link: null
title: 珠峰架构师成长计划
description: qs.js
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=69 sentences=234, words=1245

1. 手工测试

qs.js

```
function parse(str) {
    let arr = str.split('&');
    let obj = {};
    arr.forEach((item) => {
        let [key, val] = item.split('=');
        obj[key] = val;
    });
    return obj;
}
function stringify(obj) {
    let arr = [];
    for (let key in obj) {
        arr.push(key + '=' + obj[key]);
    }
    return arr.join('&');
}
exports.parse = parse;
exports.stringify = stringify;
```

2. assert断言

- 断言是表达设计人员对于系统应达到状态的一种预期
- 各种语言都内置了断言的接口
- 断言是单元测试的核心

问题 解决方案 污染 源代码里混杂了很多测试代码 从源代码中抽离出去 零散 杂乱无章，不能分组和分类 整体规划 和设计 没有持久化 没有存储 把测试文件单独存放 手动跑测试比较麻烦 自动运行并显示结果

3. 测试框架

- 通过测试框架，我们可以分离测试代码和源代码
- 测试框架可以自动运行所有的用例并输出结果
- 测试框架可能提高编写测试代码的效率

4.开发模式

`Test-Driven-Development` 测试驱动开发，

- 在TDD理念中，先有测试代码再有功能逻辑代码
- 包括测试用例和断言
- 分为模块测试和单元测试
- 有其使用场景，不能滥用
- 在实际开发中一般会使用测试框架

5. 常用测试框架

- qunit (https://github.com/qunitjs/qunit) jQuery
- mocha (https://github.com/mochajs/mocha) 支持Node&Browser express.js
- jasmine (https://github.com/jasmine/jasmine)支持Node&Browser Vue.js
- karma (https://github.com/karma-runner/karma) A Test-Runner 在不同的浏览器中跑测试用例 Angular
- jest (https://github.com/facebook/jest) React
    - 零配置
    - 内置代码覆盖率
    - 内置Mocks

6. Jest

```
cnpm i jest --save-dev
cnpm i jest -g
```

**test.js

```
jest npm jest
npm test
```

- Test Suits 测试套件，每个文件就是一个套件
- Test Group 分组 describe
- Test Case 测试用途 test()
- Assert 断言 expect()

qs.test.js

```
let { parse, stringify } = require('./qs');
describe('parse', () => {
    test('one', () => {
        expect(parse("name=zfpx").name).toBe('zfpx');
    });
    test('two', () => {
        expect(parse("name=zfpx&age=9").age).toBe('9');
    });
});

describe('stringify', () => {
    test('one', () => {
        expect(stringify({ name: 'zfpx' })).toBe('name=zfpx');
    });
    test('two', () => {
        expect(stringify({ name: 'zfpx', age: 9 })).toBe('name=zfpx&age=9');
    });
});
```

- package.json
- jest.config.js
- 命令行
- testMatch glob规则,识别哪些文件中测试文件
- testRegex 文件正则
- testEnvironment 测试环境
- rootDir 根目录
- moduleFileExtensions 模块文件扩展名

```
module.exports = {
    testMatch: ['**/__tests__/**/*.js?(x)', '**/?(*.)(spec|test).js?(x)'],
    testRegex: '(/__tests__).*|(\\.|/)(test|spec))\\.jsx?{{content}}#x27;,
    testEnvironment: 'jsdom',
    rootDir: '',
    moduleFileExtensions: ['js', 'json', 'jsx', 'node']
}
```

- 相等断言
    - toBe(value)： 比较数字、字符串
    - toEqual(value)： 比较对象、数组
    - toBeNull()
    - toBeUndefined()
- 包含断言
    - toHaveProperty(keyPath, value)： 是否有对应的属性
    - toContain(item)： 是否包含对应的值，括号里写上数组、字符串
    - toMatch(regexpOrString)： 括号里写上正则
- 逻辑断言,在JavaScript中，有六个falsy值：false，0，''，null， undefined，和NaN。其他一切都是Truthy。
    - toBeTruthy()
    - toBeFalsy()
    - oBeGreaterThan(number)： 大于
    - toBeLessThan(number)： 小于
- not 取反

```
test('matchers', () => {
    const a = {
        name: 'a',
        home: {
            name: 'beijing'
        }
    }
    const b = {
        name: 'a',
        home: {
            name: 'beijing'
        }
    }
    expect(a).toEqual(b)
    expect([1, 2, 3]).toEqual([1, 2, 3])
    expect(null).toBeNull()

    expect([1, 2, 3]).toContain(1)
    expect(b).toHaveProperty('home')
    expect('abc').toContain('b')
    expect('abc').toMatch(/^\w+$/)
    expect('123').not.toContain('4')
})
```

```
function remove(node) {
    node.parentNode.removeChild(node);
}
function on(node, type, handler) {
    node.addEventListener(type, handler);
}
exports.remove = remove;
exports.on = on;
```

```
let { remove, on } = require('../src/dom');
describe('dom', () => {
    test('remove', () => {
        document.body.innerHTML = 'hello';
        let container = document.getElementById('container');
        expect(container.nodeName.toLocaleLowerCase()).toBe('div');
        let hello = document.getElementById('hello');
        expect(hello.nodeName.toLocaleLowerCase()).toBe('span');
        remove(hello);
        let hello2 = document.getElementById('hello');
        expect(hello2).toBeNull();
    })

    test('on', () => {
        document.body.innerHTML = 'click';
        let clickMe = document.getElementById('clickMe');
        on(clickMe, 'click', () => {
            clickMe.innerHTML = 'clicked';
        });
        clickMe.click();
        expect(clickMe.innerHTML).toBe('clicked');

    })
});
```

```
<div id="tab">
    <div>
        <a href="#" class="tab-button">选项1a>
        <a href="#" class="tab-button">选项2a>
    div>
    <div>
        <div class="tab-panel">面板1div>
        <div class="tab-panel">面板2div>
    div>
div>
```

```
class Tab{
    constructor(id,buttonClass,panelClass){
        this.tab = tab = document.querySelector('#'+id);
        this.buttons = Array.from(tab.querySelectorAll('.'+buttonClass));
        this.panels = Array.from(tab.querySelectorAll('.'+panelClass));
        this.select(0);
        this.bindEvent();
    }
    select(index){
        this.buttons.forEach(button=>button.style.backgroundColor= 'white');
        this.buttons[index].style.backgroundColor= 'red';
        this.panels.forEach(panel=>panel.style.display= 'none');
        this.panels[index].style.display= 'block';
    }
    bindEvent(){
        for(let i=0;i<this.buttons.length;i++){
            this.buttons[i].addEventListener('click',()=>{
                this.select(i);
            });
        }
    }
}
module.exports = Tab;
```

```
const Tab = require('../src/tab');
const fs = require('fs');
const path = require('path');

test('tab',function(){
    document.body.innerHTML = fs.readFileSync(path.resolve(__dirname,'tab.html'),'utf8');
    const tab = new Tab('tab','tab-button','tab-panel');
    expect(tab.buttons[0].style.backgroundColor).toBe('red');
    expect(tab.buttons[1].style.backgroundColor).toBe('white');
    expect(tab.panels[0].style.display).toBe('block');
    expect(tab.panels[1].style.display).toBe('none');
    tab.buttons[1].click();
    expect(tab.buttons[0].style.backgroundColor).toBe('white');
    expect(tab.buttons[1].style.backgroundColor).toBe('red');
    expect(tab.panels[0].style.display).toBe('none');
    expect(tab.panels[1].style.display).toBe('block');
});
```

```
test('async',(done)=>{
    setTimeout(()=>{
        expect(2).toBe(2);
        done();
    },1000);
});
```

- Mocks可以擦除函数的实际实现来测试代码之间的链接
- Mocks可以捕获对函数的调用
- manual_mocks用可mock依赖的模块，放置在相应 **mocks** 目录下
- 使用mock function可以查看函数的调用次数，以及参数

tests\users.js

```
jest.mock('../js/ajax');
const getUsers = require('../js/getUsers');
test('getUsers',(done)=>{
    document.body.innerHTML = ``;
    getUsers('/users.json','users',()=>{
        const ul = document.querySelector('#users');
        const lis = ul.querySelectorAll('li');
        expect(lis.length).toBe(2);
        expect(lis[1].innerHTML).toBe('zfpx2');
        done();
    });
});
```

\js\getUsers.js

```
var ajax = require('./ajax');
function getUsers(url,id,callback){
 const users = document.getElementById(id);
 ajax(url,data=>{
     console.log('ajax ok');
     data = JSON.parse(data);
     users.innerHTML = data.map(item=>`${item.name}`).join('');
     callback&&callback();
 });
}

module.exports = getUsers;
```

\js\ajax.js

```
function ajax(url,success){
  const xhr = new XMLHttpRequest();
  xhr.onreadystatechange = ()=>{
      if(xhr.readyState == 4){
          success&success(xhr.responseText);
      }
  }
  xhr.open('get',url);
  xhr.send(null);
}
module.exports = ajax;
```

js__mocks__\ajax.js

```
const fs = require('fs');
const path = require('path');
const ajax = (url,success)=>{
   setTimeout(()=>{
       success(JSON.stringify([{name:'zfpx1'},{name:'zfpx2'}]));
   },1000);
}
module.exports = ajax;
```

```

<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Documenttitle>
head>

<body>
    <ul id="users">ul>
    <script>
        let module = {};
        let require = () => ajax;
    script>
    <script src="/js/ajax.js">script>
    <script src="/js/getUsers.js">script>
    <script>
        getUsers('http://localhost:3000/users.json', 'users');
    script>
body>

html>
```

users.html

```

    Document

        let module = {};
        let require = () => ajax;

        getUsers('http://localhost:3000/users.json', 'users');

```

server.js

```
const express = require('express');
const path = require('path');
const app = express();
app.use(express.static(__dirname));
app.get('/',function(req,res){
    res.sendFile(path.resolve(__dirname,'users.html'));
});

app.get('/users.json',function(req,res){
  res.json([{name:'zfpx1'},{name:'zfpx2'}]);
});
app.listen(3000);
```

- line coverage 行覆盖率
- function coverage 函数覆盖率
- branch coverage 分支覆盖率
- statement coverage 语句覆盖率

```
npx jest --coverage
```

7.附录

gulp内部使用了node-glob模块来实现其文件匹配功能。我们可以使用下面这些特殊的字符来匹配我们想要的文件：

匹配符 说明 星 匹配文件路径中的0个或多个字符，但不会匹配路径分隔符 ** 匹配路径中的0个或多个目录及其子目录 [...] 匹配方括号中出现的字符中的任意一个，当方括号中第一个字符为^或!时，则表示不匹配方括号中出现的其他字符中的任意一个 !(pattern pattern pattern) 匹配任何与括号中给定的任一模式都不匹配的 ?(pattern pattern pattern) 匹配括号中给定的任一模式0次或1次，类似于js正则中的? +(pattern pattern pattern) 匹配括号中给定的任一模式至少1次，类似于js正则中的+ (pattern pattern pattern) 匹配括号中给定的任一模式0次或多次，类似于js正则中的 * @(pattern pattern pattern) 匹配括号中给定的任一模式1次，类似于js正则中的

glob 匹配 * 能匹配 a.js,x.y,abc,abc/,但不能匹配a/b.js

a.js,style.css,a.b,x.y

_/.js 能匹配 a/b/c.js,x/y/z.js,不能匹配a/b.js,a/b/c/d.js ** 能匹配 abc,a/b.js,a/b/c.js,x/y/z,x/y/z/a.b,能用来匹配所有的目录和文件 a//z 能匹配 a/z,a/b/z,a/b/c/z,a/d/g/h/j/k/z a/b/z 能匹配 a/b/z,a/sb/z,但不能匹配a/x/sb/z,因为只有单_*单独出现才能匹配多级目录 ?.js 能匹配 a.js,b.js,c.js a?? 能匹配 a.b,abc,但不能匹配ab/,因为它不会匹配路径分隔符 [xyz].js 只能匹配 x.js,y.js,z.js,不会匹配xy.js,xyz.js等,整个中括号只代表一个字符 [^xyz].js 能匹配 a.js,b.js,c.js等,不能匹配x.js,y.js,z.js