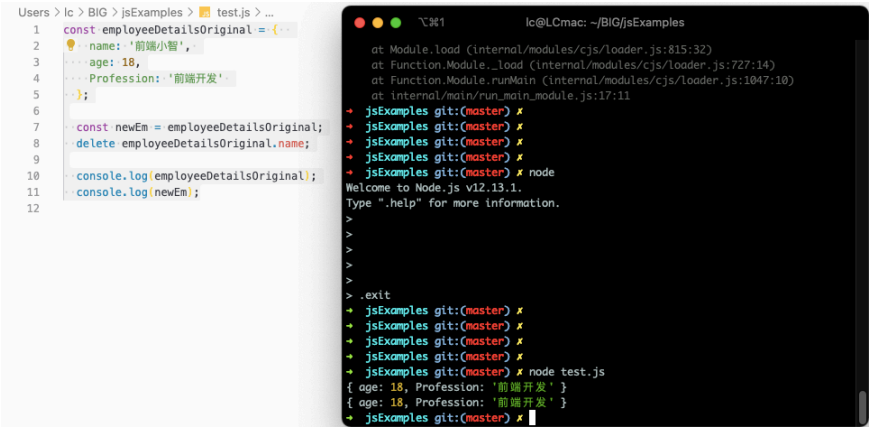
```
var arr = [function(){
  console.log(a)
}, {
  b: function(){
      console.log(b)
  }
}]

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

console.log(new_arr);
```

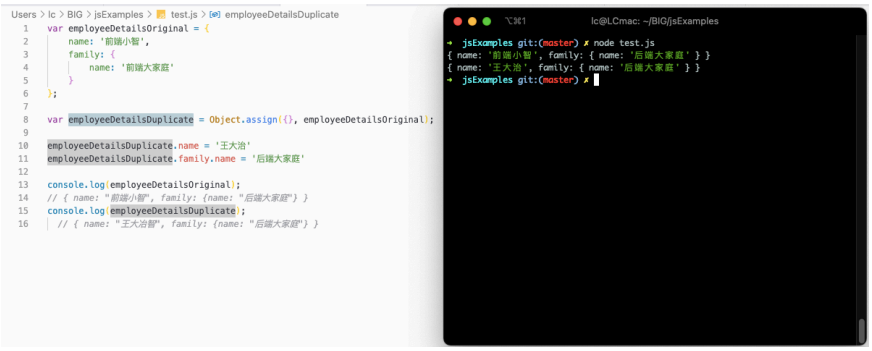
浅拷贝

赋值



Array方法之Array.from()

Object.assign() 可以处理第一层的深拷贝



slice、concat 返回一个新数组的特性来实现拷贝

基本类型拷贝

```
var arr = ['old', 1, true, null, undefined];

var new_arr = arr.concat();

new_arr[0] = 'new';

console.log(arr) // ["old", 1, true, null, undefined]
console.log(new_arr) // ["new", 1, true, null, undefined]
```

```
var arr = ['old', 1, true, null, undefined];

var new_arr = arr.concat();

new_arr[0] = 'new';

console.log(arr) // ["old", 1, true, null, undefined]
console.log(new_arr) // ["new", 1, true, null, undefined]
```

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

```
var arr = ['old', 1, true, null, undefined];

var new_arr = arr.slice();

new_arr[0] = 'new';

console.log(arr) // ["old", 1, true, null, undefined]
console.log(new_arr) // ["new", 1, true, null, undefined]
```

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

对象拷贝

```
var arr = [{old: 'old'}, ['old']];

var new_arr = arr.concat();

arr[0].old = 'new';
arr[1][0] = 'new';

console.log(arr) // [{old: 'new'}, ['new']]
console.log(new_arr) // [{old: 'new'}, ['new']]
```

```
var arr = [{old: 'old'}, ['old']];

var new_arr = arr.concat();

arr[0].old = 'new';
arr[1][0] = 'new';

console.log(arr) // [{old: 'new'}, ['new']]
console.log(new_arr) // [{old: 'new'}, ['new']]
```

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

深拷贝

```
var arr = ['old', 1, true, ['old1', 'old2'], {old: 1}]

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

console.log(new_arr);
```

```
var arr = ['old', 1, true, ['old1', 'old2'], {old: 1}]

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

console.log(new_arr);
```

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

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

```
会忽略 undefined
会忽略 symbol
不能序列化函数
不能解决循环引用的对象
```

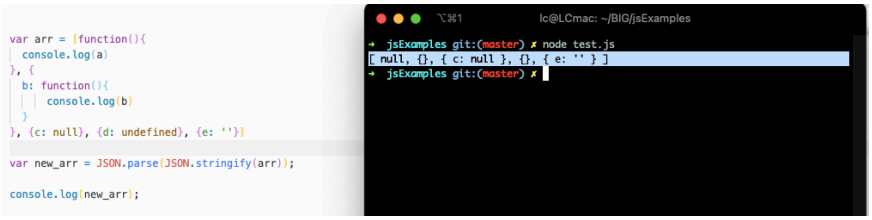
```
let a = {
    age: undefined,
```

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


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

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

console.log(new_arr);
```

拷贝 undefined 会失真

```
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);
```

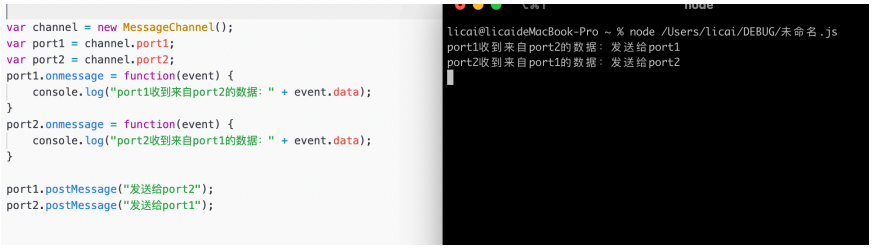


方法自定义实现深拷贝

```
function deepClone (obj) {
    if (Array.isArray(obj)) {
      return obj.map(deepClone)
    } else if (obj && typeof obj === 'object') {
      var cloned = {}
      var keys = Object.keys(obj)
      for (var i = 0, l = keys.length; i < l; i++) {
        var key = keys[i]
        cloned[key] = deepClone(obj[key])
       }
       return cloned
     } else {
       return obj
     }
   }
```

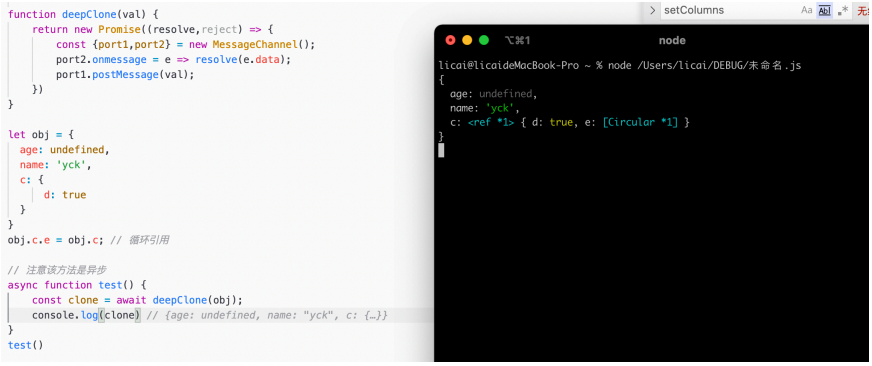
借用 MessageChannel 实现深拷贝(只能解决 undefined 和循环引用对象的问题，对于 Symbol 和 function 依然束手无策)

```
var channel = new MessageChannel();
var port1 = channel.port1;
var port2 = channel.port2;
port1.onmessage = function(event) {
    console.log("port1收到来自port2的数据: " + event.data);
}
port2.onmessage = function(event) {
    console.log("port2收到来自port1的数据: " + event.data);
}

port1.postMessage("发送给port2");
port2.postMessage("发送给port1");
```




```
function deepClone(val) {
    return new Promise((resolve,reject) => {
        const {port1,port2} = new MessageChannel();
        port2.onmessage = e => resolve(e.data);
        port1.postMessage(val);
    })
}

let obj = {
```

```
  age: undefined,
  name: 'yck',
  c: {
      d: true
  }
}
obj.c.e = obj.c; // 循环引用

// 注意该方法是异步
async function test() {
    const clone = await deepClone(obj);
    console.log(clone) // {age: undefined, name: "yck", c: {…}}
}
test()
```



lodash 的深拷贝函数

```
_.cloneDeep(value)


var objects = [{ 'a': 1 }, { 'b': 2 }];

var deep = _.cloneDeep(objects);
console.log(deep[0] === objects[0]);
// => false
```