Vue.js项目架构最佳实践

按钮权限

封装一个指令v-permission，从而实现按钮级别权限控制，src/directive/permission.js

```
import store from "@/store";

const permission = {
  inserted(el, binding) {
    // 获取指令的值：按钮要求的角色数组
    const { value:pRoles } = binding;
    // 获取用户角色
    const roles = store.getters && store.getters.roles;

    if (pRoles && pRoles instanceof Array && pRoles.length > 0) {
      // 判断用户角色中是否有按钮要求的角色
      const hasPermission = roles.some(role => {
        return pRoles.includes(role);
      });

      // 如果没有权限则删除当前dom
      if (!hasPermission) {
        el.parentNode && el.parentNode.removeChild(el);
      }
    } else {
      throw new Error(`需要指定按钮要求角色数组，如v-permission="['admin','editor']"`);
    }
  }
};

export default permission;
```

注册指令，main.js

```
import vPermission from "./directive/permission";

Vue.directive("permission", vPermission);
```

测试

```
// 添加权限按钮，About.vue
<button v-permission="['admin', 'editor']">editor button</button>
<button v-permission="['admin']">admin button</button>
```

> 该指令只能删除挂载指令的元素，对于那些额外生成的和指令无关的元素无能为力，比如：

```
<el-tabs>
<el-tab-pane label="用户管理" name="first" v-permission="['admin', 'editor']">用户管理</el-tab-pane>
<el-tab-pane label="配置管理" name="second" v-permission="['admin', 'editor']">配置管理</el-tab-pane>
<el-tab-pane label="角色管理" name="third" v-permission="['admin']">角色管理</el-tab-pane>
<el-tab-pane label="定时任务补偿" name="fourth" v-permission="['admin', 'editor']">定时任务补偿</el-tab-pane>
</el-tabs>
```

此时只能使用v-if来实现

```
<template>
    <el-tab-pane v-if="checkPermission(['admin'])">
</template>

<script>
export default {
  methods: {
      checkPermission(permissionRoles) {
          return roles.some(role => {
              return permissionRoles.includes(role);
          });
        }
  }
}
</script>
```

自定义指令参考：https://cn.vuejs.org/v2/guide/custom-directive.html

面包屑

面包屑导航是通过 `$route.matched` 数组动态生成的。

完成效果best-practice-4

- 创建Breadcrumb，./components/Breadcrumb.vue

```
<template>
  <el-breadcrumb class="app-breadcrumb" separator="/">
    <transition-group name="breadcrumb">
      <el-breadcrumb-item v-for="(item,index) in levelList" :key="item.path">
        <span
          v-if="item.redirect==='noRedirect'||index==levelList.length-1"
          class="no-redirect"
        >{{ item.meta.title }}</span>
        <a v-else @click.prevent="handleLink(item)">{{ item.meta.title }}</a>
      </el-breadcrumb-item>
```

```
    </transition-group>
  </el-breadcrumb>
</template>

<script>
import pathToRegexp from "path-to-regexp";

export default {
  data() {
    return {
      levelList: null
    };
  },
  watch: {
    $route: {
      handler(route) {
        this.getBreadcrumb();
      },
      immediate: true
    }
  },

  methods: {
    getBreadcrumb() {
      console.log(this.$route.matched);

      // 面包屑仅显示包含meta.title且item.meta.breadcrumb不为false的路由
      let matched = this.$route.matched.filter(
        item => item.meta && item.meta.title && item.meta.breadcrumb !== false
      );

      // 根路由
      const first = matched[0];

      // 根匹配只要不是home，就作为home下一级
      if (!this.isHome(first)) {
        matched = [{ path: '/', redirect: "/home", meta: { title: "首页" }
}].concat(matched);
      }

      // 处理完指定到levelList
      this.levelList = matched
    },
    isHome(route) {
      const name = route && route.name;
      if (!name) {
        return false;
      }
      return name.trim().toLocaleLowerCase() === "home".toLocaleLowerCase();
    },
    pathCompile(path) {
      const { params } = this.$route;
      var toPath = pathToRegexp.compile(path);
```

```
        return toPath(params);
      },
      handleLink(item) {
        const { redirect, path } = item;
        // 若存在重定向，按重定向走
        if (redirect) {
          this.$router.push(redirect);
          return;
        }
        // 编译path，避免存在路径参数
        this.$router.push(this.pathCompile(path));
      }
    }
};
</script>

<style scoped>
.app-breadcrumb.el-breadcrumb {
  display: inline-block;
  font-size: 14px;
  line-height: 50px;
  margin-left: 8px;
}
.app-breadcrumb.el-breadcrumb .no-redirect {
  color: #97a8be;
  cursor: text;
}

/* breadcrumb transition */
.breadcrumb-enter-active,
.breadcrumb-leave-active {
  transition: all .5s;
}

.breadcrumb-enter,
.breadcrumb-leave-active {
  opacity: 0;
  transform: translateX(20px);
}

.breadcrumb-move {
  transition: all .5s;
}

.breadcrumb-leave-active {
  position: absolute;
}
</style>
```

> 动画相关API参考

数据交互

封装request

安装axios: `npm i axios -S`

创建@/utils/request.js

```
import axios from "axios";
import { MessageBox, Message } from "element-ui";
import store from "@/store";
import { getToken } from "@/utils/auth";

// 创建axios实例
const service = axios.create({
  baseURL: process.env.VUE_APP_BASE_API, // url基础地址，解决不同数据源url变化问题
  // withCredentials: true, // 跨域时若要发送cookies需设置该选项
  timeout: 5000 // 超时
});

// 请求拦截
service.interceptors.request.use(
  config => {
    // do something

    if (store.getters.token) {
      // 设置令牌请求头
      config.headers["X-Token"] = getToken();
    }
    return config;
  },
  error => {
    // 请求错误预处理
    //console.log(error) // for debug
    return Promise.reject(error);
  }
);

// 响应拦截
service.interceptors.response.use(
  // 通过自定义code判定响应状态，也可以通过HTTP状态码判定
  response => {
    // 仅返回数据部分
    const res = response.data;

    // code不为1则判定为一个错误
    if (res.code !== 1) {
      Message({
        message: res.message || "Error",
        type: "error",
        duration: 5 * 1000
      });

      // 假设：10008-非法令牌；10012-其他客户端已登录；10014-令牌过期；
```

```
      if (res.code === 10008 || res.code === 10012 || res.code === 10014) {
        // 重新登录
        MessageBox.confirm(
          "登录状态异常，请重新登录",
          "确认登录信息",
          {
            confirmButtonText: "重新登录",
            cancelButtonText: "取消",
            type: "warning"
          }
        ).then(() => {
          store.dispatch("user/resetToken").then(() => {
            location.reload();
          });
        });
      }
      return Promise.reject(new Error(res.message || "Error"));
    } else {
      return res;
    }
  },
  error => {
    //console.log("err" + error); // for debug
    Message({
      message: error.message,
      type: "error",
      duration: 5 * 1000
    });
    return Promise.reject(error);
  }
);

export default service;
```

设置VUE_APP_BASE_API环境变量，创建.env.development文件

```
# base api
VUE_APP_BASE_API = '/dev-api'
```

> 环境变量和模式

添加token的getter方法

```
token: state => state.user.token,
```

测试代码，创建@/api/user.js

```
import request from '@/utils/request'

export function login(data) {
  return request({
```

```
    url: '/user/login',
    method: 'post',
    data
  })
}

export function getInfo() {
  return request({
    url: '/user/info',
    method: 'get'
  })
}
```

数据mock

数据模拟两种常见方式，**本地mock**和**线上esay-mock**

- 本地mock

  修改vue.config.js，给devServer添加相关代码：

```
const bodyParser = require("body-parser");

module.exports = {
  devServer: {
    before: app => {
      app.use(bodyParser.json());
      app.use(
        bodyParser.urlencoded({
          extended: true
        })
      );

      app.post("/dev-api/user/login", (req, res) => {
        const { username } = req.body;

        if (username === "admin" || username === "jerry") {
          res.json({
            code: 1,
            data: username
          });
        } else {
          res.json({
            code: 10204,
            message: "用户名或密码错误"
          });
        }
      });

      app.get("/dev-api/user/info", (req, res) => {
```

```
        const roles = req.headers['x-token'] === "admin" ? ["admin"] : ["editor"];
        res.json({
          code: 1,
          data: roles
        });
      });
    }
  }
}
```

> post请求需额外安装依赖：`npm i body-parser -D`

调用接口，@/store/modules/user.js

```
import {login, getInfo} from '@/api/user';

const actions = {
  login({ commit }, userInfo) {
    // 调用并处理结果，错误处理已拦截无需处理
    return login(userInfo).then((res) => {
      commit("SET_TOKEN", res.data);
      setToken(res.data);
    });
    // ...之前代码不需要了
  },

  // get user info
  getInfo({ commit, state }) {
    return getInfo(state.token).then(({data: roles}) => {
      commit("SET_ROLES", roles);
      return {roles}
    })
    // ...之前代码不需要了
  },
};
```

- esay-mock

  使用步骤：

  1. 登录easy-mock网站
  2. 创建一个项目
  3. 创建需要的接口
  4. 调用：修改base_url，.env.development

     ```
     VUE_APP_BASE_API = 'https://easy-mock.com/mock/5cdcc3fdde625c6ccadfd70c/kkb-cart'
     ```

解决跨域

如果请求的接口在另一台服务器上，开发时则需要设置代理避免跨域问题：

- 添加代理配置，vue.config.js

```
devServer: {
    port: port,
    proxy: {
      // 代理 /dev-api/user/login 到 http://127.0.0.1:3000/user/login
      [process.env.VUE_APP_BASE_API]: {
        target: `http://127.0.0.1:3000/`,
        changeOrigin: true,
        pathRewrite: {
          ["^" + process.env.VUE_APP_BASE_API]: ""
        }
      }
    },
}
```

- 创建一个独立接口服务器，~/test-server/index.js

```
const express = require("express");
const app = express();
const bodyParser = require("body-parser");

app.use(bodyParser.json());
app.use(
  bodyParser.urlencoded({
    extended: true
  })
);

app.post("/user/login", (req, res) => {
  const { username } = req.body;

  if (username === "admin" || username === "jerry") {
    res.json({
      code: 1,
      data: username
    });
  } else {
    res.json({
      code: 10204,
      message: "用户名或密码错误"
    });
  }
});

app.get("/user/info", (req, res) => {
  const roles = req.headers['x-token'] === "admin" ? ["admin"] : ["editor"];
  res.json({
    code: 1,
    data: roles
  });
});
```

```
app.listen(3000);
```

- 测试：把before选项注释掉

项目测试阶段

测试分类

常见的开发流程里，都有测试人员，他们不管内部实现机制，只看最外层的输入输出，这种我们称为**黑盒测试。**比如你写一个加法的页面，会设计N个用例，测试加法的正确性，这种测试我们称之为**E2E测试。**

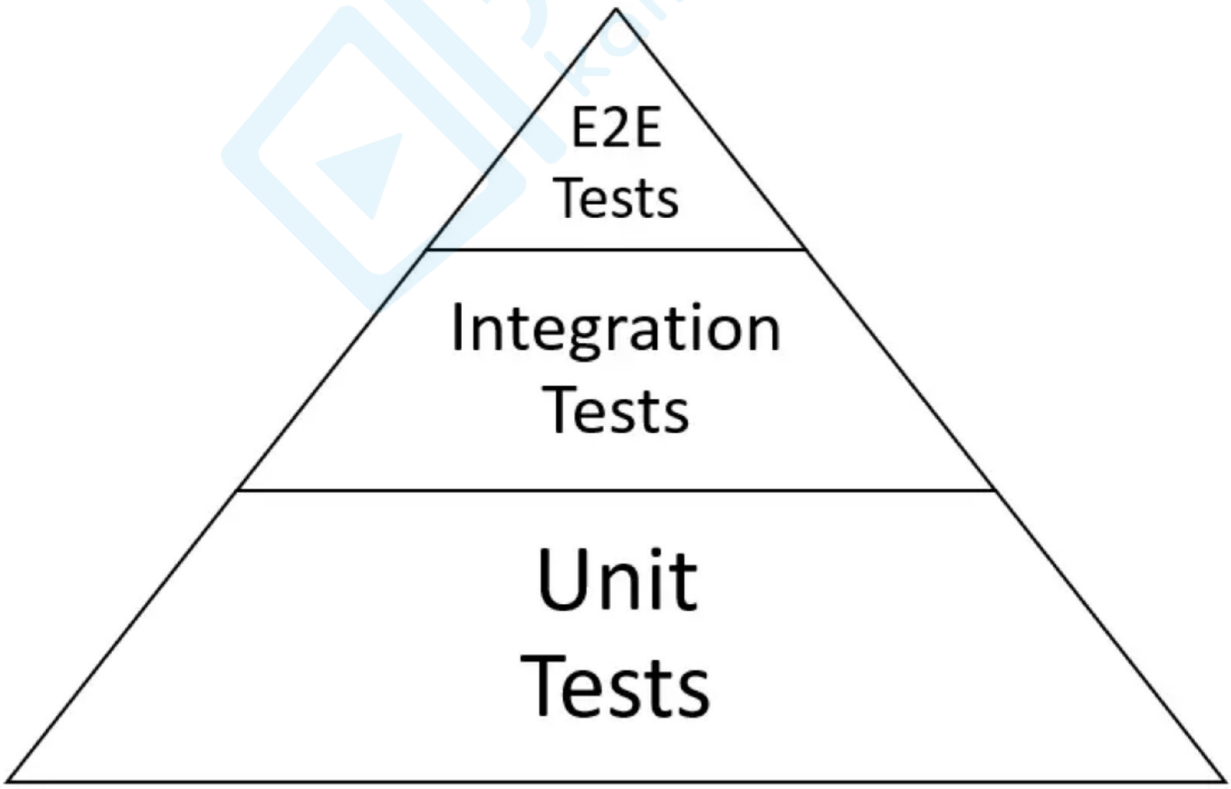
更负责一些的我们称之为**集成测试**，就是集合多个测试过的单元一起测试。

还有一种测试叫做**白盒测试**，我们针对一些内部核心实现逻辑编写测试代码，称之为**单元测试。**

编写测试代码的好处

- 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



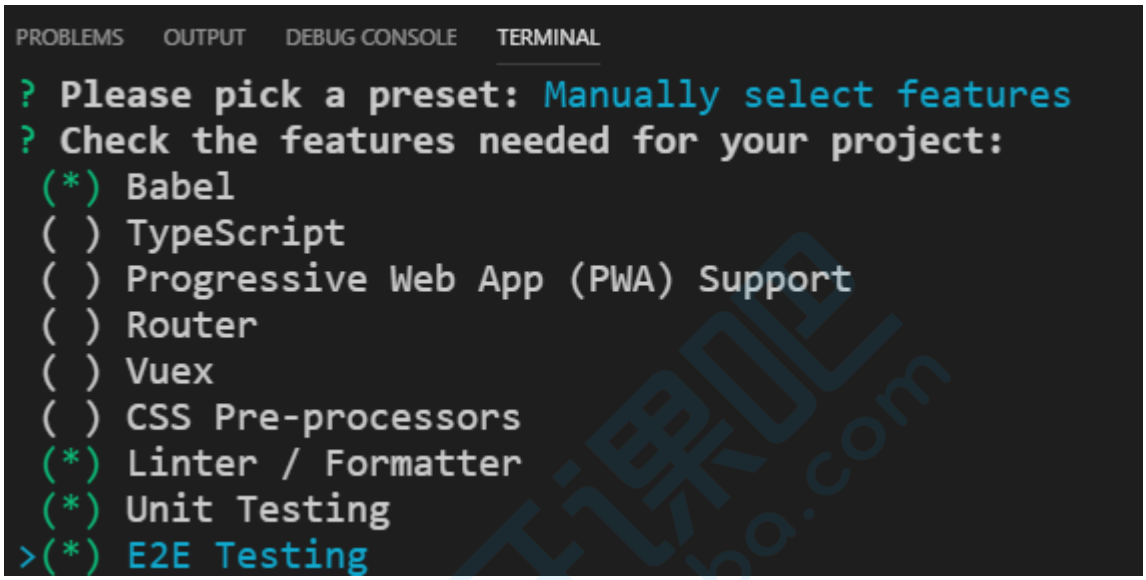
准备工作

在vue中，推荐用Mocha+Chai或者Jest，演示代码使用Jest，它们语法基本一致

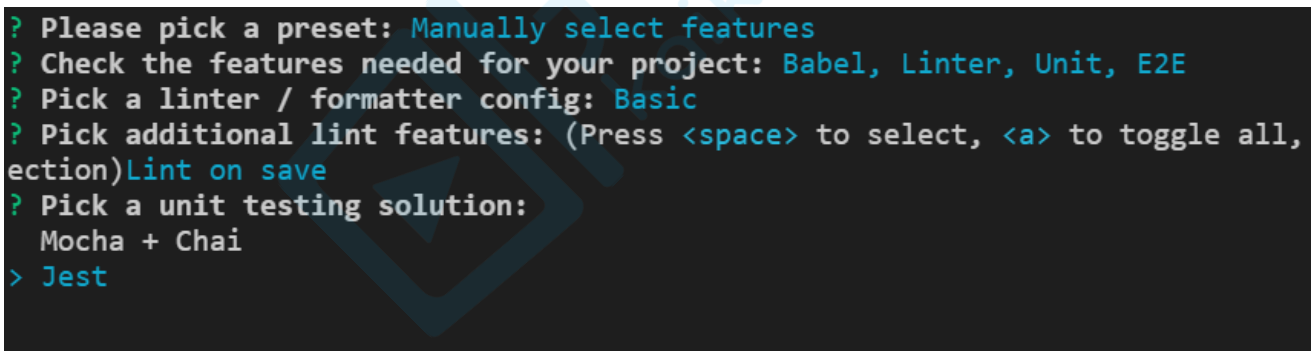
> 要完成测试任务，需要测试框架（跑测试）、断言库（编写测试）和编程框架特有的测试套件。
>
> 上面的Mocha是测试框架，Chai是断言库，Jest同时包含两者。
>
> vue中的组件等测试代码的编写需要vue-test-utils套件支持。

新建vue项目时

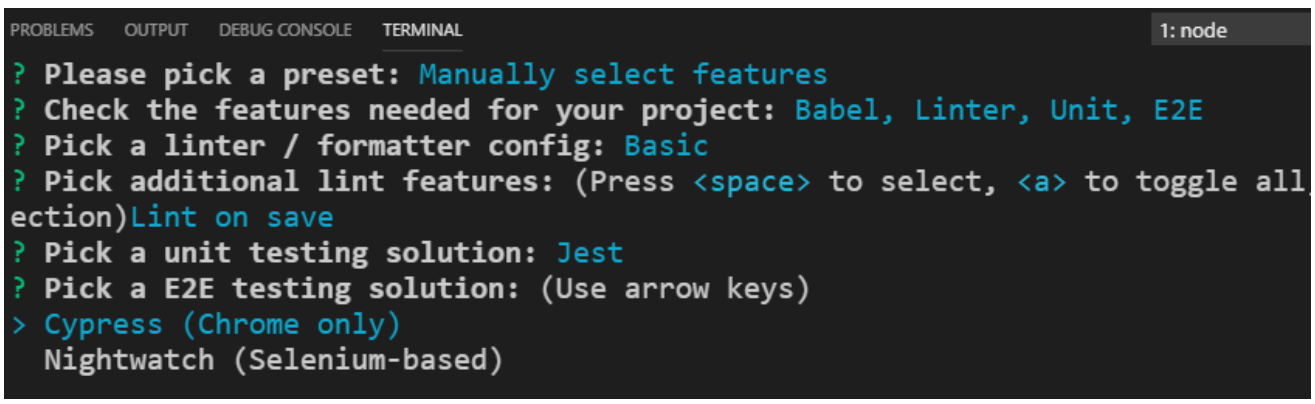
- 选择特性`Unit Testing`和`E2E Testing`



- 单元测试解决方案选择：`Jest`



- 端到端测试解决方案选择：Cypress



在已存在项目中集成

运行：`vue add @vue/unit-jest`和`vue add @vue/e2e-cypress`

编写单元测试

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。

- 新建test/unit/kaikeba.spec.js，`*.spec.js`是命名规范

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

// 测试套件 test suite
describe('Kaikeba', () => {
    // 测试用例 test case
    it('测试add函数', () => {
        // 断言 assert
        expect(add(1, 3)).toBe(3)
        expect(add(1, 3)).toBe(4)
        expect(add(-2, 3)).toBe(1)
    })
})
```

执行单元测试

- 执行：`npm run test:unit`

```
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
```

断言API简介

- `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

> 这里面仅演示了toBe，更多断言API

测试Vue组件

- 创建一个vue组件components/Kaikeba.vue

```
<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>
```

测试该组件，

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

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', () => {
    // 检查data函数存在性
    expect(typeof KaikebaComp.data).toBe('function')
    // 检查data返回的默认值
    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

用测试人员的角度去写代码，Vue提供了专门针对的测试套件 `@vue/test-utils`

安装：`npm i -D @vue/test-utils`

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

it("按钮点击后", () => {
    const wrapper = mount(KaikebaComp);
    wrapper.find("button").trigger("click");
    // 测试数据变化
    expect(wrapper.vm.message).toBe("按钮点击");
    // 测试html渲染结果
    expect(wrapper.find("span").html()).toBe("<span>按钮点击</span>");
    // 等效的方式
    expect(wrapper.find("span").text()).toBe("按钮点击");
});
```

> Vue Test Utils 是 Vue.js 官方的单元测试实用工具库。

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/**/*.{js,vue}"],
  }
```

> 若采用独立配置，则修改jest.config.js:
>
> ```
> module.exports = {
>     ...
>     "collectCoverage": true,
>     "collectCoverageFrom": ["src/**/*.{js,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) ✗
```

> %stmts是语句覆盖率（statement coverage）：是不是每个语句都执行了?
>
> %Branch分支覆盖率（branch coverage）：是不是每个if代码块都执行了?
>
> %Funcs函数覆盖率（function coverage）：是不是每个函数都调用了?
>
> %Lines行覆盖率（line coverage）：是不是每一行都执行了?

可以看到我们kaikeba.vue的覆盖率是100%，我们修改一下代码

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

<script>
export default {
  data() {
    return {
      message: "vue-text",
      count: 0
    };
  },
  created() {
    this.message = "开课吧";
  },
  methods: {
    changeMsg() {
      if (this.count > 1) {
        this.message = "count大于1";
      } else {
        this.message = "按钮点击";
      }
    },
    changeCount() {
      this.count += 1;
    }
  }
};
</script>
```

```
 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

> Vue组件单元测试cookbook
>
> Vue Test Utils使用指南

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。

运行E2E测试

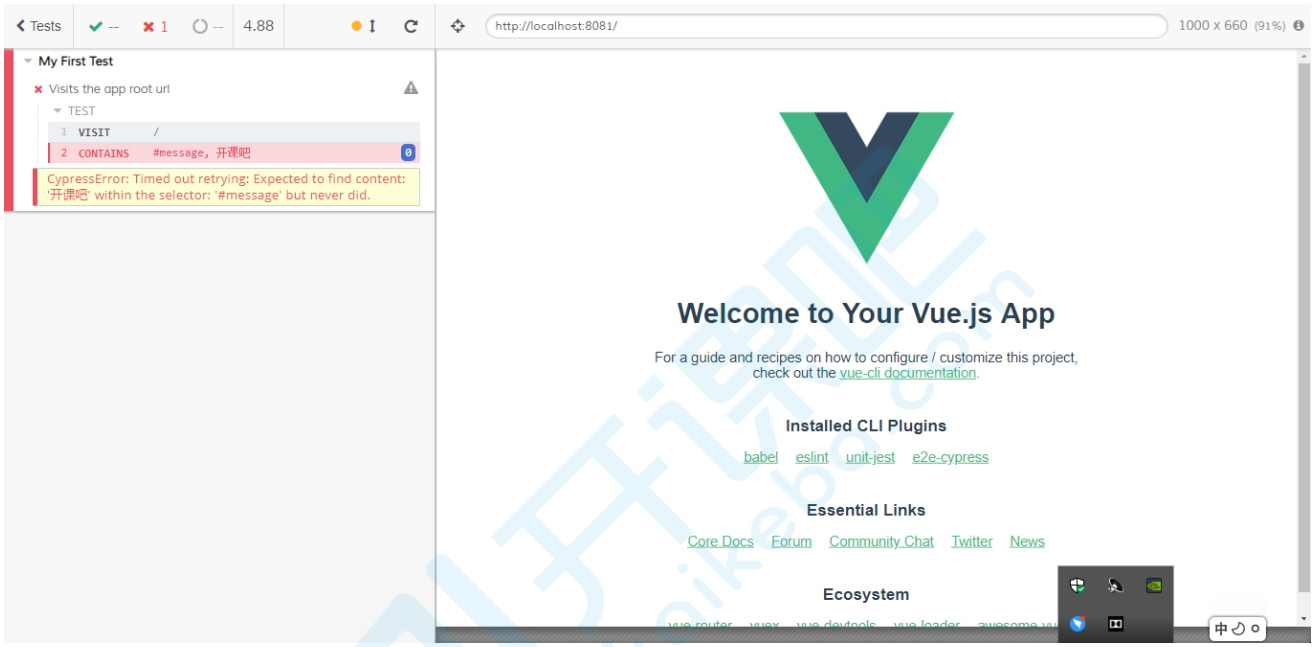
```
npm run test:e2e
```

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('span', '开课吧')

  })
})
```

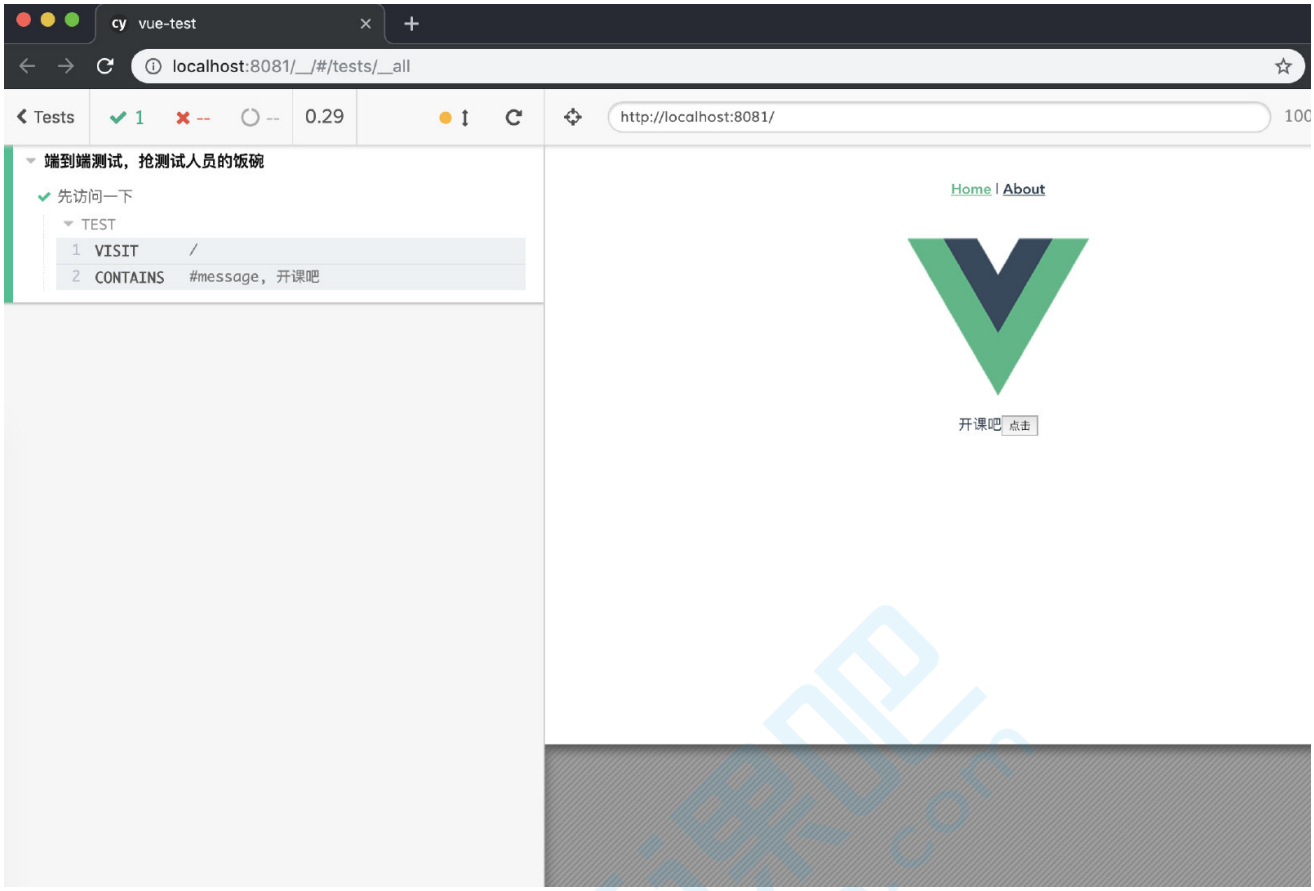


测试未通过，因为没有使用Kaikeba.vue，修改App.vue

```
<div id="app">
    <img alt="Vue logo" src="./assets/logo.png">
    <!-- <HelloWorld msg="Welcome to Your Vue.js App"/> -->
    <Kaikeba></Kaikeba>
</div>

import Kaikeba from './components/Kaikeba.vue'
export default {
  name: 'app',
  components: {
    HelloWorld,Kaikeba
  }
}
```

测试通过~



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('span', '按钮点击')

  })
})
```

项目部署阶段

创建

- 环境变量， 创建.env.production

```
# 这个前缀可以让nginx作反代
VUE_APP_BASE_API = '/api'
```

- 改一下输出地址，vue.config.js

```
outputDir: 'dist/best-practice',
```

- 构建：`npm run build`

部署

部署只需要将最终生成的静态文件，通常是 `dist` 文件夹下的静态文件发布到 cdn 或者静态服务器即可。

这里我们采用nginx作为web服务器，具体配置nginx-1.14.0/conf/nginx.conf：

```
# 项目根，大家改成自己的目录
root C:\\Users\\yt037\\Desktop\\kaikeba\\projects\\vue-study\\dist;  #项目dist目录
# history fallback
location /best-practice {
    try_files $uri /best-practice/index.html;
}
 #反向代理，实现接口转发
 location ^~ /api/ {
     # 把/api去除
    rewrite ^/api/(.*)$ /$1 break;
    # 代理到3000服务上
    proxy_pass http://localhost:3000;
 }
```

git hooks

利用git hooks在每次提交代码时执行lint

- 执行过程如下：commit => git hooks => test&&lint
- 完成这两项任务需要安装：husky和lint-staged

```
npm install husky lint-staged -D
```

- 配置，package.json

```
"husky": {
  "hooks": {
    "pre-commit": "lint-staged"
  }
},
"lint-staged": {
    "src/**/*.{js,vue}": [
        "eslint --fix",
        "git add"
    ]
}
```