Vue.js项目架构最佳实践

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。

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,
}
```

修改SideMenu/index.vue

```
<template>
  <div>
    <ul>
      <!-- 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";
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 -->
      <Icon v-if="model.meta && model.meta.icon" :icon-
class="model.meta.icon"></Icon>
      <!-- 3.设置title：如果是folder仅显示标题和展开状态 -->
      <span v-if="isFolder">
        <span v-if="model.meta && model.meta.title">{{model.meta.title}}
</span>
        [{{open ? '-' : '+'}}]
      </span>
      <!-- 4.如果是叶子节点，显示为链接 -->
      <template v-else>
        <router-link
          v-else-if="model.meta && model.meta.title"
          :to="resolvePath(model.path)"
        >{{model.meta.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>
import path from 'path'
export default {
  name: "Item",
  props: {
    // 新增basePath保存父路由path
    basePath: {
      type: String,
      default: ''
    }
  },
  methods: {
  // 拼接子路由完整path
    resolvePath(routePath) {
      return path.resolve(this.basePath, routePath)
    }
  }
};
</script>
```

利用element-ui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>
```

创建菜单标签组件，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(<Icon icon-class={icon}/>)
    }

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

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["Authorization"] = 'Bearer ' + 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'
```

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'
  })
}
```

要测试需要接口

数据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);
      setToken(res.data);
    });
  },

  // get user info
  getInfo({ commit, state }) {
    return getInfo(state.token).then(({data: roles}) => {
      commit("SET_ROLES", roles);
      return {roles}
    })
  },
};
```

线上mock：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['authorization'].split(' ')[1] ? ["admin"] :
["editor"];
  res.json({
```

```
    code: 1,
    data: roles
  });
});

app.listen(3000);
```

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**。比如根据测试用例在页面上测试业务逻辑的正确性，这种测试称之为**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

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

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

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`

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

> 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!')
  })
})
```

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，用测试的角度去写代码，vue提供了专门针对测试的`@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，修改jest.config.js:

```
module.exports = {
  "collectCoverage": true,
  "collectCoverageFrom": ["src/**/*.{js,vue}"]
}
```

在此执行npm run test:unit

> %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>
```

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~