이벤트

이벤트 드리븐 프로그래밍

- 특정 사건을 발생하여 이벤트를 발생 시키는데 이 경우 **호출될 함수를 이벤트 핸들러**라고 한다.
- 이벤트가 발생했을 때 브라우저에게 이벤트 핸들러의 호출을 위임하는 것 --> **이벤트 핸들러 등록**
- 프로그램의 흐름을 이벤트 중심으로 제어하는 프로그래밍 방식을 **이벤트 드리븐 프로그래밍**이라고 한다.

이벤트 타입

키보드 이벤트

이벤트 타입	이벤트 발생 시점
keydown	모든 키를 눌렀을 때
keypress	문자 키를 눌렀을 때 연속적으로 발생
keyup	누르고 있던 키를 놓았을 때 한 번만

마우스 이벤트

이벤트 타입	이벤트 발생 시점
click	마우스 버튼을 클릭했을 때
dblclick	마우스 버튼을 더블 클릭했을 때
mousedown	마우스 버튼을 눌렀을 때
mouseup	누르고 있던 마우스 버튼을 놓았을 때
mousemove	마우스 커서를 움직였을 때
mouseenter	마우스 커서를 HTML 요소 안으로 이동했을 때 (버블링 되지 않는다)
mouseover	마우스 커서를 HTML 요소 안으로 이동했을 때(버블링된다)
mouseleave	마우스 커서를 HTML 요소 밖으로 이동했을 때(버블링 되지 않는다)
mouseout	마우스 커서를 HTML 요소 밖으로 이동했을 때(버블링된다)

포커스 이벤트

이벤트 타입	이벤트 발생 시점
focus	HTML 요소가 포커스를 받았을 때
blur	HTML 요소가 포커스를 잃었을 때
focusin	HTML 요소가 포커스를 받았을 때(버블링된다)
focusout	HTML 요소가 포커스를 잃었을 때(버블링된다)

폼 이벤트

이벤트 타입	이벤트 발생 시점
submit	form 요소 내의 submit 버튼을 클릭 시
reset	form 요소 내의 reset 버튼 클릭 시(최근 사용 X)

값 변경 이벤트

이벤트 타입	이벤트 발생 시점
input	input, select, textarea 요소의 값이 입력 시
change	input, select, textarea 요소의 값이 변경되었을 시 --> 입력이 종료되어 값이 변경되면 이벤트 발생
readystatechange	HTML 문서의 로드와 파싱 상태를 나타내는 document.readyState 프로퍼티 값('loading', 'interactive', 'complete')이 변경될 때

DOM 뮤테이션 이벤트

이벤트 타입	이벤트 발생 시점
DOMContentLoaded	HTML 문서의 로드와 파싱이 완료되어 DOM 생성이 완료되었을 때

뷰 이벤트

이벤트 타입	이벤트 발생 시점
resize	브라우저 윈도우의 크기를 리사이즈할 때 연속적으로 발생한다.
scroll	웹페이지 또는 HTML 요소를 스크롤할 때 연속적으로 발생한다.

리소스 이벤트

이벤트 타입	이벤트 발생 시점
load	DOMContentLoaded 이벤트가 발생한 이후, 모든 리소스의 로딩이 완료되었을 때(주로 window 객체에서 발생)
unload	리소스가 언로드될 때(주로 새로운 웹페이지를 요청한 경우)
abort	리소스 로딩이 중단되었을 때
error	리소스 로딩이 실패했을 때

이벤트 핸들러 등록

1. 이벤트 핸들러 어트리뷰트 방식
2. 이벤트 핸들러 프로퍼티 방식
3. addEventListener 메서드 방식

이벤트 핸들러 어트리뷰트 방식

```
<!--html부분-->
<button onclick="sayHi('Lee')">Click me!</button>

//script부분
function sayHi(name) {
  console.log(`Hi! ${name}.`);
}
```

이벤트 핸들러 프로퍼티 방식

- 이벤트 핸들러 등록을 위해선 이벤트를 발생시킬 객체인 **이벤트 타켓**, 이벤트 종류를 나타내는 문자열인 **이벤트 타입**, **이벤트 핸들러**가 필요하다
- js와 html을 분리할 순 있지만 프로퍼티에 하나의 이벤트 핸들러만 바인딩할 수 있다는 단점 존재

```
<!--html부분-->
<button>Click me!</button>

//script부분
```

```
const $button = document.querySelector('button');

// 이벤트 핸들러 프로퍼티에 이벤트 핸들러를 바인딩
$button.onclick = function () {
  console.log('button click');
};
```

addEventListener 메서드 방식

EventTarget.prototype.addEventListener('eventType', functionName [, useCapture]) 메서드를 사용

- EventTarget: 이벤트 타겟,
- eventType: 이벤트 타입,
- functionName: 함수 이름,
- useCapture: true 인 경우 캡처링, false인 경우 버블링(기본값)

```
// addEventListener 메서드 방식
$button.addEventListener('click', function () {
  console.log('[1]button click');
});

// addEventListener 메서드는 동일한 요소에서 발생한 동일한 이벤트에 대해
// 하나 이상의 이벤트 핸들러를 등록할 수 있다.
$button.addEventListener('click', function () {
  console.log('[2]button click');
});
```

이벤트 핸들러 제거

```
const handleClick = () => console.log('button click');

// 이벤트 핸들러 등록
$button.addEventListener('click', handleClick);

// 이벤트 핸들러 제거
```

```
// addEventListener 메서드에 전달한 인수와 removeEventListener 메서드에
// 전달한 인수가 일치하지 않으면 이벤트 핸들러가 제거되지 않는다.
$button.removeEventListener('click', handleClick, true); // 실패
$button.removeEventListener('click', handleClick); // 성공

// 기명 함수를 이벤트 핸들러로 등록
$button.addEventListener('click', function foo() {
  console.log('button click');
  // 이벤트 핸들러를 제거한다. 따라서 이벤트 핸들러는 단 한 번만 호출된다.
  $button.removeEventListener('click', foo);
});
```

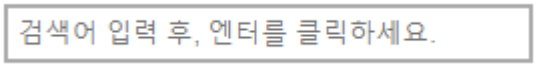
연습 문제

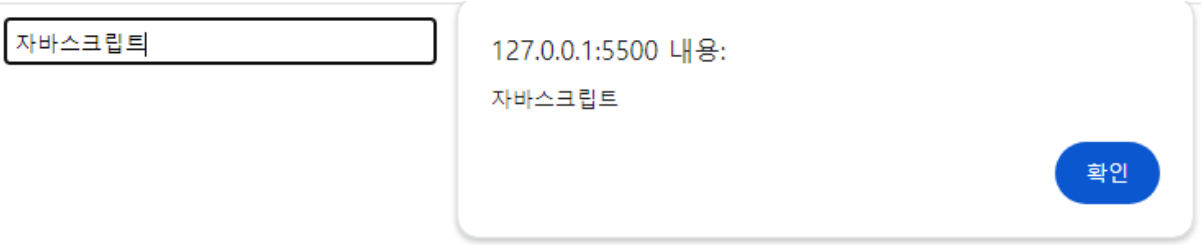
> <input>을 이용하여 검색어 입력창을 만듭니다.
>
> 검색창에 검색어를 입력한 후 엔터를 누르면, 입력한 검색어를 경고창으로 확인할 수 있습니다.
>
> 다음 자바스크립트 코드를 보고 문제를 풀어 보세요.

```
<!DOCTYPE html>
<html lang="ko">
    <head>
        <meta charset="utf-8">
        <title>이벤트 연습</title>
        <style>
            input[name="search"]{
                width: 250px;
                padding: 5px;
                border: 2px solid #aaa;
            }
        </style>
    </head>
```

```
    <body>
        <input type="text" name="search" onkeydown="mySearc
h(event)" placeholder="검색어 입력 후, 엔터를 클릭하세요.">
        <script>
            //이 곳에 자바스크립트 코드 작성
        </script>
    </body>
</html>
```

출력 결과





답

```
<!DOCTYPE html>
<html>
<head>
    <title>vanilla</title>
    <meta charset="utf-8">
    <style type="text/css">
    * { margin:0; padding:0; } li{ list-style-type:none; }
    body { font:12px/1.5 "굴림",Gulim; margin:10px; }
    h2{margin-top:20px;}

    .box1, .box2, h2+p+p{
      overflow: hidden;
      max-height: 40px;
    }
```

```
    </style>
</head>
<body>
    <div id="wrap">
      <h1>제이쿼리 Effect to 바닐라스크립트</h1>
      <p>제이쿼리에서는 화면전환, 슬라이드 다운과 업 효과 등<br />다양한 효과 메서드를 제공하며, 바닐라JS로 구현이 가능합니다.</p>

      <h2>show/hide 효과</h2>
      <p><button id="btn1">hide</button> <button id="btn2">show</button></p>
      <p class="box1">선택 요소를 숨기거나 노출하고 싶을 경우에 사용합니다.<br />
    hide(시간)/show(시간) 시간은 fast, normal, slow 등으로 입력할 수 있으며, 1000분의 1초 단위의 숫자로 표기할 수도 있습니다.</p>

      <h2>toggle 효과</h2>
      <p><button id="btn3">toggle</button></p>
      <p class="box2">선택 요소가 숨겨져 있다면 보이게 하고,<br />보이고 있다면 숨기는 역할을 합니다.</p>

      <h2>slideDown/up 효과</h2>
      <p>
        <button id="btn4">slideUp</button>
        <button id="btn5">slideDown</button>
        <button id="btn6">slideToggle</button>
      </p>
      <p>slideUp은 선택 요소를 위로 미끄러지듯 말아 올리면서 숨깁니다.<br />반면,
    slideDown은 선택 요소를 아래로 미끄러지듯 말아 내리면서 노출시킵니다.</p>

      <h2>fadeIn/fadOut</h2>
      <p>
        <button id="btn7">fadeOut</button>
        <button id="btn8">fadeIn</button>
        <button id="btn9">fadeToggle</button>
      </p>
```

```
      <p>fadeOut은 선택 요소를 천천히 사라지게 만듭니다.<br />fade
In은 선택 요소를 천천히 나타나게 합니다.</p>

      <h2>fadeTo 효과</h2>
      <p>
        <button id="btn10">fadeTo(0.3)</button>
        <button id="btn11">fadeTo(1)</button>
      </p>
      <p id="box3">fadeTo는 선택 요소를 원하는 만큼만 <br />보이거
나 숨길 수 있습니다.</p>
    </div>
</body>
</html>
```

```
const box1 = document.querySelector('.box1');
const box2 = document.querySelector('.box2');

document.querySelector('#btn1').addEventListener('click',fu
nction(){
  box1.style.transition = 'max-height .75s';
  box1.style.maxHeight = '0px'
})

document.querySelector('#btn2').addEventListener('click',fu
nction(){
  box1.style.transition = 'max-height 1s';
  box1.style.maxHeight = '40px'
})

document.querySelector('#btn3').addEventListener('click',fu
nction(){
  box2.style.transition = 'max-height .5s';

  if(!(getComputedStyle(box2).maxHeight === '0px')){
    box2.style.maxHeight = '0px';
  } else {
    box2.style.maxHeight = '40px';
  }
```

```
})

document.querySelector('#btn4').addEventListener('click',fu
nction(){
  this.parentNode.nextElementSibling.style.transition = 'ma
x-height .5s';
  this.parentNode.nextElementSibling.style.maxHeight = '0p
x';
})

document.querySelector('#btn5').addEventListener('click',fu
nction(){
  this.parentNode.nextElementSibling.style.transition = 'ma
x-height .25s';
  this.parentNode.nextElementSibling.style.maxHeight = '40p
x';
})

document.querySelector('#btn6').addEventListener('click',fu
nction(){
  const btn6ParentNext = this.parentNode.nextElementSiblin
g;
  btn6ParentNext.style.transition = 'max-height .25s';

  if(!(getComputedStyle(btn6ParentNext).maxHeight === '0p
x')){
    btn6ParentNext.style.maxHeight = '0px';
  } else {
    btn6ParentNext.style.maxHeight = '40px';
  }
})

document.querySelector('#btn7').addEventListener('click',fu
nction(){
  const btn7ParentNextStyle = this.parentNode.nextElementSi
bling.style;
  btn7ParentNextStyle.transition = 'opacity 1s';
  btn7ParentNextStyle.opacity = '0';
```

```
  setTimeout(function(){
    btn7ParentNextStyle.display = 'none';
  },1000)
})

document.querySelector('#btn8').addEventListener('click',fu
nction(){
  const btn8ParentNextStyle = this.parentNode.nextElementSi
bling.style;
  btn8ParentNextStyle.display = 'block';
  setTimeout(function(){
    btn8ParentNextStyle.transition = 'opacity .75s';
    btn8ParentNextStyle.opacity = '1';
  },1)
})

document.querySelector('#btn9').addEventListener('click',fu
nction(){
  const btn9ParentNext = this.parentNode.nextElementSiblin
g;
  btn9ParentNext.style.transition = 'opacity .5s';

  if(!(getComputedStyle(btn9ParentNext).opacity === '0')){
    btn9ParentNext.style.opacity = '0';
    setTimeout(function(){
      btn9ParentNext.style.display = 'none';
    },500)
  } else {
    btn9ParentNext.style.display = 'block';
    setTimeout(function(){
      btn9ParentNext.style.opacity = '1';
    })
  }
})

document.querySelector('#btn10').addEventListener('click',f
unction(){
  this.parentNode.nextElementSibling.style.transition = 'op
```

```
acity .75s';
  this.parentNode.nextElementSibling.style.opacity = '.3';
})

document.querySelector('#btn11').addEventListener('click',f
unction(){
  this.parentNode.nextElementSibling.style.transition = 'op
acity .75s';
  this.parentNode.nextElementSibling.style.opacity = '1';
})
```

이벤트 객체

이벤트가 발생하면 이벤트에 관련한 다양한 정보를 담고 있는 이벤트 객체가 동적으로 생성된다.

생성된 이벤트 객체는 이벤트 핸들러의 첫번째 인수로 전달된다.

```
<body onclick="showCoords(event)">
  <p>클릭하세요. 클릭한 곳의 좌표가 표시됩니다.</p>
  <em class="message"></em>

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

    // 클릭 이벤트에 의해 생성된 이벤트 객체는 이벤트 핸들러의 첫 번째
인수로 전달된다.
    function showCoords(e) {
      $msg.textContent = `clientX: ${e.clientX}, clientY:
${e.clientY}`;
    }
  </script>
```

이벤트 객체의 공통 프로퍼티

프로퍼티/메서드	타입	R/W	설명
bubbles	bool	R	이벤트가 버블링되는지

프로퍼티/메서드	타입	R/W	설명
			나타냄
cancelable	bool	R	이벤트의 기본 동작 취소 가능 여부
currentTarget	element	R	현재 이벤트를 처리중인 element
defaultPrevented	bool	R	true면 preventDefault()호출 상태
detail	integer	R	이벤트와 관련된 추가정보
eventPhase	integer	R	이벤트 핸들러가 호출된 단계(1:캡처링,2:타깃,3:버블링)
preventDefault()	Function	R	이벤트의 기본행동 취소, cancelable가 true일때 기능함
stopImmediatePropagation()	Function	R	이벤트캡처링,이벤트버블링 모두 취소하며 다른 이벤트 핸들러 호출을 막음.
stopPropagation()	Function	R	이벤트캡처링,이벤트버블링 모두 취소합니다. bubbles가 true일때 기능합니다.
target	element	R	이벤트 타깃
trusted	bool	R	브라우저에서 생성한 이벤트라면 true 개발자가 만든 자바스크립트 이벤트라면 false
type	string	R	발생한 이벤트 타입
view	AbstractView	R	이벤트와 연결된 추상화된 뷰입니다. 이벤트가 발생한 window객체와 일치

체크박스 요소의 체크상태가 변경되면 현재 체크 상태를 출력하는 구문

```
<input type="checkbox">
<em class="message">off</em>

<script>
const $checkbox = document.querySelector('input[type=checkbox]');
const $msg = document.querySelector('.message');

// change 이벤트가 발생하면 Event 타입의 이벤트 객체가 생성된다.
$checkbox.onchange = e => {
  // e.target.checked는 체크박스 요소의 현재 체크 상태를 나타낸다.
  $msg.textContent = e.target.checked ? 'on' : 'off';
};
</script>
```

마우스 정보 취득

click, dbclick, mousedown, mouseup, mousemove, mouseenter, mouseleave 이벤트가 발생하면 생성되는 마우스이벤트 타입의 이벤트 객체는 다음과 같은 고유의 프로퍼티를 갖는다.

> 마우스 포인터의 좌표 정보 : screenX / screenY, clientX / clientY, pageX / pageY, offsetX / offsetY

```
<div id="wrapper">
    <div id="box">
        <p>
            clientX = <span id="clientx">0</span><br>
            clientY = <span id="clienty">0</span><br>
            offsetX = <span id="offsetx">0</span><br>
            offsetY = <span id="offsety">0</span><br>
            pageX = <span id="pagex">0</span><br>
            pageY = <span id="pagey">0</span><br>
            screenX = <span id="screenx">0</span><br>
            screenY = <span id="screeny">0</span>
        </p>
```

```
    </div>
</div>
```

```
style
* {  margin:0;   padding:0;     }
#wrapper {  width:3000px;   height:3000px;    }
#box {
    width:400px;   height:400px;   border:2px solid blue;
margin-left:300px;
    margin-top:100px;   position:fixed;
}
#box > p {  padding:100px;   }
```

```
box.addEventListener('mousemove', (e) => {
    //document에서 스크롤 이동값을 제외한 마우스의 x,y좌표 값
  const clientx = document.querySelector('#clientx').textCo
ntent = e.clientX;
  const clienty = document.querySelector('#clienty').textCo
ntent = e.clientY;

    //이벤트 발생시 이벤트의 대상으로 부터 마우스의 x,y좌표 값
  const offsetx = document.querySelector('#offsetx').textCo
ntent = e.offsetX;
  const offsety = document.querySelector('#offsety').textCo
ntent = e.offsetY;

    //document에서 스크롤 이동값을 포함한 마우스의 x,y좌표 값
  const pagex = document.querySelector('#pagex').textConten
t = e.pageX;
  const pagey = document.querySelector('#pagey').textConten
t = e.pageY;

    //모니터 화면을 기준으로 마우스의 x,y좌표 값
  const screenx = document.querySelector('#screenx').textCo
ntent = e.screenX;
  const screeny  = document.querySelector('#screeny').textC
```

```
ontent = e.screenY;
});
```

연습 예제

이미지 다운로드 : http://naver.me/F3TEsL7M

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <script src="무브(vanilla).js" defer></script>

<style>
#m1{ width:100px; height:100px; position:absolute; top:50p
x; left:30px }
#m2{ width:100px; height:100px; position:absolute; top:300p
x; right:30px }
#m3{ width:100px; height:100px; position:absolute; bottom:8
0px; left:150px }
#cursor{
    width:100px; height:100px; cursor:url(111.png), auto; p
osition:absolute; top:0; left:0;
}


</style>

</head>
<body>
    <div id="m1"><img src="1.png" /></div>
    <div id="m2"><img src="2.png" /></div>
    <div id="m3"><img src="3.png" /></div>
    <div id="cursor"></div>
```

```
</body>
</html>
```

```
addEventListener('mousemove',function(e){
  const x = e.clientX;
  const y = e.clientY;

  const m1 = document.querySelector('#m1');
  m1.style.top = `${y/2}px`;
  m1.style.left = `${x/3}px`;

  const m2 = document.querySelector('#m2');
  m2.style.right = `${x/5}px`

  const m3 = document.querySelector('#m3');
  m3.style.bottom = `${y/6}px`

  const cursor = document.querySelector('#cursor');
  cursor.style.top = `${y}px`;
  cursor.style.left = `${x}px`;
})
```

키보드 정보 취득

keydown, keyup, keypress 이벤트가 발생하면 생성되는 키보드 이벤트 타입의 객체는 다음과 같은 고유의 프로퍼티를 갖는다.

> 버튼 정보 : altKey, ctrlKey, shiftKey, metaKey, key, keyCode

input요소의 입력 필드에 엔터 키가 입력되면 현재까지 입력 필드에 입력된 값을 출력하는 구문

```
<input type="text" />
<em class="message"></em>

<script>
```

```
const $input = document.querySelector('input[type=text]');
const $msg = document.querySelector('.message');
$input.onkeyup = e => {
  // e.key는 입력한 키 값을 문자열로 반환한다.
  // 입력한 키가 'Enter', 즉 엔터 키가 아니면 무시한다.
  if (e.key !== 'Enter') return;
  // 엔터키가 입력되면 현재까지 입력 필드에 입력된 값을 출력한다.
  $msg.textContent = e.target.value;
  e.target.value = '';
};
</script>
```

이벤트 전파

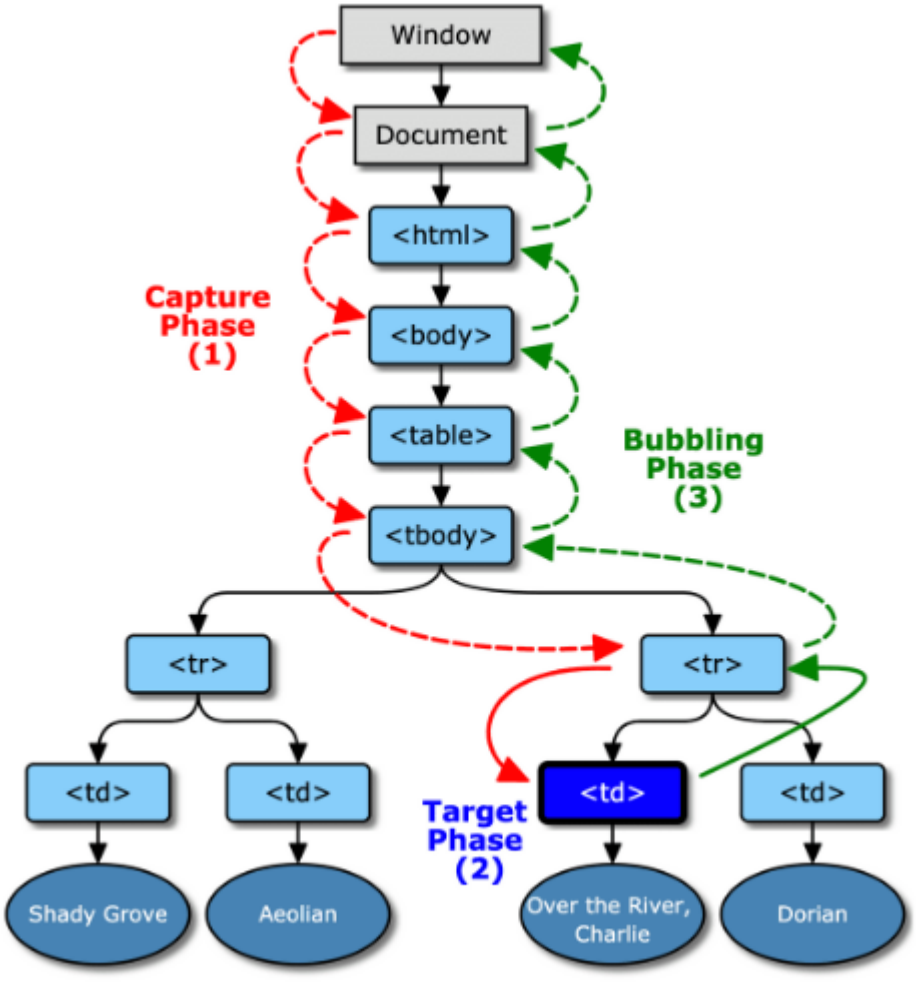


Figure 1 Graphical representation of an event dispatched in a DOM tree using the DOM event flow

- 다음 예제는 클릭하면 배경색이 바뀌도록 이벤트가 생성되어있습니다.
- 그런데 자식 엘리먼트를 클릭하면 부모 엘리먼트에도 클릭 반응이 발생합니다.
- 그 이유는 이벤트 모델의 전파 과정 때문에 발생하는 현상입니다. (버블링)

```
<style>
* {  margin:0;    padding:0;     }
body {  margin-left:300px;   margin-top:60px;    }
#wrapper {  width:200px;   height:200px;   border:1px solid
red;
        background-color: yellow;   padding:150px;      }
#targetBox {    width:200px;   height:200px;   background-c
olor: red;   color:#FFFFFF   }
</style>

<div id="wrapper">
    <div id="targetBox"></div>
</div>

<script>
const wrapper = document.querySelector('#wrapper');
const box = document.querySelector('#targetBox');
box.addEventListener('click', (e) => {
  e.stopPropagation(); //자식 객체가 클릭되었을 때 부모 객체로 전달
되지 않도록 메서드를 호출하여 버블링을 차단
  box.style.backgroundColor = 'black';
});

wrapper.addEventListener('click', (e) => {
  wrapper.style.backgroundColor = 'blue';
});
</script>
```

DOM 요소의 기본 동작 중단

```
<a href="https://www.google.com">go</a>
<input type="checkbox">
```

```
<script>
document.querySelector('a').onclick = e => {
  // a 요소의 기본 동작을 중단한다.
  e.preventDefault();
};

document.querySelector('input[type=checkbox]').onclick = e
=> {
  // checkbox 요소의 기본 동작을 중단한다.
  e.preventDefault();
};
</script>
```

이벤트 위임

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

스프레드 문법(아래 예시에서 사용)

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 복
사한다.
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(undefined + 1)을 할당한다.
  ++this.textContent;
```

```
});
</script>
```