이벤트 위임

이벤트 위임은 **하위 요소에 각각 이벤트를 붙이지 않고, 상위 요소에서 하위 요소의 이벤트들을 제어하는 방식**을 의미한다.

스프레드 문법

스프레드 문법은 **... 연산자**를 사용하여 배열이나 문자열을 개별 요소로 분해하여 결합할 수 있다.

```
const arr = [1, 2, 3];
const newArr = [...arr]; //arr을 펼쳐서 새로운 배열 newArr에 복사한다.
console.log(newArr); // 출력 결과: [1, 2, 3]


//문자열
const str1 = "javascript";
const str2 = [...str1];
console.log(str2);
// 출력 결과 : ["j", "a", "v", "a", "s", "c", "r", "i", "p", "t"];

// 배열과 배열을 결합
const arr1 = [1, 2, 3];
const arr2 = [4, 5, 6];
const mergedArr = [...arr1, ...arr2];
console.log(mergedArr); // 출력 결과: [1, 2, 3, 4, 5, 6]
```

사용자가 내비게이션 아이템(li요소)을 클릭하면 active클래스를 추가하고 그 외의 모든 내비게이션 아이템의 active클래스를 제거하는 구문

```
<style>
#fruits {  display: flex;  list-style-type: none;  padding: 0;  }
#fruits li {  width: 100px;  cursor: pointer;  }
#fruits .active {  color: red;  text-decoration: underline;
```

```
}
</style>

<nav>
  <ul id="fruits">
    <li id="apple" class="active">Apple</li>
    <li id="banana">Banana</li>
    <li id="orange">Orange</li>
  </ul>
</nav>
<div>선택된 내비게이션 아이템: <em class="msg">apple</em></div>

<script>
const $fruits = document.getElementById('fruits');
const $msg = document.querySelector('.msg');

// 사용자 클릭에 의해 선택된 내비게이션 아이템(li 요소)에 active 클래스를 추가하고
// 그 외의 모든 내비게이션 아이템의 active 클래스를 제거한다.
function activate({ target }) {
  [...$fruits.children].forEach($fruit => {
    $fruit.classList.toggle('active', $fruit === target);
    $msg.textContent = target.id;
  });
}

// 모든 내비게이션 아이템(li 요소)의 부모요소인 $fruits요소에 이벤트를 바인딩한다.
$fruits.onclick = activate;
// currentTarget프로퍼티는 $fruits요소를 가리키지만, target프로퍼티는 실제로 이벤트를 발생시킨 DOM요소를 가리킨다.
</script>
```

이벤트 핸들러 내부의 this

```
<button class="btn1">0</button>
<button class="btn2">0</button>
```

```
<script>
const $button1 = document.querySelector('.btn1');
const $button2 = document.querySelector('.btn2');

// 이벤트 핸들러 프로퍼티 방식
$button1.onclick = function (e) {
  // this는 이벤트를 바인딩한 DOM 요소를 가리킨다.
  console.log(this); // $button1
  console.log(e.currentTarget); // $button1
  console.log(this === e.currentTarget); // true

  // $button1의 textContent를 1 증가시킨다.
  ++this.textContent;
};

// addEventListener 메서드 방식
$button2.addEventListener('click', e => {
  // 화살표 함수 내부의 this는 상위 스코프의 this를 가리킨다.
  console.log(this); // window
  console.log(e.currentTarget); // $button2
  console.log(this === e.currentTarget); // false

  // this는 window를 가리키므로 window.textContent에 NaN(undef
ined + 1)을 할당한다.
  ++this.textContent;
});
</script>
```

각 버튼 별로 클릭할 경우 "버튼명"이 클릭 되었음을 alert(경고창)으로 표시해보자

```
<ul class="itemlist">
    <li><input type="button" value="button1"></li>
```

```
    <li><input type="button" value="button2"></li>
</ul>
```

JS

```
//buttons변수에 input요소들을 nodeList로 할당
let buttons = document.querySelectorAll('input');

buttons.forEach(button => button.addEventListener('click' ,
event => alert(`${event.target.value} clicked!`)))
/*buttons변수에 할당한 input요소들을 선택하고 forEach문으로 input요
소를 순환 하면서 click이벤트 생성 => alert으로 input요소의 value(입
력값)을 출력하고 clicked!라는 문자를 출력*/

//============================================================
=======

// 버튼이 동적으로 계속 추가 되는 구조일 경우
// button 3 추가
//itemlist변수에 .itemlist를 할당
let itemList = document.querySelector('.itemlist');
//li변수에 li요소를 생성하여 할당
let li = document.createElement('li');
//input변수에 input요소를 생성하여 할당
let input = document.createElement('input');
//생성한 input요소에 type속성을 button으로 설정
input.setAttribute('type', 'button');
//생성한 input요소의 value값을 button3으로 설정
input.setAttribute('value', 'button3');
//생성한 li요소의 마지막 자손요소로 생성한 input요소를 추가
li.appendChild(input);
//.itemlist요소에 생성한 li요소를 마지막 자손으로 추가
itemList.appendChild(li);


//button1과 button2는 클릭할 경우 정상적으로 alert(경고창)이 작동하
지만 button3은 이벤트 핸들러를 등록하지 않았기 때문에 클릭하더라도 아무
런 일도 발생하지 않는다.
```

```
// 다른 버튼처럼 클릭 이벤트가 활성화되기 위해서는 아래처럼 이벤트를 추가하면 된다.
input.addEventListener('click', event => alert(`${event.target.value} clicked!`))
```

이벤트위임(리팩토링)

```
<button type="button" class="btn1">버튼추가</button>
<button type="button" class="btn2">버튼제거</button>

<ul class="itemlist">
      <li><input type="button" value="button1"></li>
      <li><input type="button" value="button2"></li>
</ul>
```

JS

```
//매번 요소가 새로 추가될 때마다 이벤트리스너를 새로 작성 하는 것은 번잡하다.
//아래 코드처럼 리팩토링 할 수 있다.

//itemlist변수에 .itemlist를 할당
let itemlist = document.querySelector('.itemlist');
/*itemlist에 click이벤트 생성 alert으로 .itemlist노드 안에 있는 요소(input)의 value(입력값)를 clicked!문자와 같이 출력*/
itemlist.addEventListener('click', event => alert(event.target.value + " clicked!"));

//btn변수에 button요소를 할당.
let btn1 = document.querySelector(".btn1");

let i = 2;//변수 i에 숫자 2를 할당 => button글자뒤에 붙는 순번
btn1.addEventListener("click", () => { //btn에 클릭이벤트 생성
    i++; //변수 i에 1을 더해 다시 변수 i에 할당
```

```
    let li = document.createElement('li'); //li변수에 li요소를 생성하여 할당
    let input = document.createElement('input'); //input변수에 input요소를 생성하여 할당

      //생성한 input요소에 type속성을 button으로 설정
    input.setAttribute('type', 'button');
    //생성한 input요소의 value값을 button + i로 설정
    input.setAttribute('value', 'button'+i);
    //생성한 li요소의 마지막 자손요소로 생성한 input요소를 추가
    li.appendChild(input);
    //.itemlist요소에 생성한 li요소를 마지막 자손으로 추가
    itemlist.appendChild(li);
});
/*이벤트 버블링의 원리를 활용해 하위에서 발생한 클릭 이벤트를
상위 요소인 ul.itemlist 태그에서 감지하는 것*/

document.querySelector(".btn2").addEventListener('click',
() => {
  const li = document.querySelector(".itemlist > li:last-child")
  i--;
  li.remove();//마지막 버튼요소 제거
})
```