이벤트 버블링과 캡쳐링

이벤트를 등록하는 방법은 html 요소에 인라인 방식으로 전달하는 방법과 동적으로 addEventListener를 이용하여, 이벤트를 설정하는 방법을 배웠다.

**짧은 연습**

Html요소 버튼에 onClick메서드를 주어 경고창을 띄우는 이벤트를 인라인 방식을 먼저 구현해보자.

<body>

<button onclick=warning()>

경고

</button>

</body>

<script>

let warning = ()=>{

alert("경고 메세지")

}

</script>

다음은 id=”root”라는 div를 정적으로 만들고, 내부에 button을 동적으로 할당한 다음 해당 버튼 이벤트로 안녕하세요 ! 라는 경고창을 띄우는 버튼을 만들어보자 .

<body>

<div id="root">

</div>

</body>

<script>

const root = document.getElementById("root");

const button = document.createElement("button");

button.innerText = "경고창버튼";

button.addEventListener("click",()=>{

alert("hello world");

})

root.appendChild(button);

</script>

이처럼 addEventListener()는 웹 개발자들이 화면에 동적인 기능을 추가하기 위해 자연스럽게 접하게 되는 기본적인 기능입니다. 사용자의 입력에 따라 추가 동작을 구현 할 수 있는 방법입니다. 여기서 브라우저는 어떻게 이벤트발생을 감지 했을까요 ? 브라우저가 이벤트를 감지하는 방식 두가지 “이벤트 버블링”, “이벤트 캡쳐링” 을 알아보도록 하겠습니다.

**이벤트 버블링**

이벤트 버블링은 특정화면 요소에서 이벤트가 발생했을 때 해당 이벤트가 더 상위의 화면 요소로 전달되어 가는 특성을 의미합니다.

<body>

<div id="root">

<div id="first">

<div id="second" onclick=eventOccured()>

</div>

</div>

</div>

</body>

(그려주자)

여기서 상위 요소란 이벤트가 일어난 돔 트리의 노드의 조상들을 칭한다. 현재는 second의 상위 요소는 first와 root, body이다.

현재 div가 보기 불편하니까 style을 지정해주자.

<style>

#root{

width: 300px;

height: 300px;

border : 1px solid black;

}

#first{

width: 200px;

height: 200px;

border: 1px solid aqua;

}

#second{

width: 100px;

height: 100px;

border: 1px solid red;

}

</style>

각 div에 클릭 했을 때 이벤트가 발생된 요소의 아이디를 경고창에 띄우는 스크립트를 만들어 보자

<script>

const divs = document.querySelectorAll('div');

divs.forEach(div => {

*//forEach메서드는 해당 배열의 각 아이템을 재 구성할 수 있게 해주는 유용한 메서드이다.*

*//이 forEach메서드는 추후 자세히 설명하고 지금은 모든 div요소들에 onclick메서드를 동적으로 할당함*

div.addEventListener('click',(event)=>{

alert(event.currentTarget.id)

})

});

</script>

완성이 되었으면 가장 작은 second를 클릭해보자 => 결과는 second , first, root순으로 발생한다.

혹시 모르니 상위 요소인 second 를 클릭해보자 => 결과는 first, root순으로 발생한다.

**왜 div태그 한 개만 클릭 했을 뿐인데, 왜 3개의 이벤트가 발생되는 것인가 ?**

이유는 브라우저가 이벤트를 감지하는 방식 때문이다. 브라우저는 특정 화면요소에서 이벤트가 발생했을 때 그 이벤트를 최상위에 있는 화면 요소까지 이벤트를 전파시킵니다. 따라서 second,first,root 순으로 div태그에 등록된 이벤트들이 실행되는 것입니다. First의 경우 first->root순이구요.

여기서 중요한 것은 상위요소마다 이벤트가 전달 되는 것을 확인할 수 있습니다. 만약 이벤트가 특정 태그에만 달려 있다면 위와 같은 동작 결과를 볼 수 없습니다.

이와 같이 하위에서 상위 요소로의 이벤트 전파 방식을 이벤트버블링 이라고 합니다.

**이벤트캡쳐**

이벤트 캡쳐는 이벤트 버블링과 반대 방향인 하위로 진행되는 이벤트 전파 방식입니다.

이 이벤트 캡쳐는 다음과 같은 구현으로 구현 할 수 있습니다.

<script>

const divs = document.querySelectorAll('div');

divs.forEach(div => {

*//forEach메서드는 해당 배열의 각 아이템을 재 구성할 수 있게 해주는 유용한 메서드이다.*

*//이 forEach메서드는 추후 자세히 설명하고 지금은 모든 div요소들에 onclick메서드를 동적으로 할당함*

div.addEventListener('click',(event)=>{

alert(event.currentTarget.id)

},{

capture : true

})

});

</script>

addEventListener의 세번째 인자로 객체를 전달하고, 이 객체의 프로퍼티를 값을 true로 하면 된다.

이제 second를 클릭해보자 => root, first, second순으로 이벤트가 발생하는 것을 알 수 있다.

**Event.stopPropagation()**

그렇다면 이벤트 전파를 막을 방법은 없는 것인가 ? 그렇지 않다. 정말이지 간단한 방법으로 이벤트 전파를 막을 수 있다.

divs.forEach(div => {

*//forEach메서드는 해당 배열의 각 아이템을 재 구성할 수 있게 해주는 유용한 메서드이다.*

*//이 forEach메서드는 추후 자세히 설명하고 지금은 모든 div요소들에 onclick메서드를 동적으로 할당함*

div.addEventListener('click',(event)=>{

alert(event.currentTarget.id)

event.stopPropagation();

})

});

stopPropagation메서드를 사용하면 더 이상 이벤트 전파가 일어나지 않는다.

**Event.preventDefault()**

지난번에도 설명했지만 한번 더 설명하도록 하겠다. Submit버튼이나 a태그의 경우 기본적으로 탑재되어 있는 이벤트들이 있다. 이런 태그들은 조건에 따라 폼의 제출이나, 다운로드 같은 이벤트들을 막고 싶을 수 있다.

그런 경우 preventDefault()메서드를 사용하면 해당 폼의 제출이나 이동을 막을 수 있다.

**이벤트 위임**

우리가 지금 까지 버블링과 캡쳐링을 배웠던 이유는 사실 이벤트 위임이라는 것들을 배우기 위함이다.

Body내용

<body>

<h1>오늘의 할 일</h1>

<ul class="itemList">

<li>

<input type="checkbox" id="item1" value=”item1”>

<label for="item1">이벤트 버블링 학습</label>

</li>

<li>

<input type="checkbox" id="item2" value=”item2”>

<label for="item2">이벤트 캡쳐 학습</label>

</li>

</ul>

</body>

Check박스에 클릭하면 클릭한 값이 나오도록 변경하자 이벤트를 설정하자

var inputs = document.querySelectorAll('input');

inputs.forEach(function(input) {

input.addEventListener('click', function(event) {

alert(event.target.value);

});

});

그런데 보기가 늘어 다음과 같이 동적으로 증가 시켰다.

var li = document.createElement('li');

var input = document.createElement('input');

var label = document.createElement('label');

var labelText = document.createTextNode('이벤트 위임 학습');

input.setAttribute('type', 'checkbox');

input.setAttribute('id', 'item3');

input.setAttribute('value','item3');

label.setAttribute('for', 'item3');

label.appendChild(labelText);

li.appendChild(input);

li.appendChild(label);

itemList.appendChild(li);

그런데, 확인을 해보니 3번째 부터는 작동을 하지 않는다. 그 이유는 스크립트가 실행되고 이벤트를 폴이치를 통해 기존 2개의 체크박스에 할당하였기 때문이다. 이 추가된 요소는 이벤트를 할당하고 엘리먼트가 삽입되었기 때문에 마지막으로 삽입된 요소는 이벤트를 갖지를 못한 것이다. 그러나 새로운 요소가 생길 때 동적으로 생길 때 마다 이벤트를 위임하도록 만든다면 꽤나 복잡할 것이다.

이때 사용하는 것이 이벤트 위임이다.

var itemList = document.querySelector('.itemList');

itemList.addEventListener('click', function(event) {

alert('clicked');

console.log(event.target.value);

*// event.stopPropagation()*

});

이벤트를 체크 박스가 아닌 상위에 추가한다. 그렇기 때문에 하위에 있는 것들에서 onClick이벤트가 발생하면 상위에서 작성한 이벤트들이 발생할 것이다. 그래서 이벤트 위임만 하였다면 요소가 늘어나도 새로운 요소에 이벤트를 할당하지 않아도 되는 것이다.

**네이버 상단 웨일브라우저 광고창 예제**

네이버를 접속 하면 상단에 웨일 브라우저를 홍보하기위한 div가 생성되어 있다. 이를 크롬 검사로 확인 해보면 a태그 내부에 버튼이 생성 된 것을 볼 수 있다. 버튼을 클릭하면 다운로드가 되고, a태그 영역을 클릭하면 링크가 이동되는 것을 볼 수 있다. 지금 배운 이벤트 버블링 현상이 일어나야 함에도 불구하고 버튼을 클릭하면 a태그의 이벤트가 일어나지 않는다. 그러나 찾아본 바로는 이 부분은 클릭이벤트 전파로 인한 것이 아니였기때문에 나름의 방법으로 우회를 했다. 이 방법은 리프레시를 일으키기 때문에 좋아보이지는 않는다.

**예제 스타일링**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Document</title>

<style>

#headAdBar {

min-width: 1000px;

height: 50px;

background-color: #0a1425;

}

#areaFlex {

width: 60%;

margin: 0 auto;

height: 100%;

}

body {

padding: 0;

}

#whaleLink {

position: relative;

display: inline-block;

height: 50px;

width: 1000px;

text-align: center;

line-height: 50px;

text-decoration: none;

color: white;

background-image: url(https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcTsatOu1oSUhr097DdEjpeiEnc8xu\_\_7xWA-QjWl9UVjmQwyXPmrA&s);

background-repeat: no-repeat;

background-size: cover;

font-family: "Apple SD Gothic Neo","맑은고딕","Malgun Gothic";

}

#button {

width: 70px;

background-color: #82dbbe;

border: none;

color: #fff;

padding: 10px 0;

text-align: center;

text-decoration: none;

display: inline-block;

font-size: 10px;

cursor: pointer;

float: right;

margin: 10px 0;

border-radius: 5px;

}

bm {

font-weight: bold;

color: #00d2b3;

}

#icoWhale {

position: absolute;

top: 9px;

left: 320px;

width: 29px;

height: 30px;

background-image: url(https://s.pstatic.net/static/www/img/uit/2019/sp\_main\_v191212.png);

background-position: -143px -60px;

float: right;

}

#adTextDiv{

display: inline-block;

margin-left: 250px;

}

</style>

</head>

<body>

<div id="headAdBar">

<div id="areaFlex">

<a id="whaleLink" href="http://www.naver.com">

<span id="icoWhale"></span>

<div id="adTextDiv">

<bm>인터넷의 새로운 시작 !</bm>

네이버 웨일로 차원이 다른 웹서핑을 경험해보세요

</div>

<button id="button">

다운로드

</button>

</a>

</div>

</div>

</body>

<script>

</script>

</html>

이 부분은 좀 달랐다.

<script>

let button = document.getElementById("button");

button.addEventListener('click',(e)=>{

e.stopPropagation();

let aTag = document.getElementById("whaleLink");

alert("다운로드 중 !");

aTag.href = "";

})

</script>