[ JavaScript ] - 원시 자료형과 참조 자료형

JS에서 자료형(type)은 값(value)의 종류를 가리킵니다.

자료형에 따라서 고유한 속성과 메서드를 가지고 있으며, 자료형은 크게 원시자료형(primitive type)과 참조 자료형(reference type)으로 나눌 수 있습니다.

### 1. 원시 자료형(primitive type)

자바스크립트에서 원시 자료형에 해당하는 타입은 6개입니다.

-number

-string

-boolean

-undefined

-null

-symbol

#### 원시 자료형의 특징

* 원시 자료형을 변수에 할당하면 메모리 공간에 값 자체가 저장됩니다.
* 원시 값을 갖는 변수를 다른 변수에 할당하면 해당 값 자체가 복사되어 전달됩니다.
* 원시 자료형은 변경 불가능한 (immutable value)이며, 한 번 생성된 원시 자료형은 읽기 전용 값입니다.

let a = 10;

a = 20;

*// 메모리 내부에서는 먼저 a라는 변수에 10의 값이 저장된 공간이 생성되고*

*// 변수 a에 20이라는 값을 재할당하면 값 10이 20으로 변경되는 것이 아니라*

*// 새로운 메모리 공간에 변수 a라는 이름으로 값 20이 저장된다*

"즉, 원시 자료형은 값을 변경할 수 없습니다."

#### 가비지 콜렉터(garbage collector)

위의 코드에서 a라는 변수가 참조하는 값은 20이며 변수 a의 이전 값인 10은 현재 어떤 변수도 값으로 갖고 있지 않은 상태입니다.

10이라는 값은 어떤 식별자와도 연결되어 있지 않은, 사용되고 있지 않은 값이며 메모리의 공간만 차지하고 있는 상황이기 때문에 불필요합니다. 자바스크립트는 가비지 콜렉터라는 기능이 내장되어 있는데, 바로 10과 같이 더 이상 사용되지 않는 메모리 공간을 해제(release)하는 역할을 합니다. 하지만 언제 메모리에서 해제될 것인가는 예측할 수 없습니다.

.

#### ※ 언매니지드언어와 매니지드 언어

프로그래밍 언어는 메모리 관리 방식에 따라 언매니지드 언어와 매니지드 언어로 분류할 수 있습니다. C언어와 같은 언매니지드 언어는 개발자가 메모리를 할당하고 해제하기 위해 malloc()이나 free() 같은 저수준(low-level) 메모리 제어 기능을 제공합니다. 언매니지드 언어는 메모리 제어를 개발자가 주도할 수 있기에 개발자의 역량에 따라 최적의 성능을 확보할 수도 있지만, 반대로 치명적인 오류를 만들어 낼 가능성도 있습니다.

자바스크립트와 같은 매니지드 언어는 메모리의 할당 및 해제를 위한 메모리 관리 기능은 언어 차원에서 담당하고, 개발자의 직접적인 메모리 제어를 허용하지 않습니다. 즉 개발자가 명시적으로 메모리를 할당하고 해제할 수 없는 것입니다. 더 이상 사용하지 않는 메모리의 해제를 수행하는 가비지 콜렉터에도 개발자가 직접 관여할 수 없습니다. 매니지드 언어는 메모리를 자동으로 관리해주기 때문에 일정 수준 이상의 생산성이 확정적으로 보장되지만, 달리 말하면 개발자의 역량과는 관계 없이 메모리 관리를 통해 얻을 수 있는 성능의 한계가 명확하다는 의미입니다.

### 2. 참조 자료형(reference type)

원시 자료형이 아닌 모든 자료형은 참조 자료형입니다. 여러 개의 데이터를 한 번에 다룰 수 있는 객체와 배열이 대표적인 참조 자료형이며, 함수 또한 참조 자료형에 해당합니다.

#### 참조 자료형의 특징

* 참조 자료형을 변수에 할당하면 메모리 공간에 주솟값이 저장됩니다.
* 참조 값을 갖는 변수를 다른 변수에 할당하면 주솟값이 복사되어 전달됩니다.
* 참조 자료형은 변경이 가능한 값(mutable value)입니다.

값 자체를 메모리 공간에 저장하는 원시 자료형과 다르게 여러 개의 값을 다룰 수 있는 참조 자료형은 하나의 메모리 공간에 값을 저장할 수 없습니다. 따라서 자바스크립트에서 참조 자료형은 heap이라는 특별한 저장 공간에 값이 저장되고, 해당 값에 접근할 수 있는 주솟값을 메모리 공간에 저장합니다. 이 주솟값을 통해 heap에 저장된 값에 접근하는 것을 참조한다(refer)라고 표현합니다.

let arr = [0, 1, 2, 3];

let copiedArr = arr;

let arr = [0, 1, 2, 3];

arr[3] = '3';

arr.push(4);

arr.shift();

console.log(arr); *// [1, 2, '3', 4]*

*// 원시 자료형과 다르게 참조 자료형은 값을 변경할 수 있습니다. //*

위 코드에서 arr[3]의 값은 숫자 3 이었습니다. 하지만 arr[3]에 '3'을 재할당하여 값 자체를 변경할 수 있음을 볼 수 있고, push와 shift를 사용하여 값의 추가와 삭제 역시 가능함을 알 수 있습니다.

### 3. 원시 자료형과 참조 자료형의 복사

* 원시 자료형이 할당된 변수를 다른 변수에 할당하면 값 자체의 복사가 일어납니다. 따라서 원본과 복사본 중 하나를 변경해도 다른 하나에 영향을 미치지 않습니다.
* 참조 자료형이 할당된 변수를 다른 변수에 할당하면 주소가 복사되어 원본과 복사본이 같은 주소를 참조합니다.
* 참조 자료형의 주소값을 복사한 변수에 요소를 추가하면 같은 주소를 참조하고 있는 원본에도 영향을 미칩니다.
* 참조 자료형이 저장된 변수를 다른 변수에 할당할 경우, 두 변수는 같은 주소를 참조하고 있을 뿐 값 자체가 복사 되었다고 볼 수 없습니다.

### 4. 얕은 복사와 깊은 복사

참조 자료형을 복사하는데에는 대략 3가지 방법이 있습니다. 얕은 복사(shallow copy), 깊은 복사(deep copy), 완전한 깊은 복사입니다.

얕은 복사는 slice(), Object.assign(), spread syntax 등을 이용해 참조 자료형을 복사하는 것입니다. 이 방법을 사용해 복사하면 원본과 복사된 참조 자료형이 같은 요소를 가지고 있지만, 다른 주소를 참조하게 됩니다. 따라서 서로 값을 변경해도 서로에게 영향을 주지 않게 됩니다.

하지만, 참조 자료형 내부에 또다른 참조 자료형이 중첩되어 있는 상태에서는 가장 외부에 있는 참조 자료형만 복사되고 내부에 존재하는 참조 자료형은 같은 주소를 참조하게 됩니다.

이런 경우에 내부에 존재하는 참조 자료형까지 모두 복사하는 것이 깊은 복사입니다. 그러나 자바스크립트 내부적으로 깊은 복사를 수행할 수 있는 방법은 존재하지 않습니다. 자바스크립트의 문법을 응용해서 깊은 복사와 동일한 결과물을 만들어 낼 수 있을 뿐입니다.

이 때 사용하는 방법이 JSON.stringfy()와 JSON.parse()입니다.

let arr = [1, 2, [3, 4]];

console.log(JSON.stringify(arr)) *// [1,2,[3,4]]*

console.log(JSON.parse(JSON.stringify(arr))) *// console.log(arr)과 동일 //*

let str = JSON.stringify(arr);

console.log(str[0]); *// [ //*

console.log(str[1]); *// 1 //*

console.log(str[2]); *// 2 //*

위 방법을 사용해 깊은 복사를 할 수 있지만, 여기도 예외가 존재합니다. 바로 중첩된 참조 자료형 중에 함수가 포함되어 있을 경우 복사되지 않고 null이 되어버립니다. 따라서 이 방법 역시 완전한 깊은 복사라고 하기는 어렵습니다.

완전한 깊은 복사가 꼭 필요하다면 node.js 환경에서 외부 라이브러리인 lodash, 또는 ramda를 설치하면 됩니다. lodash와 ramda는 각각 방법으로 깊은 복사를 구현해 두었습니다. 다음은 lodash의 cloneDeep을 사용한 깊은 복사의 예시입니다.

const lodash = require('lodash');

const arr = [1, 2, [3, 4]];

const copiedArr = lodash.cloneDeep(arr);

console.log(arr); *// [1, 2, [3, 4]]*

console.log(copiedArr); *// [1, 2, [3, 4]]*

console.log(arr === copiedArr) *// false*

console.log(arr[2] === copiedArr[2]) *// false*