221017 수업 -1

키워드	node
날짜	@2022년 10월 17일 오전 9:01

Node

- DOM객체 자원은 계층 트리 형태

  → node 개념

- 부모, 자식, 형제 개념 존재

```
<script type="text/javascript>
  window.onload = function() {
    let menode;
    menode = document.getElementById("me");

    let parentnode = menode.parentNode; //body

    let grandnode = parentnode.parentNode; //html

    let my = document.getElementById("mychild");
    console.log(my.firstChild.nodeName); //li
    console.log(my.firstChild.innerText); //AA
    console.log(my.firstChild.nextSibling.innerText); //BB

    console.log(my.childNodes); //배열 리턴 Array
    console.log(my.childNodes[0]);
    console.log(my.childNodes.length);
  }
</script>

<body>
<div>A</div><div>B</div><div id="me">C</div><div>D</div><div>E</div>
<ul id="mychild"><li>AA</li><li>BB</li><li>CC</li></ul>
</body>
```

---

innerHTML, innerText

- value가 없는 태그에 대해
  - p, div, li, span ….

- 안쪽의 값을 가져옴

개선된 for > ES6 > of

```
let text = "";
for (let index in cars) {
  text += cars[index] + "<br>";
}
```

→ 인덱스가 나옴

```
let text = "";
for (let x of cars) {
  text += x + "<br>";
}
```

→ 값이 나옴

이벤트 주의사항

```
<script type="text/javascript">
  function displayDate(){
    console.log("call");
    document.getElementById("demo").innerHTML = new Date();
  }

  document.getElementById("mybtn").onclick = displayDate();
    -> 오류 발생 mybtn이 아직 읽히지 않음

  window.onload() = function(){
    console.log("load ..... ");
    //document.getElementById("mybtn").onclick = displayDate(); -> (x)
    document.getElementById("mybtn").onclick = displayDate;
  }
</script>

<body>
  <button id="mybtn">눌러봐</button>
```

```
  <br>
  <button id="mybtn2" onclick="">눌러봐2</button>
  <br>
  <p id="demo"></p>

</body>

<script type="text/javascript">
  document.getElementById("mybtn").onclick = displayDate();
    //버튼 클릭 전에 displayDate()가 호출됨

  document.getElementById("mybtn").onclick = displayDate;
    //()없이 하면 됨
<script>
```

EventListener

- 동적으로 이벤트를 추가하는 함수
- document.getElementById("myBtn").addEventListener("click", displayDate);
- removeEventListener()
- 문법 : *element*.addEventListener(*event, function, useCapture*);

1. 하나의 요소는 여러 개의 event를 가질 수 있다
2. 이벤트가 더 이상 필요하지 않다면 removeEventListener()로 제거 가능
   (단, add로 추가한 사항에 대해)

```
<body>
  <button id="mybtn">클릭</button>
  <p id="demo"></p>
</body>

<script type="text/javsscript>
  document.getElementById("mybtn").addEventListner("click", displayText);

  function displayText(){
    document.getElementById("demo").innerHTML = "hello world";
  }
<script>
```

- 다수의 이벤트 발생

```
<body>
  <button id="mybtn">클릭</button>
</body>

<script type="text/javascript">
  let x = document.getElementById("mybtn");

  x.addEventListener("mouseover", myFunc);
  x.addEventListener("mouseout", myFunc2);
  x.addEventListener("click", myFunc3);

  function myFunc(){
    document.getElementById("mybtn").innerHTML += "Mouse Over<br>";
  }
  function myFunc2(){
    document.getElementById("mybtn").innerHTML += "Mouse Out<br>";
  }
  function myFunc3(){
    document.getElementById("mybtn").innerHTML += "Click<br>";
  }

</script>
```

- 이미지 슬라이더

```
<head>
  <meta charset="UTF-8">
  <title>Insert title here</title>
  <script type="text/javascript">
    window.onload = function(){
      let myphotos = ["images/1.jpg", "images/2.jpg", "images/3.jpg", "images/4.jpg"];
      let index = 0;
      let len = myphotos.length;
      //a click 이벤트 처리하고

      document.getElementById("prv").addEventListener("click", function(){
        if(index == 0){
          index = len-1;
          document.getElementById("imgs").src = myphotos[index];
        }else{
          document.getElementById("imgs").src = myphotos[--index];
```

```
        }
      });
      document.getElementById("next").addEventListener("click", function(){
        if(index == len-1){
          index = 0;
          document.getElementById("imgs").src = myphotos[index];
        }else{
          document.getElementById("imgs").src = myphotos[++index];
        }
      });
    }
  </script>
</head>

<body>
  <h3>DOM 슬라이드</h3>
  <img src="images/1.jpg" alt="이미지" id="imgs" width="200" height="200">
  <hr>
  <a href="#" id="prv">이전</a> || <a href="#" id="next">다음</a>
</body>
```

HTML DOM Events Properties and Methods

HTML DOM Event Object

W3Schools offers free online tutorials, references and exercises in all the major languages of the web. Covering popular subjects like HTML, CSS, JavaScript, Python, SQL, Java, and many,

https://www.w3schools.com/jsref/dom_obj_event.asp

- 사건 발생하면 내부적으로 event 객체를 하나 생성
- 자동으로 생성된 Event 객체의 주소를 받아서 사용
- window.event.메서드
- window.event.속성

```
<script type="text/javascript">
  window.onload = function() {
    let btn = document.getElementById("mybtn");
    btn.onclick = function(e){ //e변수가 Event객체의 주소를 받음
```

```
      console.log(e.type); //Event객체가 가지는 type 속성, 이벤트의 종류
      console.log(e.target); //태그의 안쪽 요소
      console.log(e.currentTarget); //이벤트가 걸린 모든 요소
      console.log(e.x); //x 좌표 값
      console.log(e.y); //y 좌표 값
    }

  }
</script>

<body>
  <button id="mybtn"><span>눌러봐</span></button>
  <br>
  <button id="mybtn2">눌러봐2</button>
</body>
```

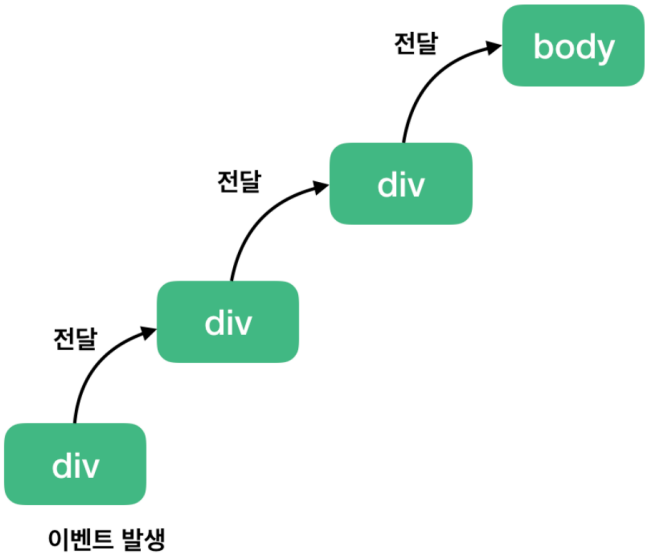
- 예제

```
<script type="text/javascript">
  window.onload = function() {
    let btn2 = document.getElementById("mybtn2");
    let handler = function(event){ //Event 객체의 주소 전달 받음
      switch(event.type){
        case "click" : alert("클릭");
          break;
        case "mouseover" : event.target.style.backgroundColor="red";
          break;
        case "mouseout" : event.target.style.backgroundColor="";
          break;
      }
    }
    btn2.onclick = handler;
    btn2.onmouseover = handler;
    btn2.onmouseout = handler;
  }
</script>

<body>
  <button id="mybtn"><span>눌러봐</span></button>
  <br>
  <button id="mybtn2">눌러봐2</button>
</body>
```

이벤트 전파 (버블링)

버블링

- 특정 화면 요소에서 이벤트가 발생했을 때 해당 이벤트가 더 상위의 화면 요소들로 전달되어 가는 특성



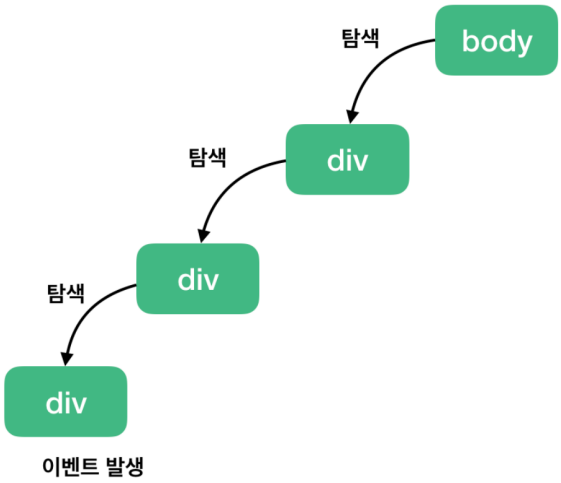

```
<body>
  <div id="myDiv1">
    <h2>Bubbling:</h2>
    <p id="myP1">Click me!</p>
  </div>
  <br>

  <div id="myDiv2">
    <h2>Capturing:</h2>
    <p id="myP2">Click me!</p>
  </div>
</body>

<script>
  document.getElementById("myP1").addEventListener("click", function() {
    alert("You clicked the white element!");
  }, false);

  document.getElementById("myDiv1").addEventListener("click", function() {
    alert("You clicked the orange element!");
  }, false);
</script>
```

→ myP1을 클릭해도 myDiv1의 alert도 반응해, alert이 2번 발생

→ alert("You clicked the white element!"); 후 alert("You clicked the orange element!");

이벤트 캡쳐

- 이벤트 버블링과 반대 방향으로 진행되는 이벤트 전파 방식




```
<body>
  <div id="myDiv1">
    <h2>Bubbling:</h2>
    <p id="myP1">Click me!</p>
  </div>
  <br>

  <div id="myDiv2">
    <h2>Capturing:</h2>
    <p id="myP2">Click me!</p>
  </div>
</body>

<script>
  document.getElementById("myP2").addEventListener("click", function() {
    alert("You clicked the white element!");
  }, true);

  document.getElementById("myDiv2").addEventListener("click", function() {
    alert("You clicked the orange element!");
  }, true);
</script>
```

→ alert("You clicked the orange element!"); 후 alert("You clicked the white element!");

---

preventDefault()

- 자기가 가진 이벤트를 막음

stopPropagation();

- 이벤트의 전파를 막음

- 리스트에서 한 요소 선택

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
<style>
#add-btn {
  padding: 5px 10px;
  border: 0;
  background: #f80;
  color: white;
  border-radius: 5px;
}

ul {
  padding: 0;
  list-style-position: inside;
}

li {
  border-bottom: 1px solid #999;
  padding: 5px 0;
}

.active {
  background: #abc;
} /* step3: 목록 클릭 스타일 */
</style>
</head>
<body>
  <h1 id='title'>할일 목록</h1>
  <button id="add-btn">목록 추가</button>
  <ul id='list'>
    <li>제목 1</li>
    <li>제목 2</li>
    <li>제목 3</li>
    <li>제목 4</li>
  </ul>
</body>
<script type="text/javascript">
  let title = document.getElementById("title");
  let list = document.getElementById("list");
  let li = document.getElementsByTagName("li");
```

```
  let addbtn = document.getElementById("add-btn");

  //console.log(title);
  //console.log(list);
  //console.log(li);
  //console.log(addbtn);

  function activeItem(event) { //event 객체를 parameter로 받기
    console.log(event.target); //li
    console.log(event.currentTarget); //ul

    if (event.target.nodeName == 'LI') { //nodeName은 대문자로 나옴.
      title.innerHTML = event.target.innerText;
    }

    //class 제거 (모든 요소)
    for(let i=0; i<event.target.parentNode.children.length; i++){
      event.target.parentNode.children[i].removeAttribute("class");
    }

    //다시 추가(한번만)
    event.target.setAttribute("class", "active");
  }

  list.addEventListener("click", activeItem);
</script>
```

동적 배열

```
let array = ['포도', '사과'];
array[2] = "바나나";

//권장하지 않음 (가능은 함)
array[10] = "애플망고";
//남은 공간이 undefined로 채워짐
```

```
let array2 = ["one", "two", "three"];
//권장하지 않아요 가능은 해요
array2.length = 2; //개수를 줄임
```

권장방법

- Array → Stack → LIFO(후입선출)
- push()
- pop()

/rn

Object

javascript 객체지향언어 OOP

클래스 정의 3가지 방법

1. 프로토타입 방식 : 일반적인 클래스 제작 방법
    - 인스턴스마다 공통된 메서드를 공유해 사용하는 장점
        - Jquery도 prototype 방식으로 설계

```
function 클래스이름() { // -> 이름의 첫글자 대문자 function Car
  this.프로퍼티1 = 초기값;
  this.프로퍼티2 = 초기값;
}

클래스이름.prototype.메서드1 = function() {
}

클래스이름.prototype.메서드2 = function() {
}

var 인스턴스 = new 클래스이름();
var carObj = new Car();
var carObj2 = new Car();
var carObj3 = new Car();
```

2. 함수 방식 : 간단한 클래스 제작 시 사용
    - 인스턴스마다 메서드가 독립적으로 만들어지는 단점

클래스 : function Car() {this.name= , this age = }

함수 : function car() { }

```
function 클래스이름() {
  this.프로퍼티1 = 초기값;
  this.프로퍼티2 = 초기값;
  this.메서드1 = function() {}
}
```

3. 리터럴 방식
   - 클래스 만드는 용도는 아님
   - 객체 만드는 가장 쉬운 방식 {"a" : "데이터"}
   - javascript 객체 표기법
   - 이기종간의 데이터 호환 → json

   ex) var myObj = { "name":"John", "age":31, "city":"New York" };
   - 재사용 불가

4. ECMA6 버전부터 class 키워드 제공

```
class Person {
  constructor(name) {
    this._name = name;
     }

  sayHi() {
    console.log(`Hi! ${this._name}`);
  }
}
```

---

javascript 객체 생성

1. 오브젝트 리터럴 방식 (객체를 만드는 방법)
   - 클래스 생성과 동시에 객체가 만들어 져요

2. 리터럴 방식
    - 제일 간단한 방법 > var obj = {}; //var objarr = [] 배열
    - JSON 표기 : {} >> JSON: JavaScript Object Notation

ex) var myObj = { "name":"John", "age":31, "city":"New York" };

** JSON >> XML (텍스트 기반의 형식화된 문서 제공)

XML :이기종간의 데이터 호환 (한 때는 서점 : xml webservice)

JSON

- 오브젝트 리터럴 방식 : 재사용을 지원하는 않는다
- 설계도를 생성과 동시에 객체 생성(장점 : 편하고 , 빠르다)
- 설계도를 미리 만들어 놓고 재사용하는 방식은 아니다
- 설계도당 하나의 객체만 생성 사용 (only object)
- 데이터로서의 의미가 강해서 메서드를 많이 만들지는 않는다

```
var product = {};
var product2 = {제품명:'사과',년도:'2018',원산지:'대구'};

var 인스턴스 ={
  프로퍼티:초기값,
  프로퍼티:초기값,

  ....

  메서드:function(){},
  메서드:function(){} ...
}
```

리터럴 방식 > 선언과 동시에 인스턴스 자동 생성

- var 인스턴스 = {}
- 특징 : 생성자 존재하지 않는다.
- 프로퍼티와 메서드만 정의 가능

- 단점 : 객체 하나 생성(재사용성 없다)
- 접근방법 : 인스턴스이름.자원 >> product2.제품명