7장 연산자

연산자는 하나이상의 표현식을 대상으로 산술,할당,비교,논리,타입,지수 연산 등을 수행하여 하나의 값을 만든다.

7.1 산술 연산자

산수를 할 때 쓰는 연산자이다.

7.1.1 이항 산술 연산자

이항binary 산술 연산자는 2개의 피연산자를 산술 연산하여 숫자 값을 만든다.

모든 이항 산술 연산자는 피연산자의 값을 변경하는 부수 효과side effect가 없다. 다시 말해, 어떤 산술 연산을 해도 피연산자의 값이 바뀌는 경우는 없고 언제나 새로운 값을 만들 뿐이다.

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

표 7-1 이항 산술 연산자

7.1.2 단항 산술 연산자

단항unary 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

증가, 감소 연산자의 경우 연산자의 위치에 따라 의미가 다르다.

```
var x = 5, result;
//var x = 5; var result;와 동일하다고 함. x 에 5와 result를 선언하겠

result = x++; // result = 5,  x = 6

result = ++x; // result = 7,  x = 7
//--도 똑같음



console.log(-(x)); // -7
//-의 경우 양수를 음수로 음수를 양수로 바꿔줌.
//+의 경우 그런 기능은 없음

//다만 아래와같이 +, - 모두 문자열 숫자를 number타입으로 바꿔줄수있음

var five = '5';

console.log(+five); // 5
console.log(typeof(+five)); // number

five = true;
```

```
console.log(+five); // 1


five = false;

console.log(+five); // 0

five = null;

console.log(+five); // 0
//null의 경우 false와 같이 0임


five = undefined;

console.log(+five); // NaN

five = 'hohoho';

console.log(+five); // NaN
```

7.1 문자열 연결 연산자

문자열끼리 + 로 연결하게 되면 두 문자열이 합쳐져 하나의 문자열이 된다.

문자열에 숫자나 다른 타입들이 더해지는 경우, 문자열로 강제 형변환된다.

```
var lego = 'lego';
var somi = 'somi';

var catdog = somi + lego; // somilego
```

```
var hoho = null
var lego = 'lego'

var hohoho = lego+hoho
console.log(hohoho)// legonull



var hoho = true
var lego = 1

var hohoho = lego+hoho
console.log(hohoho) //2
console.log(typeof(hohoho)) //number
```

문자열의 경우 undefined를 형변환해서 사용하게되면 legoundifined같은 형식이 되지만, 숫자의 경우는 NaN이됨.

7.2 할당 연산자

7.2 할당 연산자

할당 연산자assignment operator는 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

위표에서 =를 제외한 연산자들은 연산과 할당을 동시에 시행함!

연쇄 할당도 가능하다.

```
var a,b,c;

a = b = c = 0;

console.log(a,b,c) ///0 , 0, 0


//단항 산술 연산자의 증가나 감소를 연쇄할당에 사용해보려 하였으나 이는 되지
```

7.3 비교 연산자

==

동등 비교에 쓰이며, 타입은 암묵적 타입 변환 후 값을 비교 후 같으면 true를 반환한다.

```
var one = 1;
var one_ = '1';

console.log(one == one_) // true
```

===

값만 같은게 아니라 타입까지 같아야 true를 반환한다.

```
var one = 1;
var one_ = '1';

console.log(one == one_) // false
```

≠

부동등 비교 연산자는 두 값이 같지 않은지 비교후 값이 다르다면 true를 반환한다.

==와 같이 암묵적 형변환이 이루어진다.

```
var one = 1;
var one_ = '1';

console.log(one != one_) // false



var one = 1;
var one_ = 2;

console.log(one != one_) // true
```

≠=

불일치 비교 연산자는 값과 타입 중 하나라도 다른경우에 true를 반환한다.

```
var one = 1;
var one_ = '1';

console.log(one !== one_) // true
```

NaN은 자기 자신과 일치하지 않는 유일한 값이다.

따라서 NaN의 비교는 Number.isNaN함수나, Object.is를 사용하여야한다.

```
var one = NaN;
var one_ = NaN;

console.log(one === one_) // false
console.log(Number.isNaN(one)) // true
console.log(Object.is(one,one_)) //true
```

7.4 삼항 조건 연산자

```
var somi_age = 1;

var isOld = somi_age >= 7 ? true : false

console.log(isOld);
```

? 앞의 연산자는 조건식이며, 그 조건식의 결과에 따라 isOld 가 정해진다. 앞의 연산자가 참이면 클론을 기준으로 앞에 값이 선택되고, 거젓일 시 클론 뒤의 값이 선택된다.

그리고 삼항 조건 연산자는 표현식인 문이며, 따라서 값처럼 사용할 수 있다.

7.5 논리 연산자

7.5 논리 연산자

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

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

표 7-6 논리 연산자

논리 부정(!)연산자는 언제나 불리언 값을 반환한다. 만약 피연산자가 불리언이 아니라면 암묵적 형변환이 이루어진다.

논리합 또는 논리곱 연산자 표현식의 평과결과는 불리언이 아닐 수도 있다.

```
// or
```

```
console.log('dog' || 'cat') // dog
console.log('cat' || null) // cat
console.log(null || 'cat') // cat



// and
console.log('dog' && 'cat') // cat
```

7.7 그룹 연산자

괄호를 이용하며, 연산자들의 우선순위를 조절하기 위해 사용된다.

그룹 연산자는 연산자들 중 가장 우선 순위가 높다.

7.8 typeof연산자

typeof 연산자는 "string", “number”, “bolean”, “undefined”, “symbot.”, "object", “function”중하나를반환한다.

typeof 연산자는 null 값을 object로 나타내므로 null값 비교는 === 연산자를 이용하여야 한다.

7.9 지수 연산자

**를 이용한다.

제곱을 나타낼때 쓰이며, Math.Pow를 이용할 수도 있다.

```
2 ** 2 // 4
Math.Pow(2,2) //4


(-5) ** 2
```

음수를 제곱하려면 위와 같이 괄호로 묶어야한다.

지수 연산자는 이항 연산자중 가장 우선순위가 높다.

7.10 그 외의 연산자

지금까지 소개한 연산자 외에도 다양한 연산자가 있다. 다만 아직 소개하지 않은 연산자는 다른 주제와 밀접하게 연관되어 있어 해당 주제를 소개하는 장에서 살펴보기로 하자.

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"

7.11 연산자 우선순위

연산자들의 우선 순위이다.

다 외우긴 힘드니 그룹 연산자로 적절히 조절하는것을 추천한다고 한다.

우선순위	연산자
1	()
2	new(매개변수 존재), [ ](프로퍼티 접근), ( )(함수 호출), ?.(옵셔널 체이닝 연산자[9])
3	new(매개변수 미존재)
4	x++, x--
5	!x, +x, -x, ++x, --x, typeof, delete
6	**(이항 연산자 중에서 우선순위가 가장 높다)
7	*, /, %
8	+, -
9	<, <=, >, >=, in, instanceof
10	==, !=, ===, !==

9 9.4.2절 "옵셔널 체이닝 연산자" 참고

우선순위	연산자
11	??(null 병합 연산자[10])
12	&&
13	||
14	? ... : ...
15	할당 연산자(=, +=, -=, ...)
16	,

7.13 연산자 결합 순서

연산자 결합 순서란 연산자의 어느 쪽(좌항 또는 우항)부터 평가를 수행할 것인지를 나타내는 순서를 말한다. 연산자의 결합 순서는 다음과 같다.

결합 순서	연산자
좌항 → 우항	+, -, /, %, <, <=, >, >=, &&, ||, ., [], (), ??, ?., in, instanceof
우항 → 좌항	++, --, 할당 연산자(=, +=, -=, ...), !x, +x, -x, ++x, --x, typeof, delete, ? ... : ..., **