

둘째 주

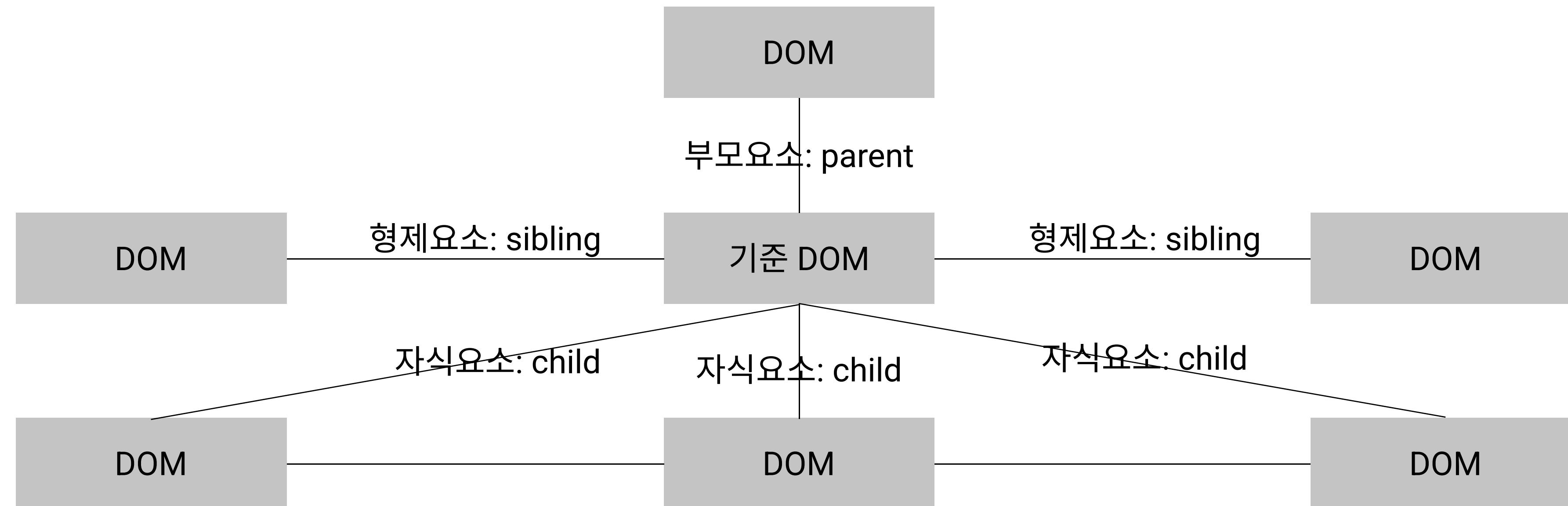
- 노드 탐색
- DOM 요소 수정하기
- 이벤트 바인딩
- 조건문

노드 탐색

앞에선 DOM에 걸려있는 다양한 속성을 이용해 해당 DOM을 가져오는 작업을 했습니다.

이번에는 **해당 DOM과 관련된 요소들을 가져오는 방법**을 알아보겠습니다.

DOM은 tree 구조를 가지며 하나의 DOM에 대해 연관된 요소들은 위치에 따라 각각 상대적인 이름을 가집니다.
그 이름들은 다음과 같습니다.



노드 탐색 - 자식 노드 탐색

DOM api 에는 해당 DOM과 관련된 요소들에 접근할 수 있는 함수를 제공하고 있습니다.
여기서는 자식 노드를 탐색하는 api들을 알아볼건데 다음과 같은 종류가 있습니다.

- childNodes
- children
- firstChild
- firstElementChild
- lastChild
- lastElementChild

각각 노드나 요소들의 정보를 가지고있는 프로퍼티 형식으로 접근할 수 있습니다.

노드 탐색 - 자식 노드 탐색

사용법은 각각의 DOM의 프로퍼티를 조회하는 형식으로 사용하면 됩니다.

```
const list = document.querySelector('.list')

// 자식 요소 목록 조회
list.childNodes
console.log('list.childNodes: ', list.childNodes);
list.children
console.log('list.children: ', list.children);

// 첫번째 자식 요소
list.firstChild
console.log('list.firstChild: ', list.firstChild);
list.firstElementChild
console.log('list.firstElementChild: ', list.firstElementChild);

// 마지막 자식 요소
list.lastChild
console.log('list.lastChild: ', list.lastChild);
list.lastElementChild
console.log('list.lastElementChild: ', list.lastElementChild);
```

노드 탐색 - 부모, 형제 노드 탐색

부모나 형제 노드에 접근하는 DOM api는 다음과 같습니다.

- parentNode
- parentElement
- previousSibling
- previousElementSibling
- nextSibling
- nextElementSibling

노드 탐색 - 부모, 형제 노드 탐색

앞의 자식 노드에 접근하는 방식처럼 사용하면 됩니다.

```
// --- 부모 노드 탐색
const item1 = document.querySelector('.item1')

item1.parentNode
console.log('item1.parentNode: ', item1.parentNode);
item1.parentElement
console.log('item1.parentElement: ', item1.parentElement);

// --- 형제 노드 탐색
const item3 = document.querySelector('.item3')

// 이전 요소
item3.previousSibling
console.log('item3.previousSibling: ', item3.previousSibling);
item3.previousElementSibling
console.log('item3.previousElementSibling: ', item3.previousElementSibling);

// 다음 요소
item3.nextSibling
console.log('item3.nextSibling: ', item3.nextSibling);
item3.nextElementSibling
console.log('item3.nextElementSibling: ', item3.nextElementSibling);
```

노드 추가,삽입,삭제

js를 이용해 html 노드를 직접 생성하거나 삽입,삭제 할 수 있습니다.

다음은 생성, 삽입, 삭제에 해당하는 DOM의 함수들입니다.

- createElement : 노드를 생성합니다.
- appendChild : 해당 요소의 자식 요소로 노드를 추가합니다. 가장 마지막에 추가시킵니다.
- insertBefore: 지정된 요소의 이전에 노드를 추가합니다.
- removeChild : 지정된 요소를 삭제합니다.

노드 추가, 삽입, 삭제

생성, 삽입, 삭제 예시 입니다.

```
// 노드 생성
const newLi = document.createElement('li')
newLi.textContent = 'this is new node'

// 노드 삽입
list.appendChild(newLi)

const newLiSecond = document.createElement('li')
newLiSecond.textContent = 'before'

list.insertBefore(newLiSecond, list.firstChild)

// 노드 삭제
list.removeChild(document.querySelector('.list .item3'))
```

DOM 요소 수정하기

html을 작성할 때 다음과 같은 코드를 본적 있을겁니다.

```

```

여기서 class나 src나 alt 들을 속성값이라 할 수 있습니다.

img dom을 javascript로 불러서 해당 속성값들에 접근할 수 있습니다.

```
const qqImage = document.querySelector('.qq-image')
```

DOM 요소 수정하기

DOM 속성에 접근

앞서 html에 img 태그의 src에 접근하려면 다음과 같이 할 수 있습니다.

```
const qqImage = document.querySelector('.qq-image')

// src 값을 볼때
qqImage.src

// src 값 바꿀 때
qqImage.src = 'url'
```

html에 적혀있는 속성값들에 접근하려면 몇몇 속성값들은 DOM 자체에 각 속성값 이름으로 들어가있는걸 알 수 있습니다.
해당값들을 변경하고 싶으면 = 연산자를 통해 값을 넣어주면 됩니다.

DOM 요소 수정하기

DOM 스타일에 접근

스타일에 접근하려면 'DOM.style.스타일속성명' 으로 접근하면 됩니다.

```
const qqImage = document.querySelector('.qq-image')

// 현재 적용된 style을 볼 때
const qqStyle = window.getComputedStyle(qqImage)

// style의 border 값을 바꿀 때
qqImage.style.border = '1px solid black'
```

해당 DOM의 스타일 값을 볼 때, 'qqImage.style.border' 값으로 접근하면 스타일 값을 볼 수 없습니다.

현재 적용중인 스타일을 알기위해선 window의 getComputedStyle 함수를 사용하면 됩니다.

DOM 요소 수정하기

DOM 클래스에 접근

이번엔 클래스에 접근해보겠습니다.

```
const qqImage = document.querySelector('.qq-image')

// 클래스를 string 형식으로 볼 때
qqImage.className

// string 형식으로 클래스를 바꿀 때
qqImage.className = qqImage.className + ' qq-big'

// list형식으로 클래스를 볼 때
qqImage.classList

// 클래스 추가
qqImage.classList.add('qq-big')

// 클래스 삭제
qqImage.classList.remove('qq-big')
```

Ex - DOM 수정하기

1. qqImage의 위치를 오른쪽으로 10px 움직여보세요
2. qqImage의 클래스에 qq-border 를 추가해보세요

이벤트

이벤트는 작성해 놓은 DOM에 대해 사용자가 여러가지 동작을 취하는 경우를 말합니다.

우리가 만들어놓은 DOM에 사용자는 **마우스**를 통해 클릭이나 드래그를 할 수 있고 **키보드**를 통해 입력을 할 수 있는데 이러한 동작들 모두 이벤트에 포함됩니다.

일반적으로 해당 DOM들은 사용자의 동작들에 대한 응답이 없거나 기본적인 응답을 가지고 있습니다.
그리고 개발자들은 이런 이벤트에 접근해 자신들이 원하는 동작을 추가시킬 수 있습니다.

이렇게 DOM 이벤트에 **새로운 동작(기능)**을 추가시키는 작업을 **이벤트 바인딩**이라고 하고, 이벤트 바인딩이 가능하도록 접근할 수 있게 해주는 것들을 **이벤트 핸들러**라고 합니다.

먼저 이벤트의 종류에 대해 알아보려 하는데, 이벤트 종류는 엄청나게 많기 때문에 모든 이벤트에 대해서 설명하기 힘듭니다(링크). 그러므로 우선 많이 쓰이는 이벤트의 종류에 대해서 알아보겠습니다.

이벤트 핸들러 종류

마우스 이벤트

마우스 이벤트입니다. 주로 마우스의 동작과 마우스와 요소의 상호작용에 대한 이벤트를 가지고 있으며, 이벤트 객체에 마우스 포인터의 좌표 같은 현재 마우스의 정보 데이터를 가지고 있습니다.

- onclick: 마우스 클릭시
- ondblclick: 마우스 더블클릭시
- onmousedown: 마우스 버튼을 누를 때
- onmouseup: 마우스 버튼에서 손을 뗄 시
- onmouseout: 마우스 포인터가 요소를 벗어날 때
- onmousemove: 마우스 포인터가 요소 위에서 움직일 때
- onmouseover: 마우스 포인터가 요소 위에 올라왔을 때

이벤트 핸들러 종류

키보드 이벤트

키보드 이벤트입니다. 주로 키보드 키보드를 누르거나 뗄 시에 대한 이벤트를 정의하고 있으며, 이벤트 객체속에 현재 어떤 키보드를 눌렀는지에 대한 정보가 들어있습니다.

- onkeydown: 키보드를 누를 시
- onkeypress: 키보드를 누르고 손을 뗄 시
- onkeyup: 키보드 키에서 손가락을 뗄 시

이벤트 핸들러 종류

폼 이벤트

키보드 이벤트입니다. 주로 키보드 키보드를 누르거나 뗄 시에 대한 이벤트를 정의하고 있으며, 이벤트 객체속에 현재 어떤 키보드를 눌렀는지에 대한 정보가 들어있습니다.

- onchange: input요소의 값이 변할 때(변경하고 포커스를 잃을 때)
- oninput: input요소 값이 변할 때(바뀐 직후)
- onblur: input요소에서 focus를 잃을 때
- onfocus: input요소를 focus 할 때(선택했을 때)
- onsubmit: 폼 제출 동작할 때

이벤트 핸들러 종류

페이지 이벤트

페이지 이벤트입니다. 주로 페이지의 시작과 종료에 관련된 이벤트를 정의하고 있습니다.

- `onload`: 페이지 첫 진입 후 모든 요소들이 로딩되었을 때
- `onunload`: 페이지에서 나갈 때

이벤트 바인딩

이벤트 바인딩이란?

javascript로 만든 함수들을 특정한 이벤트가 발생했을 때 동작할 수 있도록 연결해주는걸 이벤트 바인딩 이라고 합니다.

이벤트 바인딩 방법은 **dom요소의 이벤트 프로퍼티**를 이용한 방법과 **이벤트 리스너**를 이용하는 방법이 있습니다.

프로퍼티와 이벤트 리스너를 이용하는 방법의 차이는 이벤트 프로퍼티를 이용하는 경우 동작을 한가지만 등록이 가능하지만 이벤트 리스너를 사용할 경우 여러개의 동작을 등록시킬 수 있습니다.

먼저 DOM요소의 이벤트 프로퍼티를 이용해서 등록하는 방법을 알아봅시다.

이벤트 바인딩 - DOM 이벤트 프로퍼티

```
const bigBlock = document.querySelector('.big-block')

function bigBlockClick(event) {
  console.log('event work !!!: ', event);
}

bigBlock.onclick = bigBlockClick
```

querySelector를 통해 해당 요소를 가져와서 bigBlock 변수에 할당했습니다.

bigBlockClick이라는 함수를 정의하고 가져온 bigBlock 변수에 onclick이라는 프로퍼티에 정의한 함수를 할당하면 이벤트 바인딩 작업이 완료됩니다.

위와같이 이벤트 종류가 DOM요소에 프로퍼티로 정의되어 있으며, 우리가 원하는 함수를 해당 프로퍼티에 할당하면 해당 이벤트가 발생할 경우 함수가 동작하게 됩니다.

등록된 함수를 해제하고 싶으면 해당 이벤트 프로퍼티에 null을 할당해주면 됩니다.

```
bigBlock.onclick = null
```

이벤트 바인딩 - 이벤트 리스너

```
const bigBlock = document.querySelector('.big-block')

function bigBlockClick(event) {
  console.log('event: ', event);
}

bigBlock.addEventListener('click', bigBlockClick, false)
```

이번엔 이벤트 리스너를 통해 이벤트를 등록했습니다.

DOM요소의 `addEventListener`를 사용해서 이벤트를 등록하는데 해당 함수의 매개변수는 다음과 같습니다.

`addEventListener(이벤트명, 바인딩할 함수, 이벤트 캡처링 유무)`

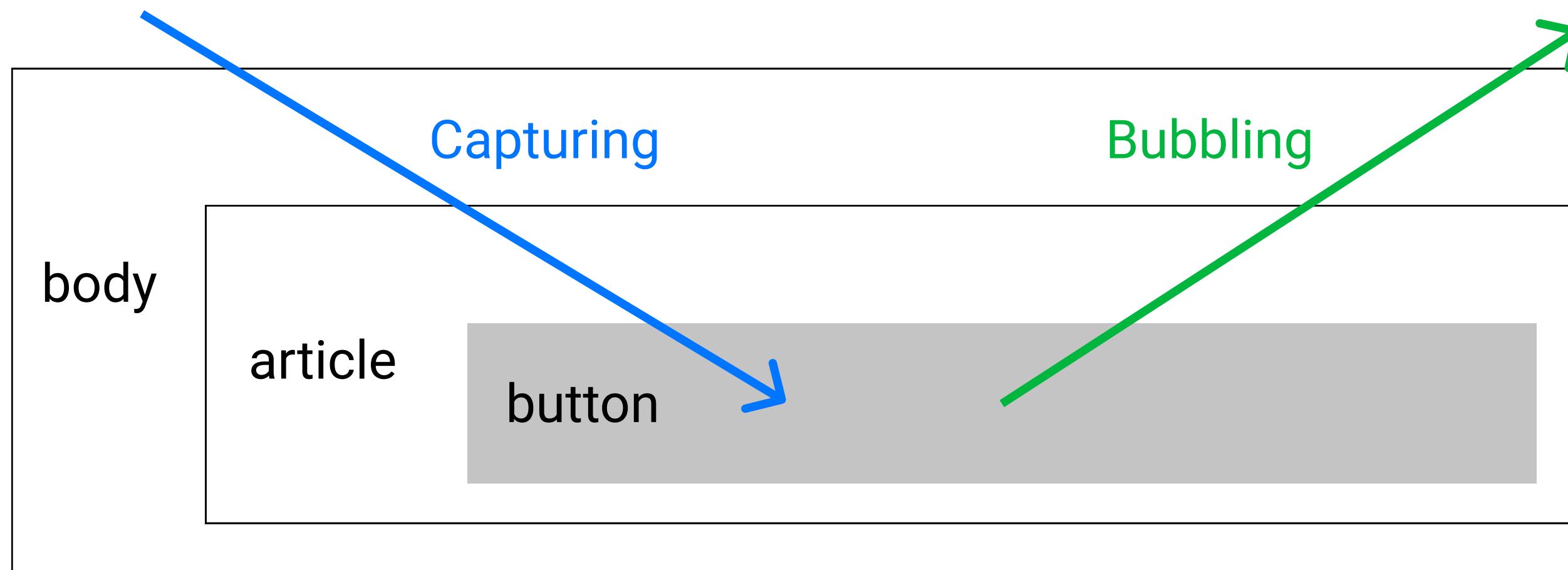
- 이벤트명: 앞서 알아본 이벤트 종류에서 'on' 을 제외시킨것이 이벤트명 입니다. 원하는 이벤트명을 넣으면 됩니다.
- 바인딩할 함수: 바인딩할 함수를 넣어줍니다.
- 이벤트 캡처링 유무: 이벤트 전파 방법에는 Event Capturing, Event Bubbling 이 있습니다. 관련해서 자세한 내용은 다음으로 이어집니다.

이벤트 바인딩 - 이벤트 리스너

이벤트 캡처링, 버블링

javascript의 이벤트는 상하 관계의 요소들에 전파됩니다.

이 전파되는 방향에 따라 명칭이 있는데요, 부모 요소에서 자식 요소방향으로 전파되는 이벤트 방법을 Capturing, 자식요소에서 부모요소로 전파되는 방법을 Bubbling 이라 합니다..



addEventListener의 세번째 매개변수는 어떤 방식으로 이벤트를 전파할 지 결정하는 변수입니다.

기본값은 false고 false일 때 bubbling 방식으로 전파됩니다.

이벤트 바인딩 - 이벤트 리스너

이런 이벤트 전파는 문제점이 있는데
의도하지 않게 원하지 않는 이벤트까지 동작하는 상황이 생긴다는 것 입니다.

예를 들면 부모 요소와 자식 요소에 click 이벤트가 있을 경우, 자식 요소를 click 했을 때 부모 요소의 click 이벤트까지 동작하게 되는경우 입니다.

이런식으로 이벤트가 전파되는 상황을 제어하는 방법 또한 javascript에서 제공하고 있습니다.

- `event.preventDefault()` : 이벤트 자체의 고유동작을 중단시킵니다.
- `event.stopPropagation()` : 이벤트의 전파를 중단시킵니다.

Ex - DOM 수정하기 + click 이벤트

버튼을 만들어서 버튼을 클릭하면 qqImage 가 오른쪽으로 10px 씩 움직이도록 만들어보세요

조건문

조건문이란?

조건문은 대부분 프로그래밍 언어에 존재하는 문법 중 하나입니다.

문법은 다음과 같습니다.

```
if(조건) {  
    로직  
}
```

if라는 예약어를 사용하고 () 안에 조건을 입력합니다. 해당하는 조건이 맞을 경우 괄호안의 로직이 실행되는 구조입니다.

```
if(1 == 1) {  
    console.log('number 1 equal number 1 !!!!')  
}
```

조건문

if ~ else

else는 if 에 들어간 조건이 성립하지 않는경우 실행되는 로직 입니다.

```
if(조건1) {  
    로직 1 <- 조건 1이 '참' 일 경우 실행됨  
} else {  
    로직 2 <- 조건 1이 '거짓' 일 경우 실행됨  
}
```

```
if(1 != 1) {  
    console.log('number 1 NOT equal number 1 !!!!')  
} else {  
    console.log('number 1 equal number 1 !!!!')  
}
```

조건문

if ~ else if

if 말고도 다른 조건을 제시하고 각각의 로직을 구현하고 싶으면 if ~ else if 문법을 사용하면 됩니다.

```
if(조건1) {  
    로직 1 <- 조건 1이 '참' 일 경우 실행됨  
} else if(조건2) {  
    로직 2 <- 조건 2이 '참' 일 경우 실행됨  
} else if(조건3) {  
    로직 3 <- 조건 3이 '참' 일 경우 실행됨  
}
```

```
if(1 != 1) {  
    console.log('number 1 NOT equal number 1 !!!!')  
} else if(1 != 2) {  
    console.log('number 1 NOT equal number 2 !!!!')  
}
```

조건문

if ~ else if ~ else

if 나 else if 로 여러 조건을 걸었지만 앞의 조건들이 모두 실행되지 않은 경우 else 문법을 통해 조건들 외의 상황일 때 동작되어야 할 로직을 구현할 수 있습니다.

```
if(조건1) {
    로직 1 <- 조건 1이 '참' 일 경우 실행됨
} else if(조건2) {
    로직 2 <- 조건 2이 '참' 일 경우 실행됨
} else if(조건3) {
    로직 3 <- 조건 3이 '참' 일 경우 실행됨
} else {
    로직 4 <- 조건 1, 조건 2, 조건 3 이 거짓일 경우
}
```

```
if(1 != 1) {
    console.log('number 1 NOT equal number 1 !!!!')
} else if(1 != 2) {
    console.log('number 1 NOT equal number 2 !!!!')
}
```

Ex - 조건문

함수를 만들어 보세요!

1. 함수의 인자값으로 숫자를 받고 해당 값이 3이면 'this is three' 출력, 아니면 'this is not three' 를 출력하면 됩니다.
2. 함수의 인자값으로 숫자를 받고
 - a. 해당값이 1이면 'this is one'
 - b. 해당값이 2이면 'this is two'
 - c. 해당값이 3이면 'this is three'
 - d. 셋 다 아니면 'wrong number'

를 출력해주세요

Ex - 조건문

쿼카를 집에 보내주세요!

요구사항

1. 버튼을 누르면 쿼카를 오른쪽으로 이동시켜주세요!
2. 쿼카가 오른쪽으로 이동하다가 집에 닿으면 '쿼카가 집에 왔어요!!!' 알림창을 띄워주세요!
3. 집에 도착하면 버튼을 눌러도 더이상 움직이지 않게 해주세요!

javascript minimap

javascript core 문법

- 변수
- 연산자
- 함수
- if ~ else

DOM 다루기

- 노드 탐색
- 요소 가져오기
- 요소 수정하기
- 이벤트 제어