4장. 표현식과 연산자

생략된 내용 엄격 모드, 비트 연산자

1. 표현식과 연산자란 무엇인가?

▸ 표현식

자바스크립트 인터프리터가 **값으로 평가**하는 구문

예) 1, “hello”, [ ], { }, x, x + y, …

▸ 연산자

값을 반환하는 구문

예) +, ++, ==, <

2. 기본 표현식

다른 표현식을 포함하지 않는 독립된 표현식. 즉, 원자성을 갖는 표현식

▸ 리터럴 (Literal)

```
1.23            // 숫자 리터럴
”hello”         // 문자열 리터럴
/pattern/       // 정규표현식 리터럴
```

▸ 예약어

```
true, false     // boolean 값
null            // null 값
this            // ‘현재’ 객체
```

▸ 변수

```
x               // 변수 x 값
fn1             // 함수 fn1 값
undefined       // undefined는 전역변수로 키워드가 아니다.
```

3. 함수의 표현식

▸ 함수의 정의

function 키워드, 매개변수, 중괄호로 둘러싸인 코드
로 구성된 리터럴 표현식으로 정의한다.

```
var hello = function(name) {
          console.log('hello ' + name);
}
```

▸ 함수의 호출

함수 표현식, 한 쌍의 중괄호, 호출인자
로 구성된 리터럴 표현식으로 정의한다.

```
hello('jaewon');
```

- 함수의 표현식 값이 호출 할 수 없는 객체라면, TypeError가 발생한다.
- 함수에 return 문이 있을 시 return 값이 반환되고, 없을 시 undefined가 반환된다.

4. 객체의 표현식

▸ 객체의 초기화

- 리터럴 표현식

```
var object1 = { p1 : ‘a’, p2 : ‘good’, ‘property 3’: ‘5’ };
```

- 생성자 호출 표현식 (함수 표현식 사용)

```
var object2 = new Object();  // 빈 객체를 생성한다.
```

▸ property 접근 표현식

- 점(.) 접근 표현식

```
object1.p1          // 식별자 property만 접근가능하다.
object1.”p1”        // 에러
```

- [“표현식”] 접근 표현식

```
object1[“p1”]                // 모든 property에 접근가능하다.
object1[“property 3”]        // 수, 문자열, 불리언, ..
```

- 객체의 표현식은 하위 표현식을 포함할 수 있기 때문에 기본 표현식이 아니다.
- . 표현식이 편하지만, 활용성 높은 [“ ”] 표현식이 필요할 때가 있다.
- 객체의 표현식 값이 null 이나 undefined 라면, TypeError가 발생한다.
- 객체에 해당 property가 없으면, 표현식은 undefined 값이된다.

5. 배열의 표현식

▸ 배열의 초기화

- 리터럴 표현식

```
var array1 = [‘a’, ‘b’, ‘c’];
```

- **생성자 호출 표현식** (함수 표현식 사용)

```
var array2 = new Array(n);  // 길이 n의 빈 배열을 생성한다.
```

▸ 배열의 원소 접근 표현식

array [index]

```
array1[0] === ‘a’  // true
array1[3] === undefined // true
```

- 배열의 표현식은 하위 표현식을 포함할 수 있기 때문에 기본 표현식이 아니다.
- 데이터가 존재하지 않는 index에 접근하면 표현식은 undefined 값이 된다.
- 마지막 쉼표는 무시된다. (IE 9 이상)

6. 연산자의 종류와 결합 방향

▸ 단항 연산자

항이 1개인 연산자로 결합방향은 **오른쪽 (R)**이다.

```
++, --, -, +, !, delete, typeof, void
```

▸ 2항 연산자

항이 2개인 연산자로 **대부분의** 결합방향은 **왼쪽 (L)**이다.

```
+, -, /, %, <, >, ==, ===, &&, ||, ..
```

* 항이 2개이면서 결합방향이 **오른쪽 (R)**인 연산자

```
할당 연산자 ( =, *=, /=, .. )
```

▸ 3항 연산자

항이 3개인 연산자로 결합방향은 **왼쪽 (L)**이다.

```
?:
```

- 각 연산자의 구체적인 기능은 책 참조.

7. 연산자의 특징과 우선순위

▸ 자동 타입 변환

대부분의 연산자는 **피연산자의 타입**과 **반환타입**이 정해져있다.
예를 들어, 곱셈 연산자는 피연산자 타입과 반환타입 모두 숫자이다.

```
'3' * 5
```

하지만 위 연산자는 문자열 * 숫자 를 연산했다.
이렇게 필요할 때 자바스크립트는 **타입 변환**을 한다.

또한 자바스크립트에서 모든 타입은 boolean 타입으로 변환 가능하다.
따라서, boolean 타입을 피연산자로 갖는 ! 연산자 같은 경우
모든 타입이 피연산자로 올 수 있다. 예) !1, !'hello', !true

[숫자] 0 = false / 0 이외의 수 = true
[문자열] 빈 문자열 = false / 비어있지 않은 문자열 = true
[기타] null = false / undefined = false

▸ 연산자의 우선순위

연산자는 각 연산자의 우선순위(1), 결합순서(2)에 따라 연산된다.
property 접근, 호출 표현식은 연산자보다 항상 우선순위가 높다.

7. 연산자의 특징과 우선순위

▸ 객체의 타입변환

연산할 때 객체의 타입변환은
숫자 연산일 때 **valueOf()** 메소드를 호출하고,
문자열 연산일 때, **toString()** 메소드를 호출한다.

테스트 해보자.

▸ 타입변환을 할 수 없는 경우 - 숫자

산술연산의 대부분의 피연산자는 숫자 타입인데,
숫자타입으로 변환할 수 없는 피연산자의 경우 **NaN**이, 연산 결과 역시 NaN이 된다.

```
'hello' * 5
```

곱 연산이기 때문에 'hello'를 숫자로 변환하려 하겠지만
'hello'는 숫자로 변환할 수 없는 문자열이다.
즉 'hello' 표현식은 NaN으로 평가되고, 전체 표현식 값 역시 NaN으로 평가된다.
그 외에도 0/0 등
숫자가 와야하지만 숫자로 표현할 수 없는 값은 **NaN**으로 평가된다.

8. 산술연산자

▸ -, /, %

마이너스(-), 나눗셈(/), 나머지(%) 연산자이다.

▸ +

덧셈 연산자는 피연산자의 타입에 따라 다르다.
둘 다 숫자 타입이라면, **덧셈 산술 연산**
한 쪽이라도 문자열 타입이라면, **문자열 붙이기 연산**을 한다.

▸ 단항 연산자 (++, --)

피연산자에 1을 더하는 연산(++), 피연산자에 1을 빼는 연산 (--)이다.

전치연산으로도, 후치연산으로도 쓸 수 있는데 차이점은
전치연산은 연산을 하고 값을 평가하고,
후치연산은 값을 평가하고 연산한다.

즉, `i = 1; j = ++i;` 이라면 `i = 2, j = 2`
`i = 1; j = i++;` 이라면 `i = 2, j = 1`

9. 관계연산자

자바스크립트는 타입이 유연한 언어이기 때문에, 두 종류의 관계연산자를 갖는다.

▸ 동치 연산자 (==, !=)

값이 같은지 비교한다.
즉, '1' == 1 이 된다.
객체의 경우에도 같은 프로퍼티, 같은 값을 가지면 같다고 평가한다.

▸ 일치 연산자 (===, !==)

값과 타입을 비교한다. 객체의 경우 참조값을 비교한다.
1 === 1 과 같이 완전히 같은 지 비교한다.

x === x가 되지 않는 한가지 예외가 NaN이다.
NaN !== NaN 이기 때문에
이 수식을 NaN을 판단하는 수식으로 이용할 수 있다.

▸ 비교 연산자 (>, >=, <, <=)

숫자 비교와 문자열 비교가 있으며
한 항이라도 숫자거나, 두 항 모두 숫자로 변환 가능하면 숫자비교를 우선시 한다.
NaN과 비교할 시 항상 false로 평가된다.

9. 논리연산자

대부분의 언어에 있는 연산자로 자세한 설명은 생략

- **AND (&&)**

  둘다 true일 때 true

- **OR (||)**

  둘중 하나라도 true면 true

- **NOT (!)**

  true면 false, false면 true

10. 기타 연산자

▸ in 연산자

```
[문자열] in [객체],  [숫자 또는 문자열] in [배열]
```

프로퍼티나 인덱스를 체크하는 연산자.
위와같이 사용하며, 좌변이 우변의 **프로퍼티** 또는 **인덱스**일 경우에
true를 반환한다. 그렇지 않을 때 false.

▸ instanceof 연산자

```
[객체] instanceof [생성자 함수]
```

객체의 클래스(생성자 함수)를 체크하는 연산자
후에 프로토타입 체이닝을 알고나면 더 잘 이해 할 수 있다.

▸ typeof 연산자

```
typeof [모든 타입]
```

피연산자의 데이터 타입을 확인하는 연산자.
몇가지 알아둬야할 부분은
typeof undefined : undefined / typeof null : object

10. 기타 연산자

▸ delete 연산자

```
delete [property]
```

객체의 프로퍼티를 삭제하는 연산자.
마치 해당 프로퍼티에 undefined를 할당하는 것 처럼 보일 수 있으나,
프로퍼티를 삭제하는 것이다. 삭제에 성공하면 true, 실패하면 false를 반환한다.
배열에 사용 시 해당 인덱스가 삭제되고 length가 변하지 않는다. (배열에는 쓰지말자)

▸ void 연산자

```
void [모든 타입]
```

피연산자를 무시하고 **undefined를 반환**하는 연산자.

▸ 쉼표(,) 연산자

```
[모든 타입], [모든 타입]
```

왼쪽을 평가하고, 오른쪽을 평가하며, **오른쪽을 반환**한다.
i = 0, j = 0; 이 표현식은 0이 된다.

▸ 삼항 연산자

```
[모든 타입] ? [모든 타입] : [모든 타입]
```

11. eval()

문자열을 자바스크립트 코드로 해석해서 평가한 값을 출력하는 전역함수
전역함수이지만 많은 기능이 제한되어 연산자처럼 동작하기에 연산자 챕터에 있다.
소스코드를 동적으로 평가하는 기능은 강력하지만
최적화에 영향을 끼치기에 삼가해야할 기능이다.

ECMA Script5부터 eval()은 호출 context의 변수환경을 그대로 사용한다.
예를 들면 다음과 같다.