# 一、搭建项目架构

## 创建项目

### 使用 Vue CLI 创建项目

```
vue create edu-boss-fed

Vue CLI v4.5.6
? Please pick a preset: Manually select features
? Check the features needed for your project: Babel, TS, Router, Vuex, CSS Pre-processors, Linter
? Use class-style component syntax? Yes
? Use Babel alongside TypeScript (required for modern mode, auto-detected polyfills, transpiling JSX)? Yes
? Use history mode for router? (Requires proper server setup for index fallback in production) No
? Pick a CSS pre-processor (PostCSS, Autoprefixer and CSS Modules are supported by default): Sass/SCSS (with dart-sass)
? Pick a linter / formatter config: Standard
? Pick additional lint features: Lint on save, Lint and fix on commit
? Where do you prefer placing config for Babel, ESLint, etc.? In dedicated config files
? Save this as a preset for future projects? No

⚓  Running completion hooks...

📄  Generating README.md...

🎉  Successfully created project topline-m-89.
🔑  Get started with the following commands:

 $ cd edu-boss-fed
 $ npm run serve
```

安装结束，启动开发服务：

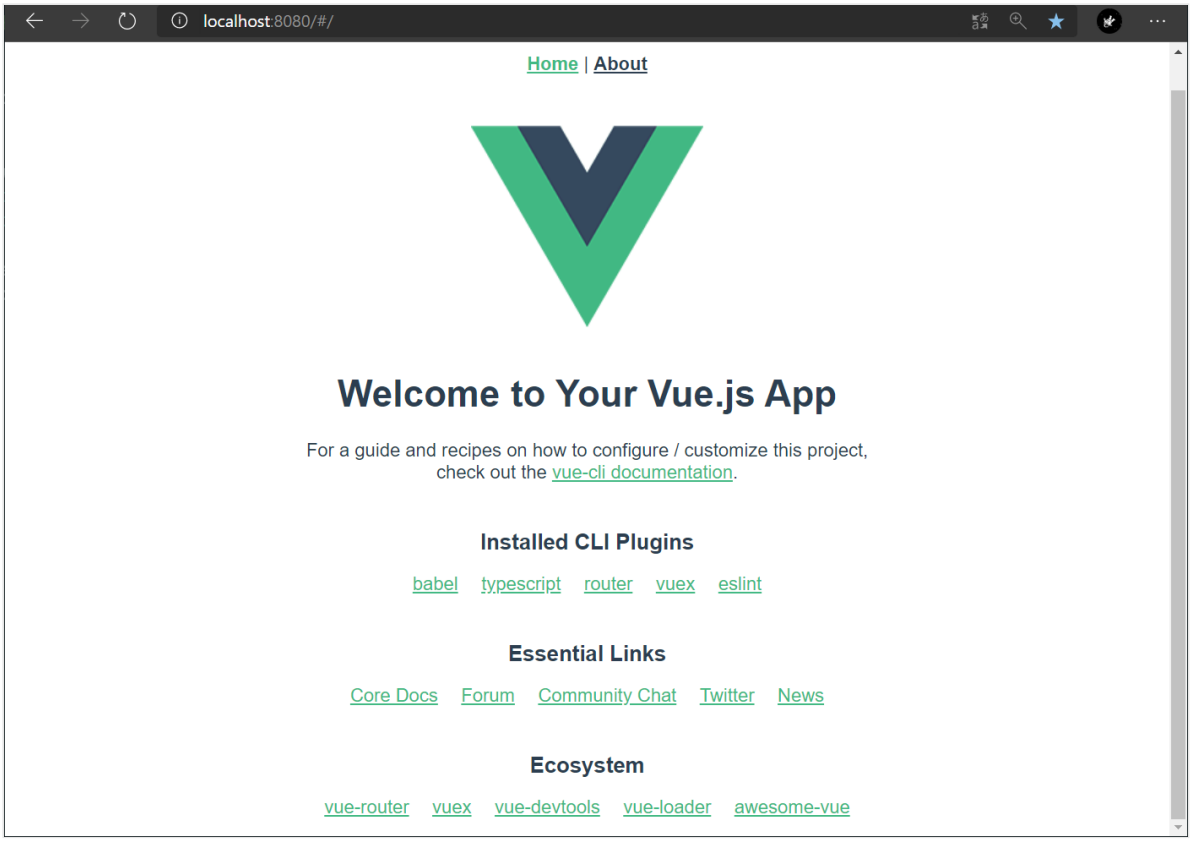
```
# 进入你的项目目录
cd edu-boss-fed

# 启动开发服务
npm run serve
```

```
  App running at:
  - Local:   http://localhost:8080/
  - Network: http://10.10.100.145:8080/

  Note that the development build is not optimized.
  To create a production build, run npm run build.
```

启动成功，根据提示访问给出的服务地址。




如果能看到该页面，恭喜你，项目创建成功了。

## 加入 Git 版本管理

（1）创建远程仓库

（2）将本地仓库推到线上

如果没有本地仓库：

```
# 创建本地仓库
git init

# 将文件添加到暂存区
git add 文件

# 提交历史记录
git commit "提交日志"

# 添加远端仓库地址
git remote add origin 你的远程仓库地址

# 推送提交
git push -u origin master
```

如果已有本地仓库：

```
# 添加远端仓库地址
git remote add origin 你的远程仓库地址

# 推送提交
git push -u origin master
```

## 初始目录结构说明

```
.
├── node_modules # 第三方包存储目录
├── public # 静态资源目录，任何放置在 public 文件夹的静态资源都会被简单的复制，而不经过 webpack
|   ├── favicon.ico
|   └── index.html
├── src
|   ├── assets # 公共资源目录，放图片等资源
|   ├── components # 公共组件目录
|   ├── router # 路由相关模块
|   ├── store # 容器相关模块
|   ├── views # 路由页面组件存储目录
|   ├── App.vue  # 根组件，最终被替换渲染到 index.html 页面中 #app 入口节点
|   ├── main.ts # 整个项目的启动入口模块
|   ├── shims-tsx.d.ts # 支持以 .tsc 结尾的文件，在 Vue 项目中编写 jsx 代码
|   └── shims-vue.d.ts # 让 TypeScript 识别 .vue 模块
├── .browserslistrc # 指定了项目的目标浏览器的范围。这个值会被 @babel/preset-env 和 Autoprefixer 用来确定需要转译的 JavaScript 特性和需要添加的 CSS 浏览器前缀
├── .editorconfig # EditorConfig 帮助开发人员定义和维护跨编辑器（或IDE）的统一的代码风格
├── .eslintrc.js # ESLint 的配置文件
├── .gitignore # Git 的忽略配置文件，告诉Git项目中要忽略的文件或文件夹
├── README.md # 说明文档
├── babel.config.js # Babel 配置文件
├── package-lock.json # 记录安装时的包的版本号，以保证自己或其他人在 npm install 时大家的依赖能保证一致
├── package.json # 包说明文件，记录了项目中使用到的第三方包依赖信息等内容
└── tsconfig.json # TypeScript 配置文件
```

## 调整初始目录结构

默认生成的目录结构不满足我们的开发需求，所以需要做一些自定义改动。

这里主要处理下面的内容：

- 删除初始化的默认文件
- 新增调整我们需要的目录结构

修改 `App.vue`：

```
<template>
  <div id="app">
    <h1>App</h1>

    <!-- 根级路由出口 -->
    <router-view/>
  </div>
</template>

<style lang="scss" scoped></style>
```

修改 `router/index.ts`：

```
import Vue from 'vue'
```

```
import VueRouter, { RouteConfig } from 'vue-router'

Vue.use(VueRouter)

// 路由规则
const routes: Array<RouteConfig> = []

const router = new VueRouter({
  routes
})

export default router
```

删除默认示例文件：

- src/views/About.vue
- src/views/Home.vue
- src/components/HelloWorld.vue
- src/assets/logo.png

创建以下内容：

- src/services 目录，接口模块
- src/utils 目录，存储一些工具模块
- src/styles 目录，存储一些样式资源

调整之后的目录结构如下。

```
.
├── public
│   ├── favicon.ico
│   └── index.html
├── src
│   ├── assets
│   ├── components
│   ├── router
│   ├── services
│   ├── store
│   ├── styles
│   ├── utils
│   ├── views
│   ├── App.vue
│   ├── main.ts
│   ├── shims-tsx.d.ts
│   └── shims-vue.d.ts
├── .browserslistrc
├── .editorconfig
├── .eslintrc.js
├── .gitignore
├── README.md
├── babel.config.js
├── package-lock.json
├── package.json
```

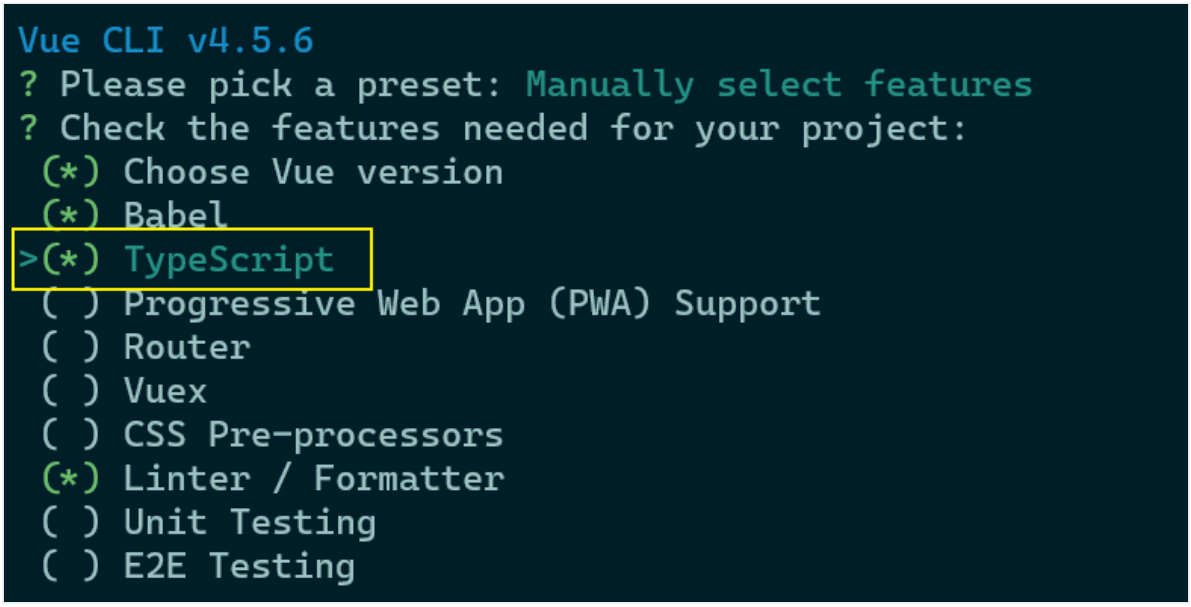
```
└── tsconfig.json
```

# 使用 TypeScript 开发 Vue 项目

## 在 Vue 项目中启用 TypeScript 支持

两种方式：

（1）全新项目：使用 Vue CLI 脚手架工具创建 Vue 项目




（2）已有项目：添加 Vue 官方配置的 TypeScript 适配插件

使用 @vue/cli 安装 TypeScript 插件：

```
vue add @vue/typescript
```

## 关于编辑器

要使用 TypeScript 开发 Vue 应用程序，我们强烈建议您使用 Visual Studio Code，它为 TypeScript 提供了极好的“开箱即用”支持。如果你正在使用单文件组件 (SFC)，可以安装提供 SFC 支持以及其他更多实用功能的 Vetur 插件。

WebStorm 同样为 TypeScript 和 Vue 提供了“开箱即用”的支持。

## TypeScript 相关配置说明

（1）安装了 TypeScript 相关的依赖项

dependencies 依赖：

| 依赖项 | 说明 |
| --- | --- |
| vue-class-component | 提供使用 Class 语法写 Vue 组件 |
| vue-property-decorator | 在 Class 语法基础之上提供了一些辅助装饰器 |

devDependencies 依赖：

| 依赖项 | 说明 |
| --- | --- |
| @typescript-eslint/eslint-plugin | 使用 ESLint 校验 TypeScript 代码 |
| @typescript-eslint/parser | 将 TypeScript 转为 AST 供 ESLint 校验使用 |
| @vue/cli-plugin-typescript | 使用 TypeScript + ts-loader + fork-ts-checker-webpack-plugin进行更快的类型检查。 |
| @vue/eslint-config-typescript | 兼容 ESLint 的 TypeScript 校验规则 |
| typescript | TypeScript 编译器，提供类型校验和转换 JavaScript 功能 |

（2）TypeScript 配置文件 `tsconfig.json`

```
{
  "compilerOptions": {
    "target": "esnext",
    "module": "esnext",
    "strict": true,
    "jsx": "preserve",
    "importHelpers": true,
    "moduleResolution": "node",
    "experimentalDecorators": true,
    "skipLibCheck": true,
    "esModuleInterop": true,
    "allowSyntheticDefaultImports": true,
    "sourceMap": true,
    "baseUrl": ".",
    "types": [
      "webpack-env"
    ],
    "paths": {
      "@/*": [
        "src/*"
      ]
```

```
    },
    "lib": [
      "esnext",
      "dom",
      "dom.iterable",
      "scripthost"
    ]
  },
  "include": [
    "src/**/*.ts",
    "src/**/*.tsx",
    "src/**/*.vue",
    "tests/**/*.ts",
    "tests/**/*.tsx"
  ],
  "exclude": [
    "node_modules"
  ]
}
```

(3) shims-vue.d.ts 文件的作用

```
// 主要用于 TypeScript 识别 .vue 文件模块
// TypeScript 默认不支持导入 .vue 模块，这个文件告诉 TypeScript 导入 .vue 文件模块都按
VueConstructor<Vue> 类型识别处理
declare module '*.vue' {
  import Vue from 'vue'
  export default Vue
}
```

(4) shims-tsx.d.ts 文件的作用

```
/**
 * 为 jsx 组件模板补充类型声明
 */

import Vue, { VNode } from 'vue'

declare global {
  namespace JSX {
    // tslint:disable no-empty-interface
    interface Element extends VNode {}
    // tslint:disable no-empty-interface
    interface ElementClass extends Vue {}
    interface IntrinsicElements {
      [elem: string]: any;
    }
  }
}
```

（5）TypeScript 模块都使用 `.ts` 后缀

# 定义组件的方式

## 使用 Options APIs

- 组件仍然可以使用以前的方式定义（导出组件选项对象，或者使用 Vue.extend()）
- 但是当我们导出的是一个普通的对象，此时 TypeScript 无法推断出对应的类型，
- 至于 VSCode 可以推断出类型成员的原因是因为我们使用了 Vue 插件，
- 这个插件明确知道我们这里导出的是一个 Vue 对象。
- 所以我们必须使用 `Vue.extend()` 方法确保 TypeScript 能够有正常的类型推断

```
const Component = {
  // 这里不会有类型推断，
  // 因为 TypeScript 不能确认这是 Vue 组件的选项
}
```

```
import Vue from 'vue'

// 1. 准确的类型推断
// 2. 结合编辑器提供的类型及错误提示等辅助功能
export default Vue.extend({
  // 类型推断已启用
  name: 'Button',
  data () {
    return {
      count: 1
    }
  },
  methods: {
    increment () {
      this.count++
    }
  }
})
```

## 使用 Class APIs

参考：

- https://class-component.vuejs.org/

在 TypeScript 下，Vue 的组件可以使用一个继承自 Vue 类型的子类表示，这种类型需要使用 Component 装饰器去修饰

装饰器函数接收的参数就是以前的组件选项对象（data、props、methods 之类）

```
import Vue from 'vue'
import Component from 'vue-class-component' // 官方库
```

```
@Component({
  props: {
    size: String
  }
})
export default class Button extends Vue {
  private count: number = 1
  private text: string = 'Click me'

  get content () {
    return `${this.text} ${this.count}`
  }

  increment () { // 事件处理函数
    this.count++
  }

  mounted () { // 生命周期函数
    console.log('button is mounted')
  }
}
```

- Data: 使用类的实例属性声明
- Method: 使用类的实例方法声明
- Computed: 使用 Getter 属性声明
- 生命周期: 使用类的实例方法声明

其它特性：例如 components, props, filters, directives 之类的，则需要使用修饰器参数传入

使用这种 class 风格的组件声明方式并没有什么特别的好处，只是为了提供给开发者多种编码风格的选择性

## 关于装饰器语法

推荐参考文档:

- https://www.typescriptlang.org/docs/handbook/decorators.html
- https://es6.ruanyifeng.com/#docs/decorator

```
import Vue from 'vue'
import Component from 'vue-class-component'

// Define the component in class-style
@Component
export default class Counter extends Vue {
  // Class properties will be component data
  count = 0

  // Methods will be component methods
  increment() {
    this.count++
  }
```

```
  decrement() {
    this.count--
  }
}
```

装饰器是 ES 草案中的一个新特性，不过这个草案最近有可能发生重大调整，所以不建议在生产环境中使用。

类的装饰器：

```
function testable (target) {
  target.isTestable = true
}

@testable
class MyTestableClass {
  // ...
}

console.log(MyTestableClass.isTestable) // true
```

如果觉得一个参数不够用，可以在装饰器外面再封装一层函数。

```
function testable(isTestable) {
  return function(target) {
    target.isTestable = isTestable;
  }
}

@testable(true)
class MyTestableClass {}
MyTestableClass.isTestable // true

@testable(false)
class MyClass {}
MyClass.isTestable // false
```

### 使用 Class APIs + vue-property-decorator

```
import { Vue, Component, Prop } from 'vue-property-decorator'

@Component
export default class Button extends Vue {
  private count: number = 1
  private text: string = 'Click me'
  @Prop() readonly size?: string

  get content () {
    return `${this.text} ${this.count}`
  }

  increment () {
    this.count++
  }

  mounted () {
    console.log('button is mounted')
  }
}
```

这种方式继续放大了 Class 这种组件定义方法。

### 总结一下

创建组件的三种方式：

1、Options APIs

```
import Vue from 'vue'

export default Vue.extend({
  data () {
    return {
      count: 0
    }
  },

  methods: {
    increment() {
      this.count++
    }
```

```
    decrement() {
      this.count--
    }
  }
})
```

2、Class APIs

```
import Vue from 'vue'
import Component from 'vue-class-component'

// Define the component in class-style
@Component
export default class Counter extends Vue {
  // Class properties will be component data
  count = 0

  // Methods will be component methods
  increment() {
    this.count++
  }

  decrement() {
    this.count--
  }
}
```

3、Class APIs + decorator

```
import { Vue, Component, Prop } from 'vue-property-decorator'

@Component
export default class YourComponent extends Vue {
  @Prop(Number) readonly propA: number | undefined
  @Prop({ default: 'default value' }) readonly propB!: string
  @Prop([String, Boolean]) readonly propC: string | boolean | undefined
}
```

个人建议：No Class APIs，只用 Options APIs。

> Class 语法仅仅是一种写法而已，最终还是要转换为普通的组件数据结构。
>
> 装饰器语法还没有正式定稿发布，建议了解即可，正式发布以后在选择使用也可以。

使用 Options APIs 最好是使用 `export default Vue.extend({ ... })` 而不是 `export default { ... }`。

# 代码格式规范

这里主要说明以下几点：

- 代码格式规范介绍
- 我们项目中配置的具体代码规范是什么
- 遇到代码格式规范错误怎么办
- 如何自定义代码格式校验规范

## 代码格式规范介绍

新手写的代码：

```
if(delayedTasks.size()==0) //no tasks
return null;
//first priority - to storage
if(getFreePlace()!=null)
for (Task task:delayedTasks)
if(((Mould)task.agent).cake.state==AT_STORAGE_ENTER)
{return task;}
//second priority - from storage
if(main.fromRisingArea.canEnter())
for(Task task:delayedTasks)
if (((Mould)task.agent).cake.state==RISING_FINISHED){
return task;}
return null;
```

有经验的同学写的代码：

```
if (delayedTasks.size() == 0) // no tasks
    return null;
// first priority - to storage
if (getFreePlace() != null)
    for (Task task : delayedTasks)
        if (((Mould) task.agent).cake.state == AT_STORAGE_ENTER) {
            return task;
        }
// second priority - from storage
if (main.fromRisingArea.canEnter())
    for (Task task : delayedTasks)
        if (((Mould) task.agent).cake.state == RISING_FINISHED) {
            return task;
        }
return null;
```

良好的代码格式规范更有利于：

- 更好的多人协作
- 更好的阅读
- 更好的维护
- ...

## 标准是什么

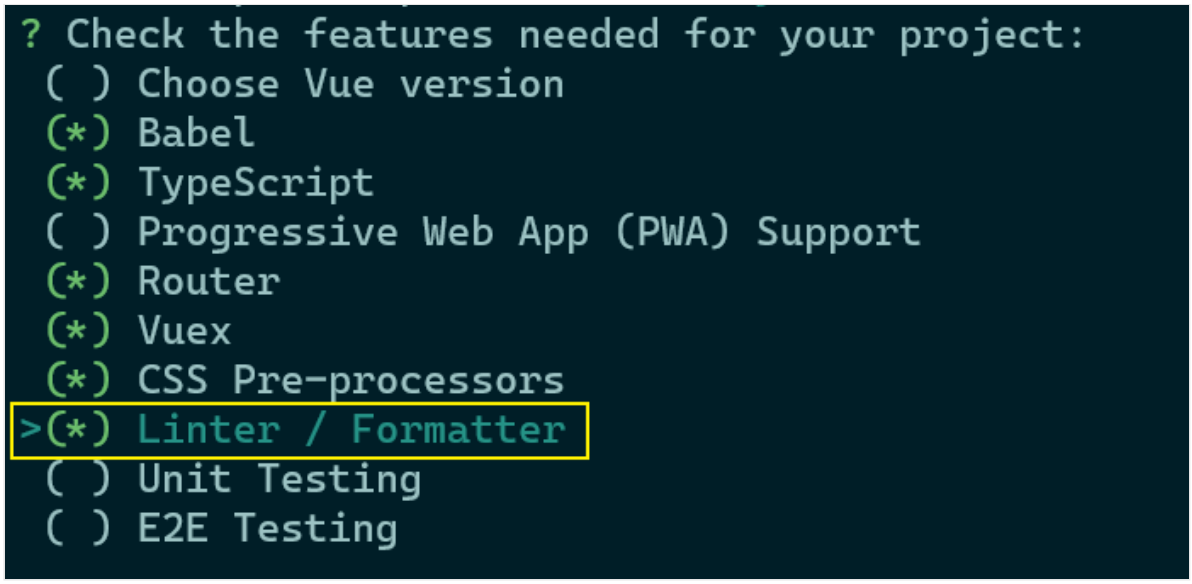
没有绝对的标准，下面是一些大厂商根据多数开发者的编码习惯制定的一些编码规范，仅供参考。

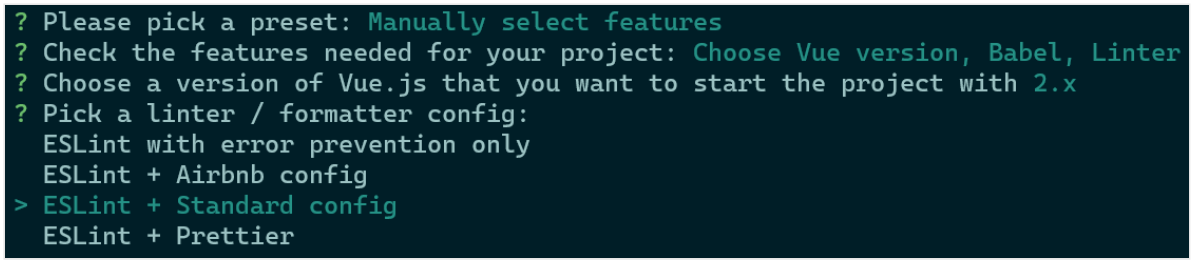
- JavaScript Standard Style
- Airbnb JavaScript Style Guide
- Google JavaScript Style Guide

## 如何约束代码规范

只靠口头约定肯定是不行的，所以要利用工具来强制执行。

- JSLint
- JSHint
- ESLint
- ...




## 项目中的代码规范是什么

ESLint 配置文件：

```
module.exports = {
  root: true,
  env: {
    node: true
  },
  // 插件：扩展了校验规则
  extends: [
    'plugin:vue/essential', // eslint-plugin-vue
```

```
    '@vue/standard', // @vue/eslint-config-standard
    '@vue/typescript/recommended' // @vue/eslint-config-typescript
  ],
  parserOptions: {
    ecmaVersion: 2020
  },

  // 自定义验证规则
  rules: {
    'no-console': process.env.NODE_ENV === 'production' ? 'warn' : 'off',
    'no-debugger': process.env.NODE_ENV === 'production' ? 'warn' : 'off'
  }
}
```

- eslint-plugin-vue
  - GitHub 仓库：https://github.com/vuejs/eslint-plugin-vue
  - 官方文档：https://eslint.vuejs.org/
  - 该插件使我们可以使用 ESLint 检查 `.vue` 文件的 `<template>` 和 `<script>`
  - 查找语法错误
  - 查找对Vue.js指令的错误使用
  - 查找违反Vue.js样式指南的行为
- @vue/eslint-config-standard
  - eslint-plugin-standard
  - JavaScript Standard Style
- @vue/eslint-config-typescript
  - 规则列表：https://github.com/typescript-eslint/typescript-eslint/tree/master/packages/eslint-plugin#supported-rules

## 如何自定义代码格式校验规范

```
{
  "rules": {
    "semi": ["error", "always"],
      "quotes": ["error", "double"]
  }
}
```

ESLint 附带有大量的规则。你可以使用注释或配置文件修改你项目中要使用的规则。要改变一个规则设置，你必须将规则 ID 设置为下列值之一：

- `"off"` 或 `0` - 关闭规则
- `"warn"` 或 `1` - 开启规则，使用警告级别的错误：`warn` (不会导致程序退出)
- `"error"` 或 `2` - 开启规则，使用错误级别的错误：`error` (当被触发的时候，程序会退出)

为了在文件注释里配置规则，使用以下格式的注释：

```
/* eslint eqeqeq: "off", curly: "error" */
```

在这个例子里，`eqeqeq` 规则被关闭，`curly` 规则被打开，定义为错误级别。你也可以使用对应的数字定义规则严重程度：

```
/* eslint eqeqeq: 0, curly: 2 */
```

这个例子和上个例子是一样的，只不过它是用的数字而不是字符串。`eqeqeq` 规则是关闭的，`curly` 规则被设置为错误级别。

如果一个规则有额外的选项，你可以使用数组字面量指定它们，比如：

```
/* eslint quotes: ["error", "double"], curly: 2 */
```

这条注释为规则 `quotes` 指定了 “double”选项。数组的第一项总是规则的严重程度（数字或字符串）。

还可以使用 `rules` 连同错误级别和任何你想使用的选项，在配置文件中进行规则配置。例如：

```
{
    "rules": {
        "eqeqeq": "off",
        "curly": "error",
        "quotes": ["error", "double"]
    }
}
```

配置定义在插件中的一个规则的时候，你必须使用 `插件名/规则ID` 的形式。比如：

```
{
    "plugins": [
        "plugin1"
    ],
    "rules": {
        "eqeqeq": "off",
        "curly": "error",
        "quotes": ["error", "double"],
        "plugin1/rule1": "error"
    }
}
```

在这些配置文件中，规则 `plugin1/rule1` 表示来自插件 `plugin1` 的 `rule1` 规则。你也可以使用这种格式的注释配置，比如：

```
/* eslint "plugin1/rule1": "error" */
```

**注意：**当指定来自插件的规则时，确保删除 `eslint-plugin-` 前缀。ESLint 在内部只使用没有前缀的名称去定位规则。

# 导入 Element 组件库

Element，一套为开发者、设计师和产品经理准备的基于 Vue 2.0 的桌面端组件库。

- 官网：https://element.eleme.cn/
- GitHub 仓库：https://github.com/ElemeFE/element

1、安装 element

```
npm i element-ui -S
```

2、在 `main.ts` 中导入配置

```
import Vue from 'vue'
import App from './App.vue'
import router from './router'
import store from './store'
import ElementUI from 'element-ui'
import 'element-ui/lib/theme-chalk/index.css'

Vue.use(ElementUI)

Vue.config.productionTip = false

new Vue({
  router,
  store,
  render: h => h(App)
}).$mount('#app')
```

3、测试使用

# 样式处理

```
src/styles
├── index.scss # 全局样式（在入口模块被加载生效）
├── mixin.scss # 公共的 mixin 混入（可以把重复的样式封装为 mixin 混入到复用的地方）
├── reset.scss # 重置基础样式
└── variables.scss # 公共样式变量
```

`variables.scss`

```
$primary-color: #40586F;
$success-color: #51cf66;
$warning-color: #fcc419;
$danger-color: #ff6b6b;
$info-color: #868e96; // #22b8cf;

$body-bg: #E9EEF3; // #f5f5f9;

$sidebar-bg: #F8F9FB;
$navbar-bg: #F8F9FB;

$font-family: system-ui, -apple-system, "Segoe UI", Roboto, Helvetica, Arial,
sans-serif;
```

index.scss

```scss
@import './variables.scss';

// globals
html {
  font-family: $font-family;
  -webkit-text-size-adjust: 100%;
  -webkit-tap-highlight-color: rgba(0, 0, 0, 0);
  // better Font Rendering
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}

body {
  margin: 0;
  background-color: $body-bg;
}

// custom element theme
$--color-primary: $primary-color;
$--color-success: $success-color;
$--color-warning: $warning-color;
$--color-danger: $danger-color;
$--color-info: $info-color;
// change font path, required
$--font-path: '~element-ui/lib/theme-chalk/fonts';
// import element default theme
@import '~element-ui/packages/theme-chalk/src/index';
// node_modules/element-ui/packages/theme-chalk/src/common/var.scss

// overrides

// .el-menu-item, .el-submenu__title {
//   height: 50px;
//   line-height: 50px;
// }

.el-pagination {
  color: #868e96;
}

// components

.status {
  display: inline-block;
  cursor: pointer;
  width: .875rem;
  height: .875rem;
  vertical-align: middle;
  border-radius: 50%;

  &-primary {
    background: $--color-primary;
  }
```

```
  &-success {
    background: $--color-success;
  }

  &-warning {
    background: $--color-warning;
  }

  &-danger {
    background: $--color-danger;
  }

  &-info {
    background: $--color-info;
  }
}
```

# 共享全局样式变量

参考：
https://cli.vuejs.org/zh/guide/css.html#%E5%90%91%E9%A2%84%E5%A4%84%E7%90%86%E5%99%A8-loader-%E4%BC%A0%E9%80%92%E9%80%89%E9%A1%B9

```
module.exports = {
  ...
  css: {
    loaderOptions: {
      sass: {
        prependData: `@import "~@/styles/variables.scss";`
      }
    }
  }
}
```

# 接口处理

## 配置接口代理

后台为我们提供了数据接口，分别是：

- https://eduboss.lagou.com
- http://edufront.lagou.com

这两个接口都没有提供 CORS 跨域请求，所以需要在客户端配置服务端代理处理跨域请求。

配置客户端层面的服务端代理跨域可以参考官方文档中的说明：

- https://cli.vuejs.org/zh/config/#devserver-proxy
- https://github.com/chimurai/http-proxy-middleware

下面是具体的操作流程。

在项目根目录下添加 `vue.config.js` 配置文件。

```
module.exports = {
  ...
  devServer: {
    proxy: {
      '/front': {
        target: 'http://edufront.lagou.com',
        changeOrigin: true // 设置请求头中的 host 为 target，防止后端反向代理服务器无法识别
      },
      '/boss': {
        target: 'http://eduboss.lagou.com',
        changeOrigin: true
      }
    }
  }
}
```

## 封装请求模块

安装 axios：

```
npm i axios
```

创建 `src/utils/request.js`：

```
import axios from 'axios'

const request = axios.create({
  // 配置选项
})

export default request
```

### 配置环境变量

> 知识点：
>
> - 配置 Vue 项目中的环境变量
> - dotenv

`.env.development`

```
VUE_APP_API=http://eduboss.lagou.com
```

`.env.production`

```
VUE_APP_API=http://eduboss.lagou.com
```

## 路由配置

我们这里先把这几个主要的页面配置出来，其它页面在随后的开发过程中配置。

| 路径 | 说明 |
| --- | --- |
| / | 首页 |
| /login | 用户登录 |
| /role | 角色管理 |
| /menu | 菜单管理 |
| /resource | 资源管理 |
| /course | 课程管理 |
| /user | 用户管理 |
| /advert | 广告管理 |
| /advert-space | 广告位管理 |

### Layout 布局

**登录页面**

**404 处理**

**其它页面**