

南大测试系统概要设计说明书

引言

- 编写目的
- 背景
- 参考资料

总体设计

- 简述
- 架构设计
 - 逻辑架构图
 - 顶层系统包图
 - 业务类包图
- 接口设计
 - 请求设计
 - 响应设计

模块设计

- 基础信息子模块
 - 子系统说明
 - 类图
 - 用户
 - 角色
 - 菜单
 - 权限
 - 公司
 - 企业认证
 - 类说明
 - 用户
 - 角色
 - 菜单
 - 权限
 - 公司
 - 企业认证
- 流程子模块
 - 子系统说明
 - 类图
 - 委托
 - 合同
 - 样品
 - 测试方案
 - 测试报告
 - 类说明
 - 委托
 - 合同
 - 样品
 - 测试方案
 - 测试报告
- 用户子模块
 - 子系统说明
 - 类图
 - 登录认证
 - 个人中心
 - 类说明
 - 登录认证

1 引言

1.1 编写目的

此概要设计说明书是为了说明整个系统的体系架构，以及需求用例的各个功能点在架构中的体现，为系统的详细设计人员进行详细设计时的输入参考文档。

1.2 背景

- 系统名称：南大软件测试中心管理系统
- 开发语言：Java、Python
- 团队协作工具：Git
- 系统框架：ruoyi-vue-pro
- 开发人员：秦嘉余、赖烨文、蒋梓栩、刘永鹏、孙文戈、张城铨

1.3 参考资料

- ruoyi-vue-pro 开发指南
- 概要设计说明书实例

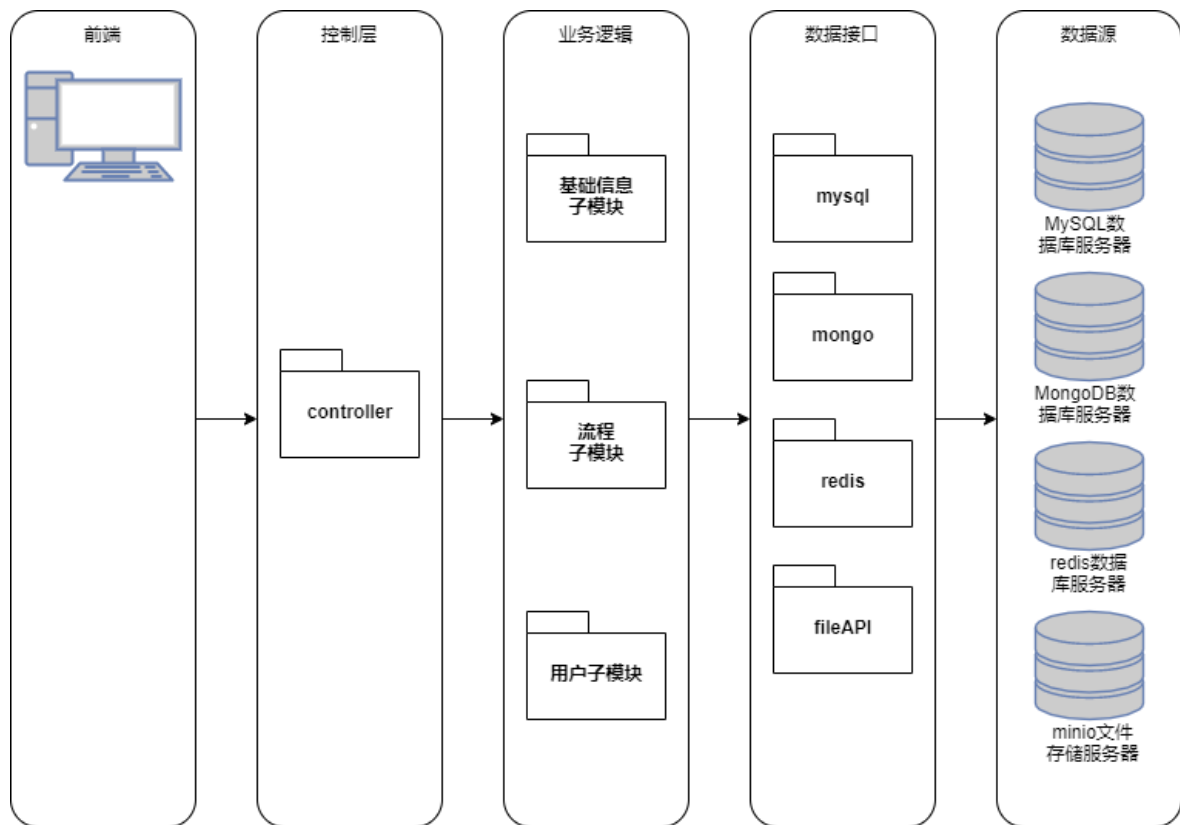
2 总体设计

2.1 简述

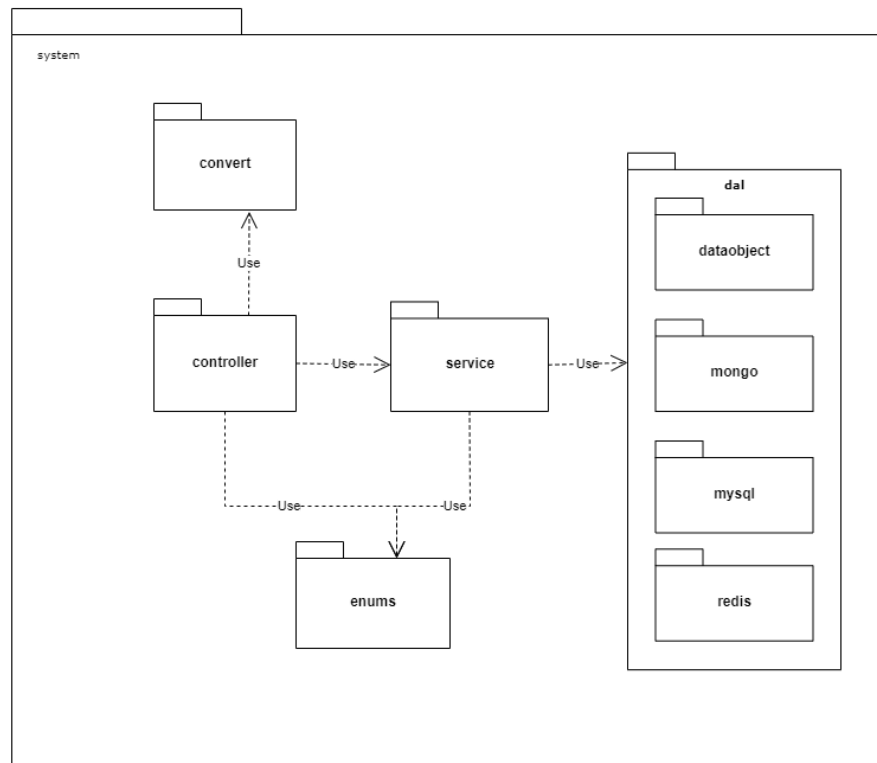
基于 Spring Boot + MyBatis Plus + Spring Security 实现了前后端分离架构的南大测试管理系统的后端部分。支持 RBAC 动态权限分配，流程流转控制，日志记录，短信通知等功能。

2.2 架构设计

2.2.1 逻辑架构图



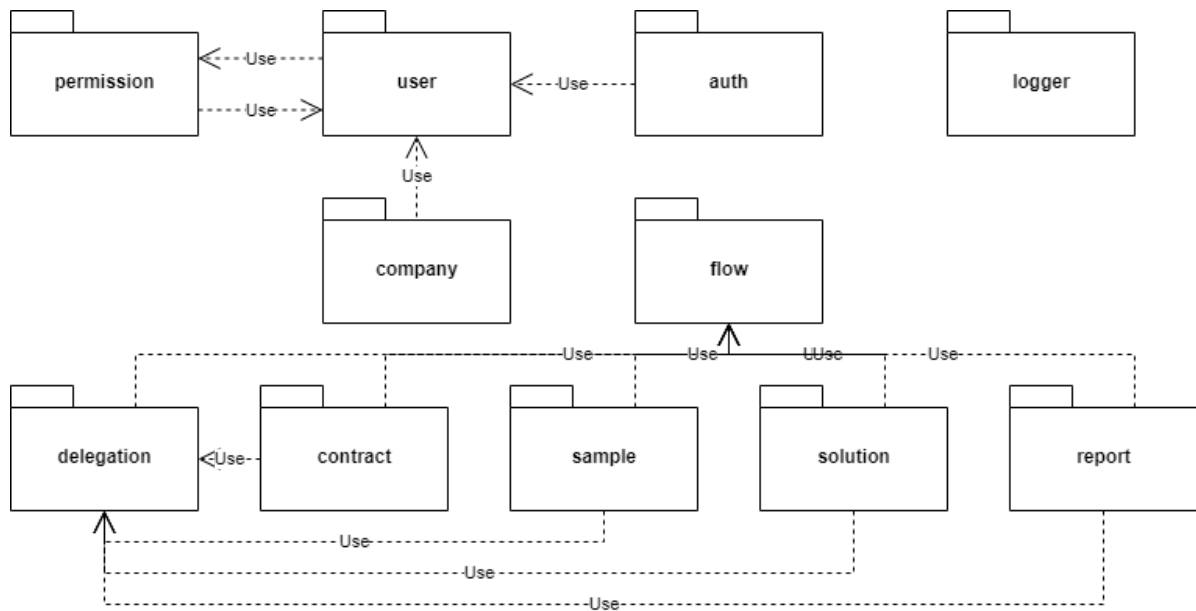
2.2.2 顶层系统包图



类包说明：

- controller包为控制层类所在包，定义了接口类型，调用service包下的service类完成逻辑操作，使用enums包下定义的枚举类，同时借助convert包下的convert类完成VO的转化
- convert包为转化类所在包，帮助controller完成VO的转化
- enums包定义了系统所使用的枚举类常量、
- service包为业务层类所在包，包含对业务数据的增删改查以及一些复杂操作
- dal包为数据访问层所在包，包含四个独立功能的子包
 - dataobject包为定义数据库实体DO类所在的包
 - mongo包为操作MongoDB文档数据的respiratory类所在的包
 - mysql包为操作MySQL数据的持久层类所在的包
 - redis为操作Redis缓存数据类所在的包

2.2.3 业务类包图



- user包对执行用户管理的业务操作
- permission包执行角色、菜单、权限管理的业务操作
- auth包执行用户登录、注册、认证的业务操作
- company包执行公司管理的业务操作
- logger包执行登录与api操作的日志记录
- delegation包执行委托管理的业务操作与流转
- contract包执行合同操作与管理的业务操作
- sample包执行样品管理的业务操作
- solution包执行测试方案管理的业务操作
- report包执行测试报告管理的业务操作
- flow包执行对流程中状态的日志记录

2.3 接口设计

2.3.1 请求设计

项目采用RESTful API 的接口设计，并配置Swagger实现接口文档。每个接口单独配置与前端交互的视图对象，即VO，以保证项目接口的灵活可变。每个接口根据所在包的位置，分配了不同的前缀，以保证接口的清晰与易读。详细接口内容见接口文档。

2.3.2 响应设计

每个接口的响应分为三部分：

- code：返回码，0代表正常返回，异常返回时设定为项目规定的全局异常码
- data：返回的数据
- message：返回的异常信息，当请求异常时才会非空，设定为异常消息提示

3 模块设计

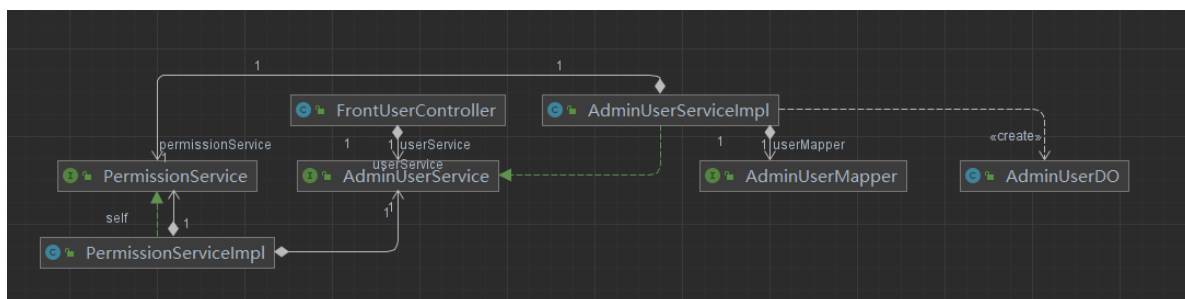
3.1 基础信息子模块

3.1.1 子系统说明

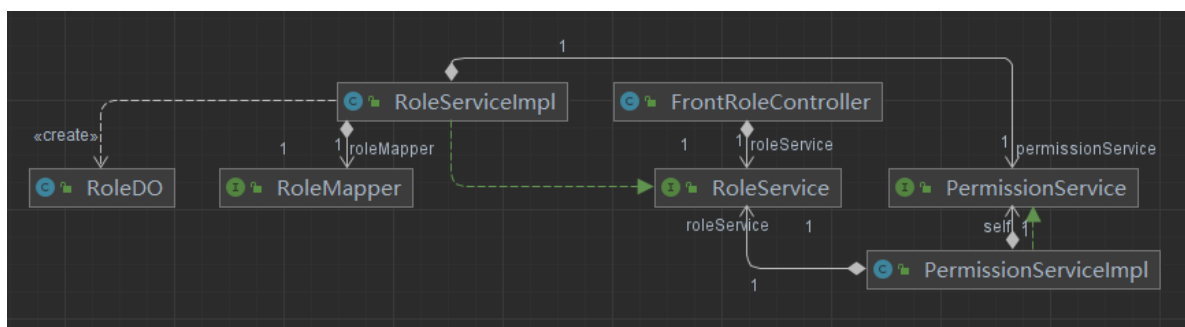
- 基础信息子模块包括：用户管理、角色管理、菜单管理、权限管理、公司管理、公司认证六部分
 - 用户管理：实现对用户基本信息的增删查改
 - 角色管理：实现对角色基本信息的增删查改
 - 菜单管理：实现对菜单基本信息的增删查改
 - 权限管理：实现对用户-角色，角色-菜单的分配以及增删查改
 - 公司管理：实现对公司基本信息的增删查改
 - 公司认证：实现对用户的公司认证以及增删查改

3.1.2 类图

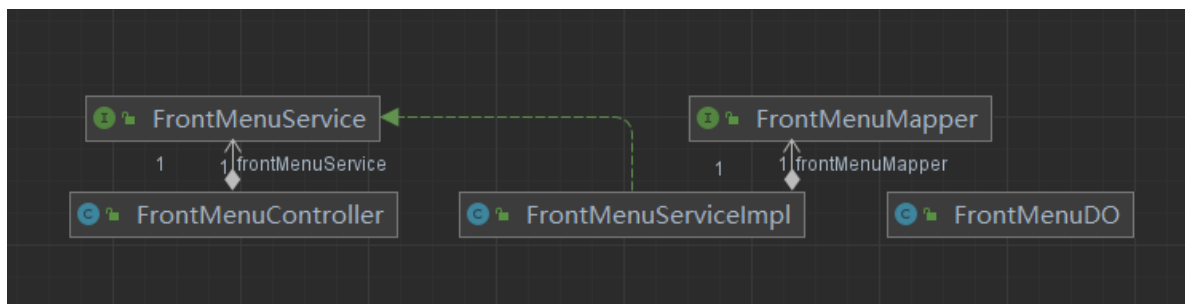
3.1.2.1 用户



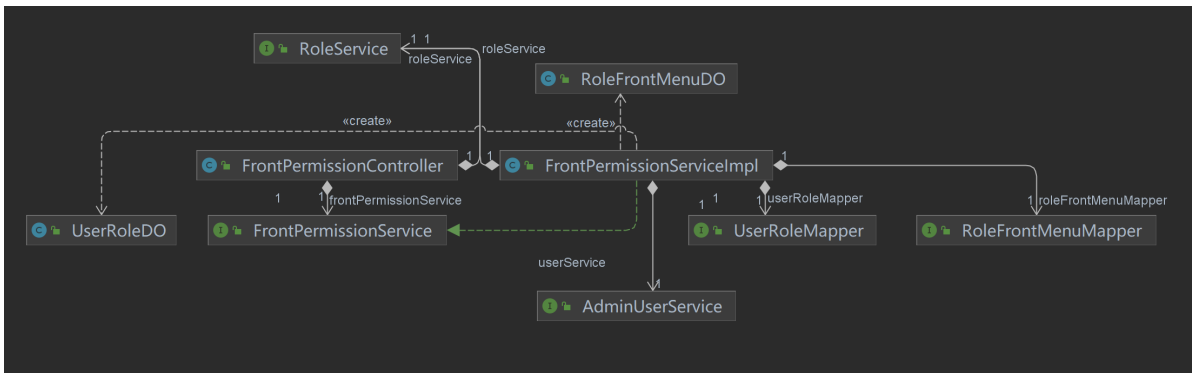
3.1.2.2 角色



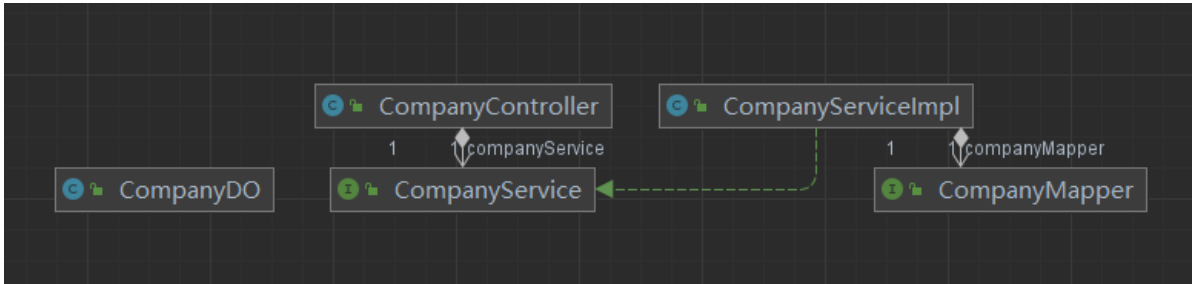
3.1.2.3 菜单



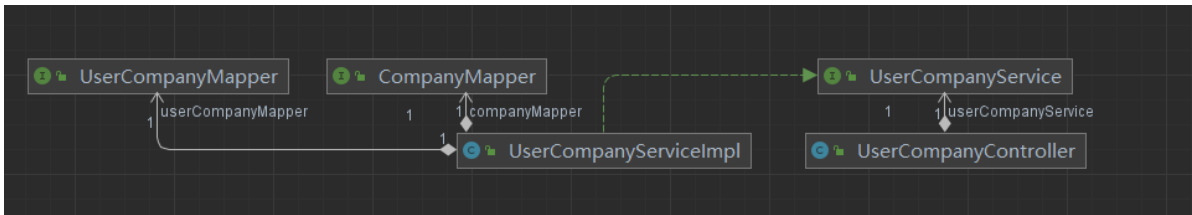
3.1.2.4 权限



3.1.2.5 公司



3.1.2.6 公司认证



3.1.3 类说明

3.1.3.1 用户

FrontUserController	
userService	AdminUserService
deptService	DeptService
updateUserPassword (FrontUserUpdatePasswordReqVO)	CommonResult<Boolean>
getInfo(Long)	CommonResult<FrontUserRespVO>
updateUserStatus (FrontUserUpdateStatusReqVO)	CommonResult<Boolean>
getUserPage (FrontUserPageReqVO)	CommonResult<PageResult<FrontUserPageItemRespVO>>
updateUser (FrontUserUpdateReqVO)	CommonResult<Boolean>
deleteUser (Long)	CommonResult<Boolean>
getSimpleUsers ()	CommonResult<List<FrontUserSimpleRespVO>>
createUser (FrontUserCreateReqVO)	CommonResult<Long>

- createUser: 传入FrontUserCreateReqVO, 调用AdminUserService的createUser方法创建用户, 返回创建好的用户的id
- updateUser: 传入FrontUserUpdateReqVO, 调用AdminUserService的updateUser方法创建用户, 返回是否创建成功
- deleteUser: 传入用户编号id, 调用AdminUserService的deleteUser方法删除用户, 返回是否删除成功



















- updateUserPassword: 传入FrontUserUpdatePasswordReqVO, 调用AdminUserService的updateUserPassword方法重置用户密码, 返回是否重置成功
- updateUserStatus: 传入FrontUserUpdateStatusReqVO, 调用AdminUserService的updateUserStatus方法修改用户状态, 返回是否修改成功
- getUserPage: 传入FrontUserPageReqVO, 调用AdminUserService的getUserPage方法获得用户分页列表, 返回用户分页列表
- getSimpleUsers: 无需传入参数, 调用AdminUserService的getUsersByStatus方法获取用户精简信息列表, 返回用户精简信息列表
- getInfo: 传入用户编号id, 调用AdminUserService的getUser方法法获得用户详情, 返回用户详情

AdminUserServiceImpl





























f	deptService	DeptService
f	postService	PostService
f	permissionService	PermissionService
f	fileApi	FileApi
f	log	Logger
f	userInitPassword	String
f	passwordEncoder	PasswordEncoder
f	tenantService	TenantService
f	userMapper	AdminUserMapper
m	updateUser (UserUpdateReqVO)	void
m	updateUserProfile (Long, UserProfileUpdateReqVO)	void
m	getUserPage (UserPageReqVO)	PageResult<AdminUserDO>
m	deleteUser (Long)	void
m	getUsersByStatus (Integer)	List<AdminUserDO>
m	checkUserExists (Long)	void
m	getUserByUsername (String)	AdminUserDO
m	validUsers (Set<Long>)	void
m	getUsers (UserExportReqVO)	List<AdminUserDO>
m	getUserByMobile (String)	AdminUserDO
m	getUsersByNickname (String)	List<AdminUserDO>
m	importUsers (List<UserImportExcelVO >, boolean)	UserImportRespVO
m	getUsersByUsername (String)	List<AdminUserDO>
m	checkCreateOrUpdate (Long, String, String, String, Long, Set<Long>)	d
m	checkUsernameUnique (Long, String)	void
m	getUsersByPostIds (Collection<Long>)	List<AdminUserDO>
m	updateUserStatus (Long, Integer)	void
m	updateUserPassword (Long, String)	void
m	getDeptCondition (Long)	Set<Long>
m	updateUserAvatar (Long, InputStream)	String
m	createUser (UserCreateReqVO)	Long
m	updateUserLogin (Long, String)	void
m	getUsersByDeptIds (Collection<Long>)	List<AdminUserDO>
m	getUsers (Collection<Long>)	List<AdminUserDO>
m	updateUserPassword (Long, UserProfileUpdatePasswordReqVO)	void
m	getUser (Long)	AdminUserDO
m	checkEmailUnique (Long, String)	void

```
checkMobileUnique(Long, String) void
checkOldPassword(Long, String) void
```

- createUser: 校验正确性后, 调用AdminUserMapper的insert方法向数据库里插入一个用户, 返回插入用户的主键id
- updateUser: 校验正确性后, 调用AdminUserMapper的updateById方法更新用户信息
- updateUserLogin: 校验正确性后, 调用AdminUserMapper的updateById方法更新用户登录Ip与日期
- updateUserProfile: 校验正确性后, 调用AdminUserMapper的updateById方法更新用户基本信息
- updateUserPassword: 校验正确性后, 调用AdminUserMapper的updateById方法更新用户密码
- updateUserAvatar: 校验正确性后, 上传传入的图片, 调用AdminUserMapper的updateById方法更新用户头像
- updateUserPassword: 校验正确性后, 调用AdminUserMapper的updateById方法更新用户密码
- updateUserStatus: 校验正确性后, 调用AdminUserMapper的updateById方法更新用户状态
- deleteUser: 校验正确性后, 调用AdminUserMapper的deleteById方法删除用户的
- getUserByUsername: 调用AdminUserMapper的selectByUsername方法通过用户名获得用户, 返回用户实体
- getUserPage: 调用AdminUserMapper的selectPage方法传入分页条件获得用户分页, 返回用户分页
- getUser: 调用AdminUserMapper的selectById通过id获得用户, 返回用户实体
- getUsers: 调用AdminUserMapper的selectBatchIds通过id列表获得用户列表, 返回用户实体列表
- validUsers: 校验传入的用户id的正确性, 包含是否存在和是否为开启状态
- getUsersByNickname: 调用AdminUserMapper的selectListByNickname通过昵称获得用户列表, 返回用户实体列表
- getUsersByUsername: 调用AdminUserMapper的selectListByUsername通过用户名获得用户列表, 返回用户实体列表
- checkCreateOrUpdate: 校验传入的参数是否正确, 校验用户存在, 校验用户名唯一, 校验手机号唯一, 校验邮箱唯一
- checkUserExists: 校验传入的用户id是否存在
- checkUsernameUnique: 校验传入的用户名是否为唯一
- checkEmailUnique: 校验传入的邮箱是否为唯一
- checkMobileUnique: 校验传入的手机号是否为唯一
- checkOldPassword: 校验旧密码与新密码是否相同
- getUsersByStatus: 调用AdminUserMapper的selectListByStatus通过状态获得用户列表, 返回用户实体列表
- getUserByMobile: 调用AdminUserMapper的selectByMobile通过手机号获得用户, 返回用户实体










AdminUserMapper		
	 selectByUsername (String)	AdminUserDO
	 selectByMobile (String)	AdminUserDO
	 selectList (UserExportReqVO , Collection<Long>)	List<AdminUserDO>
	 selectPage (UserPageReqVO , Collection<Long>)	PageResult<AdminUserDO>
	 selectListByNickname (String)	List<AdminUserDO>
	 selectListByDeptIds (Collection<Long>)	List<AdminUserDO>
	 selectListByStatus (Integer)	List<AdminUserDO>
	 selectByEmail (String)	AdminUserDO
	 selectListByUsername (String)	List<AdminUserDO>

- selectByUsername: 通过用户名查询用户，得到用户实体返回
- selectByEmail: 通过邮箱查询用户，得到用户实体返回
- selectByMobile: 通过手机号查询用户，得到用户实体返回
- selectPage: 通过传入参数从分页查询，得到用户分页返回
- selectList: 通过传入参数查询列表，得到用户列表返回
- selectListByNickname: 通过昵称查询用户列表，得到用户实体列表返回
- selectListByUsername: 通过用户名查询用户列表，得到用户实体列表返回
- selectListByStatus: 通过状态查询用户列表，得到用户实体列表返回

AdminUserDO		
	 loginIp	String
	 deptId	Long
	 sex	Integer
	 status	Integer
	 remark	String
	 loginDate	Date
	 username	String
	 postIds	Set<Long>
	 nickname	String
	 mobile	String
	 email	String
	 id	Long
	 password	String
	 avatar	String





























































- id: 用户ID
- username: 用户账号
- password: 加密后的密码
- nickname: 用户昵称
- remark: 备注
- email: 用户邮箱
- mobile: 手机号码
- sex: 用户性别
- avatar: 用户头像
- status: 帐号状态
- loginIp: 最后登录IP
- loginDate: 最后登录时间

3.1.3.2 角色

FrontRoleController	
 <i>ROLE_IDS</i>	List<Long>
 <i>roleService</i>	RoleService
 <i>createRole</i> (FrontRoleCreateReqVO)	CommonResult<Long>
 <i>getSimpleRoles</i> ()	CommonResult<List<FrontRoleSimpleRespVO>>
 <i>deleteRole</i> (Long)	CommonResult<Boolean>
 <i>getRolePage</i> (FrontRolePageReqVO)	CommonResult<PageResult<FrontRolePageRespVO>>
 <i>getRole</i> (Long)	CommonResult<FrontRoleRespVO>
 <i>updateRole</i> (FrontRoleUpdateReqVO)	CommonResult<Boolean>
 <i>updateRoleStatus</i> (FrontRoleUpdateStatusReqVO)	CommonResult<Boolean>


- createRole: 传入FrontRoleCreateReqVO, 调用RoleService的createRole方法创建角色, 返回创建好的角色的id
- updateRole: 传入FrontRoleUpdateReqVO, 调用RoleService的updateRole方法更新角色, 返回是否更新成功
- updateRoleStatus: 传入FrontRoleUpdateStatusReqVO, 调用RoleService的updateRoleStatus方法更新角色状态, 返回是否更新成功
- deleteRole: 传入角色编号id, 调用RoleService的deleteRole方法删除角色, 返回是否删除成功
- getRole: 传入角色编号id, 调用RoleService的getRole方法, 返回角色信息
- getRolePage: 传入FrontRolePageReqVO, 调用RoleService的getFrontRolePage方法获取角色分页, 返回角色分页
- getSimpleRoles: 调用RoleService的getRoles方法, 获得所有开启的角色, 返回开启的角色信息列表















RoleServiceImpl

  <i>SCHEDULER_PERIOD</i>	long
  roleProducer	RoleProducer
  permissionService	PermissionService
  maxUpdateTime	Date
  self	RoleService
  log	Logger
  roleCache	Map <Long, RoleDO>
  roleMapper	RoleMapper
  getRoleByCode (String)	RoleDO
  <i>getRoleCache()</i>	Map <Long, RoleDO>
  <i>getMaxUpdateTime()</i>	Date
  getRolePage (RolePageReqVO)	PageResult<RoleDO>
  getRoleList (RoleExportReqVO)	List<RoleDO>
  schedulePeriodicRefresh ()	void
  validRoles (Collection<Long>)	void
  loadRoleIfUpdate (Date)	List<RoleDO>?
  hasAnySuperAdmin (Collection<RoleDO>)	boolean
  getRolesFromCache (Collection<Long>)	List<RoleDO>
  updateRoleDataScope (Long, Integer, Set<Long>)	void
  checkDuplicateRole (String, String, Long)	void
  getFrontRolePage (FrontRolePageReqVO)	PageResult<RoleDO>
  initLocalCache()	void
  deleteRole (Long)	void
  updateRole (RoleUpdateReqVO)	void
  updateRoleStatus (Long, Integer)	void
  getRoleFromCache (Long)	RoleDO
  createRole (RoleCreateReqVO, Integer)	Long
  getRoles (Collection<Integer>?)	List<RoleDO>
  getRole (Long)	RoleDO
  checkUpdateRole (Long)	void

- createRole: 校验正确性后, 调用RoleMapper的insert方法向数据库里插入一个角色, 返回插入角色的主键id

- updateRole: 校验正确性后, 调用RoleMapper的updateById方法更新角色信息
- updateRoleStatus: 校验正确性后, 调用RoleMapper的updateById方法更新角色状态
- deleteRole: 校验正确性后, 调用RoleMapper的deleteById方法删除角色
- getRoles: 调用RoleMapper的selectListByStatus通过状态获取角色列表, 返回角色实体列表
- getRole: 调用RoleMapper的getRole方法获得角色, 返回角色实体
- getRolePage: 调用RoleMapper的selectPage方法传入分页条件获得角色分页, 返回角色实体分页
- getFrontRolePage: 调用RoleMapper的selectPage方法传入分页条件获得前台需要的角色分页, 返回角色实体分页
- checkDuplicateRole: 校验角色的唯一字段是否重复, 是否存在相同名字的角色, 是否存在相同编码的角色
- checkUpdateRole: 校验角色是否可以被更新
- validRoles: 检验传入的角色是否存在, 是否被禁用
- getRoleByCode: 通过code获得角色, 返回角色实体


RoleMapper

 	listRoles (RoleExportReqVO)	List<RoleDO>
 	selectByName (String)	RoleDO
 	selectFrontPage (FrontRolePageReqVO)	PageResult<RoleDO>
 	selectExistsByUpdateTimeAfter (Date)	RoleDO
 	selectListByStatus (Collection<Integer>?)	List<RoleDO>
 	selectByCode (String)	RoleDO
 	selectPage (RolePageReqVO)	PageResult<RoleDO>

- selectPage: 通过传入参数从分页查询, 得到角色分页返回
- listRoles: 通过传入参数查询角色, 得到角色列表返回
- selectByName: 通过名称查询角色, 得到角色实体返回
- selectByCode: 通过code查询角色, 得到角色实体返回
- selectListByStatus: 通过状态查询角色列表, 得到角色实体列表返回
- selectFrontPage: 通过传入参数从分页查询, 得到前台需要的角色分页返回

- id: 角色ID
- name: 角色名称
- code: 角色标识
- sort: 角色排序
- status: 角色状态
- type: 角色状态
- remark: 备注

3.1.3.3 菜单

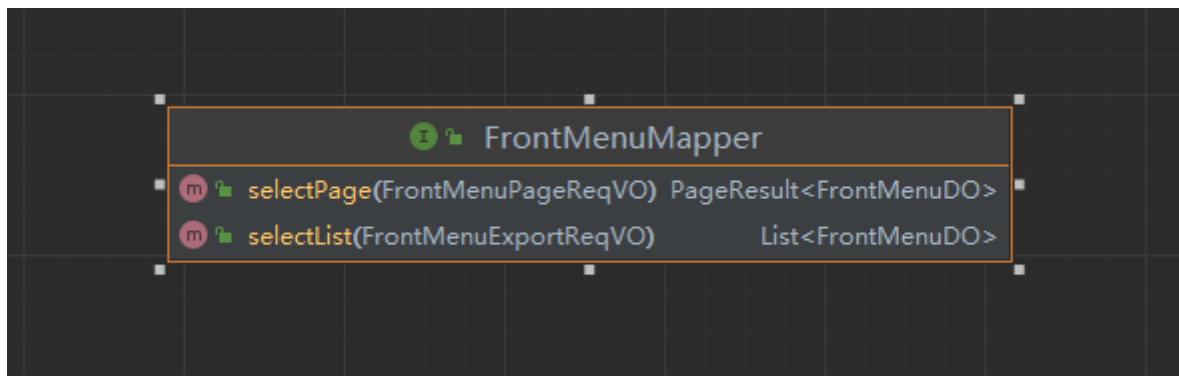
FrontMenuController		
frontMenuService		FrontMenuService
createFrontMenu(FrontMenuCreateReqVO)		CommonResult<Long>
getFrontMenu(Long)		CommonResult<FrontMenuRespVO>
deleteFrontMenu(Long)		CommonResult<Boolean>
getFrontMenuList(Collection<Long>)		CommonResult<List<FrontMenuRespVO>>
getFrontMenuPage(FrontMenuPageReqVO)		CommonResult<PageResult<FrontMenuRespVO>>
updateFrontMenu(FrontMenuUpdateReqVO)		CommonResult<Boolean>

- createFrontMenu: 传入FrontMenuCreateReqVO, 调用FrontMenuService的createFrontMenu方法创建菜单, 返回创建好的菜单的id
- updateFrontMenu: 传入FrontMenuUpdateReqVO, 调用FrontMenuService的updateFrontMenu方法更新菜单, 返回是否更新成功
- deleteFrontMenu: 传入FrontMenuUpdateReqVO, 调用FrontMenuService的deleteFrontMenu方法删除菜单, 返回是否删除成功
- getFrontMenu: 传入菜单编号id, 调用FrontMenuService的getFrontMenu获得菜单, 返回菜单信息
- getFrontMenuList: 传入菜单编号列表, 调用FrontMenuService的getFrontMenuList获得菜单列表, 返回菜单信息列表
- getFrontMenuPage: 传入FrontMenuPageReqVO, 调用FrontMenuService的getFrontMenuPage方法获取菜单分页, 返回菜单分页

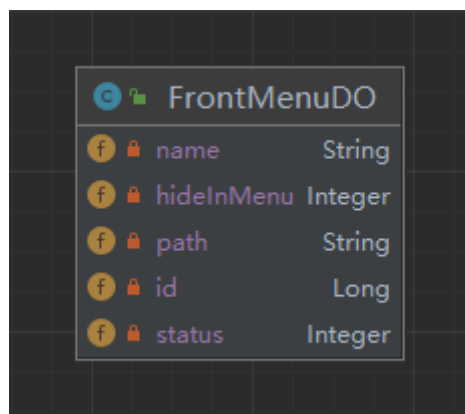
FrontMenuServiceImpl		
frontMenuMapper		FrontMenuMapper
roleFrontMenuMapper		RoleFrontMenuMapper
getFrontMenuPage(FrontMenuPageReqVO)		PageResult<FrontMenuDO>
createFrontMenu(FrontMenuCreateReqVO)		Long
deleteFrontMenu(Long)		void
getFrontMenu(Long)		FrontMenuDO
getFrontMenuList(FrontMenuExportReqVO)		List<FrontMenuDO>
validateFrontMenuExists(Long)		void
validFrontMenus(Collection<Long>)		void
updateFrontMenu(FrontMenuUpdateReqVO)		void
getFrontMenuList(Collection<Long>)		List<FrontMenuDO>

- createFrontMenu: 校验正确性后, 调用FrontMenuMapper的insert方法向数据库里插入一个菜单, 返回插入菜单的主键id
- updateFrontMenu: 校验正确性后, 调用FrontMenuMapper的updateById方法更新菜单信息
- deleteFrontMenu: 校验正确性后, 调用FrontMenuMapper的deleteById方法删除菜单
- validateFrontMenuExists: 校验传入id对应的菜单是否存在

- getFrontMenu: 调用FrontMenuMapper的selectById方法获得菜单，返回菜单实体
- getFrontMenuList: 调用FrontMenuMapper的selectBatchIds通过id列表获得菜单列表，返回菜单实体列表
- getFrontMenuPage: 调用FrontMenuMapper的selectPage方法传入分页条件获得菜单分页，返回菜单实体分页
- validFrontMenus: 检验传入的菜单是否存在，是否被禁用



- selectPage: 通过传入参数从分页查询，得到菜单分页返回
- selectList: 通过传入参数查询，得到菜单列表返回



- id: 编号
- name: 菜单名称
- path: 路由路径
- hideInMenu: 是否隐藏
- status: 菜单状态

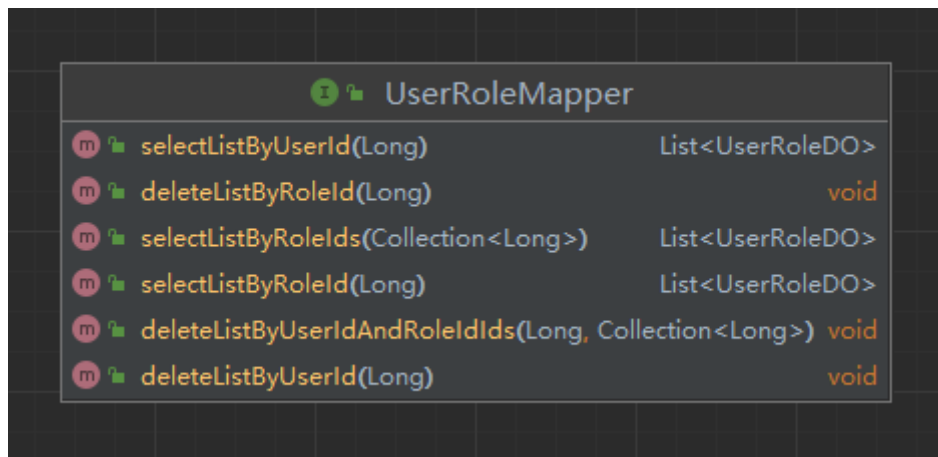
3.1.3.4 权限

FrontPermissionController		
f	tenantService	TenantService
f	frontPermissionService	FrontPermissionService
f	permissionService	PermissionService
f	roleService	RoleService
m	listAdminRoles(Long)	CommonResult<Set<Long>>
m	listRoleMenus(Long)	CommonResult<Set<Long>>
m	listSimpleUsersByRoleCode (String)	CommonResult<Set<FrontUserSimpleRespVO >>
m	assignRoleMenu (FrontPermissionAssignRoleMenuReqVO)	CommonResult<Boolean>
m	assignUserRole (FrontPermissionAssignUserRoleReqVO)	CommonResult<Boolean>
m	listUserIdsByRoleCode (String)	CommonResult<Set<Long>>

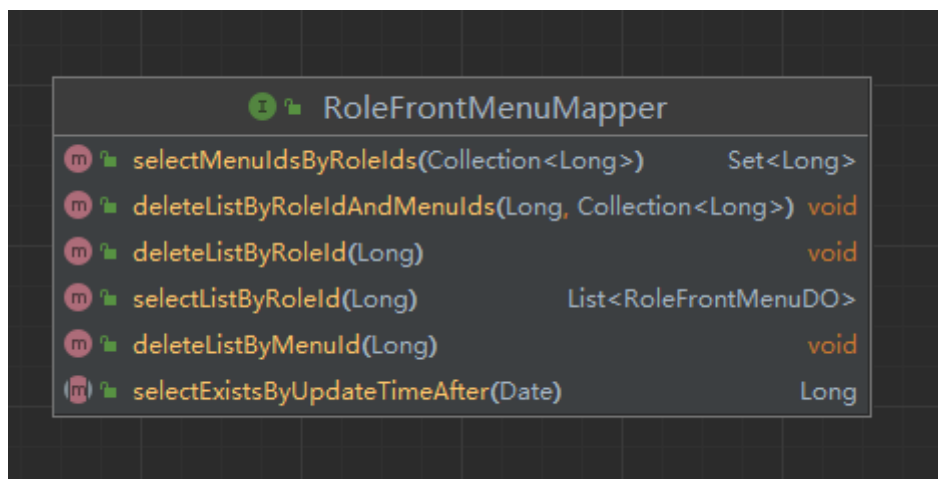
- listRoleMenus: 传入角色编号id, 调用FrontPermissionService的getRoleMenuIds方法, 返回角色拥有的前台菜单编号
- assignRoleMenu: 传入FrontPermissionAssignRoleMenuReqVO, 调用FrontPermissionService的assignRoleMenu赋予角色菜单, 返回是否赋予角色菜单成功
- listAdminRoles: 传入用户编号id, 调用FrontPermissionService的getUserRoleIdListByUserId, 返回用户拥有的角色编号列表
- assignUserRole: 传入FrontPermissionAssignUserRoleReqVO, 调用FrontPermissionService的assignUserRole赋予用户角色, 返回是否赋予用户角色成功
- listUserIdsByRoleCode: 传入角色code, 调用FrontPermissionService的getUserRoleIdListByRoleId, 返回拥有某个角色的用户编号集合
- listSimpleUsersByRoleCode: 传入角色code, 调用FrontPermissionService的getSimpleUserListByRoleId, 返回拥有某个角色的用户精简信息集合

FrontPermissionServiceImpl		
f	roleFrontMenuMapper	RoleFrontMenuMapper
f	frontMenuService	FrontMenuService
f	userRoleMapper	UserRoleMapper
f	userService	AdminUserService
f	roleService	RoleService
f	roleFrontMenuBatchInsertMapper	RoleFrontMenuBatchInsertMapper
f	userRoleBatchInsertMapper	UserRoleBatchInsertMapper
m	assignUserRole (Long, Set<Long>)	void
m	getSimpleUserListByRoleId (Long)	Set<FrontUserSimpleRespVO >
m	getRoleMenuIds (Long)	Set<Long>
m	assignRoleMenu (Long, Set<Long>)	void
m	getUserRoleIdListByUserId (Long)	Set<Long>
m	getUserRoleIdListByRoleId (Long)	Set<Long>
m	getRoleMenuList(Collection<Long>)	List<FrontMenuDO>

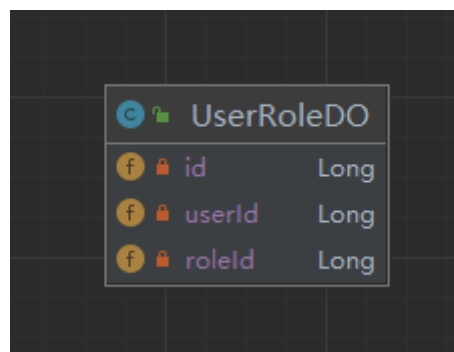
- getRoleMenuIds: 校验正确性后, 调用roleFrontMenuMapper的selectListByRoleId方法获得某个角色拥有的菜单, 返回这些菜单的id
- assignRoleMenu: 校验正确性后, 调用roleFrontMenuMapper的selectListByRoleId获得角色拥有菜单编号, 计算新增和删除的菜单编号, 执行新增和删除, 对于已经授权的菜单, 不用做任何处理。
- getUserRoleIdListByUserId: 校验正确性后, 调用userRoleMapper的selectListByUserId方法获得所有的用户角色关联实体的id, 返回这些编号id
- assignUserRole: 校验正确性后, 调用userRoleMapper的selectListByUserId获得角色拥有角色编号, 计算新增和删除的角色编号, 执行新增和删除。对于已经授权的角色, 不用做任何处理
- getUserRoleIdListByRoleId: 校验正确性后, 调用userRoleMapper的selectListByRoleId方法获得所有的用户角色关联实体的id, 返回这些编号id
- getSimpleUserListByRoleId: 校验正确性后, 调用userRoleMapper的selectListByRoleId方法获得所有的用户角色关联实体, 返回所需的用户信息VO
- getRoleMenuList: 校验正确性后, 调用RoleFrontMenuMapper的selectMenuIdsByRoleIds方法, 获得角色拥有的菜单关联, 返回这些角色菜单关联实体



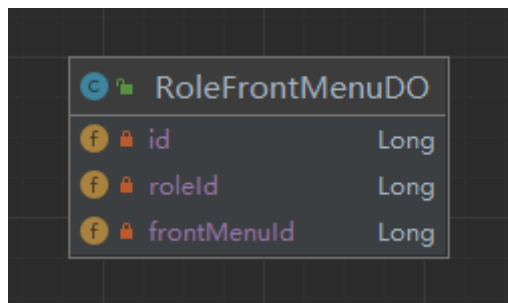
- selectListByUserId: 通过用户id查询用户角色关联, 得到用户角色关联实体列表返回
- selectListByRoleId: 通过角色id查询用户角色关联, 得到用户角色关联实体列表返回
- deleteListByUserIdAndRoleIdIds: 删除传入用户id和角色id的用户角色关联
- deleteListByUserId: 删除传入用户id的用户角色关联
- deleteListByRoleId: 删除传入角色id的用户角色关联
- selectListByRoleIds: 通过角色id列表查询用户角色关联, 得到用户角色关联实体列表返回



- selectListByRoleId: 通过角色id查询角色菜单关联, 得到角色菜单关联实体列表返回
- deleteListByRoleIdAndMenuIds: 删除传入角色id和菜单id的角色菜单关联
- deleteListByMenuId: 删除传入菜单id的角色菜单关联
- deleteListByRoleId: 删除传入角色id的角色菜单关联
- selectMenuIdsByRoleIds: 过角色id列表查询角色菜单关联, 得到角色菜单关联实体列表返回

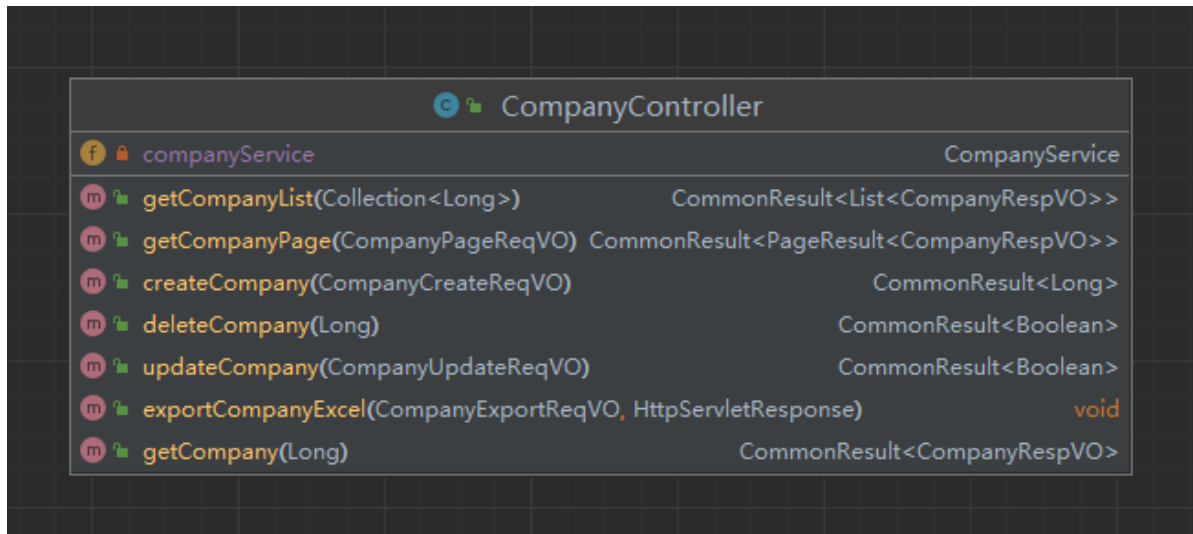


- id: 自增主键
- userId: 用户 ID
- roleId: 角色 ID

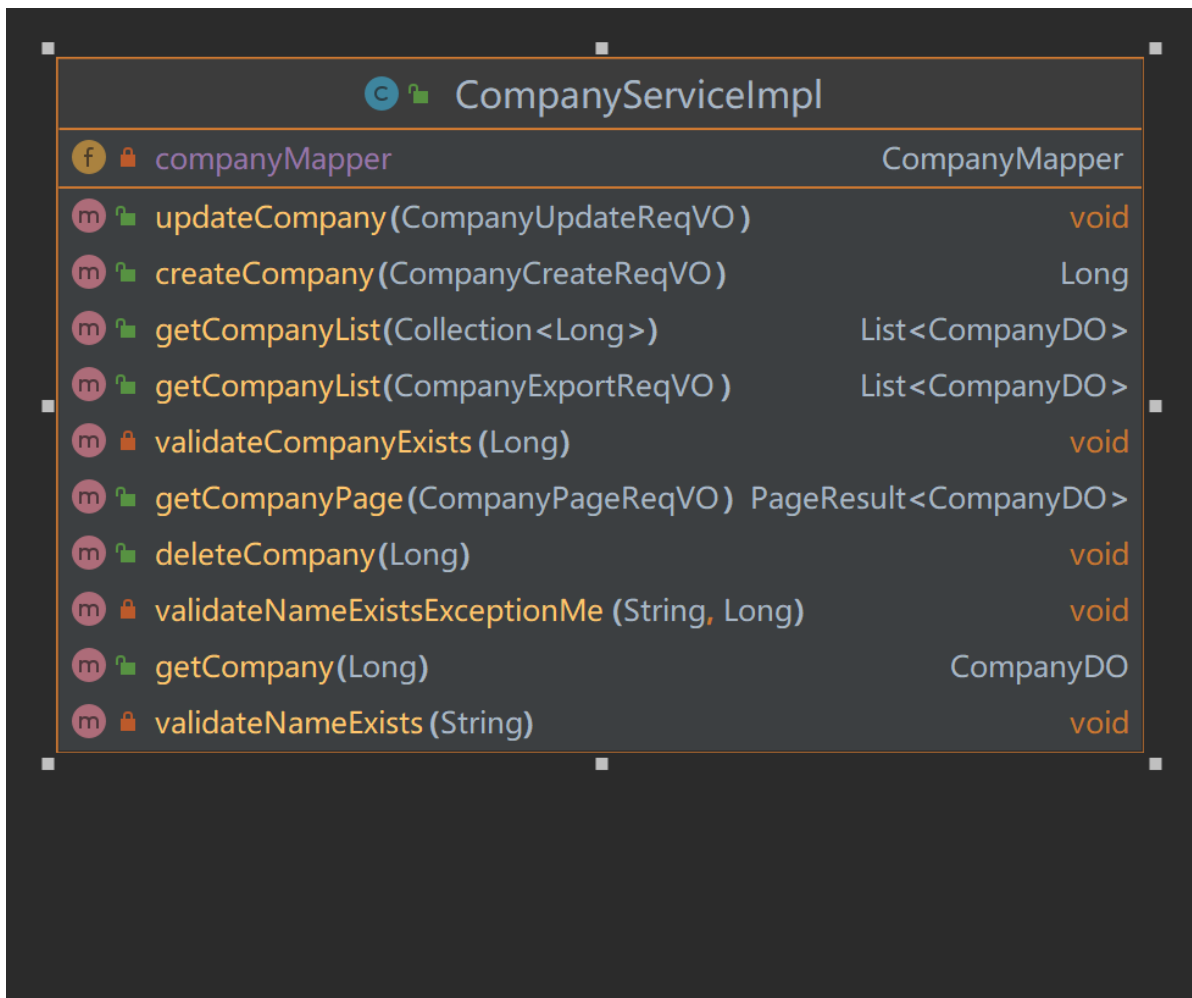


- id: 编号
- roleId: 角色编号
- frontMenuId: 菜单编号

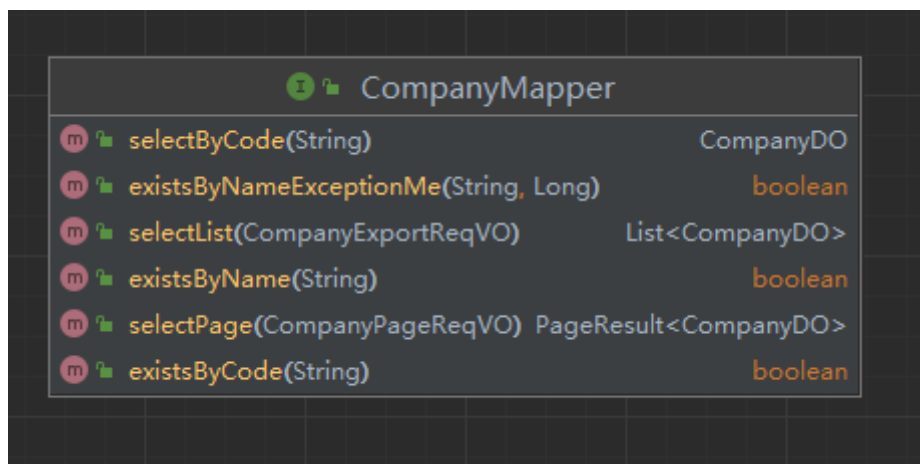
3.1.3.5 公司



- createCompany: 传入CompanyCreateReqVO, 调用CompanyService的createCompany方法新建公司, 返回公司编号id
- updateCompany: 传入CompanyUpdateReqVO, 调用CompanyService的updateCompany方法更新公司, 返回是否更新成功
- deleteCompany: 传入公司编号id, 调用CompanyService的deleteCompany方法删除公司, 返回是否删除成功
- getCompany: 传入公司编号id, 调用CompanyService的getCompany方法获得公司信息, 返回公司信息
- getCompanyList: 传入公司编号id列表, 调用CompanyService的getCompanyList方法获得公司列表, 返回公司信息列表
- getCompanyPage: 传入CompanyPageReqVO, 调用CompanyService的getCompanyPage方法获得公司信息分页, 返回公司分页

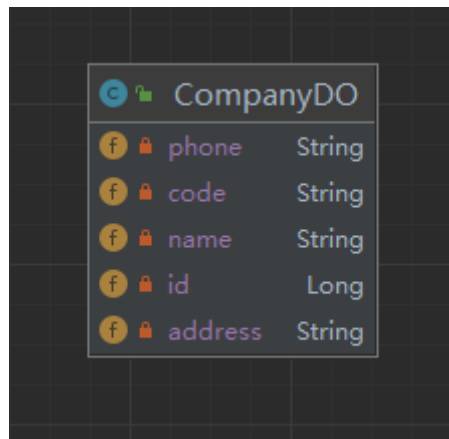


- `createCompany`: 校验正确性后, 调用`CompanyMapper`的`insert`方法向数据库里插入一个公司, 返回插入公司的主键id
- `updateCompany`: 校验正确性后, 调用`CompanyMapper`的`updateById`方法更新公司信息
- `deleteCompany`: 校验正确性后, 调用`CompanyMapper`的`deleteById`删除公司
- `validateNameExists`: 通过名字判断公司是否存在
- `validateNameExistsExceptionMe`: 通过名字判断是否存在同名公司
- `validateCompanyExists`: 通过编号id判断公司是否存在
- `getCompany`: 调用`CompanyMapper`的`selectById`方法获得公司, 返回公司实体
- `getCompanyList`: 调用`CompanyMapper`的`selectBatchIds`获得公司列表, 返回公司实体列表
- `getCompanyPage`: 调用`CompanyMapper`的`selectPage`获得公司分页, 返回公司实体分页
- `getCompanyList`: 调用`CompanyMapper`的`selectBatchIds`获得公司列表, 返回公司实体列表



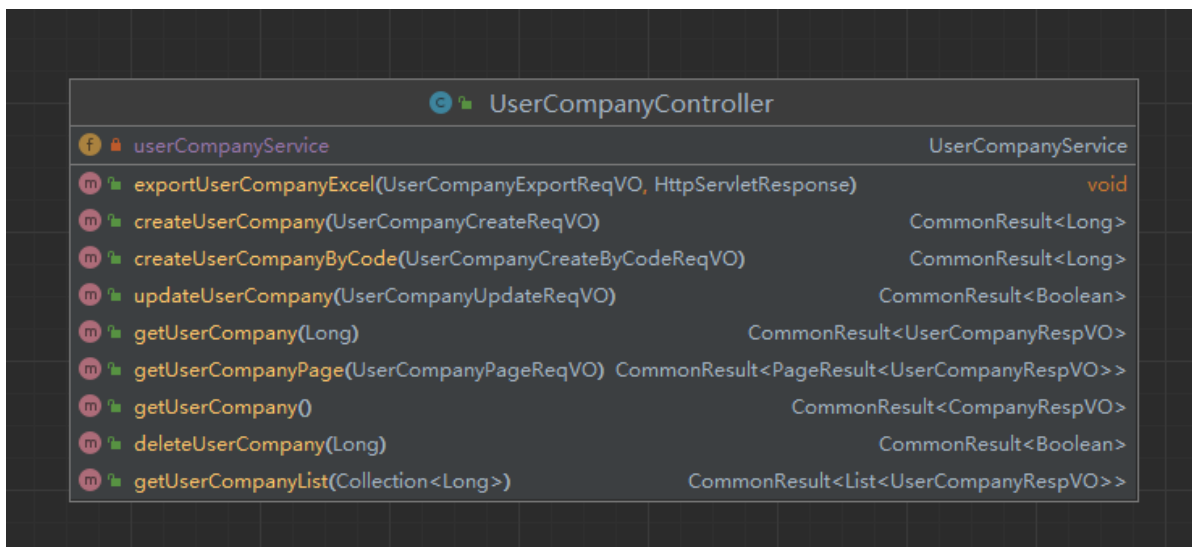
- `selectByCode`: 通过code查询公司, 得到公司实体后返回
- `existsByNameExceptionMe`: 查询是否存在名称为name但是id不为所给id的其他公司, 返回是否存在

- existsByName: 查询是否存在所给name的公司, 返回是否存在
- existsByCode: 查询是否存在所给code的公司, 返回是否存在
- selectPage: 通过传入参数从分页查询, 得到公司分页返回
- selectList: 通过传入参数查询列表, 得到公司列表返回













































- id: 编号
- name: 公司名称
- address: 公司地址
- phone: 公司联系方式
- code: 公司编码, 认证公司时使用

3.1.3.6 企业认证



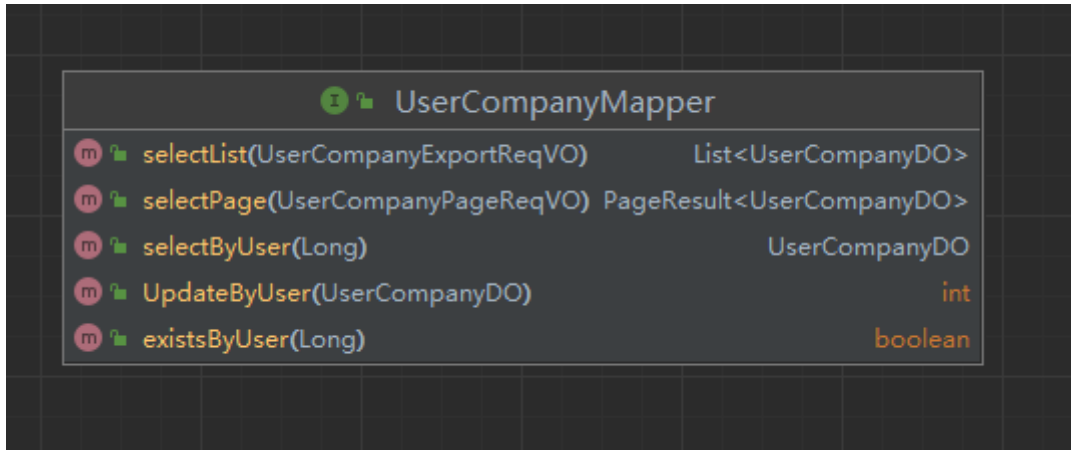
- getUserCompany: 通过调用UserCompanyService的getCompanyByUser方法获得当前用户公司信息, 返回公司信息
- createUserCompanyByCode: 用户使用, 传入UserCompanyCreateByCodeReqVO, 调用UserCompanyService的createUserCompanyByCode方法新建用户公司关联, 返回用户公司关联id
- createUserCompany: 管理人员使用, 传入UserCompanyCreateReqVO, 调用UserCompanyService的createUserCompany方法新建用户公司关联, 返回用户公司关联id
- updateUserCompany: 传入UserCompanyUpdateReqVO, 调用UserCompanyService的updateUserCompany方法更新用户公司关联, 返回是否更新用户公司关联成功
- deleteUserCompany: 传入用户公司关联编号id, 调用UserCompanyService的deleteUserCompany方法删除用户公司关联, 返回是否删除用户公司关联成功
- getUserCompany: 传入用户公司关联编号id, 调用UserCompanyService的getUserCompany方法获得用户公司关联, 返回获得用户公司关联

- getUserCompanyList: 传入用户公司关联编号id列表, 调用UserCompanyService的getUserCompanyList方法获得用户公司关联列表, 返回用户公司关联列表
- getUserCompanyPage: 传入UserCompanyPageReqVO, 调用UserCompanyService的getUserCompanyPage方法获得用户公司关联分页, 返回用户公司关联分页

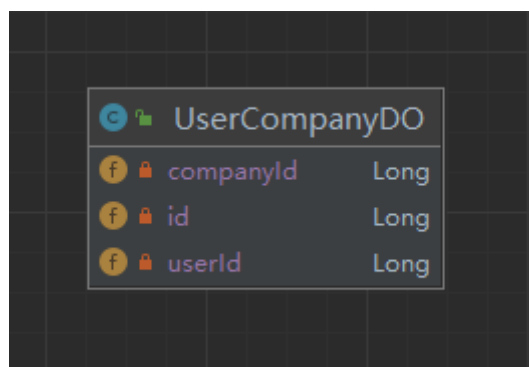
UserCompanyServiceImpl		
	 companyMapper	CompanyMapper
	 permissionService	PermissionService
	 userCompanyMapper	UserCompanyMapper
	 roleService	RoleService
	 assignCustomerRole (Long)	void
	 createUserCompanyByCode (UserCompanyCreateByCodeReqVO)	Long
	 deleteUserCompany (Long)	void
	 getCompanyByUser (Long)	CompanyDO
	 validateUserExists (Long)	void
	 validateCompanyExistsByCode (String)	void
	 validateUserCompanyExists (Long)	void
	 getUserCompanyList (UserCompanyExportReqVO)	List<UserCompanyDO >
	 createUserCompany (UserCompanyCreateReqVO)	Long
	 validateCompanyExists (Long)	void
	 updateUserCompany (UserCompanyUpdateReqVO)	void
	 deleteCustomerRole (Long)	void
	 getUserCompany (Long)	UserCompanyDO
	 getUserCompanyPage (UserCompanyPageReqVO)	PageResult<UserCompanyDO >
	 validateUserUnExists (Long)	void
	 assignNormalUserRole (Long)	void
	 getUserCompanyList (Collection<Long>)	List<UserCompanyDO >

- createUserCompany: 校验正确性后, 调用UserCompanyMapper的insert方法向数据库里插入一个用户公司关联, 返回插入用户公司关联的主键id
- updateUserCompany: 校验正确性后, 调用UserCompanyMapper的UpdateByUser方法更新用户公司关联
- deleteUserCompany: 校验正确性后, 调用UserCompanyMapper的deleteById方法删除用户公司关联
- validateUserCompanyExists: 校验用户公司关联是否存在
- getUserCompany: 调用UserCompanyMapper的selectById方法查询用户公司关联, 返回用户公司关联实体
- getUserCompanyList: 调用UserCompanyMapper的selectBatchIds方法查询用户公司关联列表, 返回用户公司关联实体列表
- getUserCompanyPage调用UserCompanyMapper的selectPage方法查询用户公司关联分页, 返回用户公司关联实体分页
- getUserCompanyList: 调用UserCompanyMapper的selectList方法查询用户公司关联列表, 返回用户公司关联实体列表
- createUserCompanyByCode: 校验正确性后, 通过code查找公司, 调用UserCompanyMapper的insert方法插入一条用户公司关联, 返回插入用户公司关联的主键id
- validateUserExists: 校验用户是否存在

- validateUserUnExists: 校验用户是否已经关联公司
- validateCompanyExists: 校验公司是否存在
- validateCompanyExistsByCode: 通过code校验公司是否存在
- getCompanyByUser: 校验正确性后, 调用UserCompanyMapper的selectByUser方法查询用户公司关联, 返回对应公司实体
- assignCustomerRole: 调用RoleService的assignUserRole方法为用户分配客户的角色
- assignNormalUserRole: 调用RoleService的assignUserRole方法为用户分配普通用户的角色
- deleteCustomerRole: 调用RoleService的assignUserRole方法为用户删除客户的角色



- selectByUser: 通过用户编号id查询用户公司关联, 返回用户公司关联实体
- existsByUser: 判断用户是否存在用户公司关联, 返回是否存在
- UpdateByUser: 更新对于用户的用户公司关联
- selectPage: 通过传入参数查询分页, 得到用户公司关联分页返回
- selectList: 通过传入参数查询列表, 得到用户公司关联列表返回



- id: 编号
- userId: 用户编号
- companyId: 公司编号

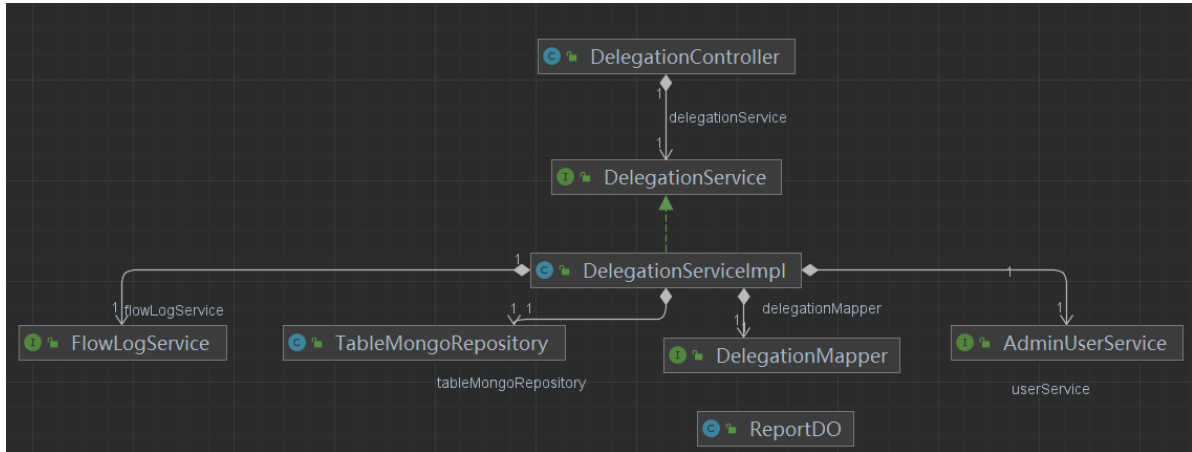
3.2 流程子模块

3.2.1 子系统说明

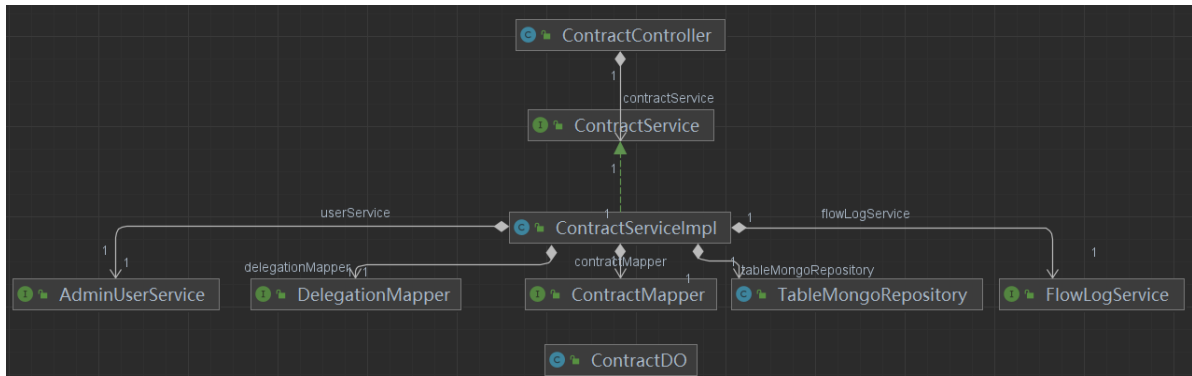
- 子模块包括: 委托管理、合同管理、样品管理、测试方案管理、测试报告管理共五部分
 - 委托管理: 实现对委托基本信息的增删查改、对委托相关表格和报价单的填写与查询
 - 合同管理: 实现对合同基本信息的增删查改、对合同相关表格的填写与查询
 - 样品管理: 实现对样品基本信息的增删查改
 - 测试方案管理: 实现对测试方案基本信息的增删查改、对测试方案相关表格的填写与查询
 - 测试报告管理: 实现对测试报告基本信息的增删查改、对测试报告相关表格的填写与查询

3.2.2 类图

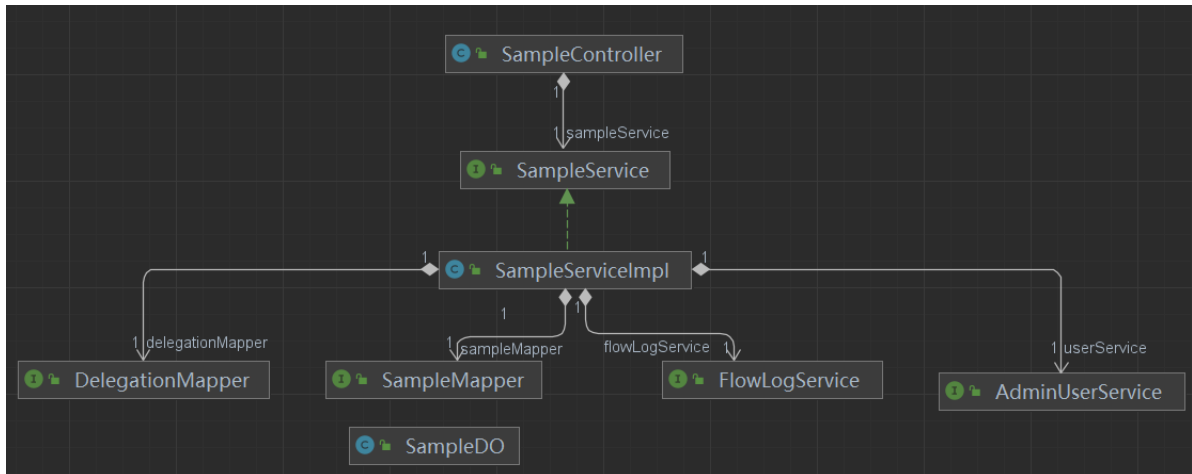
3.2.2.1 委托



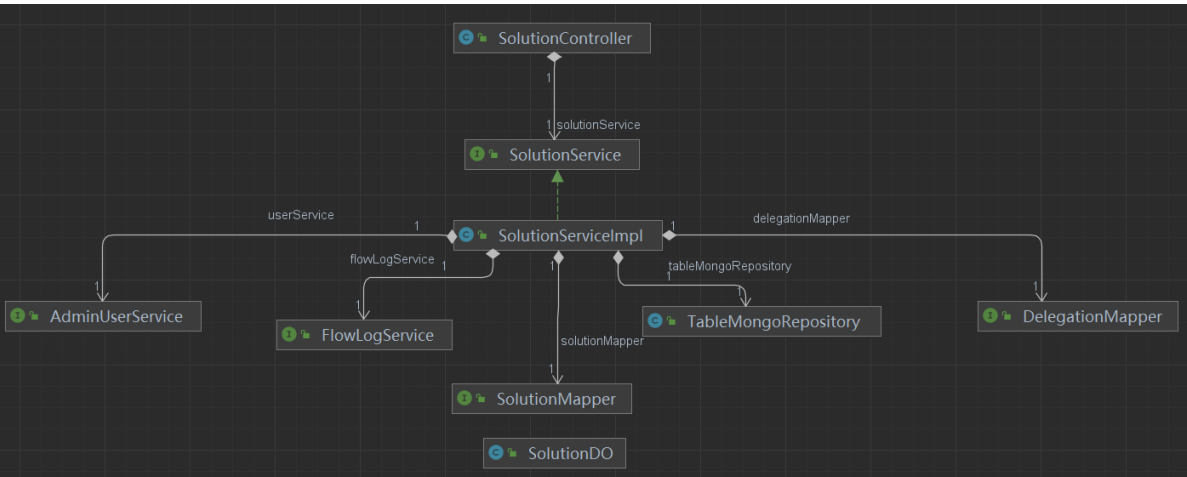
3.2.2.2 合同



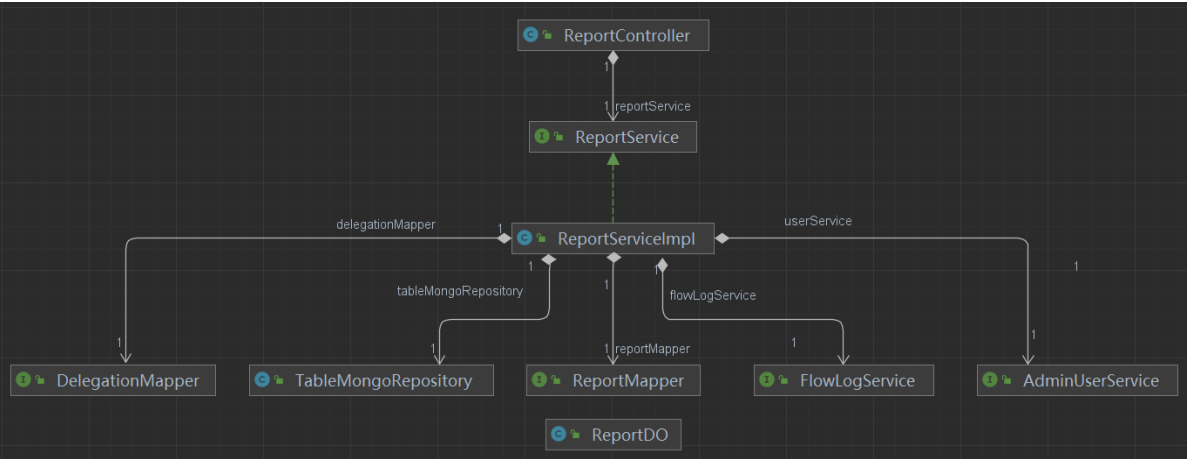
3.2.2.3 样品



3.2.2.4 测试方案



3.2.2.5 测试报告



3.2.3 类说明

3.2.3.1 委托

DelegationController		
m	getDelegationByCreator(Long)	CommonResult<List<DelegationRespVO>>
m	getDelegationTable2(String)	CommonResult<JSONObject>
m	distributeDelegationTesting(DelegationDistributeReqVO)	CommonResult<Boolean>
m	auditDelegationFailTest(DelegationAuditReqVO)	CommonResult<Boolean>
m	saveDelegationTable14(DelegationSaveTableReqVO)	CommonResult<Boolean>
m	getDelegation(Long)	CommonResult<DelegationRespVO>
m	getDelegationPage(DelegationPageReqVO)	CommonResult<PageResult<DelegationRespVO>>
m	getOffer(String)	CommonResult<JSONObject>
m	updateDelegation(DelegationUpdateReqVO)	CommonResult<Boolean>
m	getDelegationTable3(String)	CommonResult<JSONObject>
m	cancelDelegationAdmin(DelegationCancelReqVO)	CommonResult<Boolean>
m	getDelegationList(Collection<Long>)	CommonResult<List<DelegationRespVO>>
m	saveDelegationTable3(DelegationSaveTableReqVO)	CommonResult<Boolean>
m	auditDelegationFailMkt(DelegationAuditReqVO)	CommonResult<Boolean>
m	saveDelegationTable2(DelegationSaveTableReqVO)	CommonResult<Boolean>
m	exportDelegationExcel(DelegationExportReqVO, HttpServletResponse)	void
m	submitDelegation(DelegationSubmitReqVO)	CommonResult<Boolean>
m	submitOffer(OfferSubmitReqVO)	CommonResult<Boolean>
m	acceptOffer(OfferAcceptReqVO)	CommonResult<Boolean>
m	distributeDelegationMarketing(DelegationDistributeReqVO)	CommonResult<Boolean>
m	saveOffer(DelegationSaveTableReqVO)	CommonResult<Boolean>
m	rejectOffer(OfferRejectReqVO)	CommonResult<Boolean>
m	getDelegationProcessSimpleList(Long)	CommonResult<List<FlowLogSimpleResponseVO>>
m	deleteDelegation(Long)	CommonResult<Boolean>
m	auditDelegationSuccessTest(DelegationAuditReqVO)	CommonResult<Boolean>
m	getDelegationNotAccepted()	CommonResult<List<DelegationRespVO>>
m	auditDelegationSuccessMkt(DelegationAuditReqVO)	CommonResult<Boolean>
m	getDelegationTable14(String)	CommonResult<JSONObject>
m	cancelDelegationClient(DelegationCancelReqVO)	CommonResult<Boolean>
m	getDelegationByCurrentUser()	CommonResult<List<DelegationRespVO>>
m	getDelegationProcessList(Long)	CommonResult<List<FlowLogInstanceResponseVO>>
m	createDelegation(DelegationCreateReqVO)	CommonResult<Long>

- createDelegation: 传入DelegationCreateReqVO, 调用DelegationService的同名方法创建新委托。返回值为委托的编号。
- updateDelegation: 传入DelegationUpdateReqVO, 调用DelegationService的同名方法更新委托。返回值为是否更新成功。
- saveDelegationTable2: 传入DelegationSaveTableReqVO, 调用DelegationService的同名方法保存软件项目委托测试申请表。返回值为是否更新成功。
- saveDelegationTable3: 传入DelegationSaveTableReqVO, 调用DelegationService的同名方法保存委托测试软件功能列表。返回值为是否更新成功。
- saveDelegationTable14: 传入DelegationSaveTableReqVO, 调用DelegationService的同名方法保存软件文档评审表。返回值为是否更新成功。
- submitDelegation: 传入DelegationSubmitReqVO, 调用DelegationService的同名方法提交委托。返回值为是否提交成功。
- distributeDelegationMarketing: 传入DelegationDistributeReqVO, 调用DelegationService的同名方法分配委托给市场部人员。返回值为是否更新成功。

- distributeDelegationTesting: 传入DelegationDistributeReqVO, 调用DelegationService的同名方法分配委托给测试部人员。返回值为是否更新成功。
- auditDelegationSuccessMkt: 传入DelegationAuditReqVO, 调用DelegationService的同名方法审核委托通过。返回值为是否更新状态成功。
- auditDelegationSuccessTest: 传入DelegationAuditReqVO, 调用DelegationService的同名方法审核委托通过。返回值为是否更新状态成功。
- auditDelegationFailMkt: 传入DelegationAuditReqVO, 调用DelegationService的同名方法审核委托不通过。返回值为是否更新状态成功。
- auditDelegationFailTest: 传入DelegationAuditReqVO, 调用DelegationService的同名方法审核委托不通过。返回值为是否更新状态成功。
- saveOffer: 传入DelegationSaveTableReqVO, 调用DelegationService的同名方法保存报价单。返回值为是否更新成功。
- submitOffer: 传入OfferSubmitReqVO, 调用DelegationService的同名方法提交报价单。返回值为是否更新成功。
- rejectOffer: 传入OfferRejectReqVO, 调用DelegationService的同名方法不接受报价。返回值为是否更新成功。
- acceptOffer: 传入OfferAcceptReqVO, 调用DelegationService的同名方法接受报价。返回值为是否更新成功。
- cancelDelegationClient: 传入DelegationCancelReqVO, 调用DelegationService的同名方法取消委托。返回值为是否取消成功。
- cancelDelegationAdmin: 传入DelegationCancelReqVO, 调用DelegationService的同名方法取消委托。返回值为是否取消成功。
- getDelegation: 传入委托编号id字段, 调用DelegationService的同名方法获得委托。返回值为委托信息。
- getDelegationByCurrentUser: 调用DelegationService的同名方法获得当前用户的所有委托。返回值为委托列表。
- getDelegationTable2: 传入表格编号id字段, 调用DelegationService的同名方法获得软件项目委托测试申请表。返回值为json格式, 存放在data字段中, 包含表格内容。
- getDelegationTable3: 传入表格编号id字段, 调用DelegationService的同名方法获得委托测试软件功能列表。返回值为json格式, 存放在data字段中, 包含表格内容。
- getDelegationTable14: 传入表格编号id字段, 调用DelegationService的同名方法获得软件文档评审表。返回值为json格式, 存放在data字段中, 包含表格内容。
- getOffer: 传入报价单编号id字段, 调用DelegationService的同名方法获得报价单。返回值为json格式, 存放在data字段中, 包含表格内容。
- getDelegationList: 传入委托编号列表ids字段, 调用DelegationService的同名方法获得委托列表。返回值为委托列表。
- getDelegationProcessList: 传入委托编号id字段, 调用DelegationService的同名方法获得委托流程列表。返回值为委托流程列表。
- getDelegationProcessSimpleList: 传入委托编号id字段, 调用DelegationService的同名方法获得委托流程简略列表。返回值为委托流程简略列表。
- getDelegationPage: 传入DelegationPageReqVO, 调用DelegationService的同名方法获得委托分页。返回值为委托分页。

DelegationServiceImpl		
auditDelegationSuccessMkt(DelegationAuditReqVO)		void
getDelegationTable2(String)		JSONObject
getDelegationTable3(String)		JSONObject
getDelegationProcessList(Long)		List<FlowLogDO>
deleteDelegation(Long)		void
getDelegationsNotAccepted()		List<DelegationDO>
auditDelegationFailTest(DelegationAuditReqVO)		void
submitDelegation(DelegationSubmitReqVO)		void
auditDelegationSuccessTest(DelegationAuditReqVO)		void
saveOffer(DelegationSaveTableReqVO)		void
getDelegation(Long)		DelegationDO
acceptOffer(OfferAcceptReqVO)		void
saveDelegationTable3(DelegationSaveTableReqVO)		void
saveDelegationTable2(DelegationSaveTableReqVO)		void
getOffer(String)		JSONObject
distributeDelegation2Mkt(DelegationDistributeReqVO)		void
validateDelegationNameDuplicate(Long, String)		void
distributeDelegation2Test(DelegationDistributeReqVO)		void
getDelegationList(DelegationExportReqVO)		List<DelegationDO>
saveDelegationTable14(DelegationSaveTableReqVO)		void
submitOffer(OfferSubmitReqVO)		void
getDelegationsByCreator(Long)		List<DelegationDO>
getDelegationTable14(String)		JSONObject
getDelegationsByCurrentUser()		List<DelegationDO>
createDelegation(DelegationCreateReqVO)		Long
auditDelegationFailMkt(DelegationAuditReqVO)		void
rejectOffer(OfferRejectReqVO)		void
getDelegationList(Collection<Long>)		List<DelegationDO>
updateDelegation(DelegationUpdateReqVO)		void
cancelDelegationAdmin(DelegationCancelReqVO)		void
getDelegationPage(DelegationPageReqVO)		PageResult<DelegationDO>
cancelDelegationClient(DelegationCancelReqVO)		void









































- createDelegation: 校验委托名称是否重复, 调用DelegationMapper向数据库中插入一条委托记录, 返回委托id。
- updateDelegation: 校验正确性, 调用DelegationMapper更新委托的属性。
- saveDelegationTable2: 校验正确性, 调用TableMongoRepository保存软件项目委托测试申请表。
- saveDelegationTable3: 校验正确性, 调用TableMongoRepository保存委托测试软件功能列表。

- saveDelegationTable14: 校验正确性, 调用TableMongoRepository保存软件文档评审表。
- submitDelegation: 校验正确性, 调用DelegationMapper更新委托的状态。
- distributeDelegationMarketing: 校验正确性, 调用DelegationMapper更新委托的市场部负责人。
- distributeDelegationTesting: 校验正确性, 调用DelegationMapper更新委托的测试部负责人。
- auditDelegationSuccessMkt: 校验正确性, 调用DelegationMapper更新委托的状态。
- auditDelegationSuccessTest: 校验正确性, 调用DelegationMapper更新委托的状态。
- auditDelegationFailMkt: 校验正确性, 调用DelegationMapper更新委托的状态。
- auditDelegationFailTest: 校验正确性, 调用DelegationMapper更新委托的状态。
- saveOffer: 校验正确性, 调用TableMongoRepository保存报价单。
- submitOffer: 校验正确性, 调用DelegationMapper更新委托的状态。
- rejectOffer: 校验正确性, 调用DelegationMapper更新委托的状态和拒绝原因。
- acceptOffer: 校验正确性, 调用DelegationMapper更新委托的状态。
- cancelDelegationClient: 校验正确性, 调用DelegationMapper更新委托的状态和取消原因。
- cancelDelegationAdmin: 校验正确性, 调用DelegationMapper更新委托的状态和拒绝原因。
- getDelegation: 调用DelegationMapper的selectById方法获取委托并返回。
- getDelegationByCurrentUser: 调用DelegationMapper的selectList方法获取当前用户所有委托并返回。
- getDelegationTable2: 调用TableMongoRepository获取软件项目委托测试申请表。
- getDelegationTable3: 调用TableMongoRepository获取委托测试软件功能列表。
- getDelegationTable14: 调用TableMongoRepository获取软件文档评审表。
- getOffer: 调用TableMongoRepository获取报价单。
- getDelegationList: 调用DelegationMapper的selectBatchIds方法获取委托列表并返回。
- getDelegationProcessList: 调用FlowLogService的listLogs方法获取委托流程列表并返回。
- getDelegationPage: 调用DelegationMapper的selectPage方法获取委托分页并返回。

DelegationMapper	
validateDelegationStateBySample(Long, DelegationStateEnum[])	DelegationDO
validateDelegationStateBySolution(Long, DelegationStateEnum[])	DelegationDO
selectList(DelegationExportReqVO)	List<DelegationDO>
validateDelegationState(Long, DelegationStateEnum[])	DelegationDO
validateDelegationStateByReport(Long, DelegationStateEnum[])	DelegationDO
selectPage(DelegationPageReqVO)	PageResult<DelegationDO>
validateDelegationExists(Long)	DelegationDO
validateDelegationState(DelegationDO, DelegationStateEnum[])	DelegationDO
validateDelegationStateByContract(Long, DelegationStateEnum[])	DelegationDO

- validateDelegationExists: 通过传入的委托编号判断委托是否存在, 不存在则抛出异常, 返回查询到的委托。
- validateDelegationState: 判断传入的委托状态是否在传入的待判断状态中, 不在则抛出异常, 返回委托。
- validateDelegationStateByContract: 通过传入的合同编号判断对应委托是否存在及状态是否正确, 返回对应的委托。
- validateDelegationStateBySample: 通过传入的样品编号判断对应委托是否存在及状态是否正确, 返回对应的委托。
- validateDelegationStateBySolution: 通过传入的测试方案编号判断对应委托是否存在及状态是否正确, 返回对应的委托。

- validateDelegationStateByReport: 通过传入的测试报告编号判断对应委托是否存在及状态是否正确, 返回对应的委托。

DelegationDO		
 	<i>id</i>	Long
 	<i>launchTime</i>	Date
 	<i>solutionId</i>	Long
 	<i>sampleId</i>	Long
 	<i>table14Id</i>	String
 	<i>creatorId</i>	Long
 	<i>testingDeptStaffId</i>	Long
 	<i>name</i>	String
 	<i>offerId</i>	String
 	<i>cancelRemark</i>	String
 	<i>table2Id</i>	String
 	<i>url</i>	String
 	<i>table3Id</i>	String
 	<i>state</i>	Integer
 	<i>reportId</i>	Long
 	<i>marketRemark</i>	String
 	<i>offerRemark</i>	String
 	<i>contractId</i>	Long
 	<i>testingRemark</i>	String
 	<i>marketDeptStaffId</i>	Long

- id: 编号
- creatorId: 编号
- launchTime: 发起时间
- name: 委托名称
- table2Id: 软件项目委托测试申请表ID
- table3Id: 委托测试软件功能列表ID
- url: 文档材料url
- marketDeptStaffId: 分配的市场部人员id
- testingDeptStaffId: 分配的测试部人员id
- marketRemark: 市场部人员处理意见
- testingRemark: 测试部人员处理意见

- table14Id: 软件文档评审表ID
- offerId: 报价单ID
- offerRemark: 用户报价单意见
- contractId: 合同id
- sampleId: 样品id
- solutionId: 测试方案id
- reportId: 测试报告id
- state: 状态
- cancelRemark: 取消原因

3.2.3.2 合同

ContractController		
submitContractStaff	(ContractSubmitReqVO)	CommonResult<Boolean>
getContract	(Long)	CommonResult<ContractRespVO>
rejectContractClient	(ContractRejectReqVO)	CommonResult<Boolean>
acceptContractClient	(ContractAcceptReqVO)	CommonResult<Boolean>
updateContract	(ContractUploadDocReqVO)	CommonResult<Boolean>
createContract	(ContractCreateReqVO)	CommonResult<Long>
exportContractExcel	(ContractExportReqVO, HttpServletResponse)	void
getContractTable5	(String)	CommonResult<JSONObject>
getContractTable4	(String)	CommonResult<JSONObject>
rejectContractStaff	(ContractRejectReqVO)	CommonResult<Boolean>
saveContractTable5	(ContractSaveTableReqVO)	CommonResult<Boolean>
acceptContractStaff	(ContractAcceptReqVO)	CommonResult<Boolean>
getContractList	(Collection<Long>)	CommonResult<List<ContractRespVO>>
submitContractClient	(ContractSubmitReqVO)	CommonResult<Boolean>
saveContractTable4	(ContractSaveTableReqVO)	CommonResult<Boolean>
getContractPage	(ContractPageReqVO)	CommonResult<PageResult<ContractRespVO>>

- createContract: 传入ContractCreateReqVO, 调用ContractService的同名方法创建新合同。返回值为合同的编号。
- saveContractTable4: 传入ContractSaveTableReqVO, 调用ContractService的同名方法。返回值为是否保存成功。
- saveContractTable5: 传入ContractSaveTableReqVO, 调用ContractService的同名方法。返回值为是否保存成功。
- submitContractStaff: 传入ContractSubmitReqVO, 调用ContractService的同名方法。返回值为是否提交成功。
- acceptContractClient: 传入ContractAcceptReqVO, 调用ContractService的同名方法。返回值为是否更新成功。
- rejectContractClient: 传入ContractRejectReqVO, 调用ContractService的同名方法。返回值为是否更新成功。
- submitContractClient: 传入ContractSubmitReqVO, 调用ContractService的同名方法。返回值为是否更新成功。
- rejectContractStaff: 传入ContractRejectReqVO, 调用ContractService的同名方法。返回值为是否更新成功。
- acceptContractStaff: 传入ContractAcceptReqVO, 调用ContractService的同名方法。返回值为是否更新成功。

- updateContract: 传入ContractUploadDocReqVO, 调用ContractService的同名方法。返回值为是否保存成功。
- getContract: 传入合同编号id字段, 调用ContractService的同名方法获得合同。返回值为合同信息。
- getContractTable4: 传入表格编号id字段, 调用ContractService的同名方法获得软件委托测试合同。返回值为json格式, 存放在data字段中, 包含表格内容。
- getContractTable5: 传入表格编号id字段, 调用ContractService的同名方法获得软件项目委托测试保密协议。返回值为json格式, 存放在data字段中, 包含表格内容。
- getContractList: 传入合同编号列表ids字段, 调用ContractService的同名方法获得合同列表。返回值为合同列表。
- getContractPage: 传入ContractPageReqVO, 调用ContractService的同名方法获得合同分页。返回值为合同分页。

ContractServiceImpl		
m	acceptContractClient(ContractAcceptReqVO)	void
m	acceptContractStaff(ContractAcceptReqVO)	void
m	submitContractStaff(ContractSubmitReqVO)	void
m	getContractTable4(String)	JSONObject
m	rejectContractStaff(ContractRejectReqVO)	void
m	rejectContractClient(ContractRejectReqVO)	void
m	getContractList(ContractExportReqVO)	List<ContractDO>
m	deleteContract(Long)	void
m	submitContractClient(ContractSubmitReqVO)	void
m	getContract(Long)	ContractDO
m	createContract(ContractCreateReqVO)	Long
m	getContractTable5(String)	JSONObject
m	getContractPage(ContractPageReqVO)	PageResult<ContractDO>
m	validateContractExists(Long)	ContractDO
m	saveContractTable4(ContractSaveTableReqVO)	void
m	saveContractTable5(ContractSaveTableReqVO)	void
m	uploadDocument(ContractUploadDocReqVO)	void
m	getContractList(Collection<Long>)	List<ContractDO>

- createContract: 校验正确性, 调用ContractMapper向数据库中插入一条合同记录并更新对应委托的合同id字段, 返回合同id。
- saveContractTable4: 校验正确性, 调用TableMongoRepository保存软件委托测试合同。
- saveContractTable5: 校验正确性, 调用TableMongoRepository保存软件项目委托测试保密协议。
- submitContractStaff: 校验正确性, 调用DelegationMapper更新委托状态。
- submitContractClient: 校验正确性, 调用DelegationMapper更新委托状态。
- acceptContractClient: 校验正确性, 调用DelegationMapper更新委托状态。

- rejectContractClient: 校验正确性, 调用DelegationMapper更新委托状态。
- rejectContractStaff: 校验正确性, 调用DelegationMapper更新委托状态。
- acceptContractStaff: 校验正确性, 调用DelegationMapper更新委托状态。
- uploadDocument: 校验正确性, 调用DelegationMapper更新委托状态, 调用ContractMapper更新合同url字段。
- getContract: 调用ContractMapper的selectById方法获取合同并返回。
- getContractTable4: 调用TableMongoRepository获取软件委托测试合同。
- getContractTable5: 调用TableMongoRepository获取软件项目委托测试保密协议。
- getContractList: 调用ContractMapper的selectBatchIds方法获取合同列表并返回。
- getContractPage: 调用ContractMapper的selectPage方法获取合同分页并返回。

```
ContractMapper
selectPage(ContractPageReqVO) PageResult<ContractDO>
selectList(ContractExportReqVO) List<ContractDO>
```

- selectPage: 根据传入的查询参数获取合同分页并返回。

```
ContractDO
table5Id String
clientRemark String
url String
table4Id String
id Long
staffRemark String
```

- id: 合同编号。
- table4Id: 软件委托测试合同ID。
- table5Id: 软件项目委托测试保密协议ID。
- clientRemark: 客户审核合同意见。
- staffRemark: 市场部人员审核合同意见。
- url: 实体合同材料url。

3.2.3.3 样品

SampleController		
m	auditSampleFailResend(SampleAuditReqVO)	CommonResult<Boolean>
m	auditSampleSuccess(SampleAuditReqVO)	CommonResult<Boolean>
m	updateSample(SampleUpdateReqVO)	CommonResult<Boolean>
m	submitSample(SampleSubmitReqVO)	CommonResult<Boolean>
m	getSamplePage(SamplePageReqVO)	CommonResult<PageResult<SampleRespVO>>
m	createSample(SampleCreateReqVO)	CommonResult<Long>
m	exportSampleExcel(SampleExportReqVO, HttpServletResponse)	void
m	getSample(Long)	CommonResult<SampleRespVO>
m	deleteSample(Long)	CommonResult<Boolean>
m	getSampleList(Collection<Long>)	CommonResult<List<SampleRespVO>>










- createSample: 传入SampleCreateReqVO, 调用SampleService的同名方法创建样品。返回值为样品的编号。
- updateSample: 传入SampleUpdateReqVO, 调用SampleService的同名方法更新样品。返回值为是否更新成功。
- submitSample: 传入SampleSubmitReqVO, 调用SampleService的同名方法提交样品。返回值为是否提交成功。
- auditSampleSuccess: 传入SampleAuditReqVO, 调用SampleService的同名方法样品验收通过。返回值为是否更新成功。
- auditSampleFailResend: 传入SampleAuditReqVO, 调用SampleService的同名方法样品验收不通过。返回值为是否更新成功。
- getSample: 传入样品编号id字段, 调用SampleService的同名方法获得样品。返回值为样品信息。
- getSampleList: 传入样品编号列表ids字段, 调用SampleService的同名方法获得样品列表。返回值为样品列表。
- getSamplePage: 传入SamplePageReqVO, 调用SampleService的同名方法获得样品分页。返回值为样品分页。

SampleServiceImpl		
m	getSampleList(SampleExportReqVO)	List<SampleDO>
m	auditSample(SampleAuditReqVO, Boolean)	DelegationDO
m	createSample(SampleCreateReqVO)	Long
m	getSample(Long)	SampleDO
m	submitSample(SampleSubmitReqVO)	void
m	deleteSample(Long)	void
m	getSampleList(Collection<Long>)	List<SampleDO>
m	getSamplePage(SamplePageReqVO)	PageResult<SampleDO>
m	validateSampleState(Long, SampleStateEnum)	SampleDO
m	auditSampleSuccess(SampleAuditReqVO)	void
m	updateSample(SampleUpdateReqVO)	void
m	validateSampleExists(Long)	SampleDO
m	auditSampleFail(SampleAuditReqVO)	void

- createSample: 校验正确性, 调用SampleMapper向数据库中插入一条样品记录并更新对应委托的样品id字段, 返回样品id。
- updateSample: 校验正确性, 调用SampleMapper的updateById方法更新样品信息。
- submitSample: 校验正确性, 调用DelegationMapper更新委托状态。
- auditSampleSuccess: 校验正确性, 调用DelegationMapper更新委托状态。
- auditSampleFail: 校验正确性, 调用DelegationMapper更新委托状态。
- getSample: 调用SampleMapper的selectById方法获取样品并返回。
- getSampleList: 调用SampleMapper的selectBatchIds方法获取样品列表并返回。
- getSamplePage: 调用SampleMapper的selectPage方法获取样品分页并返回。













SampleMapper		
m	selectPage(SamplePageReqVO)	PageResult<SampleDO>
m	selectList(SampleExportReqVO)	List<SampleDO>

- selectPage: 根据传入的查询参数获取样品分页并返回。

SampleDO		
	<i>type</i>	String
	<i>url</i>	String
	<i>id</i>	Long
	<i>auditorId</i>	Long
	<i>remark</i>	String
	<i>state</i>	Integer
	<i>information</i>	String
	<i>processType</i>	String
	<i>modifyRemark</i>	String

- id: 样品编号
- type: 样品上传方式, 如果在线上上传则填写为线上, 其余需说明方式的具体信息
- processType: 处理方式
- url: 如果样品为线上上传, 需要填写样品的url
- information: 样品信息
- auditorId: 审核人id, 只能为选定的市场部或者测试部两个人中的一个
- remark: 审核意见
- modifyRemark: 修改意见
- state: 样品状态

3.2.3.4 测试方案

SolutionController		
	<i>createSolution</i> (SolutionCreateReqVO)	CommonResult<Long>
	<i>exportSolutionExcel</i> (SolutionExportReqVO, HttpServletResponse)	void
	<i>getSolutionTable6</i> (String)	CommonResult<JSONObject>
	<i>getSolutionList</i> (Collection<Long>)	CommonResult<List<SolutionRespVO>>
	<i>auditFail</i> (SolutionSubmitReqVO)	CommonResult<Boolean>
	<i>getSolutionTable13</i> (String)	CommonResult<JSONObject>
	<i>saveSolutionTable6</i> (SolutionSaveTableReqVO)	CommonResult<Boolean>
	<i>getSolutionPage</i> (SolutionPageReqVO)	CommonResult<PageResult<SolutionRespVO>>
	<i>submitSolutionTable6</i> (SolutionSubmitReqVO)	CommonResult<Boolean>
	<i>getSolution</i> (Long)	CommonResult<SolutionRespVO>
	<i>saveSolutionTable13</i> (SolutionSaveTableReqVO)	CommonResult<Boolean>
	<i>auditSuccess</i> (SolutionSubmitReqVO)	CommonResult<Boolean>

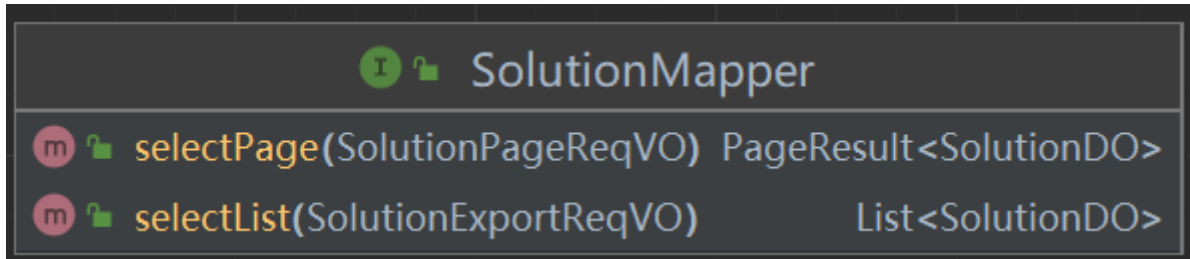
- createSolution: 传入SolutionCreateReqVO, 调用SolutionService的同名方法创建测试方案。返回值为测试方案的编号。
- saveSolutionTable6: 传入SolutionSaveTableReqVO, 调用SolutionService的同名方法保存软件测试方案表格。返回值为是否保存成功。

- saveSolutionTable13: 传入SolutionSaveTableReqVO, 调用SolutionService的同名方法保存测试方案评审表。返回值为是否保存成功。
- submitSolutionTable6: 传入SolutionSubmitReqVO, 调用SolutionService的同名方法提交软件测试方案表。返回值为是否更新成功。
- auditSuccess: 传入SolutionSubmitReqVO, 调用SolutionService的同名方法测试方案审核通过。返回值为是否更新成功。
- auditFail: 传入SolutionSubmitReqVO, 调用SolutionService的同名方法测试方案审核未通过。返回值为是否更新成功。
- getSolutionTable6: 传入表格编号id字段, 调用SolutionService的同名方法获得软件测试方案表格。返回值为json格式, 存放在data字段中, 包含表格内容。
- getSolutionTable13: 传入表格编号id字段, 调用SolutionService的同名方法获得测试方案评审表。返回值为json格式, 存放在data字段中, 包含表格内容。
- getSolution: 传入测试方案编号id字段, 调用SolutionService的同名方法获得测试方案, 返回值为测试方案信息。
- getSolutionList: 传入测试方案编号列表ids字段, 调用SolutionService的同名方法获得测试方案列表, 返回值为测试方案列表。
- getSolutionPage: 传入SolutionPageReqVO, 调用SolutionService的同名方法获得测试方案分页, 返回值为测试方案分页。

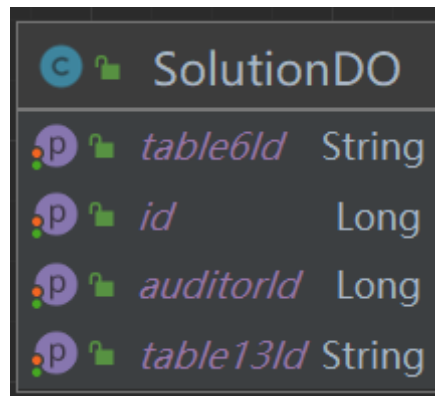
SolutionServiceImpl		
m	createSolution(SolutionCreateReqVO)	Long
m	saveSolutionTable13(SolutionSaveTableReqVO)	void
m	getSolutionTable13(String)	JSONObject
m	getSolution(Long)	SolutionDO
m	saveSolutionTable6(SolutionSaveTableReqVO)	void
m	auditSolution(SolutionSubmitReqVO)	DelegationDO
m	updateSolution(SolutionUpdateReqVO)	void
m	auditSuccess(SolutionSubmitReqVO)	void
m	getSolutionList(SolutionExportReqVO)	List<SolutionDO>
m	deleteSolution(Long)	void
m	submitSolutionTable6(SolutionSubmitReqVO)	void
m	getSolutionList(Collection<Long>)	List<SolutionDO>
m	auditFail(SolutionSubmitReqVO)	void
m	getSolutionTable6(String)	JSONObject
m	validateSolutionExists(Long)	SolutionDO
m	getSolutionPage(SolutionPageReqVO)	PageResult<SolutionDO>

- createSolution: 校验正确性, 调用SolutionMapper向数据库中插入一条测试方案记录并更新对应委托的测试方案id字段, 返回测试方案id。
- saveSolutionTable6: 调用TableMongoRepository保存软件测试方案表格。
- saveSolutionTable13: 调用TableMongoRepository保存测试方案评审表。

- submitSolutionTable6: 校验正确性, 调用DelegationMapper更新委托状态。
- auditSuccess: 校验正确性, 调用DelegationMapper更新委托状态。
- auditFail: 校验正确性, 调用DelegationMapper更新委托状态。
- getSolutionTable6: 调用TableMongoRepository获得软件测试方案表格。
- getSolutionTable13: 调用TableMongoRepository获得测试方案评审表。
- updateSolution: 校验正确性, 调用SolutionMapper更新样品信息。
- getSolution: 调用SolutionMapper的selectById方法获得测试方案并返回。
- getSolutionList: 调用SolutionMapper的selectBatchIds方法获得测试方案列表并返回。
- getSolutionPage: 调用SolutionMapper的selectPage方法获得测试方案分页并返回。





















































- selectPage: 根据传入的查询参数获取测试方案分页并返回。



- id: 测试方案编号
- table6Id: 软件测试方案ID
- table13Id: 测试方案评审表ID
- auditorId: 质量部审核人id

3.2.3.5 测试报告

ReportController

  acceptReportClient (ReportAcceptReqVO)	CommonResult <Boolean >
  archiveReport (ReportArchiveReqVO)	CommonResult <Boolean >
  saveReportTable7 (ReportSaveTableReqVO)	CommonResult <Boolean >
  rejectReportSignatory (ReportRejectReqVO)	CommonResult <Boolean >
  saveReportTable11 (ReportSaveTableReqVO)	CommonResult <Boolean >
  acceptReportSignatory (ReportAcceptReqVO)	CommonResult <Boolean >
  getReportTable10 (String)	CommonResult <JSONObject >
  acceptReportManager (ReportAcceptReqVO)	CommonResult <Boolean >
  exportReportExcel (ReportExportReqVO , HttpServletResponse)	void
  rejectReportManager (ReportRejectReqVO)	CommonResult <Boolean >
  getReportTable7 (String)	CommonResult <JSONObject >
  sendReport (ReportSendReqVO)	CommonResult <Boolean >
  saveReportTable9 (ReportSaveTableReqVO)	CommonResult <Boolean >
  getReportList (Collection<Long>)	CommonResult <List<ReportRespVO >>
  rejectReportClient (ReportRejectReqVO)	CommonResult <Boolean >
  getReportPage (ReportPageReqVO)	CommonResult <PageResult <ReportRespVO >>
  getReportTable8 (String)	CommonResult <JSONObject >
  submitReport (ReportSubmitReqVO)	CommonResult <Boolean >
  getReport (Long)	CommonResult <ReportRespVO >
  saveReportTable8 (ReportSaveTableReqVO)	CommonResult <Boolean >
  receiveReport (ReportReceiveReqVO)	CommonResult <Boolean >
  createReport (ReportCreateReqVO)	CommonResult <Long >
  getReportTable11 (String)	CommonResult <JSONObject >
  getReportTable9 (String)	CommonResult <JSONObject >
  saveReportTable10 (ReportSaveTableReqVO)	CommonResult <Boolean >

- createReport: 传入ReportCreateReqVO, 调用ReportService的同名方法创建测试报告, 返回值为测试报告编号。
- saveReportTable7: 传入ReportSaveTableReqVO, 调用ReportService的同名方法保存软件测试报告, 返回值为是否保存成功。
- saveReportTable8: 传入ReportSaveTableReqVO, 调用ReportService的同名方法保存测试用例 (电子记录), 返回值为是否保存成功。
- saveReportTable9: 传入ReportSaveTableReqVO, 调用ReportService的同名方法保存软件测试记录 (电子记录), 返回值为是否保存成功。
- saveReportTable10: 传入ReportSaveTableReqVO, 调用ReportService的同名方法保存测试报告检查表, 返回值为是否保存成功。
- saveReportTable11: 传入ReportSaveTableReqVO, 调用ReportService的同名方法保存软件测试问题清单 (电子记录), 返回值为是否保存成功。
- submitReport: 传入ReportSubmitReqVO, 调用ReportService的同名方法提交测试报告, 返回值为是否提交成功。
- acceptReportManager: 传入ReportAcceptReqVO, 调用ReportService的同名方法审核测试报告通过, 返回值为是否提交成功。
- rejectReportManager: 传入ReportRejectReqVO, 调用ReportService的同名方法审核测试报告不通过, 返回值为是否提交成功。

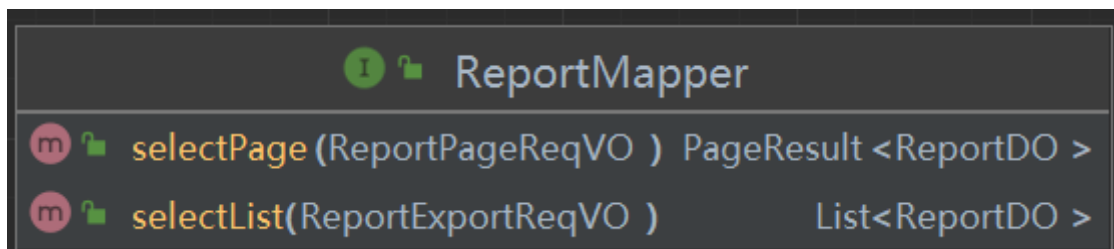
- acceptReportClient: 传入ReportAcceptReqVO, 调用ReportService的同名方法审核测试报告通过, 返回值为是否提交成功。
- rejectReportClient: 传入ReportRejectReqVO, 调用ReportService的同名方法审核测试报告不通过, 返回值为是否提交成功。
- acceptReportSignatory: 传入ReportAcceptReqVO, 调用ReportService的同名方法审核测试报告通过, 返回值为是否提交成功。
- rejectReportSignatory: 传入ReportRejectReqVO, 调用ReportService的同名方法审核测试报告不通过, 返回值为是否提交成功。
- archiveReport: 传入ReportArchiveReqVO, 调用ReportService的同名方法归档测试报告, 返回值为是否更新成功。
- sendReport: 传入ReportSendReqVO, 调用ReportService的同名方法发送测试报告, 返回值为是否发送成功。
- receiveReport: 传入ReportReceiveReqVO, 调用ReportService的同名方法确认接收测试报告, 返回值为是否确认成功。
- getReport: 传入测试报告编号id字段, 调用ReportService的同名方法获得测试报告, 返回值为测试报告信息。
- getReportTable7: 传入表格编号id字段, 调用ReportService的同名方法获得软件测试报告, 返回值为json格式, 存放在data字段中, 包含表格内容。
- getReportTable8: 传入表格编号id字段, 调用ReportService的同名方法获得测试用例(电子记录), 返回值为json格式, 存放在data字段中, 包含表格内容。
- getReportTable9: 传入表格编号id字段, 调用ReportService的同名方法获得软件测试记录(电子记录), 返回值为json格式, 存放在data字段中, 包含表格内容。
- getReportTable10: 传入表格编号id字段, 调用ReportService的同名方法获得测试报告检查表, 返回值为json格式, 存放在data字段中, 包含表格内容。
- getReportTable11: 传入表格编号id字段, 调用ReportService的同名方法获得软件测试问题清单(电子记录), 返回值为json格式, 存放在data字段中, 包含表格内容。
- getReportList: 传入测试报告列表编号列表ids字段, 调用ReportService的同名方法获得测试报告列表, 返回值为测试报告列表。
- getReportPage: 传入ReportPageReqVO, 调用ReportService的同名方法获得测试报告分页, 返回值为测试报告分页。

ReportServiceImpl

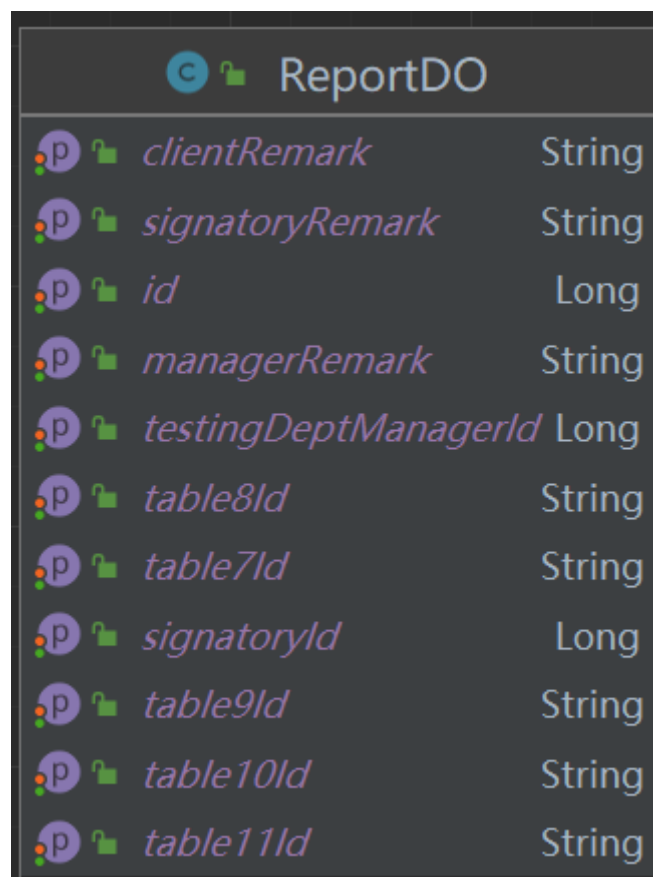
m	sendReport (ReportSendReqVO)	void
m	receiveReport (ReportReceiveReqVO)	void
m	auditReportSignatory (Long, String)	DelegationDO
m	deleteReport (Long)	void
m	getReport (Long)	ReportDO
m	getReportPage (ReportPageReqVO)	PageResult <ReportDO >
m	rejectReportClient (ReportRejectReqVO)	void
m	getReportList (Collection<Long>)	List<ReportDO >
m	submitReport (ReportSubmitReqVO)	void
m	rejectReportSignatory (ReportRejectReqVO)	void
m	validateReportExists (Long)	ReportDO
m	getReportList (ReportExportReqVO)	List<ReportDO >
m	acceptReportSignatory (ReportAcceptReqVO)	void
m	saveReportTable11 (ReportSaveTableReqVO)	void
m	createReport (ReportCreateReqVO)	Long
m	getReportTable (String, String)	JSONObject
m	saveReportTable7 (ReportSaveTableReqVO)	void
m	saveReportTable10 (ReportSaveTableReqVO)	void
m	archiveReport (ReportArchiveReqVO)	void
m	acceptReportManager (ReportAcceptReqVO)	void
m	acceptReportClient (ReportAcceptReqVO)	void
m	saveReportTable8 (ReportSaveTableReqVO)	void
m	auditReportManager (Long, String)	DelegationDO
m	rejectReportManager (ReportRejectReqVO)	void
m	saveReportTable9 (ReportSaveTableReqVO)	void
m	auditReportClient (Long, String)	DelegationDO

- createReport: 校验正确性，调用ReportMapper向数据库中插入一条测试报告记录并更新对应委托的测试报告id字段，返回测试报告id。
- saveReportTable7: 校验正确性，调用TableMongoRepository保存软件测试报告。
- saveReportTable8: 校验正确性，调用TableMongoRepository保存测试用例（电子记录）。
- saveReportTable9: 校验正确性，调用TableMongoRepository保存软件测试记录（电子记录）。

- saveReportTable10: 校验正确性, 调用TableMongoRepository保存测试报告检查表。
- saveReportTable11: 校验正确性, 调用TableMongoRepository保存软件测试问题清单 (电子记录) 。
- submitReport: 校验正确性, 调用DelegationMapper更新委托状态。
- acceptReportManager: 校验正确性, 调用DelegationMapper更新委托状态。
- rejectReportManager: 校验正确性, 调用DelegationMapper更新委托状态。
- acceptReportClient: 校验正确性, 调用DelegationMapper更新委托状态。
- rejectReportClient: 校验正确性, 调用DelegationMapper更新委托状态。
- acceptReportSignatory: 校验正确性, 调用DelegationMapper更新委托状态。
- rejectReportSignatory: 校验正确性, 调用DelegationMapper更新委托状态。
- archiveReport: 校验正确性, 调用DelegationMapper更新委托状态。
- sendReport: 校验正确性, 调用DelegationMapper更新委托状态。
- receiveReport: 校验正确性, 调用DelegationMapper更新委托状态。
- getReport: 调用DelegationMapper的selectById方法获得测试报告并返回。
- getReportTable: 调用TableMongoRepository获得测试报告相关表格。
- getReportList: 调用DelegationMapper的selectBatchIds方法获得测试报告列表并返回。
- getReportPage: 调用DelegationMapper的selectPage方法获得测试报告分页并返回。



- selectPage: 根据传入的查询参数获取测试方案分页并返回。



- id: 测试报告编号
- table7Id: 软件测试报告ID
- table8Id: 测试用例 (电子记录) ID
- table9Id: 软件测试记录 (电子记录) ID

- table11Id: 软件测试问题清单（电子记录）ID
- testingDeptManagerId: 测试部主管id
- managerRemark: 测试部主管审核意见
- table10Id: 测试报告检查表ID
- signatoryRemark: 签字人审核意见
- signatoryId: 签字人id
- clientRemark: 客户意见

3.3 用户子模块

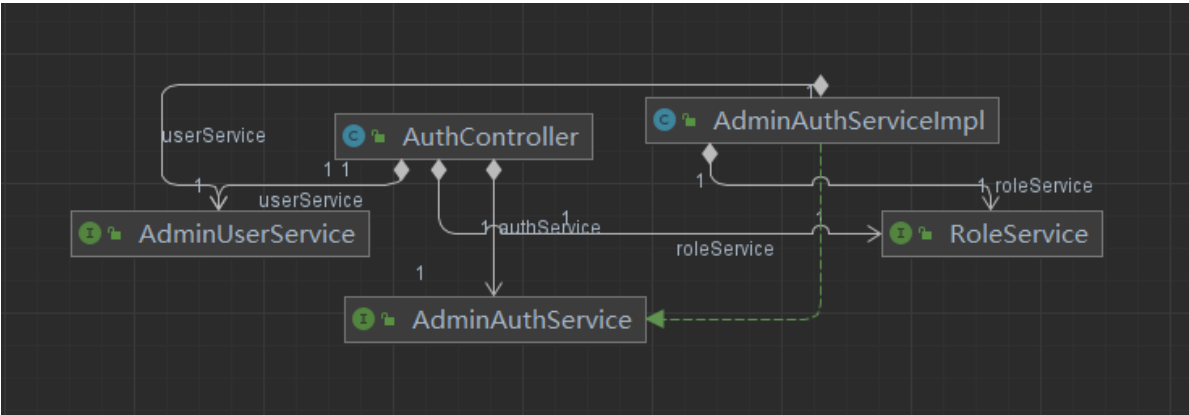
3.3.1 子系统说明

子模块包括：登录认证、个人中心两部分

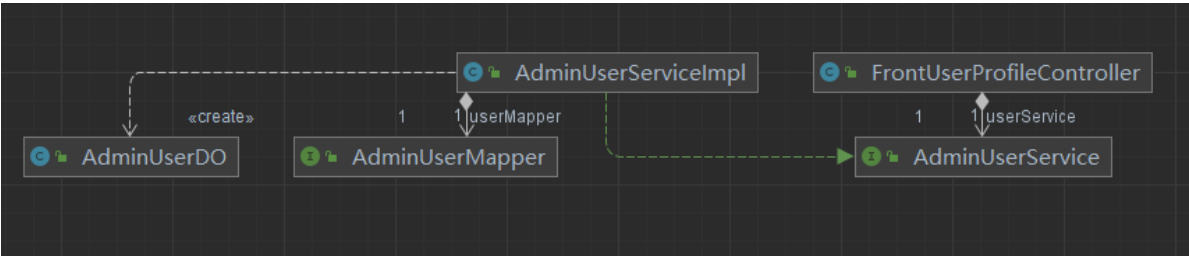
- 登录认证：实现用户的各种登录方式
- 个人中心：实现修改用户个人资料

3.3.2 类图

3.3.2.1 登录认证



3.3.2.2 个人中心



3.3.3 类说明

3.3.3.1 登录认证

AuthController		
f	authService	AdminAuthService
f	userService	AdminUserService
f	frontPermissionService	FrontPermissionService
f	socialUserService	SocialUserService
f	log	Logger
f	roleService	RoleService
f	permissionService	PermissionService
m	sendSmsCode (AuthSendSmsReqVO)	CommonResult<Boolean>
m	login(AuthMobileLoginReqVO)	CommonResult<AuthMobileLoginRespVO>
m	listSimpleMenus()	CommonResult<List<AuthSimpleMenuRespVO>>
m	login(AuthLoginReqVO)	CommonResult<AuthLoginRespVO>
m	resetPassword (AuthResetPasswordReqVO)	CommonResult<Boolean>
m	smsLogin (AuthSmsLoginReqVO)	CommonResult<AuthSmsLoginRespVO>
m	register (AuthRegisterReqVO)	CommonResult<AuthRegisterRespVO>

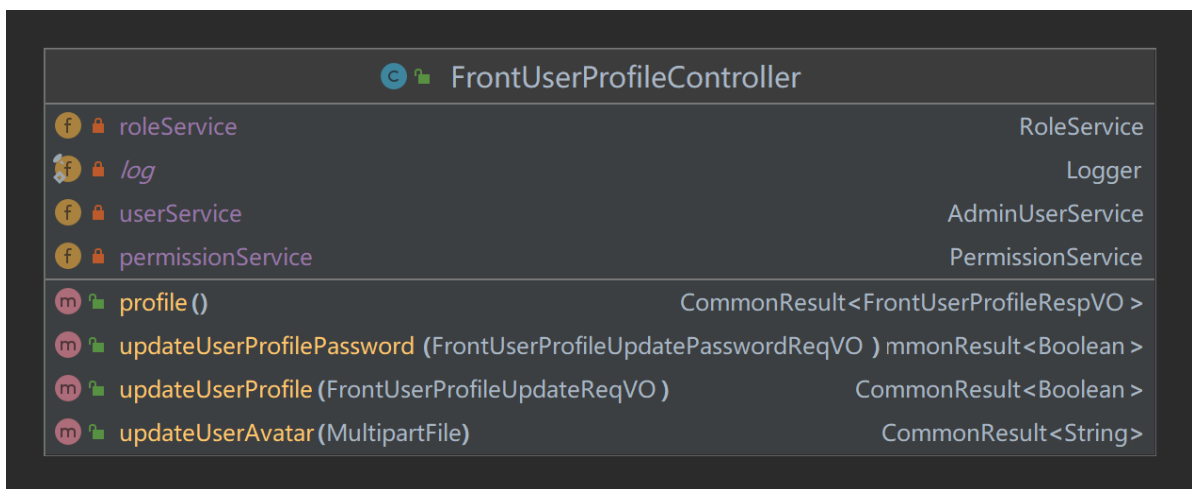
- login (/login) : 传入AuthLoginReqVO, 调用AdminAuthService的login方法使用账号密码登录
- login (/mobile-login) : 传入AuthMobileLoginReqVO, 调用AdminAuthService的mobileLogin方法使用使用手机号密码登录
- smsLogin: 传入AuthSmsLoginReqVO, 调用AdminAuthService的smsLogin方法使用手机 + 验证码登录
- sendSmsCode: 传入AuthSendSmsReqVO, 调用AdminAuthService的sendSmsCode方法发送手机验证码
- resetPassword: 传入AuthResetPasswordReqVO, 调用AdminAuthService的resetPassword方法重置密码
- register: 传入AuthRegisterReqVO, 调用AdminAuthService的register方法注册
- listSimpleMenus: 通过安全框架获得当前登录账号的角色id数组, 通过FrontPermissionService的getRoleMenuList方法获得菜单实体并返回

AdminAuthServiceImpl		
f	permissionService	PermissionService
f	captchaService	CaptchaService
f	smsCodeApi	SmsCodeApi
f	userSessionService	UserSessionService
f	loginLogService	LoginLogService
f	log	Logger
f	userService	AdminUserService
f	roleService	RoleService
f	authenticationManager	AuthenticationManager
m	createLoginLog (String, LoginLogTypeEnum, LoginResultEnum)	void
m	getUserRoleIds (Long)	Set<Long>
m	register (AuthRegisterReqVO, String, String)	String
m	buildLoginUser (AdminUserDO)	LoginUser
m	resetPassword (AuthResetPasswordReqVO)	void
m	verifyCaptcha (AuthLoginReqVO)	void
m	assignNormalUserRole (Long)	void
m	smsLogin (AuthSmsLoginReqVO, String, String)	String
m	mobileLogin (AuthMobileLoginReqVO, String, String)	String
m	logout (String)	void
m	refreshLoginUserCache (String, LoginUser)	LoginUser
m	loadUserByUsername (String)	UserDetails
m	getUserType ()	UserTypeEnum
m	createLogoutLog (Long, String)	void
m	sendSmsCode (Long, AuthSendSmsReqVO)	void
m	login0 (String, String)	LoginUser
m	createUserSessionAfterLoginSuccess (LoginUser, LoginLogTypeEnum)	
m	verifyTokenAndRefresh (String)	LoginUser
m	login (AuthLoginReqVO, String, String)	String
m	checkUserIfExists (String)	AdminUserDO

- loadUserByUsername: 通过用户名创建UserDetails, 供登录使用
- login: 校验正确性后, 调用login0函数登录, 并将创建token存到Redis中返回token的编号
- mobileLogin: 校验正确性后, 调用login函数登录, 并将创建token存到Redis中返回token的编号
- verifyCaptcha: 校验验证码是否正确
- login0: 校验正确性后, 调用安全框架接口执行登录

- createLoginLog: 生成登录日志并写入, 供登录接口使用
- createUserSessionAfterLoginSuccess: 生成登录日志并写入
- logout: 校验正确性后, 登出, 并记录登出日志
- getUserType: 获得用户类型
- createLogoutLog: 生成登出日志并写入, 供登录接口使用
- verifyTokenAndRefresh: 获得 LoginUser并刷新 LoginUser 缓存
- refreshLoginUserCache: 每 1/3 的 Session 超时时间, 刷新 LoginUser 缓存, 重新加载 AdminUserDO 信息, 刷新 LoginUser 缓存
- buildLoginUser: 生成LoginUser
- sendSmsCode: 调用smsCodeApi发送手机验证码
- smsLogin: 校验验证码后, 校验正确性后, 执行登录
- resetPassword: 调用smsCodeApi的useSmsCode方法使用验证码再调用AdminUserService的updateUserPassword方法更新密码
- checkUserIfExists: 校验用户是否存在
- register: 调用AdminUserService的createUser方法注册用户
- assignNormalUserRole: 为刚注册的用户分配普通用户角色

3.3.3.2 个人中心



- profile: 通过调用AdminUserService的getUser方法获得用户基本信息, 再通过RoleService的getRoles方法获得用户角色, 整合后返回
- updateUserProfile: 通过调用AdminUserService的updateUserProfile方法更新用户基本信息, 返回是否更新成功
- updateUserProfilePassword: 通过调用AdminUserService的updateUserPassword方法修改密码, 返回是否修改密码成功
- updateUserAvatar: 通过调用AdminUserService的updateUserAvatar方法上传用户个人头像, 返回是否修上传用户个人头像头像

4 系统异常处理设计

4.1 统一响应

后端提供 RESTful API 给前端时, 需要响应前端 API 调用是否成功:

- 如果成功, 成功的数据是什么。后续, 前端会将数据渲染到页面上
- 如果失败, 失败的原因是什么。一般, 前端会将原因弹出提示给用户

因此, 需要有**统一响应**, 而不能是每个接口定义自己的风格。一般来说, 统一响应返回信息如下:

- 成功时, 返回成功的状态码 + 数据
- 失败时, 返回失败的状态码 + 错误提示

4.1.1 CommonResult

本项目在实践时，将状态码放在 Response Body 响应内容中返回。一共有 3 个字段，通过 CommonResult 定义如下：

```
@Data
public class CommonResult<T> implements Serializable {

    /**
     * 错误码
     *
     * @see ErrorCode#getCode()
     */
    5 usages
    private Integer code;

    /**
     * 返回数据
     */
    2 usages
    private T data;

    /**
     * 错误提示, 用户可阅读
     *
     * @see ErrorCode#getMsg() ()
     */
    4 usages
    private String msg;
```

响应实例如下：

```
// 成功响应
{
    code: 0,
    data: {
        id: 1,
        username: "NJU"
    }
}

// 失败响应
{
    code: 123,
    message: "密码错误"
}
```


4.2 异常处理

RESTful API 发生异常时，需要拦截 Exception 异常，转换成**统一响应**的格式，否则前端无法处理。

4.2.1 Spring MVC 的异常

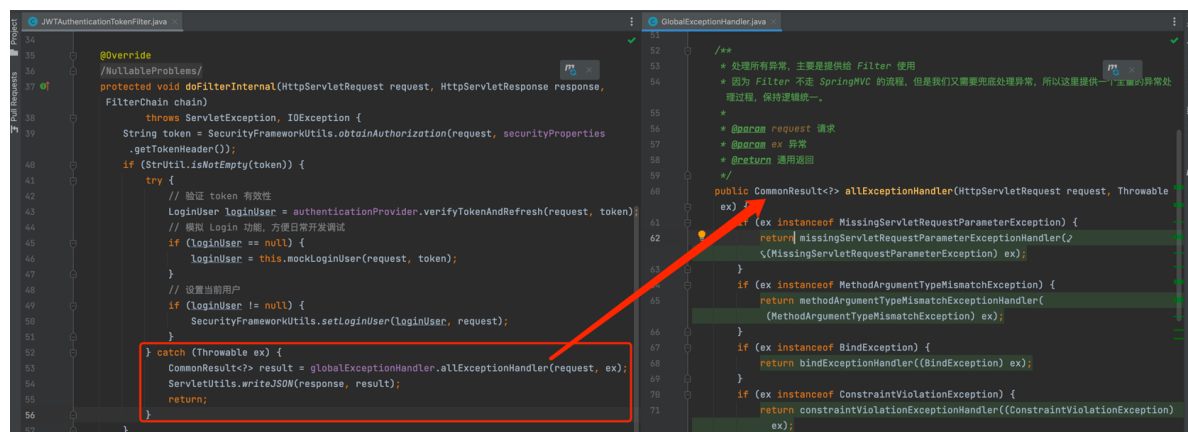
在 Spring MVC 中，通过 `@ControllerAdvice` + `@ExceptionHandler` 注解，声明将指定类型的异常，转换成对应的 `CommonResult` 响应。示例代码如下：

```
/**
 * 处理 Spring Security 权限不足的异常
 *
 * 来源是，使用 @PreAuthorize 注解，AOP 进行权限拦截
 */
1 usage
@ControllerAdvice(value = AccessDeniedException.class)
public CommonResult<?> accessDeniedExceptionHandler(HttpServletRequest req, AccessDeniedException ex) {
    log.warn("[accessDeniedExceptionHandler][userId({}) 无法访问 url({})]", WebFrameworkUtils.getLoginUserId(req),
        req.getRequestURL(), ex);
    return CommonResult.error(FORBIDDEN);
}

/**
 * 处理业务异常 ServiceException
 *
 * 例如说，商品库存不足，用户手机号已存在。
 */
1 usage
@ControllerAdvice(value = ServiceException.class)
public CommonResult<?> serviceExceptionHandler(ServiceException ex) {
    log.info("[serviceExceptionHandler]", ex);
    return CommonResult.error(ex.getCode(), ex.getMessage());
}
```

4.2.2 Filter 的异常

在请求被 Spring MVC 处理之前，需要先经过 Filter 处理，此时发生异常时，无法通过 `@ExceptionHandler` 注解来处理。只能通过 `try catch` 的方式来实现，示例代码如下：



```
JWTAuthenticationTokenFilter.java
@Override
@Nullable
protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response,
    FilterChain chain)
    throws ServletException, IOException {
    String token = SecurityFrameworkUtils.obtainAuthorization(request, securityProperties
        .getTokenHeader());
    if (StringUtil.isEmpty(token)) {
        try {
            // 验证 token 有效性
            LoginUser loginUser = authenticationProvider.verifyTokenAndRefresh(request, token);
            // 模拟 Login 功能，方便日常开发调试
            if (loginUser == null) {
                loginUser = this.mockLoginUser(request, token);
            }
            // 设置当前用户
            if (loginUser != null) {
                SecurityFrameworkUtils.setLoginUser(loginUser, request);
            }
        } catch (Throwable ex) {
            CommonResult<?> result = globalExceptionHandler.allExceptionHandler(request, ex);
            ServletUtils.writeJSON(response, result);
            return;
        }
    }
}

GlobalExceptionHandler.java
/**
 * 处理所有异常，主要是提供给 Filter 使用
 *
 * 因为 Filter 不走 SpringMVC 的流程，但是我们又要彻底处理异常，所以这里提供一个主调的异常处
    理过程，保持逻辑统一。
 *
 * @param request 请求
 * @param ex 异常
 * @return 通用返回
 */
public CommonResult<?> allExceptionHandler(HttpServletRequest request, Throwable
    ex) {
    if (ex instanceof MissingServletRequestParameterException) {
        return missingServletRequestParameterExceptionHandler(
            (MissingServletRequestParameterException) ex);
    }
    if (ex instanceof MethodArgumentTypeMismatchException) {
        return methodArgumentTypeMismatchExceptionHandler(
            (MethodArgumentTypeMismatchException) ex);
    }
    if (ex instanceof BindException) {
        return bindExceptionHandler((BindException) ex);
    }
    if (ex instanceof ConstraintViolationException) {
        return constraintViolationExceptionHandler((ConstraintViolationException)
            ex);
    }
}
```

4.3 业务异常

在 Service 发生业务异常，如委托名称已存在、相关表格未填写等情况时，使用 `ServiceException` 统一业务异常，里面有错误码和错误提示，然后进行 `throw` 抛出。

4.3.1 ServiceException

定义 ServiceException 异常类，继承 RuntimeException 异常类（非受检），用于定义业务异常。代码如下：

```
/**
 * 业务逻辑异常 Exception
 */
@Data
@EqualsAndHashCode(callSuper = true)
public final class ServiceException extends RuntimeException {

    /**
     * 业务错误码，对应 CommonResult 的 code 字段
     *
     * @see ServiceErrorCodeRange
     */
    4 usages
    private Integer code;

    /**
     * 错误提示，对应 CommonResult 的 msg 字段
     */
    4 usages
    private String message;
}
```

4.3.2 ServiceExceptionUtil

在 Service 需抛出业务异常时，通过调用 ServiceExceptionUtil 的 `exception(ErrorCode errorCode, Object... params)` 方法来构建 ServiceException 异常，然后使用 `throw` 进行抛出。代码如下：

```
public static ServiceException exception(ErrorCode errorCode) {
    String messagePattern = MESSAGES.getOrDefault(errorCode.getCode(), errorCode.getMsg());
    return exception0(errorCode.getCode(), messagePattern);
}

public static ServiceException exception(ErrorCode errorCode, Object... params) {
    String messagePattern = MESSAGES.getOrDefault(errorCode.getCode(), errorCode.getMsg());
    return exception0(errorCode.getCode(), messagePattern, params);
}
```

4.4 错误码

错误码，对应 ErrorCode 类，枚举项目中的错误，**全局唯一**，方便定位错误。

```

public class ErrorCode {

    /**
     * 错误码
     */
    1 usage
    private final Integer code;

    /**
     * 错误提示
     */
    1 usage
    private final String msg;

    public ErrorCode(Integer code, String message) {
        this.code = code;
        this.msg = message;
    }

}

```

4.4.1 错误码分类

错误码分成两类：全局的系统错误码、模块的业务错误码。

4.4.1.1 系统错误码

全局的系统错误码，使用 0-999 错误码段，和 HTTP 响应状态码对应。

系统错误码定义在 GlobalErrorCodeConstants 类，代码如下：

```

public interface GlobalErrorCodeConstants {

    ErrorCode SUCCESS = new ErrorCode( code: 0, message: "成功");

    // ===== 客户端错误段 =====

    15 usages
    ErrorCode BAD_REQUEST = new ErrorCode( code: 400, message: "请求参数不正确");
    6 usages
    ErrorCode UNAUTHORIZED = new ErrorCode( code: 401, message: "账号未登录");
    5 usages
    ErrorCode FORBIDDEN = new ErrorCode( code: 403, message: "没有该操作权限");
    1 usage
    ErrorCode NOT_FOUND = new ErrorCode( code: 404, message: "请求未找到");
    1 usage
    ErrorCode METHOD_NOT_ALLOWED = new ErrorCode( code: 405, message: "请求方法不正确");
    1 usage
    ErrorCode LOCKED = new ErrorCode( code: 423, message: "请求失败，请稍后重试"); // 并发请求，不允许
    1 usage
    ErrorCode TOO_MANY_REQUESTS = new ErrorCode( code: 429, message: "请求过于频繁，请稍后重试");

    // ===== 服务端错误段 =====

    5 usages
    ErrorCode INTERNAL_SERVER_ERROR = new ErrorCode( code: 500, message: "系统异常");

```

4.4.1.2 业务错误码

模块的业务错误码，按照模块分配错误码的区间，避免模块之间的错误码冲突。

① 业务错误码一共 10 位，分成 4 段，在 ServiceErrorCodeRange 分配，规则与代码如下图：

```

/**
 * 业务异常的错误码区间，解决：解决各模块错误码定义，避免重复，在此只声明不做实际使用
 *
 * 一共 10 位，分成四段
 *
 * 第一段，1 位，类型
 *     1 - 业务级别异常
 *     x - 预留
 * 第二段，3 位，系统类型
 *     001 - 用户系统
 *     002 - 商品系统
 *     003 - 订单系统
 *     004 - 支付系统
 *     005 - 优惠券系统
 *     ... - ...
 * 第三段，3 位，模块
 *     不限制规则。
 *     一般建议，每个系统里面，可能有多模块，可以再去做分段。以用户系统为例子：
 *         001 - OAuth2 模块
 *         002 - User 模块
 *         003 - MobileCode 模块
 * 第四段，3 位，错误码
 *     不限制规则。
 *     一般建议，每个模块自增。
 *
 * @author 芋道源码
 */
public class ServiceErrorCodeRange {

```

② 每个业务模块，定义自己的 ErrorCodeConstants 错误码枚举类。以 `yudao-module-system` 模块举例子，代码如下：

```

// ===== 委托 1002020000 =====
2 usages
ErrorCode DELEGATION_NOT_EXISTS = new ErrorCode( code: 1002020000, message: "委托不存在");
1 usage
ErrorCode DELEGATION_NAME_DUPLICATE = new ErrorCode( code: 1002020001, message: "委托名称不能重复");
1 usage
ErrorCode DELEGATION_TABLE_NOT_EXISTS = new ErrorCode( code: 1002020002, message: "委托表格不存在");
4 usages
ErrorCode DELEGATION_STATE_ERROR = new ErrorCode( code: 1002020003, message: "委托状态错误");
1 usage
ErrorCode DELEGATION_TABLE14_NOT_FILLED = new ErrorCode( code: 1002020004, message: "软件文档评审表未填写");
2 usages
ErrorCode DELEGATION_CANCELED = new ErrorCode( code: 1002020005, message: "委托已被取消");

// ===== 公司认证模块 1002022000 =====
4 usages
ErrorCode COMPANY_NOT_EXISTS = new ErrorCode( code: 1002022000, message: "公司不存在");
3 usages
ErrorCode USER_COMPANY_NOT_EXISTS = new ErrorCode( code: 1002022001, message: "用户公司关联不存在");
2 usages
ErrorCode COMPANY_NAME_EXISTS = new ErrorCode( code: 1002022002, message: "公司已存在");
1 usage
ErrorCode USER_COMPANY_EXISTS = new ErrorCode( code: 1002022003, message: "用户公司关联已存在");

```

