



深浅拷贝

1. 浅拷贝和深拷贝的引用赋值的关系

1.1 浅拷贝

声明一个对象

```
1  const info={  
2      name:'ava',  
3      age:18,  
4      address:'aaa',  
5      friend:{  
6          name:'blob',  
7          age:22,  
8          address:'bbb',  
9      }  
10 }
```

1.1.1 引用赋值

```
1  const obj1=info
```

1.1.2 展开语法

```
1 const obj2 = {...info}
```

1.1.3 Object.assign()

```
1 const obj3=Object.assign({},info)
```



浅拷贝的时，会对原始数据类型进行复制，拷贝后的原始数据类型是单独了，但是引用数据类型，只是复制的内存地址，修改数据，会影响到原有的对象的数据，为了解决这个问题，提出来深拷贝

1.2 深拷贝

1.2.1 JSON方法

```
1 const obj4 = JSON.parse(Json.stringify(info))
```

转为字符串后，解析为对象



当对象中存在函数、symbol等等属性的时，JSON无法复制，会自动跳过，且内部存在循环引用的时，也无法赋值，需要自定义深拷贝函数

2. 自定义深拷贝函数

```
1 const info={
2   name:'方佳怡\n',
3   phone:'19594736068',
4   age:18,
5   email:'cee6@9o7ft91.cn',
6   address: {
7     province:'黑龙江省',
8     city:'鹤岗市',
9     county:'萝北县',
```

```

10     town: '肇兴镇'
11   },
12   friend: {
13     name: '黎慧嘉',
14     phone: '14558873098',
15     age: 18,
16     email: 'd904@8odmq.cn',
17     address: {
18       province: '云南省',
19       city: '昆明市',
20       county: '石林彝族自治县',
21       town: '大可乡'
22     },
23   }
24 }

```

2.1 一般情况

2.1.1 基础架构

```

1  const deepCopy=(originValue)=>{
2    // 需要一个完全的新对象
3    const newObj={}
4    // 对传入的对象进行遍历
5    for (const Key in originValue) {
6      //当是普通值, 就直接引用赋值, 当时对象, 就进行递归调用赋值
7      newObj[Key] = originValue[Key]
8    }
9    return newObj
10 }

```

2.1.2 考虑到非对象传入

```

1  //判断一个标识符是否是对象
2  const isObject = (value) => {
3    //当null、object、array 都是显示为对象、function显示为函数
4    const valueType = typeof value
5    return (value!==null) && (valueType==="object" || valueType==="function")
6  }
7  }
8  const deepCopy=(originValue)=>{
9    // 如果是对象传入, 需要一个完全的新对象, 不是对象, 则返回原数据
10   if (!isObject(originValue)){

```

```

11     return originValue
12 }
13 const newObj={}
14 // 对传入的对象进行遍历
15 for (const Key in originValue) {
16     //当是普通值，就直接引用赋值，当时对象，就进行递归调用赋值
17     newObj[Key] = deepCopy(originValue[Key])
18 }
19 return newObj
20 }

```

2.1.3 数组深拷贝

当是数组是[] 当是对象时{}

```

1  const info=[
2      {name:'方语晨', phone:'13150248220'},
3      {name:'方慧嘉', phone:'15363311534'},
4      {name:'熊淑华', phone:'15595869397'},
5      {name:'李雨泽', phone:'13942751389'},
6      {name:'蒋嘉乐', phone:'18837108834'}
7  ]
8
9  //判断一个标识符是否是对象
10 const isObject = (value) => {
11     //当null、object、array 都是显示为对象、function显示为函数
12     const valueType = typeof value
13     return (value!==null) && (valueType==="object" || valueType==="function")
14 }
15 }
16 const deepCopy=(originValue)=>{
17     // 如果是对象传入，需要一个完全的新对象，不是对象，则返回原数据
18     if (!isObject(originValue)){
19         return originValue
20     }
21     //当是数组是[] 当是对象时{}
22     const newObj=Array.isArray(originValue)? [] : {}
23     // 对传入的对象进行遍历
24     for (const Key in originValue) {
25         //当是普通值，就直接引用赋值，当时对象，就进行递归调用赋值
26         newObj[Key] = deepCopy(originValue[Key])
27     }
28     return newObj
29 }
30 const newObj = deepCopy(info)
31 console.log(newObj)

```

2.1.4 深拷贝其他类型

2.1.4.1 Set

```
1    //当前值是set时
2    if (originValue instanceof Set){
3        const newSet = new Set()
4        for (const setItem of originValue) {
5            newSet.add(deepCopy(setItem))
6        }
7        return newSet
8    }
```

2.1.4.2 Function

函数类型不需要深拷贝

2.1.4.3 值是Symbol

```
1 //当前是symbol 需要创建一个新的symbol
2 if (typeof originValue==='symbol'){
3     return Symbol(originValue.description)
4 }
```

2.1.4.4 key是Symbol

```
1 const symbolkeys = Object.getOwnPropertyDescriptors(originValue)
2 for (const symbolkey in symbolkeys) {
3     newObj[Symbol(symbolkeys.description)]=deepCopy(originValue[symbolkey])
4 }
5 单独循环
```

2.1.4.5 总和

```
1 const set = new
  Set(["0DJ","8eC","N85","ALK","Cb0","QZ7","QsL","ghC","L6v","WdR"]);
2 const s1 = Symbol()
3 const info={
4     name:'方佳怡',
```

```
5     phone: '19594736068',
6     age: 18,
7     set: set,
8     [s1]: 123,
9     symbolKey: Symbol(),
10    email: 'cee6@9o7ft91.cn',
11    address: {
12        province: '黑龙江省',
13        city: '鹤岗市',
14        county: '萝北县',
15        town: '肇兴镇'
16    },
17 }
18
19 //判断一个标识符是否是对象
20 const isObject = (value) => {
21     //当null、object、array 都是显示为对象、function显示为函数
22     const valueType = typeof value
23     return (value !== null) && (valueType === "object" || valueType === "function")
24 }
25 }
26 const deepCopy = (originValue) => {
27     //当前是symbol 需要创建一个新的symbol
28     if (typeof originValue === 'symbol') {
29         return Symbol(originValue.description)
30     }
31     // 如果是对象传入，需要一个完全的新对象，不是对象，则返回原数据
32     if (!isObject(originValue)) {
33         return originValue
34     }
35     //当前值是set时
36     if (originValue instanceof Set) {
37         const newSet = new Set()
38         for (const setItem of originValue) {
39             newSet.add(deepCopy(setItem))
40         }
41         return newSet
42     }
43
44
45     //当是数组是[] 当是对象时{}
46     const newObj = Array.isArray(originValue) ? [] : {}
47     // 对传入的对象进行遍历
48     for (const Key in originValue) {
49         //当是普通值，就直接引用赋值，当时对象，就进行递归调用赋值
50         newObj[Key] = deepCopy(originValue[Key])
51     }
```

```

52     const symbolkeys = Object.getOwnPropertyDescriptor(originValue)
53     for (const symbolkey in symbolkeys) {
54         newObj[Symbol(symbolkeys.description)]=deepCopy(originValue[symbolkey])
55     }
56     return newObj
57 }
58 const newObj = deepCopy(info)
59 console.log(newObj)

```

2.1.5 深拷贝循环引用

将每一次的拷贝的值存入map，并且每一次判断是否已经存在，有就直接返回，同时为了考虑垃圾回收，需要使用weakmap

```

1  const set = new
    Set(["0DJ","8eC","N85","ALK","Cb0","QZ7","QsL","ghC","L6v","WdR"]);
2  const s1 = Symbol()
3  const info={
4      name:'方佳怡',
5      phone:'19594736068',
6      age:18,
7      set:set,
8      [s1]:123,
9      symbolKey:Symbol(),
10     email:'cee6@9o7ft91.cn',
11     address: {
12         province:'黑龙江省',
13         city:'鹤岗市',
14         county:'萝北县',
15         town:'肇兴镇'
16     },
17 }
18 info.self=info
19 //判断一个标识符是否是对象
20 const isObject = (value) => {
21     //当null、object、array 都是显示为对象、function显示为函数
22     const valueType = typeof value
23     return (value!==null) && (valueType==="object"||valueType==="function")
24 }
25 }
26 const deepCopy=(originValue,map=new WeakMap())=>{
27     //当前是symbol 需要创建一个新的symbol
28     if (typeof originValue==='symbol'){
29         return Symbol(originValue.description)
30     }

```

```

31 // 如果是对象传入，需要一个完全的新对象，不是对象，则返回原数据
32 if (!isObject(originValue)){
33     return originValue
34 }
35 //当前值是set时
36 if (originValue instanceof Set){
37     const newSet = new Set()
38     for (const setItem of originValue) {
39         newSet.add(deepCopy(setItem))
40     }
41     return newSet
42 }
43 //当是数组是[] 当是对象时{}
44 if (map.get(originValue)){
45     return map.get(originValue)
46 }
47 const newObj=Array.isArray(originValue)? [] : {}
48 map.set(originValue,newObj)
49 // 对传入的对象进行遍历
50 for (const Key in originValue) {
51     //当是普通值，就直接引用赋值，当时对象，就进行递归调用赋值
52     newObj[Key] = deepCopy(originValue[Key],map)
53 }
54 //symbol循环引用
55 const symbolkeys = Object.getOwnPropertyDescriptors(originValue)
56 for (const symbolkey in symbolkeys) {
57     newObj[Symbol(symbolkeys.description)]=deepCopy(originValue[symbolkey],map)
58 }
59 return newObj
60 }
61 const newObj = deepCopy(info)
62 console.log(newObj)

```