이벤트 위임

버블링과 캡쳐링

이벤트 위임을 알기 위해선 **이벤트 캡쳐링**과 **이벤트 버블링**이란 개념을 알아야 합니다. 부모의 이벤트가 자식에게 전달되는 현상을 **이벤트 캡쳐링**, 자식의 이벤트가 조상들에게까지 전달되는 현상을 **이벤트 버블링**이라고 합니다.

이벤트 버블링

특정 엘리먼트에 이벤트가 발생하면 **해당 엘리먼트가 그 엘리먼트의 조상들에게까지 전달**되는 현상입니다. 아래 예시를 보겠습니다.

```
<!DOCTYPE html>
<html lang="en">
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initia
    <title>Example</title>
    <link rel="stylesheet" href="style.css">
        <script></script>
</head>
<body>
    body
    <div>
        div
        <p>p</p>
    </div>
    <script src="app.js"></script>
</body>
```

```
</html>
```

```
const $p = document.querySelector('p')
const $div = document.querySelector('div')
const $body = document.querySelector('body')

function Alert(message) {
  return function() {
    alert(message)
  }
}

$p.addEventListener('click', Alert('p tag event'))
$div.addEventListener('click', Alert('div tag event'))
$body.addEventListener('click', Alert('body tag event'))
```

위 예시에서 가장 내부의 p태그를 클릭하면 해당 이벤트가 부모인 div, 조상인 body까지 전달되어 “alert p” > “alert div” > “alert body”가 순차적으로 실행되고, 이를 **이벤트 버블링**이라고 합니다.

이벤트 버블링이 일어났을 때 최초로 이벤트를 발생시킨 엘리먼트를(위 예제에서는 p태그) **타겟 엘리먼트** 라고 하고, 이는 event.target을 통해 접근이 가능합니다.

▼ this vs event.target

- event.target : 최초로 이벤트를 발생시킨 엘리먼트를 가리킵니다.
- this(=event.currentTarget) : 현재 이벤트가 발생된 엘리먼트를 가리킵니다.

위 예시에 이어 아래 예시를 통해 이벤트가 버블링된 부모가 타겟 엘리먼트를 기억하고 있음을 알 수 있습니다.

```
const bodyElement = document.querySelector('body')
```

```
function Alert(event) {
    alert(`타겟 엘리먼트: ${event.target.tagName} 현재 엘리먼트: $
}

bodyElement.addEventListener('click', Alert)
```

p 태그를 클릭하면 **타겟 엘리먼트: P 현재 엘리먼트: BODY** 가 출력된다.

그렇다면 버블링을 멈추는 방법은 없을까요?

버블링은 대체로 <html> 엘리먼트까지 올라갑니다. 이러한 이벤트를 멈추기 위해서는 최초로 이벤트가 발생되는 엘리먼트의 이벤트 핸들러에 **event.stopPropagation()** 이라는 API를 추가해주면 됩니다. 만약 하나의 이벤트에 여러 핸들러가 붙어 있는 경우 위 함수를 추가해도 다른 이벤트는 버블링이 될텐데 모든 이벤트 버블링을 멈추고 싶은 경우에는 **event.stopImmediatePropagation()** API를 추가해주면 해결됩니다.

이벤트 캡쳐링

이벤트 캡쳐링은 특정 엘리먼트에 이벤트가 발생 했을 경우 이벤트가 **최상단의 부모 엘리먼트로부터 전달되어져 내려오는 현상**입니다. 따라서 전달되는 이벤트는 부모 엘리먼트의 이벤트 핸들러를 작동시킵니다.

캡쳐링을 수행하기 위해서는 이벤트 핸들러에 true로 캡쳐링 옵션을 주어야 합니다.
(default : false)

```
const elements = document.querySelectorAll('*')

for (let elem of elements) {
    elem.addEventListener('click', e => alert(`캡쳐링: ${elem.t
}
```

p 태그를 클릭하면 HTML > BODY > DIV > P 순서로 출력되는 것을 확인할 수 있습니다. 이와 같은 현상이 이벤트 캡쳐링입니다.

캡쳐링을 막는 방법 역시 **e.stopPropagation()** 이라는 함수를 사용하면 됩니다. 즉, 해당 함수 사용 시 버블링에서는 타겟 엘리먼트에만 이벤트가 발생하도록 해주고, 캡쳐링에서는 타겟 엘리먼트 기준으로 최상단 엘리먼트에만 이벤트가 발생하도록 해줍니다.

이벤트 위임

이벤트 위임은 캡쳐링과 버블링을 이용해, 여러 엘리먼트마다 각각 이벤트 핸들러를 할당하지 않고, 공통되는 부모에 이벤트 핸들러를 할당해 이벤트를 관리하는 방식입니다.

여러 개의 자식 엘리먼트 이벤트 관리하기

정해진 액션에 따라 다른 동작을 하는 여러 버튼에 대한 이벤트는 어떻게 처리해야 할까요? 모든 버튼에 대해 이벤트 리스너를 등록할 수도 있겠지만 이벤트 위임 방식을 통해서도 이벤트를 등록할 수 있습니다.

아래 예제는 서로 다른 역할을 하는 3개의 버튼의 이벤트를 어떻게 이벤트 위임 방식으로 처리하는지 보여줍니다.

```
<!DOCTYPE html>
<html lang="en">
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initia
    <title>Example</title>
</head>
<body>
    <div id="Menu">
        <button data-action="save">저장하기</button>
```

```
        <button data-action="reset">초기화 하기</button>
        <button data-action="load">불러오기</button>
    </div>
    <script src="app.js"></script>
</body>
</html>
```

```
const $Menu = document.getElementById('Menu')

const ActionFunctions = {
  save: () => alert('저장하기'),
  reset: () => alert('초기화하기'),
  load: () => alert('불러오기'),
}

$Menu.addEventListener('click', e => {
  const action = e.target.dataset.action
  if (action) {
    ActionFunctions[action]()
  }
})
```

이런 방식으로 상단 엘리먼트에서 자식에게 이벤트를 전달할 수 있습니다.

동적 엘리먼트에 대한 이벤트 관리하기

동적으로 추가되거나 삭제되는 엘리먼트에 대한 이벤트는 어떻게 처리해야할까요? 이에 대해 매번 이벤트 리스너를 추가하고 삭제한다면 코드의 효율성도 문제이며, 제대로 리스너가 삭제되지 않을 수도 있으므로 메모리 누수 가능성도 있습니다.

이 때, 이벤트 위임 패턴을 이용해 이벤트를 관리해주면 편해집니다. 아래 예제는 글자를 작성하고 submit 하면 ul 엘리먼트 아래에 li 엘리먼트를 추가해줍니다. 그리고 추가된 li 엘리먼트를 클릭하면 li 엘리먼트 내부 text를 내용을 alert를 띄워줍니다.

여기서 li 엘리먼트의 이벤트 처리시, li 엘리먼트마다 이벤트 리스너를 추가해주는 대신, 이벤트 위임 방식을 이용해 공통 부모인 ul 엘리먼트가 자식들의 이벤트를 관리하게 한 것입니다.