项目名称：

通用后台权限管理系统

项目目标：

1. 掌握软件开发的基本流程
2. 掌握springboot框架
3. 掌握mybatis框架各知识点
4. 掌握shiro安全框架的认证和授权
5. 掌握html和javascript各知识点
6. 掌握mysql各知识点

项目大纲：

1. 背景知识
2. 需求分析
3. 功能分析
4. 架构设计
5. 代码开发

## 1、背景知识

在当前风起云涌的电商领域和广大企业信息化管理领域，软件系统的后台管理都是不可或缺的组成部分，而后台管理中的权限管理模块是衡量一个后台管理是否安全高效的重要指标。

## 2、需求分析

由于不同的后台管理系统需求多样化，本次开发的是通用型，对于大多数的后台管理系统逻辑都已足够使用，主要应用于WEB应用程序，如：网站管理后台、CMS、CRM、OA、商城后台等等。

目前常见的权限控制分为四大类：

1. **用户管理**

对于后台产品而言，是需要公司内部人员去创建账号的。而这个账号就是一把钥匙，我们通过控制账号所具备的权限，进而控制这个员工的所操作范围。

1. **角色管理**

角色往往是基于业务管理需求而预先在系统中设定好的固定标签，每个角色对应明确的系统权限，他是一个集合的概念，是众多最小权限颗粒的组成。我们通过把权限给这个角色，再把角色给账号，从而实现账号的权限，因此它承担了一个桥梁的作用。引入角色这个概念，可以帮助我们灵活的扩展，使一个账号可以具备多种角色。

1. **功能权限控制**

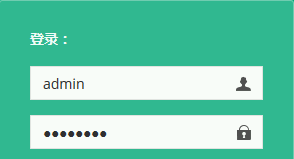
功能权限定义：为可见、可以操作的功能范围。在目录、菜单上加权限控制，有权限的就可以访问对应模块，没有的连菜单名都看不到。在业务模块的功能按钮上加权限控制，最小粒度的控制用户行为。

1. **数据权限控制**

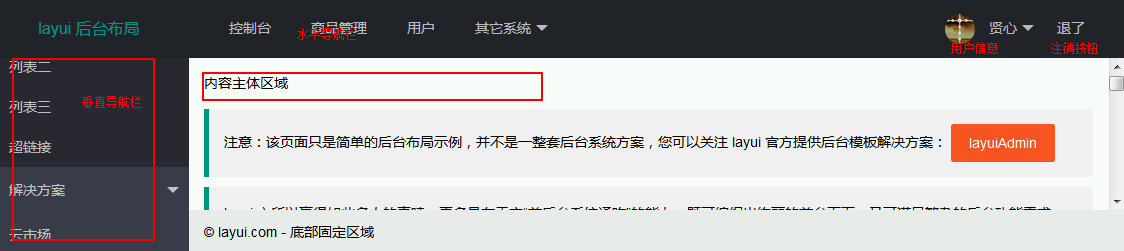
数据权限控制主要控制某条数据记录对用户是否可见，结合功能权限可以更灵活的配置业务过程中每一位员工的功能操作权限及数据可见范围，全面保障企业数据的安全性。

## 3、功能分析

1. 登录功能
   1. 登录页面



* 1. 异常登录提示错误信息



1. 用户管理
   1. 查询用户



物理删除

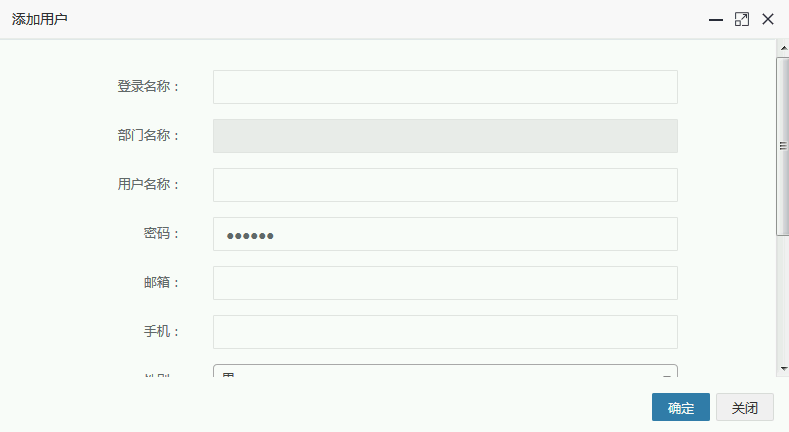
delete from table where userid=1

逻辑删除（关键数据）

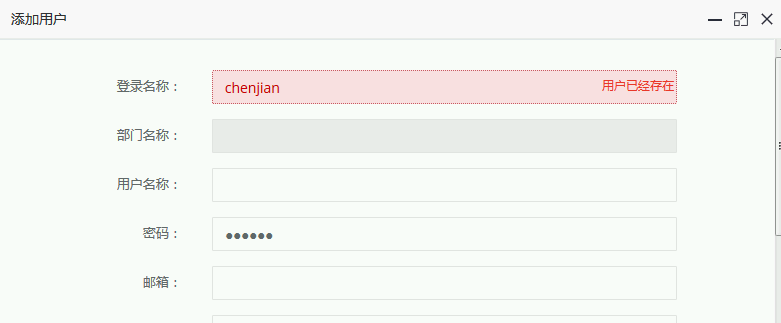
del\_flag 0 未删除 1 删除

update table set del\_flag=1 where userid=1

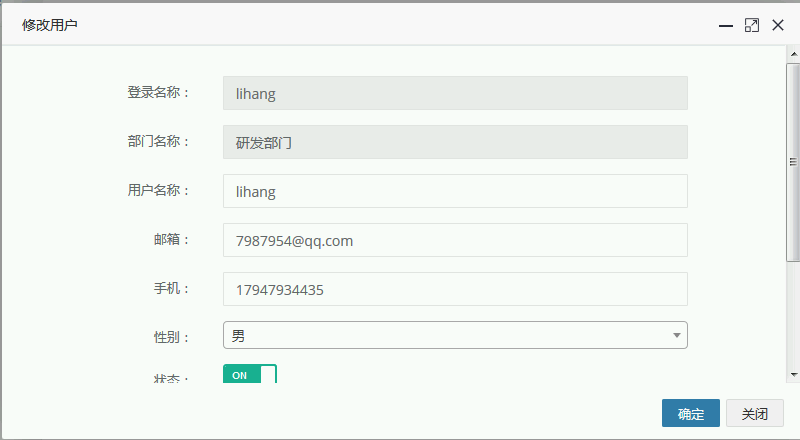
* 1. 添加用户，设置角色



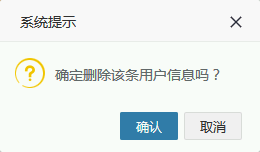
校验登录用户名是否存在onblur，layui的input校验



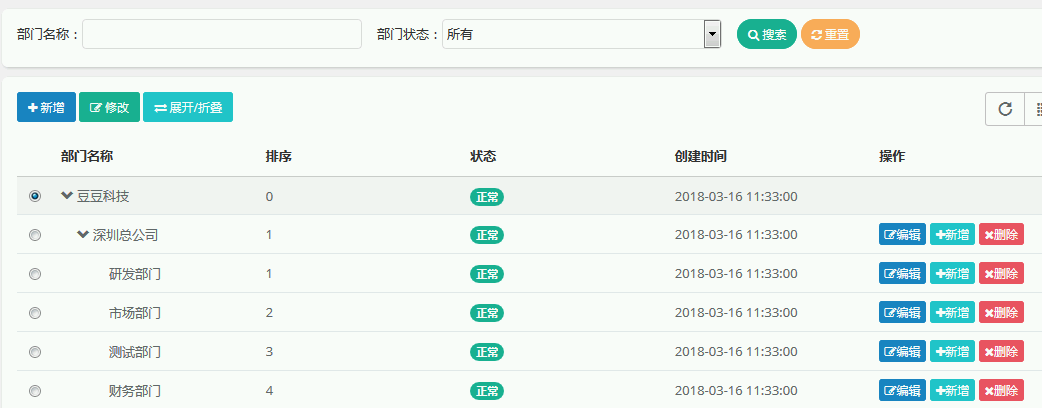
* 1. 修改用户，禁用用户



* 1. 删除用户



1. 部门管理
   1. 查询部门



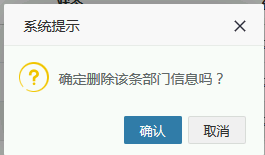
* 1. 添加部门



* 1. 修改部门，禁用部门



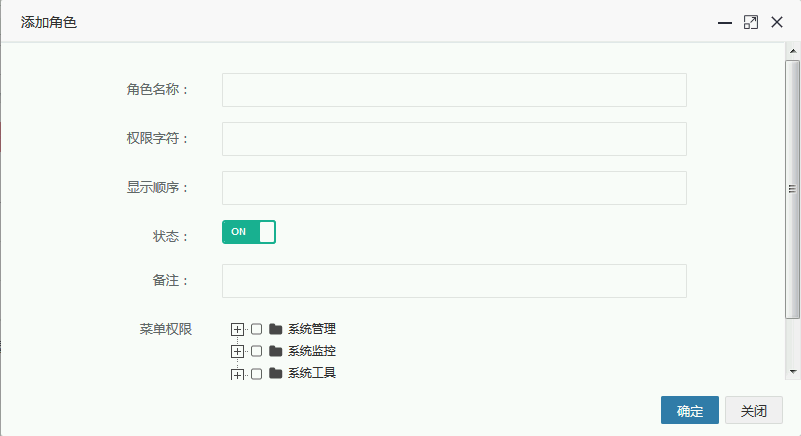
* 1. 删除部门



1. 角色管理
   1. 查询角色



* 1. 添加角色

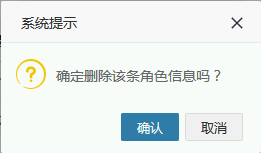


* 1. 修改角色





* 1. 删除角色

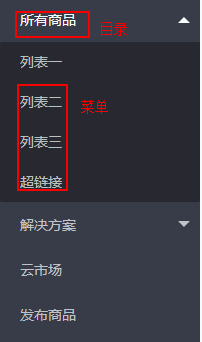


1. 菜单管理
   1. 查询菜单（treeTable）

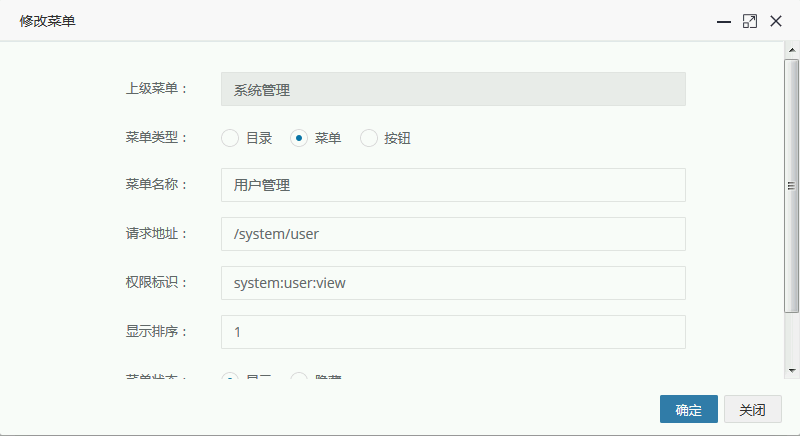


* 1. 添加菜单

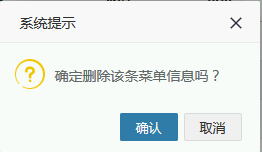




* 1. 修改菜单



* 1. 删除菜单

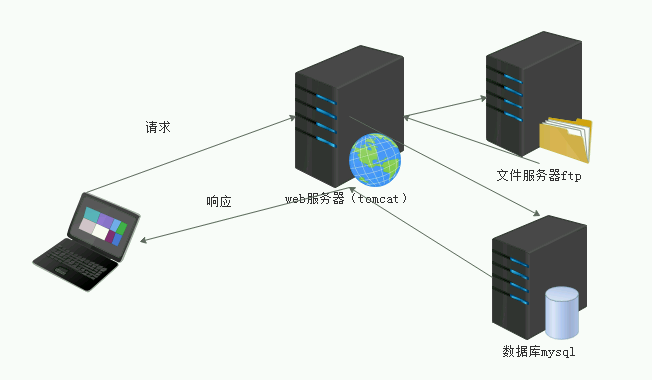


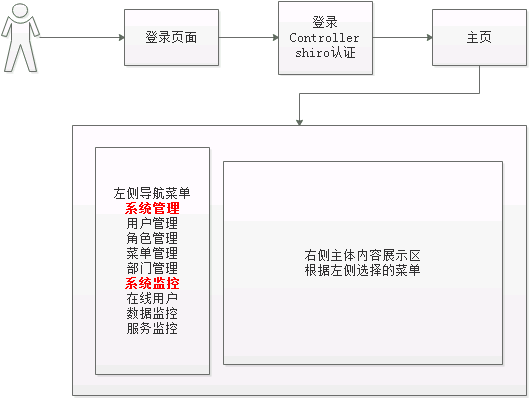
## 4、原型设计

产品经理设计产品原形

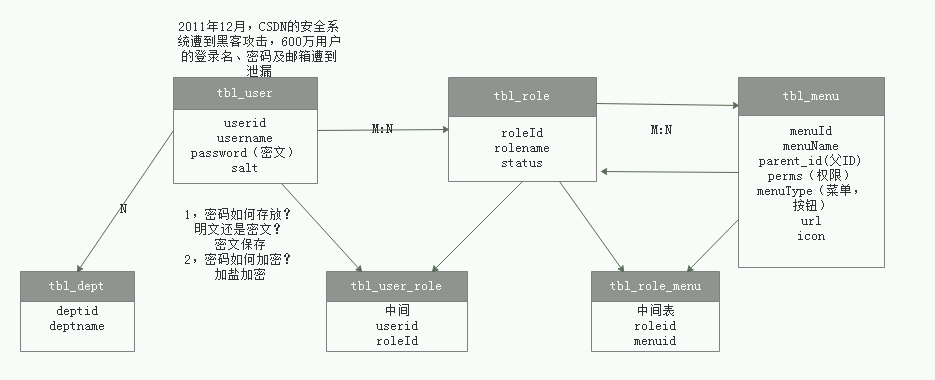
## 5、架构设计

### 5.1、整体架构设计

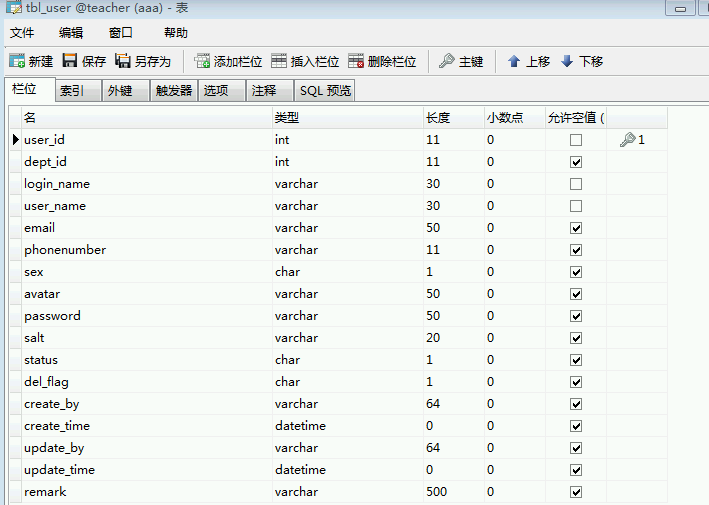




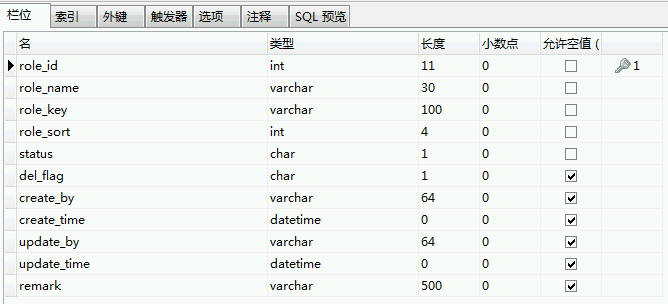
### 5.7、数据模型设计



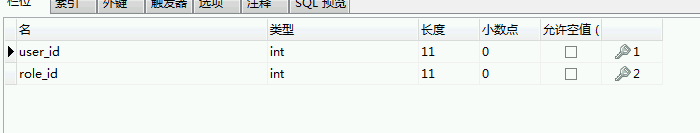
#### 5.7.1、用户表



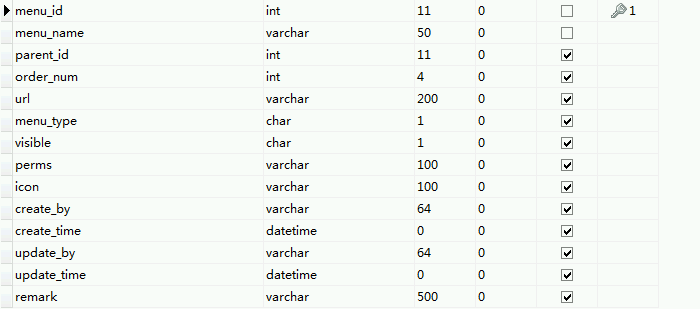
#### 5.7.2、角色表



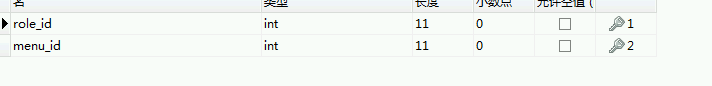
#### 5.7.3、用户角色表（中间表）



#### 5.7.4、菜单表



#### 5.7.5、角色菜单表



#### 5.7.6、部门表



## 代码开发

开发难点：

1. 菜单管理

https://fly.layui.com/jie/30625/



1. 权限菜单



<https://www.layui.com/demo/tree.html>

步骤一：根据json字符串封装java对象（gsonformat插件）

|  |
| --- |
| public class LayUiTree {   private String title;  private int id;  private String field;  private boolean checked;  private boolean spread;  private List<LayUiTree> children;   } |

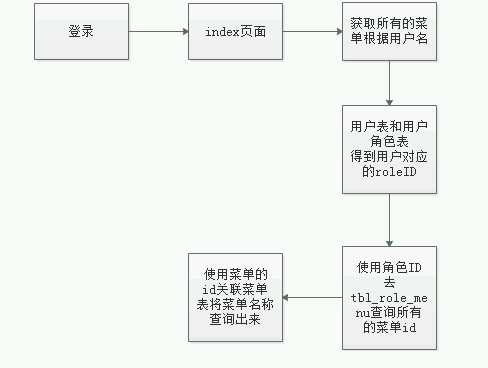
步骤二：查询所有的菜单，并且组装成LayUiTree 对象

|  |
| --- |
| package com.aaa.util;   import com.aaa.entity.LayUiTree; import com.aaa.entity.Menu;  import java.util.ArrayList; import java.util.Iterator; import java.util.List;  */\*\*  \* 权限数据处理  \*   \** ***@author*** *teacherChen  \*/* public class TreeUtils {  */\*\*  \* 根据父节点的ID获取所有子节点  \*   \** ***@param*** *list 分类表  \** ***@param*** *parentId 传入的父节点ID  \** ***@return*** *String  \*/* public static List<LayUiTree> getChildPerms(List<Menu> list, int parentId)  {  List<LayUiTree> returnList = new ArrayList<LayUiTree>();  for (Menu menu : list) {  // 一、根据传入的某个父节点ID,遍历该父节点的所有子节点  if(menu.getParentId()==parentId){  //构造tree对象  LayUiTree tree= new LayUiTree();  tree.setId(menu.getMenuId());  tree.setTitle(menu.getMenuName());  tree.setChecked(false);  //开始递归，把所有菜单和当前菜单放入  *recursionFn*(list, tree);  returnList.add(tree);  }  }  return returnList;  }   */\*\*  \* 递归列表  \*   \** ***@param*** *list  \** ***@param*** *t  \*/* private static void recursionFn(List<Menu> list, LayUiTree t)  {  // 得到t的子节点列表  List<LayUiTree> childList = *getChildList*(list, t);  t.setChildren(childList);  for (LayUiTree tChild : childList)  {  if (*hasChild*(list, tChild))  {  // 判断是否有子节点  Iterator<LayUiTree> it = childList.iterator();  while (it.hasNext())  {  LayUiTree n = it.next();  *recursionFn*(list, n);  }  }  }  }   */\*\*  \* 得到子节点列表  \*/* private static List<LayUiTree> getChildList(List<Menu> list, LayUiTree t)  {   List<LayUiTree> tlist = new ArrayList<LayUiTree>();  Iterator<Menu> it = list.iterator();  while (it.hasNext())  {  Menu n = (Menu) it.next();  if (n.getParentId() == t.getId())  {  LayUiTree tree= new LayUiTree();  tree.setId(n.getMenuId());  tree.setTitle(n.getMenuName());  tree.setChecked(false);  tlist.add(tree);  }  }  return tlist;  }   */\*\*  \* 判断是否有子节点  \*/* private static boolean hasChild(List<Menu> list, LayUiTree t)  {  int size = *getChildList*(list, t).size();  if(size>0){  return true;  }else  {  return false;  }  }  } |

1. 后台管理页面的整体布局

|  |
| --- |
| <html lang="en" xmlns:th="http://www.thymeleaf.org"> <html> <head>  <meta charset="UTF-8">  <title>AAA通用后台权限管理</title>  <script th:src="@{/js/jquery-1.10.2.js}"></script>  <script th:src="@{/layui/layui.all.js}"></script>  <link rel="stylesheet" th:href="@{/layui/css/layui.css}"> </head> <body class="layui-layout-body"> <div class="layui-layout layui-layout-admin">  <div class="layui-header">  <div class="layui-logo">AAA通用后台权限管理系统</div>  <ul class="layui-nav layui-layout-right">  <li class="layui-nav-item">  <a href="javascript:;">  <img th:src="@{/img/weixin.jpg}" class="layui-nav-img">  <span th:text="${loginName}"></span>  </a>  <dl class="layui-nav-child">  <dd><a href="">基本资料</a></dd>  <dd><a href="">安全设置</a></dd>  </dl>  </li>  <li class="layui-nav-item"><a href="logout">注销</a></li>  </ul>  </div>   <div class="layui-side layui-bg-black">  <div class="layui-side-scroll">  <!-- 左侧导航区域（可配合layui已有的垂直导航） -->  <ul class="layui-nav layui-nav-tree" lay-filter="test">  <li class="layui-nav-item layui-nav-itemed">  <a class="" href="javascript:;">所有商品</a>  <dl class="layui-nav-child">  <dd><a href="/user/toShowUser">列表一</a></dd>  <dd><a href="javascript:;">列表二</a></dd>  <dd><a href="javascript:;">列表三</a></dd>  <dd><a href="">超链接</a></dd>  </dl>  </li>  </ul>  </div>  </div>   <div class="layui-body">  <!-- 内容主体区域 -->  <iframe id="iframMain" src="" style="width: 100%;height: 100%"></iframe>  </div>   <div class="layui-footer">  <!-- 底部固定区域 -->  © layui.com - 底部固定区域  </div> </div> <script>  //JavaScript代码区域  **var** element = layui.element;  $(**function** () {  //页面加载之后执行的代码  //点击a标签的时候，修改iframe的src属性  $("dd>a").click(**function** (e) {  //阻断超链接的跳转事件  e.preventDefault();  //取得超链接对象的href属性值赋给iframe的src属性  $("#iframMain").attr("src",$(**this**).attr("href"));  })  }) </script> </body> </html> |

显示左侧的菜单栏，根据不同的用户，显示不同的菜单



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 = 'root2'

)

layui中的下拉框实现





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> |

项目优化项：

* 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;    }  } |