코어 JS -3단원

03. this

- 함수와 객체(메서드)를 구분하는 거의 유일한 기능인this

I. 상황에 따라 달라지는 this

- this는 기본적으로 실행 컨텍스트가 생성될 때 함께 결정됨

  → **함수를 호출할 때 결정됨** (∵ 실행 컨텍스트는 함수를 호출할 때 생성됨)

1. 전역 공간에서의 this

- this는 전역 객체를 가리킴

```
// 전역 공간에서만 발생하는 특이한 성질
- 전역 변수를 선언하면 JS 엔진은 이를 전역 객체의 프로퍼티로도 할당함

var a = 1;
console.log(a); // 1
console.log(window.a) // 1
console.log(this.a) // 1

- JS의 모든 변수는 특정 객체의 프로퍼티로서 동작함
```

- 전역변수 선언과 전역객체의 프로퍼티 할당은 비슷하지만 다름
  - '삭제' 명령에서 차이 有

  → var로 선언한 전역변수와 전역객체의 프로퍼티는 **호이스팅 여부** 및 **configurable 여부**에서 차이를 보임

2. 메서드로서 호출할 때 그 메서드 내부에서의 this

함수 vs. 메서드

- 이 둘을 구분하는 유일한 차이는 **독립성**에 있음
  - 함수는 그 자체로 독립적인 기능 수행
  - 메서드는 자신을 호출한 대상 객체에 관한 동작을 수행

'함수로서 호출'과 '메서드로서 호출'을 어떻게 구분할까?

- 함수 앞에 점(.)이 있는지 여부만으로도 간단하게 구분할 수 있음
- 대괄호 표기법도 메서드로서 호출할 것

```
// 함수로서의 호출, 메서드로서의 호출

var func = function(x) {
    console.log(this, x);
};
func(1); // window { ... } 1

var obj = {
    method: func
};
obj.method(2); // { method: f } 2
```

```
// 메서드로서의 호출 - 점 표기법, 대괄호 표기법

var obj = {
    method: function (x) { console.log(this, x) }
};
obj.method(1); // { method: f } 1
obj['method'](2); // { method: f } 2
```

메서드 내부에서의 this

- 어떤 함수를 메서드로서 호출하는 경우 호출 주체는 바로 함수명(프로퍼티명) 앞의 객체
- 점 표기법의 경우 마지막 점 앞에 명시된 객체가 곧 this

3. 함수로서 호출할 때 그 함수 내부에서의 this

함수 내부에서의 this

- this에는 호출한 주체에 대한 정보가 담김

  → 어떤 함수를 함수로서 호출할 경우에는 this가 지정되지 않음

  → this는 전역 객체를 바라봄

메서드 내부함수에서의 this

- 함수를 실행하는 당시의 주변 환경은 중요 X
- 오직 해당 함수를 호출하는 구문 앞에 점 or 대괄호 표기가 있는지가 관건

메서드 내부 함수에서의 this를 우회하는 방법

- var self = this; 처럼 변수를 활용 → 직전 컨텍스트의 this를 바라봄

this를 바인딩하지 않는 함수

- 함수 내부에서 this가 전역객체를 바라보는 문제 보완을 위해 ES6 화살표 함수 도입
- 화살표 함수는 실행 컨텍스트 생성 시 this 바인딩 과정이 빠지게 되어 상위 스코프의 this를 그대로 활용할 수 있음

콜백 함수 호출 시 그 함수 내부에서의 this

- 콜백 함수의 제어권을 가지는 함수가 콜백 함수에서의 this를 결정
- 따로 정의하지 않으면 전역객체를 바라봄

생성자 함수 내부에서의 this

- JS는 함수에 생성자의 역할 O
- new 명령어와 함께 함수 호출 시 해당 함수가 생성자로서 동작함

- 생성자 함수 내부에서의 this는 곧 새로 만들 구체적인 인스턴스 자신이 됨

II. 명시적으로 this를 바인딩하는 방법

1. call 메서드

- call 메서드: 메서드의 호출 주체인 함수를 즉시 실행하도록 하는 명령
- call 메서드의 첫 번째 인자를 this로 바인딩 함
- 이후 인자들을 호출할 함수의 매개변수로 함

2. apply 메서드

- call 메서드와 기능적으로 동일
- apply 메서드는 두 번째 인자를 배열로 받음

3. call / apply 메서드의 활용

a. **유사배열객체에 배열 메서드 적용**

- 유사배열객체의 경우 call 또는 apply 메서드를 이용해 배열 메서드를 적용할 수 있음
- 문지열의 경우 읽기 전용이므로 변경을 가하는 메서드는 에러 발생
- ES6에서는 유사배열객체 or 순회 가능한 모든 종류의 데이터 타입을 배열로 전환하는 Array.from 메서드 새로 도입

b. 생성자 내부에서 다른 생성자를 호출

- 생성자 내부에서 다른 생성자와 공통된 내용이 있을 경우
- call 또는 apply를 이용해 다른 생성자를 호출하면 반복을 줄일 수 있음

c. 여러 인수를 묶어 하나의 배열로 전달하고 싶을 때 - apply 활용

- 여러 개의 인수를 받는 메서드에게 하나의 배열로 인수들을 전달하고 싶을 때 apply 메서드 사용

- ES6에서는 펼치기 연산자(spread operator)를 이용하면 더욱 간단함

4. bind 메서드

- 목적: 함수에 this를 미리 적용함. 부분 적용 함수를 구현함
- 넘겨 받은 this 및 인수들을 바탕으로 새로운 함수를 반환하는 메서드
- bind 메서드를 적용해서 새로 만든 함수는 name 프로퍼티에 'bound'라는 접두어가 붙음
- 상위 컨텍스트의 this를 내부함수나 콜백 함수에 전달

5. 화살표 함수의 예외사항

- 화살표 함수 내부에는 this가 없음
- 접근하려면 스코프체인상 가장 가까운 this에 접근함

6. 별도의 인자로 this를 받는 경우 (콜백 함수 내에서의 this)

- 콜백 함수를 인자로 받는 메서드 중 일부는 추가로 this로 지정할 객체(thisArg)를 인자로 지정할 수 있는 경우 O → 콜백 함수 내부에서 this 값을 원하는 대로 변경할 수 있음
- 배열 메서드에 多
- ES6의 Set, Map 등의 메서드에도 일부 존재함

```
// 콥개 함수와 함께 this Arg를 인자로 받는 메서드
Array - forEach, map, filter, some, every, find, findIndex, f

Set, Map - forEach
```