# Javascript 이벤트

키워드 : 이벤트 (https://www.w3schools.com/js/js_events.asp),
캡처링&버블링 (https://www.w3schools.com/js/js_htmldom_eventlistener.asp)

우리는 자바스크립트 강의 초반부터 여기까지 이벤트를 다루기 위해 열심히 달려왔습니다. 지금까지는 사용자의 액션과 상호 작용할 수 없는 지극히 정적(static)인 웹 페이지에 대한 내용을 다뤘습니다. 자바스크립트는 사용자가 웹 페이지에 방문하고 특정 행동을 할 때, 그 행동에 맞는 처리를 할 수 있도록 미리 사용자의 액션에 대해 이름을 붙여 놓았습니다. 그 이름을 우리는 '이벤트(Event)'라고 부릅니다. 이번 장에서는 정적인 웹 페이지에 생기를 불어넣는 자바스크립트 이벤트에 대해서 알아보도록 하겠습니다.

## 이벤트

우선, 웹 페이지 방문자는 이미 작성된 HTML 코드를 통해 웹 페이지를 보게 됩니다. 이벤트라는 것은 방문자가 HTML 코드의 특정 엘리먼트에 대해 어떤 '행동'을 취하면, 어떤 '반응'을 할 수 있도록 자바스크립트 코드로 작성해놓는 것입니다.

우리가 잘 알고 있는 태그 중에 <a>태그가 있습니다. 사실, 이 <a>태그도 사용자가 '클릭'이라는 행동을 취하면, 그 행동에 맞는 액션을 취하게 되는 것입니다. 반대로 말하면, 이벤트에 대한 액션 처리 코드를 작성해두지 않으면, 사용자가 어떤 행동을 취하더라도 아무런 반응이 없는 심심한 웹 페이지가 되는 것입니다.

사용자가 웹 페이지에 방문하여 취할 수 있는 행동은 굉장히 여러 가지가 있습니다. 모든 이벤트를 한 번에 설명하기엔 내용이 굉장히 많습니다. 때문에 그 부분은 하나하나 알아보도록 하고, 이번 장에선 이벤트에 대해 이해하는 것을 위주로 작성하도록 하겠습니다.

우선, 자주 사용되는 대표적인 이벤트는 크게 세가지입니다.

- 웹 페이지 로딩이 끝난 경우
- 사용자 클릭에 대한 처리(마우스)
- 사용자 입력에 대한 처리(키보드)

위에 작성된 세 가지 이벤트는 우리가 처음 브라우저를 실행하고, 각종 웹 사이트에 방문할 때마다 일어나는 일들입니다. 그렇다면, 이러한 이벤트를 어떻게 처리할 수 있을까요?

## 이벤트 처리 방법(핸들러 등록)

이벤트를 처리하기 위해선 이벤트 핸들러라는 것을 등록해야 합니다. 우리는 아직 어떤 이벤트들이 존재하는지 모르기 때문에, 특정 엘리먼트를 '클릭'할 때 발생하는 onclick이벤트를 예로 이벤트를 처리하는 방법을 알아보겠습니다. 이벤트 처리를 위한 핸들러를 등록하는 방법에는 크게 세 가지가 있습니다.

#### 인라인 모델(속성으로 등록)

우선 속성으로 등록하는 방법은, 이벤트를 등록하는 가장 간단한 방법입니다. 이벤트를 등록할 엘리먼트에 속성처럼 등록합니다.

```
<!-- 기본 형식 -->
<element event = "javascript code">
<!-- 버튼 클릭시 등록 자바스크립트 myFunction 함수 호출 -->
<button onclick="myFunction();">버튼<button>
```

#### 익명 함수 선언(DOM 객체 탐색)

익명 함수로 등록하는 방법은 DOM 객체로 엘리먼트를 우선 탐색하고, 찾은 엘리먼트의 속성(Property)처럼 이벤트를 작성하는 방법입니다.

```
//기본형식
document.getElementById("id").event = function(){
    실행코드...
}

document.getElementById("test").onclick = function(){
    alert("클릭 이벤트!");
}
```

**이벤트 리스너 등록**

이벤트 리스너로 등록하는 방법은 W3C DOM에 명시된 이벤트 리스너를 등록하기 위한 방법입니다. 위에서 살펴본 인라인, 익명 함수 등록 방법은 똑같은 이벤트를 하나의 엘리먼트에 중복 등록하는 것이 불가능합니다. 반면에, 이벤트 리스너로 등록하면, 똑같은 이벤트라도 중복 등록하는 것이 가능합니다.

```
//기본 형식
document.getElementById("id").addEventListenr("Event", "Anonymous function",
"Capture")

document.getElementById("test").addEventListenr("click", function(e){
    alert("클릭 이벤트1!");
}, false);
//중복 등록 가능
document.getElementById("test").addEventListenr("click", function(e){
    alert("클릭 이벤트2!");
}, false);
```

이벤트 리스너 부분은 덧 붙일 내용이 있기 때문에 더 정리해보도록 하겠습니다. 우선, event 부분에 이벤트는 'on' 키워드가 붙지 않습니다. 클릭 이벤트라면 바로 'click'을 작성해줍니다.

다음은, Anonymous function 부분입니다. 익명 함수는 우리가 잘 알고 있는 이름이 없는 함수를 작성해줍니다. 만약 함수를 호출하고 싶을 경우엔, 익명 함수 안에 함수를 호출하는 코드를 작성하면 됩니다. 한 가지 독특한 점이 있는데요, 익명 함수의 인자로 e라는 인자를 받아오는 것을 확인할 수 있습니다.

이 e(이름은 개발자 마음대로입니다.)라는 인자는 발생한 이벤트에 대한 여러 가지 속성과 메서드 등을 반환하는 역할을 할 수 있는 인자입니다. 이 인자는 특별한 역할을 하며, 이벤트 핸들러 등록 시에 선언하면 자동으로 이벤트 관련 값을 반환해준다고 기억하시면 됩니다.

마지막으로 capture 부분은 아래에서 추가로 설명하도록 하겠습니다.

**addEevntListener vs attachEvent**

다음은 호환성과 관련된 이야기입니다. 강의에서 잠시 언급했듯이, addEventListener로 이벤트를 등록하는 방법은 W3C에서 권장하는 표현이지만, IE 하위 버전(IE 8.0 이하)과 Opera 하위 버전(Opera 6.0 이하)에서는 정상 동작하지 않습니다. 때문에 하위 버전 브라우저에서도 이벤트 핸들러가 정상적으로 동작하도록 하려면 아래와 같은 코드를 작성해주어야 합니다.

```
var el = document.getElemById("id");
function changeText(){
    document.getElementById("test").innerHTML = "changeText 함수 호출!";
}
//1번 코드(모던 브라우저)
if (el.addEventListener) {
  el.addEventListener('click', changeText, false);
//2번 코드(IE 하위 버전)
} else if (el.attachEvent)  {
  el.attachEvent('onclick', changeText);
}
```

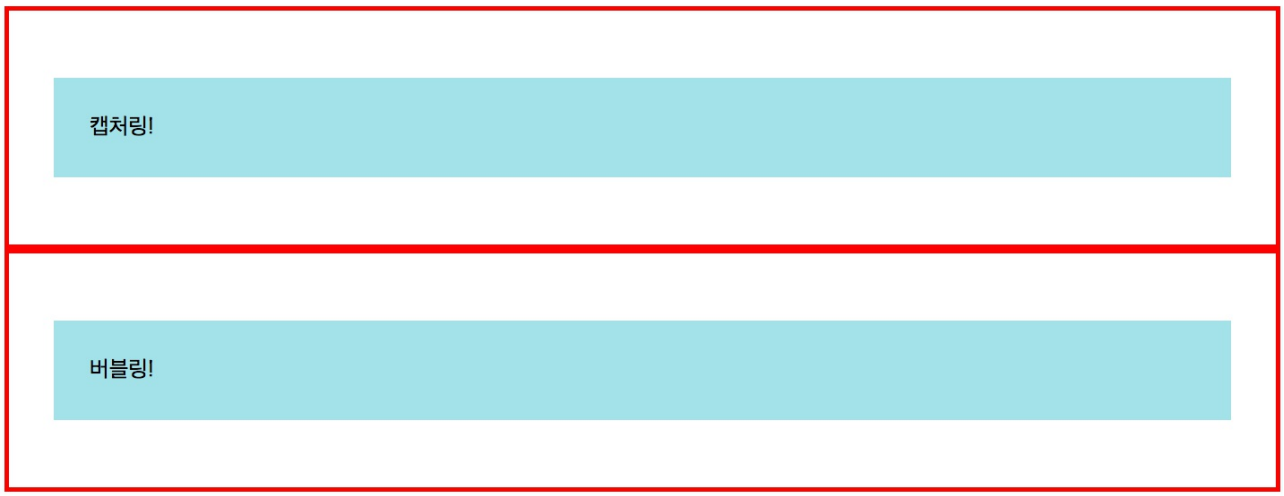
우선, 1번 코드의 경우엔 addEventListener, 2번 코드의 경우는 attachEvent 속성 값으로 `if ~ else if` 구문에서 분기하고 있는 것을 확인할 수 있습니다. 이렇듯 각각의 속성을 통해 분기할 수 있도록 코드를 작성하고, 하위 버전에서는 2번 코드처럼 attachEvent 메서드를 호출해서 작성해야 합니다. 단, IE 11 버전 이상부터는 attachEvent 메서드가 아닌 addEventListener 메서드를 이용해서 등록해야 합니다.(브라우저 호환성을 다루기가 조금 까다롭죠..? ^^)

## 이벤트 전파(Propagation), 캡처링과 버블링(Capturing & Bubbling)

우리는 HTML 엘리먼트를 구성할 때 부모 <-> 자식 관계로 코드를 작성하게 됩니다. 일반적으로 태그를 범용 요소로서 부모 태그로 지정하고, 태그 안에 여러 태그를 작성하게 됩니다. 우리가 엘리먼트에 이벤트를 등록하는 경우에 부모 엘리먼트와 하위 엘리먼트의 이벤트가 연쇄적으로 일어날 수 있습니다. 이것을 보통 이벤트 전파(Propagation)이라고 부릅니다. 이때, 일어나는 순서에 따라 구분 지은 것을 캡처링(Capturing)과 버블링(Bubbling)이라고 합니다. 아래 코드는 간단한 이벤트 캡처링과 버블링 코드 예시입니다.

- inedx.html

```html
<!doctype html>
<head>
    <meta charset="UTF-8">
    <title>Javascript 이벤트 Capturing & Bubbling.</title>
    <style>
        div{
            border: red solid 5px;
            padding: 50px;
        }
        p{
            background-color: powderblue;
            padding: 10px;
            font-size: 15px;
        }
    </style>
</head>
<body>
    <div id="capture">
        <p>캡처링!</p>
    </div>
    <div id="bubble">
        <p>버블링!</p>
    </div>
    <script type="text/javascript" src="js/main.js"></script>
</body>
</html>
```



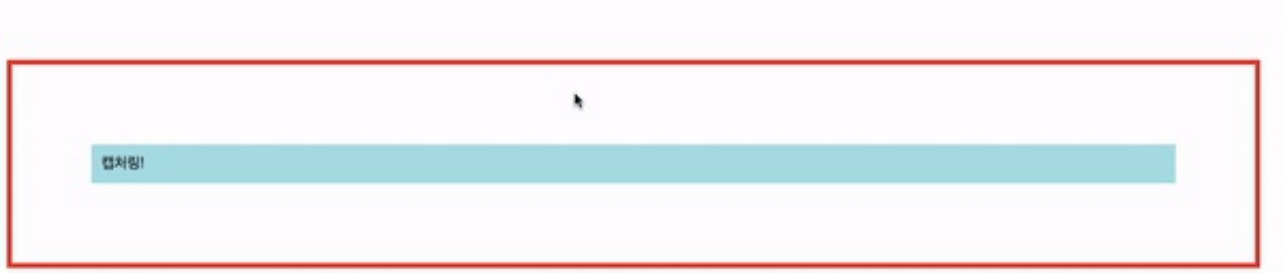

- main.js

```
//1번 코드
document.getElementById("capture").addEventListener("click", function(){
    alert("캡처 div 클릭!");
}, true);
//2번 코드
document.querySelector("div#capture > p").addEventListener("click", function(){
    alert("캡처 p 클릭!");
}, true);
//3번 코드
document.getElementById("bubble").addEventListener("click", function(){
    alert("버블 div 클릭!");
}, false);
//4번 코드
document.querySelector("div#bubble > p").addEventListener("click", function(){
    alert("버블 p 클릭!");
}, false);
```

우선 자바스크립트 코드를 먼저 살펴보겠습니다. DOM 객체를 이용해 엘리먼트를 탐색하는 코드는 어렵지 않게 이해하실 거라 생각하고, addEventListener 위주로 작성하도록 하겠습니다.

- 1번 코드는 id 속성이 capture인  태그를 클릭할 경우 alert("캡처 div 클릭!")을 호출하도록 하고 있습니다. 단, 이벤트 리스너의 마지막 인자가 true입니다.
- 2번 코드는 id 속성이 capture인  태그의 자식 <p> 태그를 클릭할 경우 alert("캡처 p 클릭!")을 호출하도록 하고 있습니다. 단, 이벤트 리스너의 마지막 인자가 true입니다.

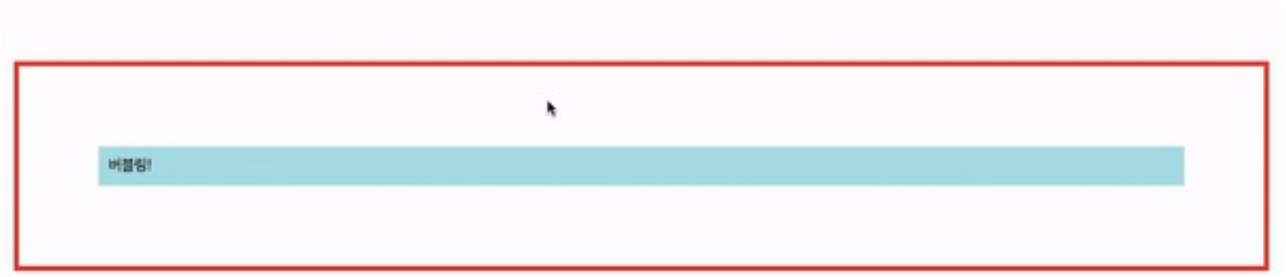
우선, 해당 코드의 결과가 어떻게 동작하는지 이벤트 코드를 확인해보겠습니다.




이벤트 캡처링은 하위 엘리먼트의 이벤트 발생 시에 부모 엘리먼트 이벤트까지 연쇄적으로 반응하는 것을 말합니다. 단, 이벤트의 발생 순서가 부모 엘리먼트 이벤트가 먼저 동작하고 하위 엘리먼트 이벤트가 동작합니다. 캡처링 동작을 위해선 이벤트 리스너의 마지막 인자를 **true**로 설정해야 합니다.

- 3번 코드는 id 속성이 bubble인  태그를 클릭할 경우 alert("버블 div 클릭!")을 호출하도록 하고 있습니다. 단, 이벤트 리스너의 마지막 인자가 false입니다.
- 4번 코드는 id 속성이 bubble인  태그의 자식 <p> 태그를 클릭할 경우 alert("버블 p 클릭!")을 호출하도록 하고 있습니다. 단, 이벤트 리스너의 마지막 인자가 false입니다.

우선, 해당 코드의 결과가 어떻게 동작하는지 이벤트 코드를 확인해보겠습니다.




이벤트 버블링은 캡처링과 반대로 이벤트의 발생 순서가 하위 엘리먼트 이벤트가 먼저 동작하고, 상위 엘리먼트 이벤트가 나중에 동작합니다. 버블링 동작을 위해선 이벤트 리스너의 마지막 인자를 **false**로 설정해야 합니다.

참고로, 이벤트 전파를 막고 싶은 경우에는 위에서 언급한 익명 함수의 e 인자를 사용하면 됩니다.

```
document.getElementById("capture").addEventListener("click", function(e){
    e.stopPropagation();
}, useCaputre);
```

위와 같이 코드를 작성하면 해당 엘리먼트 다음에 나오는 엘리먼트부터 이벤트 전파를 막을 수 있습니다.

정리하면, **캡처링: 부모 엘리먼트 >> 자식 엘리먼트**, **버블링: 자식 엘리먼트 >> 부모 엘리먼트**라고 할 수 있습니다. 그럼 어떤 것을 사용해야 할까요? 정답은 없습니다. 기본 값은 `false`이지만, 버블링과 캡처링은 좋고 나쁨을 나타내는 지표가 아니라 개발자의 요구에 맞게 사용한다는 점만 기억해주시면 됩니다.

이벤트를 어떻게 동작시키게 할지 필요에 따라서 서로 다른 방법을 적용시키면 됩니다! :) 이벤트 캡처링과 버블링에 대해서 더 궁금하시다면 여기 (https://www.quirksmode.org/js/events_order.html)를 참고해주세요!

이번 장에서는 사이트 방문자들의 액션에 대한 처리를 위한 이벤트와 이벤트 핸들러라는 것에 대해 알아보았습니다. 동적인 웹 페이지를 위해 이벤트를 처리하는 것은 굉장히 중요한 부분입니다. 사용자와 상호 작용할 수 있는 웹 페이지 제작을 위한 이벤트에 대해 많이 공부해보세요! :)