11강 대신 하는 얕은 복사, 깊은 복사

얕은 복사(Shallow Copy)

- 객체를 복사할 때 기존 값과 복사된 값이 같이 참조를 가리키고 있는 것을 뜻함
- 객체 안에 객체가 있을 경우 한 개의 객체라도 기존 변수의 객체를 참조하고 있다 → 얕은 복사

특징

- 객체를 복사 할 때, 해당 객체만 복사하여 새 객체를 생성
- 복사된 객체의 인스턴스 변수는 원본 객체의 인스턴스 변수와 같은 메모리 주소를 참조
- 메모리 주소의 값이 변경되면 원복 객체 및 복사 객체의 인스턴스 변수 값은 같이 변경됨

방법

Array.prototype.slice()

- start index부터 end index까지 기존 Array에서 추출하여 새로운 Array를 반환하는 메서드
- 여기서, 파라미터 값을 넣지 않으면 기존 Array를 전체 얕은 복사를 함

💡 밑의 코드를 실행해보고 이유를 생각해보시오.

```
const original = [
  [18, 18, 18, 18],
  [19, 19, 19, 19],
  [20, 20, 20, 20],
  [21, 21, 21, 21],
];

const copy = original.slice();

console.log(JSON.stringify(original) === JSON.stringify(copy));

copy[0][0] = 99;
copy[2].push("02");

console.log(JSON.stringify(original) === JSON.stringify(copy));
console.log(original);
console.log(copy);
```

Object.assign()

- 파라미터 소개 - (Target : 생성할 객체 , Source : 복사할 객체)
- 출처 Object들의 모든 열거 가능한 자체 속성을 복사해 Target Object에 붙여넣고 Object를 반환함
  - 해당 메서드는 두번째 파라미터에 **null || undefined**를 넣어도 예외가 발생되지 않음
  - 객체 합칠 때 자주 씀

💡 밑에 있는 코드들을 실행해 보시고 이유를 생각해보세요.

```
const obj = { a: 1 };
const newObj = Object.assign({}, obj);

newObj.a = 2;

console.log(obj);
console.log(obj === newObj);
```

```
const obj = {
  a: 1,
  b: {
    c: 2,
  },
};

const newObj = Object.assign({}, obj);

newObj.b.c = 3;

console.log(obj);
console.log(obj.b.c === newObj.b.c);
```

- 1 차원 Object는 깊은 복사가 이행되지만 2 차원 Object부터는 경우에는 얕은 복사가 이행됨
    - 속성까지는 다른 주소를 참조하지만 c와 같은 내부 Object는 같은 주소를 참조해서 일어나는 일

Spread 연산자(전개 연산자)

- Array 나 String과 같이 반복 가능한 문자를 0개 이상의 argument(함수 호출 시)또는 요소(배열 리터럴의 경우)로 확장하여, 0개 이상의 key-value 쌍으로 형식으로 Object로 확장 시킬 수 있음 - MDN 설명
- 연산자(…) 사용 시 기존 배열이나 객체의 전체 또는 일부를 다른 배열이나 객체로 빠르게 복사할 수 있음 - W3schools 설명

알잘딱 위처럼 아시죠?

```
const obj = { a: 1 };
const newObj = Object.assign({}, obj);

newObj.a = 2;

console.log(obj);
console.log(obj === newObj);
```

```
const obj = {
  a: 1,
  b: {
    c: 2,
```

```
  },
};

const newObj = { ...obj };

newObj.b.c = 3;

console.log(obj);
console.log(obj.b.c === newObj.b.c);
```

- 놀랍게도 이 기법도 2 차원 Object에서는 얕은 복사가 됨

깊은 복사 (Deep Copy)

- 객체 안에 객체가 있을 경우에도 원본과의 참조가 완전히 끊어진 객체를 뜻함
- 기존 객체의 값만 복사본으로 가져와 별도로 활용하기 위함

특징

- 객체를 복사 할 때, 해당 객체와 인스턴스 변수까지 복사함
- 전부를 복사하여 새 주소에 담기 때문에 참조를 공유하지 않음
- 객체가 참조 타입의 멤버를 포함할 경우 참조 값의 복사가 아닌 참조된 객체 자체가 복사되는 것을 의미 → 원본의 참조는 더 이상 하지 않음

JSON.parse && JSON.stringify

- JSON.stringify() 객체를 json 문자열로 변환함
    - 해당 과정에서 원본 Object와의 참조가 모두 끊어짐

```
const obj = {
  a: 1,
  b: {
    c: 2,
  },
};

const newObj = JSON.parse(JSON.stringify(obj));

newObj.b.c = 3;

console.log(obj);
console.log(obj.b.c === newObj.b.c);
```

- 단점으로는 처리 속도가 느리며 Object가 function일 경우 undefined로 처리한다는 점이 존재

```
const obj = {
  a: 1,
  b: {
    c: 2,
  },
  func: function() {
    return this.a;
  }
};

const newObj = JSON.parse(JSON.stringify(obj));

console.log(newObj.func);
```

커스텀 재귀 함수를 구현한 복사

> 💡 아래 함수를 작동 시켜보세요.

```
function deepCopy(obj) {
  if (obj === null || typeof obj !== "object") {
    return obj;
  }

  let copy = {};
  for (let key in obj) {
    copy[key] = deepCopy(obj[key]);
  }
  return copy;
}

const obj = {
  a: 1,
  b: {
    c: 2,
  },
  func: function () {
    return this.a;
  },
};

const newObj = deepCopy(obj);

newObj.b.c = 3;
```

```
console.log(obj);
console.log(obj.b.c === newObj.b.c);
```

Lodash cloneDeep 메서드 사용

💡 Lodash 라이브러리에 대해서 조사하고 cloneDeep 메서드도 조사해보세요.

```
// & npm i lodash 으로 설치
const lodash = require("lodash");

const obj = {
  a: 1,
  b: {
    c: 2,
  },
  func: function () {
    return this.a;
  },
};

const newObj = lodash.cloneDeep(obj);

newObj.b.c = 3;
console.log(obj); // { a: 1, b: { c: 2 }, func: [Function: func] }
console.log(obj.b.c === newObj.b.c); // false
```

💡 소개한 방법 이외에도 깊은복사, 얕은복사 방법을 찾아보세요
가능하다면 언어 별로 정리해도 좋고, JS만 하셔도 좋고, 단일 언어 하나만 하셔도 좋습니다.