`


愿景："让编程不在难学，让技术与生活更加有趣" 更多课程请访问xdclass.net

Vue.js devtools开发工具安装教程

- 浏览器商店直接安装

  Chrome Extension(谷歌浏览器)

  https://chrome.google.com/webstore/detail/vuejs-devtools/nhdogjmejiglipccpnnnanhbledajbpd

  Firefox Addon(火狐浏览器)

  https://addons.mozilla.org/en-US/firefox/addon/vue-js-devtools

  Workaround for Safari(Safari浏览器)

  https://github.com/vuejs/vue-devtools/blob/master/docs/workaround-for-safari.md

- (不翻墙，无法访问谷歌的)手动安装教程(需要安装node)：

  https://www.cnblogs.com/chenhuichao/p/11039427.html

第一章 课程介绍

第1集 小滴后台管理系统课程介绍

- 课前准备
  - node
  - `vue.js devtools`
- 基础部分讲解
- element组件库常用部分讲解
- 封装组件的思路
- EChart运用
- 权限管理之动态生成菜单
- 路由守卫判断`token`是否存在，跳转对应页面

第二章 Vue全家桶各部分核心知识详解

第1集 构建vue项目的利器—脚手架vue-cli3详解

简介：vue-cli3全方位对比vue-cli2

参考地址： https://cli.vuejs.org/zh/config/#vue-config-js

- 安装node
  - 选择对应系统进行下载，下载完成后直接安装即可

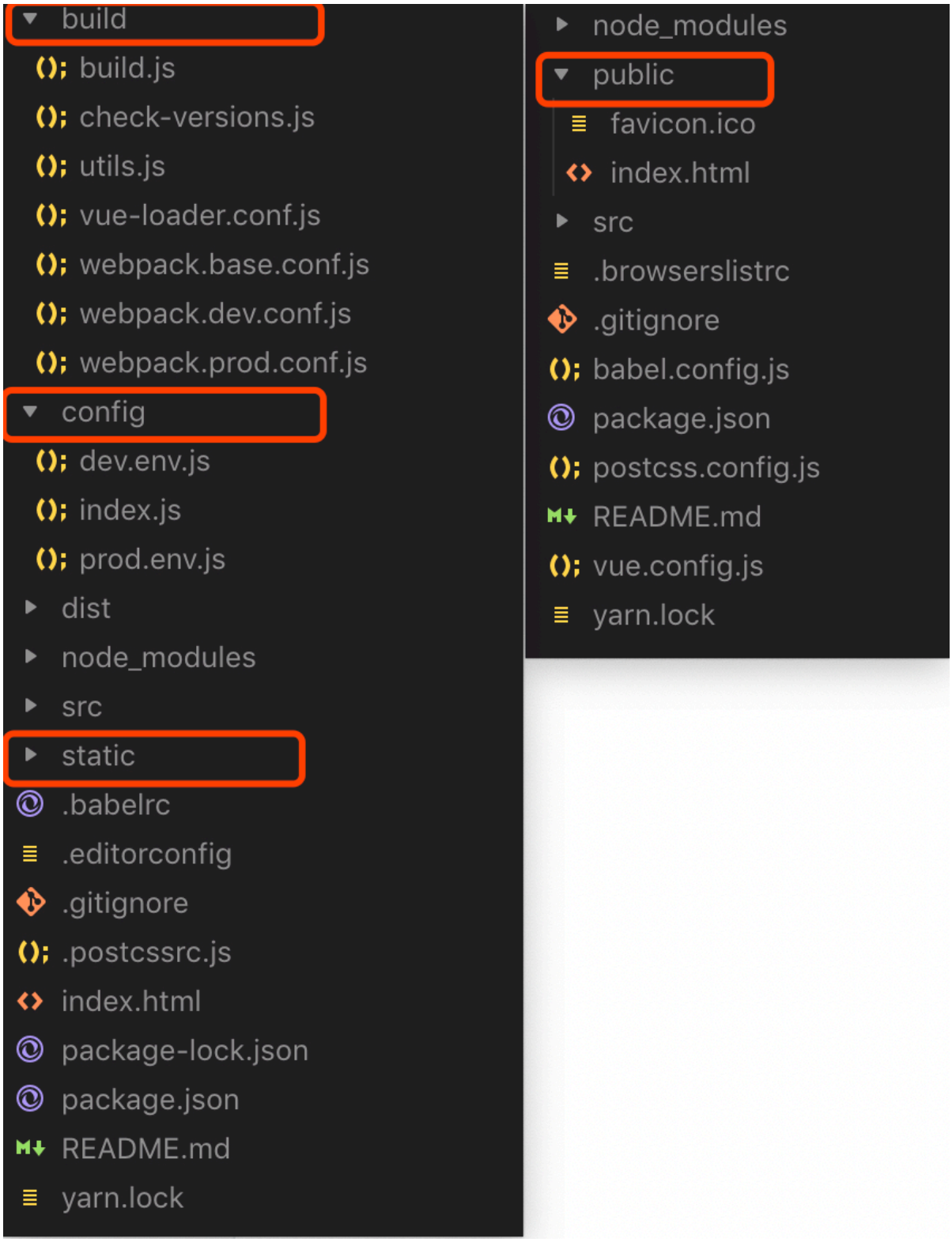
    http://nodejs.cn/download/

    ```
    // 输入一下命令，成功输出版本即为安装成功
    node -v
    ```

- 安装vue-cli3

  ```
  npm install -g @vue/cli

  // 输出版本号即为安装成功
  vue -V
  ```

- 创建项目
  - vue create project 或 可视化创建 vue ui
  - vue init webpack project
- CLI服务
  - 脚本名更改
  - 启动服务的依赖更改
    - CLi3启动方式是vue-cli-service serve
    - CLi2启动方式是webpack-dev-server --inline --progress --config
- 生成的目录结构介绍



- 3.0的目录简单了很多，少了build、config两个目录。需要对webpack进行配置的话，要手动在根目录新建一个vue.config.js文件
- vue.config.js常用配置

```
// vue.config.js 常用配置
module.exports = {
    // 基本路径, vue.cli 3.3以前请使用baseUrl
    publicPath: '/',
    // 输出文件目录
    outputDir: 'dist',
```

```
    // 用于嵌套生成的静态资产（js，css，img，fonts）的目录。
    assetsDir: '',
    // 生产环境sourceMap
    productionSourceMap: true,
    // webpack配置
    configureWebpack: () => {},
    chainWebpack: () => {},
    // css相关配置
    css: {
     // 启用 CSS modules
     modules: false,
     // 是否使用css分离插件
     extract: true,
     // 开启 CSS source maps?
     sourceMap: false,
     // css预设器配置项
     loaderOptions: {},
    },
    // webpack-dev-server 相关配置
    devServer: {
     host: '0.0.0.0',
     port: 8080,
     proxy: {}, // 设置代理
    },
    // 第三方插件配置
    pluginOptions: {
     // ...
    }
}
```

第2集 vue中组件间传值常用的几种方式(上)

- 父子组件传值
  - props / $emit
    - 子组件中通过定义props接收父组件中通过v-bind绑定的数据
    - 父组件中通过监听子组件中$emit的自定义事件接收数据
  - $parent / children
    - 子组件中通过this.$parent这个对象获取父组件中的数据
    - 父组件中通过this.$children这个数组获取子组件中的数据
  - $ref
    - 父组件中定义子组件中的ref属性后，通过*this*.$refs.定义的属性名获取子组件数据

第3集 vue中组件间传值常用的几种方式(下)

- 非父子间传值
  - 事件总线

```
// 原理上就是建立一个公共的js文件，专门用来传递消息
// bus.js
import Vue from 'vue'
export default new Vue;

// 在需要传递消息的地方引入
import bus from './bus.js'
// 传递消息
bus.$emit('msg', val)
// 接受消息
bus.$emit('msg', val => {
  console.log(val)
})
```

  - $attrs / listeners

```
// 解决多级组件间传值的问题

// $attr 将父组件中不包含props的属性传入子组件，通常配合 interitAttrs 选项一起使用。
// 如果不想在dom上出现属性，可设置interitAttrs: false
// $listeners监听子组件中数据变化，传递给父组件
```

```

```

  - vuex

第4集 玩转单页面应用的控制中心—vue-router

- 路由的基本配置
  - 基本参数
    - path

      路由的访问路径。即url
    - component

      访问路径对应的组件
  - 扩展参数
    - name

      路由指定命名，设置后可用params传参及使用name进行路由跳转
- 路由的跳转
  - router-link标签跳转
  - 编程式导航

```
// route可以是对象，或者是字符串
// 对象的时候可通过路由的path或者name进行跳转
// 字符串的话只能是路由的path
this.$router.push(route)

// 路由传递参数，query和path配合， params和name配合
query:  this.$router.push({path: '/', query: {id: 2}})
params: this.$router.push({name: 'home', params: {id: 2}})
```

- 动态路由
  - 什么是动态路由
    - 组件是同一个，只是通过不同的url参数渲染不同的数据
    - 路径参数"使用冒号"`:`标记

```
{
  path: '/home/:id',
  component: home
}
```

    - 在path里显式声明后，通过params传参后，参数不丢失同时参数被设置成必传参数

- 嵌套路由
    - 目的： 组件中嵌套不同组件
    - 实现

```
// 在需要嵌套的路由中补充children字段
{
  path: '/home/:id',
  component: home,
  children: []
}
```

- 导航守卫
    - 通过router中的beforeEach注册全局守卫，每次切换路由时触发

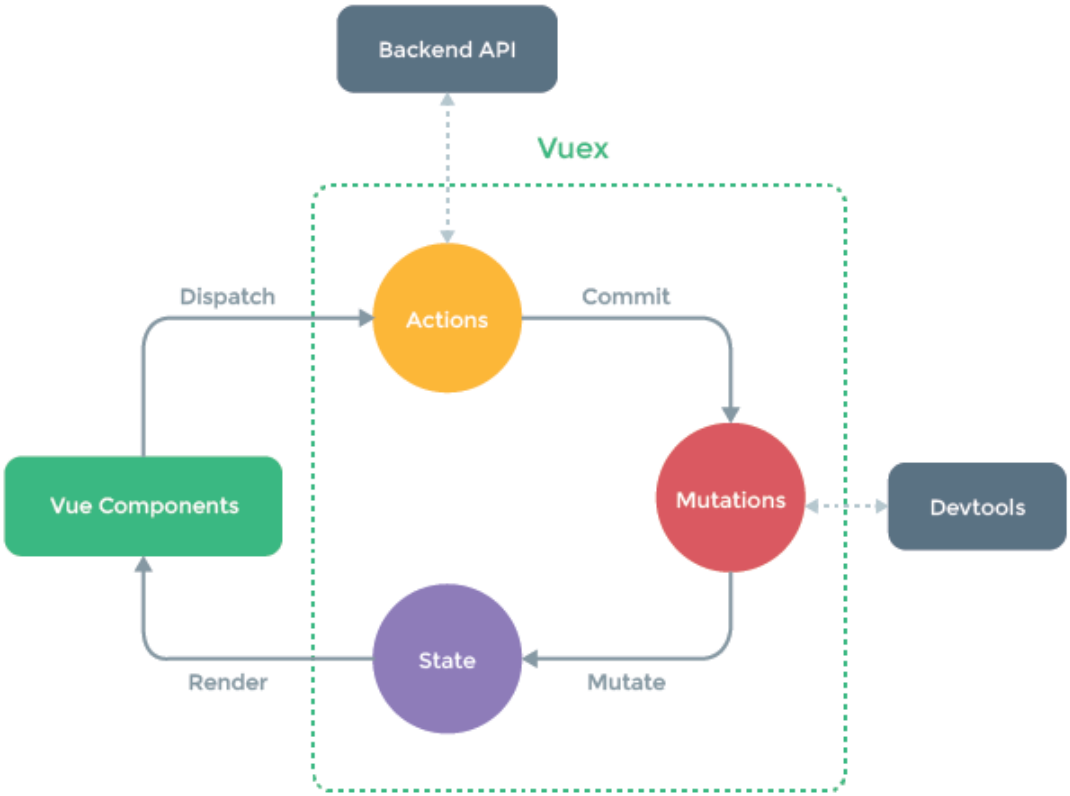
```
// to, from是路由对象，我们在路由里定义的参数都可以在这里取到，例如to.path或from.name
router.beforeEach((to, from, next) => {
  // ...
  next()
})
```

    - 参数
        - to： 将进入的路由对象
        - from： 将离开的路由对象
        - next() 确认完成操作，最后一定要调用，不然路由就不会进行切换
- 路由懒加载
    - 提高页面加载速度
    - 避免进入项目后加载全部组件
    - 在路由中的component中设置函数，用import方式进行使用

```
component: () => import('./views/Home.vue'),
```

第5集 状态管理中心—vuex的基础用法

- State
  - 数据，存放一些公用部分的数据
- Mutations
  - 数据怎么改变，定义改变state的一些方法
- Actions
  - 异步改变， 如果需要异步改变state，则在这书写



- vuex里包含的基本参数

```
export default {
  // 组件间公共数据部分
  state: {},
  // 需要改变state中的数据时，要在mutation里定义改变的方法
  mutations: {},
  // 当改变state中的数据是异步操作时，在action里定义
  actions: {}
}
```

第6集 状态管理中心—vuex的高级用法

- vuex中的计算属性—Getters
  - 当你需要依赖vuex里的state中的数据，做进一步处理时使用

```
state: {
  count: 0,
},
// 根据state中的count进一步处理，计算双倍值
getters: {
  doubleCount (state) {
    return state.count * 2
  }
},
```

- 模块化概念—Modules
  - 当vuex里的数据十分庞大时，可根据存放的数据所属模块进行划分

```
import Vue from 'vue'
import Vuex from 'vuex'
// 第一步 引入模块
import text from './text'

Vue.use(Vuex)

// 第二步 在初始化store时，加载模块
export default new Vuex.Store({
  modules: {
    text
  }
})
```

第三章 Element常用组件详解

第1集 Element常用组件之布局组件详解

- 布局组件
  - el-row、el-col
  - el-container、el-header、el-aside、el-main、el-footer

- 常用属性
  - span
  - gutter

- 如何构建5等分布局

- 布局小试

第2集 Element常用组件之弹出类型组件详解

- 常用弹出组件
  - el-dialog
  - el-popover
- sync修饰符的作用
  - 示例：ElementUI中的el-dialog

```
// 第一种写法
<el-dialog :visible.sync="dialogVisible">

// 第二种写法
<el-dialog :visible="dialogVisible" :before-close="beforeClose">

// 第一种写法关闭或是点击空白处无需特别处理，el-dialog组件内部会修改当前值状态，通过.sync修饰符传递给父组件；
// 第二种写法，需要再beforeClose方法内手动处理this.dialogVisible = false。
```

- dialog组件中的插槽
  - title
  - footer

第3集 Element常用组件之表格组件详解

- 基础表格

  在 Table 组件中，每一个表格由一个 Table-Column 组件构成，也就是表格的列

- 表格常用属性介绍

属性名	作用
height	给表格设置高度，同时固定表头
show-header	设置是否显示表头
row-class-name	设置一个函数或者固定的名字作为行的类名
border	是否显示表格竖直方向的边框，设置后可通过改变边框设置列宽

- 列常用属性介绍

属性名	作用
label	当前列的表头名称
prop	传入的表格json数据的key值
show-overflow-tooltip	是否设置文字超出列宽时悬浮显示完整内容

- 通过v-for封装适用性更好的表格

第4集 Element常用组件之表单组件详解

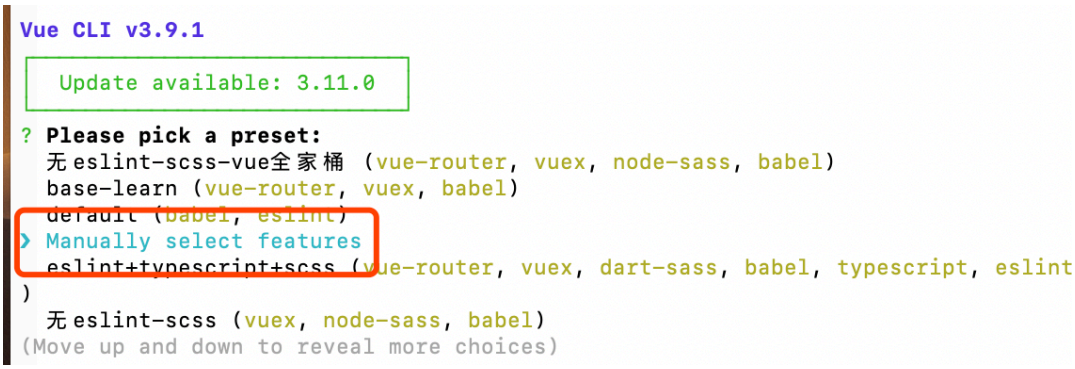
- 基础表单
  - 在 Form 组件中，每一个表单域由一个 Form-Item 组件构成，Form-item可以是下拉框、输入框、日期选择器等各种表单组件
- 添加表单验证

- 封装表单组件
  - 观察基础表单
    - 总结一个表单组件动态的参数有哪些
      - 最基础
        - label，model、type
      - 扩展
        - rule、placeholder、其他配置(自动补全，可清除等)
  - 定义循环的数据结构
    - 数组对象

第四章 实战项目之环境准备及配置改装

第1集 项目搭建及技术选型

- 通过命令行创建项目
  - Vue create <项目名>
  - 手动配置项目，上下键移动光标后，回车即可

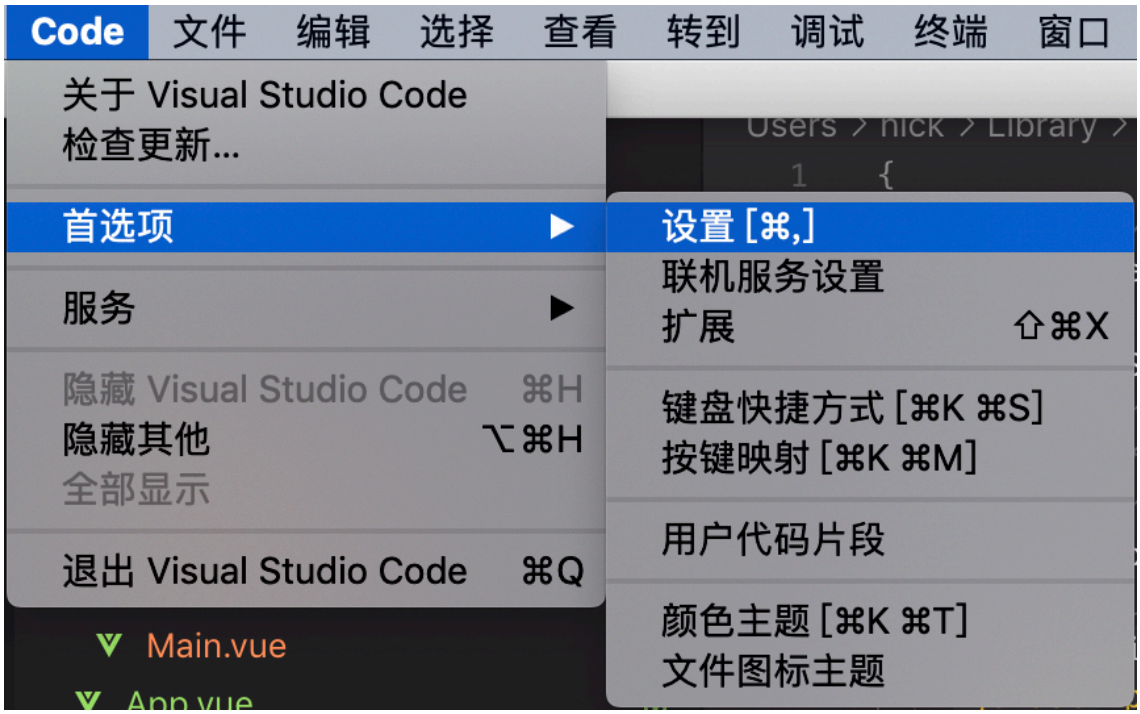
- 

  - 选择需要的配置
    - babel转译js的新特性，兼容低版本浏览器
    - CSS预处理器，设置全局变量
    - ESLint检查代码写法是否规范
    - 

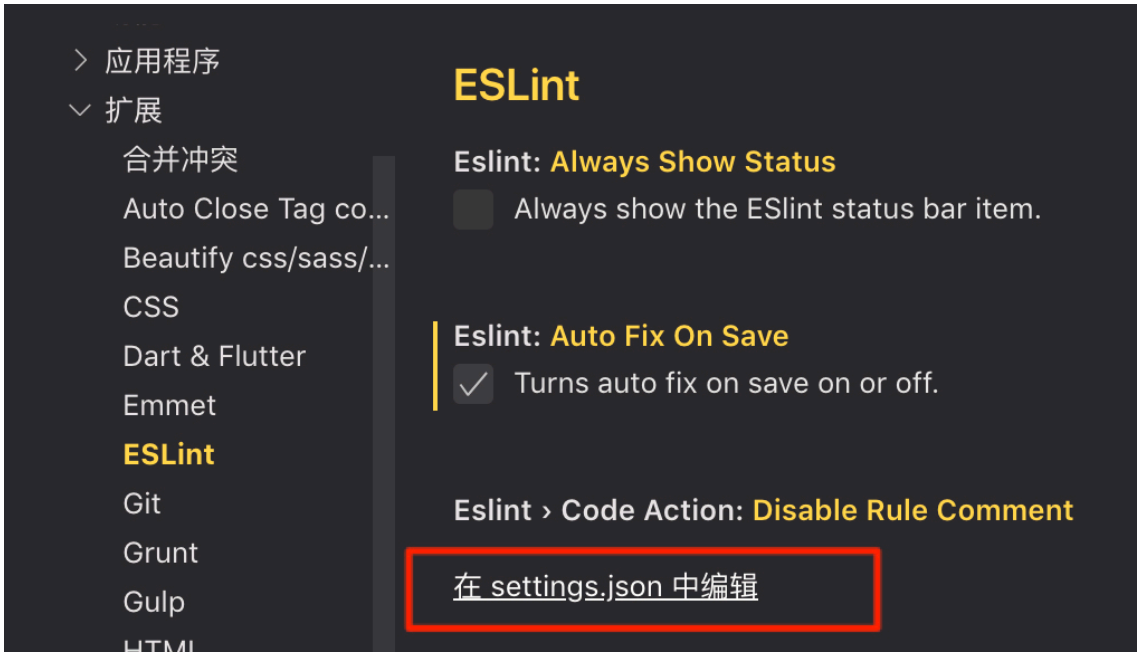
  - 是否使用

第2集 配置项目的基本环境及项目目录结构总体介绍

- 配置项目按eslint规范格式化代码
  - vscode下载`ESlint`，`Prettier`,`Vetur`插件
  - 打开vscode的设置



- 选择扩展中的ESLint，之后点击在setting.json中编辑



- 添加如图配置

```
// #每次保存的时候将代码按eslint格式进行修复
"eslint.autoFixOnSave": true,
// 添加 vue 支持
"eslint.validate": [
  "javascript",
  "javascriptreact",
  {
    "language": "vue",
    "autoFix": true
  }
],
```

```
// 每次保存的时候将代码按eslint格式进行修复
"eslint.autoFixOnSave": true,
  // 添加 vue 支持
  "eslint.validate": [
    "javascript",
    "javascriptreact",
    {
      "language": "vue",
      "autoFix": true
    }
  ],
```

- 可自定义eslint的一些规则
  - 在`.eslintrc`中覆盖prettier规则即可，覆盖是为了防止冲突
  - 在rules里配置

```
rules: {
    'no-console': process.env.NODE_ENV === 'production' ? 'error'
: 'off',
    'no-debugger': process.env.NODE_ENV === 'production' ? 'error'
: 'off',
    // 添加自定义规则
     'prettier/prettier': [
       // eslint校验不成功后，error或2则报错，warn或1则警告，off或0则无提
示
      'error',
      {
        singleQuote: true,
        semi: false
      }
    ]
  },
```

  - 配置完成后可运行`npm run lint`格式化全部文件，或者保存后自动格式化代码
  - router部分模块化
  - vuex部分模块化
  - 调整项目目录
  - 删除多余代码
- 项目目录结构调整
- 配置scss全局变量
  - 在项目的根目录下新建`vue.config.js`
  - 新建`_variable.scss`文件
  - 在`vue.config.js`文件进行如下配置

```
module.exports = {
  // 配置项目启动端口及自动打开浏览器
  devServer: {
    port: 3333,
    open: true
  },
  // 配置scss全局变量
  css: {
    loaderOptions: {
      sass: {
        data: `@import "@/assets/scss/_variable.scss";`
      }
    }
  }
}
```

第五章 小D课堂后台视频管理系统之公用部分开发

第1集 需求分析及模块划分

- 分析视频管理后台需要的功能
  - 可视化展示数据
    - 视频的成交量
    - 用户总量
    - 订单总额
  - 登录页
  - 视频管理
    - 上传视频
    - 更新视频
    - 删除视频
    - 查看已有视频
  - 用户管理
    - 更新用户信息
    - 删除用户
    - 新增用户
    - 权限管理
- 设计对应页面
  - 首页用来展示数据
    - 使用Echart柱状图、折线图及饼图展示
  - 视频管理页、用户管理页
    - 选用el-table及el-form展示和编辑数据
    - el-dialog组件实现编辑和新增功能

第2集 路由设计及左侧公用导航菜单开发

- 新建`Main.vue`组件放置公共部分组件

```
<template>
  <el-container>
    <el-aside><common-aside></common-aside></el-aside>
    <el-container>
      <el-header><common-header></common-header></el-header>
      <common-tab></common-tab>
      <el-main>
        <router-view />
      </el-main>
    </el-container>
  </el-container>
</template>

<script>
import CommonHeader from '../components/CommonHeader'
import CommonAside from '../components/CommonAside'
import CommonTab from '../components/CommonTab'
</script>
```

- 建立页面组件
  - 在`views/Home`下建立`Home.vue`
  - 在`views/UserManage`下建立`UserManage.vue`
  - 在`views/VideoManage`下建立`VideoManage.vue`
  - 在`views/Other`下建立`PageOne.vue`、`PageTwo.vue`
- 在`router/index.js`中引入路由

```
{
  // 主要配置3个参数,访问的路由地址，路由名字及懒加载加载哪个组件
  path: '',
  // name属性用于路由跳转及params传参
  name: '',
  component: ''
}
```

```
// 完整路由代码
export default new Router({
  routes: [
    {
      path: '/',
      component: () => import('@/views/Main'),
      children: [
        {
          path: '/',
          name: 'home'
          component: () => import('@/views/Home/Home'),
        },
        {
          path: '/user',
          name: 'user'
          component: () => import('@/views/UserManage/UserManage'),
        },
        {
          path: '/video',
          name: 'video'
          component: () => import('@/views/VideoManage/VideoManage'),
        },
        {
          path: '/page1',
          name: 'page1'
          component: () => import('@/views/Other/PageOne'),
        },
        {
          path: '/page2',
          name: 'page2'
          component: () => import('@/views/Other/PageTwo'),
        },
      ]
    }
  ]
})
```

第3集 顶部导航菜单及与左侧导航联动的面包屑实现(上)

- 核心点
  - 使用vuex进行传值

```
// tab.js
export default {
  state: {
    isCollapse: false,
    currentMenu: null,
    menu: [],
    tabsList: [
      {
        path: '/',
        name: 'home',
        label: '首页',
        icon: 'home'
      }
    ]
  },
  mutations: {
    selectMenu(state, val) {
      if (val.name !== 'home') {
        state.currentMenu = val
```

```
        let result = state.tabsList.findIndex(item => item.name === val.name)
        result === -1 ? state.tabsList.push(val) : ''
      } else {
        state.currentMenu = null
      }
      // val.name === 'home' ? (state.currentMenu = null) : (state.currentMenu
= val)
    },
  },
  actions: {}
}
```

第4集 顶部导航菜单及与左侧导航联动的面包屑实现(下)

vuex中的`tab.js`

```
// tab.js
export default {
  state: {
    isCollapse: false,
    currentMenu: null,
    menu: [],
    tabsList: [
      {
        path: '/',
        name: 'home',
        label: '首页',
        icon: 'home'
      }
    ]
  },
  mutations: {
    closeTab(state, val) {
      let result = state.tabsList.findIndex(item => item.name === val.name)
      state.tabsList.splice(result, 1)
    },
  },
  actions: {}
}
```

第5集 使用vuex实现切换tab页功能

- 在`Main.vue`中引入`CommonTab.vue`组件
- 在vuex里定义存取标签的`tagList`，方便非父子传递数据
- 定义`vuex`中侧边栏点击后将菜单加入到`tagList`中的方法

- 定义`vuex`中点击标签后触发删除的方法

第6集 构建页面组件，连通公共组件

- 建立每个页面组件
- 连通面包屑
- 连通侧边栏
- 连通标签栏

第7集 页面布局整体样式优化

- 侧边栏背景色改变
- tag选中样式优化
- 面包屑当前激活菜单样式优化
- 细节优化

第六章 小D课堂后台视频管理系统之首页开发

第1集 介绍mock.js及axios全局配置

- Mock.js
  - 作用：生成随机数据，拦截ajax请求
  - 安装：

```
yarn add mockjs

npm install mockjs
```

  - 核心：
    - Mock.mock()

```
// 根据数据模板生成模拟数据
Mock.mock( rurl, rtype, template)

/*
** rurl: 表示需要拦截的 URL，可以是 URL 字符串或 URL 正则
** eg. /\/domain\/list\.json/、'/domian/list.json'
*/
```

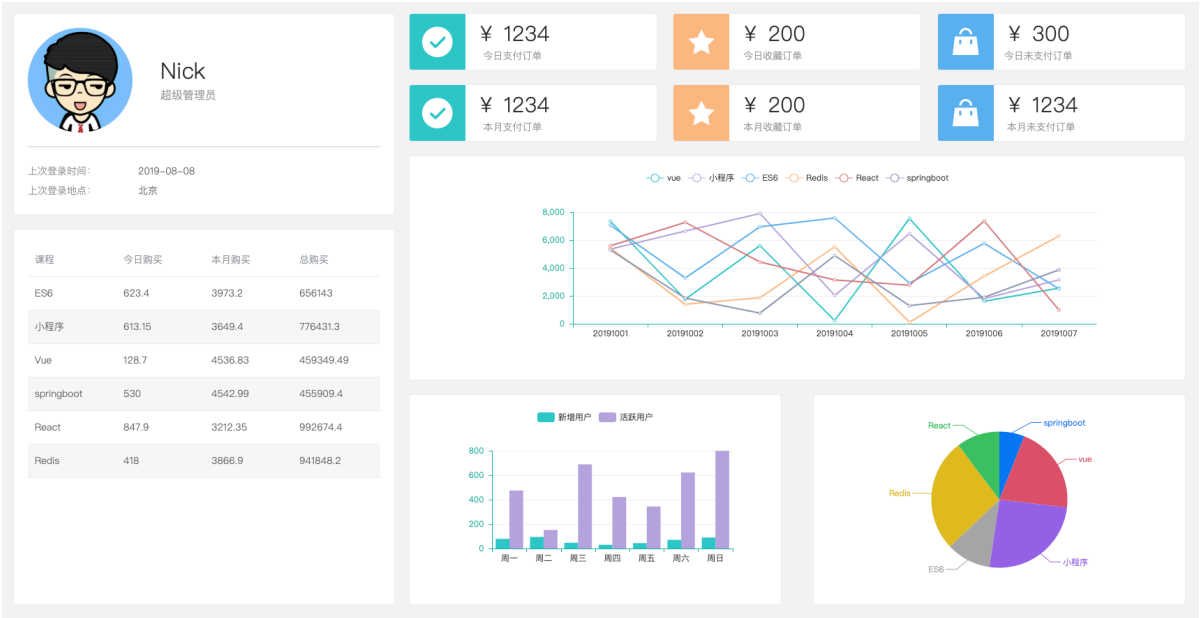
    - Mock.Random()
      - 随机生成数据

第2集 使用Mock随机返回主页数据

- 新建`mock`文件夹
- 配置mock请求时间
- 新建`home.js`存放主页的数据
- 安装mock的语法生成数据

第3集 使用element布局组件实现首页布局

- 首页截图



- 分析布局
- 选择组件

第4集 完成首页除图表外的内容

- 定义右上角统计数据格式

```
countData: [
  {
    name: '今日支付订单',
```

```
    value: 1234,
    icon: 'success',
    color: '#2ec7c9'
  },
  {
    name: '今日收藏订单',
    value: 210,
    icon: 'star-on',
    color: '#ffb980'
  },
  {
    name: '今日未支付订单',
    value: 1234,
    icon: 's-goods',
    color: '#5ab1ef'
  },
  {
    name: '本月支付订单',
    value: 1234,
    icon: 'success',
    color: '#2ec7c9'
  },
  {
    name: '本月收藏订单',
    value: 210,
    icon: 'star-on',
    color: '#ffb980'
  },
  {
    name: '本月未支付订单',
    value: 1234,
    icon: 's-goods',
    color: '#5ab1ef'
  }
],
```

- 通过v-for循环，渲染页面

```
<div class="num">
  <el-card shadow="hover" v-for="item in countData"
  :key="item.name" :body-style="{display: 'flex', padding: 0 }">
  <i class="icon" :class="`el-icon-${item.icon}`"
  :style="{ background: item.color }"></i>
  <div class="detail">
    <p class="num">¥ {{ item.value }}</p>
    <p class="txt">{{ item.name }}</p>
  </div>
  </el-card>
</div>
```

第5集 完成首页table部分及ECharts介绍

- ECharts
  - 一个使用 JavaScript 实现的开源可视化库，通过这个库可实现可视化展示数据
  - 快速入门
    - 先设置ECharts要在哪显示，设置EChart容器

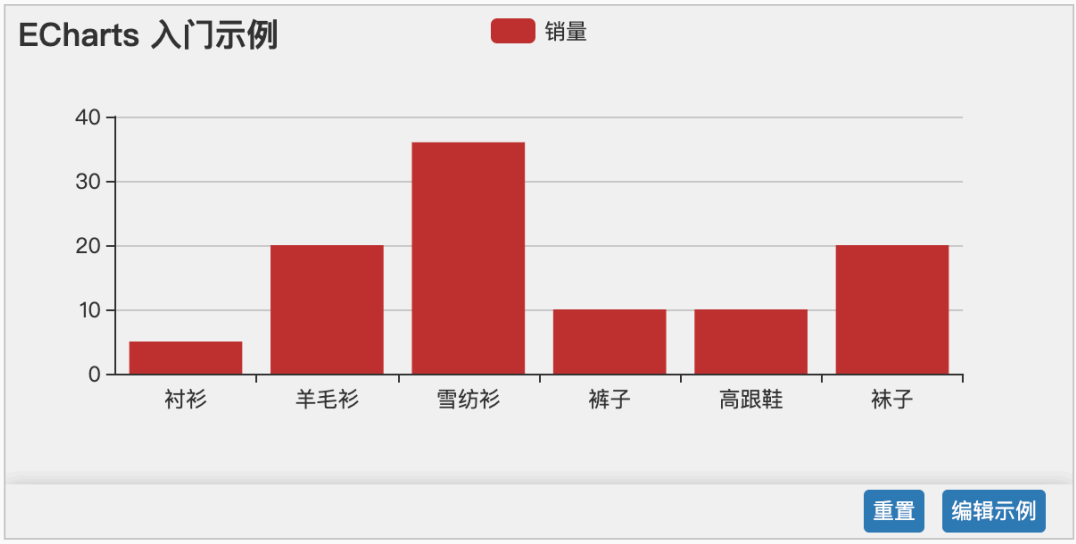
```
<body>
    <!-- 为 ECharts 准备一个具备大小（宽高）的 DOM -->
    <div id="main" style="width: 600px;height:400px;"></div>
</body>
```

    - 通过 echarts.init 方法初始化一个 echarts 实例并通过 setOption 方法生成一个简单的柱状图

```
// data中声明echart对象

// 通过ref获取dom，初始化echarts实例
this.echart = echarts.init(this.$refs.ref名字);

// 指定图表的配置项和数据
var option = {
  title: {
    text: 'ECharts 入门示例'
  },
  tooltip: {},
  legend: {
    data:['销量']
  },
  xAxis: {
    data: ["衬衫","羊毛衫","雪纺衫","裤子","高跟鞋","袜子"]
  },
  yAxis: {},
  series: [{
    name: '销量',
    type: 'bar',
    data: [5, 20, 36, 10, 10, 20]
  }]
};

// 使用刚指定的配置项和数据显示图表。
this.echart.setOption(option);
```

  - 案例
  -

这样你的第一个图表就诞生了！



第6集 谈谈封装一个EChart组件的一些想法

- 图表的大分类
- 组件的一些动态参数抽离
- 组件的基本配置
- 主题颜色

```
color: [
  '#2ec7c9',
  '#b6a2de',
  '#5ab1ef',
  '#ffb980',
  '#d87a80',
  '#8d98b3',
  '#e5cf0d',
  '#97b552',
  '#95706d',
  '#dc69aa',
  '#07a2a4',
  '#9a7fd1',
  '#588dd5',
  '#f5994e',
  '#c05050',
  '#59678c',
  '#c9ab00',
  '#7eb00a',
  '#6f5553',
  '#c14089'
]
```

第7集 上手封装一个EChart组件并处理数据展示图表

- 有坐标轴的图表
  - 数据格式

```
// 类目轴
xAxis: {
  data: [],
  type: 'category',
},

// 数据部分
yAxis: {
  type: 'value'
},
series: []
```

- 没坐标轴的图表
  - 数据格式

```
{
  series: []
}
```

第8集 修改EChart组件样式及自适应图表(上)

- 完善图表选项

```
      axisOption: {
        tooltip: {
          trigger: 'axis'
        },
        legend: {
          textStyle: {
            //图例文字的样式
            color: '#333'
          }
        },
        grid: {
          left: '20%'
        },
        color: [
          '#2ec7c9',
          '#b6a2de',
```

```
        '#5ab1ef',
        '#ffb980',
        '#d87a80',
        '#8d98b3',
        '#e5cf0d',
        '#97b552',
        '#95706d',
        '#dc69aa',
        '#07a2a4',
        '#9a7fd1',
        '#588dd5',
        '#f5994e',
        '#c05050',
        '#59678c',
        '#c9ab00',
        '#7eb00a',
        '#6f5553',
        '#c14089'
      ],
      xAxis: {
        data: [],
        type: 'category',
        axisLine: {
          lineStyle: {
            color: '#17b3a3' //x轴颜色
          }
        },
        // 字体倾斜
        axisLabel: {
          color: '#333'
        }
      },
      yAxis: {
        type: 'value',
        axisLabel: {
          formatter: '{value} '
        },
        axisLine: {
          lineStyle: {
            color: '#17b3a3' //坐标轴颜色
          }
        },
        splitLine: {
          show: true,
          lineStyle: {
            color: ['#eee'],
            width: 1,
            type: 'solid'
          }
```

```
        }
      },
      series: []
    },
```

第9集 修改EChart组件样式及自适应图表(下)

- 完善组件样式

```
normalOption: {
  tooltip: {
    trigger: 'item'
  },
    color: ['#0f78f4', '#dd536b', '#9462e5', '#a6a6a6', '#e1bb22',
'#39c362', '#3ed1cf'],
      series: []
}
```

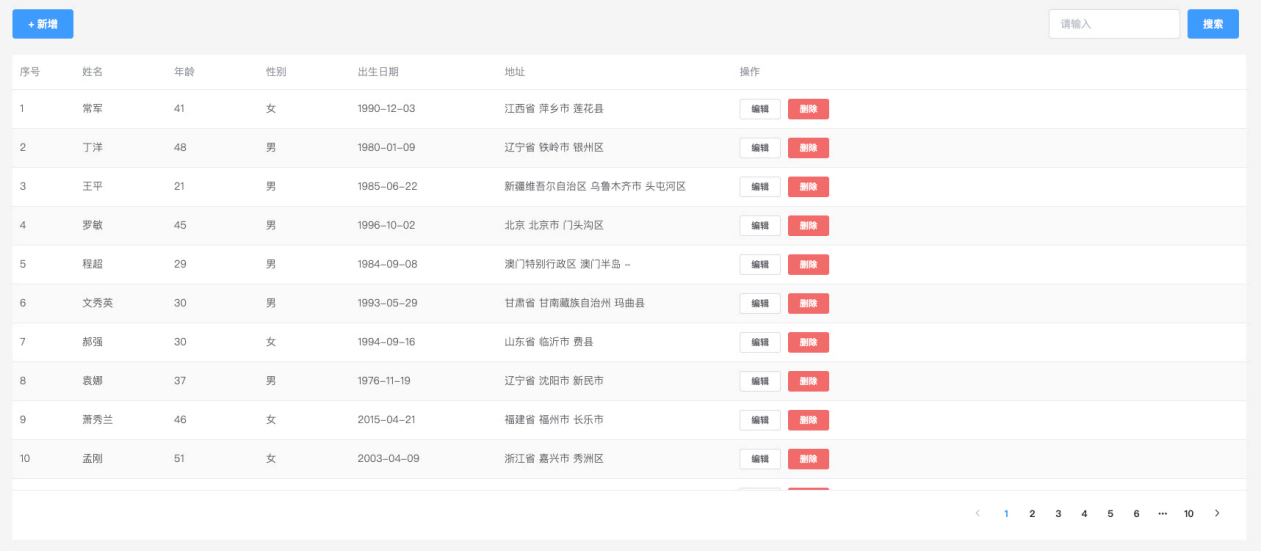
- 需要自适应的地方
  - 浏览器窗口大小发生变化
  - 折叠菜单的时候

第10集 Echart组件封装总结

- 观察文档，考虑组件需要的基本参数
- 参数筛选，分为从父组件传来的参数和自身的参数
- 完善组件，观察设计图，找不同，在文档中寻找对应的配置项
- 细节优化，考虑多种场景下，图表自适应的处理

第七章 用户管理页及详解权限管理

第1集 用户管理页介绍及页面实现思路讲解



- 管理页功能
  - 新增用户
  - 搜索用户
  - 更新用户
  - 删除用户

    - 分页展示用户列表
- 分析页面组成
- 考虑组件设计

第2集 更完善的表单组件封装及思路讲解

- 分析表单组成
    - 输入框

    - 下拉框
    - 日期选择器
    - 单选列表
    - 多选列表
    - 。。。
- 考虑基本参数
    - 绑定的表单字段
    - 表单描述文本
- 插槽拓展组件
    - 按钮组

第3集 通用表格组件封装及思路讲解

- 表格基本参数分析
  - data： 传入的数据列表
  - prop： 传入的数据字段
  - label： 列名
- 表格可选参数分析
  - width：列宽
  - type：类型
- 表格扩展
  - 分页参数
    - total： 数据条数总计
    - page： 当前页数
  - 加载状态
    - loading：布尔值

第4集 完成表格组件的封装

- 调整表格样式，固定表格高度
- 添加操作列

```
<el-table-column label="操作" min-width="180">
  <template slot-scope="scope">
   <el-button size="mini" @click="handleEdit(scope.row)">编辑</el-button>
   <el-button size="mini" type="danger" @click="handleDelete(scope.row)">删除</el-button>
  </template>
</el-table-column>
```

- 添加分页组件

```
<el-pagination
class="pager" layout="prev, pager, next"
:total="config.total" :current-page.sync="config.page"
@current-change="changePage">
</el-pagination>
```

第5集 用户管理页页面功能实现(上)

- 表格加载状态
- 分页功能
- 编辑功能
- 删除功能

第6集 用户管理页页面功能实现(下)

- 表格加载状态
  - elementUI中`v-loading`的使用
- 分页功能
- 编辑功能
- 删除功能

第7集 企业开发之权限管理思路讲解

- 什么是权限管理
  - 根据不同用户，返回不同菜单
  - 严格控制用户权限
  - 实现思路
    - 动态路由
    - 后端返回的数据格式要求
    - 触发时机
      - 登陆成功的时候触发操作
      - Cookie中存在对应数据，首次进入页面时

第8集 权限管理之动态返回菜单的实现

- 更改路由表
  - 根据是否需要权限的路由分类
- vuex里补充mutation
  - 保存菜单
  - 动态添加菜单
- 生成路由的时机
  - 登录时
  - 刷新时
- 点击退出时，清除cookie后，刷新下页面

第9集 权限管理之路由守卫判断用户登录状态

- `permission.js`中返回token
- 登录时保存token
  - 保存在vuex里
  - 保存在cookie里
- 路由守卫根据判断token存不存在，判断用户页面跳转

```
// 判断用户登录状态，未登录跳转到登录页，已登录跳转到首页
router.beforeEach((to, from, next) => {
  // 从cookie里获取token，防止刷新后token丢失
  store.commit('getToken')
  let token = store.state.user.token
  if (!token && to.name !== 'login') {
    next({ name: 'login' })
  } else {
    next()
  }
})
```


第八章 项目总结

第1集 小滴后台管理系统项目总结

- 项目当中遇到的坑以及解决思路

    - 通过vue-devtool调试
    - 通过console输出调试

- 组件的封装思路
    - 判断的基本参数
        - 哪些写死
        - 哪些是传进来
    - 拓展
        - 自定义事件，判断传出哪些参数
        - 插槽扩展
    - 优化
        - 提高他的适应性
            - vif，velse根据父组件传入的条件来生成对应的模板

- 学习一个新技术
    - EChart
        - 大局观，直接看快速教程
        - 分成几部分，在对应部分查找文档