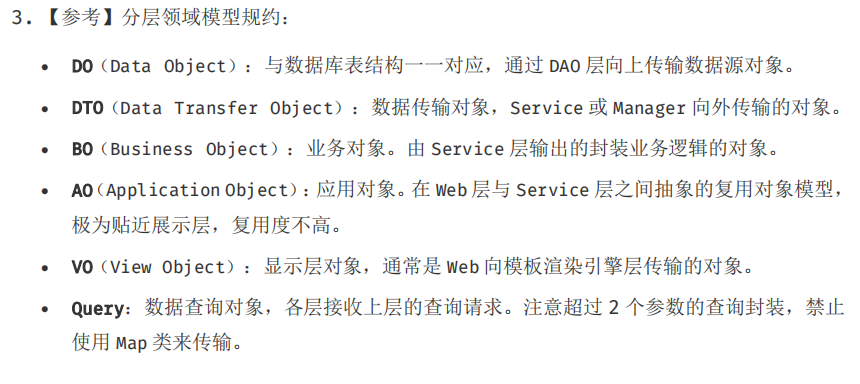
# 加油站监管系统难点答疑

# 多表查询



field: 'deptName', title: '部门名称', width: 120,  
templet: function (d) {  
 return d.dept.deptName  
}

|  |
| --- |
| @RequestMapping("/selectAllUser") @ResponseBody public LayUiTable selectAllUser(Integer page, Integer limit, String searchLoginName, String searchUserName, String searchPhonenumber){   LayUiTable table = new LayUiTable();  //多条件查询所需要的集合  Map<String ,Object> condition= new HashMap(16);   Wrapper wrapper = new EntityWrapper();  //添加模糊查询的条件  if (null != searchLoginName && !"".equals(searchLoginName)) {  wrapper.like("login\_name", searchLoginName);  condition.put("login\_name", searchLoginName);  }  if (null != searchUserName && !"".equals(searchUserName)) {  condition.put("user\_name", searchUserName);  }  if (null != searchPhonenumber && !"".equals(searchPhonenumber)) {  condition.put("phonenumber", searchPhonenumber);  }  wrapper.eq("del\_flag", 0);  condition.put("del\_flag", 0);  int userListCount = userService.selectCount(wrapper);  //如果表中没有数据。则不进行分页查询  if(userListCount>0){  //获取当前的系统毫秒数  long start = System.*currentTimeMillis*();  Page<UserVo> pageInfo = new Page(page, limit);  List<UserVo> userVoList = userService.selectUserVoList(pageInfo,condition);  //从分页结果中提取list集合  table.setCode(MyConstants.*OPERATION\_SUCCESS\_CODE*);  table.setMsg(MyConstants.*OPERATION\_SUCCESS\_MESSAGE*);  table.setData(userVoList);  table.setCount(userListCount);  long end = System.*currentTimeMillis*();  System.*out*.println("============查询花费时间："+(end-start)+"============");  return table;  }else  {   return table;  }  } |



需要自己创建一个vo对象，给页面使用

|  |
| --- |
| package com.aaa.entity;  import com.baomidou.mybatisplus.annotations.TableField; import lombok.Data;  import java.io.Serializable; import java.util.Date;  */\*\*  \* <p>  \* 用户信息表  \* </p>  \*  \** ***@author*** *AAATeacherChen  \** ***@since*** *2020-06-09  \*/* @Data public class UserVo implements Serializable{   private static final long *serialVersionUID* = 1L;   */\*\*  \* 用户ID  \*/* private Integer userId;  */\*\*  \* 部门ID  \*/* private Integer deptId;  */\*\*  \* 登录账号  \*/* private String loginName;  */\*\*  \* 用户昵称  \*/* private String userName;  */\*\*  \* 用户邮箱  \*/* private String email;  */\*\*  \* 手机号码  \*/* private String phonenumber;  */\*\*  \* 用户性别（0男 1女 2未知）  \*/* private String sex;  private String avatar;  */\*\*  \* 密码  \*/* private String password;  */\*\*  \* 盐加密  \*/* private String salt;  */\*\*  \* 帐号状态（0正常 1停用）  \*/* private String status;  */\*\*  \* 删除标志（0代表存在 2代表删除）  \*/* private String delFlag;  */\*\*  \* 创建者  \*/* private String createBy;  */\*\*  \* 创建时间  \*/* private Date createTime;  */\*\*  \* 更新者  \*/* private String updateBy;  */\*\*  \* 更新时间  \*/* private Date updateTime;  */\*\*  \* 备注  \*/* private String remark;   private Dept dept;   } |

需要在mybatis的mapper映射文件中，添加resultMap

|  |
| --- |
| <resultMap id="UserDeptResultMap" type="com.aaa.entity.UserVo">  <id column="user\_id" property="userId" />  <result column="dept\_id" property="deptId" />  <result column="login\_name" property="loginName" />  <result column="user\_name" property="userName" />  <result column="email" property="email" />  <result column="phonenumber" property="phonenumber" />  <result column="sex" property="sex" />  <result column="avatar" property="avatar" />  <result column="password" property="password" />  <result column="salt" property="salt" />  <result column="status" property="status" />  <result column="del\_flag" property="delFlag" />  <result column="create\_by" property="createBy" />  <result column="create\_time" property="createTime" />  <result column="update\_by" property="updateBy" />  <result column="update\_time" property="updateTime" />  <result column="remark" property="remark" />  <association property="dept" column="dept\_id" javaType="com.aaa.entity.Dept">  <result column="dept\_name" property="deptName"></result>  </association> </resultMap> |

最后需要自己编写多表关联查询的sql

|  |
| --- |
| <!--多条件查询--> <select id="selectUserVoList" resultMap="UserDeptResultMap">  select u.\*,d.dept\_name from tbl\_user u INNER JOIN tbl\_dept d on u.dept\_id=d.dept\_id  <where>  <if test="condition.login\_name != null and condition.login\_name != ''">  AND u.login\_name like concat('%', #{condition.login\_name},'%')  </if>  <if test="condition.user\_name != null and condition.user\_name != ''">  AND u.user\_name like concat('%', #{condition.user\_name},'%')  </if>  <if test="condition.phonenumber != null and condition.phonenumber != ''">  AND u.phonenumber like concat('%', #{condition.phonenumber},'%')  </if>  <if test="condition.del\_flag != null and condition.del\_flag != ''">  AND u.del\_flag = #{condition.del\_flag}  </if>  </where> </select> |

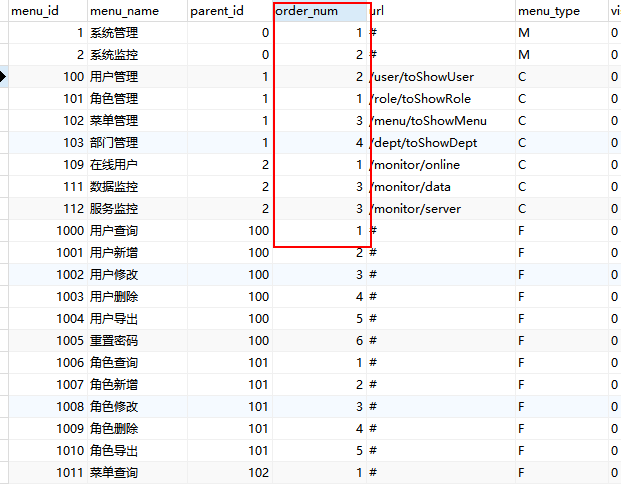
# 批量删除

|  |
| --- |
| //对于删除或者是其他敏感操作，一定要让用户确认。 layer.confirm('确定要删除吗？', function () {  //请求后台批量删除接口  $.ajax({  url: '/dept/deleteBatchDept',  dataType: 'json',  contentType: "application/json;charset=UTF-8",  type: 'POST',  data: ***JSON***.stringify(data),//将js对象转成json串传入到后台  success: function (result) {  layer.closeAll();//关闭对话框  layer.msg(result.msg);  myTable.reload();//重新加载表格数据  }  })  }) |
| @RequestMapping(value = "/deleteBatchDept") @ResponseBody public Result deleteBatchDept(@RequestBody List<Dept> deptList) {  List<Dept> deptListNew = new ArrayList<>();  //此处是逻辑删除，修改delflag  for (Dept dept : deptList) {  Dept deptNew = new Dept();  deptNew.setDelFlag("1");  deptNew.setDeptId(dept.getDeptId());  deptListNew.add(deptNew);  }  boolean update = deptService.updateBatchById(deptListNew);  if (update) {  return super.success();  } else {  return super.error();  } } |

# 排序

要求表中必须有排序列

在查询的时候按照排序列进行正序排列



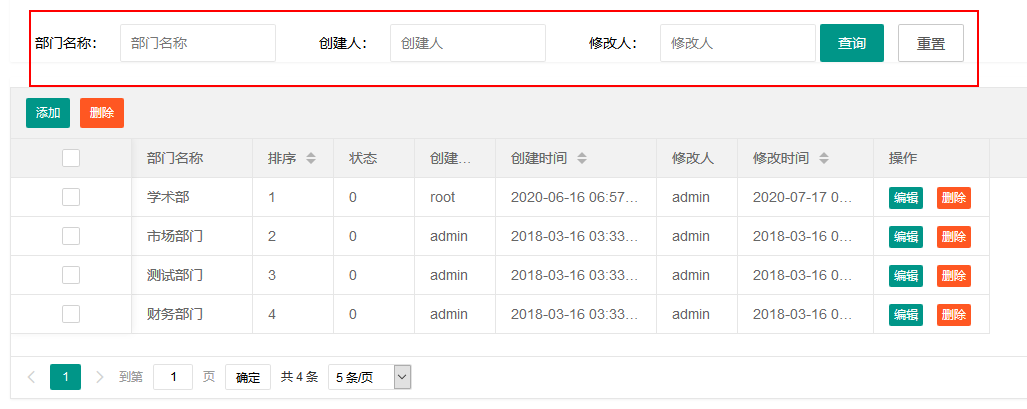
//排序  
wrapper.orderBy("order\_num",true);

# 多条件查询

多条件查询的使用场景？

数据量大，并且查询频繁的。经常查询的关键字需要设置为查询条件，一般不超过所有列的50%。

多条查询的实现：



步骤一：编写多条件查询的form表单

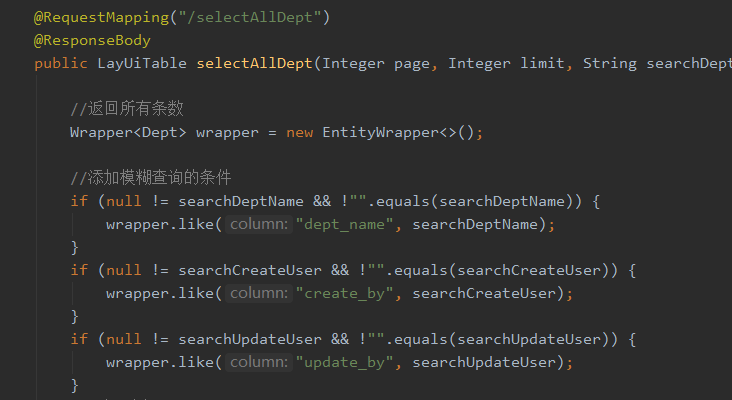
步骤二：在查询表单上绑定提交事件，在事件触发时调用表格重载。

数据表格重载可以参考官方文档。需要在指定表格初始化时定义一个id。





步骤三：在controller中接收三个条件，并判断是否为空。

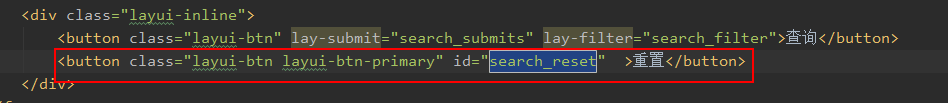


# 重置按钮

在条件查询的时候，需要增加重置按钮，作用是清空条件输入框，

重载数据表格。

坑：layui的form表单中的reset按钮阻断了click事件。需要手动自己定义click事件。



//点击重置按钮，清除查询关键字，并且重新加载表格数据  
$("#search\_reset").on("click",function () {  
 $("#searchDeptName").val=null;  
 $("#searchCreateUser").val=null;  
 $("#searchUpdateUser").val=null;  
 myTable.reload();//重新加载表格数据  
})

# 下拉框实现





官方网站的下拉框数据是写死的，我们的数据是需要从数据库查询出来。

|  |
| --- |
| @RequestMapping("/toShowUser") public String toShowUser(Model model){  //查询所有的部门信息，填充到页面下拉框中  List<Dept> deptList = deptService.selectList(null);  model.addAttribute("deptList", deptList);  return "user/showUser"; } |

|  |
| --- |
| <div class="layui-form-item">  <label class="layui-form-label">部门</label>  <div class="layui-input-block">  <select name="city" lay-verify="required" >  <option th:each="dept:${deptList}" th:value="${dept.deptId}" th:text="${dept.deptName}"></option>  </select>  </div> </div> |

# 用户添加

1用户添加的时候，明文密码是用户手动输入，保存到数据库的密码是密文+盐值。

1. 添加用户的时候选择的角色信息保存在盐值中，随着表单一起提交给后端。

<div class="layui-form-item">  
 <label class="layui-form-label">部门</label>  
 <div class="layui-input-block">  
 <select name="deptId" lay-verify="required">  
 <option th:each="dept:${deptList}" th:value="${dept.deptId}" th:text="${dept.deptName}"></option>  
 </select>  
 </div>  
</div>

<div class="layui-form-item" pane="">  
 <label class="layui-form-label">角色</label>  
 <div class="layui-input-block">  
 <input th:each="role:${roleList}" name="role" type="checkbox" th:value="${role.roleId}"  
 lay-skin="primary"  
 th:text="${role.roleName}">  
 </div>  
 <input hidden="hidden" name="salt">  
</div>

提交表单的时候，需要获取用户对应的角色

//监听修改用户表单的提交  
form.on('submit(formUpdateUser)', function (data) {  
 data.field.status = data.field.status ? 0 : 1;  
 var checkeds = selectCheckeds();  
 //盐值在添加的时候没有用处，所以将角色信息放入到盐值中传入后端  
 data.field.salt = checkeds;  
 $.ajax({  
 url: '/user/updateUser',  
 dataType: 'json',  
 type: 'POST',  
 data: data.field,  
 success: function (result) {  
 layer.closeAll();//关闭对话框  
 layer.msg(result.msg);  
 myTable.reload();//重新加载表格数据  
 }  
 })  
 return false;  
});

下面的方法是获取角色复选框选中的角色

// 获取选中复选框项  
function selectCheckeds(name) {  
 var checkeds = "";  
 // 保存用户  
 if (name != null) {  
 $('input:checkbox[name="' + name + '"]:checked').each(function (i) {  
 if (0 == i) {  
 checkeds = $(this).val();  
 } else {  
 checkeds += ("," + $(this).val());  
 }  
 });  
 //修改用户  
 } else {  
 $('.role\_check:checked').each(function (i) {  
 if (0 == i) {  
 checkeds = $(this).val();  
 } else {  
 checkeds += ("," + $(this).val());  
 }  
 });  
 }  
  
 return checkeds;  
}

# 用户修改



不能在修改页面修改的内容有哪些？

userId

createTime

createBy

password

loginname

修改页面需要数据的回显

回显的难点有以下两个地方



, "deptId": data.deptId



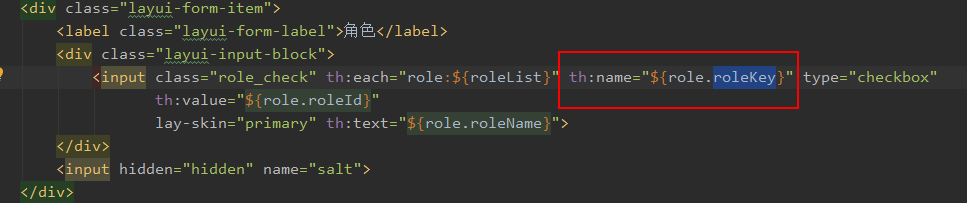


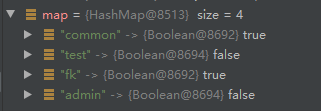
上面的查询是返回当前用户的角色

下面的查询是所有的角色









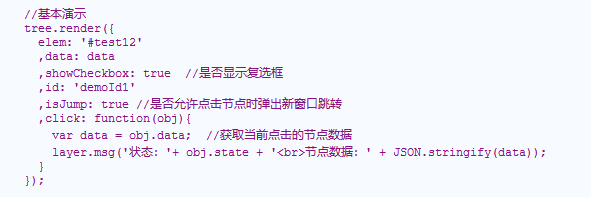
*/\*\*  
 \** ***@create*** *by: Teacher陈  
 \** ***@description:*** *根据用户id查询自己的角色  
 \** ***@create*** *time: 2020/7/18 21:28  
 \*/*@RequestMapping("/getRoleCheckByUserId")  
@ResponseBody  
public Result getRoleCheckByUserId(Integer userId){  
 List<Role> roleUserList = roleService.selectListByUserId(userId);  
 List<Role> roleList = roleService.selectList(null);  
 Map map = new HashMap();  
 for (Role role : roleList) {  
 //判断集合包含对象，必须重写equals和hashcode方法  
 if(roleUserList.contains(role)){  
 map.put(role.getRoleKey(), true);  
 }else {  
 map.put(role.getRoleKey(), false);  
 }  
 }  
 System.*out*.println(map.toString());  
 return success(map);  
}

修改页面的提交

public boolean updateUserAndSalt(User user) {  
 String roleIds = user.getSalt();  
 //将用户保存到数据库  
 Integer update = userDao.updateUserColumById(user);  
 //如果插入成功，并且页面传入的角色id不为空  
 if(update>0){  
 boolean b = updateRoleUser(user, roleIds);  
 if(b){  
 return true;  
 }  
 }  
  
 return false;  
}

# 角色修改

菜单树的官方代码



$.ajax({  
 url: '/menu/findAllMenus',  
 dataType: 'json',  
 contentType: "application/json;charset=UTF-8",  
 type: 'GET',  
 success: function (result) {  
 //基本演示  
 tree.render({  
 elem: '#menuTree'  
 , data: result  
 , showCheckbox: true //是否显示复选框  
 , id: 'roleTree'  
 , isJump: true //是否允许点击节点时弹出新窗口跳转  
  
 });  
 }  
})

*/\*\*  
 \* create by: Teacher陈  
 \* description: 根据登录名字查询树状菜单  
 \* create time: 2020/7/17 14:51  
 \*  
 \** ***@Param:*** *loginName  
 \** ***@return*** *java.util.List*<*com.aaa.entity.LayUiTree*>  
 *\*/*public List<LayUiTree> findMenus(String loginName) {  
 //查询出所有的menu  
 List<Menu> menuList = menuDao.findMenusByLoginName(loginName);  
 //将List<Menu>换成 List<LayUiTree>  
 List<LayUiTree> treeList = TreeUtil.*fromMenuListToTreeList*(menuList);  
 return treeList;  
}

# 项目代码优化

## 统一ajax返回结果。

|  |
| --- |
| package com.aaa.controller;  import com.aaa.entity.Result; import com.aaa.util.MyConstants;  */\*\*  \** ***@author*** *：Teacher陈  \** ***@date*** *：Created in 2020/6/17 10:36  \** ***@description：所有的controller的父类*** *\** ***@modified*** *By：  \** ***@version:*** *1.0  \*/* public class BaseController {   */\*\*  \* create by: Teacher陈  \* description:成功且带数据  \* create time: 2020/6/20 10:17  \*  \** ***@return*** *a  \** ***@Param*** *null  \*/* public Result success(Object object){  Result result = new Result();  result.setCode(MyConstants.*OPERATION\_SUCCESS\_CODE*);  result.setMsg(MyConstants.*OPERATION\_SUCCESS\_MESSAGE*);  result.setData(object);  return result;  }   */\*\*  \** ***@create*** *by: Teacher陈  \** ***@description:*** *成功但不带数据  \** ***@create*** *time: 2020/6/20 10:19  \*  \** ***@return*** *a  \** ***@Param:*** *null \*/* public Result success(){   return success(null);  }   */\*\*  \* create by: Teacher陈  \* description: 失败  \* create time: 2020/6/20 10:18  \*  \** ***@return*** *a  \** ***@Param:*** *null \*/* public Result error(Integer code, String msg){  Result result = new Result();  result.setCode(code);  result.setMsg(msg);  return result;  }   */\*\*  \* create by: Teacher陈  \* description: 失败但默认提示信息  \* create time: 2020/6/20 10:18  \*  \** ***@return*** *a  \** ***@Param:*** *null \*/* public Result error(){  Result result = new Result();  result.setCode(MyConstants.*OPERATION\_FAIL\_CODE*);  result.setMsg(MyConstants.*OPERATION\_FAIL\_MESSAGE*);  return result;  }  */\*\*  \* create by: Teacher陈  \* description: 文件上传下载  \* create time: 2020/6/23 14:03  \*  \** ***@Param:*** *null  \** ***@return*** *\*/  /\*\*  \* create by: Teacher陈  \* description: 统一异常处理  \* create time: 2020/6/23 14:03  \*  \** ***@Param:*** *null  \** ***@return*** *\*/* } |

## 统一批量增加创建和修改的信息。

使用aop面向切面编程和反射以及jskson工具。

想法：定义一个注解，加入到需要补充创建信息和修改信息的方法上。

package com.aaa.aop;  
  
import java.lang.annotation.\*;  
  
*/\*\*  
 \** ***@Author:*** *陈建  
 \** ***@Date:*** *2020/6/13 0013 10:37  
 \** ***@Version*** *1.0  
 \* 此注解使用的场合是在需要匹配创建人和创建时间，  
 \* 或者修改人和修改时间的方法上。  
 \*/*//此注解使用在方法上  
@Target(ElementType.*METHOD*)  
//注解在运行的时候生效  
@Retention(RetentionPolicy.*RUNTIME*)  
public @interface SaveOrUpdateEntityAnn {  
 //实体类的类型  
 Class<?> entityClass();  
}

编写切面类

|  |
| --- |
| package com.aaa.aop;  import com.aaa.entity.User; import com.aaa.util.MyConstants; import com.fasterxml.jackson.databind.ObjectMapper; import org.apache.shiro.SecurityUtils; import org.aspectj.lang.ProceedingJoinPoint; import org.aspectj.lang.annotation.Around; import org.aspectj.lang.annotation.Aspect; import org.aspectj.lang.annotation.Pointcut; import org.aspectj.lang.reflect.MethodSignature; import org.springframework.stereotype.Component;  import java.lang.reflect.Method; import java.util.Date;  */\*\*  \** ***@Author:*** *陈建  \** ***@Date:*** *2020/6/13 0013 10:32  \** ***@Version*** *1.0  \* 补充添加修改信息  \*/* @Aspect @Component public class EntityAspect {  @Pointcut("@annotation(com.aaa.aop.SaveOrUpdateEntityAnn)")  public void pointCutSaveOrUpdate(){}  @Around("pointCutSaveOrUpdate()")  */\*\*  \* create by: Teacher陈  \* description: 通过AOP环绕方法将controller中的保存和修改方法统一  \* 加上时间和人  \* create time: 2020/6/17 10:  \*  \** ***@Param:*** *joinPoint  \** ***@return*** *java.lang.Object  \*/* public Object aroundMethod(ProceedingJoinPoint joinPoint) throws Throwable {  //获取所有的参数  Object[] joinPointArgs = joinPoint.getArgs();  //获取方法的签名  MethodSignature signature = (MethodSignature) joinPoint.getSignature();  //获取方法  Method method = signature.getMethod();  String methodName= method.getName();  //获取注解  SaveOrUpdateEntityAnn annotation = method.getAnnotation(SaveOrUpdateEntityAnn.class);  //获取注解的值  Class<?> entityClass = annotation.entityClass();  //遍历所有的参数，并修改参数的内容  for (int i = 0; i <joinPointArgs.length ; i++) {  Object fromObject=joinPointArgs[i];  //通过jackson工具转换对象  ObjectMapper objectMapper = new ObjectMapper();  //将原始对象，通过jakson工具转换成具体的实体对象  Object toObject = objectMapper.convertValue(fromObject, entityClass);  //获取当前登录的用户  User user = (User) SecurityUtils.*getSubject*().getPrincipal();  String loginName = user.getLoginName();  //如果是保存方法，自动匹配创建人和创建时间  if(methodName.contains(MyConstants.*SAVE\_OPERATION*)){  Method setCreateTime = entityClass.getMethod("setCreateTime", Date.class);  setCreateTime.invoke(toObject,new Date());  Method setCreateBy = entityClass.getMethod("setCreateBy",String.class);  setCreateBy.invoke(toObject,loginName);   }  //如果是修改方法，自动匹配修改人和修改时间  if(methodName.contains(MyConstants.*UPDATE\_OPERATION*)){  Method setUpdateTime = entityClass.getMethod("setUpdateTime", Date.class);  setUpdateTime.invoke(toObject,new Date());  Method setUpdateBy = entityClass.getMethod("setUpdateBy",String.class);  setUpdateBy.invoke(toObject,loginName);  }  //将参数修改完之后，再重新设置回去  joinPointArgs[i]=toObject;  }  Object proceed = joinPoint.proceed(joinPointArgs);  return proceed;    }  } |

# 使用shiro安全框架完成登录功能

## 为什么使用shiro安全框架。

在后台管理系统中，用户的认证和授权非常重要。

1. 不同的用户登录到系统，显示的菜单功能不同。
2. 没有登录的用户，不能访问任何未授权的页面。如果有未登录用户访问该页面，需要拦截并定向到登录页面。
3. 登录验证功能，提示用户名错，密码错。
4. 对于同一个页面中的不同按钮，需要匹配相关权限才能可见。
5. 在做后台接口的时候，也可以通过shiro注解对接口方法进行权限验证。

## 如何使用shiro安全框架。

1. 引入jar包

|  |
| --- |
| <!--shiro安全框架--> <dependency>  <groupId>org.apache.shiro</groupId>  <artifactId>shiro-spring</artifactId>  <version>1.4.0</version> </dependency> <!--shiro的权限标签--> <dependency>  <groupId>com.github.theborakompanioni</groupId>  <artifactId>thymeleaf-extras-shiro</artifactId>  <version>2.0.0</version> </dependency> |

1. 编写配置类

|  |
| --- |
| package com.aaa.shiro;  import at.pollux.thymeleaf.shiro.dialect.ShiroDialect; import com.aaa.util.Constants; import org.apache.shiro.authc.credential.CredentialsMatcher; import org.apache.shiro.authc.credential.HashedCredentialsMatcher; import org.apache.shiro.spring.security.interceptor.AuthorizationAttributeSourceAdvisor; import org.apache.shiro.spring.web.ShiroFilterFactoryBean; import org.apache.shiro.web.mgt.DefaultWebSecurityManager; import org.springframework.aop.framework.autoproxy.DefaultAdvisorAutoProxyCreator; import org.springframework.context.annotation.Bean; import org.springframework.context.annotation.Configuration;  import java.util.LinkedHashMap; import java.util.Map;  */\*\*  \** ***@Author:*** *陈建  \** ***@Date:*** *2020/5/25 0025 16:56  \** ***@Version*** *1.0  \* shiro安全框架的配置类，相当于之前的shiro.xml  \*/* @Configuration public class ShiroConfig {  */\*\*  \* 定义一个bean，id为方法名  \* <bean id="fdsaf" class=">  \*  \* </bean>  \** ***@return*** *\*/* @Bean  public MyRealm myRealm(){  MyRealm myRealm = new MyRealm();  myRealm.setCredentialsMatcher(credentialsMatcher());  return myRealm;  }  */\*\*  \* securityManager  \*/* @Bean  public DefaultWebSecurityManager defaultWebSecurityManager(){  DefaultWebSecurityManager defaultWebSecurityManager= new DefaultWebSecurityManager();  defaultWebSecurityManager.setRealm(myRealm());  return defaultWebSecurityManager;  }  */\*\*  \* shiroFilterFactorybean  \* shiro的安全过滤器，过滤所有的请求，对请求分类拦截  \*/* @Bean  public ShiroFilterFactoryBean shiroFilterFactoryBean(){  ShiroFilterFactoryBean shiroFilterFactoryBean = new ShiroFilterFactoryBean();  shiroFilterFactoryBean.setSecurityManager(defaultWebSecurityManager()); */\*\*  \* 认证过滤器的分类  \* anon:无需认证  \* authc:必须认证才能到达  \* user:使用rememberme的时候才用  \* perms：访问的资源需要某个权限才能到达  \* roles:访问的资源需要某个角色才能到达  \*/* Map<String, String> map = new LinkedHashMap<>();  //放行login  map.put("/login","anon");  map.put("/css/\*\*","anon");  map.put("/img/\*\*","anon");  map.put("/js/\*\*","anon");  map.put("/json/\*\*","anon");  map.put("/layui/\*\*","anon");   //过滤所有的请求  map.put("/\*\*","authc");  //授权页面  /\* map.put("/user/toShowUser","perms[system:user:view]");\*/  shiroFilterFactoryBean.setFilterChainDefinitionMap(map);  //修改登录页面，所有的未认证的请求都给我滚，滚去登录  shiroFilterFactoryBean.setLoginUrl("/toLogin");  return shiroFilterFactoryBean;  }  */\*\*  \* 实例化密码比较器  \*/* @Bean  public CredentialsMatcher credentialsMatcher(){  HashedCredentialsMatcher credentialsMatcher= new HashedCredentialsMatcher();  //使用md5加密  credentialsMatcher.setHashAlgorithmName(Constants.*ALGORITHM\_NAME*);  //加密1000次  credentialsMatcher.setHashIterations(Constants.*HASH\_ITERATIONS*);  return credentialsMatcher;  }  */\*\*  \* 设置shiro的方言  \** ***@return*** *\*/* @Bean  public ShiroDialect shiroDialect(){  return new ShiroDialect();  }  */\*\*  \* 开启Shiro注解(如@RequiresRoles,@RequiresPermissions),  \* 需借助SpringAOP扫描使用Shiro注解的类,并在必要时进行安全逻辑验证  \* 配置以下两个bean(DefaultAdvisorAutoProxyCreator和AuthorizationAttributeSourceAdvisor)  \*/* @Bean  public DefaultAdvisorAutoProxyCreator advisorAutoProxyCreator(){  DefaultAdvisorAutoProxyCreator advisorAutoProxyCreator = new DefaultAdvisorAutoProxyCreator();  advisorAutoProxyCreator.setProxyTargetClass(true);  return advisorAutoProxyCreator;  }  */\*\*  \* 开启aop注解支持  \*/* @Bean  public AuthorizationAttributeSourceAdvisor authorizationAttributeSourceAdvisor() {  AuthorizationAttributeSourceAdvisor authorizationAttributeSourceAdvisor = new AuthorizationAttributeSourceAdvisor();  authorizationAttributeSourceAdvisor.setSecurityManager(defaultWebSecurityManager());  return authorizationAttributeSourceAdvisor;  } } |

@configuration注解是表示此类为配置类，相当于一个xml文件。

1. 编写realm

|  |
| --- |
| */\*\*  \* shiro安全框架的认证,  \** ***@param*** *token  \** ***@return*** *AuthenticationInfo ，假如返回的是null就说明认证失败  \** ***@throws*** *AuthenticationException  \*/* @Override protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) throws AuthenticationException {  System.*out*.println("认证开始");  //开始校验用户名和密码  //取出令牌信息  UsernamePasswordToken usernamePasswordToken= (UsernamePasswordToken) token;  //登录验证分两个步骤，步骤一查询用户是否存在  String username=usernamePasswordToken.getUsername();  //User userInfo = userBizImpl.selectUserByUsername(username);  Wrapper<User> wrapper = new EntityWrapper<>();  Wrapper<User> userWrapper = wrapper.eq("login\_name", username);  User user = userService.selectOne(userWrapper);  if(null==user){  return null;  }  //步骤二，查询密码是否正确  //数据库中的密码  String password=user.getPassword();  //Object principal, Object credentials, String realmName  */\*\*  \* \* @param principal the 'primary' principal associated with the specified realm.  \* \* @param hashedCredentials the hashed credentials that verify the given principal.  \* \* @param credentialsSalt the salt used when hashing the given hashedCredentials  \* \* @param realmName the realm from where the principal and credentials were acquired.  \*/* String salt = user.getSalt();  ByteSource byteSource=ByteSource.Util.*bytes*(salt);  SimpleAuthenticationInfo simpleAuthenticationInfo= new SimpleAuthenticationInfo(user,password,byteSource,getName());  return simpleAuthenticationInfo; } |

1. 编写登录方法

|  |
| --- |
| @RequestMapping("/login") public String login(String username,String password,Model model){  //使用shiro安全框架的认证实现登录逻辑  //登录验证  System.*out*.println(username+password);  //获取shiro的连接器  Subject subject = SecurityUtils.*getSubject*();  //构建用户登录令牌  UsernamePasswordToken token= new UsernamePasswordToken(username,password);  try {  subject.login(token);  }catch (UnknownAccountException e){  model.addAttribute("message","用户名错误");  return "login";  }catch (IncorrectCredentialsException e){  model.addAttribute("message","密码错误");  return "login";  }  return "helloLayUi"; } |

1. 在页面上使用shiro权限标签
2. 在controller中的方法上加入shiro注解

## 登录成功之后，显示首页信息。

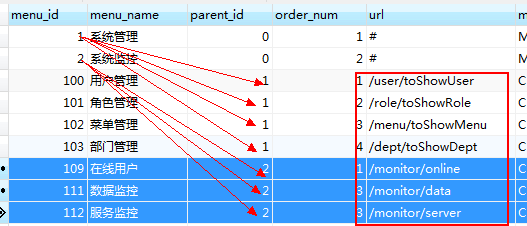


可以参考layui官方网站的后台布局

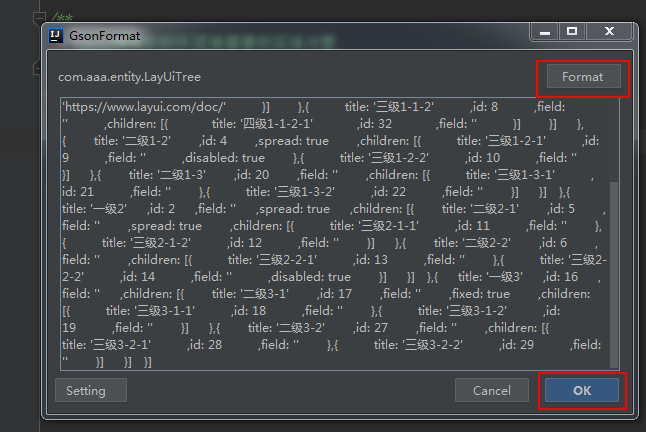








树形结构数据在页面上有两个地方使用给，一个是首页的垂直导航菜单，还有一个地方是角色对应的权限树。



将所有的菜单对象转换成layuitree对象。

SELECT

m.\*

FROM

tbl\_menu m

INNER JOIN tbl\_role\_menu rm ON m.menu\_id = rm.menu\_id

WHERE

rm.role\_id IN ( SELECT ur.role\_id FROM tbl\_user u INNER JOIN tbl\_user\_role ur ON u.user\_id = ur.user\_id WHERE u.login\_name = 'admin' )

需要使用递归查询，将所有的menu集合转换成tree集合。

## 在页面上显示树状菜单。

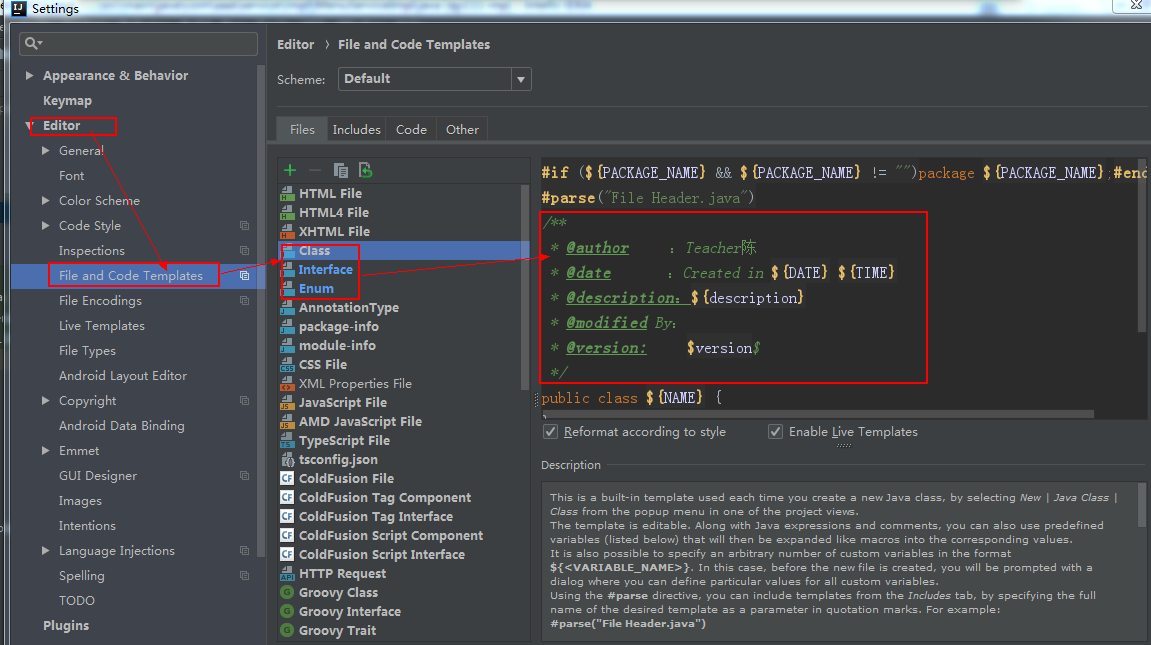


将List<Menu>换成 List<LayUiTree>

|  |
| --- |
| package com.aaa.util;  import com.aaa.entity.LayUiTree; import com.aaa.entity.Menu;  import java.util.ArrayList; import java.util.List;  */\*\*  \** ***@author*** *：Teacher陈  \** ***@date*** *：Created in 2020/6/17 15:30  \** ***@description：树状菜单工具类*** *\** ***@modified*** *By：  \** ***@version:*** *1.0  \*/* public class TreeUtil {  */\*\*  \* create by: Teacher陈  \* description:将List<Menu>换成 List<LayUiTree>  \* create time: 2020/6/17 15:32  \*  \** ***@return*** *List*<*LayUiTree*>  *\** ***@Param:*** *menuList  \*/* public static List<LayUiTree> fromMenuListToTreeList(List<Menu> menuList){  List<LayUiTree> treeList = new ArrayList<>();  //遍历所有的menu对象，然后发现menu对象有孩子，就继续便利孩子，递归操作  for (Menu menu : menuList) {  //如果父亲的id为0，就说明是一级目录  if(menu.getParentId()==0){  //将menu转成tree对象  LayUiTree tree=*fromMenuToTree*(menu);  //找自己的孩子,给tree对象设置孩子children  LayUiTree treeChilren = *setTreeChilren*(tree, menuList);  treeList.add(treeChilren);  }  }  return treeList;  }    */\*\*  \* create by: Teacher陈  \* description:找自己的孩子,给tree对象设置孩子children  \* create time: 2020/6/17 15:44  \*  \** ***@return*** *a  \** ***@Param:*** *null  \*/* public static LayUiTree setTreeChilren( LayUiTree tree,List<Menu> menuList){  //此集合封装所有的孩子  List<LayUiTree> children = new ArrayList<>();  for (Menu menu : menuList) {  //tree的id是他所有孩子的父亲id  if(menu.getParentId()==tree.getId())  {  //将menu转成tree对象  LayUiTree layUiTree=*fromMenuToTree*(menu);  children.add(*setTreeChilren*(layUiTree,menuList));  }  }  tree.setChildren(children);  return tree;  }   */\*\*  \* create by: Teacher陈  \* description: 将menu转成tree对象  \* create time: 2020/6/17 15:55  \*  \** ***@return*** *a  \** ***@Param:*** *null  \*/* public static LayUiTree fromMenuToTree(Menu menu){  LayUiTree layUiTree = new LayUiTree();  layUiTree.setId(menu.getMenuId());  layUiTree.setTitle(menu.getMenuName());  layUiTree.setUrl(menu.getUrl());  return layUiTree;  } } |

## 在idea中设置代码注释模板。

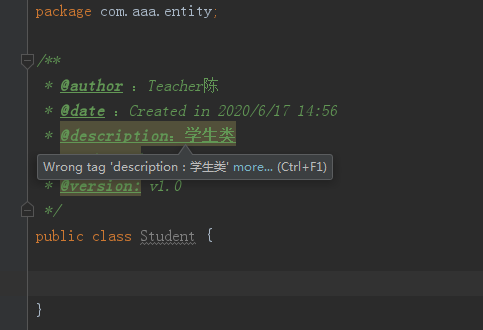
1. 设置类，接口，枚举的注释模板。



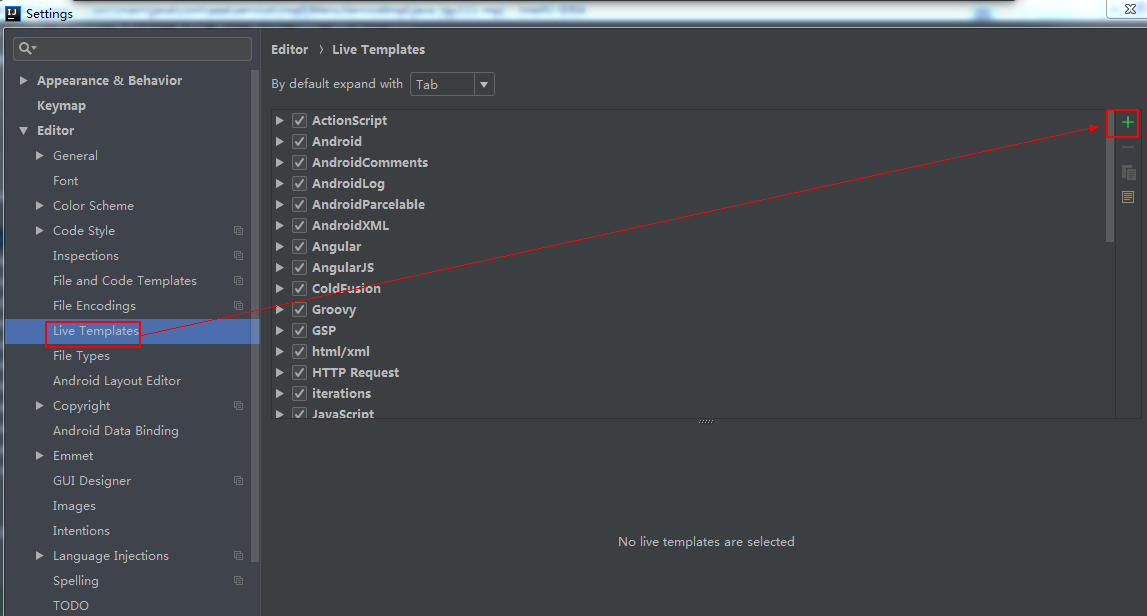
|  |
| --- |
| /\*\*  \* @author ：Teacher陈  \* @date ：Created in **${**DATE**} ${**TIME**}** \* @description：**${**description**}** \* @modified By：  \* @version: **${**version}  \*/ |

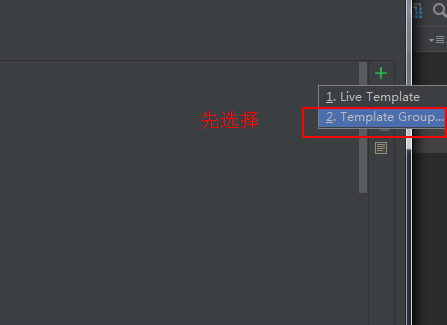
上面的四个变量${DATE} ${TIME}系统自带的。

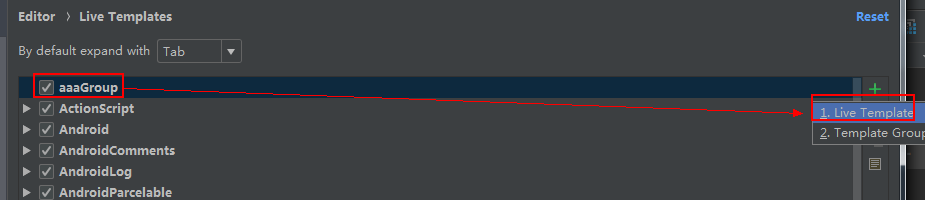
**${**description**}和$｛**version**｝需要在创建类的时候输入的。**

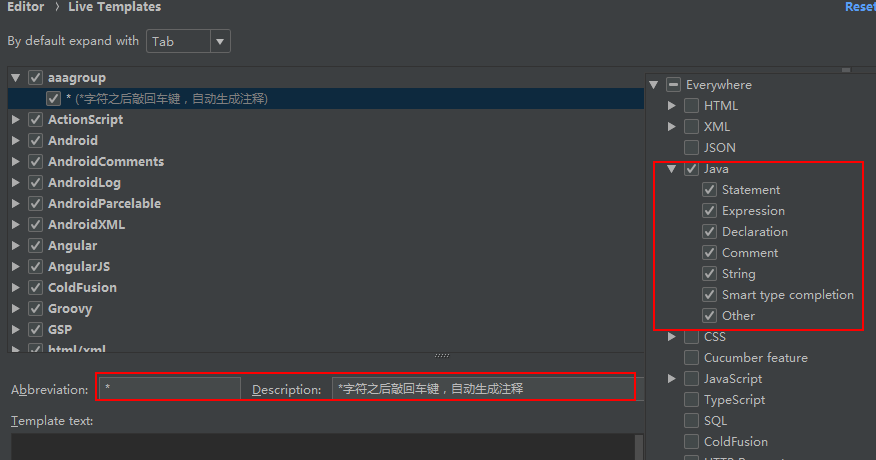


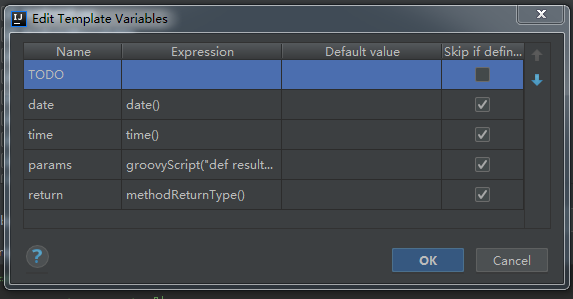
1. 设置方法的注释模板



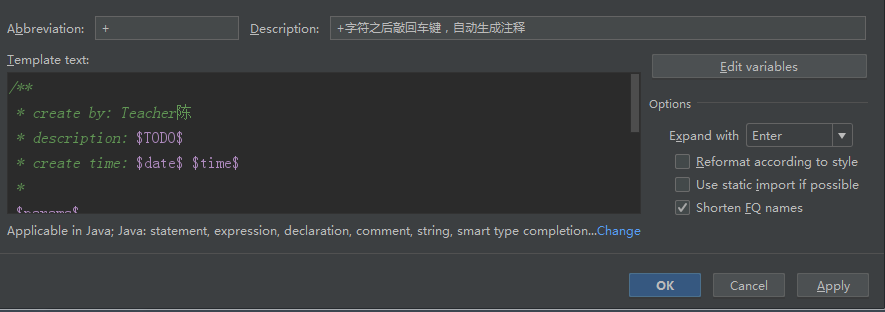








|  |
| --- |
| /\*\*  \* create by: Teacher陈  \* description: $TODO$  \* create time: $date$ $time$  \*  $params$  \* @return $return$  \*/      groovyScript("def result=''; def params=\"${\_1}\".replaceAll('[\\\\[|\\\\]|\\\\s]', '').split(',').toList(); for(i = 0; i < params.size(); i++) {result+=' \* @'+'Param: ' + params[i] + ((i < params.size() - 1) ? '\\n'+'\\b' : '')}; return result", methodParameters()) |



## 开发角色管理页面的时候需要使用权限树。



前台代码：

|  |
| --- |
| <!DOCTYPE html> <html lang="en" xmlns:th="http://www.thymeleaf.org"> <head>  <meta charset="UTF-8">  <title>显示权限树页面</title>  <link rel="stylesheet" th:href="@{/layui/css/layui.css}">  <script th:src="@{/layui/layui.js}"></script> </head> <body>  <h1>显示权限树</h1>  <div class="layui-btn-container">  <button type="button" class="layui-btn layui-btn-sm" lay-demo="getChecked">获取选中节点数据</button>  <button type="button" class="layui-btn layui-btn-sm" lay-demo="setChecked">勾选指定节点</button>  <button type="button" class="layui-btn layui-btn-sm" lay-demo="reload">重载实例</button>  </div>   <div id="test12" class="demo-tree-more"></div> </body> <script>  layui.use(['tree', 'util','jquery'], **function**() {  **var** $ = layui.jquery;  **var** tree=layui.tree;  $.ajax({  url: '/menu/findAllMenus',  dataType: 'json',  contentType: "application/json;charset=UTF-8",  type: 'GET',  success: **function** (result) {  //基本演示  tree.render({  elem: '#test12'  , data: result  , showCheckbox: **true** //是否显示复选框  , id: 'demoId1'  , isJump: **true** //是否允许点击节点时弹出新窗口跳转  , click: **function** (obj) {  **var** data = obj.data; //获取当前点击的节点数据  layer.msg('状态：' + obj.state + '<br>节点数据：' + JSON.stringify(data));  }  });  }  })    }); </script> </html> |

后台代码：

|  |
| --- |
| <!--使用resultType的场景是数据库列名和实体类属性名完全一致，如果不一致，需要自己定义resultmap-->  <select id="findMenusByLoginName" parameterType="java.lang.String" resultMap="BaseResultMap">  SELECT  DISTINCT m.\* FROM  tbl\_menu m  INNER JOIN tbl\_role\_menu rm ON m.menu\_id = rm.menu\_id WHERE  rm.role\_id IN (  SELECT  ur.role\_id  FROM  tbl\_user u  INNER JOIN tbl\_user\_role ur ON u.user\_id = ur.user\_id  <trim prefix="WHERE" prefixOverrides="AND |OR |and |or ">  <if test="loginName != null ">  AND u.login\_name = #{loginName}  </if>  </trim>  )  </select> |

|  |
| --- |
| package com.aaa.controller;   import com.aaa.entity.LayUiTree; import com.aaa.service.MenuService; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.stereotype.Controller; import org.springframework.web.bind.annotation.ResponseBody;  import java.util.List;  */\*\*  \* <p>  \* 菜单权限表 前端控制器  \* </p>  \*  \** ***@author*** *AAATeacherChen  \** ***@since*** *2020-06-09  \*/* @Controller @RequestMapping("/menu") public class MenuController {  @Autowired  private MenuService menuService;   // List<LayUiTree> layUiTreeList = menuService.findMenus(username);   @RequestMapping("/findAllMenus")  @ResponseBody  */\*\*  \* create by: Teacher陈  \* description: 查询所有的权限菜单  \* create time: 2020/6/17 16:54  \*  \** ***@Param:*** *\** ***@return*** *java.util.List*<*com.aaa.entity.LayUiTree*>  *\*/* public List<LayUiTree> findAllMenus(){  List<LayUiTree> layUiTreeList = menuService.findMenus(null);  return layUiTreeList;  } } |

# 菜单管理（treeTable）



# 权限标签

目的：让有权限的用户可以显示特定的按钮（删除按钮），没有删除权限的用户登录之后，看不见该按钮。

步骤一：需要在页面添加shiro的权限标签



步骤二：在realm中实现授权代码

|  |
| --- |
| @Override protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {  System.out.println("授权开始");  User user = (User) principals.getPrimaryPrincipal();  //将权限字符串添加到授权对象中  Set<String> allPermsByLoginName = menuService.findAllMenusByLoginName(user.getLoginName());  SimpleAuthorizationInfo info = new SimpleAuthorizationInfo();  info.setStringPermissions(allPermsByLoginName);  return info; } |