# 一．环境配置

## （一）vue环境配置

安装vue脚手架

## （二）mysql环境

本地的mysql；

用户名：root

本地连接密码：root，注意黑马项目中的mysql密码配置的也是root

# 二、实现步骤

## （一）登录界面

### 1.布局



注意一开始测试接口时，login接口的参数是设置在body中的，而不是header之中！！！

### 2.项目架构

src中的main.js是项目的入口文件，导入vue，App组件，router路由，通过创建vue实例将App根组件渲染到了页面中，同时把路由router挂载到实例中。

App根组件中template是ui结构，script是行为，style是样式；

### 3.登录组件Login.vue

vue组件都是由三部分组成， 结构，行为，样式；

注意单文件组件的样式需要用scoped属性来限制组件的样式不会扩散到全局之中，仅仅是该组件中使用该样式。

<template>

    <div>

        登录组件

    </div>

</template>

<script>

export default {

}

</script>

<style lang="less" scoped>

</style>

创建好Login.vue组件后，到router.js中导入该组件（注意导入组件的路径一定要正确），并且创建相应的路由规则，如下所示，访问的是/login,则转到Login组件中：

const routes = [

  {

    path: '/login',

    component: Login

  }

]

最后在根组件App中添加一个路由占位符，能够将所有扫描到的路由组件都显示出来。

<template>

  <div id="app">

    <h1>APP根组件</h1>

    <!-- 路由占位符 -->

    <router-view></router-view>

  </div>

</template>

当然，如果用户访问的是/根目录，则会重定向到/login页面之中，因此需要修改路由规则：

const routes = [

  {

    path: '/',

    redirect: '/login'

  },

  {

    path: '/login',

    component: Login

  }

]

less相关依赖报错，版本过高，因此使用如下的版本：

图形化界面提供的less-loader和less版本过高。---卸载上两个依赖后，在项目文件夹终端执行 npm install less-loader@4.1.0 -D | 以及 npm install less@3.9.0 -D 即可安装正常依赖

此时查看显示页面中，可以看到组件并没有撑满屏幕，因此需要设置一个全局的样式，为html,body,#app设置样式：

/\* 全局样式表 \*/

html,body,#app {

    height: 100%;

    margin: 0;

    padding: 0;

}

然后在main.js入口中导入全局样式表：

// 导入全局 样式表

import './assets/css/global.css'

创建一个登陆盒子div，并且为它设置样式，将它放到页面中间，注意看样式中使用的是绝对位置，首先基于页面左上角进行偏移50%，再进行移动；

.login\_box {

    width: 450px;

    height: 300px;

    background-color: #fff;

    border-radius: 3px;

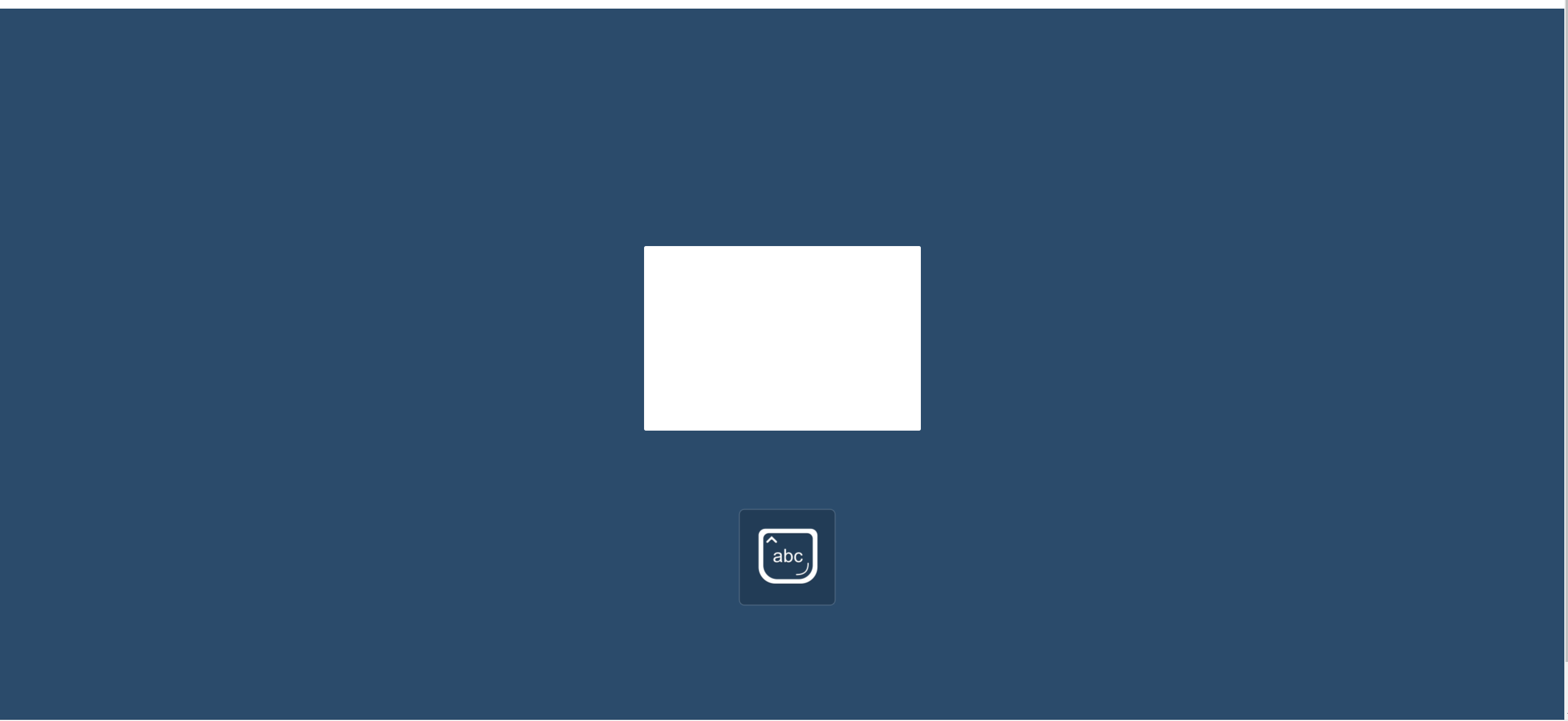
    position: absolute;

    left: 50%;

    top: 50%;

    transform: translate(-50%,-50%);

}



接下来设置图片以及样式，首先为图片在上面的盒子中再创建一个小盒子，并且撑满该子盒子，这样图片就显示在了父盒子的左上角；

然后为子盒子添加边界显示，同时为边界添加圆角样式，并且为图片设置圆角样式；

最后为图片盒子调整位置，设置绝对位置，向右以及向上偏移

.avatar\_box {

    height: 130px;

    width: 130px;

    border: 1px solid #eee;

    border-radius: 50%;

    padding: 10px;

    box-shadow: 0 0 10px #ddd;

    position: absolute;

    left: 50%;

    transform: translate(-50%, -50%);

    background-color: #fff;

    img {

        width: 100%;

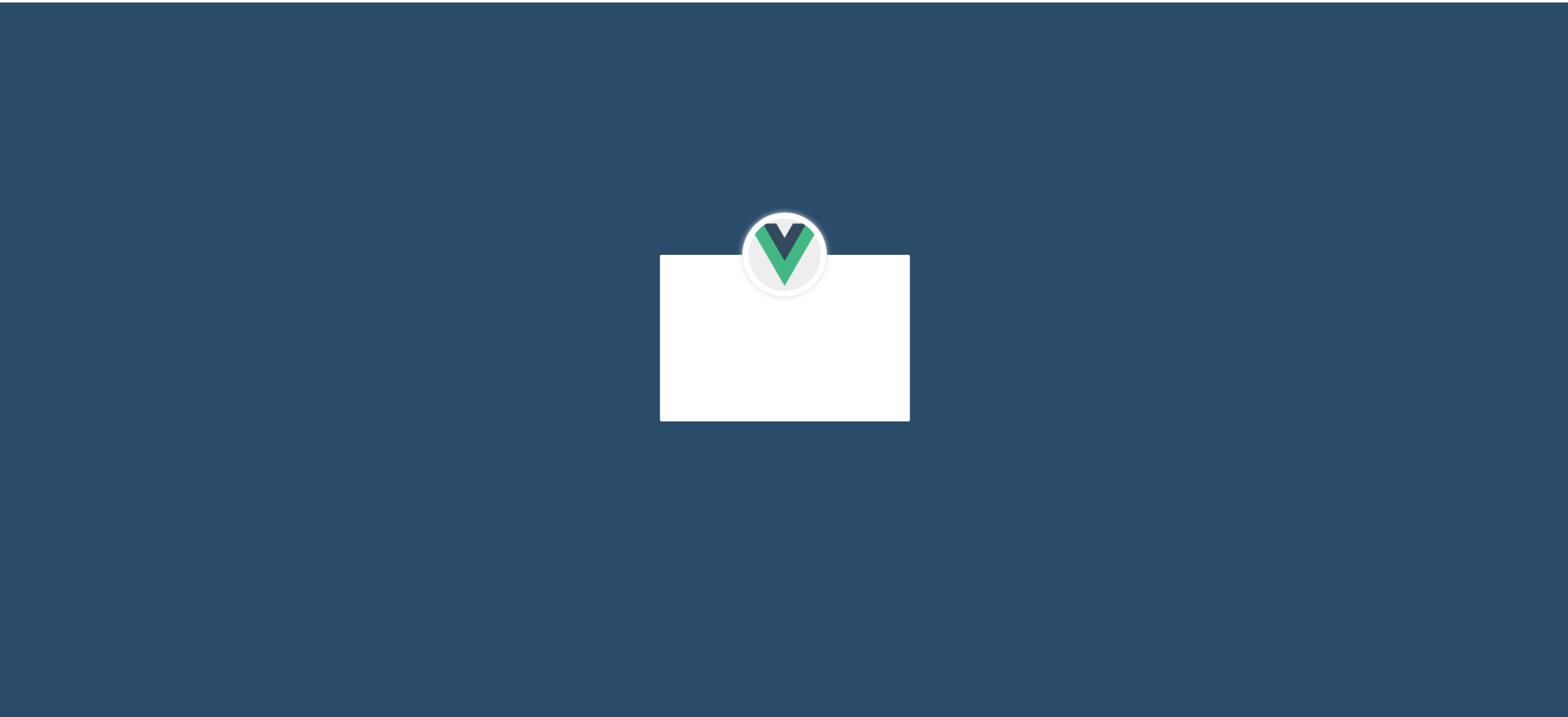
        height: 100%;

        border-radius: 50%;

        background-color: #eee;

    }

}



表单的编写，直接通过element ui进行调用，直接使用代码即可，但是会发现报错了，因为element ui是按需导入的，使用哪些el组件就导入哪些组件。因此，在element.js中进行导入：

import Vue from 'vue'

import { Button,Form,FormItem,Input } from 'element-ui'

Vue.use(Button)

Vue.use(Form)

Vue.use(FormItem)

Vue.use(Input)

接下来添加两个按钮，并且设置按钮的样式，尾部对齐：

.login\_form {

    position: absolute;

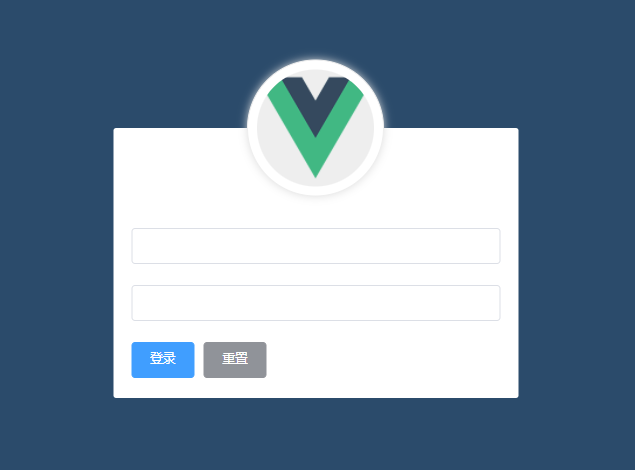
    bottom: 0;

    width: 100%;

    padding: 0 20px;

    box-sizing: border-box;

}



下面为输入框中添加小图标，通过icon属性来添加即可，从element ui组件中查询icon图标有哪些可以直接使用，或者使用第三方的图标库也行：

                <!-- 用户名 -->

                <el-form-item >

                    <el-input prefix-icon="el-icon-user"></el-input>

                </el-form-item>

                <!-- 密码 -->

                <el-form-item >

                    <el-input prefix-icon="el-icon-lock"></el-input>

                </el-form-item>



接下来为登录组件绑定数据源，也就是与后端连接。首先查看element ui官网中的form表单中的代码展示，通过:model属性（v-model）绑定一个在script中定义的数据对象，表单中的每一项都对应着该数据对象中的属性。

因此，我们需要根据表单中展示的数据定义数据对象loginForm，如username和password。然后为表单中的每一项绑定数据对象信息。

如果是密码文本框，则添加属性type=”password”即可：

            <!-- 登录表单区域 -->

            <el-form :model="loginForm" label-width="0px" class="login\_form">

                <!-- 用户名 -->

                <el-form-item >

                    <el-input v-model="loginForm.username" prefix-icon="el-icon-user"></el-input>

                </el-form-item>

                <!-- 密码 -->

                <el-form-item >

                    <el-input type="password" v-model="loginForm.password" prefix-icon="el-icon-lock"></el-input>

                </el-form-item>

                <!-- 按钮区域 -->

                <el-form-item class="btns">

                    <el-button type="primary">登录</el-button>

                    <el-button type="info">重置</el-button>

                </el-form-item>

            </el-form>

接下来对表单进行验证操作，通过element ui表单组件进行编写，直接通过一个验证规则对象就可以验证。

首先在表单区域的标签el-form标签中添加:rules绑定对象loginFormRules，然后在script中定义规则对象，分别定义验证用户名和密码的规则：

            //表单验证规则对象

            loginFormRules: {

                //验证用户名

                username: [

                    {required: true, message: "请输入登录用户名", trigger:"blur"},

                    {min: 3, max: 10, message: "长度在3-10个字符", trigger:"blur"}

                ],

                //验证密码

                password: [

                    {required: true, message: "请输入登录密码", trigger:"blur"},

                    {min: 6, max: 15, message: "长度在6-15个字符", trigger:"blur"}

                ]

            }

最后，在用户名和密码的输入框标签中绑定规则对象：

                <!-- 用户名 -->

                <el-form-item prop="username">

                    <el-input v-model="loginForm.username" prefix-icon="el-icon-user"></el-input>

                </el-form-item>

                <!-- 密码 -->

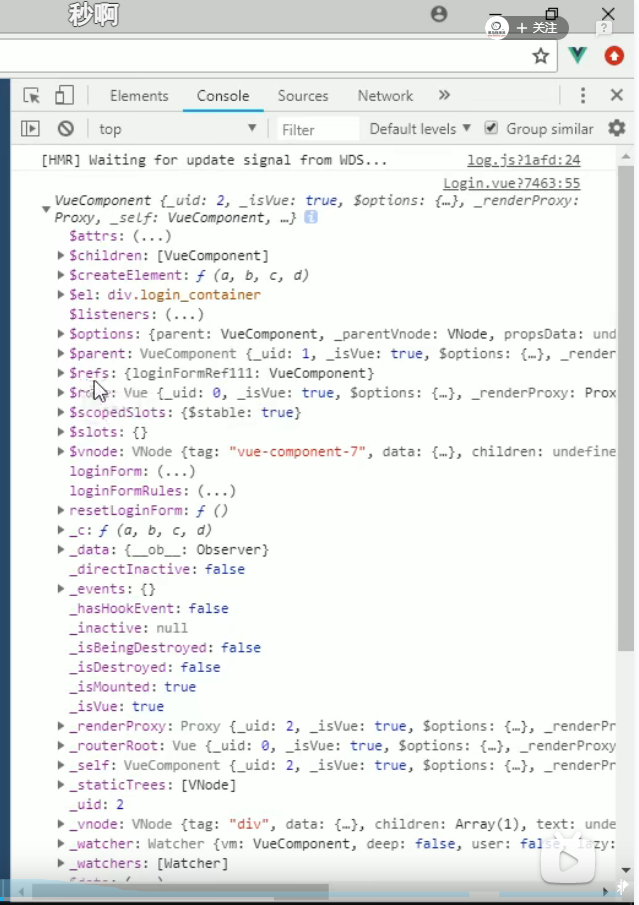
                <el-form-item prop="password">

                    <el-input type="password" v-model="loginForm.password" prefix-icon="el-icon-lock"></el-input>

                </el-form-item>

接下来，实现重置表单按钮。首先为按钮绑定重置函数，重置函数中通过获取表单对象，并且执行element ui中的form表单的函数resetFields()即可实现。

因此，需要获取表单的实例对象，通过ref标签来获取，这些都是可以通过chrome的开发者模式看到的，因此才能够通过引用对象直接获取表单实例对象。



    methods: {

        //点击重置按钮，重置登录表单

        resetLoginForm() {

            this.$refs.loginFormRef.resetFields();

        }

    }

接下来进行表单的预验证，通过调用表单的validate()函数进行校验；

首先，为登录button绑定函数login，在该函数中获取表单实例对象，调用validate()函数验证：

        //登录前的预验证

        login() {

            this.$refs.loginFormRef.validate(valid => {

                console.log(valid);

            });

        }

在验证完表单之后，需要根据验证结果发送http请求，因此在main.js中导入axios包，设置请求根路径：

//导入axios

import axios from 'axios'

//配置请求的根路径

axios.defaults.baseURL = 'http://127.0.0.1:8888/api/private/v1/'

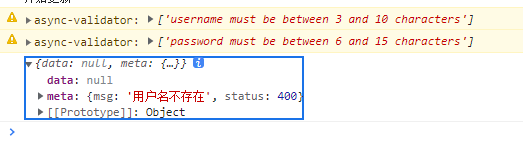
//全局属性，所有组件都可以访问

Vue.prototype.$http = axios

然后在login表单中的验证函数中进行调用post请求，参数是地址和传递数据对象；

通过f12查看返回结果，如果是promise对象，则使用await, async修饰；

从返回的data对象中直接取出res对象，也就是结果，如下所示：



        //登录前的预验证

        login() {

            this.$refs.loginFormRef.validate(async valid => {

                // console.log(valid);

                if(!valid) return;

                //获取data对象中的res

                const { data: res} = await this.$http.post('login', this.loginForm);

                console.log(res);

            });

        }

接下来进行登录结果的弹窗提示，导入element ui中的Message组件即可，在element.js中：

                if(res.meta.status != 200) return this.$message.error("登录失败");

                this.$message.success("登录成功");

登录成功后，需要记录后台返回的用户token信息，并且将token信息保存在浏览器的sessionStorage中，因为token只应该在网站打开期间有效；

同时在登录成功或失败后需要跳转到后台主页中，路由地址是/home，创建一个[Home.vue组件，代表后台主页，然后在router.js](http://Home.vue组件，代表后台主页，然后在router.js)中定义路由规则；这里由于不知道后台的密码规则，所以即使登录失败，也可以进入主页之中；

const routes = [

  {

    path: '/',

    redirect: '/login'

  },

  {

    path: '/login',

    component: Login

  },

  {

    path: '/home',

    component: Home

  }

]



接下来添加路由导航守卫控制访问权限，当用户未登陆下，直接访问有权限的页面时需要重新导航到登录页面中。

使用的是router的beforeEach方法，参数是to，from，next；要去哪个url，从哪个url来，回调函数next表示放行；

//挂载路由导航守卫

router.beforeEach((to,from,next) => {

  if(to.path == '/login') return next()

  //获取token,这里由于登录失败没有token，因此暂且不适用

  const tokenStr = window.sessionStorage.getItem('token')

  if(!tokenStr) return next('/login')

  next()

})

接下来是退出功能，基于token的退出只需要消耗本地的token即可，如果需要重新登录就生成一个新的token即可。

在点击了退出按钮后，就会清除本地的token信息，并且跳转到login页面之中；

<script>

export default {

    methods: {

        logout() {

            window.sessionStorage.clear();

            //跳转

            this.$router.push("/login");

        }

    }

}

</script>

## （二）主页

### 1.布局

主页的整体布局分为上下以及左右布局，先上下划分，再左右划分。

包括头部，侧边栏区域以及主体区域。



### 2.头部header

从Element ui中的Container布局容器中查找我们想要的布局效果，可以直接将代码复制过来。同时注意，要将Container,Header,Aside,Main等组件在element.js中导入。

为布局的头部，边栏，主体设置样式，充满屏幕以及设置高度：

<el-container class="home-container">

        <!-- 头部区域 -->

        <el-header>

            Header

            <el-button type="info" @click="logout">

            退出

            </el-button>

        </el-header>

        <!-- 页面主体区域 -->

        <el-container>

            <!-- 侧边栏 -->

            <el-aside width="200px">

                Aside

            </el-aside>

            <!-- 右侧内容主体 -->

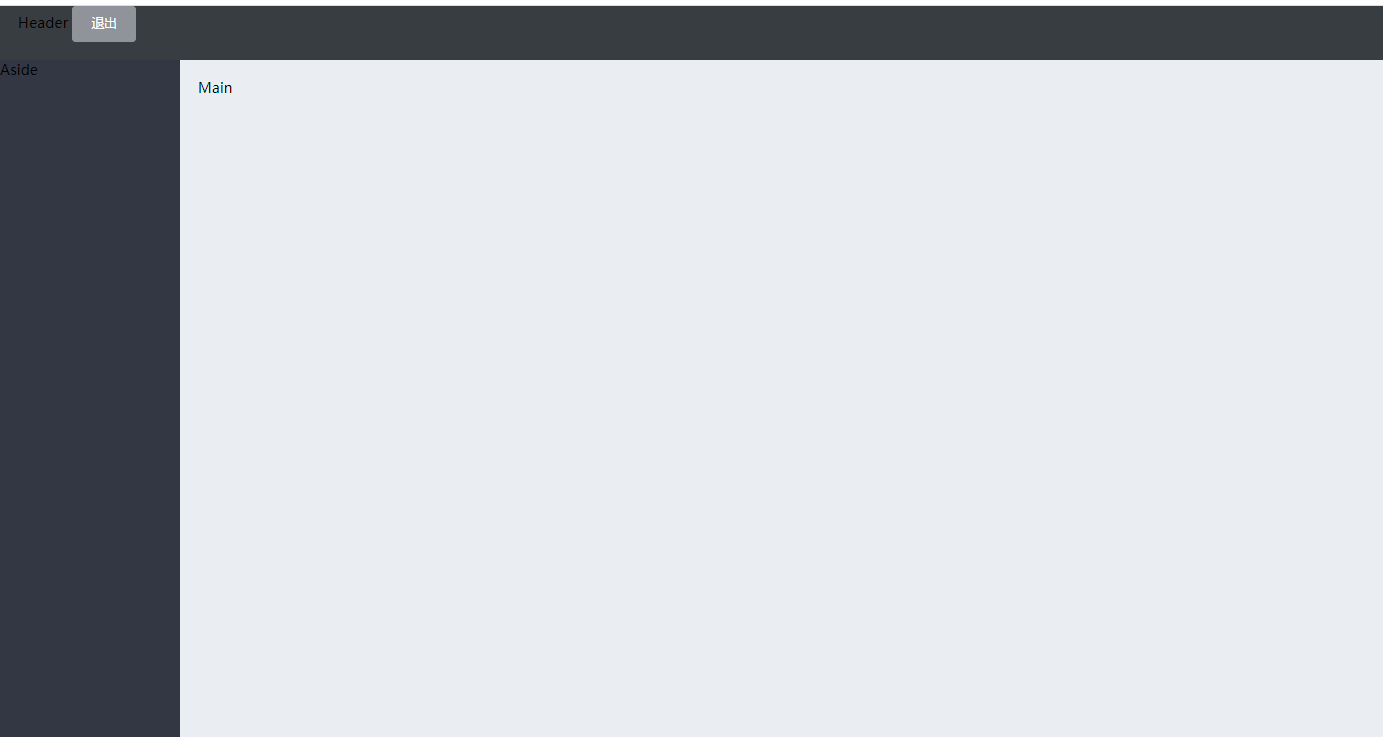
            <el-main>

                Main

            </el-main>

        </el-container>

    </el-container>



接下来编写头部区域，设置左侧的图片，文本，并设置样式；

<!-- 头部区域 -->

        <el-header>

            <div>

                <img src="../assets/logo.png" alt="">

                <span>电商后台管理系统</span>

            </div>

            <el-button type="info" @click="logout">

            退出

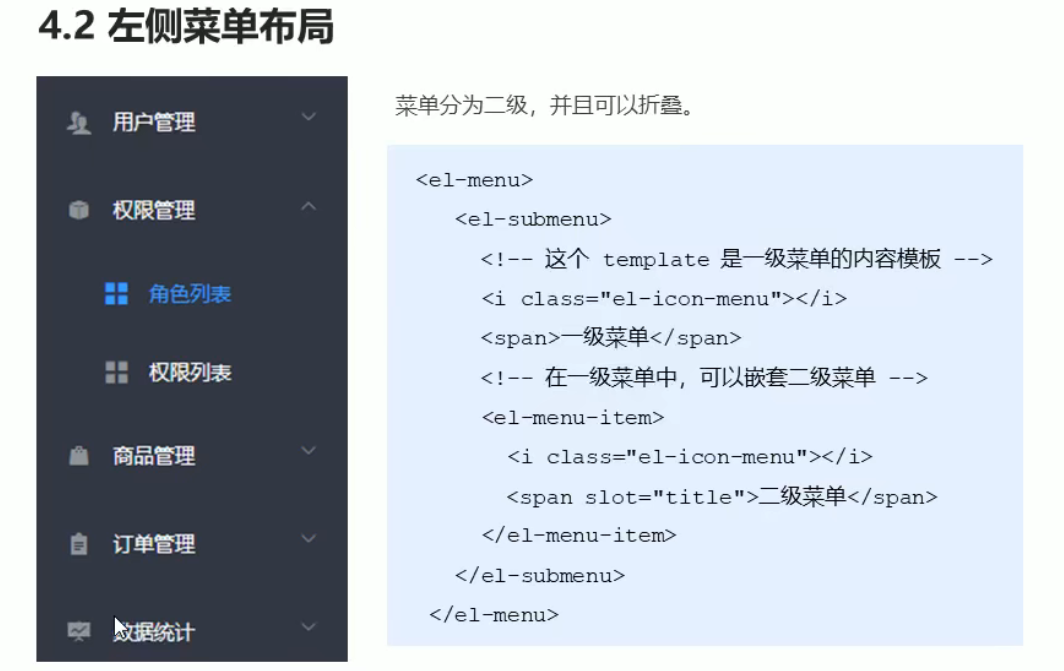
            </el-button>

        </el-header>

### 3.左侧栏aside

（1）布局

菜单分为两级，每一级菜单都有图标和文本显示，使用element ui中的组件制作。



（2）制作

首先从element ui中导入侧边栏的依赖Menu, Submenu, MenuItemGroup, MenuItem，然后使用粘贴来的代码；

由于我们只需要二级菜单，因此设置两级菜单；

<el-menu

          background-color="#333744"

          text-color="#fff"

          active-text-color="#ffd04b"

        >

          <!-- 一级菜单 -->

          <el-submenu index="1">

            <!-- 一级菜单的模板区域 -->

            <template slot="title">

              <!-- 图标 -->

              <i class="el-icon-location"></i>

              <!-- 文本 -->

              <span>导航一</span>

            </template>

            <el-menu-item index="1-4-1">

              <!-- 二级菜单的模板区域 -->

              <template slot="title">

                <!-- 图标 -->

                <i class="el-icon-location"></i>

                <!-- 文本 -->

                <span>导航一</span>

              </template>

            </el-menu-item>

          </el-submenu>

        </el-menu>



由于多个数据接口需要token认证，因此前端发送请求之前，需要通过axios拦截器添加token信息，config对象就是发送的信息对象，需要为config对象的Authorization中添加token信息，如下是在main.js中配置拦截器：

//配置拦截器

axios.interceptors.request.use(config => {

  console.log(config);

  config.headers.Authorization = window.sessionStorage.getItem('token')

  //最后必须return config

  return config

})

接下来获取左侧菜单的数据信息，通过接口获取。因此在Home组件中通过script获取接口数据，访问menus接口，注意此时已经携带了token信息了。

    //获取所有的菜单

    async getMenuList() {

        //获取menus地址的数据

        const {data: res } = await this.$http.get('menus')

        console.log(res);

        if(res.meta.status != 200) return this.$message.error(res.meta.msg)

        this.menulist = res.data

    }

要使用接口中返回的数据，就需要通过for循环将数据全部展示出来；

注意，菜单中的index属性，每级菜单各自应该都是唯一的，如果菜单的index属性相同则会导致多个相同index的菜单同时动作的效果，因此需要使用接口返回的id信息，同时注意index属性应该使用字符串，因此需要设置为字符串：

          <el-submenu :index="item.id+''" v-for="item in menulist" :key="item.id">

            <!-- 一级菜单的模板区域 -->

            <template slot="title">

              <!-- 图标 -->

              <i class="el-icon-location"></i>

              <!-- 文本 -->

              <span>{{ item.authName }}</span>

            </template>

            <el-menu-item :index="subItem.id + ''" v-for="subItem in item.children" :key="subItem.id">

              <!-- 二级菜单的模板区域 -->

              <template slot="title">

                <!-- 图标 -->

                <i class="el-icon-location"></i>

                <!-- 文本 -->

                <span>{{ subItem.authName }}</span>

              </template>

            </el-menu-item>

          </el-submenu>



下面设置Welcome页面，在Home页面中显示，因此welcome路径是.home路径的子路由，也就是说welcome页面在home页面之中展示出来。只要访问了/home，就会重定向到/welcome路径中，因此需要设置重定向路由规则。

最后在Home组件的主体区域放置路由占位符router-view，这样Welcome组件才能够显示。

  {

    path: '/home',

    component: Home,

    // 重定向

    redirect: '/welcome',

    // 子路由

    children: [

      {

        path: '/welcome',

        component: Welcome

    }]

  }

下面设置侧边栏的跳转路由，也就是说点击每个侧边栏时会跳转到某个组件中，我们需要设置侧边栏为router路由模式，通过为el-menu菜单添加属性router为true即可，那么每个菜单都会根据index属性作为路由进行跳转。因此，我们需要设置index属性的路径。

因此，我们需要在传回数据时也传回该项的index属性为路由跳转地址。

            <el-menu-item

              :index="'/' + subItem.path"

              v-for="subItem in item.children"

              :key="subItem.id"

            >

当点击用户管理中的用户列表时，路由路径就会变为了users：



### 4.用户管理

（1）布局

最上面是组件显示，中间是一个文字框，最下面是用户列表的显示。



（2）制作

首先创建用户管理页面的组件Users.vue，然后在路由中添加到Home组件中的子路由之中，这样用户管理就会在Home组件之中了。

    // 子路由

    children: [

      {

        path: '/welcome',

        component: Welcome

      },

      {

        path: '/users',

        component: Users

      }

    ]

接下来对选中的菜单栏实现高亮，这就需要通过click函数存储选中的路由链接到sessionStorage中，然后进行高亮显示。

接下来制作页面上方的导航显示，也就是面包屑组件。通过element ui中的组件即可以使用，同样的需要导入Breadcrumb, BreadcrumbItem组件。

      <!-- 面包屑导航区域 -->

    <el-breadcrumb separator-class="el-icon-arrow-right">

      <el-breadcrumb-item :to="{ path: '/' }">首页</el-breadcrumb-item>

      <el-breadcrumb-item>活动管理</el-breadcrumb-item>

      <el-breadcrumb-item>活动列表</el-breadcrumb-item>

      <el-breadcrumb-item>活动详情</el-breadcrumb-item>

    </el-breadcrumb>



接下来制作卡片视图区域，也就是中间的输入框与添加用户按钮。首先，添加el-input组件，即输入框，再添加一个button。然后为该区域设置全局样式，设置固定的宽度，可以使用element ui中的栅格layout布局设置，通过el-row代表一行，el-col代表一列，span表示宽度，gutter表示每一列之间的距离。注意需要导入Row和Col组件。

    <!-- 卡片视图区域 -->

    <el-card class="box-card">

      <!-- 通过layout布局 -->

      <el-row :gutter="20">

        <!-- 每一列 -->

        <el-col :span="7">

          <!-- 搜索域添加区域 -->

          <el-input placeholder="请输入内容">

            <el-button slot="append" icon="el-icon-search"></el-button>

          </el-input>

        </el-col>

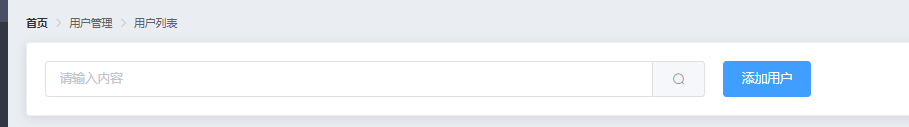
        <el-col :span="4">

            <el-button type="primary">添加用户</el-button>

        </el-col>

      </el-row>

    </el-card>



接下来获取用户列表信息，通过ajax请求获取。首先在生命周期函数中通过getUserList()函数获取用户列表信息，则在页面创建期间就会获取用户列表信息。

    async getUserList() {

      const {data: res } = await this.$http.get("users", { params: this.queryInfo });

      console.log(res);

      if(res.meta.status != 200){

          return this.$message.error('获取用户列表信息失败！')

      }

      this.userList = res.data.users

      this.total = res.data.total

    },

获取到用户列表信息后，就需要将信息渲染到表格之中，使用element ui中的table表格组件，data表示动态绑定数据对象，table-column中的prop代表该列在数据对象中的属性值，通过prop获取对应的数据，label表示显示标题；

对表格添加边框线，斑马线效果；

      <!-- 用户列表区域 -->

      <el-table :data="userList" border stripe>

          <el-table-column label="姓名" prop="username"></el-table-column>

          <el-table-column label="邮箱" prop="email"></el-table-column>

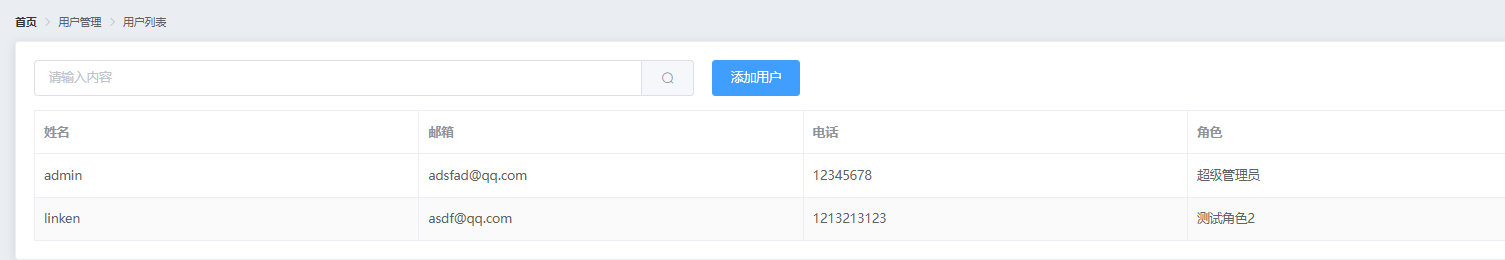
          <el-table-column label="电话" prop="mobile"></el-table-column>

          <el-table-column label="角色" prop="role\_name"></el-table-column>

          <el-table-column label="状态" prop="mg\_state"></el-table-column>

          <el-table-column label="操作" ></el-table-column>

      </el-table>



为状态列设置按钮效果，也就是该按钮能够修改数据，不仅仅是显示数据，使用作用域插槽，然后导入el-switch组件的按钮效果。

          <el-table-column label="状态" prop="mg\_state">

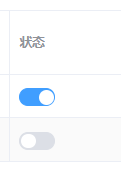
              <template slot-scope="scope">

                  <el-switch v-model="scope.row.mg\_state">

                  </el-switch>

              </template>

          </el-table-column>



接下来设置操作中的多个按钮，也是通过作用域插槽来实现的，同时还需要设置按钮的样式和鼠标注释。



下面制作数据分页效果，依然使用element ui中的el-pagination分页组件，只需要配置一些参数即可实现功能；

如页面的size的修改监听函数，当前页码修改的监听函数，当前页码，页面的size列表，页面的size，以及分页的布局，最后是总体的数据信息。

当页码的size更新后，就需要即时更新页码的size，

      <!-- 分页区域 -->

      <el-pagination @size-change="handleSizeChange" @current-change="handleCurrentChange"

      :current-page="queryInfo.pagenum" :page-sizes="[1,2,5,10]"

      :page-size="queryInfo.pagesize" layout="total, sizes, prev, pager, next, jumper"

      :total="total">

    //监听pagesize改变的事件

    handleSizeChange(newSize) {

        this.queryInfo.pagesize = newSize

        this.getUserList()

    },

    //监听 页码值 改变的事件

    handleCurrentChange(newPage) {

        this.queryInfo.pagenum = newPage

        this.getUserList()

    }

下面制作用户状态的修改与后端匹配，也就是说前台的状态按钮修改后，需要同步到后端之中。

首先监听按钮的修改，通过change事件监听，设置回调函数userStateChanged，参数是该列用户的信息对象，通过获取switch中的信息修改后台中的数据。

    //监听switch开关状态的改变

    async userStateChanged(userinfo) {

        console.log(userinfo);

        //注意这里不应该是单引号，而是反引号，模板字符串

        const { data: res} = await this.$http.put(`users/${userinfo.id}/state/${userinfo.mg\_state}`)

        if(res.meta.status != 200){

            userinfo.mg\_state = !userinfo.mg\_state

            return this.$message.error('更新用户状态信失败！')

        }

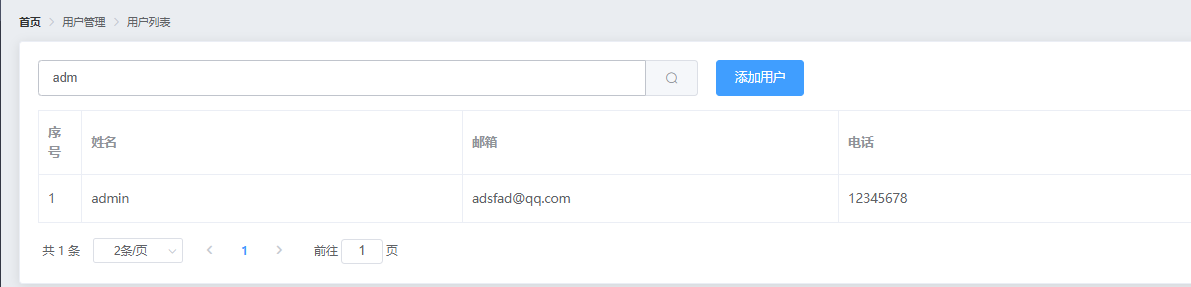
        this.$message.success('更新用户状态成功')

    }

接下来编写搜索功能，后台已经写好了搜索功能，我们只需要通过button按钮绑定函数getUserList，传入关键字就可以搜索得出结果。

输入框的绑定数据通过v-model实现双向绑定。

同时，还可以优化输入框中的删除功能，全部删除后自动调用回调函数恢复所有数据显示。



接下来编写添加用户的功能，通过一个弹出的对话框编写，使用element ui中的dialog组件实现。visible.sync表示是否显示对话框，下面是内容主体区域，以及底部的按钮区域，同样要为每个按钮绑定点击事件，将对话框显示的flag设置为flase

点击“添加用户”按钮，就会弹出对话框，设置@click事件中为true即可显示，

    <!-- 添加用户的对话框 -->

    <el-dialog title="提示" :visible.sync="addDialogVisible" width="50%">

        <!-- 内容主体区域 -->

        <span>这是一段信息</span>

        <!-- 底部区域 -->

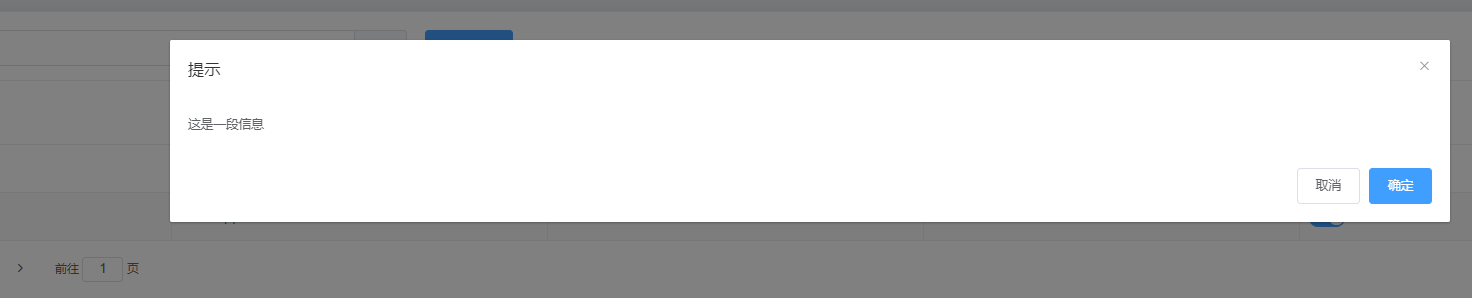
        <span slot="footer" class="dialog-footer">

            <el-button @click="addDialogVisible=false">取消</el-button>

            <el-button type="primary" @click="addDialogVisible=false">确定</el-button>

        </span>

    </el-dialog>



下面在对话框中添加一个表单，表单中需要对应的表单数据对象，表单校验规则，表单引用对象。通过prop属性来绑定表单中每一项的验证规则。

<!-- 添加用户的表单 -->

      <el-form

        :model="addForm"

        :rules="addFormRules"

        ref="addFormRef"

        label-width="70px"

      >

        <el-form-item label="用户名" prop="username">

          <el-input v-model="addForm.username"></el-input>

        </el-form-item>

        <el-form-item label="密码" prop="password">

          <el-input v-model="addForm.password"></el-input>

        </el-form-item>

        <el-form-item label="邮箱" prop="email">

          <el-input v-model="addForm.email"></el-input>

        </el-form-item>

        <el-form-item label="手机" prop="mobile">

          <el-input v-model="addForm.mobile"></el-input>

        </el-form-item>

      </el-form>



接下来设置表单关闭时的自动重置表单所有选项的功能，也就是说表单在关闭后，所有选项需要变为初始值，通过el-dialog中的属性@close实现，绑定一个函数即可，函数的功能就是清空所有值。

    //监听添加用户对话框的关闭事件

    addDialogClosed() {

        this.$refs.addFormRef.resetFields()

    }

接下来为确定按钮添加一个点击事件，校验表单中的所有数据是否符合要求，返回boolean；

    //点击按钮，添加新用户

    addUser() {

        this.$refs.addFormRef.validate(valid => {

            if(!valid) return

            //验证成功，可以发起HTTP请求

        })

    }

接下来发起http请求，根据请求的结果判断是否添加用户成功！！

注意如果要异步获取返回的结果，需要使用async 和await同时使用；

    //点击按钮，添加新用户

    addUser() {

        this.$refs.addFormRef.validate(async valid => {

            if(!valid) return

            //验证成功，可以发起HTTP请求

            const {data: res} = await this.$http.post('users', this.addForm)

            console.log(res);

            if(res.meta.status != 201){

                this.$message.error('添加用户失败！')

            }

            this.$message.success('添加用户成功！')

            //隐藏对话框

            this.addDialogVisible = false

            //添加了新用户，需要重新获取用户列表数据

            this.getUserList()

        })

    }

接下来进行修改用户信息的操作，通过每个用户的操作按钮，弹出用户信息的对话框，修改对话框中的信息。与上面的对话框操作类似。

同时，注意，用户名是不能被更改的，因此设置属性为disabled就可以了。

<!-- 修改用户的对话框 -->

    <el-dialog title="修改用户" :visible.sync="editDialogVisible"

    width="50%">

      <!-- 内容主体区域 -->

      <!-- 添加用户的表单 -->

      <el-form

        :model="editForm"

        :rules="editFormRules"

        ref="editFormRef"

        label-width="70px"

      >

        <el-form-item label="用户名" prop="username">

          <el-input v-model="editForm.username" disabled></el-input>

        </el-form-item>

        <el-form-item label="邮箱" prop="email">

          <el-input v-model="editForm.email"></el-input>

        </el-form-item>

        <el-form-item label="手机" prop="mobile">

          <el-input v-model="editForm.mobile"></el-input>

        </el-form-item>

      </el-form>

      <!-- 底部区域 -->

      <span slot="footer" class="dialog-footer">

        <el-button @click="editDialogVisible = false">取消</el-button>

        <el-button type="primary" @click="editDialogVisible = false"

          >确定</el-button>

      </span>

    </el-dialog>

接下来进行表单提交前的预验证操作，直接在确定按钮中添加一个@click事件，验证成功则可以发送http请求，

    //修改用户信息提交

    editUserInfo() {

        this.$refs.editFormRef.validate(async valid => {

            if(!valid) return

            //发起修改用户信息的请求

            const {data: res} = await this.$http.put('users/' + this.editForm.id, {

                email: this.editForm.email,

                mobile: this.editForm.mobile

            })

            if(res.meta.status != 200){

                return this.$message.error('更新用户信息失败！')

            }

            //关闭对话框

            this.editDialogVisible = false

            //刷新数据列表

            this.getUserList()

            //提示修改成功

            this.$message.success('更新用户信息成功！')

        })

    }

接下来做删除用户的操作，需要确认弹窗的组件，因此导入MessageBox中的confirm组件作为全局组件使用。

然后为该按钮绑定一个删除函数，删除指定id的用户信息，注意需要添加防误删除的弹窗。

    //根据Id删除对应的用户信息

    async removeUserById(id) {

        //弹框询问用户是否删除数据

        const confirmResult = await this.$confirm('此操作将永久删除该用户，是否继续？',

        '提示',

        {

            confirmButtonText: '确定',

            cancelButtonText: '取消',

            type: 'warning'

        }).catch(err => err)

        //如果用户确认删除，则返回值为confirm，否则为cancel

        if(confirmResult != 'confirm'){

            return this.$message.info('已取消删除！')

        }

        const {data: res} = await this.$http.delete('users/' + id)

        if(res.meta.status != 200){

            return this.$message.error('删除用户失败！')

        }

        this.$message.success('删除用户成功！')

        //刷新用户列表

        this.getUserList()

    }

### 5.权限管理

（1）布局

还是一个面包屑+卡片视图

（2）制作

直接复制之前的面包屑和卡片视图即可；

首先获取权限列表数据，直接通过生命周期函数调用接口即可；

然后将数据通过table进行渲染呈现，首先绑定一个数据对象，然后从该数据对象中获取每一列的数据信息显示。

如果想要为权限等级设置好看的标签，则需要引入Tag组件，通过等级来判断应该显示哪一个tag标签；

    <!-- 面包屑导航区域 -->

    <el-breadcrumb separator-class="el-icon-arrow-right">

      <el-breadcrumb-item :to="{ path: '/home' }">首页</el-breadcrumb-item>

      <el-breadcrumb-item>权限管理</el-breadcrumb-item>

      <el-breadcrumb-item>权限列表</el-breadcrumb-item>

    </el-breadcrumb>

    <!-- 卡片视图 -->

    <el-card>

        <el-table :data="rightsList" border stripe>

            <el-table-column type="index"></el-table-column>

            <el-table-column label="权限名称" prop="authName"></el-table-column>

            <el-table-column label="路径" prop="path"></el-table-column>

            <el-table-column label="权限等级" prop="level">

                <template slot-scope="scope">

                    <!-- 标签 -->

                    <el-tag v-if="scope.row.level == '0' ">一级</el-tag>

                    <el-tag type="success" v-else-if="scope.row.level == '2' ">一级</el-tag>

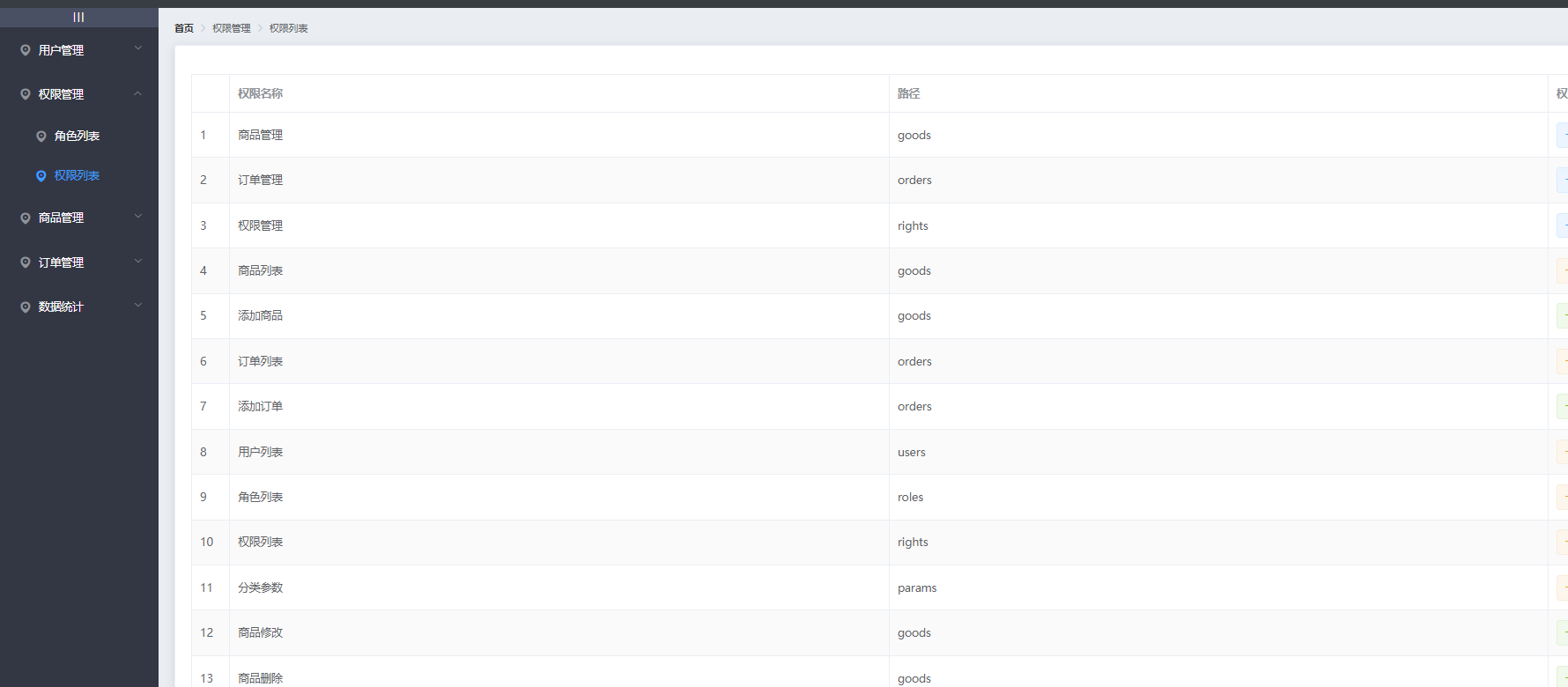
                    <el-tag type="warning" v-else>一级</el-tag>

                    </template>

            </el-table-column>

        </el-table>

    </el-card>



### 6.角色管理

（1）布局

面包屑+添加按钮+角色表格

（2）制作

面包屑和卡片视图直接复制；

使用layout布局，通过el-row和el-col进行布局；

注意表格中有一个展开列，能够展开一行的数据，可以进行二级数据的展示。

        <!-- 角色列表区域 -->

        <el-table :data="roleList" border stripe>

            <!-- 展开列 -->

            <el-table-column type="expand"></el-table-column>

            <!-- 索引列 -->

            <el-table-column type="index"></el-table-column>

            <el-table-column label="角色名称" prop="roleName"></el-table-column>

            <el-table-column label="角色描述" prop="roleDesc"></el-table-column>

            <el-table-column label="操作" width="300px">

                <template slot-scope="scope">

                    <el-button size = "mini" type="primary" icon="el-icon-edit">编辑</el-button>

                    <el-button size = "mini" type="danger" icon="el-icon-delete">删除</el-button>

                    <el-button size = "mini" type="warning" icon="el-icon-setting">分配权限</el-button>

                </template>

            </el-table-column>

        </el-table>



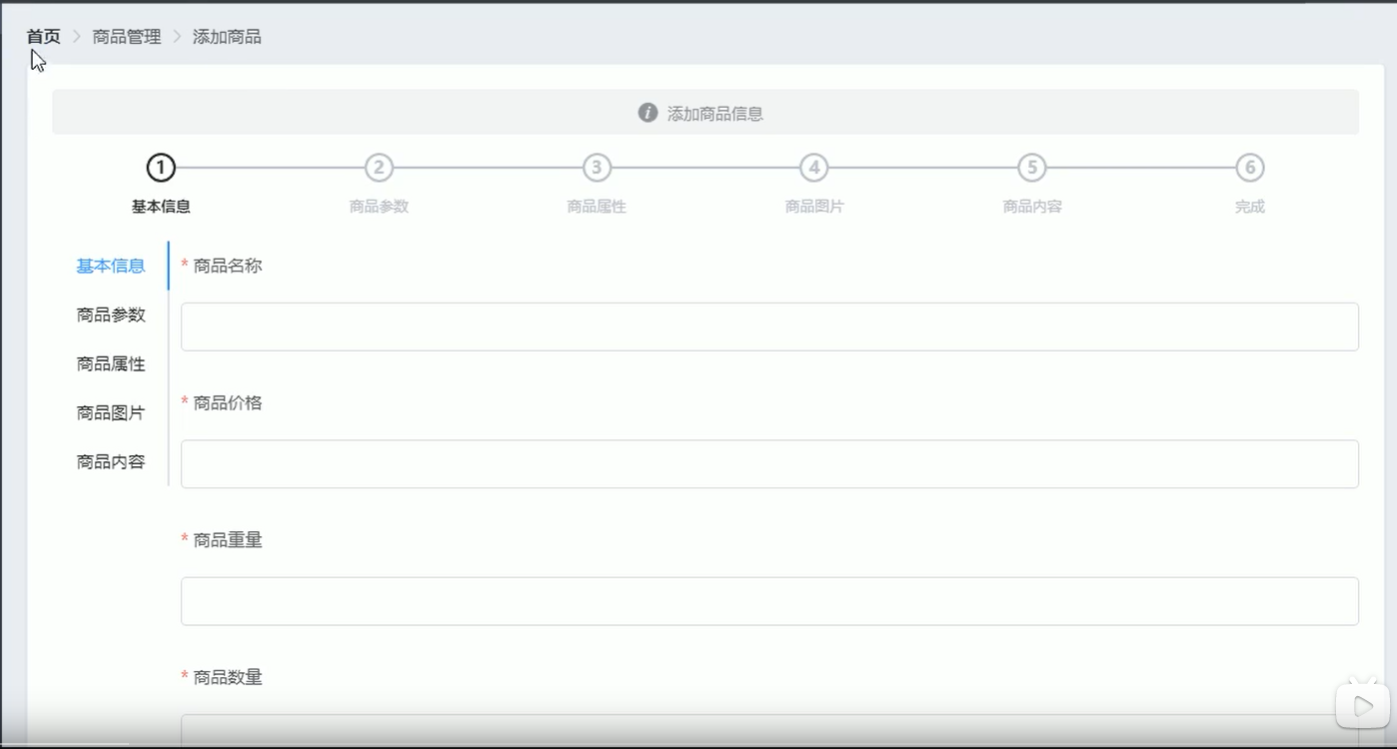
### 7.添加商品页面流程

添加商品有多个步骤，因此需要通过一个添加流程来添加商品信息；

首先为按钮设置点击函数，

1. 布局

添加商品的流程页面如下所示，包括面包屑，警告栏，进度条，面板。



1. 制作

首先还是复制面包屑，警告栏，步骤条的element ui实例代码，然后进行相应的修改，最后添加路由规则。

    <!-- 卡片视图 -->

    <el-card>

        <!-- 提示区域 -->

        <el-alert title="添加商品信息" type="info" center show-icon :closable="false"></el-alert>

        <!-- 步骤条区域 -->

        <el-steps :space="200" :active="1" finish-status="success">

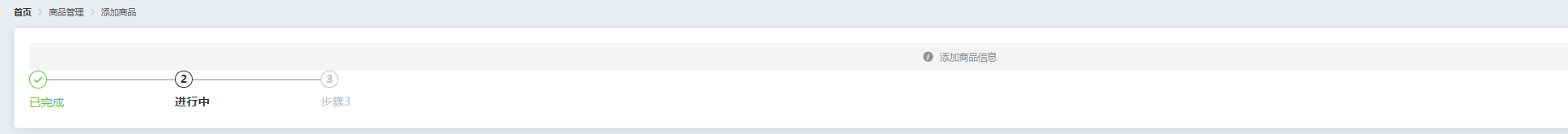
            <el-step title="已完成"></el-step>

            <el-step title="进行中"></el-step>

            <el-step title="步骤3"></el-step>

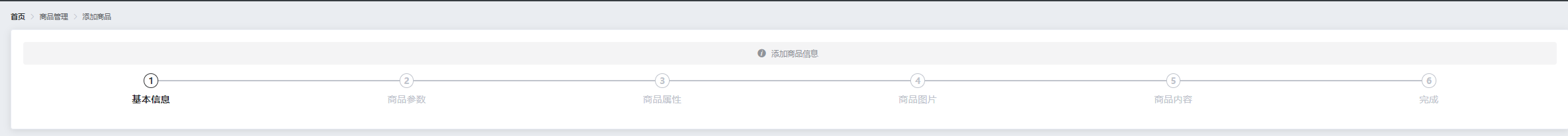
        </el-steps>

    </el-card>

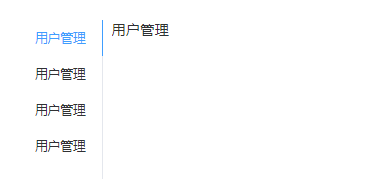


接下来设置样式，首先调整上下的margin高度差，再调整文字大小，通过steps组件的属性align center整体居中，如果没有居中，则调整center属性；

注意步骤条之中的激活选项初始应为0，即第一个选项激活。



接下来添加左侧显示的tab栏区域，通过tabs和tab-pane组件实现。

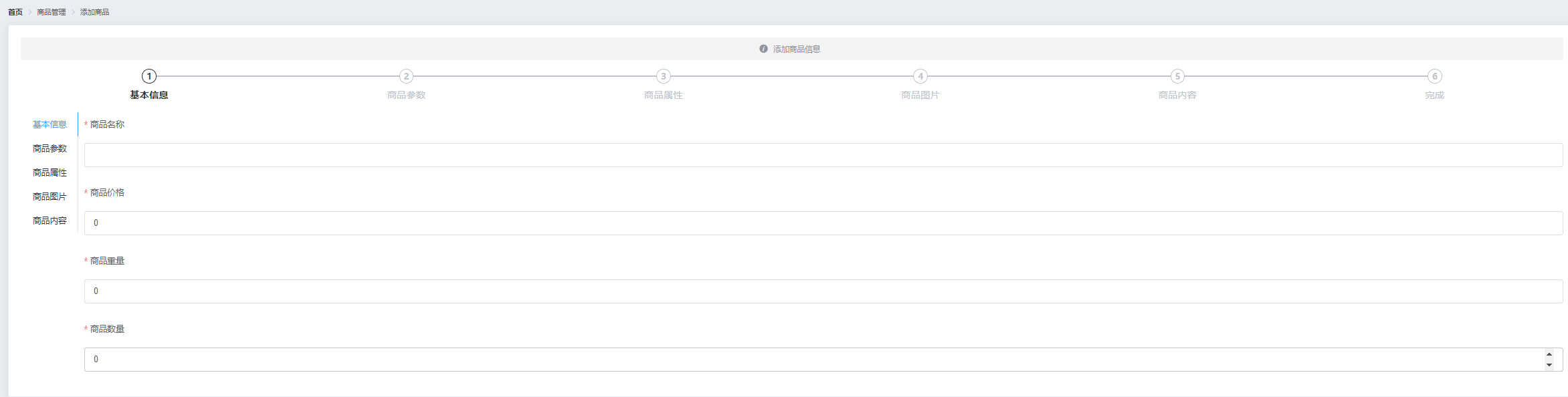


接下来让步骤条和tab栏联动显示同一个选项，那么就需要使用同一个数据项activeIndex；

接下来为每个tab页面绑定form表单，因为每个tab页面都需要通过form表单填写数据，因此使用form表单包裹tabs；



接下来为每个表单项绑定数据对象，校验规则对象，设置输入框的类型。发现卡片的整体高度不够，是因为之前我们设置了高度，因此删除tabs中的高度设置即可；



下面需要设置防止随意切换tab页面，只有在当前页都完成之后才能切换到下一页之中，通过tabs中的属性before-leave在切换tab标签时校验页面的信息是否可以进行切换，通过校验函数进行校验。校验函数的参数包括旧的标签页以及新的标签页，我们进行自行判断即可。

通过表单的validate功能，验证每个表单项是否填写即可。

    beforeTabLeave(activeName, oldActiveName) {

      console.log("即将离开的标签页：" + oldActiveName);

      console.log("即将进入的标签页：" + activeName);

      if (oldActiveName == "0") {

        //校验是否填写

        this.$refs.addFormRef.validate((valid) => {

            console.log(valid);

          if (!valid) {

            //提示

            this.$message.error("当前页面有信息未填写！");

          } else {

            this.changeTab = true

          }

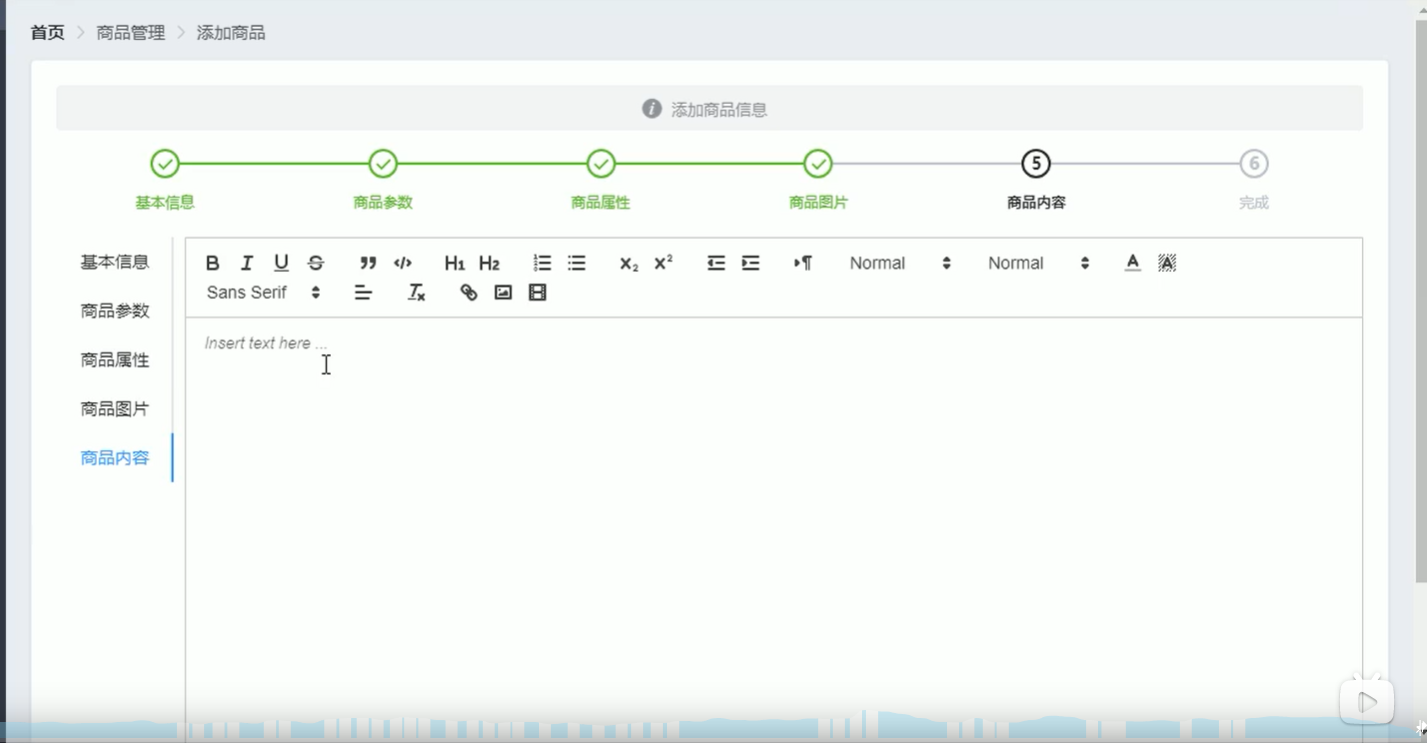
        });

        return this.changeTab

      }

    },

接下来添加富文本编辑器，如下所示：



首先在vue可视化工具中导入依赖vue-quill-editor；

然后在main.js中导入富文本编辑器以及对应的css样式，最后进行编辑器的全局注册，注册为全局可用的组件；

//导入富文本编辑器

import VueQuillEditor from 'vue-quill-editor'

//导入富文本编辑器对应的样式

import 'quill/dist/quill.core.css'

import 'quill/dist/quill.snow.css'

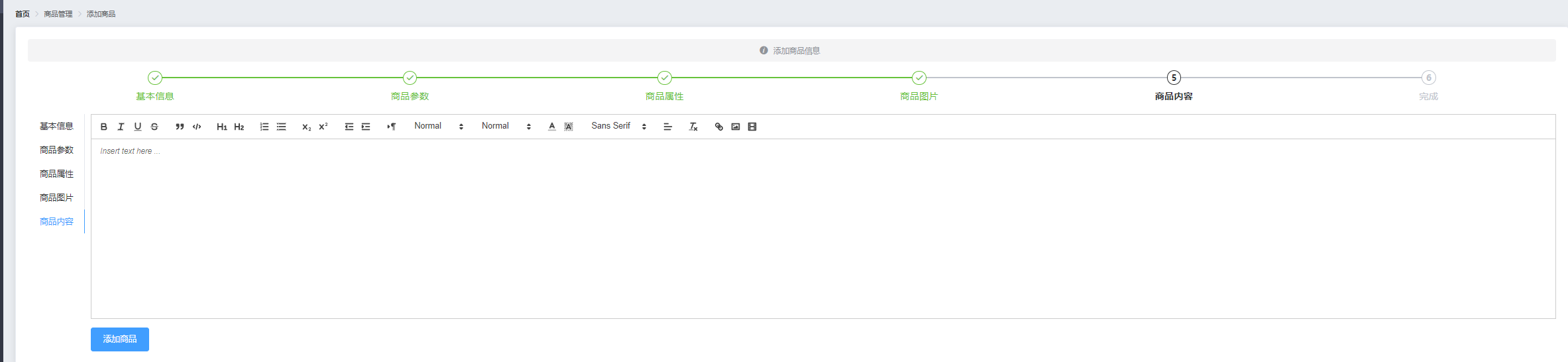
import 'quill/dist/quill.bubble.css'

//将富文本编辑器，注册为全局可用的组件

Vue.use(VueQuillEditor)

接下来在页面中使用富文本编辑器；

并添加一个按钮，添加商品；



下面添加图标功能；

首先在vue的可视化页面中添加依赖echarts组件，

在Add.vue中导入echarts；

在vue组件中准备一个具备大小的Dom；

基于准备好的dom，初始化echarts实例；

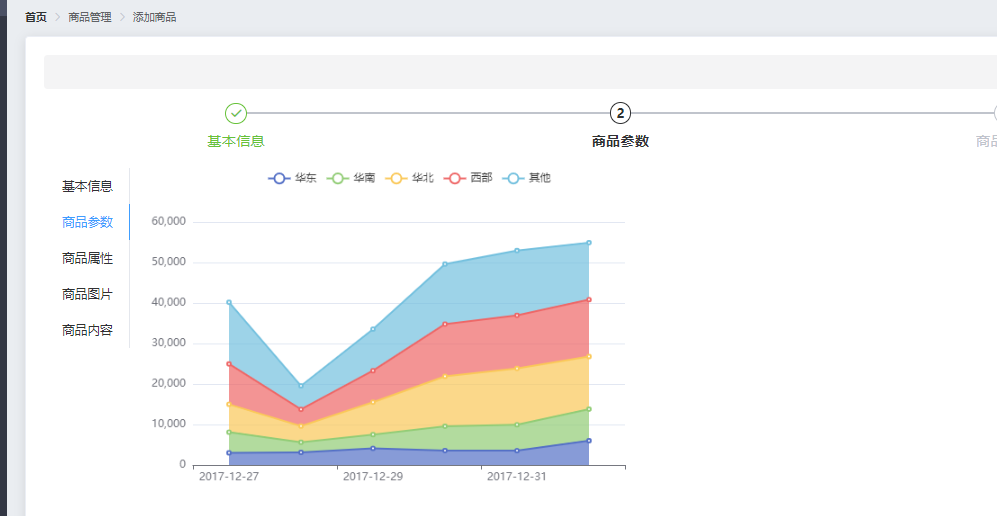
准备数据和配置项；

最后通过调用函数展示数据即可；

（上述步骤可在Echarts官方文档中查看）



折线图：



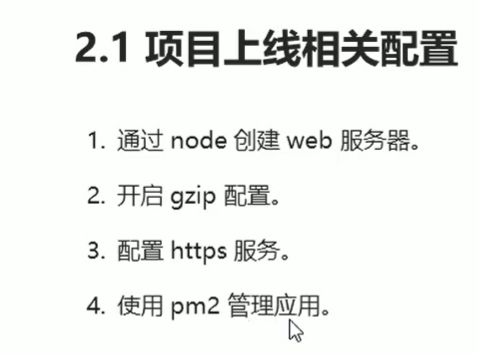
# 三．项目上线

## （一）项目上线优化





## （二）项目上线





创建新的项目admin-manage-server，从管理员终端中执行如下命令：

npm init -y

再安装express包

npm I express -S

然后将原admin-manage项目之中的编译生成的dist目录复制到当前项目中；

创建app.js作为入口文件；

首先导入express

// 导入express

const express = require('express')

// 创建express实例对象

const app = express()

// 托管静态资源

app.use(express.static('./dist'))

// 监听端口

app.listen(80, () => {

    console.log('server running at http://127.0.0.1');

})

在终端中执行node 命令：

node .\app.js

出现了端口占用错误，说明我们监听的80端口被占用了，因此修改端口，首先创建ServerConf.js文件设置端口：

const ServerConf = {

    ServicePort: 3036

};

module.exports=ServerConf

接着在app.js中引入端口的设置，同时修改了端口监听：

// 监听端口

app.listen(3036, () => {

    console.log('server running at http://127.0.0.1');

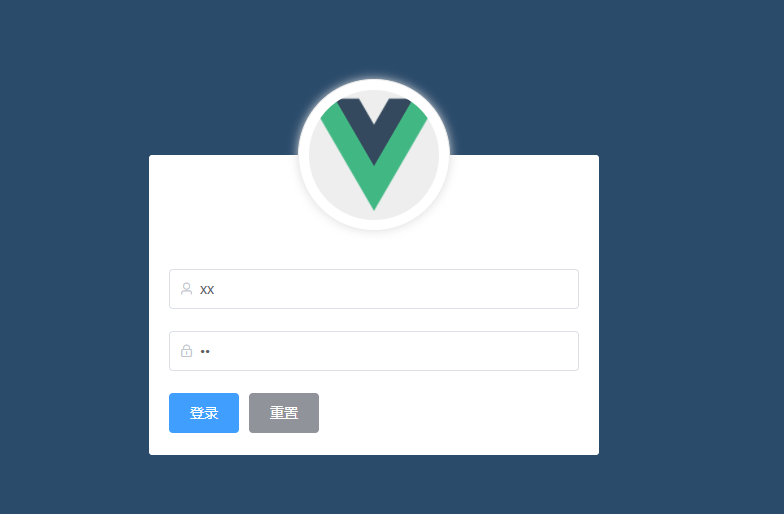
})

//设置端口号

var ServerConf = require("./serverConf");

process.env.PORT=ServerConf.ServicePort;

重新启动node项目，打开网址 127.0.0.1:3306



功能一切正常！！！