实验室设备管理系统

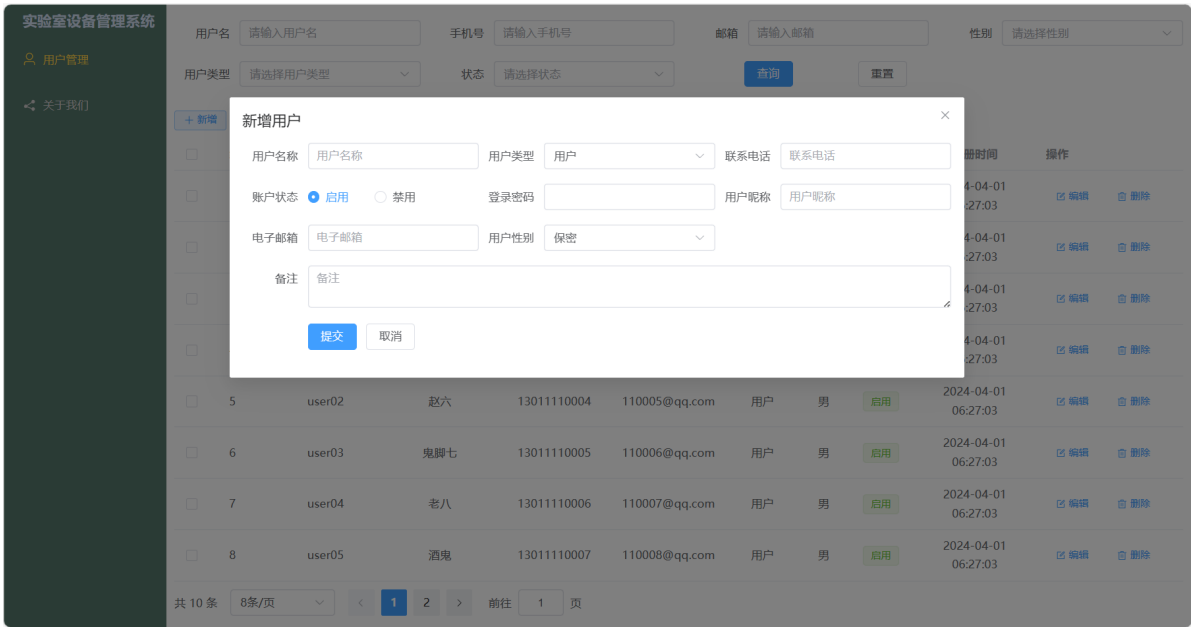
> 我们的项目是一个前后端分离项目，所以项目便分出了：
>
> 前端页面展示（即：Vue 2）
>
> 后端业务接口实现（即：Spring Boot）

项目原型

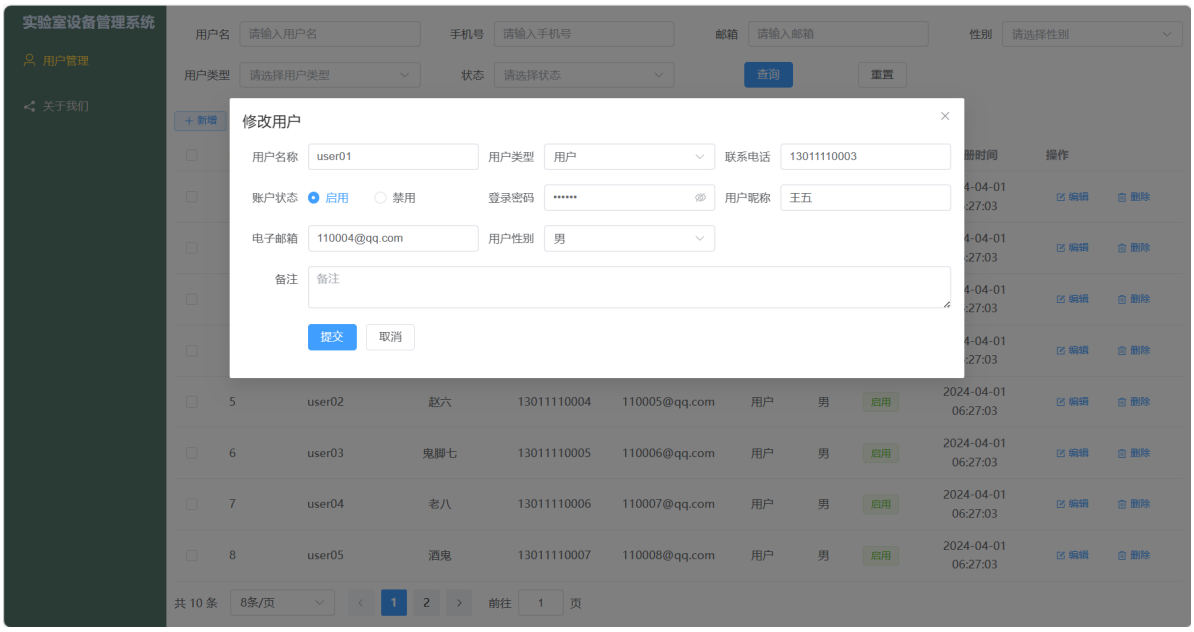
用户管理

查询

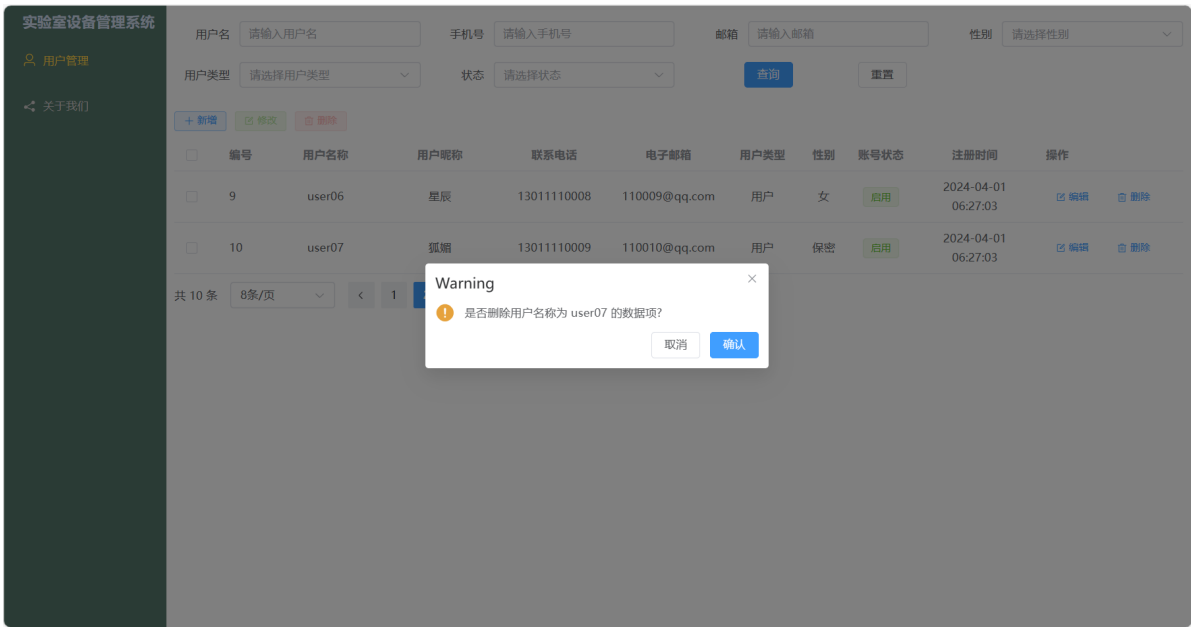
新增



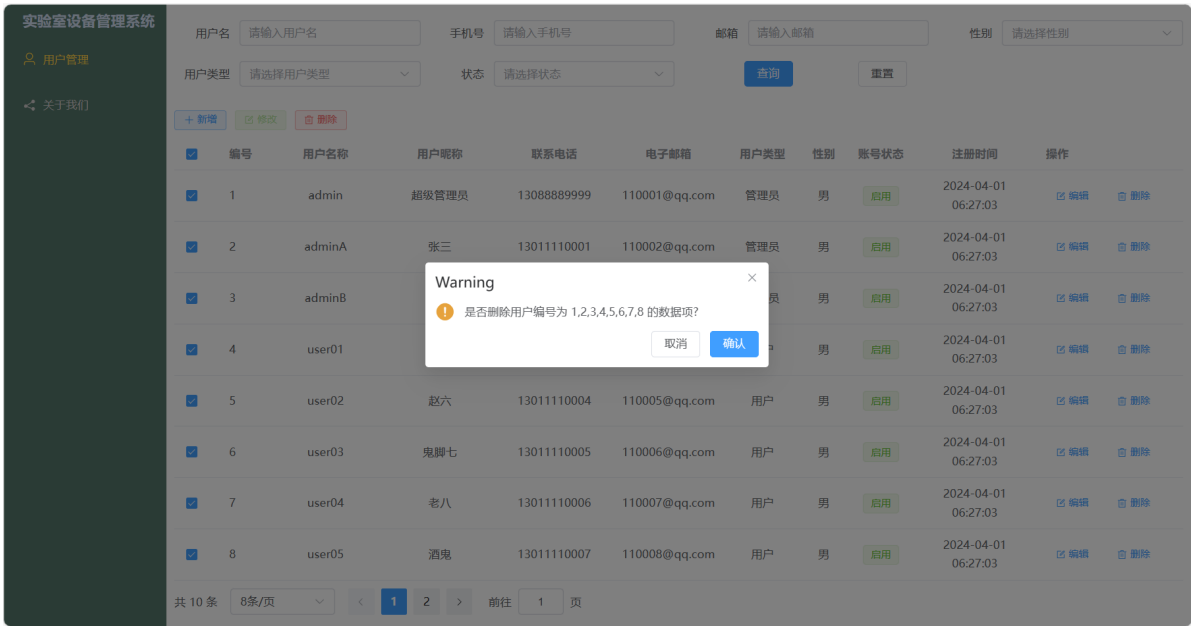
修改


实验室设备管理系统
用户管理
关于我们
修改用户
用户名称 user01
用户类型 用户
联系电话 13011110003
账户状态 启用 禁用
登录密码
用户昵称 王五
电子邮箱 110004@qq.com
用户性别 男
备注
提交
取消

删除


实验室设备管理系统
用户管理
关于我们
Warning
是否删除用户名称为 user07 的数据项?
取消
确认

批量删除



搭建项目

安装nodeJs

> 前置条件：安装nvm
>
> 查看nvm安装路径：nvm root
>
> 配置镜像：在nvm目录下，打开编辑setting.txt文件，添加或替换下面两行
>
> node_mirror: https://npmmirror.com/mirrors/node/
> npm_mirror: https://npmmirror.com/mirrors/npm/

验证是否安装nvm

```
nvm -v
```

```
C:\Users\lenovo>nvm -v
1.1.12
```

查看已安装的node版本列表

```
nvm ls
```

```
C:\Users\lenovo>nvm ls

    20.11.1
  * 16.13.0 (Currently using 64-bit executable)
    0.0.0
```

安装对应版本的node

```
nvm install 16.13.0
```

切换对应版本的nodejs

```
nvm use 16.13.0
```

验证是否安装node

```
node -v
```

```
C:\Users\lenovo>node -v
v16.13.0
```

安装Vue Cli

> 必要环境：nodejs

在终端中输入命令进行安装

```
npm install -g @vue/cli
```

创建Vue项目

在要创建项目的目录下打开终端，输入以下命令

```
vue create manage_ui
```

手动选择安装

```
Vue CLI v5.0.8
? Please pick a preset:
  vue2-DIY ([Vue 2] less, babel, router, vuex, eslint)
  Default ([Vue 3] babel, eslint)
  Default ([Vue 2] babel, eslint)
> Manually select features      手动选择
```

根据项目需求选择

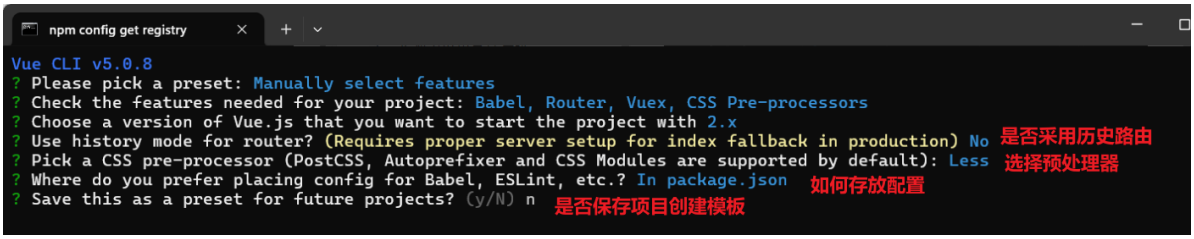
```
Vue CLI v5.0.8
? Please pick a preset: Manually select features
? Check the features needed for your project: (Press <space> to select, <a> to toggle all, <i> to inve
<enter> to proceed)
 (*) Babel           转码器。可以将ES6代码转为ES5代码
 ( ) TypeScript
 ( ) Progressive Web App (PWA) Support
 (*) Router  Vue路由
 (*) Vuex    Vuex状态管理
>(*) CSS Pre-processors     Css预处理器
 ( ) Linter / Formatter
 ( ) Unit Testing
 ( ) E2E Testing

      上下键选择，空格选中或者取消选择
```

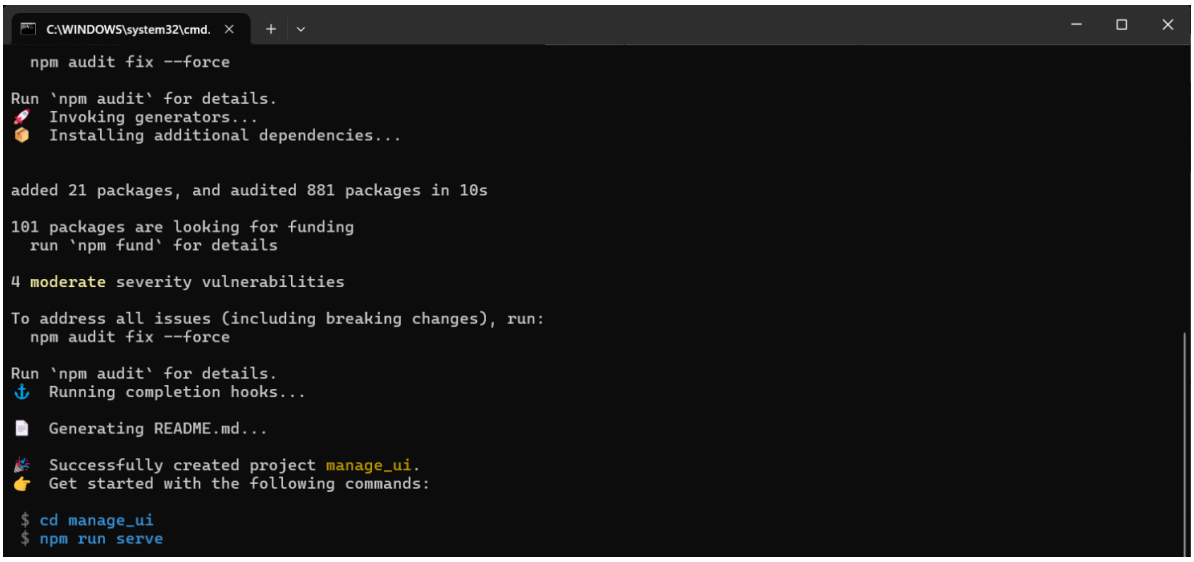
选择vue版本

```
npm config get registry
Vue CLI v5.0.8
? Please pick a preset: Manually select features
? Check the features needed for your project: Babel, Router, Vuex, CSS Pre-processors
? Choose a version of Vue.js that you want to start the project with
  3.x
> 2.x       选择Vue 2版本
```

其他配置



项目完成创建



搭建spring boot项目

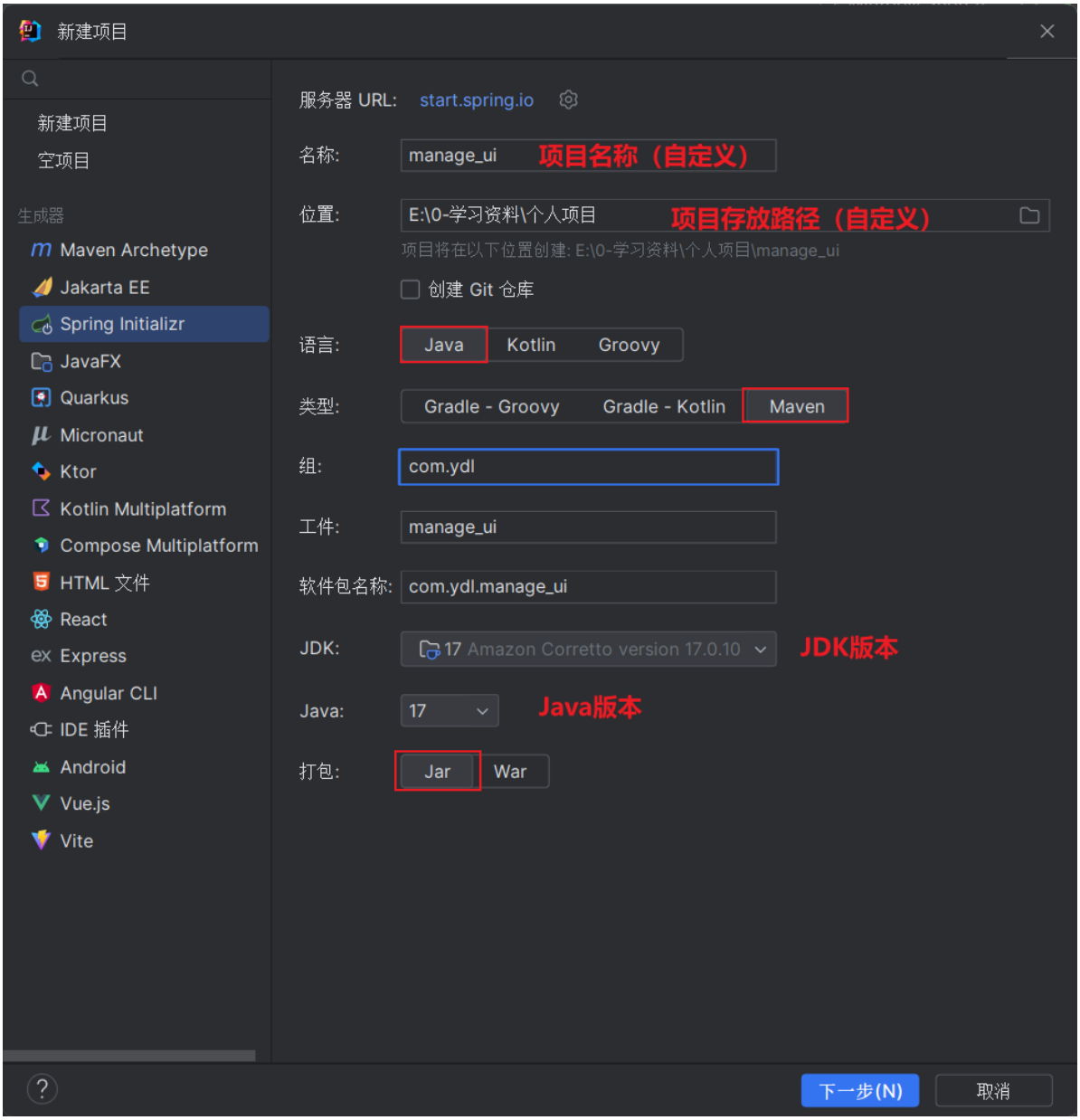
准备数据表（tb_user）

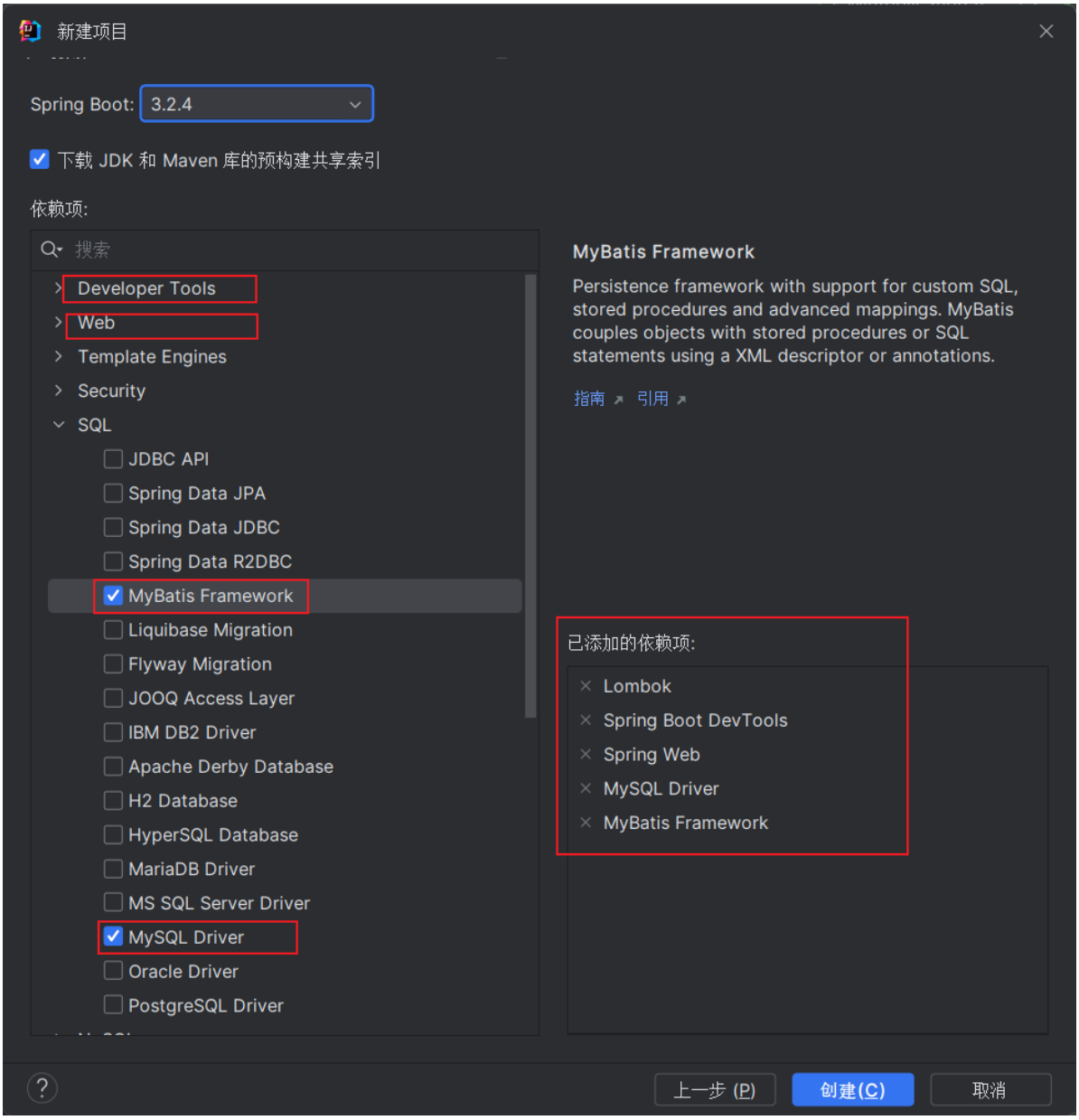
```
-- 创建数据库
CREATE DATABASE IF NOT EXISTS manage_db DEFAULT CHARSET utf8mb4 COLLATE
utf8mb4_0900_ai_ci;

-- 创建用户表
CREATE TABLE tb_user (
id BIGINT AUTO_INCREMENT PRIMARY KEY COMMENT '用户ID',
username varchar(128) NOT NULL UNIQUE KEY COMMENT '用户名',
tel varchar(64) NOT NULL COMMENT '手机号',
email varchar(255) NULL DEFAULT NULL COMMENT '邮箱',
password varchar(255) NULL DEFAULT NULL COMMENT '密码',
icon varchar(255) NULL DEFAULT NULL COMMENT '头像地址',
nickname varchar(255) NULL DEFAULT NULL COMMENT '用户昵称',
gender int NOT NULL DEFAULT 0 COMMENT '性别，0：保密，1：男，2：女',
age int NULL DEFAULT NULL COMMENT '年龄',
type int NOT NULL DEFAULT 1 COMMENT '用户类型，0：管理员，1：用户',
status int NOT NULL DEFAULT 1 COMMENT '账号状态，1：正常，0：停用',
remark varchar(128) NULL DEFAULT NULL COMMENT '备注',
create_by varchar(128) NOT NULL COMMENT '创建人',
create_time timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',
update_by varchar(128) NULL DEFAULT NULL COMMENT '修改人',
update_time timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP COMMENT '最后修改时间',
username_update_time timestamp NULL DEFAULT NULL COMMENT '用户名最后修改时间',
is_del int NOT NULL DEFAULT 0 COMMENT '是否删除，0：否，1：是'
) COMMENT '用户表';
```

```sql
-- 插入用户表数据
INSERT INTO tb_user (id, username, tel, email, password, nickname, gender, age,
type, create_by) VALUES
(1,   'admin',  '13088889999',  '110001@qq.com', 'admin',   '超级管理员',    1,
26, 0, 'admin'),
(2,   'adminA', '13011110001',  '110002@qq.com', '123456',  '张三',         1,
24, 0, 'admin'),
(3,   'adminB', '13011110002',  '110003@qq.com', '123456',  '李四',         1,
25, 0, 'admin'),
(4,   'user01', '13011110003',  '110004@qq.com', '123456',  '王五',         1,
18, 1, 'admin'),
(5,   'user02', '13011110004',  '110005@qq.com', '123456',  '赵六',         1,
19, 1, 'admin'),
(6,   'user03', '13011110005',  '110006@qq.com', '123456',  '鬼脚七',       1,
20, 1, 'admin'),
(7,   'user04', '13011110006',  '110007@qq.com', '123456',  '老八',         1,
20, 1, 'admin'),
(8,   'user05', '13011110007',  '110008@qq.com', '123456',  '酒鬼',         1,
21, 1, 'admin'),
(9,   'user06', '13011110008',  '110009@qq.com', '123456',  '时间',         0,
23, 1, 'admin'),
(10,  'user07', '13011110009',  '110010@qq.com', '123456',  '诗怡',         2,
19, 1, 'admin'),
(11,  'user08', '13011110010',  '110011@qq.com', '123456',  '霜柳',         2,
18, 1, 'admin'),
(12,  'user09', '13011110011',  '110012@qq.com', '123456',  '泗水',         0,
22, 1, 'admin');
```

创建springboot项目


新建项目
新建项目
空项目
生成器
Maven Archetype
Jakarta EE
Spring Initializr
JavaFX
Quarkus
Micronaut
Ktor
Kotlin Multiplatform
Compose Multiplatform
HTML 文件
React
Express
Angular CLI
IDE 插件
Android
Vue.js
Vite
服务器 URL: start.spring.io
名称: manage_ui
项目名称（自定义）
位置: E:\0-学习资料\个人项目
项目存放路径（自定义）
项目将在以下位置创建: E:\0-学习资料\个人项目\manage_ui
创建 Git 仓库
语言: Java Kotlin Groovy
类型: Gradle - Groovy Gradle - Kotlin Maven
组: com.ydl
工件: manage_ui
软件包名称: com.ydl.manage_ui
JDK: 17 Amazon Corretto version 17.0.10
JDK版本
Java: 17
Java版本
打包: Jar War
下一步(N)
取消



> 由于使用start.spring.io进行创建项目，spring boot的版本过高，处于spring 3。需要人为将版本将为spring boot 2

修改pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.5.15</version> <!-- 修改springboot版本-->
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.ydl</groupId>
  <artifactId>manage_ui</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>manage_ui</name>
  <description>manage_ui</description>
```

```
  <properties>
    <java.version>17</java.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
      <groupId>com.baomidou</groupId>
      <artifactId>mybatis-plus-boot-starter</artifactId>
      <version>3.4.2</version>
    </dependency>
    <!-- pageHelper 分页 -->
    <dependency>
      <groupId>com.github.pagehelper</groupId>
      <artifactId>pagehelper-spring-boot-starter</artifactId>
      <version>1.4.1</version>
    </dependency>
    <!--huTool 工具 -->
    <dependency>
      <groupId>cn.hutool</groupId>
      <artifactId>hutool-all</artifactId>
      <version>5.7.17</version>
    </dependency>

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-devtools</artifactId>
      <scope>runtime</scope>
      <optional>true</optional>
    </dependency>
    <!-- MySQL依赖-->
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>8.0.28</version>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.projectlombok</groupId>
      <artifactId>lombok</artifactId>
      <optional>true</optional>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring
          -boot-maven-plugin</artifactId>
        <configuration>
          <excludes>
```

```
            <exclude>
              <groupId>org.projectlombok</groupId>
              <artifactId>lombok</artifactId>
            </exclude>
          </excludes>
        </configuration>
      </plugin>
    </plugins>
  </build>

</project>
```

> 修改完pom文件后，需要刷新以下maven

创建项目架构

我们项目后端采用 spring boot + MybatisPlus 进行开发。所以，我们的项目包将采用MSCM架构，分有：
model： 数据模型层。存放实体类，与数据库中的属性值基本保持一致。 一般命名有 pojo、entity、domain

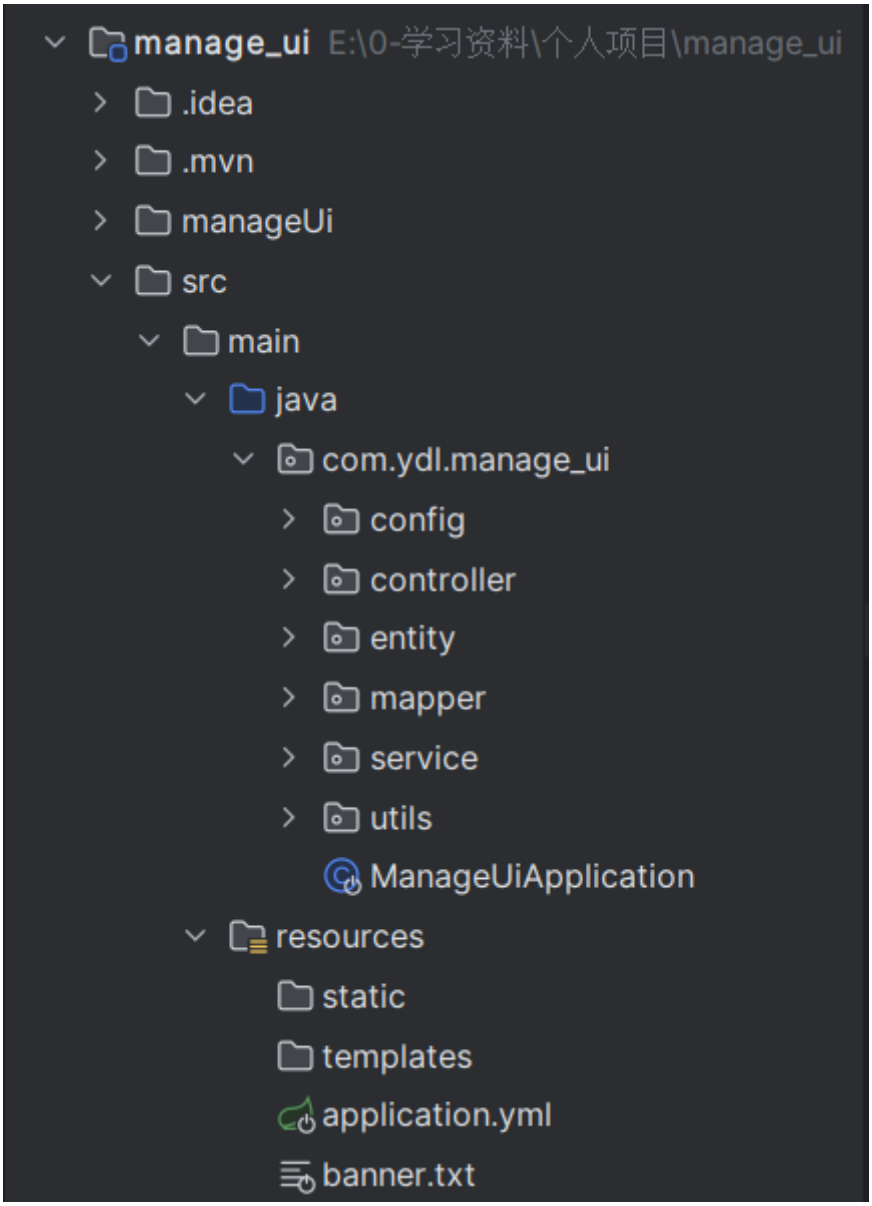
mapper：数据操作层。mapper层直接与数据库打交道（执行SQL语句），接口提供给service层。一般命名有mapper

service：业务逻辑层。主要是针对具体的问题的操作，把一些数据层的操作进行组合，间接与数据库打交道（提供操作数据库的方法）。一般命名为：service

且做这一层，需要先设计接口（属于service包下），再实现类（属于service下，impl包下）。

controller：控制层。controller通过service的接口来控制业务流程，也可通过接收前端传过来的参数进行业务操作。一般命名为：controller

项目具体架构图如下



修改spring boot配置文件

> 配置文件名称： application.yml
>
> ps：若是properties后缀，可改为yml

```yaml
spring:
  application:
    name: manage_ui # 项目名称
  datasource: # 配置数据源
    driver-class-name: com.mysql.cj.jdbc.Driver
    # manage_db 数据库名称（需要根据实际数据库名而定）
    url: jdbc:mysql://localhost:3306/manage_db?
useUnicode=true&characterEncoding=utf8&zeroDateTimeBehavior=convertToNull&useSSL
=true&serverTimezone=GMT%2B8
    username: root
    password: mysql@123456

mybatis-plus:
  global-config:
    db-config:
      id-type: auto # 主键自增
  # 搜索指定包别名 实体类所在的包
  type-aliases-package: com.ydl.manage_ui.entity
```

```yaml
  # 配置mapper的扫描，找到所有的mapper.xml映射文件
  mapper-locations: classpath*:mapper/*Mapper.xml

# 开发环境配置
server:
  # 服务器的HTTP端口，默认为8080
  port: 8080
  servlet:
    # 应用访问的根路径
    context-path: "/api"
  tomcat:
    # tomcat的URI编码
    uri-encoding: UTF-8
    # 连接数满后的排队数，默认为100
    accept-count: 1000
    threads:
      # tomcat最大线程数，默认为200
      max: 800
      # Tomcat启动初始化的线程数，默认值10
      min-spare: 100

# 分页插件PageHelper配置
pagehelper:
  helper-dialect: mysql
  reasonable: true
  support-methods-arguments: true
  params: count=countSql
  page-size-zero: true
```

创建实体类

创建实体类父类

为了避免重复类成员，可以创建一个父类对相同的类成员进行维护。且还可以利用父类，拓展一些业务需要的类成员。如分页需要的pageNumber（页码）、pageSize（数量）。

```java
package com.ydl.manage_ui.entity.base;

import com.baomidou.mybatisplus.annotation.TableField;
import com.baomidou.mybatisplus.annotation.TableId;
import com.fasterxml.jackson.annotation.JsonFormat;
import lombok.Data;

import java.util.Date;
import java.util.List;

@Data
public class BaseEntity {
    /** 页码 */
    @TableField(exist = false) // 该注解注明，该字段在表中不存在
    private Long pageNumber;
    /** 页数 */
    @TableField(exist = false)
    private Long pageSize;
```

```
    /** 主键 */
    @TableId("id")
    @JsonFormat(shape = JsonFormat.Shape.STRING) // 将Long类型转为String类型传给前端，避免long类型精度浮动
    private Long id;

    /** 创建人 */
    @TableField("create_by") // 该注解注明，该属性对应的表字段
    private String createBy;
    /** 创建时间 */
    @TableField("create_time")
    @JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss")
    private Date createTime;
    /** 更新人 */
    @TableField("update_by")
    private String updateBy;
    /** 更新时间 */
    @TableField("update_time")
    @JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss")
    private Date updateTime;
    /** 备注 */
    @TableField("remark")
    private String remark;

    /** IDs */
    @TableField(exist = false)
    private List<Long> ids;
}
```

创建实体类（TbUser）

```
package com.ydl.manage_ui.entity;

import com.baomidou.mybatisplus.annotation.*;
import com.fasterxml.jackson.annotation.JsonFormat;
import com.ydl.manage_ui.entity.base.BaseEntity;
import lombok.Data;

import java.util.Date;

@Data
@TableName("tb_user") // 表名
public class TbUser extends BaseEntity { // 继承父类
    /** 用户名 */
    @TableField("username") // 用户实体类对应用户表的字段
    private String username;

    /** 手机号 */
    @TableField("tel")
    private String tel;

    /** 邮箱 */
    @TableField("email")
    private String email;

    /** 密码 */
```

```
    @TableField("password")
    private String password;

    /** 头像地址 */
    @TableField("icon")
    private String icon;

    /** 用户昵称 */
    @TableField("nickname")
    private String nickname;

    /** 性别 */
    @TableField("gender")
    private Integer gender;

    /** 年龄 */
    @TableField("age")
    private Integer age;

    /** 用户类型 0：管理员，1：用户 */
    @TableField("type")
    private Integer type;

    /** 账号状态 0：停用，1：正常 */
    @TableField("status")
    private Integer status;

    /** 用户名最后修改时间 */
    @TableField("username_update_time")
    @JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss")
    private Date usernameUpdateTime;

    /** 是否删除 0：否，1：是 */
    @TableLogic
    @TableField("is_del")
    private Integer isDel;
}
```

创建mapper层

创建UserMapper

```
package com.ydl.manage_ui.mapper;

import com.baomidou.mybatisplus.core.mapper.BaseMapper;
import com.ydl.manage_ui.entity.TbUser;
import org.apache.ibatis.annotations.Mapper;

// 该注解代表代表该类属于mapper层，后续也更方便springboot集成mp对其识别
@Mapper
public interface UserMapper extends BaseMapper<TbUser> { // 使用mp后，mapper接口需要继承BaseMapper
}
```

创建Service层

创建IUserService接口

```
package com.ydl.manage_ui.service;

import com.baomidou.mybatisplus.extension.service.IService;
import com.ydl.manage_ui.entity.TbUser;
import org.springframework.transaction.annotation.Transactional;

// 开启事务回滚 执行业务代码中遇到异常，会将sql事务进行回滚
@Transactional
public interface IUserService extends IService<TbUser>{
}
```

实现IUserService接口，创建UserServiceImpl实现类

```
package com.ydl.manage_ui.service.impl;

import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;
import com.ydl.manage_ui.entity.TbUser;
import com.ydl.manage_ui.mapper.UserMapper;
import com.ydl.manage_ui.service.IUserService;
import org.springframework.stereotype.Service;

// 该注解代表该类属于Service层
@Service
public class UserServiceImpl extends ServiceImpl<UserMapper, TbUser> implements
IUserService {
}
```

创建Controller层

封装ResultVO

为了更好的跟前端打好关系，我们通常需要对后端返回的数据进行包装一下。让我们的接口返回的数据更加规范，更加的优雅美观。

```
package com.ydl.manage_ui.entity.vo;

import lombok.Data;

/**
 * 返回数据封装类
 *
 * @author 林武泰
 */
@Data
public class ResultVO {
    /** 返回状态码 */
    private Integer code;

    /** 返回数据 */
    private Object data;
```

```java
    /** 返回消息 */
    private String message;

    /**
     * 请求结果返回封装
     * @param code 返回结果
     * @param data 返回数据
     * @param message 返回消息
     */
    public ResultVO(Integer code, Object data, String message) {
        this.code = code;
        this.data = data;
        this.message = message;
    }

    /**
     * 操作成功，无参
     *
     * @return ResultVO
     */
    public static ResultVO success() {
        return new ResultVO(200, null, "操作成功");
    }

    /**
     * 操作成功，只返回消息
     *
     * @param message 返回消息
     * @return ResultVO
     */
    public static ResultVO success(String message) {
        return new ResultVO(200, null, message);
    }

    /**
     * 操作成功，只返回数据
     *
     * @param data 返回数据
     * @return ResultVO
     */
    public static ResultVO success(Object data) {
        return new ResultVO(200, data, "操作成功");
    }

    /**
     * 操作成功，返回数据与消息
     *
     * @param data 返回数据
     * @param message  返回消息
     * @return ResultVO
     */
    public static ResultVO success(Object data, String message) {
        return new ResultVO(200, data, message);
    }

    /**
     * 操作失败，无参
     *
```

```
     * @return ResultVO
     */
    public static ResultVO error() {
        return new ResultVO(500, null, "操作失败");
    }

    /**
     * 操作失败，只返回数据
     *
     * @param data 返回数据
     * @return ResultVO
     */
    public static ResultVO error(Object data) {
        return new ResultVO(500, data, "操作失败");
    }

    /**
     * 操作失败
     *
     * @param errMessage 返回失败消息
     * @return ResultVO
     */
    public static ResultVO error(String errMessage) {
        return new ResultVO(500, null, errMessage);
    }
}
```

封装PageResult

```
package com.ydl.manage_ui.entity.vo;

import lombok.Data;

import java.util.List;

/**
 * 返回分页数据封装类
 *
 * @author 林武泰
 */
@Data
public class PageResult {
    /** 数据集合 */
    private List<?> records;
    /** 总条数 */
    private long total;
    /** 当前页数量 */
    private long size;
    /** 当前页码 */
    private long current;
}
```

创建分页工具类

```
package com.ydl.manage_ui.utils;

import com.github.pagehelper.PageInfo;
import com.ydl.manage_ui.entity.vo.PageResult;

import java.util.Collections;
import java.util.List;

/**
 * 分页工具
 *
 * @author 林武泰
 */
public class PageUtil {
    public static PageResult getPageResult(PageInfo<?> pageInfo) {
        PageResult pageResult = new PageResult();
        // 分页数据
        pageResult.setRecords(pageInfo.getList());
        // 当前页数
        pageResult.setSize(pageInfo.getPageSize());
        // 数据总数
        pageResult.setTotal(pageInfo.getTotal());
        // 当前页码
        pageResult.setCurrent(pageInfo.getPageNum());

        return pageResult;
    }

    /**
     * 自定义数据分页
     *
     * @param pageNumber 页码
     * @param pageSize 页数
     * @param allData 分页数据
     * @return 分页结果
     * @param <T> 数据类型
     */
    public static <T> PageResult myStartPage(Integer pageNumber, Integer
pageSize, List<T> allData) {
        PageResult pageResult = new PageResult();
        List<T> records;
        long total = 0;

        if (allData == null || allData.isEmpty()) {
            records = Collections.emptyList();
        } else {
            Integer number = pageNumber;
            Integer size = pageSize;
            total = allData.size();

            if (pageNumber == null || pageNumber <= 0) {
                pageNumber = 1;
                number = pageNumber;
            }
```

```
            if (pageSize == null || pageSize <= 0 || size > allData.size()) {
                pageSize = allData.size();
                size = allData.size();
            }

            if (number == 1) {
                // 含头不含尾
                records = allData.subList(0, size);
            } else {
                // 获取每页开始值 页码 * 页数 获取开始截取值
                int eachNumber = number * size;
                // 获取当前页数量 每页开始值 + 页数 获取结束截取值
                int eachSize = eachNumber + size;
                if (eachSize > allData.size() || eachNumber > allData.size()) {
                    records = Collections.emptyList();
                } else {
                    // 含头不含尾
                    records = allData.subList(eachNumber, eachSize);
                }
            }
        }

        // 分页数据
        pageResult.setRecords(records);
        // 当前页数
        pageResult.setSize(pageSize == null ? -1 : pageSize);
        // 数据总数
        pageResult.setTotal(total);
        // 当前页码
        pageResult.setCurrent(pageNumber);

        return pageResult;
    }
}
```

创建Controller父类

为了更好的帮助Controller接口的业务实现，我们可以创建一个Controller父类，将必要的一些数据封装，开启分页进行封装。

```
package com.ydl.manage_ui.entity.base;

import com.github.pagehelper.PageHelper;
import com.github.pagehelper.PageInfo;
import com.ydl.manage_ui.entity.vo.ResultVO;
import com.ydl.manage_ui.utils.PageUtil;

import java.util.List;

public class BaseController {
    // 默认成功消息
    String SUCCESS_MESSAGE = "操作成功";
    // 默认失败消息
    String ERROR_MESSAGE = "操作失败";
```

```
    /** 返回成功消息 */
    protected ResultVO success() {
        return ResultVO.success(SUCCESS_MESSAGE);
    }

    /**
     * 返回成功消息
     *
     * @param data 返回数据
     */
    protected ResultVO success(Object data) {
        return ResultVO.success(data, SUCCESS_MESSAGE);
    }

    /**
     * 返回成功消息
     *
     * @param message 返回消息
     */
    protected ResultVO success(String message) {
        return ResultVO.success(message);
    }

    /**
     * 返回成功消息
     *
     * @param data 返回数据
     * @param message 返回消息
     */
    protected ResultVO success(Object data, String message) {
        return ResultVO.success(data, message);
    }

    /** 返回失败消息 */
    protected ResultVO error() {
        return ResultVO.error(ERROR_MESSAGE);
    }

    /**
     * 返回失败消息
     *
     * @param data 返回数据
     */
    protected ResultVO error(Object data) {
        return ResultVO.error(data);
    }

    /**
     * 返回失败消息
     *
     * @param message 返回消息
     */
    protected ResultVO error(String message) {
        return ResultVO.error(message);
    }

    /**
     * 返回消息
```

```
     *
     * @param flag 操作结果
     */
    protected ResultVO toResult(boolean flag) {
        return flag ? success() : error();
    }

    /**
     * 设置开启分页
     */
    protected void startPage(Long pageNumber, Long pageSize)
    {
        PageHelper.startPage(pageNumber.intValue(), pageSize.intValue());
    }

    /**
    * 返回分页数据
    */
    protected ResultVO getPageResult(List<?> list) {
        PageInfo<?> pageInfo = new PageInfo<>(list);
        return success(PageUtil.getPageResult(pageInfo));
    }
}
```

创建UserController

当dao层和service层准备完成后，我们就可以创建一个Controller，将业务写为一个个的接口。

```
package com.ydl.manage_ui.controller;

import cn.hutool.core.util.StrUtil;
import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
import com.ydl.manage_ui.entity.TbUser;
import com.ydl.manage_ui.entity.base.BaseController;
import com.ydl.manage_ui.entity.vo.ResultVO;
import com.ydl.manage_ui.service.IUserService;
import org.springframework.web.bind.annotation.*;

import javax.annotation.Resource;
import java.util.List;

// 表明该类为rest风格的控制器
@RestController
@RequestMapping("/user") // 统一接口归属 如：/user/list
public class UserController extends BaseController {
    // 自动注入业务逻辑层
    @Resource
    private IUserService userService;

    @GetMapping("/{id}") // 加上配置文件中的根路径，该接口的访问为
http://127.0.0.1:8080/api/user/1
    public ResultVO getId(@PathVariable("id") Long id) {
        return success(userService.getById(id));
    }

    @PostMapping("/list")
    public ResultVO list(@RequestBody TbUser tbUser) {
```

```
        // 获取页码
        Long pageNumber = tbUser.getPageNumber();
        // 获取条数
        Long pageSize = tbUser.getPageSize();
        // 开启分页
        startPage(pageNumber, pageSize);

        QueryWrapper<TbUser> queryWrapper = new QueryWrapper<>();
        String username = tbUser.getUsername();
        if (StrUtil.isNotBlank(username)) {
            queryWrapper.like("username", username);
        }
        String tel = tbUser.getTel();
        if (StrUtil.isNotBlank(tel)) {
            queryWrapper.like("tel", tel);
        }
        String email = tbUser.getEmail();
        if (StrUtil.isNotBlank(email)) {
            queryWrapper.like("email", email);
        }
        Integer gender = tbUser.getGender();
        if (gender != null) {
            queryWrapper.eq("gender", gender);
        }
        Integer status = tbUser.getStatus();
        if (status != null) {
            queryWrapper.eq("status", status);
        }
        Integer type = tbUser.getType();
        if (type != null) {
            queryWrapper.eq("type", type);
        }

        return getPageResult(userService.list(queryWrapper));
    }

    @PostMapping
    public ResultVO save(@RequestBody TbUser tbUser) {
        tbUser.setCreateBy("admin");
        return toResult(userService.save(tbUser));
    }

    @PutMapping
    public ResultVO update(@RequestBody TbUser tbUser) {
        tbUser.setUpdateBy("admin");
        return toResult(userService.updateById(tbUser));
    }

    @DeleteMapping("/{id}")
    public ResultVO delete(@PathVariable("id") Long id) {
        return toResult(userService.removeById(id));
    }

    @DeleteMapping
    public ResultVO deleteBatch(@RequestParam List<Long> ids) {
        return toResult(userService.removeByIds(ids));
    }
}
```

解决跨域

创建一个mvc配置类

```
package com.ydl.manage_ui.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

/**
 * WebMvc配置类
 *
 * @author 林武泰
 */
@Configuration
public class MvcConfig implements WebMvcConfigurer {
    /**
     * 解决跨域问题
     */
    @Override
    public void addCorsMappings(CorsRegistry registry) {
        // 设置允许跨域的路径
        registry.addMapping("/**")
                // 设置允许跨域请求的域名
                .allowedOriginPatterns("*")
                // 设置允许cookie
                .allowCredentials(true)
                // 设置允许请求方式
                .allowedMethods("GET","POST","DELETE","PUT","OPTIONS")
                // 设置允许的header请求
                .allowedHeaders("*")
                // 设置跨域允许时间
                .maxAge(3600);
    }
}
```