Chap22 this

22-1 this 키워드

▼ 단어 정리

- arguments : 함수에 전달된 인수에 해당하는 `Array` 형태의 객체이다.

```
function func1(a, b, c) {
  console.log(arguments[0]);
  // Expected output: 1

  console.log(arguments[1]);
  // Expected output: 2

  console.log(arguments[2]);
  // Expected output: 3
}

func1(1, 2, 3); //인수
```

- this 바인딩 : this와 this가 가리킬 객체를 바인딩하는 것

this란?

- 자신이 속한 객체 또는 자신이 생성할 인스턴스를 가리키는 **자기 참조 변수**
- js엔진에 의해 암묵적으로 생성되며, 코드 어디서든 참조할 수 있다.
- **함수 호출 방식에 따라** this에 바인딩 될 값이 **동적으로 결정**된다.

    - 일반함수에서는 `전역객체` window가 바인딩된다.
    - 생성자함수에서는 생성자 함수가 생성할 인스턴스가 바인딩된다.
- strict mode 역시 this바인딩에 영향을 준다.
    - strict mode가 아닌 경우, 일반 함수 호출에서 `this`는 전역 객체를 가리킵니다.
    - strict mode에서는 일반 함수 호출에서 `this`는 `undefined`가 됩니다.
    - strict mode에서도 `new` 키워드를 사용한 생성자 함수 호출에서는 `this`가 새로 생성된 객체를 가리킵니다.

22-2 함수 호출 방식과 this 바인딩

1. **일반 함수에서 호출**

- 전역객체가 바인딩된다.
- 즉 일반 함수로 호출된 모든 함수(중첩 함수, 콜백 함수 포함)내부의 `this`에는 **전역 객체**가 바인딩 된다.
- 다만 **strict모드 일반함수**면 `undefined`가 반인딩된다.

2. **메서드에서 호출**

- 메서드 내부의 this에는 메서드를 호출한 객체, 즉 메서드를 호출할 때 마침표(.) 연산자 앞에 기술한 **객체**가 바인딩된다.

주의) 메서드 내부 this는 메소드를 소유한 객체가 아닌 **메서드를 호출한 객체**에 바인딩된다는 것이다.

- 만약 메서드를 일반 변수에 할당하여 호출하면 일반 함수로 호출된다.
- 프로토타입 메서드 내부에서 사용된 `this`도 일반 메서드와 동일하게 해당 메서드를 호출한 객체에 바인딩 된다.

3. **생성자 함수에서 호출**

- 생성자 함수가 생성할 인스턴스가 바인딩된다.'

```
// 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
```

4. Funtion.prototype.apply/call/bind 메서드에 의한 간접 호출

- `Function.prototype.apply`, `Function.prototype.call` **메서드**

`apply`와 `call`메서드는 함수에 `this`로 사용할 객체를 전달하고 함수를 호출한다.

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

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

// getThisBinding 함수를 호출하면서 인수로 전달한 객체를 getThisBindi
// 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):
// {a: 1}

// call 메서드는 호출할 함수의 인수를 쉼표로 구분한 리스트 형식으로 전달한
console.log(getThisBinding.call(thisArg, 1, 2, 3));
```

```
// Arguments(3) [1, 2, 3, callee: ƒ, Symbol(Symbol.iterator):
// {a: 1}
```

`apply` 메서드는 함수의 인수를 **배열로 묶어 전달**하고, `call` 메서드는 함수의 인수를 **쉼표로 구분한 리스트 형식으로 전달**한다.

`apply` 와 `call` 메서드의 대표적인 용도는 `arguments` 객체와 같은 **유사 배열 객체에 배열 메서드를 사용하는 경우**이다.

`arguments` 객체는 배열이 아니기 때문에(유사 배열) `Array.prototype.slice` 같은 배열 메서드를 사용할 수 없으나, `apply` 나 `call` 메서드를 이용하면 가능하다.

- `Function.prototype.bind` **메서드**

`bind` 메서드는 `apply` 와 `call` 메서드와 달리 함수를 호출하지는 않고 `this` 로 사용할 객체만 전달한다.(명시적으로 호출 `()` 을 해줘야 함.

정리

함수 호출 방식	**`this` 바인딩**
일반 함수 호출	전역 객체
메서드 호출	메서드를 호출한 객체
생성자 함수 호출	생성자 함수가 (미래에)생성할 인스턴스
`Function.prototype.apply/call/bind` 메서드에 의한 간접 호출	`Function.prototype.apply/call/bind` 메서드에 첫 번째 인수로 전달된 객체