项目设计文档

车一晗 181250009

黄婉红 181840096

纳思彧 181250107

王博 181250133

2021 年 6 月 18 日

摘要

本文档为 Heap 小组在 2021 年春学期《软件工程与计算三》课程作业迭代三中为项目所撰写的项目设计文档。

目录

1 引言

1.1 编写目的

本文档提供 COIN 知识图谱系统的软件架构概览，采用若干架构视图描述系统的不同方面，以便表示构造系统所需要的重要架构决策。

1.2 对象与范围

本文档的读者是 Heap 团队内部的开发和管理人员，参考了 RUP 的《软件架构文档模板》，用于指导下一循环的代码开发和测试工作。

1.3 参考文献

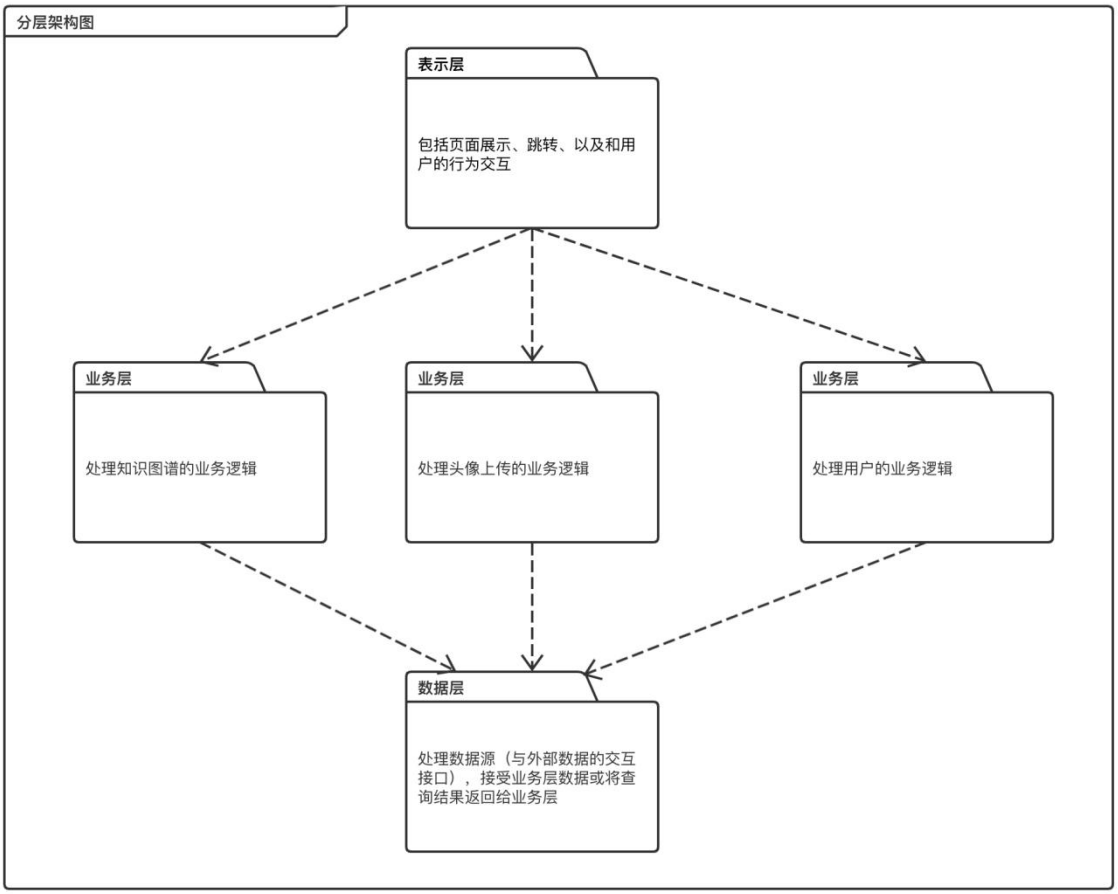
《软件需求规格说明书》, Heap;
《软件架构文档模板》, Rational Software Corporation;

1.4 名词与术语

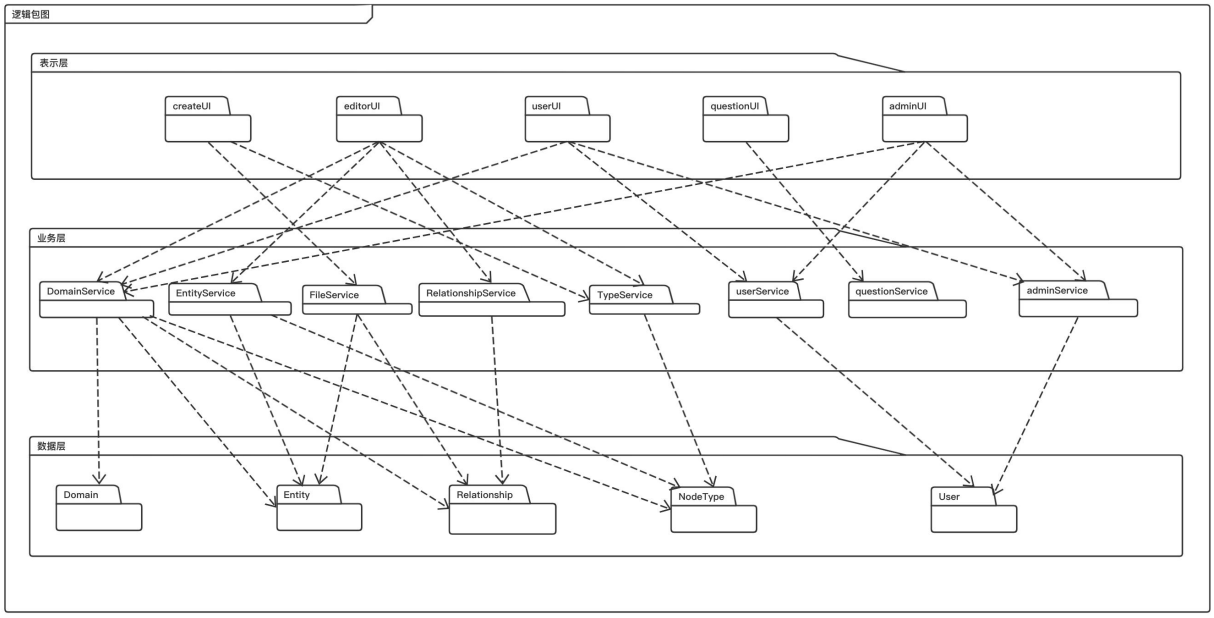
COIN：COnstructing and vIsualizing kNowledge graph 知识图谱可视化系统

2　逻辑视角

2.1 分层架构包图



2.2 逻辑包图

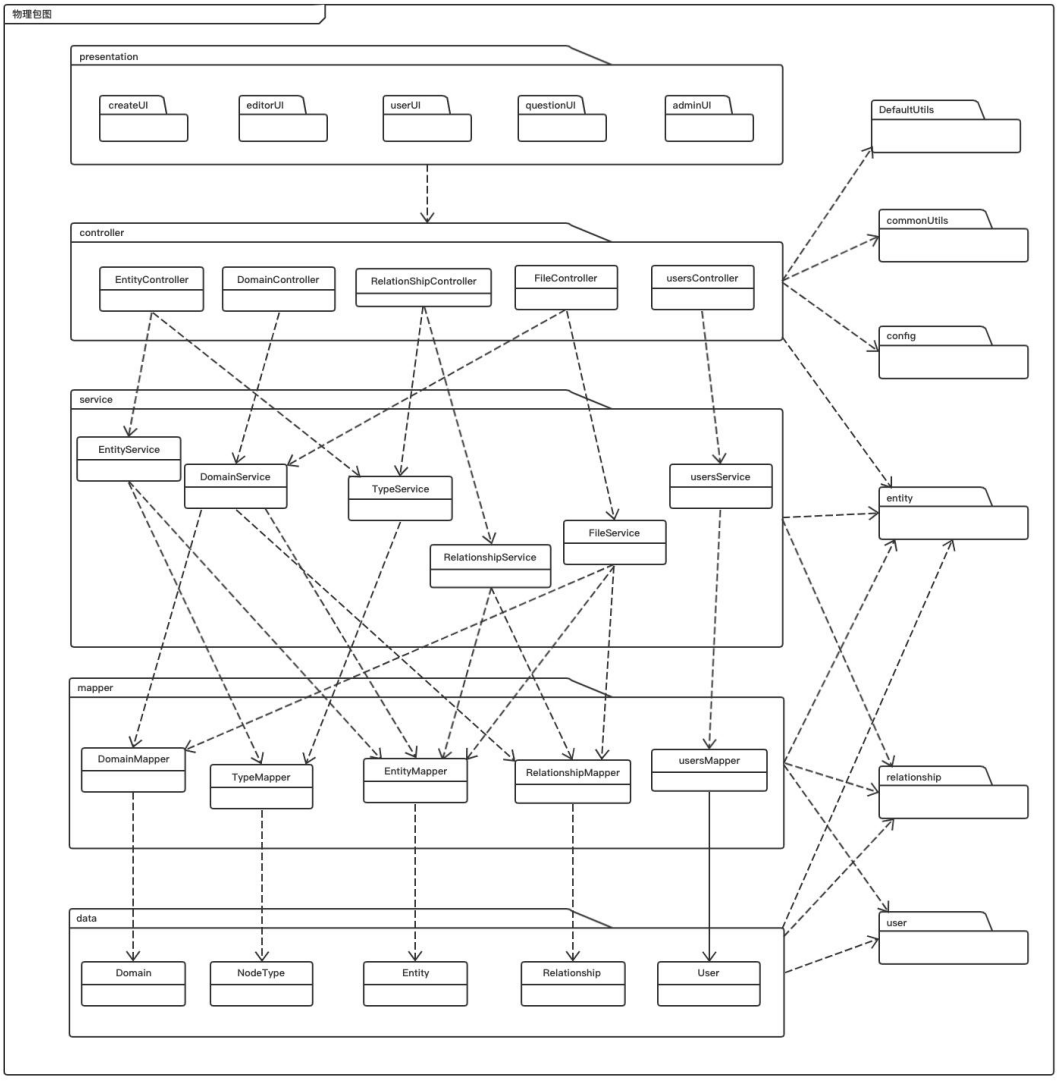

逻辑包图
表示层
createUI
editorUI
userUI
questionUI
adminUI
业务层
DomainService
EntityService
FileService
RelationshipService
TypeService
userService
questionService
adminService
数据层
Domain
Entity
Relationship
NodeType
User

3　组合视角

3.1 物理包划分

开发包	依赖的其他开发包
commonUtils	
config	
controller	service, commonUtils, config, utils
entity	
service	entity
impl	service, utils
mapper	entity
utils	

3.2 物理包图


物理包图
presentation
createUI
editorUI
userUI
questionUI
adminUI
controller
EntityController
DomainController
RelationShipController
FileController
usersController
service
EntityService
DomainService
TypeService
RelationshipService
FileService
usersService
mapper
DomainMapper
TypeMapper
EntityMapper
RelationshipMapper
usersMapper
data
Domain
NodeType
Entity
Relationship
User
DefaultUtils
commonUtils
config
entity
relationship
user

4　接口视角

4.1 模块的职责

层	职责
启动模块	负责初始化网络通信机制，启动用户界面
用户界面层	客户端用户界面
后台界面层	后台管理系统界面
接口模块	负责客户端和服务器端的通信和数据传递
业务逻辑层	对于用户界面和后台管理系统界面的输入进行响应并进行业务处理逻辑
数据服务层	抽象出的数据操作接口
数据层	负责数据的持久化和访问

4.2 模块的接口规范

4.2.1 用户界面层分解

服务名	服务
entity.createNodeAPI	创建节点
entity.getNodesByDomainIdAPI	根据域名 id 得到该域内的所有节点
entity.updateNodeAPI	更新节点信息
entity.deleteNodeAPI	删除节点
entity.countNodeAPI	统计节点个数
entity.countNodeByTypeAPI	统计特定类型节点个数
entity.updateXYAPI	更新全体节点坐标
entity.getTypesAPI	得到类型列表
entity.getNodeByTypeAPI	得到特定类型节点
entity.updateTypeAPI	更新节点类型

relationship.createLinkAPI	创建关系
relationship.updateLinkAPI	更新关系信息
relationship.deleteLinkAPI	删除关系
relationship.getLinkByDomainIdAPI	根据域名 id 得到该域内的所有关系
relationship.getGraphScreenAPI	按节点筛选关系图
relationship.countLinkAPI	统计关系个数
domain.createDomainAPI	创建域
domain.updateDomainAPI	更新域信息
domain.deleteDomainAPI	删除域
domain.selectDomainByIdAPI	根据域名 id 得到对应的域
domain.selectAllDomainAPI	获取所有的域
editor.createFromFileAPI	根据文件创建图谱
file.getCsvFileAPI	得到 csv 文件
file.downloadAPI	下载文件
users.loginUserAPI	用户登录系统
users.registerUserAPI	用户注册系统
users.getUserInfoAPI	获得用户信息
users.updateUserInfoAPI	更新用户信息
users.updateUserInfoAdminAPI	管理员更新用户信息
users.updateAvatarAPI	更新用户头像
users.updateUserPwdAPI	修改用户密码
users.resetPwdAPI	重置用户密码
users.getUserListPageAPI	条件查询用户
users.deleteUserAPI	删除用户
users.addUserAPI	添加用户
users.disableUserAPI	注销用户或撤销注销
users.setVipUserAPI	将用户设置为 VIP

4.2.2 业务逻辑层分解

4.2.2.1 EntityService 的接口规范

模块		职责
EntityService		负责对节点的职责
提供的服务（供接口）		
EntityService.createNode	语法	Result createNode(Entity entity)
	前置条件	entity 对象中的 name、domainId、type 不为空
	后置条件	根据输入创建节点信息
EntityService.updateNode	语法	Result updateNode(Entity entity)
	前置条件	待被更新的节点存在
	后置条件	根据输入更新节点信息
EntityService.deleteNode	语法	Result deleteNode(Entity entity)
	前置条件	待被删除的节点存在
	后置条件	根据输入删除节点及其关系
EntityService.getNodesByDomainId	语法	Result getNodesByDomainId(int domainId)
	前置条件	输入的 domainId 存在
	后置条件	根据输入返回符合条件的节点列表
EntityService.countNode	语法	Result countNode(int domainId)
	前置条件	输入的 domainId 存在
	后置条件	根据输入返回该域节点数目
EntityService.countNodeByType	语法	Result countNodeByType(int domainId,String type)
	前置条件	输入的 domainId 存在
	后置条件	根据输入返回该域特定类型节点数目
EntityService.updateXY	语法	Result updateXY(List<Entity> entities)
	前置条件	输入的节点均存在
	后置条件	更新节点位置信息

EntityService.getTypes	语法	Result getTypes(int domainId)
	前置条件	输入的 domainId 存在
	后置条件	根据输入返回该域类型一栏
EntityService.getNodeByType	语法	Result getNodeByType(int domainId,String type)
	前置条件	输入的 domainId 存在
	后置条件	根据输入返回同类型节点
EntityService.updateType	语法	Result updateType(id,oldType,newType,domainId)
	前置条件	输入的 domainId 和 id 存在，且两类型不同
	后置条件	根据输入更新节点类型

4.2.2.2 RelationshipService 的接口规范

模块		**职责**
RelationshipService		负责对节点关系的职责
提供的服务（供接口）		
RelationshipService.createLink	语法	Result createLink(Long fromId, Long toId, String name)
	前置条件	fromId 和 toId 均存在且不为空
	后置条件	根据输入创建两节点间关系
RelationshipService.updateLink	语法	Result updateLink(Relationship relationship)
	前置条件	待被更新的关系存在
	后置条件	根据输入，修改关系信息
RelationshipService.deleteLink	语法	Result deleteLink(Relationship relationship)
	前置条件	待被删除的关系存在

	后置条件	根据输入，删除关系信息
RelationshipService.getLinkByDomainId	语法	Result getLinkByDomainId(int domainId)
	前置条件	输入 domianId 合法
	后置条件	根据输入，返回域内所有关系图谱（含孤立节点）
RelationshipService.getGraphScreen	语法	Result getGraphScreen(int domainId,List<Entity> entities)
	前置条件	输入 domianId 合法，节点列表不为空
	后置条件	根据输入，筛选所有关系图谱（含孤立节点，仅显示这些节点及其附属子节点关系）
RelationshipService.countLink	语法	Result countLink(int domainId)
	前置条件	输入 domianId 合法
	后置条件	根据输入，返回域内关系个数

4.2.2.3 DomainService 的接口规范

模块		**职责**
DomainService		负责对图谱的职责
提供的服务（供接口）		
DomainService.createDomain	语法	Result createDomain(Domain domain)
	前置条件	数据库不存在同名域
	后置条件	根据输入，创建域
DomainService.updateDomain	语法	Result updateDomain(Domain domain)
	前置条件	待被更新的域已存在
	后置条件	根据输入，修改域名

DomainService.deleteDomain	语法	Result deleteDomain(int domainId)
	前置条件	待被删除的域存在
	后置条件	根据输入，删除该域及域内的关系和节点
DomainService.selectDomainById	语法	Result selectDomainById(int domainId)
	前置条件	数据库存在该域
	后置条件	根据输入，返回该域
DomainService.selectAllDomain	语法	Result selectAllDomain()
	前置条件	无
	后置条件	根据输入，返回所有域
DomainService.getDomainListPage	语法	Result getDomainListPage(long current, long limit, DomainQuery domainQuery)
	前置条件	无
	后置条件	根据查询条件，返回符合条件的所有域

4.2.2.4 FileService 的接口规范

模块		**职责**
FileService		负责对文件流控制
提供的服务（供接口）		
FileService.getCsvFile	语法	Result getCsvFile(MultipartFile file)
	前置条件	csv 文件符合输入规范（域已创建）
	后置条件	根据输入，在域内新建关系图谱
FileService.download	语法	Result download(String domainName, int type, final HttpServletRequest request, final HttpServletResponse response)

	前置条件	需要的图谱存在
	后置条件	下载对应图谱的 xml 文件
需要的服务（需接口）		
服务名		服务
EntityService.createNode		创建节点
DomainService.getDomainById		根据域名 id 得到该域

4.2.2.5 UsersService 的接口规范

模块		**职责**
usersService		负责对图谱的职责
提供的服务（供接口）		
usersService.loginUser	语法	Result loginUser(LoginVo loginVo)
	前置条件	该用户已注册
	后置条件	根据输入的验证信息进行登录
usersService.registerUser	语法	Result registerUser(RegisterVO registerVO)
	前置条件	该用户未注册
	后置条件	根据输入注册用户
usersService.getUserInfo	语法	Result getUserInfo(HttpServletRequest request)
	前置条件	前端用户处于已登录的状态
	后置条件	查出用户信息
usersService.updateUserInfo	语法	Result updateUserInfo(String id, String nickname, String sign)
	前置条件	前端用户处于已登录的状态
	后置条件	更新用户信息

usersService.updateUserInfoAdmin	语法	Result updateUserInfoAdmin (UserInfoAdminVO userInfoAdminVO)
	前置条件	登录管理员账户
	后置条件	更新用户信息，包括等级、会员等
usersService.updateAvatar	语法	Result updateAvatar(UserInfoVO userInfoVO)
	前置条件	前端用户处于已登录的状态
	后置条件	更新用户头像
usersService.updateUserPwd	语法	Result updateUserPwd(String id, String oldPwd, String newPwd)
	前置条件	前端用户处于已登录的状态
	后置条件	修改用户密码
usersService.resetPwd	语法	Result resetPwd(String mobile, String password)
	前置条件	前端用户处于已登录的状态
	后置条件	重置用户密码
usersService.getUserListPage	语法	Result getUserListPage(long current, long limit, UserQuery userQuery)
	前置条件	无
	后置条件	根据查询条件查询用户
usersService.deleteUser	语法	Result deleteUser(String id)
	前置条件	该用户存在
	后置条件	用户被删除
usersService.addUser	语法	Result addUser(User user)
	前置条件	要添加的用户不存在
	后置条件	用户被添加进数据库
usersService.disableUser	语法	Result disableUser(String id)
	前置条件	该用户存在
	后置条件	用户账号被注销，无法再次登录

usersService.setVipUser	语法	Result setVipUser(String id, int days)
	前置条件	该用户存在
	后置条件	该用户设置为 vip

4.2.3 数据层分解

4.2.3.1 EntityMapper 的接口规范

模块		**职责**
EntityMapper		负责对节点数据的访问控制
提供的服务（供接口）		
EntityMapper.getNodeByDomainId	语法	List<Entity> getNodeByDomainId(int domainId)
	前置条件	域名 domainId 不为空且存在
	后置条件	根据输入，返回该域内所有节点
EntityMapper.updateNode	语法	Entity updateNode(Long id, String name, String description,int shape, double x, double y)
	前置条件	节点 id 不为空且存在
	后置条件	根据其他输入，更新节点内容
EntityMapper.getUnLinkNodes	语法	List<Entity> getUnLinkNodes(int domainId)
	前置条件	域名 domainId 不为空且存在
	后置条件	根据输入，返回该域内所有孤立节点
EntityMapper.deleteNodeWithLink	语法	void deleteNodeWithLink(Long id)
	前置条件	数据库存在该节点
	后置条件	删除该节点及其附属关系
EntityMapper.findByName	语法	Entity findByName(String name, int

		domainId);
	前置条件	输入合法
	后置条件	查找该名称的节点
EntityMapper.countAllEntity	语法	int countAllEntity(int domainId);
	前置条件	域名存在
	后置条件	返回域内节点个数
EntityMapper.countEntitiesByType	语法	int countEntitiesByType(int domainId, String type);
	前置条件	域名存在
	后置条件	返回域内特定类型节点个数
EntityMapper.updateXY	语法	void updateXY(Long id, double x, double y);
	前置条件	节点已存在
	后置条件	更新一个节点的坐标
EntityMapper.getNodeByType	语法	List<Entity> getNodeByType(int domainId, String type);
	前置条件	域名存在
	后置条件	返回一个类型的节点数据
EntityMapper.updateType	语法	void updateXY(Long id, String type, String bgColor);
	前置条件	节点已存在
	后置条件	更新一个节点类型和方法

4.2.3.2 RelationshipMapper 的接口规范

模块	**职责**
RelationshipMapper	负责对关系数据的访问控制
提供的服务（供接口）	

RelationshipMapper.updateLink	语法	Relationship updateLink(Long id, String name)
	前置条件	关系 id 不为空且存在
	后置条件	根据其他输入，更新关系内容
RelationshipMapper.getLinkByDomainId	语法	List<Relationship> getLinkByDomainId(int domainId)
	前置条件	域名 domainId 不为空且存在
	后置条件	根据输入，返回该域内所有关系
RelationshipMapper.countAllLink	语法	int countAllLink(int domainId);
	前置条件	域名 domainId 不为空且存在
	后置条件	根据输入，返回该域内关系个数

4.2.3.3 DomainMapper 的接口规范

模块		**职责**
DomainMapper		负责对域的访问控制
提供的服务（供接口）		
DomainMapper.createDomain	语法	void createDomain(Domain domain)
	前置条件	数据库没有重复域
	后置条件	根据输入，新增域
DomainMapper.updateDomain	语法	void updateDomain(String name, int id)
	前置条件	数据库存在该域
	后置条件	根据输入，更新该域
DomainMapper.deleteDomain	语法	void deleteDomain(int id)
	前置条件	数据库存在该域
	后置条件	根据输入，删除该域
DomainMapper.selectDomain	语法	Domain selectDomain(int id)
	前置条件	域名 id 不为空且存在

	后置条件	根据输入，返回该域
DomainMapper.selectAllDomain	语法	List<Domain> selectAllDomain()
	前置条件	无
	后置条件	根据输入，返回所有域
DomainMapper.lastInsertId	语法	int lastInsertId();
	前置条件	已进行过新增域操作
	后置条件	根据输入，返回最新新增域的 id

4.2.3.4 TypeMapper 的接口规范

模块		**职责**
TypeMapper		负责对类型的访问控制
提供的服务（供接口）		
TypeMapper.insertType	语法	insertType(int domainId,String color, String nodeType);
	前置条件	域名合法且存在
	后置条件	根据输入，新增一个类型
TypeMapper.deleteType	语法	void deleteType(int domainId, String type);
	前置条件	数据库存在该类型
	后置条件	根据输入，删除该类型
TypeMapper.searchColorByType	语法	String searchColorByType(int domainId,String nodeType);
	前置条件	数据库存在对应记录数据
	后置条件	根据输入，返回类型对应的颜色
TypeMapper.searchAll	语法	List<String> searchAll(int domainId);
	前置条件	域名 id 不为空且存在
	后置条件	根据输入，返回域持有的类型一栏

5　信息视角

5.1 VO 定义

5.1.1 Result

含义	属性	字段
是否成功	Boolean	success
返回码	Integer	code
返回消息	String	message
返回数据	Map<String, Object>	data

5.1.2 Entity

含义	属性	字段
id 标识码	Long	id
节点名称	String	name
节点颜色	String	bgColor
节点样式类型	int	shape
所属域	int	domainId
节点属性类型	String	type
节点描述	String	description
节点 x 坐标	double	x
节点 y 坐标	double	y
节点半径	double	r
节点文本字体	int	fontSize

5.1.3 Relationship

含义	属性	字段
id 标识码	Long	id
关系名称	String	name
关系样式类型	int	type
所属域	int	domainId
起始节点	Entity	startEntity
起始节点码	Long	fromId
终止节点	Enitiy	endEntity
终止节点码	Long	toId

5.1.4 Domain

含义	属性	字段
域标识码	int	id
域名	String	name
所属用户 id	String	user_id
创建时间	Date	createTime
更新时间	Date	modifyTime

5.1.5 DomainQuery

含义	属性	字段
图谱名称	String	name
所属用户 id	String	userId
查询开始时间	Date	begin
查询结束时间	Date	end

5.1.6 User

含义	属性	字段
用户 id	String	id
手机号码	String	mobile
密码	String	password
用户名	String	nickname
等级	Integer	level
头像	String	avatar
是否注销	Boolean	isDisabled
是否为会员	Boolean	isVip
会员到期时间	Date	vipEndTime
用户签名	String	sign
创建时间	Date	createTime
更新时间	Date	modifyTime

5.1.7 LoginVO

含义	属性	字段
手机号	String	mobile
密码	String	password

5.1.8 RegisterVO

含义	属性	字段
手机号	String	mobile
密码	String	password
用户名	String	nickname

5.2 数据库表

数据库表包含 entity 表、relationship 表、domain 表、nodeType 表和 user 表

6 Pipeline 脚本

```
pipeline {
    agent any

    stages {
        stage('Maven Build') {
            steps{
                echo 'Maven Build and Cobertura Stage'
                sh '/opt/apache-maven-3.6.3/bin/mvn clean -DskipTests=true
package'
            }
        }

        stage('Jacoco Report') {
                    steps{
                        echo 'Jacoco Stage'
                        sh '/opt/apache-maven-3.6.3/bin/mvn test'
                    }
                }

        stage('Image Clear'){
            steps{
                echo 'Image Clear Stage'
                sh "if (docker ps -a| grep coin) then (docker container stop coin &&
docker container rm coin) fi"
                sh "if (docker images | grep coin) then (docker rmi \$(docker images
coin -q)) fi"
            }
        }

        stage('Image Build'){
```

```
                steps{
                    echo 'Image Build Stage'
                    sh "docker build . -t coin:${BUILD_ID}"
                }
            }

        stage('Image-oss Clear'){
            steps{
                echo 'Image-oss Clear Stage'
                sh "mv Dockerfile Dockerfile-default"
                sh "mv Dockerfile-oss Dockerfile"
                sh "if (docker ps -a| grep coin-oss) then (docker container stop coin-oss
&& docker container rm coin-oss) fi"
                sh "if (docker images | grep coin-oss) then (docker rmi \$(docker
images coin-oss -q)) fi"
            }
        }

         stage('Image-oss Build'){
            steps{
                echo 'Image-oss Build Stage'
                sh "docker build . -t coin-oss:${BUILD_ID}"
            }
        }

        stage('Image-user Clear'){
            steps{
                echo 'Image-user Clear Stage'
                sh "mv Dockerfile Dockerfile-oss"
                sh "mv Dockerfile-user Dockerfile"
                sh "if (docker ps -a| grep coin-user) then (docker container stop
coin-user && docker container rm coin-user) fi"
                sh "if (docker images | grep coin-user) then (docker rmi \$(docker
images coin-user -q)) fi"
            }
        }

         stage('Image-user Build'){
            steps{
                echo 'Image-user Build Stage'
                sh "docker build . -t coin-user:${BUILD_ID}"
            }
        }
```

```
        stage('Deploy'){
            steps{
                sh "docker run -p 8002:8002 --name coin -v /log:/log -d
coin:${BUILD_ID}"
                sh "docker run -p 8003:8003 --name coin-oss -v /log:/log -d
coin-oss:${BUILD_ID}"
                sh "docker run -p 8004:8004 --name coin-user -v /log:/log -d
coin-user:${BUILD_ID}"
            }
        }
    }
    post {
            success {
                // publish html
                publishHTML target: [
                    allowMissing: false,
                    alwaysLinkToLastBuild: false,
                    keepAll: true,
                    reportDir: '',
                    reportFiles: 'index.html',
                    reportName: 'Jacoco Report'
                ]
            }
        }
}
```