link: null
title: 珠峰架构师成长计划
description: src\math.js
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=248 sentences=649, words=3970

1.为什么需要测试

- 软件测试是一种实际输出与预期输出之间的审核或者比较过程
- 测试可以尽早发现 BUG
- 测试可以提高代码质量
- 测试可以让我们自信地重构

1.1 创建项目

```
mkdir zhufeng_test
cd zhufeng_test
npm init -y
mkdir src
```

1.2 手工测试

src\math.js

```
function add(a, b) {
    return a + b;
}
function minus(a, b) {
    return a - b;
}
function multiply(a, b) {
    return a * b;
}
function divide(a, b) {
    return a / b;
}

console.log(add(4, 2));
console.log(minus(4, 2));
console.log(multiply(4, 2));
console.log(divide(4, 2));
```

1.3 断言

- 表达程序设计人员对于系统应达状态的一种预期
- 断言是测试的核心,很多语言都内置了断言接口

```
function add(a, b) {
    return a + b;
}
function minus(a, b) {
    return a - b;
}
function multiply(a, b) {
    return a * b;
}
function divide(a, b) {
    return a / b;
}

console.assert(add(4, 2) == 6, true);
console.assert(minus(4, 2) == 2, true);
console.assert(multiply(4, 2) == 8, true);
console.assert(divide(4, 2) == 2, true);
```

1.4 测试框架

- 测试用例可用来测试程序是否正确工作
- 通常把一组相关的测试称为一个测试套件(test suite)

手工断言 测试框架 污染源代码 可能分离测试代码和源代码 散落在各个文件中 测试代码可以集中存放 没有办法持久化保存 放置到单独的文件中 手动执行和对比麻烦不自动 可以自动运行、显示测试结果

math.js

```
function add(a, b) {
    return a + b;
}
function minus(a, b) {
    return a - b;
}
function multiply(a, b) {
    return a * b;
}
function divide(a, b) {
    return a / b;
}

module.exports = {
    add,
    minus,
    multiply,
    divide
}
```

math.test.js

```
let { add, minus, multiply, divide } = require('./math');
describe('测试add', function () {
    test('测试1+1', function () {
        assert(add(1, 1) == 2, '1+1没有等于2');
    });
    test('测试2+2', function () {
        assert(add(2, 2) == 4, '2+2没有等于4');
    });
});
describe('测试minus', function () {
    test('测试1+1', function () {
        assert(minus(1, 1) == 0, '1-1没有等于0');
    });
    test('测试2+2', function () {
        assert(minus(2, 2) == 0, '2-2没有等于0');
    });
});
function describe(message, testSuite) {
    console.log('测试套件', message);
    testSuite();
}
function test(message, testCase) {
    console.log('测试用例', message);
    testCase();
}
function assert(assertion, message) {
    if (!assertion) {
        throw new Error(message);
    }
}
```

5.Jest

- jest (https://jestjs.io/docs/zh-Hans/getting-started)由Facebook出品,非常适合React测试
- 零配置
- 内置代码覆盖率
- 强大的Mocks

5.1 安装

```
cnpm install --save-dev jest
```

5.2 初体验

src\math.spec.js

```
let { add, minus, multiply, divide } = require('./math');
describe('测试add', function () {
    test('测试1+1', function () {
        expect(add(1, 1)).toBe(2);
    });
    test('测试2+2', function () {
        expect(add(2, 2)).toBe(4);
    });
});
describe('测试minus', function () {
    test('测试1-1', function () {
        expect(minus(1, 1)).toBe(0);
    });
    test('测试2-2', function () {
        expect(minus(2, 2)).toBe(0);
    });
});
```

package.json

```
  "scripts": {
+    "test": "jest  --watchAll"
  }
```

```
npm  test
```

```
> jest

 PASS  src/math.test.js
  测试add
    √ 测试1+1 (2ms)
    √ 测试2+2 (1ms)
  测试minus
    √ 测试1-1
    √ 测试2-2

Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        1.918s, estimated 2s
Ran all test suites.
```

5.3 测试覆盖率

- 代码覆盖率是软件测试中的一种度量,描述程序中源代码被测试的比例和程度,所得比例称为代码覆盖率
- 尽量穷尽
- 办界考虑完整
- 简单
- 独立
- 重复执行

类型 说明 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

package.json

```
  "scripts": {
    "test": "jest",
+    "coverage": "jest --coverage"
  },
```

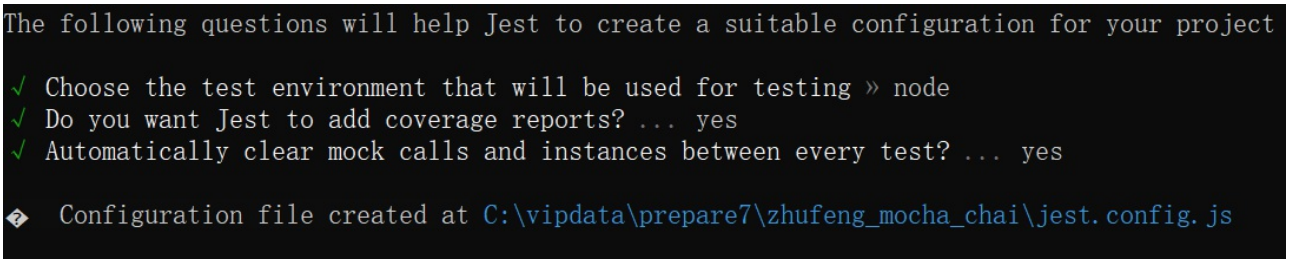
```
npx jest --coverage
npm run coverage
```

```
 PASS  src/math.test.js
  测试add
    √ 测试1+1 (3ms)
    √ 测试2+2
  测试minus
    √ 测试1-1 (1ms)
    √ 测试2-2

----------|----------|----------|----------|----------|-------------------|
File      |  % Stmts | % Branch |  % Funcs |  % Lines | Uncovered Line #s |
----------|----------|----------|----------|----------|-------------------|
All files |       60 |      100 |       50 |       60 |                   |
 math.js  |       60 |      100 |       50 |       60 |              8,11 |
----------|----------|----------|----------|----------|-------------------|
Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        2.199s
Ran all test suites.
```

```
 PASS  src/math.test.js
  测试add
    √ 测试1+1 (2ms)
    √ 测试2+2
  测试minus
    √ 测试1-1
    √ 测试2-2
  测试multiply
    √ 测试1-1
    √ 测试2-2 (1ms)
  测试divide
    √ 测试1-1
    √ 测试2-2

----------|----------|----------|----------|----------|-------------------|
File      |  % Stmts | % Branch |  % Funcs |  % Lines | Uncovered Line #s |
----------|----------|----------|----------|----------|-------------------|
All files |      100 |      100 |      100 |      100 |                   |
 math.js  |      100 |      100 |      100 |      100 |                   |
----------|----------|----------|----------|----------|-------------------|
Test Suites: 1 passed, 1 total
Tests:       8 passed, 8 total
Snapshots:   0 total
Time:        2.186s
Ran all test suites.
```

5.4 配置 #

- 可以通过配置文件configuration (https://jestjs.io/docs/zh-Hans/configuration#defaults)对Jest进行配置



5.4.1 生成配置文件

- 支持测试 ts和 jsx

```
jest --init
yarn add --dev babel-jest  @types/jest @babel/core @babel/preset-env @babel/preset-react @babel/preset-typescript typescript
```

5.4.2 babel.config.js

```
module.exports = {
    presets: [
        [
            '@babel/preset-env',
            {
                targets: {
                    node: 'current',
                }
            }
        ],
        '@babel/preset-typescript',
        '@babel/preset-react'
    ],
};
```

5.4.3 jest.config.js

- jsdom (https://github.com/jsdom/jsdom)是web标准的纯JS实现

jest.config.js默认配置

```
module.exports = {
    testMatch: ["**/__tests__/**/*.[jt]s?(x)","**/?(*.)+(spec|test).[tj]s?(x)"],
    testRegex: [],
    testEnvironment:'jsdom',
    rootDir:null,
    moduleFileExtensions:["js","json","jsx","ts","tsx","node"],
    clearMocks:true
}
```

选项 说明 testMatch 用来检测测试文件的glob模式 testRegex 用来检测测试文件的正则表达式或正则表达式数组 testEnvironment 用来跑测试的测试环境,可以选择 jsdom node

rootDir Jest用来描述测试文件或模块的根目录，默认是 package.json

所在的目录 moduleFileExtensions 使用的模块的文件扩展名数组 clearMocks 在每一次测试时自动清除mock调用和实例 coverageDirectory 输出代码覆盖率的目录

5.4.4 tsconfig.json

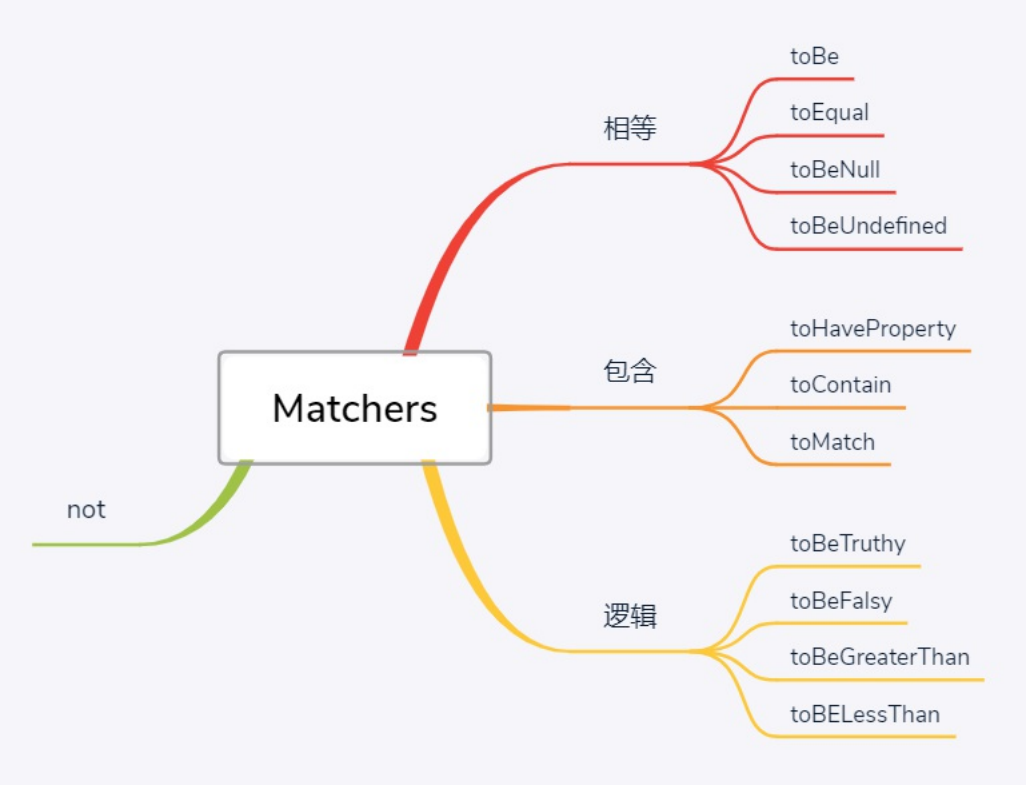
```
tsx --init
```

tsconfig.json

```
{
  "compilerOptions": {
+   "target": "ES2015",
    "module": "commonjs",
+    "jsx": "preserve",
+    "strict": false,
+    "noImplicitAny": false,
+    "strictNullChecks": false,
    "esModuleInterop": true
  }
}
```

5.5 Matchers

- Jest使用匹配器 (https://jestjs.io/docs/zh-Hans/using-matchers)让你可以用各种方式测试你的代码
- 完整的匹配器API (https://jestjs.io/docs/zh-Hans/expect)
- Matchers也可以称之为断言库



5.5.1 matches.tsx

tests\matches.tsx

```
declare var expect;
it('matchers', () => {
    expect(1).toEqual(1);
    expect([1, 2, 3]).toEqual([1, 2, 3]);
    expect(null).toBeNull();
    expect([1, 2, 3]).toContain(1);
    expect({ name: 'zhufeng' }).toHaveProperty('name');
    expect('123').toContain('2');
    expect('123').toMatch(/^\d+$/);
    expect([1, 2, 3]).not.toContain(4);
});
```

5.6 测试DOM

- Jest里面可以直接操作DOM,是因为内置了jsdom (https://github.com/jsdom/jsdom)
- jsdom是一套模拟的DOM环境,可以在浏览器上运行
- 可以修改 `jest.config.js`,设置 `testEnvironment: "jsdom"`

5.6.1 src\domUtils.tsx

src\domUtils.tsx

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

5.6.2 tests\domUtils.tsx

tests\domUtils.tsx

```
import { remove, on } from '../src/domUtils';
declare var describe;
declare var test;
declare var expect;
describe('domUtils', () => {
    test('remove', function () {
        document.body.innerHTML = (
            `

              儿子

            `
        );
        let parent = document.getElementById('parent');
        expect(parent.nodeName.toLocaleLowerCase()).toBe('div');
        const child = document.getElementById('child');
        expect(child.nodeName.toLocaleLowerCase()).toBe('div');
        remove(child);
        expect(document.getElementById('child')).toBeNull();
    });
    test('on', function () {
        document.body.innerHTML = 'click';
        let clickMe = document.getElementById('clickMe');
        on(clickMe, 'click', () => {
            clickMe.innerHTML = 'clicked';
        });
        clickMe.click();
        expect(clickMe.innerHTML).toBe('clicked');
    });
})
```

5.7 异步请求

- asynchronous (https://jestjs.io/docs/zh-Hans/asynchronous)**5.7.1 src\api.tsx #**src\api.tsx

```
export const callCallback = (onSuccess) => {
  setTimeout(() => {
      onSuccess({ code: 0 });
  }, 3000);
}
export const callPromise = () => {
  return new Promise(function (resolve) {
      setTimeout(() => {
          resolve({ code: 0 });
      }, 3000);
  });
}
```

** 5.7.2 tests\api.tsx #**

tests\api.tsx

```
import { callCallback, callPromise } from '../src/api';
declare var describe;
declare var it;
declare var expect;
describe('测试异步接口', () => {
    it('测试 callCallback', (done) => {
        callCallback(result => {
            expect(result.code).toBe(0);
            done();
        });
    });
    it('测试 callPromise', () => {
        return callPromise().then((result: any) => {
            expect(result.code).toBe(0);
        });
    });
    it('测试 callAsync', async () => {
        let result: any = await callPromise();
        expect(result.code).toBe(0);
    });
    it('测试 resolves', async () => {
        expect(callPromise()).resolves.toMatch({ code: 0 });
    });
});
```

5.8 钩子函数

- 钩子函数 (https://jestjs.io/docs/en/api#afterallfn-timeout)对不同测试执行阶段提供了对应的回调接口
- beforeAll 在所有测试用例执行之前执行
- beforeEach 每个测试用例执行前执行
- afterEach 每个测试用例执行结束时
- afterAll 等所有测试用例都执行之后执行
- only (https://jestjs.io/docs/en/api#testonlyname-fn-timeout)的意思是只调用特定的测试用例

```
let counter = 0;
declare var describe;
declare var beforeAll;
declare var beforeEach;
declare var afterEach;
declare var afterAll;
declare var test;
describe('counter测试代码', () => {
    beforeAll(() => {
        console.log('BeforeAll'); counter++;
    })

    beforeEach(() => {
        console.log('BeforeEach'); counter++;
    })

    afterEach(() => {
        console.log('AfterEach'); counter++;
    })

    afterAll(() => {
        console.log('AfterAll'); counter++;
        console.log(counter);
    })

    describe('测试用例', () => {
        test('测试用例1', () => {
            console.log('测试用例1'); counter++;
        });
        test('测试用例2', () => {
            console.log('测试用例2'); counter++;
        });
    });
});
```

5.9 mock

- mock-functions (https://jestjs.io/docs/zh-Hans/mock-functions)可以擦除函数的实际实现来测试代码
- 使用 `mock function`可以查看函数的调用次数以及入参的情况
- 原生的定时器函数并不是很方便测试,我们通过jest.useFakeTimers() (https://jestjs.io/docs/zh-Hans/timer-mocks)来模拟定时器函数

** 5.9.1 src\mock.tsx #**

src\mock.tsx

```
import axios from 'axios';
export function exec(callback) {
    callback('123');
    callback('456');
}
export function createInstance(ClassName) {
    return new ClassName();
}
export function getData() {
    return axios.get('/api/users');
}
export function delay(callback, ms) {
    setTimeout(() => {
        callback(ms);
    }, ms);
}
```

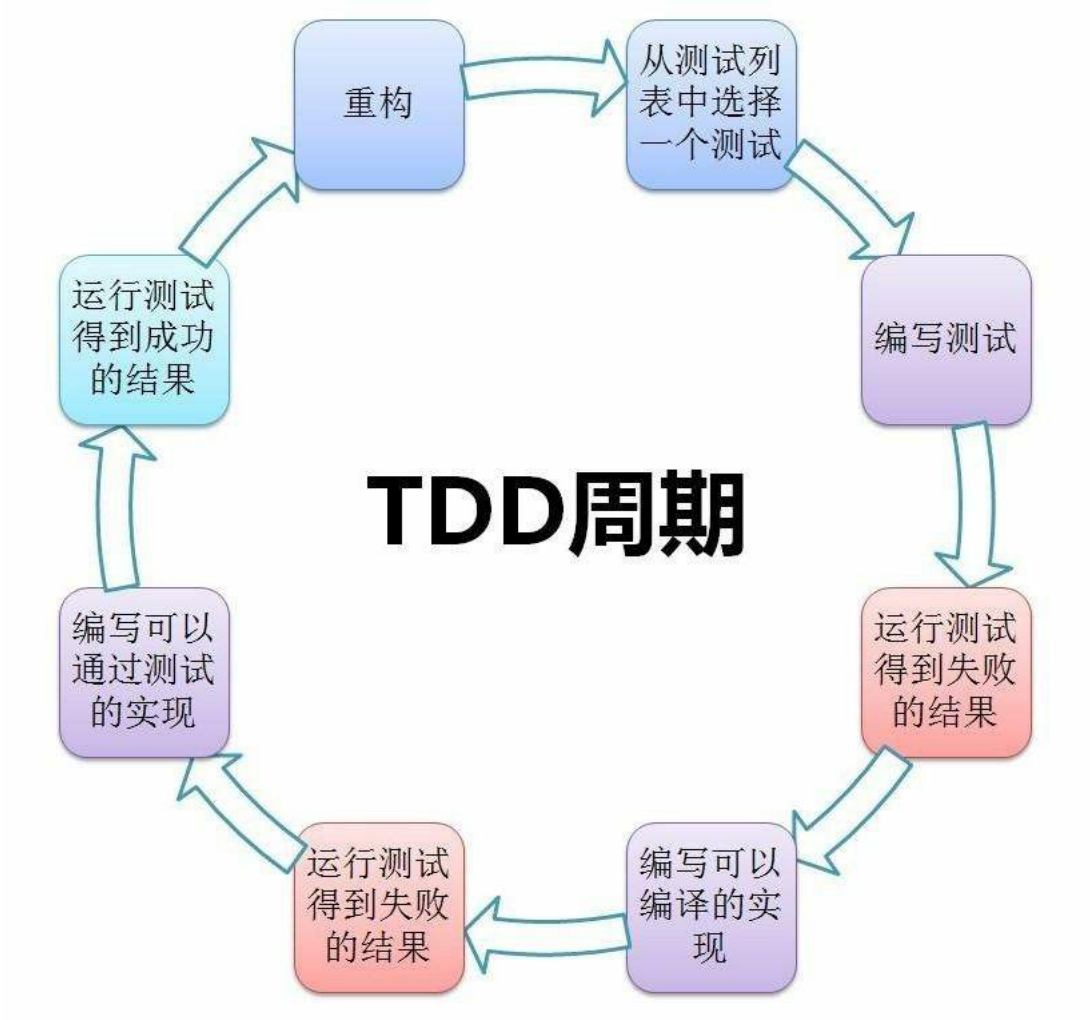
** 5.9.2 tests\mock.tsx #**

tests\mock.tsx

```
jest.mock('axios');
import { exec, createInstance, getData, delay } from '../src/mock';
import axios from 'axios';
declare var jest;
declare var expect;
test('测试exec', () => {

    let callback = jest.fn();
    callback.mockReturnValueOnce('abc');
    callback.mockReturnValueOnce('def');
    exec(callback);
    expect(callback).toBeCalled();
    expect(callback).toBeCalledTimes(2);
    expect(callback).toBeCalledWith('123');
});
test('测试createInstance', () => {

    let callback = jest.fn(function (this: any) {
        this.name = 'zhufeng';
    });
    createInstance(callback);
});

test('测试getData', async () => {
    (axios.get as any).mockResolvedValue({ data: { code: 0 } });
    let result = await getData();
    console.log('result', result);
    expect(result.data).toEqual({ code: 0 });
});
jest.useFakeTimers();
test('测试delay', (done) => {
    delay((result) => {
        expect(result).toBe(1000);
        done();
    }, 1000);
    jest.runAllTimers();
});
```

6. TDD(Test-Driven-Development)

- 测试驱动开发模式 需求分析->拆分模块->编写测试用例->编写代码通过测试用例->重构->发布
- 重构时可以减少BUG、提高代码质量、提高可维护性



6.1 创建项目 #

项目预览 (http://img.zhufengpeixun.cn/msg.html)

```
create-react-app  zhufeng_message_app --typescript
cd zhufeng_message_app
cnpm install --save-dev enzyme @types/enzyme enzyme-adapter-react-16 @types/enzyme-adapter-react-16 -D
yarn start
```

6.2 环境准备

** 6.2.1 src\react-app-env.d.ts #**

src\react-app-env.d.ts ** 6.2.2 src\setupTests.ts #**

src\setupTests.ts

```
import '@testing-library/jest-dom/extend-expect';
import Enzyme from 'enzyme';
import Adaptor from 'enzyme-adapter-react-16';
Enzyme.configure({ adapter: new Adaptor() });
```

** 6.2.3 tsconfig.json #**

tsconfig.json

```
{
  "compilerOptions": {
    "jsx": "react",
+    "noImplicitAny": false
  }
}
```

6.3 Message.tsx

- shallow (https://airbnb.io/enzyme/docs/api/shallow.html)
- mount (https://airbnb.io/enzyme/docs/api/mount.html)
- expect-enzyme (http://npm.taobao.org/package/expect-enzyme)

** 6.3.1 测试用例 #**

src__tests__\components\Message.tsx

```
import React from 'react';
import Message from '../../components/Message';
import { shallow } from 'enzyme';
describe('测试Message', () => {
    test('应该渲染出来一个li,类名list-group-item,内容是zhufeng', () => {
        let message = { id: '1', content: 'zhufeng' };
        const wrapper = shallow(<Message message={message} />);
        const li = wrapper.find(`li`);
        expect(li).toHaveLength(1);
        expect(li.prop('className')).toBe('list-group-item');
        expect(li.prop('children')).toContain('zhufeng');
    });
});
```

** 6.3.2 重构实现 #**

src\components\Message.tsx

```
import React from 'react';
export default function (props) {
    let { content } = props.message;
    return (
        <li className="list-group-item" >{content}li>
    )
}
```

6.4 MessageList.tsx

** 6.4.1 测试用例 #**

src__tests__\components\MessageList.tsx

```
import React from 'react';
import MessageList from '../../components/MessageList';
import { mount } from 'enzyme';
describe('测试MessageList', () => {
  test('传入2个留言,应该渲染出来2个li', () => {
    let messages = [{ id: '1', content: 'zhufeng' }, { id: '2', content: 'jiagou' }];
    const wrapper = mount(<MessageList messages={messages} />);
    const listItems = wrapper.find(`li`);
    expect(listItems).toHaveLength(2);
    expect(listItems.at(0).text()).toContain('zhufeng');
    expect(listItems.at(1).text()).toContain('jiagou');
  });
});
```

** 6.4.2 重构实现 #**

src\components\MessageList.tsx

```
import React from 'react';
import Message from './Message';
export default function (props) {
    let messages = props.messages;
    return (
        <ul className="list-group">
            {
                messages.map((message) => (
                    <Message message={message} key={message.id} />
                ))
            }
        ul>
    )
}
```

6.5 MessageForm.tsx

** 6.5.1 测试用例 #**

src__tests__\components\MessageForm.tsx

```
import React from 'react';
import MessageForm from '../../components/MessageForm';
import { shallow, mount } from 'enzyme';
describe('测试MessageForm', () => {
    test('应该渲染出来一个表单,表单里有input和button', () => {
        let addMessage = jest.fn();
        const wrapper = shallow(<MessageForm addMessage={addMessage} />);
        const form = wrapper.find(`form`);
        const input = wrapper.find(`input`);
        const button = wrapper.find(`button`);
        expect(form).toHaveLength(1);
        expect(input).toHaveLength(1);
        expect(button).toHaveLength(1);
    });
    test('在输入框里输入内容,如果内容为空点击提交按钮不会添加留言', () => {
        let addMessage = jest.fn();
        const wrapper = mount(<MessageForm addMessage={addMessage} />);
        const input = wrapper.find(`input`);
        const button = wrapper.find(`button`);
        const newValue = '';
        input.simulate('change', { target: { value: newValue } });
        button.simulate('click');
        expect(addMessage).not.toHaveBeenCalled();
    });
    test('在输入框里输入内容,如果内容不为空点击提交按钮会添加留言', () => {
        let addMessage = jest.fn();
        const wrapper = mount(<MessageForm addMessage={addMessage} />);
        const input = wrapper.find(`input`);
        const button = wrapper.find(`button`);
        const newValue = '新留言';
        input.simulate('change', { target: { value: newValue } });
        button.simulate('click');
        expect(addMessage).toHaveBeenLastCalledWith(newValue)
    });
});
```

** 6.5.2 重构实现 #**

src\components\MessageForm.tsx

```
import React, { useState, useCallback } from 'react';
export default function (props) {
    let [content, setContent] = useState('');
    const handleSubmit = useCallback((event) => {
        event.preventDefault();
        if (content) {
            props.addMessage(content);
            setContent('');
        }
    }, [content])
    return (
        <form>
            <div className="form-group">
                <input type="text"
                    value={content}
                    onChange={event => setContent(event.target.value)}
                    className="form-control"
                    placeholder="请输入内容"
                />
            div>
            <div className="form-group">
                <button type="button" className="btn btn-primary" onClick={handleSubmit}>
                    发表
            button>
            div>
        form>
    )
}
```

6.6 MessageApp.tsx

src__tests__\components\MessageApp.tsx
** 6.6.1 测试用例 #**

```
import React from 'react';
import MessageApp from '../../components/MessageApp';
import MessageList from '../../components/MessageList';
import MessageForm from '../../components/MessageForm';
import { mount } from 'enzyme';
describe('测试MessageApp', () => {
    test('应该渲染出来一个面板', () => {
        const wrapper = mount(<MessageApp />);
        const container = wrapper.find(`.container`);
        const panel = wrapper.find(`.panel.panel-default`);
        const panelHeading = wrapper.find(`.panel-heading`);
        const panelBody = wrapper.find(`.panel-body`);
        const panelFooter = wrapper.find(`.panel-footer`);
        expect(container).toHaveLength(1);
        expect(panel).toHaveLength(1);
        expect(panelHeading).toHaveLength(1);
        expect(panelBody).toHaveLength(1);
        expect(panelFooter).toHaveLength(1);
    });
    test('默认状态是空数组', () => {
        const wrapper = mount(<MessageApp />);
        expect(wrapper.state()).toMatchObject({ messages: [] });
    });
    test('MessageList组件存在，并且给MessageList传递messages属性', () => {
        const wrapper = mount(<MessageApp />);
        const messageList = wrapper.find(MessageList);
        expect(messageList.prop('messages')).toBe((wrapper.instance() as MessageApp).state.messages);
    });
    test('MessageForm组件存在，并且给MessageForm传递addMessage属性', () => {
        const wrapper = mount(<MessageApp />);
        const messageForm = wrapper.find(MessageForm);
        expect(messageForm.prop('addMessage')).toBe((wrapper.instance() as MessageApp).addMessage);
    });
    test('点击提交按钮添加条目的时候，应该可以改变在MessageApp的state中添加一个新条目', () => {
        let wrapper = mount(<MessageApp />);
        let messageList = wrapper.find(MessageList);
        let messageForm = wrapper.find(MessageForm);
        expect(wrapper.state('messages')).toHaveLength(0);
        let content = '我想你';
        const input = messageForm.find(`input`);
        const button = messageForm.find(`button`);
        input.simulate('change', { target: { value: content } });
        button.simulate('click');
        expect(wrapper.state('messages')).toHaveLength(1);
        let newMessages = [{ id: expect.any(String), content }]
        expect(wrapper.state('messages')).toEqual(newMessages);
        messageList = wrapper.find(MessageList);
        expect(messageList.prop('messages')).toEqual(newMessages);
    });
});
```

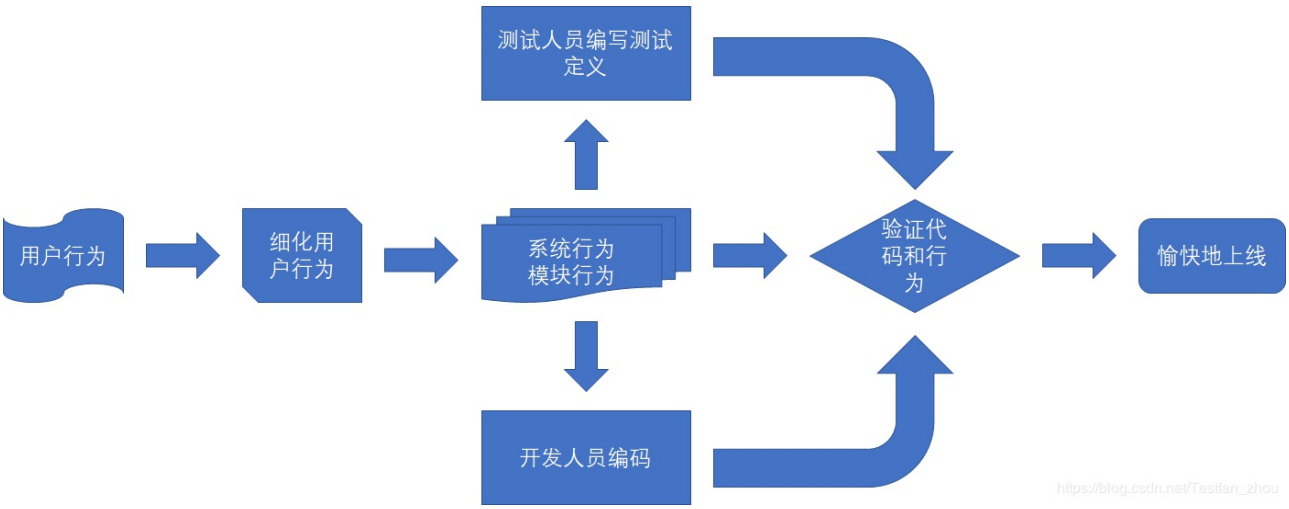
** 6.5.2 重构实现 #**

src\components\MessageApp.tsx

```
import React from 'react';
import MessageList from './MessageList';
import MessageForm from './MessageForm';
export default class MessageApp extends React.Component {
    state = { messages: [] }
    addMessage = (content: string) => {
        let newMessage = { id: Date.now() + '', content };
        this.setState({ messages: [...this.state.messages, newMessage] });
    }
    render() {
        return (
            <div className="container" style={{ marginTop: 50 }}>
                <div className="col-md-8 col-md-offset-2">
                    <div className="panel panel-default">
                        <div className="panel-heading">
                            <h1 style={{ textAlign: 'center' }}>珠峰留言版h1>
                        div>
                        <div className="panel-body">
                            <MessageList messages={this.state.messages} />
                        div>
                        <div className="panel-footer">
                            <MessageForm addMessage={this.addMessage} />
                        div>
                    div>
                div>
            div>
        )
    }
}
```

7. BDD(Behavior Driven Development)

- BDD(Behavior Driven Development)行为驱动开发是一种敏捷软件开发的技术，它鼓励软件项目中的开发者、QA和非技术人员或商业参与者之间的协作



7.1 创建项目

项目预览 (http://img.zhufengpeixun.cn/counter.html)

```
create-react-app  zhufeng_message_app --typescript
cd zhufeng_message_app
cnpm install --save-dev enzyme @types/enzyme enzyme-adapter-react-16 @types/enzyme-adapter-react-16 -D
yarn start
```

7.2 环境准备

** 7.2.1 src\react-app-env.d.ts #**

src\react-app-env.d.ts ** 7.2.2 src\setupTests.ts #**

src\setupTests.ts

```
import '@testing-library/jest-dom/extend-expect';
import Enzyme from 'enzyme';
import Adaptor from 'enzyme-adapter-react-16';
Enzyme.configure({ adapter: new Adaptor() });
```

** 7.2.3 tsconfig.json #**

tsconfig.json

```
{
  "compilerOptions": {
    "jsx": "react",
+    "noImplicitAny": false
  }
}
```

7.3 开发功能

** 7.3.1 index.tsx #**

src\index.tsx

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './containers/App';
ReactDOM.render(
    <App />
    , document.getElementById('root'));
```

** 7.3.2 containers\App.tsx #**

src\containers\App.tsx

```
import React from 'react';
import Counter1 from '../components/Counter1';
import Counter2 from '../components/Counter2';
import store from '../store';
import { Provider } from 'react-redux';
import { BrowserRouter as Router, Route, Link } from 'react-router-dom';
import Header from '../components/Header';
export default function (props) {
    return (
        <Provider store={store}>
            <Router>
                <Header />
                <Route path="/counter1" component={Counter1} />
                <Route path="/counter2" component={Counter2} />
            Router>
        Provider>
    )
}
```

** 7.3.3 components\Counter1.tsx #**

src\components\Counter1.tsx

```
import React, { Component } from 'react';
import actions from '../store/actions/counter1';
import { connect } from 'react-redux';
class Counter1 extends Component<any> {
    render() {
        return (
            <div>
                <p>Counter1:{this.props.number}p>
                <button id="counter1-increment" onClick={this.props.increment1}>+button>
                <button id="counter1-decrement" onClick={this.props.decrement1}>-button>
            div>
        )
    }
}

let mapStateToProps = state => state.counter1
export default connect(
    mapStateToProps,
    actions
)(Counter1)
```

** 7.3.4 components\Counter2.tsx #**

src\components\Counter2.tsx

```
import React, { Component } from 'react';
import actions from '../store/actions/counter2';
import { connect } from 'react-redux'
class Counter2 extends Component<any> {
    render() {
        return (
            <div>
                <p>Counter2:{this.props.number}p>
                <button id="counter2-increment" onClick={this.props.increment2}>+button>
                <button id="counter2-decrement" onClick={this.props.decrement2}>-button>
            div>
        )
    }
}

let mapStateToProps = state => state.counter2
export default connect(
    mapStateToProps,
    actions
)(Counter2)
```

src\components\Header.tsx

```
import React from 'react';
import { withRouter } from 'react-router-dom';
function Header(props) {
    return (
        <ul>
            <li><button id="counter1" onClick={() => props.history.push('/counter1')}>counter1button>li>
            <li><button id="counter2" onClick={() => props.history.push('/counter2')}>counter2button>li>
        ul>
    )
}
export default withRouter(Header);
```

** 7.3.6 store\index.tsx #**

src\store\index.tsx

```
import { createStore } from 'redux';
import combinedReducer from './reducers';
const store = createStore(combinedReducer);
export default store;
```

** 7.3.7 store\action-types.tsx #**

src\store\action-types.tsx

```
export const INCREMENT1 = 'INCREMENT1';
export const DECREMENT1 = 'DECREMENT1';

export const INCREMENT2 = 'INCREMENT2';
export const DECREMENT2 = 'DECREMENT2';

export const RESET = 'RESET';
```

** 7.3.8 store\reducers\index.tsx #**

src\store\reducers\index.tsx

```
import counter1 from './counter1';
import counter2 from './counter2';
import { combineReducers } from 'redux';
let reducers = {
    counter1,
    counter2
}
let combinedReducer = combineReducers(reducers);
export default combinedReducer;
```

** 7.3.9 store\reducers\counter1.tsx #**

src\store\reducers\counter1.tsx

```
import * as types from '../action-types';
let initialState = { number: 0 }
export default function (state = initialState, action) {
    switch (action.type) {
        case types.INCREMENT1:
            return { number: state.number + 1 };
        case types.DECREMENT1:
            return { number: state.number - 1 };
        case types.RESET:
            return initialState;
        default:
            return state;
    }
}
```

** 7.3.10 store\reducers\counter2.tsx #**

src\store\reducers\counter2.tsx

```
import * as types from '../action-types';
let initialState = { number: 0 }
export default function (state = initialState, action) {
    switch (action.type) {
        case types.INCREMENT2:
            return { number: state.number + 1 };
        case types.DECREMENT2:
            return { number: state.number - 1 };
        case types.RESET:
            return initialState;
        default:
            return state;
    }
}
```

** 7.3.11 store\actions\counter1.tsx #**

src\store\actions\counter1.tsx

```
import * as types from '../action-types';
export default {
    increment1() {
        return { type: types.INCREMENT1 };
    },
    decrement1() {
        return { type: types.DECREMENT1 };
    }
}
```

** 7.3.12 store\actions\counter2.tsx #**

src\store\actions\counter2.tsx

```
import * as types from '../action-types';
export default {
    increment2() {
        return { type: types.INCREMENT2 };
    },
    decrement2() {
        return { type: types.DECREMENT2 };
    }
}
```

** 7.3.13 index.html #**

```
+  CounterApp
```

7.4 编写测试用例

** 7.4.1 reducers\counter1.spec.tsx #**

src\store\reducers\counter1.spec.tsx

```
import counter1 from './counter1';
import * as types from '../action-types';

describe('测试counter1', () => {
    it('counter1存在', () => {
        expect(counter1).toBeTruthy();
    })

    it('获取默认值', () => {
        expect(counter1(undefined, { type: '@@REDUX/INIT' })).toEqual({ number: 0 });
    })

    it('INCREMENT1可以让number加1', () => {
        expect(counter1({ number: 0 }, { type: types.INCREMENT1 })).toEqual({ number: 1 });
    })
    it('DECREMENT1可以让number加1', () => {
        expect(counter1({ number: 0 }, { type: types.DECREMENT1 })).toEqual({ number: -1 });
    })
})
```

** 7.4.2 actions\counter1.spec.tsx #**

src\store\actions\counter1.spec.tsx

```
import actions from './counter1';
import * as types from '../action-types';
describe('测试counter1', () => {
    it('counter1存在', () => {
        expect(actions.increment1).toBeTruthy();
    })
    it('counter1返回值是{ type: types.INCREMENT1 }', () => {
        let mockCounter1 = jest.fn(actions.increment1);
        mockCounter1();
        expect(mockCounter1).toHaveReturnedWith({ type: types.INCREMENT1 });
    })
})
```

** 7.4.3 store\store.spec.ts #**

src\store\store.spec.ts

```
import store from './';
import * as types from './action-types';
describe('测试store', () => {
    beforeEach(() => {
        store.dispatch({ type: types.RESET });
    });
    it('store存在', () => {
        expect(store).toBeTruthy();
    })
    it('store应该有默认的状态', () => {
        expect(store.getState()).toMatchObject({
            counter1: { number: 0 },
            counter2: { number: 0 }
        });
    })
    it('派发INCREMENT1动作后后仓库counter1状态应该变为{ number: 1 }', () => {
        store.dispatch({ type: types.INCREMENT1 });
        expect(store.getState()).toMatchObject({
            counter1: { number: 1 },
            counter2: { number: 0 }
        });
    })
    it('派发DECREMENT1动作后后仓库counter1状态应该变为{ number: -1 }', () => {
        store.dispatch({ type: types.DECREMENT1 });
        expect(store.getState()).toMatchObject({
            counter1: { number: -1 },
            counter2: { number: 0 }
        });
    })
})
```

** 7.4.4 tests\containers\Counter1.tsx #**

src__tests__\containers\Counter1.tsx

```
import React from 'react';
import Counter1 from '../../components/Counter1';
import { shallow, mount, render } from 'enzyme';
import { Link } from 'react-router-dom';
import store from '../../store';
import { Provider } from 'react-redux';
describe('测试Counter1', () => {
    test(`我想要显示一个数字，点+变成1，再点-变成0`, () => {
        let wrapper = mount(<Provider store={store}> <Counter1 /> Provider >);
        let p = wrapper.find('p');
        expect(p).toHaveLength(1);
        expect(p.text()).toContain('0');

        let button = wrapper.find('button');
        expect(button).toHaveLength(2);

        button.at(0).simulate('click');
        p = wrapper.find('p');
        expect(p.text()).toContain('1');

        button.at(1).simulate('click');
        p = wrapper.find('p');
        expect(p.text()).toContain('0');
    });
});
```

** 7.4.5 tests\containers\App.tsx #**

src__tests__\containers\App.tsx

```
import React from 'react';
import App from '../../containers/App';
import { shallow, mount, render } from 'enzyme';
import { Link } from 'react-router-dom';
describe('测试App', () => {
    test(`测试App`, () => {
        let wrapper = mount(<App history={history} />);
        let p = wrapper.find('p');
        expect(p).toHaveLength(0);
        let button = wrapper.find('button');
        button.at(0).simulate('click');
        p = wrapper.find('p');
        expect(p.text()).toContain('Counter1');
        button.at(1).simulate('click');
        p = wrapper.find('p');
        expect(p.text()).toContain('Counter2');
    });
});
```

8. UI测试

- 用puppeteer做集成测试的时候测试用例是真正在真实的浏览器上执行的

8.1 puppeteer

- 使用puppeteer控制chromium
- puppeteer是Chrome团队开发的一个node库
- 可以通过api来控制浏览器的行为，比如点击，跳转，刷新，在控制台执行js脚本等等
- 通过这个工具可以用来写爬虫，自动签到，网页截图，生成pdf，自动化测试等

```
cnpm install --save-dev jest-puppeteer puppeteer jest
```

```
let puppeteer = require('puppeteer');
const delay = (ms) => new Promise(function (resolve) {
    setTimeout(() => {
        resolve();
    }, ms);
});
(async () => {
    const browser = await puppeteer.launch({ headless: false });
    const page = await browser.newPage();
    await page.goto('http://localhost:3000');
    await page.click('button[id="counter1"]');
    await delay(1000);
    await page.click('button[id="counter1-increment"]');
    await delay(1000);
    await page.click('button[id="counter2"]');
    await delay(1000);
    await page.click('button[id="counter2-decrement"]');
    await browser.close();
})();
```

8.2 jest-puppeteer

- jest-puppeteer (https://github.com/smooth-code/jest-puppeteer)
- 切记删除 testEnvironment配置

** 8.2.1 jest.config.js #**

```
{
+  "preset": "jest-puppeteer"
}
```

** 8.2.2 CounterApp.ts #**

```
declare var beforeAll;
declare var page;
describe('测试CounterApp', () => {
    beforeAll(async () => {
        await page.goto('http://localhost:3000/');
    });

    it('标题是CounterApp', async () => {
        await expect(page.title()).resolves.toMatch('CounterApp');
    });

    it('点击第一个button会跳转到/counter1', async () => {
        await page.click('button[id="counter1"]');
        let text = await page.$eval('p', el => el.innerText);
        expect(text).toBe('Counter1:0');
        await page.click('button[id="counter2"]');
        text = await page.$eval('p', el => el.innerText);
        expect(text).toBe('Counter2:0');
    });
});
```

9.参考

9.1 jest

- jest matchers (https://jestjs.io/docs/zh-Hans/using-matchers)
- jest断言 (https://jestjs.io/docs/zh-Hans/expect#expectanything)
- jest异步 (https://jestjs.io/docs/zh-Hans/asynchronous)

9.2 enzyme

- enzyme (https://airbnb.io/enzyme)
- shallow (https://airbnb.io/enzyme/docs/api/shallow.html)
- mount (https://airbnb.io/enzyme/docs/api/mount.html)
- expect-enzyme (http://npm.taobao.org/package/expect-enzyme)
- selector (https://airbnb.io/enzyme/docs/api/selector.html)

9.3 puppeteer

- puppeteer api (https://github.com/puppeteer/puppeteer/blob/master/docs/api.md)

- jest-puppeteer (https://github.com/smooth-code/jest-puppeteer)
- expect-puppeteer (https://github.com/smooth-code/jest-puppeteer/blob/master/packages/expect-puppeteer/README.md#api)
- puppeteer-api-zh_CN (https://zhaoqize.github.io/puppeteer-api-zh_CN/#/)

9.4 glob

- glob返回匹配指定模式的文件名或目录**9.4.1 glob规则#*

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1次，类似于js正则中的
** 9.4.2 glob示例 #**

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
/

*l.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