

复习

- 使用 elementui 完成 vue 前端登录页面布局
- 利用 axios 发请求,完成登录
- 登录成功使用路由跳转主页面
- 主页面布局,菜单,显示登录人名,退出按钮
- 在 main 主区域设置卡片,内置表格准备展现数据库数据

主页表格展示数据

了解表格组件

```

<template>
<div>
  <el-card>
    <!--el-table 表格组件-->
    <!--:data=tableData 是绑定表格数据,在下方 data 中定义,是一个数组类型-->
    <!--数组是对象,下面 prop 就是对象的属性-->
    <el-table :data="tableData" border style="width: 100%">
      <!--一个 el-table-column,就是一列-->
      <el-table-column prop="date" label="日期" width="280"></el-table-column>
      <el-table-column prop="name" label="姓名" width="180"></el-table-column>
      <el-table-column prop="address" label="地址"></el-table-column>
      <el-table-column prop="birthday" label="生日"></el-table-column>
    </el-table>
  </el-card>
</div>
</template>

```

```

<script>
export default {
  name: "StuInfoView",
  data() {
    return {
      // 此处就是表格中的数据,现在是写固定的
      // 后续就需要从数据库查询返回到前端
      // 前端再展现
      tableData: [{
        date: '2016-05-02',
        name: '王小虎',
        address: '上海市普陀区金沙江路 1518 弄',
        birthday: '1111-11-11'
      }, {
        date: '2016-05-04',
        name: '王小虎',
        address: '上海市普陀区金沙江路 1517 弄'
      }, {
        date: '2016-05-01'
      }
    ]
  }
}

```

前端发送请求查询全部用户

问题: 什么时候发请求?

思路: 页面**加载时同时就发请求**查后端数据,等页面加载完就展现数据

如果实现: vue 框架有生命周期函数,会在页面加载时自动触发函数,那就可以在这个函数这发请求查数据

```
<script>
// 导入 axios
import axios from "axios";
export default {
  name: "StuInfoView",
  // data 是定义的数据变量
  data() {
    return {
      // 此处就是表格中的数据,现在是写固定的
      // 后续就需要从数据库查询返回到前端
      // 前端再展现
      tableData: []
    }
  },
  // vue 声明周期函数,在 vue 对象创建完毕时自动触发
  created() {
    // 在此处发请求
    axios.get("http://localhost:8888/stu/list")
      .then(res => {
        console.log("后端返回的数据", res)
      })
  }
}
```

后端接收请求

- 后端 StudentController 定义一个方法,接收前端的请求
- 调用 studentMapper 查询数据库,获得全部数据
- 返回给前端

// StudentController 接收请求

```
/**
 * 接收前端请求, 查询全部数据, 返回集合
 * 返回值 List<Student>, 是一个链表集合, 其中存储学生对象
 */
@RequestMapping("/stu/list")
public List<Student> findAll(){
    List<Student> list = studentMapper.findAll( );
    System.out.println(list );
    return list;
}
```

// StudentMapper.java 接口中添加方法

```
/**
 * 查询全部
 */
List<Student> findAll();
```

// StudentMapper.xml 文件中添加 sql

```
<select id="findAll" resultType="com.qf.model.Student">
    select * from tb_stu
</select>
```

```

<template>
<div>
  <el-card>
    <!--el-table 表格组件-->
    <!--:data=tableData 是绑定表格数据,在下方 data 中定义,是一个数组类型-->
    <!--数组是对象,下面 prop 就是对象的属性-->
    <el-table :data="tableData" border style="width: 100%">
      <!--一个 el-table-column,就是一列-->
      <!--prop 的值是根据后端返回的数据定的-->
      <el-table-column prop="id" label="学号" ></el-table-column>
      <el-table-column prop="username" label="姓名" ></el-table-
column>
      <el-table-column prop="password" label="密码"></el-table-
column>
      <el-table-column prop="age" label="年龄"></el-table-column>
      <el-table-column prop="sex" label="性别"></el-table-column>
      <el-table-column prop="score" label="分数"></el-table-
column>
      <el-table-column prop="birthday" label="生日"></el-table-
column>
    </el-table>
  </el-card>
</div>
</template>

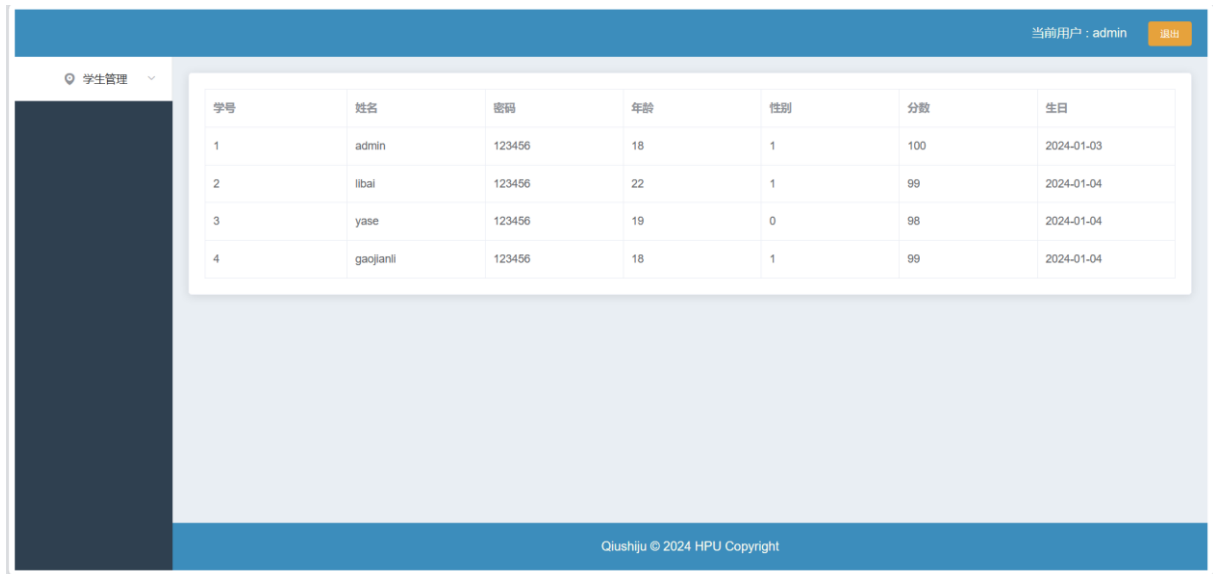
```

```

<script>
// 导入 axios
import axios from "axios";
export default {
  name: "StuInfoView",
  // data 是定义的数据变量
  data() {
    return {
      // 此处就是表格中的数据,现在是写固定的
      // 后续就需要从数据库查询返回到前端
      // 前端再展现
      tableData: []
    }
  },

```

// 最终效果



学号	姓名	密码	年龄	性别	分数	生日
1	admin	123456	18	1	100	2024-01-03
2	libai	123456	22	1	99	2024-01-04
3	yase	123456	19	0	98	2024-01-04
4	gaojianli	123456	18	1	99	2024-01-04

Qiushiju © 2024 HPU Copyright

日期显示 bug

给后端 Student.java 类的 birthday 属性上面加上@JsonFormat(pattern = "yyyy-MM-dd",timezone = "GMT+8")

```
public class Student {  
  
    // private是修饰符,私有的意思  
    private int id;  
    // int是整型,String是字符串,double浮点型  
    private String username;  
    private String password;  
    private int age;  
    private int sex;  
    private double score;  
    // java.util.Date的不要导错包  
    // Date也是数据类型,代表日期  
    @JsonFormat(pattern = "yyyy-MM-dd",timezone = "GMT+8")  
    private Date birthday;  
}
```

分页展示数据

了解分页 sql 语句

select * from tb_stu 这是全部数据


```
-- 查询全部数据
select * from tb_stu

-- 查询数据条数
select count(*) from tb_stu

-- 分页查询,就是使用 limit 语句
-- limit x,y
-- x 是第 x 行从 0 开始,y 是多少条
-- 规定每页 10 条,查第一页
select * from tb_stu limit 0,10
-- 第二页
select * from tb_stu limit 10,10

-- 多少页可以算出来,总页数/每页多少条数据
```

了解前端分页组件

分页组件文档 <https://element.eleme.cn/#/zh-CN/component/pagination>

```

<div style="margin-top: 10px">
<el-card style="height: 50px;text-align: center">
  <!--
    el-pagination 分页组件
      @size-change 事件,页面尺寸变化时触发函数
      @current-change 事件,当前页面改变时触发
      :current-page ,变量,当前页码,下方 data 中定义了变量
      :page-sizes ,提供页面可以大小的选项
      :page-size , 默认页面大小
      layout="total, sizes, prev, pager, next, jumper"
      layout 是布局显示,total 显示总条数,sizes 显示可选大小,prev 上一
      页,pager 页码,next 下一页,jumper 跳转按钮
      :total , 变量,显示数据总条数
  -->
  <el-pagination
    @size-change="handleSizeChange"
    @current-change="handleCurrentChange"
    :current-page="currentPage"
    :page-sizes="[5,10, 15, 20]"
    :page-size="pageSize"
    layout="total, sizes, prev, pager, next, jumper"
    :total="total">
  </el-pagination>
</el-card>
</div>

```

```

data() {
  return {
    // 此处就是表格中的数据,现在是写固定的
    // 后续就需要从数据库查询返回到前端
    // 前端再展现
    tableData: [],
    currentPage: 1, // 默认当前页 1
    total: 0, // 默认总条数 0
    pageSize: 10
  }
},
methods: {
  // 当页面大小时触发
  handleSizeChange(val) {
    console.log(`每页 ${val} 条`);
  },
  // 当页码改变时触发
  handleCurrentChange(val) {
    console.log(`当前页: ${val}`);
  }
},

```

发请求分页查询

问题: 什么时候发,在哪里发请求分页查询?

思路: 就在页面加载时,就直接分页查询,即修改上次查询全部的代码,添加请求参数

```

// vue声明周期函数,在vue对象创建完毕时自动触发
created(){
  // 在此处发请求
  axios.get( url: "http://localhost:8888/stu/list", config: {
    params:{
      currentPage : this.currentPage,
      pageSize : this.pageSize
    }
  })
}

```

后端接收分页查询请求

问题: 哪里接收?

思路: 改动之前查询全部的代码,加上两个参数接收前端发送请求参数(当前页,页面大小)

```
@RequestMapping("/stu/list")  
public List<Student> findAll(int currentPage,int pageSize){  
    System.out.println("currentPage = " + currentPage);  
    System.out.println("pageSize = " + pageSize);  
    List<Student> list = studentMapper.findAll( );  
    System.out.println(list );  
    return list;  
}
```

问题: 如何完成分页 sql 语句编写?

解决: 我们不需要编写,但是需要加一个框架,帮我们实现分页,它就是 pagehelper

步骤: 1 加依赖 2 开启分页 3 返回数据

1. pom.xml 导入依赖

```
<!-- pageHelper 依赖-->  
<dependency>  
    <groupId>com.github.pagehelper</groupId>  
    <artifactId>pagehelper-spring-boot-starter</artifactId>  
    <version>1.4.2</version>  
</dependency>
```

2. 开始分页

3. 收集分页数据,返回给 前端

```

/**
 * 接收前端请求, 查询全部数据, 返回集合
 * 返回值 List<Student>, 是一个链表集合, 其中存储学生对象
 * 添加两个参数 int currentPage, int pageSize
 * 返回值变成 PageInfo<Student>
 */
@RequestMapping("/stu/list")
public PageInfo<Student> findAll(int currentPage, int
pageSize) {
    System.out.println("currentPage = " + currentPage);
    System.out.println("pageSize = " + pageSize);

    /**
     * 在查询全部前, 设置分页
     * 参数 1 当前页码
     * 参数 2 页面大小
     * -----
     * PageHelper 工具会自动帮我拼接 limit 语句, 我们就不需要改 sql
     */
    PageHelper.startPage(currentPage, pageSize);
    // 开查询
    List<Student> list = studentMapper.findAll( );
    System.out.println(list );
    // 收集分页信息
    PageInfo<Student> info = new PageInfo<>(list);
    return info;
}

```

前端获得数据,展示

```
// vue声明周期函数,在vue对象创建完毕时自动触发
created(){
  // 在此处发请求
  axios.get( url: "http://localhost:8888/stu/list", config: {
    params:{
      currentPage : this.currentPage,
      pageSize : this.pageSize
    }
  })
  .then(res => {
    console.log("后端返回的数据",res)
    // 将后端数据赋值给表格数组
    this.tableData = res.data.list;
    this.pageSize = res.data.pageSize;
    this.total = res.data.total;
    this.currentPage = res.data.pageNum;
  })
},
```

实现点击页码翻页

需求: 点击页码翻页

思路: 分页组件中@current-change="handleCurrentChange" 页码改变时触发函数 **handleCurrentChange**,在函数中重新发请求查询一遍

发现问题: 重新查一遍代码会重复,所以抽取查询代码,封装成函数 **getData()**,复用

```
67 methods: {
68   // 定义一个函数,用于查询数据,复用
69   getData(){
70     // 在此处发请求
71     axios.get( url: "http://localhost:8888/stu/list", config: {
72       params:{
73         currentPage : this.currentPage,
74         pageSize : this.pageSize
75       }
76     }).then(res => {
77       console.log("后端返回的数据",res)
78       // 将后端数据赋值给表格数组
79       this.tableData = res.data.list;
80       this.pageSize = res.data.pageSize;
81       this.total = res.data.total;
82       this.currentPage = res.data.pageNum;
83     })
84   },
85   // 当页面大小时触发
86   handleSizeChange(val) {
87     console.log(` 每页 ${val} 条`);
88   },
89   // 当页码改变时触发
90   handleCurrentChange(val) {
91     console.log(` 当前页: ${val} `);
92     // 改变当前页码
93     this.currentPage = val;
94     this.getData();
95   }
96 }
```

实现改变页面大小查询

需求: 改变页面大小,比如每页 10 条,20 条这样.....

思路： 分页组件中@size-change="handleSizeChange" 页面改变时触发函数
重新查一遍

```
},  
// 当页面大小时触发  
handleSizeChange(val) {  
  console.log(`每页 ${val} 条`);  
  this.pageSize = val;  
  this.getData();  
},
```

改变性别显示

```
<el-table-column prop="sex" label="性别">  
  <template slot-scope="scope">  
    <span style="margin-left: 10px">  
      <el-tag v-if="scope.row.sex == 0">男</el-tag>  
      <el-tag v-if="scope.row.sex == 1" type="warning">女</el-tag>  
    </span>  
  </template>  
</el-table-column>
```

前端完整代码

```

<template>
<div>
  <el-card>
    <!--el-table 表格组件-->
    <!--:data=tableData 是绑定表格数据,在下方 data 中定义,是一个数组类型-->
    <!--数组是对象,下面 prop 就是对象的属性-->
    <el-table :data="tableData" border style="width: 100%" max-height="450">
      <!--一个 el-table-column,就是一列-->
      <!--prop 的值是根据后端返回的数据定的-->
      <el-table-column prop="id" label="学号" ></el-table-column>
      <el-table-column prop="username" label="姓名" ></el-table-column>
      <el-table-column prop="password" label="密码"></el-table-column>
      <el-table-column prop="age" label="年龄"></el-table-column>
      <el-table-column prop="sex" label="性别">
        <template slot-scope="scope">
          <span style="margin-left: 10px">
            <el-tag v-if="scope.row.sex == 0">男</el-tag>
            <el-tag v-if="scope.row.sex == 1" type="warning">女
          </el-tag>
        </span>
      </template>
    </el-table-column>
    <el-table-column prop="score" label="分数"></el-table-column>
    <el-table-column prop="birthday" label="生日"></el-table-column>
  </el-table>
</el-card>
<div style="margin-top: 10px">
  <el-card style="height: 50px;text-align: center">
    <!--
    el-pagination 分页组件
    @size-change 事件,页面尺寸变化时触发函数
    @current-change 事件,当前页面改变时触发
    :current-page ,变量,当前页码,下方 data 中定义了变量
    :page-sizes ,提供页面可以大小的选项

```


当前用户: admin

学生管理

学生信息管理

宿舍信息管理

选课信息管理

成绩信息管理

学号	姓名	密码	年龄	性别	分数	生日
1	admin	123456	18	女	100	2024-01-03
2	libai	123456	22	女	99	2024-01-04
3	yase	123456	19	男	98	2024-01-04
4	gaojianli	123456	18	女	99	2024-01-04
5	zs0	111	22	男	98	2024-01-04
6	zs1	111	22	男	98	2024-01-04
7	zs2	111	22	女	98	2024-01-04

共 104 条 10条/页 < 1 2 3 4 5 6 ... 11 > 前往 1 页

Qiushiju © 2024 HPU Copyright

更新学生信息

需求: 在每一行数据后,设置更新|删除按钮,点击更新按钮,弹出对话框,其中有表单,表单内有回显的之前的数据,填完信息发送数据到数据,执行更新语句,前端对话框消失,并且页面重新查询最新的数据

设置更新|删除按钮

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

```
handleEdit(index, row) {
  console.log(index, row);
},
handleDelete(index, row) {
  console.log(index, row);
}
```

弹出对话框

点击更新按钮,弹出对话框,其中有表单,表单内有回显的之前的数据,

```

60 <!-- 对话框组件-->
61 <el-dialog
62   title="更新学生信息"
63   :visible.sync="editDialogVisible"
64   width="30%"
65   <el-form ref="stu" :model="stu" label-width="80px">
66     <el-form-item label="用户名">
67       <el-input v-model="stu.username"></el-input>
68     </el-form-item>
69     <el-form-item label="密码">
70       <el-input v-model="stu.password"></el-input>
71     </el-form-item>
72     <el-form-item label="年龄">
73       <el-input v-model="stu.age"></el-input>
74     </el-form-item>
75     <el-form-item label="分数">
76       <el-input v-model="stu.score"></el-input>
77     </el-form-item>
78     <el-form-item label="性别">
79       <el-radio-group v-model="stu.sex">
80         <el-radio :label="0">男</el-radio>
81         <el-radio :label="1">女</el-radio>
82       </el-radio-group>
83     </el-form-item>
84     <el-form-item label="生日">
85       <el-col :span="11">
86         <el-date-picker type="date" placeholder="选择日期" v-model="stu.birthday" style="width: 100%;"></el-date-picker>
87       </el-col>
88     </el-form-item>
89   </el-form>
90   <span slot="footer" class="dialog-footer">
91     <el-button @click="editDialogVisible = false">取消</el-button>
92     <el-button type="primary" @click="editDialogVisible = false">确定</el-button>

```

```

data() {
  return {
    // 此处就是表格中的数据,现在是写固定的
    // 后续就需要从数据库查询返回到前端
    // 前端再展现
    tableData: [],
    currentPage: 1, // 默认当前页1
    total: 0, // 默认总条数0,
    pageSize: 10, // 默认页面大小
    editDialogVisible: false, // 更新对话框默认不可见
    stu: {
      id: 0,
      username: "",
      password: "",
      age: 0,
      sex: 0,
      score: 0.0,
      birthday: "2000-01-01",
    }
  }
}

```

```

// 处理点击更新按钮时
handleEdit(index, row) {
  console.log(index, row);
  this.editDialogVisible = true; // 让对话框可见
  this.stu = row;
}

```

发请求执行更新

后端接收请求,处理更新

在 StudentController 加上一个方法接收请求

```

/**
 * 更新
 */
@RequestMapping("/stu/edit")
public boolean editById(@RequestBody Student student){
    System.out.println("更新用的对象"+ student );
    return false;
}

```

在 StudentMapper.java 接口中定义更新的方法

```

/**
 * 更新
 */
boolean editById(Student student);
}

```

在 StudentMapper.xml 写 sql

```

<!-- 更新 -->
<update id="editById">
    update tb_stu set
        username=#{username},
        password=#{password},
        age=#{age},
        sex=#{sex},
        score=#{score},
        birthday=#{birthday}
    where id = #{id}
</update>

```

在 StudentController 的更新方法中调用该方法

```

/**
 * 更新
 */
@RequestMapping("/stu/edit")
public boolean editById(@RequestBody Student student){
    System.out.println("更新用的对象"+ student );
    boolean b = studentMapper.editById(student);
    return b;
}

```

前端对话框消失,重查

```
// 处理更新请求
submitEdit(){
  console.log(this.stu);
  // 发请求,带数据,到后端
  axios.post( url: "http://localhost:8888/stu/edit",this.stu)
    .then(res => {
      console.log("更新后,后端返回",res)
      if (res.data){
        // 弹框提示
        this.$message( options: {
          message: '更新成功',
          type: 'success',
        });
        // 对话框消失
        this.editDialogVisible = false;
        // 重新查询
        this.getData();
      }else{
        // 弹框提示
        this.$message( options: {
          message: '更新失败',
          type: 'error',
        });
      }
    })
}
```

前端完整代码

```

<template>
<div>
  <el-card>
    <!--el-table 表格组件-->
    <!--:data=tableData 是绑定表格数据,在下方 data 中定义,是一个数组类型-->
    <!--数组是对象,下面 prop 就是对象的属性-->
    <el-table :data="tableData" border style="width: 100%" max-height="450">
      <!--一个 el-table-column,就是一列-->
      <!--prop 的值是根据后端返回的数据定的-->
      <el-table-column prop="id" label="学号" ></el-table-column>
      <el-table-column prop="username" label="姓名" ></el-table-column>
      <el-table-column prop="password" label="密码"></el-table-column>
      <el-table-column prop="age" label="年龄"></el-table-column>
      <el-table-column prop="sex" label="性别">
        <template slot-scope="scope">
          <span style="margin-left: 10px">
            <el-tag v-if="scope.row.sex == 0">男</el-tag>
            <el-tag v-if="scope.row.sex == 1" type="warning">女
          </el-tag>
        </span>
      </template>
    </el-table-column>
    <el-table-column prop="score" label="分数"></el-table-column>
    <el-table-column prop="birthday" label="生日"></el-table-column>
    <el-table-column label="操作">
      <template slot-scope="scope">
        <el-button
          type="warning"
          size="mini"
          @click="handleEdit(scope.$index, scope.row)">编辑
        </el-button>
        <el-button
          size="mini"
          type="danger"

```

删除学生信息

需求: 点击删除按钮,弹出确认框,确定是否删除,如果确定删除发请求到后端,执行删除

删除按钮弹框确认

```
// 处理删除
handleDelete(index, row) {
  console.log(index, row);
  this.$confirm( message: '此操作将永久删除该文件, 是否继续?', title: '提示', options: {
    confirmButtonText: '确定',
    cancelButtonText: '取消',
    type: 'warning'
  }).then(() => {
    this.$message( options: {
      type: 'success',
      message: '删除成功!'
    });
  }).catch(() => {
    this.$message( options: {
      type: 'info',
      message: '已取消删除'
    });
  });
},
// 处理更新请求
```

确定删除,发请求到后端

```
}).then(() => {
  // 发请求到后端
  axios.get( url: "http://localhost:8888/stu/del", config: {
    params: {
      id: row.id
    }
  }).then(res => {
    if (res.data) {
      this.$message( options: {
        type: 'success',
        message: '删除成功!'
      });
    } else {
      this.$message( options: {
        type: 'error',
        message: '删除失败!'
      });
    }
  })
}).catch(() => {
```

后端接收请求,执行删除

// StudentController 中设计方法接收请求

```
/**
 * 删除
 */
@RequestMapping("/stu/del")
public boolean delById(int id){
    System.out.println("删除用的 id"+ id );
    // 调用 mapper 完成删除
    boolean b = studentMapper.delById(id);
    return b;
}
```

// 在 StudentMapper.java 中定义删除的方法

```
/**
 * 删除
 */
boolean delById(int id);
```

// 在 StudentMapper.xml 中定义删除的 sql


```
<delete id="delById">
    delete from tb_stu where id = #{id}
</delete>
```

前端接收返回结果,弹框提示,重查

```
// 发请求到后端
axios.get( url: "http://localhost:8888/stu/del", config: {
  params: {
    id: row.id
  }
}).then(res => {
  if (res.data) {
    this.$message( options: {
      type: 'success',
      message: '删除成功!'
    });
    // 重新查询最新数据
    this.getData();
  } else {
    this.$message( options: {
      type: 'error',
      message: '删除失败!'
    });
  }
})
}).catch(() => {
```

```

<template>
<div>
  <el-card>
    <!--el-table 表格组件-->
    <!--:data=tableData 是绑定表格数据,在下方 data 中定义,是一个数组类型-->
    <!--数组是对象,下面 prop 就是对象的属性-->
    <el-table :data="tableData" border style="width: 100%" max-height="450">
      <!--一个 el-table-column,就是一列-->
      <!--prop 的值是根据后端返回的数据定的-->
      <el-table-column prop="id" label="学号" ></el-table-column>
      <el-table-column prop="username" label="姓名" ></el-table-column>
      <el-table-column prop="password" label="密码"></el-table-column>
      <el-table-column prop="age" label="年龄"></el-table-column>
      <el-table-column prop="sex" label="性别">
        <template slot-scope="scope">
          <span style="margin-left: 10px">
            <el-tag v-if="scope.row.sex == 0">男</el-tag>
            <el-tag v-if="scope.row.sex == 1" type="warning">女
          </el-tag>
        </span>
      </template>
    </el-table-column>
    <el-table-column prop="score" label="分数"></el-table-column>
    <el-table-column prop="birthday" label="生日"></el-table-column>
    <el-table-column label="操作">
      <template slot-scope="scope">
        <el-button
          type="warning"
          size="mini"
          @click="handleEdit(scope.$index, scope.row)">编辑
        </el-button>
        <el-button
          size="mini"
          type="danger"

```

添加学生信息

设置添加按钮

```
StuInfoView.vue x LoginView.vue x App.vue x AdminView.vue x
1 <template>
2 <div>
3   <el-card>
4
5     <div style="text-align: left">
6       <!-- 添加按钮 -->
7       <el-button type="primary">添加</el-button>
8     </div>
9   <!-- el-table 表格组件 -->
```

按钮-弹框

cv 之前更新弹出框,改一个控制弹出框展示/隐藏的变量记录

按钮绑定事件,点击添加按钮时 隐藏框的变量为 true,即可展示框

```
<!-- 添加对话框组件 -->
<el-dialog
  title="添加学生信息"
  :visible.sync="addDialogVisible"
  width="30%">
  <el-form ref="stu" :model="stu" label-width="80px">
    <el-form-item label="用户名">
      <el-input v-model="stu.username"></el-input>
    </el-form-item>
    <el-form-item label="密码">
```

变量,控制隐藏和显示

下面的代码省略....

```
143 import axios from "axios";
144 export default {
145   name: "StuInfoView",
146   // data是定义的数据变量
147   data() {
148     return {
149       // 此处就是表格中的数据,现在是写固定的
150       // 后续就需要从数据库查询返回到前端
151       // 前端再展现
152       tableData: [],
153       currentPage: 1, // 默认当前页1
154       total: 0, // 默认总条数0,
155       pageSize: 10, // 默认页面大小
156       editDialogVisible: false, // 更新对话框默认不可见
157       addDialogVisible: false, // 添加对话框默认不可见
158       stu: {
159         id: 0,
160         username: "",
161         password: "",
162         age: null,
```

```
<div style="text-align: left">
  <!-- 添加按钮 -->
  <el-button type="primary" @click="addDialogVisible = true">添加</el-button>
</div>
<!-- el-table 表格组件 -->
```

发送添加请求到后端

```
// 处理添加请求|
submitAdd(){
  axios.post( url: "http://localhost:8888/stu/add",this.stu)
    .then(res => {
      console.log("执行添加,后端返回结果",res)
    })
}
```

后端接收请求,执行插入

// StudentController.java 添加方法接收请求

```
/**
 * 添加
 */
@RequestMapping("/stu/add")
public boolean addStu(@RequestBody Student student){
    System.out.println("添加用的对象"+ student );
    boolean b = studentMapper.addStu(student);
    return b;
}
```

// StudentMapper.java 设置一个添加学生 的方法

```
/**
 * 添加学生
 */
boolean addStu(Student stu);
```

// StudentMapper.xml 写 sql

```
<insert id="addStu">
    insert into tb_stu
        (username,password,sex,age,score,birthday)
    values

    (#{username},#{password},#{sex},#{age},#{score},#{birthday})
</insert>
```

前端根据结果,隐藏对话框,重查

```
},
// 处理添加请求
submitAdd(){
    axios.post( url: "http://localhost:8888/stu/add",this.stu)
    .then(res => {
        console.log("执行添加,后端返回结果",res)
        if (res.data) {
            // 弹框提示
            this.$message( options: {
                message: '添加成功',
                type: 'success',
            });
            // 对话框消失
            this.addDialogVisible = false;
            // 重新查询
            this.getData();
        } else {
            // 弹框提示
            this.$message( options: {
                message: '添加失败',
                type: 'error',
            });
        }
    })
}
```

任务

我讲过的都要实现!