vue项目单元测试宝典

# 序

## 关于本文

本文的制作初衷是让读者能完成任何vue项目的单元测试，具体有两种方式：

1. 通读本文在理解单元测试思路的基础上，对自己的项目进行组件分拆整理然后写单元测试（推荐）
2. 只通过本文中所列出的例子，对应到自己的项目中，相同部分写相同的单元测试（不推荐），但是再NB的教程也不可能把实际项目中的所有情况都列举出来，所以应该重在理解单元测试思路。另外强调一点：所有情况都可以测试，所有可能性都需要测试

就国内的开发环境而言，要求写测试的项目不多，甚至很少。这也直接导致很大一部分的程序员根本不会写单元测试，而且网上很多的技术博客中对测试都是一笔带过，意思就是不会写测试也不影响写代码做开发，这话没毛病，如果你对自己的要求只是能写代码做开发就够了的话，本文以下的内容将对你毫无意义。

既然你选择了继续往下看，说明你是一个有理想的码农，本文也将会从单元测试方面为实现你的理想添砖加瓦。

那么会写单元测试和不会写单元测试的程序员到底有什么区别呢？有句俗话叫无知者无畏，当我不会写测试的时候，我以为开发不过就是做出客户需要的功能和网页而已，具体怎么做，怎么实现全是我自己说了算，（虽然有代码规范，但是细节上有很大发挥空间）甚至一些组件的写法都是随心所欲，还自诩为个性写法。直到我学会了单元测试我才发现以前的自己是多么可笑，一切开发都应该做到一致性，不存在个性更不能随心所欲，就个人而言单元测试对我的提升包括以下方面

学会一项技能：单元测试

保证了代码质量，提升了组件可复用性，

优化组件构成和代码结构

规范代码写法

项目结构清晰明了，非前端人员也能明白每个文件夹里是什么，每个文件的作用是什么

增加对自己写的代码的自信

相信单元测试对每个人都会有不同程度的提升，所以请坚定信念继续前行。

本文中所提到的测试均是指单元试，e2e本文中不做讨论。

本文是笔者根据个人开发经验总结所著，参考了vue官方测试插件以及很多网络资源和书籍，在此强调，均是原著，所参考资源都是网上公开知识。

测试工具有几种可以选择：Jest，Mocha，Karma，本文使用的是Jest（其他也可用）

理由有如下几点：

性能非常好，

用法非常简单，快速上手，

容易安装和运行，无需任何配置

自带覆盖率统计工具

可以在沙盒环境运行

自动watch你的代码变动并运行测试

自动mock函数

其他测试框架都没有的快照（snapshot）测试

非常简单地就能测试异步代码

Vue,Angular,React框架等等都能用

## 适用读者

本文的想定读者：对vue的基础知识有一定了解，使用vue作为项目开发框架，不会写单元测试或者从没写过单元测试，且项目需要写单元测试、或者对自己要求较高，想要给自己的代码写单元测试的有志骚年。

## 开发环境（笔者代码环境）

OS：windows10

Vue-Cli：v3.3.0

Node.js: v8.12.0

Npm: v6.4.1

"vue": "^2.6.6",

"vue-router": "^3.0.1",

"vuex": "^3.0.1"

"axios": "^0.18.0",

"@vue/cli-plugin-unit-jest": "^3.4.0"

本文中示例代码为vue SPA项目

github仓库地址：https://github.com/holylovelqq/vue-project-unitTest-with-jest

## 特别说明

1. 免责声明。作者不能保证每个人都能喜欢文中的单元测试方式或者代码，如有不足欢迎指出，实际是否适用请根据项目组或个人需求斟酌使用，对于因使用本文中知识所造成的一切损失，均由使用者自己负责，笔者概不负责
2. 版权所有。本文仅供个人学习所用，未经笔者同意不得用于商业用途，违者必究。

# 为什么要写单元测试

【为什么要写单元测试】【有必要写单元测试么】【单元测试有什么用】这种问题就不在这里详细解释了，只要百度一下就会有很多很多回答，答案肯定是非常建议你写测试，但是你要是就是不写别人也没办法，不过总有一天你还是会写的，不然你不会看本文。

这里借用vue官网上的话（不知道读者朋友们在使用vue的时候有没有注意到）

组件的单元测试有很多好处：

1. 提供描述组件行为的文档
2. 节省手动测试的时间
3. 减少研发新特性时产生的 bug
4. 改进设计
5. 促进重构

自动化测试使得大团队中的开发者可以维护复杂的基础代码。

特别要提出vue有官方的单元测试组件库，如果是没必要的东西，何必花力气来维护一个组件库呢。

另外就个人而言，测试最大的意义在于不希望自己写的代码对别人来说就是一坨shit，这里说的别人包括三个月后的自己，如果自己看到三个月前自己写的代码都要抑制不住的想骂人的话，难道还想让别人夸你么？其实如果只是骂几句的话也不是什么大事，最大的问题是浪费时间、降低效率、破坏心情……花三个月写出来的代码，别人要花四个月才能看明白，这是一种什么操作？说你做的无用功吧，可代码真真就摆在这里的。

说再多其实就是要肯定单元测试的必要性，下面会详细介绍组件的分类和如何写单元测试

# 易于测试的组件与组件分类

## 易于测试的组件

易于测试的组件一般都有如下特征

* 组件内功能尽可能的少
* 组件内的依赖尽可能的少
* 组件尽可能不会根据data或者props而发生大的变化

与上面三点相反就是不易测试的组件，实际开发中应尽量避免

## 组件分类

组建分类不仅有利于测试，更重要的是性能优化，组件尽可能分类细化有利于快速更新视图、渲染页面、提高用户体验

前端组件分类一般分为两类：Presentation Component和Container Component

* Presentation Component

这类组件的主要功能是DOM展示，数据一般是通过props获取数据，然后展示出来

基本特征：

* 不依赖其他组件，与store和router无关联
* 不进行CRUD操作（如果必要可使用$emit在夫组件内操作）
* 尽量不使用生命周期函数
* 尽量具有可复用性
* Container Component

这类组件的主要功能是数据操作（API联调）

基本特征：

* 子组件的状态改变（props、$emit）
* CRUD操作、store内数据操作、router切换

以上这两类组件是项目开发中定义组件的思想，实际项目中区分这两种组件还是有一定的困难的，主要还是要看项目的规模及组件的使用，下一章文件树结构会更详细的按照文件夹内所存放的组件来讲解

# 文件树的构成

笔者一般时候的文件树构成如下

-src

-assets -------------CSS，JS，image，icon

-basics--------------基础组件（不含其他组件，便于复用）

-components-------大型组件（含其他组件，会被复用）

-containers---------容器组件（包含其他组件，不会被复用，数据处理，通常用于复杂页面分担views压力）

-views---------------页面组件（路由切换使用，展示页面用，数据处理）

-App.vue

-main.js

.

.

.

.

下面相应讲解一下每个文件夹内对应的组件类别

Bsics文件夹内存放的一般是最小的组件单元和可复用的组件，比如：按钮组件AppButton.vue、输入框组件AppInput.vue 等等，属于presentation component

Components文件夹内存放的是集合了basics或者components组件的父组件，属于presentation component

Containers文件夹内存放的是比自身组件小的组件basics或者components组件的父组件，主要是给子组件传递props等数据，属于containers component

views文件夹内存放的是页面组件，内涵页面内所有子组件，主要是操作数据和路由切换，属于containers component

以上几类文件夹内会根据项目需求创建子文件和子文件夹

# vue项目中哪些需要测试

Vue组件的基本模板主要有三大标签组成：template，script，Style（字符串模板不在讨论范围内），理论上来讲，这三大标签内的所有项都应该进行测试，但是本文涉及的测试是使用jest对template和script两大标签的内容进行测试，如果要测试style内的CSS样式的话，可以使用storybook，不在本文的讨论范围内，有兴趣的可以单独联系

下面就详细的介绍两大标签内可能会出现的内容

## Template标签

本标签内主要是DOM结构，可以使用jest的snapshot功能进行测试，也可进行精确测试，即断言某DOM元素存在与否

v-on（依赖函数存在于script标签内）

v-bind（依赖变量存在于script标签内）

slot（vue2.6.0以上版本更新了此语法，本文完成时并未影响到测试）

v-if/v-show（依赖变量存在于script标签内）

filter（依赖函数存在于script标签内）

测试要配合script标签的内容进行

## Script标签

本标签内主要是逻辑实现可能出现的内容有

Props（⭐必须进行测试）

Data（一般配合v-bind/v-if/v-show等一起测试）

Filters（⭐必须进行测试）

Methods（⭐必须进行测试）

$emit（⭐必须进行测试）

eventHub（空vue实例，用于分发事件，在使用的组件内进行测试）

watch（⭐必须进行测试）

computed（⭐必须进行测试）

router（⭐必须进行测试）

axios（⭐必须进行测试）

vuex（⭐必须进行测试）

其他插件：element-ui/momentJS等插件本身不需要我们进行测试，开发者已经测试过，但是被测试组件如果使用了此插件的话，需要我们挂载在临时vue上，避免测试时报错

# 从零开始写测试

## 安装开发环境

参见序中的开发环境，示例代码再此环境下完美执行，建议各软件版本不低于此开发环境，最好是相同版本，避免版本不同导致的bug

## 创建项目

本文假定您已了解vue的基础知识，并能够使用vuecli创建项目，记得选中Unit Testing并选择Jest（2.x版本的vuecli需要单独安装@vue/test-utils）。

（如确实不会的话，参见：<https://cli.vuejs.org/zh/guide/creating-a-project.html#vue-create>）

且读到此处默认您已经阅读完了第二三章，并对已有的项目目录进行了调整，对组件也进行了相应的分割（如果没有建议您看一下第二三章，或者看一下笔者的github代码），虽然您不进行这些操作也不影响写测试。但是您的代码可能会看起来像一坨shit，包括测试代码。当然您也可以边写测试边分割组件，如果您大脑够清晰的话。不管如何，建议您先将项目结构清晰化，组件尽可能地优化，易于测试

## 写单元测试

原则上讲单元测试要做到100%覆盖率，但有些时候很难达到，所以在实际中要根据项目情况，尽可能的趋近于100%

Github仓库内的示例项目的覆盖率是100%，测试的写法可以参考，下一章将会根据示例代码详细解说单元测试的写法

# 可能的需要测试项目及测试方法（如有不足欢迎在github提issue）

## 常用的断言语句

1. toBe() 测试具体的值
2. toEqual() 测试对象类型的值
3. toBeCalled() 测试函数被调用
4. toHaveBeenCalledTimes() 测试函数被调用的次数
5. toHaveBeenCalledWith() 测试函数被调用时的参数
6. toBeNull() 结果是null
7. toBeUndefined()结果是undefined
8. toBeDefined()结果是defined
9. toBeTruthy() 结果是true
10. toBeFalsy() 结果是false
11. toContain() 数组匹配，检查是否包含
12. toMatch() 匹配字符型规则，支持正则
13. toBeCloseTo() 浮点数
14. toThrow() 支持字符串，浮点数，变量
15. toMatchSnapshot() jest特有的快照测试
16. .not.+matcher，eg. .not.toBe() 前面加上.not就是否定形式，

以上只是一部分matcher，需要更多请查看jest官方文档 <https://jestjs.io/docs/zh-Hans/expect#expectextendmatchers>

## 测试的写法

以下根据上面提到的需要测试的项目，一一详解测试的写法

1. Snapshot　测试

// 测试内容：snapshot->概括的测试DOM结构

// 如果组件内存在比较特殊的需要测试的DOM结构的话，可以单独测试

it('matches snapshot', () => {

const wrapper = shallowMount(AppButton)

expect(wrapper.html()).toMatchSnapshot()

wrapper.destroy()

})

1. DOM测试

// 测试内容：精准DOM结构测试示例

it('DOM test', () => {

const wrapper = shallowMount(AppButton)

expect(wrapper.contains('button')).toBeTruthy()

wrapper.destroy()

})

1. v-on　：以click事件为例，测试思路：点击，断言对应函数是否被触发，出发几次，参数是否为预期参数（如果有参数）

.vue文件

<template>

<button :class="cssClasses" v-on:click="onClick" :disabled="disabled">

<slot>submit</slot>

</button>

</template>

<script>

export default {

methods: {

onClick () {

this.$emit('click', 'i am params') // 此处的参数'i am params'纯为测试准备，并未真正使用

}

}

}

</script>

测试代码（仅AppButton组件的测试文件中wrapper在it内创建，其他的都在describe下创建，详见代码仓库）

// 测试内容：func测试

// 点击按钮组件时，正确触发点击事件

it('click button onClick is clled', () => {

const wrapper = shallowMount(AppButton)

// 创建mock函数

const mockFn = jest.fn()

// 设置 Wrapper vm 的方法并强制更新。

wrapper.setMethods({

onClick: mockFn

})

// 获取buttonDOM元素

const button = wrapper.find('button')

// 测试点击按钮后有没有正确触发函数

// 点击按钮

button.trigger('click')

// 断言函数被触发，且被触发一次

expect(mockFn).toBeCalled()

expect(mockFn).toHaveBeenCalledTimes(1)

wrapper.destroy()

})

1. v-bind（依赖变量存在于script标签内）

.vue文件

<template>

<div class="header">

<div class="user-info">

<span class="el-dropdown-link">

<img class="user-logo" src="../assets/logo.png">

{{name}}

</span>

</div>

</div>

</template>

<script>

export default {

data () {

return {

collapse: checkWidth(),

name: 'admin'

}

}

}

</script>

测试代码

// 测试内容：data v-bind

// 断言data中变量的默认值

it('data test', () => {

expect(wrapper.vm.name).toBe('admin')

expect(wrapper.vm.collapse).toBeTruthy() // 函数checkWidth()在测试环境下的值一致都是true

wrapper.vm.name = 'holy'

expect(wrapper.find('.el-dropdown-link').text()).toBe('holy')

})

1. slot（vue2.6.0以上版本更新了此语法，本文完成时并未影响到测试）

slots分文普通插槽和具名插槽和作用域插槽，均需测试

.vue文件内

<template>

<button :class="cssClasses" v-on:click="onClick" :disabled="disabled">

<slot>submit</slot>

<!-- 下面的命名插槽和作用域插槽，实际并未使用，是为了测试而存在 -->

<slot name="namedSlot"></slot>

<slot name="scopedSlot" :user="user"></slot>

</button>

</template>

测试文件

/ 测试内容：slots 普通插槽

// 测试默认值

it('slots default value test', () => {

const wrapper = shallowMount(AppButton)

const button = wrapper.find('button')

expect(button.text()).toBe('submit')

wrapper.destroy()

})

// 测试内容：slots 普通插槽

// mount时传入自定义的内容作为slots，然后再断言自定义的内容存在与否

// 自定义的内容可能会是text，html，componets等允许的内容

it('slots test', () => {

const wrapper = shallowMount(AppButton, {

slots: {

default: 'i am slots text' // 自定义slots内容

}

})

const button = wrapper.find('button')

expect(button.text()).toBe('i am slots text')

wrapper.destroy()

})

// 测试内容：slots具名插槽->vue2.6更新后的新语法v-slots，3.0中会延续使用，并废除旧语法（在此提醒注意，不影响测试代码的书写）

// 测试方法与slots普通插槽相同，此处传入为html,

// 当传入组件时，只需断言wrapper中是否包含组件的DOM元素即可

// expect(wrapper.contains('.container')).toBe(true)

it('named slots test', () => {

const wrapper = shallowMount(AppButton, {

slots: {

namedSlot: `<span>i am slots html</span>` // 自定义slots内容

}

})

const button = wrapper.find('button')

expect(button.contains('span')).toBe(true)

const span = wrapper.find('button span')

expect(span.text()).toBe('i am slots html')

wrapper.destroy()

})

// 测试内容：slots作用域插槽->vue2.6更新后的新语法v-slots，3.0中会延续使用，并废除旧语法（在此提醒注意，不影响测试代码的书写）

// 测试方法与slots具名插槽相同，此处传入为html,

// 当传入组件时，只需断言wrapper中是否包含组件的DOM元素即可

// expect(wrapper.contains('.container')).toBe(true)

it('scoped slots test', () => {

const wrapper = shallowMount(AppButton, {

scopedSlots: {

scopedSlot: `<span slot-scope="foo">{{ foo.user.lastName }}</span>` // 自定义slots内容

}

})

const button = wrapper.find('button')

expect(button.contains('span')).toBe(true)

const span = wrapper.find('button span')

expect(span.text()).toBe('holy')

wrapper.destroy()

})

1. v-if/v-show（依赖变量存在于script标签内）

.vue组件内

<template>

<!-- filter的单元测试例 -->

<div class="example filter">

<h2 class="example\_title">filter的单元测试例</h2>

<el-input class="input" v-model="userInput" placeholder="输入长度大于7的内容，下方将显示过滤后内容"></el-input>

<div v-show="toggleShow" class="text format">{{userInput | formatText}}</div>

<div v-show="!toggleShow" class="text noformat">{{userInput}}</div>

<app-button @click="changeShow">切换显示方式</app-button>

</div>

</template>

<script>

export default {

data () {

return {

userInput: '',

toggleShow: true

}

}

}

</script>

测试文件

// 测试内容：v-show 测试，v-if v-bind的测试思路基本相同，

// 设置变量的值，断言对应的DOM结构显示与否

it('v-show test', () => {

// true时显示的div

const truediv = wrapper.find('.text.format')

// false时显示的div

const falsediv = wrapper.find('.text.noformat')

// toggleShow默认值为true

expect(truediv.isVisible()).toBe(true)

expect(falsediv.isVisible()).toBe(false)

// 设置为false

wrapper.vm.toggleShow = false

expect(truediv.isVisible()).toBe(false)

expect(falsediv.isVisible()).toBe(true)

})

1. filter

.vue组件内

<template>

<!-- filter的单元测试例 -->

<div class="example filter">

<h2 class="example\_title">filter的单元测试例</h2>

<el-input class="input" v-model="userInput" placeholder="输入长度大于7的内容，下方将显示过滤后内容"></el-input>

<div v-show="toggleShow" class="text format">{{userInput | formatText}}</div>

<div v-show="!toggleShow" class="text noformat">{{userInput}}</div>

<app-button @click="changeShow">切换显示方式</app-button>

</div>

</template>

<script>

export default {

data () {

return {

userInput: '',

toggleShow: true

}

},

filters: {

formatText: function (value) {

// console.log(value)

if (!value) return ''

if (value.length > 7) {

let frontVal = value.slice(0, 2)

let endVal = value.slice(value.length - 2, value.length)

return frontVal + '...' + endVal

} else {

return value

}

}

},

methods: {

changeShow () {

this.toggleShow = !this.toggleShow

}

}

}

</script>

测试文件

import { shallowMount, createLocalVue } from '@vue/test-utils'

import FilterTest from '@/components/FilterTest.vue'

// 测试内容：filter----filter不能通过wrapper或者vm获取，只能通过组件获取

// filter需要测试函数的所有可能性

it('filter test', () => {

// console.log(FilterTest.filters)

expect(FilterTest.filters.formatText('12345678')).toBe('12...78')

expect(FilterTest.filters.formatText('12345')).toBe('12345')

expect(FilterTest.filters.formatText()).toBe('')

})

1. Props（⭐必须进行测试）

<template>

<button :class="cssClasses" v-on:click="onClick" :disabled="disabled">

<slot>submit</slot>

</button>

</template>

<script>

export default {

name: 'AppButton',

props: {

type: {

type: String,

default: 'primary'

},

size: {

type: String,

default: 'md'

},

disabled: {

type: Boolean,

default: false

}

}

</script>

测试文件

// 测试内容：props

// 自定义props传递给AppButton组件，判断组件有获取到props

it('props test', () => {

const buttonProps = {

type: 'danger',

size: 'lg',

disabled: true

}

const wrapper = shallowMount(AppButton, {

propsData: buttonProps

})

// 断言已经获取到props

expect(wrapper.props().size).toBe('lg')

expect(wrapper.props().type).toBe('danger')

expect(wrapper.props().disabled).toBe(true)

// 每个it最后都应该销毁wrapper

wrapper.destroy()

})

1. Data（见①中跟v-bind一起测试）
2. Methods（⭐func函数测试）

.vue文件内

<template>

<!-- filter的单元测试例 -->

<div class="example filter">

<app-button @click="changeShow">切换显示方式</app-button>

</div>

</template>

<script>

export default {

data () {

return {

userInput: '',

toggleShow: true

}

},

methods: {

changeShow () {

this.toggleShow = !this.toggleShow

}

}

}

</script>

<style scoped>

</style>

测试文件

// 测试内容：changeShow()函数

// changeShow()函数被调用时，能正确执行

it('called changeShow()', () => {

// 手动将变量的值设置为false,默认值是true

wrapper.vm.toggleShow = false

// 执行函数

wrapper.vm.changeShow()

// 期望结果

expect(wrapper.vm.toggleShow).toBe(true)

// 再次执行函数

wrapper.vm.changeShow()

// 期望结果

expect(wrapper.vm.toggleShow).toBe(false)

})

1. $emit（⭐必须进行测试）

.vue文件内

<template>

<button :class="cssClasses" v-on:click="onClick" :disabled="disabled">

<slot>submit</slot>

</button>

</template>

<script>

export default {

methods: {

onClick () {

this.$emit('click', 'i am params') // 此处的参数'i am params'纯为测试准备，并未真正使用

}

}

}

</script>

测试文件

// 测试内容：$emit

// 函数被触发后，emit的函数也会被触发

it('when onClick is called $emit is called', () => {

const wrapper = shallowMount(AppButton)

// 测试$emmit函数被正确触发

// mock函数替代点击按钮后$emit的函数，此处函数名相同，依然为click

const mockFn1 = jest.fn()

wrapper.vm.$on('click', mockFn1)

// 测试mock函数是否被触发，触发的次数,以及参数

wrapper.vm.onClick()

expect(mockFn1).toBeCalled()

expect(mockFn1).toHaveBeenCalledTimes(1)

expect(mockFn1).toHaveBeenCalledWith('i am params')

// 第二次点击button，依然测试mock函数是否被触发，触发的次数

wrapper.vm.onClick()

expect(mockFn1).toBeCalled()

expect(mockFn1).toHaveBeenCalledTimes(2)

expect(mockFn1).toHaveBeenCalledWith('i am params')

wrapper.destroy()

})

1. eventHub（空vue实例，用于分发事件，在使用的组件内进行测试）

.vue组件内

import eventHub from '@/assets/js/EventHub.js'

created () {

// 通过 Event Bus 进行组件间通信，来折叠侧边栏

eventHub.$on('collapse', msg => {

this.collapse = msg

})

}

测试文件

// 测试内容：eventHub $on

it('eventHub $on test', () => {

wrapper.vm.collapse = false

eventHub.$emit('collapse', true)

expect(wrapper.vm.collapse).toBeTruthy()

})

1. watch（⭐必须进行测试）
2. computed（⭐必须进行测试）

.vue文件

<template>

<button :class="cssClasses" v-on:click="onClick" :disabled="disabled">

<slot>submit</slot>

</button>

</template>

<script>

export default {

name: 'AppButton',

props: {

type: {

type: String,

default: 'primary'

},

size: {

type: String,

default: 'md'

},

disabled: {

type: Boolean,

default: false

}

},

computed: {

cssClasses () {

let classes = ['app-button']

classes.push('app-button--' + this.size)

if (this.type && !this.disabled) {

classes.push('app-button--' + this.type)

}

if (this.disabled) {

classes.push('app-button--disabled')

}

return classes.join(' ')

}

}

}

</script>

测试文件

// 测试内容：computed(要注意计算属性不是函数，是变量，测试时很容易看着组件内写法按照函数测试)

// 改变props的type，size，disable值时，cssClasses的值也会跟着改变

it('computed test', () => {

const wrapper = shallowMount(AppButton)

// 设置props 断言computed计算属性（注意props有default值）

wrapper.setProps({ type: 'danger' })

expect(wrapper.vm.cssClasses).toBe('app-button app-button--md app-button--danger')

wrapper.setProps({ size: 'lg' })

expect(wrapper.vm.cssClasses).toBe('app-button app-button--lg app-button--danger')

wrapper.setProps({ disabled: true })

expect(wrapper.vm.cssClasses).toBe('app-button app-button--lg app-button--disabled')

wrapper.destroy()

})

1. router（⭐必须进行测试）

.vue文件

<template>

<!-- 路由跳转的单元测试例 -->

<div class="example router">

<h2 class="example\_title">路由跳转的单元测试例</h2>

<app-button size="lg" @click="goVIPs">跳转到VIP用户页面</app-button>

</div>

</template>

<script>

export default {

methods: {

goVIPs () {

this.$router.push({

path: '/vips'

})

}

}

}

</script>

<style scoped>

</style>

测试文件

import { shallowMount } from '@vue/test-utils'

import RouterTest from '@/components/RouterTest.vue'

// vue-router测试

// 不建议直接在localVue上挂载vue-router

// 使用mock的$route和$router更加灵活，方便测试

const $route = {

path: '/some'

// ...其他属性

}

const mockPush = jest.fn()

const $router = {

push: mockPush

// ... 其他属性

}

describe('RouterTest.vue', () => {

// A Wrapper is an object that contains a mounted component or vnode and methods to test the component or vnode

// wrapper是一个对象，包含了已经mounted的组件或vnode，及测试该组件或 vnode 的方法。

// wrapper.vm是组件实例，包含该实例的所有方法和属性

let wrapper

beforeEach(() => {

wrapper = shallowMount(RouterTest, {

stubs: ['app-button'],

mocks: {

$route,

$router

}

})

})

afterEach(() => {

wrapper.destroy()

})

// 测试内容：goVIPs()被调用->触发mock的push函数

it('goVIPs() is called', () => {

// 执行函数

wrapper.vm.goVIPs()

// 期望结果

expect(mockPush).toBeCalled()

})

})

1. axios（⭐必须进行测试）

.vue组件

<template>

<!-- Axios的单元测试例 -->

<div class="example axios">

<h2 class="example\_title">Axios的单元测试例</h2>

<app-button size="lg" @click="getData">从后台接口获取数据</app-button>

</div>

</template>

<script>

export default {

data () {

return {

usersInfo: []

}

},

methods: {

getData () {

// 伪造的请求，只是为单元测试提供一个例子

// 下面的两处return都是为了配合单元测试,方便单元测试

// 如果没有下面的return.在测试时就无法获取promis和最终请求结果,导致无法进行断言

return this.$axios

.get('users/')

.then(res => {

this.usersInfo = res.data

return res

})

.catch(e => e)

}

}

}

</script>

测试文件

/\*\*

\* AxiosTest.vue组件

\* 测试内容包括以下

\* 自定义func

\* axios

\*/

import { shallowMount, createLocalVue } from '@vue/test-utils'

import AxiosTest from '@/components/AxiosTest.vue'

import axios from 'axios'

// mock掉整个axios模块

// 返回值在使用的时候自定义

jest.mock('axios')

// 创建临时Vue实例，挂载组件中使用的插件

const localVue = createLocalVue()

localVue.prototype.$axios = axios // 挂载axios

describe('AxiosTest.vue', () => {

let wrapper

beforeEach(() => {

axios.mockClear()

wrapper = shallowMount(AxiosTest, { localVue,

stubs: ['app-button']

})

})

afterEach(() => {

wrapper.destroy()

})

// 测试内容：func ->getData()

// 点击按钮函数被触发(注意此处的click事件是子组件(按钮组件$emit)的事件,在父组件内不属于DOM原生事件,所以触发方式不能使用trigger,而应该使用$emit)

it('when button is clicked getData will be called', () => {

// 创建mock函数

const mockFn = jest.fn()

// 设置 Wrapper vm 的方法并强制更新。

wrapper.setMethods({ getData: mockFn })

// 获取对应按钮

const axiosButton = wrapper.find('.axios app-button-stub')

// 点击按钮->注意触发方式不能使用trigger

axiosButton.vm.$emit('click')

// 断言函数被触发且只触发一次

expect(mockFn).toBeCalled()

expect(mockFn).toHaveBeenCalledTimes(1)

})

// 测试内容：axios->getData()函数

// 为了配合axios测试,需要在组件代码的两处增加return,参见AxiosTest组件

it('axios test', () => {

// 此处只是使用了get,post/patch/delete/...与get相同

// 自定义get的返回值

const mockData = { data: { name: 'Bob' } }

axios.get.mockResolvedValue(mockData)

return wrapper.vm.getData().then(result => {

expect(result).toEqual(mockData)

expect(wrapper.vm.usersInfo).toEqual(mockData.data)

})

})

// 测试内容：axios->getData()函数请求rejeced的情况

it('axios test', () => {

// 自定义get被拒绝时返回值

axios.get.mockRejectedValue('error')

return wrapper.vm.getData().catch(e => expect(e).toMatch('error'))

})

// 测试内容：snapshot->概括的测试DOM结构

// 如果组件内存在比较特殊的需要测试的DOM结构的话，可以单独测试（详见AppButton测试文件）

it('snapshot test', () => {

expect(wrapper.html()).toMatchSnapshot()

})

})

1. vuex（⭐必须进行测试）

.vue组件

<template>

<!-- vuex的单元测试例 -->

<div class="example vuex">

<h2 class="example\_title">vuex的单元测试例</h2>

<div class="text">Clicked: [ {{ $store.state.count }} ] times, count is [ {{ evenOrOdd }} ]</div>

<app-button class="add" size="lg" @click="increment">+1</app-button>&nbsp;&nbsp;&nbsp;

<app-button class="minus" size="lg" @click="decrement">-1</app-button>

&nbsp;&nbsp;&nbsp;

<app-button class="dispatchAdd" size="lg" @click="dispatchIncrement">通过dispatch+1</app-button>

&nbsp;&nbsp;&nbsp;

<app-button class="mutationsMinus" size="lg" @click="mutationsDecrement">通过mutations-1</app-button>

</div>

</template>

<script>

import { mapGetters, mapActions } from 'vuex'

export default {

methods: {

...mapActions([

'increment',

'decrement'

]),

// 为了展示测试方法而存在

dispatchIncrement () {

this.$store.dispatch('increment')

},

// 为了展示测试方法而存在

mutationsDecrement () {

this.$store.commit('decrement')

}

},

computed: mapGetters([

'evenOrOdd'

])

}

</script>

测试文件

/\*\*

\* VuexTest.vue组件

\* 测试内容包括以下

\* 临时vue实例上挂载element-ui

\* func

\* vuex store内的state mutations actions getters

\*/

import { shallowMount, createLocalVue } from '@vue/test-utils'

import VuexTest from '@/components/VuexTest.vue'

import Vuex from 'vuex'

// 创建临时Vue实例，挂载组件中使用的插件

const localVue = createLocalVue()

localVue.use(Vuex)

describe('VuexTest.vue', () => {

// A Wrapper is an object that contains a mounted component or vnode and methods to test the component or vnode

// wrapper是一个对象，包含了已经mounted的组件或vnode，及测试该组件或 vnode 的方法。

// wrapper.vm是组件实例，包含该实例的所有方法和属性

let wrapper

let actions

let state

let getters

let mutations

let store

beforeEach(() => {

// 伪造actions state getters mutations

actions = {

increment: jest.fn(),

decrement: jest.fn()

}

mutations = {

increment: jest.fn(),

decrement: jest.fn()

}

state = {

count: 0

}

getters = {

evenOrOdd: () => 'gettersVal' // 伪造的越简单越好

}

// 伪造store

store = new Vuex.Store({

state,

actions,

mutations,

getters

})

// 挂载store

wrapper = shallowMount(VuexTest, {

localVue,

store,

stubs: ['app-button']

})

})

afterEach(() => {

wrapper.destroy()

})

// 测试内容：state

// 只需测试伪造的state值是否存在于dom中

it('getters test', () => {

const text = wrapper.find('.text')

expect(text.text()).toContain(state.count)

})

// 测试内容：actions-通过点击按钮直接调用

// 点击按钮测试伪造的函数是否被调用

it('actions test', () => {

const buttonAdd = wrapper.find('.add')

const buttonMinus = wrapper.find('.minus')

buttonAdd.vm.$emit('click')

expect(actions.increment).toHaveBeenCalled()

expect(actions.increment).toHaveBeenCalledTimes(1)

buttonMinus.vm.$emit('click')

expect(actions.decrement).toHaveBeenCalled()

expect(actions.decrement).toHaveBeenCalledTimes(1)

})

// 测试内容：dispatchIncrement()

// 点击按钮测试函数dispatchIncrement()是否被调用

it('dispatchIncrement test', () => {

const dispatchAdd = wrapper.find('.dispatchAdd')

const mockAdd = jest.fn()

wrapper.setMethods({

dispatchIncrement: mockAdd

})

dispatchAdd.vm.$emit('click')

expect(mockAdd).toHaveBeenCalled()

expect(mockAdd).toHaveBeenCalledTimes(1)

})

// 测试内容：actions--通过dispatch调用

// 调用函数测试伪造的函数是否被调用

it('dispatch actions test', () => {

wrapper.vm.dispatchIncrement()

expect(actions.increment).toHaveBeenCalled()

expect(actions.increment).toHaveBeenCalledTimes(1)

})

// 测试内容：mutationsDecrement()

// 点击按钮测试函数mutationsDecrement()是否被调用

it('mutationsDecrement test', () => {

const mutationsMinus = wrapper.find('.mutationsMinus')

const mockMinus = jest.fn()

wrapper.setMethods({

mutationsDecrement: mockMinus

})

mutationsMinus.vm.$emit('click')

expect(mockMinus).toHaveBeenCalled()

expect(mockMinus).toHaveBeenCalledTimes(1)

})

// 测试内容：mutations

// 调用函数测试伪造的函数是否被调用

it('mutations test', () => {

wrapper.vm.mutationsDecrement()

expect(mutations.decrement).toHaveBeenCalled()

expect(mutations.decrement).toHaveBeenCalledTimes(1)

})

// 测试内容：getters

// 只需测试伪造的getters值是否存在于dom中

it('getters test', () => {

const text = wrapper.find('.text')

expect(text.text()).toContain(getters.evenOrOdd())

})

// 测试内容：snapshot->概括的测试DOM结构

// 如果组件内存在比较特殊的需要测试的DOM结构的话，可以单独测试（详见AppButton测试文件末尾）

it('snapshot test', () => {

expect(wrapper.html()).toMatchSnapshot()

})

})

1. store.js

import Vue from 'vue'

import Vuex from 'vuex'

Vue.use(Vuex)

const state = {

count: 0

}

// mutations/actions/getters 前面的export是为了配合测试

export const mutations = {

increment: state => state.count++,

decrement: state => state.count--

}

export const actions = {

increment: ({ commit }) => commit('increment'),

decrement: ({ commit }) => commit('decrement')

}

export const getters = {

evenOrOdd: state => state.count % 2 === 0 ? 'even' : 'odd'

}

export default new Vuex.Store({

state,

mutations,

actions,

getters

})

测试文件

/\*\*

\* store.js

\* 测试内容包括以下

\* 测试思路：mutations/getters/actions 分别测试，测试其中一个的时候，其他依赖伪造mock

\* 本质还是测试方法（官网列出了两种测试方法，我们选择简单易懂便于测试的）

\*/

import { mutations, getters, actions } from '@/store.js'

describe('store.js', () => {

// 测试内容：mutations

// 伪造（mock）state 测试mutations下的方法

it('mutations test', () => {

const state = {

count: 0

}

mutations.increment(state)

expect(state.count).toBe(1)

mutations.decrement(state)

expect(state.count).toBe(0)

})

// 测试内容：getters

// 伪造（mock）state 测试evenOrOdd的值

it('getters test even', () => {

const state = {

count: 0

}

expect(getters.evenOrOdd(state)).toBe('even')

})

// 测试内容：getters

// 伪造（mock）state 测试evenOrOdd的值

it('getters test odd', () => {

const state = {

count: 1

}

expect(getters.evenOrOdd(state)).toBe('odd')

})

// 测试内容：actions 比前两个测试要稍微复杂

// 伪造（mock）commit 测试mutations下的方法

it('actions test', () => {

// 伪造commit

const commit = jest.fn()

// increment

actions.increment({ commit })

expect(commit).toBeCalled()

expect(commit).toBeCalledWith('increment')

// decrement

actions.decrement({ commit })

expect(commit).toBeCalled()

expect(commit).toBeCalledWith('decrement')

})

})