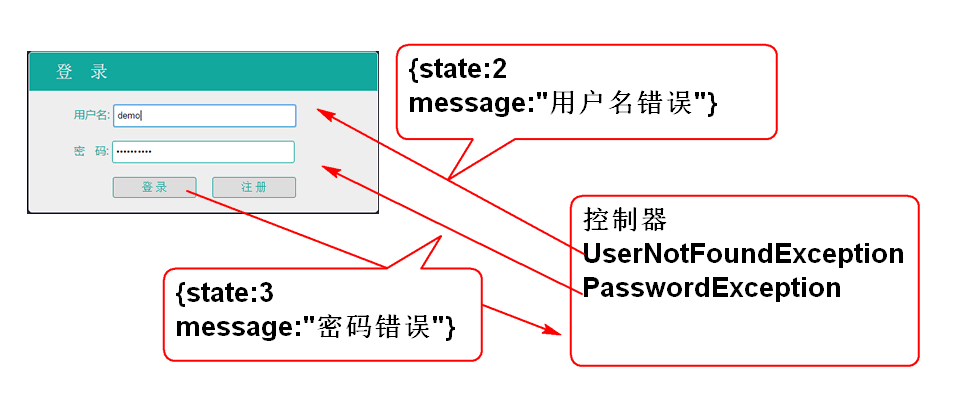
* **登录(续)**

**显示错误消息**

原理:



1. 重构控制器增加异常处理方法 UserController

@ExceptionHandler( UserNotFoundException.class)

@ResponseBody public JsonResult handleUserNotFound( UserNotFoundExceptione){ e.printStackTrace(); return newJsonResult(2,e); }

@ExceptionHandler(PasswordException.class)

@ResponseBody public JsonResult handlePassword( PasswordException e){ e.printStackTrace(); return new JsonResult(3,e); }

1. 重构JsonResult 添加 构造器

public JsonResult(int state, Throwable e) { this.state = state; this.message = e.getMessage(); }

1. 重构 login.js 的loginAction方法, 显示错误信息

var msg = result.message;

if(result.state==2){ $('#count').next().html(msg);

}else if(result.state==3){

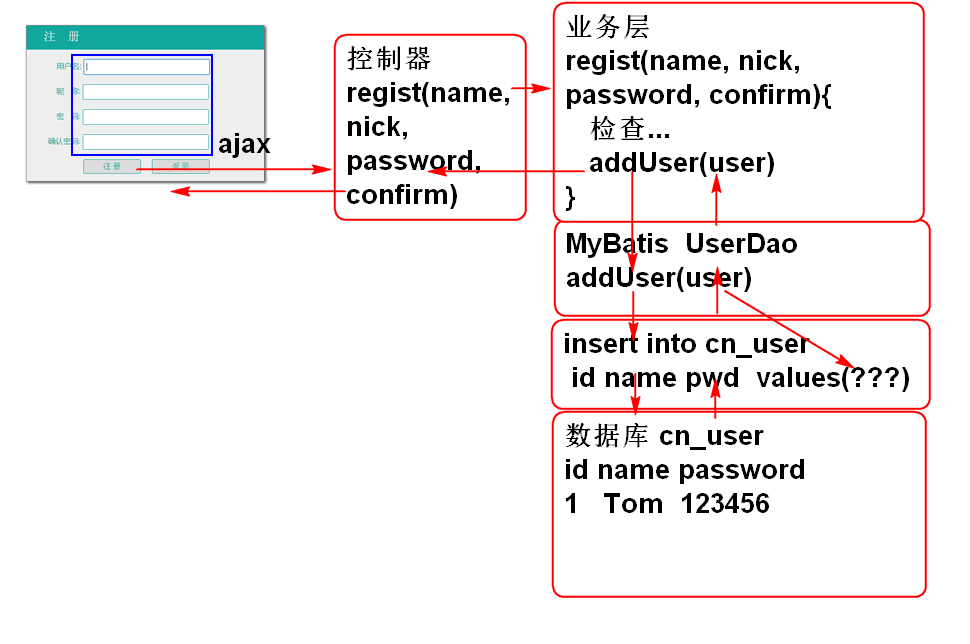
$('#password').next().html(msg);

}else{ alert(msg); }

1. 测试

**注册功能**

原理:



**1. 持久层**

1. 声明持久层方法: UserDao

int addUser(User user);

1. 声明SQL UserMappeer.xml

<insert id="addUser" parameterType="cn.tedu.note.entity.User">

insert into cn\_user ( cn\_user\_id, cn\_user\_name, cn\_user\_password, cn\_user\_token, cn\_user\_nick ) values ( #{id}, #{name}, #{password}, #{token}, #{nick} )

</insert>

1. 测试 UserDaoTest:

UserDao dao;

@Before public void initDao(){

dao = ctx.getBean( "userDao", UserDao.class); }

@Test

public void testAddUser(){

String id=UUID.randomUUID().toString();

String name = "Tom"; String salt = "今天你吃了吗?";

String password = DigestUtils.md5Hex(salt+"123456");

String token = ""; String nick = "";

User user = new User( id, name, password, token, nick);

int n = dao.addUser(user);

System.out.println(n); }

**2. 业务层**

1. 声明业务层方法 UserService

/\*\* \* UserService 中添加注册功能

\* @param name \* @param nick \* @param password \* @param confirm \* @return 注册成功的用户信息 \* @throws UserNameException 用户名异常

\* @throws PasswordException 密码异常

\*/ User regist(String name, String nick, String password, String confirm) throws UserNameException, PasswordException;

1. 声明业务层异常 UserNameException

public class UserNameException extends RuntimeException { private static final long serialVersionUID = 6435296194529486206L; public UserNameException() { } public UserNameException(String message) { super(message); } public UserNameException(Throwable cause) { super(cause); } public UserNameException(String message, Throwable cause) { super(message, cause); } public UserNameException(String message, Throwable cause, boolean enableSuppression, boolean writableStackTrace) { super(message, cause, enableSuppression, writableStackTrace); } }

1. 重构 jdbc.properties 和 UserServiceImpl, 将salt存储到配置文件, 利用Spring注入到属性中:

# jdbc.properties salt=\u4ECA\u5929\u4F60\u5403\u4E86\u5417?

// UserServiceImpl @Value("#{jdbc.salt}") private String salt;

1. 实现业务层方法: UserServiceImpl

public User regist(String name, String nick, String password, String confirm) throws UserNameException, PasswordException {

//检查name, 不能重复

if(name==null || name.trim().isEmpty()){ throw new UserNameException("不能空"); }

User one = userDao.findUserByName(name);

if(one!=null){ throw new UserNameException("已注册"); }

//检查密码

if(password==null || password.trim().isEmpty()){ throw new PasswordException("不能空"); }

if(! password.equals(confirm)){ throw new PasswordException("确认密码不一致"); }

//检查nick if(nick ==null || nick.trim().isEmpty()){ nick = name; }

String id = UUID.randomUUID().toString(); String token = "";

password = DigestUtils.md5Hex(salt+password);

User user = new User( id, name, password, token, nick);

int n = userDao.addUser(user);

if(n!=1){ throw new RuntimeException("添加失败!"); } return user; }

1. 测试:UserServiceTest

UserService service;

@Before

public void initService(){ service = ctx.getBean("userService", UserService.class); }

@Test

public void testRegist(){ User user = service.regist( "Andy", "Andy", "123456", "123456"); System.out.println(user); }

**3. 控制器**

1. 添加控制器方法 UserController

@RequestMapping("/regist.do")

@ResponseBody

public JsonResult regist(String name, String nick, String password, String confirm){

User user = userService.regist( name, nick, password, confirm);

return new JsonResult(user); }

1. 测试

http://localhost:8080/note/user/regist.do?name=Jerry&nick=AN&password=12345&confirm=12345

**4. 添加注册JS脚本**

1. 更新 log\_in.html 取消页面检查js脚本:

window.onload=function(){

var t =setTimeout("get('zc').style.visibility='visible'",800); //get('final\_password').onblur=function(){

// var npassword=get('regist\_password').value;

// var fpassword=get('final\_password').value;

// if(npassword!=fpassword){

// get('warning\_3').style.display='block'; // } //} //get('regist\_password').onblur=function(){

// var npassword=get('regist\_password').value.length;

// if(npassword<6&&npassword>0){

// get('warning\_2').style.display='block'; // } //} //get('regist\_password').onfocus=function(){ // get('warning\_2').style.display='none'; //} //get('final\_password').onfocus=function(){ // get('warning\_3').style.display='none'; //} }

1. 添加注册对话框事件脚本 login.js

$('#regist\_button').click(registAction);

$('#regist\_username').blur(checkRegistName); $('#regist\_password').blur(checkRegistPassword);

$('#final\_password').blur(checkConfirm);

1. 添加注册对话框数据检验方法:login.js

function checkConfirm(){ var pwd2 = $('#final\_password').val();

var pwd = $('#regist\_password').val();

//pwd 如果是空值表示 false, 非空则是true

if(pwd && pwd==pwd2){

$('#final\_password').next().hide(); return true; } $('#final\_password').next().show() .find('span').html('确认密码不一致'); return false; } function checkRegistPassword(){

var pwd = $('#regist\_password').val().trim();

var rule = /^\w{4,10}$/;

if(rule.test(pwd)){ $('#regist\_password').next().hide(); return true; } $('#regist\_password').next().show() .find('span').html('4~10个字符'); return false; } function checkRegistName(){

var name = $('#regist\_username').val().trim();

var rule = /^\w{4,10}$/;

if(rule.test(name)){ $('#regist\_username').next().hide(); return true; } $('#regist\_username').next().show() .find('span').html('4~10字符'); return false; }

1. 添加注册对话框注册按钮事件方法 login.js

function registAction(){ console.log('registAction');

//检验界面参数 var n = checkRegistName() + checkRegistPassword() + checkConfirm(); if(n!=3){ return ; }

//获取界面中表单数据

var name = $('#regist\_username').val().trim();

var nick = $('#nickname').val(); var password = $('#regist\_password').val();

var confirm = $('#final\_password').val();

//发起AJAX请求

var url = 'user/regist.do';

var data = {name:name, nick:nick, password:password, confirm:confirm};

//console.log(data);

// $.post 是 $.ajax的简化版

$.post(url, data, function(result){ console.log(result);

if(result.state==0){ //退回登录界面 $('#back').click();

var name = result.data.name;

$('#count').val(name); $('#password').focus();

//清空表单 $('#regist\_username').val('');

$('#nickname').val(''); $('#regist\_password').val('');

$('#final\_password').val('');

}elseif(result.state==4){ $('#regist\_username').next().show() .find('span').html(result.message); }else if(result.state==3){ $('#regist\_password').next().show() .find('span').html(result.message); }else{ alert(result.message); } }); //得到响应以后, 更新界面 }

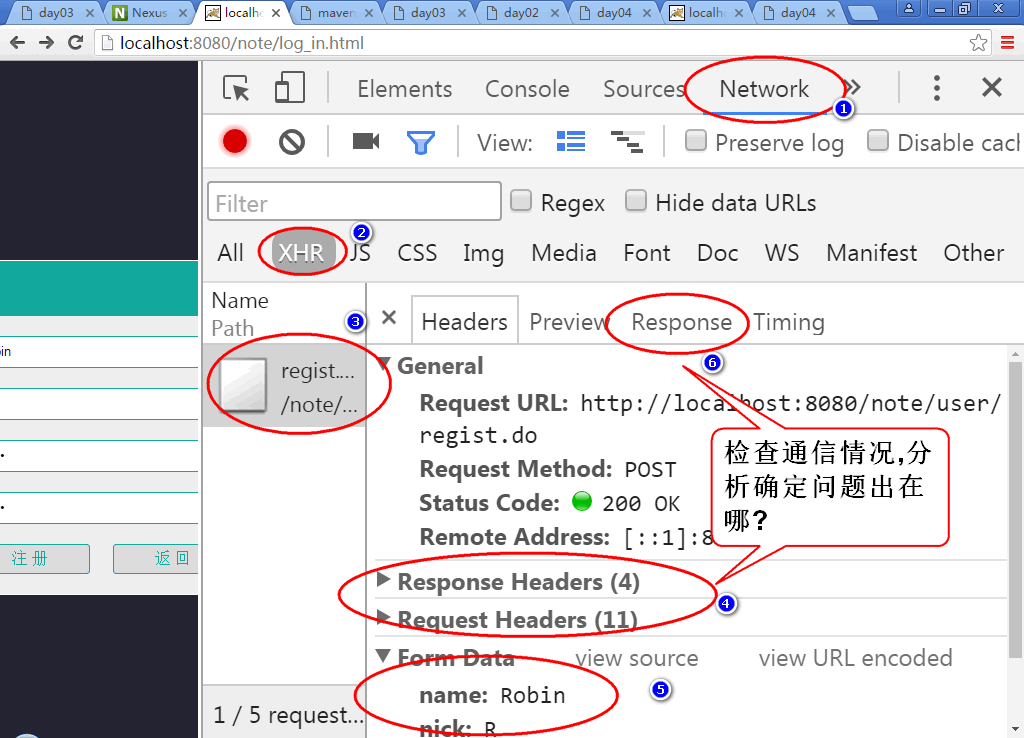
1. 重构控制器UserController 增加事件处理方法:

@ExceptionHandler(UserNameException.class)

@ResponseBody public JsonResult handleUserName( UserNameException e){ e.printStackTrace(); return new JsonResult(4,e); }

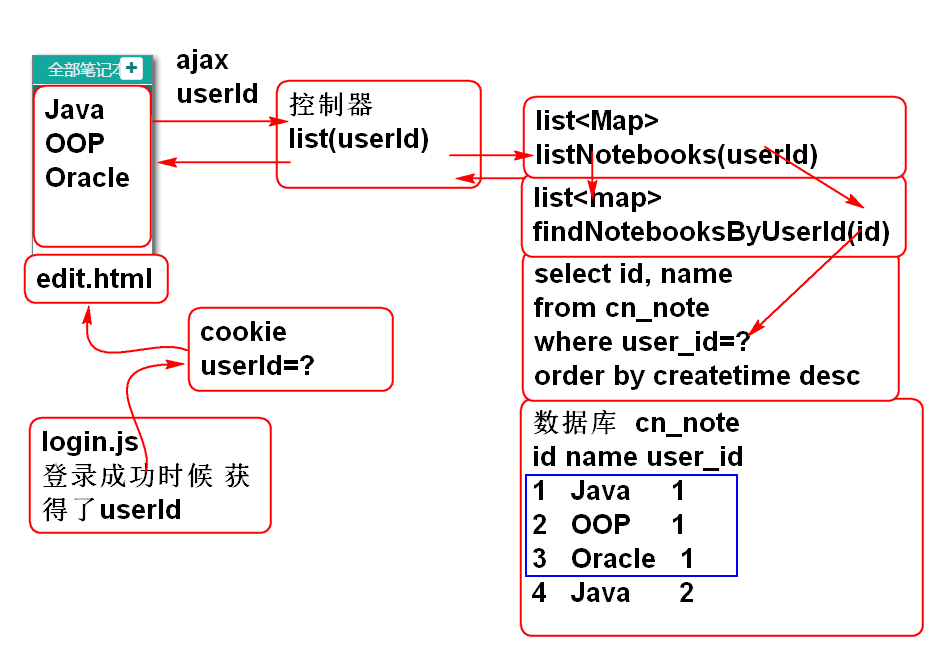
1. 测试...

调试工具:



* **笔记本列表功能**

原理:



**1. 持久层**

1. 添加持久层接口 NotebookDao:

public interface NotebookDao { List<Map<String, Object>> findNotebooksByUserId( String userId); }

1. 添加Mapper文件: NotebookMapper.xml

<mapper namespace="cn.tedu.note.dao.NotebookDao">

<select id="findNotebooksByUserId" parameterType="string" resultType="map">

select cn\_notebook\_id as id, cn\_notebook\_name as name from cn\_notebook where cn\_user\_id = #{userId} order by cn\_notebook\_createtime desc

</select> </mapper>

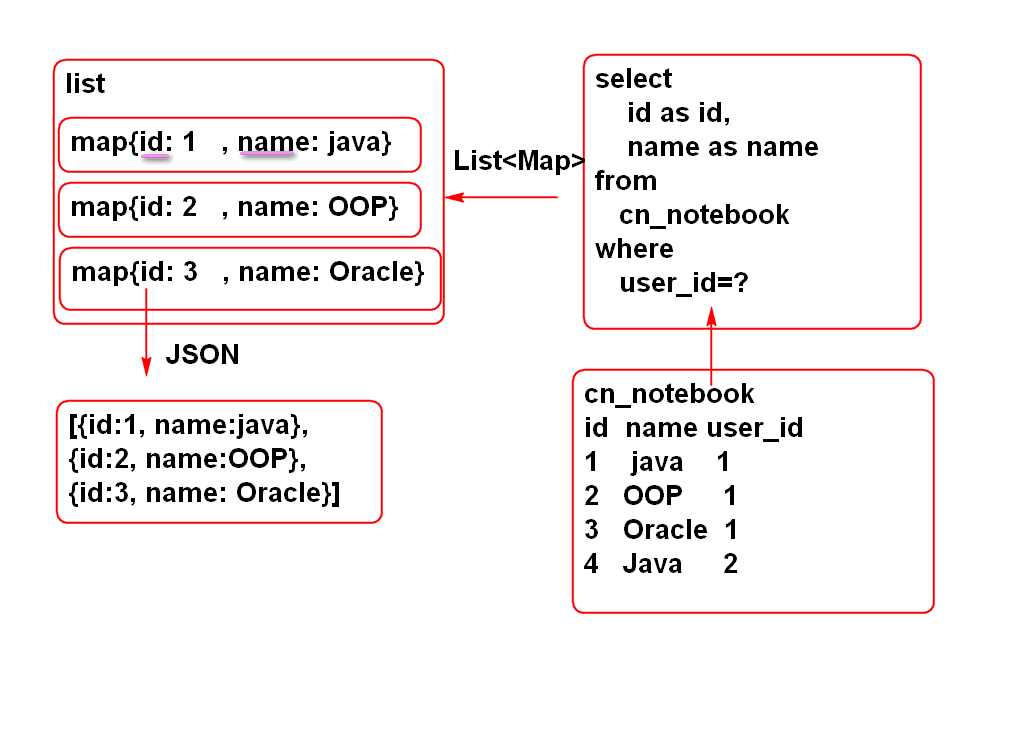
1. 测试:

public class NotebookDaoTest extends BaseTest{ NotebookDao dao; @Before public void initDao(){ dao = ctx.getBean("notebookDao", NotebookDao.class); } @Test //select cn\_user\_id from cn\_notebook; public void testFindNotebooksByUserId(){ String userId="52f9b276-38ee-447f-a3aa-0d54e7a736e4"; List<Map<String, Object>> list= dao.findNotebooksByUserId(userId); for (Map<String, Object> map : list) { System.out.println(map); } } }

提示: 需要先到数据库中获得拥有笔记的用户ID userId

select cn\_user\_id from cn\_notebook;

List 封装查询结果:



**2. 业务层**

1. 声明业务接口NotebookService

public interface NotebookService { List<Map<String, Object>> listNotebooks(String userId) throws UserNotFoundException; }

1. 实现业务接口 NotebookServiceImpl

@Service("notebookService") public class NotebookServiceImpl implements NotebookService { @Resource private NotebookDao notebookDao; @Resource private UserDao userDao; public List<Map<String, Object>> listNotebooks(String userId) throws UserNotFoundException { if(userId==null || userId.trim().isEmpty()){ throw new UserNotFoundException("ID不能空"); } User user = userDao.findUserById(userId); if(user==null){ throw new UserNotFoundException("用户不存在"); } return notebookDao .findNotebooksByUserId(userId); } }

1. 添加查询方法 UserDao

User findUserById(String userId);

1. 添加SQL UserMapper

<select id="findUserById" parameterType="string" resultType="cn.tedu.note.entity.User"> select cn\_user\_id as id, cn\_user\_name as name, cn\_user\_password as password, cn\_user\_token as token, cn\_user\_nick as nick from cn\_user where cn\_user\_id = #{userId} </select>

1. 测试:

public class NotebookServiceTest extends BaseTest { NotebookService service; @Before public void initService(){ service = ctx.getBean("notebookService", NotebookService.class); } @Test public void testListNotebooks(){ String userId="52f9b276-38ee-447f-a3aa-0d54e7a736e4"; List<Map<String, Object>> list= service.listNotebooks(userId); for (Map<String, Object> map : list) { System.out.println(map); } } }

* **笔记本列表**

**3. 控制器**

1. 添加控制器父类AbstractController, 封装公共的异常处理方法:

public abstract class AbstractController { /\*\* \* 在其他控制器方法执行出现异常时候, 执行 \* 异常处理方法 handleException \*/ @ExceptionHandler(Exception.class) @ResponseBody public Object handleException( Exception e){ e.printStackTrace(); return new JsonResult(e); } }

重构 UserController, 使其继承于 AbstractController

略...

1. 创建控制器类 NotebookController:

@RequestMapping("/notebook") @Controller public class NotebookController extends AbstractController { @Resource private NotebookService notebookService; @RequestMapping("/list.do") @ResponseBody public JsonResult list(String userId){ List<Map<String, Object>> list= notebookService.listNotebooks(userId); return new JsonResult(list); } }

1. 测试

http://localhost:8080/note/notebook/list.do?userId=52f9b276-38ee-447f-a3aa-0d54e7a736e4 http://localhost:8080/note/notebook/list.do?userId=abc http://localhost:8080/note/notebook/list.do

测试结果体现了控制器能够利用父类的异常处理方法处理异常情况.

**4. 表现层脚本编程**

1. 更新log\_in.html 引入cookie操作API:

<script type="text/javascript" src="scripts/cookie\_util.js"></script>

1. 更新 login.js 的loginAction方法, 在登录成功时候将userId保存到cookie中:

... //登录成功以后将userId保存到cookie中 addCookie("userId", user.id); ...

1. 在edit.html添加JS脚本:

<script type="text/javascript" src="scripts/cookie\_util.js"></script> <script type="text/javascript" src="scripts/note.js"></script>

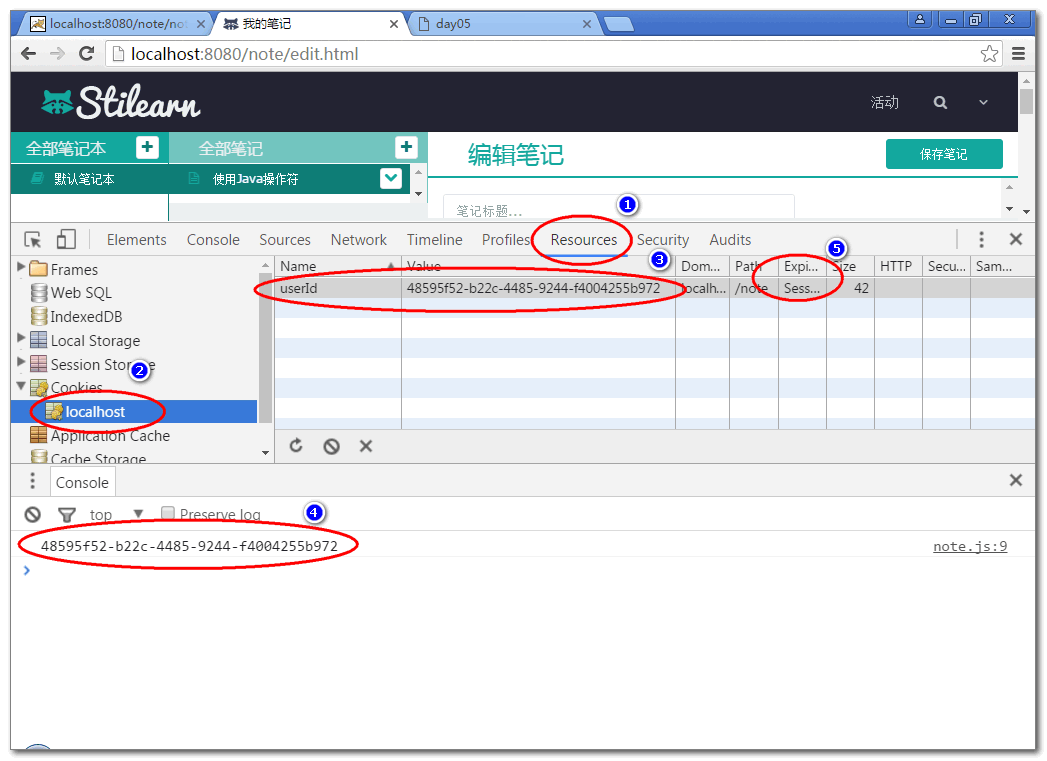
为了提高页面的用户体验, 减少页面加载延迟, 将脚本在页面后部加载.

1. 添加scripts/note.js 文件:

var SUCCESS = 0; var ERROR = 1; $(function(){ var userId = getCookie('userId'); console.log(userId); });

scripts/note.js 编码一定是 utf-8

1. 测试: 登录以后可以再edit.html 的控制台上能够看到 userId, 可以通过浏览器的控制台检查cookie保存情况:



1. 在ready函数中调用方法 loadNotebooks() 在页面加载后立即加载笔记本列表数据:

//网页加载以后, 立即读取笔记本列表 loadNotebooks();

1. 添加方法loadNotebooks, 利用ajax加载笔记本列表数据:

/\*\* 加载笔记本列表数据 \*/ function loadNotebooks(){ //利用ajax从服务器获取(get)数据, 使用getJSON方法 var url = 'notebook/list.do'; var data = {userId:getCookie('userId'), name:'demo'}; $.getJSON(url, data, function(result){ console.log(result); if(result.state==SUCCESS){ var notebooks = result.data; //在showNotebooks方法中将全部的 //笔记本数据 notebooks 显示到 // notebook-list 区域 showNotebooks(notebooks); }else{ alert(result.message); } }); }

提示: 为了保存代码的简洁, 将显示笔记本列表的算法封装到showNotebooks方法中.

1. 声明showNotebooks方法, 显示在notebook-list区域的ul中显示笔记本列表信息:

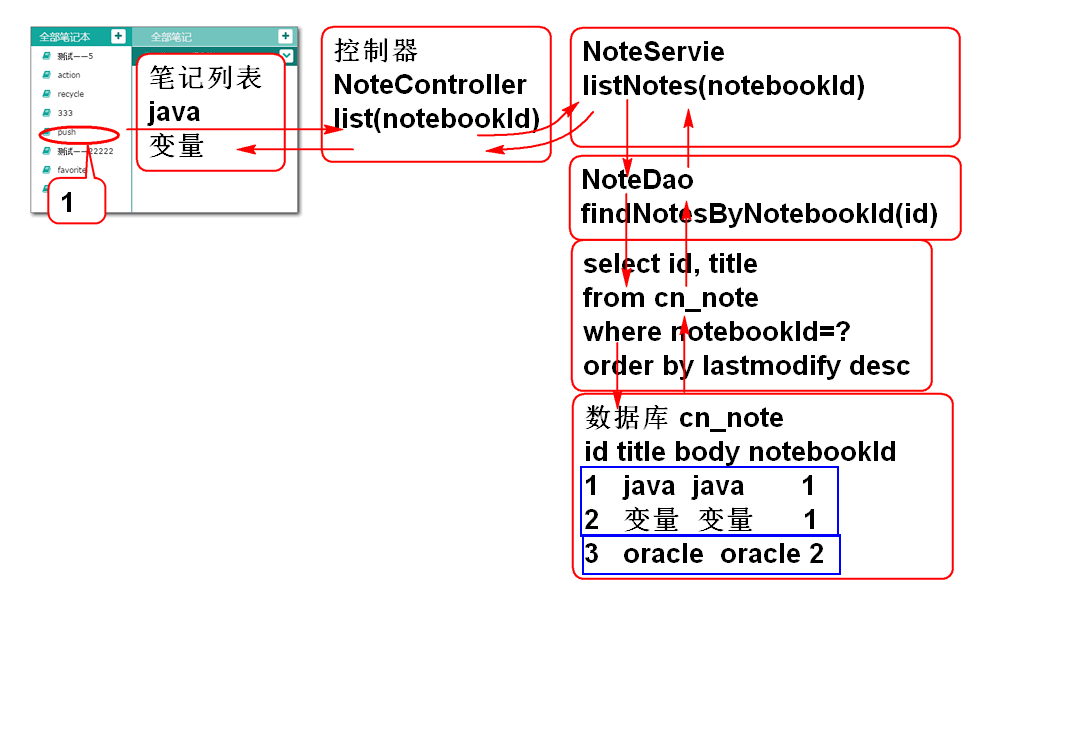
/\*\* 在notebook-list区域中显示笔记本列表 \*/ function showNotebooks(notebooks){ //算法: //找显示笔记本列表的区域的ul元素 //遍历notebooks数组, 将为每个对象创建一个li //元素, 添加到 ul元素中. var ul = $('#notebook-list ul'); ul.empty();//清除ul中原有的内容 for(var i=0; i<notebooks.length; i++){ var notebook = notebooks[i]; var li = notebookTemplate.replace( '[name]', notebook.name); li = $(li); ul.append(li); } } var notebookTemplate = '<li class="online">'+ '<a><i class="fa fa-book" title="online" '+ 'rel="tooltip-bottom"></i> [name]</a>'+ '</li>';

提示: notebookTemplate 变量定义了显示笔记本的li元素模板. 来自edit.html页面中.

1. 测试...

* **笔记列表功能**

原理:



**1. 持久层**

1. 声明持久层查询方法 NoteDao

public interface NoteDao { List<Map<String,Object>> findNotesByNotebookId( String notebookId); }

1. 声明SQL NoteMapper.xml

<mapper namespace="cn.tedu.note.dao.NoteDao"> <select id="findNotesByNotebookId" parameterType="string" resultType="map"> select cn\_note\_id as id, cn\_note\_title as title from cn\_note where cn\_notebook\_id = #{notebookId} order by cn\_note\_last\_modify\_time desc </select> </mapper>

1. 测试:

public class NoteDaoTest extends BaseTest{ NoteDao dao; @Before public void initDao(){ dao = ctx.getBean("noteDao", NoteDao.class); } @Test //select cn\_notebook\_id from cn\_note; public void testFindNotesByNotebookId(){ String id="fa8d3d9d-2de5-4cfe-845f-951041bcc461"; List<Map<String, Object>> list= dao.findNotesByNotebookId(id); for (Map<String, Object> map : list) { System.out.println(map); } } }

**2. 业务层**

1. 声明业务层方法 NoteService

public interface NoteService { List<Map<String, Object>> listNotes( String notebookId) throws NotebookNoteFoundExcepotion; }

1. 声明业务异常:NotebookNoteFoundExcepotion

略...

1. 实现业务层方法 NoteServiceImpl

@Service("noteService") public class NoteServiceImpl implements NoteService{ @Resource private NoteDao noteDao; @Resource private NotebookDao notebookDao; public List<Map<String, Object>> listNotes( String notebookId) throws NotebookNoteFoundExcepotion { if(notebookId==null || notebookId.trim().isEmpty()){ throw new NotebookNoteFoundExcepotion("ID为空"); } //Notebook notebook = notebookDao // .findNotebookById(notebookId); //if(notebook==null){ // throw new NotebookNoteFoundExcepotion("没有笔记本"); //} int n = notebookDao.countNotebookById( notebookId); if(n!=1){ throw new NotebookNoteFoundExcepotion("没有笔记本"); } return noteDao.findNotesByNotebookId(notebookId); } }

1. 重构NotebookDao, 添加统计方法 countNotebookById

int countNotebookById(String notebookId);

1. 添加sql NotebookMapper.xml

<select id="countNotebookById" parameterType="string" resultType="int"> select count(\*) as cnt from cn\_notebook where cn\_notebook\_id = #{notebookId} </select>

1. 测试 NoteServiceTest:

public class NoteServiceTest extends BaseTest{ NoteService service; @Before public void initService(){ service = ctx.getBean("noteService", NoteService.class); } @Test public void testListNotes(){ String id="fa8d3d9d-2de5-4cfe-845f-951041bcc461"; List<Map<String, Object>> list= service.listNotes(id); for (Map<String, Object> map : list) { System.out.println(map); } } }

**3. 控制器**

1. 添加控制器 NoteController

@Controller @RequestMapping("/note") public class NoteController extends AbstractController{ @Resource private NoteService noteService; @RequestMapping("/list.do") @ResponseBody public JsonResult list(String notebookId){ List<Map<String, Object>> list= noteService.listNotes(notebookId); return new JsonResult(list); } }

1. 测试:

http://localhost:8080/note/note/list.do?notebookId=d0b0727f-a233-4a1f-8600-f49fc1f25bc9

**4. 表现层脚本**

1. 重构edit.html, 设置笔记本列表ID notebook-list

<div class="col-xs-2" style='padding:0;' id='notebook-list'>

1. 在note.js 的ready方法中绑定 笔记本列表点击事件:

//绑定笔记本列表区域的点击事件 $('#notebook-list').on( 'click','.notebook', loadNotes);

提示: 这里利用事件冒泡, 在父元素上绑定点击事件, Jquery提供的on方法可以非常方便区别发生事件的事件源.

重构: 重构notebookTemplate, 为li元素添加 notebook 类:

var notebookTemplate = '<li class="online notebook">'+ '<a><i class="fa fa-book" title="online" '+ 'rel="tooltip-bottom"></i> [name]</a>'+ '</li>';

1. 添加事件处理方法 loadNotes:

/\*\* 笔记本项目点击事件处理方法, 加载全部笔记 \*/ function loadNotes(){ var li = $(this);//当前被点击的对象li //在被点击的笔记本li增加选定效果 li.parent().find('a').removeClass('checked'); li.find('a').addClass('checked'); var url = 'note/list.do'; //li.data('notebookId') 方法可以获取绑定到li //元素上的数据notebookId, 这个notebookId在 //showNotebooks方法中绑定li元素上的! var data={notebookId:li.data('notebookId')}; console.log(data); $.getJSON(url, data, function(result){ if(result.state==SUCCESS){ var notes = result.data; showNotes(notes); }else{ alert(result.message); } }); }

提示: 笔记本点击时候的选择效果是后续重构过程中添加的.

重构: 重构showNotebooks方法, 将notebookId绑定li元素上

... for(var i=0; i<notebooks.length; i++){ var notebook = notebooks[i]; var li = notebookTemplate.replace( '[name]', notebook.name); li = $(li); //将 notebook.id 绑定到 li li.data('notebookId', notebook.id); ul.append(li); } ...

1. 添加 showNotes 方法显示笔记到笔记列表区域:

/\*\* 将笔记列表信息显示到屏幕上 \*/ function showNotes(notes){ console.log(notes); //将每个笔记对象显示到屏幕的ul区域 var ul = $('#note-list ul'); ul.empty(); for(var i=0; i<notes.length; i++){ var note = notes[i]; var li = noteTemplate.replace( '[title]', note.title); li = $(li); ul.append(li); } } var noteTemplate = '<li class="online">'+ '<a>'+ '<i class="fa fa-file-text-o" title="online" rel="tooltip-bottom"></i> [title]<button type="button" class="btn btn-default btn-xs btn\_position btn\_slide\_down"><i class="fa fa-chevron-down"></i></button>'+ '</a>'+ '<div class="note\_menu" tabindex="-1">'+ '<dl>'+ '<dt><button type="button" class="btn btn-default btn-xs btn\_move" title="移动至..."><i class="fa fa-random"></i></button></dt>'+ '<dt><button type="button" class="btn btn-default btn-xs btn\_share" title="分享"><i class="fa fa-sitemap"></i></button></dt>'+ '<dt><button type="button" class="btn btn-default btn-xs btn\_delete" title="删除"><i class="fa fa-times"></i></button></dt>'+ '</dl>'+ '</div>'+ '</li>';

重构: edit.html, 设置笔记列表ID note-list

<div class="col-xs-3" style='padding:0;' id='note-list'>

1. 测试...