**JavaScript var 변수 Hoisting에 대하여**

소프트웨어 공학 01분반

1434859 이선환

**1. var 에 대한 개념정리**

JavaScript의 var 변수는 다른 언어들과 조금 다르게 동작합니다. 이는 때때로 개발자로 하여금 의도하지 않은 결과를 발생하기도 합니다.

if(true){

*var* name = 'sean';

}

*console*.log(name);

for(*var* i=0; i<5; i++){

// do something

}

*console*.log(i);

일반적인 프로그래밍 언어에서 변수는 블록 스코프 { } 안에서 유효하기 때문에, 블록이 종료된

시점에서 console.log를 호출하면 정상적으로 동작하지 않을 것이라 예상됩니다.

하지만 입력한 소스코드의 결과로 **‘sean’**과 **‘5’**가 콘솔에 정상적으로 출력이 됩니다.

이러한 현상이 발생하는 이유가 바로 var 변수가 **Hoisting** 되기 때문입니다.

**- Hoisting**이란?

Hoisting은 var을 통해 정의된 변수의 선언문을 **유효 범위의 최상단**으로 끌어올리는 행위를 말합니다. ‘선언과 할당의 분리’라고 생각할 수도 있습니다.

if(true){

*var* name = 'sean';

}

*console*.log(name);

위 코드는 Hoisting에 의해 아래와 같은 모양으로 바뀌게 됩니다.

*var* name;

if(true){

name = 'sean';

}

*console*.log(name);

var name = ‘sean’이라고 정의한 변수가 실제로는 선언(var name)과 할당(name = ‘sean’)로 분리되는 것입니다. 이러한 이유로 if 구문 밖에서 name 변수가 사용 가능하고, for 구문 밖에서 I 변수가 사용 가능한 현상이 발생하는 것입니다.

**-** 그렇다면 **Hoisting** 시 **유효 범위** 란?

자바스크립트의 var는 단순한 블록{ }이 아닌 함수 블록 function{ } 안에서 유효합니다.

이를 var는 **“함수 스코프에서 유효하다”** 라고 합니다.

*function* ho1(){

if(true){

*var* name = 'sean';

}

*console*.log(name);

}

*function* ho2(){

for(*var* i=0; i<5; i++){

// do something

}

*console*.log(i);

}

if(true){

*var* score = 100;

}

*console*.log(score);

위 코드는 Hoisting에 의해 내부에서 아래와 같은 모양으로 바뀌게 됩니다.

*var* score;

*function* ho1(){

*var* name;

if(true){

name = 'sean';

}

*console*.log(name);

}

*function* ho2(){

*var* i;

for(i=0; i<5; i++){

// do something

}

*console*.log(i);

}

if(true){

score = 100;

}

*console*.log(score);

var는 함수 스코프에서 유효하기 때문에 Hoisting 시, 선언문은 글로벌 스코프가 아닌 유효 범위(function) 내부의 최상단에 위치하게 됩니다.

글로벌 스코프 또한 하나의 함수 스코프처럼 동작하기 때문에 Hoisting이 일어납니다.

**- 함수의 Hoisting**?

함수의 선언 역시 Hoisting의 대상입니다. 때문에 스코프 내에서 어떤 위치에서 함수 선언을 하든지 호출할 수 있습니다.

sayName();

*function* sayName(){

*console*.log(‘sean’);

}

위 코드는 함수 hoisting으로 인해 아래와 같은 모양으로 변하게 됩니다.

*function* sayName(){

*console*.log(‘sean’);

}

sayName();

함수 선언 역시 최상단으로 끌어올려지기 때문에 sayName()을 먼저 호출하고 함수 정의를 이후에 하여도 정상적으로 동작하게 됩니다.

**2. var 에 대한 문제점**

학부에서 C, C++, Java, Python 등의 프로그래밍 언어를 배우다가 JavaScript를 접하니,

var 변수가 Function scope 를 갖는 것에 대해 우려가 생깁니다. var 변수의 우려되는 점에서 발생하는 문제점을 정리하겠습니다.

**1) var로 변수 선언 시 var 키워드 생략 가능**

**chicken = 'nice';**

***console*.log(chicken); // nice**

var 키워드로 시작하는 선언문을 작성하지 않았으나 선언하여 사용이 가능합니다.

이는 직관적인 코드 리뷰가 어려워지는 문제를 유발합니다.

**2) 변수 선언 전 사용**

변수 선언 전에 변수를 사용하여 변수가 undefined 값을 갖는 경우입니다. 위에서 언급한 것처럼 JavaScript에서는 **Hoisting** 을 해서 변수나 함수 선언이 인터프리터에 의해 최상위로 끌어올려집니다.

for (var i = 0; i < rows.length; i++) {

var obj = {

patternId: patternId,

sortOrder: idx++

};

rowOrderArr.push(obj);

}

var patternId = $("#patternNameLabel").attr("patternId");

$.post('/delete.do', { order: JSON.stringify(rowOrderArr) });

위 코드에서 obj 객체의 patternId 속성은 항상 undefined 값으로 설정됩니다.

하지만 **에러는 발생하지 않기 때문에** 실제로 돌려보는 시점에서야 문제를 알 수 있게 됩니다.

가령 rowOrderArr 객체로 서버에 AJAX 호출을 했을 때에야 알 수 있습니다.

(클라이언트에서 patternId 값을 안 채워줘서 서버 에러가 납니다.)

**3) 변수 중복 선언**

함수 내에 중복된 변수 선언이 존재하는 경우입니다.

개발자들이 변수의 범위를 일반적으로 생각하는 범위인 블럭 범위(block scope)로 생각하고

아래와 같이 pjtCode 변수가 각 블럭에 한정되게 작성을 한다고 가정해보겠습니다.

var idx = nextUrl.toUpperCase().indexOf('&PJTCODE=');

if (idx > -1) {

var pjtCode = nextUrl.substr(idx + 9, 15);

if (idx > 9) {

var pjtCode = 'BAD\_CODE';

}

console.log(pjtCode);

}

console.log(pjtCode);

JavaScript는 변수가 함수 범위(function scope)를 가지고 Hoisting에 의해 선언이 끌어올려지므로, 실제로 pjtCode 변수는 함수 내에서 하나로 유지됩니다.

즉, 두 개의 콘솔 출력문 모두 BAD\_CODE로 출력됩니다.(idx가 9보다 클 경우)

이와 같이 Hoisting은 JavaScript 코드 해석을 비직관적으로 만드는 측면이 있습니다.

**4) 변수 재정의**

변수를 재정의함으로써 이전에 정의한 변수가 사용되지 않는 경우입니다.

if ($.browser.msie == true) {

target = url + "userName" + userName; // 1)

target = url.replace(/\.|\?|\&|\/|\=|\:|\-|\s/gi,""); // 2)

}

위 코드에서 1)에 정의된 target 변수가 2)에서 재정의되면서 1)에서 할당한 username 파라미터가 무시됩니다. 따라서 의도한 대로 결과를 얻으려면 다음과 같이 수정되어야 합니다.

if ($.browser.msie == true) {

target = url + "userName" + userName;

target = target.replace(/\.|\?|\&|\/|\=|\:|\-|\s/gi,"");

}

**3. 문제점에 대한 해결방안**

ES6(ECMAScript 6th edition, 매 년 발표하는 자바스크립트 표준) 이전에는 변수를 선언하는 방법이 var 를 이용하는 방법 밖에 없었습니다. 하지만 위에서 언급한 것과 같은 여러가지 문제들이 발생했고, ES6에서 추가된 **let, const** 사용함으로써 문제를 해결할 수 있게 되었습니다.

**1) block-level-scope variable(let, const)**

Var 에서는 function-level-scope 라서 전역 변수가 남발하는 문제들이 많았으나, let, const 는 block-level-scope 라서 이러한 문제를 해결할 수 있습니다.

***let* chicken = 'nice'; // 전역 변수**

**{**

**// ReferenceError: chicken is not defined**

***console*.log(chicken);**

***let* chicken = 'nice';**

***console*.log(chicken); // good**

**}**

**2) 변수 중복 선언 불가**

Var 에서는 변수를 위에 선언 후 다시 아래에서 재선언이 가능했습니다. 하지만 let, const 는 재선언 시 문법 에러가 뜹니다.

***let* chicken = 'nice'; // 전역 변수**

**{**

**// Identifier ‘chicken’ has already been declared**

***let* chicken = 'good';**

**}**

**3) 호이스팅 불가**

위에서 설명했듯이 var로 변수 생성 시 선언과 초기화가 동시에 이루어지기 때문에 Hoisting 현상이 나타났습니다. 하지만 let, const 는 변수 생성 시 선언과 초기화가 분리되어서 진행이 됩니다. let, const 는 변수를 생성하고 선언까지 한 후 **일시적 사각지대**가 선언과 초기화 사이에 생기며, 할당문에서 변수 값이 할당 될 때 초기화 및 값 할당이 이루어집니다.

**// error: chicken is not defined**

***Console.log(*chicken);**

***let* chicken = 'nice';**

**정리하자면, ES6 를 사용 가능하다면 var 대신 let, const를 사용하면 문제들을 해결할 수 있습니다.**

let 으로 변수 선언 후 값을 재할당 가능하지만, const 로 선언 후 값을 재할당 할 수 없습니다. 값을 재할당 할 것 같으면 let을 사용하고 변하지 않는 값에는 const(상수)를 사용하면 됩니다.

**4. 참고 사이트**

- 자바스크립트 호이스팅(Hoisting)

(<https://yuddomack.tistory.com/entry/%EC%9E%90%EB%B0%94%EC%8A%A4%ED%81%AC%EB%A6%BD%ED%8A%B8-%ED%98%B8%EC%9D%B4%EC%8A%A4%ED%8C%85Hoisting>)

- var를 사용할 때 발생하는 문제들

(<https://www.daleseo.com/js-var-issues/>)

- JavaScript에서 변수 사용 시 주의할 점

(<https://cimfalab.github.io/deepscan/2016/07/variable-misuse>)

- JavaScript let, const란? 그리고 왜 써야만 하는가? (ES6)

<(https://happycording.tistory.com/entry/let-const-%EB%9E%80-%EC%99%9C-%EC%8D%A8%EC%95%BC%EB%A7%8C-%ED%95%98%EB%8A%94%EA%B0%80-ES6>)