

자바스크립트로 살펴보는 얕은 복사와 깊은 복사

GDSC Soongsil Member
Server / Cloud 정명준





얕은 복사와 깊은 복사?

많은 분들이 경험해본 내용을 주제로 삼은 이유



얕은 복사와 깊은 복사?

많은 분들이 경험해본 내용을 주제로 삼은 이유



나는 정말 자바스크립트 복사에 대해 잘 알고 있는가?

저는 아니었습니다



Google

자바스크립트 얕은복사 깊은 복사

전체 이미지 동영상 뉴스 지도 더보기 도구

검색결과 약 21,700개 (0.44초)

<https://velog.io> > Javascript-얕은-복사-깊은-복사

[JavaScript] 얕은 복사, 깊은 복사 - velog

2020. 2. 7. — 자바스크립트에서 값은 원시값과 참조값으로 나뉜다. 원시값 Number String Boolean Null Undefined 참조값 Object Symbol 원시값은 값을 복사 할 때 ...

참조값 · Object.assign() · 전개연산자

이 페이지를 22. 3. 28에 방문했습니다.

<https://velog.io> > JavaScript-얕은-복사Shallow-Copy와...

[JavaScript] 얕은 복사(Shallow Copy)와 깊은 복사(Deep Copy)

2020. 12. 14. — 깊은 복사와 얕은 복사에 대해 알아보겠다. 이 글의 초반 내용은 이전 포스팅의 (원시 타입과 참조 타입의 차이과 맥락이 비슷하며, 위 포스팅은 원시 ...

<https://hanamon.kr> > javascript-shallow-copy-deep-copy

[JavaScript] 얕은 복사(shallow copy) vs 깊은 복사(deep copy)

2021. 6. 11. — 위의 예시처럼 객체를 직접 대입하는 경우 참조에 의한 할당이 이루어지므로 둘은 같은 데이터(주소)를 가지고 있다. 이것이 얕은 복사이다. const obj1 = ...

<https://bbangson.tistory.com> > ...

[JavaScript] 깊은 복사, 얕은 복사 - 뽕슨 개발 블로그 - 티스토리

2021. 7. 12. — 결론부터 말하자면 얕은 복사는 객체의 참조값(주소 값)을 복사하고, 깊은 복사는 객체의 실제 값을 복사합니다. 먼저, 자바스크립트에서 값은 원시값 ...

함께 검색한 항목

- jquery 깊은복사
- JavaScript Deep Copy
- C 얕은 복사 깊은 복사
- Javascript shallow copy, Deep copy
- 얕은 복사, 깊은 복사 자바
- 스프레드 연산자 깊은 복사

구글 1페이지에 나오는 결과가 제대로 된 방법이 아닐 수도 있습니다



본론에 들어가기 전에

자바스크립트가 변수를 어떻게 관리하는지를 알아야
얕은 복사와 깊은 복사를 이해 할 수 있다.



소스코드

```
let a = 12;
```

메모리 주소는 모두 개념적 표현

변수 영역	주소	...	1000	1001	1002	1003	...
	데이터			식별자 : a 값 : @3001			
데이터 영역	주소	...	3000	3001	3002	3003	...
	데이터			12			

변수 영역에 값을 직접 대입하지 않고 데이터 영역을 새롭게 구성하는 이유

- 자바스크립트는 숫자형은 8바이트, 문자형은 가변적인 메모리 공간을 구성한다.
- 확보된 공간을 가변적으로 늘린다면 중간에 있는 데이터를 늘리기 위해 모든 데이터를 이동 시키는 비효율적인 작업



소스코드

```
let a = 12;  
a = 'hello~';
```

메모리 주소는 모두 개념적 표현

변수 영역	주소	...	1000	1001	1002	1003	...
	데이터			식별자 : a 값 : @3002			
데이터 영역	주소	...	3000	3001	3002	3003	...
	데이터			12	'hello~'		

@3001에 있는 값을 바꾸지 않고 @3002라는 새로운 공간에 데이터를 할당한다
식별자 a는 참조하는 주소만 @3001에서 @3002로 바뀐다.

@3001는 더 이상 식별자에게 가리키는 주소가 아니기 때문에 GC를 통해 제거 🙄

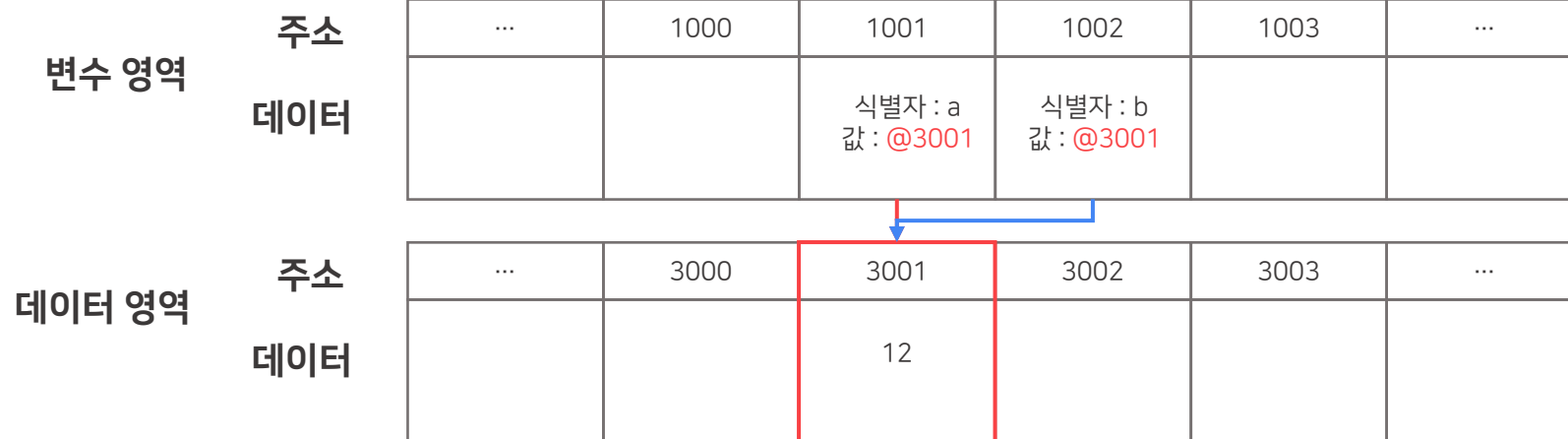


소스코드

```
let a = 12;  
let b = a;
```

```
b = 33;
```

메모리 주소는 모두 개념적 표현



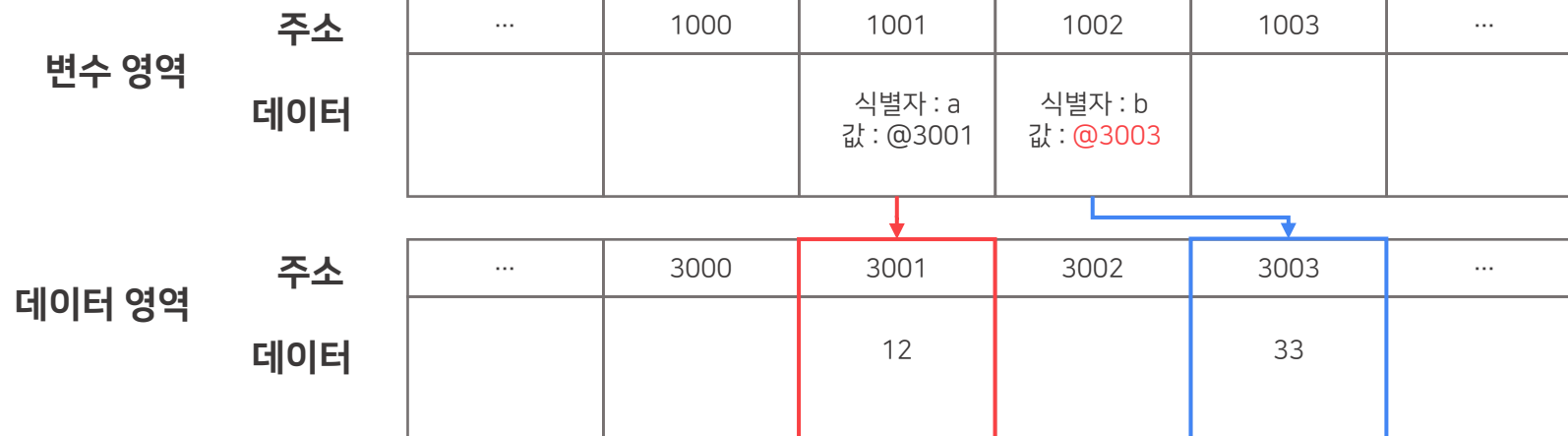
식별자 b에 a를 대입하였기 때문에 동일한 주소값 (= 데이터 영역) 을 가지게 된다.



소스코드

```
let a = 12;  
let b = a;  
  
b = 33;
```

메모리 주소는 모두 개념적 표현



A와 b는 동일한 주소 값을 바라보고 있었지만, b가 33이라는 값을 할당 받았기 때문에 새로운 데이터 영역의 주소를 받아오게 되면서 a의 데이터 영역 주소 값 연결이 끊어진다.



여기까지 참 쉽죠?



여기서 생기는 의문

그럼 복사가 잘 되는 거 아니가요?



여기서 생기는 의문

~~그럼 복사가 잘 되는 거 아닌가요?~~



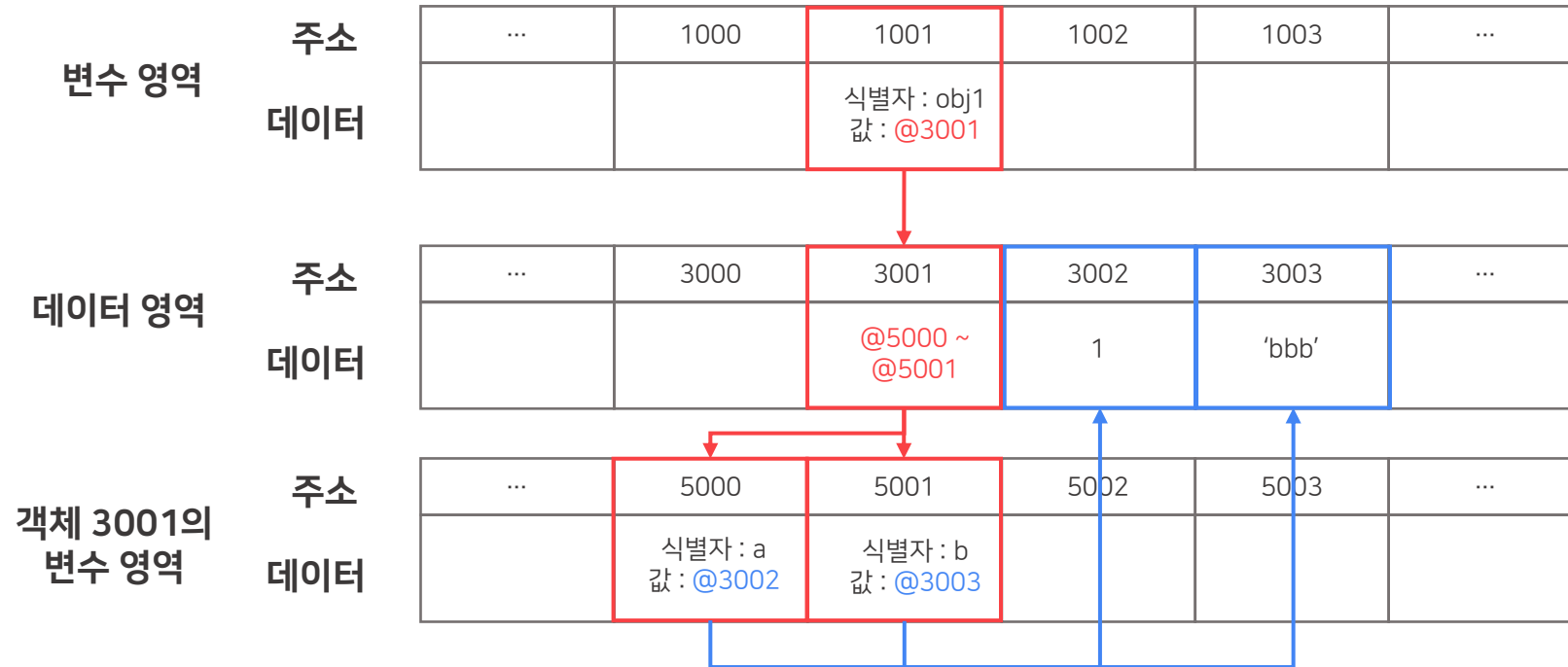
Number, String, Boolean, null, undefined, Symbol은
기본형 데이터로 한 번 할당했다면 지워지기 전까지 값이 바뀌지 않습니다.
(불변형)

저희 문제는 가변형에서 발생 합니다



소스코드

```
let obj1 = {  
  a: 1,  
  b: 'bbb'  
};
```

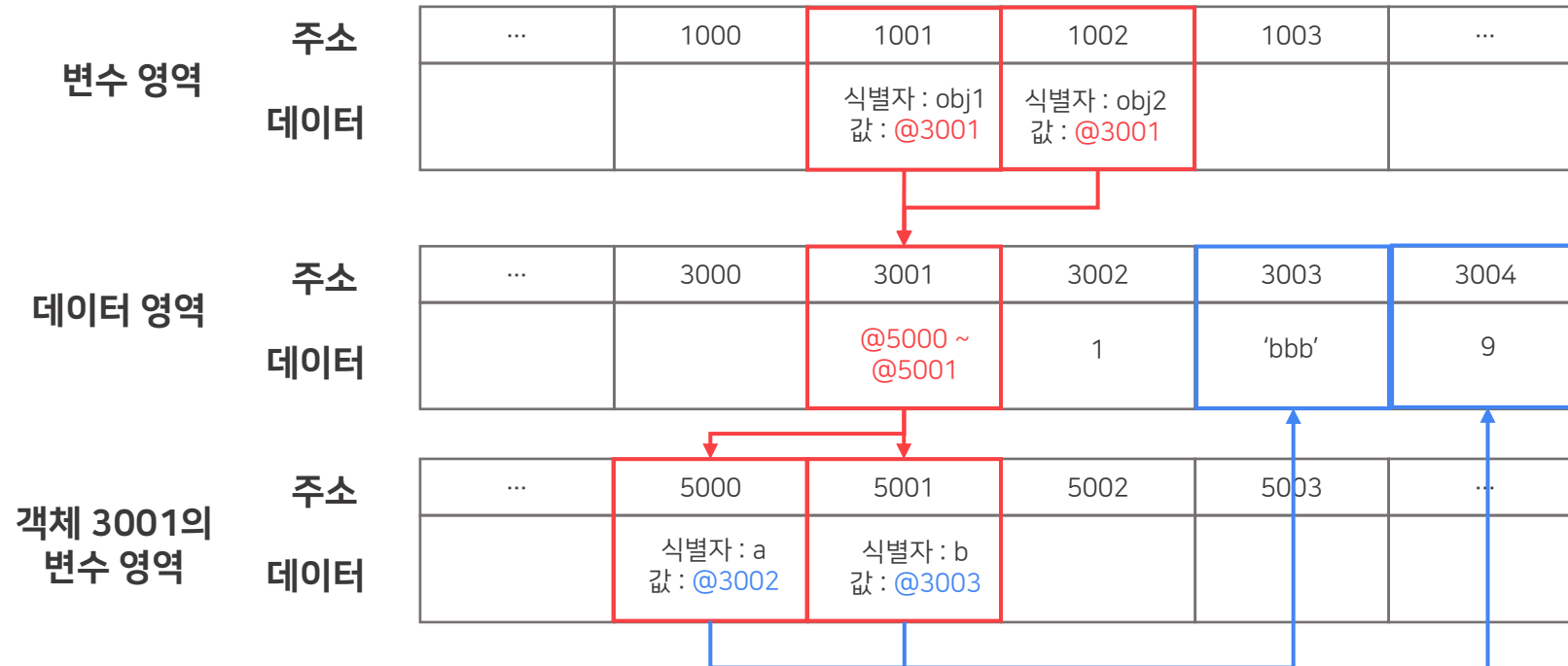


객체의 변수 영역 (프로퍼티) 영역이 별도로 존재하고 프로퍼티는 다른 값으로 변경이 가능하다.



소스코드

```
let obj1 = {  
  a : 1,  
  b : 'bbb'  
};  
  
let obj2 = obj1;  
obj2.a = 9;
```



복사를 해도 실제 데이터 영역이 바뀌는 것이 아니고 프로퍼티의 영역이 바뀌기 때문에 원하는 복사가 이루어지지 않는다.



지금까지 여정

```
> let obj1 = {  
    a: 1,  
    b: 'bbb'  
};  
let obj2 = obj1;  
  
obj2.a = 9  
  
console.log(obj1);  
console.log(obj2);
```

결과

```
▶ {a: 9, b: 'bbb'}  
▶ {a: 9, b: 'bbb'}
```

복사(?)를 했지만 제대로 복사가 되지 않는 것을 확인했다.

이것은 “얕은 복사”로 바로 1deep 주소 값만 복사되어 해당 프로퍼티에 대한 원본과 사본이 동일한 참조형 데이터를 바라본다.

> 불변형과 다르게 가변형은 사본을 바꿔도 원본이 바뀌고 원본을 바꾸면 사본도 바뀐다.



Deep Copy의 시도

그럼 어떻게 해야 깊은 복사를 할 수 있나요?



방법1 Object의 경우 function을 새롭게 만들어서 return

```
let user = {
  name: 'myungJun',
  gender: 'male'
};

const changeName = (user, newName) => {
  return {
    name: newName,
    gender: user.gender
  };
}

const user2 = changeName(user, 'soongsil');
console.log(user.name, user2.name); // myungJun soongsil
```

function Return을 해서 새로운 값을 만드는 형태로 복사를 해냈지만
깊어질수록 프로퍼티를 모두 개발자가 손수 작성 해줘야 한다.



방법1 Object의 경우 function을 새롭게 만들어서 return

```
let user = {  
  name: 'myungJun',  
  gender: 'male'  
};  
  
const changeName = (user, newName) => {  
  return {  
    name: newName,  
    gender: user.gender  
  };  
}  
  
const user2 = changeName(user, 'soongsil');  
console.log(user.name, user2.name); // myungJun soongsil
```

function Return을 해서 새로운 값을 만드는 형태로 복사를 해냈지만
깊어질수록 프로퍼티를 모두 개발자가 손수 작성 해줘야 한다.



방법2 Array에서 Slice 사용하기

```
const arr = [1, 2, 3];  
const copied = arr.slice();  
  
arr[0] = 2;  
copied[0] = 99;  
  
console.log(arr); // [2, 2, 3]  
console.log(copied); // [99, 2, 3]
```

▶ (3) [2, 2, 3]

▶ (3) [99, 2, 3]

Array를 복사하려고 할 때 코드에서 주로 볼 수 있는 방법
정상적으로 복사가 되는 것처럼 보인다.



방법2 Array에서 Slice 사용하기

```
const arr = [[22,33], 2, 3];  
const copied = arr.slice();  
  
copied[0][1] = 2;  
  
console.log(arr);  
console.log(copied);
```

→ ▼ (3) [Array(2), 2, 3] ⓘ
 ▶ 0: (2) [22, 2]
 1: 2
 2: 3
 length: 3
 ▶ [[Prototype]]: Array(0)

→ ▼ (3) [Array(2), 2, 3] ⓘ
 ▶ 0: (2) [22, 2]
 1: 2
 2: 3
 length: 3
 ▶ [[Prototype]]: Array(0)

2Deep가 들어가자마자 복사가 제대로 안되는 모습을 보여준다.
Slice는 1deep만 순차적으로 복사하기 때문에 독립적으로 복사되지 않는다.

원하는 방법 아님



여기서 생기는 의문2

JSON.parse 또는 재귀함수 function 만들면 되잖아요



여기서 생기는 의문2

JSON.parse 또는 재귀함수 function 만들면 되잖아요



많은 책과 구글 검색에서 나오는 해답... 으로 알려져 있다



방법3 JSON.parse & JSON.stringify

```
const test1 = {  
  intTest: 1,  
  arrayTest: [111, 222, 333],  
  deep1: {  
    deep2: {  
      message: '하이 빵가루'  
    }  
  }  
}  
  
const copyTest1 = JSON.parse(JSON.stringify(test1));  
copyTest1.deep1.deep2.message = '안녕~~';
```

```
▼ {intTest: 1, arrayTest: Array(3), deep1: {...}} ⓘ  
  ► arrayTest: (3) [111, 222, 333]  
  ▼ deep1:  
    ▼ deep2:  
      message: "하이 빵가루"  
      ► [[Prototype]]: Object  
      ► [[Prototype]]: Object  
      intTest: 1  
      ► [[Prototype]]: Object  
copyTest1 ---  
▼ {intTest: 1, arrayTest: Array(3), deep1: {...}} ⓘ  
  ► arrayTest: (3) [111, 222, 333]  
  ▼ deep1:  
    ► deep2: {message: '안녕~~'}  
    ► [[Prototype]]: Object  
    intTest: 1  
    ► [[Prototype]]: Object
```

JSON.parse는 Object를 JSON으로 변환하고, 변환한 JSON을 다시 원래 객체로 돌려주는 역할을 담당한다.

이 과정에서 Object는 String으로 변환이 되고, 불변성 데이터 타입으로 전환되기 때문에 모든 체인이 끊기게 되고 다시 JSON.parse를 통해 원본 객체로 돌아온다.

프로퍼티가 깊어도 매우 잘 동작한다



방법3 JSON.parse & JSON.stringify

```
const test1 = {
  intTest: 1,
  arrayTest: [111, 222, 333],
  deep1: {
    deep2: {
      message: '하이 빵가루'
    }
  }
}

const copyTest1 = JSON.parse(JSON.stringify(test1));
copyTest1.deep1.deep2.message = '안녕~~';
```

```
▼ {intTest: 1, arrayTest: Array(3), deep1: {...}} ⓘ
  ► arrayTest: (3) [111, 222, 333]
  ▼ deep1:
    ▼ deep2:
      message: "하이 빵가루"
      ► [[Prototype]]: Object
      ► [[Prototype]]: Object
      intTest: 1
      ► [[Prototype]]: Object
copyTest1 ---
▼ {intTest: 1, arrayTest: Array(3), deep1: {...}} ⓘ
  ► arrayTest: (3) [111, 222, 333]
  ▼ deep1:
    ► deep2: {message: '안녕~~'}
    ► [[Prototype]]: Object
    intTest: 1
    ► [[Prototype]]: Object
```

JSON.parse는 Object를 JSON으로 변환하고, 변환한 JSON을 다시 원래 객체로 돌려주는 역할을 담당한다.

이 과정에서 Object는 String으로 변환이 되고, 불변성 데이터 타입으로 전환되기 때문에 모든 체인이 끊기게 되고 다시 JSON.parse를 통해 원본 객체로 돌아온다.

프로퍼티가 깊어도 매우 잘 동작한다

이것도 사용하면 안되는 방법이다



방법3 JSON.parse & JSON.stringify

왜 쓰면 안되나요?

변환하는 과정은 매우 리소스를 많이 잡아먹는 과정으로 속도에 큰 영향을 미치는 이유도 있지만, 자바스크립트의 객체와 JSON은 완벽한 호환성을 보장하지 않기 때문이다.

ECMA2020에 추가된 BigInt는 아예 컴파일 단계에서 막아버린다.

```
> const test1 = {  
  bigIntTest: BigInt(9007199254740994), // int의 기본 최대값은 900719925474092  
}
```

```
const test1Copy = JSON.parse(JSON.stringify(test1));
```

```
✖ ▶ Uncaught TypeError: Do not know how to serialize a BigInt  
  at JSON.stringify (<anonymous>)  
  at <anonymous>:5:35
```

Function은 복사가 되지 않는다.

```
> const arr = [function() {}, () => {}];  
const copied = JSON.parse(JSON.stringify(arr));  
console.log(arr);  
console.log(copied);
```

```
▼ (2) [f, f] ⓘ  
  ▶ 0: f ()  
  ▶ 1: () => {}  
    length: 2  
  ▶ [[Prototype]]: Array(0)
```

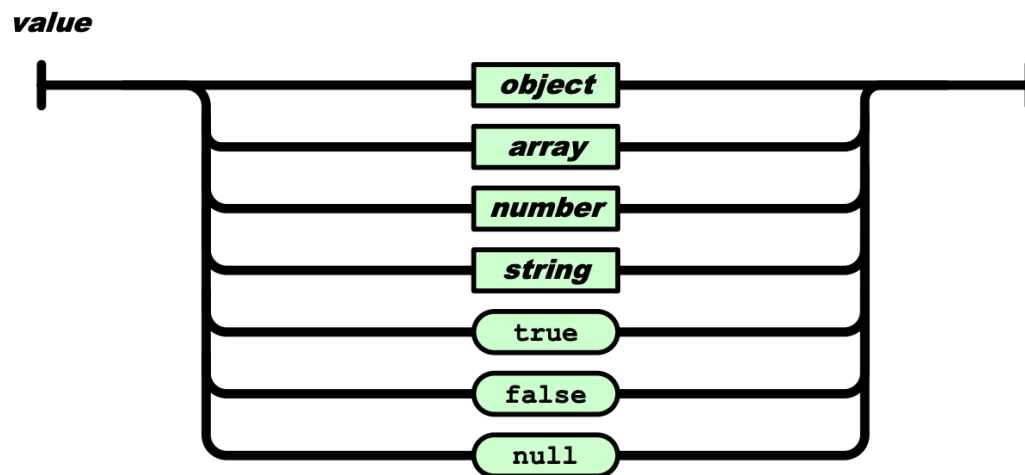
```
▼ (2) [null, null] ⓘ  
  ▶ 0: null  
  ▶ 1: null  
    length: 2  
  ▶ [[Prototype]]: Array(0)
```



방법3 JSON.parse & JSON.stringify

자바스크립트와 JSON의 명세서가 다르다.

자바스크립트의 표준 국제 이름은 ECMAScript-262이고 JSON은 ECMA-404라는 명세서를 가지고 있다.



이 값들만 JSON으로 인정하며 다른 값은 모두 JSON으로 인정하지 않는다.
그렇기에 자바스크립트의 함수 (function)은 JSON으로 사용 될 수 없다.



방법3 JSON.parse & JSON.stringify

더 해보기

```
const test1 = {  
  //bigIntTest: BigInt(9007199254740994), // int의 기본 최대값은 900719925474092  
  intTest: 9007199254740993,  
  date: new Date(),  
  test2: new RegExp(/ab+c/, 'i'),  
  arrayTest: [111, 222, 333]  
}  
  
const copyTest1 = JSON.parse(JSON.stringify(test1));
```

```
▼ {intTest: 9007199254740992, date: Sat Apr 02 2022 01:37:05 GMT+0900 (한국 표준시), test2: /ab+c/i, arrayTest: Array(3)} ⓘ  
  ▶ arrayTest: (3) [111, 222, 333]  
  ▶ date: Sat Apr 02 2022 01:37:05 GMT+0900 (한국 표준시) {}  
  ▶ intTest: 9007199254740992  
  ▶ test2: /ab+c/i  
  ▶ [[Prototype]]: Object
```

getDate: object

copyTest1 ---

```
▼ {intTest: 9007199254740992, date: '2022-04-01T16:37:05.207Z', test2: {...}, arrayTest: Array(3)} ⓘ  
  ▶ arrayTest: (3) [111, 222, 333]  
  ▶ date: "2022-04-01T16:37:05.207Z"  
  ▶ intTest: 9007199254740992  
  ▶ test2: {}  
  ▶ [[Prototype]]: Object
```

getDate: string



방법4 DeepCopy 재귀함수 만들기

```
const deepCopy = (obj) => {  
  const clone = {};  
  for (const key in obj) {  
    if (typeof obj[key] == "object" && obj[key] != null) {  
      clone[key] = deepCopy(obj[key]);  
    } else {  
      clone[key] = obj[key];  
    }  
  }  
  return clone;  
}  
  
const test2 = {  
  bigIntTest: BigInt(9007199254740994),  
  intTest: 9007199254740993  
}
```

JSON.parse에서 발생했던 BigInt 문제도 해결되었고 속도도 이전과 다르게 매우 빠르다.
재귀 함수이기 때문에 프로퍼티가 깊어도 매우 잘 동작한다

```
▼ {bigIntTest: 9007199254740994n, intTest: 9007199254740992} ⓘ  
  bigIntTest: 9007199254740994n  
  intTest: 9007199254740992  
  ► [[Prototype]]: Object
```



방법4 DeepCopy 재귀함수 만들기

```
const deepCopy = (obj) => {  
  const clone = {};  
  for (const key in obj) {  
    if (typeof obj[key] == "object" && obj[key] != null) {  
      clone[key] = deepCopy(obj[key]);  
    } else {  
      clone[key] = obj[key];  
    }  
  }  
  return clone;  
}  
  
const test2 = {  
  bigIntTest: BigInt(9007199254740994),  
  intTest: 9007199254740993,  
  test4: new RegExp(/ab+c/, 'i'),  
  date: new Date(),  
  arrayTest: [111, 222, 333]  
}
```

> deepCopy(test2);

▼ {bigIntTest: 9007199254740994n, intTest: 9007199254740992, test4: {}, date: {}, arrayTest: {}} ⓘ



▶ arrayTest: {0: 111, 1: 222, 2: 333}



▶ bigIntTest: 9007199254740994n



▶ date: {}

▶ intTest: 9007199254740992

▶ test4: {}

▶ [[Prototype]]: Object

BigInt가 해결되었지만 Array는 Object 형태가 되었고 아까 전과 똑같이 정규식과 function도 복사가 되지 않는 모습을 볼 수 있다.



이제 우리는 알 수 있습니다



자바스크립트 얇은복사 깊은 복사



전체 이미지 동영상 뉴스 지도 더보기

도구

검색결과 약 21,700개 (0.44초)

<https://velog.io> > Javascript-얇은-복사-깊은-복사

[JavaScript] 얇은 복사, 깊은 복사 - velog

2020. 2. 7. — 자바스크립트에서 얇은 원시값과 참조값으로 나뉜다. 원시값 Number String Boolean Null Undefined 참조값 Object Symbol 원시값은 얇은 복사 할 때 ...

참조값 · Object.assign() · 전개연산자

이 페이지를 22. 3. 28에 방문했습니다.

<https://velog.io> > JavaScript-얇은-복사Shallow-Copy와...

[JavaScript] 얇은 복사(Shallow Copy)와 깊은 복사(Deep Copy)

2020. 12. 14. — 얇은 복사와 깊은 복사에 대해 알아보겠다. 이 글의 초반 내용은 이전 포스팅의 (원시 타입과 참조 타입의 차이과 맥락이 비슷하며, 위 포스팅은 원시 ...

<https://hanamon.kr> > javascript-shallow-copy-deep-copy

[JavaScript] 얇은 복사(shallow copy) vs 깊은 복사(deep copy)

2021. 6. 11. — 위의 예시처럼 객체를 직접 대입하는 경우 참조에 의한 할당이 이루어지므로 둘은 같은 데이터(주소)를 가지고 있다. 이것이 얇은 복사이다. const obj1 = ...

<https://bbangson.tistory.com> > ...

[JavaScript] 깊은 복사, 얇은 복사 - 뽀슨 개발 블로그 - 티스토리

2021. 7. 12. — 결론부터 말하자면 얇은 복사는 객체의 참조값(주소 값)을 복사하고, 깊은 복사는 객체의 실제 값을 복사합니다. 먼저, 자바스크립트에서 얇은 원시값 ...

함께 검색한 항목

jquery 깊은복사 Javascript shallow copy, Deep copy

JavaScript Deep Copy 얇은 복사, 깊은 복사 자바

C 얇은 복사 깊은 복사 스프레드 연산자 깊은 복사



얇은 복사 Deep Copy

깊은 복사된 객체는 객체안에 객체가 있을 경우에도 원본과의 참조가 완전히 끊어진 객체를 말한다.

1. 재귀함수를 이용한 복사

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

function copyObj(obj) {
  const result = {};

  for (let key in obj) {
    if (typeof obj[key] === 'object') {
      result[key] = copyObj(obj[key]);
    } else {
      result[key] = obj[key];
    }
  }

  return result;
}

const copiedObj = copyObj(obj);
copiedObj.b.c = 3;

obj.b.c === copiedObj.b.c //false
```

2. JSON.stringify()

JSON.stringify()는 객체를 json 문자열로 변환하는데 이과정에서 원본 객체와의 참조가 모두 끊어진다. 객체를 json 문자열로 변환후 JSON.parse()를 이용해 다시 자바스크립트 객체로 만들어주면 깊은 복사가 된다.

하지만 이방법은 사용하기는 쉽지만 다른 방법예비해 아주 느리다고 알려져있다.

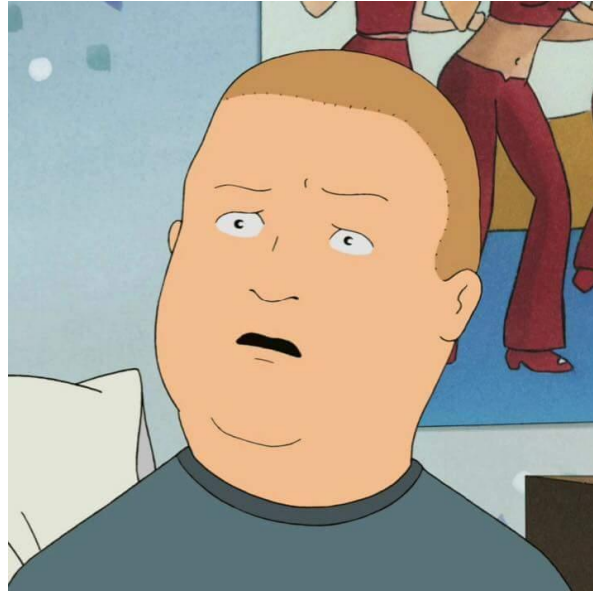
```
const obj = {
  a: 1,
  b: {
    c: 2,
  },
};

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

copiedObj.b.c = 3;

obj.b.c === copiedObj.b.c //false
```

이거 다 제대로 된
깊은 복사 아님




덧) 도대체 그럼 어떻게 깊은 복사를 하는 건데요




lodash

4.17.21 • Public • Published a year ago

 [Readme](#)

 [Explore](#) 

 0 Dependencies

 152,510 Dependents

 114 Versions

lodash v4.17.21

The **Lodash** library exported as **Node.js** modules.

Installation

Using npm:


```
$ npm i -g npm  
$ npm i --save lodash
```

In Node.js:


Install

```
> npm i lodash
```

Repository

 github.com/lodash/lodash

Homepage

 lodash.com/

Weekly Downloads

50,027,873





방법5 Lodash 사용하기

```
const test2 = {  
  bigIntTest: BigInt(9007199254740994),  
  intTest: 9007199254740993,  
  test4: new RegExp(/ab+c/, 'i'),  
  date: new Date(),  
  arrayTest: [111, 222, 333]  
}
```

```
const lodaShCopy = lodash.cloneDeep(test2);  
console.log(lodaShCopy);
```

```
test2 ---  
{  
  bigIntTest: 9007199254740994n,  
  intTest: 9007199254740992,  
  test4: /ab+c/i,  
  date: 2022-04-01T16:55:48.209Z,  
  arrayTest: [ 111, 222, 333 ]  
}
```

원본

```
{  
  bigIntTest: 9007199254740994n,  
  intTest: 9007199254740992,  
  test4: /ab+c/i,  
  date: 2022-04-01T16:55:48.209Z,  
  arrayTest: [ 111, 222, 333 ]  
}
```

복사판

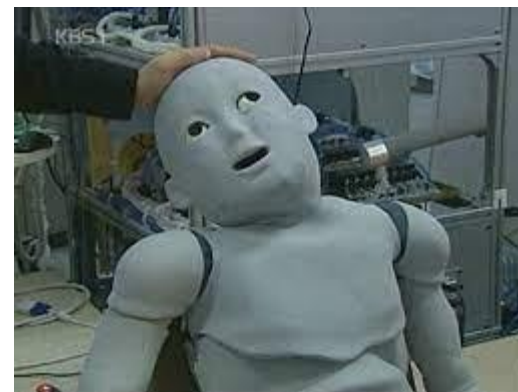
Bigint도 잘 받고 Date도, Array도 정규식도 잘 받는다.



방법5 Lodash 사용하기

이게 어떻게 가능한 걸까?

```
32 const boolTag = '[object Boolean]'  
33 const dateTag = '[object Date]'  
34 const errorTag = '[object Error]'  
35 const mapTag = '[object Map]'  
36 const numberTag = '[object Number]'  
37 const objectTag = '[object Object]'  
38 const regexpTag = '[object RegExp]'  
39 const setTag = '[object Set]'  
40 const stringTag = '[object String]'  
41 const symbolTag = '[object Symbol]'  
42 const weakMapTag = '[object WeakMap]'  
43  
44 const arrayBufferTag = '[object ArrayBuffer]'  
45 const dataViewTag = '[object DataView]'  
46 const float32Tag = '[object Float32Array]'  
47 const float64Tag = '[object Float64Array]'  
48 const int8Tag = '[object Int8Array]'  
49 const int16Tag = '[object Int16Array]'  
50 const int32Tag = '[object Int32Array]'  
51 const uint8Tag = '[object Uint8Array]'  
52 const uint8ClampedTag = '[object Uint8ClampedArray]'  
53 const uint16Tag = '[object Uint16Array]'  
54 const uint32Tag = '[object Uint32Array]'
```



모든 상황에 대한 경우를 다 넣어줬다






마지막으로...

자바스크립트는 그럼 왜 Deep Copy메소드를 지원하지 않나요?


Why Object.assign can't implement deep copy? #1319

Closed godkun opened this issue on 3 Oct 2018 · 1 comment



godkun commented on 3 Oct 2018

Why Object.assign can't implement deep copy, if you can add a flag parameter to achieve deep copy, it will be perfect!!



ljharb commented on 3 Oct 2018

Object.assign is variadic, so there's no place to accept a flag.

See <https://github.com/tc39/ecma262/blob/master/CONTRIBUTING.md> for how to suggest a feature the language, but a deep copy that isn't a structured clone (for which generic algorithms don't yet exist in the language) would be very fragile.

In practice, I've found when i need a deep copy, it's a code smell, and i can achieve a better solution by rearchitecting to not need one - but that's just my personal experience.

👍 1

TC39 멤버의 답변

Object.assign은 가변적이므로 플래그를 사용할 수 없습니다.

<https://github.com/tc39/ecma262/blob/master/CONTRIBUTING.md>를 참조해 주세요. UTING.md에서 언어의 기능을 제안하는 방법에 대해 설명합니다.
단, 구조화 클론이 아닌 딥 카피(일반 알고리즘이 아직 언어에 존재하지 않는 것)는 매우 취약합니다.

실제로 깊이 있는 복사가 필요할 때는 코드 냄새이며, 필요하지 않도록 아키텍처를 수정함으로써 더 나은 솔루션을 얻을 수 있다는 것을 알게 되었습니다. 그것은 제 개인적인 경험일 뿐입니다.

자바스크립트는 언어적으로 **Deep Copy**를 할 수 없도록 모델링 되어 있고 사용자의 **Needs**를 해결해주기 위해 낮은 퍼포먼스를 가지더라도 모든 요소를 복사해줄 수 있는 라이브러리들이 구현되어 있다.

즉, 깊은 복사를 하기 전에는 다른 효율적인 방법이 있는지 찾아보는게 중요하다.



감사합니다.