KWEB 준회원 스터디

JavaScript 재입문하기

@KWEB

목차

1. 왜 자바스크립트 인가?
2. 타입과 객체
3. 함수와 프로토타입 체이닝
4. 스코프와 클로저

왜 자바스크립트 인가?

> 잠시 멈춰서 우리가 왜 자바스크립트를 공부하고 있었는지 상기 해봅시다.

과거

- 개발자들에게 하루면 충분히 사용할 수 있는 언어?
- 불완전한 언어. 제대로 된 취급조차 받지 못한 비운의 언어(…)

1 단계 - 스마트폰의 폭발적 증가는 사용자에게 웹을 통해 어디서든 원하는 작업을 할 수 있음을 부각.
- OSMU(One Source Multi Use)

2 단계 - 웹 브라우저의 복잡도 증가
- 앱스러운 UI, UX 구현에는 자바스크립트가 필수

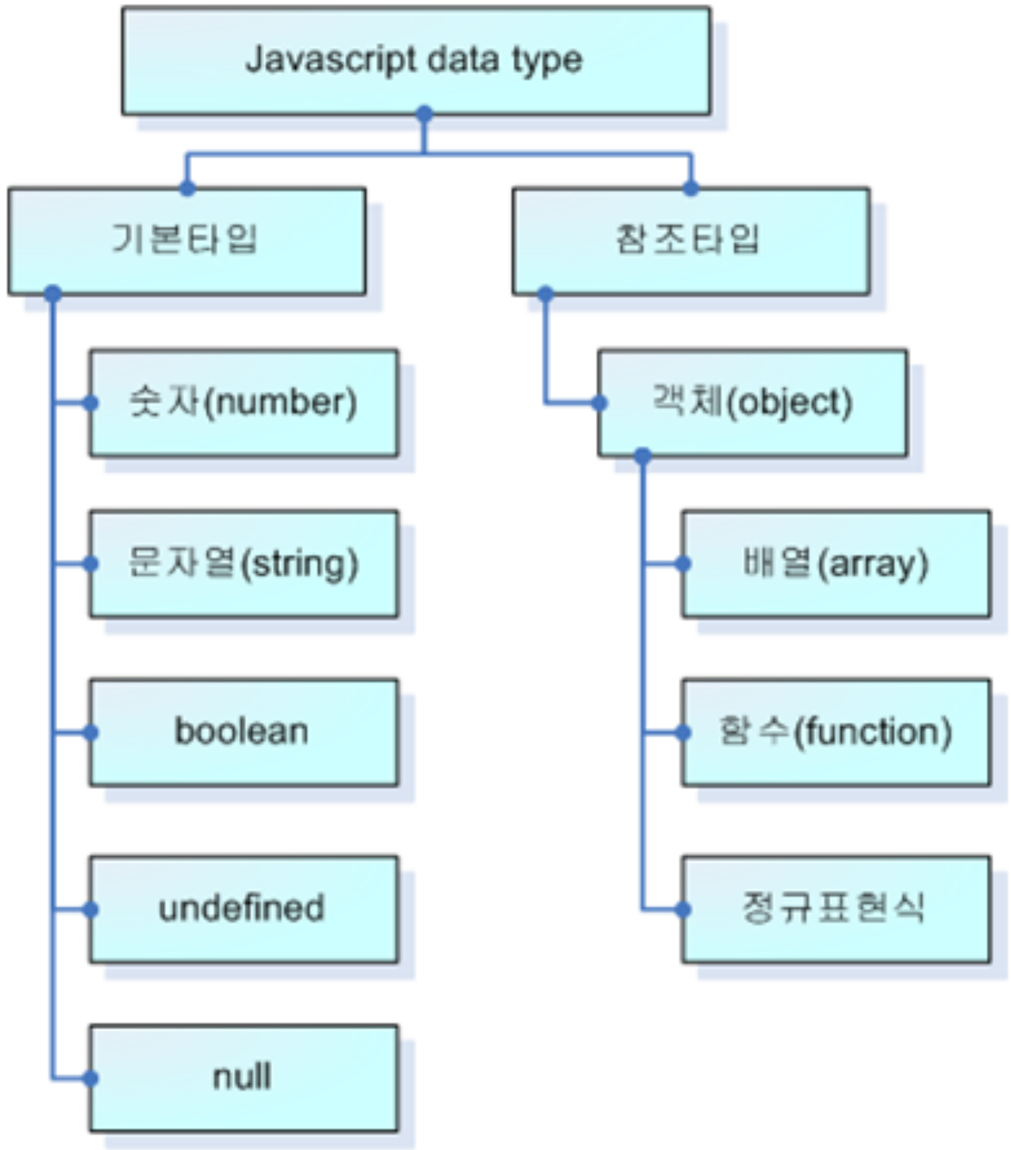
현재

- **“자바스크립트가 세상을 먹어치우고 있다!”**
- 웹 브라우저를 벗어나 서버, 모바일, 데스크탑까지 진출
- 깃허브 레포지토리, 커밋 및 푸쉬 1위

타입과 객체

> 자바스크립트의 타입과 객체에 대해서 더 자세히 알아봅시다.

데이터 타입


Javascript data type
기본타입
참조타입
숫자(number)
문자열(string)
boolean
undefined
null
객체(object)
배열(array)
함수(function)
정규표현식

숫자(Number)

모든 숫자를 64비트 부동 소수점 형태로 저장한다. (C언어의 double형과 유사)

실수 연산을 할 때는 주의! (0.1 + 0.2 !== 0.3)

안전하게 표현할 수 있는 최대 정수는 2^53 - 1

- **NaN**

  경계값의 일종. “Not a Number”를 의미함

  Math.sqrt(-1) 또는 3 / 0 처럼 연산에 실패하였을 때 반환되는 값

  자기 자신과 동등하지 않은 유일한 값(?!) (NaN !== NaN)

null과 undefined

‘값이 비어있음’을 나타낸다.

각각 null가 undefined라는 값을 갖고 있다. (즉, 데이터 타입이라는 집합에 원소가 하나)

기본적으로 값이 활당되지 않은 변수는 **undefined** 타입이다.

null의 경우 개발자가 명시적으로 값이 비어있음을 나타낼 때 사용한다.

웃긴 점(?)

```
typeof null === 'object' // true
```

null을 확인할 때는

```
var nullVal = null;
nullVal === null; // true
```

객체

기본 타입을 제외한 모든 값. 배열, 함수, 정규표현식 등도 모두 객체.

‘이름(key):값(value)’ 형태의 프로퍼티들을 저장하는 컨테이너.

- **프로퍼티 읽기**

```
var foo = {};

foo.nickname;
console.log(foo.nickname);
```

- **프로퍼티 갱신**

```
foo.nickname = 'say my name!';
foo['major'] = 'hello world!';
```

객체

객체의 모든 연산은 실제 값이 아닌 참조값으로 처리된다. (포인터!)

```
var objA = {
    val: 40
};
var objB = objA;

console.log(objA.val); // 40
console.log(objB.val); // 40

objB.val = 50;

console.log(objA.val) // 50
console.log(objB.val) // 50
```

함수와 프로토타입 체이닝

> 자바스크립트는 프로토타입 기반의 객체 지향 언어입니다.

함수도 객체다.

```
function add(x, y) {
    return x + y;
}

add.result = add(3, 2);
add.status = 'OK';

console.log(add.result); // 5
console.log(add.status); // 'OK'
```

함수도 객체다.

함수의 코드는 객체의 **[[Code]] 내부 프로퍼티**에 자동으로 저장되어 메소드 호출 시에 해당 코드를 실행한다.

- **일급 객체 (First Class)**

  리터럴에 의한 생성

  변수나 배열의 요소, 객체의 프로퍼티 등에 할당 가능

  함수의 인자로 전달 가능

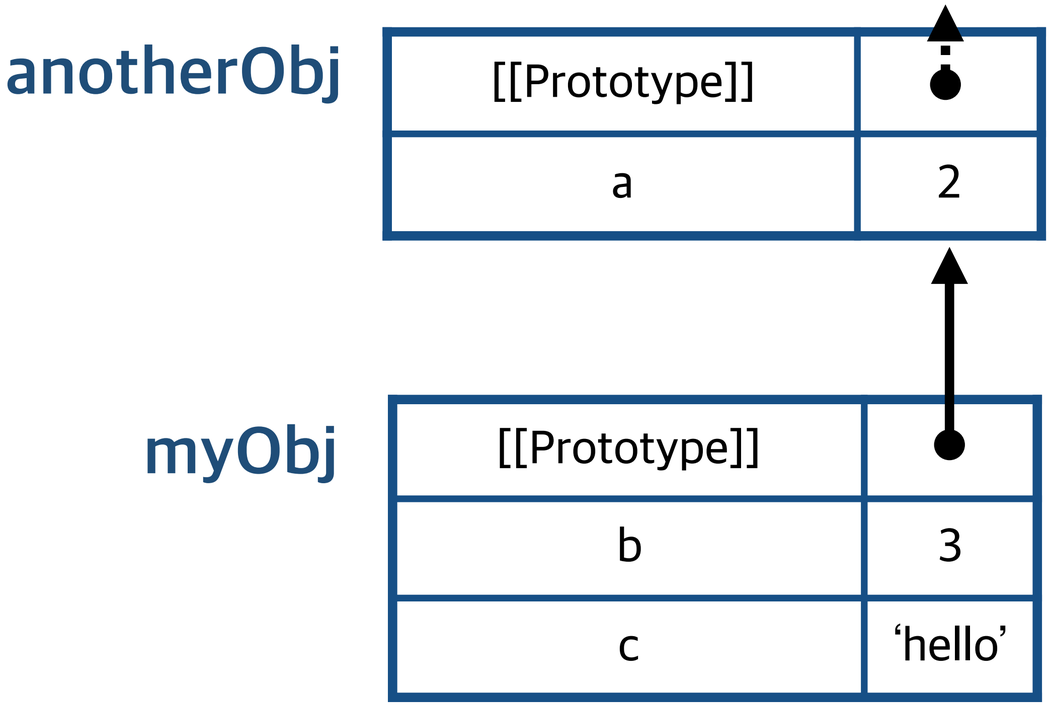
  함수의 리턴값으로 리턴 가능

  동적으로 프로퍼티를 생성 및 할당 가능

프로토타입

자바스크립트의 모든 객체는 자신의 부모 역할을 하는 객체와 연결되어 있다.

내부 프로퍼티인 [[Prototype]]은 자신의 프로토타입 객체를 가리킨다.

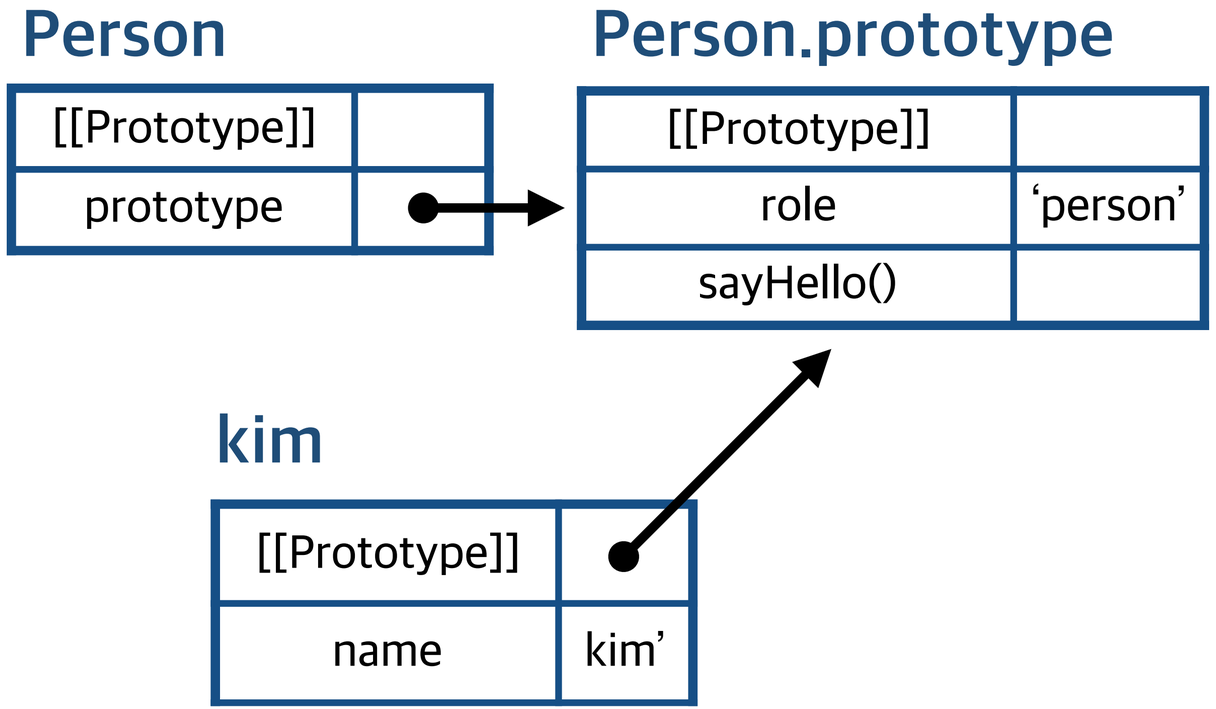



```
var anotherObj = {
    a: 2
};

var myObj = Object.create(anotherObj);
myObj.b = 3;
myObj.c = 'hello';

myObj.a // 2
```

함수와 프로토타입

모든 함수는 객체로서 prototype 프로퍼티를 가지고 있다.

함수가 생성자로서 쓰일 때 prototype 프로퍼티가 생성자로 생성된 객체의 프로토타입 객체가 된다.



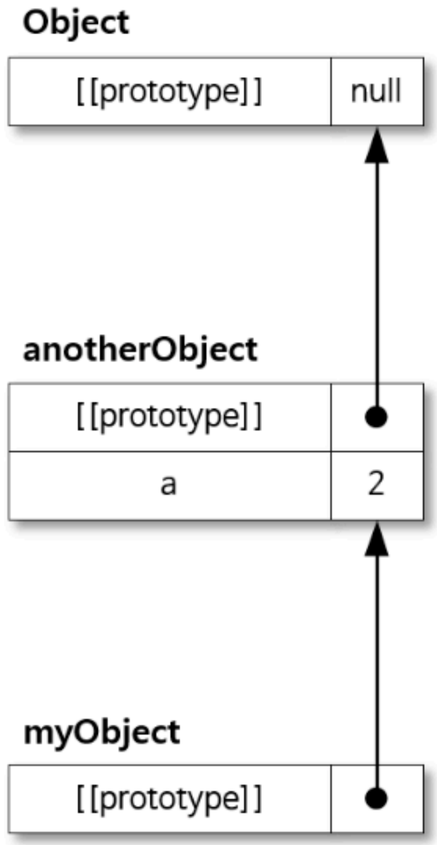

```
function Person(name) {
    this.name = name;
}

Person.prototype.role = 'person';

Person.prototype.sayHello = function () {
    return 'Hello, ' + this.name;
};

var kim = new Person('kim');

console.log(kim.role); // 'person'
console.log(kim.sayHello()); // 'Hello, kim'
```

프로토타입 체이닝

객체의 프로퍼티를 찾지 못하면 해당 객체의 프로토타입 객체에서 그 프로퍼티를 찾는다.

[[Prototype]] 링크를 따라 프로퍼티를 수색: **프로토타입 체이닝**

[[Prototype]] 체인이 끝나는 지점은 Object.prototype이다.



스코프와 클로저

> 자바스크립트에서 변수의 유효 범위

실행 컨텍스트

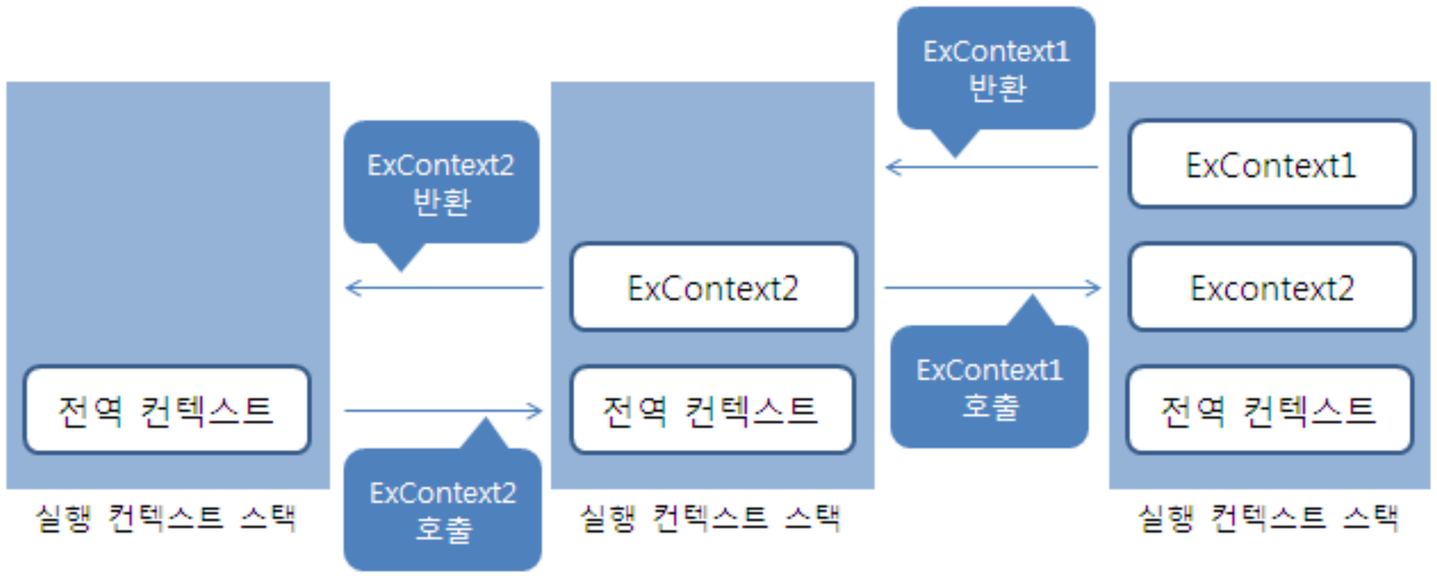
실행 가능한 자바스크립트 코드 블록이 실행되는 환경

전역 코드, eval() 함수로 실행되는 코드, 함수 안의 코드를 실행할 경우 생성됨

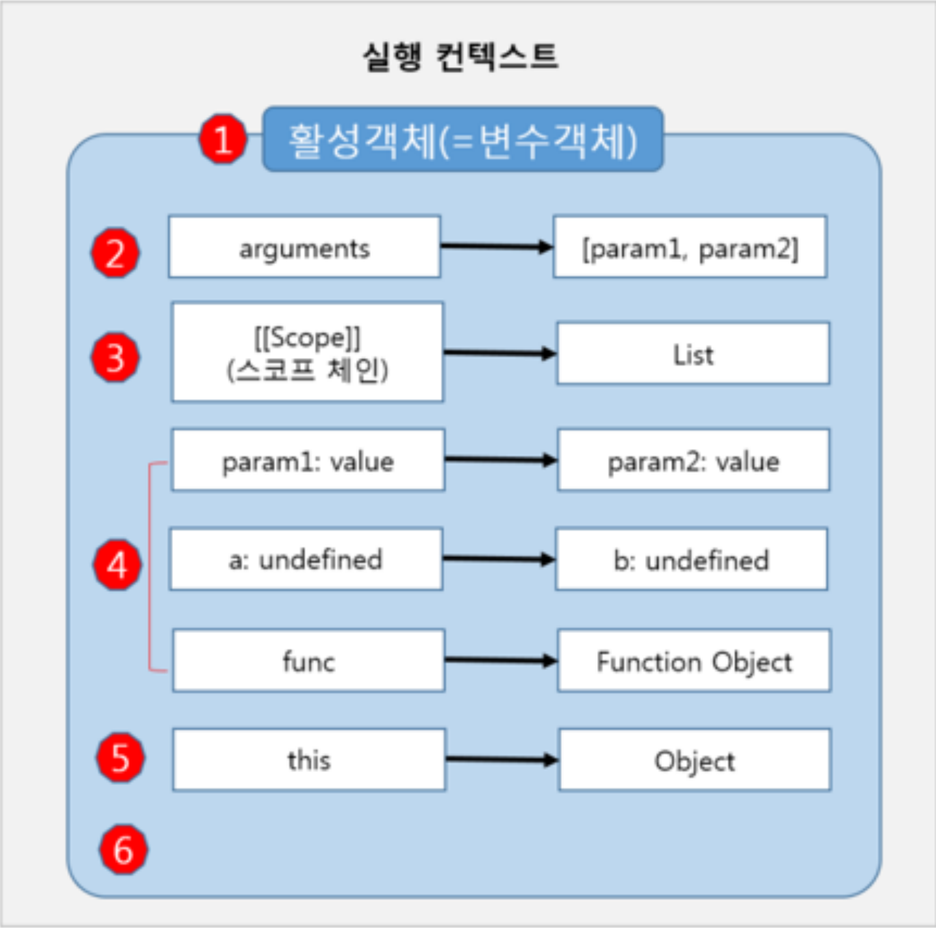
코드가 실행되면 실행 컨텍스트가 하나씩 차곡차곡 쌓이고 제일 위에 위치하는 실행 컨텍스트가 현재 실행되고 있는 컨텍스트다.

실행 컨텍스트

```
console.log('전역 컨텍스트');

function exContext1() {
    console.log('ExContext1');
}

function exContext2() {
    exContext1();
    console.log('ExContext2');
}

exContext2();


// 전역 컨텍스트
// ExContext1
// ExContext2
```



실행 컨텍스트 생성 과정



(1) 활성객체 생성

(2) arguments 객체 생성

(3) 스코프 정보 생성

(4) 변수 생성

(5) this 바인딩

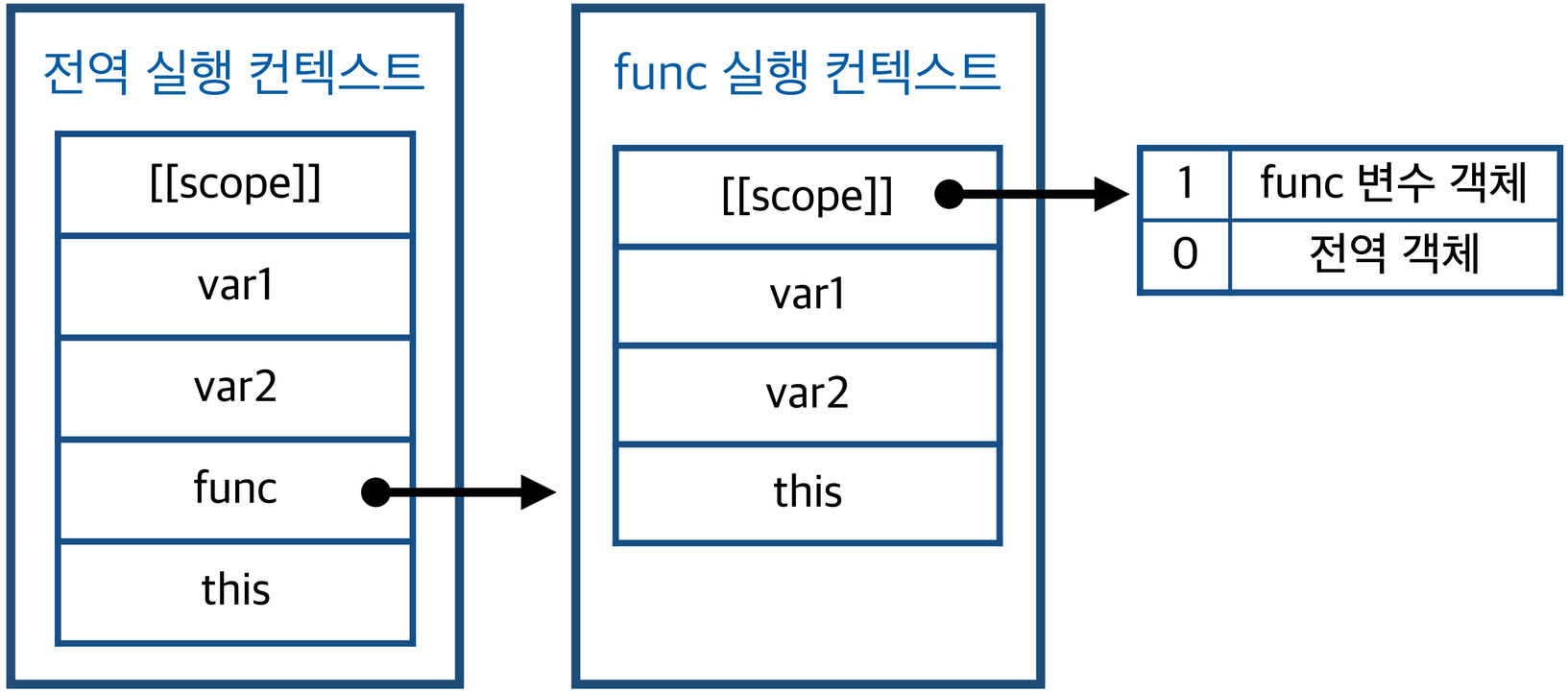
(6) 코드 실행

스코프 체인

스코프는 어디서 어떻게 변수를 찾는가를 결정하는 규칙의 집합이다.

변수의 유효 범위를 나타내는 스코프가 [[scope]] 내부 프로퍼티로 각 함수 객체 내에서 연결리스트의 형식으로 관리된다. -> 스코프 체인

```
var var1 = 1;
var var2 = 2;

function func() {
    var var1 = 10;
    var var2 = 20;

    console.log(var1);
    console.log(var2);
}

func();
console.log(var1);
console.log(var2);
```



클로저

```
function foo() {
    var count = 0;

    function increase() {
        count++;
    }

    function getCount() {
        return count;
    }

    return {
        increase: increase,
        getCount: getCount
    };
}

var baz = foo();

baz.increase();
baz.getCount(); // 1
```

이미 생명 주기가 끝난 외부 함수의 변수를 참조하여 스코프 체인이 그대로 남아있어 접근이 가능한 객체.

foo() 함수의 실행 컨텍스트가 종료되어도 객체 baz는 foo()의 스코프가 중첩되어 있다. -> foo()의 스코프에 접근이 가능하다.

감사합니다

> KWEB 1학기 준회원 마지막 스터디입니다. 모두들 수고하셨습니다 :)