Chap13 스코프

13-1 스코프란?
13-2 스코프의 종류
13-3 스코프 체인
13-4 함수 레벨 스코프
13-5 렉시컬 스코프

13-1 스코프란?

▼ 스코프

식별자(변수 이름, 함수 이름, 클래스 이름 등)가 유효한 범위

자바스크립트 엔진이 식별자를 검색할 때 사용하는 규칙

▼ 식별자 결정

자바스크립트 엔진이 이름이 같은 변수가 있을 때 어떤 변수를 참조할 지 결정하는 것

13-2 스코프의 종류

▼ 전역 스코프

- 전역에서 선언된 변수(전역 변수)가 갖는 스코프
- 전역변수는 어디에서든 참조 가능

▼ 지역 스코프

- 지역에서 선언된 변수(지역변수)가 갖는 스코프
- 지역 변수는 지역 스코프와 하위 지역 스코프에서 유효하다.

13-3 스코프 체인

함수는 함수 몸체 내부에 또다른 함수를 정의할 수 있다. 이를 `함수의 중첩`이라하고 함수 몸체 내부에 정의한 함수를 `중첩 함수` 라고 하고 중첩 함수를 포함하는 함수를 `외부 함수` 라고 한다.

이때 함수가 중첩되면 스코프도 `계층적 구조` 를 갖는다. 이것을 **`스코프 체인`** 이라고 한다.

```
// 전역 함수
function foo() {
  console.log('global function foo');
}
//외부 함수
function bar() {
  // 중첩 함수
  function foo() {
    console.log('local function foo');
  }

  foo(); // ①
}

bar();
```

자바스크립트 엔진은 코드를 실행하기 전에 렉시컬 환경을 실제로 생성한다. 변수 선언이 실행되면 변수 식별자가 이 자료구조(렉시컬 환경)에 키로 등록되고, 변수 할당이 일어나면 이 자료구조의 변수 식별자에 해당하는 값을 변경한다.

▼ 렉시컬 환경

코드가 어디서 실행되며 어떤 코드가 있는지 코드의 문맥을 이루는 환경. 자세한 내용은 23자에서....

13-4 함수 레벨 스코프

▼ 함수 레벨 스코프와 블록 레벨 스코프

▼ 1. 함수 레벨 스코프

- 함수 내에서 선언된 변수는 함수 내에서만 유효하며 함수 외부에서는 참조할 수 없다.
- 즉, 함수 내부에서 선언한 변수는 지역 변수이며 함수 외부에서 선언한 변수는 모두 전역 변수이다.

▼ 2. 블록 레벨 스코프

- 모든 코드 블록(함수, if 문, for 문, while 문, try/catch 문 등) 내에서 선언된 변수는 코드 블록 내에서만 유효하며 코드 블록 외부에서는 참조할 수 없다.

- 즉, 코드 블록 내부에서 선언한 변수는 지역 변수이다.
- 예외) `var키워드` 로 선언한 변수가 있다면 이는 전역 변수이다.

▼ var let const

var는 함수 레벨 스코프에서만 선언되었을 때 지역 변수가 인정되고 이외에서는 모두 전역변수로 인정한다.

이를 보완하기 위해 ES6에서 도입된 let, cosnt를 이용하면 된다.

13-5 렉시컬 스코프

```
var x = 1;

function foo() {
  var x = 10;
  bar();
}

function bar() {
  console.log(x);
}

foo(); // ?
bar(); // ?
```

foo(); 를 호출하면 1이 나오고 bar();를 호출해도 1이 나온다. 왜일까?

자바스크립트는 렉시컬 스코프를 따르기 때문이다.

▼ 렉시컬 스코핑 vs 동적 스코핑

1. **렉시컬 스코핑** (Lexical Scoping):
   - 함수가 어디서 호출되는지와 상관없이, 함수가 정의된 위치에 따라 스코프가 결정됩니다.
   - 자바스크립트는 렉시컬 스코핑을 사용합니다.
2. **동적 스코핑** (Dynamic Scoping):
   - 함수가 어디서 호출되는지에 따라 스코프가 결정됩니다.
   - 함수 호출 시점의 실행 컨텍스트를 기반으로 변수의 값을 찾습니다.

예제 코드

다시 한 번 코드를 보겠습니다:

```
javascript코드 복사
var x = 1;

function foo() {
  var x = 10;
  bar();
}

function bar() {
  console.log(x);
}

foo(); // ?
bar(); // ?
```

렉시컬 스코핑에서의 결과

자바스크립트의 렉시컬 스코핑을 적용하면 결과는 다음과 같습니다:

- `foo()` 호출 시 `bar()`는 전역 스코프의 `x`를 참조하여 `1`을 출력합니다.
- 전역에서 `bar()`를 호출해도 전역 스코프의 `x`를 참조하여 `1`을 출력합니다.

즉,

```
javascript코드 복사
foo(); // 1
bar(); // 1
```

동적 스코핑에서의 결과

동적 스코핑을 가정하면 함수가 호출된 위치에 따라 스코프가 결정됩니다. 따라서, `bar` 함수가 호출될 때의 실행 컨텍스트를 사용합니다.

1. `foo()` **호출 시**:

- `foo` 함수가 실행되면 `var x = 10;`이 정의됩니다.
- `bar()`가 호출될 때, 현재 실행 컨텍스트는 `foo`입니다.
- 동적 스코핑에서는 `bar`가 `foo` 내부에서 호출되었으므로, `bar`는 `foo`의 스코프 체인에서 변수를 찾습니다.
- 따라서 `bar`는 `foo`의 `x`를 참조하여 `10`을 출력합니다.

2. **전역에서 `bar()` 호출 시**:
   - 전역에서 `bar()`를 호출하면, 현재 실행 컨텍스트는 전역 컨텍스트입니다.
   - 동적 스코핑에서는 `bar`가 전역에서 호출되었으므로, 전역 스코프에서 변수를 찾습니다.
   - 따라서 `bar`는 전역 변수 `x`를 참조하여 `1`을 출력합니다.