目录

[构建模块 2](#_Toc141261229)

[创建project 2](#_Toc141261230)

[加入mybatis-plus 5](#_Toc141261231)

[创建数据库 5](#_Toc141261232)

[修改配置文件 6](#_Toc141261233)

[编写测试代码来测试mybatis-plus 8](#_Toc141261234)

[创建文件结构 8](#_Toc141261235)

[在实体类中写入user类 8](#_Toc141261236)

[在service中写UserService接口 9](#_Toc141261237)

[创建实现类 10](#_Toc141261238)

[在Controller中写测试代码 10](#_Toc141261239)

[如果想写一个自定义的查询呢？ 11](#_Toc141261240)

[编写增删改查方法 13](#_Toc141261241)

[进行测试 14](#_Toc141261242)

[使用分页插件 18](#_Toc141261243)

[添加分页拦截器 20](#_Toc141261244)

[添加自定义查找请求 21](#_Toc141261245)

[封装返回数据的封装 22](#_Toc141261246)

[新增测试请求 23](#_Toc141261247)

[创建Vue项目 24](#_Toc141261248)

[导入到idea 24](#_Toc141261249)

[配置启动程序 24](#_Toc141261250)

[常见报错 26](#_Toc141261251)

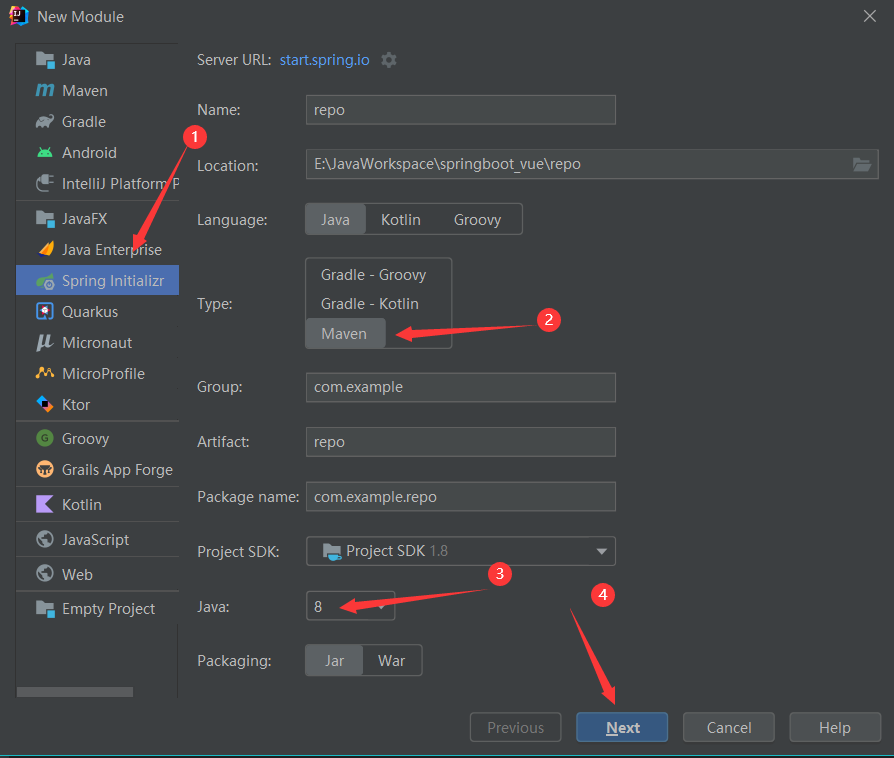
[参考 27](#_Toc141261252)

# 构建模块

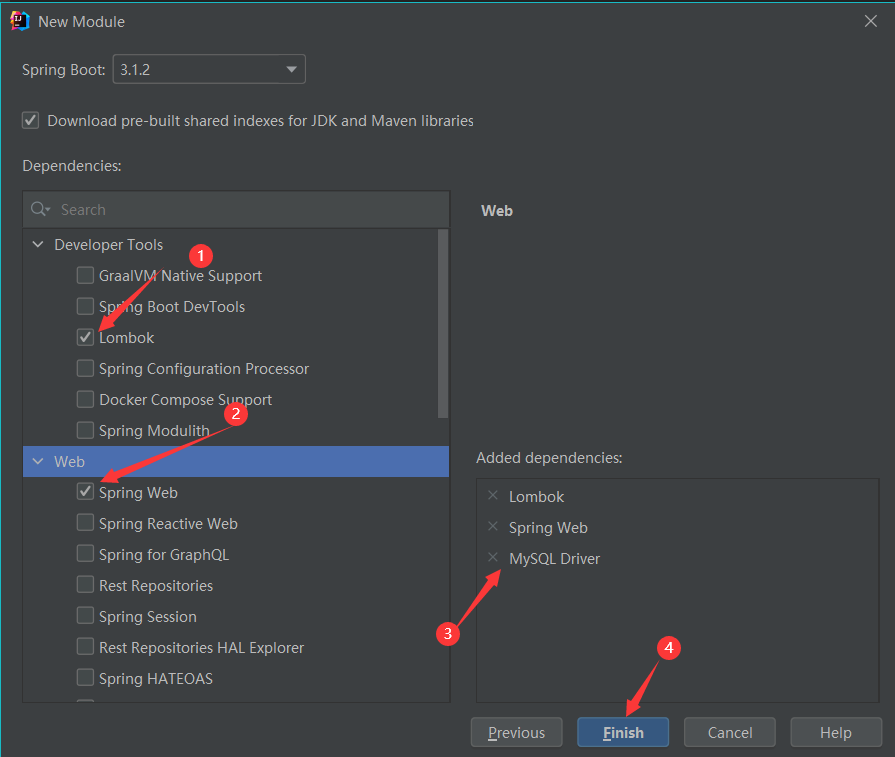
## 创建project

在project目录中创建module，此为后端项目

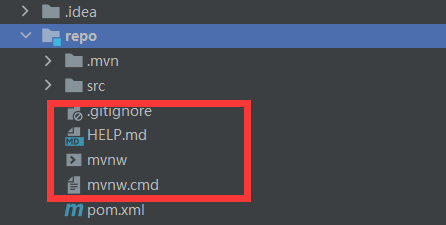
1. 第一步



1. 第二步

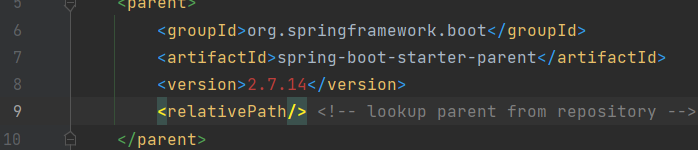


1. 第三步：勾住的没用，删掉



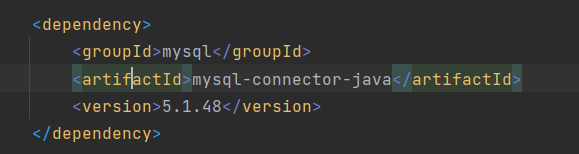
1. 修改pom文件

修改springboot版本

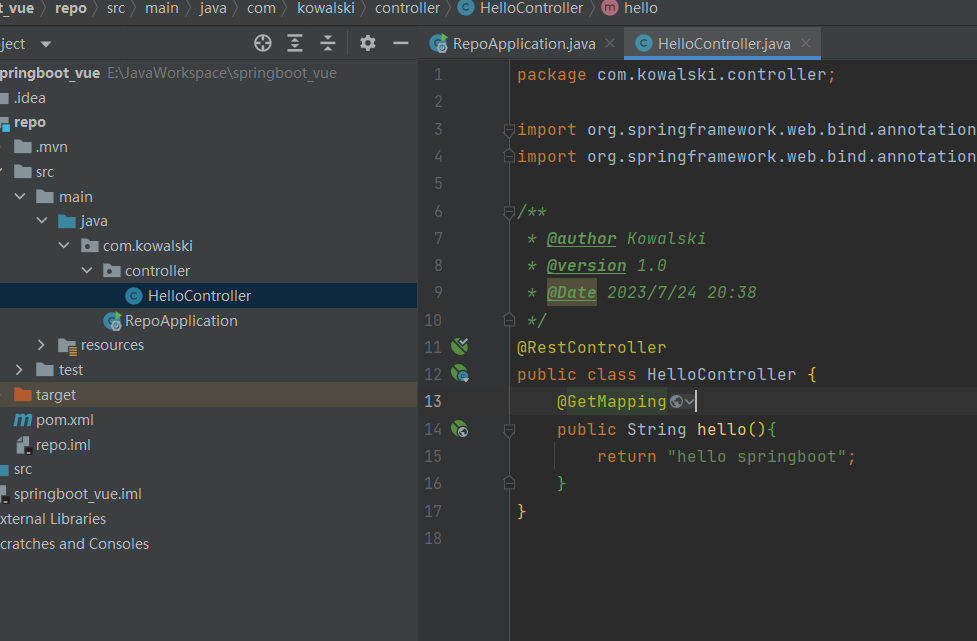


版本过高会提示类文件具有错误的版本 61.0, 应为 52.0 请删除该文件或确保该文件位于正确的类路径子目录中。

修改mysql依赖。

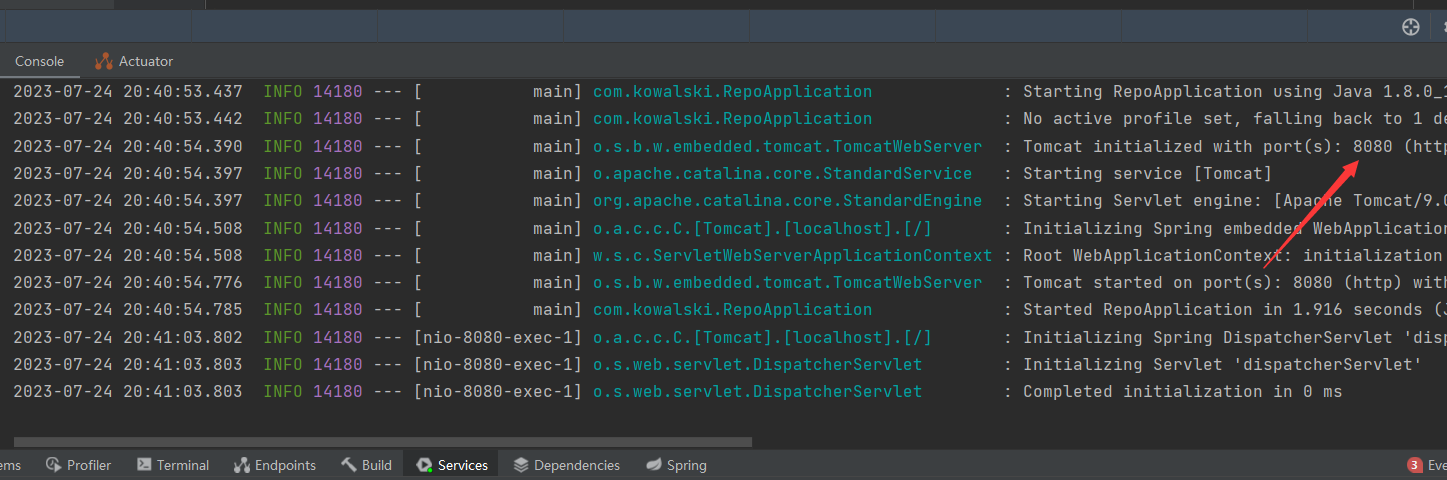


1. 创建测试类

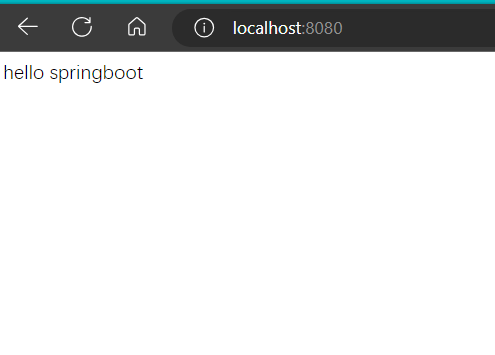


在包中创建controller层的包，写一个测试类，运行springboot项目。

1. 端口号会打印到控制台，访问localhost:8080来测试项目



1. 测试成功



## 加入mybatis-plus

<dependency>

<groupId>com.baomidou</groupId>

<artifactId>mybatis-plus-boot-starter</artifactId>

<version>最新版本</version>

</dependency>

在pom文件中引入这个依赖即可。

## 创建数据库

Create database manage\_system;

User manage\_system;

CREATE TABLE user(

id int auto\_increment comment '主键' PRIMARY KEY,

no VARCHAR(20) NULL COMMENT '账号',

`name` VARCHAR(100) not null COMMENT '姓名',

`password` VARCHAR(100) not NULL,

age int null,

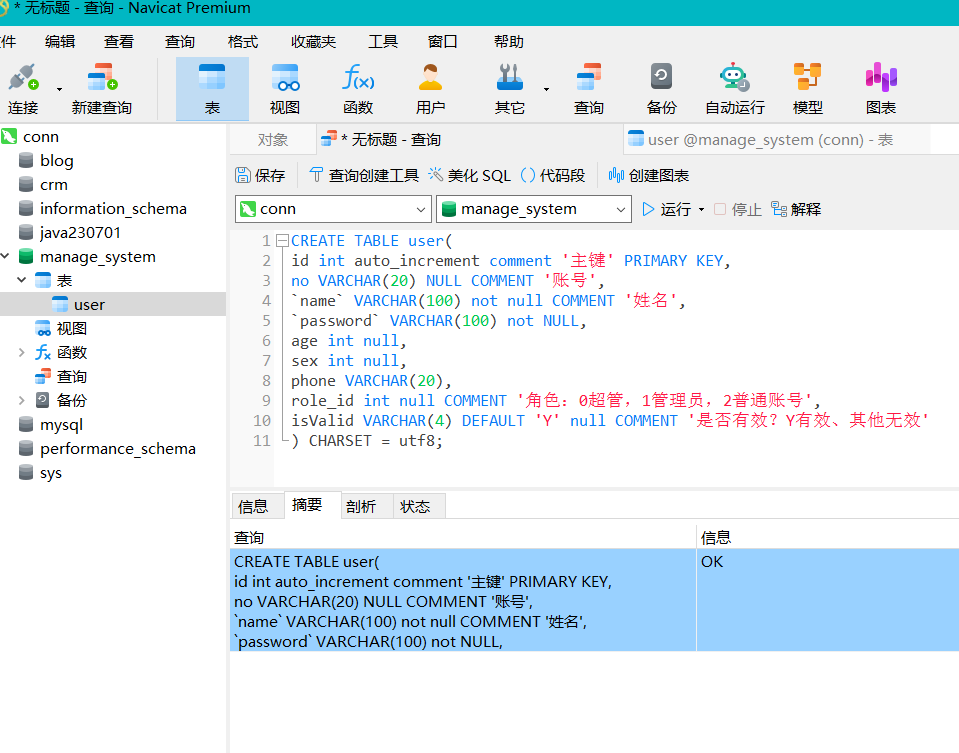
sex int null,

phone VARCHAR(20),

role\_id int null COMMENT '角色：0超管，1管理员，2普通账号',

isValid VARCHAR(4) DEFAULT 'Y' null COMMENT '是否有效？Y有效、其他无效'

) CHARSET = utf8;



## 修改配置文件

在 application.yml 配置文件中添加数据库的相关配置：

把application.properties文件修改为application.yml，打开文件输入以下内容。设置端口号必须与后端项目不一样，否则在启动前端项目时会发生端口占用冲突。

数据源为刚刚创建的数据库名。

用户名和密码为自己的设定。

server:

port: 8081

spring:

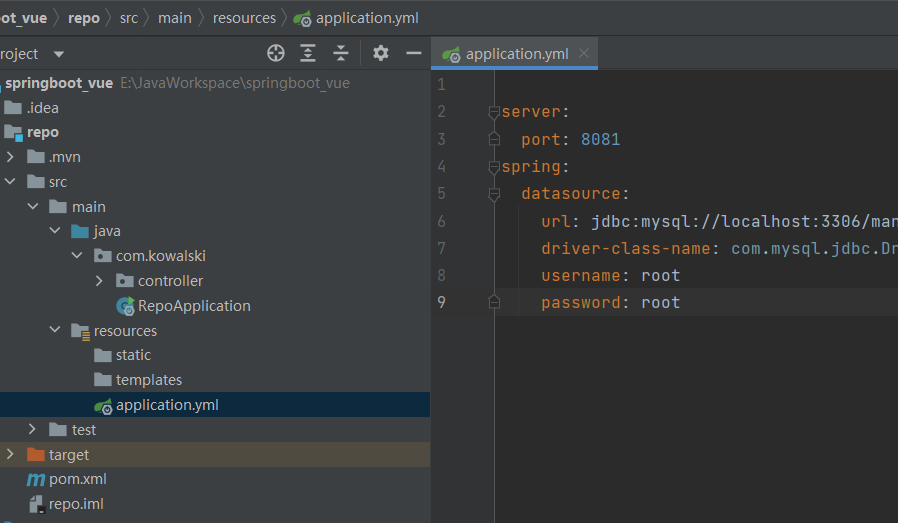
datasource:

url: jdbc:mysql://localhost:3306/manager\_system?useSSL=false&useUnicode=true&characterEncoding=utf-8&serverTimezone=GMT%2B8

driver-class-name: com.mysql.jdbc.Driver

username: root

password: root

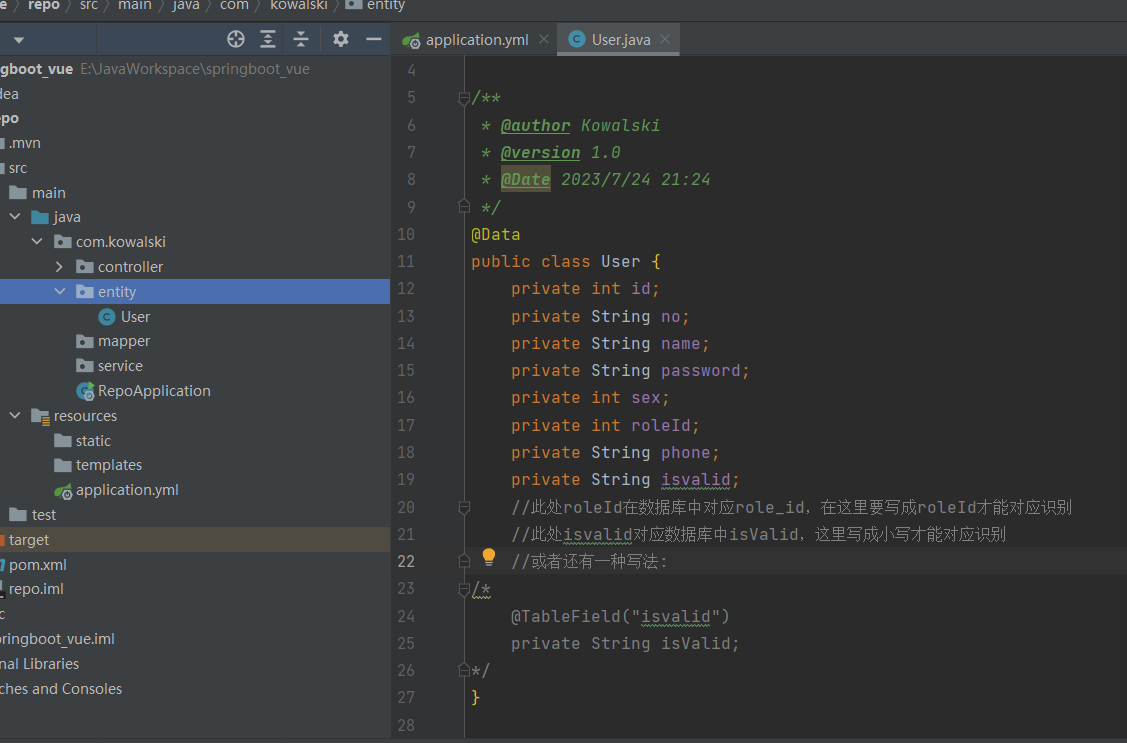


# 编写测试代码来测试mybatis-plus

## 创建文件结构

包括entity实体类、mapper、service包

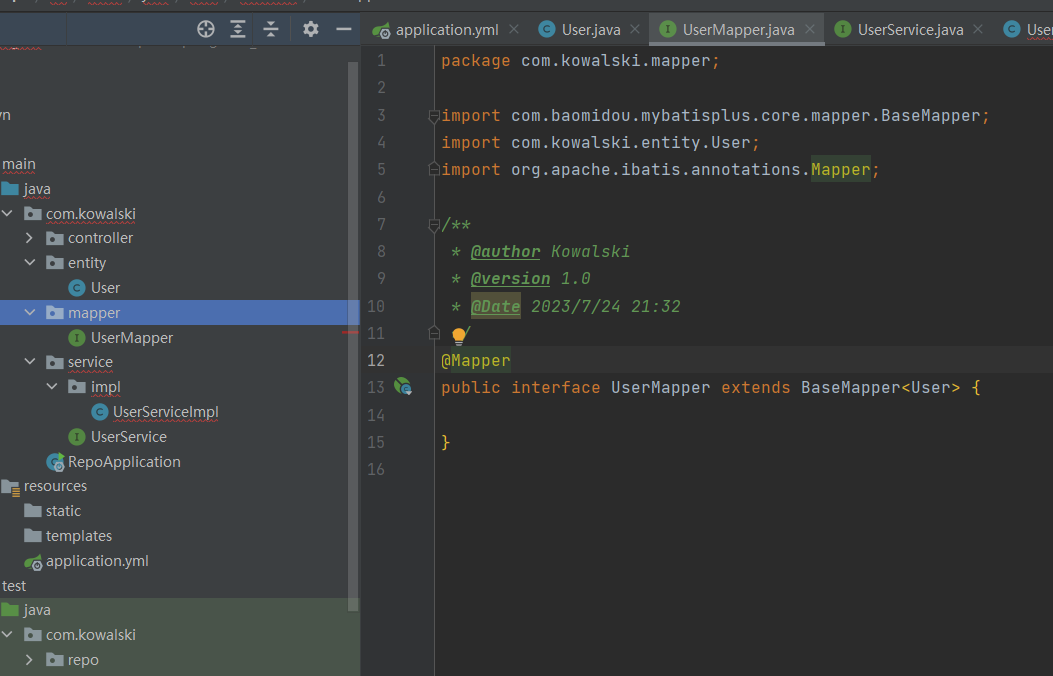
## 在实体类中写入user类



在mapper中创建UserMapper接口

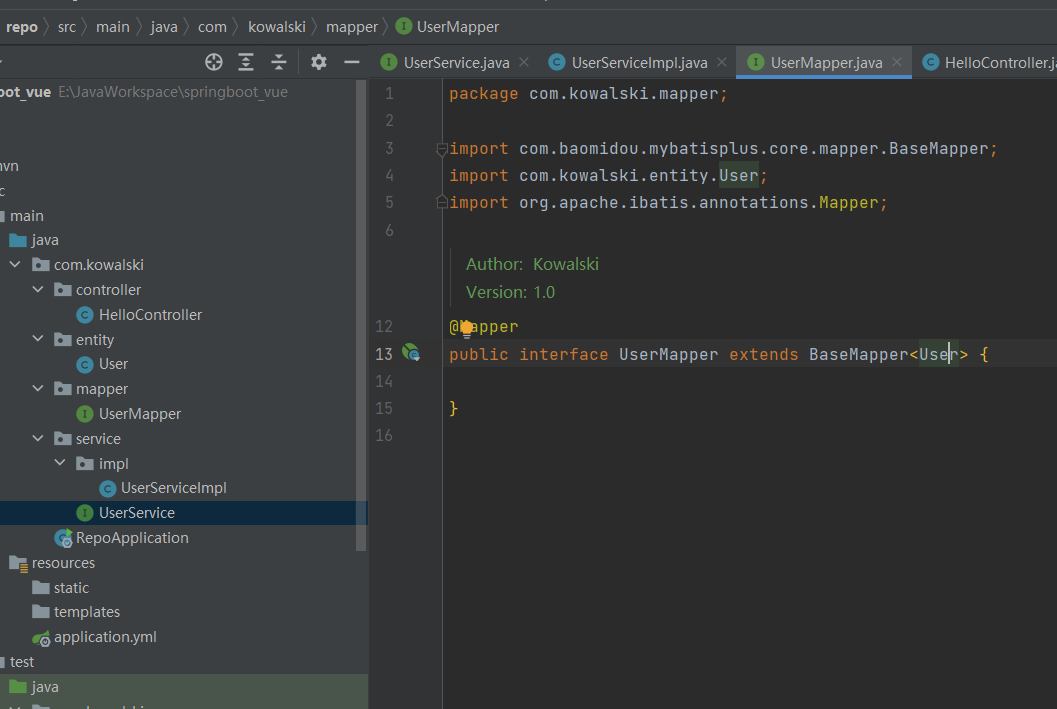
让接口继承BaseMapper<User>

在类前标注@Mapper注解，将接口交给spring管理



## 在service中写UserService接口

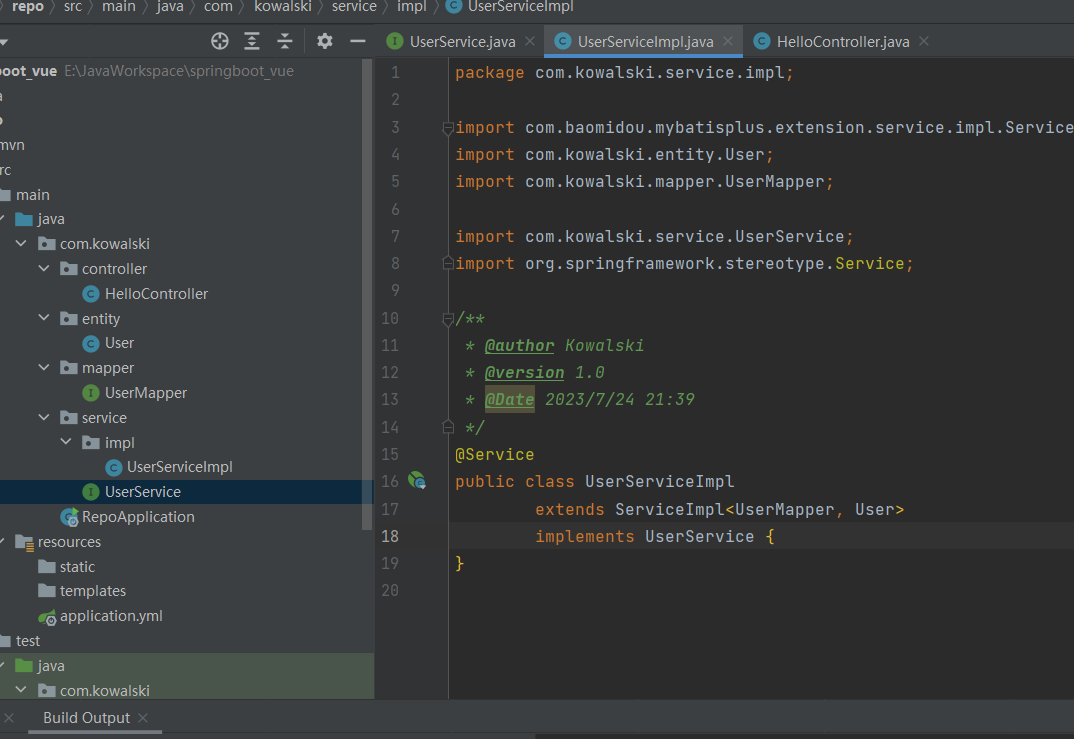
接口需要继承IService<User>



## 创建实现类

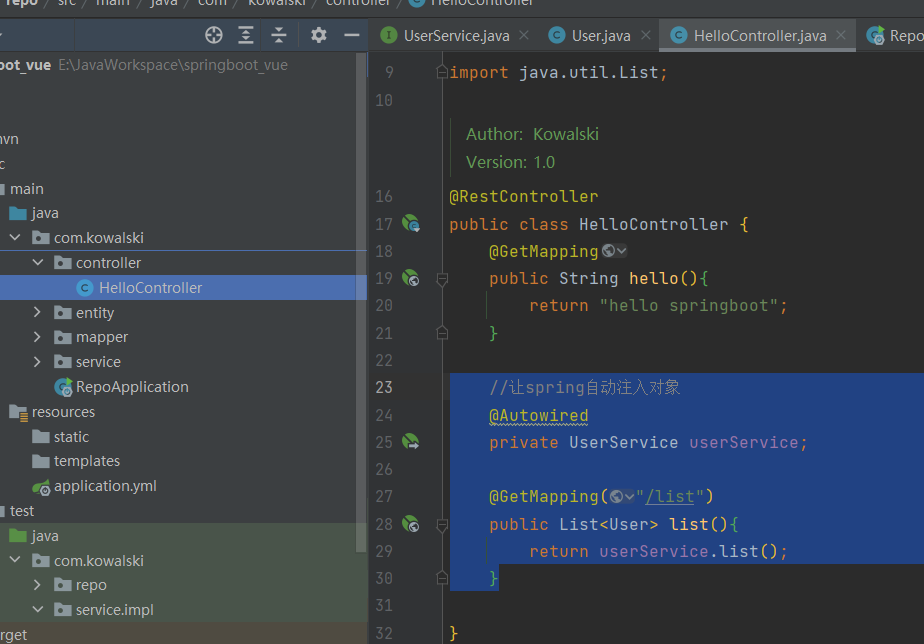
在service包中创建impl实现类包，创建UserServiceImpl类。

这个类需要继承ServiceImpl<UserMapper, User>，泛型第一个是接口，第二个是实体类，还要实现UserService接口。

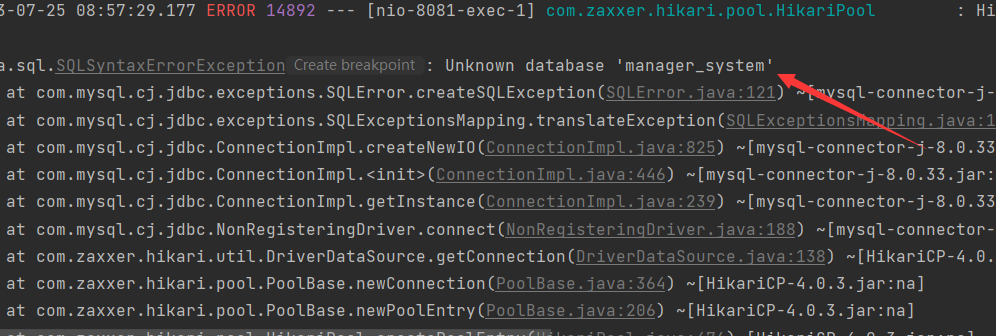


## 在Controller中写测试代码

加入以下代码，启动服务器访问localhost:8081/list以查看结果。



此时报错了。页面回报错误代码500，查看控制台，发现数据库名写错了。

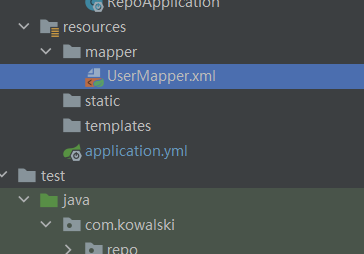


在yml配置文件中改正后即可查询成功。

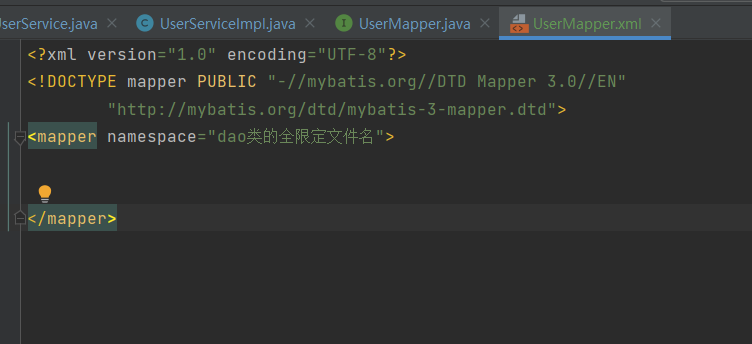
## 如果想写一个自定义的查询呢？

例如想写一个listAll方法：

1. 先在controller中写好方法
2. 去userService接口中写这个方法
3. 让实现类去实现方法
4. 让类实现方法，方法体要自己写
   1. 使用@Resource注解注入UserMapper（同一个controller文件中只需要注入一次）
   2. 返回值填userMapper.listAll()//此时方法名是自己写的。
5. 方法名爆红，直接去UserMapper中补上这个方法
6. 在resources文件夹中创建mapper文件，在文件夹中创建UserMapper.xml配置文件



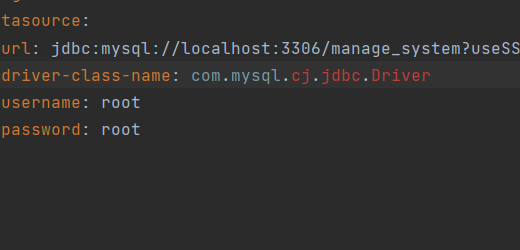
1. 配置内部文件
   1. 修改映射头，到网上搜mybatis mapper映射头配置



* 1. namespace=com.kowalski.mapper.UserMapper
  2. 补充标签：<select id=”listAll” resultType=”com.kowalski.entity.User”></select>
  3. 标签内部填写sql语句
  4. 随后可以运行程序，测试自定义查询是否成功

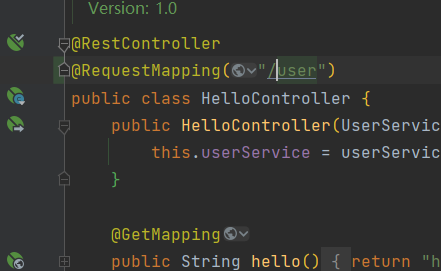
此时报错“Error creating bean with name 'helloController' defined in file”

经查资料与检查发现yml文件中曾经修改过mysql的驱动包，导致spring无法识别，修改之后就成功了。

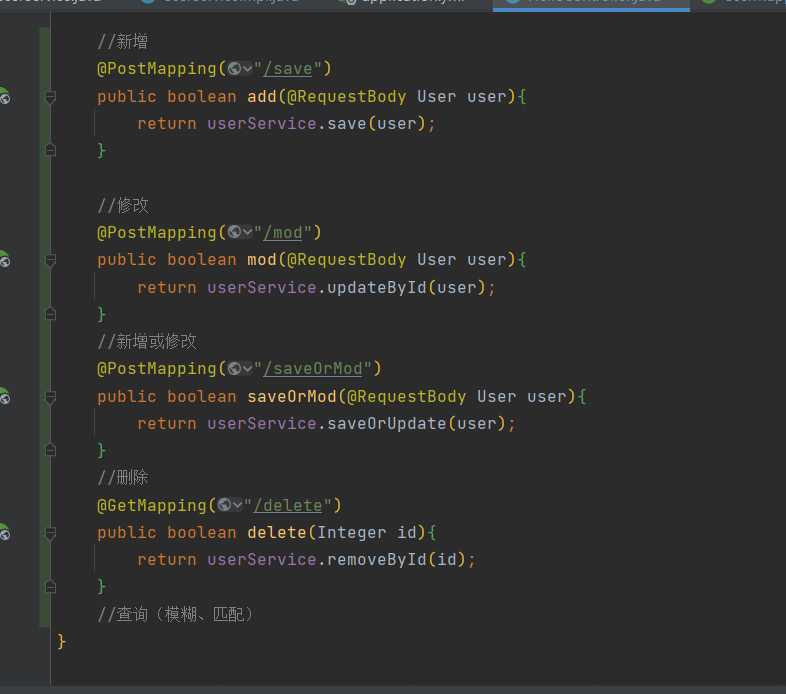


# 编写增删改查方法

在controller前加上@RequestMapping(“/user”)注解



添加各种请求



## 进行测试

下载Postman软件，用谷歌账号登录。

建立连接，输入url，切换数据模式为JSON，输入JSON格式的测试数据，更改请求为POST。

本次测试使用的数据：数据字段视情况增删，格式为JSON

{

    "id":"23",

    "no": "admi2n",

    "name": "小明",

    "password": "123",

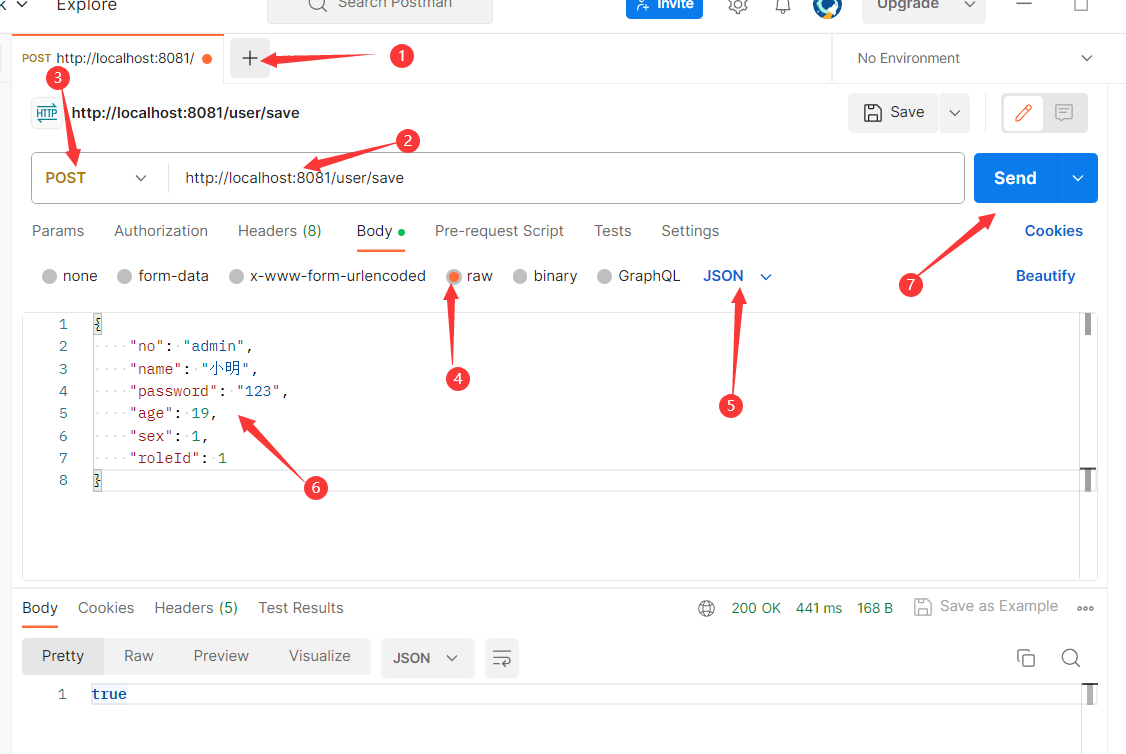
    "age": 19,

    "sex": 1,

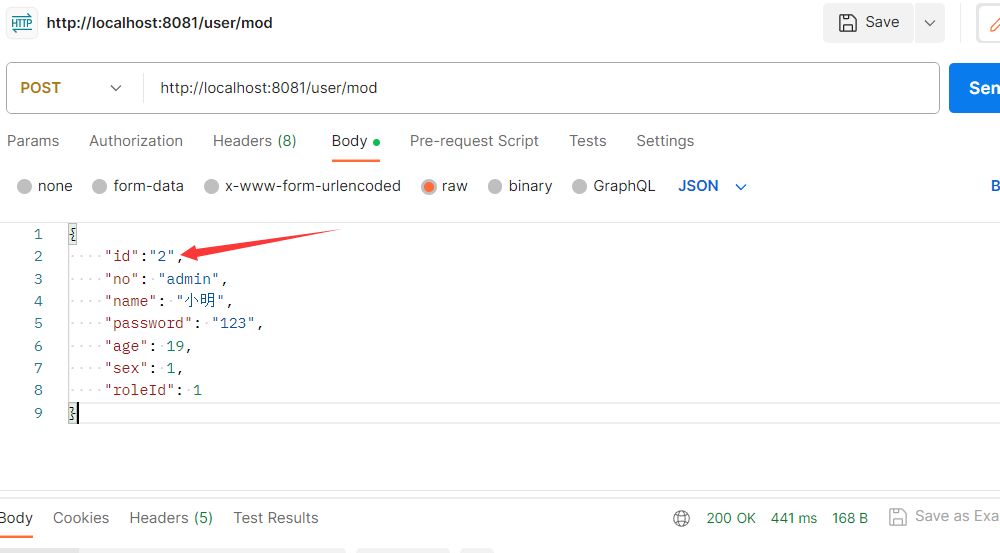
    "roleId": 1

}

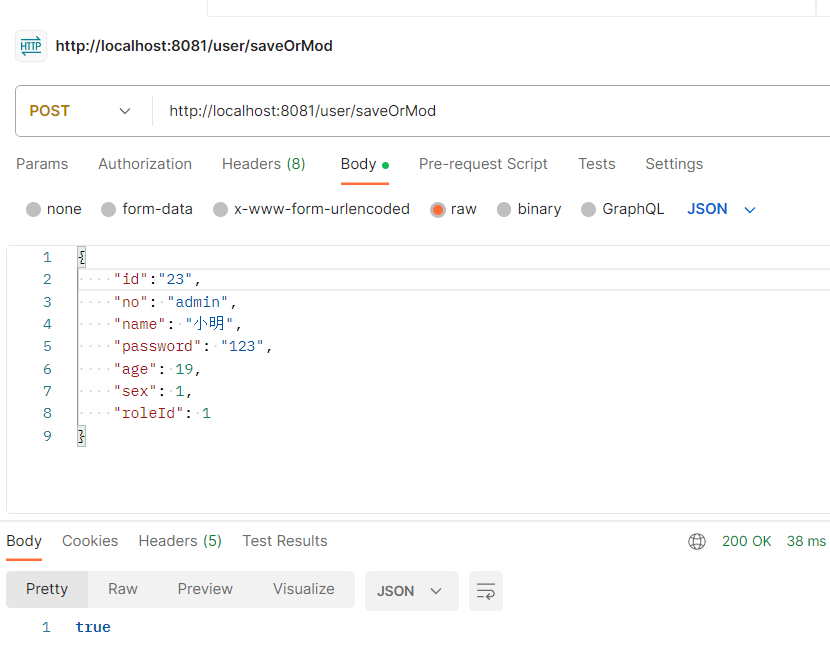
进行对/user/save的测试：



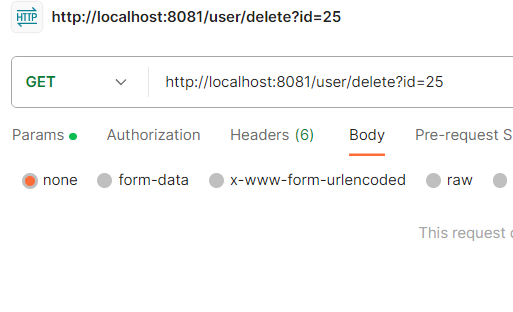
进行对/user/mod的测试（mod方法需要获取id，所以传入的数据需要增加id字段）：



进行对/user/saveOrMod的测试（这个方法会去判断是否有此数据，有的话就修改，没有的话就增加，id不是必须的字段，但是没有id的话这个方法一定是去新增数据）：

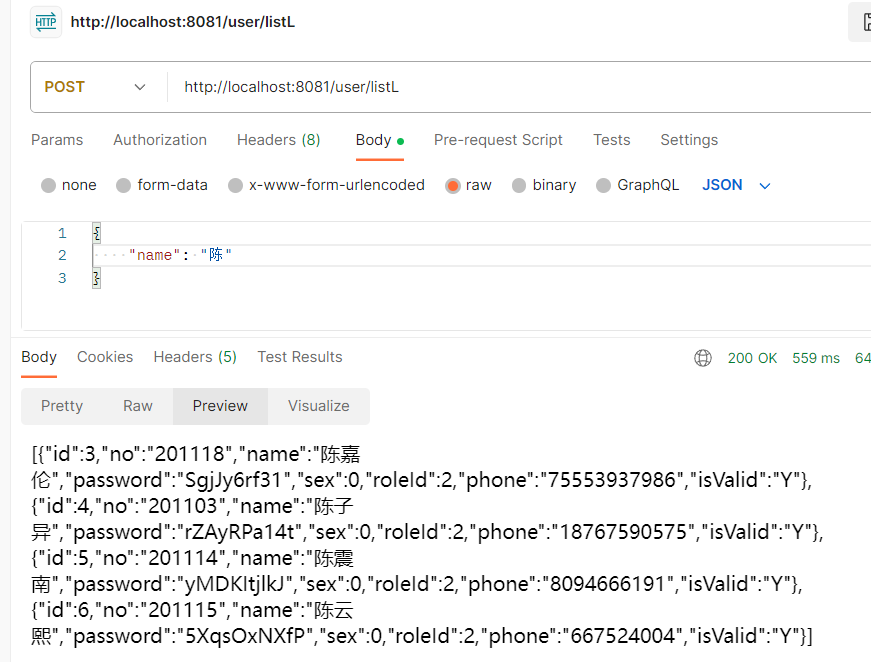
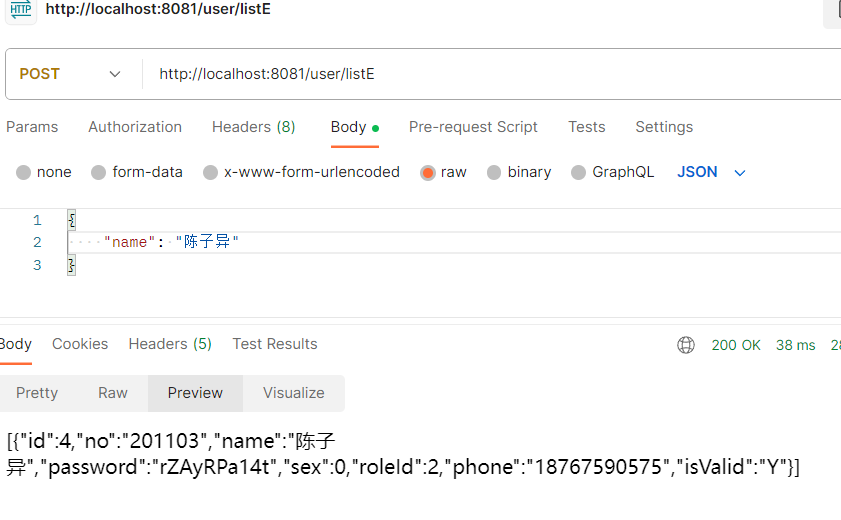


对/user/delete进行测试（设置请求为get，直接在url后拼接参数，关闭参数列表）：



进行对/user/ListE和/user/ListL的测试：

这两个方法分别对应完全匹配Equals和模糊匹配Like



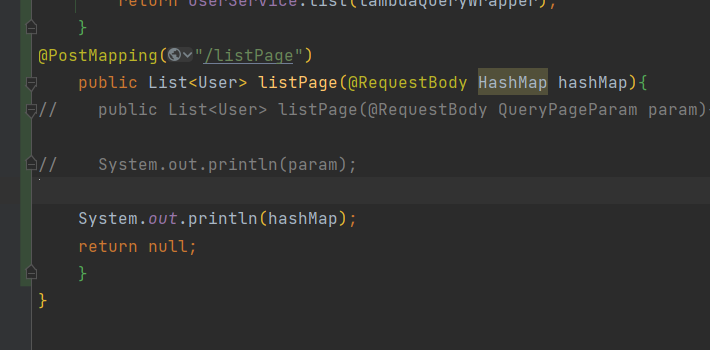
报错：Method Not Allowed(err405) --请求方法有错，把默认的get改为post去测试save方法。

报错：Unsupported Media Type(415) --传输数据有误，把默认的html格式改为JSON格式去测试。

报错：argument type mismatch –参数类型不匹配，在实体类中为主键id增加注解：@TableId(type = IdType.AUTO)。

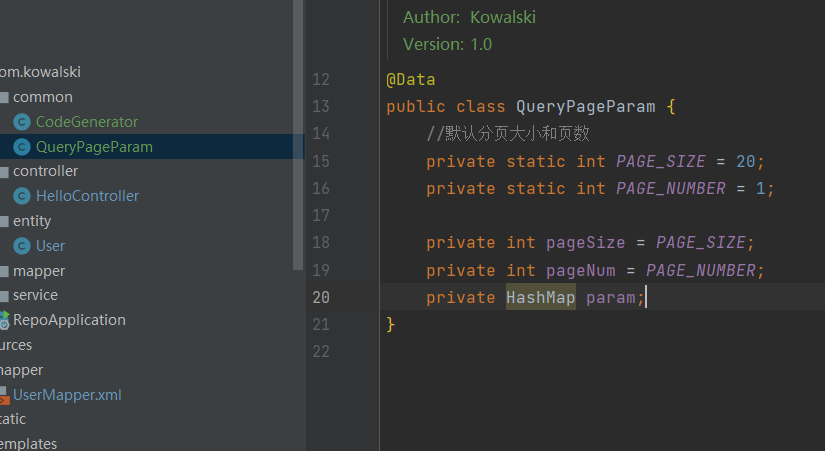
# 使用分页插件

一种方式：直接用HashMap接受参数并返回：

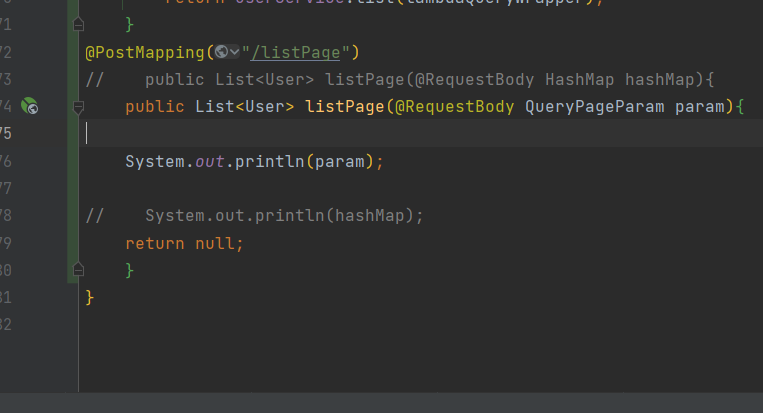


另一种方式：封装自定义分页类。（封装到com/kowalski/common中）

其中，param是除了分页数据之外的所有数据，例如id，name之类的。



写完这里之后，到listPage中修改方法：



随后进行测试。

下面给出测试数据（JSON格式）：

{

    "pageSize":5,

    "pageNum":1,

    "param":{

        "no":"admin",

        "name":"张"

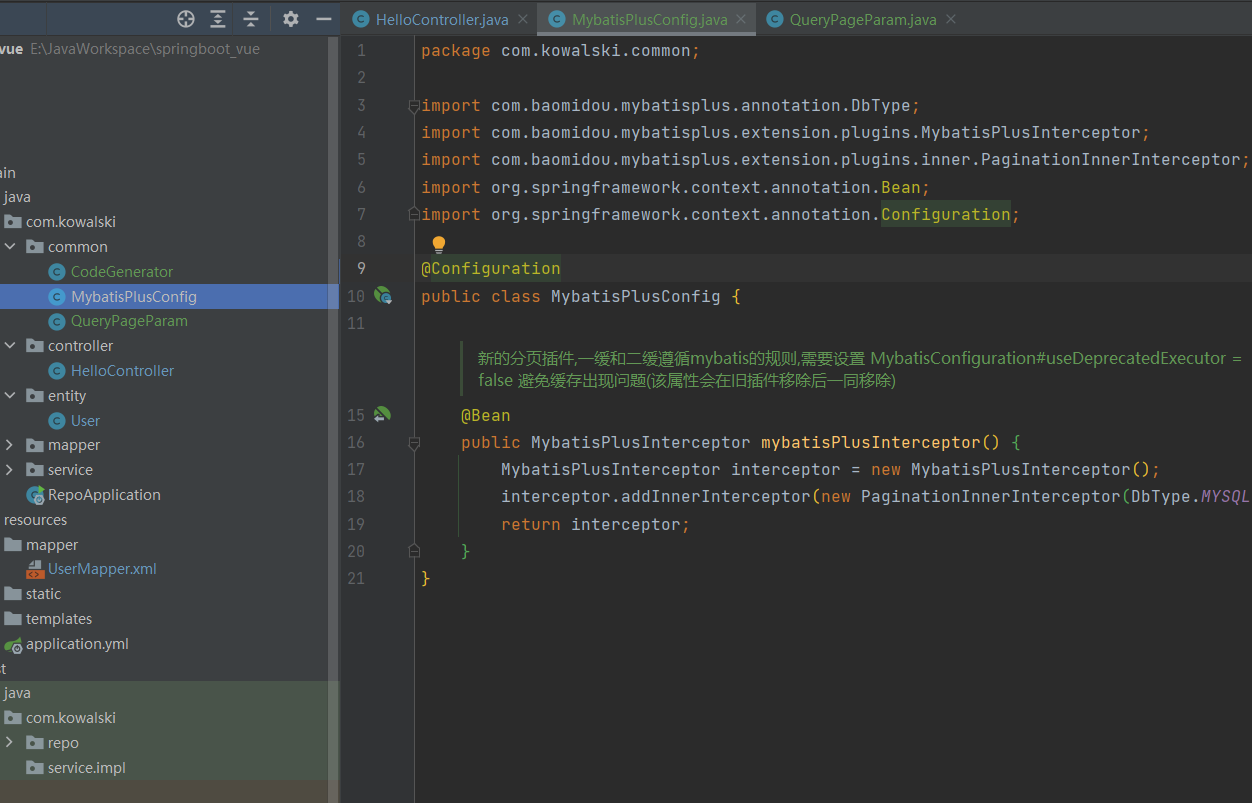
    }

}

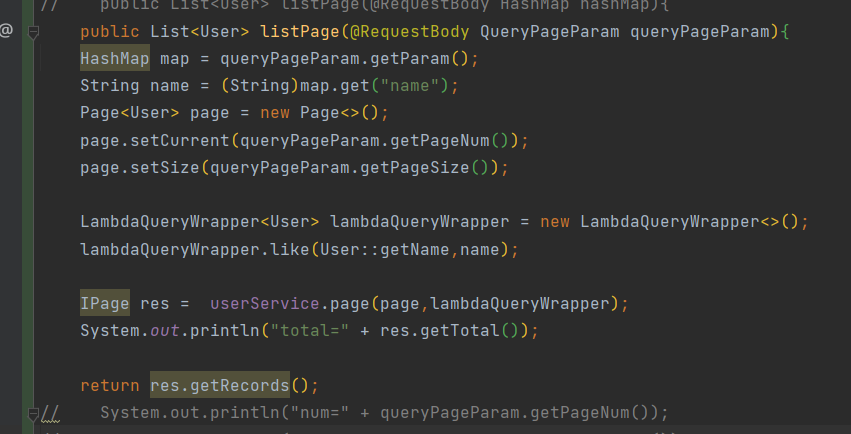
## 添加分页拦截器

在mybatis-plus官网插件主体中找到springboot的配置，添加到common包中。

[插件主体 | MyBatis-Plus (baomidou.com)](https://baomidou.com/pages/2976a3/#spring-boot)



修改listPage方法逻辑



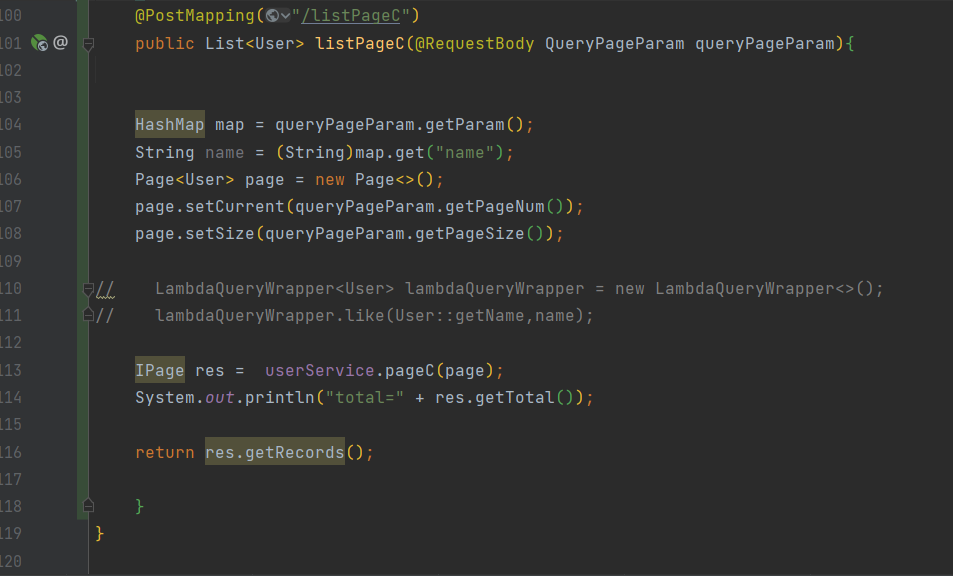
如此即可使用mybatisplus提供的分页拦截器。

## 添加自定义查找请求

1. 复制一份listPage，修改部分代码。
2. 完成一系列新建
   1. 创建Ipage对象，使用userService中的pageC方法，此时接口中无此方法
   2. 去接口中创建方法IPage pageC(IPage<User> page);
   3. 在实现类中实现该方法，返回值为UserMapper.pageC。如果前面没有注入UserMapper，还要进行[注入操作](#_如果想写一个自定义的查询呢？" \o "4.1红字)。

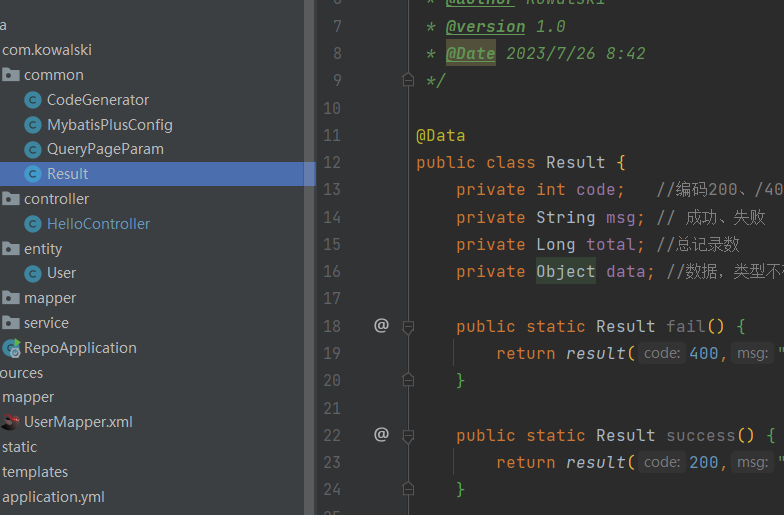
@Override  
public IPage pageC(IPage<User> page) {  
 return userMapper.pageC(page);  
}

* 1. 在对应的xml配置文件中写自定义sql



# 封装返回数据的封装

在common包中创建Result包装类来封装查询结果，生成多种默认返回类型，包括成功（带参无参）、失败。



## 新增测试请求

复制一份ListPageC，在返回值上稍作修改来测试包装返回结果。

返回值改为return Result.success(res.getRecords(), res.getTotal());

即可。[[1]](#footnote-1)

下面给出测试数据：

{

    "pageSize":5,

    "pageNum":1,

    "param":{

        "name":""

    }

}

Name可以自行调整，如”陈”/”陈子异”。

# 创建Vue项目

在原project路径中创建vue模块的文件夹，在文件夹中使用vue create Kowalski-web来创建项目。

在cmd中提示的vue2和vue3中选择vue2，进行安装。

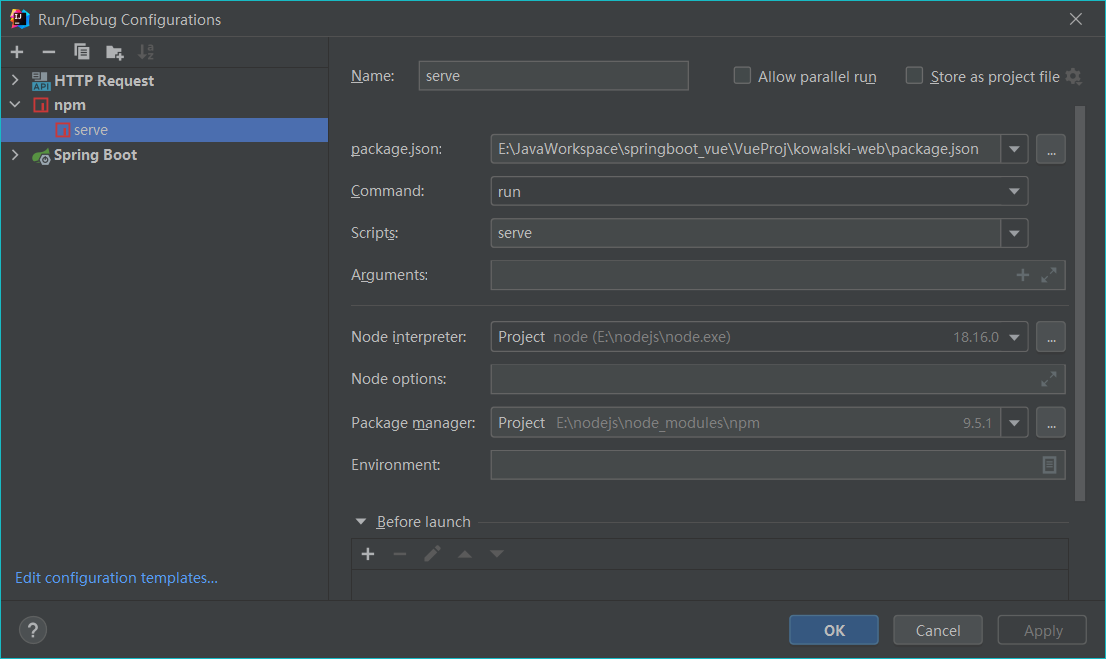
安装成功后进入Kowalski-web目录，执行npm run serve来启动项目，根据cmd给出的url访问服务器，即可测试是否成功。

## 导入到idea

在项目中打开project structure配置，在module中选择import，把刚刚创建好的vue项目导入进来。

## 配置启动程序

在run按钮那里选择edit configuration，新增npm启动方式，在启动配置中选择刚刚创建好的项目中的package.json文件即可自动识别，并勾选serve。



如果没有安装node.js，还要去提前安装。

## 引入element-ui

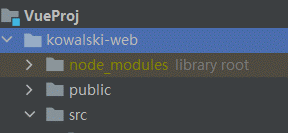
在项目路径下打开终端，输入npm i element-ui -S来安装插件。

不要使用最新版本的elementui。会出现诸多问题。输入

npm install element-ui@2.8.0来安装低版本。

安装成功后，会多一个文件夹node\_modules。

修改vue.config.js中的配置信息： 添加lintOnSave: false



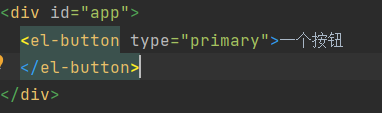
在main.js中写入以下内容

import ElementUI from 'element-ui';

import 'element-ui/lib/theme-chalk/index.css';

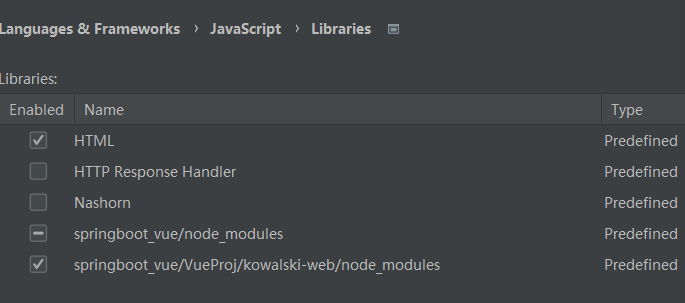
Vue.use(ElementUI);

然后去App.vue中添加标签以测试是否安装成功。



问题：element组件标黄提示无法识别的tag。

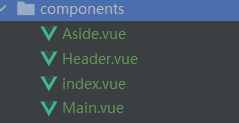
解决办法：在settings中把element状态更改为勾选。



[IDEA编辑器下Vue项目中Element标签出现标黄（Unknown html tag el-form）问题解决方案! - EEEEET - 博客园 (cnblogs.com)](https://www.cnblogs.com/imyjy/p/16922739.html)

## 拆分页面

把index页面中的Aside（左侧菜单栏），Header（顶部条），Main（主区域显示数据）分开放到各自的vue组件中，原文只对其进行引用。这样的写法可以使index的结构更清晰，方便修改大组件的样式。

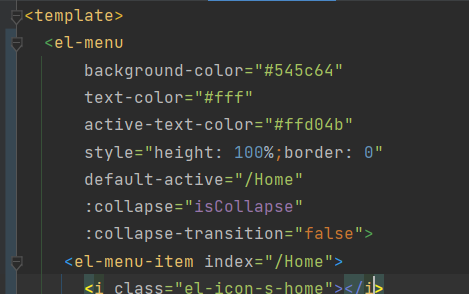


## 添加自定义的导航栏

在导航子组件中修改导航栏内容和样式，不过这里的内容现在是随便写，因为后期要根据数据库动态显示导航栏。



## 修改导航栏收缩动效

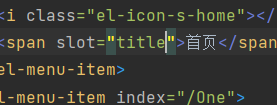


添加:collapse-transition=”false”即可关闭动画，这个属性默认是TRUE。

首先在index父组件的Aside标签中定义isCollapse属性，在props中传入该属性，传到Aside子组件中接受，接收到之后同样应用到属性中。

## 为导航栏提供提示气泡（插槽功能slot）（未生效）

给每个span标签增加这个属性：slot=”title”



## 使收起导航栏时，按钮能够变化

在index中为header设置属性



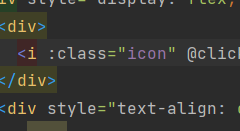
再到下面的data中定义它。



在下面使用props接受父组件传入的属性，



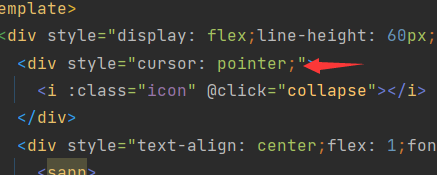
去Aside子组件同步更改这个属性，让固定不变的i标签使用传入的变量。



回到父组件中，为属性提供变更逻辑。展开时的图标为fold，折叠时的图标为unfold。

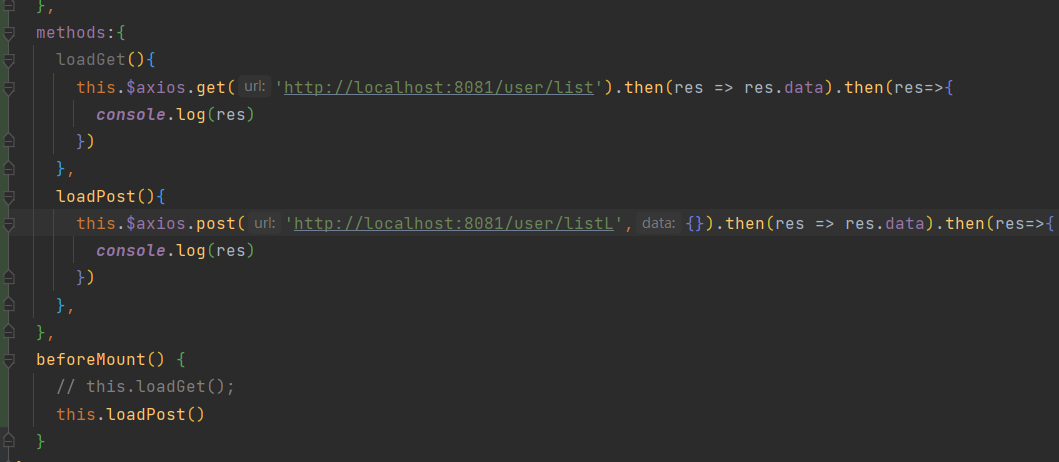


在按钮的i标签的父标签上使用cursor:pointer来让指针移到按钮上面时可以变化。（其实放在i标签里也可以，而且逻辑性更好）



## 让后端的数据传到前端

在main页面中添加loadGet或者loadPost请求方法，让页面渲染时调用请求。



现在直接调用的话，传到前台的数据是空的。此时需要在后台修改ListL请求，让请求中的user不为空时再去执行操作。



## 把前半截地址设置为全局地址

Localhost:8080全局都要使用，所以把它设置为全局地址可以避免多次输入。

在main.js中设置Vue.prototype.$HttpUrl=’http://localhost:8081’即可。

那么现在就可以把之前设置过的路径都改掉，改成$HttpUrl + ‘路径’，而且以后要改路径，就只需要在main.js中修改。

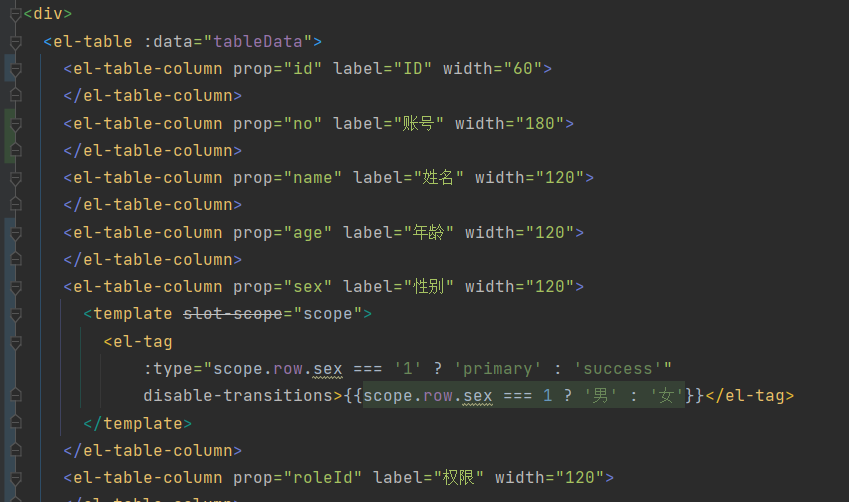
*另一种方法是设置拦截器，这里没有教。*

## 把从后端获取到的数据展示到页面上

先设置数据源



再把数据传给表单，其中，性别使用了el-tag标签，可以在页面上以tag形式显示。



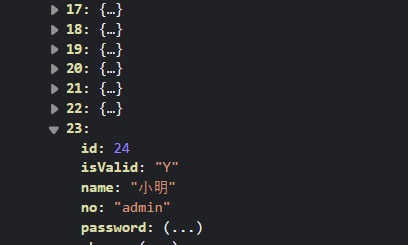
## 添加编辑删除按钮

修改操作栏的尺寸，让两个按钮能放得下并且好看。



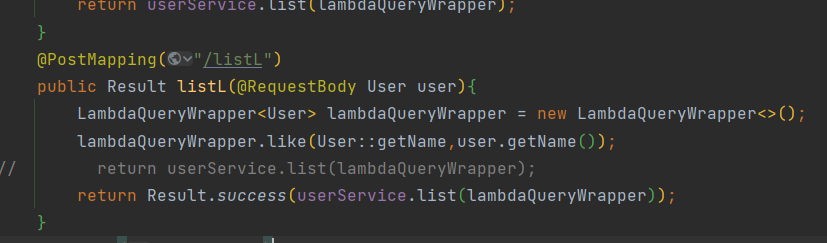
## 优化返回到前台的数据

原来返回到前台的数据是这样：



现在把它封装到Result中，优化返回到前台的数据。

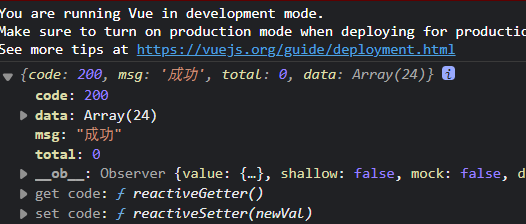
修改调用的请求的返回值为Result



此时是无法显示的，要去前端获取数据的地方进行一次判断，然后再修改传给tableData的数据。

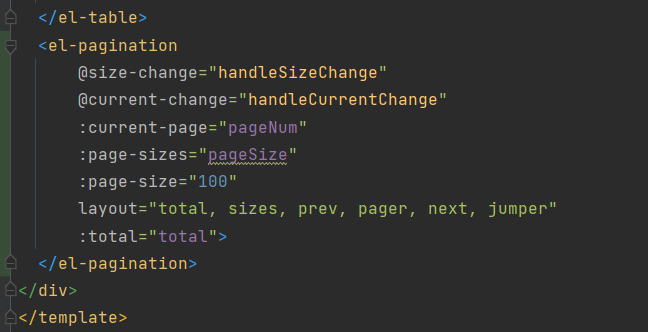


此时页面回复正常显示，并且封装了后台返回的Result信息。

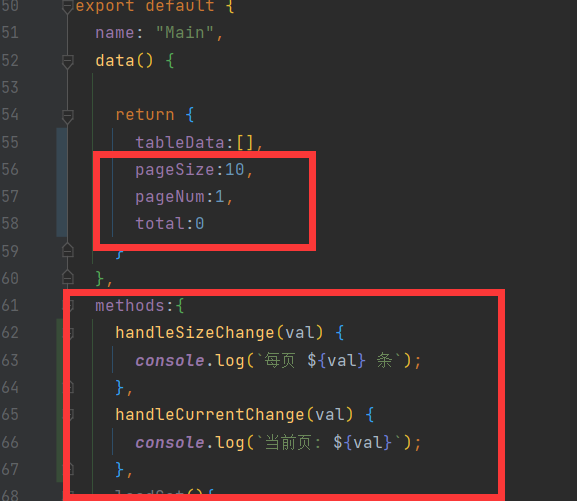


# 实现分页功能

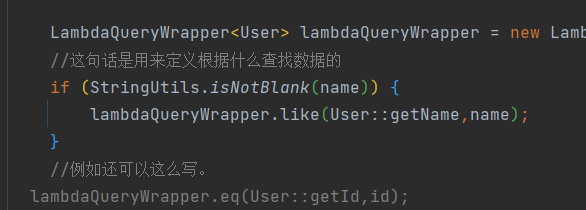
从官网引入分页组件



设置对应方法和参数，这里先设置一些默认值。



接下来把loadPost的请求url改为listPageC1，这个方法能够获取带分页信息的参数。但是这个方法没有判空，所以要进行改进。

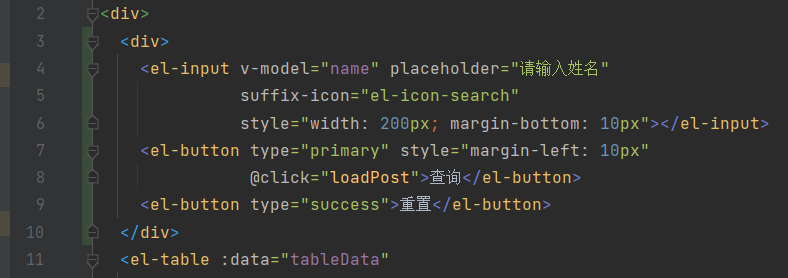


然后修改从官网抄下来的方法，让他能够实现切换页面的功能和切换页面大小的功能。

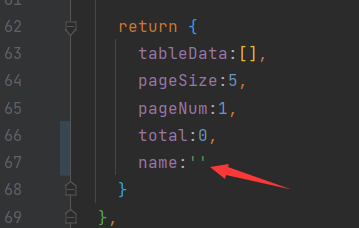


# 添加带参查询功能

先写一个输入框，为其设置样式



查询按钮需要设置点击事件为loadPost，给loadPost函数添加一个参数name，给data的返回值中增加一个参数name，将输入框中获取的参数传进去。



*工作逻辑：从input标签的v-model实时获取的name，这个name会传到data的return中，然后loadPost获取到的就是this.name，把这个值传给loadPost中的name参数，即可返回到后端。*

此时去测试，搜索框已经能够正常通过获取的name进行模糊查询了

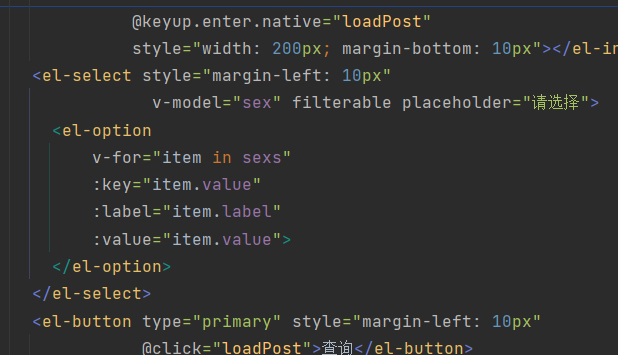


## 增加回车按键搜索的功能

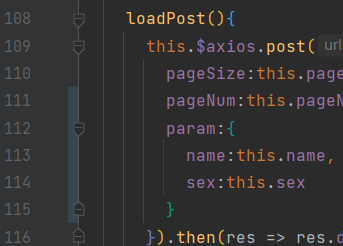
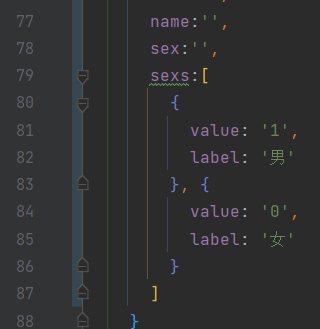
给input元素增加一条属性，@keyup.enter.native="loadPost" 即可实现回车搜索。

## 增加下拉搜索框

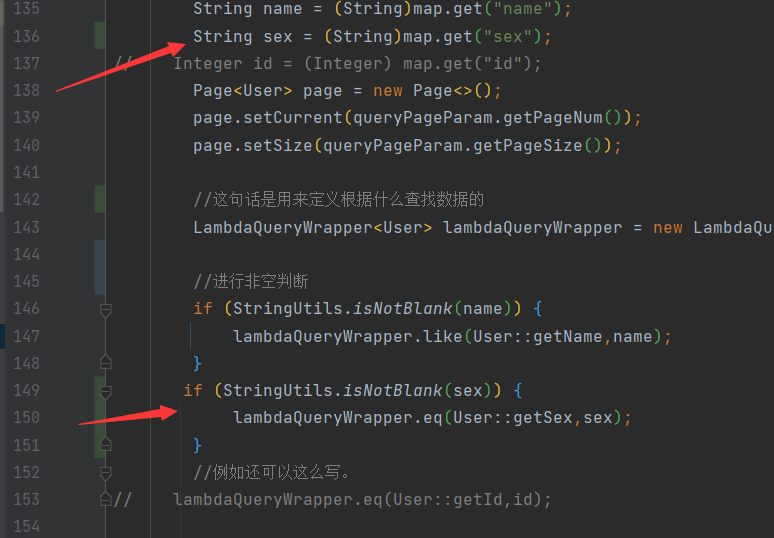
设置搜索框的v-model为sex，内置选项的参数为sexs数组



在data中定义对应的属性，并在loadPost函数中增加相应的查找字段



再去后端增加对应的查询语句即可实现按性别查找。



# 增加添加数据的操作

## 完成前端页面搭建

在重置按钮后面复制一个查询按钮，把点击事件换为新增的add，在method中增加add方法，先设定方法体为空。

到element官网找dialog对话框组件，复制居中布局的el-dialog组件到分页组件后面。



将方法中的参数写到data中，默认值为false（默认不打开对话框）。centerDialogVisible:"false"。

这时再把add方法体中写入this.centerDialogVisible = true

即当点击事件发生后，修改对话框属性为TRUE，即可让对话框弹出来。

为表单增加表项，其中性别使用单选框的形式。同时需要把这些表项放到data的form对象中。



## 提交数据

为表单的确认按钮添加点击事件save，在method中添加该方法。方法体和loadPost一样，直接复制过来，修改传入的data为this.form，修改调用的请求为/user/save。

修改后端的save方法的返回值，让其返回Result对象，只需要修改返回值就可以了。

return userService.save(user) ? Result.success() : Result.fail();

重启后端服务后进行测试，点击确认后弹窗提示添加成功，数据库中相应数据增加了。

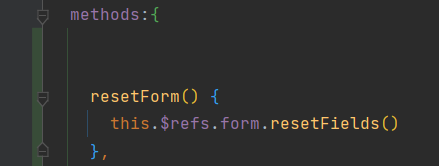


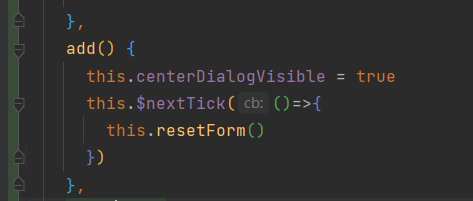
## 进行优化

当提交表单后，不使用alert弹窗，使用消息提示框。并且在添加成功后要关闭对话框且自动刷新表单。使用element的this.$message来提示操作信息。在成功后再次调用this.loadpost来刷新页面。



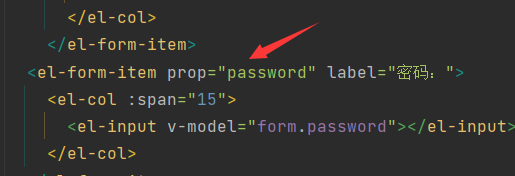
在提交完之后，或者关闭新建窗口后，要重置表单。在method中写一个resetForm方法，然后在按下添加按钮后的点击事件add中调用这个方法来实现该功能。但是这个函数要写到渲染窗口后，否则可能会出bug。这个方法以现在的时间轴来说是不能用的，在写下一个功能时要给每个表项增加prop属性，这样resetForm就生效了。



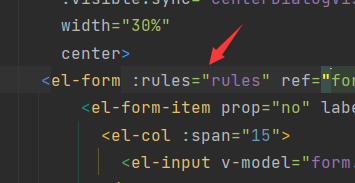


## 增加表单验证功能

验证每个表项的数据合法性，先给每个el-form-item添加prop属性，属性和各自提交的参数一致。



给el-from增加这个属性，并且在data的return中设置rules。



设置的rules为每个表项的合法性检验，其中年龄有个独立的检验方法，要放到data中





其中，年龄和手机号的检验使用了正则表达式匹配，比较复杂。

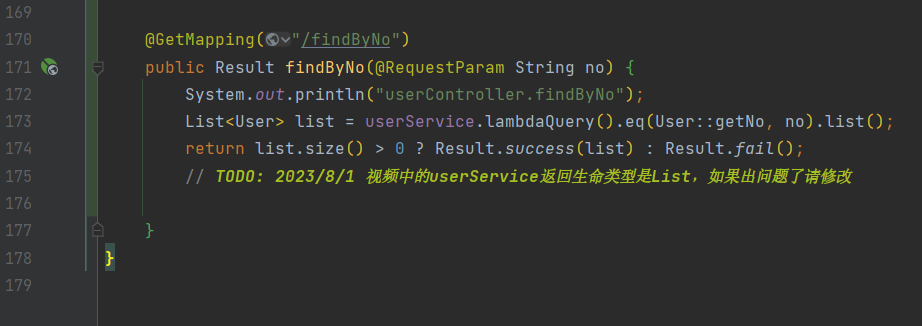


## 添加账号唯一验证校验

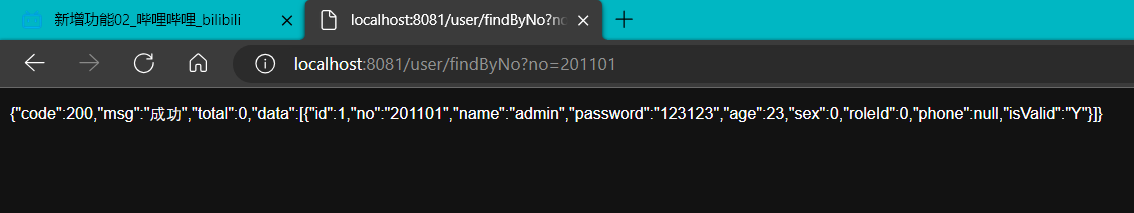
现在data中写一个方法checkDuplication，在方法中发请求到后台查询有没有前台传入的数据，如果有就返回code200，前台提示账号已存在，否则返回400，允许使用该账号注册。



这里从后端获得的数据是一个data，这个data中有包括code，成功失败信息以及查询出来的data。[[2]](#footnote-2)所以前端拿到数据后需要先进行一层解析res=>res.data然后才能从内层的data中获得相应的账号信息。



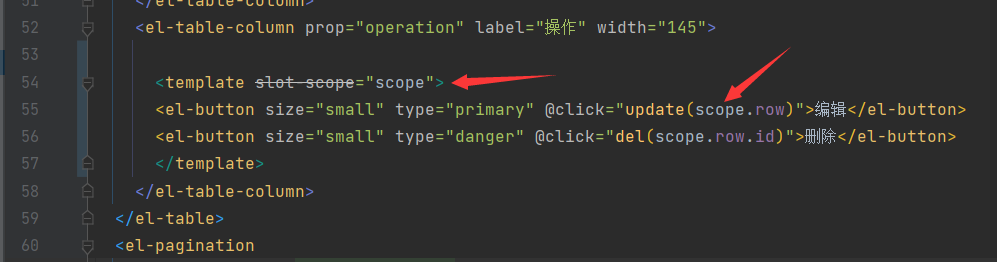
写完之后可以在浏览器中手动输入url进行测试，页面会返回相应的数据。注意，后端项目的端口是8081。



# 添加编辑功能

## 编写前端代码

用一个标签包裹编辑和删除按钮，并且为标签添加插槽属性slot，给两个按钮分别添加对应的点击事件，传入对应scope.row的参数。

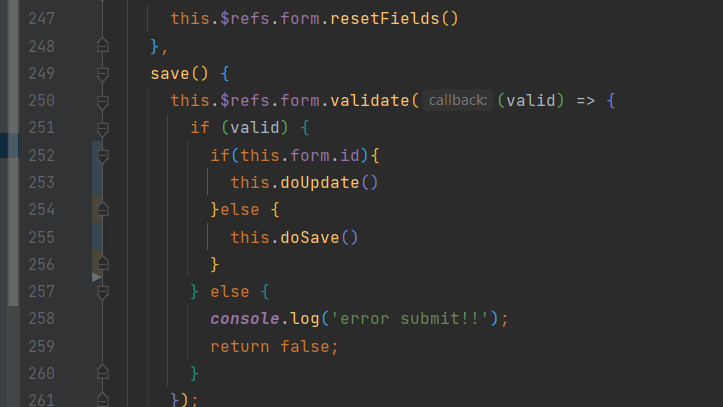


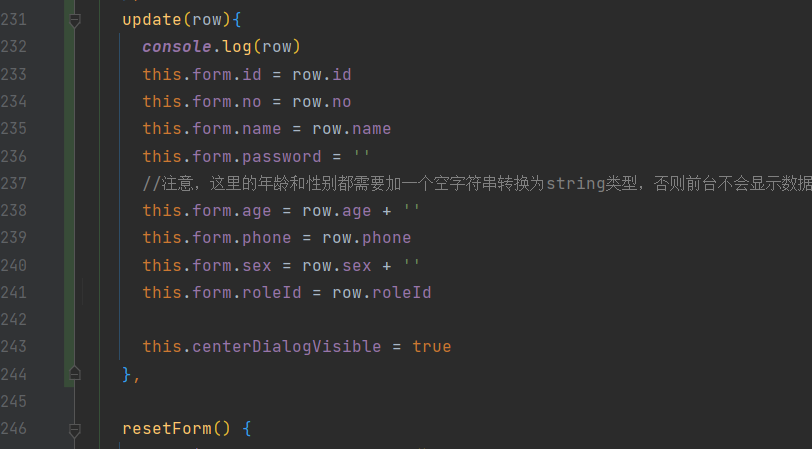
首先需要修改save方法，这个方法是点击对话框的确认按钮所对应的事件，这个事件现在需要对表单获取的id是否为空来判断用户是添加操作还是修改操作。

只要加一个对this.form.id的空值判断即可，如果为空就让其跳转到原来的save方法，如果不为空则让其跳转到update方法。

原来这个方法中的发请求逻辑直接移动到save方法里就行了，然后再复制到update方法中把请求的url改成update就是编辑的逻辑。

在方法区编写对应方法，update方法需要获取被点击的按钮的所在行信息，并且要对若干数据进行格式转换，否则数据无法回显到页面上。

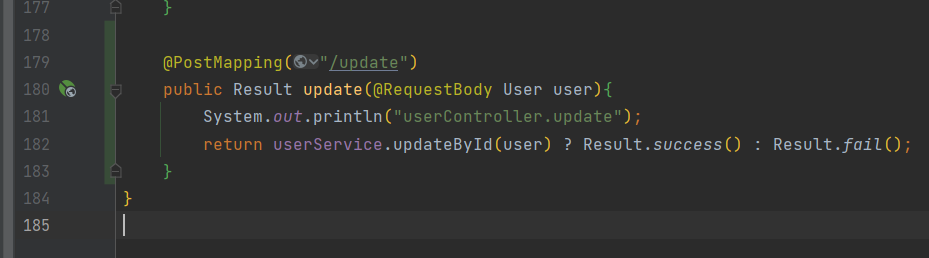






## 编写后端代码

后端只需要重新写一个请求即可。注意updateById返回的是一个Boolean值，可以用三目运算符来直接让请求返回成功与否的Result。

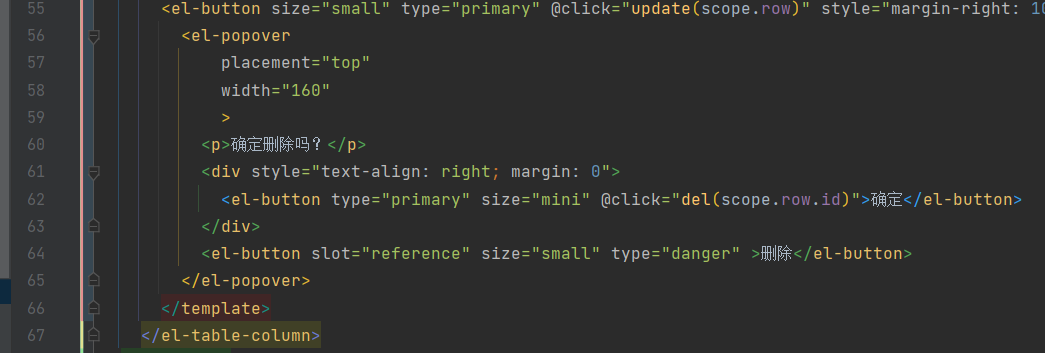


# 添加删除功能

## 编写前端代码

这个项目使用的是elementUI2.8版本的，需要使用低版本的气泡提示框，官网组件中叫popover。

使用官网提供的模板，把代码中的删除按钮放到里面，再进行一些修改，调用del方法即可。



在方法区中写del的执行内容，逻辑和save、update一样，只要把请求改成get请求，url改成del，参数直接在url后拼接即可。

## 编写后端代码

直接复制一份update的方法，把请求注解改成GetMapping，修改方法名和url，修改参数为@RequestParam String id。修改调用的service为deleteById就完成了。然后重启服务器进行测试，在点击确认删除后即可成功删除数据。

*但是有个bug，如果这一页就剩一个数据了，那么把它删掉就会变成暂无数据。这个bug同样在切换页面大小时也会发生：如果一共24条数据，每页5条，如果当前页在3页以后，修改为每页10条，就会显示暂无数据。*

# 常见报错

#The driver has not received any packets from the server

Servlet.service() for servlet [dispatcherServlet] in context with path [] threw exception [Request processing failed; nested exception is org.mybatis.spring.MyBatisSystemException: nested exception is org.apache.ibatis.exceptions.PersistenceException:

### Error querying database. Cause: org.springframework.jdbc.CannotGetJdbcConnectionException: Failed to obtain JDBC Connection;

原因：数据库未启动

# 参考

1、[解决mybatisplus插入报错argument type mismatch的问题\_java\_脚本之家 (jb51.net)](https://www.jb51.net/article/200592.htm)

2、[一则“HTTP 405 Method Not Allowed”的解决办法\_405 (method not allowed)\_蚁方阵的3博客-CSDN博客](https://blog.csdn.net/yiifaa/article/details/80928487)

3、[Unsupported Media Type Http请求415错误\_smile radiantly的博客-CSDN博客](https://blog.csdn.net/zhixingwu/article/details/104759739)

1. 至此已经完成后端项目基础的构建。 [↑](#footnote-ref-1)
2. 参考Result实体类 [↑](#footnote-ref-2)