Table of Contents

基础 1.1

- 版权 1.1.1
- 介绍 1.1.2
- 安装 1.1.3
- 业务实战 1.1.4
- 构建 & 部署 1.1.5

进阶 1.2

- 布局 & 路由 1.2.1
- 样式 & 主题 1.2.2
- 图标 1.2.3
- 国际化 1.2.4

版权说明

> 本文档为付费文档，版权归人人开源（renren.io）所有，并保留一切权利，本文档及其描述的内容受有关法律的版权保护，对本文档以任何形式的非法复制、泄露或散布到网络提供下载，都将导致相应的法律责任。

免责声明

> 本文档仅提供阶段性信息，所含内容可根据项目的实际情况随时更新，以人人开源社区公告为准。如因文档使用不当造成的直接或间接损失，人人开源不承担任何责任。

文档更新

> 本文档由人人开源于2018年9月30日最后修订。

介绍

Renren-cloud-admin人人微服务平台（前端）基于Vue-cli3.0+脚手架搭建而成。它使用了Vue2.0+、Element2.0+等前端技术栈开发，提供一套更优的前端管理后台解决方案。

前序

考虑到项目的用户群体以后台开发人员居多。以下几个建议希望能帮助到您快速开发和定位问题。

- `编辑器` 使用VSCode，读写文件快，插件安装简单并且丰富，满足前端开发需求。
- `浏览器` 使用Chrome，它自带的“开发者工具”是前端开发必不可少的利器。在开发时一定要经常留意Console控制台是否有报错信息，就如同您在JAVA开发时关注IDEA / Eclipse编辑器Console控制台一样。
- 官方的 `API文档` 和 `教程` 永远是最棒的。比如Vue官方教程就是Vue最棒的 `入门学习教程` ，没有之一。

技术栈

提前了解和学习这些知识会对使用本项目有很大的帮助。

- Node.js
- ES6
- Webpack
- Vue-cli
- Vue
- Vue-router
- Vuex
- Vue-i18n
- Axios
- Element
- JS-cookie

兼容性

项目兼容Chrome、IE10+、Firefox、Safari浏览器版本。

目录结构

```
├── dist                        // 构建生成的部署文件
├── node_modules                // node插件
├── public                      // 静态资源（如：favicon.ico、第三方插件等，无需再次处理的文件）
├── src
│   ├── assets                  // 静态资源（如：scss、img等，需脚手架编译、压缩等再次处理的文
```

```
件)
│  ├── components            // 公共组件
│  ├── element-ui             // element配置
│  ├── i18n                   // 国际化
│  ├── icons                  // 图标
│  ├── mixins                 // 混入
│  ├── router                 // 路由
│  ├── store                  // 状态管理
│  ├── utils                  // 工具类
│  ├── views                  // 业务相关
│  ├── App.vue
│  ├── main.js                // 入口
├── ...
├── package-lock.json
├── package.json
└── vue.config.js             // vue-cli脚手架配置
```

安装

您需要提前在本地安装Node.js，使用Git克隆项目或者直接下载项目后，再通过 `终端命令行` 执行以下命令。

```
# 切换到项目根目录

# 安装插件
npm install

# 启动项目
npm run serve
```

> 如网络不稳定，安装时出错或进度过慢！请移步cnpm淘宝镜像进行安装。

启动完成后，会自动打开浏览器访问http://localhost:8001，如您看到下面的页面代表 `前端项目` 运行成功！因为前后端分离项目，需保证 `前端项目` 和 `后台项目` 分别独立正常运行。

请留意下面的页面，其中 `验证码` 未能正常显示，控制台有 `API请求` 报错信息！这时需检查 `后台项目` 是否正常运行。

常见问题

如何修改API请求地址?

- 修改 `/src/pubilc/index.html` 文件中 `<!-- 开发环境 -->` 注释下的 `window.SITE_CONFIG['apiURL']` 变量值。

```
<!-- 开发环境 -->
<% if (process.env.VUE_APP_NODE_ENV === 'dev') { %>
  <script>window.SITE_CONFIG['apiURL'] = 'http://localhost:8080';</script>
<% } %>
```

业务实战

一切准备就绪，我们进入业务实战。

需求

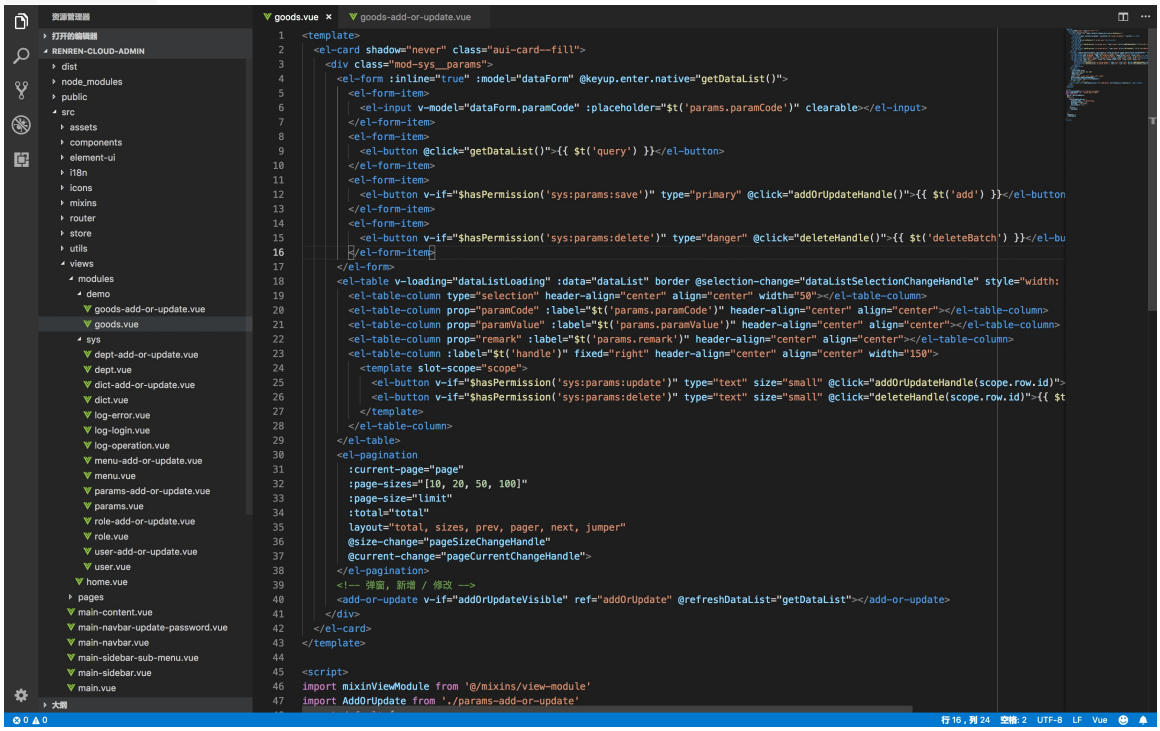
实现一个商品管理的列表、删除、增加、修改、导出功能。这里假设商品管理API接口已经提供，并且对象拥有以下属性：

```
Goods = {
  id,     // Int类型，自增ID
  name,   // String类型，名称
  type,   // Int类型，类型（0：服装，1：零食，2：电子产品）
  price   // Double类型，价格
}
```

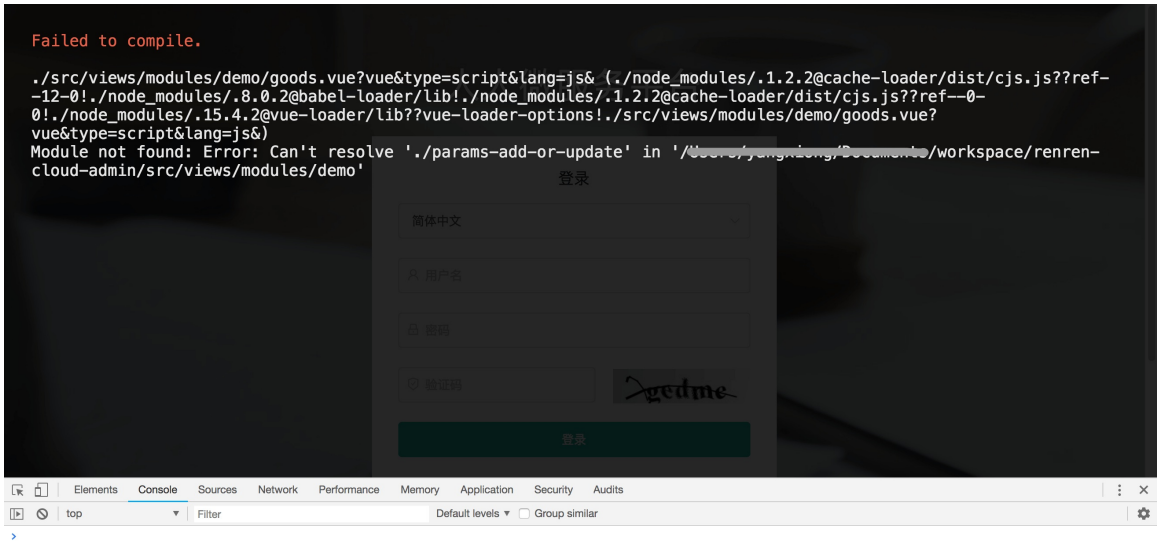
准备工作

在真实的业务开发中，建议先拷贝一份 `较简易` 的现有业务做基础，然后再进行具体的业务逻辑代码修改。这里使用 `参数管理` 模块做基础。

1. 在项目 `/src/views/modules` 目录中创建 `demo` 文件夹。
2. 将 `/src/views/modules/sys` 目录中的 `参数管理` 模块 `params.vue` 和 `params-add-or-update.vue` 两个文件拷贝到 `demo` 目录中，并重命名为 `goods.vue` 和 `goods-add-or-update.vue` 。



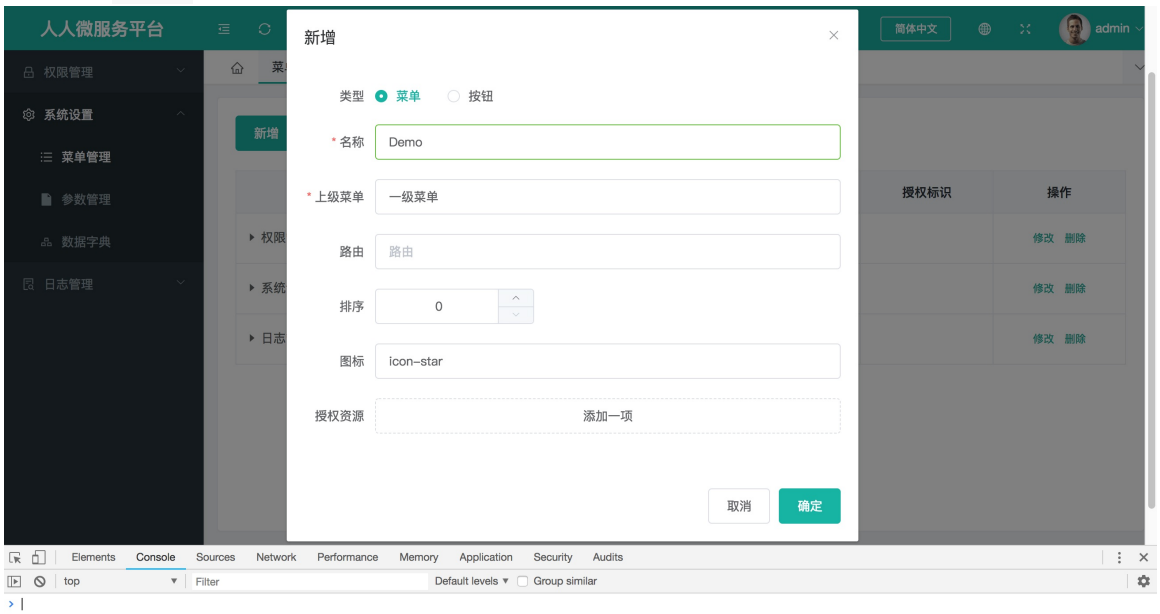
3. 这时浏览器会有报错信息。解决办法只需将 `goods.vue` 文件中的 `import AddOrUpdate from './params-add-or-update'` 修改为 `import AddOrUpdate from './goods-add-or-update'` 即可。



项目中，业务文件务必放在 `/src/views/modules` 目录中，并做好归类。比如，现在的Demo演示就归类到 `demo` 中， 最终目录 为 `/src/views/modules/demo/goods.vue` 。

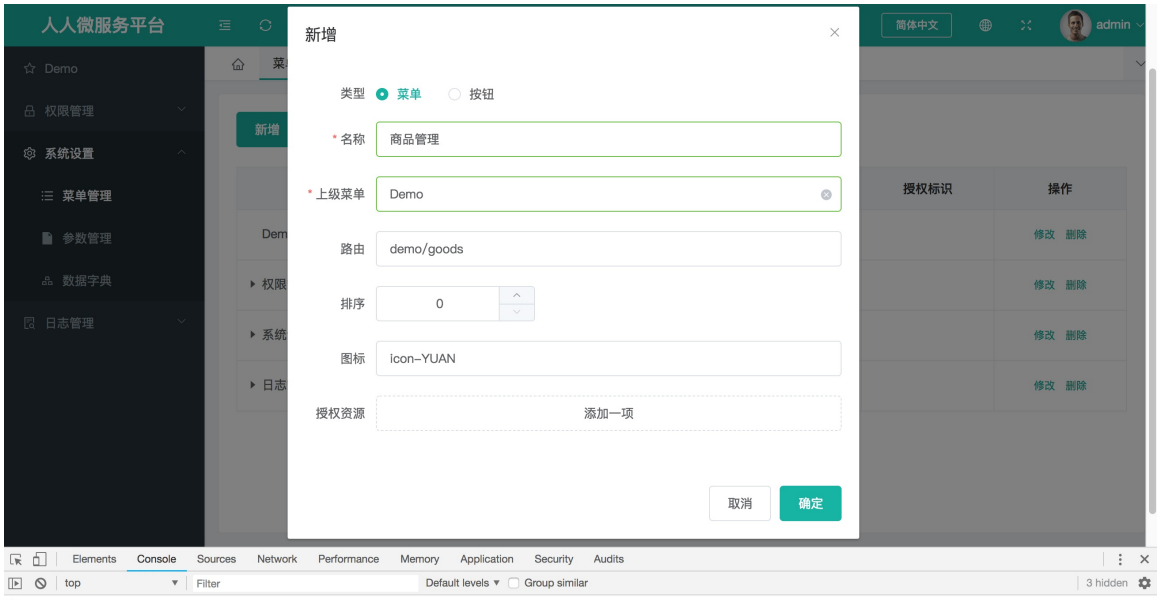
菜单配置

1. 登录项目，打开 `菜单管理` 菜单，新增一个 `Demo` 菜单。新增成功后，即可在 `菜单管理` 列表中看到新增的 `Demo` 菜单。

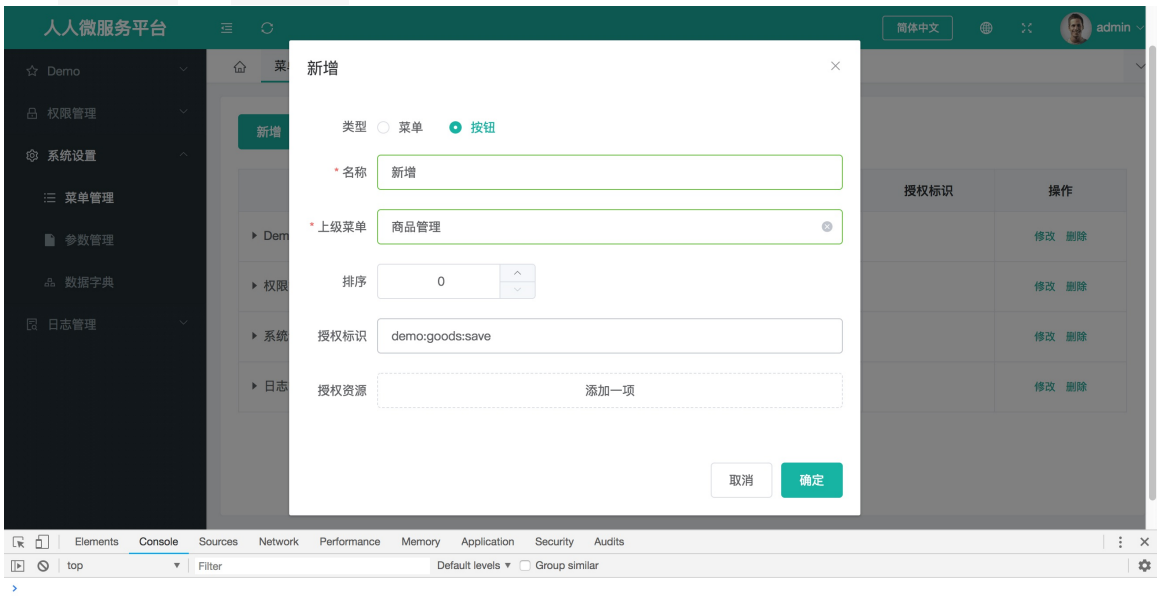


> 需要 `刷新` 浏览器，才能看到 `左侧边栏` 新增的菜单。

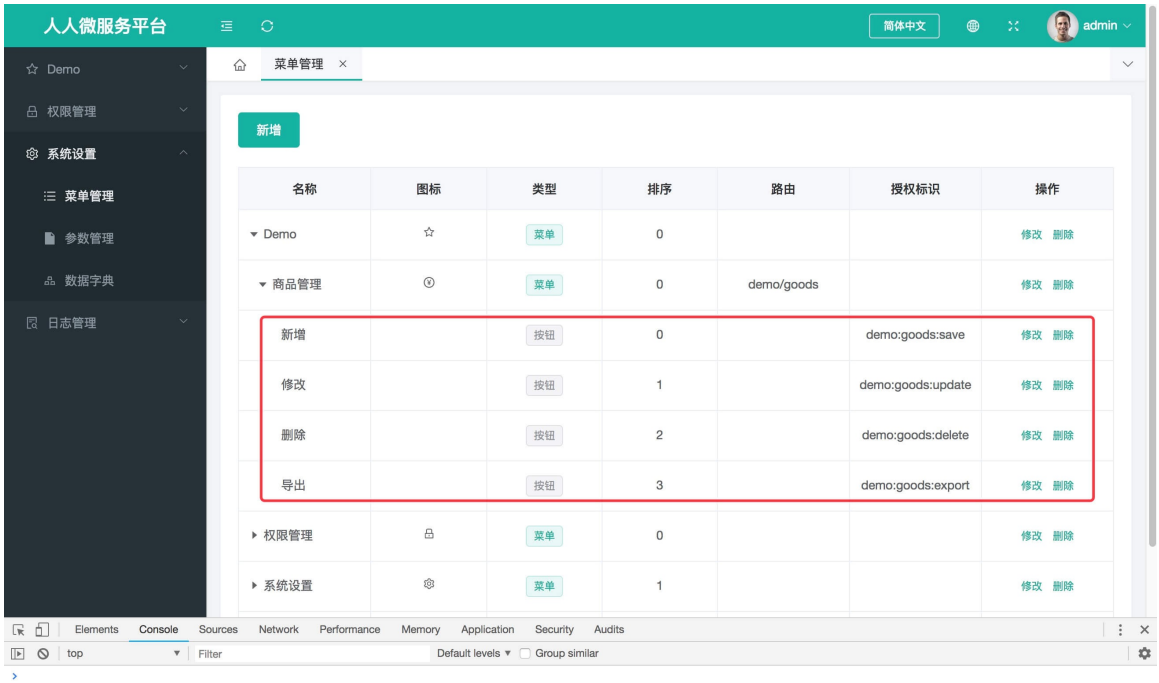
2. 再新增一个 `商品管理` 菜单。
   - 新增菜单的 `上级菜单` 为 `Demo` 菜单。
   - 新增菜单的 `路由` 为之前 `准备工作` 中创建的 `demo` 文件夹加上业务文件名 `demo/goods` 。



3. 新增 商品管理 菜单 按钮权限 。



一共新增4个按钮权限。



新增成功后，`刷新`浏览器。即可正常打开浏览`商品管理`菜单，但展示的内容还是`参数管理`模块的信息。

接下来，将进入功能开发。您会在代码中看到很多`$t('xxx.xx')`类似方法，请不要疑惑！这是在处理国际化。通过Vue-i18n插件实现，具体代码在`/src/i18n/index.js`中。

- 如您没有国际化需求，在开发中使用`简体中文`编写即可，因为国际化默认为`简体中文`。
- 对于那些已经完成的国际化，您`只需`将页面中`语言切换功能`删除即可，现有的国际化代码不会有其他负面影响。

列表 & 删除 & 导出

常见业务列表页都是由`搜索栏`和`数据列表`组成。其中：

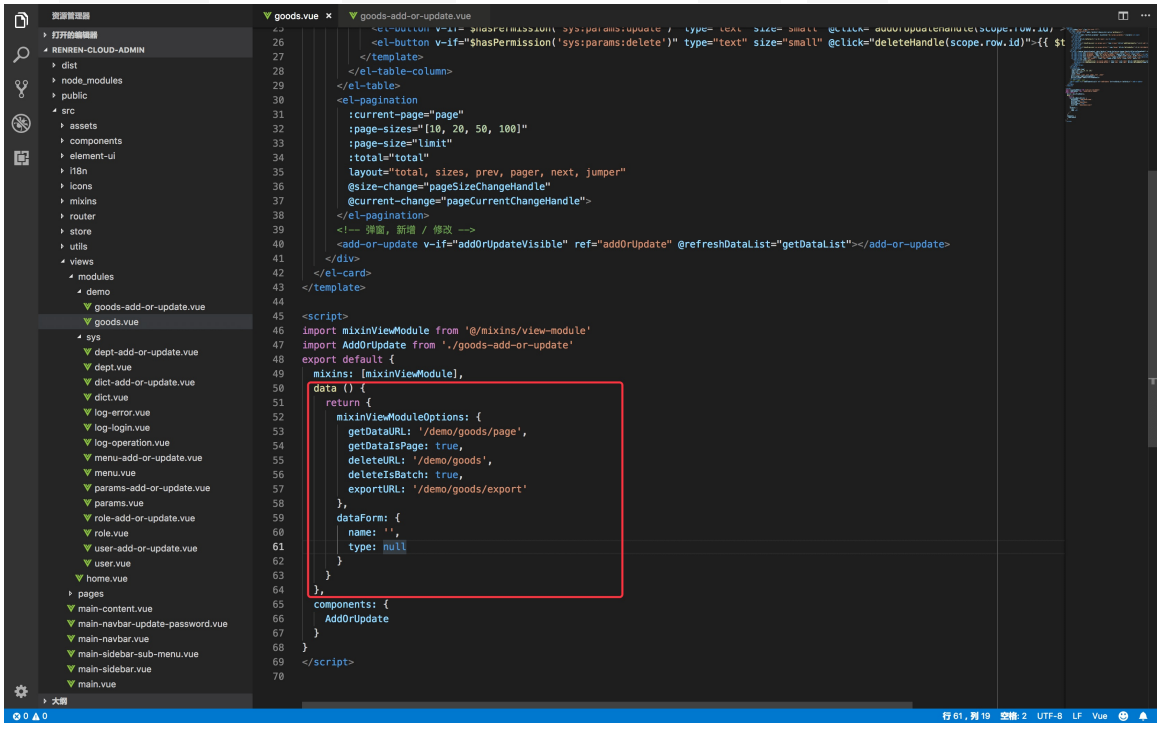
- 搜索栏包含`搜索条件`、`新增`、`批量xx`、`导出`等对`数据列表`全局操作功能项。
- 数据列表包含`分页`和每条数据末尾的`操作`项，用于对当前这条数据进行`修改`、`删除`等操作。

这里将`列表`、`删除`、`导出`功能放在一起实现，是因为它们都属于直接在`列表页`中进行操作的功能，同时项目中已将`列表页`中`常用的业务功能`封装成一个`简易`的`业务可复用类`。具体代码在`/src/mixins/view-module.js`中，其中可配置属性如下：

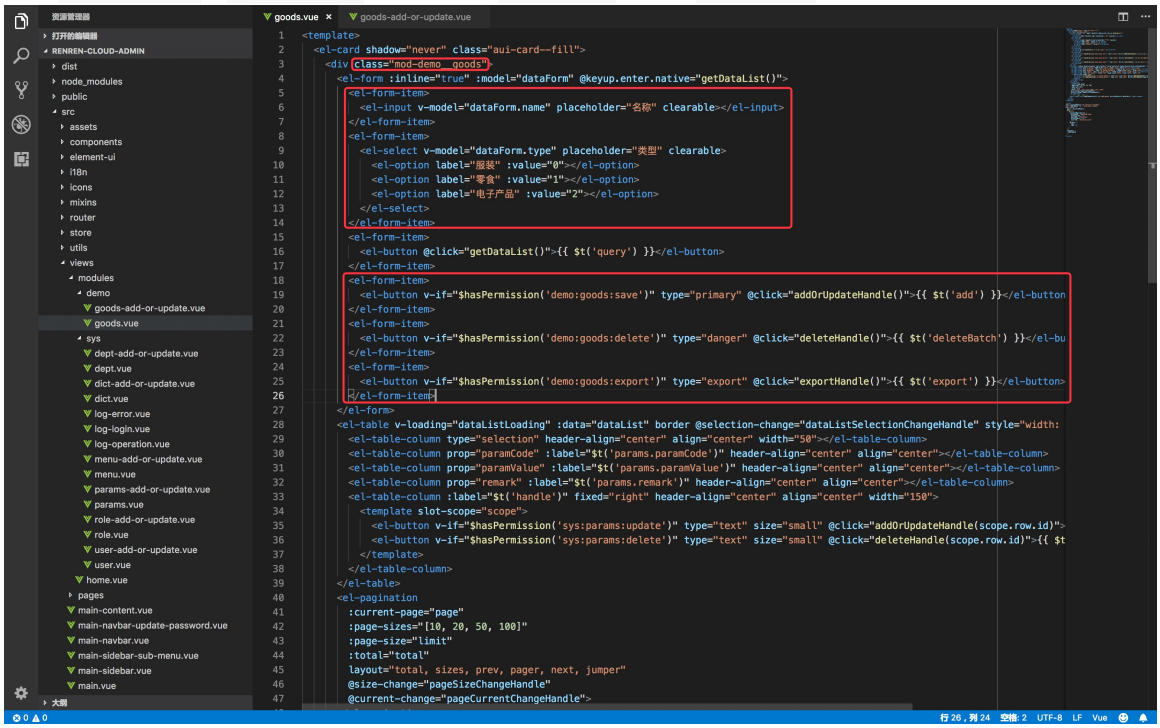
```
data () {
  /* eslint-disable */
  return {
    // 设置属性
```

```
    mixinViewModuleOptions: {
      activatedIsNeed: true,     // 此页面是否在激活（进入）时，调用查询数据列表接口?
      getDataListURL: '',        // 数据列表接口，API地址
      getDataListIsPage: false, // 数据列表接口，是否需要分页?
      deleteURL: '',             // 删除接口，API地址
      deleteIsBatch: false,      // 删除接口，是否需要批量?
      deleteIsBatchKey: 'id',    // 删除接口，批量状态下由那个key进行标记操作? 比如：pid，ui
d...
      exportURL: ''              // 导出接口，API地址
    },
    // 默认属性
    dataForm: {},                // 查询条件
    dataList: [],                // 数据列表
    order: '',                   // 排序，asc / desc
    orderField: '',              // 排序，字段
    page: 1,                     // 当前页码
    limit: 10,                   // 每页数
    total: 0,                    // 总条数
    dataListLoading: false,      // 数据列表，loading状态
    dataListSelections: [],      // 数据列表，多选项
    addOrUpdateVisible: false    // 新增 / 更新，弹窗visible状态
  }
  /* eslint-enable */
},
```

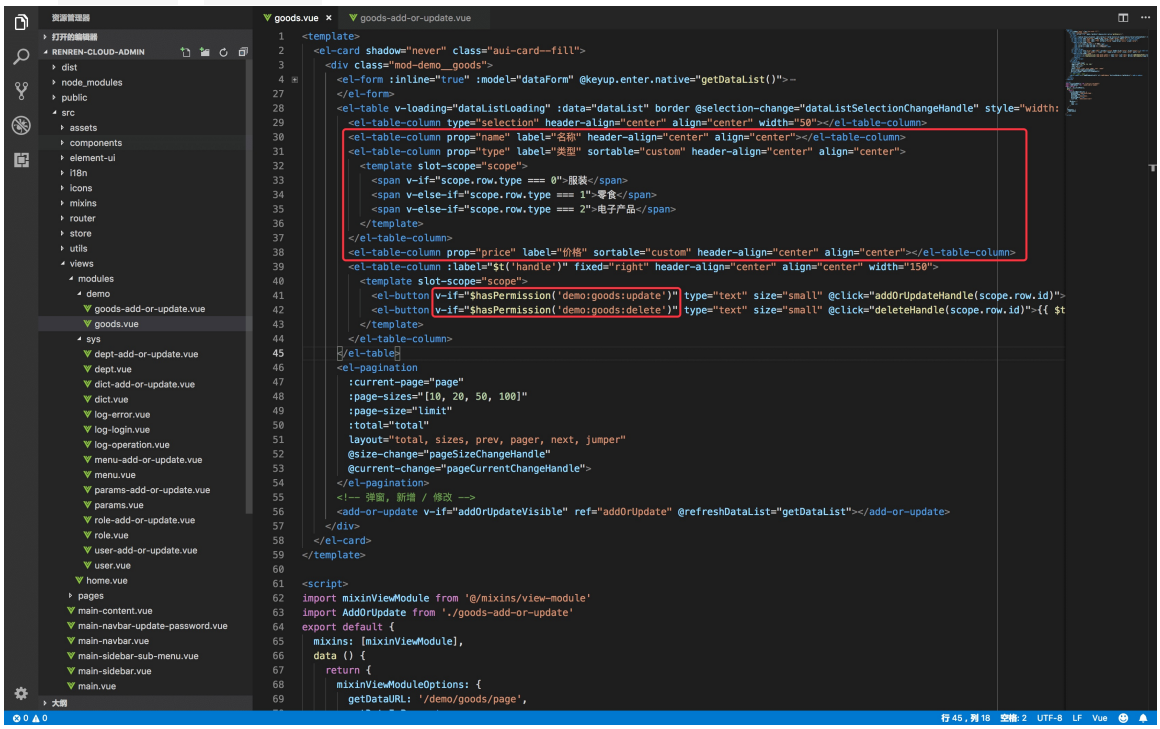
1. 打开 `goods.vue` 文件，修改 `API请求` 相关配置属性，新增 `name` 和 `type` 查询条件。



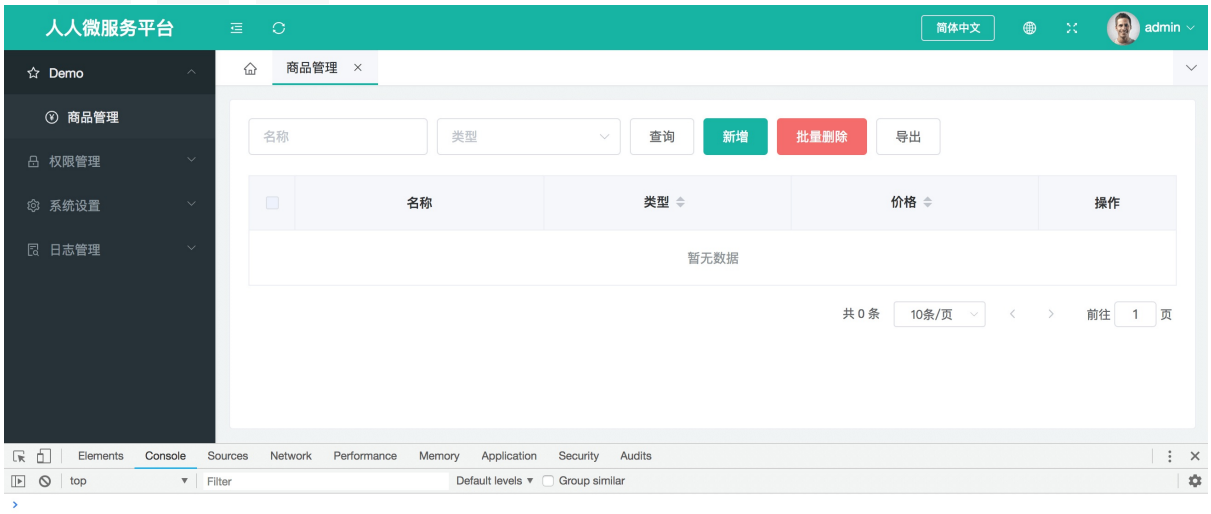
2. 修改 `class名称` 、 `dataForm 查询条件`、 `按钮权限` ， 新增 `导出按钮` 。



3. 修改 `数据列表` 和 `按钮权限` 。



至此 列表 、 删除 、 导出 功能开发完毕。

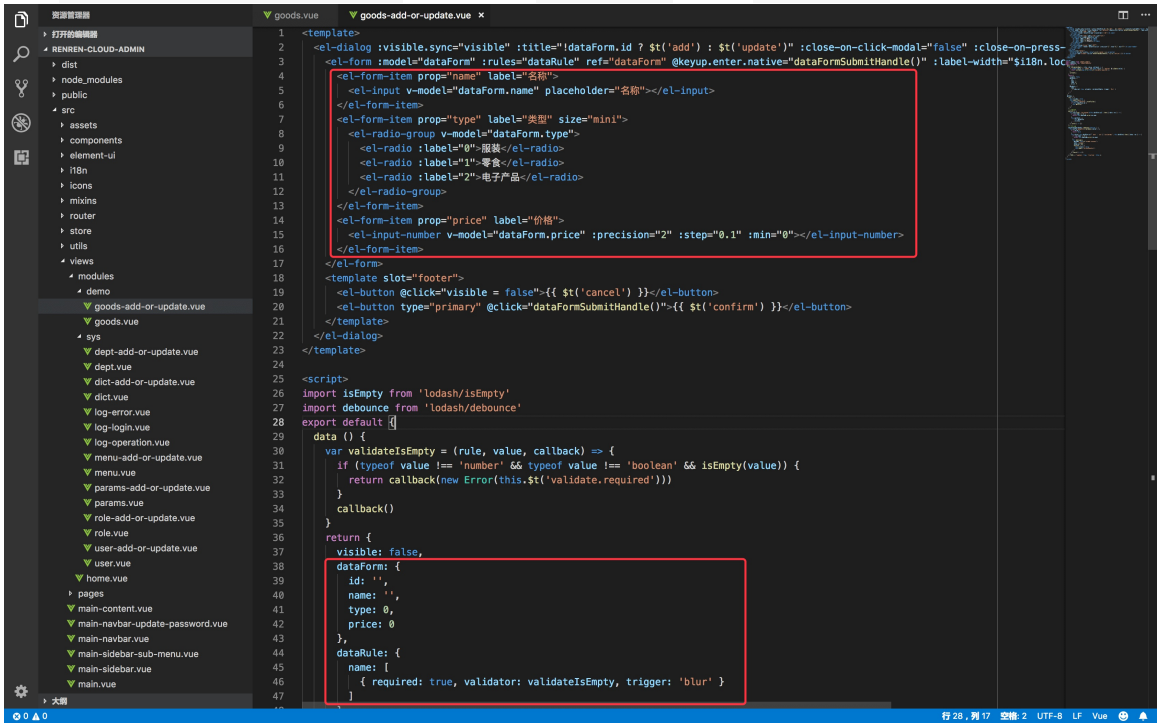


增加 & 修改

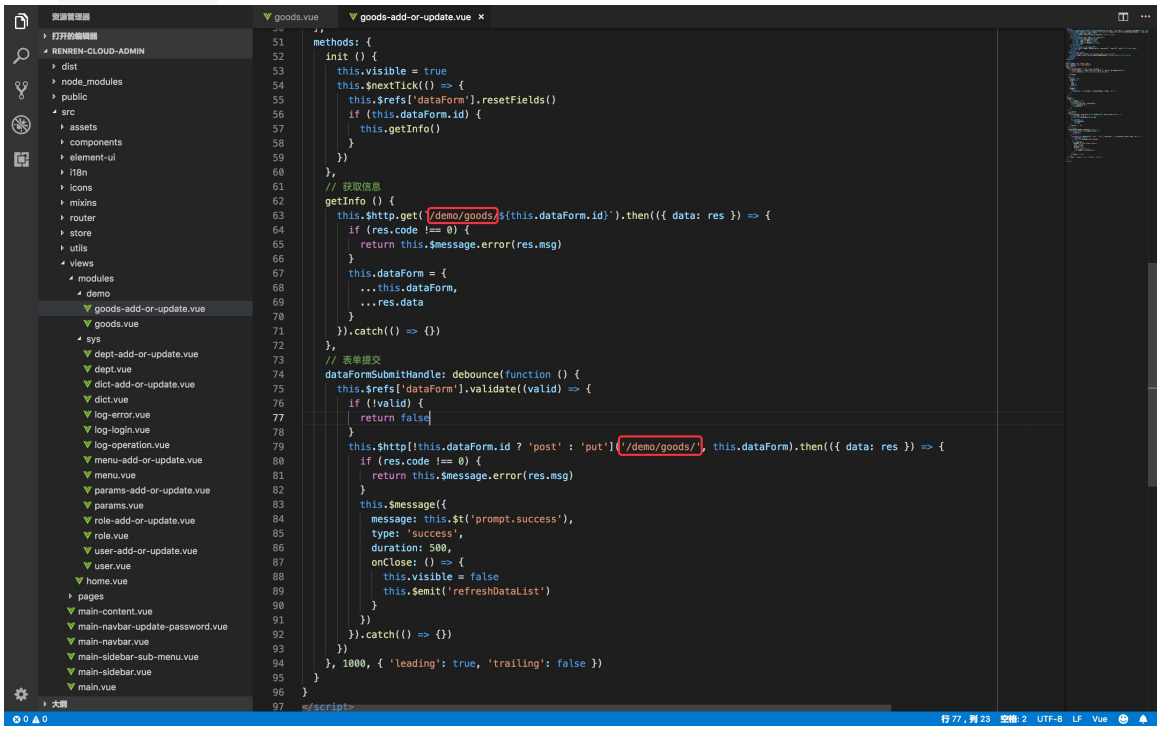
常见业务 新增 和 修改 功能几乎一样，唯一区别在于它们 自增ID 是否存在，所以业务将它们放在同一弹窗中处理。

> 考虑到它们的功能会存在一些特定业务处理，就没有封装 业务可复用类 。

1. 打开 goods-add-or-update.vue 文件，修改 dataForm 属性和 dataRule 验证信息。



2. 修改 API请求地址 。



至此商品管理的列表、删除、增加、修改、导出功能全部开发完毕。

构建 & 部署

项目预留3个构建环境，具体代码在 `/public/index.html` 文件中。

```
<!-- 集成测试环境 -->
<% if (process.env.VUE_APP_NODE_ENV === 'prod:sit') { %>
  <script>window.SITE_CONFIG['apiURL'] = 'http://localhost:8080';</script>
<% } %>
<!-- 验收测试环境 -->
<% if (process.env.VUE_APP_NODE_ENV === 'prod:uat') { %>
  <script>window.SITE_CONFIG['apiURL'] = 'http://localhost:8080';</script>
<% } %>
<!-- 生产环境 -->
<% if (process.env.VUE_APP_NODE_ENV === 'prod') { %>
  <script>window.SITE_CONFIG['apiURL'] = 'http://localhost:8080';</script>
<% } %>
```

您需要配置相应环境 `window.SITE_CONFIG['apiURL']` 变量值后，再通过 `终端命令行` 执行以下构建命令。

```
# 集成测试环境
npm run build:sit

# 验收测试环境
npm run build:uat

# 生产环境
npm run build:prod
```

构建完成后，生成的静态资源文件在 `/dist` 目录下。

部署只需将 `/dist` 目录下的静态资源文件放置服务器即可。

常见问题

项目除 `开发环境` 和 `生产环境` 外，没有 / 或者只有一个测试环境，该如何配置?

- 只需配置项目 `需要的环境` ，并执行 `相应环境` 的构建命令即可。

布局 & 路由

页面视图由布局实现简化、路由实现交互，它们是整个项目的基架。

项目中将页面分为 `独立页面` 和 `基于主入口布局页面` 2种 `页面类型` ，对应的区分出不同类型的 `文件目录` 和 `路由变量` ，用于划分业务和 `动态菜单路由` 功能的实现和管理。

- 页面类型： `独立页面` 和 `基于主入口布局页面`
- 文件目录： `/src/views/pages` 和 `/src/views/modules`
- 路由变量： `pageRoutes` 页面路由和 `moduleRoutes` 模块路由

> 使用：将同类型的页面，放入对应的文件目录中做好归类，并在对应的路由变量中定义路由。

布局

这就是上述提到的 `基于主入口布局页面` ，它主要包含 `navbar导航条` 、 `sidebar侧边栏` 、 `content内容` 3个部分，具体代码在 `/src/views/main.vue` 文件中。

项目中的页面除了 `登录页面` 、 `注册页面` 、 `400页面` 等属于 `独立页面` 类型外，其他都属于 `基于主入口布局页面` 类型。

路由

本项目属于 `SPA单页` Web应用，页面中的跳转都是通过路由Vue-router插件实现。分为 `pageRoutes` 页面路由和 `moduleRoutes` 模块路由2种类型。具体代码在 `/src/router/index.js` 文件中。

- `pageRoutes` 页面路由，它相对简单、易理解，不再重述之前介绍。

```
// 页面路由(独立页面)
export const pageRoutes = [
  {
```

```
    path: '/404',
    component: () => import('@/views/pages/404'),
    name: '404',
    meta: { title: '404未找到' },
    beforeEnter (to, from, next) {
      // 拦截处理特殊业务场景
      // 如果, 重定向路由包含__双下划线, 为临时添加路由
      if (/__.*/.test(to.redirectedFrom)) {
        return next(to.redirectedFrom.replace(/__.*/, ''))
      }
      next()
    }
  },
  { path: '/login', component: () => import('@/views/pages/login'), name: 'logi
n', meta: { title: '登录' } }
]
```

- `moduleRoutes` 模块路由，它负责 `基于主入口布局页面` 类型页面的路由。同时与 `sidebar侧边栏` 菜单、 `content内容` tabs标签页功能绑在一起，通过一些约定规则实现 `动态菜单路由` 、 `动态路由` 、 `通过或不通过Tab展示内容` 等功能。

```
// 模块路由(基于主入口布局页面)
export const moduleRoutes = {
  path: '/',
  component: () => import('@/views/main'),
  name: 'main',
  redirect: { name: 'home' },
  meta: { title: '主入口布局' },
  children: [
    { path: '/home', component: () => import('@/views/modules/home'), name: 'ho
me', meta: { title: '首页', isTab: true } }
  ]
}
```

动态菜单路由

通过 `约定规则` 定义路由名称， `自动映射` 路由目录文件。

1. 在 `菜单管理` 菜单中添加配置 `路由` 名称。
2. 在 `/src/views/modules` 目录中添加对应的 `目录文件` 即可实现功能。

具体操作可查阅业务实战。相关 `约定规则` 如下：

- 配置 `路由` 名称为 `demo/goods` 对应目录 `/src/views/modules` 目录中 `/demo/goods.vue` 文件
- 配置 `路由` 名称为 `sys/user` 对应目录 `/src/views/modules` 目录中 `/sys/user.vue` 文件
- 配置 `路由` 名称为 `sys/user-info` 对应目录 `/src/views/modules` 目录中 `/sys/user-info.vue` 文件
- 配置 `路由` 名称为 `sys/user/info` 对应目录 `/src/views/modules` 目录

中 `/sys/user/info.vue` 文件
- 配置 `路由` 名称为 `https://www.renren.io/` 不再对应目录，而是使用iframe访问 `https://www.renren.io/` 外部地址

> 注意：`菜单管理` 菜单中配置的 `路由` 名称对应目录必须为 `/src/views/modules` 目录中，否则无效！

自定义路由

通过 `动态菜单路由` 可解决大部分常用需求，但肯定会存在一些特殊场景需求。比如：

- 路由不属于菜单，如何通过Tab展示内容?
- 交互复杂 / 内容过多等原因，不想通过Dialog弹窗展示，如何添加一个Tab展示内容，并传参数?

在这之前，我们先了解下项目提供的 `window.SITE_CONFIG['contentTabDefault']` 内容标签页默认属性对象，它可快速的帮助我们将需求解决。具体代码在 `/public/index.html` 文件中，配有详细注释说明。

```
window.SITE_CONFIG['contentTabDefault'] = {
  'name': '',         // 名称，由 this.$route.name 自动赋值（默认，名称 === 路由名称 === 路由路径）
  'params': {},       // 参数，由 this.$route.params 自动赋值
  'query': {},        // 查询参数，由 this.$route.query 自动赋值
  'menuId': '',       // 菜单id（用于选中侧边栏菜单，与this.$store.state.sidebarMenuActivName进行匹配）
  'title': '',        // 标题
  'isTab': true,      // 是否通过tab展示内容?
  'iframeURL': ''     // 是否通过iframe嵌套展示内容？（以http[s]://开头，自动匹配）
};
```

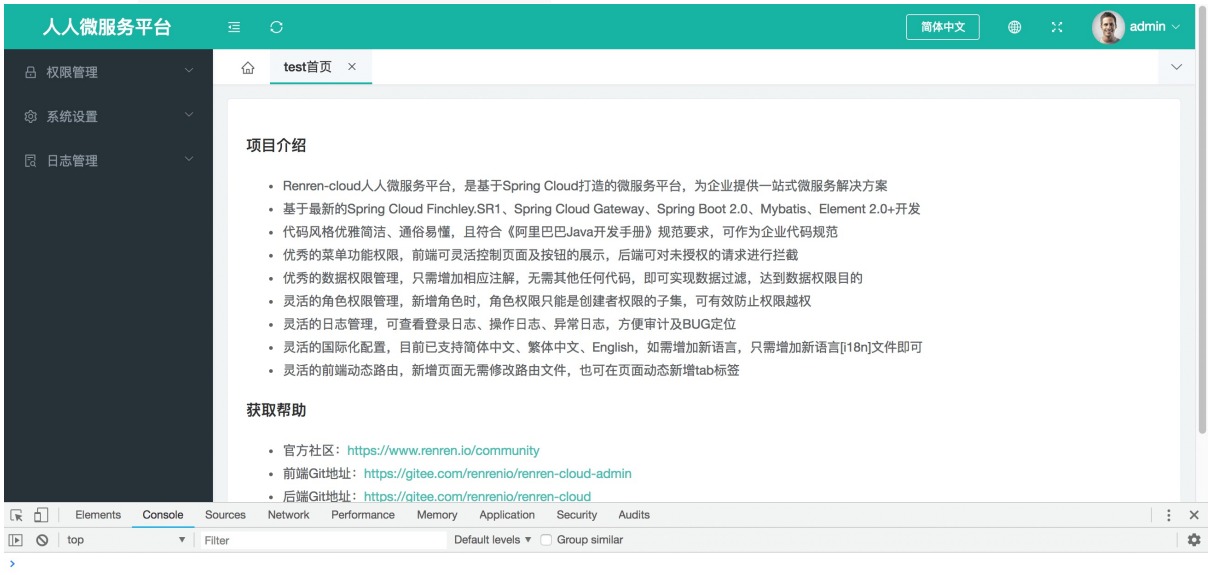
如何使用这个属性对象?

设置每一个 `模块路由` 对象时，可以通过 `meta` 对象配置 `window.SITE_CONFIG['contentTabDefault']` 对象中定义的所有属性。比如添加一个 `test路由` 代码如下：

```
// 模块路由(基于主入口布局页面)
export const moduleRoutes = {
  path: '/',
  component: () => import('@/views/main'),
  name: 'main',
  redirect: { name: 'home' },
  meta: { title: '主入口布局' },
  children: [
    { path: '/home', component: () => import('@/views/modules/home'), name: 'home',
 meta: { title: '首页', isTab: true } },
    // test 路由
    {
```

```
      path: '/test',
      component: () => import('@/views/modules/home'),
      name: 'test',
      meta: {
        title: 'test首页',
        isTab: true
      }
    }
  ]
}
```

浏览器访问 `http://localhost:8001/#/test` 效果如下：



这样，第1个特殊场景需求已解决。我们还可以设置：

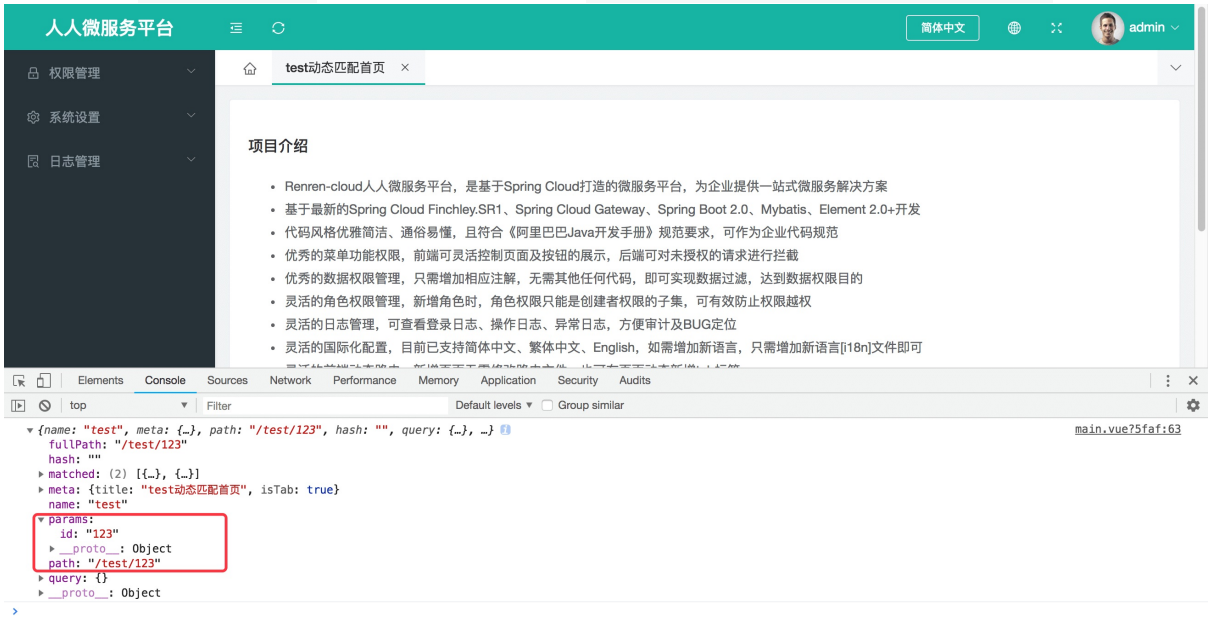
- `isTab: false` 不通过Tab展示内容。
- `component: null` 和 `iframeUrl: 'https://renren.io'` 通过iframe外链访问人人官网。

然后，再通过设置这个属性对象配合 `Vue-router插件` 动态路由匹配解决第2个特殊场景需求。代码如下：

```
// 模块路由(基于主入口布局页面)
export const moduleRoutes = {
  path: '/',
  component: () => import('@/views/main'),
  name: 'main',
  redirect: { name: 'home' },
  meta: { title: '主入口布局' },
  children: [
    { path: '/home', component: () => import('@/views/modules/home'), name: 'home',
 meta: { title: '首页', isTab: true } },
    // test 路由
    {
      path: '/test/:id',
```

```
        component: () => import('@/views/modules/home'),
        name: 'test',
        meta: {
          title: 'test动态匹配首页',
          isTab: true
        }
      }
    ]
  }
```

代码中打印 `this.$route` 路由信息，浏览器访问 `http://localhost:8001/#/test/123` 效果如下：



动态路由

不难发现 `自定义路由` 中提到的解决方案，都需要提前定义路由再处理需求。当业务变得庞大时，则需要提前定义很多的路由，这样代码冗余严重！

接下来由 `动态路由` 又称 `临时添加路由` 解决这些问题。

1. 在 `<template></template>` 中添加一个按钮并绑定 `动态路由` 事件，代码如下：

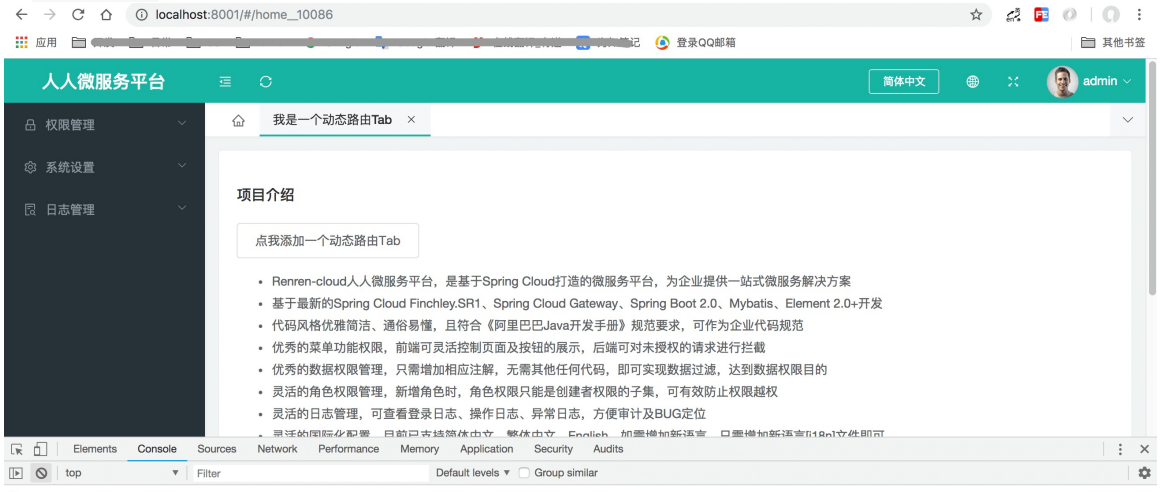
```
<el-button @click="dynamicAddRouteHandle()">点我添加一个动态路由Tab</el-button>
```

2. 在 `<script></script>` 中引入 `模块路由` 对象和 `动态路由` 实现方法，代码如下：

```
import { moduleRoutes } from '@/router'
export default {
 methods: {
   dynamicAddRouteHandle () {
     // 组装路由名称，并判断是否已添加，如是：则直接跳转
     var routeName = `home__10086`
     var route = window.SITE_CONFIG['dynamicRoutes'].filter(item => item.name =
```

```
== routeName)[0]
      if (route) {
        return this.$router.push({ name: routeName })
      }
      // 否则: 添加并全局变量保存, 再跳转
      route = {
        path: routeName,
        component: () => import('@/views/modules/home'),
        name: routeName,
        meta: {
          ...window.SITE_CONFIG['contentTabDefault'],
          menuId: this.$route.meta.menuId,
          title: '我是一个动态路由Tab'
          // 这里可根据`自定义路由`中提到的window.SITE_CONFIG['contentTabDefault']对象
属性进行配置
        }
      }
      this.$router.addRoutes([
        {
          ...moduleRoutes,
          name: `main-dynamic__${route.name}`,
          children: [route]
        }
      ])
      window.SITE_CONFIG['dynamicRoutes'].push(route)
      this.$router.push({ name: route.name })
    }
  }
}
```

3. 在 `home首页` 添加按钮测试，点击效果如下：



动态路由与 `自定义路由` 的功能完全一致。唯一需要注意代码中 `routeName` 组装路由名称变量的定义规则：

- 必须以 `真实存在的业务路由` 开头，通过 `__` 双下划线拼接当前 `动态路由` 的唯一 `Key` 组成。比如：`sys-user__${user.id}`

此规则是为了通过 `__` 双下划线进行分割路由区分 `真实业务路由` 和 `动态路由` 。再通过当前 `动态路由` 的唯一 `Key` ，保证 `动态路由` 的唯一性，防止重复添加。

假如处于 `动态路由` 状态下 `刷新页面` ，会通过拦截处理 `404路由` 将 `动态路由` 回归到 `真实业务路由` 中。比如：`http://localhost:8001/#/home__10086` 回归到 `http://localhost:8001/#/home` 。具体代码如下：

```
{
  path: '/404',
  component: () => import('@/views/pages/404'),
  name: '404',
  meta: { title: '404未找到' },
  beforeEnter (to, from, next) {
    // 拦截处理特殊业务场景
    // 如果, 重定向路由包含__双下划线, 为临时添加路由
    if (/__.*/.test(to.redirectedFrom)) {
      return next(to.redirectedFrom.replace(/__.*/, ''))
    }
    next()
  }
}
```

此功能已在 `数据字典` 菜单 `查看下级类型` 业务中实践。具体代码在 `/src/views/modules/sys/dict.vue` 文件中。

样式 & 主题

使用CSS预处理器SCSS进行样式开发，通过 `scss` 变量进行主题修改定制。

样式

样式 `.scss` 代码文件都放在 `/src/assets/scss` 目录中。结构如下：

```
scss
  ├── modules                    // 模块样式，对应`/src/views/modules`目录中文件
  ├── pages                      // 页面样式，对应`/src/views/pages`目录中文件
  ├── aui.scss                   // 入口
  ├── common.scss                // 公共
  ├── normalize.scss             // 重置，插件
  └── variables.scss             // 全局变量
```

结构中预留 `pages` 和 `modules` 目录，可将对应的业务样式文件放入其中进行归类。并在 `aui.scss` 文件中引入。代码如下：

```
// 变量
@import "~@/element-ui/theme-variables.scss";
@import "./variables.scss";
// 公共
@import "./normalize.scss";
@import "./common.scss";
// 页面
@import "./pages/login.scss";
@import "./pages/404.scss";
// 模块
@import "./modules/home.scss";
```

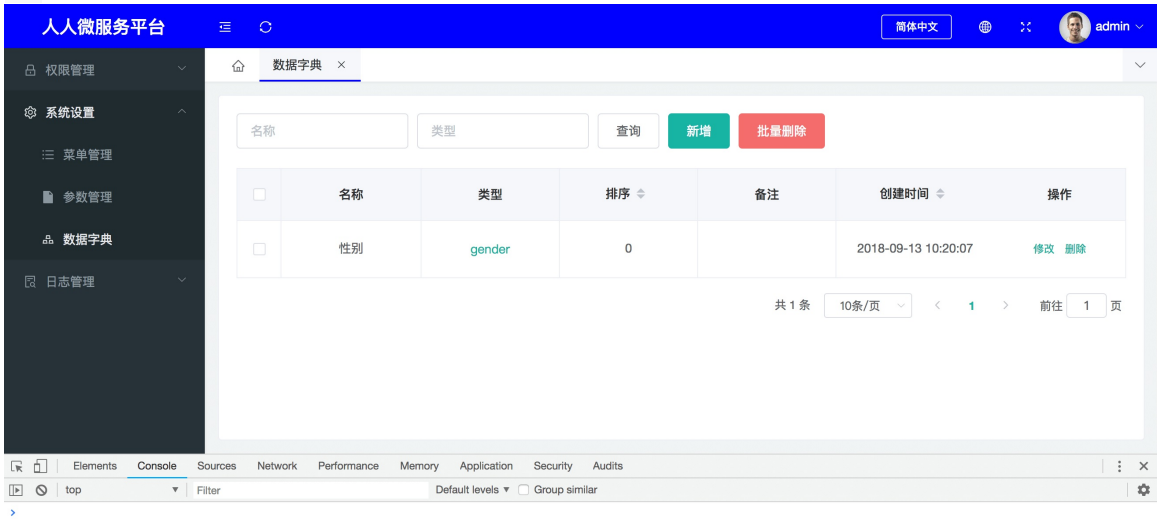
也可在 `.vue` 文件中使用 `<style lang="scss"></style>` 编写样式。代码如下：

```
<style lang="scss">
.mod-sys__dept {
  .dept-list {
    .el-input__inner,
    .el-input__suffix {
      cursor: pointer;
    }
  }
}
</style>
```

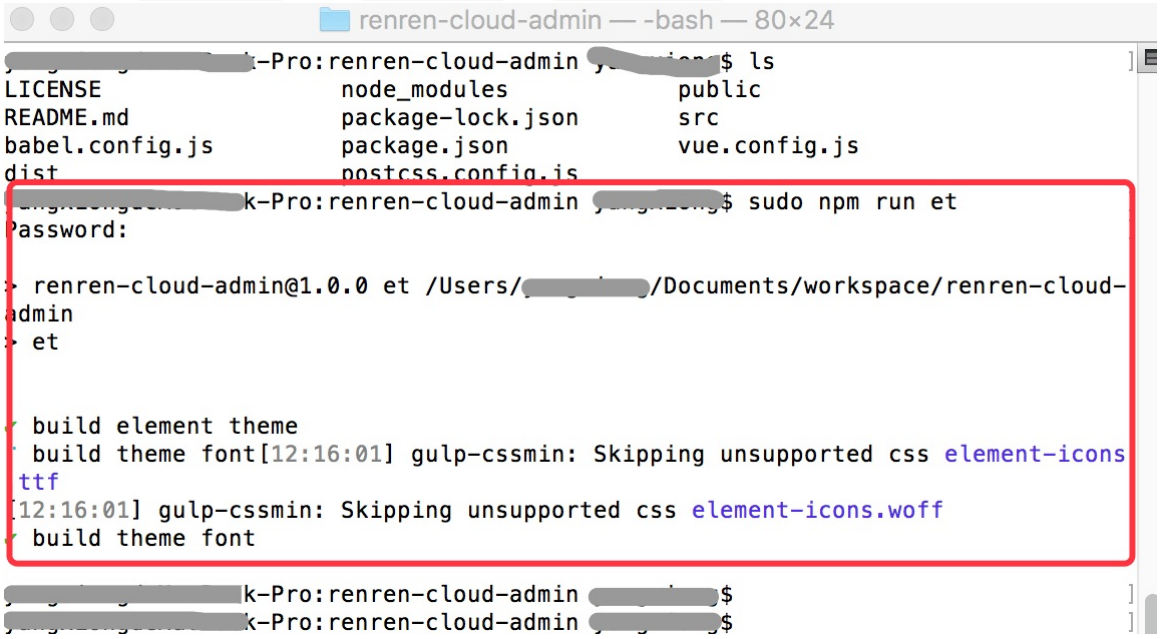
主题

项目中已经将 `Element` 组件库的自定义主题和 `主入口布局` 等 `SCSS` 变量配置好。只需修改相应变量，即可进行主题修改定制。具体操作如下：

1. 修改 `/src/element-ui/theme-variables.scss` 文件中 `$--color-primary` 变量值为 `$--color-primary: blue !default;` 保存。马上能在浏览器看到 `主入口布局` 头部背景色变成 `蓝色` 。



2. 另开启一个 `终端命令行` 切换到 `项目根目录` 执行 `npm run et` 命令。



效果如下：

人人微服务平台
简体中文
admin
权限管理
系统设置
菜单管理
参数管理
数据字典
日志管理
数据字典
名称
类型
查询
新增
批量删除
名称
类型
排序
备注
创建时间
操作
性别
gender
0
2018-09-13 10:20:07
修改 删除
共 1 条
10条/页
1
前往 1 页
Elements
Console
Sources
Network
Performance
Memory
Application
Security
Audits
top
Filter
Default levels
Group similar

图标

统一使用SVG Sprite矢量图标。项目中已引入532个icon图标，它们来自阿里矢量图标站下的一套Ant Design 官方图标库，具体代码在 `/src/icons/index.js` 文件中。

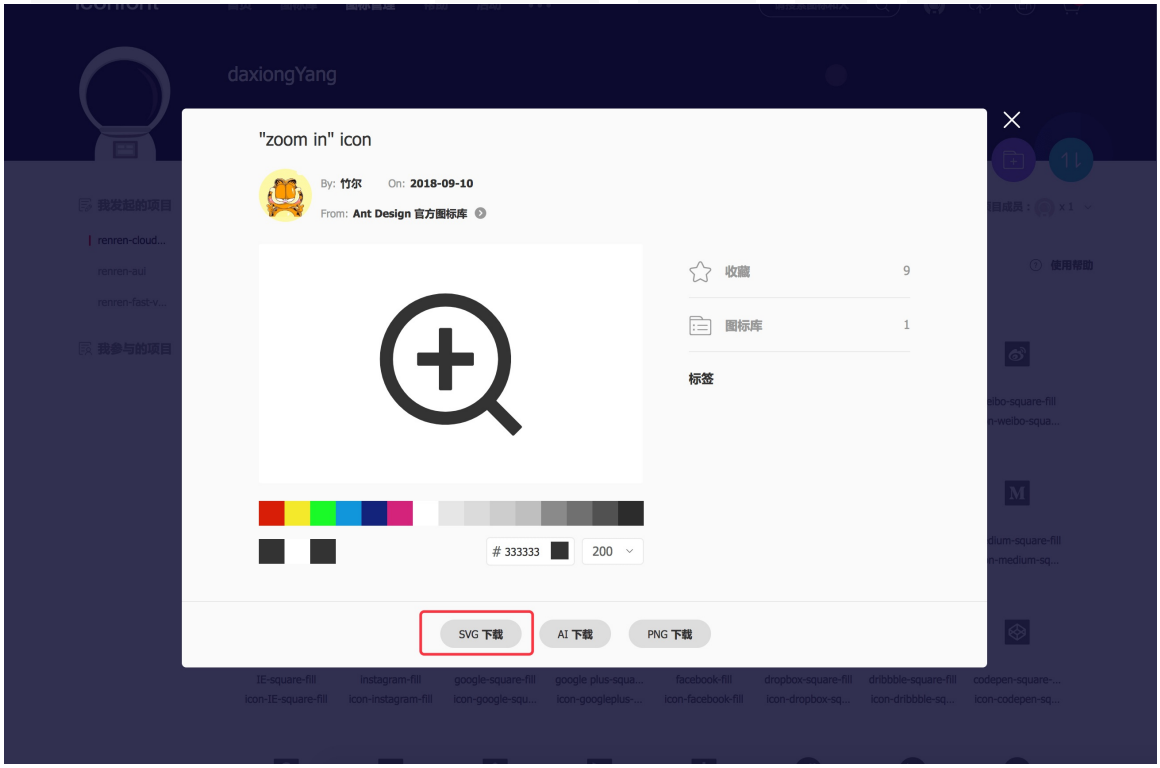
```
import './iconfont'

const svgFiles = require.context('./svg', true, /\.svg$/)
svgFiles.keys().map(item => svgFiles(item))
```

使用

通过 `<svg class="icon-svg" aria-hidden="true"><use xlink:href="#icon-zoomin"></use></svg>` 代码使用图标。其中 `icon-zoomin` 为icon的名称。

单个引入图标

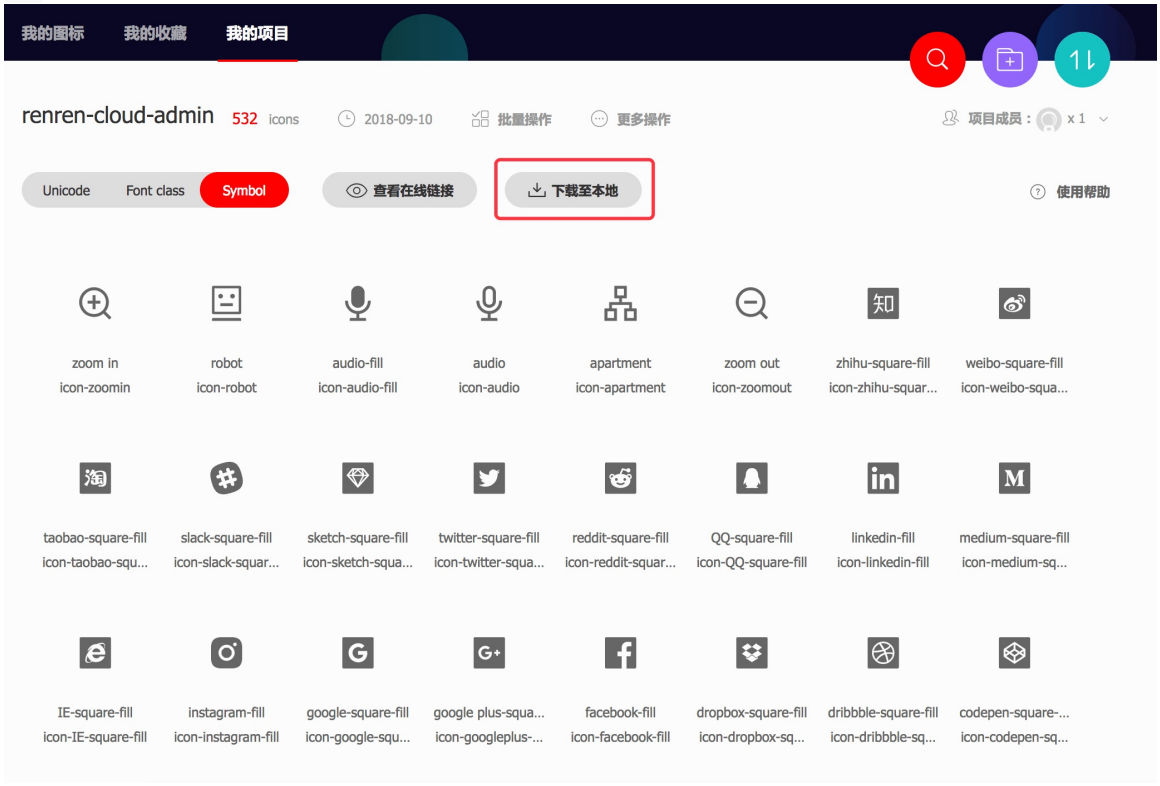
1. 在 阿里矢量图标站 搜索图标，并进行 `鼠标选中图标` -> `复制代码` -> `下载` -> `SVG下载` 操作。



2. 将下载的本地 `xxx.svg` 重命名为上一步 `复制代码` 操作 `粘贴代码.svg`，并放入 `/src/icons/svg` 目录中。

批量引入图标

1. 在 `阿里矢量图标站` 创建一个项目，搜索图标，并进行 `添加购物车` -> `添加至项目` -> `下载至本地` 操作。



2. 将下载的本地文件 `iconfont.js` 重命名为 `xxx.js`，并放入 `/src/icons/` 目录中。
3. 并在 `/src/icons/index.js` 文件中引入 `xxx.js` 。

```
import './iconfont'
import './xxx'

const svgFiles = require.context('./svg', true, /\.svg$/)
svgFiles.keys().map(item => svgFiles(item))
```

国际化

项目中提供了i18n国际化，通过Vue-i18n插件实现。具体代码在 `/src/i18n/index.js` 中。

```
import Vue from 'vue'
import VueI18n from 'vue-i18n'
import Cookies from 'js-cookie'
import zhCNLocale from 'element-ui/lib/locale/lang/zh-CN'
import zhTWLocale from 'element-ui/lib/locale/lang/zh-TW'
import enLocale from 'element-ui/lib/locale/lang/en'
import zhCN from './zh-CN'
import zhTW from './zh-TW'
import enUS from './en-US'

Vue.use(VueI18n)

export const messages = {
  'zh-CN': {
    '_lang': '简体中文',
    ...zhCN,
    ...zhCNLocale
  },
  'zh-TW': {
    '_lang': '繁體中文',
    ...zhTW,
    ...zhTWLocale
  },
  'en-US': {
    '_lang': 'English',
    ...enUS,
    ...enLocale
  }
}

export default new VueI18n({
  locale: Cookies.get('language') || 'zh-CN',
  messages
})
```

使用

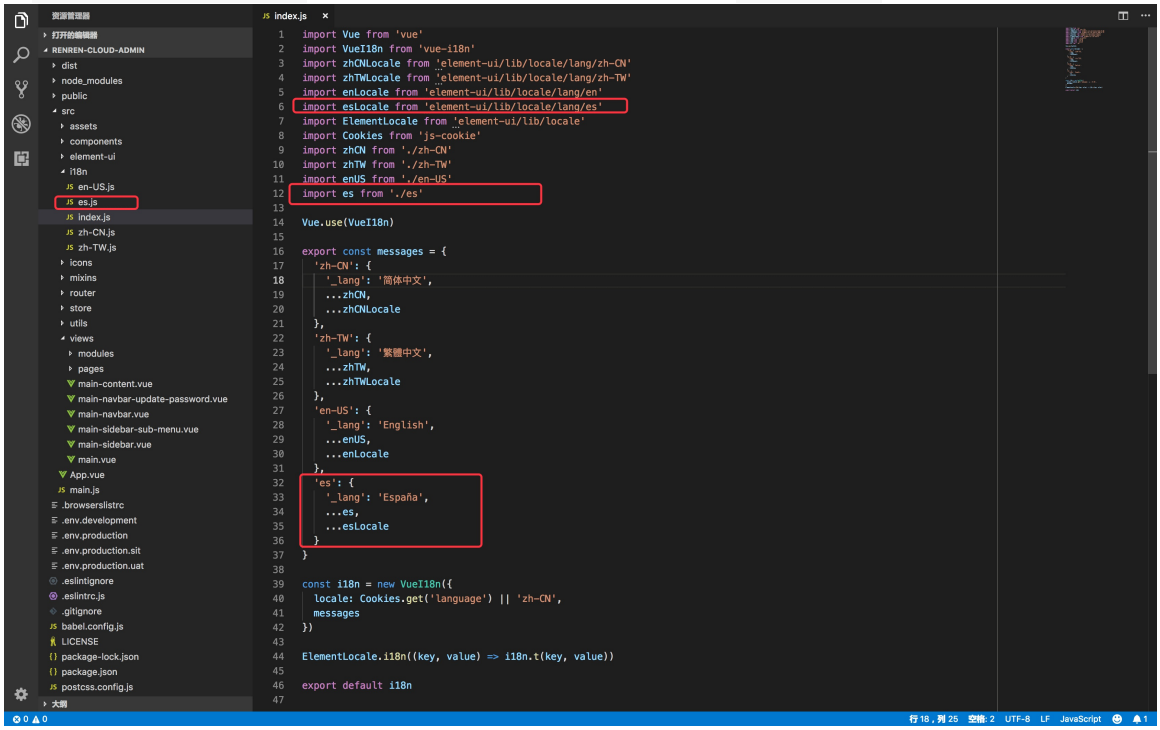
通过Vue-i18n插件提供的 `$t('xxx.xx')` 全局方法，获取定义的国际化 `xxx.xx` 属性值。

如果在组件 `data` 中通过 `$t('xxx.xx')` 方法使用国际化，在切换语言时，无法正常切换！需通过 Vue `computed` 计算和 `watch` 监听方式处理。

项目中使用 `computed` 计算处理 `表单验证信息` 。具体代码 `login登录页` 如下：

```
import debounce from 'lodash/debounce'
import { messages } from '@/i18n'
import { getUUID } from '@/utils'
export default {
  data () {
    return {
      i18nMessages: messages,
      captchaPath: '',
      dataForm: {
        username: '',
        password: '',
        uuid: '',
        captcha: ''
      }
    }
  },
  computed: {
    dataRule () {
      return {
        username: [
          { required: true, message: this.$t('validate.required'), trigger: 'blur'
}
        ],
        password: [
          { required: true, message: this.$t('validate.required'), trigger: 'blur'
}
        ],
        captcha: [
          { required: true, message: this.$t('validate.required'), trigger: 'blur'
}
        ]
      }
    }
  },
  created () {
    this.getCaptcha()
  },
  methods: {
    // 获取验证码
    getCaptcha () {
      this.dataForm.uuid = getUUID()
      this.captchaPath = `${window.SITE_CONFIG['apiURL']}/auth/captcha?uuid=${this.
dataForm.uuid}`
    },
    // 表单提交
    dataFormSubmitHandle: debounce(function () {
      this.$refs['dataForm'].validate((valid) => {
        if (!valid) {
          return false
        }
```

```
        this.$http.post('/auth/login', this.dataForm).then(({ data: res }) => {
          if (res.code !== 0) {
            this.getCaptcha()
            return this.$message.error(res.msg)
          }
          sessionStorage.setItem('token', res.data.token)
          this.$router.replace({ name: 'home' })
        }).catch(() => {})
      })
    }, 1000, { 'leading': true, 'trailing': false })
  }
}
```

新增语言

1. 在 `/src/i18n` 目录中创建对应的 `xxx.js` 语言文件。
2. 在 `/src/i18n/index.js` 文件中添加语言代码。下面假设新增 `es西班牙语` 。