8253jang@daum.net
장희정

ES6(ECMAScript 6) and JS

✓ ES5

ES5에서 변수를 선언할 수 있는 유일한 방법은 var 키워드를 사용하는 것.

- var 키워드로 선언된 변수의 특징

1) (Function-level scope)

: 전역 변수함수 레벨 스코프의 남발

: for loop 초기화식에서 사용한 변수를 for loop 외부 또는 전역에서 참조할 수 있다.

2) var 키워드 생략 허용

: 의도하지 않은 변수의 전역화

3) 중복 선언 허용

: 의도하지 않은 변수 값 변경

4) 변수 호이스팅

: 변수를 선언하기 전에 참조가 가능하다

✓ ES5

대부분의 문제는 전역 변수로 인해 발생한다. 전역 변수는 간단한 애플리케이션의 경우, 사용이 편리하다는 장점이 있지만 불가피한 상황을 제외하고 사용을 억제해야 한다. 전역 변수는 유효 범위(scope)가 넓어서 어디에서 어떻게 사용될 것인지 파악하기 힘들며, 비순수 함수(Impure function)에 의해 의도하지 않게 변경될 수도 있어서 복잡성을 증가시키는 원인이 된다. 따라서 변수의 유효 범위(scope)는 좁을수록 좋다.

ES6는 이러한 var의 단점을 보완하기 위해 **let과 const 키워드를 도입**하였다.

✓ ES6의 let

블록 레벨 스코프

대부분의 C-family 언어는 블록 레벨 스코프(Block-level scope)를 지원하지만 자바스크립트는 함수 레벨 스코프(Function-level scope)를 갖는다.

✓ ES6의 let

함수 레벨 스코프(Function-level scope)

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.

블록 레벨 스코프(Block-level scope)

코드 블록 내에서 선언된 변수는 코드 블록 내에서만 유효하며 코드 블록 외부에서는 참조할 수 없다.

✓ ES6의 let

블록 레벨 스코프를 지원하지 않는 var 키워드의 특성상, 코드 블록 내의 변수 foo는 전역변수이다.

그런데 이미 전역변수 foo가 선언되어 있다.

var 키워드를 사용하여 선언한 변수는 중복 선언이 허용되므로 위의 코드는 문법적으로 아무런 문제가 없다.

단, 코드 블록 내의 변수 foo는 전역변수이기 때문에 전역에서 선언된 전역변수 foo의 값 123을 대체하는 새로운 값 456을 재할당한다.

```
console.log(foo); // undefined
var foo = 123;
console.log(foo); // 123
{
  var foo = 456;
}
console.log(foo); // 456
```

✓ ES6의 let

ES6는 **블록 레벨 스코프**를 갖는 변수를 선언하기 위해 let 키워드를 제공한다.

let 키워드로 선언된 변수는 블록 레벨 스코프를 갖는다. 앞 예제에서 코드 블록 내에 선언된 변수 foo는 블록 레벨 스코프를 갖는 지역 변수이다 . 전역에서 선언된 변수 foo와는 다른 변수이다.

또한 변수 bar도 블록 레벨 스코프를 갖는 지역 변수이다. 따라서 전역에서는 변수 bar를 참조할 수 없다.

```
let foo = 123;
{
  let foo = 456;
  let bar = 456;
}
console.log(foo); // 123
console.log(bar); // ReferenceError: bar is not defined
```

✓ ES6의 let

var 키워드로는 이름이 같은 변수를 중복해서 선언할 수 있었지만, let 키워드로는 이름이 같은 변수를 중복해서 선언하면 문법 에러(SyntaxError)가 발생한다.

```
var foo = 123;
var foo = 456;  // 중복 선언 허용

let bar = 123;
let bar = 456;  // Uncaught SyntaxError: Identifier 'bar' has already been declared
```

✓ 호이스팅(Hoisting)

자바스크립트는 ES6에서 도입된 let, const를 포함하여 모든 선언(var, let, const, function, function*, class)을 호이스팅 한다.

호이스팅(Hoisting)이란, var 선언문이나 function 선언문 등을 해당 스코프의 선두로 옮긴 것처럼 동작하는 특성을 말한다.

하지만 var 키워드로 선언된 변수와는 달리 let 키워드로 선언된 변수를 선언문 이전에 참조하면 참조 에러(ReferenceError)가 발생한다. 이는 let 키워드로 선언된 변수는 스코프의 시작에서 변수의 선언까지 **일시적 사각지대(Temporal Dead Zone; TDZ)**에 빠지기 때문이다.

✓ 호이스팅(Hoisting)

```
console.log(foo); // undefined
var foo;

console.log(bar); // Error: Uncaught ReferenceError: bar is not defined
let bar;
```

```
let foo = 1; // 전역 변수

{
  console.log(foo); // ReferenceError: foo is not defined
  let foo = 2; // 지역 변수
}
```

✓ 전역객체와 let

전역 객체(Global Object)는 모든 객체의 유일한 최상위 객체를 의미하며 일반적으로 Browser-side 에서는 window 객체

var 키워드로 선언된 변수를 전역 변수로 사용하면 전역 객체의 프로퍼티가 된다.

```
var foo = 123; // 전역변수

console.log(window.foo); // 123
```

✓ 전역객체와 let

let 키워드로 선언된 변수를 전역 변수로 사용하는 경우, let 전역 변수는 전역 객체의 프로퍼티가 아니다. 즉, window.foo와 같이 접근할 수 없다. let 전역 변수는 보이지 않는 개념적인 블록 내에 존재하게 된다.

```
let foo = 123; // 전역변수

console.log(window.foo); // undefined
```

✓ const

const는 상수(변하지 않는 값)

let은 재 할당 가능하나 const 재할당 금지

```
const FOO = 123;

FOO = 456; // TypeError: Assignment to constant variable.
```

✓ const

주의할 점은 **const는 반드시 선언과 동시에 할당이 이루어져야 한다**는 것이다.

그렇지 않으면 다음처럼 문법 에러(SyntaxError)가 발생한다.

```
const FOO; // SyntaxError: Missing initializer in const declaration
```

✓ const

const는 let과 마찬가지로 블록 레벨 스코프를 갖는다

```
{
  const FOO = 10;
  console.log(FOO); //10
}
console.log(FOO); // ReferenceError: FOO is not defined
```

✓ const

상수는 가독성과 유지보수의 편의를 위해 적극적으로 사용해야 한다.

```
// 10의 의미를 알기 어렵기 때문에 가독성이 좋지 않다.
if (rows > 10) {
}

// 값의 의미를 명확히 기술하여 가독성이 향상되었다.
const MAXROWS = 10;
if (rows > MAXROWS) {
}
```