

# 一、回顾

---

- Vue2
  - js
  - new Vue({el:"",data(){ retrun {} },methods:{}})
  - {{ }}
  - vue的指令
    - v-for
      - v-for="(item,index) in array"
    - v-if
      - v-else-if
      - v-else
    - v-on
      - v-on:click === @click
    - v-bind
      - img v-bind:src=""
      - :src
    - v-model
      - 双向绑定
      - 表单元素
- ElementUI
  - el-button
  - el-form \*\*
    - 校验
    - 表单元素
- vue-admin-template(前端项目模板)
- jdbc
- mybatis
  - sql写到了xml映射文件中

- 写一个接口对应这个xml文件
  - 接口名称 - xml文件名称
  - 接口的全路径 --- xml的namespace一致
  - 接口的方法名 xml中sql的id一致
  - 接口的参数 xml中sql的参数一致
  - 接口的返回值 xml中sql的resultType/resultMap保持一致
- springboot
  - 提供接口，提供功能
- springboot+mybatis

## 二、本周计划

---

- 能够独立使用
  - 后端：spring boot+mybatis
  - 前端：vue+elementUI
  - 完成一些后台管理的相关功能
- 项目分组
  - 布置项目任务
  - 分析项目功能
  - 教大家如何做需求分析
- 下周项目的编写
- 项目的验收

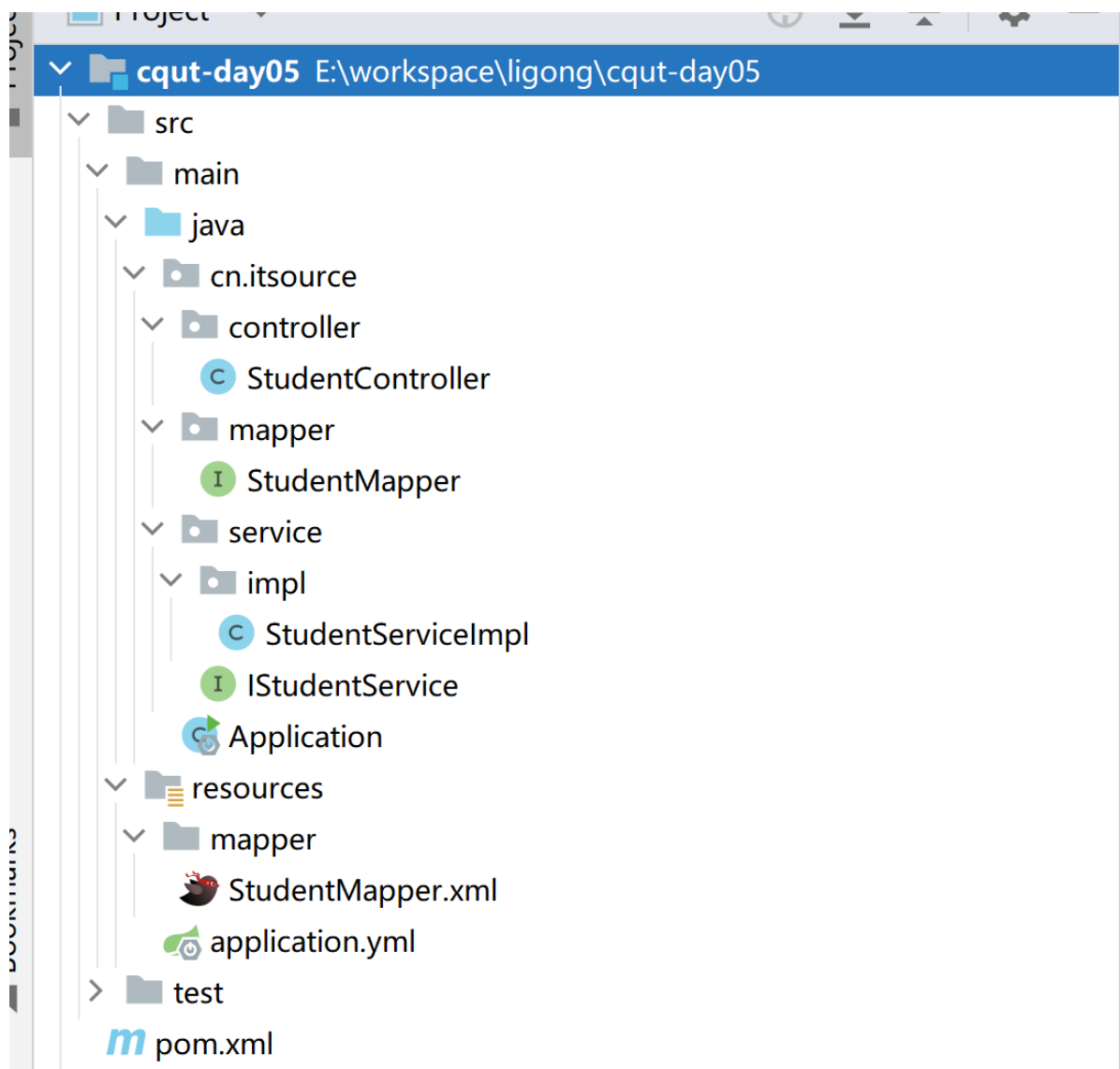
## 三、axios

---

### 1、后端的三层结构

---

controller、service（接口层、实现层）、mapper



## 2、分析

学生管理 | 教师管理 | 课程管理

添加

	姓名	性别	出生日期	手机号码	学历	状态	简介	操作
暂无数据								

- 当页面加载成功之后调用后端的<http://localhost:8080/stduent/listAll>接口查询数据
- 把查询到的结果给data中的students赋值
- vue如果stduents内容改变了，页面中也会自动渲染

## 3、axios的入门

<http://www.axios-js.com/>



## (1) 项目中安装axios

```
npm install axios
```

## (2) 在需要使用axios的组件中引入axios

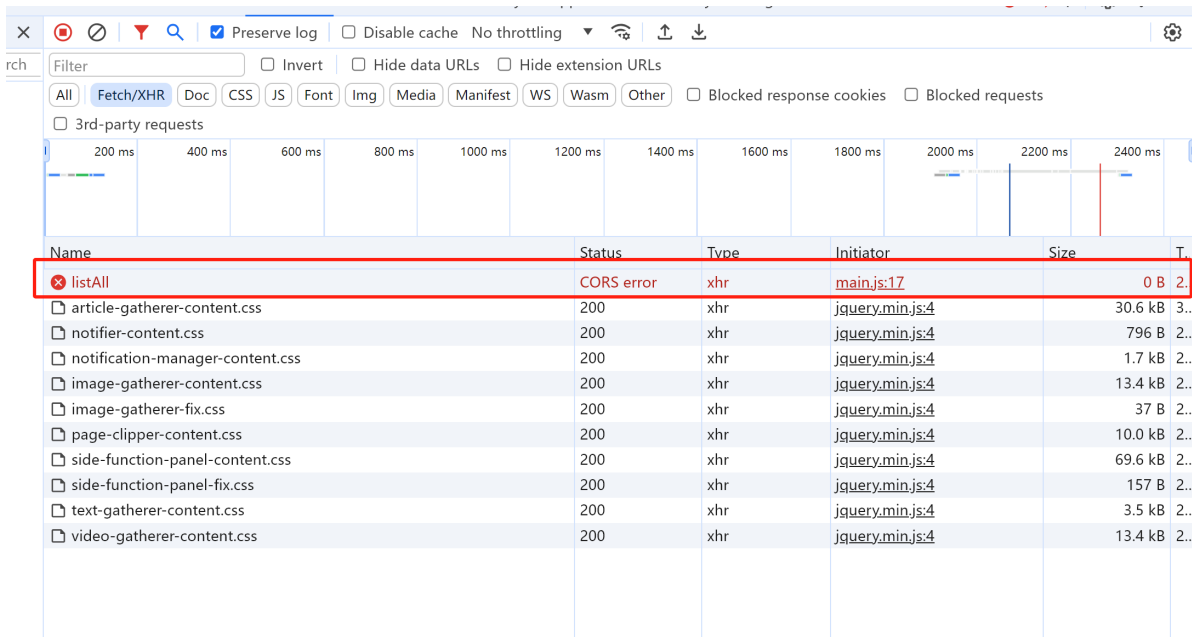
```
import axios from 'axios'
```

## (3) 封装函数，发送请求加载数据

```
loadStudents () {  
  axios.get("http://localhost:8080/student/listAll")  
  // 请求成功的回调  
  .then(res => {  
    this.students = res.data  
  })  
}
```

## (4) 在页面加载成功后执行函数

```
Student.vue 1 x App.vue
src > views > Student.vue > Vetur > {} "Student.vue"
1 <template>
102
103 <script>
104 // data和methods
105 import axios from 'axios'
106 export default {
107 >   data() { ...
162   },
163 >   methods: { ...
220   },
221 // 页面加载(挂载)成功之后执行的代码
222   mounted() {
223     this.loadStudents()
224   }
225 }
226 </script>
227 |
228 <style>
229 .myDIV{
230   color: red
231 }
232 </style>
```



The screenshot shows the Chrome DevTools Network tab with a list of requests. The first request, 'listAll', is highlighted with a red box and shows a 'CORS error' status. The table below lists the requests and their details.

Name	Status	Type	Initiator	Size	T...
listAll	CORS error	xhr	main.js:17	0 B	2...
article-gatherer-content.css	200	xhr	jquery.min.js:4	30.6 kB	3...
notifier-content.css	200	xhr	jquery.min.js:4	796 B	2...
notification-manager-content.css	200	xhr	jquery.min.js:4	1.7 kB	2...
image-gatherer-content.css	200	xhr	jquery.min.js:4	13.4 kB	2...
image-gatherer-fix.css	200	xhr	jquery.min.js:4	37 B	2...
page-clipper-content.css	200	xhr	jquery.min.js:4	10.0 kB	2...
side-function-panel-content.css	200	xhr	jquery.min.js:4	69.6 kB	2...
side-function-panel-fix.css	200	xhr	jquery.min.js:4	157 B	2...
text-gatherer-content.css	200	xhr	jquery.min.js:4	3.5 kB	2...
video-gatherer-content.css	200	xhr	jquery.min.js:4	13.4 kB	2...

## 4、跨域问题

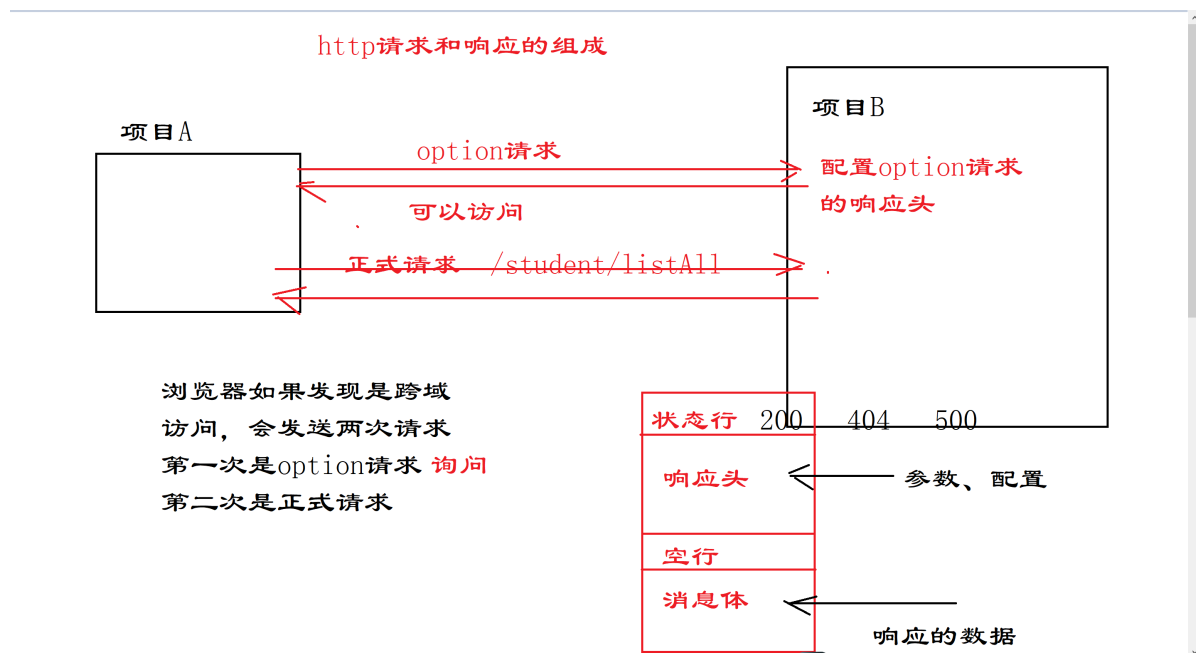
在前后端分离的项目中，一定会遇到跨域问题

浏览器的保护机制

当一个域通过ajax发送请求，请求另一个域的资源的时候，浏览器会自动拦截不让发送

192.168.0.201:8080 ---- 192.168.0.202:8080 跨域访问 跨域必然有跨域问题

192.168.0.201:8080 ---- 192.168.0.201:8081 跨域访问 跨域必然有跨域问题



我们需要在项目B（后端）中配置跨域

配置哪些域可以跨域访问我，如果你可以访问我，那么就在响应头中告诉浏览器

springboot中配置跨域：超级简单



## 5、前端全局使用和配置axios

main.js

```
Student.vue 1 | JS main.js | App.vue
src > JS main.js > ...
1 import Vue from 'vue' // 导入vue.js
2 import App from './App.vue'
3 import router from './router' // 导入的是./router/index.js
4
5 // 引入ElementUI
6 import ElementUI from 'element-ui';
7 import 'element-ui/lib/theme-chalk/index.css';
8
9 import axios from 'axios'
10 axios.defaults.baseURL = 'http://localhost:8080'
11
12 Vue.prototype.axios = axios
13
14
15 Vue.use(ElementUI);
16
17
18 Vue.config.productionTip = false
19
20
21
22 new Vue({
23   router: router, // 路由的配置
24   render: h => h(App)
25 }).$mount('#app')
26
```

axios.defaults.baseURL 配置了之后，其他所有发送请求的地方都不需要加“<http://localhost:8080>”这个前缀了

在vue组件中使用axios发送请求

不需要import axios，直接通过this.axios发送请求

```
209 ...
210 ...
211 ...
212 loadStudents() {
213   this.axios.get("/student/listAll")
214   // 请求成功的回调
215   .then(res => {
216     this.students = res.data
217   })
218 }
219 ...
220 // 页面加载(挂载)成功之后执行的代码
221 mounted() {
222   this.loadStudents()
223 }
224 }
225 </script>
226
227 <style>
```

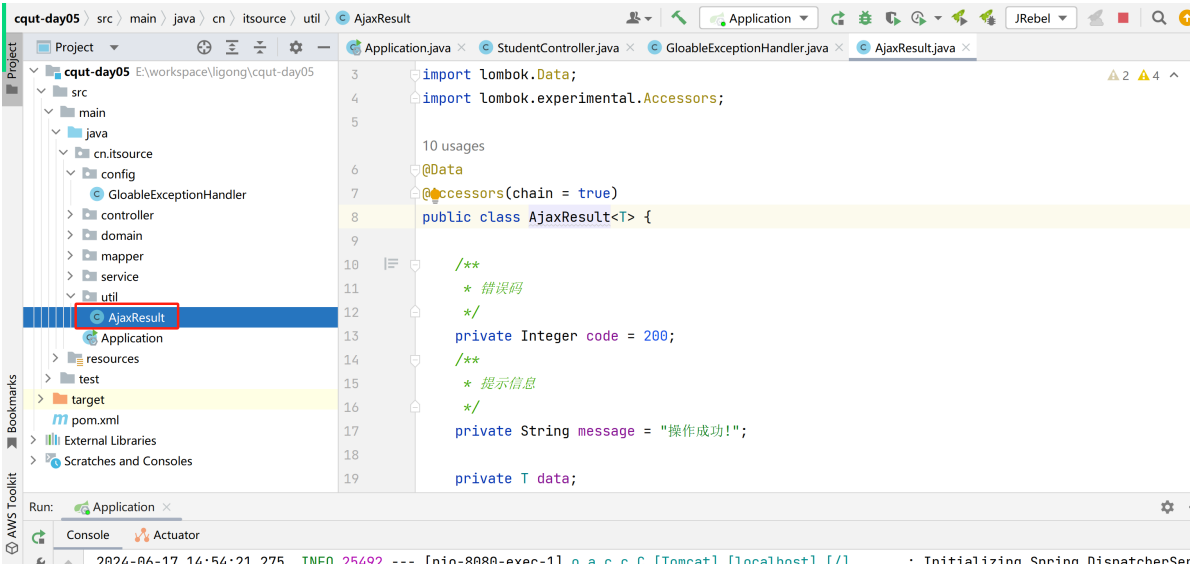
## 6、测试

<div> <div>+</div> <div>添加</div> </div>		姓名	性别	出生日期	手机号码	学历	状态	简介	操作
<input type="checkbox"/>	1	张三	男	2024-06-13	18888888888	博士后	冻结	是个狠人	<div> <div>编辑</div> <div>删除</div> </div>
<input type="checkbox"/>	2	里斯	男	2024-06-14	16666666666	大学	冻结	这个更狠	<div> <div>编辑</div> <div>删除</div> </div>
<input type="checkbox"/>	3	王五	男	2024-06-14	1999999999	中学	正常	链式编程	<div> <div>编辑</div> <div>删除</div> </div>
<input type="checkbox"/>	4	王五	女				冻结		<div> <div>编辑</div> <div>删除</div> </div>
<input type="checkbox"/>	5	田七	女		18888888888		冻结		<div> <div>编辑</div> <div>删除</div> </div>
<input type="checkbox"/>	6	田七	女		18888888888		冻结		<div> <div>编辑</div> <div>删除</div> </div>



## 四、一些封装

### 1、统一的响应格式



```

package cn.itsource.util;

import lombok.Data;
import lombok.experimental.Accessors;

@Data
@Accessors(chain = true)
public class AjaxResult<T> {

    /**

```



```
    * 错误码
    */
private Integer code = 200;
/**
    * 提示信息
    */
private String message = "操作成功!";

private T data;

// 不能通过new创建AjaxResult对象
private AjaxResult(){}

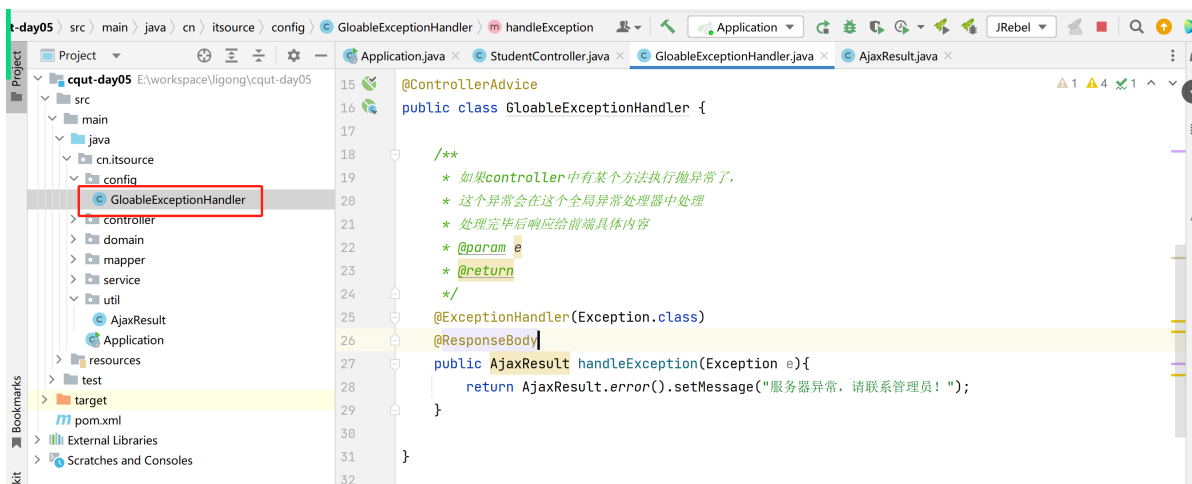
/**
    * 成功
    * @return
    */
public static AjaxResult success(){
    return new AjaxResult<>();
}

/**
    * 失败
    * @return
    */
public static AjaxResult error(){
    return new AjaxResult<>
().setCode(500).setMessage("操作失败! ");
}

}
```



## 2、全局异常处理



```
package cn.itsource.config;
```

```
import cn.itsource.util.AjaxResult;
```

```
import
```

```
org.springframework.web.bind.annotation.ControllerAdvice;
```

```
import
```

```
org.springframework.web.bind.annotation.ExceptionHandler;
```

```
import
```

```
org.springframework.web.bind.annotation.ResponseBody;
```

```
/**
```

```
 * 全局异常处理器
```

```
 * controller层如果有异常，会自动被当前处理器捕获，
```

```
 * 根据异常封装Ajax里面错误提示信息
```

```
 * spring 的 aop（感兴趣自己去了解）
```

```
 * spring中的两大核心：IOC      AOP
```

```
 */
```

```

@ControllerAdvice
public class GlobalExceptionHandler {

    /**
     * 如果controller中有某个方法执行抛异常了，
     * 这个异常会在这个全局异常处理器中处理
     * 处理完毕后响应给前端具体内容
     * @param e
     * @return
     */
    @ExceptionHandler(Exception.class)
    @ResponseBody
    public AjaxResult handleException(Exception e){
        return AjaxResult.error().setMessage("服务器异常，请联系管理员！");
    }

}

```

## 五、实现增删改

## 六、高级分页查询

### 1、分页sql

```
select * from student limit 0,2
```

limit 后面的0, 2

0 表示 开始查询的行数（从0开始）

2 表示 检索的行数，检索两条

```
select * from student limit a,b
```

每页显示2条数据

页数	a	b
第一页	0	2
第二页	2	2
第三页	4	2

找规律：

一般来说，前端在分页的时候会给到后端两个参数：

pageNum：当前的页数，第几页

pageSize：每页显示多少条数据

`select * from student limit (pageNum - 1)*pageSize, pageSize`

## 2、分页插件的渲染

<input type="checkbox"/>	日志编号	系统模块	操作类型	操作人员	操作地址	操作地点	操作状态	操作日期	消耗时间	操作
<input type="checkbox"/>	1379	代码生成	生成代码	admin	113.133.208.52	陕西省 西安市	成功	2024-06-17 15:31:29	17毫秒	详情
<input type="checkbox"/>	1378	代码生成	生成代码	admin	101.68.68.38	浙江省 杭州市	成功	2024-06-17 15:23:56	22毫秒	详情
<input type="checkbox"/>	1377	代码生成	生成代码	admin	222.240.195.20	湖南省 长沙市	成功	2024-06-17 15:12:17	20毫秒	详情
<input type="checkbox"/>	1376	代码生成	生成代码	admin	116.3.13.214	辽宁省 大连市	成功	2024-06-17 15:09:52	20毫秒	详情
<input type="checkbox"/>	1375	代码生成	生成代码	admin	124.127.75.127	北京市 北京市	成功	2024-06-17 15:07:37	22毫秒	详情
<input type="checkbox"/>	1374	代码生成	生成代码	admin	61.153.57.102	浙江省 衢州市	成功	2024-06-17 15:07:33	27毫秒	详情
<input type="checkbox"/>	1373	代码生成	生成代码	admin	113.132.8.187	陕西省 西安市	成功	2024-06-17 15:03:36	21毫秒	详情
<input type="checkbox"/>	1372	代码生成	修改	admin	117.182.45.201	广西 桂林市	成功	2024-06-17 15:03:30	17毫秒	详情
<input type="checkbox"/>	1371	代码生成	生成代码	admin	223.100.134.76	辽宁省 大连市	成功	2024-06-17 15:00:29	29毫秒	详情
<input type="checkbox"/>	1370	代码生成	生成代码	admin	218.12.16.214	河北省 石家...	成功	2024-06-17 14:53:23	20毫秒	详情

共 1280 条 10条/页

根据数据库中的总条目数进行渲染

< 1 2 3 4 5 6 ... 128 >

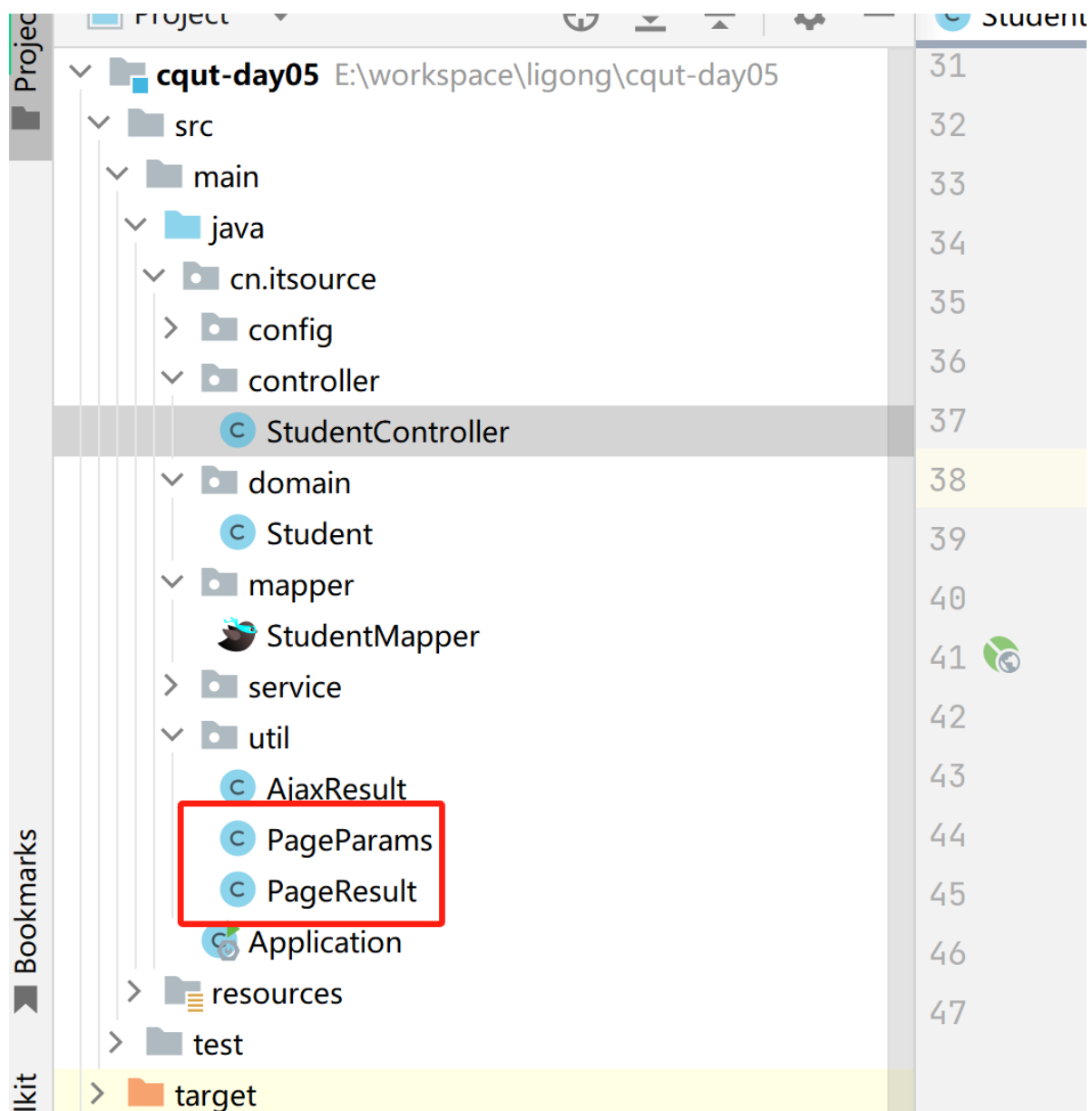
前往 1 页

后端需要执行两次查询

一次执行limit语句查询当前页的数据

一次执行select count(\*)... 查询总条目数

## 3、代码实现



- 封装实体类 - PageParams

```
package cn.itsource.util;

import lombok.Data;

@Data
public class PageParams {

    private Integer pageNum;

    private Integer pageSize;

    public Integer getStartIndex(){
        return (pageNum - 1) * pageSize;
    }
}
```

```
}
```

- 封装实体类 - PageResult

```
package cn.itsource.util;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.ArrayList;
import java.util.List;
@Data
@NoArgsConstructor
@AllArgsConstructor
public class PageResult<T> {

    /**
     * 当前页的数据
     */
    private List<T> rows = new ArrayList<>();
    /**
     * 总的条目数
     */
    private Long total = 0L;

}
```

Controller:

```
package cn.itsource.controller;

import cn.itsource.domain.Student;
import cn.itsource.service.IStudentService;
import cn.itsource.service.impl.StudentServiceImpl;
import cn.itsource.util.AjaxResult;
import cn.itsource.util.PageParams;
import cn.itsource.util.PageResult;
```

```
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@Controller
@RestController // @Controller + @ResponseBody
@RequestMapping("/student")
@CrossOrigin("*") // 后端配置跨域访问 * 表示所有的域都可以访问
public class StudentController {

    @Autowired
    private IStudentService studentService;

    /**
     * 查询所有学生
     * @return
     */
    @GetMapping("/listAll")
    public AjaxResult<List<Student>> listAll(){
        List<Student> students =
studentService.listAll();
        int num = 1/0; // 抛出异常
        return AjaxResult.success().setData(students);
    }

    /**
     * 完整的分页
     * @param pageParams
     * @return
     */
    @PostMapping("/page")
    public AjaxResult<PageResult<Student>>
page(@RequestBody PageParams pageParams){
        PageResult<Student> pageResult =
studentService.page(pageParams);
        return
AjaxResult.success().setData(pageResult);
    }
}
```

```
}
```

## IStudentService.java

```
package cn.itsource.service;

import cn.itsource.domain.Student;
import cn.itsource.util.PageParams;
import cn.itsource.util.PageResult;

import java.util.List;

public interface IStudentService {
    /**
     * 查询所有学生信息
     * @return
     */
    List<Student> listAll();

    /**
     * 分页查询
     * @param pageParams
     * @return
     */
    PageResult<Student> page(PageParams pageParams);
}
```

## StudentServiceImpl.java

```
package cn.itsource.service.impl;

import cn.itsource.domain.Student;
import cn.itsource.mapper.StudentMapper;
import cn.itsource.service.IStudentService;
import cn.itsource.util.PageParams;
import cn.itsource.util.PageResult;
import
org.springframework.beans.factory.annotation.Autowired;
```



```
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class StudentServiceImpl implements
IStudentService {

    @Autowired
    private StudentMapper studentMapper;

    /**
     * 查询所有学生信息
     * @return
     */
    @Override
    public List<Student> listAll() {
        List<Student> students =
studentMapper.selectAll();
        // 对所有学生按照需求进行一些额外的运算
        return students;
    }

    /**
     * 分页查询
     * @param pageParams
     * @return
     */
    @Override
    public PageResult<Student> page(PageParams
pageParams) {
        // 查询当前页的数据
        List<Student> students =
studentMapper.selectPage(pageParams);
        // 查询总条目数
        Long total = studentMapper.selectCount();
        PageResult<Student> pageResult = new
PageResult<>(students, total);

        return pageResult;
    }
}
```

```
}
```

## StudentMapper.java

```
package cn.itsource.mapper;

import cn.itsource.domain.Student;
import cn.itsource.util.PageParams;
import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Param;

import java.util.List;

public interface StudentMapper {

    /**
     * 查询所有学生信息
     * @return
     */
    List<Student> selectAll();

    /**
     * 分页查询
     * @param pageParams
     * @return
     */
    List<Student> selectPage(PageParams pageParams);

    /**
     * 查询总条目数
     * @return
     */
    Long selectCount();
}
```

## StudentMapper.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
```

```

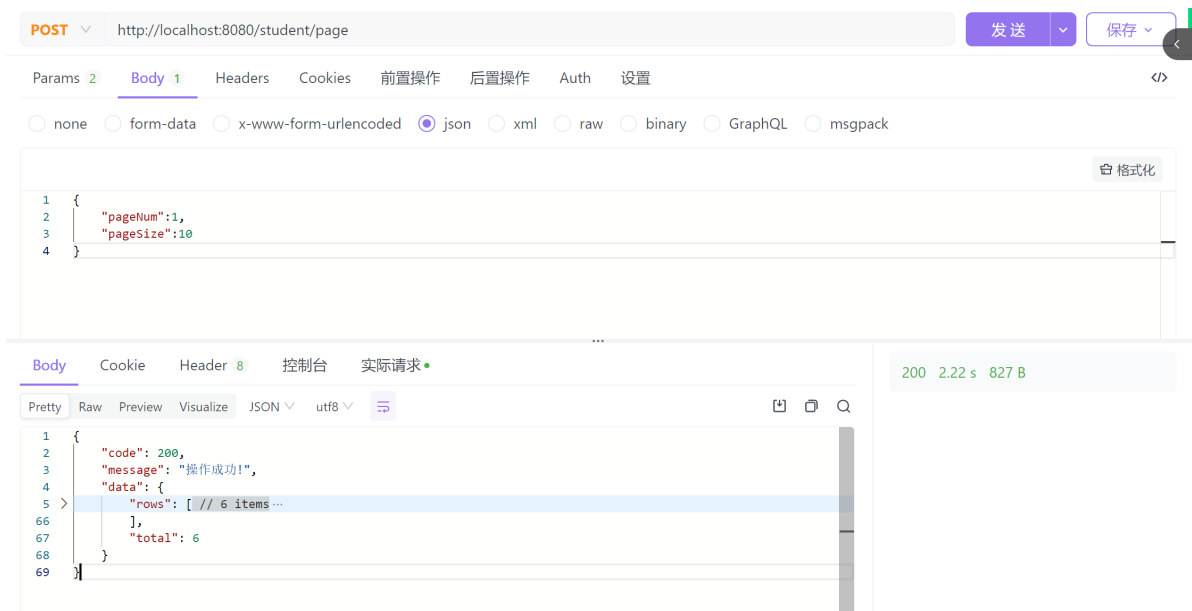
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper
3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd"
>
<mapper namespace="cn.itsource.mapper.StudentMapper">
    <select id="selectAll"
resultType="cn.itsource.domain.Student">
        select * from student
    </select>
    <!--
        #{xxx} 的原理是 调用对象的 getXxx方法
        再底层的原理就是java的反射
    -->
    <select id="selectPage"
resultType="cn.itsource.domain.Student">
        select * from student
        limit #{startIndex},#{pageSize}
    </select>
    <!--
        一个select标签只能执行一次查询
    -->
    <select id="selectCount"
resultType="java.lang.Long">
        select count(*) from student
    </select>

</mapper>

```

## 4、测试

---



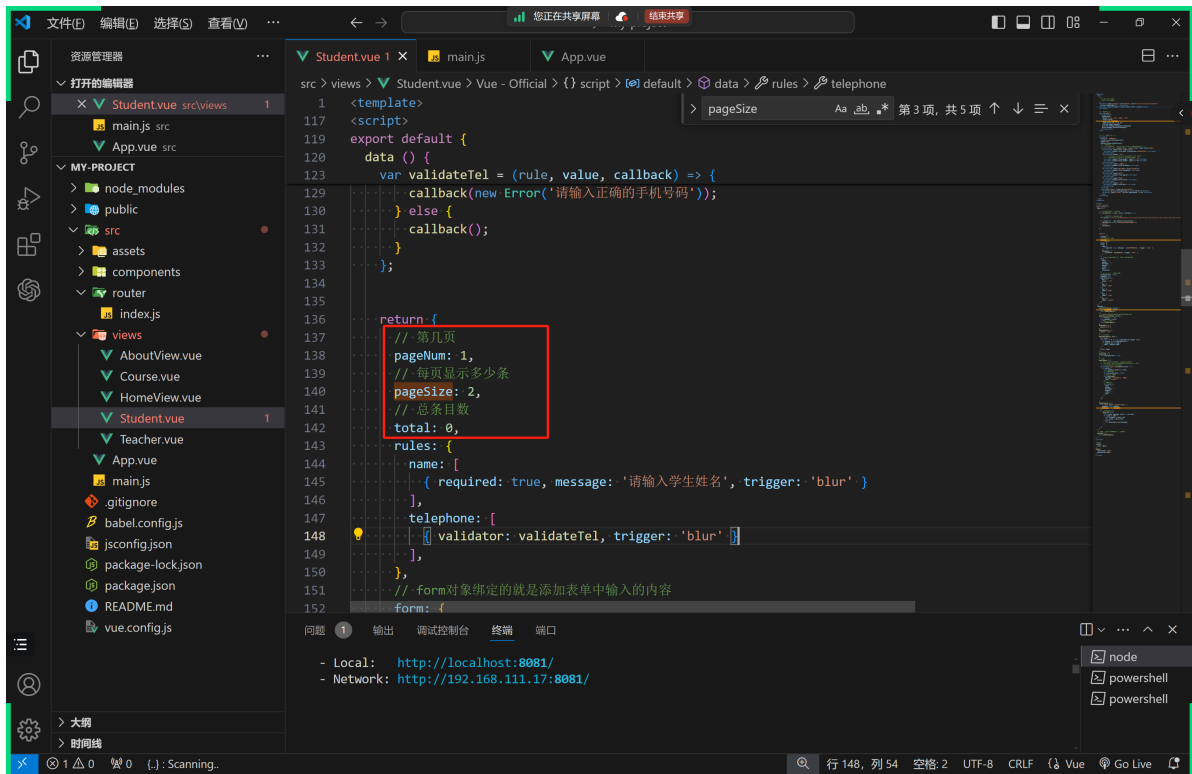
## 七、elementUI分页插件

```
<!-- 分页条 -->
<div id="page">
  <el-pagination
    background
    layout="sizes, prev, pager, next"
    :total="total"
    :page-size="pageSize"
    :page-sizes="[2, 4, 6, 8]"
    :current-page="pageNum"
    @current-change="handleCurrentChange"
    @size-change="handleSizeChange">
  </el-pagination>
</div>
```

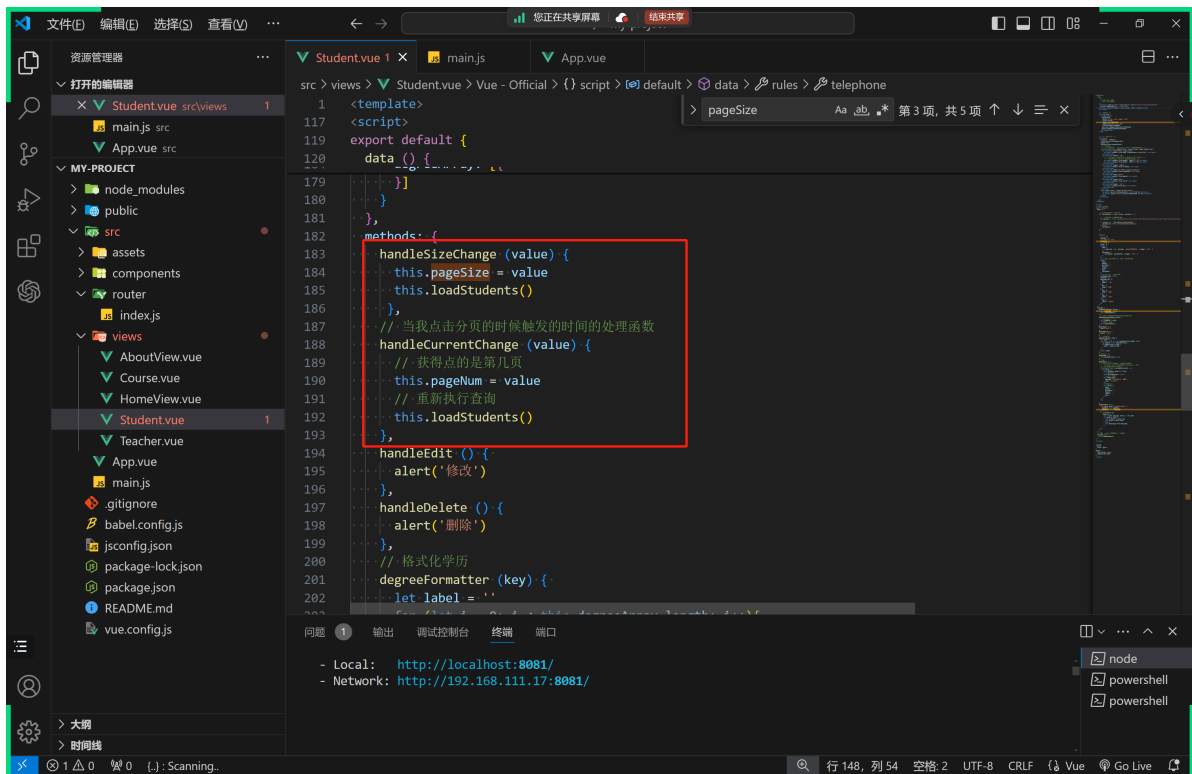
每个属性的含义对照官网去看

<https://element.eleme.io/#/zh-CN/component/pagination>

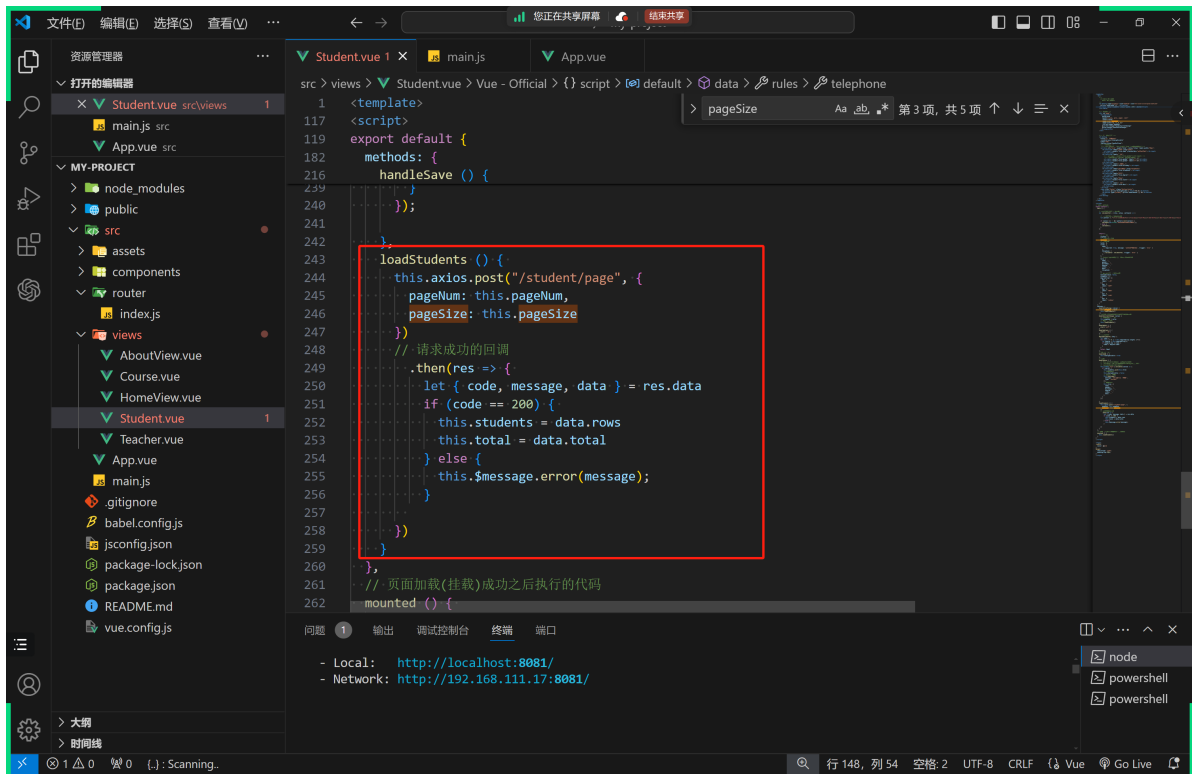
data中添加分页数据的绑定 total、pageSize、pageNum



## 监听size的改边和page的改变



## 数据查询，由原来的listAll接口改为最新的分页接口

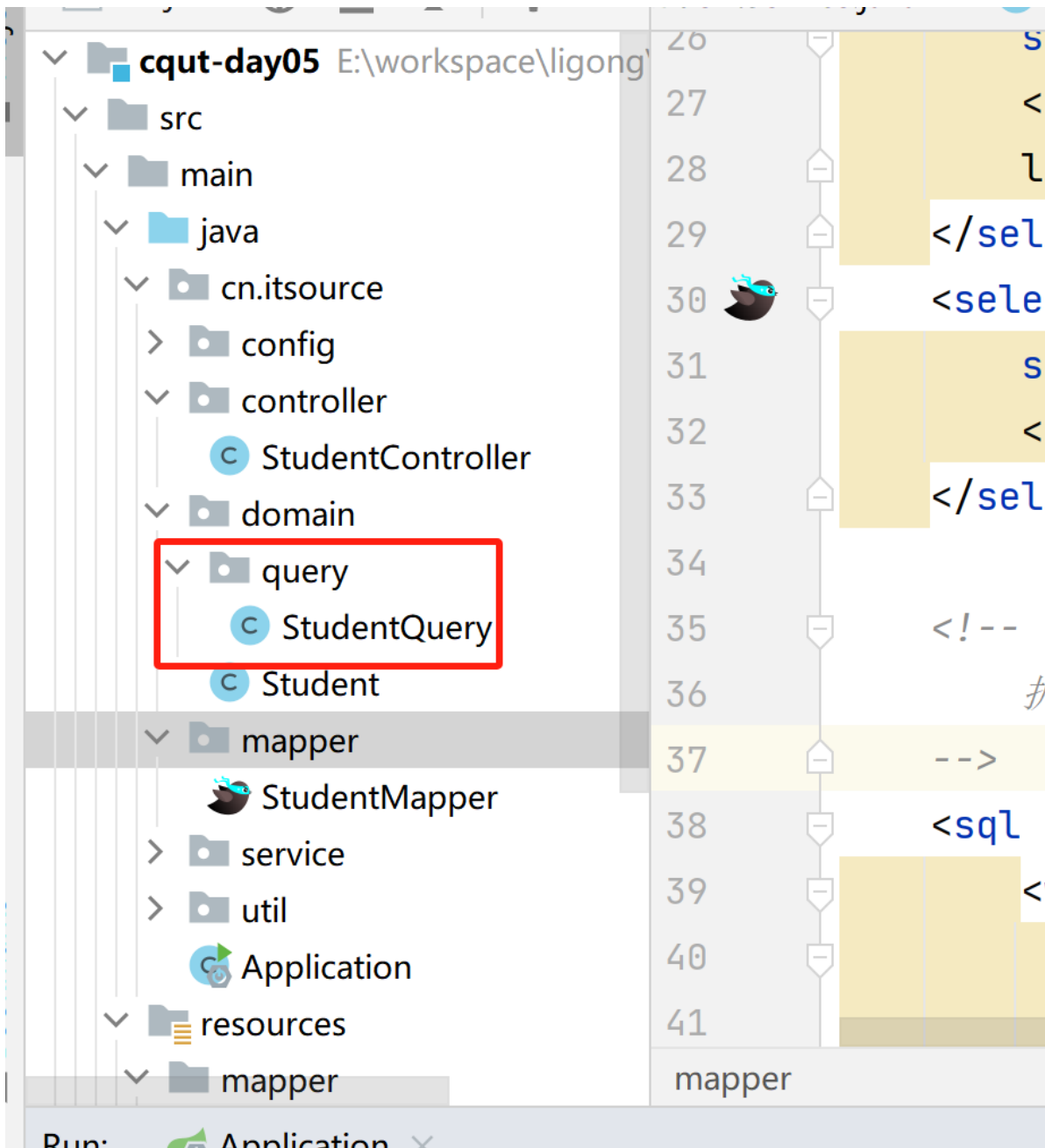


扩展：

axios的post传参：body传参形式

## 八、分页+条件查询

### 1、接收请求参数：封装请求参数的实体类



```
package cn.itsource.domain.query;

import cn.itsource.util.PageParams;
import lombok.Data;

import java.util.Date;
@Data
public class StudentQuery extends PageParams {

    /**
     * 性别
     */
    private String gender;
    /**
```

```

    * 出生日期 - 开始
    */
private Date birthdayBegin;
/**
    * 出生日期 - 结束
    */
private Date birthdayEnd;
/**
    * 学历
    */
private String degree;
/**
    * 简介
    */
private String desc;

}

```

## 2、使用动态sql进行条件查询

StudentMapper.java

```

/**
    * 分页条件查询
    * @param query
    * @return
    */
List<Student> queryPage(StudentQuery query);

/**
    * 条件查询总数
    * @param query
    * @return
    */
Long queryCount(StudentQuery query);

```

StudentMapper.xml

```

<!--
    mybatis的动态sql - mybatis中很牛逼的语法
-->

```



```

<select id="queryPage"
resultType="cn.itsource.domain.Student">
    select * from student
    <include refid="querySql"></include>
    limit #{startIndex},#{pageSize}
</select>
<select id="queryCount" resultType="java.lang.Long">
    select count(*) from student
    <include refid="querySql"></include>
</select>

<!--
    扩延：抽取sql片段
-->
<sql id="querySql">
    <where>
        <if test="gender != null and gender != ''">
            and gender = #{gender}
        </if>
        <if test="birthdayBegin != null">
            and birthday >= #{birthdayBegin}
        </if>
        <if test="birthdayEnd != null">
            and birthday <= #{birthdayEnd}
        </if>
        <if test="degree != null and degree != ''">
            and degree = #{degree}
        </if>
        <if test="desc != null and desc != ''">
            and `desc` like concat('%',{desc},'%')
        </if>
    </where>
</sql>

```

### 3、测试

---

POST

http://localhost:8080/student/pageQuery

发送

保存

Params 2

Body 1

Headers

Cookies

前置操作

后置操作

Auth

设置

</>

none

form-data

x-www-form-urlencoded

json

xml

raw

binary

GraphQL

msgpack

格式化

1 {

2   "pageNum":1,

3   "pageSize":2,

4   "gender":1

5 }

Body

Cookie

Header 8

控制台

实际请求

Pretty

Raw

Preview

Visualize

JSON

utf8

🔍

1 {

2   "code": 200,

3   "message": "操作成功!",

4   "data": {

5     "rows": [

6       {

7         "id": "1",

8         "name": "张三",

9         "gender": "1",

10        "birthday": "2024-06-13",

11        "phonenum": "18888888888",

12        "degree": "5",

13        "desc": "是个狼人",

14        "state": "0"

15       },

16       {

17         "id": "2",

18         "name": "里斯",

19         "gender": "1",

20         "birthday": "2024-06-14",

21         "phonenum": "16666666666",

22         "degree": "2",

23         "desc": "这个更狠",

24         "state": "0"

25       }

26     ],

27   }

200

35 ms

338 B

Cookie 管理