자바스크립트

스코프

목차

스코프

- 스코프 (Scope)의 개념
  - 변수가 유효성을 가지는 범위
  - 스코프는 기본적으로 프로세스 메모리 맵의 구조에 의존적.
  - 선언된 위치에 따라 스코프가 결정
  - 같은 이름으로 변수가 선언 된 경우 스코프가 겹칠 수 있음.
  - 스코프가 겹칠때 어떤 변수가 우선권을 가지는지 판단하는 기능이 필요하며 이러한 기능을 식별자 결정(identifier resolution)이라고 함.

스코프의 종류

- 스코프의 종류

구분	전역 스코프	지역 스코프
변수의 의미	전역변수	지역변수
선언위치	코드의 가장 바깥부분	함수의 내부
유효영역	전역 영역과 그 하위 영역	함수 자신과 그 하위 영역
특징	Var와 같은 선언자를 생략시 전역스코프	ES5(var)는 함수 레벨 스코프 지원 ES6(let, const)는 블록 레벨 스코프 지원

** 블록레벨 스코프 : C++, JAVA언어 등에서 지원하는 스코프. if, for, try/catch등이 스코프영역으로 판단된다.

식별자 결정

- 식별자 결정
  - 식별자 결정은 가장 가까운 스코프를 우선으로 한다.

```
const sc = 'global';

function func()
{
    console.log(sc);
}

func();
```

결과 : global

```
const sc = 'global';

function func()
{
    const sc = 'local';
    console.log(sc);
}

func();
```

결과 : local

식별자 결정

- 함수 레벨 스코프
  - ES5의 var는 함수 레벨 스코프를 가진다. 즉 함수내에서 유효.

```
function func()
{
    for (var i = 0; i < 5; i++)
    {
        i;
    }
    console.log(i);
}
```

결과 : 5

  - ES6의 const나 let은 블록 레벨 스코프를 가진다. 즉 블록내에서만 유효

```
function func()
{
    for (let i = 0; i < 5; i++)
    {
        i;
    }
    console.log(i);
}
```

결과 : ERROR, i is not defined

식별자 결정

- 식별자 결정시 문맥해석
  - Javascript는 변수 선언 없이 변수를 사용 할 수 있다.

```
function func()
{
    sc1 = 'local';
    console.log(sc1);
}
```

sc1은 전역변수 var sc1으로 해석

  - 하지만 아래 코드는 sc1을 전역 변수인 sc1에 값을 대입한다.

```
let sc1 = 'global';

function func()
{
    sc1 = 'local';
    console.log(sc1);
}

func();
console.log(sc1);
```

식별자 결정

- 식별자 결정의 난해함
  - 아래 코드는 무한 루프를 발생
  - f1()에서의 i는 for문에서 선언한 i를 초기화.
  - f1에 i가 없기 때문에 스코프 체이닝으로 인하여 for문의 i를 참조.

```
function f1()
{
    i = 0;
}

for (var i = 0; i < 10; i++)
{
    f1();

}
```

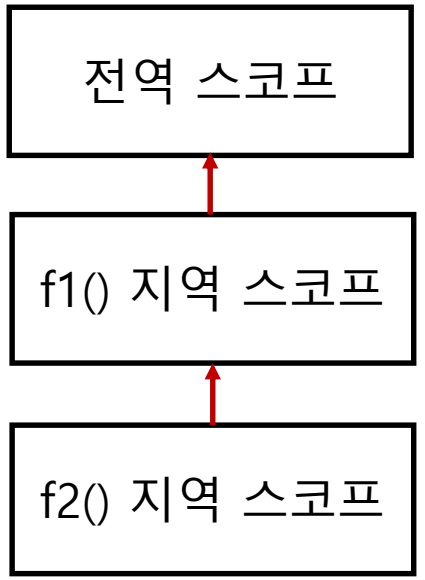
스코프 체인

- 스코프 체인의 개념
  - 함수의 중첩 구조로 인해 스코프도 중첩 구조가 생김
  - 식별자 결정을 위해 가까운 스코프부터 전역스코프까지 계층적으로 연결된 구조를 스코프 체인이라고 한다.

```
let a = 30;
const f1 = function()
{
    let a = 20;
    const f2 = function()
    {
        let a = 10;
        console.log(a);

    }

    f2();

}

f1();
```

```
let a = 30;
const f1 = function()
{
    let a = 20;
    const f2 = function()
    {

        console.log(a);

    }

    f2();

}

f1();
```

```
let a = 30;
const f1 = function()
{

    const f2 = function()
    {

        console.log(a);

    }

    f2();

}

f1();
```



클로저

- 외부 함수와 내부함수의 개념
  - 내부함수는 외부함수의 변수에 접근 가능 (스코프 체이닝)

```
const outfunc = function() {
    const value = 10;
    const infunc = function() {
        console.log(value);
    }

    infunc();

}

outfunc();
```

내부함수

외부함수

클로저

- 클로저의 개념
  - 내부 함수에서 외부함수로 스코프 체이닝을 통해 외부함수의 변수로 접근 가능한 함수의 조합

  - 원인 : 스코프 체이닝, 결과 : 클로저

클로저

- 클로저의 활용
  - title지역변수는 외부에서 접근할 수 없음
  - 클래스와 비슷한 구조로 활용

```
const makeInstance = function(title)
{
    return {
        getTitle : function()
        {
            return title;

        },

        setTitle : function(_title)
        {
            title = _title;

        }

    }

}
```

```
const book1 = makeInstance('title1');

console.log(book1.getTitle());
book1.setTitle('title2')
console.log(book1.getTitle());
```

중복선언

- var의 중복선언
  - var의 중복 선언시 var를 생략한 것과 같은 효과가 발생.
  - 프리미티브의 경우 메모리의 재할당이 이루어 짐.

```
var a = 10;
console.log(a);

var a = 20;
console.log(a);
```

  - const나 let은 이러한 중복 선언을 허용하지 않음.

전역변수의 사용

- 전역변수의 문제점
  - 일반적으로 자바스크립트는 외부라이브러리의 의존도가 높음.
  - 따라서 전역변수 사용시 이름이 겹치는 경우 오류발생

- 해결방안
  - 전역 객체를 이용
    - 필요한 모든 전역 변수들을 전역 객체의 속성으로 등록하여 사용
  - 즉시 호출함수를 이용

```
(function()
{
    const myGlobal = {

        value1 : 1,                     // 전역변수
        array1 : [11, 22, 33, 44, 55]   // 전역변수

    };

    const func = function()
    {
        return myGlobal.array1[0] + myGlobal.value1;
    }

    console.log(func());

})()
```