**通用后台权限管理系统需求规格说明书**

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

### 登录功能

#### 登录页面



#### 异常登录提示错误信息



### 用户管理

#### 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

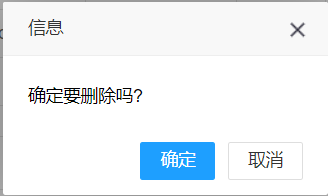
#### 添加用户，设置角色



#### 修改用户信息



#### 删除用户



### 部门管理

#### 查询部门



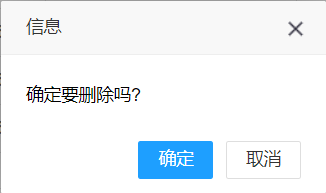
#### 添加部门



#### 修改部门，禁用部门



#### 删除部门

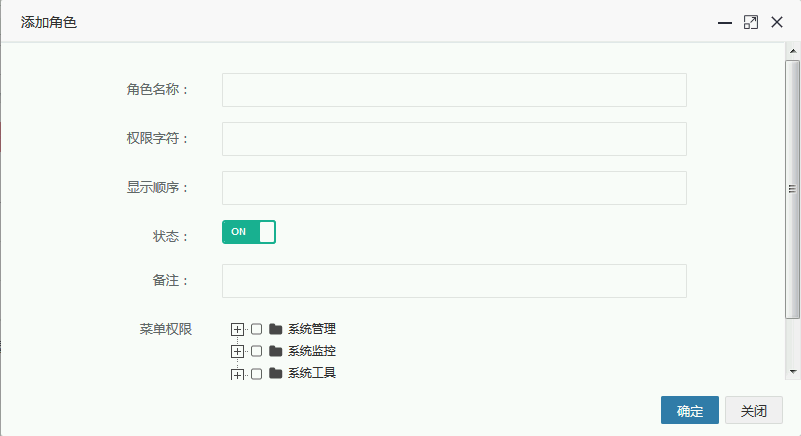


### 角色管理

#### 查询角色



#### 添加角色

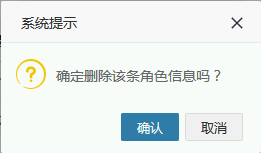


#### 修改角色





#### 删除角色



### 菜单管理

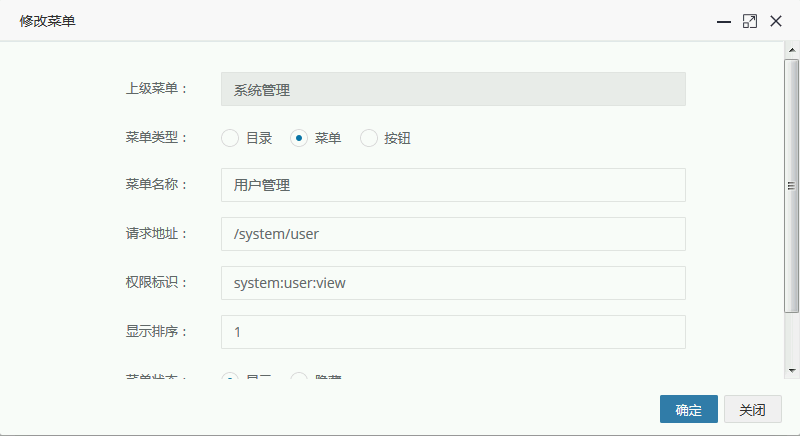
#### 查询菜单



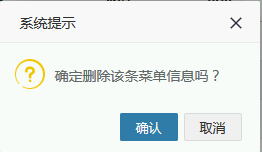
#### 添加菜单



#### 修改菜单



#### 删除菜单

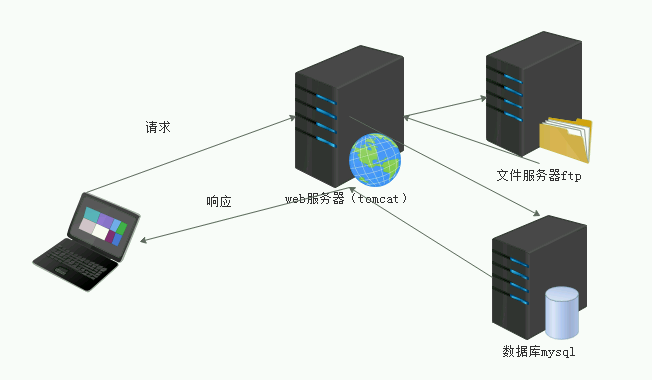


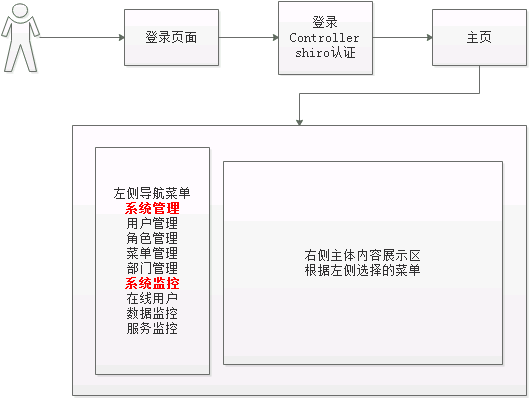
## 4、原型设计

产品经理设计产品原形

## 5、架构设计

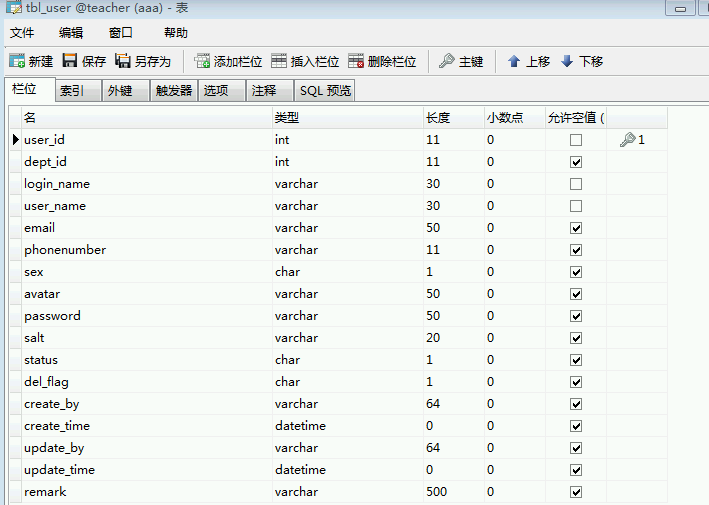
### 5.1、整体架构设计



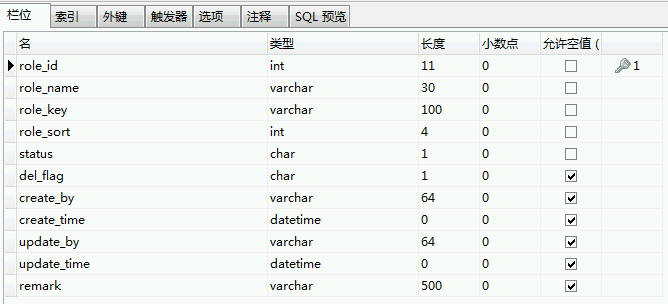


### 5.7、数据模型设计

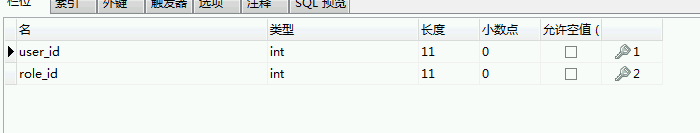
#### 5.7.1、用户表



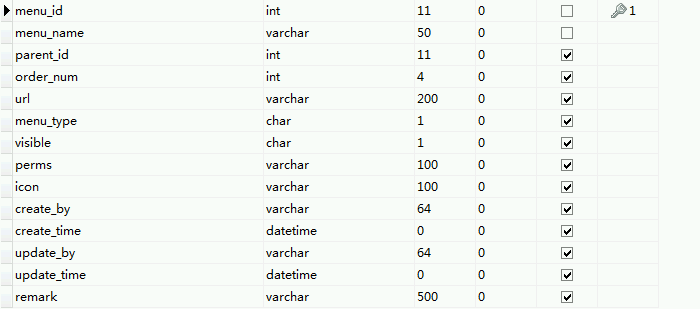
#### 5.7.2、角色表



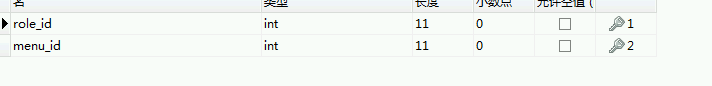
#### 5.7.3、用户角色表（中间表）



#### 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**：

### 菜单管理

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



### 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）

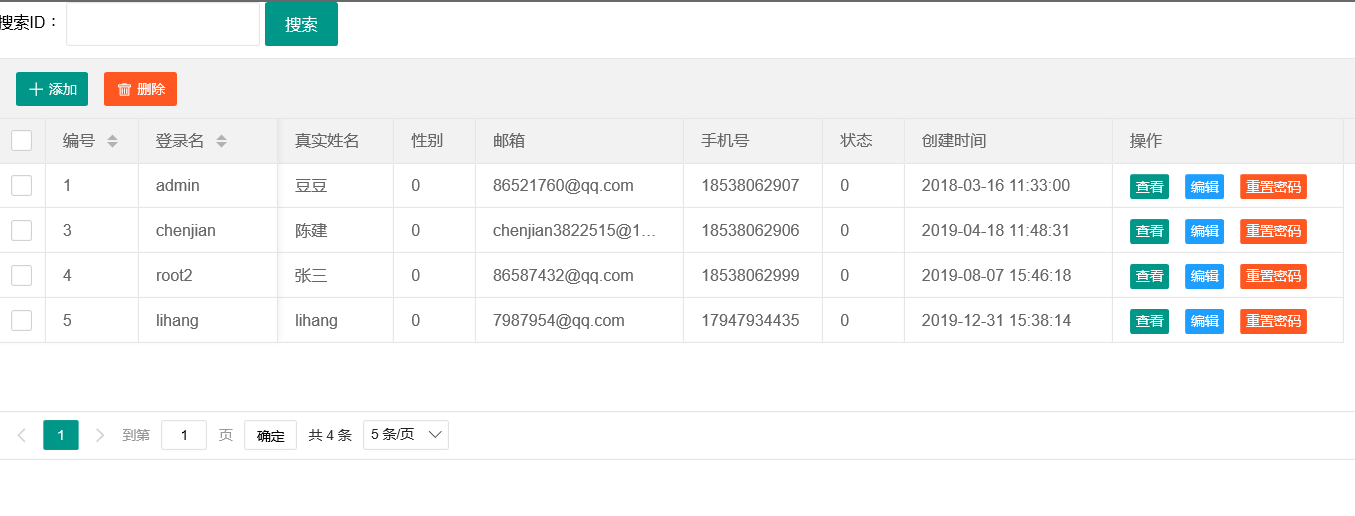
|  |
| --- |
| public class LayUiTree {   private String title;  private int id;  private String field;  private boolean checked;  private boolean spread;  private List<LayUiTree> children;   } |

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;  }  }  } |

### 小组成果

#### 用户管理



Controller包

public LayUiTable showUserLayui(int page, int limit) {  
 *//开始查询* PageInfo<User> pageInfo = userBizImpl.selectAllUser(page, limit);  
 LayUiTable layUiTable = new LayUiTable();  
 layUiTable.setCode(0);  
 layUiTable.setMsg("返回消息");  
 *//设置分页之后的返回值* layUiTable.setCount(pageInfo.getTotal());  
 layUiTable.setData(pageInfo.getList());

分页

public PageInfo<User> selectAllUser(int page, int limit) {  
 *//开始分页,第一个参数是当前第几页，第二个参数是一页显示多少行* PageHelper.*startPage*(page, limit);  
 List<User> myUserInfos = userMapper.selectAllUser();  
 *//结束分页,pageInfo封装了分页之后所有数据* PageInfo<User> pageInfo = new PageInfo(myUserInfos);  
 return pageInfo;  
  
}

密码加盐

@Override  
public int insertSelective(User record) {  
 *//次处要进行密码加盐加密* String salt = UUID.*randomUUID*().toString();  
 String message = record.getPassword();  
 String encryption = ShiroUtil.*encryptionBySalt*(salt, message);  
 record.setPassword(encryption);  
 record.setSalt(salt);  
 return userMapper.insertSelective(record);

详细代码见项目程序

#### 角色管理



Controller包

@Controller  
@RequestMapping("/role")  
public class RoleController {  
 @Autowired  
 private RoleBiz roleBizImpl;  
 @RequestMapping("/toShowRole")  
 public String toShowRole(){  
 return "role/showRole";  
 }  
  
 @RequestMapping("/showRole")  
 @ResponseBody  
 public LayUiTable showRoleLayui() {  
 *//开始查询* List<Role> roleList = roleBizImpl.selectAllRole();  
 LayUiTable layUiTable = new LayUiTable();  
 layUiTable.setCode(0);  
 layUiTable.setMsg("返回消息");  
 layUiTable.setData(roleList);  
 return layUiTable;  
 }

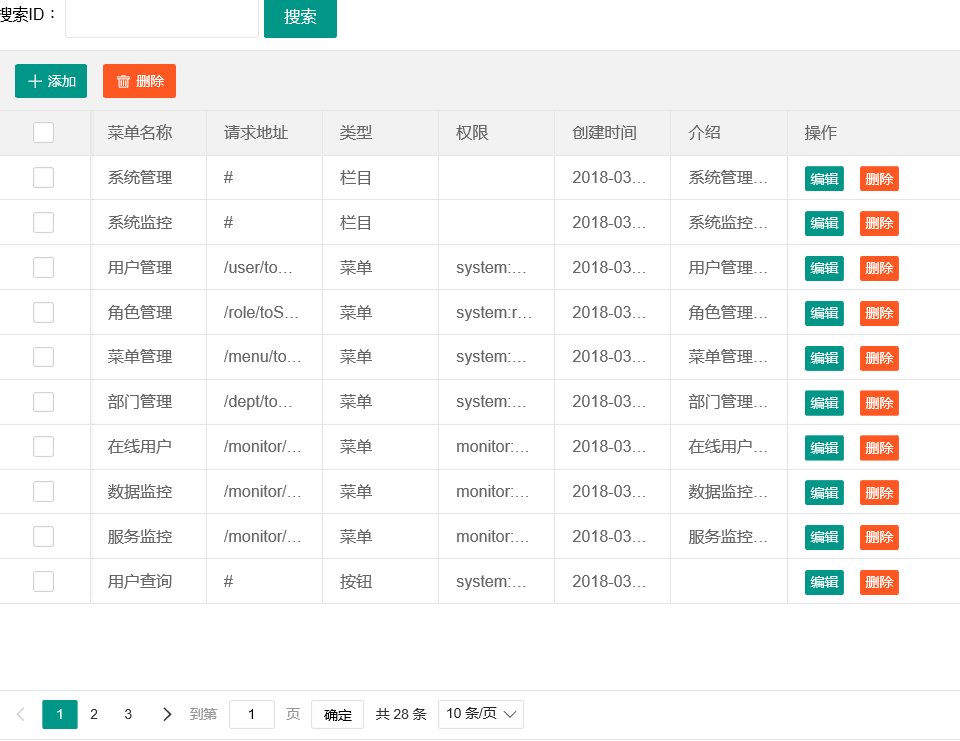
@RequestMapping("/editRole")  
@ResponseBody  
public Object editRole(Role role){  
 if(role.getStatus()==null){role.setStatus("1");}  
 System.*out*.println(role.getStatus());  
 int i = roleBizImpl.updateByPrimaryKeySelective(role);  
 Map map= new HashMap<>();  
 if(i>0){  
 map.put("code",MyConstants.*successCode*);  
 map.put("message",MyConstants.*editSuccessMsg*);  
 }else {  
 map.put("code",MyConstants.*failCode*);  
 map.put("message",MyConstants.*editFailMsg*);  
 }  
 return map;  
}

树型结构

@Override  
public List<LayUiTree> selectAllMenu() {  
 *//查询所有的菜单* List<Menu> menus = menuMapper.selectAllMenu();  
 *//并组装成tree格式的* return TreeUtils.*getChildPerms*(menus, 0);  
}  
@Override  
public List<LayUiTable> selectMenu(){return menuMapper.selectMenu();}  
  
@Override  
public List<LayUiTree> selectAllMenuByName(String loginName) {  
 *//查询所有的菜单* List<Menu> menus = menuMapper.selectAllMenuByName(loginName);  
 *//并组装成tree格式的* return TreeUtils.*getChildPerms*(menus, 0);  
}

详细代码见项目程序

#### 菜单管理



Controller包

@RequestMapping("/toShowMenuTree")  
public String toShowMenuTree(){  
 return "menu/showMenuTree2";  
}  
@RequestMapping("/showAllMenu")  
public LayUiTable showAllMenu(){  
 List<LayUiTable> list = menuBiz.selectMenu();  
 LayUiTable layUiTable = new LayUiTable();  
 layUiTable.setCode(0);  
 layUiTable.setMsg("返回消息");  
 *//设置分页之后的返回值* layUiTable.setCount(1000);  
 layUiTable.setData(list);  
 System.*out*.println(layUiTable);  
 return layUiTable;}

分页

public PageInfo<Menu> selectAllMenu(int page, int limit) {  
 *//开始分页,第一个参数是当前第几页，第二个参数是一页显示多少行* PageHelper.*startPage*(page, limit);  
 List<Menu> mymenus = menuMapper.selectAllMenu();  
 *//结束分页,pageInfo封装了分页之后所有数据* PageInfo<Menu> pageInfo= new PageInfo(mymenus);  
 return pageInfo;  
}

树型结构

@Override  
public List<LayUiTree> selectAllMenu() {  
 *//查询所有的菜单* List<Menu> menus = menuMapper.selectAllMenu();  
 *//并组装成tree格式的* return TreeUtils.*getChildPerms*(menus, 0);  
}  
@Override  
public List<LayUiTable> selectMenu(){return menuMapper.selectMenu();}  
  
@Override  
public List<LayUiTree> selectAllMenuByName(String loginName) {  
 *//查询所有的菜单* List<Menu> menus = menuMapper.selectAllMenuByName(loginName);  
 *//并组装成tree格式的* return TreeUtils.*getChildPerms*(menus, 0);  
}

详细代码见项目程序

#### 部门管理



Controller包

@Controller  
@RequestMapping("/dept")  
public class DeptController {  
 @Autowired  
 private DeptBiz deptBizImpl;  
 @RequestMapping("/toShowDept")  
 public String toShowDept(){  
 return "dept/showDept";  
 }  
  
 @RequestMapping("/showDept")  
 @ResponseBody  
 public LayUiTable showDept() {  
 *//开始查询* List<Dept> deptList = deptBizImpl.selectAllDept();  
 LayUiTable layUiTable = new LayUiTable();  
 layUiTable.setCode(0);  
 layUiTable.setMsg("返回消息");  
 layUiTable.setData(deptList);  
 *//System.out.println(deptList);* return layUiTable;  
 }

详细代码见项目程序