7장 연산자

연산자(operator)는 하나 이상의 표현식을 대상으로 산술, 할당, 비교, 논리, 타입, 지수 연산등을 수행해 하나의 값을 만든다.

연산의 대상을 피연산자(operand)라고 한다. 또한 피연산자와 연산자의 조합으로 이뤄진 연산자 표현식도 값으로 평가될 수 있는 표현식이다.

```
5 * 4 // 산술 연산자

'My name is ' + 'Lee' // 문자열 연결 연산자

3 > 5 // 비교 연산자
```

피연산자가 '값'이라는 명사의 역할을 한다면, 연산자는 '피연산자를 연산하여 새로운 값을 만든다'라는 동사의 역할을 한다.

산술 연산자

산술 연산자는 피연산자를 대상으로 수학적 계산을 수행해 새로운 값을 만든다.

산술 연산이 불가능한 경우, NaN을 반환한다. (Not a Number)

또한 산술 연산자는 **이항 산술 연산자**와 **단항 산술 연산자**로 분류된다.

이항 산술 연산자

이항 산술 연산자는 **2개의 피연산자**를 산술하여 숫자 값을 만든다.

모든 이항 산술 연산자는 피연산자의 값을 변경하는 부수 효과(side effect)가 없다.

이항 산술 연산자	의미	부수 효과
+	덧셈	×
-	뺄셈	×
*	곱셈	×
/	나눗셈	×
%	나머지	×

표 7-1 이항 산술 연산자

어떤 산술 연산을 해도 피연산자의 값이 바뀌는 경우는 없고 언제나 새로운 값을 만들 뿐이다.

단항 산술 연산자

단항(unary) 산술 연산자는 1개의 피연산자를 산술 연산하여 숫자 값을 만든다.

단항 산술 연산자	의미	부수 효과
++	증가	○
--	감소	○
+	어떠한 효과도 없다. 음수를 양수로 반전하지도 않는다.	×
-	양수를 음수로, 음수를 양수로 반전한 값을 반환한다.	×

표 7-2 단항 산술 연산자

주의할 점은 이항 산술 연산자와는 달리 증가/감소(++/—) 연산자는 피연산자의 값을 변경하는 부수 효과가 있다는 것이다.

```
var x = 1;

x++; // x = x + 1;
```

증가/감소 연산자는 위치에 의미가 있다.

- 전위 증가/감소 연산자는 먼저 피연산자의 값을 증가/감소 시킨 후, 다른 연산을 수행한다.
- 후위 증가/감소 연산자는 먼저 다른 연산을 수행한 후, 피연산자의 값을 증가/감소 시킨다.

```
var x = 5, result;

result = x++; // 선 처리, 후 증가
console.log(result); // 5

result = ++x; // 선 증가, 후 처리
console.log(result); // 7
```

단항 연산자 +

단항 연산자 `+`는 피연산자에 어떠한 효과도 없다. 음수를 양수로 반전하지도 않는다.

숫자 타입이 아닌 피연산자에 `+`단항 연산자를 사용하면 피연산자를 숫자 타입으로 변환하여 반환한다.

이때 피연산자를 변경하는 것은 아니고 숫자 타입으로 변환한 값을 생성해서 반환한다.

```
var x = '1';

console.log(+x); // 1

console.log(x); // "1"

x = true;
console.log(+x); // 1, true를 숫자로 변환하면 1이다.
```

```
x = "Hello";
console.log(+x); // NaN
```

단항 연산자 -

단항 연산자 `-` 는 피연산자의 부호를 반전한 값을 반환한다.

단항 연산자 `+` 의 기능과 다른 점은 숫자 타입이 아닌 값은 숫자 타입으로 바꿔 반환하지만, 부호를 반대로 바꿔 반환한다는 점이다.

문자열 연결 연산자

연산자 `+` 는 피연산자 중 하나 이상이 문자열인 경우 문자열 연결 연산자로 동작한다. (자바와 동일)

```
'1'+ 2; // '12' , 문자열 연결 연산

1 + false; // 1

1 + true; // 2

+undefined; // NaN
1 + undefined; // NaN
```

위 코드에서 `1+true` 를 연산하면 자바스크립트 엔진은 암묵적으로 불리언 타입의 값은 true를 숫자 타입인 1로 타입을 강제로 변환한 후 연산을 수행한다.

이를 암묵적 타입 변환(implicit coercion) 또는 타입 강제 변환(type coercion)이라고 한다.

할당 연산자

할당 연산자는 우항에 있는 피연산자의 평가 결과를 좌항에 있는 변수에 할당한다.

할당 연산자는 좌항의 변수에 값을 할당하므로 변수 값이 변하는 부수 효과가 있다.

할당 연산자	예	동일 표현	부수 효과
=	x = 5	x = 5	○
+=	x += 5	x = x + 5	○
-=	x -= 5	x = x - 5	○
*=	x *= 5	x = x * 5	○
/=	x /= 5	x = x / 5	○
%=	x %= 5	x = x % 5	○

표 7-3 할당 연산자

표현식은 값으로 평가될 수 있는 문이고, 문에는 표현식인 문과 표현식이 아닌 문이 있다고 했다.

그렇다면 할당문은 표현식인 문일까, 표현식이 아닌 문일까?

```
var x;
console.log(x=10); // 할당문은 표현식인 문이다.
```

할당문은 값으로 평가되는 표현식인 문으로서 할당된 값으로 평가된다.

```
var a,b,c;

a = b = c = 0;
// c = 0 : 이 할당문 자체는 할당값 자체로 평가된다.(0)
```

```
// b = 0 : 위와 동일
// a = 0 : 위와 동일
```

비교 연산자

비교 연산자는 좌항과 우항의 피연산자를 비교한 다음 그 결과를 불리언 값으로 반환한다.

동등/일치 비교 연산자

동등 비교(loose equality) 연산자와 일치 비교 (strict equality) 연산자는 좌항과 우항의 피연산자가 같은 값으로 평가되는지 비교해 불리언 값을 반환한다. 하지만 비교하는 엄격성의 정도가 다르다.

비교 연산자	의미	사례	설명	부수 효과
==	동등 비교	x == y	x와 y의 값이 같음	X
===	일치 비교	x === y	x와 y의 값과 타입이 같음	X
!=	부동등 비교	x != y	x와 y의 값이 다름	X
!==	불일치 비교	x !== y	x와 y의 값과 타입이 다름	X

표 7-4 비교 연산자

동등 비교(==) 연산자는 좌항과 우항의 피연산자를 비교할 때 먼저 암묵적 타입 변환을 통해 타입을 일치시킨 후 같은 값인지 비교한다.

```
5 == 5; // true

5 == '5'; // true
```

동등 비교 연산자는 편리한 경우도 있지만, 결과를 예측하기 어렵고 실수하기 쉽다.

```
'0' ==  ; // false
0 == ''; // true
0 == '0'; // true
false == 'false'; // false
false == '0';
false == null;
false == undefined; // false
```

이처럼 동등 비교(==) 연산자는 예측하기 어려운 결과를 만들어낸다. 따라서 동등 비교 연산자는 사용하지 않는 편이 좋다. 대신 일치 비교(===) 연산자를 사용한다.

일치 비교 (===) 연산자는 좌항과 우항의 피연산자 타입도 같고 값도 같은 경우에 한하여 true를 반환한다.

```
5 === 5; // true

5 === '5'; // false

NaN === NaN; // false
```

일치 비교 연산자에서 주의할 것은 NaN이다.

NaN은 자신과 일치하지 않는 유일한 값이다. 따라서 숫자가 NaN인지 조사하려면 빌트인 함수 `Number.isNaN` 을 사용해야 한다.

```
Number.isNaN(NaN); // true
Number.isNaN(10); // false
Number.isNaN(1+undefined); // true
```

숫자 0도 주의하자. 자바스크립트에는 양의 0과 음의 0이 있는데 이들을 비교하면 true를 반환한다.

```
0 === -0;
0 == -0;
```

Object.is 메소드

동등 비교 연산자(==)와 일치 비교 연산자(===)는 +0과 -0을 동일하다고 평가한다.

또한 동일한 값인 NaN과 NaN을 비교하면 다른 값이라고 평가한다.

ES6에서 도입된 Object.is 메소드는 다음과 같이 예측 가능한 정확한 비교 결과를 반환한다.

그 외에는 일치 비교 연산자(===)와 동일하게 동작한다.

```
-0 === +0 // true
Object.is(-0, +0); // false

NaN === NaN; // false
Object.is(NaN,NaN); // true
```

부동등 비교 연산자(`!=`)와 불일치 비교 연산자 (`!==`)는 각각 동등 비교 연산자와 일치 비교 연산자의 반대 개념이다.

대소 관계 비교 연산자

피연산자의 크기를 비교하여 불리언 값을 반환함

대소 관계 비교 연산자	예제	설명	부수 효과
>	x > y	x가 y보다 크다	X
<	x < y	x가 y보다 작다	X
>=	x >= y	x가 y보다 크거나 같다	X
<=	x <= y	x가 y보다 작거나 같다	X

표 7-5 대소 관계 비교 연산자

삼항 연산자

if-else 구문을 한 줄로 처리하는 연산자라고 생각하면 된다.

조건식의 평가 결과에 따라 반환할 값을 결정한다. 자바스크립트의 유일한 삼항 연산자이며, 부수 효과는 없다.

`조건식 ? 조건식이 true일 때 반환할 값 : 조건식이 false일 때 반환할 값`

```
var x = 2;

var result = x%2 ? "홀수" : "짝수"; // x%2가 짝수이면 나머지가 0이므로

console.log(result); // 짝수
```

삼항 연산자는 if-else 문과 비슷하지만 차이가 있다.

삼항 연산자는 값 처럼 사용할 수 있지만, if-else 구문은 값처럼 사용할 수 없다.

if-else는 표현식이 아닌 문이다. 따라서 if-else 문은 값처럼 사용할 수 없다.

하지만 삼항 연산자 표현식은 값으로 평가할 수 있는 표현식인 문이다.

조건에 따라 어떤 값을 결정해야 한다면 if-else 문보다 삼항 연산자 표현식을 사용하는 편이 유리하다.

하지만 조건에 따라 수행해야 할 문이 하나가 아니라 여러 개 라면 if-else 문의 가독성이 더 좋다.

논리 연산자

논리 연산자는 우항과 좌항의 피연산자(부정 논리 연산자의 경우 우항의 피연산자)를 논리 연산한다.

논리 연산자	의미	부수 효과
||	논리합(OR)	X
&&	논리곱(AND)	X
!	부정(NOT)	X

표 7-6 논리 연산자

```
true || true; // true
true || false; // true
false || true; // true
false || false; // false
```

논리 부정 연산자(!)는 언제나 불리언 값을 반환한다.

단, 피연산자가 반드시 불리언 값일 필요는 없다. 만약 피연산자가 불리언 값이 아니라면 불리언 타입으로 암묵적 타입 변환된다.

논리합(||) 논리곱(&&) 연산자 표현식의 평가 결과는 불리언 값이 아닐 수도 있다.

논리합(||) 또는 논리곱 (&&) 연산자 표현식은 언제나 2개의 피연산자 중 어느 한쪽으로 평가된다.

`'Cat' && 'Dog'; // 'Dog'` → 단축 평가

드모르간의 법칙

논리 연산자로 구성된 복잡한 표현식은 가독성이 좋지 않아 한눈에 이해하기 어려울 때가 있다.

이러한 경우 드모르간 법칙을 활용하면 복잡한 표현식을 좀 더 가독성 좋은 표현식으로 변환할 수 있다.

```
!(x || y) === (!x && !y)
!(x && y) === (!x || !y)
```

쉼표 연산자

쉼표(,) 연산자는 왼쪽 피연산자부터 차례대로 피연산자를 평가하고 마지막 피연산자의 평가가 끝나면 마지막 피연산자의 결과를 반환한다.

그룹 연산자

소괄호()로 피연산자를 감싸는 그룹 연산자는 자신의 피연산자인 표현식을 가장 먼저 평가한다.

따라서 그룹 연산자를 사용하면 연산자의 우선순위를 조절할 수 있다.

그룹 연산자는 우선 순위가 가장 높다.

`수학 기호 ()와 똑같다.`

typeof 연산자

typeof 연산자는 피연산자의 데이터 타입을 문자열로 반환한다.

typeof 연산자는 7가지 문자열 `string, number, boolean, undefined, symbol, object, function` 중 하나를 반환한다.

null을 반환하는 경우는 없으며, 함수의 경우 function을 반환한다.

```
typeof '' // string
typeof 1 // number
typeof NaN // number
typeof true // boolean
typeof undefined // undefined
typeof Symbol() // symbol
typeof null // object
typeof [] // object
typeof {} // object
typeof new Date() // object
typeof /test/gi // object
typeof function() {} // function
```

typeof 연산자로 null 값을 연산해보면 null이 아닌, object를 반환한다는 데 주의하자

이는 자바스크립트의 첫 번째 버전의 버그다. 값이 null 타입인지 확인할 때는 typeof 연산자를 사용하지 말고 === 일치 연산자를 사용하자.

```
var foo = null;

typeof foo === null; // false
foo === null; // true
```

또 하나 주의해야 할 것이 있다. 선언하지 않은 식별자는 typeof 연산자로 연산해 보면 ReferenceError가 발생하지 않고 undefined를 반환한다.

지수 연산자

ES7에서 도입된 지수 연산자는 좌항의 피연산자를 밑으로, 우항의 피연산자를 지수로 거듭 제곱하여 숫자 값을 반환한다.

```
2 ** 2;
2 ** 2.5; // 5.656...
```

지수 연산자가 도입되기 이전에는 Math.pow 메소드를 사용했다.

```
Math.pow(2,2);
Math.pow(2,2.5);
Math.pow(2,0);
Math.pow(2,-2);
```

지수 연산자는 다음의 상황에서 Math.pow 메소드보다 가독성이 좋다.

```
2 ** (3 ** 2);
Math.pow(2, Math.pow(3,2));
```

음수를 거듭 제곱의 밑으로 사용하려면 다음과 같이 괄호로 묶어야 한다.

`(-5) ** 2; // 25`

`-5 ** 2 // Syntax Error`

지수 연산자는 다른 산술 연산자와 마찬가지로 할당 연산자와 함께 사용할 수 있다.

```
var num = 5;
num **= 2; // 25
```

그 외의 연산자

연산자	개요	참고
?.	옵셔널 체이닝 연산자	9.4.2절 "옵셔널 체이닝 연산자"
??	null 병합 연산자	9.4.3절 "null 병합 연산자"
delete	프로퍼티 삭제	10.8절 "프로퍼티 삭제"
new	생성자 함수를 호출할 때 사용하여 인스턴스를 생성	17.2.6절 "new 연산자"
instanceof	좌변의 객체가 우변의 생성자 함수와 연결된 인스턴스인지 판별	19.10절 "instanceof 연산자"
in	프로퍼티 존재 확인	19.13.1절 "in 연산자"

연산자의 부수 효과

대부분의 연산자는 다른 코드에 영향을 주지 않는다.

예를 들어, 1*2는 다른 코드에 어떠한 영향도 주지 않고 새로운 값 2를 생성할 뿐이다.

하지만 일부 연산자는 다른 코드에 영향을 주는 부수 효과가 있다.

부수 효과가 있는 연산자는 할당 연산자(=), 증가/감소 연산자(++/—), delete 연산자이다.

`delete` 연산자는 객체의 프로퍼티를 삭제하는 부수 효과가 있다. 이는 o 객체를 사용하는 다른 코드에 영향을 준다.

```
var o = { a:1};
delete o.a;
console.log(o); // {}
```