9장 타입 변환과 단축 평가

정의

- 값의 타입은 개발자의 의도에 따라 다른 타입으로 변환할 수 있음
- 개발자가 의도적으로 값의 타입을 변환하는 것을 **명시적 타입 변환** 또는 **타입 캐스팅**

```
var x = 10;
var str = x.toString();
console.log(typeof str, str);
console.log(typeof x, x);
```

- 가끔 개발자의 의도와 상관없이 표현식을 평가하는 도중에 JS 엔진에 의해 암묵적으로 타입이 자동 변환되지도 함
- **암묵적 타입변환** 또는 **타입 강제 변환**

```
var x = 10;
var str = x + '';
console.log(typeof str, str);
console.log(typeof x, x);
```

- 명시적 타입 변환이나 암묵적 타입 변환이 기존 원시 값을 직접 변경하는 것은 아님
    - 원시 값은 변경 불가능한 값이므로 변경할 수 없으며 타입 변환이라는 건 기존 원시 값을 이용해 다른 타입의 새로운 원시 값을 생성하는 것을 뜻함
    - 위 코드를 참조해서 보자면 JS엔진은 x + ‘’ 라는 표현식을 평가하기 위해 x 변수의 숫자 값을 바탕으로 새로운 문자열 값 10을 생성하고 이것으로 표현식 10+” 을 평가함

이 때 암묵적으로 생성된 문자열’10’은 x에 할당되지 않음

- 암묵적 타입 변환
  - 기존 변수 값을 재할당하여 변경하는 것이 아니라 JS엔진에서 표현식을 에러 없이 평가하기 위해 피연산자의 값을 암묵적 타입 변환해 새로운 타입의 값을 만들어 내고 한번 사용한 후 변수에 재할당 되지 않음
- 명시적 타입 변환과 암묵적 타입 변환의 차이점 중 또 다른 하나는 개발자의 의지가 코드에서 나타나는가 안나타나는가로 구분이 가능함

그렇다면 암묵적 타입 변환이 무조건적으로 좋지 않은 문화이자 기능인가?

암묵적 타입 변환

- 개발자의 의도와 상관없이 JS엔진에서 강제적으로 변경되는 변환

```
'10' + 2 // -> '102' String
5 * '10' // -> 50 number
!0 // -> true boolean
```

문자열 타입으로 변환

```
`1 + 1 = ${1+1}` // -> " 1 + 1 = 2"
```

숫자 타입으로 변환

```
1 - '1' // -> 0
1 * '10' // -> 10
1 / NaN // -> NaN
```

- 이 때 피연산자를 숫자 타입으로 변환할 수 없는 경우는 산술 연산을 수행할 수 없으므로 표현식의 평가 결과는 NaN임

불리언 타입으로 변환

```
if (' ') console.log(x);아
```

- if 문이나 for문과 같은 제어문 또는 삼항 조건 연산자의 조건식은 불리언 값, 즉 논리적 참/거짓으로 평가되어야 하는 표현식
  - JS엔진 불리언 타입이 아닌 값을 Truthy 값 또는 Falsy 값으로 구분
  - false 값으로 변환되는 것은 false, underfined, null, 0, -0, NaN, ' '

명시적 타입 변환

- 표준 빌트인 생성자 함수를 new 연산자 없이 호출하는 방법과 빌트인 메서드를 사용하는 방법이 존재함
  - 표준 빌트인 생성자 함수

    객체를 생성하기 위한 함수이며 new 연산자와 함께 호출
  - 표준 빌트인 메서드

    JS에서 기본 제공하는 빌트인 객체의 메서드

문자열 타입 변환

1. String 생성자 함수를 new 연산자 없이 호출하는 방법
2. Object.prototype.toString 메서드를 사용하는 방법
3. 문자열 연결 연산자를 이용하는 방법

```
String(1);
String(NaN);
String(Infinity);
String(true);
String(false);

(1).toString();
(NaN).toString();
(Infinity).toString();
(true).toString();
(false).toString();

1 + '';
NaN + '';
Infinity + '';
true + '';
false + '';
```

숫자 타입 변환

1. Number 생성자 함수를 new 연산자 없어 호출하는 방법
2. parseInt, parseFloat 함수를 사용하는 방법(문자열만 숫자 타입으로 변환 가능)
3. + 단항 산술 연산자를 이용하는 방법
4. * 산술 연산자를 이용하는 방법

```
Number('1');
Number(true);
Number(false);

parseInt('0');

+'0';
+true;
+false;

'0'*1;
true*1;
false*1;
```

불리언 타입으로 변환

1. Boolean 생성자 함수를 new 연산자 없이 호출하는 방법
2. ! 부정 논리연산자를 두 번 사용하는 방법

```
Boolean('x');
Boolean(true);
Boolean(false);

Boolean(0);
Boolean(1);
Boolean(NaN);
Boolean(Infinity);

Boolean(null);
Boolean(undefined);

Boolean({});
Boolean([]);

!!'x';
!!'';
!!'false';

!!0;
!!1;
!!NaN;
!!Infinity;
```

```
!!null;
!!undefined;
!!{};
!![];
```