搭建项目架构

创建项目

使用 Vue CLI 创建项目

安装 Vue CLI：

```
npm i -g @vue/cli
```

```
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
```

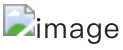
安装结束，启动开发服务：

```
# 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`：

```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`：

```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
```

风格指南

本项目的风格指南主要是参照 `vue` 官方的风格指南。在真正开始使用该项目之前建议先阅读一遍指南，这能帮助让你写出更规范和统一的代码。当然每个团队都会有所区别。其中大部分规则也都配置在了eslint-plugin-vue之中，当没有遵循规则的时候会报错，详细内容见eslint章节。

当然也有一些特殊的规范，是不能被 eslint 校验的。需要人为的自己注意，并且来遵循。最主要的就是文件的命名规则，这里拿`vue-element-admin`来举例。

Component

所有的`Component`文件都是以大写开头 (PascalCase)，这也是官方所推荐的。

但除了 `index.vue`。

例子：

- `@/components/BackToTop/index.vue`
- `@/components/Charts/Line.vue`
- `@/views/example/components/Button.vue`

JavaScript 文件

所有的`.js`文件都遵循横线连接 (kebab-case)。

例子：

- `@/utils/open-window.js`
- `@/views/svg-icons/require-icons.js`
- `@/components/MarkdownEditor/default-options.js`

Views

在 `views` 文件下，代表路由的 `.vue` 文件都使用横线连接 (kebab-case)，代表路由的文件夹也是使用同样的规则。

例子：

- `@/views/svg-icons/index.vue`
- `@/views/svg-icons/require-icons.js`

使用横线连接 (kebab-case)来命名`views`主要是出于以下几个考虑。

- 横线连接 (kebab-case) 也是官方推荐的命名规范之一 文档
- `views`下的`.vue`文件代表的是一个路由，所以它需要和`component`进行区分(component 都是大写开头)
- 页面的`url` 也都是横线连接的，比如`https://www.xxx.admin/export-excel`，所以路由对应的`view`应该要保持统一
- 没有大小写敏感问题

代码规范和 ESLint

不管是多人合作还是个人项目，代码规范都是很重要的。这样做不仅可以很大程度地避免基本语法错误，也保证了代码的可读性。

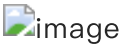
这里主要说明以下几点：

- 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

代码规范介绍

新手写的代码：

有经验的同学写的代码：



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

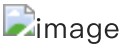
没有绝对的标准，下面是一些大厂商根据多数开发者的编码习惯制定的一些编码规范，仅供参考。

- 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-typescrip
t
  ],
  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：

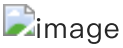
```
/* eslint "plugin1/rule1": "error" */
```

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。

vscode 配置 ESLint

这所谓工欲善其事，必先利其器，个人推荐 eslint+vscode 来写 vue，绝对有种飞一般的感觉。效果如图：

每次保存，vscode 就能标红不符合 eslint 规则的地方，同时还会做一些简单的自我修正。安装步骤如下：

 image

安装并配置完成 ESLint 后，我们继续回到 VSCode 进行扩展设置，依次点击 文件 > 首选项 > 设置 打开 VSCode 配置文件,添加如下配置

```
{
  "files.autoSave": "off",
  "eslint.validate": [
    "javascript",
    "javascriptreact",
    "vue-html",
    {
      "language": "vue",
      "autoFix": true
    }
  ],
  "eslint.run": "onSave",
  "eslint.autoFixOnSave": true
}
```

自动修复

```
npm run lint -- --fix
```

Git Hooks

布局

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
```

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

初始化路由页面组件

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

```
<template>
  <el-container class="wrapper">
    <app-sidebar />
    <el-container direction="vertical">
      <app-navbar />
      <el-main>
        <router-view />
      </el-main>
    </el-container>
  </el-container>
</template>

<script lang="ts">
import Vue from 'vue'
import Component from 'vue-class-component'
import AppSidebar from './components/sidebar.vue'
import AppNavbar from './components/navbar.vue'

@Component({
  components: {
```

```
    AppNavbar,
    AppSidebar
  }
})
export default class LayoutIndex extends Vue {
}
</script>

<style>
.wrapper {
  min-height: 100vh;
}
</style>
```

`sidebar.vue`

```
<template>
  <el-aside class="sidebar" width="auto">
    <!-- 如果已收起，则作用 collapse 类名 -->
    <router-link
      class="logo"
      :to="{ name: 'home' }"
      title="Back to Home"
    >
      <img src="@/assets/logo.png" alt="LagouEdu" />
      <h1>Edu Boss</h1>
    </router-link>
    <!-- 如果菜单已收起，则给 el-submenu 作用 collapse 属性 -->
    <el-menu router>
      <el-submenu index="1">
        <template slot="title">
          <i class="el-icon-lock"></i>
          <span>权限管理</span>
        </template>
        <el-menu-item index="1-1">
          <i class="el-icon-lock"></i>
          <span>角色列表</span>
        </el-menu-item>
```

```
        <el-menu-item index="1-2">
          <i class="el-icon-lock"></i>
          <span>菜单列表</span>
        </el-menu-item>
        <el-menu-item index="1-3">
          <i class="el-icon-lock"></i>
          <span>资源列表</span>
        </el-menu-item>
      </el-submenu>
      <el-menu-item index="2">
        <i class="el-icon-film"></i>
        <span slot="title">课程管理</span>
      </el-menu-item>
      <el-menu-item index="3">
        <i class="el-icon-user"></i>
        <span slot="title">用户管理</span>
      </el-menu-item>
      <el-submenu index="4">
        <template slot="title">
          <i class="el-icon-location"></i>
          <span>广告管理</span>
        </template>
        <el-menu-item index="4-1">
          <i class="el-icon-menu"></i>
          <span>广告列表</span>
        </el-menu-item>
        <el-menu-item index="4-2">
          <i class="el-icon-menu"></i>
          <span>广告位列表</span>
        </el-menu-item>
      </el-submenu>
    </el-menu>
  </el-aside>
</template>

<script lang="ts">
import Vue from 'vue'
import Component from 'vue-class-component'

@Component
```

```
63 export default class AppSidebar extends Vue {
64 }
65 </script>
66
67 <style lang="scss" scoped>
68 .sidebar {
69   position: sticky;
70   top: 0;
71   max-height: 100vh;
72   background-color: $sidebar-bg;
73   user-select: none;
74
75   &::-webkit-scrollbar {
76     width: 5px;
77   }
78
79   &::-webkit-scrollbar-track {
80     border-radius: 2em;
81   }
82
83   &::-webkit-scrollbar-thumb {
84     background-color: rgba(0, 0, 0, 0.1);
85     border-radius: 2em;
86   }
87
88   .logo {
89     display: flex;
90     justify-content: center;
91     align-items: center;
92     color: #495057;
93     line-height: 50px;
94     text-decoration: none;
95     text-align: center;
96
97     &:hover {
98       color: #343a40;
99       background-color: rgba(0, 0, 0, 0.05);
100    }
101
102    img {
```

```
      margin: 10px;
      width: 30px;
    }

    h1 {
      display: inline-block;
      margin: 0;
      width: 95px;
      overflow: hidden;
      font-size: 20px;
      white-space: nowrap;
      transition: width 0.3s;
    }

    &.collapse {
      h1 {
        width: 0;
      }
    }
  }

  .el-menu {
    border-right: 0 !important;
    background-color: transparent;

    &:not(.el-menu--collapse) {
      width: 200px;
    }

    .el-menu-item:hover,
    .el-menu-item:focus,
    .el-submenu__title:hover {
      background-color: $body-bg;
    }
  }
}
</style>
```

`navbar.vue`

```vue
<template>
  <el-header class="navbar" height="auto">
    <!-- :icon="sidebar.collapse ? 'el-icon-s-unfold' : 'el-icon-s-fold'" -->
    <el-button
      class="hamburger"
      type="text"
      icon="el-icon-s-unfold"
    ></el-button>
    <el-breadcrumb separator="/" replace>
      <el-breadcrumb-item :to="{ name: 'Home' }">Home</el-breadcrumb-item>
      <el-breadcrumb-item>Item 1</el-breadcrumb-item>
      <el-breadcrumb-item>Item 2</el-breadcrumb-item>
    </el-breadcrumb>
    <el-dropdown>
      <img
        class="avatar"
        src="https://edu-lagou.oss-cn-beijing.aliyuncs.com/images/2020/07/10/15943594999396473.png"
      />
      <el-dropdown-menu slot="dropdown">
        <el-dropdown-item command="profile">Admin</el-dropdown-item>
        <el-dropdown-item command="logout" divided>登出</el-dropdown-item>
      </el-dropdown-menu>
    </el-dropdown>
  </el-header>
</template>

<script lang="ts">
import Vue from 'vue'
import Component from 'vue-class-component'

@Component
export default class AppNavbar extends Vue {
}
```

```
</script>

<style lang="scss" scoped>
.navbar {
  position: sticky;
  top: 0;
  z-index: 100;
  display: flex;
  align-items: center;
  padding: 0 !important;
  background-color: $navbar-bg;

  .hamburger {
    margin-right: 10px;
    padding: 15px;
    font-size: 20px;
    border: 0;
    border-radius: 0;

    &:hover {
      background-color: rgba(0, 0, 0, 0.1);
    }
  }

  .el-dropdown {
    margin-left: auto;
  }

  .avatar {
    display: block;
    margin: 10px;
    width: 30px;
    height: 30px;
    border-radius: 50%;
    box-sizing: border-box;
  }
}
</style>
```

路由和侧边栏

路由懒加载

当打包构建应用时，Javascript 包会变得非常大，影响页面加载速度。如果我们能把不同路由对应的组件分割成不同的代码块，然后当路由被访问的时候才加载对应组件，这样就更加高效了。

结合 Vue 的异步组件和 Webpack 的代码分割功能，轻松实现路由组件的懒加载。如：

```
const Foo = () => import(/* webpackChunkName: 'foo' */ './Foo.vue'
)
```

当你的项目页面越来越多之后，在开发环境之中使用 `lazy-loading` 会变得不太合适，每次更改代码触发热更新都会变得非常的慢。所以建议只在生产环境之中使用路由懒加载功能。

1. 安装依赖 `npm install babel-plugin-dynamic-import-node -S -D`
2. 在 `babel.config.js` 中添加插件

```
module.exports = {
  presets: ['@vue/cli-plugin-babel/preset'],
  env: {
    development: {
      plugins: ['dynamic-import-node']
    }
  }
}
```

> 提示：`webpack5` 即将发布，大幅提高了打包和编译速度，之后可能完全不需要搞这么复杂了，再多的页面热更新，都能很快，完全就不需要前面提到的解决方案了。

登录页面

错误页面

错误处理

页面

404

页面级的错误处理由 `vue-router` 统一处理，所有匹配不到正确路由的页面都会进 `404` 页面。

```
// Vue 内部会把 * 放到所有路由规则之后，所以这个规则写到哪里都可以，但我们建议把它写到最后
{ path: '*', redirect: '/404' }
```

401

在 `@/permission.js` 做了权限控制，所有没有权限进入该路由的用户都会被重定向到 `401` 页面。

请求

项目里所有的请求都会走`@/utils/request.js`里面创建的的 axios 实例，它统一做了错误处理，完整代码。

你可以在`service.interceptors.response` response 拦截器之中根据自己的实际业务统一针对不同的状态码或者根据自定义 code 来做错误处理。如：

```
service.interceptors.response.use(
  response => {
    /**
     * code为非20000是抛错 可结合自己业务进行修改
     */
    const res = response.data
    if (res.code !== 20000) {
      Message({
        message: res.data,
```

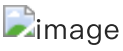
```
        type: 'error',
        duration: 5 * 1000
      })

      // 50008:非法的token; 50012:其他客户端登录了;  50014:Token 过期了;
      if (res.code === 50008 || res.code === 50012 || res.code === 50014) {
        MessageBox.confirm(
          '你已被登出，可以取消继续留在该页面，或者重新登录',
          '确定登出',
          {
            confirmButtonText: '重新登录',
            cancelButtonText: '取消',
            type: 'warning'
          }
        ).then(() => {
          store.dispatch('FedLogOut').then(() => {
            location.reload() // 为了重新实例化vue-router对象 避免bug
          })
        })
      }
      return Promise.reject('error')
    } else {
      return response.data
    }
  },
  error => {
    console.log('err' + error) // for debug
    Message({
      message: error.message,
      type: 'error',
      duration: 5 * 1000
    })
    return Promise.reject(error)
  }
)
```

因为所有请求返回的是 `promise`，所以你也可以对每一个请求通过`catch` 错误，从而进行单独的处理。

```
getInfo()
  .then(res => {})
  .catch(err => {
    xxxx
  })
```

代码

本项目也做了代码层面的错误处理，如果你开启了 `eslint` 在编写代码的时候就会提示错误。如：



当然还有很多不能被`eslint`检查出来的错误，vue 也提供了全局错误处理钩子errorHandler，所以本项目也做了相对应的错误收集。

样式处理

CSS Modules

目录结构

自定义 element-ui 样式

父组件改变子组件样式 深度选择器

建议你使用`::v-deep`的写法，它不仅兼容了 css 的`>>>`写法，还兼容了 sass `/deep/`的写法。而且它还是 vue 3.0 RFC 中指定的写法。

而且原本 `/deep/` 的写法也本身就被 Chrome 所废弃，你现在经常能在控制台中发现 Chrome 提示你不要使用`/deep/`的警告。

```
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`

```
@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.sc
ss

// 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%;
```

```
50
51   &-primary {
52     background: $--color-primary;
53   }
54
55   &-success {
56     background: $--color-success;
57   }
58
59   &-warning {
60     background: $--color-warning;
61   }
62
63   &-danger {
64     background: $--color-danger;
65   }
66
67   &-info {
68     background: $--color-info;
69   }
70 }
```

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

```
1 module.exports = {
2   ...
3   css: {
4     loaderOptions: {
5       sass: {
6         prependData: `@import "~@/styles/variables.scss";`
7       }
8     }
9   }
```

```
10 }
```

和服务端交互

基本逻辑

在 `vue-element-admin` 中，一个完整的前端 UI 交互到服务端处理流程是这样的：

1. UI 组件交互操作；
2. 调用统一管理的 api service 请求函数；
3. 使用封装的 request.js 发送请求；
4. 获取服务端返回；
5. 更新 data；

从上面的流程可以看出，为了方便管理维护，统一的请求处理都放在 `@/api` 文件夹中，并且一般按照 model 纬度进行拆分文件，如：

```
api/
  login.js
  article.js
  remoteSearch.js
  ...
```

服务端接口说明

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。

接口跨域问题

平时被问到最多的问题还是关于跨域的，其实跨域问题真的不是一个很难解决的问题。这里我来简单总结一下我推荐的几种跨域解决方案。

我最推荐的也是我工作中在使用的方式就是： `cors` 全称为 Cross Origin Resource Sharing（跨域资源共享）。这种方案对于前端来说没有什么工作量，和正常发送请求写法上没有任何区别，工作量基本都在后端这里。每一次请求，浏览器必须先以 `OPTIONS` 请求方式发送一个预请求（也不是所有请求都会发送 options，展开介绍 点我），通过预检请求从而获知服务器端对跨源请求支持的 `HTTP` 方法。在确认服务器允许该跨源请求的情况下，再以实际的 `HTTP` 请求方法发送那个真正的请求。推荐的原因是：只要第一次配好了，之后不管有多少接口和项目复用就可以了，一劳永逸的解决了跨域问题，而且不管是开发环境还是正式环境都能方便的使用。详细 MDN 文档

但总有后端觉得麻烦不想这么搞，那纯前端也是有解决方案的。

在 `dev` 开发模式下可以下使用 webpack 的 `proxy` 使用也是很方便，参照 文档 就会使用了，楼主一些个人项目使用的该方法。但这种方法在生产环境是不能使用的。在生产环境中需要使用 `nginx` 进行反向代理。不管是 `proxy` 和 `nginx` 的原理都是一样的，通过搭建一个中转服务器来转发请求规避跨域的问题。

开发环境	生产环境
cors	cors
proxy	nginx

这里我只推荐这两种方式跨域，其它的跨域方式都还有很多但都不推荐，真心主流的也就这两种方式。

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'
import { 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.BASE_API, // api的base_url
  timeout: 5000 // 请求超时时间
})

// request拦截器
service.interceptors.request.use(config => {
  // Do something before request is sent
  if (store.getters.token) {
    config.headers['X-Token'] = getToken() // 让每个请求携带token--
```

```
['X-Token']为自定义key 请根据实际情况自行修改
  }
  return config
}, error => {
  // Do something with request error
  console.log(error) // for debug
  Promise.reject(error)
})

// respone拦截器
service.interceptors.response.use(
  response => response,
  /**
  * 下面的注释为通过response自定义code来标示请求状态，当code返回如下情况为权限有问题，登出并返回到登录页
  * 如通过xmlhttprequest 状态码标识 逻辑可写在下面error中
  */
  //  const res = response.data;
  //     if (res.code !== 20000) {
  //       Message({
  //         message: res.message,
  //         type: 'error',
  //         duration: 5 * 1000
  //       });
  //       // 50008:非法的token; 50012:其他客户端登录了;  50014:Token过期了;
  //       if (res.code === 50008 || res.code === 50012 || res.code === 50014) {
  //         MessageBox.confirm('你已被登出，可以取消继续留在该页面，或者重新登录', '确定登出', {
  //           confirmButtonText: '重新登录',
  //           cancelButtonText: '取消',
  //           type: 'warning'
  //         }).then(() => {
  //           store.dispatch('FedLogOut').then(() => {
  //             location.reload();// 为了重新实例化vue-router对象 避免bug
  //           });
  //         })
  //       }
```

```
  //       return Promise.reject('error');
  //     } else {
  //       return response.data;
  //     }
  error => {
    console.log('err' + error)// for debug
    Message({
      message: error.message,
      type: 'error',
      duration: 5 * 1000
    })
    return Promise.reject(error)
  })

export default service
```

使用示例：

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

//使用
export function getInfo(params) {
  return request({
    url: '/user/info',
    method: 'get',
    params
  });
}
```

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
```