link: null
title: 珠峰架构师成长计划
description: MDN call
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=155 sentences=469, words=3018

1. call 和 apply 的区别是什么，哪个性能更好一些

- call：第一个参数是为函数内部指定this指向，后续的参数则是函数执行时所需要的参数，一个一个传递。
- apply：第一个参数与call相同，为函数内部this指向，而函数的参数，则以数组的形式传递，作为apply第二参数。
- call 的性能更好
- apply (https://tc39.es/ecma262/#sec-function.prototype.apply)
- call-apply-test (https://jsperf.com/call-apply-segu)

```
Obj.prototype.target.apply(this, ["YES"]);
Obj.prototype.target.call(this, "YES");
```

MDN call (https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Function/call)

```
Function.prototype.call = function (obj, ...arg) {
    const context = obj;
    const fn = Symbol();
    context[fn] = this;
    const ret = context[fn](...arg);
    delete context[fn];
    return ret;
}
```

MDN apply (https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Function/apply)

```
Function.prototype.apply = function(obj, arg) {
      const context = obj;
      const fn = Symbol();
      context[fn] = this;
      const ret = context[fn](...arg);
      delete context[fn]
      return ret;
}
```

```
function fn1 () {
    console.log(1);
}
function fn2 () {
    console.log(2);
}
fn1.call.call(fn2);
```

2. ES6 代码转成 ES5 代码的实现思路是什么？#

- AST编译解析 (http://www.javascriptpeixun.cn/course/12/task/345/show)

3.数组里面有10万个数据，取第一个元素和第10万个元素的时间相差多少

- 一样

4.XHR具体底层原理和API?

- XMLHttpRequest (https://developer.mozilla.org/zh-CN/docs/Web/API/XMLHttpRequest)
- axios (http://www.javascriptpeixun.cn/course/12/task/56919/show)

5.prototype根本上解决的是什么问题？

- prototype (http://www.javascriptpeixun.cn/course/12/task/74600/show)

6. 编写parse函数，实现访问对象里属性的值

```
let obj = { a: 1, b: { c: 2 }, d: [1, 2, 3], e: [{ f: [4, 5, 6] }] };
let r1 = parse(obj, 'a');
let r2 = parse(obj, 'b.c');
let r3 = parse(obj, 'd[2]');
let r4 = parse(obj, 'e[0].f[0]');

function parse(obj, str) {
    return new Function('obj', 'return obj.' + str.replace(/\.(\d+)/g, '\[$1\]'))(obj);
}
function parse(obj, str) {
    str = str.replace(/\[(\d+)\]/g, '.$1');
    arr = str.split('.');
    arr.forEach(function (item) {
        obj = obj[item];
    })
    return obj;
}

console.log(r1, r2, r3, r4);
```

7.数组扁平化flat方法的多种实现？

```
let arr = [
    [1],
    [2, 3],
    [4, 5, 6, [7, 8, [9, 10, [11]]]],
    12
];

flat

console.log(arr.toString().split(',').map(item => Number(item)));

console.log(JSON.stringify(arr).replace(/\[|\]/g, '').split(',').map(item => Number(item)));

while (arr.some(item => Array.isArray(item))) {
    arr = [].concat(...arr);
}
console.log(arr);

Array.prototype.flat = function () {
    let result = [];
    let _this = this;
    function _flat(arr) {
        for (let i = 0; i < arr.length; i++) {
            let item = arr[i];
            if (Array.isArray(item)) {
                _flat(item);
            } else {
                result.push(item);
            }
        }
    }
    _flat(_this);
    return result;
}
console.log(arr.flat());
```

8.实现一个不可变对象

- 无论是不可扩展，密封，还是冻结，都是浅层控制的，即只控制对象本身属性的增删改。如果对象属性是一个引用类型，比如数组 subArr 或对象 subObj等，虽然subArr、subObj 的不可被删改，但subArr、subObj 的属性仍然可增删改
- 由于每个对象都有一个属性 `__proto__`,该属性的值是该对象的原型对象，也是引用类型，由于冻结是浅层的所以原型对象并不会被连着冻结，仍然可以通过给对象的原型对象加属性达到给当前对象新增属性的效果。所以如果想进一步冻结还需要把原型对象也冻结上

8.1 不可扩展

- `Object.preventExtensions()`可以使一个对象不可再添加新的属性，参数为目标对象，返回修改后的对象

```
var obj = Object.preventExtensions({
    name: 'zhufeng'
});
var obj = { name: 'zhufeng' };
console.log(Object.isExtensible(obj));
Object.preventExtensions(obj);
console.log(Object.isExtensible(obj));

obj.age = 10;
console.log(obj.age);
```

8.2 密封

- `Object.seal()` 可以使一个对象无法添加新属性的同时，也无法删除旧属性。参数是目标对象，返回修改后的对象
- 其本质是通过修改属性的 `configurable` 为 `false` 来实现的
- `configurable` 为 `false` 时，其他配置不可改变， `writable` 只能 `true` 变 `false`，且属性无法被删除。而由于只要 `writable` 或 `configurable` 其中之一为 `true`，则 `value` 可改，所以密封之后的对象还是可以改属性值的
- `Object.isSealed()` 可以检测一个对象是否密封，即是否可以增删属性。参数是目标对象，返回布尔值，true 代表被密封不可增删属性，false 代表没被密封可增删属性

```

var obj = new Object();
Object.isExtensible(obj);
Object.isSealed(obj);
Object.seal(obj);
Object.isExtensible(obj);
Object.isSealed(obj);

var obj = { name: 'zhufeng' };
console.log(Object.getOwnPropertyDescriptor(obj, 'name'));

Object.seal(obj);
console.log(Object.getOwnPropertyDescriptor(obj, 'name'));

```

8.3 冻结

- `Object.freeze()` 可以使对象一个对象不能再添加新属性，也不可以删除旧属性，且不能修改属性的值。参数是目标对象，返回修改后的对象。
- `Object.isFrozen()` 可以检测一个对象是否冻结，即是否可以增删改。参数是目标对象，返回布尔值，true 表示已经冻结不可再增删改，false 反之

```
var obj = new Object();
Object.isExtensible(obj);
Object.isSealed(obj);
Object.isFrozen(obj);
Object.freeze(obj);
Object.isExtensible(obj);
Object.isSealed(obj);
Object.isFrozen(obj);
var obj = Object.freeze({ name: 'zhufeng' });

Object.defineProperty(obj, 'name', {
    value: 'zhufeng'
});

obj.name = 'zhufeng';
obj.name;

delete obj.name;

obj.name = 'jiagou';
obj.name;
```

9.给定一组url，利用js的异步实现并发请求，并按顺序输出结果

```
function printOrder(urlArr) {
    Promise.all(urlArr.map(url => new Promise(function (resolve, reject) {
        let xhr = new XMLHttpRequest;
        xhr.open('GET', url, true);
        xhr.responseType = 'json';
        xhr.onreadystatechange = function () {
            if (xhr.readyState === 4 && xhr.status === 200) {
                resolve(xhr.response);
            }
        }
        xhr.send();
    }))).then(result => {
        console.log(result);
    });
}
printOrder(['/1.json?ts=' + Date.now(), '/2.json?ts=' + Date.now()]);
```

```
function printOrder(urlArr, callback) {
    let result = {};
    function sendRequest(url, index) {
        let xhr = new XMLHttpRequest;
        xhr.open('GET', url, true);
        xhr.responseType = 'json';
        xhr.onreadystatechange = function () {
            if (xhr.readyState === 4 && xhr.status === 200) {
                result[index] = xhr.response;
                if (Object.keys(result).length == urlArr.length) {
                    result.length = Object.keys(result).length;
                    callback(null, Array.from(result));
                }
            }
        }
        xhr.send();
    }
    urlArr.forEach(function (url, index) {
        sendRequest(url, index);
    });
}

printOrder(['/1.json?ts=' + Date.now(), '/2.json?ts=' + Date.now()], (err, result) => {
    console.log(result);
});
```

10.说下防抖和节流,能实现么？#

- 防抖和节流 (http://www.javascriptpeixun.cn/course/12/task/1608/show)

11.说下 Reflect Proxy

- Proxy 可以理解成，在目标对象之前架设一层"拦截"，外界对该对象的访问，都必须先通过这层拦截，因此提供了一种机制，可以对外界的访问进行过滤和改写。Proxy 这个词的原意是代理，用在这里表示由它来 &#x4EE3;&#x7406;某些操作，可以译为 &#x201C;代理器`
- Proxy (https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Proxy)

```
let target = {
    name: 'zhufeng',
    age: 10
}
let handler = {
    get: function (target, key) {
        return target[key];
    },
    set: function (target, key, value) {
        target[key] = value;
    }
}
let proxy = new Proxy(target, handler)
console.log(proxy.name);
proxy.age = 25;
console.log(proxy.age);
```

- 将Object对象的一些明显属于语言内部的方法（比如Object.defineProperty），放到Reflect对象上
- Reflect (https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Reflect)

```
let target = {
    name: 'zhufeng',
    age: 10
}

Reflect.defineProperty(target, 'home', {
    value: '北京'
})
console.log(target.home);

console.log(Reflect.has(target, 'home'));
```

- [Vue3.0中的应用 (http://www.javascriptpeixun.cn/course/12/task/49302/show)](http://www.javascriptpeixun.cn/course/12/task/49302/show)

12.写一个函数，可以控制最大并发数

- [semaphore (https://www.npmjs.com/package/semaphore)](https://www.npmjs.com/package/semaphore)

```
let Semaphore = require('semaphore');
let semaphore = new Semaphore(2);
console.time('cost');
semaphore.take(function () {
    setTimeout(() => {
        console.log(1);
        semaphore.leave();
    }, 1000);
});
semaphore.take(function () {
    setTimeout(() => {
        console.log(1);
        semaphore.leave();
    }, 2000);
});
semaphore.take(function () {
    console.log(3);
    semaphore.leave();
    console.timeEnd('cost');
});
```

```
class Semaphore {
    constructor(available) {
        this.available = available;
        this.waiters = [];
        this._continue = this._continue.bind(this);
    }

    take(callback) {
        if (this.available > 0) {
            this.available--;
            callback();
        } else {
            this.waiters.push(callback);
        }
    }

    leave() {
        this.available++;
        if (this.waiters.length > 0) {
            process.nextTick(this._continue);
        }
    }

    _continue() {
        if (this.available > 0) {
            if (this.waiters.length > 0) {
                this.available--;
                const callback = this.waiters.pop();
                callback();
            }
        }
    }
}
```

13.说下 js模块化（commonjs/AMD/CMD/ES6）

- 使用过哪些模块化？说说对模块化和组件化的理解

14. promise、async await、Generator的区别

15. 如何让 (a == 1 && a == 2 && a == 3) 的值为true?

15.1 valueOf和toString

- ==的时候会涉及到类型转换，如果双等号两边数据类型不同会尝试将他们转化为同一类型
- `valueOf`和 `toString`这两个方法是每个对象都自带的(继承自Object原型)
- toString返回一个字符串"[object Object]",valueOf则是直接返回对象本身

```
var obj = {a:1};

console.log(obj == "[object Object]");
```

15.2 类型转换

- 三个等号不转数据类型,二个等号不转数据类型,直接返回false 值一样,类型需要一样
- NaN和谁都不相等,包括NaN自己
- null==undefined null和undefined两个等号相等，三个等号不相等
- 对象 == 字符串 会把对象转成字符串再比较
- 剩余的都转换为数字进行比较

15.2.1 对象转原始类型步骤

- 如果部署了[Symbol.toPrimitive] 接口，那么调用此接口，若返回的不是基本数据类型，抛出错误
- 如果没有部署 [Symbol.toPrimitive] 接口，那么根据要转换的类型，先调用 valueOf / toString

    - 如果hint是number,调用顺序为 valueOf=>toString
    - 如果hint是string(针对Date),调用顺序为toString=>valueOf
    - 如果hint是default,调用顺序为valueOf=>toString

- 执行 a==1时,js引擎会尝试把对象类型a转化为数字类型，首先调用a的 valueOf方法来判断，不行则继续调用 toString方法
- 然后再把 toString返回的字符串转化为数字类型再去和a作比较
- 重写 valueOf方法也可以实现，而且转化时会优先调用valueOf方法
- toPrimitive (https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Symbol/toPrimitive)

```
let a = {
  [Symbol.toPrimitive]: (function () {
      let i = 1;
      return () => i++;
  })()
}
if (a == 1 && a == 2 && a == 3) {
  console.log('成功');
}
```

- Proxy (https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Proxy)

```
let a = new Proxy({}, {
    i: 1,
    get() {
        return () => this.i++;
    }
});
if (a == 1 && a == 2 && a == 3) {
    console.log('成功');
}
```

```
var a = {
    count:1,
    toString(){
        return this.count++;
    },

};
if(a == 1 && a ==2 && a == 3){
  console.log('成功');
}
```

```
Object.defineProperty(window,'a',{
    get(){
        return i++;
    }
})
if(a == 1 && a ==2 && a ==3 ){
    console.log('相等');
}
```

```
var a = [1,2,3];
a.toString = a.shift;

if(a == 1 && a ==2 && a ==3 ){
    console.log('相等');
}
```

16. console.log([]==![]);//true

- 比较的时候要转成数字进行比较

```
console.log([]==![]);
[]..toString() 得到空字符串=>Number('')=0
![] = false =>Number(false)=>0
```

17. +号

- 两个操作数如果是 number则直接相加出结果
- 如果其中有一个操作数为string，则将另一个操作数隐式的转换为string，然后进行字符串拼接得出结果
- 如果操作数为对象或者是数组这种复杂的数据类型，那么就将两个操作数都转换为字符串，进行拼接
- 如果操作数是像boolean这种的简单数据类型，那么就将操作数转换为number相加得出结果
- [ ] + { } 因为[]会被强制转换为"", 然后+运算符 链接一个{ }, { }强制转换为字符串就是"[object Object]"
- { } 当作一个空代码块,+[]是强制将[]转换为number,转换的过程是 +[] => +"" =>0 最终的结果就是0

```
[]+{}
{}+[]
{}+0
[]+0
```

18. 说一下arrayBuffer 和 Buffer 的区别 以及应用场景

- Buffer (http://www.zhufengpeixun.cn/2020/html/9.Buffer.html#t01.%20%E4%BB%80%E4%B9%88%E6%98%AFBuffer)
- XMLHttpRequest (https://developer.mozilla.org/zh-CN/docs/Web/API/XMLHttpRequest)
- responseType (https://developer.mozilla.org/zh-CN/docs/Web/API/XMLHttpRequest/responseType)
- ArrayBuffer (https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/ArrayBuffer)
- TypedArray (https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/TypedArray)
- DataView (https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/DataView)

18.1 1.server.js

node\1.server.js

```
let http = require('http');
let fs = require('fs');
let path = require('path');
http.createServer((req, res) => {
    if (req.url === '/') {
        let content = fs.readFileSync(path.join(__dirname, 'index.html'));
        res.setHeader('Content-Type', 'text/html');
        res.end(content);
    } else if (req.url === '/data') {
        let buffer1 = Buffer.from('abc');
        let buffer2 = Buffer.from('def');
        let buffer = Buffer.concat([buffer1, buffer2]);
        res.end(buffer);
    } else {
        res.end('');
    }
}).listen(8000);;
```

18.2 index.html

```
 <script>
        var xhr = new XMLHttpRequest();
        xhr.open('GET', '/data', true);
        xhr.responseType = 'arraybuffer';
        xhr.onload = function (e) {
            buffer = xhr.response;

            var dataView = new DataView(buffer);
            console.log(dataView.getInt8(0));
            console.log(dataView.getInt8(1));
            console.log(dataView.getInt8(2));
        };
        xhr.send();
    script>
```

```
console.log(97..toString(2));
console.log(98..toString(2));
console.log(98..toString(2) + 97..toString(2));

console.log(parseInt('0110001001100001', 2));

let buffer = Buffer.from('ab');

console.log(buffer);

console.log(parseInt('0X61', 16));
```

19. 柯理化

19.1 bind

- bind (https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Function/bind)
- create (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/create)

```
~function (prototype) {
    function bind(context = global, ...outerArgs) {
        return (...innerArgs) => {
            return this.call(context, ...outerArgs, ...innerArgs);
        }
    }
    prototype.bind = bind;
}(Function.prototype);

function sum(...args) {
    return this.prefix + args.reduce((acc, curr) => acc + curr, 0);
}
let obj = { prefix: '{{content}}#x27; };
let bindSum = sum.bind(obj, 1, 2, 3);
console.log(bindSum(4, 5));
```

```
~(function () {
    Object.create = function (proto) {
        function F() { }
        F.prototype = proto;
        return new F();
    };

    Function.prototype.bind = function (oThis, ...outerArgs) {
        var thatFunc = this,
            fBound = function (...innerArgs) {
                return thatFunc.apply(
                    this instanceof thatFunc ? this : oThis, [...outerArgs, ...innerArgs])
            };
        fBound.prototype = Object.create(thatFunc.prototype);
        return fBound;
    }
})();

function Point(x, y) {
    this.x = x;
    this.y = y;
}

Point.prototype.toString = function () {
    return this.x + ',' + this.y;
};
var emptyObj = {};
var YAxisPoint = Point.bind(null, 1);
var axisPoint = new YAxisPoint(2);
console.log(axisPoint.toString());

console.log(axisPoint instanceof Point);
console.log(axisPoint instanceof YAxisPoint);
```

19.2 add

```
console.log(add(1, 2, 3, 4, 5));
console.log(add(1)(2, 3)(4, 5));
console.log(add(1)(2)(3)(4)(5));
```

```
const add = (function (length) {
    let allArgs = [];
    function _add(...args) {
        allArgs = [...allArgs, ...args];
        if (allArgs.length >= length) {
            let sum = allArgs.reduce((acc, curr) => acc + curr, 0);
            allArgs.length = 0;
            return sum;
        } else {
            return _add;
        }
    }
    return _add;
})(5);
```

```
alert(add(1, 2, 3, 4, 5));
alert(add(1)(2, 3)(4));
alert(add(1)(2)(3));
function add(...args) {
    var _add = add.bind(null, ...args);
    _add.toString = function () {
        return args.reduce((sum, item) => sum + item, 0);
    }
    return _add;
}
```

- 函数柯里化就是把接受多个参数的函数变换成接受一个单一参数的函数，并且返回接受余下参数返回结果的技术

```
function curry(fn, ...args) {
    return args.length < fn.length ? (...extraArgs) => curry(fn, ...args, ...extraArgs) : fn(...args)
}
function addFn(a, b, c, d, e) {
    return a + b + c + d + e;
}
let add = curry(addFn);
console.log(add(1, 2, 3, 4, 5));
console.log(add(1)(2, 3)(4, 5));
console.log(add(1)(2)(3)(4)(5));
```

20. 拷贝

20.1 JSON.parse

- 无法支持所有类型,比如函数

```
let obj = { name: 'zhufeng', age: 10 };
console.log(JSON.parse(JSON.stringify(obj)));
```

20.2 浅拷贝

```
function clone(source) {
    let target = {};
    for (const key in source) {
        target[key] = source[key];
    }
    return target;
};
```

20.3 深拷贝

- 支持对象和数组

```
let obj = {
    name: 'zhufeng',
    age: 10,
    home: { name: '北京' },
    hobbies: ['抽烟', '喝酒', '烫头']
};
function clone(source) {
    if (typeof source === 'object') {
        let target = Array.isArray(source) ? [] : {};
        for (const key in source) {
            target[key] = clone(source[key]);
        }
        return target;
    }
    return source;

};
let cloned = clone(obj);
console.log(Array.isArray(cloned.hobbies));
```

20.4 循环引用

```
let obj = {
    name: 'zhufeng',
    age: 10,
    home: { name: '北京' },
    hobbies: ['抽烟', '喝酒', '烫头']
};
obj.obj = obj;
function clone(source, map = new Map()) {
    if (typeof source === 'object') {
        if (map.get(source)) {
            return map.get(source);
        }
        let target = Array.isArray(source) ? [] : {};
        map.set(source, target);
        for (const key in source) {
            target[key] = clone(source[key], map);
        }
        return target;
    }
    return source;

};

let cloned = clone(obj);
console.log(cloned.obj);
```

20.5 while

```
let obj = {
    name: 'zhufeng',
    age: 10,
    home: { name: '北京' },
    hobbies: ['抽烟', '喝酒', '烫头']
};
obj.obj = obj;
function clone(source, map = new Map()) {
    if (typeof source === 'object') {
        if (map.get(source)) {
            return map.get(source);
        }
        let target = Array.isArray(source) ? [] : {};
        map.set(source, target);
        let keys = Object.keys(source);
        let length = keys.length;
        let index = 0;
        while (index < length) {
            target[keys[index]] = clone(source[keys[index]], map);
            index++;
        }
        return target;
    }
    return source;
};
```

```
function getType(source) {
    return Object.prototype.toString.call(source);
}
```

20.6 精确类型

20.6.1 判断类型方式

- typeof
  - 返回结果都是字符串
  - 字符串中包含了对应的数据类型 number string boolean undefined symbol
  - typeof null == 'object'
  - typeof {} [] /&$/ Date== 'object'
- instanceof
- Object.prototype.toString.call

20.6.2 判断类型

- flags (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/RegExp/flags)

```
let obj = {
    married: true,
    age: 10,
    name: 'zhufeng',
    girlfriend: null,
    boyfriend: undefined,
    flag: Symbol('man'),
    home: { name: '北京' },
    set: new Set(),
    map: new Map(),
    getName: function () { },
    hobbies: ['抽烟', '喝酒', '烫头'],
    error: new Error('error'),
    pattern: /^regexp$/ig,
    math: Math,
    json: JSON,
    document: document,
    window: window
};
obj.set.add(1);
obj.map.set('name', 'value');
obj.obj = obj;

let OBJECT_TYPES = [{}, [], new Map(), new Set(), new Error(), new Date(), /^$/].map(item => getType(item));
const MAP_TYPE = getType(new Map());
const SET_TYPE = getType(new Set());
const CONSTRUCT_TYPE = [new Error(), new Date()].map(item => getType(item));
const SYMBOL_TYPE = getType(Symbol('1'));
const REGEXP_TYPE = getType(/^$/);
function clone(source, map = new Map()) {
    let type = getType(source);
    if (!OBJECT_TYPES.includes(type)) {
        return source;
    }
    if (map.get(source)) {
        return map.get(source);
    }
    if (CONSTRUCT_TYPE.includes(type)) {
        return new source.constructor(source);
    }
    let target = new source.constructor();
    map.set(source, target);

    if (SYMBOL_TYPE === type) {
        return Object(Symbol.prototype.valueOf.call(source));
    }
    if (REGEXP_TYPE === type) {
        const flags = /\w*$/;
        const target = new source.constructor(source.source, flags.exec(source));
        target.lastIndex = source.lastIndex;
        return target;
    }
    if (SET_TYPE === type) {
        source.forEach(value => {
            target.add(clone(value, map));
        });
        return target;
    }
    if (MAP_TYPE === type) {
        source.forEach((value, key) => {
            target.set(key, clone(value, map));
        });
        return target;
    }
    let keys = Object.keys(source);
    let length = keys.length;
    let index = 0;
    while (index < length) {
        target[keys[index]] = clone(source[keys[index]], map);
        index++;
    }
    return target;
};

function getType(source) {
    return Object.prototype.toString.call(source);
}
let cloned = clone(obj);
console.log(cloned);
console.log(obj.home === cloned.home);
console.log(obj.set === cloned.set);
console.log(obj.map === cloned.map);
```

庄同学提供

```javascript
let obj = {
    married: true,
    age: 10,
    name: 'zhufeng',
    girlfriend: null,
    boyfriend: undefined,
    flag: Symbol('man'),
    home: { name: '北京' },
    set: new Set(),
    map: new Map(),
    getName: function () { },
    hobbies: ['抽烟', '喝酒', '烫头'],
    error: new Error('error'),
    pattern: /^regexp$/ig,
    date: new Date()

};
obj.set.add(1);
obj.map.set('name', 'value');
obj.obj = obj;

const getType = (o) => Object.prototype.toString.call(o);
const arrayTag = '[object Array]'
const dateTag = '[object Date]'
const errorTag = '[object Error]'
const mapTag = '[object Map]'
const objectTag = '[object Object]'
const regexpTag = '[object RegExp]'
const setTag = '[object Set]'
const symbolTag = '[object Symbol]'
const objectTags = [arrayTag, objectTag, regexpTag, symbolTag, setTag, mapTag, errorTag, dateTag];
const instanceTags = [regexpTag, errorTag, dateTag];
function clone(source, map = new Map()) {
    let target;
    const sourceType = getType(source);
    if (!objectTags.includes(sourceType)) return source;
    if (map.get(source)) return map.get(source);
    const constructor = source.constructor;
    if (instanceTags.includes(sourceType)) return new constructor(source);
    if (sourceType === symbolTag) {
        return Object(Symbol.prototype.valueOf.call(source));
    }
    target = new constructor();
    map.set(source, target);
    if (sourceType === mapTag) {
        source.forEach((value, key) => target.set(key, clone(value, map)));
        return target;
    }
    if (sourceType === setTag) {
        source.forEach((value) => target.add(clone(value, map)));
        return target;
    }
    for (const key in source) {
        target[key] = clone(source[key], map);
    }
    return target;
}
const obj2 = clone(obj);

console.log('obj', obj);
console.log('obj2', obj2);
```