Ch33 7번째 데이터 타입 Symbol

심벌 값의 생성

다른 원시값(문자열, 숫자, 불리언, undefined, null 타입)은 리터럴 표기법을 통해 값을 생성하지만 심벌 값은 `Symbol` **함수**를 호출하여 생성해야 한다.

심벌 값은 외부로 노출되지 않아 확인할 수 없으며, **다른 값과 절대 중복되지 않는 유일무이한 값**이다.

→ 주로 이름의 충돌 위험이 없는 유일한 프로퍼티 키를 만들기 위해 사용함 (프로퍼티 키로 사용할 수 있는 값은 빈 문자열을 포함하는 모든 문자열 또는 심벌 값임)

Symbol 함수는 String, Number, Boolean 생성자 함수와는 달리 **new 연산자와 함께 호출하지 않음**. (new 연산자를 사용하면 `TypeError`가 발생함)

new 연산자와 함께 생성자 함수 또는 클래스를 호출하면 객체가 생성되지만 심벌 값은 변경 불가능한 원시 값이다.

Symbol 함수에는 선택적으로 문자열을 인수로 전달할 수 있다. 이 문자열은 생성된 심벌 값에 대한 설명으로 디버깅 용도로만 사용되며, 심벌 값 생성에 어떠한 영향도 주지 않는다.

즉, 심벌 값에 대한 설명이 같더라도 생성된 심벌 값은 유일무이한 값이다.

```
const mySymbol1 = Symbol('mySymbol');
const mySymbol2 = Symbol('mySymbol');

console.log(mySymbol1 === mySymbol2); //false
```

심벌 값은 암묵적으로 문자열이나 숫자 타입으로 변환되지 않지만 불리언 타입으로는 암묵적으로 타입 변환된다. (if문 등에서 존재 확인 가능)

Symbol.for / Symbol.keyFor 메서드

`Symbol.for` 메서드는 인수로 전달받은 문자열을 키로 사용하여 키와 심벌 값의 쌍들이 저장되어 있는 전역 심벌 레지스토리에서 해당 키와 일치하는 심벌 값을 검색한다.

- **검색에 성공**하면 새로운 심벌 값을 생성하지 않고 **검색된 심벌 값을 반환**한다.
- **검색에 실패**하면 **새로운 심벌 값을 생성**하여 `Symbol.for` 메서드의 인수로 전달된 키로 전역 심벌 레지스토리에 저장한 후, **생성된 심벌 값을 반환**한다.

```
//새로운 심벌 값 생성
const s1 = Symbol.for('mySymbol');
//기존 심볼 값 반환
const s2 = Symbol.for('mySymbol');

console.log(s1 === s2); //true
```

`Symbol.for` 메서드를 사용하면 애플리케이션 전역에서 중복되지 않는 유일무이한 상수인 심벌 값을 단 하나만 생성하여 전역 심벌 레지스토리를 통해 공유할 수 있다.

`Symbol.keyFor` 메서드를 사용하면 전역 심벌 레지스토리에 저장된 심벌 값의 키를 추출할 수 있다.

```
//전역 심벌 레지스토리에 mySymbol이라는 키로 저장된 심벌 값이 없으면 새로운 심벌 값을 생성
const s1 = Symbol.for('mySymbol');
//전역 심벌 레지스토리에 저장된 심벌 값의 키를 추출
Symbol.keyFor(s1); //mySymbol

//Symbol 함수를 호출하여 생성한 심벌 값은 전역 심벌 레지스토리에 등록되어 관리되지 않는다.
const s2 = Symbol('foo');
```

```
//전역 심벌 레지스토리에 저장된 심벌 값의 키를 추출
Symbol.keyFor(s2); //undefined
```

심벌과 상수

상수 값이 변경될 수 있으며 다른 변수 값과 중복될 수 있는 경우 변경/중복될 가능성이 있는 무의미한 상수 대신 중복될 가능성이 없는 유일무이한 심벌 값을 사용할 수 있다.

💡 **enum**

enum은 명명된 숫자의 집합으로 열거형이라고 부른다. enum을 지원하지 않는 자바스크립트에서 enum을 흉내 내어 사용하려면 객체의 변경을 방지하기 위해 객체를 동결하는 `Object.freeze` 메서드와 `심벌 값`을 사용한다.

심벌과 프로퍼티 키

객체의 프로퍼티 키는 빈 문자열을 포함하는 모든 문자열 또는 심벌 값으로 만들 수 있으며 동적으로 생성 할 수도 있다.

심벌 값을 프로퍼티 키로 사용하려면 프로퍼티 키로 사용할 심벌 값에 **대괄호**를 사용해야 한다. (프로퍼티에 접근할 때도 마찬가지)

심벌 값으로 프로퍼티 키를 만들면 다른 프로퍼티 키와 절대 충돌하지 않는다.

심벌과 프로퍼티 은닉

심벌 값을 프로퍼티 키로 사용하여 생성한 프로퍼티는 은닉되므로 for ... in 문이나 Object.keys, Object.getOwnPropertyNames 메서드로 찾을 수 없지만 ES6에서 도입된 `Object.getOwnPropertySymbols` 메서드를 사용하면 찾을 수 있다.

```
const obj = {
//심벌 값으로 프로퍼티 키를 생성
[Symbol('mySymbol')]: 1
};

//getOwnPropertySymbols 메서드는 인수로 전달한 객체의 심벌 프로퍼티 키를 배열로 반환한다.
console.log(Object.getOwnPropertySymbols(obj)); //[Symbol(mySymbol)]

//getOwnPropertySymbols 메서드로 심벌 값도 찾을 수 있다.
const symbolKey1 = Object.getOwnPropertySymbols(obj)[0];
console.log(obj[symbolKey1]); //1
```

심벌과 표준 빌트인 객체 확장

직접 추가한 메서드와 미래에 표준 사양으로 추가될 메서드의 이름이 중복될 수 있기 때문에 표준 빌트인 객체에 사용자 정의 메서드를 직접 추가하여 확장하는 것은 권장하지 않고 **읽기 전용으로 사용**하는 것이 좋다. 하지만 중복될 가능성이 없는 심벌 값으로 프로퍼티 키를 생성하여 표준 빌트인 객체를 확장하면 어떤 프로퍼티 키와도 충돌하지 않아 안전하게 표준 빌트인 객체를 확장할 수 있다.

Well-known Symbol

Well-known Symbol은 자바스크립트가 기본 제공하는 빌트인 심벌 값을 ECMAScript 사양에서 부르는 것이고 자바스크립트 엔진의 내부 알고리즘에 사용된다.

for ... of 문으로 순회 가능한 빌트인 이터러블은 Well-known Symbol인 `Symbol.iterator`를 키로 갖는 메서드를 가지며 Symbol.iterator 메서드를 호출하면 이터레이터를 반환하도록 규정되어 있다.