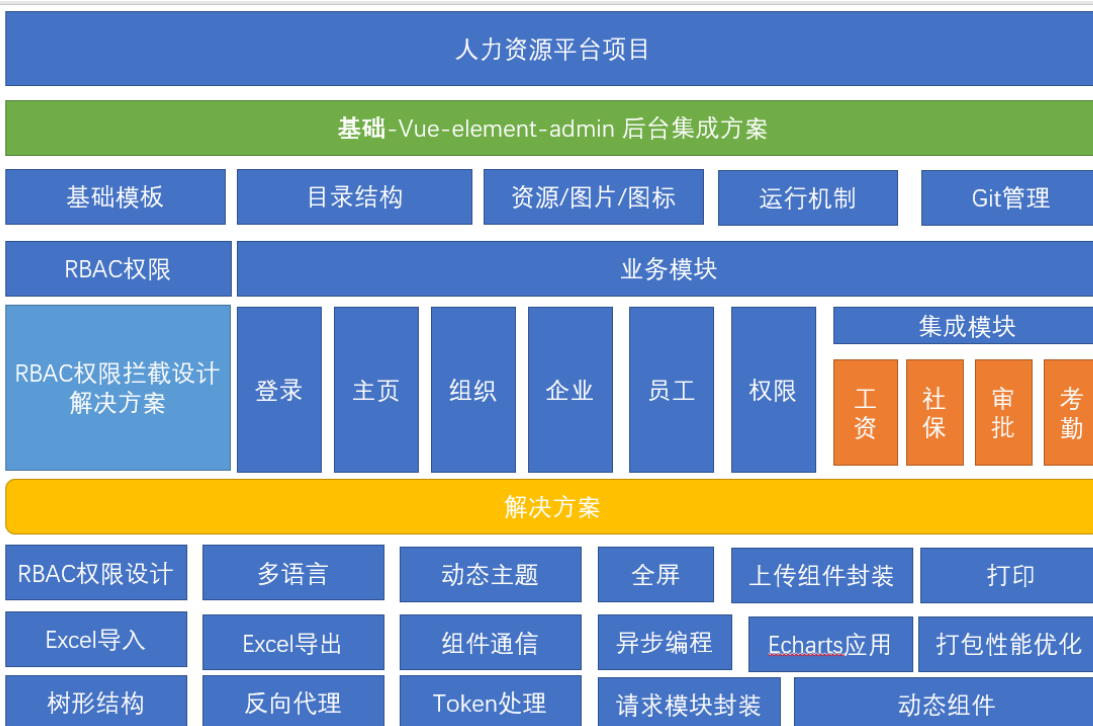


# 项目课设计



## 人力资源的环境搭建

## vue-element-admin的了解和介绍

## 目标：学习和了解通用的vue后台集成方案 vue-element-admin

[vue-element-admin](#) 是一个后台前端解决方案，它基于 [vue](#) 和 [element-ui](#) 实现。它使用了最新的前端技术栈，内置了 i18n 国际化解决方案，动态路由，权限验证，提炼了典型的业务模型，提供了丰富的功能组件，它可以帮助你快速搭建企业级中后台产品原型。

[vue-element-admin](#) 是一个后台集成方案, 集成了PC项目中很多的业务场景和功能, 尤其在当下SPA的趋势下, 我们可以从中获得很多成熟的解决方案.

[vue-element-admin](#) 有一个成熟的集成方案, 里面包含了所有的业务功能和场景, 并不适合直接拿来  
进行二次开发, 但是可以通过该项目中的一个案例来进行学习和使用.

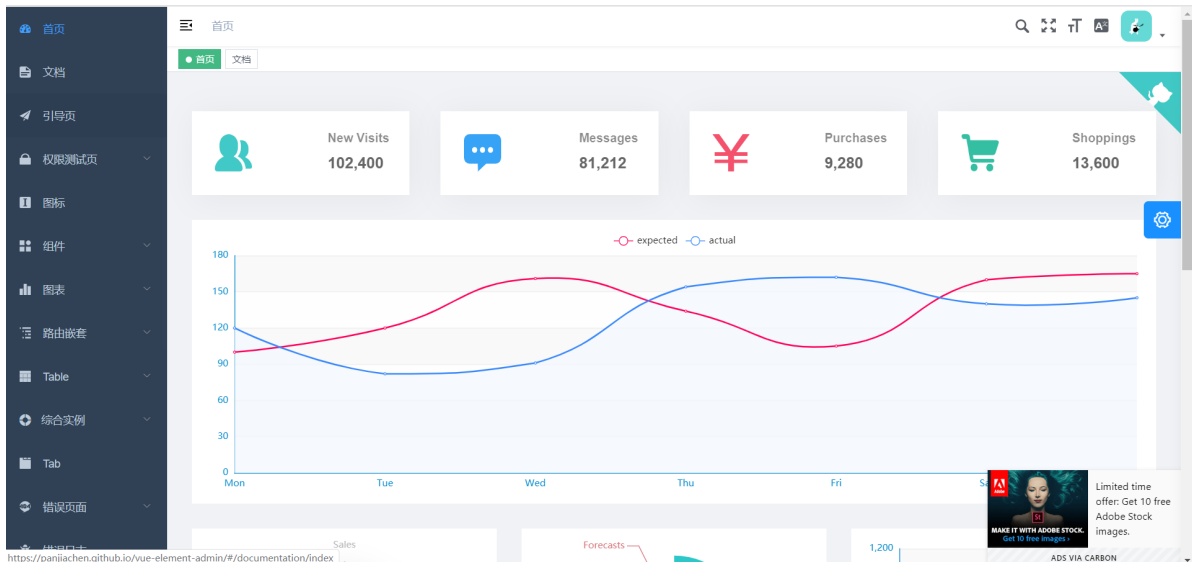
这里是[官网地址](#)

这里是[线上demo地址](#)

如果你想查看该项目的具体功能和效果，可以拉取代码，启动进行预览。

```
$ git clone https://github.com/PanJiaChen/vue-element-admin.git #拉取代码
$ cd vue-element-admin #切换到具体目录下
$ npm run dev #启动开发调试模式 查看package.json文件的scripts可知晓启动命令
```

**注意：**当前项目下载速度如果过慢，可以直接下载代码的压缩包运行



集成方案并不适合我们直接拿来进行二次开发，[基础模板](#)则是一个更好的选择

基础模板，包含了基本的 **登录 / 鉴权 / 主页布局** 的一些基础功能模板，我们可以直接在该模板上进行功能的扩展和项目的二次开发

**本节任务：** 浏览vue-element-admin的文档，了解这个集成方案

## 搭建项目前的一些基本准备

**目标：** 介绍搭建一个vue中台项目,需要环境和工具

接下来要做的是一个大型的项目，我们需要更好的环境准备和资源前置，所以提前检查我们的环境和资源

### nodejs环境

nodejs是当下前端工程化开发必不可少的环境，使用 nodejs的 **npm** 功能来管理依赖包

查看node 和 npm的版本

```
$ node -v #查看node版本
$ npm -v #查看npm版本
```

Windows PowerShell

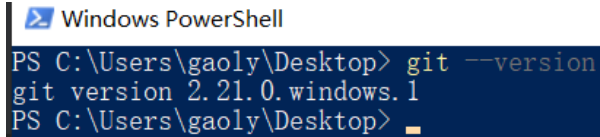
```
PS C:\Users\gaoly\Desktop> node -v
v12.10.0
PS C:\Users\gaoly\Desktop> npm -v
6.10.3
PS C:\Users\gaoly\Desktop>
```

### git版本控制

git版本控制工具是目前最为流行的分布式版本管理工具,代码的 **提交**, **检出**, **日志**, 都需要通过git完成

查看git安装版本

```
$ git --version #查看git安装版本
```



```
Windows PowerShell
PS C:\Users\gaoly\Desktop> git --version
git version 2.21.0.windows.1
PS C:\Users\gaoly\Desktop> █
```

## npm淘宝镜像

npm是非常重要的npm管理工具,由于npm的服务器位于国外, 所以一般建议 将 npm设置成国内的 淘宝镜像

设置淘宝镜像

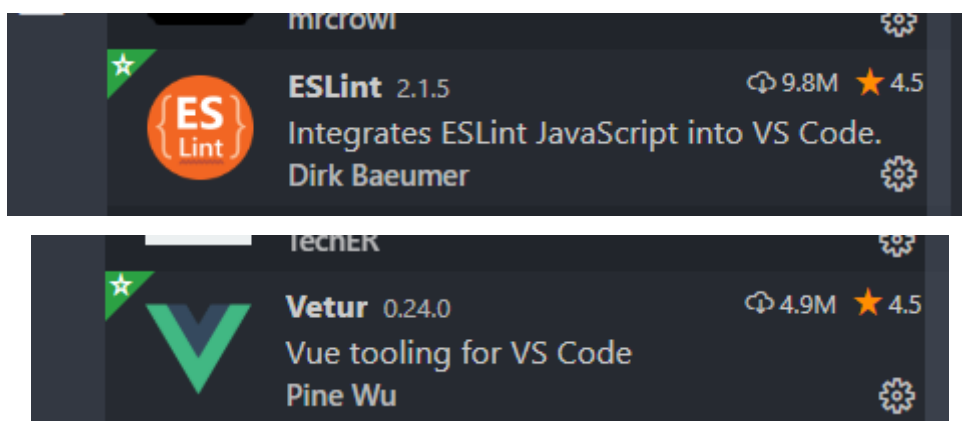
```
$ npm config set registry https://registry.npm.taobao.org/ #设置淘宝镜像地址
$ npm config get registry #查看镜像地址
```

## vscode编辑器

vscode编辑器是目前前端开发的编码利器, 以及丰富的插件系统, 非常适合开发前端项目

vscode编辑器插件 + **vetur** + **eslint**

vetur是基于 单文件组件开发的支持插件, eslint是 基于代码校验的插件工具



除此之外, eslint需要在vscode中进行一些参数的配置

```
{
  "eslint.enable": true,
  "eslint.run": "onType",
  "eslint.options": {
    "extensions": [
      ".js",
    ]
  }
}
```

```

        ".vue",
        ".jsx",
        ".tsx"
      ],
    },
    "editor.codeActionsOnSave": {
      "source.fixAll.eslint": true
    }
  }
}

```

本项目的技术栈 本项目技术栈基于 [ES2015+](#)、[vue](#)、[vuex](#)、[vue-router](#)、[vue-cli](#)、[axios](#) 和 [element-ui](#)

**本节任务：** 大家检查各自的开发环境和资源，尤其是npm的淘宝镜像额外需要注意

## 项目模板启动和目录介绍

**目标：** 拉取项目的基础模板,并对目录进行介绍

vue-element-admin的基础 模板和我们之前开发的项目一样吗？本章节，我们对该项目目录进行一下介绍

### git拉取基础项目模板

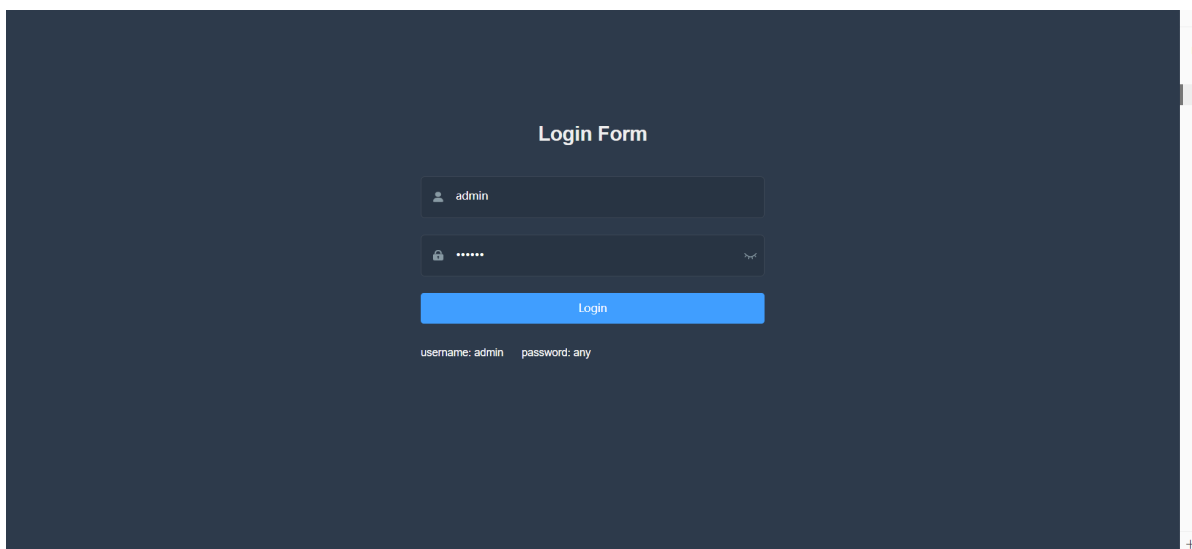
```
$ git clone https://github.com/PanJiaChen/vue-admin-template.git hrsaas #拉取基础模板到hrsaas目录
```

### 安装项目依赖（定位到项目目录下）

```
$ npm install #安装依赖
```

### 启动项目

```
$ npm run dev #启动开发模式的服务
```



项目运行完毕，浏览器会自动打开基础模板的登录页，如上图

### 目录结构

本项目已经为你生成了一个基本的开发框架，提供了涵盖中后台开发的各类功能和坑位，下面是整个项目的目录结构。

```
├── build                # 构建相关
├── mock                 # 项目mock 模拟数据
├── public               # 静态资源
│   ├── favicon.ico     # favicon图标
│   └── index.html       # html模板
├── src                  # 源代码
│   ├── api             # 所有请求
│   ├── assets           # 主题 字体等静态资源
│   ├── components      # 全局公用组件
│   ├── icons            # 项目所有 svg icons
│   ├── layout           # 全局 layout
│   ├── router           # 路由
│   ├── store            # 全局 store管理
│   ├── styles           # 全局样式
│   ├── utils            # 全局公用方法
│   ├── vendor           # 公用vendor
│   ├── views            # views 所有页面
│   ├── App.vue          # 入口页面
│   ├── main.js          # 入口文件 加载组件 初始化等
│   ├── permission.js    # 权限管理
│   └── settings.js      # 配置文件
├── tests                # 测试
├── .env.xxx             # 环境变量配置
├── .eslintrc.js         # eslint 配置项
├── .babelrc             # babel-loader 配置
├── .travis.yml          # 自动化CI配置
├── vue.config.js        # vue-cli 配置
├── postcss.config.js    # postcss 配置
└── package.json         # package.json
```

此时,你可能会**眼花缭乱**, 因为生成的目录里面有太多的文件 我们在做项目时 其中最关注的就是 **src** 目录, 里面是所有的源代码和资源, 至于其他目录, 都是对项目的环境和工具的配置

**本节任务**： 按照操作和讲解步骤，进行拉取代码，安装依赖，运行项目，阅读目录和文件的操作

**本节注意** 需要注意自己的npm是否已经设置了淘宝镜像

## 项目运行机制和代码注释

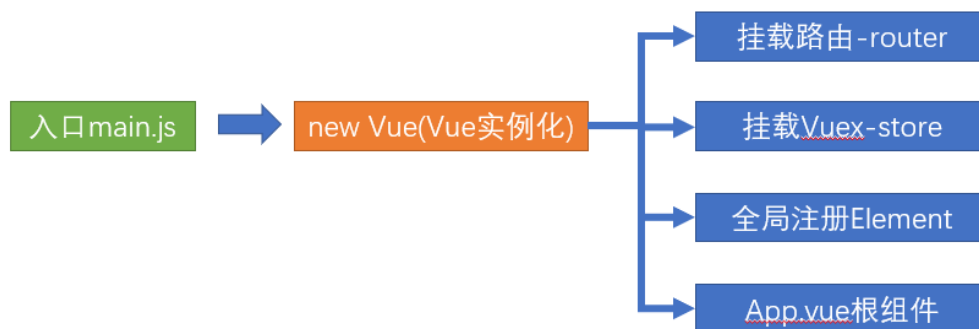
**目标**：了解当前模板的基本运行机制和基础架构

眼花缭乱的目录和文件到底是怎么工作的？ 我们进行一下最基本的讲解，帮助大家更好的去理解和开发

```
├── src                  # 源代码
│   ├── api             # 所有请求
│   ├── assets           # 主题 字体等静态资源
│   ├── components      # 全局公用组件
│   ├── icons            # 项目所有 svg icons
│   ├── layout           # 全局 layout
│   ├── router           # 路由
│   ├── store            # 全局 store管理
│   └── styles           # 全局样式
```

utils	# 全局公用方法
vendor	# 公用vendor
views	# views 所有页面
App.vue	# 入口页面
main.js	# 入口文件 加载组件 初始化等
permission.js	# 权限管理
settings.js	# 配置文件

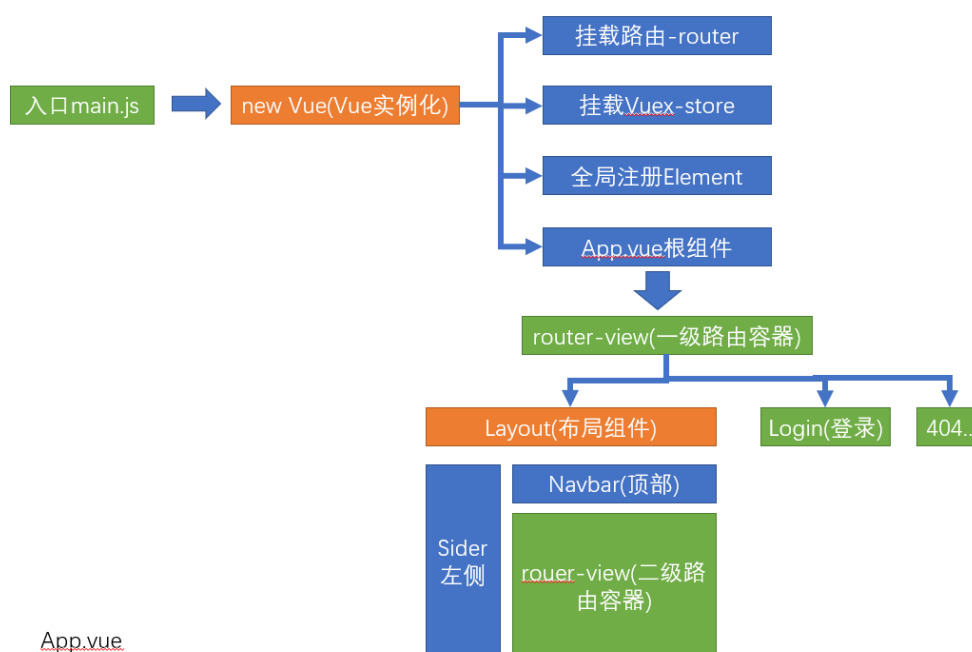
## main.js



请注释掉 **mock数据** 的部分，删除src下的 **mock** 文件夹，我们开发的时候用不到模拟数据，如图  
同时，请注释掉 **vue.config.js** 中的 **before: require('./mock/mock-server.js')**

```
// if (process.env.NODE_ENV === 'production') {
//   const { mockXHR } = require('../mock')
//   mockXHR()
// }
```

## App.vue



## permission.js

src下，除了main.js还有两个文件， `permission.js` 和 `settings.js`

`permission.js` 是控制**页面登录权限**的文件，此处的代码没有经历构建过程会很难理解，所以先将此处的代码进行注释，等我们构建权限功能时，再从0到1进行构建。

### 注释代码

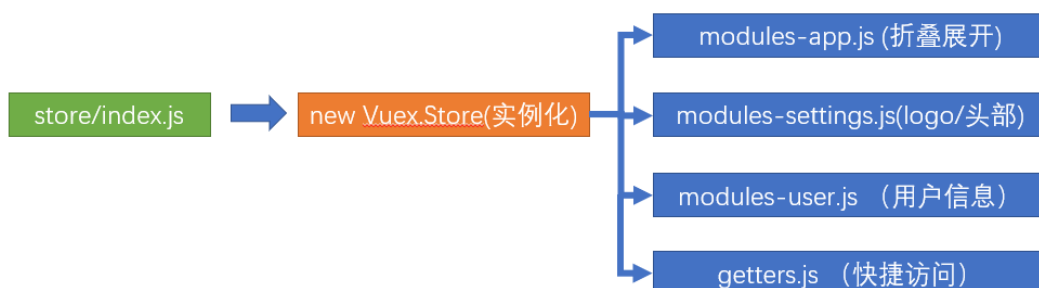
```
1 // import router from './router'
2 // import store from './store'
3 // import { Message } from 'element-ui'
4 // import NProgress from 'nprogress' // progress bar
5 // import 'nprogress/nprogress.css' // progress bar style
6 // import { getToken } from '@/utils/auth' // get token from cookie
7 // import getPageTitle from '@/utils/get-page-title'
8
```

`settings.js` 则是对于一些项目信息的配置，里面有三个属性 `title` (项目名称)，`fixedHeader` (固定头部)，`sidebarLogo` (显示左侧菜单logo)

`settings.js` 中的文件在其他的位置会引用到，所以这里暂时不去对该文件进行变动

## Vuex结构

当前的Vuex结构采用了模块形式进行管理共享状态，其架构如下



其中app.js模块和settings.js模块，功能已经完备，不需要再进行修改。user.js模块是我们后期需要重点开发的内容，所以这里我们将user.js里面的内容删除，并且导出一个默认配置

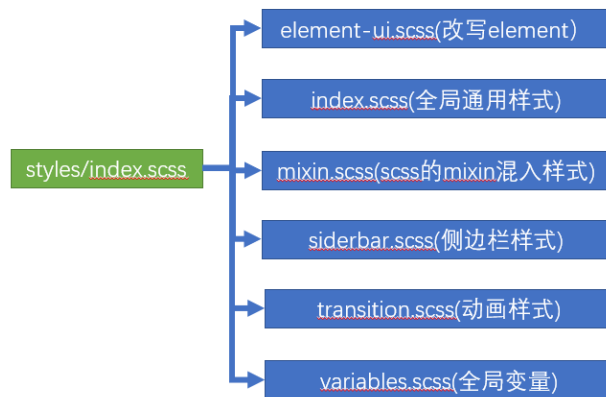
```
export default {
  namespaced: true,
  state: {},
  mutations: {},
  actions: {}
}
```

同时，由于getters中引用了user中的状态，所以我们将getters中的状态改为

```
const getters = {
  sidebar: state => state.app.sidebar,
  device: state => state.app.device
}
export default getters
```

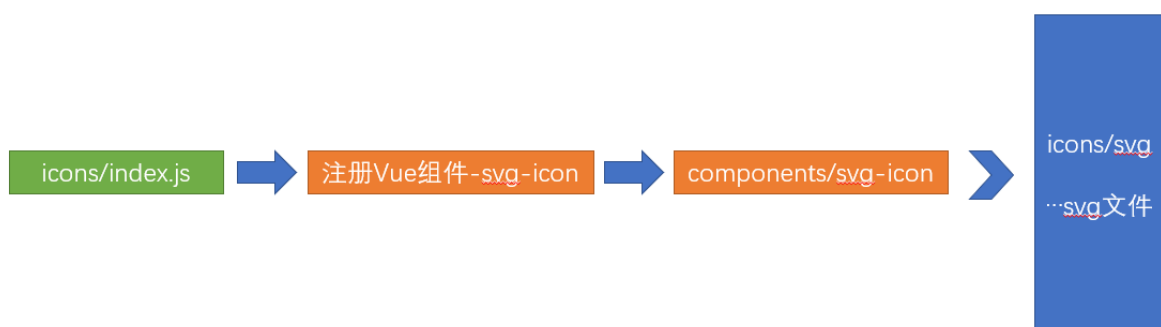
## SCSS

该项目还使用了[scss](#)作为css的扩展语言，在 `styles` 目录下，我们可以发现scss的相关文件，相关用法 我们[下一小节](#) 进行讲解



## icons

icons的结构如下



以上就是vue-element-admin的基础和介绍,希望大家在这过程中体会 一个基础的模板运行机制

**本节任务：** 大家根据目录结构和设计图，对以上的内容进行了解



# SCSS处理的了解和使用

**目标：**了解和学习Scss处理器的规范和用法

[官方文档](#)

首先注意,这里的sass和我们的scss是什么关系

sass和scss其实是 **一样的** css预处理语言, SCSS 是 Sass 3 引入新的语法, 其后缀名是分别为 .sass 和.scss两种。

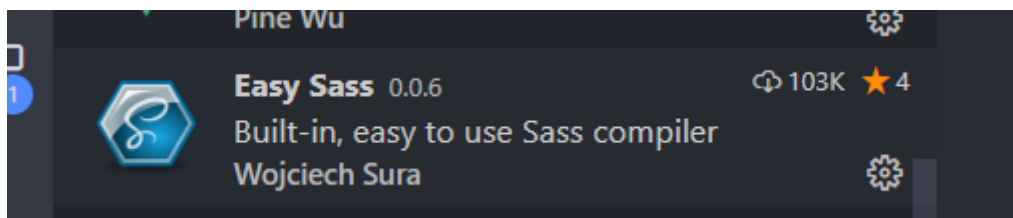
SASS版本3.0之前的后缀名为.sass, 而版本3.0之后的后缀名.scss。

两者是有不同的, 继sass之后scss的编写规范基本和css一致, sass时代是有严格的缩进规范并且没有'{'和';'。

而scss则和css的规范是一致的。

## 搭建小型测试环境

为了方便应用scss, 我们可以在vscode中安装一个名为 **easy sass** 的插件, 但是我们只在该项目中工作区中应用该插件, 因为在项目中, 不需要该插件的辅助



首先我们新建一个文件夹test, 然后我们在test下新建一个index.html, 并新建一个test.scss

页面结构如下

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <link rel="stylesheet" href="./test.css">
</head>
<body>
  <nav> Scss样式 </nav>
```

```

<div id="app">
  Hello World
</div>
<div id="content">
  <article>
    <h1>文章标题</h1>
    <p>文章内容 <a href="">百度</a> </p>
  </article>
  <aside>
    侧边栏
  </aside>
</div>
</body>
</html>

```

我们使用的 `easy sass` 插件会自动的帮助我们吧 `test.scss`  $\Rightarrow$  `test.css`

## 变量

sass 使用 `$` 符号来标识变量

```
$highlight-color: #f90
```

上面我们声明了一个 名为 `$highlight-color` 的变量, 我们可以把该变量用在任何位置

```

#app {
  background-color: $highlight-color;
}

```

以空格分割的多属性值也可以标识变量

```
$basic-border: 1px solid black;
```

```

#app {
  background-color: $highlight-color;
  border: $basic-border
}

```

## 变量范围

与 CSS 属性不同, 变量可以在 css 规则块定义之外存在。当变量定义在 css 规则块内, 那么该变量只能在此规则块内使用。如果它们出现在任何形式的 `{ ... }` 块中 (如 `@media` 或者 `@font-face` 块), 情况也是如此:

```

$nav-color: #F90;
nav {
  $width: 100px;
  width: $width;
  color: $nav-color;
  background-color: black
}

```

# 编译后

```
nav {
  width: 100px;
  color: #F90;
  background-color: black;
}
```

在这段代码中，`$nav-color` 这个变量定义在了规则块外边，所以在这个样式表中都可以像 `nav` 规则块那样引用它。`$width` 这个变量定义在了 `nav` 的 `{ }` 规则块内，所以它只能在 `nav` 规则块 内使用。这意味着是你可以样式表的其他地方定义和使用 `$width` 变量，不会对这里造成影响。

## 嵌套语法

和less一样,scss同样支持 **嵌套型** 的语法

```
#content {
  article {
    h1 { color: #1dc08a }
    p { font-style: italic; }
  }
  aside { background-color: #f90 }
}
```

转化后

```
#content article h1 {
  color: #1dc08a;
}

#content article p {
  font-style: italic;
}

#content aside {
  background-color: #f90;
}
```

## &父选择器

假如你想针对某个特定子元素 进行设置

比如

```
#content {
  article {
    h1 { color: #1dc08a }
    p { font-style: italic; }
    a {
      color: blue;
      &:hover { color: red }
    }
  }
  aside { background-color: #f90 }
}
```

学到这里,我们会发现scss和less有很多相似之处,最大的区别就在于声明变量的方式,less采用的是 `@变量名`,而scss采用的 `$变量名`

此时,我们再来看一下模板中的 `styles/variables.scss`

```
// sidebar
$menuText: #bfc9d9;
$menuActiveText: #409EFF;
$subMenuActiveText: #f4f4f5; //https://github.com/ElementFE/element/issues/12951

$menuBg: #304156;
$menuHover: #263445;

$subMenuBg: #1f2d3d;
$subMenuHover: #001528;

$sideBarWidth: 210px;
```

上述文件,实际上定义了我们的一些基础数值,方便大家在某个文件统一的处理.

**本节任务**：对scss进行了解和掌握

## 建立远程Git仓库并完成初始提交

**目标** 在[码云](#)或者[github](#)上建立相应的远程仓库,并将代码分支提交

### 建立远程仓库

远程仓库建立只需要在网站上直接操作即可

### 本地项目提交

**注意**：由于我们之前的项目是直接从 vue-element-admin **克隆** 而来,里面拥有原来的提交记录,为了避免冲突,先将原来的 `.git` 文件夹删除掉

并且对项目进行git初始化

```
$ git init #初始化项目
```

```
$ git add . #将修改添加到暂存
$ git commit -m '人资项目初始化' #将暂存提到本地仓库
```

### 查看版本日志

```
$ git log #查看版本日志
```

### 推送到远程仓库

推送到远程仓库一般先将 **远程仓库地址** 用本地仓库别名代替

```
$ git remote add origin <远程仓库地址> #添加远程仓库地址
```

当我们不清楚自己的仓库对应的origin地址时, 我们可以通过命令查看当前的远程仓库地址

```
$ git remote -v #查看本地仓库的远程仓库地址映射
```

## 推送master分支到远程仓库

```
$ git push -u origin master #将master分支推送到origin所代表的远程仓库地址
```

**本节任务：**同学们 根据以上操作,将拉取下的项目提交到自己的仓库里面

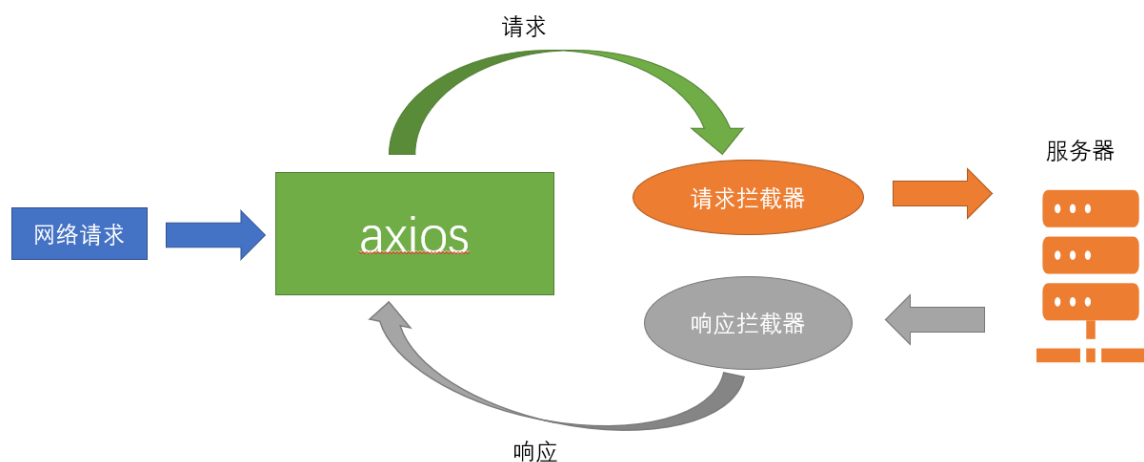
## API模块和请求封装模块介绍

**目标** 介绍API模块的单独请求和 request模块的封装

### Axios的拦截器介绍

该项目采用了API的单独模块封装和axios拦截器的方式进行开发

axios的拦截器原理如下



### axios拦截器

axios作为网络请求的第三方工具, 可以进行请求和响应的拦截

通过create创建了一个新的axios实例

```
// 创建了一个新的axios实例
const service = axios.create({
  baseURL: process.env.VUE_APP_BASE_API, // url = base url + request url
  // withCredentials: true, // send cookies when cross-domain requests
  timeout: 5000 // 超时时间
})
```

### 请求拦截器

请求拦截器主要处理 token的 统一注入问题

```
// axios的请求拦截器
service.interceptors.request.use(
  config => {
    // do something before request is sent

    if (store.getters.token) {
      // let each request carry token
      // ['X-Token'] is a custom headers key
      // please modify it according to the actual situation
    }
  }
```

```

    config.headers['X-Token'] = getToken()
  }
  return config
},
error => {
  // do something with request error
  console.log(error) // for debug
  return Promise.reject(error)
}
)

```

## 响应拦截器

响应拦截器主要处理 返回的 **数据异常** 和 **数据结构** 问题

```

// 响应拦截器
service.interceptors.response.use(
  response => {
    const res = response.data

    // if the custom code is not 20000, it is judged as an error.
    if (res.code !== 20000) {
      Message({
        message: res.message || 'Error',
        type: 'error',
        duration: 5 * 1000
      })
      if (res.code === 50008 || res.code === 50012 || res.code === 50014) {
        // to re-login
        MessageBox.confirm('You have been logged out, you can cancel to stay on this page, or log in again', 'Confirm logout', {
          confirmButtonText: 'Re-Login',
          cancelButtonText: 'Cancel',
          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)
  }
)

```

这里为了后续更清楚的书写代码,我们将原有代码注释掉,换成如下代码

```
// 导出一个axios的实例 而且这个实例要有请求拦截器 响应拦截器
import axios from 'axios'
const service = axios.create() // 创建一个axios的实例
service.interceptors.request.use() // 请求拦截器
service.interceptors.response.use() // 响应拦截器
export default service // 导出axios实例
```

## api模块的单独封装

我们习惯性的将所有的网络请求 放置在api目录下统一管理,按照模块进行划分

**单独封装代码**

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

export function login(data) {
  return request({
    url: '/vue-admin-template/user/login',
    method: 'post',
    data
  })
}

export function getInfo(token) {
  return request({
    url: '/vue-admin-template/user/info',
    method: 'get',
    params: { token }
  })
}

export function logout() {
  return request({
    url: '/vue-admin-template/user/logout',
    method: 'post'
  })
}
```

上面代码中,使用了封装的request工具,每个接口的请求都单独 **导出** 了一个方法,这样做的好处就是,**任何位置需要请求的话,可以直接引用我们导出的请求方法**

为了后续更好的开发,我们可以先将**user.js**代码的方法设置为空, 后续在进行更正

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

export function login(data) {
}

export function getInfo(token) {
```

```
}  
  
export function logout() {  
  
}
```

### 提交代码

**本节任务：** 将request和用户模块的代码进行清理，理解request和模块封装

## 公共资源图片和统一样式

**目标** 将一些公共的图片和样式资源放入到 规定目录中

我们已经将整体的基础模块进行了简单的介绍，接下来，我们需要将该项目所用到的图片和样式进行统一的处理

### 图片资源

图片资源在课程资料的图片文件中，我们只需要将 `common` 文件夹拷贝放置到 `assets` 目录即可

### 样式

样式资源在 资源/样式目录下

修改 `variables.scss`

新增 `common.scss`

我们在 `variables.scss` 添加了一些基础的变量值

我们提供了一份公共的 `common.scss` 样式,里面内置了一部分内容的样式,在开发期间可以帮助我们快速的实现页面样式和布局

将两个文件放置到`styles`目录下，然后在 `index.scss` 中引入该样式

```
@import './common.scss'; //引入common.scss样式表
```

### 提交代码

**本节注意：** 注意在 `scss` 文件中，通过`@import` 引入其他样式文件，需要注意最后加分号，否则会报错

**本节任务** 将公共资源的图片和样式放置到规定位置