前端笔记

1. node环境

官网：https://nodejs.org

```
C:\Users\dacai>node -v
v16.12.0

C:\Users\dacai>npm -v
8.1.0
```

注意，node可以比我稍低，但不要更高

2. 下载vue-admin-template

https://panjiachen.gitee.io/vue-element-admin-site/zh/guide/

项目展开

public中的index.html为项目展开唯一的html，这个是一个单页应用，favicon.ico为浏览器上的小图标，

src中的api封装发给后端的请求，layout封装跟布局有关的组件，store为vuex

utils/request.js封装了拦截器，这里封装了请求拦截器和响应拦截器，请求拦截器为

```
// request interceptor
service.interceptors.request.use(
  config => {
    // do something before request is sent

    if (store.getters.token) {
      // let each request carry token
      // ['X-Token'] is a custom headers key
      // please modify it according to the actual situation
      config.headers['X-Token'] = getToken()
    }
    return config
  },
  error => {
    // do something with request error
    console.log(error) // for debug
    return Promise.reject(error)
  }
)
```

响应拦截器为

```
// response interceptor
service.interceptors.response.use(
  /**
   * If you want to get http information such as headers or status
   * Please return  response => response
  */

  /**
   * Determine the request status by custom code
   * Here is just an example
   * You can also judge the status by HTTP Status Code
   */
  response => {
    const res = response.data

    // if the custom code is not 20000, it is judged as an error.

    // 这里有前后端对接的标准，正常情况下返回20000
    if (res.code !== 20000) {
      Message({
        message: res.message || 'Error',
        type: 'error',
        duration: 5 * 1000
      })

      // 50008: Illegal token; 50012: Other clients logged in; 50014: Token expired;
      if (res.code === 50008 || res.code === 50012 || res.code === 50014) {
        // to re-login
        MessageBox.confirm('You have been logged out, you can cancel to stay on this
page, or log in again', 'Confirm logout', {
          confirmButtonText: 'Re-Login',
          cancelButtonText: 'Cancel',
          type: 'warning'
        }).then(() => {
          store.dispatch('user/resetToken').then(() => {
            location.reload()
          })
        })
      }
      return Promise.reject(new Error(res.message || 'Error'))
    } else {
      return res
    }
  },
  error => {
    console.log('err' + error) // for debug
    Message({
      message: error.message,
      type: 'error',
```

```
      duration: 5 * 1000
    })
    return Promise.reject(error)
  }
)
```

App.vue为根组件，main.js为说明文件

.env.development为开发配置，.env.production为生产配置，.env.staging为测试配置

package.json定义依赖，vue.config.js为一些配置文件

3. 项目初始化

1. 解压至非中文无空格目录下
2. vscode打开项目
3. 安装依赖

   ```
   npm config set registry http://registry.npm.taobao.org/
   ```

   ```
   npm install
   ```

4. 运行测试

   ```
   npm run dev
   ```

5. 运行报错：

   ```
   (node:internal/fs/read_file_context:68:3) {
     opensslErrorStack: [ 'error:03000086:digital envelope routines::initialization
   error' ],
     library: 'digital envelope routines',
     reason: 'unsupported',
     code: 'ERR_OSSL_EVP_UNSUPPORTED'
   }
   Waiting for the debugger to disconnect...
   ```

   修改方案：在package.json中修改dev部分

   ```
   "dev":"export  NODE_OPTIONS=--openssl-legacy-provider && vue-cli-service serve"
   ```

6. 配置修改

   vue.config.js

   ```
   16  const port = 8888
   30    lintOnSave: false,
   34      open: false,
   ```

默认打开端口号，open：false关闭默认打开的浏览器，lintOnSave：不断的语法修改受不了，改为false

```
devServer: {
  open: false,
  ...
  before: require('./mock/mock-server')
  //默认是否使用mock加载数据，暂时保留
}
```

Get-page-title.js

这里的title进行改动

```
const title = defaultSettings.title
```

src\settings.js

```
3 | title: '神盾局特工管理系统',
```

src\router\index.js

```
54 | | | meta: { title: '神盾局特工管理系统', icon: 'dashboard' }
```

7. 重启测试

4. 登录页修改

1. 中文描述

   在 src/views/login/index.vue中进行修改

   然后对

```
<div class="tips">
     <span style="margin-right:20px;">username: admin</span>
     <span> password: any</span>
</div>
```

   这段提示进行删除

2. 背景图

   图片放在assets里面，然后修改.login-container

```
background-image: url('../../assets/bg.jpeg');
```

   登录框调整

```
.login-container {
  min-height: 100%;
  width: 100%;
  background-color: $bg;
  background-image: url(../../assets/bg.jpg);
  background-position: center;
  background-size: 100%;
  background-repeat: no-repeat;
  overflow: hidden;
  display: flex;
  align-items: center;

  .login-form {
    position: relative;
    width: 520px;
    max-width: 100%;
    padding: 35px 35px 10px;
    margin: 0 auto;
    background-color: #2d3a4b;
    border-radius: 8px;
    opacity: 0.9;
    overflow: hidden;
  }

```

使用background-size: 100%默认100%平铺，display: flex使表单平铺，border-radius代表圆角

3. 登录用户名取消限制

   src\utils\validate.js

```
export function validUsername(str) {
  /* const valid_map = ['admin', 'editor']
  return valid_map.indexOf(str.trim()) >= 0 */
  return true;
}
```

如果输入admin123的时候，会提示请输入正确的用户名

因为在utils/auth.js中有个验证

```
const validateUsername = (rule,value,callback) => {
   if (!validUsername(value)) {
      callback(new Error('请输入正确的用户名'))
   }
   else
   {
      callback()
   }
}
```

5. 修改右上角用户下拉菜单

这个对应的页面在src/layout/components/Navbar.vue中

src > layout > components > Navbar.vue > {} "Navbar.vue" > template > div.navbar > div.right-menu > el-dropdown.avatar-container > el-dropdown-menu.user-dropdown > a

```
<el-dropdown class="avatar-container" trigger="click">
  <div class="avatar-wrapper">
    <img :src="avatar+'?imageView2/1/w/80/h/80'" class="user-avatar">
    <i class="el-icon-caret-bottom" />
  </div>
  <el-dropdown-menu slot="dropdown" class="user-dropdown">
    <router-link to="/">
      <el-dropdown-item>
        个人信息
      </el-dropdown-item>
    </router-link>
    <a target="_blank" href="https://www.baidu.com">
      <el-dropdown-item>公司主页</el-dropdown-item>
    </a>
    <el-dropdown-item divided @click.native="logout">
      <span style="display:block;">注销</span>
    </el-dropdown-item>
  </el-dropdown-menu>
</el-dropdown>
```

6. 首页面包屑导航

src\components\Breadcrumb\index.vue

```
matched = [{ path: '/dashboard', meta: { title: '首页' }}].concat(matched)
```

7. 菜单初始化

1. src\views下删除
2. 在src\views目录下创建sys模块目录、test模块目录（充数用，后续可用作权限分配测试）
3. 在sys下创建user.vue、role.vue两个组件文件

   在test下创建test1.vue、test2.vue、test3.vue
4. **修改路由配置，在src/router/index.js中**

```
{
    path: '/sys',
    component: Layout,
    redirect: '/sys/user',
    name: 'sys',
    meta: { title: '系统管理', icon: 'sys' },
    //这个icon图标必须放在icons/svg/eye.svg里面
    children: [
```

```
      {
        path: 'user',
        name: 'user',
        component: () => import('@/views/sys/user'),
        meta: { title: '用户管理', icon: 'userManage' }
      },
      {
        path: 'role',
        name: 'role',
        component: () => import('@/views/sys/role'),
        meta: { title: '角色管理', icon: 'roleManage' }
      }
    ]
  },

  {
    path: '/test',
    component: Layout,
    redirect: '/test/test1',
    name: 'test',
    meta: { title: '功能测试', icon: 'form' },
    children: [
      {
        path: 'test1',
        name: 'test1',
        component: () => import('@/views/test/test1'),
        meta: { title: '测试点一', icon: 'form' }
      },
      {
        path: 'test2',
        name: 'test2',
        component: () => import('@/views/test/test2'),
        meta: { title: '测试点二', icon: 'form' }
      },
      {
        path: 'test3',
        name: 'test3',
        component: () => import('@/views/test/test3'),
        meta: { title: '测试点三', icon: 'form' }
      }
    ]
  }
```

图标svg文件可上 https://www.iconfont.cn/ 下载

并且可以在src/sys/role.vue中修改对应路由的显示内容

```
  ```javascript
  <template>
```
```

```
    <div>
      角色管理，开发中...
    </div>
  </template>

  <script>
  export default {

  }
  </script>

  <style>

  </style>
  ```
```

## 修改一下dashboard的标题内容(src/components/Breadcrumb/index.vue)

```
if (!this.isDashboard(first)) {
  matched = [{ path: '/dashboard', meta: { title: '标题' }}].concat(matched)
}
```

8. 标签栏导航

1. **@/layout/components/AppMain.vue**
```

src > layout > components > AppMain.vue > { } "AppMain.vue" > style

```vue
<template>
  <section class="app-main">
    <transition name="fade-transform" mode="out-in">
      <keep-alive :include="cachedViews">
        <router-view :key="key" />
      </keep-alive>
    </transition>
  </section>
</template>

<script>
export default {
  name: 'AppMain',
  computed: {
    key() {
      return this.$route.path
    },
    cachedViews() {
      return this.$store.state.tagsView.cachedViews
    }
  }
}
</script>
```

```vue
<keep-alive :include="cachedViews">
    <router-view :key="key" />
</keep-alive>
```

```
cachedViews() {
    return this.$store.state.tagsView.cachedViews
}
```

2. **复制vue-element-admin项目中的文件到相应的目录中，从哪个文件里面拿取的就放入到哪个目录文件下去**

   @/layout/components/TagsView
   @/store/modules/tagsView.js
   @/store/modules/permission.js

3. **修改文件@store/getters.js**

src > store > JS getters.js > ...

```
const getters = {
  sidebar: state => state.app.sidebar,
  device: state => state.app.device,
  token: state => state.user.token,
  avatar: state => state.user.avatar,
  name: state => state.user.name,

  visitedViews: state => state.tagsView.visitedViews,
  cachedViews: state => state.tagsView.cachedViews,
  permission_routes: state => state.permission.routes
}
```

```
visitedViews: state => state.tagsView.visitedViews,
cachedViews: state => state.tagsView.cachedViews,
permission_routes: state => state.permission.routes
```

4. **修改文件@store/index.js**

src > store > JS index.js > ...

```
import Vue from 'vue'
import Vuex from 'vuex'
import getters from './getters'
import app from './modules/app'
import settings from './modules/settings'
import user from './modules/user'
import tagsView from './modules/tagsView'

Vue.use(Vuex)

const store = new Vuex.Store({
  modules: {
    app,
    settings,
    user,
    tagsView
  },
  getters
})
```

5. **修改文件@\layout\index.vue**

src > layout > index.vue > { } "index.vue" > script > default > components > TagsView

```vue
<template>
  <div :class="classObj" class="app-wrapper">
    <div v-if="device==='mobile'&&sidebar.opened" class="drawer-bg" @click="handleCli
    <sidebar class="sidebar-container" />
    <div class="main-container">
      <div :class="{'fixed-header':fixedHeader}">
        <navbar />
        <tags-view />
      </div>
      <app-main />
    </div>
  </div>
</template>

<script>
import { Navbar, Sidebar, AppMain ,TagsView} from './components'
import ResizeMixin from './mixin/ResizeHandler'

export default {
  name: 'Layout',
  components: {
    Navbar,
    Sidebar,
    AppMain,
    TagsView
  },
```

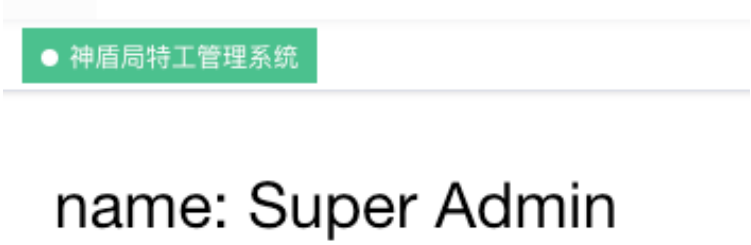
6. **修改文件@layout\components\index.js**

   新增

```
export { default as TagsView } from './TagsView'
```

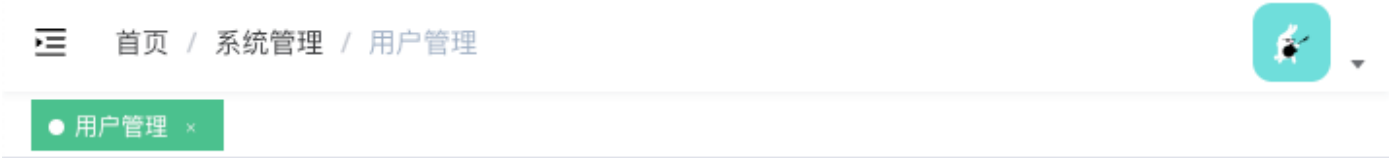
7. **Affix 固钉**

   当在声明路由是 添加了 Affix 属性，则当前tag会被固定在 tags-view中（不可被删除），也就是这里定义了首页不能够被关闭

8. 

```
46  {
47    path: '/',
48    component: Layout,
49    redirect: '/dashboard',
50    children: [{
51      path: 'dashboard',
52      name: 'Dashboard',
53      component: () => import('@/views/dashboard/index'),
54      meta: { title: '首页', icon: 'dashboard', affix:true }
55    }]
56  },
```

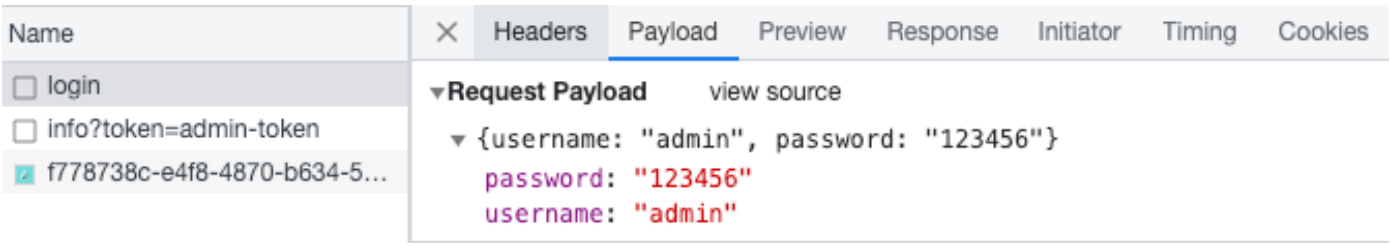
这些都加入完成之后，点开页面会出现相应的选项卡

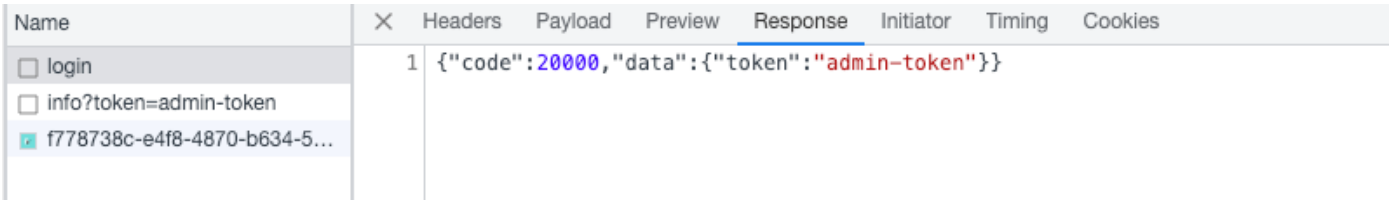


用户管理，开发中...

9. 登录接口梳理

接口	url	method
登录	/user/login	post
获取用户信息	/user/info	get
注销	/user/logout	post





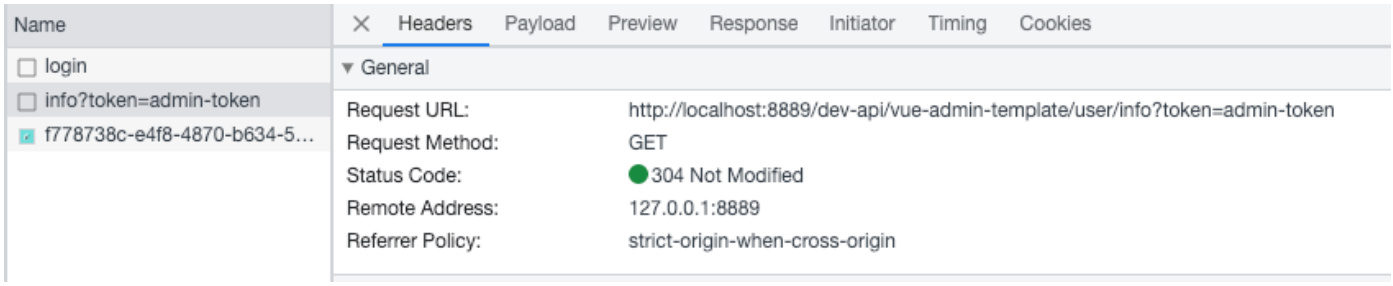
登录的时候调用下面的1和2两个接口

1.输入的时候login携带用户名和密码，返回code以及token值

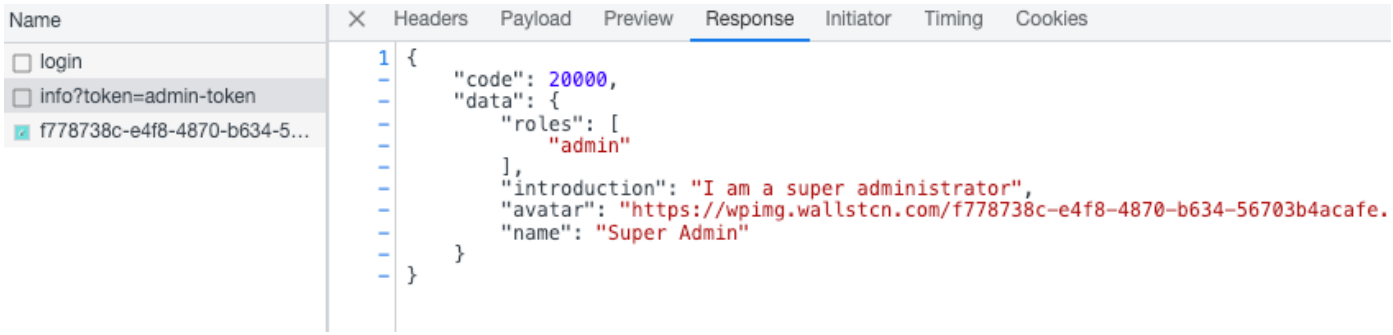
```
{"code":20000,"data":{"token":"admin-token"}}
```

2.输入的时候info返回角色和介绍

info传入的时候带了个token过来，后端根据token验证是否有效



如果有效返回一些数据给前端

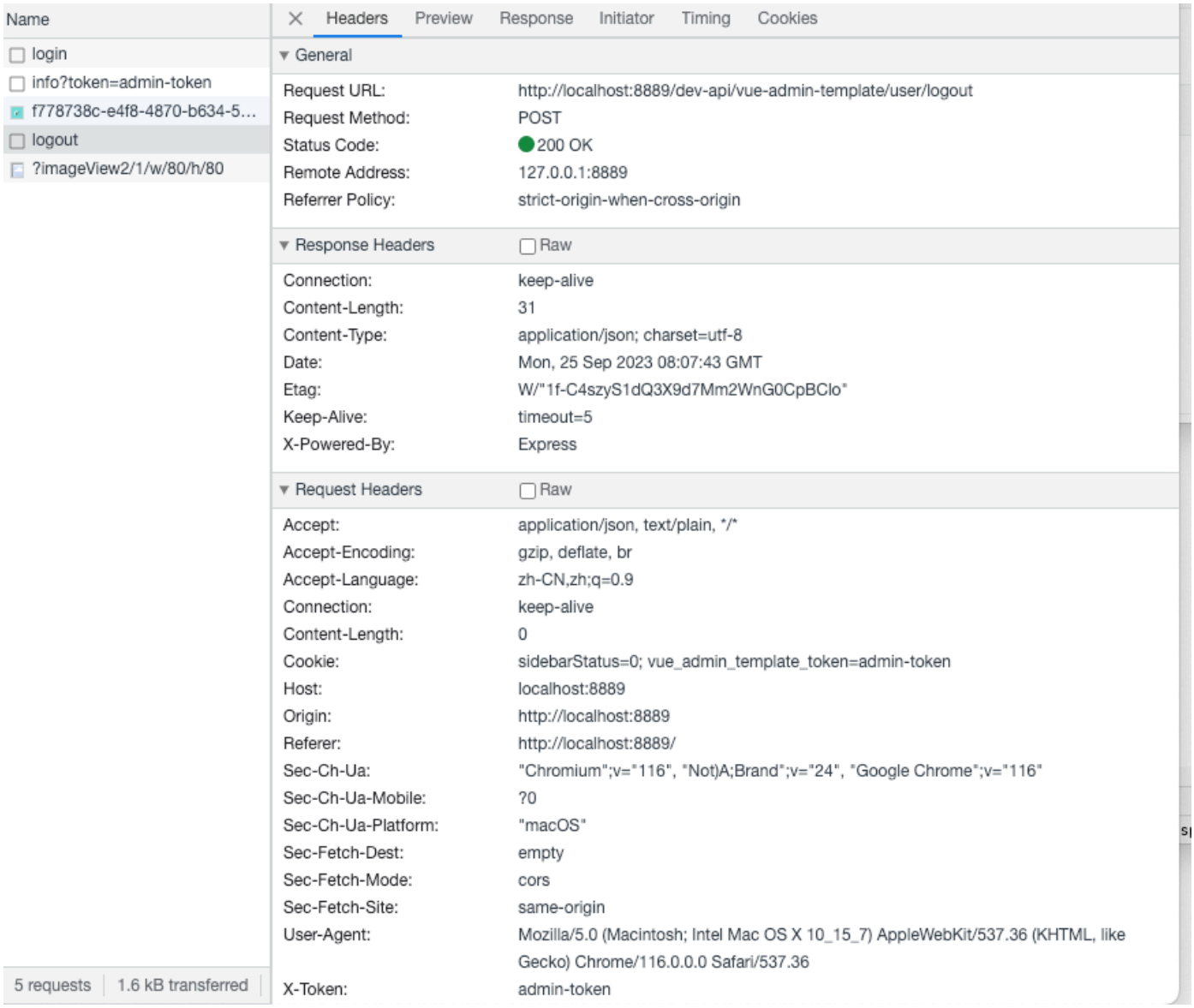



```
{"code":20000,"data":{"roles":["admin"],"introduction":"I am a super administrator","avatar":"https://wpimg.wallstcn.com/f778738c-e4f8-4870-b634-56703b4acafe.gif","name":"Super Admin"}}
```


其中avatar为头像


```
{"code":20000,"data":"success"}
```


3.注销接口



注销接口也会在返回头中带一个token，在X-Token中，后台可以拿到token后针对清除处理并返回给前端

这里进入后端教程之中！处理完成之后再返回前端！

10. 对接后端接口

1. 修改 .env.development 中的base api，打包部署的话要修改.env.production

```
VUE_APP_BASE_API = 'http://localhost:9999'
```

这里发送的请求时候会将两个网址进行拼接，VUE_APP_BASE_API和请求的src/api/user.js中的请求拼接在一起

```
export function login(data) {
  return request({
    url: '/user/login',
    method: 'post',
    data
  })
}

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

这里的http://localhost:8899/user/login就是完整的向后端发送的url地址了

2. 修改vue.config.js(在根目录下面)，屏蔽mock请求

```
39    //before: require('./mock/mock-server.js')
```

此时打开前后端界面，发送请求之后会报错

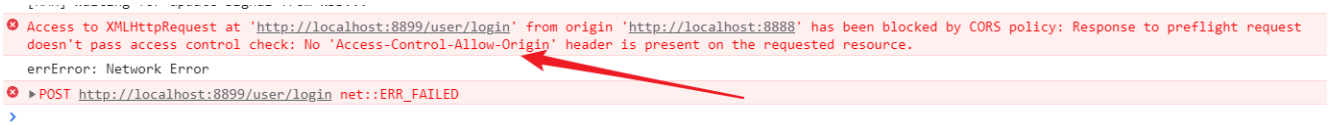
意味着存在着跨域的问题，因为端口不一致，它认为是存在跨域的问题，回到后端部分

跨域问题解决：1.在前端axios进行反向代理2.在后端进行处理

3. 修改src\api\user.js，将url中的/vue-admin-template去掉

```
export function login(data) {
  return request({
    url: '/user/login',
    method: 'post',
    data
  })
}

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

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

1. 测试，预期会出现跨域错误

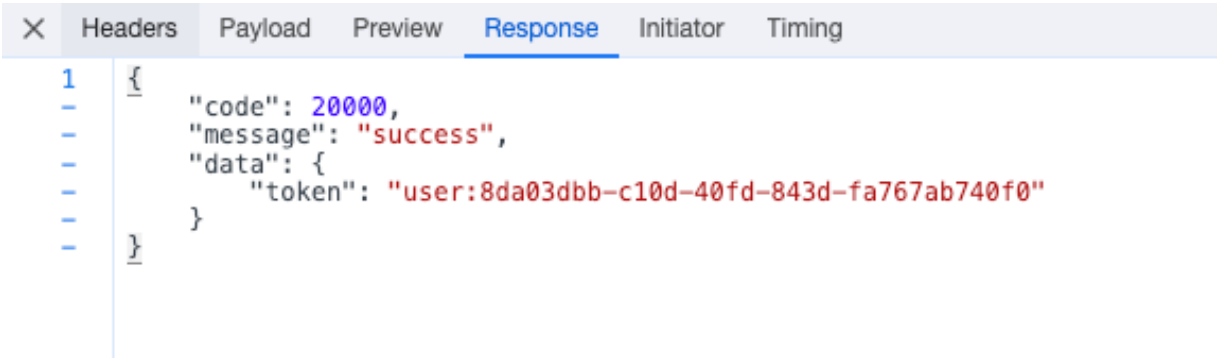


2. 后端做跨域处理测试应该成功，并可在调试窗口观察接口调用情况

!!!注意点：1.返回500很有可能是redis没开。2.在MyCorsConfig中只能使用config.addAllowedOriginPattern(" * ")而不能使用config.addAllowedOrigin(" * ")3.config.addAllowedOrigin("http://localhost:8888")这里面的8888必须为前端的端口号

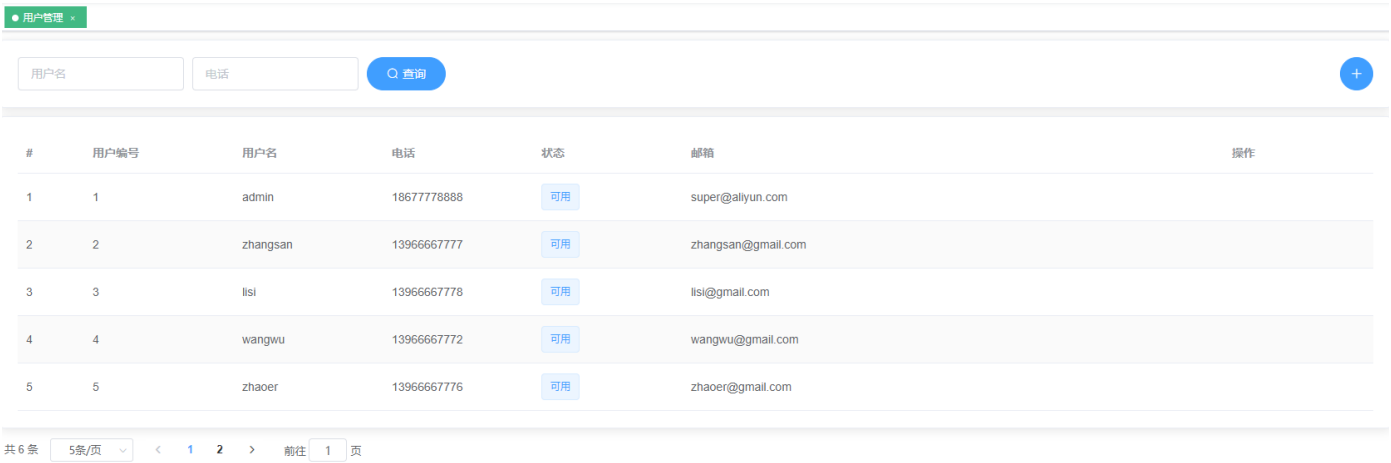
其中vue的端口在package.json中配置，而spring的端口在application.yml中配置，注意不要漏掉@Configuration和@Bean

返回的Response内容



11. 用户管理

预览



界面的话前面有一些查询的按钮，上面是一些查询的条件，下面则是显示的内容，最下面为分页，右上角有一个新增的按钮，在官网上看el-card的卡片用法

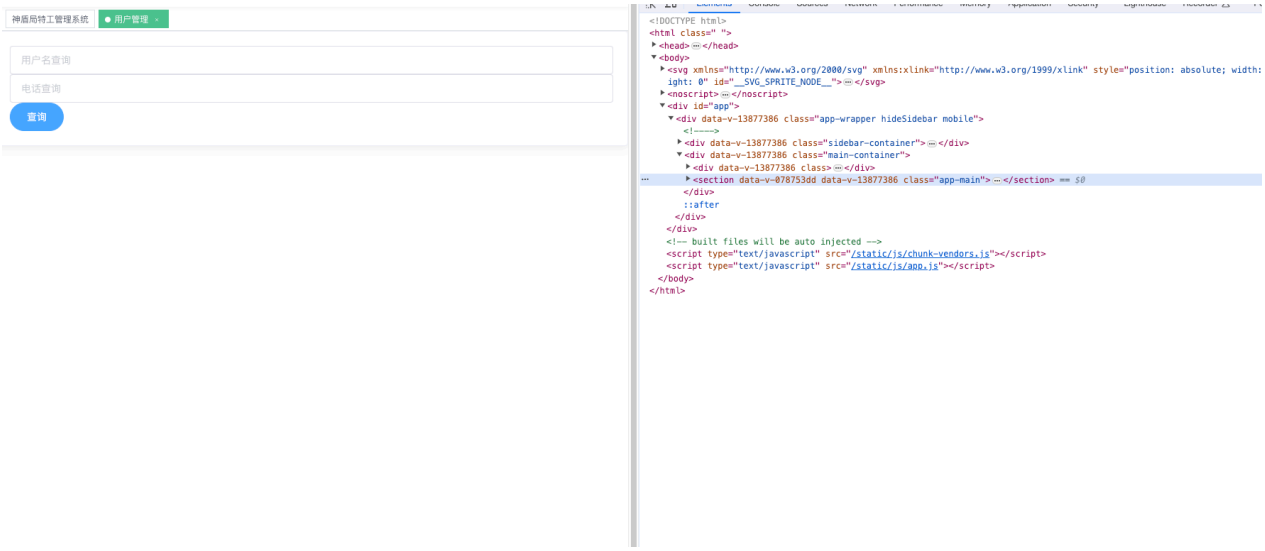
打开前端，找到user.vue文件

- **用户查询**

1. 定义src/sys/user.vue

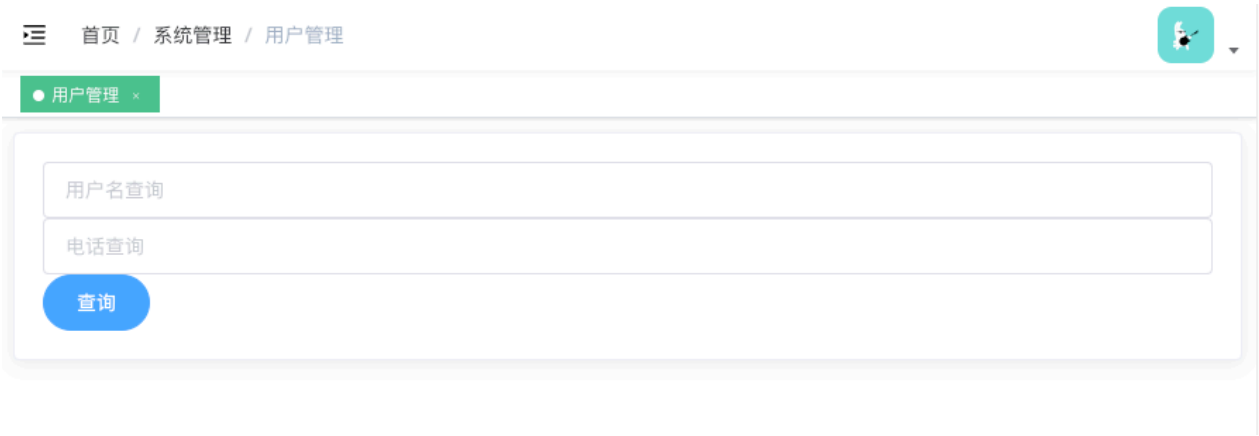
```
<template>
    <div>
        <!--搜索栏-->
        <el-card>
            <el-input v-model="input" placeholder="用户名查询"></el-input>
            <el-input v-model="input" placeholder="电话查询"></el-input>
            <el-button type="primary" round>查询</el-button>
        </el-card>
    </div>
</template>

<script>
export default {

}
</script>

<style>

</style>
```

2. 定义边框两侧的间距，在App.vue中，因为App.vue为它的父类



这里为选中为app-main，因此可以定义app-main的样式

```
<style>
  .app-main{
    padding: 10px
  }
</style>
```



3. 修改width，让不同组件能够放在同一页上
4. 加上前往按钮的组件布局(sys/user.vue)

```
<template>
    <div>
        <!--搜索栏-->
        <el-card id="search">
            <el-row>
              <el-col :span="20">
                <el-input v-model="searchModel.username" placeholder="用户名查
询"></el-input>
                <el-input v-model="searchModel.phone" placeholder="电话查询">
</el-input>
                <el-button type="primary" round icon="el-icon-search">查询</el-
button>
```

```
                <!--icon可以在官网上查询对应图标-->
              </el-col>
              <el-col :span="4" align="right">
    `           <el-button type="primary" icon="el-icon-plus" circle ></el-
button>
              </el-col>
            </el-row>
        </el-card>

        <!--结果列表-->
        <!--在这里加入element-ui中的表格-->
        <el-card>
            <el-table :data="userList" stripe style="width: 100%">
              <el-table-column type="index" label="#" width="180">

              </el-table-column>
              <el-table-column prop="id" label="用户ID" width="180">

              </el-table-column>
              <el-table-column prop="username" label="用户名" width="180">

              </el-table-column>
              <el-table-column prop="phone" label="电话" width="180">

              </el-table-column>
              <el-table-column prop="email" label="电子邮件">

              </el-table-column>
              <!--这里prop显示什么值就显示什么属性-->
              <el-table-column prop="操作" width="180">

              </el-table-column>
            </el-table>
        </el-card>

        <!--分页组件-->
        <template>
            <div class="block">
              <span class="demonstration">显示总数</span>
              <el-pagination
                @size-change="handleSizeChange"
                @current-change="handleCurrentChange"
                :current-page="searchModel.pageNo"
                :page-sizes="[5,10,20,50]"
                :page-size="searchModel.pageSize"
                layout="total, sizes, prev, pager, next, jumper"
                :total="total">
              </el-pagination>
            </div>
```

```
        </template>
    </div>
</template>

<script>
export default {
    data(){
        return {
            total: 0,
            searchModel:{
                pageNo: 1,
                pageSize: 10
            },
            userList:[]
        }
    },
    methods:{
        handleSizeChange(){

        },
        handleCurrentChange(){

        },

    }
}
</script>

<style>
#search .el-input{
    width: 200px;
    margin-right: 10px;
}
</style>
```

5. 发现组件是英文，修改为中文，修改main.js

```
import locale from 'element-ui/lib/locale/lang/en' // lang i18n
```

```
import locale from 'element-ui/lib/locale/lang/zh-CN'
```

写完之后的界面

从上面看返回的数据应该包含list和total

6. 后端返回值之后前端处理

首先在src/api/userManager.js中定义方法

```
import request from '@/utils/request'
export default{
  /*
  searchModel:{
    pageNo: 1,
    pageSize: 10
  }
   */
  getUserList(searchModel){
    return request({
      url: '/user/list',
      method: 'get',
      params: {
        pageNo: searchModel.pageNo,
        pageSize: searchModel.pageSize,
        username: searchModel.username,
        phone: searchModel.phone
      }
    });
  }
}
```

注意这里的params别打错，否则pageNo和pageSize都为null就会报page为null

接下来修改src/sys/user.vue中的代码

```
methods:{
    handleSizeChange(){

    },
    handleCurrentChange(){

    },
    getUserList(){
        userApi.getUserList(this.searchModel).then(response => {
```

```
            this.userList = response.data.rows;
            this.total = response.data.total;
        });
    }
},
created(){
    this.getUserList();
}
```

接下来查询按钮也绑定userApi.getUserList方法

7. 分页序号处理

处理到这一步的时候，分页操作还无法完成，因为src/views/sys/user.vue方法中的handleSizeChange和handleCurrentChange还没有定义

```
<el-pagination
  @size-change="handleSizeChange"
  @current-change="handleCurrentChange"
  ......
>
</el-pagination>
```

这里在src/views/sys/user.vue中定义

```
handleSizeChange(pageSize){
            this.searchModel.pageSize = pageSize;
            this.getUserList();
        },
        handleCurrentChange(pageNo){
            this.searchModel.pageNo = pageNo;
            this.getUserList();
        },
        /*
        这里修改之后能够生效的原因在于在分页组件中调用了pageSize和pageNo两个属性
        <!--分页组件-->
        <template>
            <div class="block">
              <span class="demonstration">显示总数</span>
              <el-pagination
                @size-change="handleSizeChange"
                @current-change="handleCurrentChange"
                :current-page="searchModel.pageNo"
                :page-sizes="[5,10,20,50]"
                :page-size="searchModel.pageSize"
                layout="total, sizes, prev, pager, next, jumper"
                :total="total">
              </el-pagination>
```

```
            </div>
        </template>
        */
```

如果报错

```
Error in v-on handler: "TypeError: handler.apply is not a function"
```

这是界面定义出错了

@current-page="searchModel.pageNo"错写成:current-page="searchModel.pageNo"

这样加完之后存在问题，第二页的第一行记录编号为1

此时我们需要修改一下label="#"下面的计算方式

(pageNo-1)*pageSize+index+1，因此在user.vue中配置

```
<!--结果列表-->
<!--在这里加入element-ui中的表格-->
<el-card>
    <el-table :data="userList" stripe style="width: 100%">
      <el-table-column type="index" label="#" width="180">
        <template slot-scope="scope">
            <!--(pageNo-1) * pageSize + index + 1-->
            {{(searchModel.pageNo-1) * searchModel.pageSize + scope.$index +
1}}
            <!--这里通过scope.$index拿到当前行的索引号-->
        </template>
      </el-table-column>
      <el-table-column prop="id" label="用户ID" width="180">

      </el-table-column>
      <el-table-column prop="username" label="用户名" width="180">
```

```
      </el-table-column>
      <el-table-column prop="phone" label="电话" width="180">

      </el-table-column>
      <el-table-column prop="email" label="电子邮件">

      </el-table-column>
      <el-table-column prop="操作" width="180">

      </el-table-column>
    </el-table>
</el-card>
```

加入按钮清除功能

```
<el-input v-model="searchModel.username" placeholder="用户名查询" clearable></el-input>
<el-input v-model="searchModel.phone" placeholder="电话查询" clearable></el-input>
```

文本框后面会出现x按键，点击叉号可清除文本框的内容，这里面v-model为属性绑定的属性名

- **用户新增**

  点击新增按钮需要触发一个新增的对话框，里面有输入用户名、密码等内容，这里使用element-ui中的dialogue对话框

```
<!--用户信息编辑对话框，点击右上角的添加按钮之后会自动地弹出-->
<el-dialog :title="title" :visible.sync="dialogFormVisible">
    <!--注意这里使用的如果为:，则后面调用的是title变量，如果没有加:则后面直接调用的是title，visible.sync为表单是否可见-->
    <el-form :model="userForm">
      <el-form-item label="用户名" :label-width="formLabelWidth">
        <!--label-width指的是到左边框的宽度，v-model为绑定的变量-->
        <el-input v-model="userForm.username" autocomplete="off"></el-input>
      </el-form-item>
      <el-form-item label="登录密码" :label-width="formLabelWidth">
        <!--label-width指的是到左边框的宽度，v-model为绑定的变量-->
        <el-input type="password" v-model="userForm.password" autocomplete="off">
</el-input>
      </el-form-item>
    <el-form-item label="联系电话" :label-width="formLabelWidth">
        <!--label-width指的是到左边框的宽度，v-model为绑定的变量-->
        <el-input v-model="userForm.phone" autocomplete="off"></el-input>
    </el-form-item>
    <el-form-item label="用户状态" :label-width="formLabelWidth">
        <!--label-width指的是到左边框的宽度，v-model为绑定的变量-->
        <el-switch
        v-model="userForm.status"
        :active-value="1"
        :inactive-value="0">
```

```
        </el-switch>
    </el-form-item>
    <el-form-item label="电子邮件" :label-width="formLabelWidth">
        <!--label-width指的是到左边框的宽度，v-model为绑定的变量-->
        <el-input v-model="userForm.email" autocomplete="off"></el-input>
      </el-form-item>
    </el-form>
    <div slot="footer" class="dialog-footer">
      <el-button @click="dialogFormVisible = false">取 消</el-button>
      <el-button type="primary" @click="dialogFormVisible = false">确 定</el-button>
    </div>
</el-dialog>
```

我点+号不可见，这里我需要让它可见，要修改dialogFormVisible这个属性，需要调整+号调用的方法，修改user.vue中的内容

```
<!--搜索栏-->
<el-card id="search">
    <el-row>
      <el-col :span="20">
        <el-input v-model="searchModel.username" placeholder="用户名查询" clearable>
</el-input>
        <el-input v-model="searchModel.phone" placeholder="电话查询" clearable></el-
input>
        <el-button type="primary" @click="getUserList" round icon="el-icon-search">
查询</el-button>
        <!--icon可以在官网上查询对应图标-->
      </el-col>
      <el-col :span="4" align="right">
`           <el-button @click="openEditUI" type="primary" icon="el-icon-plus"
circle ></el-button>
      </el-col>
    </el-row>
</el-card>
```

定义openEditUI的方法

```
methods:{
    openEditUI(){
        this.title = '新增用户',
        this.dialogFormVisible = true
    },
```

问题：

1. 窗口关闭后数据还在

   此时需要清除数据，有三个部分需要清理：点击叉号、点击确定、点击取消这三个位置需要清理数据

   这里对话框提供了一个关闭事件，可以通过监听close，清理表单

Events

事件名称	说明	回调参数
open	Dialog 打开的回调	—
opened	Dialog 打开动画结束时的回调	—
close	Dialog 关闭的回调	—
closed	Dialog 关闭动画结束时的回调	—

定义user.vue中的关闭方法

```
 <el-dialog @close="clearForm" :title="title"
:visible.sync="dialogFormVisible">
     ......
 </el-dialog>
```

然后清空表单的内容

```
this.userForm = {}
```

2. 表单数据验证

   常规验证

   自定义验证

   定义非空验证，还是在src/views/sys/user.vue中进行修改rules

```
<el-form :model="userForm" :rules="rules">
     ......
</el-form>
```

这里的rules可以在官网上找到，其中trigger: 'blur'表示失去焦点的时候触发

然后在每个插件上添加prop，这里下面rules中定义的规则必须跟上面的prop对应上

```
<el-form-item label="用户名" prop="username" :label-width="formLabelWidth">
  ...
</el-form-item>
```

然后在下面的data()中定义rules

```
data(){
    var checkEmail = (rule, value, callback) => {
      var reg =
```

```
        /^[a-zA-Z0-9]+([-_.][a-zA-Z0-9]+)*@[a-zA-Z0-9]+([-_.][a-zA-Z0-9]+)*\.
[a-z]{2,}$/;
      if (!reg.test(value)) {
        return callback(new Error("邮箱格式错误"));
      }
      callback();
    };
    return {
        ......
        userList:[],
        rules: {
          username: [
            { required: true, message: "请输入用户名", trigger: "blur" },
            {
              min: 3,
              max: 50,
              message: "长度在 3 到 50 个字符",
              trigger: "blur",
            },
          ],
          password: [
            { required: true, message: "请输入登录初始密码", trigger: "blur" },
            {
              min: 6,
              max: 16,
              message: "长度在 6 到 16 个字符",
              trigger: "blur",
            },
          ],
          email: [
            { required: true, message: "请输入电子邮件", trigger: "blur" },
            { validator: checkEmail, trigger: "blur" },
            //这里的验证器采用上面的checkEmail函数
          ],
        }
    }
},
```

此时鼠标移开之后就会报错：

3. 窗口关闭后上次验证结果还在

   关闭再打开之后，数据被清理了，但是窗口的提示还在

   此时查看官网，存在clearValidate方法

clearValidate	移除表单项的校验结果。传入待移除的表单项的 prop 属性或者 prop 组成的数组，如不传则移除整个表单的校验结果	Function(props: array | string)

然后我们查看调用的过程

```
submitForm(formName) {
  this.$refs[formName].validate((valid) => {
    if (valid) {
      alert('submit!');
    } else {
      console.log('error submit!!');
      return false;
```

因此这里我们需要在user.vue中定义ref

```
<el-form :model="userForm" ref="userFormRef" :rules="rules">
  ......
</el-form>
```

然后在methods中的clearForm进行处理

```
methods:{
  clearForm(){
    this.userForm = {};
    this.$refs.userFormRef.clearValidate();
  }
}
```

确定按钮的校验

如果什么都没有输入的情况下，点击确定按钮，也要能够把错误显示出来

```
<el-button type="primary" @click="saveUser">确定</el-button>
```

然后在methods中定义saveUser方法

```
saveUser(){
  //触发表单验证，如果错误会直接有提示
  this.$refs.userFormRef.validate((valid) => {
    if(valid) {
      // 提交请求给后台
```

```
        alert('submit!');
      } else {
        //
        console.log('error submit!!');
        return false;
      }
    })
    //提交请求给后台
}
```

在src/ap/userManage.js中新增请求的方法

```
addUser(user){
  //这里的user就是那个表单
  return request({
    url: '/user',
    method: 'post',
    data: user
  });
}
```

此时点击新增之后，能够看到绿色的窗口新增用户成功

如果想要新增的放在最前面，在@GetMapping("/list")中改一下排序顺序

```
wrapper.orderByDesc(User::getId);
```

- **在el-table-column的显示位置加入用户状态，如果为0的时候显示一个标签，为1的时候显示另外一个标签**

```
<el-table-column prop="status" label="用户状态" width="180">
    <template slot-scope="scope">
      <el-tag v-if="scope.row.status == 1">正常</el-tag>
      <el-tag type="danger" v-if="scope.row.status == 0">禁用 </el-tag>
    </template>
</el-table-column>
```

- **用户加密**

  见后端部分

- **用户修改**

  写完前端接口之后，定义el-table-column操作中的组件

```
</el-table-column>
<el-table-column prop="操作" width="180">
  <template slot-scope="scope">
    <el-button type="primary" icon="el-icon-edit" circle ></el-button>
    <el-button type="danger" icon="el-icon-delete circle"></el-button>
  </template>
</el-table-column>
```

接下来需要点击编辑按钮之后，能够看到界面，首先配置前端调用后端的接口，在src/api/userManager.js中配置端口

```
getUserById(id){
  return request({
    url: `/user/${id}`,
    method: 'get'
  });
},
updateUser(user){
  return request({
    url: `/user`,
    method: 'put',
    data: user
  });
}
//!!!注意这里`引号的方向!!!
```

然后需要定义el-button的点击事件，在操作按钮中定义openEditUI函数的点击事件

```
<el-table-column prop="操作" width="180">
  <template slot-scope="scope">
    <el-button @click="openEditUI(scope.row.id)" type="primary" icon="el-icon-edit"
size="mini" circle></el-button>
    <el-button type="danger" icon="el-icon-delete" size="mini" circle></el-button>
  </template>
</el-table-column>
```

修改一下之前右上角+号的点击事件

```
<el-col :span="4" align="right">
`   <el-button @click="openEditUI(null)" type="primary" icon="el-icon-plus" circle
></el-button>
</el-col>
```

两个组件共享登录密码，因此需要定义当id为null或者id未被定义的时候，登录密码这一栏不显示

```
<el-form-item v-if="userForm.id == null || userForm.id == undefined" label="登录密码" prop="password" :label-width="formLabelWidth">
  <!--label-width指的是到左边框的宽度，v-model为绑定的变量-->
  <el-input type="password" v-model="userForm.password" autocomplete="off"></el-input>
</el-form-item>
```

同时目前由于确定按钮是共享的，因此点击确定按钮会新增数据，需要点击确定时候的调用逻辑，当user.id为null的时候，说明数据库不存在当前的user，因此为添加user，而当user.id不为null的时候为修改

```
<!--用户信息编辑对话框，点击右上角的添加按钮之后会自动地弹出的窗口布局-->
<el-dialog @close="clearForm" :title="title" :visible.sync="dialogFormVisible">
    <!--注意这里使用的如果为:，则后面调用的是title变量，如果没有加:则后面直接调用的是title，visible.sync为表单是否可见-->
    <el-form :model="userForm" ref="userFormRef" :rules="rules">
      <el-form-item label="用户名" prop="username" :label-width="formLabelWidth">
        <!--label-width指的是到左边框的宽度，v-model为绑定的变量-->
        <el-input v-model="userForm.username" autocomplete="off"></el-input>
      </el-form-item>
      <el-form-item v-if="userForm.id == null || userForm.id == undefined" label="登录密码" prop="password" :label-width="formLabelWidth">
        <!--label-width指的是到左边框的宽度，v-model为绑定的变量-->
        <el-input type="password" v-model="userForm.password" autocomplete="off">
</el-input>
      </el-form-item>
    <el-form-item label="联系电话" prop="phone" :label-width="formLabelWidth">
        <!--label-width指的是到左边框的宽度，v-model为绑定的变量-->
        <el-input v-model="userForm.phone" autocomplete="off"></el-input>
    </el-form-item>
    <el-form-item label="用户状态" prop="status" :label-width="formLabelWidth">
        <!--label-width指的是到左边框的宽度，v-model为绑定的变量-->
        <el-switch
        v-model="userForm.status"
        :active-value="1"
        :inactive-value="0">
        </el-switch>
    </el-form-item>
    <el-form-item label="电子邮件" prop="email" :label-width="formLabelWidth">
        <!--label-width指的是到左边框的宽度，v-model为绑定的变量-->
        <el-input v-model="userForm.email" autocomplete="off"></el-input>
      </el-form-item>
    </el-form>
    <div slot="footer" class="dialog-footer">
      <el-button @click="dialogFormVisible = false">取 消</el-button>
      <el-button type="primary" @click="saveUsers">确 定</el-button>
    </div>
</el-dialog>
```

然后定义saveUsers的方法，根据传入参数的不同进行不同的调用

注意!!!Error in v-on handler: "TypeError: _api_userManagerWEBPACK_IMPORTED_MODULE_2.default.saveUser is not a function"一般为方法没有被定义或者方法跳转错误!!!

```
saveUsers() {
  //触发表单验证
  this.$refs.userFormRef.validate((valid) => {
    if(valid) {
      //提交请求给后台
      this.saveUser(this.userForm).then(response => {
        // 成功提示，需要关闭对话框，并且刷新表格
        this.$message({
          message: response.message,
          type: 'success'
        });
      })};
      //关闭对话框
      this.dialogFormVisible = false;
      //刷新表格
      this.getUserList();
  })
},

saveUser(user){
  if(user.id == null || user.id == undefined)
  {
    return userApi.addUser(user);
  }
  return userApi.updateUser(user);
  //目前问题：添加按钮和编辑按钮都无法点击
},
```

- **用户删除**

  先写前端的删除方法

```
//!!!注意这里`引号的方向!!!
deleteUserById(id){
  return request({
    url: `/user/${id}`,
    method: 'delete'
  });
}
```

然后在操作的时候把删除用户的方法绑定上去

```
<el-table-column prop="操作" width="180">
  <template slot-scope="scope">
    <el-button @click="openEditUI(scope.row.id)" type="primary" icon="el-icon-edit"
size="mini" circle></el-button>
    <el-button @click="deleteUser(scope.row)" type="danger" icon="el-icon-delete"
size="mini" circle></el-button>
  </template>
```

这里传入scope.row整行的信息是因为想要提示删除的用户对象是谁

然后定义deleteUser方法，此处需要一个确认消息的弹窗