this에 대해서

- 자바스크립트에서 모든 함수는 실행될 때마다 함수 내부에 `this`라는 객체가 추가된다. `arguments`라는 유사 배열 객체와 함께 함수 내부로 암묵적으로 전달되는 것이다. 그렇기 때문에 자바스크립트에서의 `this`는 함수가 호출된 상황에 따라 그 모습을 달리한다.

▼ 상황 1. 객체의 메서드를 호출할 때

- 객체의 프로퍼티가 함수일 경우 메서드라고 부른다. `this`는 함수를 실행할 때 함수를 소유하고 있는 객체(메서드를 포함하고 있는 인스턴스)를 참조한다. 즉 해당 메서드를 호출한 객체로 바인딩된다. `A.B`일 때 `B`함수 내부에서의 `this`는 `A`를 가리키는 것이다.

```
var myObject = {
  name: "foo",
  sayName: function() {
    console.log(this);
  }
};
myObject.sayName();
// console> Object {name: "foo", sayName: sayName()}
```

▼ 상황 2. 함수를 호출할 때

- 특정 객체의 메서드가 아니라 함수를 호출하면, 해당 함수 내부 코드에서 사용된 this 는 전역객체에 바인딩 된다. `A.B`일 때 `A`가 전역 객체가 되므로 `B`함수 내부에서의 `this`는 당연히 전역 객체에 바인딩 되는 것이다.

```
var value = 100;
var myObj = {
  value: 1,
  func1: function() {
    console.log(`func1's this.value: ${this.value}`);

    var func2 = function() {
      console.log(`func2's this.value ${this.value}`);
    };
    func2();
  }
};

myObj.func1();
```

```
// console> func1's this.value: 1
// console> func2's this.value: 100
```

▼ 상황 3. 생성자 함수를 통해 객체를 생성할 때

- 그냥 함수를 호출하는 것이 아니라 `new`키워드를 통해 생성자 함수를 호출할 때는 또 `this`가 다르게 바인딩 된다. `new` 키워드를 통해서 호출된 함수 내부에서의 `this`는 객체 자신이 된다. 생성자 함수를 호출할 때의 `this` 바인딩은 생성자 함수가 동작하는 방식을 통해 이해할 수 있다.

  `new` 연산자를 통해 함수를 생성자로 호출하게 되면, 일단 빈 객체가 생성되고 this가 바인딩 된다. 이 객체는 함수를 통해 생성된 객체이며, 자신의 부모인 프로토타입 객체와 연결되어 있다. 그리고 return 문이 명시되어 있지 않은 경우에는 `this`로 바인딩 된 새로 생성한 객체가 리턴된다.

```
var Person = function(name) {
  console.log(this);
  this.name = name;
};

var foo = new Person("foo"); // Person
console.log(foo.name); // foo
```

▼ 상황 4. bind, call, apply 를 통한 호출

- 상황 1, 상황 2, 상황 3 에 의존하지 않고 `this`를 자바스크립트 코드로 주입 또는 설정할 수 있는 방법이다. 상황 2 에서 사용했던 예제 코드를 다시 한 번 보고 오자. `func2`를 호출할 때, `func1`에서의 this 를 주입하기 위해서 위 세가지 메소드를 사용할 수 있다. 그리고 세 메소드의 차이점을 파악하기 위해 `func2`에 파라미터를 받을 수 있도록 수정한다.

- `bind` 메소드 사용

```
var value = 100;
var myObj = {
  value: 1,
  func1: function() {
    console.log(`func1's this.value: ${this.value}`);

    var func2 = function(val1, val2) {
      console.log(`func2's this.value ${this.value} and ${val1} and ${val2}`);
    }.bind(this, `param1`, `param2`);
    func2();
  }
};
```

```
myObj.func1();
// console> func1's this.value: 1
// console> func2's this.value: 1 and param1 and param2
```

- `call` 메소드 사용

```
var value = 100;
var myObj = {
  value: 1,
  func1: function() {
    console.log(`func1's this.value: ${this.value}`);

    var func2 = function(val1, val2) {
      console.log(`func2's this.value ${this.value} and ${val1} and ${val2}`);
    };
    func2.call(this, `param1`, `param2`);
  }
};

myObj.func1();
// console> func1's this.value: 1
// console> func2's this.value: 1 and param1 and param2
```

- `apply` 메소드 사용

```
var value = 100;
var myObj = {
  value: 1,
  func1: function() {
    console.log(`func1's this.value: ${this.value}`);

    var func2 = function(val1, val2) {
      console.log(`func2's this.value ${this.value} and ${val1} and ${val2}`);
    };
    func2.apply(this, [`param1`, `param2`]);
  }
};

myObj.func1();
// console> func1's this.value: 1
// console> func2's this.value: 1 and param1 and param2
```

- `bind` VS `apply`, `call`
  - 우선 `bind`는 함수를 선언할 때, `this`와 파라미터를 지정해줄 수 있으며, `call`과 `apply`는 함수를 호출할 때, `this`와 파라미터를 지정해준다.
- `apply` VS `bind`, `call`

- `apply` 메소드에는 첫번째 인자로 `this`를 넘겨주고 두번째 인자로 넘겨줘야 하는 파라미터를 배열의 형태로 전달한다. `bind`메소드와 `call`메소드는 각각의 파라미터를 하나씩 넘겨주는 형태이다.