**深克隆**

**定义**

浅拷贝：创建一个新对象，这个对象有着原始对象属性值的一份精确拷贝。如果属性是基本类型，拷贝的就是基本类型的值，如果属性是引用类型，拷贝的就是内存地址 ，所以如果其中一个对象改变了这个地址，就会影响到另一个对象。

深拷贝：

将一个对象从内存中完整的拷贝一份出来,从堆内存中开辟一个新的区域存放新对象,且修改新对象不会影响原对象

浅克隆

1. Object.assign(target, ...sources)
2. let target = {};
3. let source = { a: { b: 2 } };
4. Object.assign(target, source);
5. target.a===source.a //true

**判断类型**

**typeof**

1. bigint、undefined、function、object、number、boolean、symbol、string

**instanceof**

原理就是只要右边变量的 prototype 在左边变量的原型链上即可。

**Object.prototype.toString**

ES5 规范地址：https://es5.github.io/#x15.2.4.2。

**丐中丐：业务最实用**

1. let co = JSON.parse(JSON.stringify(o));

正则、拷贝函数、循环引用等

测试用例

1. function Obj() {
2. this.func = function () {
3. alert(1)
4. };
5. this.obj = {a:1};
6. this.arr = [1,2,3];
7. this.und = undefined;
8. this.reg = /123/;
9. this.date = new Date(0);
10. this.NaN = NaN
11. this.infinity = Infinity
12. this.sym = Symbol(1)
13. this.set = new Set([1,2,3])
14. this.map = new Map([['a',1],['b',9]])
15. }
16. let obj1 = new Obj();
17. Object.defineProperty(obj1,'innumerable',{
18. enumerable:false,
19. value:'innumerable'
20. })
21. console.log({obj1});
22. let str = JSON.stringify(obj1);
23. let obj2 = JSON.parse(str);
24. console.log({obj2});

**基础版本**

1. function deepCopy(obj) {
2. if(typeof obj === "object") {
3. if(obj.constructor === Array) {
4. var newArr = []
5. for(var i = 0; i < obj.length; i++) newArr.push(obj[i])
6. return newArr
7. } else {
8. var newObj = {}
9. for(var key in obj) {
10. newObj[key] = this.deepCopy(obj[key])
11. }
12. return newObj
13. }
14. } else {
15. return obj
16. }
17. }

**复杂版本**

<https://github.com/lodash/lodash/blob/master/.internal/baseClone.js>

正确但无面试意义版本



**面试终极武器**

1. const isComplexDataType = obj => (typeof obj === 'object' || typeof obj === 'function') && (obj !== null)
3. const deepClone = function (obj, hash = new WeakMap()) {
4. if (hash.has(obj)) return hash.get(obj)
5. let type = [Date,RegExp,Set,Map,WeakMap,WeakSet]
6. if (type.includes(obj.constructor)) return new obj.constructor(obj);
7. //如果成环了,参数obj = obj.loop = 最初的obj 会在WeakMap中找到第一次放入的obj提前返回第一次放入WeakMap的cloneObj
9. let allDesc = Object.getOwnPropertyDescriptors(obj); //遍历传入参数所有键的特性
10. let cloneObj = Object.create(Object.getPrototypeOf(obj), allDesc); //继承原型
11. hash.set(obj, cloneObj)
13. for (let key of Reflect.ownKeys(obj)) { //Reflect.ownKeys(obj)可以拷贝不可枚举属性和符号类型
14. // 如果值是引用类型(非函数)则递归调用deepClone
15. cloneObj[key] =
16. (isComplexDataType(obj[key]) && typeof obj[key] !== 'function') ?
17. deepClone(obj[key], hash) : obj[key];
18. }
19. return cloneObj;
20. };
22. let obj = {
23. bigInt: BigInt(12312),
24. set:new Set([2]),
25. map:new Map([['a',22],['b',33]]),
26. num: 0,
27. str: '',
28. boolean: true,
29. unf: undefined,
30. nul: null,
31. obj: {
32. name: '我是一个对象',
33. id: 1
34. },
35. arr: [0, 1, 2],
36. func: function () {
37. console.log('我是一个函数')
38. },
39. date: new Date(0),
40. reg: new RegExp('/我是一个正则/ig'),
41. [Symbol('1')]: 1,
42. };
44. Object.defineProperty(obj, 'innumerable', {
45. enumerable: false,
46. value: '不可枚举属性'
47. });
49. obj = Object.create(obj, Object.getOwnPropertyDescriptors(obj))
51. obj.loop = obj
53. let cloneObj = deepClone(obj);
55. console.log('obj', obj);
56. console.log('cloneObj', cloneObj);
58. for (let key of Object.keys(cloneObj)) {
59. if (typeof cloneObj[key] === 'object' || typeof cloneObj[key] === 'function') {
60. console.log(`${key}相同吗？ `, cloneObj[key] === obj[key])
61. }
62. }