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

```
const info={
    name:'ava',
    age:18,
    address:'aaa',
    friend:{
        name:'blob',
        age:22,
        address:'bbb',
    }
}
```

1.1.1 引用赋值

```
const obj1=info
```

1.1.2 展开语法

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

1.1.3 Object.assign()

```
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

```
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

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

```
        town:'肇兴镇'
    },
    friend:{
        name:'黎慧嘉',
        phone:'14558873098',
        age:18,
        email:'d904@8odmq.cn',
        address: {
            province:'云南省',
            city:'昆明市',
            county:'石林彝族自治县',
            town:'大可乡'
        },
    }
}
```

2.1 一般情况

2.1.1 基础架构

```
const deepCopy=(originValue)=>{
    // 需要一个完全的新对象
    const newObj={}
    // 对传入的对象进行遍历
    for (const Key in originValue) {
        //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
        newObj[Key] = originValue[Key]
    }
    return newObj
}
```

2.1.2 考虑到非对象传入

```
//判断一个标识符是否是对象
const isObject = (value) => {
    //当null、object、array 都是显示为对象、function显示为函数
    const valueType = typeof value
    return (value!==null) && (valueType==="object"||valueType==="function")

}
const deepCopy=(originValue)=>{
    // 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
    if (!isObject(originValue)){
```

```
        return originValue
    }
    const newObj={}
    // 对传入的对象进行遍历
    for (const Key in originValue) {
        //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
        newObj[Key] = deepCopy(originValue[Key])
    }
    return newObj
}
```

2.1.3 数组深拷贝

当是数组是[] 当是对象时{}

```
const info=[
    {name:'方语晨', phone:'13150248220'},
    {name:'方慧嘉', phone:'15363311534'},
    {name:'熊淑华', phone:'15595869397'},
    {name:'李雨泽', phone:'13942751389'},
    {name:'蒋嘉乐', phone:'18837108834'}
]

//判断一个标识符是否是对象
const isObject = (value) => {
    //当null、object、array 都是显示为对象、function显示为函数
    const valueType = typeof value
    return (value!==null) && (valueType==="object"||valueType==="function")

}
const deepCopy=(originValue)=>{
    // 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
    if (!isObject(originValue)){
        return originValue
    }
    //当是数组是[] 当是对象时{}
    const newObj=Array.isArray(originValue)? [] : {}
    // 对传入的对象进行遍历
    for (const Key in originValue) {
        //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
        newObj[Key] = deepCopy(originValue[Key])
    }
    return newObj
}
const newObj = deepCopy(info)
console.log(newObj)
```

2.1.4 深拷贝其他类型

2.1.4.1 Set

```
    //当前值是set时
    if (originValue instanceof Set){
        const newSet = new Set()
        for (const setItem of originValue) {
            newSet.add(deepCopy(setItem))
        }
        return newSet
    }
```

2.1.4.2 Function

函数类型不需要深拷贝

2.1.4.3 值是Symbol

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

2.1.4.4 key是Symbol

```
const symbolkeys = Object.getOwnPropertyDescriptor(originValue)
for (const symbolkey in symbolkeys) {
    newObj[Symbol(symbolkeys.description)]=deepCopy(originValue[symbolkey])
}
单独循环
```

2.1.4.5 总和

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

```
    phone:'19594736068',
    age:18,
    set:set,
    [s1]:123,
    symbolKey:Symbol(),
    email:'cee6@9o7ft91.cn',
    address: {
        province:'黑龙江省',
        city:'鹤岗市',
        county:'萝北县',
        town:'肇兴镇'
    },
}

//判断一个标识符是否是对象
const isObject = (value) => {
    //当null、object、array 都是显示为对象、function显示为函数
    const valueType = typeof value
    return (value!==null) && (valueType==="object"||valueType==="function")

}
const deepCopy=(originValue)=>{
    //当前是symbol 需要创建一个新的symbol
    if (typeof originValue==='symbol'){
        return Symbol(originValue.description)
    }
    // 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
    if (!isObject(originValue)){
        return originValue
    }
    //当前值是set时
    if (originValue instanceof Set){
        const newSet = new Set()
        for (const setItem of originValue) {
            newSet.add(deepCopy(setItem))
        }
        return newSet
    }


    //当是数组是[] 当是对象时{}
    const newObj=Array.isArray(originValue)? [] : {}
    // 对传入的对象进行遍历
    for (const Key in originValue) {
        //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
        newObj[Key] = deepCopy(originValue[Key])
    }
```

```
    const symbolkeys = Object.getOwnPropertyDescriptor(originValue)
    for (const symbolkey in symbolkeys) {
        newObj[Symbol(symbolkeys.description)]=deepCopy(originValue[symbolkey])
    }
    return newObj
}
const newObj = deepCopy(info)
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

```
const set = new Set(["0DJ","8eC","N85","ALK","CbO","QZ7","QsL","ghC","L6v","WdR"]);
const s1 = Symbol()
const info={
    name:'方佳怡',
    phone:'19594736068',
    age:18,
    set:set,
    [s1]:123,
    symbolKey:Symbol(),
    email:'cee6@9o7ft91.cn',
    address: {
        province:'黑龙江省',
        city:'鹤岗市',
        county:'萝北县',
        town:'肇兴镇'
    },
}
info.self=info
//判断一个标识符是否是对象
const isObject = (value) => {
    //当null、object、array 都是显示为对象、function显示为函数
    const valueType = typeof value
    return (value!==null) && (valueType==="object"||valueType==="function")

}
const deepCopy=(originValue,map=new WeakMap())=>{
    //当前是symbol 需要创建一个新的symbol
    if (typeof originValue==='symbol'){
        return Symbol(originValue.description)
    }
```

```
    // 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
    if (!isObject(originValue)){
        return originValue
    }
    //当前值是set时
    if (originValue instanceof Set){
        const newSet = new Set()
        for (const setItem of originValue) {
            newSet.add(deepCopy(setItem))
        }
        return newSet
    }
    //当是数组是[] 当是对象时{}
    if (map.get(originValue)){
        return map.get(originValue)
    }
    const newObj=Array.isArray(originValue)? [] : {}
    map.set(originValue,newObj)
    // 对传入的对象进行遍历
    for (const Key in originValue) {
        //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
        newObj[Key] = deepCopy(originValue[Key],map)
    }
    //symbol循环引用
    const symbolkeys = Object.getOwnPropertyDescriptor(originValue)
    for (const symbolkey in symbolkeys) {

newObj[Symbol(symbolkeys.description)]=deepCopy(originValue[symbolkey],map)
    }
    return newObj
}
const newObj = deepCopy(info)
console.log(newObj)
```