用户-注册-控制器

5.1 创建响应结果类

状态码、状态描述信息、数据、这部分功能封装在一个类中，将这个类作为方法的返回值，返回给前端浏览器。

创建 com.fx.store.util.JsonResult 响应结果类型

```
package com.fx.store.util;
import java.io.Serializable;
/**
 * Json 格式的数据进行响应
 * 泛型：当一个类中有泛型的数据类型，这个类声明时也要声明成泛型
 */
public class JsonResult<E> implements Serializable {
    //状态码
    private Integer state;
    //描述信息
    private String message;
    //数据：数据类型不确定，用泛型
    private E data;
    //定义构造方法，方便后期调用（根据参数不同进行调用）
    public JsonResult() {
    }
    public JsonResult(Integer state) {
        this.state = state;
    }
        /** 出现异常时调用 */
    public JsonResult(Throwable e) {
        // 获取异常对象中的异常信息
        this.message = e.getMessage() ;
    }
    public JsonResult(Integer state, E data) {
        this.state = state;
        this.data = data;
    }
    public Integer getState() {
        return state;
    }
    public void setState(Integer state) {
        this.state = state;
    }
    public String getMessage() {
        return message;
    }
```

```
    public void setMessage(String message) {
        this.message = message;
    }
    public E getData() {
        return data;
    }
    public void setData(E data) {
        this.data = data;
    }
}
```

5.2 设计请求

设计用户提交的请求，并设计响应的方式

请求路径：/users/reg

请求参数：User user

请求类型：POST（有敏感数据就用 POST,没有就用 GET）

响应结果：JsonResult<Void>（没有数据，泛型为 void 类型）

5.3 处理请求

1.创建 com.fx.store.controller.UserController 控制器类，此类依赖业务层 IUserService 接口，在类的声明之前添加@RestController 和@RequestMapping("users")注解，在类中添加 IUserService 业务对象并使用@Autowired 注解修饰。2.然后在类中添加处理请求的用户注册方法。

```
package com.fx.store.controller;

import com.fx.store.entity.User;
import com.fx.store.service.IUserService;
import com.fx.store.service.ex.InsertException;
import com.fx.store.service.ex.UsernameDuplicateException;
import com.fx.store.util.JsonResult;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

//@Controller//用于指示 Spring 类的实例是一个控制器。表示当前类交给 spring 管理
@RestController //@Controller+@ResponseBody
@RequestMapping("users")//什么样的请求会被拦截到此类中
public class UserController {
    @Autowired
    private IUserService iUserService;
    @RequestMapping("reg")
    //@ResponseBody//表示此方法的响应结果以 json 格式进行数据的响应到前端
    public JsonResult<Void> reg(User user){
```

```
        //创建响应结果对象
        JsonResult<Void> result = new JsonResult<Void>();
        try {
            //调用业务对象执行注册
            iUserService.reg(user);
            //响应成功
            result.setState(200);
            result.setMessage("用户名注册成功");
        } catch (UsernameDuplicateException e) {
            //用户名被占用
            result.setState(4000);
            result.setMessage("用户名已经被占用");
        }catch (InsertException e){
            result.setState(5000);
            result.setMessage("注册失败，请联系系统管理员");
        }
        return result;
    }
}
```

3.完成后启动项目（运行 springboot 的启动类），打开浏览器访问 http://localhost:8080/users/reg?username=lucy01&password=123456 请求进行测试。

```
{"state":200,"message":"用户名注册成功","data":null}
```

5.4 控制器层的调整

1.然后创建提供控制器类的基类 com.fx.store.controller.BaseController，在其中定义表示响应成功的状态码及统一处理异常的方法。

@ExceptionHandler 注解用于统一处理方法抛出的异常。当我们使用这个注解时，需要定义一个异常的处理方法，再给这个方法加上@ExceptionHandler 注解，这个方法就会处理类中其他方法（被@RequestMapping 注解）抛出的异常。@ExceptionHandler 注解中可以添加参数，参数是某个异常类的 class，代表这个方法专门处理该类异常。

```
package com.fx.store.controller;

import com.fx.store.service.ex.InsertException;
import com.fx.store.service.ex.ServiceException;
import com.fx.store.service.ex.UsernameDuplicateException;
import com.fx.store.util.JsonResult;
import org.springframework.web.bind.annotation.ExceptionHandler;

/**
 * 控制类的基类
 */
```

```
public class BaseController {
    /**
     * 操作成功的状态码
     */
    public static final int ok = 200;
    /**
     * handleException：请求处理方法，这个方法的返回值就是需要传递给前端的数据
     * @ExceptionHandler：
     * 自动将异常对象传递给此方法的参数列表上
     * 当前项目中产生了异常，被统一拦截到此方法中，这个方法此时就充当的是请求处理方法，方法的返回值直接给前端
     */
    @ExceptionHandler(ServiceException.class)//用于统一处理抛出的异常
    public JsonResult<Void> handleException(Throwable e){
        JsonResult<Void> result = new JsonResult<Void>(e);
        if (e instanceof UsernameDuplicateException)
        {
            //用户名被占用
            result.setState(4000);
            result.setMessage("用户名已经被占用");
        }else if (e instanceof InsertException){
            result.setState(5000);
            result.setMessage("注册失败，请联系系统管理员");
        }
        return result;
    }
}
```

2.最后简化 UserController 控制器类中的用户注册 reg()方法的代码。

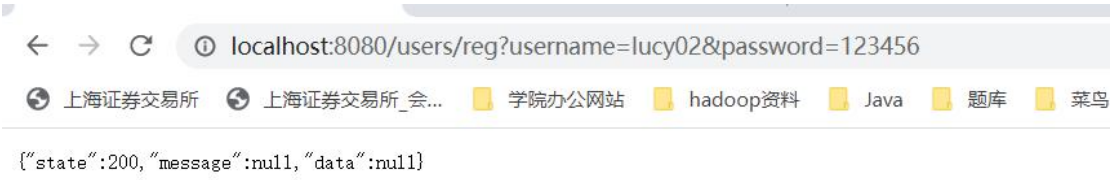
```
package com.fx.store.controller;

import com.fx.store.entity.User;
import com.fx.store.service.IUserService;
import com.fx.store.service.ex.InsertException;
import com.fx.store.service.ex.UsernameDuplicateException;
import com.fx.store.util.JsonResult;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.client.OkHttp3ClientHttpRequestFactory;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

//@Controller//用于指示 Spring 类的实例是一个控制器。表示当前类交给 spring 管理
@RestController //@Controller+@ResponseBody
@RequestMapping("users")//什么样的请求会被拦截到此类中
```

```
public class UserController extends BaseController {
    @Autowired
    private IUserService iUserService;
    @RequestMapping("reg")
    //@ResponseBody//表示此方法的响应结果以 json 格式进行数据的响应到前端
    public JsonResult<Void> reg(User user) {
        //调用业务对象执行注册
        iUserService.reg(user);
        return new JsonResult<Void>(ok);
    }
}
```

3.完成后启动项目，打开浏览器访问 http://localhost:8080/users/reg?username=lucy02&password=123456 请求进行测试。



用户-注册-前端页面

1. AJAX = 异步 JavaScript 和 XML。

AJAX 是一种用于创建快速动态网页的技术。

通过在后台与服务器进行少量数据交换，AJAX 可以使网页实现异步更新。这意味着可以在不重新加载整个网页的情况下，对网页的某部分进行更新。

传统的网页（不使用 AJAX）如果需要更新内容，必需重载整个网页面。

2.jQuery 封装了一个函数，称之为$. ajax()函数，通过对象调用 ajax()函数，可以异步加载相关的请求。依靠的是 javaScript 提供的一个对象 XHR（xmlHttpResponse），封装了这个对象。

3.ajax()使用方式。需要传递一个方法体作为方法的参数来使用，一对大括号称之为方法体。Ajax 接收多个参数，参数与参数之间要求使用“,”进行分割，每一组参数之间使用“:”进行分割，参数的组成部分是一个参数的名称（不能随意定义），是参数的值，参数的值要求是用字符串来标识。参数的声明顺序没有要求，语法结构：

```
$.ajax({
    url:"",
    type: "",
    data: "",
    dataType:"",
    success:function(){}
    error:function(){}
});
```

参数	功能描述

url	标识请求的地址（url），不能包含参数列表部分的内容。例如：url: localhost:8080/users/reg
type	请求类型（GET 和 POST 请求的类型）。例如：type: ”POST”
data	向指定请求 url 地址提交的数据。例如：data: ” username=lucy02&password=123456”
dataType	提交的数据类型，数据类型一般指定为 json 类型。dataType:”json”
success	当服务器正常响应客户端时，会自动调用 success 参数的方法，并且将服务器返回的数据以参数的形式传递给这个方法的参数上
error	当服务器未正常响应客户端时，会自动调用 error 参数的方法，并且将服务器返回的数据以参数的形式传递给这个方法的参数上

4.js 代码可以独立声明在一个 js 的文件里或者声明在一个 script 标签中（该标签可以在页面的任意位置：head、body）。

5.在 register.html 页面中 body 标签内部的最后，添加 script 标签用于编写 JavaScript 程序实现发送请求的方法，点击事件来完成，先选中对应的按钮（$(“选择器”)），再去添加点击的事件，$.ajax()函数发送请求。

serialize()方法通过序列化表单值，创建 URL 编码文本字符串。

```
<script type="text/javascript">
    //1.监听注册按钮是否被点击,如果被点击则可以执行一个方法
    $("#btn-reg").click(function () {
        //在检测到用户注册时，先输出看看序列化后的结果是否为下面的拼接结果
        console.log($("#form-reg").serialize())
        //2.是，则发送 ajax()的异步请求来完成用户注册功能
        $.ajax({
            url:"/users/reg",
            type:"POST",
            //数据在表单里，获取整个表单
            //调用 serialize()序列化返回的值自动拼接：username=lucy02&password=123456
            data:$("#form-reg").serialize(),
            dataType:"json",
            success:function (json) {
                if (json.state == 200){
                    alert("注册成功");
                }else {
                    alert("注册失败");
                }
            },
            error:function (xhr) {
                alert("注册时产生未知的错误"+xhr);
            }
        });
    });

</script>
```

启动项目后，打开注册界面：http://localhost:8080/web/register.html

输入注册数据，点击注册,出现以下：