삼육대학교 SW융합교육원 김진호

자바스크립트 기초교육

<class>
</body>
<head>
<title>

03. Symbol (심볼)

3-1. Symbol basic syntax

3-2. Symbol feature

3. Symbol

- 자바스크립트가 ECMAScript로 표준화 된 이래로 자바스크립트에는 6개의 타입 (문자열, 숫자, 불리언, undefined, null, 객체)이 있었다.
- Symbol은 ES6에서 도입 된 7번째 데이터 타입으로 변경 불가능한 원시 타입의 값이다.
- Symbol은 다른 값과 중복 되지 않는 유일무이한 값으로 주로 이름 충돌의 위험이 없는 유일한 프로퍼티 키를 만들기 위해 사용 된다.

3-1. Symbol basic syntax (심볼 기본 문법)

3-1-1. Symbol basic syntax (심볼 기본 문법)

- Symbol()을 사용하면 심볼 값을 만들 수 있다.
- 디버깅 시 유용한 심볼 이름 설명을 붙여 심볼 값을 만들 수도 있다.
- 심볼은 유일성이 보장되는 자료형이기 때문에, 설명이 동일한 심볼을 여러 개 만들어도 각 심볼값은 다르다.
- 심볼 이름은 어떤 것에도 영향을 주지 않는 이름표 역할만을 한다.
- 전역 심볼 레지스트리(global symbol registry)에 심볼을 만들고 해당 심볼에 접근하면, 이름이 같은 경우 항상 동일한 심볼을 반환한다.
- Symbol.for(key)를 사용해 이름이 key인 심볼을 전역 심볼 레지스트리에서 읽는다.
- Symbol.keyFor(symbol)를 사용하면 이름을 얻을 수 있다.

3. Symbol

3-1. Symbol basic syntax (심볼 기본 문법)

3-1-1. Symbol basic syntax (심볼 기본 문법)

```
// Symbol()을 사용하면 심볼 값을 만들 수 있다.
let symbol1 = Symbol();

// 디버깅 시 유용한 심볼 이름 설명을 붙여 심볼 값을 만들 수도 있다.
let symbol2 = Symbol("mySymbol");

// 심볼은 유일성이 보장되는 자료형이기 때문에, 설명이 동일한 심볼을 여러 개 만들어도
각 심볼값은 다르다.
// 심볼 이름은 어떤 것에도 영향을 주지 않는 이름표 역할만을 한다.
let symbol3 = Symbol("mySymbol");
console.log(symbol2 == symbol3);    // false - symbol2와 symbol3는 이름만
같을 뿐 다른 값이다.

// 전역 심볼 레지스트리(global symbol registry)에 심볼을 만들고 해당 심볼에 접근하면,
이름이 같은 경우 항상 동일한 심볼을 반환한다.

// Symbol.for(key)를 사용해 이름이 key인 심볼을 전역 심볼 레지스트리에서 읽는다.
let symbol = Symbol.for("id"); // 심볼이 존재하지 않으면 새로운 심볼을 만든다.

// 동일한 이름을 이용해 심볼을 다시 읽는다.
let idAgain = Symbol.for("id");

console.log(symbol === idAgain); // true - 두 심볼은 같다.

// 반대로 Symbol.keyFor(symbol)를 사용하면 이름을 얻을 수 있다.
let sym = Symbol.for("name");
let sym2 = Symbol.for("id");
console.log(Symbol.keyFor(sym)); // name
console.log(Symbol.keyFor(sym2)); // id
```

3. Symbol

3-1. Symbol basic syntax (심볼 기본 문법)

3-1-1. Symbol basic syntax (심볼 기본 문법)

- 심볼은 이름이 같더라도 값이 항상 다르므로 이름이 같을 때 값도 같길 원한다면 전역 레지스트리를 사용해야 한다.
- 전역 심볼 레지스트리는 애플리케이션 곳곳에서 심볼 이름을 이용해 특정 프로퍼티에 접근해야 할 경우 사용할 수 있다.

3. Symbol

3-2. Symbol feature (심볼 특징)

3-2-1. Symbol feature (심볼 특징)

- Symbol을 이용하면 외부 코드에서 접근이 불가능하고 값도 덮어쓸 수 없는 숨김 프로퍼티를 만들 수 있다.

```
let student = {
    name : "홍길동"
};

// id 심볼 생성 후 프로퍼티로 추가한다.
let id = Symbol("id");
student[id] = 1;

// student 객체의 키 값, 프로퍼티 이름 등에 name만 나타나
고 id는 나타나지 않는다.
console.log(Object.keys(student));
console.log(Object.getOwnPropertyNames(student));

console.log(student[id]);
```

3. Symbol

3-2. Symbol feature (심볼 특징)

3-2-1. Symbol feature (심볼 특징)

- 객체 리터럴 안에서 사용할 경우 대괄호를 사용해 심볼형 키를 만들어야 한다.

```
let student2 = {
    name : "유관순",
    age : 16,
    [id] : 2
}
```

3. Symbol

3-2. Symbol feature (심볼 특징)

3-2-1. Symbol feature (심볼 특징)

- 키가 심볼인 프로퍼티는 for...in 반복문에서 배제된다.
- 외부 스크립트나 라이브러리는 심볼 정보를 갖고 있지 않아서 프로퍼티에 직접 접근하는 것이 불가능하다.
- 따라서 심볼형 키를 사용하면 프로퍼티가 우연히라도 사용되거나 덮어씌워 지는 걸 예방할 수 있다.

```
// 키가 심볼인 프로퍼티는 for...in 반복문에서 배제된다.
for(let key in student2) console.log(key);
```

3. Symbol

3-2. Symbol feature (심볼 특징)

3-2-1. Symbol feature (심볼 특징)

- 심볼은 완전히 숨길 수는 없다. 내장 메서드 Object.getOwnPropertySymbols(obj)를 사용하면 모든 심볼을 볼 수 있다.
- 하지만 대부분의 라이브러리, 내장 함수 등은 이런 메서드를 사용하지 않는다.

```
console.log(Object.getOwnPropertySymbols(student));
console.log(Object.getOwnPropertySymbols(student2));
```

- System Symbol을 이용하면 내장 메서드 등의 기본 동작을 변경할 수 있다.
- Symbol은 중복 되지 않는 상수 값을 생성하는 것은 물론 기존에 작성 된 코드에 영향을 주지 않고 새로운 프로퍼티를 추가하기 위해, 즉 하위 호환성을 보장하기 위해 도입 되었다고 할 수 있다.