이벤트 기초 2

이벤트 위임 (@)

- 캡처링과 버블링을 활용하면 이벤트 핸들링 패턴인 **이벤트 위임(event delegation)을 구현** 가능
  - 이벤트 위임은 **비슷한 방식으로 여러 요소를 다뤄야 할 때 사용**됨
  - 이벤트 위임을 사용하면 **요소마다 핸들러를 할당하지 않고, 요소의 공통 조상에 이벤트 핸들러를 단 하나만 할당**해서 여러 요소에 필요한 공통 로직을 수행할 수 있음

event_delegation.html

```
<style>
.highlight { background: yellow; }
</style>
<script>
document.addEventListener('DOMContentLoaded', function() {
    let selected;

    function highlight(target) {
        if(selected) {
            selected.classList.remove('highlight');
        }
        selected = target;
        selected.classList.add('highlight');
    }

    // 선택된 리스트 요소를 강조해주고 타이틀 내용을 출력하는 작업 진행 (모두 공통적으로 해줘야 할 작업이라고 가정)
    list.onclick = function(e) {
        let li = event.target.closest('li');

        if(!li) return;
        else {
            let h1 = li.querySelector('h1');
            alert(h1.textContent);
            highlight(li);
        }
    }
});
</script>
<body>
    <ul id="list">
        <li>
            <div>
                <h1>title #1</h1>
                <p>paragraph #1</p>
            </div>
        </li>
        <li>
            <div>
                <h1>title #2</h1>
```

```
                <p>paragraph #2</p>
            </div>
        </li>
    </ul>
</body>
```

without_event_delegation.html

```
<style>
.highlight { background: yellow; }
</style>
<script>
document.addEventListener('DOMContentLoaded', function() {
    // 이벤트 위임 없이 똑같은 기능 구현하기
    let selected;
    document.querySelectorAll("#list li").forEach(li => {
        li.addEventListener('click', function(e) {
            if(selected) {
                selected.classList.remove('highlight');
            }
            const li = e.currentTarget;
            selected = li;
            li.classList.add('highlight');
            const h1 = li.querySelector('h1');
            alert(h1.textContent);
        });
    });
});
</script>
<body>
    <ul id="list">
        <li>
            <div>
                <h1>title #1</h1>
                <p>paragraph #1</p>
            </div>
        </li>
        <li>
            <div>
                <h1>title #2</h1>
                <p>paragraph #2</p>
            </div>
        </li>
    </ul>
</body>
```

브라우저 기본 동작

- 상당수 이벤트는 발생 즉시 브라우저에 의해 **특정 동작을 자동으로 수행**함

```
* 링크를 클릭하면 해당 URL로 이동
* 폼 전송 버튼을 클릭하면 서버로 폼을 전송
* 마우스 버튼을 누른 채로 글자 위에서 커서를 움직이면(=드래그하면) 글자가 강조되어 선택
```

- 브라우저 기본 동작을 취소할 수 있는 방법은 두 가지
  - **방법1)** event 객체에서 제공하는 **preventDefault 메서드를 호출** (*)
  - **방법2)** 핸들러가 addEventListener가 아닌 on<event> 속성을 사용해 할당되었다면, 이벤트 핸들러 **함수에서 false를 반환**하도록 코드 작성 가능

```
<!-- 방법 1 -->
<a href="/" onclick="event.preventDefault()">이곳을</a> 클릭해주세요.
<!-- 방법 2 -->
<a href="/" onclick="return false">이곳</a>
```

이벤트 위임 방식을 이용해서 리스트 내부의 a 태그 처리하기 (+ 기본 동작 취소)

```
<ul id="menu" class="menu">
    <li><a href="/html">HTML</a></li>
    <li><a href="/javascript">JavaScript</a></li>
    <li><a href="/css">CSS</a></li>
</ul>
<script>
// (이벤트 버블링을 이용하여) 상위의 ul 요소에서 내부 a 요소의 이벤트 처리
menu.onclick = function(event) {
    // 클릭된 요소가 a 요소인 경우에만 작동
    if (event.target.nodeName != 'A') return;

    let href = event.target.getAttribute('href');
    // 서버에서 데이터를 읽어오거나, UI를 새로 만든다거나 하는 등의 작업 코드를 작성
    alert(href);
    // 주소로 이동하지 않도록 브라우저 기본 동작을 취소
    // (addEventListener 메서드 호출이 아닌, onclick 속성에 함수를 대입하는 방식으로 처리했으므로 false 값의 반환이 가능함을 유의)
    return false;
};
</script>
```