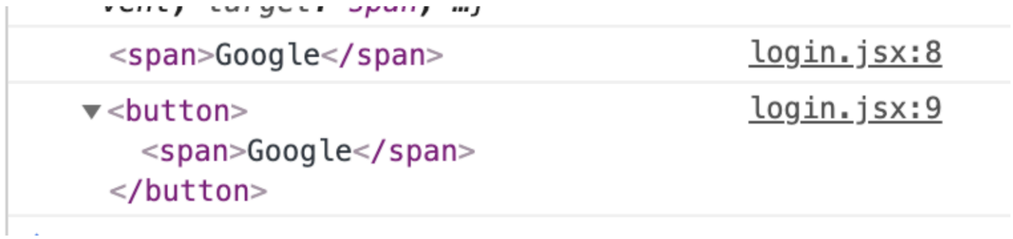
currentTarget은 이벤트 핸들러가 부착된 것을 가리킨다.
즉, event.target은 부모로부터 이벤트가 위임되어 발생하는 자식의 위치, 내가 클릭한 자식 요소를 반환한다.
하지만 currentTarget은 이벤트가 부착된 부모의 위치를 반환한다.

event.target & event.currentTarget

<코드>

```
<li>
    <button onClick={onLogin}>
        <span>Google</span>
    </button>
</li>

const onLogin = (event) => {
    console.log(event.target);
    console.log(event.currentTarget);
};
```

<실행결과>



event.target은 자식 요소인 span을 반환하고, event.currentTarget은 부모 요소인 button을 반환

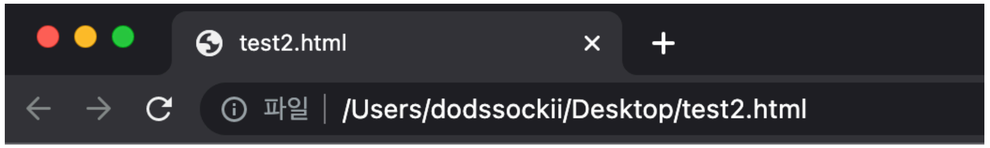
이벤트 위임 - Event Delegation

하위 요소에 각각 이벤트를 붙이지 않고 상위 요소에서 하위 요소의 이벤트들을 제어하는 방식

<코드>

```
<h1> 오늘의 할 일 </h1>
<ul>
    <li>
        <input type="checkbox" id="item1">
        <label for="item1">이벤트 버블링 학습</label>
    </li>
    <li>
        <input type="checkbox" id="item2">
        <label for="item2">이벤트 캡쳐링 학습</label>
    </li>
</ul>

var inputs = document.querySelectorAll('input');
inputs.forEach(function(input){
    input.addEventListener('click', function(event){
        alert('clicked');
    });
});
```

<실행결과>



오늘의 할 일

- ☐ 이벤트 버블링 학습
- ☐ 이벤트 캡쳐 학습

이벤트 위임 - Event Delegation

할 일이 더 생겨서 리스트 아이템을 추가해야한다면?

```
// ...

// 새 리스트 아이템을 추가하는 코드
var itemList = document.querySelector('.itemList');

var li = document.createElement('li');
var input = document.createElement('input');
var label = document.createElement('label');
var labelText = document.createTextNode('이벤트 위임 학습');

input.setAttribute('type', 'checkbox');
input.setAttribute('id', 'item3');
label.setAttribute('for', 'item3');
label.appendChild(labelText);
li.appendChild(input);
li.appendChild(label);
itemList.appendChild(li);
```

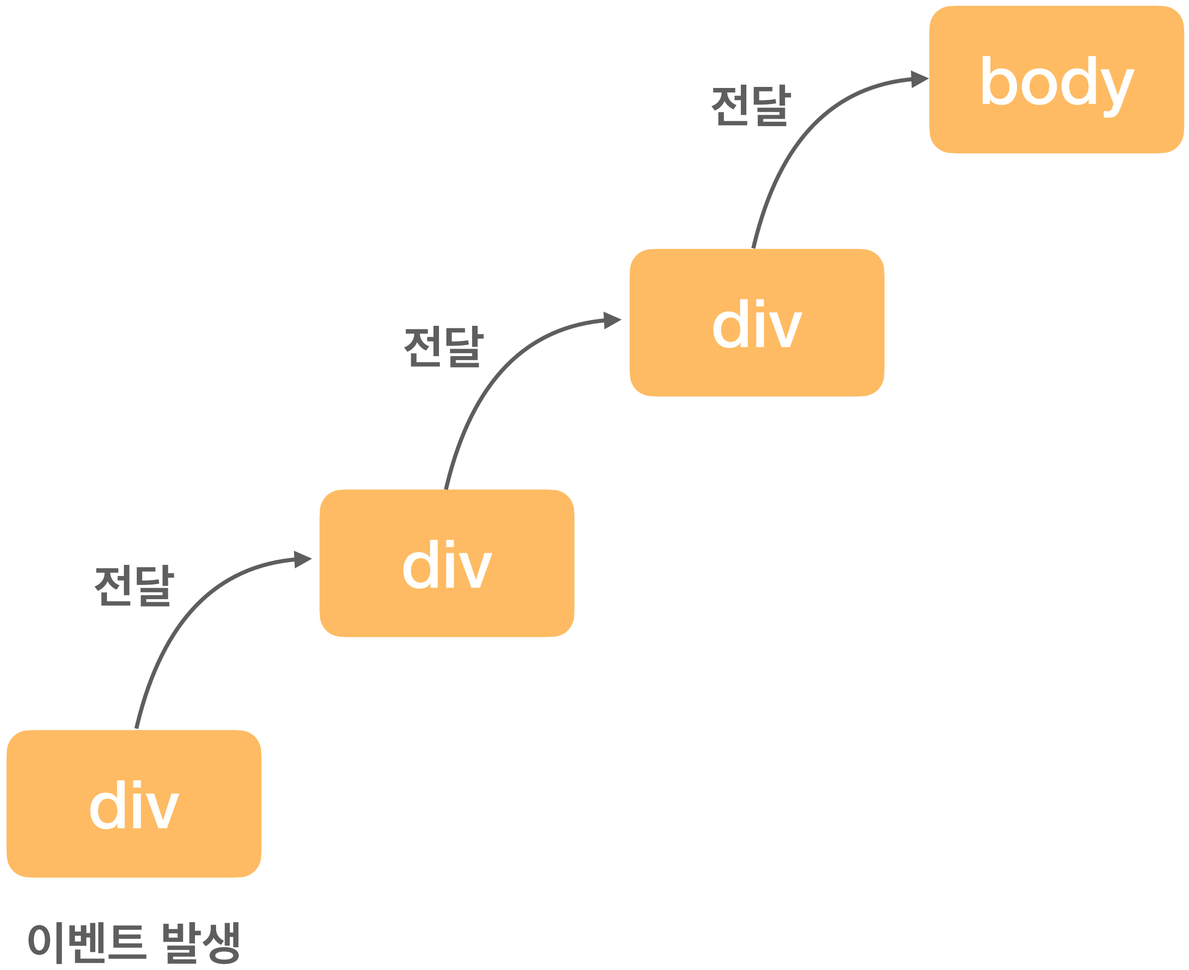
이벤트 위임 - Event Delegation

```
// var inputs = document.querySelectorAll('input');
// inputs.forEach(function(input) {
//  input.addEventListener('click', function() {
//      alert('clicked');
//  });
// });

var itemList = document.querySelector('.itemList');
itemList.addEventListener('click', function(event) {
    alert('clicked');
});

// 새 리스트 아이템을 추가하는 코드
// ...
```

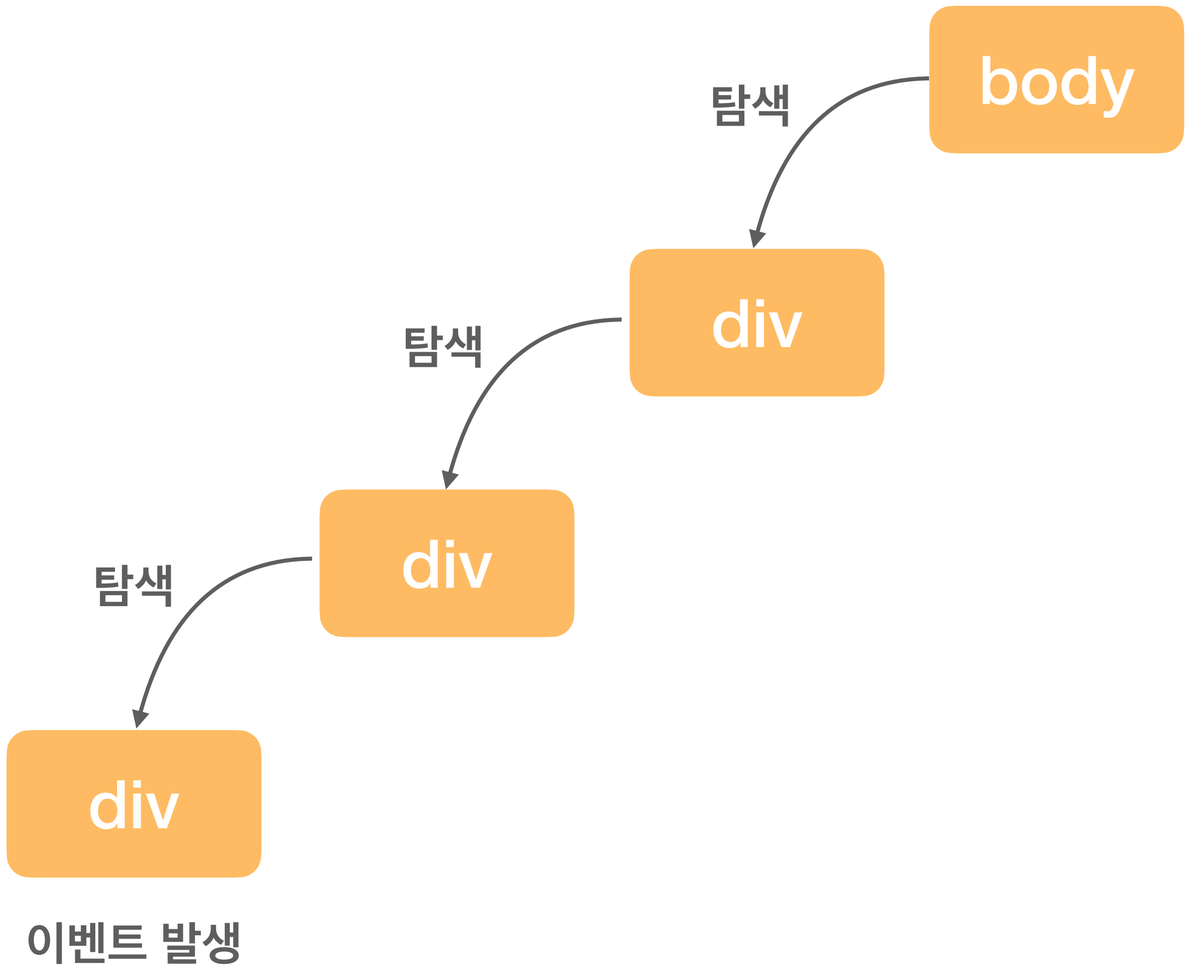
모든 input 박스에 일일이 이벤트 리스너를 추가하는 대신 input 박스의 상위 요소인 ul 태그, .itemList에 이벤트 리스너를 달아놓고 하위에서 발생한 클릭 이벤트를 감지. 여기서 쓰인 방법이 이벤트 버블링.

Thank You

별첨



별첨



별첨

https://joshua1988.github.io/web-development/javascript/event-propagation-delegation/#%EB%93%A4%EC%96%B4%EA%B0%80%EB%A9%B0

https://www.youtube.com/watch?v=6riJ7r6HF3o

https://velog.io/@edie_ko/JavaScript-event-target%EA%B3%BC-currentTarget%EC%9D%98-%EC%B0%A8%EC%9D%B4%EC%A0%90