**单元测试说明书**

1. **测试内容**

本系统的单元测试将对于Service模块的各个具体业务类进行逐一测试，包括UserService，HouseService，NewsService，AuditService，SystemService，ReportService。每个业务模块均测试两个层次，分别为Controller层和Service层，其中Controller层的单元测试是为了测试后端方法对于前端传参的处理，Service层的测试是为了测试具体的业务逻辑。

1. **测试方法**
2. 人工静态分析。后端程序员交叉查阅对方代码，检查有无逻辑错误。
3. 人工动态测试。由后端程序员设置输入参数和预计输出，然后核对工作交由测试程序去完成。
4. **UserService模块单元测试**

**3.1 Controller层**

**3.1.1 测试描述**

UserService模块的Controller层单元测试主要是通过模拟Http请求，对UserController类的各个方法处理参数的能力做测试。

1. 测试对象

Controller层的测试对象即为各个Controller类中的具体方法，详细描述如下表所示：

|  |  |  |
| --- | --- | --- |
| 类名 | UserController | |
| 方法名 | 方法描述 | 参数说明 |
| getUserList | 获取用户列表 | 传入查询条件，包括用户名等 |
| getUserDetail | 获取用户详情 | 传入用户id |
| manageRole | 管理用户角色 | 传入用户角色id |
| userEmpowerment | 用户赋予角色 | 传入用户角色id |
| getRoleList | 获取角色列表 | 传入查询条件，包括角色名等 |
| getRoleDetail | 获取角色详情 | 传入角色id |
| addRole | 添加角色 | 传入角色信息，包括角色名等 |
| getPermissionList | 获取权限列表 | 传入查询条件，包括权限名等 |
| addPermission | 添加权限 | 传入权限信息 |
| managePermission | 管理角色权限 | 传入角色权限id |

1. 测试用例设计

Controller层的职责为处理校验参数的合法性，合法参数可进行下一步操作，不合法参数便直接返回不合法信息，为了提高用户体验，对于输入的参数Controller层均能返回HTTP200的状态码，但非法信息会夹带自定义的非法状态码。所以测试用例的设计不能只顾于返回的HTTP状态码，还要具体校验返回的信息。

**3.1.2 测试用例**

对于每个测试对象都给出多个测试用例，以覆盖所有的参数输入情况，包括正常输入、边界值输入、空指输入等传参输入，并给出期望输出，和实例代码。

**3.1.2.1 getUserList**

（1）用例说明。其中参数均为JSON格式

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 输入参数 | 期望结果 | 参数说明 |
| US\_UCON\_1 | {} | code:200 | 测试参数为null情况 |
| US\_UCON\_2 | {userId: 10000} | code:200 | 测试查询具体用户 |
| US\_UCON\_3 | {roleId:10000,orderText:”ur.CREATE\_TIME asc”} | code:200 | 测试排序 |
| US\_UCON\_4 | {name: “张三”} | code:200 | 测试查询姓名 |
| US\_UCON\_5 | {pageSize:2} | code:200 | 测试分页 |

（2）测试代码实例

@ParameterizedTest  
@ValueSource(strings = {"{\"userId\":10000}","{}","{\"roleId\":10000,\"orderText\":\"ur.CREATE\_TIME asc\"}","{\"name\":\"张三\"}","{\"pageSize\":2}"})  
void getUserList(String arg) throws Exception {  
 MvcResult mvcResult = null;  
 try {  
 mvcResult = mockMvc.perform(MockMvcRequestBuilders.*post*("/user/getUserList")  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .content(arg))  
 .andExpect(MockMvcResultMatchers.*status*().isOk())  
 .andDo(MockMvcResultHandlers.*print*())  
 .andReturn();  
 System.*out*.println(mvcResult);  
 } catch (Exception e){  
 e.printStackTrace();  
 }  
}

**3.1.2.2 getUserDetail**

（1）用例说明。其中参数均为JSON格式

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 输入参数 | 期望结果 | 参数说明 |
| US\_UCON\_6 | {userId:10000} | code:200 | 测试正确传参 |
| US\_UCON\_7 | {} | code:500 | 测试传空 |
| US\_UCON\_8 | {roleId:10000} | code:500 | 测试不穿userId |

（2） 测试代码部分实例

@ParameterizedTest  
@ValueSource(strings = {"{}","{\"roleId\":10000,\"orderText\":\"ur.CREATE\_TIME asc\"}","{\"name\":\"张三\"}"})  
void getUserDetail2(String arg){  
 MvcResult mvcResult = null;  
 try {  
 mvcResult = mockMvc.perform(MockMvcRequestBuilders.*post*("/user/getUserDetail")  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .content(arg))  
 .andExpect(MockMvcResultMatchers.*status*().isOk())  
 .andDo(MockMvcResultHandlers.*print*())  
 .andReturn();  
 Result<Map<String, Object>> result = JSONObject.*parseObject*(mvcResult.toString(), Result.class);  
 int code = result.getCode();  
 Assertions.*assertEquals*(500, code);  
 } catch (Exception e){  
 e.printStackTrace();  
 }  
}

**3.1.2.3 manageRole**

（1）用例说明。其中参数均为JSON格式

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 输入参数 | 期望结果 | 参数说明 |
| US\_UCON\_9 | {userRoleId:10000, lockedMark:”NO”} | code:200 | 测试正确传参 |
| US\_UCON\_10 | {userRoleId:10000} | code:500 | 不传lockedMark |
| US\_UCON\_11 | {lockedMark:”NO”} | code:500 | 不穿userRoleId |
| US\_UCON\_12 | {userRoleId:10000, lockedMark:”DD”} | code:500 | 传入非法lockedMark |
| US\_UCON\_13 | {} | code:500 | 传空 |

（2） 测试代码部分实例

@ParameterizedTest  
@ValueSource(strings = {"{\"userRoleId\":10000}","{\"lockedMark\":\"NO\"}","{\"userId\":10000,\"lockedMark\":\"DD\"}"})  
void manageRole2(String arg){  
 MvcResult mvcResult = null;  
 try {  
 mvcResult = mockMvc.perform(MockMvcRequestBuilders.*post*("/user/manageRole")  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .content(arg))  
 .andExpect(MockMvcResultMatchers.*status*().isOk())  
 .andDo(MockMvcResultHandlers.*print*())  
 .andReturn();  
 Result<Map<String, Object>> result = JSONObject.*parseObject*(mvcResult.toString(), Result.class);  
 int code = result.getCode();  
 Assertions.*assertEquals*(500, code);  
 } catch (Exception e){  
 e.printStackTrace();  
 }  
}

**3.1.2.4 userEmpowerment**

（1）用例说明。其中参数均为JSON格式

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 输入参数 | 期望结果 | 参数说明 |
| US\_UCON\_14 | {roleId:10000,userId:10002} | code:200 | 测试正确传参 |
| US\_UCON\_15 | {roleId:10000} | code:500 | 不传userId |
| US\_UCON\_16 | {userId:10002} | code:500 | 不传roleId |

（2） 测试代码部分实例

@ParameterizedTest  
@ValueSource(strings = {"{\"roleId\":\"10000","{\"userId\":10002}"})  
void userEmpowerment2(String arg){  
 MvcResult mvcResult = null;  
 try {  
 mvcResult = mockMvc.perform(MockMvcRequestBuilders.*post*("/user/userEmpowerment")  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .content(arg))  
 .andExpect(MockMvcResultMatchers.*status*().isOk())  
 .andDo(MockMvcResultHandlers.*print*())  
 .andReturn();  
 Result<Map<String, Object>> result = JSONObject.*parseObject*(mvcResult.toString(), Result.class);  
 int code = result.getCode();  
 Assertions.*assertEquals*(500, code);  
 } catch (Exception e){  
 e.printStackTrace();  
 }  
}

**3.1.2.5 getRoleList**

（1）用例说明。其中参数均为JSON格式

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 输入参数 | 期望结果 | 参数说明 |
| US\_UCON\_17 | {pageSize:5} | code:200 | 测试分页 |
| US\_UCON\_18 | {} | code:200 | 传空 |
| US\_UCON\_19 | {roleId:10004} | code:200 | 查询具体角色 |
| US\_UCON\_20 | {roleName:”管理”} | code:200 | 测试模糊查询 |

（2） 测试代码部分实例

@ParameterizedTest  
@ValueSource(strings = {"{\"pageSize\":5}","{}","{\"roleId\":10004}","{\"pageNo\":1,\"roleName\":\"管理\"}"})  
void getRoleList(String arg){  
 MvcResult mvcResult = null;  
 try {  
 mvcResult = mockMvc.perform(MockMvcRequestBuilders.*post*("/user/getRoleList")  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .content(arg))  
 .andExpect(MockMvcResultMatchers.*status*().isOk())  
 .andDo(MockMvcResultHandlers.*print*())  
 .andReturn();  
 } catch (Exception e){  
 e.printStackTrace();  
 }  
}

**3.1.2.6 getRoleDetail**

（1）用例说明。其中参数均为JSON格式

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 输入参数 | 期望结果 | 参数说明 |
| US\_UCON\_21 | {roleId:10000} | code:200 | 测试正确用例 |
| US\_UCON\_22 | {} | code:500 | 传空 |
| US\_UCON\_23 | {roleName:”管理员”} | code:500 | roleId传空 |

（2） 测试代码部分实例

@ParameterizedTest  
@ValueSource(strings = {"{\"roleName\":\"管理员\"}"})  
void getRoleDetail2(String arg){  
 MvcResult mvcResult = null;  
 try {  
 mvcResult = mockMvc.perform(MockMvcRequestBuilders.*post*("/user/getRoleDetail")  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .content(arg))  
 .andExpect(MockMvcResultMatchers.*status*().isOk())  
 .andDo(MockMvcResultHandlers.*print*())  
 .andReturn();  
 Result<Map<String, Object>> result = JSONObject.*parseObject*(mvcResult.toString(), Result.class);  
 int code = result.getCode();  
 Assertions.*assertEquals*(500, code);  
 } catch (Exception e){  
 e.printStackTrace();  
 }  
}

**3.1.2.7 addRole**

（1）用例说明。其中参数均为JSON格式

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 输入参数 | 期望结果 | 参数说明 |
| US\_UCON\_24 | {roleName：“测试”, permissions:[10000,10001]} | code:200 | 测试正确用例 |
| US\_UCON\_25 | {} | code:500 | 传空 |
| US\_UCON\_26 | {roleName:”测试”} | code:200 | 权限传空 |
| US\_UCON\_27 | {permissions:[10000,10001]} | code:500 | roleName传空 |

（2） 测试代码部分实例

@ParameterizedTest  
@ValueSource(strings = {"{\"roleId\":10000,\"permissions\":[10000]}"})  
void addRole2(String arg){  
 MvcResult mvcResult = null;  
 try {  
 mvcResult = mockMvc.perform(MockMvcRequestBuilders.*post*("/user/addRole")  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .content(arg))  
 .andExpect(MockMvcResultMatchers.*status*().isOk())  
 .andDo(MockMvcResultHandlers.*print*())  
 .andReturn();  
 Result<Map<String, Object>> result = JSONObject.*parseObject*(mvcResult.toString(), Result.class);  
 int code = result.getCode();  
 Assertions.*assertEquals*(500, code);  
 } catch (Exception e){  
 e.printStackTrace();  
 }  
}

**3.1.2.8 getPermissionList**

（1）用例说明。其中参数均为JSON格式

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 输入参数 | 期望结果 | 参数说明 |
| US\_UCON\_28 | {pageSize:10} | code:200 | 测试分页 |
| US\_UCON\_29 | {} | code:200 | 传空 |
| US\_UCON\_30 | {permissionId:10003} | code:200 | 查询具体权限 |
| US\_UCON\_31 | {url:”/user/test”} | code:200 | 查询url |

（2） 测试代码部分实例

@ParameterizedTest  
@ValueSource(strings = {"{}","{\"pageSize\":10, \"pageNo\":1}","{\"permissionId\":10003}"})  
void getPermissionList(String arg){  
 MvcResult mvcResult = null;  
 try {  
 mvcResult = mockMvc.perform(MockMvcRequestBuilders.*post*("/user/getPermissionList")  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .content(arg))  
 .andExpect(MockMvcResultMatchers.*status*().isOk())  
 .andDo(MockMvcResultHandlers.*print*())  
 .andReturn();  
 } catch (Exception e){  
 e.printStackTrace();  
 }  
}

**3.1.2.9 addPermission**

（1）用例说明。其中参数均为JSON格式

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 输入参数 | 期望结果 | 参数说明 |
| US\_UCON\_32 | {} | code:500 | 传空 |
| US\_UCON\_33 | {permissionName:”测试”} | code:500 | 不传url |
| US\_UCON\_34 | {url:”/user/test”} | code:500 | 不传权限名 |
| US\_UCON\_35 | {permissionName:”测试”,url:”/user/test”} | code:200 | 正确用例 |

（2） 测试代码部分实例

@ParameterizedTest  
@ValueSource(strings = {"{\"permissionName\":\"测试\", \"url\":\"\\user\\test\"}"})  
void addPermission1(String arg){  
 MvcResult mvcResult = null;  
 try {  
 mvcResult = mockMvc.perform(MockMvcRequestBuilders.*post*("/user/addPermission")  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .content(arg))  
 .andExpect(MockMvcResultMatchers.*status*().isOk())  
 .andDo(MockMvcResultHandlers.*print*())  
 .andReturn();  
 } catch (Exception e){  
 e.printStackTrace();  
 }  
}

**3.1.2.10 managePermission**

（1）用例说明。其中参数均为JSON格式

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 输入参数 | 期望结果 | 参数说明 |
| US\_UCON\_36 | {} | code:500 | 传空 |
| US\_UCON\_37 | {rolePermissionId:10000} | code:500 | 不传lockedMark |
| US\_UCON\_38 | {lockedMark:”NO”} | code:500 | 不传角色权限id |
| US\_UCON\_39 | {rolePermissionId:10000,lockedMaek:”NO”} | code:200 | 正确用例 |

（2） 测试代码部分实例

@ParameterizedTest  
@ValueSource(strings = {"{\"permissionId\":10000, \"lockedMark\":\"NO\"}"})  
void managePermission1(String arg){  
 MvcResult mvcResult = null;  
 try {  
 mvcResult = mockMvc.perform(MockMvcRequestBuilders.*post*("/user/manageRole")  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .content(arg))  
 .andExpect(MockMvcResultMatchers.*status*().isOk())  
 .andDo(MockMvcResultHandlers.*print*())  
 .andReturn();  
 } catch (Exception e){  
 e.printStackTrace();  
 }  
}

**3.2 Service层**

**3.2.1 测试描述**

UserService模块的Service层单元测试主要是通过传入测试人员设置好的参数，查看Service层各个函数的具体输出是否符合逻辑，比如返回的数量，返回的属性，是否会抛出异常等等。涉及到的Service类包括UserService、RoleService、PermissionService。

（1）测试对象

Service层的测试对象即为各个Service类中的具体方法，详细描述如下表所示：

|  |  |  |
| --- | --- | --- |
| 类名 | UserService | |
| 方法名 | 方法描述 | 参数说明 |
| selectUserList | 获取用户列表 | 传入查询条件，包括用户名等 |
| getUserDetail | 获取用户详情 | 传入用户id |
| manageRole | 管理用户角色 | 传入用户角色id |
| userEmpowerment | 用户赋予角色 | 传入用户角色id |
| getEmails | 获取用户邮箱 | 传入用户id |
| getModuleList | 获取菜单栏列表 | 传入用户id |
| 类名 | RoleService | |
| getRoleList | 获取角色列表 | 传入查询条件，包括角色名等 |
| getRoleDetail | 获取角色详情 | 传入角色id |
| addRole | 添加角色 | 传入角色信息，包括角色名等 |
| managePermission | 管理角色权限 | 传入角色权限id |
| 类名 | PermissionService | |
| getPermissionList | 获取权限列表 | 传入查询条件，包括权限名等 |
| addPermission | 添加权限 | 传入权限信息 |

（2）测试用例设计

Service层的职责在于处理具体的业务逻辑。其中UserService模块中的Service层主要是为了后台管理服务的，主要业务逻辑为数据库的增删改查，而对于算法没有什么要求，所以对于该Service层的单元测试主要是为了校验其返回值的主要属性，包括数量、主键是否正确，至于是否符合业务，需要到接口测试才能体现。

**3.2.2 测试用例**

对于每个测试对象都给出多个测试用例，以覆盖所有的参数输入情况，包括正常输入、边界值输入、空指输入等传参输入，并给出期望输出，和实例代码。

**3.2.2.1 selectUserList**

（1）用例说明。其中参数均为UserForm的封装属性。

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 输入参数 | 期望结果 | 参数说明 |
| US\_USER\_1 | {} | 非空 | 传空 |
| US\_USER\_2 | userId:10000 | total:1,userId:10000 | 查询具体用户 |
| US\_USER\_3 | pageNo:1,pageSize:2 | data.size:2 | 测试分页 |
| US\_UCON\_4 | roleId:10000 | roleId:10000 | 测试具体查询条件 |

（2） 测试代码部分实例

@Test  
void selectUserList(){  
 UserForm userForm = new UserForm();  
 Assertions.*assertNotNull*(userService.selectUserList(userForm).get("data"));  
 userForm.setUserId(10000l);  
 Assertions.*assertEquals*(1, (int)userService.selectUserList(userForm).get("total"));  
 userForm = new UserForm();  
 userForm.setPageNo(1);  
 userForm.setPageSize(2);  
 Assertions.*assertEquals*(2, ((List<UserVo>)userService.selectUserList(userForm).get("data")).size());  
}

**3.2.2.2 getUserDetail**

（1）用例说明。其中参数均为UserForm的封装属性。

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 输入参数 | 期望结果 | 参数说明 |
| US\_USER\_5 | userId:10000 | userId:10000 | 测试正确用例 |
| US\_USER\_6 |  | flag:false | 传空 |

（2） 测试代码部分实例

@Test  
void getUserDetail(){  
 UserForm userForm = new UserForm();  
 userForm.setUserId(10000l);  
 Map<String, Object> result = userService.getUserDetail(userForm);  
 UserVo userVo = (UserVo) result.get("data");  
 Assertions.*assertEquals*(10000l, userVo.getUserId());  
}

**3.2.2.3 manageRole**

（1）用例说明。其中参数均为UserForm的封装属性。

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 输入参数 | 期望结果 | 参数说明 |
| US\_USER\_7 | userRoleId:1,lockedMark:”NO” | flag:true | 正确用例 |
| US\_USER\_8 |  | flag:flase | 传空 |
| US\_USER\_9 | userRoleId:1 | flag:false | lockedMark传空 |
| US\_USER\_10 | lockedMark:”NO” | flag:false | userRoleId传空 |

（2） 测试代码部分实例

@Test  
void manageRole(){  
 UserForm userForm = new UserForm();  
 userForm.setUserRoleId(2l);  
 userForm.setLockedMark("NO");  
 Boolean flag = (Boolean) userService.manageRole(userForm).get("flag");  
 Assertions.*assertEquals*(true, flag);  
 userForm = new UserForm();  
 userForm.setUserId(2l);  
 flag = (Boolean) userService.manageRole(userForm).get("flag");  
 Assertions.*assertEquals*(false, flag);  
 userForm = new UserForm();  
 userForm.setLockedMark("NO");  
 flag = (Boolean) userService.manageRole(userForm).get("flag");  
 Assertions.*assertEquals*(false, flag);  
}

**3.2.2.4 userEmpowerment**

（1）用例说明。其中参数均为UserForm的封装属性。

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 输入参数 | 期望结果 | 参数说明 |
| US\_USER\_11 | userId:10002,roleId:10001 | flag:true | 正确用例 |
| US\_USER\_12 | userId:10000,roleId:10000 | flag:flase | 重复赋予角色 |
| US\_USER\_13 |  | flag:false | 传空 |
| US\_USER\_14 | userId:10002 | flag:false | roleId传空 |
| US\_USER\_15 | roleId:10003 | flag:false | userId传空 |

（2） 测试代码部分实例

@Test  
void userEmpowerment(){  
 UserForm userForm = new UserForm();  
 userForm.setUserId(10002l);  
 userForm.setRoleId(10000);  
 Boolean flag = (Boolean) userService.userEmpowerment(userForm).get("flag");  
 Assertions.*assertEquals*(true, flag);  
 userForm = new UserForm();  
 userForm.setUserId(10002l);  
 flag = (Boolean) userService.userEmpowerment(userForm).get("flag");  
 Assertions.*assertEquals*(false, flag);  
 userForm = new UserForm();  
 userForm.setRoleId(10000);  
 flag = (Boolean) userService.userEmpowerment(userForm).get("flag");  
 Assertions.*assertEquals*(false, flag);  
 userForm = new UserForm();  
 userForm.setUserId(10002l);  
 userForm.setRoleId(10001);  
 flag = (Boolean) userService.userEmpowerment(userForm).get("flag");  
 Assertions.*assertEquals*(false, flag);  
}

**3.2.2.5 getEmails**

（1）用例说明。

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 输入参数 | 期望结果 | 参数说明 |
| US\_USER\_16 | userIds:[10000,10001] | 非空 | 正确用例 |
| US\_USER\_17 |  | null | 传空 |

（2） 测试代码部分实例

@Test  
void getEmails(){  
 List<Long> userIds = new ArrayList<>();  
 userIds.add(10000l);  
 userIds.add(10001l);  
 List<String> emails = (List<String>) userService.getEmail(userIds).get("data");  
 Assertions.*assertNotNull*(emails);  
 userIds = new ArrayList<>();  
 emails = (List<String>) userService.getEmail(userIds).get("data");  
 Assertions.*assertEquals*(null, emails);  
}

**3.2.2.6 getModuleList**

（1）用例说明。其中参数均为UserForm的封装属性。

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 输入参数 | 期望结果 | 参数说明 |
| US\_USER\_18 | userId:10000 | 非空 | 正确用例 |
| US\_USER\_19 |  | null | 传空 |

（2） 测试代码部分实例

@Test  
void getModuleList(){  
 Long userId = 10000l;  
 Assertions.*assertNotNull*(userService.getModuleList(userId));  
}

**3.2.2.7 getRoleList**

（1）用例说明。其中参数均为RoleForm的封装属性。

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 输入参数 | 期望结果 | 参数说明 |
| US\_USER\_20 |  | 非空 | 传空 |
| US\_USER\_21 | roleId:10000 | total:1,roleId:10000 | 查询某个角色 |
| US\_USER\_22 | pageSize:2,pageNo:1 | data.size:2 | 测试分页 |

（2） 测试代码部分实例

@Test  
void getRoleList(){  
 RoleForm roleForm = new RoleForm();  
 List<RoleVo> list = (List<RoleVo>) roleService.getRoleList(roleForm).get("data");  
 Assertions.*assertNotNull*(list);  
 roleForm.setRoleId(10000);  
 Assertions.*assertEquals*(1, (int)roleService.getRoleList(roleForm).get("total"));  
 roleForm = new RoleForm();  
 roleForm.setPageSize(2);  
 Assertions.*assertEquals*(2, ((List<RoleVo>) roleService.getRoleList(roleForm)).size());  
}

**3.2.2.8 getRoleDetail**

（1）用例说明。其中参数均为RoleForm的封装属性。

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 输入参数 | 期望结果 | 参数说明 |
| US\_USER\_23 |  | flag:false | 传空 |
| US\_USER\_24 | roleId:10000 | roleId:10000 | 正确用例 |

（2） 测试代码部分实例

@Test  
void getRoleDetail(){  
 RoleForm roleForm = new RoleForm();  
 roleForm.setRoleId(10001);  
 Map<String, Object> result = roleService.getRoleDetail(roleForm);  
 RoleVo roleVo = (RoleVo) result.get("data");  
 Assertions.*assertEquals*(10001, roleVo.getRoleId());  
}

**3.2.2.9 managePermission**

（1）用例说明。其中参数均为RoleForm的封装属性。

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 输入参数 | 期望结果 | 参数说明 |
| US\_USER\_25 |  | flag:false | 传空 |
| US\_USER\_26 | rolePermissionId:10000 | flag:false | lockedMark传空 |
| US\_USER\_27 | lockedMark:”NO” | flag:false | rolePermissionId传空 |
| US\_USER\_28 | rolePermissionId:10000,lockedMark:”NO” | flag:true | 正确用例 |

（2） 测试代码部分实例

@Test  
void managePermission(){  
 RoleForm roleForm = new RoleForm();  
 roleForm.setRolePermissionId(10003);  
 roleForm.setLockedMark("NO");  
 Boolean flag = (Boolean) roleService.managePermission(roleForm).get("flag");  
 Assertions.*assertEquals*(true, flag);  
 roleForm = new RoleForm();  
 roleForm.setLockedMark("NO");  
 flag = (Boolean) roleService.managePermission(roleForm).get("flag");  
 Assertions.*assertEquals*(false, flag);  
 roleForm = new RoleForm();  
 roleForm.setRolePermissionId(10003);  
 flag = (Boolean) roleService.managePermission(roleForm).get("flag");  
 Assertions.*assertEquals*(false, flag);  
}

**3.2.2.10 addRole**

（1）用例说明。其中参数均为RoleForm的封装属性。

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 输入参数 | 期望结果 | 参数说明 |
| US\_USER\_29 |  | flag:false | 传空 |
| US\_USER\_30 | roleName:”测试” | flag:true,roleId非空 | permissions传空 |
| US\_USER\_31 | roleName:”测试”, permissions:[10000,10001] | flag:true,roleId非空 | 正确用例 |

（2） 测试代码部分实例

@Test  
void addRole(){  
 RoleForm roleForm = new RoleForm();  
 Boolean flag = (Boolean) roleService.addRole(roleForm).get("flag");  
 Assertions.*assertEquals*(false, flag);  
 roleForm.setRoleName("测试");  
 Map<String, Object> result = roleService.addRole(roleForm);  
 flag = (Boolean) result.get("flag");  
 Assertions.*assertEquals*(true, flag);  
 Assertions.*assertNotNull*(result.get("roleId"));  
 roleForm = new RoleForm();  
 roleForm.setRoleName("测试");  
 List<Integer> list = new ArrayList<>();  
 list.add(10000);  
 list.add(10001);  
 roleForm.setPermissions(list);  
 flag = (Boolean) roleService.addRole(roleForm).get("flag");  
 Assertions.*assertEquals*(true, flag);  
}

**3.2.2.11 getPermissionList**

（1）用例说明。其中参数均为PermissionForm的封装属性。

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 输入参数 | 期望结果 | 参数说明 |
| US\_USER\_32 |  | 非空 | 传空 |
| US\_USER\_33 | pageSize:3,pageNo:2 | data.size:3,total != 3 | 测试分页 |
| US\_USER\_34 | permissionId:10000 | total:1,permissionId:10000 | 查询具体权限 |

（2） 测试代码部分实例

@Test  
void getPermissionList(){  
 PermissionForm permissionForm = new PermissionForm();  
 Map<String, Object> result = permissionService.getPermissionList(permissionForm);  
 List<PermissionVo> list = (List<PermissionVo>) result.get("data");  
 Assertions.*assertNotNull*(list);  
 permissionForm.setPageSize(3);  
 result = permissionService.getPermissionList(permissionForm);  
 list = (List<PermissionVo>) result.get("data");  
 Assertions.*assertEquals*(3, list.size());  
 permissionForm.setPermissionId(10000);  
 Assertions.*assertEquals*(1, (int) permissionService.getPermissionList(permissionForm).get("total"));  
}

**3.2.2.12 addPermission**

（1）用例说明。其中参数均为PermissionForm的封装属性。

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 输入参数 | 期望结果 | 参数说明 |
| US\_USER\_35 |  | flag:false | 传空 |
| US\_USER\_36 | permissonName:”测试” | flag:false | url为空 |
| US\_USER\_37 | url:”/user/test” | flag:false | 权限名为空 |
| US\_USER\_38 | permissionName:”测试”,url:”/user/test” | flag:true,permissionId非空 | 正确用例 |

（2） 测试代码部分实例

@Test  
void addPermission(){  
 PermissionForm permissionForm = new PermissionForm();  
 permissionForm.setPermissionName("测试");  
 permissionForm.setUrl("/user/test");  
 Boolean flag = (Boolean) permissionService.addPermission(permissionForm).get("flag");  
 Assertions.*assertEquals*(true, flag);  
 permissionForm = new PermissionForm();  
 permissionForm.setUrl("/user/test");  
 flag = (Boolean) permissionService.addPermission(permissionForm).get("flag");  
 Assertions.*assertEquals*(false, flag);  
 permissionForm = new PermissionForm();  
 permissionForm.setPermissionName("测试");  
 flag = (Boolean) permissionService.addPermission(permissionForm).get("flag");  
 Assertions.*assertEquals*(false, flag);  
}

1. **ReportService模块单元测试**

**4.1 Controller层**

**4.1.1 测试描述**

ReportService模块的Controller层单元测试主要是通过模拟Http请求，对UserController类的各个方法处理参数的能力做测试。

（1）测试对象

Controller层的测试对象即为各个Controller类中的具体方法，详细描述如下表所示：

|  |  |  |
| --- | --- | --- |
| 类名 | ReportController | |
| 方法名 | 方法描述 | 参数说明 |
| addReport | 添加举报记录 | 传入举报基本信息 |
| getReportList | 获取举报记录 | 传入查询信息比如类别id等 |
| getReportDetail | 获取举报详情 | 传入举报id |
| doAudit | 举报审核 | 传入举报和审核信息 |

（2）测试用例设计

Controller层的职责为处理校验参数的合法性，合法参数可进行下一步操作，不合法参数便直接返回不合法信息，为了提高用户体验，对于输入的参数Controller层均能返回HTTP200的状态码，但非法信息会夹带自定义的非法状态码。所以测试用例的设计不能只顾于返回的HTTP状态码，还要具体校验返回的信息。

**4.1.2 测试用例**

对于每个测试对象都给出多个测试用例，以覆盖所有的参数输入情况，包括正常输入、边界值输入、空指输入等传参输入，并给出期望输出，和实例代码。

**4.1.2.1 selectReportList**

（1）用例说明。其中参数均为ReportForm的封装属性。

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 输入参数 | 期望结果 | 参数说明 |
| RS\_RCON\_1 | {} | code:200 | 传空 |
| RS\_RCON\_2 | {reportId:10000} | code:200 | 查询具体举报 |
| RS\_RCON\_3 | {pageNo:1,pageSize:2} | code:200 | 测试分页 |
| RS\_RCON\_4 | {orderText:”CREATE\_TIME desc”} | code:200 | 测试排序 |

（2） 测试代码部分实例

@ParameterizedTest  
@ValueSource(strings = {"{\"objtypeId\":10000}"})  
void testGetReportList(String arg) throws Exception {  
 // Setup  
 *when*(mockReportService.getReportList(new ReportForm())).thenReturn(new HashMap<>());  
  
 // Run the test  
 final MockHttpServletResponse response = mockMvc.perform(*get*("/report/getReportList")  
 .content(arg).contentType(MediaType.*APPLICATION\_JSON*)  
 .accept(MediaType.*APPLICATION\_JSON*))  
 .andReturn().getResponse();  
  
 // Verify the results  
 *assertThat*(response.getStatus()).isEqualTo(HttpStatus.*OK*.value());  
}

**4.1.2.2 addReport**

（1）用例说明。其中参数均为ReportForm的封装属性。

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 输入参数 | 期望结果 | 参数说明 |
| RS\_RCON\_5 | {} | code:500 | 传空 |
| RS\_RCON\_6 | {defense:10000, reason:”xx”} | code:500 | 举报类型为空 |
| RS\_RCON\_7 | {defense:10000, objtypeId:10000} | code:500 | 举报理由为空 |
| RS\_RCON\_8 | {reason:”xx”, objtypeId:10000} | code:500 | 被举报对象为空 |
| RS\_RCON\_9 | {defense:10000,reason:”xx”,objtypeId:10000} | code:200 | 正确用例 |

（2） 测试代码部分实例

@Test  
void testAddReport() throws Exception {  
 // Setup  
 *when*(mockReportService.addReport(new ReportForm())).thenReturn(new HashMap<>());  
  
 // Run the test  
 final MockHttpServletResponse response = mockMvc.perform(*post*("/report/addReport")  
 .content("{}").contentType(MediaType.*APPLICATION\_JSON*)  
 .accept(MediaType.*APPLICATION\_JSON*))  
 .andReturn().getResponse();  
  
 // Verify the results  
 *assertThat*(response.getStatus()).isEqualTo(HttpStatus.*OK*.value());  
 *assertThat*(response.getContentAsString()).isEqualTo("expectedResponse");  
}

**4.1.2.3 getReportDetail**

（1）用例说明。其中参数均为ReportForm的封装属性。

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 输入参数 | 期望结果 | 参数说明 |
| RS\_RCON\_10 | {} | code:500 | 传空 |
| RS\_RCON\_11 | {reportId:10000} | code:200 | 正确用例 |

（2） 测试代码部分实例

@ParameterizedTest  
@ValueSource(strings = {"{\"objtypeId\":10000}"})  
void testGetReportDetail(String arg) throws Exception {  
 // Setup  
 *when*(mockReportService.getReportDetail(new ReportForm())).thenReturn(new HashMap<>());  
  
 // Run the test  
 final MockHttpServletResponse response = mockMvc.perform(*get*("/report/getReportDetail")  
 .content(arg).contentType(MediaType.*APPLICATION\_JSON*)  
 .accept(MediaType.*APPLICATION\_JSON*))  
 .andReturn().getResponse();  
  
 // Verify the results  
 *assertThat*(response.getStatus()).isEqualTo(HttpStatus.*OK*.value());  
 *assertThat*(response.getContentAsString()).isEqualTo("expectedResponse");  
}

**4.1.2.4 doAudit**

（1）用例说明。其中参数均为ReportForm的封装属性。

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 输入参数 | 期望结果 | 参数说明 |
| RS\_RCON\_12 | {objtypeId:10000, defense:10000} | code:200 | 正确用例 |
| RS\_RCON\_13 | {} | code:500 | 传空 |
| RS\_RCON\_14 | {objtypeId:10000} | code:500 | defense传空 |
| RS\_RCON\_15 | {defense:10000} | code:500 | objtypeId传空 |

（2） 测试代码部分实例

@Test  
void testDoAudit() throws Exception {  
 // Setup  
 *when*(mockReportService.doAudit(new ReportForm())).thenReturn(false);  
 // Run the test  
 final MockHttpServletResponse response = mockMvc.perform(*post*("/report/doAudit")  
 .content("content").contentType(MediaType.*APPLICATION\_JSON*)  
 .accept(MediaType.*APPLICATION\_JSON*))  
 .andReturn().getResponse();  
 // Verify the results  
 *assertThat*(response.getStatus()).isEqualTo(HttpStatus.*OK*.value());  
 *assertThat*(response.getContentAsString()).isEqualTo("expectedResponse");  
}

**4.2 Service层**

**4.2.1 测试描述**

ReportService模块的Service层单元测试主要是通过传入测试人员设置好的参数，查看Service层各个函数的具体输出是否符合逻辑，比如返回的数量，返回的属性，是否会抛出异常等等。涉及到的Service类包括ReportService。

（1）测试对象

Service层的测试对象即为各个Service类中的具体方法，详细描述如下表所示：

|  |  |  |
| --- | --- | --- |
| 类名 | ReportService | |
| 方法名 | 方法描述 | 参数说明 |
| addReport | 添加举报记录 | 传入举报基本信息 |
| getReportList | 获取举报记录 | 传入查询信息比如类别id等 |
| getReportDetail | 获取举报详情 | 传入举报id |
| doAudit | 举报审核 | 传入举报和审核信息 |

（2）测试用例设计

Service层的职责在于处理具体的业务逻辑。其中ReportService模块中的Service层主要是为了后台管理服务的，主要业务逻辑为数据库的增删改查，而对于算法没有什么要求，所以对于该Service层的单元测试主要是为了校验其返回值的主要属性，包括数量、主键是否正确，至于是否符合业务，需要到接口测试才能体现。

**4.2.2 测试用例**

对于每个测试对象都给出多个测试用例，以覆盖所有的参数输入情况，包括正常输入、边界值输入、空指输入等传参输入，并给出期望输出，和实例代码。

**4.2.2.1 getReportList**

（1）用例说明。其中参数均为ReportForm的封装属性。

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 输入参数 | 期望结果 | 参数说明 |
| RS\_RSER\_1 |  | 非空 | 传空 |
| RS\_RSER\_2 | pageSize:3,pageNo:1 | data.size:3 | 测试分页 |
| RS\_RSER\_3 | objtypeId:10000,defense:10000,prosecution:10000 | total:1 | 查询具体举报 |
| RS\_RSER\_4 | isHandle:”YES” | 非空 | 查询已处理的举报 |

（2） 测试代码部分实例

@Test  
void testGetReportList() {  
 // Setup  
 final ReportForm reportForm = new ReportForm();  
 reportForm.setReportId(0);  
 reportForm.setCreateTime(Date.*valueOf*(LocalDate.*of*(2020, 1, 1)));  
 reportForm.setObjtypeId(0);  
 reportForm.setDefense(0L);  
 reportForm.setReason("reason");  
 reportForm.setProsecution(0L);  
 reportForm.setHandleMark("handleMark");  
 reportForm.setDeleteMark("deleteMark");  
 reportForm.setPageSize(0);  
 reportForm.setPageNo(0);  
  
 // Run the test  
 final Map<String, Object> result = reportServiceImplUnderTest.getReportList(reportForm);  
  
 // Verify the results  
}

**4.2.2.2 getReportDetail**

（1）用例说明。其中参数均为ReportForm的封装属性。

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 输入参数 | 期望结果 | 参数说明 |
| RS\_RSER\_5 |  | null | 传空 |
| RS\_RSER\_6 | pageSize:3,pageNo:1 | data.size:3 | 测试分页 |
| RS\_RSER\_7 | objtypeId:10000,defense:10000 | 非空 | 查询详情 |
| RS\_RSER\_8 | isHandle:”YES” | null | 查询已处理的举报 |

（2） 测试代码部分实例

@Override  
public Map<String, Object> getReportDetail(ReportForm reportForm) {  
 Map<String, Object> params = new HashMap<>();  
 params.put("start", (reportForm.getPageNo() - 1) \* reportForm.getPageSize());  
 params.put("size", reportForm.getPageSize());  
 params.put("objtypeId", reportForm.getObjtypeId());  
 params.put("objId", reportForm.getDefense());  
  
 Map<String, Object> result = new HashMap<>();  
  
 List<ReportVo> list = new ArrayList<>();  
  
 Integer total = baseMapper.countReportDetail(params);  
 if(total > 0){  
 list = baseMapper.getReportList(params);  
 }  
  
 result.put("total", total);  
 result.put("data", list);  
 result.put("pageNo", reportForm.getPageNo());  
 result.put("pageSize", reportForm.getPageSize());  
  
 return result;  
}

**4.2.2.3 addReport**

（1）用例说明。其中参数均为ReportForm的封装属性。

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 输入参数 | 期望结果 | 参数说明 |
| RS\_RSER\_9 |  | flag:false | 传空 |
| RS\_RSER\_10 | defense:10000,reason:”xx”,objtypeId:10000,prosecution:10000 | flag:true,reportId:非空 | 正确用例 |
| RS\_RSER\_11 | objtypeId:10000,defense:10000 | flag:false | 理由和举报人为空 |

（2） 测试代码部分实例

@Test  
void testAddReport() {  
 // Setup  
 final ReportForm reportForm = new ReportForm();  
 reportForm.setReportId(0);  
 reportForm.setCreateTime(Date.*valueOf*(LocalDate.*of*(2020, 1, 1)));  
 reportForm.setObjtypeId(0);  
 reportForm.setDefense(0L);  
 reportForm.setReason("reason");  
 reportForm.setProsecution(0L);  
 reportForm.setHandleMark("handleMark");  
 reportForm.setDeleteMark("deleteMark");  
 reportForm.setPageSize(0);  
 reportForm.setPageNo(0);  
  
 // Run the test  
 final Map<String, Object> result = reportServiceImplUnderTest.addReport(reportForm);  
  
 // Verify the results  
}

**4.2.2.3 doAudit**

（1）用例说明。其中参数均为ReportForm的封装属性。

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 输入参数 | 期望结果 | 参数说明 |
| RS\_RSER\_12 |  | flag:false | 传空 |
| RS\_RSER\_13 | defense:10000,objtypeId:10000 | flag:true,update:非空 | 正确用例 |

（2） 测试代码部分实例

@Test  
void testDoAudit() {  
 // Setup  
 final ReportForm reportForm = new ReportForm();  
 reportForm.setReportId(0);  
 reportForm.setCreateTime(Date.*valueOf*(LocalDate.*of*(2020, 1, 1)));  
 reportForm.setObjtypeId(0);  
 reportForm.setDefense(0L);  
 reportForm.setReason("reason");  
 reportForm.setProsecution(0L);  
 reportForm.setHandleMark("handleMark");  
 reportForm.setDeleteMark("deleteMark");  
 reportForm.setPageSize(0);  
 reportForm.setPageNo(0);  
  
 *when*(mockReportDao.updateIsHandle(Arrays.*asList*(new Report()))).thenReturn(0);  
  
 // Run the test  
 final Boolean result = reportServiceImplUnderTest.doAudit(reportForm);  
  
 // Verify the results  
 *assertThat*(result).isTrue();  
}

1. **AuditService模块单元测试**

**5.1 Controller**

**5.1.1 测试描述**

AuditService模块的Controller层单元测试主要是通过模拟Http请求，对AuditController类的各个方法处理参数的能力做测试。

（1）测试对象

Controller层的测试对象即为各个Controller类中的具体方法，详细描述如下表所示：

|  |  |  |
| --- | --- | --- |
| 类名 | AuditController | |
| 方法名 | 方法描述 | 参数说明 |
| getAuditList | 获取审核列表 | 传入查询信息包括审核id等 |
| addAudit | 添加审核记录 | 传入查询信息比如类别id等 |

（2）测试用例设计

Controller层的职责为处理校验参数的合法性，合法参数可进行下一步操作，不合法参数便直接返回不合法信息，为了提高用户体验，对于输入的参数Controller层均能返回HTTP200的状态码，但非法信息会夹带自定义的非法状态码。所以测试用例的设计不能只顾于返回的HTTP状态码，还要具体校验返回的信息。

**5.1.2 测试用例**

对于每个测试对象都给出多个测试用例，以覆盖所有的参数输入情况，包括正常输入、边界值输入、空指输入等传参输入，并给出期望输出，和实例代码。

**5.1.2.1 getAuditList**

（1）用例说明。其中参数均为AuditForm的封装属性。

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 输入参数 | 期望结果 | 参数说明 |
| AS\_ACON\_1 | {} | code:200 | 传空 |
| AS\_ACON\_2 | {auditId:10000} | code:200 | 查询具体举报 |
| AS\_ACON\_3 | {pageNo:1,pageSize:2} | code:200 | 测试分页 |
| AS\_ACON\_4 | {orderText:”CREATE\_TIME desc”} | code:200 | 测试排序 |

（2） 测试代码部分实例

@Test  
void testGetAuditList() throws Exception {  
 // Setup  
 // Configure AuditService.getAuditList(...).  
 final AuditVo auditVo = new AuditVo();  
 auditVo.setAuditId(0);  
 auditVo.setCreateTime(Date.*valueOf*(LocalDate.*of*(2020, 1, 1)));  
 auditVo.setObjtypeId(0);  
 auditVo.setObjId(0L);  
 auditVo.setOperator(0L);  
 auditVo.setOper(0);  
 auditVo.setOperation("operation");  
 auditVo.setMessage("message");  
 auditVo.setDeleteMark("deleteMark");  
 final List<AuditVo> auditVos = Arrays.*asList*(auditVo);  
 *when*(mockAuditService.getAuditList(new AuditForm())).thenReturn(auditVos);  
  
 // Run the test  
 final MockHttpServletResponse response = mockMvc.perform(*get*("/audit/getAuditList")  
 .content("content").contentType(MediaType.*APPLICATION\_JSON*)  
 .accept(MediaType.*APPLICATION\_JSON*))  
 .andReturn().getResponse();  
  
 // Verify the results  
 *assertThat*(response.getStatus()).isEqualTo(HttpStatus.*OK*.value());  
 *assertThat*(response.getContentAsString()).isEqualTo("expectedResponse");  
}

**5.1.2.2 addAudit**

（1）用例说明。其中参数均为AuditForm的封装属性。

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 输入参数 | 期望结果 | 参数说明 |
| AS\_ACON\_5 | {} | code:500 | 传空 |
| AS\_ACON\_6 | {objtypeId:10000,objId:10000,oper:10000} | code:200 | 正确用例 |
| AS\_ACON\_7 | {objtypeId:10000,objId:10000} | code:500 | 操作为空 |

（2） 测试代码部分实例

@Test  
void testAddAudit() throws Exception {  
 // Setup  
 *when*(mockAuditService.addAudit(new AuditForm())).thenReturn(new HashMap<>());  
  
 // Run the test  
 final MockHttpServletResponse response = mockMvc.perform(*post*("/audit/addAudit")  
 .content("content").contentType(MediaType.*APPLICATION\_JSON*)  
 .accept(MediaType.*APPLICATION\_JSON*))  
 .andReturn().getResponse();  
  
 // Verify the results  
 *assertThat*(response.getStatus()).isEqualTo(HttpStatus.*OK*.value());  
 *assertThat*(response.getContentAsString()).isEqualTo("expectedResponse");  
}

**5.2 Service层**

**5.2.1 测试描述**

AuditService模块的Service层单元测试主要是通过传入测试人员设置好的参数，查看Service层各个函数的具体输出是否符合逻辑，比如返回的数量，返回的属性，是否会抛出异常等等。涉及到的Service类包括AuditService。

（1）测试对象

Service层的测试对象即为各个Service类中的具体方法，详细描述如下表所示：

|  |  |  |
| --- | --- | --- |
| 类名 | AuditService | |
| 方法名 | 方法描述 | 参数说明 |
| addAudit | 添加审核记录 | 传入审核信息 |
| getAuditList | 获取审核记录 | 传入查询信息比如类别id等 |

（2）测试用例设计

Service层的职责在于处理具体的业务逻辑。其中AuditService模块中的Service层主要是为了后台管理服务的，主要业务逻辑为数据库的增删改查，而对于算法没有什么要求，所以对于该Service层的单元测试主要是为了校验其返回值的主要属性，包括数量、主键是否正确，至于是否符合业务，需要到接口测试才能体现。

**5.2.2 测试用例**

对于每个测试对象都给出多个测试用例，以覆盖所有的参数输入情况，包括正常输入、边界值输入、空指输入等传参输入，并给出期望输出，和实例代码。

**5.2.2.1 addAudit**

（1）用例说明。其中参数均为AuditForm的封装属性。

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 输入参数 | 期望结果 | 参数说明 |
| AS\_ASER\_1 |  | flag:false | 传空 |
| AS\_ASER\_2 | objtypeId:10000,objId:10000,oper:10000 | flag:true,auditId:非空 | 正确用例 |
| AS\_ASER\_3 | objtypeId:10000,objId:10000 | flag:false | 操作为空 |

（2） 测试代码部分实例

@Test  
void testAddAudit() {  
 // Setup  
 final AuditForm auditForm = new AuditForm();  
 auditForm.setAuditId(0);  
 auditForm.setCreateTime(Date.*valueOf*(LocalDate.*of*(2020, 1, 1)));  
 auditForm.setObjtypeId(0);  
 auditForm.setObjId(0L);  
 auditForm.setStatus(0);  
 auditForm.setOperator(0L);  
 auditForm.setOper(0);  
 auditForm.setMessage("message");  
 auditForm.setDeleteMark("deleteMark");  
  
 // Run the test  
 final Map<String, Object> result = auditServiceImplUnderTest.addAudit(auditForm);  
  
 // Verify the results  
}

**5.2.2.2 getAuditList**

（1）用例说明。其中参数均为AuditForm的封装属性。

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 输入参数 | 期望结果 | 参数说明 |
| AS\_ASER\_4 |  | 非空 | 传空 |
| AS\_ASER\_5 | pageSize:3,pageNo:1 | data.size:3 | 测试分页 |
| AS\_ASER\_6 | objtypeId:10000,objId:10000 | 非空 | 测试查询条件 |
| AS\_ASER\_7 | orderText:CREATE\_TIME desc | 非空 | 测试排序 |
| AS\_ASER\_8 | auditId:10000 | total:1 | 插叙具体审核 |

（2） 测试代码部分实例

@Test  
void testGetAuditList() {  
 // Setup  
 final AuditForm auditForm = new AuditForm();  
 auditForm.setAuditId(0);  
 auditForm.setCreateTime(Date.*valueOf*(LocalDate.*of*(2020, 1, 1)));  
 auditForm.setObjtypeId(0);  
 auditForm.setObjId(0L);  
 auditForm.setStatus(0);  
 auditForm.setOperator(0L);  
 auditForm.setOper(0);  
 auditForm.setMessage("message");  
 auditForm.setDeleteMark("deleteMark");  
  
 final AuditVo auditVo = new AuditVo();  
 auditVo.setAuditId(0);  
 auditVo.setCreateTime(Date.*valueOf*(LocalDate.*of*(2020, 1, 1)));  
 auditVo.setObjtypeId(0);  
 auditVo.setObjId(0L);  
 auditVo.setOperator(0L);  
 auditVo.setOper(0);  
 auditVo.setOperation("operation");  
 auditVo.setMessage("message");  
 auditVo.setDeleteMark("deleteMark");  
 final List<AuditVo> expectedResult = Arrays.*asList*(auditVo);  
  
 // Run the test  
 final List<AuditVo> result = auditServiceImplUnderTest.getAuditList(auditForm);  
  
 // Verify the results  
 *assertThat*(result).isEqualTo(expectedResult);  
}

1. **SystemService模块单元测试**

**6.1 Controller**

**6.1.1 测试描述**

SystemService模块的Controller层单元测试主要是通过模拟Http请求，对SyslogController、MessageController类的各个方法处理参数的能力做测试。

（1）测试对象

Controller层的测试对象即为各个Controller类中的具体方法，详细描述如下表所示：

|  |  |  |
| --- | --- | --- |
| 类名 | MessageController | |
| 方法名 | 方法描述 | 参数说明 |
| launchMessage | 发送信息 | 传入信息基本内容 |
| 类名 | SyslogController | |
| 方法名 | 方法描述 | 参数说明 |
| addLog | 添加日志 | 传入日志内容 |

（2）测试用例设计

Controller层的职责为处理校验参数的合法性，合法参数可进行下一步操作，不合法参数便直接返回不合法信息，为了提高用户体验，对于输入的参数Controller层均能返回HTTP200的状态码，但非法信息会夹带自定义的非法状态码。所以测试用例的设计不能只顾于返回的HTTP状态码，还要具体校验返回的信息。

**6.1.2 测试用例**

对于每个测试对象都给出多个测试用例，以覆盖所有的参数输入情况，包括正常输入、边界值输入、空指输入等传参输入，并给出期望输出，和实例代码。

**6.1.2.1 launchMessage**

（1）用例说明。其中参数均为MessageForm的封装属性。

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 输入参数 | 期望结果 | 参数说明 |
| SS\_SCON\_1 |  | code:500 | 传空 |
| SS\_SCON\_2 | {scope:10000, content:”xxxx”} | code:500 | 接收者为空 |
| SS\_SCON\_3 | {content:”xxx”,specificUsers:[10000,10001]} | code:500 | 范围为空 |
| SS\_SCON\_4 | {scope:10000,specificUsers:[10000,10001]} | code:500 | 内容为空 |
| SS\_SCON\_5 | {scope:10000,specificUsers:[10000,10001],content:”xxx”} | code:200 | 正确用例 |

（2） 测试代码部分实例

@Test  
void testLaunchMessage() throws Exception {  
 // Setup  
 *when*(mockMessageService.launchMessage(new MessageForm())).thenReturn(new HashMap<>());  
 *when*(mockEmailService.sendEmail(*any*(String[].class), *eq*("content"))).thenReturn(false);  
  
 // Run the test  
 final MockHttpServletResponse response = mockMvc.perform(*post*("/sys/launchMessage")  
 .content("content").contentType(MediaType.*APPLICATION\_JSON*)  
 .accept(MediaType.*APPLICATION\_JSON*))  
 .andReturn().getResponse();  
  
 // Verify the results  
 *assertThat*(response.getStatus()).isEqualTo(HttpStatus.*OK*.value());  
 *assertThat*(response.getContentAsString()).isEqualTo("expectedResponse");  
 *verify*(mockEmailService).sendEmail(*any*(String[].class), *eq*("content"));  
}

**6.1.2.2 addSyslog**

（1）用例说明。其中参数均为SyslogForm的封装属性。

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 输入参数 | 期望结果 | 参数说明 |
| SS\_SCON\_6 |  | code:500 | 传空 |
| SS\_SCON\_7 | {objtypeId:20000} | code:500 | 类别id非法 |
| SS\_SCON\_8 | {objtypeId:10000, objId:10000, operator: 10000, oper:10000} | code:200 | 正确用例 |
| SS\_SCON\_9 | {objtypeId:10000, objId:10000, oper:10000} | code:500 | 操作人为空 |
| SS\_SCON\_10 | {objtypeId:10000, operator: 10000, oper:10000} | code:500 | 对象id为空 |

（2） 测试代码部分实例

@Test  
void testAddLog() throws Exception {  
 // Setup  
 *when*(mockSyslogService.addLog(new SyslogForm())).thenReturn(0L);  
  
 // Run the test  
 final MockHttpServletResponse response = mockMvc.perform(*post*("/sys/addLog")  
 .content("content").contentType(MediaType.*APPLICATION\_JSON*)  
 .accept(MediaType.*APPLICATION\_JSON*))  
 .andReturn().getResponse();  
  
 // Verify the results  
 *assertThat*(response.getStatus()).isEqualTo(HttpStatus.*OK*.value());  
 *assertThat*(response.getContentAsString()).isEqualTo("expectedResponse");  
}

**6.2 Service层**

**6.2.1 测试描述**

SystemService模块的Service层单元测试主要是通过传入测试人员设置好的参数，查看Service层各个函数的具体输出是否符合逻辑，比如返回的数量，返回的属性，是否会抛出异常等等。涉及到的Service类包括SyslogService，MessageService。

（1）测试对象

Service层的测试对象即为各个Service类中的具体方法，详细描述如下表所示：

|  |  |  |
| --- | --- | --- |
| 类名 | MessageService | |
| 方法名 | 方法描述 | 参数说明 |
| launchMessage | 发送信息 | 传入信息基本内容 |
| 类名 | SyslogService | |
| 方法名 | 方法描述 | 参数说明 |
| addLog | 添加日志 | 传入日志内容 |

（2）测试用例设计

Service层的职责在于处理具体的业务逻辑。其中SystemService模块中的Service层主要是为了后台管理服务的，主要业务逻辑为数据库的增删改查，而对于算法没有什么要求，所以对于该Service层的单元测试主要是为了校验其返回值的主要属性，包括数量、主键是否正确，至于是否符合业务，需要到接口测试才能体现。

**6.2.2 测试用例**

对于每个测试对象都给出多个测试用例，以覆盖所有的参数输入情况，包括正常输入、边界值输入、空指输入等传参输入，并给出期望输出，和实例代码。

**6.2.2.1 launchMessage**

（1）用例说明。其中参数均为MessageForm的封装属性。

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 输入参数 | 期望结果 | 参数说明 |
| SS\_SSER\_1 |  | flag:false | 传空 |
| SS\_SSER\_2 | scope:10000,specificUsers:[10000,10001],content:”xxx” | flag:true,messageId:非空 | 正确用例 |
| SS\_SSER\_3 | scope:10000,specificUsers:[10000,10001] | flag:false | 内容为空 |

（2） 测试代码部分实例

@Test  
void testLaunchMessage() {  
 // Setup  
 final MessageForm messageForm = new MessageForm();  
 messageForm.setMessageId(0L);  
 messageForm.setCreateTime(Date.*valueOf*(LocalDate.*of*(2020, 1, 1)));  
 messageForm.setDeleteMark("deleteMark");  
 messageForm.setScope(0);  
 messageForm.setScopeDetail("scopeDetail");  
 messageForm.setContent("content");  
 messageForm.setSpecificUsers(Arrays.*asList*(0L));  
 messageForm.setTitle("title");  
 messageForm.setEmails(new String[]{"emails"});  
  
 // Run the test  
 final Map<String, Object> result = messageServiceImplUnderTest.launchMessage(messageForm);  
  
 // Verify the results  
}

**6.2.2.2 addSyslog**

（1）用例说明。其中参数均为SyslogForm的封装属性。

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 输入参数 | 期望结果 | 参数说明 |
| SS\_SSER\_4 |  | flag:false | 传空 |
| SS\_SSER\_5 | objtypeId:10000, objId:10000, operator: 10000, oper:10000 | flag:true,syslogId:非空 | 正确用例 |
| SS\_SSER\_6 | objtypeId:10000, objId:10000, operator: 10000 | flag:false | 操作为空 |

（2） 测试代码部分实例

@Test  
void testAddLog() {  
 // Setup  
 final SyslogForm syslogForm = new SyslogForm();  
 syslogForm.setLogId(0L);  
 syslogForm.setCreateTime(Date.*valueOf*(LocalDate.*of*(2020, 1, 1)));  
 syslogForm.setDeleteMark("deleteMark");  
 syslogForm.setObjtypeId(0);  
 syslogForm.setObjId(0L);  
 syslogForm.setOperator(0L);  
 syslogForm.setOperation(0);  
 syslogForm.setMessage("message");  
 syslogForm.setObjtype("objtype");  
  
 *when*(mockRedisTemplate.opsForValue()).thenReturn(null);  
  
 // Run the test  
 final Long result = syslogServiceImplUnderTest.addLog(syslogForm);  
  
 // Verify the results  
 *assertThat*(result).isEqualTo(0L);  
}