COIN 项目设计文档

1 文档修改历史

修改人员	日期	修改原因	版本号
李宇轩	2021.03.27	初始版本	V1.0

2 引言

2.1 编制目的

本文档提供COIN知识图谱可视化系统的软件架构的概要设计，采用若干架构层面识图描述系统达到指导详细设计与开发的目的，同时实现与测试人员及用户的沟通。本报告面向开发人员、测试人员及最终用户而编写，是了解系统的导航。

2.2 词汇表

词汇名称	词汇含义	备注
COIN	知识图谱定义及可视化系统	无

2.3 参考资料

[1] CSEIII 01-COIN项目介绍文档

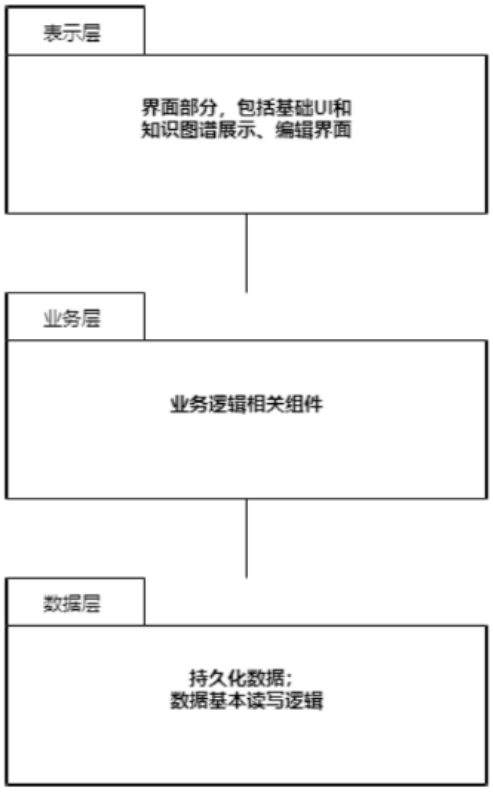
[2] COIN需求规格说明文档

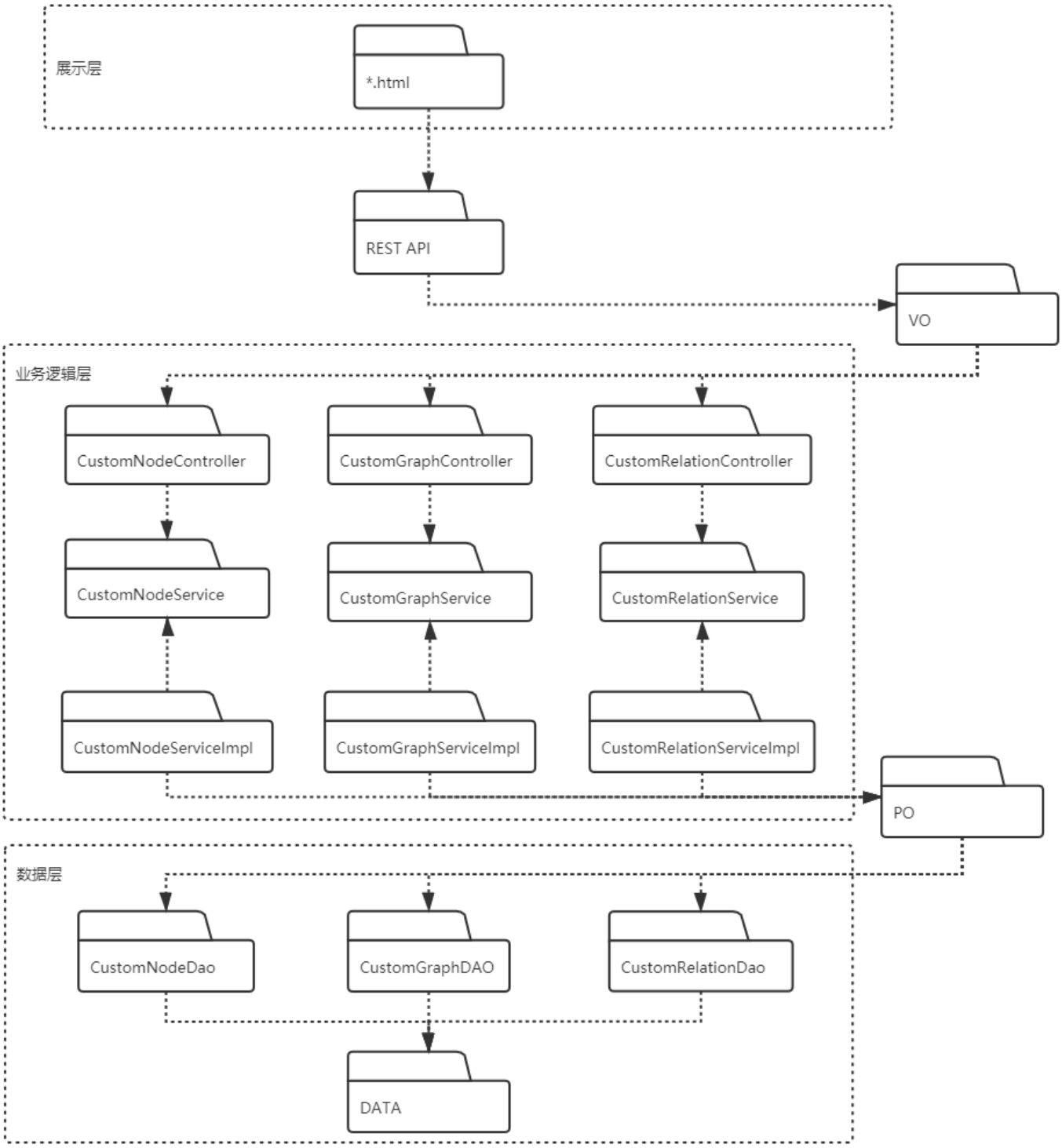
[3]《软件工程与计算（卷三） 团队与软件开发实践》，骆斌、刘嘉、张瑾玉、黄蕾，机械工业出版社

3 逻辑视角

3.1 系统的分层架构

COIN系统中，选择了分层体系结构风格，将系统分为3层（展示层、业务逻辑层、数据层）能够更好地示意整个高层抽象。展示层包含GUI页面的实现，业务逻辑层包含业务逻辑处理的实现，数据层负责数据的持久化和访问。





3.2 系统架构设计

- **系统架构中的对象**
  - UI对象，负责处理系统数据的展现和用户的交互

Controller包中的对象

- CustomGraphController对象，负责获取用户输入，并调用CustomGraphServiceImpl接口的服务
- CustomNodeController对象，负责获取用户输入，并调用CustomNodeServiceImpl接口的服务
- CustomRelationController对象，负责获取用户输入，并调用CustomRelationServiceImpl接口的服务

bl包中的对象

- CustomGraphService对象，关于CustomGraph对象业务逻辑服务的抽象接口，获取从数据端组装好的数据
- CustomNodeService对象，关于CustomNode对象业务逻辑服务的抽象接口，获取从数据端组装好的数据
- CustomRelationService对象，关于CustomRelation对象业务逻辑服务的抽象接口，获取从数据端组装好的数据

blmpl包中的对象

- CustomGraphServiceImpl对象，负责CustomGraphService接口的实现
- CustomNodeServiceImpl对象，负责CustomNodeService接口的实现
- CustomRelationServiceImpl对象，负责CustomRelationService接口的实现

DAO包中的对象

- CustomNodeDao对象，操作neo4j数据库节点label的接口，获取数据
- CustomRelationDao对象，操作neo4j数据库关系label的接口，获取数据
- CustomGraphDAO对象，操作neo4j数据库关系label的接口，获取数据

PO包中的对象

- CustomGraph对象，用于图的持久化数据封装
- CustomNode对象，用于节点的持久化数据封装
- CustomRelation对象，用于关系的持久化数据封装

VO包中的对象

- CustomGraphVO对象，用于图的表现层数据封装
- CustomNodeVO对象，用于节点的表现层数据封装
- CustomRelationVO对象，用于关系的表现层数据封装
- ResponseVO对象，用于请求结果的表现层数据封装

- **系统对外接口设计**

  CustomGraphController包中包含的服务

	提供的服务（供接口）	
CustomGraphController.add	语法	ResponseVO add(String name);
	API调用格式	/api/customGraph/add/{name}
	前置条件	用户请求添加一个图
	后置条件	调用service，在Dao层完成添加一个图的逻辑
CustomGraphController.delete	语法	ResponseVO delete(Long id);
	API调用格式	/api/customGraph/delete/{id}
	前置条件	用户请求删除一个图
	后置条件	调用service，在Dao层完成删除一个图的逻辑
CustomGraphController.edit	语法	ResponseVO edit(CustomGraphVO customGraphVO);
	API调用格式	/api/customGraph/edit
	前置条件	用户请求编辑一个图
	后置条件	调用service，在Dao层完成编辑一个图的逻辑
CustomGraphController.retrieveAllNode	语法	List retrieveAllNode(Long id);
	API调用格式	/api/customGraph/retrieveAllNode/{id}
	前置条件	用户请求获取图的所有节点
	后置条件	调用service，在Dao层完成获取图的逻辑
CustomGraphController.retrieveAllRelation	语法	List retrieveAllRelation(Long id);
	API调用格式	/api/customGraph/retrieveAllRelation/{id}
	前置条件	用户请求获取图的全部关系
	后置条件	调用service，在Dao层完成获取图的全部关系的逻辑
CustomGraphController.loadGraph	语法	List loadGraph(MultipartFile multipartFile)
	API调用格式	/api/customGraph/loadGraph
	前置条件	用户请求加载一个图谱
	后置条件	调用service，在Dao层完成获取加载一个图谱的逻辑
CustomGraphController.fuzzyMatching	语法	List fuzzyMatching(String jsonString);
	API调用格式	/api/customGraph/fuzzyMatching
	前置条件	用户请求模糊搜索节点信息
	后置条件	调用service，在Dao层完成获取模糊搜索节点信息的逻辑
CustomGraphController.typesetting	语法	List typesetting(Long id);
	API调用格式	/api/customGraph/typesetting/{id}
	前置条件	用户请求对图谱进行排版模式
	后置条件	调用service，在Dao层完成获取排版模式的逻辑

CustomNodeController包中包含的服务

	提供的服务（供接口）	
CustomNodeController.add	语法	ResponseVO add(CustomNodeVO customNodeVO);
	API调用格式	/api/customNode/add
	前置条件	用户请求添加一个节点
	后置条件	调用service，在Dao层完成添加节点的逻辑

CustomNodeController.delete	语法	ResponseVO delete(long id);
	API调用格式	/api/customNode/delete/{id}
	前置条件	用户请求删除一个节点
	后置条件	调用service，在Dao层完成删除节点的逻辑
CustomNodeController.edit	语法	ResponseVO edit(CustomNodeVO customNodeVO);
	API调用格式	/api/customNode/edit
	前置条件	用户请求编辑一个节点
	后置条件	调用service，在Dao层完成编辑节点的逻辑
CustomNodeController.retrieve	语法	Optional retrieve(long id);
	API调用格式	/api/customNode/retrieve/{id}
	前置条件	用户请求获取一个节点
	后置条件	调用service，在Dao层完成获取节点的逻辑
CustomNodeController.retrieveAll	语法	List retrieveAll();
	API调用格式	/api/customNode/retrieveAll
	前置条件	用户请求获取全部节点
	后置条件	调用service，在Dao层完成获取全部节点的逻辑
CustomNodeController.retrieveSubNodes	语法	ResponseVO retrieveSubNodes(long id);
	API调用格式	/api/customNode/retrieveSubNodes/{id}
	前置条件	用户请求获取一个节点的全部子节点
	后置条件	调用service，在Dao层完成获取全部子节点的逻辑

CustomRelationController包中包含的服务

提供的服务（供接口）

customRelationController.add	语法	ResponseVO add(CustomRelationVO customRelationVO);
	API调用格式	/api/customRelation/add
	前置条件	用户请求添加一个关系
	后置条件	调用service，在Dao层完成添加关系的逻辑
customRelationController.delete	语法	ResponseVO delete(long fromId, long toId);
	API调用格式	/api/customRelation/delete/{id}
	前置条件	用户请求删除一个关系的相关节点
	后置条件	调用service，在Dao层完成删除关系的逻辑
customRelationController.delete	语法	ResponseVO delete(long id);
	API调用格式	/api/customRelation/delete/{id}
	前置条件	用户请求删除一个关系
	后置条件	调用service，在Dao层完成删除关系的逻辑
customRelationController.edit	语法	ResponseVO edit(CustomNodeVO customNodeVO);
	API调用格式	/api/customRelation/edit
	前置条件	用户请求编辑一个关系
	后置条件	调用service，在Dao层完成编辑节点的逻辑
customRelationController.retrieve	语法	CustomRelation retrieve(Long fromId, Long toId);
	API调用格式	/api/customRelation/retrieve/{fromId}/{toId}
	前置条件	用户请求获取一个关系的相关节点
	后置条件	调用service，在Dao层完成获取关系的逻辑
customRelationController.retrieve	语法	Optional retrieve(Long id);
	API调用格式	/api/customRelation/retrieve/{id}
	前置条件	用户请求获取一个关系
	后置条件	调用service，在Dao层完成获取关系的逻辑
customRelationController.retrieveAll	语法	List retrieveAll();
	API调用格式	/api/customRelation/retrieveAll
	前置条件	用户请求获取全部关系
	后置条件	调用service，在Dao层完成获取全部关系的逻辑

4 Jenkins部署

前端项目Pipeline脚本

```
    pipeline {
        agent any
        options {
            timestamps()
            timeout(10)
        }
        stages {
            stage('git') {
                steps {
                    checkout([$class:'GitSCM', branches: [[name:
                    '*/master']],extensions: [],
                    userRemoteConfigs: [[credentialsId:
                    'e6580780-e75d-4e95-a44d-ed8e783caf03', url:
                    'http://212.129.149.40/181250076_ac/frontend-coin.git']]])
                }
            }
```

```
            stage('build') {
                steps {
                    nodejs('nodejs') {
                    }
                    sh 'npm install'
                    sh 'npm run build'
                }
            }
            stage('mv&del') {
                steps {
                    sh 'rm -rf /tomcat/apache-tomcat-8.5.64/webapps/dist'
                    sh 'mv /var/lib/jenkins/workspace/frontend_coin/dist /tomcat/apache-tomcat8.5.64/webapps'
                }
            }
            stage('clean') {
                steps {
                    cleanWs()
                }
            }
        }
    }
```

后端项目Pipeline脚本

```
pipeline {
        agent any
        options {
            timestamps()
            timeout(10)
        }
        stages {
            stage('git') {
                steps {
                    checkout([$class:'GitSCM', branches: [[name:
                    '*/master'], [name:
                    '*/release']],
                    extensions: [],userRemoteConfigs: [[credentialsId:
                    'e6580780-e75d-4e95-a44d-ed8e783caf03', url:
                    'http://212.129.149.40/181250076_ac/backend-projectname.git']]])
                }
            }
            stage('build') {
                steps {
                    sh 'mvn clean package -DskipTests'
                }
            }
            stage('publish code') {
                steps {
                    deploy adapters: [tomcat8(credentialsId:'bd1c8fe9-582f-49eb-bdce-e13e8705a567',
                            path:'', url:'http://101.200.52.46:8080/')],contextPath:
                    null, war:'target/*.war'
                }
            }
        }
    }
```