UI/UX 전문가 과정비트캠프

2021년 01월 28일 22회 차

장 해 솔

이메일 : wkdgothf@gmail.com

github : https://github.com/legossol

JavaScript

목차

1. VAR 과 LET

1)VAR이란?

VAR은 글로벌한 변수같은 존재이다. 끝까지 메모리 안에 남아있으며 어느곳에 있건 변수명의 값을 변화시킬 수 있다.

만약 여러개의 값이 존재 하는 변수명이라면 가장 마지막에 선언된 값이 최종 값이 된다.

확인을 위해 아래와 같이 코드를 짜보았다.

결과는 DOG 두개가 출력된다. 하지만 CONSOLE을 확인해보면 위의 LILY는 정의되지 않았다고 뜬다.

즉, 최상단에 있는 콘솔 로그는 바로 아래있는 LILY = "CAT"중에서 변수명 LILY만 가져온 것이고 변수의 값을가져온 것은 아니다.(HOISTING)

그리고 아래로 내려가서 VAR LILY = "DOG"가 선언되고 나서야 VAR이 선언안된 CAT을 가져오는게 아닌 VAR로 선언된 LILY의 DOG라는 값을 가져온 것이다.

그럼 여기서 아래의 AnnouncementValuecat은 뭔가

그걸 확인하기위해 아래와 같이 만들어보면

```
function AnnouncementValue() {

    lily = "cat"
    console.log("AnnouncementValue" +lily)
    lily = "spider"

    console.log("AnnouncementValue" +lily)
    var lily = "dog"
```

AnnouncementValuecat

AnnouncementValuespider

?

```
function AnnouncementValue() {
    lily = "meandyou"
    lily = "me"
    lily = "spider"
    console.log("AnnouncementValue" +lily)
    lily = "cat"

    lily = "you"

    console.log("AnnouncementValue" +lily)
    var lily = "dog"
```

```
AnnouncementValuespider
AnnouncementValueyou
```

???????..● 모르겠다 강사님

질문입니다.
HOISKING은 아래서 가져오는 것인데 만약 이렇게 되면 어떤 변수를 가져온 것인가요...?

2)LET

LET은 변수를 선언하는 유일한 키워드 이다. 만약 {}중괄호 안에 들어있는 경우 {}중괄호가 끝나면 메모리가 해제되기 때문에 메모리 낭비가 발생하지 않는다.

만약 코드가 너무 많아 LET을 여러 개 생성했을 경우 다음과 같이 된다.

CASE1.

```
function AnnouncementValue() {
    let i = 0;
    var lily = "dog"
    let lily1 = "baram's nera"
    console.log("AnnouncementValue" +lily)
     lily = "cat"
    if(i = 0){
        let lily1 = "car"
    console.log("AnnouncementValue" +lily1)
    }
    var lily = "spider"
```

dog, Baram's nara

```
function AnnouncementValue() {
    let lily = "dog"
    let i = 0
    if(i ==0){
        let a = 3;
        console.log("AnnouncementValue" +a)
    }
    let a = 10;
    console.log("AnnouncementValue" +lily)

    if(i == 0){
        let a = 5;
    console.log("AnnouncementValue" +a)
}
```

```
dog, 10,10
```

위에서 알 수 있듯이 {}중괄호에 없는 곳에서 먼저 선언된 LET의 값이 최종 값이 된다.

만약 아래와 같이 똑같은 블록에 변수의 값을 다르게 주게 되면 오류가 발생한다.

CASE2.

```
function AnnouncementValue() {

    let lily = "dog"

    console.log("AnnouncementValue" +lily)
    let lily = "cat"

    console.log("AnnouncementValue" +lily)
```

CASE3.

마지막으로 만약 다음과 같이 {}중괄호 안에 처음 LET을 선언했을경우 밖에서 그 변수 값을 받고 싶어하면 에러가 발생한다.

```
function AnnouncementValue() {
    let lily = "dog"
    let i = 0

    console.log("AnnouncementValue" +lily)

    if(i == 0){
        let a = 5;
    console.log("AnnouncementValue" +a)
}
console.log("AnnouncementValue" +a)
```

3)VAR과 LET의 차이점

```
var d = 0;
{
  var d = 10;
  console.log(d);
}
console.log(d);

10
10
```

```
let d = 0;
{
  let d = 10;
  console.log(d);
}
console.log(d);

10
0
```

```
let d = 0;
{
  var d = 10;
  console.log(d);
}
console.log(d);

SyntaxError: Identifier
'd' has already been
declared
```

```
var d = 0;
{
  let d = 10;
  console.log(d);
}
console.log(d);

10
0
```

2.CONST(상수)

CONST의 특징은 한번 저장되면 바뀌지 않는다는 점이다. 그렇기때문에 다른사람이 실수로 값을 바꾸는 것을 방지할 수 있으며 여러 쓰레드가 동시에 접근해도 안정성을 보장할 수 있다.

3.NULL과 UNDEFINED의 차이

NULL은 값이 없다고 명시적으로 저장한 것.

UNDEFINED는 변수만 선언하고 값을 저장하지 않은 것.

4.SYMBOL

심볼은 정보의 은닉화를 위해서 사용한다.(뚫을 수 있는 방법은 있음)

```
function getObject() {
    const world = Symbol();

    return {
        hello: 10,
        [world]: 10,
    };
}

const a = getObject();
// a.hello === 10
a.hello = 20;
// a.hello is now 20.
// a.... key?
```

```
// --- counter.js ---
const count = Symbol();

export default class Counter {
    [count] = 0;

    add() {
        this[count] += 1;
        return this;
    }

    get() { return this[count]; }
}

// --- index.js ---
import Counter from "./counter";

const counter = new Counter();
console.log(counter.get()); // 0

counter.add().add().add();
console.log(counter.get()); // 3
```

왼쪽은 SYMBOL에 접근할 수 없다는 것을 알려주는 사진이고

오른쪽은 접근할 수 없기에 아무나 COUNT를 건들지 못하는 것을 보여주는 사진이다.
많은 시간을 소요했지만 내가 추려낸 정보를 꼽자면
1.정보의 은닉화를 위해 사용한다.
2.필드 내부에 문자열을 그냥 선언하는 것보다 다른 프로그래머가 코드를 보았을 때 PRIVATE FILED라는 것을 직관적으로 알 수 있다.(큰장점)
3.암묵적 형변환이 불가하다
예를들어 'A' + 1의 식이라면 1이 암묵적으로 문자열이 되어{A1}로 출력되고
TRUE+1의 식이라면 TRUE가 1이니까 암묵적으로 1+1로 변형해서 2가되는 것같은 것이 불가하다는 것이다
4.SYMBOL( ) 에서 소괄호에는 아무것도 안써도 상관없고 써도 상관없기에 보통은 주석같이 사용된다고 한다.

이정도만 알고 코드를 짜보자.

```
function TestSymbolTest() {
    let 마나물약 = Symbol(liquid)
    let 생명수 = Symbol(liquid)

    let 현철중검 = Symbol("sword")
    let 도깨비방망이 = Symbol("sword")

    마나물약 === 생명수 ? //false
    마나물약 ==  생명수 ? //false
    현철중검 === 도깨비방망이 ? //false
    현철중검 ==  도깨비방망이 ? //false

    //그럼 만약
    let 마나물약 = 동충하초
    마나물약 === 동충하초//true
```

중간부터 다음으로 중요한 내용이 등장한다. 같은 이름을 가진 심볼에 있다고 하더라도 그안에 속한 것의 데이터만도 타입포함해서도 모두 다르다. 마지막에 억지로 같게 만들고 싶다면 만들 순 있다.
다음으로 간단하게 아래와 같이 심볼이 어떻게 은닉성을 가지고 있는지 확인 할 수 있다.

```
const x = () =>{
    const a
    return {
        [a] : 10
    }
}
//true
const y = x();//y는 x()를 실행한 결과를 담아라
              //즉 리턴된 [a]:10이 담길 것.
//Error
y[Symbol(a)];같은 의미지만 오류가 뜨게됨
y.a//Error
y['a']//Error
```

만약 조금 바꿔서

```
//만약 위의 예제를 아래와같이 바꾸면
const x = () =>{
    const a = symbol('a');
    return {
        [a] : 10
        a: a
    }
}
//y를 출력시 a:Symbol(a)가 있게됨
y.a; //Symbol(a)
y[y.a] //10
```

이와 같이 RETURN{}안에 `a: a` **가 있다면 정보를 가져올 수 있다.**

```
[a] : 10//값을 쓸수 있고 노출은 되어있지만
a: a    //public한 상태로 접근할 수 있는 권한을 주느냐마느냐에 따라 다름
```

아까 말했던 억지로 정보를 뭔지 보고 싶어서 보고싶다고 한다면

```
const b = Reflect.ownKeys(y)//이렇게하면 접근가능
y[b(0)]//10                  //하지만 내부에 몇번째에
                             //뭐가 있는지 모르면 사용해봤자 의미가 없음
                             //지금은 10하나라 바로 알 수 있음
```

이런 방법이 있다고는 한다(방법이 더 있음 한, 두가지)

아래는 심볼을 활용한 예시.

```
const NAME = Symbol('이름')
const GENDER = Symbol('성별')
const ssol = {
    [NAME]: '장해솔'
    [GENDER]: 'male'
    age:21
}
const jinwook = {
    [NAME]: '오진욱'
    [GENDER]: 'male'
    age:21
}
console.log(ssol,jinwook)
//{age: 21, Symbol(이름):"장해솔", Symbol(성별):"male"}
//{age: 21, Symbol(이름):"오진욱", Symbol(성별):"male"}
```