this (8월 5일)

배정	
상태	8월
속성	@2021년 8월 5일 오후 9:16
언어	

this의 개념

> 자신이 속한 객체 또는 자신이 생성할 인스턴스를 가리키는 자기 참조 변수

> 함수 실행시 호출(invocation) 방법에 의해 동적으로 결정되는 특별한 객체

this 바인딩은 함수가 호출되는 방식에 따라 동적으로 결정된다.

Aa 함수 호출 방식	this가 바인딩되는 객체
일반함수 호출	전역 객체
메서드 호출	메서드를 호출한 객체
생성자 함수 호출	생성된 인스턴스
apply/call/bind 메서드에 의한 간접 호출	메서드에 첫번째 인수로 전달한 객체
제목 없음	

1. 일반함수 호출 예시 코드

```
// var 키워드로 선언한 전역 변수 value는 전역 객체의 프로퍼티다.
var value = 1;
// const 키워드로 선언한 전역 변수 value는 전역 객체의 프로퍼티가 아니다.
// const value = 1;

const obj = {
```

```
  value: 100,
  foo() {
    console.log("foo's this: ", this);  // {value: 100, foo: ƒ}
    console.log("foo's this.value: ", this.value); // 100

    // 메서드 내에서 정의한 중첩 함수
    function bar() {
      // console.log("bar's this: ", this); // window
      console.log("bar's this.value: ", this.value); // 1
    }

    // 메서드 내에서 정의한 중첩 함수도 일반 함수로 호출되면 중첩 함수 내부의 this에는 전역 객체가 바인딩
된다.
    bar();
  }
};

obj.foo(); // "foo's this: " { value: 100, foo: ƒ foo() }
          // "foo's this.value: " 100
          // "bar's this.value: " 1
```

2. 메서드 호출 예시 코드

```
const person = {
  name: 'Lee',
  getName() {
    // 메서드 내부의 this는 메서드를 호출한 객체에 바인딩된다.
    return this.name;
  }
};

// 메서드 getName을 호출한 객체는 person이다.
console.log(person.getName()); // Lee

const anotherPerson = {
  name: 'Kim'
};
// getName 메서드를 anotherPerson 객체의 메서드로 할당
anotherPerson.getName = person.getName;

// getName 메서드를 호출한 객체는 anotherPerson이다.
console.log(anotherPerson.getName()); // Kim

// getName 메서드를 변수에 할당
const getName = person.getName;

// getName 메서드를 일반 함수로 호출
console.log(getName()); // ''
// 일반 함수로 호출된 getName 함수 내부의 this.name은 브라우저 환경에서 window.name과 같다.
// 브라우저 환경에서 window.name은 브라우저 창의 이름을 나타내는 빌트인 프로퍼티이며 기본값은 ''이다.
// Node.js 환경에서 this.name은 undefined다.
```

3. 생성자 함수 호출 예시 코드

```
// 생성자 함수
function Circle(radius) {
  // 생성자 함수 내부의 this는 생성자 함수가 생성할 인스턴스를 가리킨다.
  this.radius = radius;
  this.getDiameter = function () {
    return 2 * this.radius;
  };
}

// 반지름이 5인 Circle 객체를 생성
const circle1 = new Circle(5);
// 반지름이 10인 Circle 객체를 생성
const circle2 = new Circle(10);

console.log(circle1.getDiameter()); // 10
console.log(circle2.getDiameter()); // 20

// new 연산자와 함께 호출하지 않으면 생성자 함수로 동작하지 않는다. 즉, 일반적인 함수의 호출이다.
const circle3 = Circle(15);

// 일반 함수로 호출된 Circle에는 반환문이 없으므로 암묵적으로 undefined를 반환한다.
console.log(circle3); // undefined

// 일반 함수로 호출된 Circle 내부의 this는 전역 객체를 가리킨다.
console.log(radius); // 15
```

4. apply와 call , bind 개념과 예시 코드

💡 apply 메소드 : 호출할 함수의 인수를 **배열**로 묶어 전달

💡 call 메소드 : 호출할 함수의 인수를 **'쉼표'** 로 구분한 리스트 형식으로 전달

```
function getThisBinding() {
  console.log(arguments);
  return this;
}

// this로 사용할 객체
const thisArg = { a: 1 };

// getThisBinding 함수를 호출하면서 인수로 전달한 객체를 getThisBinding 함수의 this에 바인딩한다.
// apply 메서드는 호출할 함수의 인수를 배열로 묶어 전달한다.
console.log(getThisBinding.apply(thisArg, [1, 2, 3]));
// Arguments(3) [1, 2, 3, callee: ƒ, Symbol(Symbol.iterator): ƒ]
```

```
// {a: 1}

// call 메서드는 호출할 함수의 인수를 쉼표로 구분한 리스트 형식으로 전달한다.
console.log(getThisBinding.call(thisArg, 1, 2, 3));
// Arguments(3) [1, 2, 3, callee: ƒ, Symbol(Symbol.iterator): ƒ]
// {a: 1}
```

bind 메소드 : 함수를 호출하지 않고 **this를 사용할 객체만** 전달

```
function getThisBinding() {
  return this;
}

// this로 사용할 객체
const thisArg = { a: 1 };

// bind 메서드는 함수에 this로 사용할 객체를 전달한다.
// bind 메서드는 함수를 호출하지는 않는다.
console.log(getThisBinding.bind(thisArg)); // getThisBinding
// bind 메서드는 함수를 호출하지는 않으므로 명시적으로 호출해야 한다.
console.log(getThisBinding.bind(thisArg)()); // {a: 1}
```

화살표 함수에서의 this

- lexical this

화살표 함수는 함수 자체의 this 바인딩을 갖지 않는다. 따라서 화살표 함수 내부에서 this를 참조하면 상위 스코프의 this를 그대로 참조한다. 이를 lexical this라고 한다.

```
// 화살표 함수는 상위 스코프의 this를 참조한다.
() => this.x;

// 익명 함수에 상위 스코프의 this를 주입한다. 위 화살표 함수와 동일하게 동작한다.
(function () { return this.x; }).bind(this);
```

- 화살표함수와 화살표함수의 중첩

화살표함수와 화살표함수가 중첩되어 있다면 가장 가까운 상위 함수중에서 화살표 함수가 아닌 함수의 this를 참조한다.(상위 화살표 함수에도 this 바인딩이 없기때문)

```
// 중첩 함수 foo의 상위 스코프는 즉시 실행 함수다.
// 따라서 화살표 함수 foo의 this는 상위 스코프인 즉시 실행 함수의 this를 가리킨다.
```

```
(function () {
  const foo = () => console.log(this);
  foo();
}).call({ a: 1 }); // { a: 1 }

// bar 함수는 화살표 함수를 반환한다.
// bar 함수가 반환한 화살표 함수의 상위 스코프는 화살표 함수 bar다.
// 하지만 화살표 함수는 함수 자체의 this 바인딩을 갖지 않으므로 bar 함수가 반환한
// 화살표 함수 내부에서 참조하는 this는 화살표 함수가 아닌 즉시 실행 함수의 this를 가리킨다.
(function () {
  const bar = () => () => console.log(this);
  bar()();
}).call({ a: 1 }); // { a: 1 }
```

- 화살표함수가 전역 함수일 경우

화살표 함수가 전역 함수라면 화살표 함수의 this는 **전역 객체**를 가리킨다.

```
// 전역 함수 foo의 상위 스코프는 전역이므로 화살표 함수 foo의 this는 전역 객체를 가리킨다.
const foo = () => console.log(this);
foo(); // window
```

- 프로퍼티에 할당한 화살표함수

가장 가까운 상위 함수 중에서 화살표 함수가 아닌 함수의 this를 참조한다.

```
// increase 프로퍼티에 할당한 화살표 함수의 상위 스코프는 전역이다.
// 따라서 increase 프로퍼티에 할당한 화살표 함수의 this는 전역 객체를 가리킨다.
const counter = {
  num: 1,
  increase: () => ++this.num
};

console.log(counter.increase()); // NaN
```

- 화살표는 함수자체의 this 바인딩을 갖지않음

this를 교체 할 수 없고 언제나 상위스코프의 this 바인딩을 참조한다.

```
const add = (a, b) => a + b;

console.log(add.call(null, 1, 2));    // 3
console.log(add.apply(null, [1, 2])); // 3
console.log(add.bind(null, 1, 2)());  // 3

------------------------------------------------------------------------------
window.x = 1;

const normal = function () { return this.x; };
```

```
// 상위 스코프  window 를 참조함
const arrow = () => this.x;

console.log(normal.call({ x: 10 })); // 10
// 상위 스코프를 참고함.
console.log(arrow.call({ x: 10 }));  // 1
```

- 메서드를 화살표 함수로 정의 하는것은 피해야한다.

좋은예와 나쁜예

결론, 즉시실행함수 일경우 this가 스코프내의 'name : lee' 를 참조하는데 화살표함수일 경우 this가 **'전역'** 을 참조하게 된다.

```
// Bad
const person = {
  name: 'Lee',
  sayHi: () => console.log(`Hi ${this.name}`)
};

// sayHi 프로퍼티에 할당된 화살표 함수 내부의 this는 상위 스코프인 전역의 this가 가리키는
// 전역 객체를 가리키므로 이 예제를 브라우저에서 실행하면 this.name은 빈 문자열을 갖는
// window.name과 같다. 전역 객체 window에는 빌트인 프로퍼티 name이 존재한다.
person.sayHi(); // Hi

--------------------------------------------------------------------------------

// Good
const person = {
  name: 'Lee',
  sayHi() {
    console.log(`Hi ${this.name}`);
  }
};

person.sayHi(); // Hi Lee
```

- 클래스 필드의 용법례

```
// Bad
class Person {
  // 클래스 필드 정의 제안
  name = 'Lee';
  sayHi = () => console.log(`Hi ${this.name}`);
}

const person = new Person();
person.sayHi(); // Hi Lee

--------------------------------------------------------------------------------
// Good
class Person {
```

```
  // 클래스 필드 정의
  name = 'Lee';

  sayHi() { console.log(`Hi ${this.name}`); }
}
const person = new Person();
person.sayHi(); // Hi Lee
```