前端使用vue.css渲染动态表格

（未渲染界面）



渲染界面



关键代码



**new Vue({**

**el: '#app',**

**data: {**

**users: [**

**{ id: '1', breakfast: '豆浆油条', lunch: '鱼香肉丝', dinner: '爆炒青椒' }**

**],**

**newUser: {**

**id: '',**

**breakfast: '',**

**lunch: '',**

**dinner: ''**

**}**

**},**

**methods: {**

**addUser() {**

**this.users.push({**

**id: this.newUser.id,**

**breakfast: this.newUser.breakfast,**

**lunch: this.newUser.lunch,**

**dinner: this.newUser.dinner**

**});**

**this.clearInputs();**

**},**

**deleteUser(index) {**

**this.users.splice(index, 1);**

**},**

**clearInputs() {**

**this.newUser = {**

**id: '',**

**breakfast: '',**

**lunch: '',**

**dinner: ''**

**};**

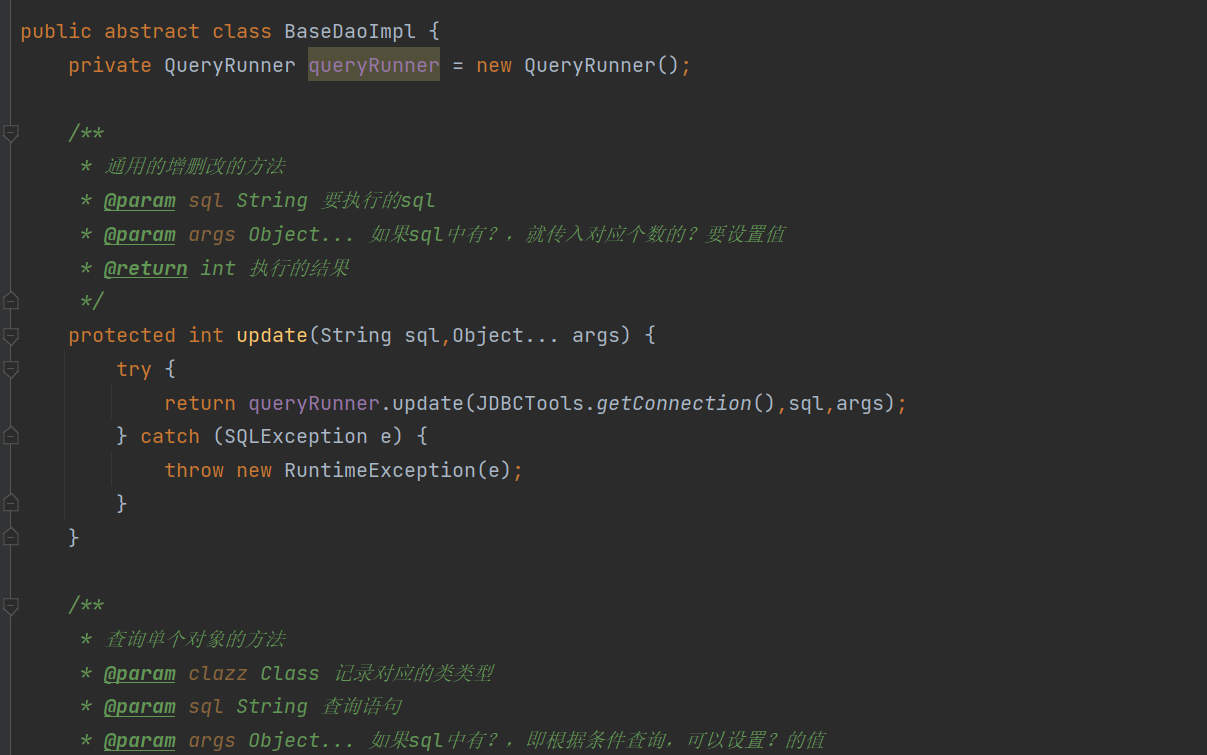
**}**

**}**

**});**

**</script>**

# 使用beasdao层封装底层增删改查代码，优化项目，解耦合



**public abstract class BaseDaoImpl {**

**private QueryRunner queryRunner = new QueryRunner();**

**/\*\***

**\* 通用的增删改的方法**

**\* @param sql String 要执行的sql**

**\* @param args Object... 如果sql中有？，就传入对应个数的？要设置值**

**\* @return int 执行的结果**

**\*/**

**protected int update(String sql,Object... args) {**

**try {**

**return queryRunner.update(JDBCTools.getConnection(),sql,args);**

**} catch (SQLException e) {**

**throw new RuntimeException(e);**

**}**

**}**

**/\*\***

**\* 查询单个对象的方法**

**\* @param clazz Class 记录对应的类类型**

**\* @param sql String 查询语句**

**\* @param args Object... 如果sql中有？，即根据条件查询，可以设置？的值**

**\* @param <T> 泛型方法声明的泛型类型**

**\* @return T 一个对象**

**\*/**

**protected <T> T getBean(Class<T> clazz, String sql, Object... args){**

**List<T> list = getList(clazz, sql, args);**

**if(list != null && list.size()>0) {**

**return getList(clazz, sql, args).get(0);**

**}**

**return null;**

**}**

**/\*\***

**\* 通用查询多个对象的方法**

**\* @param clazz Class 记录对应的类类型**

**\* @param sql String 查询语句**

**\* @param args Object... 如果sql中有？，即根据条件查询，可以设置？的值**

**\* @param <T> 泛型方法声明的泛型类型**

**\* @return List<T> 把多个对象放到了List集合**

**\*/**

**protected <T> List<T> getList(Class<T> clazz, String sql, Object... args){**

**try {**

**return queryRunner.query(JDBCTools.getConnection(),sql,new BeanListHandler<T>(clazz),args);**

**} catch (SQLException e) {**

**throw new RuntimeException(e);**

**}**

**}**

**protected Object getValue(String sql,Object... args){**

**try {**

**return queryRunner.query(JDBCTools.getConnection(),sql,new ScalarHandler(),args);**

**} catch (SQLException e) {**

**throw new RuntimeException(e);**

**}**

**}**

**protected void batch(String sql,Object[][] args){**

**try {**

**queryRunner.batch(JDBCTools.getConnection(),sql,args);**

**} catch (SQLException e) {**

**throw new RuntimeException(e);**

**}**

**}**

**}**

# 创建CommonResult封装json对象。让前端更好渲染后端传来的key-valuer数据



/\*\*

\* @Author chenxin

\* @date 2023/10/16

\* @Version 1.0

\* 功能：设置异步请求响应结果的格式

\* 最终是将CommonResult的对象变为json字符串返给js

\*/

public class CommonResult {

private boolean flag; //确定本次响应的结果是否正常

private Object resultData; //响应结果的数据

private String message; //信息

/\*\*

\* 处理请求成功

\* @return

\*/

public static CommonResult ok(){

return new CommonResult().setFlag(true);

}

/\*\*

\* 处理请求失败

\* @return

\*/

public static CommonResult error(){

return new CommonResult().setFlag(false);

}

public boolean isFlag() {

return flag;

}

private CommonResult setFlag(boolean flag) {

this.flag = flag;

return this;

}

public Object getResultData() {

return resultData;

}

public CommonResult setResultData(Object resultData) {

this.resultData = resultData;

return this;

}

public String getMessage() {

return message;

}

public CommonResult setMessage(String message) {

this.message = message;

return this;

}

@Override

public String toString() {

return "CommonResult{" +

"flag=" + flag +

", resultData=" + resultData +

", message='" + message + '\'' +

'}';

}

}

# 使用sping boot框架简化配置

。各种XML、Annotation配置会让人眼花缭乱，而且配置多的话，如果出错了也很难找出原因。Spring Boot更多的是采用 Java Config 的方式，对 Spring 进行配置。举个例子：

我新建一个类，但是我不用 @Service注解，也就是说，它是个普通的类，只需要@Configuration 和@Bean两个注解即可，如下：

public class TestService {

public String sayHello () {

return "Hello Spring Boot!";

}

}

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

@Configuration

public class JavaConfig {

@Bean

public TestService getTestService() {

return new TestService();

}

}

@Configuration表示该类是个配置类，@Bean表示该方法返回一个 Bean。这样就把TestService作为 Bean 让 Spring 去管理了，在其他地方，我们如果需要使用该 Bean，和原来一样，直接使用@Resource注解注入进来即可使用，非常方便。

@Resource private TestService testService;

另外，部署配置方面，原来 Spring 有多个 xml 和 properties配置，在 Spring Boot 中只需要个 application.yml即可。