

Javascript 기초 (객체 그리고 this)

📎 자료	<u>Javascrtip</u>
☰ 구분	Javascript
⋮ 과목	

1. JavaScript 객체

Javascript에서 객체는 key와 value로 구성된 property들의 모음이라고 했다.

```
<script>
const person = {
  name: "kevin",
  age: 40,
  greeting: function () {
    console.log(`안녕하세요 저는 ${this.name} 입니다.`);
  },
};
</script>
```

- person 이라는 이름으로 객체를 하나 만들었다. 객체는 { } 중괄호를 이용해서 작성한다.
- name 그리고 age라는 property(속성)을 가진 객체이다.
 - 속성은 key:value 형태로 작성하는데
 - key 는 문자열만 가능하고 value는 모든 자료형이 다 가능하다.
- greeting 이라는 method(메서드)는 인사말을 출력하는 코드로 구성했다.
 - greeting method 안에는 this 키워드는 메서드가 호출 될 때, 메서드가 속해 있는 객체를 참조할 때 사용하는 키워드다
 - 여기서 this는 greeting이 속해있는 person 객체를 의미한다. 따라서 person.name과 같은 의미 이다.

2. Javascript 에서의 this 가 의미하는 것은?

JavaScript에서 this는 굉장히 많이 쓰인다.

그런데 this는 사용하는 언제 어떻게 사용하는가에 따라서 this가 의미하는 것이 달라진다.

1. 함수(메서드)의 선언 방식에 따라

a. 일반함수 vs 화살표함수

(일반함수 = 함수선언식, 일반함수표현식을 말하는것임)

2. 그리고 호출 되는 방식에 따라

a. 전역에서 호출 될 때

b. 객체의 메서드 안에서 호출 될 때

c. 생성자 함수로써 호출 될 때

각각 this가 의미하는 값은 달라진다.

그래서 우리는 상황에 따라 this가 무엇을 참조 하는 지를 잘 구분해 보자.

2-1. 전역에서 사용하는 함수가 일반함수일 경우

this는 호출되는 상황에 따라 의미하는 것이 다르다고 했다.

전역에서 this를 호출되면window 객체를 가리키지만,

객체의 메서드로써 this를 호출하면 이때 this는 해당 객체를 가리킨다.

```
// 1. 전역에서 함수로 호출 (일반함수)
```

```
function person(){  
  return console.log(this)  
}
```

```
person() // this는 전역 객체를 가리킴 (window)
```

```
const test=function(){  
  return console.log(this)
```

```
}
```

```
test() // this는 전역 객체를 가리킴 (window)
```

전역 스코프에서 함수를 하나 사용했는데 함수 선언식으로 정의를 했다.

이때 this는 해당 함수를 감싸고 있는 객체는 최상위 객체인 window이다.

따라서 이때의 this는 최상위 객체인 window를 가르킨다.

이는 함수를 함수표현식 으로 정의를 할 때도 마찬가지 이다.

2-2. 전역에서 사용하는 함수가 화살표함수일 경우

그렇다면 화살표 함수로 바꾸면 어떨까?

```
// 1-2. 전역에서 함수로 호출 (화살표함수)
```

```
const person = () => {  
  console.log(this);  
};  
  
person();
```

이때에도 window가 출력이 된다.

일반함수 그리고 화살표함수에서 모두 window가 출력이 된다.

그러나 window가 출력이 되는 과정은 조금 다르다.

- 일반함수(선언식,표현식)에서의 this 는 함수를 감싸고 있는 객체를 의미한다. 따라서 person 함수를 감싸고 있는 객체, 즉 브라우저의 최상위 객체인 window가 출력이 된다. 그러나
- 화살표 함수에서의 this는 독특하다. 왜냐하면 화살표 함수는 자신만의 this를 가지지 않고,

정의된 함수를 감싸고 있는 함수의 this를 따른다.

지금 과 같이 함수가 전역으로 정의되었다면, this는 window를 참조하게 된다.

(전역 함수는 최상위 함수라고 할수 있으며, 그러한 전역함수를 감싸는 객체는 window가 된다.)

따라서 화살표 함수에서의 this는 **함수가 어디에서 만들어졌느냐에 따라** 가리키는 대상이 달라진다.

함수의 호출 방식이 아니라, 정의된 위치에 따라 결정된다는 것을 기억하자.

3-1. 메서드가 일반함수일때 this 가 의미하는것

이번에는 일반함수 그리고 화살표함수가 객체의 메서드로 사용 되는 경우를 보겠다.

객체의 메서드로 this가 호출이 되었을 때 this가 무엇을 가리키는지 확인해 보자.

```
// 2. 객체의 메서드로 호출 (일반함수)
```

```
const naver = {  
  value: 40,  
  method: function() {  
    console.log(this);  
  }  
};  
naver.method(); // this는 메서드를 호출한 객체를 가리킴 (naver)
```

naver 라는 객체안에 method를 정의해 주었는데 일반함수를 사용했다.

그리고 일반함수 안에는 this라는 키워드를 사용했다.

여기서 this는 메서드를 호출한 객체 (naver)를 가리킨다.

[참고]

하지만 주의할 점은 이때, this의 바인딩이 함수가 어떻게 호출되느냐? 에 따라 달라 질 수 있다.

예를 들어, 메서드를 다른 변수에 할당 한 후 호출을 하면 this가 전역 객체인 window를 가리킨다.

```
const person = {  
  value: 40,  
  method: function () {  
    console.log(this);  
  }  
};
```

```
const test = person.method;
test(); // this는 전역 객체를 가리킴 (window)
```

this 가 해당 메서드를 호출한 객체를 가르키지 않고 전역객체인 window를 가르킨다.

왜일까? 메서드를 다른 변수에 할당해서 사용을 하면, 해당 메서드의 바인딩은 해제되기 때문이다.

(연결이 잠시 끊어짐)

다시 말해, 메서드가 더 이상 객체의 메서드로 인식되지 않기 때문에 this는 전역객체인 window를 가리키게 되는 것이다.

3-2. 메서드가 화살표함수일때 this 가 의미하는것

화살표 함수는 this를 바인딩하지 않고, **자신이 작성된 코드 위치의 상위 스코프(함수)에서 this를 가져온다.** 이로 인해, 객체 안에서 화살표 함수를 메서드로 사용할 경우에도 this는 그 객체가 아니라,

함수가 만들어질 당시의 바깥환경에서 결정된 this를 참조한다.

예시를 보자.

```
const naver = {
  value: 40,
  method: () => {
    console.log(this);
  }
};

naver.method(); // 출력 결과: Window {..}
```

this는 메서드를 감싸고 있는 객체를(naver)를 가르키는 것이 아니라 상위 객체인 window를 가르킨다. 왜일까?

화살표 함수에서 this 라는 것이 존재하지 않는다. (인정하지 않는다.)

그래서 method라는 화살표 함수를 감싸고 있는 함수는 전역에서 정의된 화살표 함수이며, 전역함수에서의 객체는 최상위 객체인 window를 의미한다.

만약에 메서드를 화살표 함수로 표현 했지만 객체를 가르키도록 할 수는 없을까?

사실 방법은 없다. 그냥 객체를 직접 참조해야 한다.

```
const naver = {
  value: 40,
  method: () => {
    console.log(naver); // this 대신 naver를 직접 참조
  }
};

naver.method(); // 출력 결과: naver 객체
```

일반함수 그리고 화살표 함수를 비교하는 예시코드를 더 살펴보자.

아래 코드는 kakao라는 객체 안에 naver라는 객체가 존재한다.

그리고 naver객체 안에는 b 라는 이름의 메서드가 존재한다.

b 메서드를 일반함수 그리고 화살표함수로 놓았을때 this가 가르키는 것을 비교해 보자.

일반함수

```
const kakao = {
  name: "kevin",
  naver: {
    a: 1,
    name: "MINHO",
    b: function () {
      console.log(this.name);
    },
  },
};

kakao.naver.b();
```

화살표 함수

```
const kakao = {
  name: "kevin",
  naver: {
    a: 1,
```

```

    name: "MINHO",
    b: () => console.log(this.name),
  },
};

kakao.naver.b();

```

- 일반함수에서의 this는 메서드를 감싸고 있는 객체를(naver)를 가르키는 것이기 때문에 naver.name인 MINHO가 출력이 되는것이다.

- 화살표함수에서는 this 라는 것이 존재하지 않는다고 했다. 즉, 자신만의 this를 가지지 않는다.

따라서 kakao.naver.b();를 호출하면 this.name는 naver 객체를 감싸고 있는 외부에서 this로 할 값을 끌어온다. 즉, **정의된 위치의 상위 스코프** window의 this를 참조한다.

- b함수는 naver객체 안에 있지만, 객체는 스코프를 만들지 않기 때문에 this는 전역 객체(window)를 참조하게 된다. 따라서 this.name은 window.name을 의미함으로 브라우저에서는 아무것도 출력이 안되고 node.js에서는 undefined가 출력이 될것이다.

이러한 이유 때문에 화살표 함수는 객체의 메서드로는 잘 사용하지 않고, 뒤에 학습할 콜백 함수 에서 많이 사용한다.

예시를 하나 더 살펴보자.

```

const person = {
  name: 'Kevin',
  greeting: function() {
    const inner = () => {
      console.log(`안녕하세요, 저는 ${this.name}입니다.`);
    };
    inner();
  }
}

```

```
};  
person.greeting(); // '안녕하세요, 저는 Kevin입니다.'
```

inner 함수는 화살표 함수로 되어 있다.

화살표 함수는 this를 스스로 정하지 않는다고 했다.

대신, 그 함수가 "정의된 위치"의 외부 함수에서 this를 물려받는다는 특징이 있다고 했다.

이 경우 inner는 greeting 함수 안에서 정의 되었으므로

inner 함수의 this는 greeting 함수의 this를 따라간다.

greeting 함수는 person객체의 메서드로 호출되었기 때문에

greeting의 this는 person 객체를 가르킨다.

따라서 inner함수 안의 this는 person 객체를 가르키기 때문에

출력결과는 "저는 kevin입니다."가 된다.

4. 생성자 함수에서의 this (일반함수 vs 화살표 함수)

자 이번엔 생성자 함수 안에서 this 를 사용해 보자.

먼저 생성자 함수가 무엇인지 확인해 보자.

생성자 함수란? - 객체를 생성하기 위한 틀, 프레임을 의미한다.

예를 통해서 살펴보자.

만약에 key=name 이고 value가 choi 인 객체를 3개 만든다면?

```
const a1={  
  name1:"choi",  
}  
  
const b1={  
  name1:"choi",  
}
```



```
const c1={  
  name1:"choi",  
}
```

이런식으로 표현할 수 있다. 그러나 조금 더 효율적으로 표현 해 보자.

함수를 사용해서 객체를 찍어 내겠다.

(즉, 생성자 함수를 만들어서 인스턴스를 찍어 내겠다)

생성자 함수를 만드는 방법은 다음과 같다.

```
// 생성자 함수는 함수명을 대문자로 많이 시작한다.  
// 1. 함수 생성  
  
function Test(name){  
  this.name=name  
}  
  
// 2. 객체를 생성자 함수를 통해서 찍어낸다.  
const d1=new Test('choi')  
const e1=new Test('choi')
```

위 코드가 생성자 함수에서의 this를 사용한 예가 되겠다.

여기서 this 가 의미 하는 것은 새로 생성 될 "인스턴스"를 의미한다.

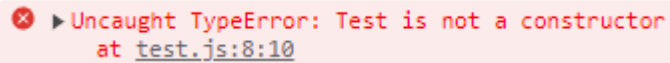
지금 잘 보면 위의 코드는 함수 선언식으로 생성자 함수를 만들었다.

만약에 화살표 함수 (ArrowFunction)으로 생성자 함수를 만들어 보면 어떨까?

```
// 생성자 함수는 함수명을 대문자로 많이 시작한다.  
// 1. 함수 생성  
  
const Test=(name)⇒{this.name=name}
```

```
// 2. 객체를 생성자 함수를 통해서 찍어낸다.  
const d1=new Test('choi')  
const e1=new Test('choi')
```

바로

A screenshot of a JavaScript error message in a browser console. It shows a red 'x' icon followed by the text 'Uncaught TypeError: Test is not a constructor' and 'at test.js:8:10'.

에러가 발생한다.

그 이유는 아까도 말했듯이 화살표 함수는 This 라는것이 존재하지 않기 때문에 new와 함께 실행 할 수 없다. 즉, 인스턴스를 새로 만들 수 없다!

[중요한 결론]

따라서 화살표 함수는 객체를 생성하는 용도로 사용할 수 없다 !!

이러한 특성으로 화살표 함수는 보통 콜백함수로는 많이 사용하지만

생성자 함수로는 사용할 수 없다는 점을 반드시 유의하자!

생성자 함수를 정의 해 줄때는 함수 선언식으로 쓰자.

5. 콜백 함수 (일반함수 vs 화살표 함수)

콜백 함수가 무엇인가?

함수는 필요에 따라 매개변수가 있을 수가 있으며, 그 매개변수에 들어가는 값을 인자값 이라고 한다.

만약에 A라는 함수가 다른 함수의 인자값으로 들어간다면,

인자값으로 들어가는 A 함수를 콜백함수 라고 부른다.

5-1. 콜백 함수가 일반함수

```
const kakao = {  
  numbers: [1, 2, 3],  
  test: function () {  
    this.numbers.forEach(function (number) {  
      console.log(this)  
    })  
  }  
}  
kakao.test()
```

kakao라는 객체 안에 test라는 메소드가 있다.

test 메소드는 kakao 객체의 number라는 속성값을 출력하는 메소드 이다.

this.numbers.forEach() 구문에서 this 는 kakao 라는 객체를 가르키며
forEach문의 콜백함수는 함수 선언식으로 작성 되었다.

forEach 함수에 인자 값으로 들어가는 함수 (콜백함수)를 살펴보자

```
function (number) {  
  console.log(this) // window  
}
```

여기서의 this는 window를 가르킨다.

그 이유는 콜백 함수는 단지 인자값으로 들어가는 일반 함수이다.

콜백 함수는 (kakao 객체와 상관없는) 변수로써의 함수다.

그래서 콜백함수는 전역에서 일반적인 함수 호출을 한 것과 다름이 없기 때문에,
함수 표현식을 일반 함수로 호출 되었을때와 마찬가지로 this가 의미하는 것은
전역객체(window) 브라우저의 최상위 객체를 의미한다.

만약에 콜백함수를 화살표 함수로 작성했다면 어떨까?

5-2. 콜백 함수가 화살표 함수

```
const kakao = {  
  numbers: [1, 2, 3],  
  test: function () {  
    this.numbers.forEach((number) => {  
      console.log(this) // kakao  
    })  
  }  
}  
  
console.log(kakao.test())
```

마찬가지로 kakao라는 객체 안에 test라는 메소드가 있다.

test 메소드는 kakao 객체의 number라는 속성값을 출력하는 메소드 이다.

this.numbers.forEach() 구문에서 this 는 kakao 라는 객체를 가르키며

forEach문 안의 콜백함수는 화살표 함수로 작성 되었다.

```
(number) => {  
  console.log(this)
```

콜백함수를 화살표함수로 했을 경우, this는 window 가 아니라, kakao를 가르킨다.

앞에서 화살표함수는 This라는 것이 존재하지 않는다고 했다.

화살표 함수에서 this를 사용 했다면 해당 화살표 함수를 감싸고 있는 외부스코프(함수)의 this를 그대로 상속받는다고 했다.

지금 forEach문의 콜백함수를 감싸고 있는 함수는 test 함수 이다.

따라서 화살표 함수는 test함수의 this를 그대로 상속받기 때문에 기서 this는 kakao 객체를 가리키게 된다.

6. 결론

따라서 **this 키워드를 사용**하는데 있어서 결론은 다음과 같다.

1. 전역 함수 호출 시 this

- **일반 함수:** 전역에서 호출되면 this는 브라우저 환경에서는 window 객체를 가리킨다.
- **화살표 함수:** 함수를 정의할 당시의 감싸고 있는 외부 스코프(함수)의 this를 참조한다.
- **결론:** `forEach`, `map` 등의 메서드의 콜백 함수에서 this를 사용 시 화살표 함수를 사용하는 것이 좋다.

2. 객체 메서드 호출 시 this

- **일반 함수:** 메서드가 호출된 객체를 가리킨다. 즉, this는 메서드를 호출한 "그" 객체를 의미한다.
- **화살표 함수:** 정의할 당시 감싸고 있는 외부 스코프의 this를 참조하므로, 화살표함수를 객체의 메서드로 사용하면 예상치 못한 결과를 초래할 수 있다.
- **결론:** 객체의 메서드 내에서 this를 사용할 때는 일반 함수를 사용하는 것이 좋다.

3. 생성자 함수에서 this

- **일반 함수:** 정상적으로 동작한다.
- **화살표 함수:** `new` 키워드와 함께 사용할 수 없으므로 생성자 함수로 사용할 수 없다.
- **결론:** 생성자 함수를 정의할 때는 일반 함수를 사용하는 것이 필수다.