**云笔记**

**弹出笔记子菜单**

**1. 利用事件冒泡在 note-list 上绑定事件弹出笔记子菜单:**

1. 重构笔记列表模板, 为笔记子菜单触发按钮添加类 btn-note-menu

var noteTemplate = '<li class="online note">'+ '<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 btn-note-menu"><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>';

1. 在ready方法中绑定 触发事件

//绑定笔记子菜单的触发事件 $('#note-list').on('click', '.btn-note-menu', showNoteMenu);

1. 添加事件处理方法

/\*\* 显示笔记子菜单处理方法 \*/ function showNoteMenu(){ //找到菜单对象, 调用show() 方法 var btn = $(this); //如果当前是被选定的 笔记项目 就弹出子菜单 btn.parent('.checked').next().toggle(); //btn.parent('.checked') 获取当前按钮的父元素 //这个元素必须符合选择器'.checked', 如果不 //符合就返回空的JQuery元素. return false;//阻止点击事件的继续传播!避免传播到document对象时候, 触发关闭菜单事件 }

1. ready方法中绑定document的点击事件, 利用事件冒泡关闭子菜单

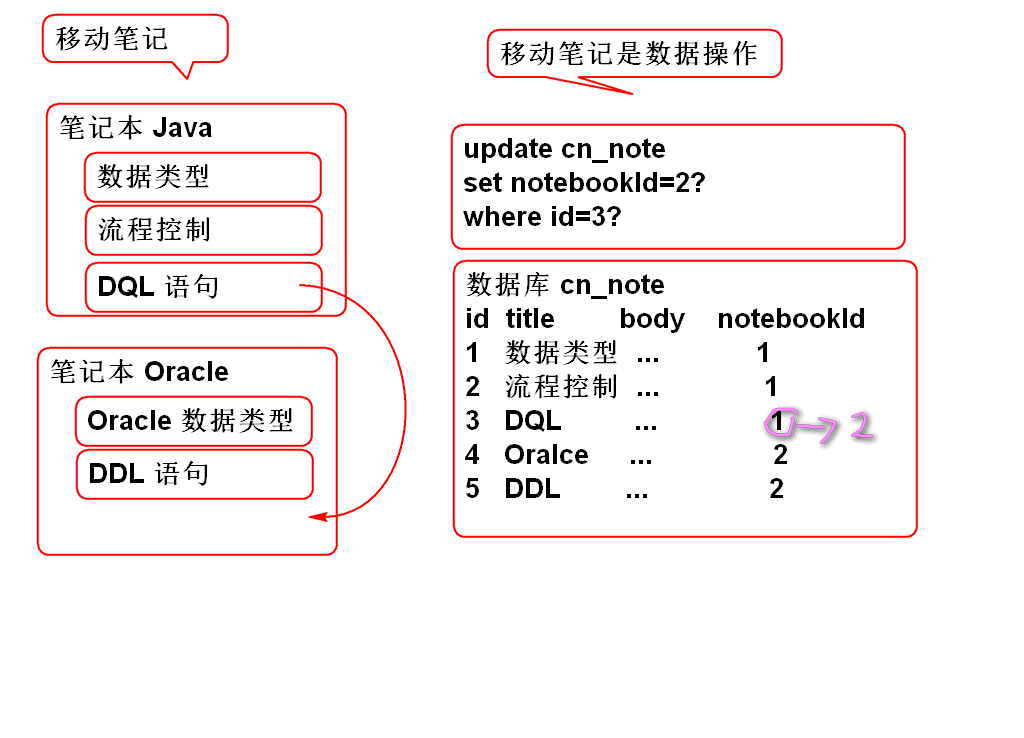
//监听整体的文档区域, 任何位置点击都要关闭笔记子菜单 $(document).click(hideNoteMenu);

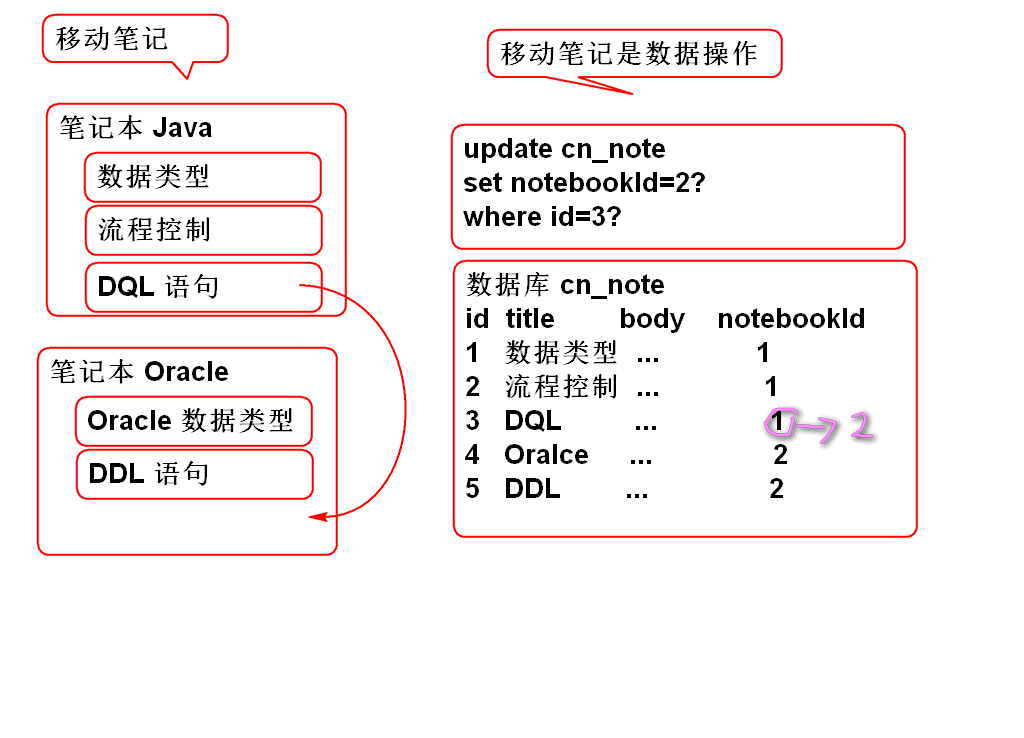
1. 添加事件处理方法关闭子菜单

/\*\* 关闭笔记子菜单事件处理方法 \*/ function hideNoteMenu(){ $('.note\_menu').hide(); }

**移动笔记功能**

原理:





**1. 持久层: 重用 NoteDao.updateNote 方法**

略...

**2. 业务层**

1. 业务层方法 NoteService

boolean moveNote(String noteId, String notebookId) throws NoteNotFoundException, NotebookNotFoundException;

1. 实现业务层方法: NoteServiceImpl

public boolean moveNote(String noteId, String notebookId) throws NoteNotFoundException, NotebookNotFoundException { if(noteId==null || noteId.trim().isEmpty()){ throw new NoteNotFoundException("ID不能空"); } Note note = noteDao.findNoteById(noteId); if(note==null){ throw new NoteNotFoundException("没有对应的笔记"); } if(notebookId==null||notebookId.trim().isEmpty()){ throw new NotebookNotFoundException("ID空"); } int n=notebookDao.countNotebookById(notebookId); if(n!=1){ throw new NotebookNotFoundException("没有笔记本"); } Note data = new Note(); data.setId(noteId); data.setNotebookId(notebookId); data.setLastModifyTime(System.currentTimeMillis()); n = noteDao.updateNote(data); return n==1; }

1. 测试

**3. 表现层**

1. 在ready方法中监听移动按钮点击事件打开对话框

//监听笔记子菜单中移动按钮的点击 $('#note-list').on('click', '.btn\_move', showMoveNoteDialog);

1. 添加事件处理方法, 打开对话框, 打开对话框以后加载笔记本下拉列表

/\*\* 显示移动笔记对话框 \*/ function showMoveNoteDialog(){ var id = $(document).data('note').id; if(id){ $('#can').load('alert/alert\_move.html', loadNotebookOptions); $('.opacity\_bg').show(); return; } alert('必须选择笔记!'); }

1. 添加加载笔记本列表事件方法

/\*\* 加载移动笔记对话框中的笔记本列表 \*/ function loadNotebookOptions(){ var url = 'notebook/list.do'; var data={userId:getCookie('userId')}; $.getJSON(url, data, function(result){ if(result.state==SUCCESS){ var notebooks = result.data; //清楚全部的笔记本下拉列表选项 //添加新的笔记本列表选项 $('#moveSelect').empty(); var id=$(document).data('notebookId'); for(var i=0; i<notebooks.length; i++){ var notebook = notebooks[i]; var opt=$('<option></option>') .val(notebook.id) .html(notebook.name); //默认选定当时笔记的笔记本ID if(notebook.id==id){ opt.attr('selected','selected'); } $('#moveSelect').append(opt); } }else{ alert(result.message); } }); }

1. 在ready方法中监听 移动笔记对话框中的确定按钮

//监听移动笔记对话框中的确定按钮 $('#can').on('click', '.move-note', moveNote);

重构 alert\_move.html, 在确定按钮上增加 类 move-note

<button type="button" class="btn btn-primary sure move-note">确 定</button>

1. 添加移动笔记对话框确认事件:

/\*\* 移动笔记事件处理方法 \*/ function moveNote(){ var url = 'note/move.do'; var id = $(document).data('note').id; var bookId=$('#moveSelect').val(); //笔记本ID没有变化, 就不移动了! if(bookId==$(document).data('notebookId')){ return; } var data = {noteId:id, notebookId:bookId}; $.post(url, data, function(result){ if(result.state==SUCCESS){ //移动成功, 在当前笔记列表中删除移动的笔记 //将笔记列表中的第一个设置为当前笔记, 否则清空边编辑区域 var li = $('#note-list .checked').parent(); var lis = li.siblings(); if(lis.size()>0){ lis.eq(0).click(); }else{ $('#input\_note\_title').val(""); um.setContent(""); } li.remove(); closeDialog();//关闭对话框! }else{ alert(result.message); } }); }

1. 测试

**删除笔记功能**

**1. 持久层**

1. 重用 NoteDao.updateNote 方法.

略

1. 重构SQL, 笔记本列表中显示没有删除的笔记: NoteMapper.xml

<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} and cn\_note\_status\_id = '1' order by cn\_note\_last\_modify\_time desc </select>

注意：　增加了查询条件 cn*note*status\_id = '1'

**2. 业务层**

1. 添加业务层方法 NoteService

boolean deleteNote(String noteId) throws NoteNotFoundException;

1. 实现业务层方法 NoteServiceImpl

public boolean deleteNote(String noteId) throws NoteNotFoundException { if(noteId==null || noteId.trim().isEmpty()){ throw new NoteNotFoundException("ID不能空"); } Note note = noteDao.findNoteById(noteId); if(note==null){ throw new NoteNotFoundException("没有对应的笔记"); } Note data = new Note(); data.setId(noteId); data.setStatusId("0"); data.setLastModifyTime(System.currentTimeMillis()); int n = noteDao.updateNote(data); return n==1; }

1. 测试

略

**3. 控制器**

1. 添加控制器方法 NoteController

@RequestMapping("/delete.do") @ResponseBody public JsonResult delete(String noteId) { boolean b = noteService.deleteNote( noteId); return new JsonResult(b); }

1. 测试

略

**4. 表现层**

1. 在ready 方法中监听 笔记子菜单中的删除按钮:

//监听笔记子菜单中删除按钮的点击 $('#note-list').on('click', '.btn\_delete', showDeleteNoteDialog);

添加事件处理方法

/\*\* 打开删除笔记对话框 \*/ function showDeleteNoteDialog(){ var id = $(document).data('note').id; if(id){ $('#can').load('alert/alert\_delete\_note.html', loadNotebookOptions); $('.opacity\_bg').show(); return; } alert('必须选择笔记!'); }

1. 在ready中监听删除笔记对话框中的确定按钮事件

//监听删除笔记对话框中的确定按钮 $('#can').on('click', '.delete-note', deleteNote);

重构 alert*detete*note.html 在确定按钮上增加 类 delete-note

<button type="button" class="btn btn-primary sure delete-note">删 除</button>

事件监听方法

/\*\* 删除笔记功能 \*/ function deleteNote(){ var url = 'note/delete.do'; var id = $(document).data('note').id; var data = {noteId:id}; $.post(url, data, function(result){ if(result.state==SUCCESS){ //删除成功, 在当前笔记列表中删除笔记 //将笔记列表中的第一个设置为当前笔记, 否则清空边编辑区域 var li = $('#note-list .checked').parent(); var lis = li.siblings(); if(lis.size()>0){ lis.eq(0).click(); }else{ $('#input\_note\_title').val(""); um.setContent(""); } li.remove(); closeDialog();//关闭对话框! }else{ alert(result.message); } }); }

1. 测试

略...

**JS 面向对象编程**

**如何创建JS对象**

1. JSON语法声明对象(直接量声明对象)

var obj = {};

1. 使用 Object 创建对象

var obj = new Object();

**JS对象可以后期添加属性**

案例

var obj = {}; obj.name = "Tom"; var obj2 = new Object(); obj2.name = "Jerry";

对象特点:

1. new Object() 和 JSON 语法创建的对象没有差别!
   * JSON语法简洁方便, 更加容易使用
2. 对象可以随时添加属性
   * 对象.属性 = 值
3. 不存在的属性, 值是 undefined
   * undefined 相当于 false, 利用这个特点可以用于检测属性是否存在

if(! obj.age){ console.log('没有age属性'); } if(obj.age){ console.log('年龄:'+obj.age); }

1. 可以随时删除对象的属性

delete 对象.属性;

1. JS Object的底层本质是一个散列表!
   * 为对象添加属性, 本质是添加了 key:value, key是属性名, value是属性值.
   * 访问对象属性, 本质是get(key)

JS对象没有封装性可言! 因为不能完整支持面向对象3大特性, 所有JS不是面向对象的编程语言!

**JS 对象的方法**

js对象的方法, 本质是一个属性, 是一个值是函数对象的属性!

var obj = {}; obj.name = "Tom"; obj.who = function(){ console.log(this.name); };

调用方法与访问属性

obj.who();//调用方法 obj.who; 访问属性的值, 函数对象

可以像属性一样删除方法

delete obj.who

可以像属性一样, 修改方法!

obj.who=function(){ console.log(this.name); }; obj.who=function(){ console.log('Hello World!'); }; //who方法引用最后一个函数对象

JS方法没有重载!!

obj.add = function(a, b){ return a+b; } obj.add = function(a, b, c){ return a+b+c; } //最后只保留最后的方法: add = a+b+c obj.add(1,2) //返回未定义 obj.add(1,2,3) //返回6

**使用JSON直接声明属性和方法**

var obj = { name:'Tom', age:18, who:function(){ console.log(this.name); } }; //后期扩展属性 obj.price=25;

**默认的变量和函数**

在网页值直接声明的变量和函数, 是window对象的属性和方法

也可以利用赋值, 修改window提供的属性和方法:

//重写JS原生alert函数 window.alert=function(e){ $('#can').load('./alert/alert\_error.html',function(){ $('#error\_info').text(' '+e); $('.opacity\_bg').show(); }); }

**更正**

1. 重构 note.js

更正: 增加 li.data('noteId', note.id) 否则无法再次点击笔记了!

function addNote(){ var url = 'note/add.do'; var notebookId=$(document).data('notebookId'); var title = $('#input\_note').val(); var data = {userId:getCookie('userId'), notebookId:notebookId, title:title}; //console.log(data); $.post(url, data, function(result){ if(result.state==SUCCESS){ var note=result.data; //console.log(note); showNote(note); //找到显示笔记列表的ul对象 var ul = $('#note-list ul'); //创建新新的笔记列表项目 li var li = noteTemplate.replace( '[title]', note.title); li = $(li); //绑定笔记ID到LI li.data('noteId', note.id) //设置选定效果 ul.find('a').removeClass('checked'); li.find('a').addClass('checked'); //插入到笔记列表的第一个位置 ul.prepend(li); //关闭添加对话框 closeDialog(); }else{ alert(result.message); } }); }

1. 更正 NoteServiceImpl

更正: addNote 方法中 String statusId = "1" 否则笔记是删除状态的!

public Note addNote(String userId, String notebookId, String title) throws UserNotFoundException, NotebookNotFoundException { if(userId==null||userId.trim().isEmpty()){ throw new UserNotFoundException("ID空"); } User user=userDao.findUserById(userId); if(user==null){ throw new UserNotFoundException("木有人"); } if(notebookId==null||notebookId.trim().isEmpty()){ throw new NotebookNotFoundException("ID空"); } int n=notebookDao.countNotebookById(notebookId); if(n!=1){ throw new NotebookNotFoundException("没有笔记本"); } if(title==null || title.trim().isEmpty()){ title="葵花宝典"; } String id = UUID.randomUUID().toString(); String statusId = "1"; String typeId = "1"; String body = ""; long time=System.currentTimeMillis(); Note note = new Note(id, notebookId, userId, statusId, typeId, title, body, time, time); n = noteDao.addNote(note); if(n!=1){ throw new NoteNotFoundException("保存失败"); } return note; }

**作业**

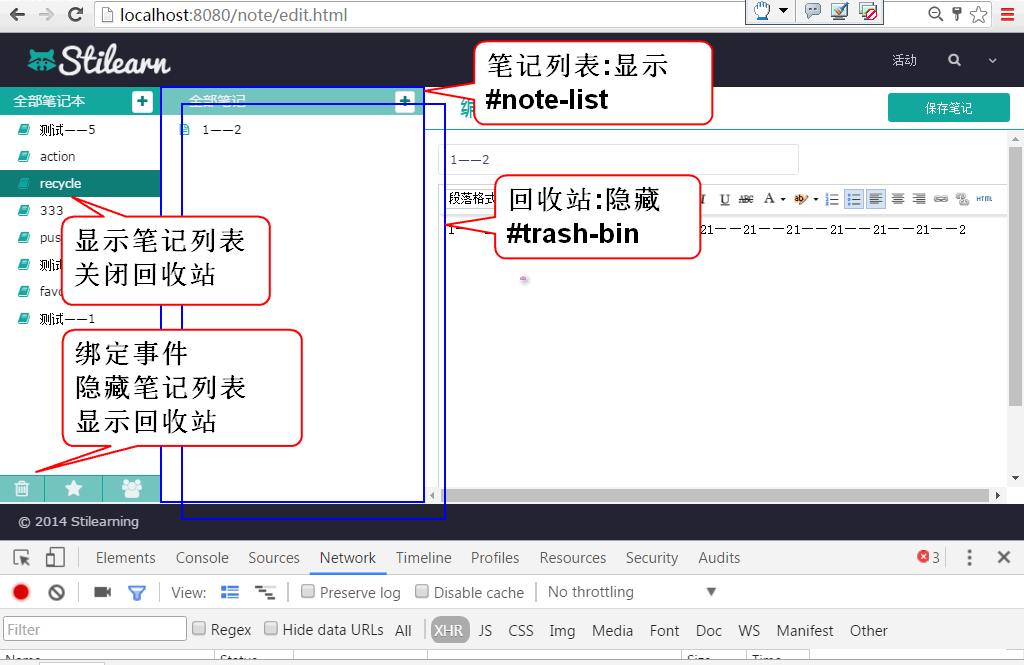
1. 完成笔记的移动功能
2. 完成笔记的的删除功能

**云笔记**

**回收站**

**1. 显示回收站**

原理:



1. 重构 edit.html 为回收站和回收站按钮设置ID

重构 118 行, 设置 id='trash-bin'

<div class="col-xs-3" style='padding:0;display:none;' id='trash-bin'>

重构 81 行, 设置 id='trash\_button'

<div class="col-xs-4 click" id='trash\_button' title='回收站'><i class='fa fa-trash-o' style='font-size:20px;line-height:31px;'></i></div>

1. 在ready方法中绑定按钮事件:

//监听回收站按钮被点击 $('#trash\_button').click(showTrashBin);

添加事件处理方法:

/\*\* 监听回收站按钮被点击 \*/ function showTrashBin(){ $('#trash-bin').show() ; $('#note-list').hide() ; //loadTrashBin(); 加载删除笔记列表 }

**2. 持久层**

1. 添加数据访问方法 NoteDao

List<Map<String, Object>> findDeleteNotesByUserId(String userId);

1. 添加SQL NoteMapper.xml

<select id="findDeleteNotesByUserId" parameterType="string" resultType="map"> select cn\_note\_id as id, cn\_note\_title as title from cn\_note where cn\_user\_id = #{userId} and cn\_note\_status\_id = '0' order by cn\_note\_last\_modify\_time desc </select>

1. 测试

...

**3. 业务层**

1. 添加业务层方法 NoteService

List<Map<String, Object>> listNotesInTrashBin(String userId) throws UserNotFoundException;

1. 实现业务层方法 NoteServiceImpl

public List<Map<String, Object>> listNotesInTrashBin( String userId) throws UserNotFoundException { if(userId==null||userId.trim().isEmpty()){ throw new UserNotFoundException("ID空"); } User user=userDao.findUserById(userId); if(user==null){ throw new UserNotFoundException("木有人"); } return noteDao.findDeleteNotesByUserId(userId); }

1. 测试

...

**4. 表现层**

1. 添加 loadTrashBin 方法利用Ajax加载回收站笔记列表:

/\*\* 加载回收站中的笔记列表 \*/ function loadTrashBin(){ var url = 'note/trash.do'; var data = {userId: getCookie('userId')}; $.getJSON(url, data, function(result){ if(result.state==SUCCESS){ showNotesInTrashBin(result.data); }else{ alert(result.message); } }); }

1. 添加显示笔记列表到回收站方法 showNotesInTrashBin

function showNotesInTrashBin(notes){ var ul = $('#trash-bin ul'); ul.empty(); for(var i=0; i<notes.length; i++){ var note = notes[i]; var li = trashBinItem.replace('[title]', note.title); li = $(li); li.data('noteId', note.id); ul.append(li); } } var trashBinItem = '<li class="disable">'+ '<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\_delete">'+ '<i class="fa fa-times"></i>'+ '</button>'+ '<button type="button" class="btn btn-default btn-xs btn\_position\_2 btn\_replay">'+ '<i class="fa fa-reply"></i>'+ '</button></a>'+ '</li>';

其中 trashBinItem 是回收站笔记项目的模板

1. 重构 showTrashBin 方法, 在显示回收站后加载以删除笔记列表

/\*\* 监听回收站按钮被点击 \*/ function showTrashBin(){ $('#trash-bin').show() ; $('#note-list').hide() ; loadTrashBin();// 加载已删除笔记列表 }

1. 测试

...

**恢复删除项目**

**1. 持久层, 重用 NoteDao updateNote 方法**

略

**2. 业务层**

1. 声明业务方法 NoteService

boolean replayNote(String noteId, String notebookId) throws NoteNotFoundException, NotebookNotFoundException;

1. 实现业务方法 NoteServiceImpl

public boolean replayNote(String noteId, String notebookId) throws NoteNotFoundException, NotebookNotFoundException { if(noteId==null || noteId.trim().isEmpty()){ throw new NoteNotFoundException("ID不能空"); } Note note = noteDao.findNoteById(noteId); if(note==null){ throw new NoteNotFoundException("没有对应的笔记"); } if(notebookId==null||notebookId.trim().isEmpty()){ throw new NotebookNotFoundException("ID空"); } int n=notebookDao.countNotebookById(notebookId); if(n!=1){ throw new NotebookNotFoundException("没有笔记本"); } Note data = new Note(); data.setId(noteId); data.setStatusId("1"); data.setNotebookId(notebookId); data.setLastModifyTime(System.currentTimeMillis()); n = noteDao.updateNote(data); return n==1; }

1. 测试

...

**3. 控制器**

1. 添加控制器方法 NoteController

@RequestMapping("/replay.do") @ResponseBody public JsonResult replay(String noteId, String notebookId) { boolean b = noteService.replayNote( noteId, notebookId); return new JsonResult(b); }

1. 测试

...

**3. 表现层**

1. 在ready方法中添加事件监听方法, 打开恢复对话框:

//恢复笔记到笔记本按钮事件监听 $('#trash-bin').on( 'click', '.btn\_replay', showReplayDialog);

添加事件方法

/\*\* 显示恢复笔记对话框 \*/ function showReplayDialog(){ var li = $(this).parent().parent() var id = li.data('noteId'); $(document).data('replayItem', li); if(id){ $('#can').load('alert/alert\_replay.html', loadReplayOptions); $('.opacity\_bg').show(); return; } alert('必须选择笔记!'); }

提示: 需要在事件中保存 li 到 document中, 在恢复时候需利用这个li获取被恢复的笔记ID

1. 添加方法loadReplayOptions, 在显示窗口以后加载笔记本列表到恢复对话框中:

function loadReplayOptions(){ var url = 'notebook/list.do'; var data={userId:getCookie('userId')}; $.getJSON(url, data, function(result){ if(result.state==SUCCESS){ var notebooks = result.data; //清楚全部的笔记本下拉列表选项 //添加新的笔记本列表选项 $('#replaySelect').empty(); var id=$(document).data('notebookId'); for(var i=0; i<notebooks.length; i++){ var notebook = notebooks[i]; var opt=$('<option></option>') .val(notebook.id) .html(notebook.name); //默认选定当时笔记的笔记本ID if(notebook.id==id){ opt.attr('selected','selected'); } $('#replaySelect').append(opt); } }else{ alert(result.message); } }); }

1. 监听恢复对话框中的确定方法:

$('#can').on('click', '.btn-replay', replayNote);

添加事件处理方法

function replayNote(){ var li = $(document).data('replayItem'); var id = li.data('noteId'); var url = 'note/replay.do'; var nid = $('#replaySelect').val(); var data = {noteId: id, notebookId:nid}; $.post(url, data, function(result){ if(result.state==SUCCESS){ closeDialog(); li.slideUp(200, function(){$(this).remove()}); }else{ alert(result.message); } }); }

提示: li对象为显示对话框事件中保存到document对象的li.

提示: li.slideUp 方法可以为删除li时候添加动画效果, 这样增加视觉效果可以提高用户的体验.

1. 测试

**权限检验**

原理

**1. 使用Servlet Filter 检查 html 文件的访问权限**

1. 重构 登录控制器, 登录以后将用户信息保存到Session, UserController

@RequestMapping("/login.do") @ResponseBody public Object login( String name, String password, HttpSession session){ User user = userService.login( name, password); //登录成功时候, 将user信息保存到session //用于在过滤器中检查登录情况 session.setAttribute("loginUser", user); return new JsonResult(user); }

1. 添加权限检查过滤器:

public class AccessFilter implements Filter { public void destroy() { } private String login = "/log\_in.html"; public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException { HttpServletRequest req = (HttpServletRequest)request; HttpServletResponse res = (HttpServletResponse)response; HttpSession session = req.getSession(); //放过 log\_in.html String path = req.getRequestURI(); System.out.println("access:"+path); if(path.endsWith(login)){ chain.doFilter(request, response); return; } //放过 alert\_error.html if(path.endsWith("alert\_error.html")){ chain.doFilter(request, response); return; } //检查用户是否登录 User user = (User)session .getAttribute("loginUser"); //如果没有登录就重定向到 登录页 if(user==null){//没有登录 //重定向到登录页 res.sendRedirect( req.getContextPath()+login); return; } //如果登录就放过 chain.doFilter(request, response); } public void init(FilterConfig fConfig) throws ServletException { } }

在web.xml 中配置过滤器:

<filter> <display-name>AccessFilter</display-name> <filter-name>AccessFilter</filter-name> <filter-class>cn.tedu.note.web.AccessFilter</filter-class> </filter> <filter-mapping> <filter-name>AccessFilter</filter-name> <url-pattern>\*.html</url-pattern> </filter-mapping>

1. 测试

**2. 使用 Spring MVC 拦截器检查 Ajax请求是否登录**

1. 添加拦截器Bean

@Component public class AccessInterceptor implements HandlerInterceptor { public boolean preHandle( HttpServletRequest req, HttpServletResponse res, Object handle) throws Exception { String path=req.getRequestURI(); System.out.println("Interceptor:"+path); HttpSession session = req.getSession(); User user = (User)session .getAttribute("loginUser"); //如果没有登录就返回错误的JSON消息 if(user==null){ JsonResult result = new JsonResult("需要重新登录!"); //利用response 对象反馈结果 res.setContentType( "application/json;charset=UTF-8"); res.setCharacterEncoding("UTF-8"); ObjectMapper mapper = new ObjectMapper(); String json=mapper .writeValueAsString(result); res.getWriter().println(json); res.flushBuffer(); return false; } //如果登录了就放过请求 return true;//放过请求 } public void postHandle(HttpServletRequest arg0, HttpServletResponse arg1, Object arg2, ModelAndView arg3) throws Exception { } public void afterCompletion(HttpServletRequest arg0, HttpServletResponse arg1, Object arg2, Exception arg3) throws Exception { } }

1. 配置拦截器 spring-mvc.xml

<!-- 扫描拦截器组件 --> <context:component-scan base-package="cn.tedu.note.web"/> <!-- 拦截器bean的ID是accessInterceptor --> <!-- 配置拦截器 --> <mvc:interceptors> <mvc:interceptor> <mvc:mapping path="/note/\*"/> <mvc:mapping path="/notebook/\*"/> <ref bean="accessInterceptor"/> </mvc:interceptor> </mvc:interceptors>

1. 测试...

**3. 利用心跳检查, 保持session的活跃**

1. 在ready中添加心跳检查方法:

startHeartbeat();

1. 实现心跳检查方法:

function startHeartbeat(){ var url = "user/heartbeat.do"; setInterval(function(){ $.getJSON(url, function(result){ console.log(result.data); }); }, 5000); }

1. 添加控制器处理心跳检查 UserController

@RequestMapping("/heartbeat.do") @ResponseBody public JsonResult heartbeat(){ Object ok = "ok"; return new JsonResult(ok); }

**作业**

1. 完成回收站功能
2. 完成登录权限检查功能