前端 Vue 3 单元测试入门

Jest | Vitest

作者: https://github.com/pdsuwwz

```
expect(1 + 1).toBe(2)
```

断言 1 + 1 的结果应该等于2

前端为什么需要单元测试?

前端为什么需要单元测试?

代码质量	帮助开发者发现和修复潜在的错误场景和边界条件
可维护性	一种形式文档，旨在帮助开发者更清晰理解代码的行为
正确性	验证代码的正确性，从而能够在使用组件时做到心里有底
自动化	在开发过程中自动运行，相比手动测试能够节省时间和精力

前端为什么需要单元测试?

代码质量	帮助开发者发现和修复潜在的错误场景和边界条件
可维护性	一种形式文档，旨在帮助开发者更清晰理解代码的行为
正确性	验证代码的正确性，从而能够在使用组件时做到心里有底
自动化	在开发过程中自动运行，相比手动测试能够节省时间和精力

前端为什么需要单元测试?

代码质量	帮助开发者发现和修复潜在的错误场景和边界条件
可维护性	一种形式文档，旨在帮助开发者更清晰理解代码的行为
正确性	验证代码的正确性，从而能够在使用组件时做到心里有底
自动化	在开发过程中自动运行，相比手动测试能够节省时间和精力

前端为什么需要单元测试?

代码质量	帮助开发者发现和修复潜在的错误场景和边界条件
可维护性	一种形式文档，旨在帮助开发者更清晰理解代码的行为
正确性	验证代码的正确性，从而能够在使用组件时做到心里有底
自动化	在开发过程中自动运行，相比手动测试能够节省时间和精力

前端为什么需要单元测试?

代码质量	帮助开发者发现和修复潜在的错误场景和边界条件
可维护性	一种形式文档，旨在帮助开发者更清晰理解代码的行为
正确性	验证代码的正确性，从而能够在使用组件时做到心里有底
自动化	在开发过程中自动运行，相比手动测试能够节省时间和精力

前端为什么需要单元测试?

代码质量	帮助开发者发现和修复潜在的错误场景和边界条件
可维护性	一种形式文档，旨在帮助开发者更清晰理解代码的行为
正确性	验证代码的正确性，从而能够在使用组件时做到心里有底
自动化	在开发过程中自动运行，相比手动测试能够节省时间和精力

先决条件

- Node.js LTS 版本 (>= 16.x)
- Pnpm 包管理工具 (>= 8.x)

大纲

1. 单元测试框架选择
2. 测试环境搭建
3. Vue3 测试用例编写
4. Vitest 的使用

大纲

1. 单元测试框架选择
2. 测试环境搭建
3. Vue3 测试用例编写
4. Vitest 的使用

大纲

1. 单元测试框架选择
2. 测试环境搭建
3. Vue3 测试用例编写
4. Vitest 的使用

大纲

1. 单元测试框架选择
2. 测试环境搭建
3. Vue3 测试用例编写
4. Vitest 的使用

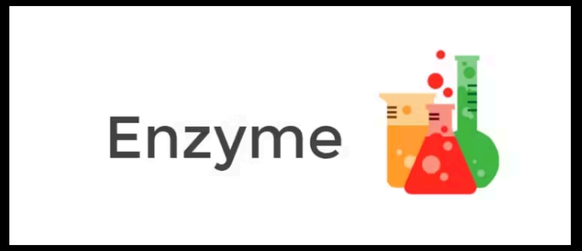
大纲

1. 单元测试框架选择
2. 测试环境搭建
3. Vue3 测试用例编写
4. Vitest 的使用

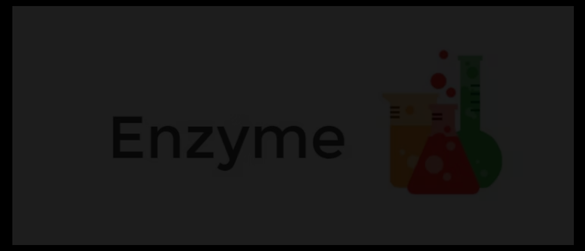
大纲

1. 单元测试框架选择
2. 测试环境搭建
3. Vue3 测试用例编写
4. Vitest 的使用

单元测试框架选择



单元测试框架选择



单元测试框架选择

Jest

https://jestjs.io

Vitest

https://cn.vitest.dev

单元测试框架选择

Jest 是一个流行的 JS 测试框架，由 Facebook 开发并维护。它专注于提供简单的 API 和强大的功能，使测试变得简单

一个 Vite 原生的单元测试框架。非常的快！

单元测试框架选择

Jest

单元测试框架选择

特点

Jest

- ✅ 配置简洁
- ✅ 自带断言
- ✅ 并行测试
- ✅ 丰富的 API 生态

Vitest

- ✅ Vite 支持
- ✅ 无缝兼容 Jest
- ✅ 智能 HMR
- ✅ 天然支持 ES Module

单元测试框架选择

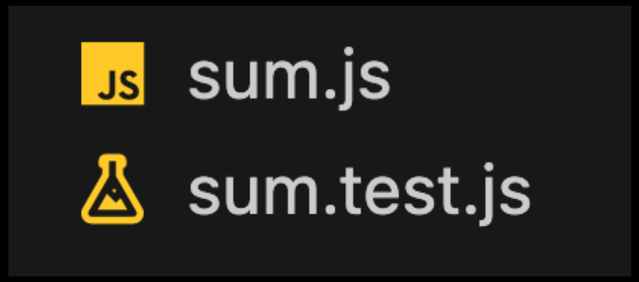
Jest

测试环境搭建

(1) 快速起步

- 安装 Jest
- 创建并编写测试用例：

```
$ pnpm add -D jest
```



代码演示时间

其实很短

测试环境搭建

(2) 支持 ES Module

- 改写为 es6 语法
- 更改两个测试文件

```
function sum(num1, num2) {
  return num1 + num2
}
module.exports = {
  sum
}
```

```
export function sum(num1, num2) {
  return num1 + num2
}
```

```
const { sum } = require('./sum')
```

```
import { sum } from './sum'
```

测试环境搭建

(3) Jest 协同 Babel

- 安装 Babel
- 配置 babel.config.js

```
$ pnpm add -D babel-jest @babel/core @babel/preset-env
```

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

Vue3 测试用例编写

Vue3 测试用例编写

(1) 在 Vue3 组件中使用 Jest

- 创建 Vue 组件和对应单测文件

```
$ pnpm add vue
$ pnpm add -D @vue/vue3-jest
$ pnpm add -D @vue/test-utils
```

HelloWorld.vue M

src > HelloWorld.vue > ...

You, 6 seconds ago | 1 author (You)

```vue
<template>
  <div>{{ msg }}</div>
</template>

<script setup>
defineOptions({
  name: 'HelloWorld'
})

defineProps({
  msg: ''
})

</script>
```

```
import { mount } from '@vue/test-utils'
import HelloWorld from './src/HelloWorld.vue'

Run | Debug
test('Test Vue Running', () => {

  const textMsg = '测试'
  const instance = mount(HelloWorld, {
    props: {
      msg: textMsg
    }
  })

  expect(instance.text()).toContain(textMsg)
})
```

Vue3 测试用例编写

(2) 使 Jest 识别 Vue3 语法

- 安装 jest-environment-jsdom 模拟 DOM 环境
- 高版本 Jest 需要添加 testEnvironmentOptions 来保证测试正常运行

```
module.exports = {
  testEnvironment: 'jsdom',
  testEnvironmentOptions: {
    customExportConditions: [
      'node',
      'node-addons'
    ]
  },
  transform: {
    '^.+\\.js$': 'babel-jest',
    '^.+\\.vue$': '@vue/vue3-jest'
  }
}
```

Vue3 测试用例编写

(2) 使 Jest 识别 Vue3 语法

- 安装 jest-environment-jsdom 模拟 DOM 环境
- 高版本 Jest 需要添加 testEnvironmentOptions 来保证测试正常运行

```
module.exports = {
  testEnvironment: 'jsdom',
  testEnvironmentOptions: {
    customExportConditions: [
      'node',
      'node-addons'
    ]
  },
  transform: {
    '^.+\\.js$': 'babel-jest',
    '^.+\\.vue$': '@vue/vue3-jest'
  }
}
```

Vitest 的使用

我更倾向于推荐你使用 Vitest 而不是 Jest

Vitest 的使用

(1) 快速起步

```
$ pnpm add -D vitest @vue/test-utils
```

- 准备一个 Vite + Vue3 项目
- 安装 Vitest

```ts
import { defineConfig } from 'vite'
import vue from '@vitejs/plugin-vue'

export default defineConfig({
  plugins: [vue()],
  test: {
    globals: true,
    dir: '__tests__',
    environment: 'jsdom',
  }
})
```

Vitest 的使用

(2) 代码迁移

- 代码粘过来，再显示地导入测试函数即可

hello.test.js > ...

```
import { mount } from '@vue/test-utils'
import HelloWorld from './src/HelloWorld.vue'

test('Test Vue Running', () => {

  const textMsg = '测试'
  const instance = mount(HelloWorld, {
    props: {
      msg: textMsg
    }
  })

  expect(instance.text()).toContain(textMsg)
})

```

→

```
import { test, expect } from 'vitest'
import { mount } from '@vue/test-utils'
import HelloWorld from './src/HelloWorld.vue'

test('Test Vue Running', () => {

  const textMsg = '测试'
  const instance = mount(HelloWorld, {
    props: {
      msg: textMsg
    }
  })

  expect(instance.text()).toContain(textMsg)
})

```

来个稍微复杂的

其实很简单，但也算是复杂

Vitest 的使用

(3) 实现一个按钮点击后，count自增并将其值显示在template中的组件

- 测试：点击事件的触发
- 测试：count 值是否自增
- 测试：快照的使用方法（一般用于对比测试用）

常用的一些断言方法

常用的一些断言方法

toContain	验证字符串是否所属另一个字符串/数组的子集
toBe	验证基础对象/引用是否相等（浅比较）
toEqual	验证对象及结构是否相等（递归对比）
.not.	用于否定断言
toBeTruthy	将待验证值转换为布尔值，断言该值是否为 true
toThorw	验证函数在调用时是否抛出指定错误
toMatch	验证字符串是否匹配指定的正则表达式或字符串
toHaveLength	验证测试字符串和数组类型的length是否符合预期

其他的考虑

需求是提不完的...

- 其他框架可以使用吗? 如 React, Nuxt, Ruby, Solid…
- 我想用 TypeScript, 是否支持呢?
- Vitest 已自带了断言库, 我还想根据喜好自己扩展断言可以吗?
- 仅限使用了 Vite 的项目吗, Webpack 的可以不?
- 可以和 Puppeteer 一起使用吧?
- 支持跨端使用吗?
- … 省略999+

推荐学习仓库

- https://github.com/vuejs/router/tree/main/packages/router/__tests__
- https://github.com/element-plus/element-plus/blob/dev/packages/components/button
- https://github.com/tusen-ai/naive-ui/blob/main/src/button/tests/Button.spec.tsx
- https://github.com/vuejs/pinia/blob/v2/packages/pinia/__tests__/state.spec.ts

总结

如果你想给你的新项目使用 vitest/jest 或者将旧项目的测试方案从 jest 迁移到 vitest，你可以从以下几个方面着手：

- 拥抱 ES Module
- 熟悉对应的迁移文档
- 参考大型开源项目仓库的测试用法