vue项目最佳实践

复习

- 编译器工作原理
  - 解析 parse：HTML string =》 AST
  - 优化 optimize：静态根节点和静态节点
  - 生成 generate：render函数代码字符串
- 组件化
  - 组件声明、注册：Vue.component('comp', {})
  - 组件实例化和挂载：两个createComponent()
    - createComponent create-component.js

      获取自定义组件vnode，添加初始化钩子
    - createComponent patch.js

      用初始化钩子实例化并且挂载

作业

- 尝试解答双向绑定实现原理
  - 编译阶段：对v-model进行特殊处理

```
// 生成的渲染函数
(function anonymous() {
    with(this){return _c('div',{attrs:{"id":"demo"}},[
        _c('h1',[_v("双向绑定机制")]),_v(" "),
        _c('input',{directives:[{name:"model",rawName:"v-model",value:
(foo),expression:"foo"}],attrs:{"type":"text"},domProps:{"value":
(foo)},on:{"input":function($event)
{if($event.target.composing)return;foo=$event.target.value}}}),_v(" "),
        _c('comp',{model:{value:(foo),callback:function ($$v)
{foo=$$v},expression:"foo"}})
    ],1)}
})

// input
_c('input',{
    directives:[{
        name:"model",
        rawName:"v-model",
```

```
        value:(foo),
        expression:"foo"}],
    attrs:{"type":"text"},
    domProps:{"value":(foo)},
    on:{
        "input":function($event){
            if($event.target.composing) return;
            foo=$event.target.value
        }
    }
})
// comp
// <comp value="foo" @input="">
_c('comp',{
    model:{
        value:(foo),
        callback:function ($$v) {foo=$$v},
        expression:"foo"
    }
})
```

- 初始化阶段：对节点赋值及事件监听

  对节点赋值 platforms\web\runtime\modules\dom-props.js

  事件监听 platforms\web\runtime\modules\events.js

  额外的model指令 platforms\web\runtime\directives\model.js

  自定义组件会转换为属性和事件 core/vdom/create-component.js

  自定义组件事件监听 core\instance\events.js

- 不同类型输入项编译结果和后续处理是不同的

```
_c('input', {
    directives: [{ name: "model", rawName: "v-model", value: (foo),
expression: "foo" }],
    attrs: { "type": "checkbox" },
    domProps: { "checked": Array.isArray(foo) ? _i(foo, null) > -1 :
(foo) },
    on: {
        "change": function ($event) {
            var $$a = foo,
                $$el = $event.target,
                $$c = $$el.checked ? (true) : (false);
            if (Array.isArray($$a)) {
                var $$v = null, $$i = _i($$a, $$v);
                if ($$el.checked) { $$i < 0 && (foo =
$$a.concat([$$v])) }
                else {
                    $$i > -1 && (foo = $$a.slice(0,
$$i).concat($$a.slice($$i + 1)))
                }
            } else {
                foo = $$c
            }
```

```
        }
    }
})
```

- 自定义组件可以指定v-model事件名和属性名

```
model:{
    prop:'foo',
    event:"change"
}
```

- 回答类似面试题
  - WWWH
  - what
  - why
  - where
  - H

资源

- Vue-CLI 3.0
- vue-element-admin

知识点

项目配置

创建vue.config.js，指定应用上下文、端口号、主页title

```
// vue.config.js
const port = 7070;
const title = "vue项目最佳实践";

module.exports = {
  publicPath: '/best-practice', // 部署应用包时的基本 URL
  devServer: {
    port: port,
  },
  configureWebpack: {
    // 向index.html注入标题
    name: title
  }
};
```

链式操作：svg icon引入

安装依赖：svg-sprite-loader

```
npm i svg-sprite-loader -D
```

下载图标，存入src/icons/svg中

修改规则和新增规则，vue.config.js

```
// resolve定义一个绝对路径获取函数
const path = require('path')

function resolve(dir) {
  return path.join(__dirname, dir)
}
//...
chainWebpack(config) {
    // 配置svg规则排除icons目录中svg文件处理
    config.module
        .rule("svg")
            .exclude.add(resolve("src/icons"))
            .end();
    // 新增icons规则，设置svg-sprite-loader处理icons目录中的svg
    config.module
        .rule("icons")
            .test(/\.svg$/)
                .include.add(resolve("src/icons"))
                .end()
            .use("svg-sprite-loader")
            .loader("svg-sprite-loader")
                .options({ symbolId: "icon-[name]" })
                .end();
}
```

图标自动导入

```
// icons/index.js
const req = require.context('./svg', false, /\.svg$/)
req.keys().map(req);

// main.js
import './icons'
```

创建SvgIcon组件，./components/SvgIcon.vue

```
<template>
  <svg :class="svgClass" aria-hidden="true" v-on="$listeners">
    <use :xlink:href="iconName" />
  </svg>
</template>

<script>
export default {
  name: 'SvgIcon',
  props: {
    iconClass: {
      type: String,
      required: true
```

```
    },
    className: {
      type: String,
      default: ''
    }
  },
  computed: {
    iconName() {
      return `#icon-${this.iconClass}`
    },
    svgClass() {
      if (this.className) {
        return 'svg-icon ' + this.className
      } else {
        return 'svg-icon'
      }
    }
  }
}
</script>

<style scoped>
.svg-icon {
  width: 1em;
  height: 1em;
  vertical-align: -0.15em;
  fill: currentColor;
  overflow: hidden;
}
</style>
```

权限控制和动态路由

路由定义

路由分为两种：`constantRoutes`和`asyncRoutes`，router.js

```
import Vue from "vue";
import Router from "vue-router";
import Layout from '@/layout'; // 布局页

Vue.use(Router);

// 通用页面：不需要守卫，可直接访问
export const constRoutes = [
  {
    path: "/login",
    component: () => import("@/views/Login"),
    hidden: true // 导航菜单忽略该项
  },
  {
    path: "/",
    component: Layout,// 应用布局
    redirect: "/home",
    children: [
```

```
      {
        path: "home",
        component: () =>
          import(/* webpackChunkName: "home" */ "@/views/Home.vue"),
        name: "home",
        meta: {
            title: "Home", // 导航菜单项标题
            icon: "qq" // 导航菜单项图标
        }
      }
    ]
  }
];

// 权限页面：受保护页面，要求用户登录并拥有访问权限的角色才能访问
export const asyncRoutes = [
  {
    path: "/about",
    component: Layout,
    redirect: "/about/index",
    children: [
      {
        path: "index",
        component: () =>
          import(/* webpackChunkName: "home" */ "@/views/About.vue"),
        name: "about",
        meta: {
            title: "About",
            icon: "qq",
            roles: ['admin', 'editor']
        },
      }
    ]
  }
];

export default new Router({
  mode: "history",
  base: process.env.BASE_URL,
  routes: constRoutes
});
```

创建布局页面，layout/index.vue

```
<template>
  <div class="app-wrapper">
    <!-- <sidebar class="sidebar-container" /> -->
    <div class="main-container">
      <router-view />
    </div>
  </div>
</template>
```

创建用户登录页面，views/Login.vue

```
<template>
```

```
  <div>
    <h2>用户登录</h2>
    <div>
      <input type="text" v-model="username">
      <button @click="login">登录</button>
    </div>
  </div>
</template>
<script>
export default {
  data() {
    return {
      username: "admin"
    };
  },
  methods: {
    login() {}
  }
};
</script>
```

用户登录状态维护

vuex根模块实现，创建store/index.js

```
import Vue from 'vue'
import Vuex from 'vuex'
import user from './modules/user'

Vue.use(Vuex)

const store = new Vuex.Store({
  modules: {user}
})

export default store
```

user模块：store/modules/user.js

```
const state = {
  token: localStorage.getItem('token'),
  // 其他用户信息
};

const mutations = {
  SET_TOKEN: (state, token) => {
    state.token = token;
  }
};

const actions = {
  // 模拟用户登录
  login({ commit }, userInfo) {
    const { username } = userInfo;
    return new Promise((resolve, reject) => {
```

```
      setTimeout(() => {
        if (username === "admin" || username === "jerry") {
          commit("SET_TOKEN", username);
          localStorage.setItem('token', username);
          resolve();
        } else {
          reject("用户名、密码错误");
        }
      }, 1000);
    });
  }
};

export default {
  namespaced: true,
  state,
  mutations,
  actions
};
```

请求登录，Login.vue

```
login() {
    this.$store
        .dispatch("user/login", { username: this.username })
        .then(() => {
        this.$router.push({
            path: this.$route.query.redirect || "/"
        });
    }).catch(error => {
        alert(error);
    });
}
```

路由守卫

创建./src/permission.js

```
import router from './router'

const whiteList = ['/login'] // 无需令牌白名单

router.beforeEach((to, from, next) => {

  // 获取令牌判断用户是否登录
  const hasToken = localStorage.getItem('token')

  // 已登录
  if (hasToken) {
    if (to.path === '/login') {
      // 若已登录没有必要显示登录页，重定向至首页
      next({ path: '/' })
    } else {
      // 去其他路由，暂时放过
```

```
      next()
      // 接下来执行用户角色逻辑，todo
    }
  } else {// 未登录
    if (whiteList.indexOf(to.path) !== -1) {
      // 白名单中路由放过
      next()
    } else {
      // 重定向至登录页
      next(`/login?redirect=${to.path}`)
    }
  }
})
```

用户角色获取和维护

添加用户角色和获取逻辑，store/modules/user.js

```
const state = {
  roles: []
  // 其他用户信息
};

const mutations = {
  SET_ROLES: (state, roles) => {
    state.roles = roles;
  }
};

const actions = {
  // 模拟获取用户信息，角色和令牌内容一致
  getInfo({ commit, state }) {
    return new Promise((resolve) => {
      setTimeout(() => {
        const roles = state.token === 'admin' ? ['admin'] : ['editor']
        commit("SET_ROLES", roles);
        resolve({roles});
      }, 1000);
    });
  }
};
```

获取用户角色，判断用户是否拥有访问权限，permission.js

```
// 引入store
import store from './store'

router.beforeEach(async (to, from, next) => {
  // ...
  if (hasToken) {
    if (to.path === '/login') {}
    else {
      // next()
      // 若用户角色已附加则说明权限以判定，动态路由已添加
```

```
      const hasRoles = store.state.user.roles && store.state.user.roles.length >
0
      if (hasRoles) {
        next() // 继续即可
      } else {
        try {
          // 先请求获取用户信息
          const { roles } = await store.dispatch('user/getInfo')

          // 动态路由生成，todo
          next() // 先放过，待会删掉
        } catch (error) {
          // 出错需重置令牌并重新登录（令牌过期、网络错误等原因）
          await store.dispatch('user/resetToken')
          next(`/login?redirect=${to.path}`)
          alert(error || '未知错误')
        }
      }
    }
  } else {// 未登录
  }
})
```

添加动态路由

根据用户角色过滤出可访问路由并动态添加到router。

创建permission模块， store/modules/permission.js

```
import { asyncRoutes, constRoutes } from "@/router";

const state = {
  routes: [], // 完整路由表
  addRoutes: [] // 用户可访问路由表
};

const mutations = {
  SET_ROUTES: (state, routes) => {
    state.addRoutes = routes;
    state.routes = constRoutes.concat(routes);
  }
};

const actions = {
  // 路由生成：在得到用户角色后会第一时间调用
  generateRoutes({ commit }, roles) {
    return new Promise(resolve => {
      // 根据角色做过滤处理
      const accessedRoutes = filterAsyncRoutes(asyncRoutes, roles);;
      commit("SET_ROUTES", accessedRoutes);
      resolve(accessedRoutes);
    });
  }
};
```

```
/**
 * 递归过滤AsyncRoutes路由表
 * @routes 待过滤路由表，首次传入的就是AsyncRoutes
 * @roles 用户拥有角色
 */
export function filterAsyncRoutes(routes, roles) {
  const res = [];

  routes.forEach(route => {
    // 复制一份
    const tmp = { ...route };
    // 如果用户有访问权则加入结果路由表
    if (hasPermission(roles, tmp)) {
      // 如果存在子路由则递归过滤之
      if (tmp.children) {
        tmp.children = filterAsyncRoutes(tmp.children, roles);
      }
      res.push(tmp);
    }
  });

  return res;
}

/**
 * 根据路由meta.role确定是否当前用户拥有访问权限
 * @roles 用户拥有角色
 * @route 待判定路由
 */
function hasPermission(roles, route) {
  // 如果当前路由有roles字段则需判断用户访问权限
  if (route.meta && route.meta.roles) {
    // 若用户拥有的角色中有被包含在待判定路由角色表中的则拥有访问权
    return roles.some(role => route.meta.roles.includes(role));
  } else {
    // 没有设置roles则无需判定即可访问
    return true;
  }
}

export default {
  namespaced: true,
  state,
  mutations,
  actions
};
```

调用路由生成逻辑，./permission.js

```
// 根据当前用户角色动态生成路由
const accessRoutes = await store.dispatch('permission/generateRoutes', roles)

// 添加这些路由至路由器
router.addRoutes(accessRoutes)

// 继续路由切换，确保addRoutes完成
next({ ...to, replace: true })
```

> 异步获取路由表
>
> 可以当用户登录后**向后端请求可访问的路由表**，从而动态生成可访问页面，操作和原来是相同的，这里多了一步将后端返回路由表中**组件名称和本地的组件映射**步骤：

```
// 前端组件名和组件映射表
const map = {
    //login: require('@/views/login').default  // 同步的方式
    login: () => import('@/views/login')     // 异步的方式
}
// 服务端返回的map类似于
const serviceMap = [
    { path: '/login', component: 'login', hidden: true }
]
// 遍历serviceMap，将component替换为map[component]，动态生成asyncRoutes
function mapComponent(serviceMap) {
 serviceMap.forEach(route => {
     route.component = map[route.component];
     if(route.children) {
       route.children.map(child => mapComponent(child))
     }
 })
}
mapComponent(serviceMap)
```

权限逻辑修改

演示项目中权限控制逻辑是通过获取当前用户的角色去比对路由表，生成当前用户有权访问的路由表，通过 `router.addRoutes` 动态挂载到 `router` 上。若要自定义，可以修改@/permission.js

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

导航菜单生成

导航菜单是根据路由信息并结合权限判断而动态生成的。它需要支持路由的多级嵌套，所以这里要用到递归组件。

low版实现

菜单结构是典型递归组件，利用之前实现的tree组件，快速看效果

数据准备，添加getter方法，store/index.js

```
getters: {
    permission_routes: state => state.permission.routes,
}
```

复制components/tree，重命名SideMenu

修改SideMenu/index.vue

```
<template>
  <div>
    <ul>
      <!-- 2.遍历permission_routes -->
      <!-- 传递base-path是由于子路由是相对地址 -->
      <item
        :model="route"
        v-for="route in permission_routes"
        :key="route.path"
        :base-path="route.path"
      ></item>
    </ul>
  </div>
</template>
<script>
import { mapGetters } from "vuex";
import Item from "./Item"; // 修改引入
```

```
export default {
  components: { Item },
   computed: {
    ...mapGetters(["permission_routes"])
   }
};
</script>
```

Item.vue改造

```
<template>
  <!-- 1.hidden存在则不显示 -->
  <li v-if="!model.hidden">
        <div @click="toggle">
      <!-- 2.设置icon才显示图标 -->
      <svg-icon v-if="hasIcon" :icon-class="model.meta.icon"></svg-icon>

      <span v-if="isFolder">
        <!-- 3.设置标题才显示 -->
        <span v-if="hasTitle">{{model.meta.title}}</span>
        [{{open ? '-' : '+'}}]
      </span>

      <!-- 4.如果是叶子节点，显示为链接 -->
      <template v-else>
        <router-link :to="resolvePath(model.path)">{{title}}</router-link>
      </template>
    </div>

    <!-- 5.子树设置base-path -->
    <ul v-show="open" v-if="isFolder">
      <item
        class="item"
        v-for="route in model.children"
        :model="route"
        :key="route.path"
        :base-path="resolvePath(model.path)"
      ></item>
    </ul>
  </li>
</template>
<script>
// resolvePath方法需要用到path库
import path from 'path'
export default {
  name: "Item",
  props: {
    // 新增basePath保存父路由path
    basePath: {
      type: String,
      default: ''
    }
  },
  computed: {
    hasIcon() { return this.model.meta && this.model.meta.icon },
    hasTitle() { return this.model.meta && this.model.meta.title },
```

```
    title() {
      return this.hasTitle
        ? this.model.meta.title
        : this.model.name
        ? this.model.name
        : this.model.path;
    }
  },
  methods: {
    // 拼接子路由完整path
    resolvePath(routePath) {
      return path.resolve(this.basePath, routePath)
    }
  }
};
</script>
```

在layout/index.vue应用

利用element做一个更高逼格的导航

创建侧边栏组件，components/Sidebar/index.vue

```
<template>
  <div>
    <el-scrollbar wrap-class="scrollbar-wrapper">
      <el-menu
        :default-active="activeMenu"
        :background-color="variables.menuBg"
        :text-color="variables.menuText"
        :unique-opened="false"
        :active-text-color="variables.menuActiveText"
        :collapse-transition="false"
        mode="vertical"
      >
        <sidebar-item
          v-for="route in permission_routes"
          :key="route.path"
          :item="route"
          :base-path="route.path"
        />
      </el-menu>
    </el-scrollbar>
  </div>
</template>

<script>
import { mapGetters } from "vuex";
import SidebarItem from "./SidebarItem";

export default {
  components: { SidebarItem },
  computed: {
    ...mapGetters(["permission_routes"]),
    activeMenu() {
```

```
      const route = this.$route;
      const { meta, path } = route;
      // 默认激活项
      if (meta.activeMenu) {
        return meta.activeMenu;
      }
      return path;
    },
    variables() {
      return {
        menuText: "#bfcbd9",
        menuActiveText: "#409EFF",
        menuBg: "#304156"
      };
    }
  }
};
</script>
```

创建侧边栏菜项目组件，layout/components/Sidebar/SidebarItem.vue

```
<template>
  <div v-if="!item.hidden" class="menu-wrapper">

    <template v-if="hasOneShowingChild(item.children,item) &&
(!onlyOneChild.children||onlyOneChild.noShowingChildren)&&!item.alwaysShow">
      <router-link v-if="onlyOneChild.meta"
:to="resolvePath(onlyOneChild.path)">
        <el-menu-item :index="resolvePath(onlyOneChild.path)" :class="{'submenu-
title-noDropdown':!isNest}">
          <item :icon="onlyOneChild.meta.icon||(item.meta&&item.meta.icon)"
:title="onlyOneChild.meta.title" />
        </el-menu-item>
      </router-link>
    </template>

    <el-submenu v-else ref="subMenu" :index="resolvePath(item.path)" popper-
append-to-body>
      <template v-slot:title>
        <item v-if="item.meta" :icon="item.meta && item.meta.icon"
:title="item.meta.title" />
      </template>
      <sidebar-item
        v-for="child in item.children"
        :key="child.path"
        :is-nest="true"
        :item="child"
        :base-path="resolvePath(child.path)"
        class="nest-menu"
      />
    </el-submenu>
  </div>
</template>

<script>
import path from 'path'
```

```
import Item from './Item'

export default {
  name: 'SidebarItem',
  components: { Item },
  props: {
    // route object
    item: {
      type: Object,
      required: true
    },
    isNest: {
      type: Boolean,
      default: false
    },
    basePath: {
      type: String,
      default: ''
    }
  },
  data() {
    this.onlyOneChild = null
    return {}
  },
  methods: {
    hasOneShowingChild(children = [], parent) {
      const showingChildren = children.filter(item => {
        if (item.hidden) {
          return false
        } else {
          // 如果只有一个子菜单时设置
          this.onlyOneChild = item
          return true
        }
      })

      // 当只有一个子路由，该子路由默认显示
      if (showingChildren.length === 1) {
        return true
      }

      // 没有子路由则显示父路由
      if (showingChildren.length === 0) {
        this.onlyOneChild = { ... parent, path: '', noShowingChildren: true }
        return true
      }

      return false
    },
    resolvePath(routePath) {
      return path.resolve(this.basePath, routePath)
    }
  }
}
</script>
```

创建侧边栏菜单项组件，layout/components/Sidebar/Item.vue

```
<script>
export default {
  name: 'MenuItem',
  functional: true,
  props: {
    icon: {
      type: String,
      default: ''
    },
    title: {
      type: String,
      default: ''
    }
  },
  render(h, context) {
    const { icon, title } = context.props
    const vnodes = []

    if (icon) {
      vnodes.push(<svg-icon icon-class={icon}/>)
    }

    if (title) {
      vnodes.push(<span slot='title'>{(title)}</span>)
    }
    return vnodes
  }
}
</script>
```

测试：layout/index.vue中添加sidebar

```
<script>
import Sidebar from '@/components/Sidebar'

export default {
  components: {
    Sidebar
  },
}
</script>
```

> 可添加子菜单测试一下

面包屑

面包屑导航是通过 `$route.matched` 数组动态生成的。

low版实现，创建./components/Bread.vue

```
<template>
  <div>
    <span v-for="(item, idx) in items" :key="item.path">
      <!-- 最后一项时只显示文本 -->
      <span v-if="idx === items.length - 1">{{item.meta.title}}</span>
      <!-- 否则显示超链接 -->
      <span v-else>
        <router-link :to="item">{{item.meta.title}}</router-link>
        <span>/</span>
      </span>
    </span>
  </div>
</template>

<script>
export default {
  computed: {
    items() {
      console.log(this.$route.matched);

      // 根据matched数组获取面包屑数组
      // 要求必须有title且breadcrumb不为false
      return this.$route.matched.filter(
        item => item.meta && item.meta.title && item.meta.breadcrumb !== false
      );
    }
  }
};
</script>
```

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

使用面包屑，layout/index.vue

数据交互

数据交互流程：

api service => request => local mock/esay-mock/server api

> 主要问题分析：
>
> 1. 有时需要对请求头、响应进行统一预处理
> 2. 请求不同数据源时url会变化，需要能根据环境自动修改url
> 3. 可能出现的跨域问题

封装request

解决前两个问题需要统一封装请求代码。

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";

// 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
    const token = localStorage.getItem('token')
    if (token) {
      // 设置令牌请求头
      config.headers["Authorization"] = 'Bearer ' + token;
    }
    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

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'
  })
}
```

> 下面创建mock数据

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 auth = req.headers["authorization"];
        const roles = auth.split(' ')[1] === "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);
      localStorage.setItem("token", 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
2. 创建一个项目
3. 创建需要的接口

```
// user/login
{
  "code": function({
    _req
  }) {
    const {
      username
    } = _req.body;
    if (username === "admin" || username === "jerry") {
      return 1
    } else {
      return 10008
    }
  },
  "data": function({
    _req
  }) {
    const {
      username
    } = _req.body;
    if (username === "admin" || username === "jerry") {
      return username
    } else {
      return ''
    }
  }
}

// user/info
{
```

```
  code: 1,
  "data": function({
    _req
  }) {
    return _req.headers['authorization'].split(' ')[1] === 'admin' ?
['admin'] : ['editor']
  }
}
```

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['authorization'].split(' ')[1] ? ["admin"] :
["editor"];
  res.json({
    code: 1,
    data: roles
  });
});

app.listen(3000);
```

~测试

项目测试

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**。

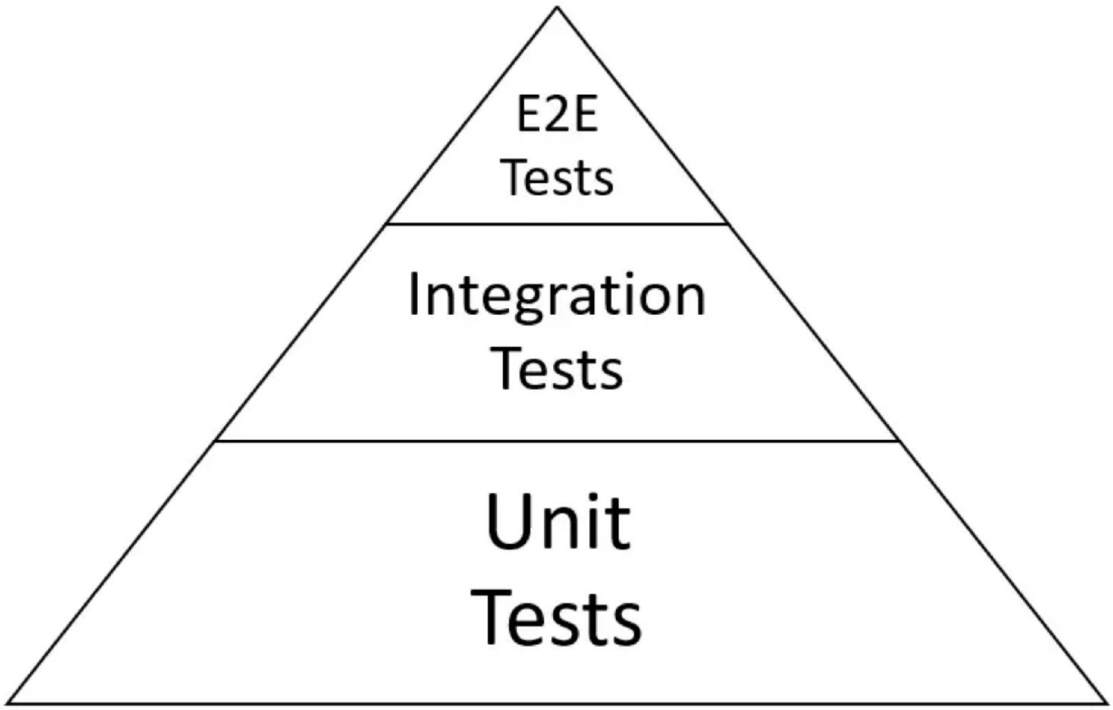
还有一种测试叫做**白盒测试**，我们针对一些内部核心实现逻辑编写测试代码，称之为**单元测试**。

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。

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`

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
? Please pick a preset: Manually select features
? Check the features needed for your project:
 (*) Babel
 ( ) TypeScript
 ( ) Progressive Web App (PWA) Support
 ( ) Router
 ( ) Vuex
 ( ) CSS Pre-processors
 (*) Linter / Formatter
 (*) Unit Testing
>(*) E2E Testing
```

- 单元测试解决方案选择：`Jest`

```
? Please pick a preset: Manually select features
? Check the features needed for your project: Babel, Linter, Unit, E2E
? Pick a linter / formatter config: Basic
? Pick additional lint features: (Press <space> to select, <a> to toggle all,
ection)Lint on save
? Pick a unit testing solution:
  Mocha + Chai
> Jest
```

- 端到端测试解决方案选择：Cypress

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL                                1: node
? Please pick a preset: Manually select features
? Check the features needed for your project: Babel, Linter, Unit, E2E
? Pick a linter / formatter config: Basic
? Pick additional lint features: (Press <space> to select, <a> to toggle all
ection)Lint on save
? Pick a unit testing solution: Jest
? Pick a E2E testing solution: (Use arrow keys)
> Cypress (Chrome only)
  Nightwatch (Selenium-based)
```

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

vue官方提供了用于单元测试的实用工具库`@vue/test-utils`

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，test/unit/kaikeba.spec.js

```
import Kaikeba from '@/components/Kaikeba.vue'

describe('Kaikeba.vue', () => {
  // 检查组件选项
  it('要求设置created生命周期', () => {
    expect(typeof Kaikeba.created).toBe('function')
  })
  it('message初始值是vue-test', () => {
    // 检查data函数存在性
    expect(typeof Kaikeba.data).toBe('function')
    // 检查data返回的默认值
    const defaultData = Kaikeba.data()
    expect(defaultData.message).toBe('vue-test')
  })
})
```

```
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之后预期结果

使用@vue/test-utils挂载组件

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

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

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");
});
```

> 解决sample.spec.js中的错误
>
> ```
> const wrapper = shallowMount(HelloWorld, {
> propsData: { msg },
> provide: {foo:{$options:{name:'aaa'}}}
> })
> ```
>
> 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/**/*.{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

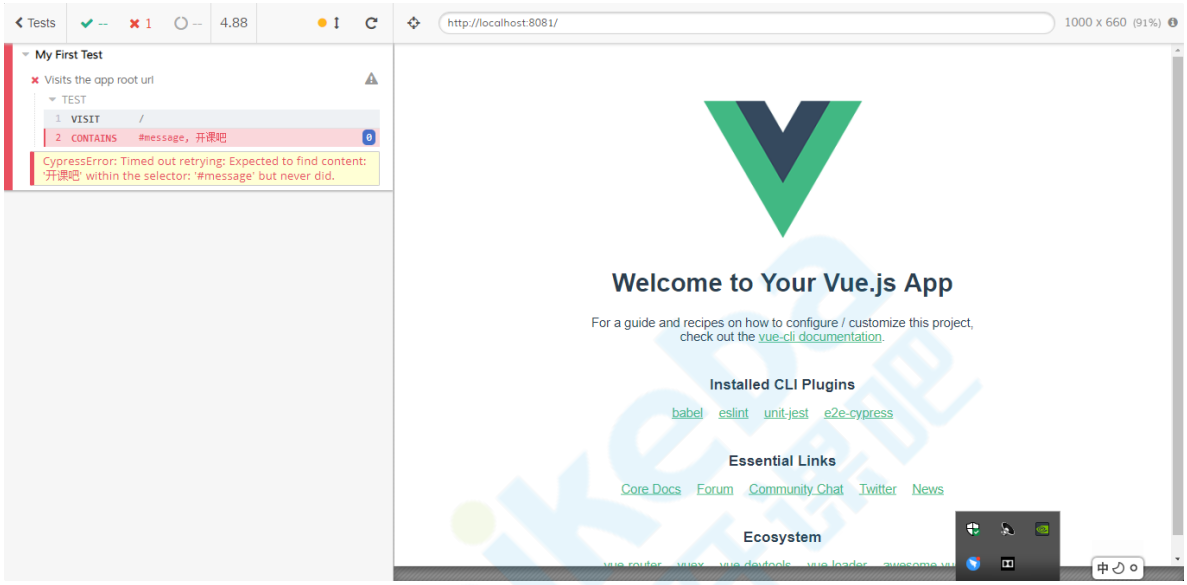
`npm run test:e2e`

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')

  })
})
```

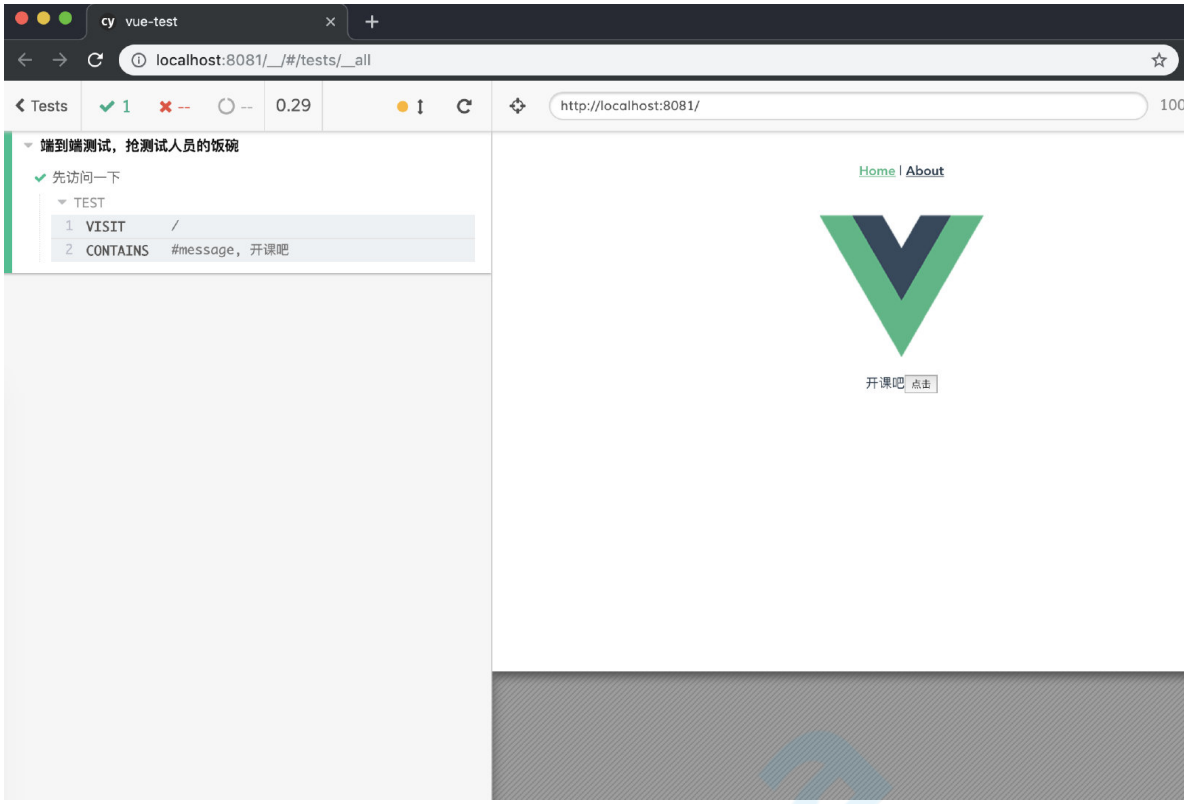


测试未通过，因为没有使用Kaikeba.vue，修改App.vue

```
<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')

  })
})
```