

```
1 var arr = [function(){
2     console.log(a)
3 }, {
4     b: function(){
5         console.log(b)
6     }
7 }]
8
9 var new_arr = JSON.parse(JSON.stringify(arr));
10
11 console.log(new_arr);
```

## 浅拷贝

## 赋值

```
1  const employeeDetailsOriginal = { ...
2    name: '前端小智',
3    age: 18,
4    Profession: '前端开发'
5  };
6
7  const newEm = employeeDetailsOriginal;
8  delete employeeDetailsOriginal.name;
9
10 console.log(employeeDetailsOriginal);
11 console.log(newEm);
12
```

```

    at Module.load (internal/modules/cjs/loader.js:815:32)
    at Function.Module._load (internal/modules/cjs/loader.js:727:14)
    at Function.Module.runMain (internal/modules/cjs/loader.js:1047:10)
    at internal/main/run_main_module.js:17:11
+ jsExamples git:(master) x
+ jsExamples git:(master) x
+ jsExamples git:(master) x
+ jsExamples git:(master) x
+ jsExamples git:(master) x node
Welcome to Node.js v12.13.1.
Type ".help" for more information.
>
>
>
>
> .exit
+ jsExamples git:(master) x
+ jsExamples git:(master) x
+ jsExamples git:(master) x
+ jsExamples git:(master) x
+ jsExamples git:(master) x node test.js
{ age: 18, Profession: '前端开发' }
{ age: 18, Profession: '前端开发' }
+ jsExamples git:(master) x

```

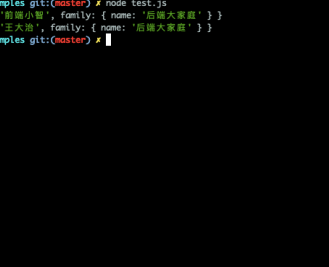
## Array方法之Array.from()

## Object.assign() 可以处理第一层的深拷贝

```

1  user > lc > BIG > jsExamples > test.js > employeeDetailsDuplicate
2  1 var employeeDetailsOriginal = {
3    name: "前嫌小智",
4    family: {
5      name: "前嫌大家族"
6    }
7  };
8
9  2 var employeeDetailsDuplicate = Object.assign({}, employeeDetailsOriginal);
10
11 3 employeeDetailsDuplicate.name = "王大治";
12 4 employeeDetailsDuplicate.family.name = "后嫌大家族";
13
14 5 console.log(employeeDetailsOriginal);
15 6 // { name: "前嫌小智", family: { name: "后嫌大家族" } }
16 7 console.log(employeeDetailsDuplicate);
17 8 // { name: "王大治", family: { name: "后嫌大家族" } }

```



```

● ● ●  温度
It@LLCmac: ~/BIG/jsExamples

+ .jsExamples git:(master) * node test.js
{ name: '熊貓小翠', family: { name: '潘曉大豪莊' } }
{ name: '王大地', family: { name: '潘曉大豪莊' } }
+ .jsExamples git:(master) * █

```

**slice、concat** 返回一个新数组的特性来实现拷贝

## 基本类型拷贝

```
1 var arr = ['old', 1, true, null, undefined];
2
3 var new_arr = arr.concat();
4
5 new_arr[0] = 'new';
6
7 console.log(arr) // ["old", 1, true, null, undefined]
8 console.log(new_arr) // ["new", 1, true, null, undefined]
```

```

1 var arr = ['old', 1, true, null, undefined];
2
3 var new_arr = arr.concat();
4
5 new_arr[0] = 'new';
6
7 console.log(arr) // ["old", 1, true, null, undefined]
8 console.log(new_arr) // ["new", 1, true, null, undefined]

```

```

lc@LCmac: ~/BIG/JsExamples
+ jsExamples git:(master) x node test.js
[ 'old', 1, true, , undefined ]
[ 'new', 1, true, , undefined ]
+ jsExamples git:(master) x

```

```

1 var arr = ['old', 1, true, null, undefined];
2
3 var new_arr = arr.slice();
4
5 new_arr[0] = 'new';
6
7 console.log(arr) // ["old", 1, true, null, undefined]
8 console.log(new_arr) // ["new", 1, true, null, undefined]

```

```

lc@LCmac: ~/BIG/JsExamples
+ jsExamples git:(master) x node test.js
[ 'old', 1, true, , undefined ]
[ 'new', 1, true, , undefined ]
+ jsExamples git:(master) x

```

## 对象拷贝

```

1 var arr = [{old: 'old'}, ['old']];
2
3 var new_arr = arr.concat();
4
5 arr[0].old = 'new';
6 arr[1][0] = 'new';
7
8 console.log(arr) // [{old: 'new'}, ['new']]
9 console.log(new_arr) // [{old: 'new'}, ['new']]

```

```

1 var arr = [{old: 'old'}, ['old']];
2
3 var new_arr = arr.concat();
4
5 arr[0].old = 'new';
6 arr[1][0] = 'new';
7
8 console.log(arr) // [{old: 'new'}, ['new']]
9 console.log(new_arr) // [{old: 'new'}, ['new']]

```

```

lc@LCmac: ~/BIG/JsExamples
+ jsExamples git:(master) x node test.js
[ { old: 'new' }, [ 'new' ] ]
[ { old: 'new' }, [ 'new' ] ]
+ jsExamples git:(master) x

```

## 深拷贝

```

1 var arr = ['old', 1, true, ['old1', 'old2'], {old: 1}]
2
3 var new_arr = JSON.parse( JSON.stringify(arr) );
4
5 console.log(new_arr);

```

```

1 var arr = ['old', 1, true, ['old1', 'old2'], {old: 1}]
2
3 var new_arr = JSON.parse( JSON.stringify(arr) );
4
5 console.log(new_arr);

```

```

lc@LCmac: ~/BIG/JsExamples
+ jsExamples git:(master) x node test.js
[ 'old', 1, true, [ 'old1', 'old2' ], { old: 1 } ]
+ jsExamples git:(master) x

```

## 使用JSON.parse(JSON.stringify(object))实现深拷贝的局限

- 1 会忽略 undefined
- 2 会忽略 symbol
- 3 不能序列化函数
- 4 不能解决循环引用的对象

```

1 let a = {
2   age: undefined,

```

```

3   sex: Symbol('male'),
4   jobs: function() {},
5   name: 'yck'
6 }
7 let b = JSON.parse(JSON.stringify(a))
8 console.log(b) // {name: "yck"}

```

```

let a = {
  age: undefined,
  sex: Symbol('male'),
  jobs: function() {},
  name: 'yck'
}
let b = JSON.parse(JSON.stringify(a))
console.log(b) // {name: "yck"}

```

```

- zsh
lica@licaideMacBook-Pro ~ % node /Users/licai/DEBUG/未命名.js
{ name: 'yck' }
lica@licaideMacBook-Pro ~ %

```

```

1 var arr = [function(){
2   console.log(a)
3 }, {
4   b: function(){
5     console.log(b)
6   }
7 }]
8
9 var new_arr = JSON.parse(JSON.stringify(arr));
10
11 console.log(new_arr);

```

```

var arr = [function(){
  console.log(a)
}, {
  b: function(){
    console.log(b)
  }
}]

var new_arr = JSON.parse(JSON.stringify(arr));
console.log(new_arr);

```

```

lica@LCmac: ~/BIG/jsExamples
+ jsExamples git:(master) ✗ node test.js
[ null, {} ]
+ jsExamples git:(master) ✗

```

拷贝 undefined 会失真

```

1 var arr = [function(){
2   console.log(a)
3 }, {
4   b: function(){
5     console.log(b)
6   }
7 }, {c: null}, {d: undefined}, {e: ''}]
8
9 var new_arr = JSON.parse(JSON.stringify(arr));
10
11 console.log(new_arr);

```

```

var arr = [function(){
  console.log(a)
}, {
  b: function(){
    console.log(b)
  }
}, {c: null}, {d: undefined}, {e: ''}]

var new_arr = JSON.parse(JSON.stringify(arr));
console.log(new_arr);

```

```

lica@LCmac: ~/BIG/jsExamples
+ jsExamples git:(master) ✗ node test.js
[ null, {}, { c: null }, {}, { e: '' } ]
+ jsExamples git:(master) ✗

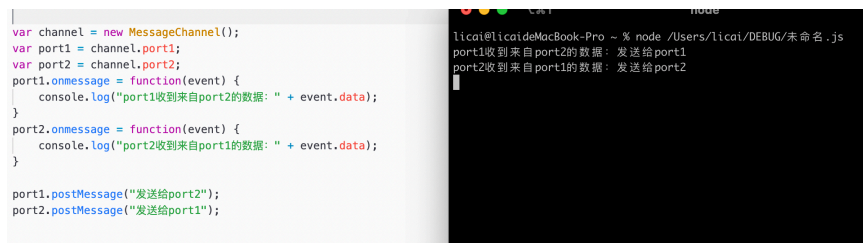
```

## 方法自定义实现深拷贝

```
1 function deepClone (obj) {
2   if (Array.isArray(obj)) {
3     return obj.map(deepClone)
4   } else if (obj && typeof obj === 'object') {
5     var cloned = {}
6     var keys = Object.keys(obj)
7     for (var i = 0, l = keys.length; i < l; i++) {
8       var key = keys[i]
9       cloned[key] = deepClone(obj[key])
10    }
11    return cloned
12  } else {
13    return obj
14  }
15 }
```

借用 MessageChannel 实现深拷贝(只能解决 undefined 和循环引用对象的问题, 对于 Symbol 和 function 依然束手无策)

```
1 var channel = new MessageChannel();
2 var port1 = channel.port1;
3 var port2 = channel.port2;
4 port1.onmessage = function(event) {
5   console.log("port1收到来自port2的数据: " + event.data);
6 }
7 port2.onmessage = function(event) {
8   console.log("port2收到来自port1的数据: " + event.data);
9 }
10
11 port1.postMessage("发送给port2");
12 port2.postMessage("发送给port1");
```



```
1 function deepClone(val) {
2   return new Promise((resolve, reject) => {
3     const {port1, port2} = new MessageChannel();
4     port2.onmessage = e => resolve(e.data);
5     port1.postMessage(val);
6   })
7 }
8
9 let obj = {
```

```

10   age: undefined,
11   name: 'yck',
12   c: {
13     d: true
14   }
15 }
16 obj.c.e = obj.c; // 循环引用
17
18 // 注意该方法是异步
19 async function test() {
20   const clone = await deepClone(obj);
21   console.log(clone) // {age: undefined, name: "yck", c: {...}}
22 }
23 test()

```



## lodash 的深拷贝函数

```

1  _.cloneDeep(value)
2
3
4  var objects = [{ 'a': 1 }, { 'b': 2 }];
5
6  var deep = _.cloneDeep(objects);
7  console.log(deep[0] === objects[0]);
8  // => false

```