

自动化测试

- 1. 课前准备
- 2. 课堂主题
- 3. 课堂目标
- 4. 知识点

测试分类

单测

api介绍

测试Vue组件

检查mounted之后

用户点击

测试覆盖率

E2E测试

测试用户点击

TDD

React 自动化测试

Node自动化测试

1. 课前准备

- 1. 了解自动化测试
- 2. [cypress](#)

2. 课堂主题

- 1. 单测
- 2. E2E测试

3. 课堂目标

- 1. 掌握Vue测试
- 2. 写易于测试的Vue组件和代码

4. 知识点

测试分类

常见的开发流程里，都有测试人员，这种我们成为黑盒测试，测试人员不管内部实现机制，只看最外层的输入输出，比如你写一个加法的页面，会设计N个case，测试加法的正确性，这种代码里，我们称之为E2E测试

更负责一些的 我们称之为集成测试，就是集合多个测试过的单元一起测试

还有一种测试叫做白盒测试，我们针对一些内部机制的核心逻辑 使用代码 进行编写 我们称之为单元测试

这仨都是我们前端开发人员进阶必备的技能

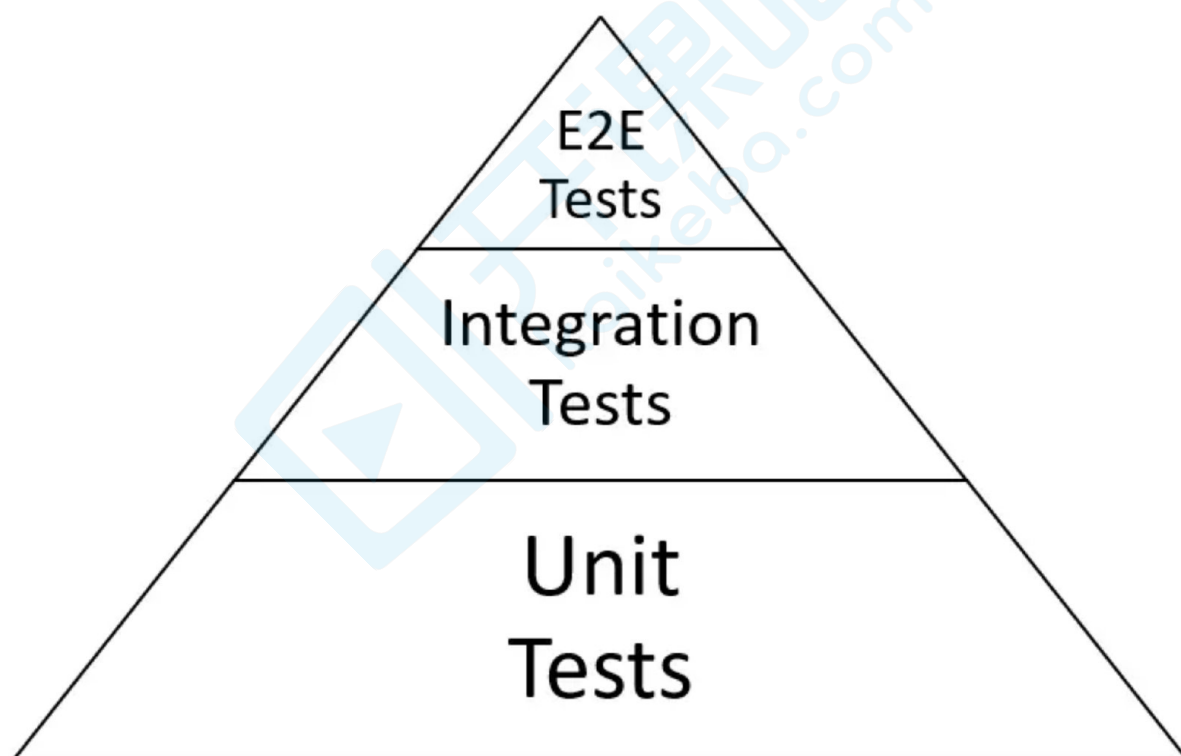
我们其实日常使用console，算是测试的雏形吧，`console.log(add(1,2) == 3)`

测试的好处

组件的单元测试有很多好处：

- 提供描述组件行为的文档
- 节省手动测试的时间
- 减少研发新特性时产生的 bug
- 改进设计
- 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。让你改代码不再小心翼翼



单测

单元测试（unit testing），是指对软件中的最小可测试单元进行检查和验证。

在vue中，推荐用Mocha+chai 或者jest，咱们使用jest演示，语法基本一致

新建kaikeba.spec.js，.spec.js是命名规范，写下一下代码

```
function add(num1, num2) {
  return num1 + num2
}

describe('kaikeba', () => {
  it('测试加法', () => {
    expect(add(1, 3)).toBe(3)
    expect(add(1, 3)).toBe(4)
    expect(add(-2, 3)).toBe(1)
  })
})
```

执行 npm run test:unit

The terminal output shows a failed test. The first test suite, 'Kaikeba > 测试加法', failed. The error message is 'expect(received).toBe(expected) // Object.is equality'. The expected value is 3, and the received value is 4. The test suite 'Kaikeba' has 1 failed test and 1 passed test. The second test suite, 'example.spec.js', passed. The overall test results are 1 failed, 1 passed, and 2 total tests. The time taken for the tests is 1.703s.

```
FAIL tests/unit/kaikeba.spec.js
  • Kaikeba > 测试加法

    expect(received).toBe(expected) // Object.is equality

    Expected: 3
    Received: 4

      6 | describe('Kaikeba', () => {
      7 |   it('测试加法', () => {
    >  8 |     expect(add(1, 3)).toBe(3)
        |                               ^
      9 |     expect(add(1, 3)).toBe(4)
     10 |     expect(add(-2, 3)).toBe(1)
     11 |   })
        at Object.toBe (tests/unit/kaikeba.spec.js:8:27)

PASS tests/unit/example.spec.js

Test Suites: 1 failed, 1 passed, 2 total
Tests:       1 failed, 1 passed, 2 total
Snapshots:   0 total
Time:        1.703s
```

api介绍

- `describe`: 定义一个测试套件
- `it`: 定义一个测试用例
- `expect`: 断言的判断条件
- `toBe`: 断言的比较结果

测试Vue组件

一个简单的组件

```
<template>
  <div>
    <span>{{ message }}</span>
    <button @click="changeMsg">点击</button>
  </div>
</template>

<script>
  export default {
    data () {
      return {
        message: 'vue-text'
      }
    },
    created () {
      this.message = '开课吧'
    },
    methods: {
      changeMsg() {
        this.message = '按钮点击'
      }
    }
  }
</script>
```

```
// 导入 vue.js 和组件, 进行测试
import Vue from 'vue'
import KaikebaComp from '@/components/Kaikeba.vue'

// 这里是一些 Jasmine 2.0 的测试, 你也可以使用你喜欢的任何断言库或测试工具。

describe('KaikebaComp', () => {
  // 检查原始组件选项
  it('由created生命周期', () => {
    expect(typeof KaikebaComp.created).toBe('function')
  })

  // 评估原始组件选项中的函数的结果
  it('初始data是vue-text', () => {
    expect(typeof KaikebaComp.data).toBe('function')
  })
})
```

```
const defaultData = KaikebaComp.data()
expect(defaultData.message).toBe('hello!')
})

})
```

FAIL tests/unit/kaikeba.spec.js

- KaikebaComp > 初始data是vue-text

expect(received).toBe(expected) // Object.is equality

Expected: "hello!"
Received: "vue-text"

```
33 |
34 |     const defaultData = KaikebaComp.data()
> 35 |     expect(defaultData.message).toBe('hello!')
    |                                   ^
36 |   })
37 |
38 | //    // 检查 mount 中的组件实例

at Object.toBe (tests/unit/kaikeba.spec.js:35:33)
```

PASS tests/unit/example.spec.js

检查mounted之后

```
it('mount之后测data是开课吧', () => {
  const vm = new Vue(KaikebaComp).$mount()
  expect(vm.message).toBe('开课吧')
})
```

用户点击

和写vue 没啥本质区别，只不过我们用测试的角度去写代码，vue提供了专门针对测试的 `@vue/test-utils`

测试覆盖率

jest自带覆盖率，如果用的mocha，需要使用istanbul来统计覆盖率

package.json里修改jest配置

```
"jest": {
  "collectCoverage": true,
  "collectCoverageFrom": ["src/**/*.vue"],
}
```

在此执行npm run test:unit

```
> vue-cli-service test:unit

PASS tests/unit/kaikeba.spec.js
PASS tests/unit/example.spec.js

-----|-----|-----|-----|-----|
File      | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s |
-----|-----|-----|-----|-----|-----|
All files | 15.79   | 100      | 0       | 15.79   |                   |
src       | 0       | 100      | 0       | 0       |                   |
  main.js | 0       | 100      | 0       | 0       | 1,2,3,4,6,8,11    |
  router.js | 0       | 100      | 0       | 0       | 1,2,3,5,22        |
  store.js | 0       | 100      | 100      | 0       | 1,2,4             |
src/components | 100     | 100      | 100      | 100      |                   |
  Kaikeba.vue | 100     | 100      | 100      | 100      |                   |
src/views | 0       | 100      | 100      | 0       |                   |
  Home.vue | 0       | 100      | 100      | 0       | 10                |
-----|-----|-----|-----|-----|

Test Suites: 2 passed, 2 total
Tests:       5 passed, 5 total
Snapshots:   0 total
Time:        1.653s
Ran all test suites.
→ vue-test git:(master) x
```

可以看到我们kaikeba.vue的覆盖率是100%

进行修改之后

```
PASS tests/unit/kaikeba.spec.js
PASS tests/unit/example.spec.js

-----|-----|-----|-----|-----|
File      | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s |
-----|-----|-----|-----|-----|-----|
All files | 18.18   | 50        | 0        | 18.18   |                    |
src       | 0        | 100       | 0        | 0        |                    |
  main.js | 0        | 100       | 0        | 0        | 1,2,3,4,6,8,11    |
  router.js | 0        | 100       | 0        | 0        | 1,2,3,5,22        |
  store.js | 0        | 100       | 100       | 0        | 1,2,4              |
src/components | 66.67   | 50        | 100       | 66.67   |                    |
  Kaikeba.vue | 66.67   | 50        | 100       | 66.67   | 22,28              |
src/views  | 0        | 100       | 100       | 0        |                    |
  Home.vue | 0        | 100       | 100       | 0        | 10                  |
-----|-----|-----|-----|-----|

Test Suites: 2 passed, 2 total
Tests:       5 passed, 5 total
Snapshots:   0 total
Time:        2.08s
Ran all test suites
```

现在的代码，依然是测试没有报错，但是覆盖率只有66%了，而且没有覆盖的代码行数，都标记了出来，继续努力加测试吧

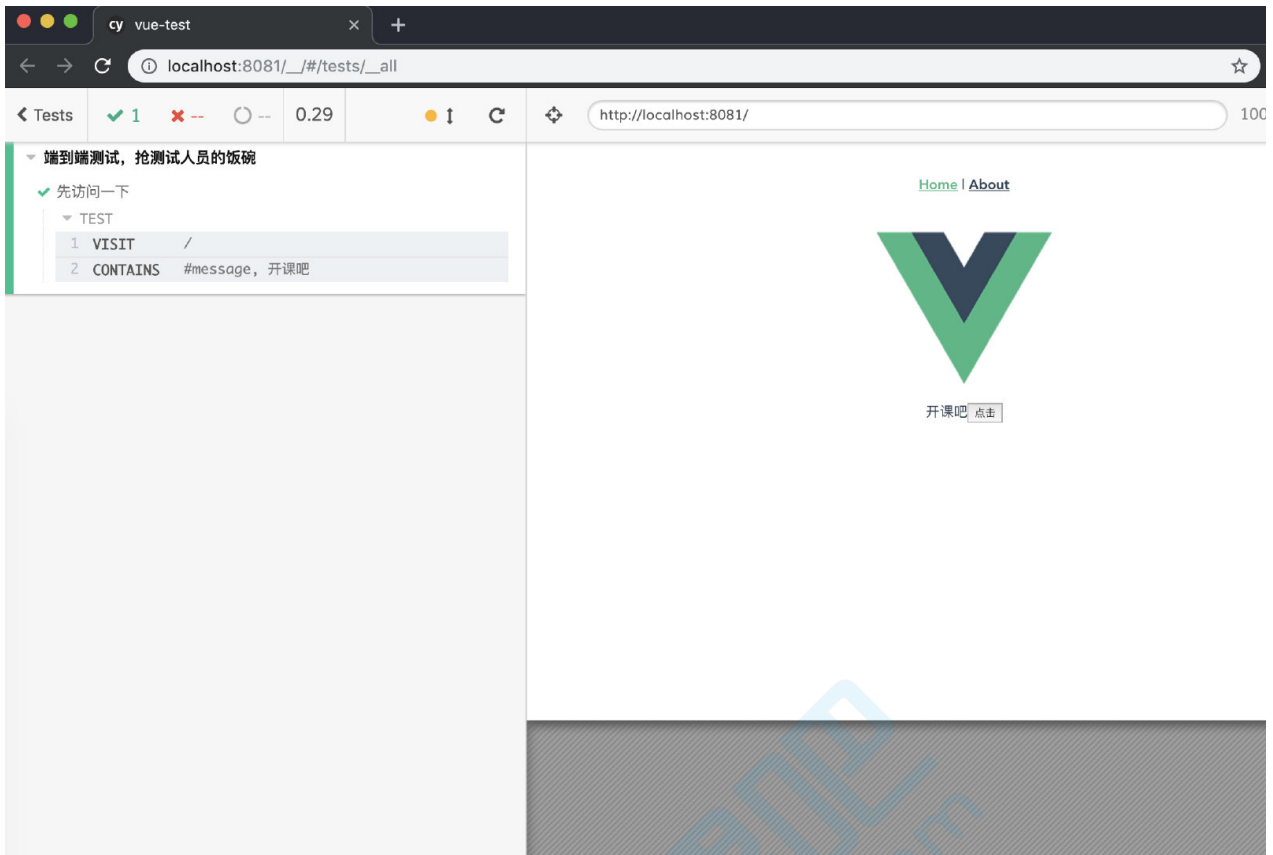
E2E测试

借用浏览器的能力，站在用户测试人员的角度，输入框，点击按钮等，完全模拟用户，这个和具体的框架关系不大，完全模拟浏览器行为

修改e2e/spec/test.js

```
// https://docs.cypress.io/api/introduction/api.html

describe('端到端测试，抢测试人员的饭碗', () => {
  it('先访问一下', () => {
    cy.visit('/')
    // cy.contains('h1', 'welcome to Your Vue.js App')
    cy.contains('#message', '开课吧')
  })
})
```



可以看到是打开了一个浏览器进行测试

测试用户点击

```
// https://docs.cypress.io/api/introduction/api.html

describe('端到端测试, 抢测试人员的饭碗', () => {
  it('先访问一下', () => {
    cy.visit('/')
    // cy.contains('h1', 'Welcome to Your Vue.js App')
    cy.contains('#message', '开课吧')

    cy.get('button').click()
    cy.contains('#message', '按钮点击')
  })
})
```

TDD

所以TDD 就是测试驱动开发模式，就是我们开发一个新功能，先把测试写好，然后测试跑起来，会报错，我们才开始写代码，挨个的把测试跑绿，功能也就完成了

开课吧web全栈架构师

React 自动化测试

React中，也是使用jest来做自动化测试，我们来体验一下

Node自动化测试

node中单测，除了类似vue中的输入输出测试，node很多都是网络接口数据，我们如何是去测试这些数据呢

