link: null
title: 珠峰架构师成长计划
description: str = str.split('').sort().join('');
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=99 sentences=314, words=2192

1.call的实现

```
Function.prototype.call = function(context,...args){
    let cxt = context || window;
    let func = Symbol()
    cxt[func] = this;
    args = args ? args : []
    const res = args.length > 0 ? cxt[func](...args) : cxt[func]();
    delete cxt[func];
    return res;
}
```

2. apply的实现

```
Function.prototype.apply = function(context,args = []){
    let cxt = context || window;
    let func = Symbol()
    cxt[func] = this;
    const res = args.length > 0 ? cxt[func](...args) : cxt[func]();
    delete cxt[func];
    return res;
}
```

3. bind的实现

```
Function.prototype.bind = function (context, ...args) {
    const fn = this
    args = args ? args : []
    return function newFn(...newFnArgs) {
        if (this instanceof newFn) {
            return new fn(...args, ...newFnArgs)
        }
        return fn.apply(context, [...args,...newFnArgs])
    }
}
```

4. 寄生式组合继承

```
function Person(obj) {
    this.name = obj.name
    this.age = obj.age
}
Person.prototype.add = function(value){
    console.log(value)
}
var p1 = new Person({name:"zhangsan", age: 18})

function Person1(obj) {
    Person.call(this, obj)
    this.sex = obj.sex
}

Person1.prototype = Object.create(Person.prototype);
Person1.prototype.constructor = Person1;

Person1.prototype.play = function(value){
    console.log(value)
}
var p2 = new Person1({name:"lisi", age: 18, sex: "男"})
```

5. ES6继承

```
class Person{
  constructor(name='zhangsan',age=18){
    this.name = name;
    this.age = age;
  }
  desc(){
    console.log(`${this.name} ${this.age}`)
  }
}

class Man extends Person{
   constructor(name = 'lisi',age = 18){
     super(name, age);
   }
   desc(){
      super.eat()
   }
}
```

6. new的实现

```
function new(ctor,...args){
    if(typeof ctor !== 'function'){
      throw 'the first param must be a function';
    }
    var newObj = Object.create(ctor.prototype);
    var ctorReturnResult = ctor.apply(newObj, args);
    var isObject = typeof ctorReturnResult === 'object' && ctorReturnResult !== null;
    var isFunction = typeof ctorReturnResult === 'function';
    if(isObject || isFunction){
        return ctorReturnResult;
    }
    return newObj;
}
let c = new(Ctor);
```

7.instanceof的实现

```
function myInstanceOf(a,b){
    let left = a.__proto__;
    let right = b.prototype;
    while(true){
        if(left == null){
            return false
        }
        if(left == right){
            return true
        }
        left = left.__proto__
    }
}
function myInstanceof(left, right) {
    let proto = Object.getPrototypeOf(left),
    prototype = right.prototype;
    while (true) {
        if (!proto) return false;
        if (proto === prototype) return true;
        proto = Object.getPrototypeOf(proto);
    }
}
```

8.Object.create()的实现

```

function myCreate(obj){
    function C(){};
    C.prototype = obj;
    return new C()
}

if (typeof Object.create !== "function") {
    Object.create = function (proto, propertiesObject) {
        if (typeof proto !== 'object' && typeof proto !== 'function') {
            throw new TypeError('Object prototype may only be an Object: ' + proto);
        } else if (proto === null) {
            throw new Error("This browser's implementation of Object.create is a shim and doesn't support 'null' as the first argument.");
        }

        if (typeof propertiesObject !== 'undefined') throw new Error("This browser's implementation of Object.create is a shim and doesn't support a second
argument.");

        function F() {}
        F.prototype = proto;

        return new F();
    };
}
```

9.实现 Object.assign

```
Object.assign2 = function(target, ...source) {
    if (target == null) {
        throw new TypeError('Cannot convert undefined or null to object')
    }
    let ret = Object(target)
    source.forEach(function(obj) {
        if (obj != null) {
            for (let key in obj) {
                if (obj.hasOwnProperty(key)) {
                    ret[key] = obj[key]
                }
            }
        }
    })
    return ret
}
```

10.Ajax的实现

```
function ajax(url,method,body,headers){
    return new Promise((resolve,reject)=>{
        let req = new XMLHttpRequest();
        req.open(methods,url);
        for(let key in headers){
            req.setRequestHeader(key,headers[key])
        }
        req.onreadystatechange(()=>{
            if(req.readystate == 4){
                if(req.status >= '200' && req.status 300){
                    resolve(req.responseText)
                }else{
                    reject(req)
                }
            }
        })
        req.send(body)
    })
}
```

11.实现防抖函数（debounce）

```
let debounce = (fn,time = 1000) => {
    let timeLock = null
    return function (...args){
        clearTimeout(timeLock)
        timeLock = setTimeout(()=>{
            fn(...args)
        },time)
    }
}
```

12.实现节流函数（throttle）

```
let throttle = (fn,time = 1000) => {
    let flag = true;
    return function (...args){
        if(flag){
            flag = false;
            setTimeout(()=>{
                flag = true;
                fn(...args)
            },time)
        }
    }
}
```

13. 深拷贝

```
function deepClone(obj,hash = new WeakMap()){
    if(obj instanceof RegExp) return new RegExp(obj);
    if(obj instanceof Date) return new Date(obj);
    if(obj === null || typeof obj !== 'object') return obj;
    if(hash.has(obj)){
        return hash.get(obj)
    }
    let constr = new obj.constructor();
    hash.set(obj,constr);
    for(let key in obj){
        if(obj.hasOwnProperty(key)){
            constr[key] = deepClone(obj[key],hash)
        }
    }
    let symbolObj = Object.getOwnPropertySymbols(obj)
    for(let i=0;iif(obj.hasOwnProperty(symbolObj[i])){
            constr[symbolObj[i]] = deepClone(obj[symbolObj[i]],hash)
        }
    }
    return constr
}
```

14. 数组扁平化的实现(flat)

```
let arr = [1,2,[3,4,[5,[6]]]]
console.log(arr.flat(Infinity))
```

```
function fn(arr){
   return arr.reduce((prev,cur)=>{
      return prev.concat(Array.isArray(cur)?fn(cur):cur)
   },[])
}
```

15. 函数柯里化

```
function sumFn(a,b,c){return a+ b + c};
let sum = curry(sumFn);
sum(2)(3)(5)
sum(2,3)(5)
```

```
function curry(fn,...args){
  let fnLen = fn.length,
      argsLen = args.length;
  if(fnLen > argsLen){
    return function(...arg2s){
      return curry(fn,...args,...arg2s)
    }
  }else{
    return fn(...args)
  }
}
```

16.使用闭包实现每隔一秒打印 1,2,3,4

```
for (var i=1; i5; i++) {
  (function (i) {
    setTimeout(() => console.log(i), 1000*i)
  })(i)
}
```

17. 手写一个 jsonp

```
const jsonp = function (url, data) {
    return new Promise((resolve, reject) => {
        let dataString = url.indexOf('?') === -1 ? '?' : ''
        let callbackName = `jsonpCB_${Date.now()}`
        url += `${dataString}callback=${callbackName}`
        if (data) {
            for (let k in data) {
                url += `${k}=${data[k]}`
            }
        }
        let jsNode = document.createElement('script')
        jsNode.src = url
        window[callbackName] = result => {
            delete window[callbackName]
            document.body.removeChild(jsNode)
            if (result) {
                resolve(result)
            } else {
                reject('没有返回数据')
            }
        }

        jsNode.addEventListener('error', () => {
            delete window[callbackName]
            document.body.removeChild(jsNode)
            reject('JavaScript资源加载失败')
        }, false)

        document.body.appendChild(jsNode)
    })
}
jsonp('http://127.0.0.1/jsonp', {
    a: 1,
    b: 'hello'
})
.then(result => {
    console.log(result)
})
.catch(err => {
    console.error(err)
})
```

18.手写一个观察者模式

```
class Subject{
  constructor(name){
    this.name = name
    this.observers = []
    this.state = 'off'
  }
  attach(observer){
    this.observers.push(observer)
  }

  setState(newState){
    this.state = newState
    this.observers.forEach(o=>{
      o.update(newState)
    })
  }
}

class Observer{
  constructor(name){
    this.name = name
  }

  update(newState){
    console.log(`${this.name}say:${newState}`)
  }
}

let sub = new Subject('')
let zhangsan = new Observer('zhangsan')
let lisi = new Observer('lisi')

sub.attach(zhangsan)
sub.attach(lisi)

sub.setState('on')
```

19. EventEmitter 实现

```
EventEmitter 实现
class EventEmitter {
    constructor() {
        this.events = {};
    }
    on(event, callback) {
        let callbacks = this.events[event] || [];
        callbacks.push(callback);
        this.events[event] = callbacks;
        return this;
    }
    off(event, callback) {
        let callbacks = this.events[event];
        this.events[event] = callbacks && callbacks.filter(fn => fn !== callback);
        return this;
    }
    emit(event, ...args) {
        let callbacks = this.events[event];
        callbacks.forEach(fn => {
            fn(...args);
        });
        return this;
    }
    once(event, callback) {
        let wrapFun = function (...args) {
            callback(...args);
            this.off(event, wrapFun);
        };
        this.on(event, wrapFun);
        return this;
    }
}
```

20. 生成随机数的各种方法？

```
function getRandom(min, max) {
  return Math.floor(Math.random() * (max - min)) + min
}
```

21. 如何实现数组的随机排序？

```
let arr = [1,2,3,4,5]
arr.sort(randomSort)
function randomSort(a, b) {
  return Math.random() > 0.5 ? -1 : 1;
}
```

22. 写一个通用的事件侦听器函数

```
const EventUtils = {
  addEvent: function(element, type, handler) {
    if (element.addEventListener) {
      element.addEventListener(type, handler, false);
    } else if (element.attachEvent) {
      element.attachEvent("on" + type, handler);
    } else {
      element["on" + type] = handler;
    }
  },
  removeEvent: function(element, type, handler) {
    if (element.removeEventListener) {
      element.removeEventListener(type, handler, false);
    } else if (element.detachEvent) {
      element.detachEvent("on" + type, handler);
    } else {
      element["on" + type] = null;
    }
  },
  getTarget: function(event) {
    return event.target || event.srcElement;
  },
  getEvent: function(event) {
    return event || window.event;
  },
  stopPropagation: function(event) {
    if (event.stopPropagation) {
      event.stopPropagation();
    } else {
      event.cancelBubble = true;
    }
  },
  preventDefault: function(event) {
    if (event.preventDefault) {
      event.preventDefault();
    } else {
      event.returnValue = false;
    }
  }
};
```

23. 使用迭代的方式实现 flatten 函数

```
var arr = [1, 2, 3, [4, 5], [6, [7, [8]]]]
function wrap() {
    var ret = [];
    return function flat(a) {
        for (var item of
            a) {
                if (item.constructor === Array) {
                    ret.concat(flat(item))
                } else {
                    ret.push(item)
                }
        }
        return ret
    }
}
console.log(wrap()(arr));
```

24. 怎么实现一个sleep

```
function sleep(delay) {
  var start = (new Date()).getTime();
  while ((new Date()).getTime() - start < delay) {
    continue;
  }
}

function test() {
  console.log('111');
  sleep(2000);
  console.log('222');
}

test()
```

25. 实现正则切分千分位（10000 => 10,000）

```
let num1 = '1321434322222'
num1.replace(/(\d)(?=(\d{3})+$)/g,'$1,')

let num2 = '342243242322.3432423'
num2.replace(/(\d)(?=(\d{3})+\.)/g,'$1,')
```

26. 对象数组去重

```
输入:
[{a:1,b:2,c:3},{b:2,c:3,a:1},{d:2,c:2}]
输出:
[{a:1,b:2,c:3},{d:2,c:2}]
```

```
function objSort(obj){
    let newObj = {}
    Object.keys(obj).sort().map(key => {
        newObj[key] = obj[key]
    })
    return JSON.stringify(newObj)
}

function unique(arr){
    let set = new Set();
    for(let i=0;ilet str = objSort(arr[i])
        set.add(str)
    }
    arr = [...set].map(item => {
        return JSON.parse(item)
    })
    return arr
}
```

27.解析 URL Params 为对象

```
let url = 'http://www.domain.com/?user=zhangsan&id=100&id=200&city=beijing&enabled';
parseParam(url)
```

```
function parseParam(url) {
  const paramsStr = /.+\?(.+)$/.exec(url)[1];
  const paramsArr = paramsStr.split('&');
  let paramsObj = {};
  paramsArr.forEach(param => {
    if (/=/.test(param)) {
      let [key, val] = param.split('=');
      val = decodeURIComponent(val);
      val = /^\d+$/.test(val) ? parseFloat(val) : val;

      if (paramsObj.hasOwnProperty(key)) {
        paramsObj[key] = [].concat(paramsObj[key], val);
      } else {
        paramsObj[key] = val;
      }
    } else {
      paramsObj[param] = true;
    }
  })

  return paramsObj;
}
```

28.模板引擎实现

```
let template = '我是{{name}}，年龄{{age}}，性别{{sex}}';
let data = {
  name: '姓名',
  age: 18
}
render(template, data);
```

```
function render(template, data) {
  const reg = /\{\{(\w+)\}\}/;
  if (reg.test(template)) {
    const name = reg.exec(template)[1];
    template = template.replace(reg, data[name]);
    return render(template, data);
  }
  return template;
}
```

29.转化为驼峰命名

```
var s1 = "get-element-by-id"
```

```
var f = function(s) {
    return s.replace(/-\w/g, function(x) {
        return x.slice(1).toUpperCase();
    })
}
```

30.查找字符串中出现最多的字符和个数

- 例: abbcccddddd -> 字符最多的是d，出现了5次 ````js let str = "abcabcabcbbccccc"; let num = 0; let char = '';

str = str.split('').sort().join('');

let re = /(\w)\1+/g; str.replace(re,($0,$1) => { if(num < $0.length){ num = $0.length; char = $1;
} }); console.log(字符最多的是${char}，出现了${num}次);

```
## 31.&#x56FE;&#x7247;&#x61D2;&#x52A0;&#x8F7D;
```js
let imgList = [...document.querySelectorAll('img')]
let length = imgList.length

const imgLazyLoad = function() {
    let count = 0
    return (function() {
        let deleteIndexList = []
        imgList.forEach((img, index) => {
            let rect = img.getBoundingClientRect()
            if (rect.top < window.innerHeight) {
                img.src = img.dataset.src
                deleteIndexList.push(index)
                count++
                if (count === length) {
                    document.removeEventListener('scroll', imgLazyLoad)
                }
            }
        })
        imgList = imgList.filter((img, index) => !deleteIndexList.includes(index))
    })()
}

document.addEventListener('scroll', imgLazyLoad)
```
```