Ch.11 원시 값과 객체의 비교

▼ 🌟🌟🌟 원시 타입 vs 객체 타입

- 원시 타입의 값은 변경 불가능한 값이다.
  객체 타입의 값은 변경 가능한 값이다.
  불변성을 갖는 원시 값을 할당한 변수는 재할당 이외에 변수 값을 변경할 수 있는 방법이 없다.
- 원시 값을 변수에 할당하면 변수(확보된 메모리 공간)에는 **실제 값**이 저장된다.
  객체를 변수에 할당하면 변수에는
  참조값이 저장된다.
- 원시 값을 갖는 **변수를 다른 변수에 할당하면** 원본의 **원시 값이 복사되어 전달**된다.
  이를 값에 의한 전달이라 한다.
  객체를 가리키는
  변수를 다른 변수에 할당하면 원본의 **참조값이 복사되어 전달**된다. 이를 참조에 의한 전달이라 한다.

p138(e172)

+값이 변경 불가능 하다는 것이지 변수가 변경불가능 하다는게 아니다. 변수는 언제든지 재할당을 통해 변수 값을 교체 할 수 있다.
+상수와 변경 불가능한 값을 동일시하는 것은 곤란하다. 상수는 재할당이 금지된 변수일 뿐이다.

예제 11-01

```
// const 키워드를 사용해 선언한 변수인 상수.
const o = { };


// const 키워드를 사용해 선언한 상수에 할당한 객체는 변경할 수 있다.
```

```
o.a = 1;
console.log(o); // {a: 1}
```

▼ 🌟 🌟 🌟 문자열과 불변성

- 원시값을 저장하려면 먼저 확보해야 하는 메모리 공간의 크기를 결정해야 한다.

  ECMAScript 사양에 문자열 타입(2바이트)과 숫자 타입(8바이트) 이외의 원시 타입은 크기를 명확히 규정하고 있지는 않아서 브라우저 제조사의 구현에 따라 원시 타입의 크기는 다를 수 있다.

  문자열은 **0개 이상의 문자로 이뤄진 집합**을 말하며 1개의 문자는 2바이트의 메모리 공간에 저장된다. 따라서 문자열은 몇개의 문자로 이뤄졌느냐에 따라 필요한 메모리 공간의 크기가 결정된다. 숫자값은 1도 1000000도 동일한 8바이트가필요하지만 문자열의 경우(실제와는 다르지만 단순하게 계산했을때) 1개의 문자로 이뤄진 문자열은 2바이트, 10개의 문자로 이뤄진 문자열은 20바이트가 필요하다.

- 문자열은 **유사배열객체** 이면서 **이터러블**이므로 배열과 유사하게 각 문자에 접근할 수 있다.

  원시값인 문자열이 객체일수도 있다니 혼란스러울 수 있지만, **원시 값을 객체처럼 사용하면 원시값을 감싸는 래퍼 객체로 자동 변환된다.** p141(e175)

- 문자열은 원시 값이므로 변경할 수 없다. 예제 11-05

```
var str = 'string';

str[0] = 'S';  // 문자열은 유사배열이므로 인덱스를 사용해 각 문

console.log(str);  // string 문자열은 원시값이므로 변경할 수
```

+유사 배열 객체:
마치 배열처럼
인덱스로 프로퍼티 값에 접근할 수 있고 length프로퍼티를 갖는 객체를 말한다.
문자열은 마치 배열처럼 인덱스를 통해 각 문자에 접근할 수 있으며 length프로퍼티를 갖기 때문에 유사배열 객체이고 for문으로 순회할 수도 있다.

▼ 🌟 🌟🌟 값에 의한 전달

변수에 원시 값을 갖는 변수를 할당하면 할당받는 변수에는 **할당되는 변수의 원시 값이 복사되어 전달**된다. 이를 값에 의한 전달이라 한다.

p143(e177) 그림 11-3 copy = score에서 score는 변수 값 80으로 평가되므로 copy 변수에도 80이 할당되는데 이때 새로운 숫자 값 80이 생성되어 copy변수에 할당된다. score변수와 copy변수의 값 80은 **다른 메모리 공간에 저장된 별개의 값이다.**

따라서 score의 변수값을 변경해도 copy의 변수값은 여전히 80이다. 그림 11-4, 11-5

엄격하게 표현하면 변수에는 값이 전달되는 것이 아니라 메모리 주소가 전달된다. 변수가 같은 식별자는 값이 아니라 메모리 주소를 기억하고 있기 때문이다.

값에 의한 전달도 사실은 값을 전달하는게 아니라 메모리 주소를 전달한다. 단, 전달된 메모리 주소를 통해 메모리 공간에 접근하면 값을 참조할 수 있다.

p145(e179) 예제 [11-09]

```
var x = 10;
```

할당연산자는 숫자 리터럴 10에 의해 생성된 숫자 값 10이 저장된 메모리 공간의 주소를 전달한다. 이로써 식별자 x는 메모리 공간에 저장된 숫자 값 10을 식별할 수 있다.

p145(e179) 예제 [11-10]

```
var score = 80;

var copy = score;
```

위 예제의 경우 두가지 평가 방식이 가능하다.

- 새로운 80을 생성(복사)해서 메모리 주소를 전달하는 그림 11-4 방식. 이 방식은 할당 시점에 두 변수가 기억하는 메모리 주소가 다르다.
- score의 변수값 80의 메모리 주소를 그대로 전달하는 그림 11-5방식. 이 방식은 할당 시점에 두 변수가 기억하는 메모리 주소가 같다.

+p144(e178) 파이썬의 경우, 변수에 원시 값을 갖는 변수를 할당하는 시점에는 두 변수가 같은 원시 값을 참조하다가 어느 한쪽의 변수에 재할당이 이뤄졌을때 비로소 새로운 메모리 공간에 재할당된 값을 저장하도록 동작한다.

변수에 원시 값을 갖는 변수를 할당하면 변수 할당 시점이든, 두 변수 중 어느 하나의 변수에 값을 재할당하는 시점이든 중요한 것은 **결국은 두 변수의 원시 값은 서로 다른 메모리 공간에 저장된 별개의 값이 되어 어느 한쪽에서 재할당을 통해 값을 변경하더라도 서로 간섭할 수 없다는 것이다.**

▼ 🌟 🌟 객체

- 객체는 프로퍼티의 개수가 정해져있지 않으며 동적으로 추가되고 삭제할 수 있다. 또한 프로퍼티 값에도 제약이 없다. 따라서 객체는 원시 값과 같이 확보해야 할 메모리 공간의 크기를 사전에 정해 둘 수 없다.
- **변경 가능한 값**
  객체를 할당한 변수에 재할당을 하지 않고 변경 가능하므로 객체를 할당한 변수의 참조값은 변경되지 않는다.
- 객체를 할당한 변수를 참조하면 메모리에 저장되어 있는 **참조 값**을 통해 실제 객체에 접근한다. 참조값은 생성된 객체가 저장된 메모리 공간의 주소 그 자체다. 변수는 이 참조 값을 통해 객체에 접근할 수 있다. p148(e182) 그림 11-7
  객체를 할당한 변수의 경우 “변수는 객체를 참조하고 있다, 가리키고 있다” 라고 표현한다.
- **여러개의 식별자가 하나의 객체를 공유할 수 있다.**

+자바스크립트 객체의 관리 방식 p146(e180)

자바스크립트 객체는 프로퍼티 키를 인덱스로 사용하는 해시 테이블이라고 생각할 수 있다. (해시 테이블은 연관 배열, map, dictionary, lookup table이라 부르기도 한다.)

+ p149(e183) 객체를 생성하고 관리하는 방식은 매우 복잡하며 비용이 많이 든다. 객체를 변경할때마다 원시 값처럼 이전 값을 복사해서 새롭게 생성한다면 명확하고 신뢰성이 확보되겠지만 객체는 크기가 매우 클 수도 있고 원시 값처럼 크기가 일정하지도 않으며 프로퍼티 값이 객체일 수도 있어서 복사해서 생성하는 비용이 많이 든다. 즉, 메모리의 효율적 소비가 어렵고 성능이 나빠진다.
따라서 메모리를 효율적으로 사용하고 객체를 복사해 생성하는 비용을 절약해 성능을 향상시키기 위해 객체는 변경 가능한 값으로 설계되어 있다.
메모리 사용의 효율성과 성능을 위해 어느정도 구조적인 단점을 감안한 설계라고 할 수 있다.

객체는 이러한 구조적 단점에 따른 부작용이 있다. 원시 값과는 다르게 **여러 개의 식별자가 하나의 객체를 공유할 수 있다는 것이다.**

▼ 🌟 🌟 얕은 복사 vs 깊은 복사

p150(e184) 예제 11-14, 11-15

객체를 프로퍼티 값으로 갖는 객체의 경우, 얕은 복사는 1단계 까지만 복사하는 것을 말하고
깊은 복사는
객체에 중첩되어 있는 객체까지 모두 복사하는 것을 말한다.

```
const o = { x: {y: 1} };

// 얕은 복사
const c1 = { ...o };  // 스프레드 문법
console.log(c1 === o);  // false
console.log(c1.x === o.x);  // true

// lodash의 cloneDeep을 사용한 깊은 복사
const _ = require('lodash');
const c2 = _.cloneDeep(o);
console.log(c2 === o); // false
console.log(c2.x === o.x); // false
```

얕은 복사와 깊은 복사로 생성된 객체는 원본과는 다른 객체다. 즉, 원본과 복사본은 참조값이 다른 별개의 객체다.

얕은 복사는 객체에 중첩되어 있는 객체의 경우 **참조값을 복사하고**
깊은 복사는
객체에 중첩되어 있는 객체까지 모두 복사해서 원시 값처럼 완전한 복사본을 만든다는 차이가 있다.

▼ 🌟 🌟🌟 참조에 의한 전달

객체를 가리키는 변수를 다른 변수에 할당하면 **원본의 참조 값이 복사되어 전달**된다. 이를 참조에 의한 전달이라 한다. 그림 11-9 p151(e185)

이것은 **두개의 식별자가 하나의 객체를 공유**한다는 것을 의미한다.
따라서

원본 또는 사본 중 어느 한쪽에서 객체를 변경하면 서로 영향을 주고 받는다. (재할당이 아니라 객체의 프로퍼티 값을 변경, 추가, 삭제하면)

+결국 값에 의한 전달과 참조에 의한 전달은 식별자가 기억하는 메모리 공간에 저장되어 있는 값을 복사해서 전달한다는 면에서 동일하다. 다만 식별자가 기억하는 메모리 공간, 즉 변수에 저장되어 있는 값이 원시 값이냐 참조 값이냐의 차이만 있을 뿐이다. 따라서 자바스크립트에는 참조에 의한 전달은 존재하지 않고 값에 의한 전달만 존대한다고 말할 수 있다.
p152(e186)